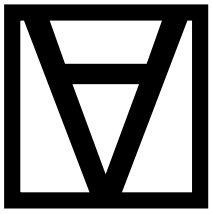
Sample Logic Text

Open Logic Project



Contents

Part I

Sets, Relations, Functions

Chapter 1

Sets

1.1 Basics

Sets are the most fundamental building blocks of mathematical objects. In fact, almost every mathematical object can be seen as a set of some kind. In logic, as in other parts of mathematics, sets and set-theoretical talk is ubiquitous. So it will be important to discuss what sets are, and introduce the notations necessary to talk about sets and operations on sets in a standard way.

Definition 1.1 (Set)**.** A *set* is a collection of objects, considered independently of the way it is specified, of the order of the objects in the set, or of their multiplicity. The objects making up the set are called *elements* or *members* of the set. If a is an element of a set X, we write $a \in X$ (otherwise, $a \notin X$). The set which has no elements is called the *empty* set and denoted by the symbol $\emptyset$.

Example 1.2. Whenever you have a bunch of objects, you can collect them together in a set. The set of Richard's siblings, for instance, is a set that contains one person, and we could write it as $S = \{\text{Ruth}\}$. In general, when we have some objects $a_1, \ldots, a_n$, then the set consisting of exactly those objects is written $\{a_1, \ldots, a_n\}$. Frequently we'll specify a set by some property that its elements share—as we just did, for instance, by specifying S as the set of Richard's siblings. We'll use the following shorthand notation for that: $\{x \mid \ldots x \ldots\}$, where the $\ldots x \ldots$ stands for the property that x has to have in order to be counted among the elements of the set. In our example, we could have specified S also as

$$S = \{x \mid x \text{ is a sibling of Richard}\}.$$

When we say that sets are independent of the way they are specified, we mean that the elements of a set are all that matters. For instance, it so happens

that

$$\{\text{Nicole, Jacob}\},$$
$$\{x \mid \text{is a niece or nephew of Richard}\}, \text{ and}$$
$$\{x \mid \text{is a child of Ruth}\}$$

are three ways of specifying one and the same set.

Saying that sets are considered independently of the order of their elements and their multiplicity is a fancy way of saying that

$$\{\text{Nicole, Jacob}\} \text{ and}$$
$$\{\text{Jacob, Nicole}\}$$

are two ways of specifying the same set; and that

$$\{\text{Nicole, Jacob}\} \text{ and}$$
$$\{\text{Jacob, Nicole, Nicole}\}$$

are also two ways of specifying the same set. In other words, all that matters is which elements a set has. The elements of a set are not ordered and each element occurs only once. When we *specify* or *describe* a set, elements may occur multiple times and in different orders, but any descriptions that only differ in the order of elements or in how many times elements are listed describes the same set.

Definition 1.3 (Extensionality)**.** If X and Y are sets, then X and Y are *identical*, $X = Y$, iff every element of X is also an element of Y, and vice versa.

Extensionality gives us a way for showing that sets are identical: to show that $X = Y$, show that whenever $x \in X$ then also $x \in Y$, and whenever $y \in Y$ then also $y \in X$.

1.2 Some Important Sets

Example 1.4. Mostly we'll be dealing with sets that have mathematical objects as members. You will remember the various sets of numbers: $\mathbb{N}$ is the set of *natural* numbers $\{0, 1, 2, 3, \dots\}$; $\mathbb{Z}$ the set of *integers*,

$$\{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\};$$

$\mathbb{Q}$ the set of *rational* numbers ($\mathbb{Q} = \{z/n \mid z \in \mathbb{Z}, n \in \mathbb{N}, n \neq 0\}$); and $\mathbb{R}$ the set of *real* numbers. These are all *infinite* sets, that is, they each have infinitely many elements. As it turns out, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ have the same number of elements, while $\mathbb{R}$ has a whole bunch more—$\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$ are "countable and infinite" whereas $\mathbb{R}$ is "uncountable".

We'll sometimes also use the set of positive integers $\mathbb{Z}^+ = \{1, 2, 3, \dots\}$ and the set containing just the first two natural numbers $\mathbb{B} = \{0, 1\}$.

Example 1.5 (Strings)**.** Another interesting example is the set A^* of *finite strings* over an alphabet A: any finite sequence of elements of A is a string over A. We include the *empty string* Λ among the strings over A, for every alphabet A. For instance,

$$\mathbb{B}^* = \{\Lambda, 0, 1, 00, 01, 10, 11, \\ 000, 001, 010, 011, 100, 101, 110, 111, 0000, \ldots\}.$$

If $x = x_1 \ldots x_n \in A^*$is a string consisting of n "letters" from A, then we say *length* of the string is n and write $\text{len}(x) = n$.

Example 1.6 (Infinite sequences)**.** For any set A we may also consider the set A^ω of infinite sequences of elements of A. An infinite sequence $a_1a_2a_3a_4\ldots$ consists of a one-way infinite list of objects, each one of which is an element of A.

1.3 Subsets

Sets are made up of their elements, and every element of a set is a part of that set. But there is also a sense that some of the elements of a set *taken together* are a "part of" that set. For instance, the number 2 is part of the set of integers, but the set of even numbers is also a part of the set of integers. It's important to keep those two senses of being part of a set separate.

Definition 1.7 (Subset)**.** If every element of a set X is also an element of Y, then we say that X is a *subset* of Y, and write $X \subseteq Y$.

Example 1.8. First of all, every set is a subset of itself, and $\emptyset$ is a subset of every set. The set of even numbers is a subset of the set of natural numbers. Also, $\{a, b\} \subseteq \{a, b, c\}$.

But $\{a, b, e\}$ is not a subset of $\{a, b, c\}$.

Note that a set may contain other sets, not just as subsets but as elements! In particular, a set may happen to *both* be an element and a subset of another, e.g., $\{0\} \in \{0, \{0\}\}$ and also $\{0\} \subseteq \{0, \{0\}\}$.

Extensionality gives a criterion of identity for sets: $X = Y$ iff every element of X is also an element of Y and vice versa. The definition of "subset" defines $X \subseteq Y$ precisely as the first half of this criterion: every element of X is also an element of Y. Of course the definition also applies if we switch X and Y: $Y \subseteq X$ iff every element of Y is also an element of X. And that, in turn, is exactly the "vice versa" part of extensionality. In other words, extensionality amounts to: $X = Y$ iff $X \subseteq Y$ and $Y \subseteq X$.

Definition 1.9 (Power Set)**.** The set consisting of all subsets of a set X is called the *power set of* X, written $\wp(X)$.

$$\wp(X) = \{Y \mid Y \subseteq X\}$$

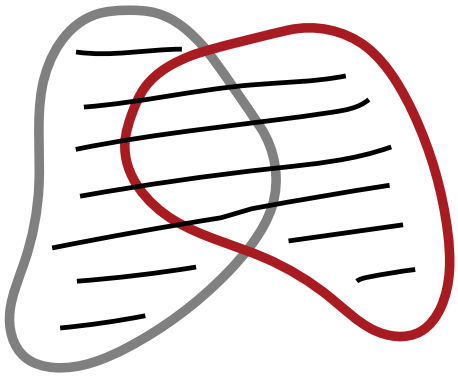


Figure 1.1: The union $X \cup Y$ of two sets is set of elements of X together with those of Y.

Example 1.10. What are all the possible subsets of $\{a, b, c\}$? They are: $\emptyset$, $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{a, c\}$, $\{b, c\}$, $\{a, b, c\}$. The set of all these subsets is $\wp(\{a, b, c\})$:

$$\wp(\{a, b, c\}) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}\}$$

1.4 Unions and Intersections

We can define new sets by abstraction, and the property used to define the new set can mention sets we've already defined. So for instance, if X and Y are sets, the set $\{x \mid x \in X \vee x \in Y\}$ defines a set which consists of all those objects which are elements of either X or Y, i.e., it's the set that combines the elements of X and Y. This operation on sets—combining them—is very useful and common, and so we give it a name and a symbol.

Definition 1.11 (Union)**.** The *union* of two sets X and Y, written $X \cup Y$, is the set of all things which are elements of X, Y, or both.

$$X \cup Y = \{x \mid x \in X \vee x \in Y\}$$

Example 1.12. Since the multiplicity of elements doesn't matter, the union of two sets which have an element in common contains that element only once, e.g., $\{a, b, c\} \cup \{a, 0, 1\} = \{a, b, c, 0, 1\}$.

The union of a set and one of its subsets is just the bigger set: $\{a, b, c\} \cup \{a\} = \{a, b, c\}$.

The union of a set with the empty set is identical to the set: $\{a, b, c\} \cup \emptyset = \{a, b, c\}$.

The operation that forms the set of all elements that X and Y have in common is called their *intersection*.

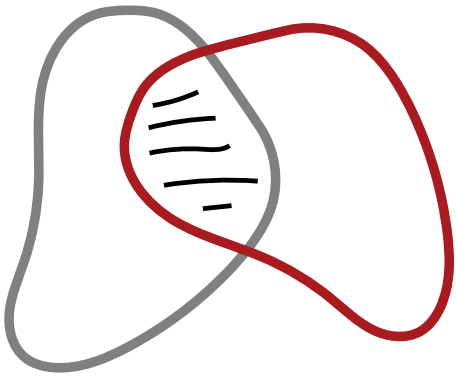


Figure 1.2: The intersection $X \cap Y$ of two sets is the set of elements they have in common.

Definition 1.13 (Intersection)**.** The *intersection* of two sets X and Y, written $X \cap Y$, is the set of all things which are elements of both X and Y.

$$X \cap Y = \{x \mid x \in X \,\&\, x \in Y\}$$

Two sets are called *disjoint* if their intersection is empty. This means they have no elements in common.

Example 1.14. If two sets have no elements in common, their intersection is empty: $\{a, b, c\} \cap \{0, 1\} = \emptyset$.

If two sets do have elements in common, their intersection is the set of all those: $\{a, b, c\} \cap \{a, b, d\} = \{a, b\}$.

The intersection of a set with one of its subsets is just the smaller set: $\{a, b, c\} \cap \{a, b\} = \{a, b\}$.

The intersection of any set with the empty set is empty: $\{a, b, c\} \cap \emptyset = \emptyset$.

We can also form the union or intersection of more than two sets. An elegant way of dealing with this in general is the following: suppose you collect all the sets you want to form the union (or intersection) of into a single set. Then we can define the union of all our original sets as the set of all objects which belong to at least one element of the set, and the intersection as the set of all objects which belong to every element of the set.

Definition 1.15. If Z is a set of sets, then $\bigcup Z$ is the set of elements of elements of Z:

$$\bigcup Z = \{x \mid x \text{ belongs to an element of } Z\}, \text{ i.e.,}$$
$$\bigcup Z = \{x \mid \text{there is a } Y \in Z \text{ so that } x \in Y\}$$

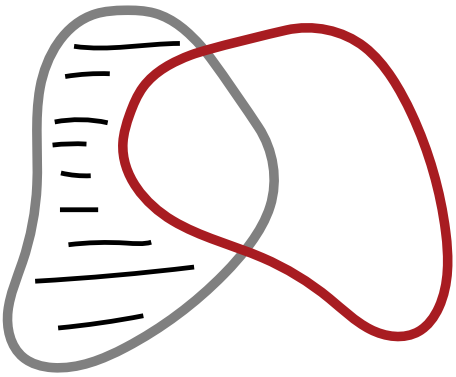


Figure 1.3: The difference $X \setminus Y$ of two sets is the set of those elements of X which are not also elements of Y.

Definition 1.16. If Z is a set of sets, then $\bigcap Z$ is the set of objects which all elements of Z have in common:

$$\bigcap Z = \{x \mid x \text{ belongs to every element of } Z\}, \text{ i.e.,}$$
$$\bigcap Z = \{x \mid \text{for all } Y \in Z, x \in Y\}$$

Example 1.17. Suppose $Z = \{\{a, b\}, \{a, d, e\}, \{a, d\}\}$. Then $\bigcup Z = \{a, b, d, e\}$ and $\bigcap Z = \{a\}$.

We could also do the same for a sequence of sets $X_1, X_2, \ldots$

$$\bigcup_i X_i = \{x \mid x \text{ belongs to one of the } X_i\}$$
$$\bigcap_i X_i = \{x \mid x \text{ belongs to every } X_i\}.$$

Definition 1.18 (Difference)**.** The *difference* $X \setminus Y$ is the set of all elements of X which are not also elements of Y, i.e.,

$$X \setminus Y = \{x \mid x \in X \text{ and } x \notin Y\}.$$

1.5 Pairs, Tuples, Cartesian Products

Sets have no order to their elements. We just think of them as an unordered collection. So if we want to represent order, we use *ordered pairs* $\langle x, y\rangle$. In an unordered pair $\{x, y\}$, the order does not matter: $\{x, y\} = \{y, x\}$. In an ordered pair, it does: if $x \neq y$, then $\langle x, y\rangle \neq \langle y, x\rangle$.

Sometimes we also want ordered sequences of more than two objects, e.g., *triples* $\langle x, y, z\rangle$, *quadruples* $\langle x, y, z, u\rangle$, and so on. In fact, we can think of

triples as special ordered pairs, where the first element is itself an ordered pair: $\langle x, y, z\rangle$ is short for $\langle\langle x, y\rangle, z\rangle$. The same is true for quadruples: $\langle x, y, z, u\rangle$ is short for $\langle\langle\langle x, y\rangle, z\rangle, u\rangle$, and so on. In general, we talk of *ordered n-tuples* $\langle x_1, \ldots, x_n\rangle$.

Definition 1.19 (Cartesian product)**.** Given sets X and Y, their *Cartesian product* $X \times Y$ is $\{\langle x, y\rangle \mid x \in X \text{ and } y \in Y\}$.

Example 1.20. If $X = \{0, 1\}$, and $Y = \{1, a, b\}$, then their product is

$$X \times Y = \{\langle 0, 1\rangle, \langle 0, a\rangle, \langle 0, b\rangle, \langle 1, 1\rangle, \langle 1, a\rangle, \langle 1, b\rangle\}.$$

Example 1.21. If X is a set, the product of X with itself, $X \times X$, is also written X^2. It is the set of *all* pairs $\langle x, y\rangle$ with $x, y \in X$. The set of all triples $\langle x, y, z\rangle$ is X^3, and so on. We can give an inductive definition:

$$X^1 = X$$
$$X^{k+1} = X^k \times X$$

Proposition 1.22. *If X has n elements and Y has m elements, then $X \times Y$ has $n \cdot m$ elements.*

Proof. For every element x in X, there are m elements of the form $\langle x, y\rangle \in X \times Y$. Let $Y_x = \{\langle x, y\rangle \mid y \in Y\}$. Since whenever $x_1 \neq x_2$, $\langle x_1, y\rangle \neq \langle x_2, y\rangle$, $Y_{x_1} \cap Y_{x_2} = \emptyset$. But if $X = \{x_1, \ldots, x_n\}$, then $Y = Y_{x_1} \cup \cdots \cup Y_{x_n}$, so has $n \cdot m$ elements.

To visualize this, arrange the elements of $X \times Y$ in a grid:

$$\begin{array}{rccccc} Y_{x_1} = & \{\langle x_1, y_1\rangle & \langle x_1, y_2\rangle & \ldots & \langle x_1, y_m\rangle\} \\ Y_{x_2} = & \{\langle x_2, y_1\rangle & \langle x_2, y_2\rangle & \ldots & \langle x_2, y_m\rangle\} \\ \vdots & & \vdots & & \\ Y_{x_n} = & \{\langle x_n, y_1\rangle & \langle x_n, y_2\rangle & \ldots & \langle x_n, y_m\rangle\} \end{array}$$

Since the x_i are all different, and the y_j are all different, no two of the pairs in this grid are the same, and there are $n \cdot m$ of them. □

Example 1.23. If X is a set, a *word* over X is any sequence of elements of X. A sequence can be thought of as an n-tuple of elements of X. For instance, if $X = \{a, b, c\}$, then the sequence "*bac*" can be thought of as the triple $\langle b, a, c\rangle$. Words, i.e., sequences of symbols, are of crucial importance in computer science, of course. By convention, we count elements of X as sequences of length 1, and $\emptyset$ as the sequence of length 0. The set of *all* words over X then is

$$X^* = \{\emptyset\} \cup X \cup X^2 \cup X^3 \cup \ldots$$

1.6 Russell's Paradox

We said that one can define sets by specifying a property that its elements share, e.g., defining the set of Richard's siblings as

$$S = \{x \mid x \text{ is a sibling of Richard}\}.$$

In the very general context of mathematics one must be careful, however: not every property lends itself to *comprehension*. Some properties do not define sets. If they did, we would run into outright contradictions. One example of such a case is Russell's Paradox.

Sets may be elements of other sets—for instance, the power set of a set X is made up of sets. And so it makes sense, of course, to ask or investigate whether a set is an element of another set. Can a set be a member of itself? Nothing about the idea of a set seems to rule this out. For instance, surely *all* sets form a collection of objects, so we should be able to collect them into a single set—the set of all sets. And it, being a set, would be an element of the set of all sets.

Russell's Paradox arises when we consider the property of not having itself as an element. The set of all sets does not have this property, but all sets we have encountered so far have it. $\mathbb{N}$ is not an element of $\mathbb{N}$, since it is a set, not a natural number. $\wp(X)$ is generally not an element of $\wp(X)$; e.g., $\wp(\mathbb{R}) \notin \wp(\mathbb{R})$ since it is a set of sets of real numbers, not a set of real numbers. What if we suppose that there is a set of all sets that do not have themselves as an element? Does

$$R = \{x \mid x \notin x\}$$

exist?

If R exists, it makes sense to ask if $R \in R$ or not—it must be either $\in R$ or $\notin R$. Suppose the former is true, i.e., $R \in R$. R was defined as the set of all sets that are not elements of themselves, and so if $R \in R$, then R does not have this defining property of R. But only sets that have this property are in R, hence, R cannot be an element of R, i.e., $R \notin R$. But R can't both be and not be an element of R, so we have a contradiction.

Since the assumption that $R \in R$ leads to a contradiction, we have $R \notin R$. But this also leads to a contradiction! For if $R \notin R$, it does have the defining property of R, and so would be an element of R just like all the other non-self-containing sets. And again, it can't both not be and be an element of R.

Chapter 2

Relations

2.1 Relations as Sets

You will no doubt remember some interesting relations between objects of some of the sets we've mentioned. For instance, numbers come with an *order relation* $<$ and from the theory of whole numbers the relation of *divisibility without remainder* (usually written $n \mid m$) may be familar. There is also the relation *is identical with* that every object bears to itself and to no other thing. But there are many more interesting relations that we'll encounter, and even more possible relations. Before we review them, we'll just point out that we can look at relations as a special sort of set. For this, first recall what a *pair* is: if a and b are two objects, we can combine them into the *ordered pair* $\langle a, b\rangle$. Note that for ordered pairs the order *does* matter, e.g, $\langle a, b\rangle \neq \langle b, a\rangle$, in contrast to unordered pairs, i.e., 2-element sets, where $\{a, b\} = \{b, a\}$.

If X and Y are sets, then the *Cartesian product* $X \times Y$ of X and Y is the set of all pairs $\langle a, b\rangle$ with $a \in X$ and $b \in Y$. In particular, $X^2 = X \times X$ is the set of all pairs from X.

Now consider a relation on a set, e.g., the $<$-relation on the set $\mathbb{N}$ of natural numbers, and consider the set of all pairs of numbers $\langle n, m\rangle$ where $n < m$, i.e.,

$$R = \{\langle n, m\rangle \mid n, m \in \mathbb{N} \text{ and } n < m\}.$$

Then there is a close connection between the number n being less than a number m and the corresponding pair $\langle n, m\rangle$ being a member of R, namely, $n < m$ if and only if $\langle n, m\rangle \in R$. In a sense we can consider the set R to *be* the $<$-relation on the set $\mathbb{N}$. In the same way we can construct a subset of $\mathbb{N}^2$ for any relation between numbers. Conversely, given any set of pairs of numbers $S \subseteq \mathbb{N}^2$, there is a corresponding relation between numbers, namely, the relationship n bears to m if and only if $\langle n, m\rangle \in S$. This justifies the following definition:

Definition 2.1 (Binary relation)**.** A *binary relation* on a set X is a subset of X^2. If $R \subseteq X^2$ is a binary relation on X and $x, y \in X$, we write Rxy (or xRy) for $\langle x, y\rangle \in R$.

Example 2.2. The set $\mathbb{N}^2$ of pairs of natural numbers can be listed in a 2-dimensional matrix like this:

$$\begin{array}{cccccc}
\langle \mathbf{0},\mathbf{0}\rangle & \langle 0,1\rangle & \langle 0,2\rangle & \langle 0,3\rangle & \dots \\
\langle 1,0\rangle & \langle \mathbf{1},\mathbf{1}\rangle & \langle 1,2\rangle & \langle 1,3\rangle & \dots \\
\langle 2,0\rangle & \langle 2,1\rangle & \langle \mathbf{2},\mathbf{2}\rangle & \langle 2,3\rangle & \dots \\
\langle 3,0\rangle & \langle 3,1\rangle & \langle 3,2\rangle & \langle \mathbf{3},\mathbf{3}\rangle & \dots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

The subset consisting of the pairs lying on the diagonal, i.e.,

$$\{\langle 0,0\rangle, \langle 1,1\rangle, \langle 2,2\rangle, \dots\},$$

is the *identity relation on* $\mathbb{N}$. (Since the identity relation is popular, let's define $\mathrm{Id}_X = \{\langle x, x\rangle \mid x \in X\}$ for any set X.) The subset of all pairs lying above the diagonal, i.e.,

$$L = \{\langle 0,1\rangle, \langle 0,2\rangle, \dots, \langle 1,2\rangle, \langle 1,3\rangle, \dots, \langle 2,3\rangle, \langle 2,4\rangle, \dots\},$$

is the *less than* relation, i.e., Lnm iff $n < m$. The subset of pairs below the diagonal, i.e.,

$$G = \{\langle 1,0\rangle, \langle 2,0\rangle, \langle 2,1\rangle, \langle 3,0\rangle, \langle 3,1\rangle, \langle 3,2\rangle, \dots\},$$

is the *greater than* relation, i.e., Gnm iff $n > m$. The union of L with I, $K = L \cup I$, is the *less than or equal to* relation: Knm iff $n \leq m$. Similarly, $H = G \cup I$ is the *greater than or equal to relation*. L, G, K, and H are special kinds of relations called *orders*. L and G have the property that no number bears L or G to itself (i.e., for all n, neither Lnn nor Gnn). Relations with this property are called *irreflexive*, and, if they also happen to be orders, they are called *strict orders.*

Although orders and identity are important and natural relations, it should be emphasized that according to our definition *any* subset of X^2 is a relation on X, regardless of how unnatural or contrived it seems. In particular, $\emptyset$ is a relation on any set (the *empty relation*, which no pair of elements bears), and X^2 itself is a relation on X as well (one which every pair bears), called the *universal relation*. But also something like $E = \{\langle n, m\rangle \mid n > 5 \text{ or } m \times n \geq 34\}$ counts as a relation.

2.2 Special Properties of Relations

Some kinds of relations turn out to be so common that they have been given special names. For instance, $\leq$ and $\subseteq$ both relate their respective domains

(say, $\mathbb{N}$ in the case of $\leq$ and $\wp(X)$ in the case of $\subseteq$) in similar ways. To get at exactly how these relations are similar, and how they differ, we categorize them according to some special properties that relations can have. It turns out that (combinations of) some of these special properties are especially important: orders and equivalence relations.

Definition 2.3 (Reflexivity)**.** A relation $R \subseteq X^2$ is *reflexive* iff, for every $x \in X$, Rxx.

Definition 2.4 (Transitivity)**.** A relation $R \subseteq X^2$ is *transitive* iff, whenever Rxy and Ryz, then also Rxz.

Definition 2.5 (Symmetry)**.** A relation $R \subseteq X^2$ is *symmetric* iff, whenever Rxy, then also Ryx.

Definition 2.6 (Anti-symmetry)**.** A relation $R \subseteq X^2$ is *anti-symmetric* iff, whenever both Rxy and Ryx, then $x = y$ (or, in other words: if $x \neq y$ then either $\sim Rxy$ or $\sim Ryx$).

In a symmetric relation, Rxy and Ryx always hold together, or neither holds. In an anti-symmetric relation, the only way for Rxy and Ryx to hold together is if $x = y$. Note that this does not *require* that Rxy and Ryx holds when $x = y$, only that it isn't ruled out. So an anti-symmetric relation can be reflexive, but it is not the case that every anti-symmetric relation is reflexive. Also note that being anti-symmetric and merely not being symmetric are different conditions. In fact, a relation can be both symmetric and anti-symmetric at the same time (e.g., the identity relation is).

Definition 2.7 (Connectivity)**.** A relation $R \subseteq X^2$ is *connected* if for all $x, y \in X$, if $x \neq y$, then either Rxy or Ryx.

Definition 2.8 (Partial order)**.** A relation $R \subseteq X^2$ that is reflexive, transitive, and anti-symmetric is called a *partial order*.

Definition 2.9 (Linear order)**.** A partial order that is also connected is called a *linear order*.

Definition 2.10 (Equivalence relation)**.** A relation $R \subseteq X^2$ that is reflexive, symmetric, and transitive is called an *equivalence relation*.

2.3 Orders

Very often we are interested in comparisons between objects, where one object may be less or equal or greater than another in a certain respect. Size is the most obvious example of such a comparative relation, or *order*. But not all such relations are alike in all their properties. For instance, some comparative relations require any two objects to be comparable, others don't. (If they do,

we call them *linear* or *total*.) Some include identity (like $\leq$) and some exclude it (like $<$). Let's get some order into all this.

Definition 2.11 (Preorder)**.** A relation which is both reflexive and transitive is called a *preorder.*

Definition 2.12 (Partial order)**.** A preorder which is also anti-symmetric is called a *partial order*.

Definition 2.13 (Linear order)**.** A partial order which is also connected is called a *total order* or *linear order.*

Example 2.14. Every linear order is also a partial order, and every partial order is also a preorder, but the converses don't hold. The universal relation on X is a preorder, since it is reflexive and transitive. But, if X has more than one element, the universal relation is not anti-symmetric, and so not a partial order. For a somewhat less silly example, consider the *no longer than* relation $\preccurlyeq$ on $\mathbb{B}^*$: $x \preccurlyeq y$ iff $\mathrm{len}(x) \leq \mathrm{len}(y)$. This is a preorder (reflexive and transitive), and even connected, but not a partial order, since it is not anti-symmetric. For instance, $01 \preccurlyeq 10$ and $10 \preccurlyeq 01$, but $01 \neq 10$.

The relation of *divisibility without remainder* gives us an example of a partial order which isn't a linear order: for integers n, m, we say n (evenly) divides m, in symbols: $n \mid m$, if there is some k so that $m = kn$. On $\mathbb{N}$, this is a partial order, but not a linear order: for instance, $2 \nmid 3$ and also $3 \nmid 2$. Considered as a relation on $\mathbb{Z}$, divisibility is only a preorder since anti-symmetry fails: $1 \mid -1$ and $-1 \mid 1$ but $1 \neq -1$. Another important partial order is the relation $\subseteq$ on a set of sets.

Notice that the examples L and G from Example 2.2, although we said there that they were called "strict orders," are not linear orders even though they are connected (they are not reflexive). But there is a close connection, as we will see momentarily.

Definition 2.15 (Irreflexivity)**.** A relation R on X is called *irreflexive* if, for all $x \in X$, $\sim Rxx$.

Definition 2.16 (Asymmetry)**.** A relation R on X is called *asymmetric* if for no pair $x, y \in X$ we have Rxy and Ryx.

Definition 2.17 (Strict order)**.** A *strict order* is a relation which is irreflexive, asymmetric, and transitive.

Definition 2.18 (Strict linear order)**.** A strict order which is also connected is called a *strict linear order.*

A strict order on X can be turned into a partial order by adding the diagonal Id_X, i.e., adding all the pairs $\langle x, x \rangle$. (This is called the *reflexive closure* of R.) Conversely, starting from a partial order, one can get a strict order by removing Id_X.

Proposition 2.19. *1. If R is a strict (linear) order on X, then $R^+ = R \cup \mathrm{Id}_X$ is a partial order (linear order).*

2. If R is a partial order (linear order) on X, then $R^- = R \setminus \mathrm{Id}_X$ is a strict (linear) order.

Proof. 1. Suppose R is a strict order, i.e., $R \subseteq X^2$ and R is irreflexive, asymmetric, and transitive. Let $R^+ = R \cup \mathrm{Id}_X$. We have to show that R^+ is reflexive, antisymmetric, and transitive.

R^+ is clearly reflexive, since for all $x \in X$, $\langle x, x\rangle \in \mathrm{Id}_X \subseteq R^+$.

To show R^+ is antisymmetric, suppose R^+xy and R^+yx, i.e., $\langle x, y\rangle$ and $\langle y, x\rangle \in R^+$, and $x \neq y$. Since $\langle x, y\rangle \in R \cup \mathrm{Id}_X$, but $\langle x, y\rangle \notin \mathrm{Id}_X$, we must have $\langle x, y\rangle \in R$, i.e., Rxy. Similarly we get that Ryx. But this contradicts the assumption that R is asymmetric.

Now suppose that R^+xy and R^+yz. If both $\langle x, y\rangle \in R$ and $\langle y, z\rangle \in R$, it follows that $\langle x, z\rangle \in R$ since R is transitive. Otherwise, either $\langle x, y\rangle \in \mathrm{Id}_X$, i.e., $x = y$, or $\langle y, z\rangle \in \mathrm{Id}_X$, i.e., $y = z$. In the first case, we have that R^+yz by assumption, $x = y$, hence R^+xz. Similarly in the second case. In either case, R^+xz, thus, R^+ is also transitive.

If R is connected, then for all $x \neq y$, either Rxy or Ryx, i.e., either $\langle x, y\rangle \in R$ or $\langle y, x\rangle \in R$. Since $R \subseteq R^+$, this remains true of R^+, so R^+ is connected as well.

2. Exercise.

□

Example 2.20. $\leq$ is the linear order corresponding to the strict linear order $<$. $\subseteq$ is the partial order corresponding to the strict order $\subsetneq$.

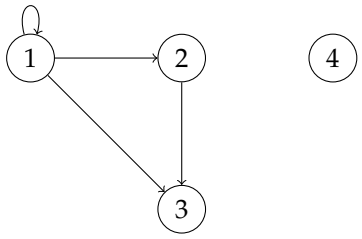
2.4 Graphs

A *graph* is a diagram in which points—called "nodes" or "vertices" (plural of "vertex")—are connected by edges. Graphs are a ubiquitous tool in discrete mathematics and in computer science. They are incredibly useful for representing, and visualizing, relationships and structures, from concrete things like networks of various kinds to abstract structures such as the possible outcomes of decisions. There are many different kinds of graphs in the literature which differ, e.g., according to whether the edges are directed or not, have labels or not, whether there can be edges from a node to the same node, multiple edges between the same nodes, etc. *Directed graphs* have a special connection to relations.

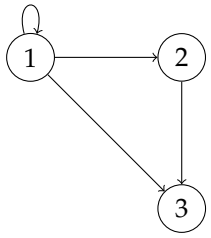
Definition 2.21 (Directed graph)**.** A *directed graph* $G = \langle V, E\rangle$ is a set of *vertices* V and a set of *edges* $E \subseteq V^2$.

According to our definition, a graph just is a set together with a relation on that set. Of course, when talking about graphs, it's only natural to expect that they are graphically represented: we can draw a graph by connecting two vertices v_1 and v_2 by an arrow iff $\langle v_1, v_2 \rangle \in E$. The only difference between a relation by itself and a graph is that a graph specifies the set of vertices, i.e., a graph may have isolated vertices. The important point, however, is that every relation R on a set X can be seen as a directed graph $\langle X, R \rangle$, and conversely, a directed graph $\langle V, E \rangle$ can be seen as a relation $E \subseteq V^2$ with the set V explicitly specified.

Example 2.22. The graph $\langle V, E \rangle$ with $V = \{1, 2, 3, 4\}$ and $E = \{\langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 3 \rangle\}$ looks like this:



This is a different graph than $\langle V', E \rangle$ with $V' = \{1, 2, 3\}$, which looks like this:



2.5 Operations on Relations

It is often useful to modify or combine relations. We've already used the union of relations above (which is just the union of two relations considered as sets of pairs). Here are some other ways:

Definition 2.23. Let $R, S \subseteq X^2$ be relations and Y a set.

1. The *inverse* R^{-1} of R is $R^{-1} = \{\langle y, x \rangle \mid \langle x, y \rangle \in R\}$.
2. The *relative product* $R \mid S$ of R and S is

$$(R \mid S) = \{\langle x, z \rangle : \text{for some } y, Rxy \text{ and } Syz\}$$

3. The *restriction* $R \restriction Y$ of R to Y is $R \cap Y^2$
4. The *application* $R[Y]$ of R to Y is

$$R[Y] = \{y : \text{for some } x \in Y, Rxy\}$$

Example 2.24. Let $S \subseteq \mathbb{Z}^2$ be the successor relation on $\mathbb{Z}$, i.e., the set of pairs $\langle x, y \rangle$ where $x + 1 = y$, for $x, y \in \mathbb{Z}$. Sxy holds iff y is the successor of x.

1. The inverse S^{-1} of S is the predecessor relation, i.e., $S^{-1}xy$ iff $x - 1 = y$.
2. The relative product $S \mid S$ is the relation x bears to y if $x + 2 = y$.
3. The restriction of S to $\mathbb{N}$ is the successor relation on $\mathbb{N}$.
4. The application of S to a set, e.g., $S[\{1, 2, 3\}]$ is $\{2, 3, 4\}$.

Definition 2.25 (Transitive closure)**.** The *transitive closure* R^+ of a relation $R \subseteq X^2$ is $R^+ = \bigcup_{i=1}^{\infty} R^i$ where $R^1 = R$ and $R^{i+1} = R^i \mid R$.

The *reflexive transitive closure* of R is $R^* = R^+ \cup \mathrm{Id}_X$.

Example 2.26. Take the successor relation $S \subseteq \mathbb{Z}^2$. S^2xy iff $x + 2 = y$, S^3xy iff $x + 3 = y$, etc. So R^*xy iff for some $i \geq 1$, $x + i = y$. In other words, S^+xy iff $x < y$ (and R^*xy iff $x \leq y$).

Chapter 3

Functions

3.1 Basics

A *function* is a mapping which pairs each object of a given set with a single partner in another set. For instance, the operation of adding 1 defines a function: each number n is paired with a unique number $n + 1$. More generally, functions may take pairs, triples, etc., of inputs and returns some kind of output. Many functions are familiar to us from basic arithmetic. For instance, addition and multiplication are functions. They take in two numbers and return a third. In this mathematical, abstract sense, a function is a *black box*: what matters is only what output is paired with what input, not the method for calculating the output.

Definition 3.1 (Function)**.** A *function* $f\colon X \to Y$ is a mapping of each element of X to an element of Y. We call X the *domain* of f and Y the *codomain* of f. The elements of X are called inputs or *arguments* of f, and the element of Y that is paired with an argument x by f is called the *value of* f for argument x, written $f(x)$.

The *range* $\operatorname{ran}(f)$ of f is the subset of the codomain consisting of the values of f for some argument; $\operatorname{ran}(f) = \{f(x) \mid x \in X\}$.

Example 3.2. Multiplication takes pairs of natural numbers as inputs and maps them to natural numbers as outputs, so goes from $\mathbb{N} \times \mathbb{N}$ (the domain) to $\mathbb{N}$ (the codomain). As it turns out, the range is also $\mathbb{N}$, since every $n \in \mathbb{N}$ is $n \times 1$.

Multiplication is a function because it pairs each input—each pair of natural numbers—with a single output: $\times\colon \mathbb{N}^2 \to \mathbb{N}$. By contrast, the square root operation applied to the domain $\mathbb{N}$ is not functional, since each positive integer n has two square roots: $\sqrt{n}$ and $-\sqrt{n}$. We can make it functional by only returning the positive square root: $\sqrt{}\colon \mathbb{N} \to \mathbb{R}$. The relation that pairs each

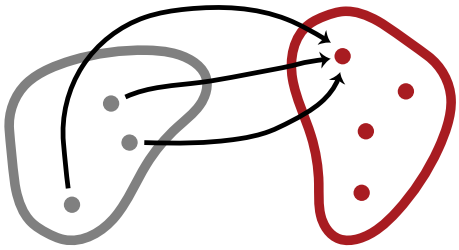


Figure 3.1: A function is a mapping of each element of one set to an element of another. An arrow points from an argument in the domain to the corresponding value in the codomain.

student in a class with their final grade is a function—no student can get two different final grades in the same class. The relation that pairs each student in a class with their parents is not a function—generally each student will have at least two parents.

We can define functions by specifying in some precise way what the value of the function is for every possible argment. Different ways of doing this are by giving a formula, describing a method for computing the value, or listing the values for each argument. However functions are defined, we must make sure that for each argment we specify one, and only one, value.

Example 3.3. Let $f\colon \mathbb{N} \to \mathbb{N}$ be defined such that $f(x) = x + 1$. This is a definition that specifies f as a function which takes in natural numbers and outputs natural numbers. It tells us that, given a natural number x, f will output its successor $x + 1$. In this case, the codomain $\mathbb{N}$ is not the range of f, since the natural number 0 is not the successor of any natural number. The range of f is the set of all positive integers, $\mathbb{Z}^+$.

Example 3.4. Let $g\colon \mathbb{N} \to \mathbb{N}$ be defined such that $g(x) = x + 2 - 1$. This tells us that g is a function which takes in natural numbers and outputs natural numbers. Given a natural number n, g will output the predecessor of the successor of the successor of x, i.e., $x + 1$. Despite their different definitions, g and f are the same function.

Functions f and g defined above are the same because for any natural number x, $x + 2 - 1 = x + 1$. f and g pair each natural number with the same output. The definitions for f and g specify the same mapping by means of different equations, and so count as the same function.

Example 3.5. We can also define functions by cases. For instance, we could define $h\colon \mathbb{N} \to \mathbb{N}$ by

$$h(x) = \begin{cases} \frac{x}{2} & \text{if } x \text{ is even} \\ \frac{x+1}{2} & \text{if } x \text{ is odd.} \end{cases}$$

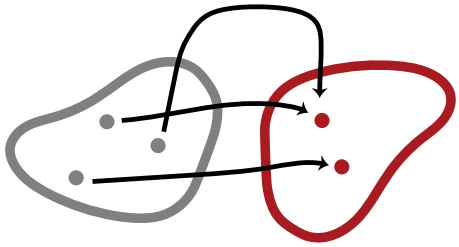


Figure 3.2: A surjective function has every element of the codomain as a value.

Figure 3.3: An injective function never maps two different arguments to the same value.

Since every natural number is either even or odd, the output of this function will always be a natural number. Just remember that if you define a function by cases, every possible input must fall into exactly one case. In some cases, this will require a a proof that the cases are exhaustive and exclusive.

3.2 Kinds of Functions

Definition 3.6 (Surjective function)**.** A function $f\colon X \to Y$ is *surjective* iff Y is also the range of f, i.e., for every $y \in Y$ there is at least one $x \in X$ such that $f(x) = y$.

If you want to show that a function is surjective, then you need to show that every object in the codomain is the output of the function given some input or other.

Definition 3.7 (Injective function)**.** A function $f\colon X \to Y$ is *injective* iff for each $y \in Y$ there is at most one $x \in X$ such that $f(x) = y$.

Any function pairs each possible input with a unique output. An injective function has a unique input for each possible output. If you want to show that a function f is injective, you need to show that for any elements x and x' of the domain, if $f(x) = f(x')$, then $x = x'$.

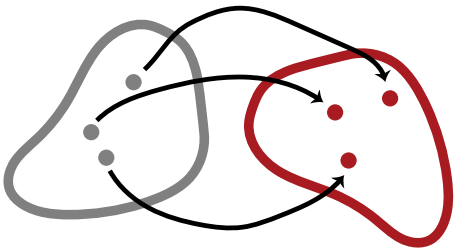


Figure 3.4: A bijective function uniquely pairs the elements of the codomain with those of the domain.

An example of a function which is neither injective, nor surjective, is the constant function $f\colon \mathbb{N} \to \mathbb{N}$ where $f(x) = 1$.

An example of a function which is both injective and surjective is the identity function $f\colon \mathbb{N} \to \mathbb{N}$ where $f(x) = x$.

The successor function $f\colon \mathbb{N} \to \mathbb{N}$ where $f(x) = x + 1$ is injective, but not surjective.

The function

$$f(x) = \begin{cases} \frac{x}{2} & \text{if } x \text{ is even} \\ \frac{x+1}{2} & \text{if } x \text{ is odd.} \end{cases}$$

is surjective, but not injective.

Definition 3.8 (Bijection)**.** A function $f\colon X \to Y$ is *bijective* iff it is both surjective and injective. We call such a function a *bijection* from X to Y (or between X and Y).

3.3 Inverses of Functions

One obvious question about functions is whether a given mapping can be "reversed." For instance, the successor function $f(x) = x + 1$ can be reversed in the sense that the function $g(y) = y - 1$ "undoes" what f does. But we must be careful: While the definition of g defines a function $\mathbb{Z} \to \mathbb{Z}$, it does not define a function $\mathbb{N} \to \mathbb{N}$ ($g(0) \notin \mathbb{N}$). So even in simple cases, it is not quite obvious if functions can be reversed, and that it may depend on the domain and codomain. Let's give a precise definition.

Definition 3.9. A function $g\colon Y \to X$ is an *inverse* of a function $f\colon X \to Y$ if $f(g(y)) = y$ and $g(f(x)) = x$ for all $x \in X$ and $y \in Y$.

When do functions have inverses? A good candidate for an inverse of $f\colon X \to Y$ is $g\colon Y \to X$ "defined by"

$$g(y) = \text{"the" } x \text{ such that } f(x) = y.$$

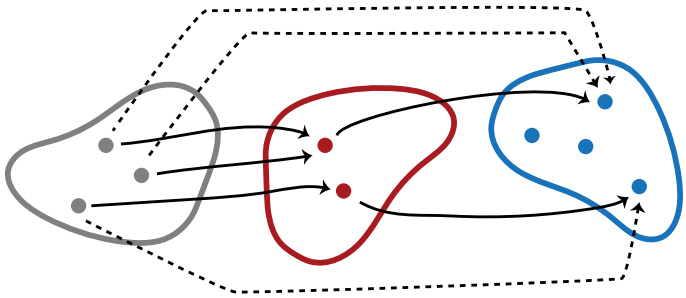


Figure 3.5: The composition $g \circ f$ of two functions f and g.

The scare quotes around "defined by" suggest that this is not a definition. At least, it is not in general. For in order for this definition to specify a function, there has to be one and only one x such that $f(x) = y$—the output of g has to be uniquely specified. Moreover, it has to be specified for every $y \in Y$. If there are x_1 and $x_2 \in X$ with $x_1 \neq x_2$ but $f(x_1) = f(x_2)$, then $g(y)$ would not be uniquely specified for $y = f(x_1) = f(x_2)$. And if there is no x at all such that $f(x) = y$, then $g(y)$ is not specified at all. In other words, for g to be defined, f has to be injective and surjective.

Proposition 3.10. *If $f\colon X \to Y$ is bijective, f has a unique inverse $f^{-1}\colon Y \to X$.*

Proof. Exercise. □

3.4 Composition of Functions

We have already seen that the inverse f^{-1} of a bijective function f is itself a function. It is also possible to compose functions f and g to define a new function by first applying f and then g. Of course, this is only possible if the ranges and domains match, i.e., the range of f must be a subset of the domain of g.

Definition 3.11 (Composition)**.** Let $f\colon X \to Y$ and $g\colon Y \to Z$. The *composition* of f with g is the function $(g \circ f)\colon X \to Z$, where $(g \circ f)(x) = g(f(x))$.

The function $(g \circ f)\colon X \to Z$ pairs each member of X with a member of Z. We specify which member of Z a member of X is paired with as follows—given an input $x \in X$, first apply the function f to x, which will output some $y \in Y$. Then apply the function g to y, which will output some $z \in Z$.

Example 3.12. Consider the functions $f(x) = x + 1$, and $g(x) = 2x$. What function do you get when you compose these two? $(g \circ f)(x) = g(f(x))$. So that means for every natural number you give this function, you first add one,

and then you multiply the result by two. So their composition is $(g \circ f)(x) = 2(x+1)$.

3.5 Isomorphism

An *isomorphism* is a bijection that preserves the structure of the sets it relates, where structure is a matter of the relationships that obtain between the elements of the sets. Consider the following two sets $X = \{1, 2, 3\}$ and $Y = \{4, 5, 6\}$. These sets are both structured by the relations successor, less than, and greater than. An isomorphism between the two sets is a bijection that preserves those structures. So a bijective function $f : X \to Y$ is an isomorphism if, $i < j$ iff $f(i) < f(j)$, $i > j$ iff $f(i) > f(j)$, and j is the successor of i iff $f(j)$ is the successor of $f(i)$.

Definition 3.13 (Isomorphism)**.** Let U be the pair $\langle X, R\rangle$ and V be the pair $\langle Y, S\rangle$ such that X and Y are sets and R and S are relations on X and Y respectively. A bijection f from X to Y is an *isomorphism* from U to V iff it preserves the relational structure, that is, for any x_1 and x_2 in X, $\langle x_1, x_2\rangle \in R$ iff $\langle f(x_1), f(x_2)\rangle \in S$.

Example 3.14. Consider the following two sets $X = \{1, 2, 3\}$ and $Y = \{4, 5, 6\}$, and the relations less than and greater than. The function $f : X \to Y$ where $f(x) = 7 - x$ is an isomorphism between $\langle X, <\rangle$ and $\langle Y, >\rangle$.

3.6 Partial Functions

It is sometimes useful to relax the definition of function so that it is not required that the output of the function is defined for all possible inputs. Such mappings are called *partial functions*.

Definition 3.15. A *partial function* $f : X \nrightarrow Y$ is a mapping which assigns to every element of X at most one element of Y. If f assigns an element of Y to $x \in X$, we say $f(x)$ is *defined*, and otherwise *undefined*. If $f(x)$ is defined, we write $f(x)\downarrow$, otherwise $f(x)\uparrow$. The *domain* of a partial function f is the subset of X where it is defined, i.e., $\mathrm{dom}(f) = \{x \mid f(x)\downarrow\}$.

Example 3.16. Every function $f : X \to Y$ is also a partial function. Partial functions that are defined everywhere on X—i.e., what we so far have simply called a function—are also called *total* functions.

Example 3.17. The partial function $f : \mathbb{R} \nrightarrow \mathbb{R}$ given by $f(x) = 1/x$ is undefined for $x = 0$, and defined everywhere else.

3.7 Functions and Relations

A function which maps elements of X to elements of Y obviously defines a relation between X and Y, namely the relation which holds between x and y iff $f(x) = y$. In fact, we might even—if we are interested in reducing the building blocks of mathematics for instance—*identify* the function f with this relation, i.e., with a set of pairs. This then raises the question: which relations define functions in this way?

Definition 3.18 (Graph of a function)**.** Let $f : X \pfun Y$ be a partial function. The *graph* of f is the relation $R_f \subseteq X \times Y$ defined by

$$R_f = \{\langle x, y\rangle \mid f(x) = y\}.$$

Proposition 3.19. *Suppose $R \subseteq X \times Y$ has the property that whenever Rxy and Rxy' then $y = y'$. Then R is the graph of the partial function $f : X \pfun Y$ defined by: if there is a y such that Rxy, then $f(x) = y$, otherwise $f(x) \uparrow$. If R is also* serial*, i.e., for each $x \in X$ there is a $y \in Y$ such that Rxy, then f is total.*

Proof. Suppose there is a y such that Rxy. If there were another $y' \neq y$ such that Rxy', the condition on R would be violated. Hence, if there is a y such that Rxy, that y is unique, and so f is well-defined. Obviously, $R_f = R$ and f is total if R is serial. □

Chapter 4

The Size of Sets

4.1 Introduction

When Georg Cantor developed set theory in the 1870s, his interest was in part to make palatable the idea of an infinite collection—an actual infinity, as the medievals would say. Key to this rehabilitation of the notion of the infinite was a way to assign sizes—"cardinalities"—to sets. The cardinality of a finite set is just a natural number, e.g., $\emptyset$ has cardinality 0, and a set containing five things has cardinality 5. But what about infinite sets? Do they all have the same cardinality, ∞? It turns out, they do not.

The first important idea here is that of an enumeration. We can list every finite set by listing all its elements. For some infinite sets, we can also list all their elements if we allow the list itself to be infinite. Such sets are called countable. Cantor's surprising result was that some infinite sets are not countable.

4.2 Countable Sets

One way of specifying a finite set is by listing its elements. But conversely, since there are only finitely many elements in a set, every finite set can be enumerated. By this we mean: its elements can be put into a list (a list with a beginning, where each element of the list other than the first has a unique predecessor). Some infinite sets can also be enumerated, such as the set of positive integers.

Definition 4.1 (Enumeration)**.** Informally, an *enumeration* of a set X is a list (possibly infinite) of elements of X such that every element of X appears on the list at some finite position. If X has an enumeration, then X is said to be *countable*. If X is countable and infinite, we say X is countably infinite.

A couple of points about enumerations:

1. We count as enumerations only lists which have a beginning and in which every element other than the first has a single element immediately preceding it. In other words, there are only finitely many elements between the first element of the list and any other element. In particular, this means that every element of an enumeration has a finite position: the first element has position 1, the second position 2, etc.

2. We can have different enumerations of the same set X which differ by the order in which the elements appear: 4, 1, 25, 16, 9 enumerates the (set of the) first five square numbers just as well as 1, 4, 9, 16, 25 does.

3. Redundant enumerations are still enumerations: 1, 1, 2, 2, 3, 3, ... enumerates the same set as 1, 2, 3, ... does.

4. Order and redundancy *do* matter when we specify an enumeration: we can enumerate the positive integers beginning with 1, 2, 3, 1, ..., but the pattern is easier to see when enumerated in the standard way as 1, 2, 3, 4, ...

5. Enumerations must have a beginning: ..., 3, 2, 1 is not an enumeration of the natural numbers because it has no first element. To see how this follows from the informal definition, ask yourself, "at what position in the list does the number 76 appear?"

6. The following is not an enumeration of the positive integers: 1, 3, 5, ..., 2, 4, 6, ... The problem is that the even numbers occur at places $\infty + 1$, $\infty + 2$, $\infty + 3$, rather than at finite positions.

7. Lists may be gappy: 2, $-$, 4, $-$, 6, $-$, ... enumerates the even positive integers.

8. The empty set is enumerable: it is enumerated by the empty list!

Proposition 4.2. *If X has an enumeration, it has an enumeration without gaps or repetitions.*

Proof. Suppose X has an enumeration $x_1, x_2, \ldots$ in which each x_i is an element of X or a gap. We can remove repetitions from an enumeration by replacing repeated elements by gaps. For instance, we can turn the enumeration into a new one in which x'_i is x_i if x_i is an element of X that is not among $x_1, \ldots, x_{i-1}$ or is $-$ if it is. We can remove gaps by closing up the elements in the list. To make precise what "closing up" amounts to is a bit difficult to describe. Roughly, it means that we can generate a new enumeration $x''_1, x''_2, \ldots$, where each x''_i is the first element in the enumeration $x'_1, x'_2, \ldots$ after x''_{i-1} (if there is one). □

The last argument shows that in order to get a good handle on enumerations and countable sets and to prove things about them, we need a more precise definition. The following provides it.

Definition 4.3 (Enumeration)**.** An *enumeration* of a set X is any surjective function $f\colon \mathbb{Z}^+ \to X$.

Let's convince ourselves that the formal definition and the informal definition using a possibly gappy, possibly infinite list are equivalent. A surjective function (partial or total) from $\mathbb{Z}^+$ to a set X enumerates X. Such a function determines an enumeration as defined informally above: the list $f(1)$, $f(2)$, $f(3)$, ... Since f is surjective, every element of X is guaranteed to be the value of $f(n)$ for some $n \in \mathbb{Z}^+$. Hence, every element of X appears at some finite position in the list. Since the function may not be injective, the list may be redundant, but that is acceptable (as noted above).

On the other hand, given a list that enumerates all elements of X, we can define a surjective function $f\colon \mathbb{Z}^+ \to X$ by letting $f(n)$ be the nth element of the list that is not a gap, or the final element of the list if there is no nth element. There is one case in which this does not produce a surjective function: if X is empty, and hence the list is empty. So, every non-empty list determines a surjective function $f\colon \mathbb{Z}^+ \to X$.

Definition 4.4. A set X is countable iff it is empty or has an enumeration.

Example 4.5. A function enumerating the positive integers ($\mathbb{Z}^+$) is simply the identity function given by $f(n) = n$. A function enumerating the natural numbers $\mathbb{N}$ is the function $g(n) = n - 1$.

Example 4.6. The functions $f\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ and $g\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ given by

$$f(n) = 2n \text{ and}$$
$$g(n) = 2n + 1$$

enumerate the even positive integers and the odd positive integers, respectively. However, neither function is an enumeration of $\mathbb{Z}^+$, since neither is surjective.

Example 4.7. The function $f(n) = (-1)^n \lceil \frac{(n-1)}{2} \rceil$ (where $\lceil x \rceil$ denotes the *ceiling* function, which rounds x up to the nearest integer) enumerates the set of integers $\mathbb{Z}$. Notice how f generates the values of $\mathbb{Z}$ by "hopping" back and forth between positive and negative integers:

$$\begin{array}{cccccccc} f(1) & f(2) & f(3) & f(4) & f(5) & f(6) & f(7) & \dots \\ -\lceil \frac{0}{2} \rceil & \lceil \frac{1}{2} \rceil & -\lceil \frac{2}{2} \rceil & \lceil \frac{3}{2} \rceil & -\lceil \frac{4}{2} \rceil & \lceil \frac{5}{2} \rceil & -\lceil \frac{6}{2} \rceil & \dots \\ 0 & 1 & -1 & 2 & -2 & 3 & \dots & \end{array}$$

You can also think of f as defined by cases as follows:

$$f(n) = \begin{cases} 0 & \text{if } n = 1 \\ n/2 & \text{if } n \text{ is even} \\ -(n-1)/2 & \text{if } n \text{ is odd and } > 1 \end{cases}$$

That is fine for "easy" sets. What about the set of, say, pairs of natural numbers?

$$\mathbb{Z}^+ \times \mathbb{Z}^+ = \{\langle n, m\rangle \mid n, m \in \mathbb{Z}^+\}$$

We can organize the pairs of positive integers in an *array*, such as the following:

	1	**2**	**3**	**4**	...
1	$\langle 1,1\rangle$	$\langle 1,2\rangle$	$\langle 1,3\rangle$	$\langle 1,4\rangle$	...
2	$\langle 2,1\rangle$	$\langle 2,2\rangle$	$\langle 2,3\rangle$	$\langle 2,4\rangle$	...
3	$\langle 3,1\rangle$	$\langle 3,2\rangle$	$\langle 3,3\rangle$	$\langle 3,4\rangle$	...
4	$\langle 4,1\rangle$	$\langle 4,2\rangle$	$\langle 4,3\rangle$	$\langle 4,4\rangle$	...
⋮	⋮	⋮	⋮	⋮	⋱

Clearly, every ordered pair in $\mathbb{Z}^+ \times \mathbb{Z}^+$ will appear exactly once in the array. In particular, $\langle n, m\rangle$ will appear in the nth column and mth row. But how do we organize the elements of such an array into a one-way list? The pattern in the array below demonstrates one way to do this:

	1	2	4	7	...
	3	5	8	...	...
	6	9	...	...	...
	10	...	...	...	...
	⋮	⋮	⋮	⋮	⋱

This pattern is called *Cantor's zig-zag method*. Other patterns are perfectly permissible, as long as they "zig-zag" through every cell of the array. By Cantor's zig-zag method, the enumeration for $\mathbb{Z}^+ \times \mathbb{Z}^+$ according to this scheme would be:

$$\langle 1,1\rangle, \langle 1,2\rangle, \langle 2,1\rangle, \langle 1,3\rangle, \langle 2,2\rangle, \langle 3,1\rangle, \langle 1,4\rangle, \langle 2,3\rangle, \langle 3,2\rangle, \langle 4,1\rangle, \ldots$$

What ought we do about enumerating, say, the set of ordered triples of positive integers?

$$\mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+ = \{\langle n, m, k\rangle \mid n, m, k \in \mathbb{Z}^+\}$$

We can think of $\mathbb{Z}^+ \times \mathbb{Z}^+ \times \mathbb{Z}^+$ as the Cartesian product of $\mathbb{Z}^+ \times \mathbb{Z}^+$ and $\mathbb{Z}^+$, that is,

$$(\mathbb{Z}^+)^3 = (\mathbb{Z}^+ \times \mathbb{Z}^+) \times \mathbb{Z}^+ = \{\langle\langle n, m\rangle, k\rangle \mid \langle n, m\rangle \in \mathbb{Z}^+ \times \mathbb{Z}^+, k \in \mathbb{Z}^+\}$$

and thus we can enumerate $(\mathbb{Z}^+)^3$ with an array by labelling one axis with the enumeration of $\mathbb{Z}^+$, and the other axis with the enumeration of $(\mathbb{Z}^+)^2$:

	1	**2**	**3**	**4**	...
$\langle \mathbf{1}, \mathbf{1} \rangle$	$\langle 1,1,1 \rangle$	$\langle 1,1,2 \rangle$	$\langle 1,1,3 \rangle$	$\langle 1,1,4 \rangle$	...
$\langle \mathbf{1}, \mathbf{2} \rangle$	$\langle 1,2,1 \rangle$	$\langle 1,2,2 \rangle$	$\langle 1,2,3 \rangle$	$\langle 1,2,4 \rangle$	...
$\langle \mathbf{2}, \mathbf{1} \rangle$	$\langle 2,1,1 \rangle$	$\langle 2,1,2 \rangle$	$\langle 2,1,3 \rangle$	$\langle 2,1,4 \rangle$	...
$\langle \mathbf{1}, \mathbf{3} \rangle$	$\langle 1,3,1 \rangle$	$\langle 1,3,2 \rangle$	$\langle 1,3,3 \rangle$	$\langle 1,3,4 \rangle$	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Thus, by using a method like Cantor's zig-zag method, we may similarly obtain an enumeration of $(\mathbb{Z}^+)^3$.

4.3 Uncountable Sets

Some sets, such as the set $\mathbb{Z}^+$ of positive integers, are infinite. So far we've seen examples of infinite sets which were all countable. However, there are also infinite sets which do not have this property. Such sets are called *uncountable*.

First of all, it is perhaps already surprising that there are uncountable sets. For any countable set X there is a surjective function $f\colon \mathbb{Z}^+ \to X$. If a set is uncountable there is no such function. That is, no function mapping the infinitely many elements of $\mathbb{Z}^+$ to X can exhaust all of X. So there are "more" elements of X than the infinitely many positive integers.

How would one prove that a set is uncountable? You have to show that no such surjective function can exist. Equivalently, you have to show that the elements of X cannot be enumerated in a one way infinite list. The best way to do this is to show that every list of elements of X must leave at least one element out; or that no function $f\colon \mathbb{Z}^+ \to X$ can be surjective. We can do this using Cantor's *diagonal method*. Given a list of elements of X, say, x_1, x_2, ..., we construct another element of X which, by its construction, cannot possibly be on that list.

Our first example is the set $\mathbb{B}^\omega$ of all infinite, non-gappy sequences of 0's and 1's.

Theorem 4.8. *$\mathbb{B}^\omega$ is uncountable.*

Proof. Suppose, by way of contradiction, that $\mathbb{B}^\omega$ is countable, i.e., suppose that there is a list s_1, s_2, s_3, s_4, ... of all elements of $\mathbb{B}^\omega$. Each of these s_i is itself an infinite sequence of 0's and 1's. Let's call the j-th element of the i-th sequence in this list $s_i(j)$. Then the i-th sequence s_i is

$$s_i(1), s_i(2), s_i(3), \ldots$$

We may arrange this list, and the elements of each sequence s_i in it, in an array:

	1	2	3	4	...
1	$\mathbf{s_1(1)}$	$s_1(2)$	$s_1(3)$	$s_1(4)$	...
2	$s_2(1)$	$\mathbf{s_2(2)}$	$s_2(3)$	$s_2(4)$	...
3	$s_3(1)$	$s_3(2)$	$\mathbf{s_3(3)}$	$s_3(4)$	...
4	$s_4(1)$	$s_4(2)$	$s_4(3)$	$\mathbf{s_4(4)}$	...
⋮	⋮	⋮	⋮	⋮	⋱

The labels down the side give the number of the sequence in the list $s_1, s_2, \ldots$; the numbers across the top label the elements of the individual sequences. For instance, $s_1(1)$ is a name for whatever number, a 0 or a 1, is the first element in the sequence s_1, and so on.

Now we construct an infinite sequence, $\bar{s}$, of 0's and 1's which cannot possibly be on this list. The definition of $\bar{s}$ will depend on the list s_1, s_2, ... Any infinite list of infinite sequences of 0's and 1's gives rise to an infinite sequence $\bar{s}$ which is guaranteed to not appear on the list.

To define $\bar{s}$, we specify what all its elements are, i.e., we specify $\bar{s}(n)$ for all $n \in \mathbb{Z}^+$. We do this by reading down the diagonal of the array above (hence the name "diagonal method") and then changing every 1 to a 0 and every 1 to a 0. More abstractly, we define $\bar{s}(n)$ to be 0 or 1 according to whether the n-th element of the diagonal, $s_n(n)$, is 1 or 0.

$$\bar{s}(n) = \begin{cases} 1 & \text{if } s_n(n) = 0 \\ 0 & \text{if } s_n(n) = 1. \end{cases}$$

If you like formulas better than definitions by cases, you could also define $\bar{s}(n) = 1 - s_n(n)$.

Clearly $\bar{s}$ is a non-gappy infinite sequence of 0's and 1's, since it is just the mirror sequence to the sequence of 0's and 1's that appear on the diagonal of our array. So $\bar{s}$ is an element of $\mathbb{B}^\omega$. But it cannot be on the list $s_1, s_2, \ldots$ Why not?

It can't be the first sequence in the list, s_1, because it differs from s_1 in the first element. Whatever $s_1(1)$ is, we defined $\bar{s}(1)$ to be the opposite. It can't be the second sequence in the list, because $\bar{s}$ differs from s_2 in the second element: if $s_2(2)$ is 0, $\bar{s}(2)$ is 1, and vice versa. And so on.

More precisely: if $\bar{s}$ were on the list, there would be some k so that $\bar{s} = s_k$. Two sequences are identical iff they agree at every place, i.e., for any n, $\bar{s}(n) = s_k(n)$. So in particular, taking $n = k$ as a special case, $\bar{s}(k) = s_k(k)$ would have to hold. $s_k(k)$ is either 0 or 1. If it is 0 then $\bar{s}(k)$ must be 1—that's how we defined $\bar{s}$. But if $s_k(k) = 1$ then, again because of the way we defined $\bar{s}$, $\bar{s}(k) = 0$. In either case $\bar{s}(k) \neq s_k(k)$.

We started by assuming that there is a list of elements of $\mathbb{B}^\omega$, s_1, s_2, ... From this list we constructed a sequence $\bar{s}$ which we proved cannot be on the

list. But it definitely is a sequence of 0's and 1's if all the s_i are sequences of 0's and 1's, i.e., $\bar{s} \in \mathbb{B}^\omega$. This shows in particular that there can be no list of *all* elements of $\mathbb{B}^\omega$, since for any such list we could also construct a sequence $\bar{s}$ guaranteed to not be on the list, so the assumption that there is a list of all sequences in $\mathbb{B}^\omega$ leads to a contradiction. □

This proof method is called "diagonalization" because it uses the diagonal of the array to define $\bar{s}$. Diagonalization need not involve the presence of an array: we can show that sets are not countable by using a similar idea even when no array and no actual diagonal is involved.

Theorem 4.9. *$\wp(\mathbb{Z}^+)$ is not countable.*

Proof. We proceed in the same way, by showing that for every list of subsets of $\mathbb{Z}^+$ there is a subset of $\mathbb{Z}^+$ which cannot be on the list. Suppose the following is a given list of subsets of $\mathbb{Z}^+$:

$$Z_1, Z_2, Z_3, \ldots$$

We now define a set $\overline{Z}$ such that for any $n \in \mathbb{Z}^+$, $n \in \overline{Z}$ iff $n \notin Z_n$:

$$\overline{Z} = \{n \in \mathbb{Z}^+ \mid n \notin Z_n\}$$

$\overline{Z}$ is clearly a set of positive integers, since by assumption each Z_n is, and thus $\overline{Z} \in \wp(\mathbb{Z}^+)$. But $\overline{Z}$ cannot be on the list. To show this, we'll establish that for each $k \in \mathbb{Z}^+$, $\overline{Z} \neq Z_k$.

So let $k \in \mathbb{Z}^+$ be arbitrary. We've defined $\overline{Z}$ so that for any $n \in \mathbb{Z}^+$, $n \in \overline{Z}$ iff $n \notin Z_n$. In particular, taking $n = k$, $k \in \overline{Z}$ iff $k \notin Z_k$. But this shows that $\overline{Z} \neq Z_k$, since k is an element of one but not the other, and so $\overline{Z}$ and Z_k have different elements. Since k was arbitrary, $\overline{Z}$ is not on the list $Z_1, Z_2, \ldots$ □

The preceding proof did not mention a diagonal, but you can think of it as involving a diagonal if you picture it this way: Imagine the sets $Z_1, Z_2, \ldots$, written in an array, where each element $j \in Z_i$ is listed in the j-th column. Say the first four sets on that list are $\{1, 2, 3, \ldots\}$, $\{2, 4, 6, \ldots\}$, $\{1, 2, 5\}$, and $\{3, 4, 5, \ldots\}$. Then the array would begin with

$$\begin{array}{lccccccc}
Z_1 = \{ & \mathbf{1}, & 2, & 3, & 4, & 5, & 6, & \ldots\} \\
Z_2 = \{ & & \mathbf{2}, & & 4, & & 6, & \ldots\} \\
Z_3 = \{ & 1, & 2, & & & 5 & & \} \\
Z_4 = \{ & & & 3, & \mathbf{4}, & 5, & 6, & \ldots\} \\
\vdots & & & & & \ddots & &
\end{array}$$

Then $\overline{Z}$ is the set obtained by going down the diagonal, leaving out any numbers that appear along the diagonal and include those j where the array has a gap in the j-th row/column. In the above case, we would leave out 1 and 2, include 3, leave out 4, etc.

4.4 Reduction

We showed $\wp(\mathbb{Z}^+)$ to be uncountable by a diagonalization argument. We already had a proof that $\mathbb{B}^\omega$, the set of all infinite sequences of 0s and 1s, is uncountable. Here's another way we can prove that $\wp(\mathbb{Z}^+)$ is uncountable: Show that *if* $\wp(\mathbb{Z}^+)$ *is countable then* $\mathbb{B}^\omega$ *is also countable*. Since we know $\mathbb{B}^\omega$ is not countable, $\wp(\mathbb{Z}^+)$ can't be either. This is called *reducing* one problem to another—in this case, we reduce the problem of enumerating $\mathbb{B}^\omega$ to the problem of enumerating $\wp(\mathbb{Z}^+)$. A solution to the latter—an enumeration of $\wp(\mathbb{Z}^+)$—would yield a solution to the former—an enumeration of $\mathbb{B}^\omega$.

How do we reduce the problem of enumerating a set Y to that of enumerating a set X? We provide a way of turning an enumeration of X into an enumeration of Y. The easiest way to do that is to define a surjective function $f\colon X \to Y$. If $x_1, x_2, \ldots$ enumerates X, then $f(x_1), f(x_2), \ldots$ would enumerate Y. In our case, we are looking for a surjective function $f\colon \wp(\mathbb{Z}^+) \to \mathbb{B}^\omega$.

Proof of Theorem 4.9 by reduction. Suppose that $\wp(\mathbb{Z}^+)$ were countable, and thus that there is an enumeration of it, $Z_1, Z_2, Z_3, \ldots$

Define the function $f\colon \wp(\mathbb{Z}^+) \to \mathbb{B}^\omega$ by letting $f(Z)$ be the sequence s_k such that $s_k(n) = 1$ iff $n \in Z$, and $s_k(n) = 0$ otherwise. This clearly defines a function, since whenever $Z \subseteq \mathbb{Z}^+$, any $n \in \mathbb{Z}^+$ either is an element of Z or isn't. For instance, the set $2\mathbb{Z}^+ = \{2, 4, 6, \ldots\}$ of positive even numbers gets mapped to the sequence $010101\ldots$, the empty set gets mapped to $0000\ldots$ and the set $\mathbb{Z}^+$ itself to $1111\ldots$.

It also is surjective: Every sequence of 0s and 1s corresponds to some set of positive integers, namely the one which has as its members those integers corresponding to the places where the sequence has 1s. More precisely, suppose $s \in \mathbb{B}^\omega$. Define $Z \subseteq \mathbb{Z}^+$ by:

$$Z = \{n \in \mathbb{Z}^+ \mid s(n) = 1\}$$

Then $f(Z) = s$, as can be verified by consulting the definition of f.

Now consider the list

$$f(Z_1), f(Z_2), f(Z_3), \ldots$$

Since f is surjective, every member of $\mathbb{B}^\omega$ must appear as a value of f for some argument, and so must appear on the list. This list must therefore enumerate all of $\mathbb{B}^\omega$.

So if $\wp(\mathbb{Z}^+)$ were countable, $\mathbb{B}^\omega$ would be countable. But $\mathbb{B}^\omega$ is uncountable (Theorem 4.8). Hence $\wp(\mathbb{Z}^+)$ is uncountable. □

It is easy to be confused about the direction the reduction goes in. For instance, a surjective function $g\colon \mathbb{B}^\omega \to X$ does *not* establish that X is uncountable. (Consider $g\colon \mathbb{B}^\omega \to \mathbb{B}$ defined by $g(s) = s(1)$, the function that maps a sequence of 0's and 1's to its first element. It is surjective, because

some sequences start with 0 and some start with 1. But $\mathbb{B}$ is finite.) Note also that the function f must be surjective, or otherwise the argument does not go through: $f(x_1)$, $f(x_2)$, ... would then not be guaranteed to include all the elements of Y. For instance, $h\colon \mathbb{Z}^+ \to \mathbb{B}^\omega$ defined by

$$h(n) = \underbrace{000\ldots0}_{n\ 0\text{'s}}$$

is a function, but $\mathbb{Z}^+$ is countable.

4.5 Equinumerous Sets

We have an intuitive notion of "size" of sets, which works fine for finite sets. But what about infinite sets? If we want to come up with a formal way of comparing the sizes of two sets of *any* size, it is a good idea to start with defining when sets are the same size. Let's say sets of the same size are *equinumerous*. We want the formal notion of equinumerosity to correspond with our intuitive notion of "same size," hence the formal notion ought to satisfy the following properties:

Reflexivity: Every set is equinumerous with itself.

Symmetry: For any sets X and Y, if X is equinumerous with Y, then Y is equinumerous with X.

Transitivity: For any sets X, Y, and Z, if X is equinumerous with Y and Y is equinumerous with Z, then X is equinumerous with Z.

In other words, we want equinumerosity to be an *equivalence relation*.

Definition 4.10. A set X is *equinumerous* with a set Y, $X \approx Y$, if and only if there is a bijective $f\colon X \to Y$.

Proposition 4.11. *Equinumerosity defines an equivalence relation.*

Proof. Let X, Y, and Z be sets.

Reflexivity: Using the identity map $1_X\colon X \to X$, where $1_X(x) = x$ for all $x \in X$, we see that X is equinumerous with itself (clearly, 1_X is bijective).

Symmetry: Suppose that X is equinumerous with Y. Then there is a bijective $f\colon X \to Y$. Since f is bijective, its inverse f^{-1} exists and also bijective. Hence, $f^{-1}\colon Y \to X$ is a bijective function from Y to X, so Y is also equinumerous with X.

Transitivity: Suppose that X is equinumerous with Y via the bijective function $f\colon X \to Y$ and that Y is equinumerous with Z via the bijective function $g\colon Y \to Z$. Then the composition of $g \circ f\colon X \to Z$ is bijective, and X is thus equinumerous with Z.

Therefore, equinumerosity is an equivalence relation. □

Theorem 4.12. *Suppose X and Y are equinumerous. Then X is countable if and only if Y is.*

Proof. Let X and Y be equinumerous. Suppose that X is countable. Then either $X = \emptyset$ or there is a surjective function $f\colon \mathbb{Z}^+ \to X$. Since X and Y are equinumerous, there is a bijective $g\colon X \to Y$. If $X = \emptyset$, then $Y = \emptyset$ also (otherwise there would be an element $y \in Y$ but no $x \in X$ with $g(x) = y$). If, on the other hand, $f\colon \mathbb{Z}^+ \to X$ is surjective, then $g \circ f\colon \mathbb{Z}^+ \to Y$ is surjective. To see this, let $y \in Y$. Since g is surjective, there is an $x \in X$ such that $g(x) = y$. Since f is surjective, there is an $n \in \mathbb{Z}^+$ such that $f(n) = x$. Hence,

$$(g \circ f)(n) = g(f(n)) = g(x) = y$$

and thus $g \circ f$ is surjective. We have that $g \circ f$ is an enumeration of Y, and so Y is countable. □

4.6 Comparing Sizes of Sets

Just like we were able to make precise when two sets have the same size in a way that also accounts for the size of infinite sets, we can also compare the sizes of sets in a precise way. Our definition of "is smaller than (or equinumerous)" will require, instead of a bijection between the sets, a total injective function from the first set to the second. If such a function exists, the size of the first set is less than or equal to the size of the second. Intuitively, an injective function from one set to another guarantees that the range of the function has at least as many elements as the domain, since no two elements of the domain map to the same element of the range.

Definition 4.13. X is *no larger than* Y, $X \preceq Y$, if and only if there is an injective function $f\colon X \to Y$.

Theorem 4.14 (Schröder-Bernstein)**.** *Let X and Y be sets. If $X \preceq Y$ and $Y \preceq X$, then $X \approx Y$.*

In other words, if there is a total injective function from X to Y, and if there is a total injective function from Y back to X, then there is a total bijection from X to Y. Sometimes, it can be difficult to think of a bijection between two equinumerous sets, so the Schröder-Bernstein theorem allows us to break the comparison down into cases so we only have to think of an injection from

the first to the second, and vice-versa. The Schröder-Bernstein theorem, apart from being convenient, justifies the act of discussing the "sizes" of sets, for it tells us that set cardinalities have the familiar anti-symmetric property that numbers have.

Definition 4.15. X is *smaller than* Y, $X \prec Y$, if and only if there is an injective function $f\colon X \to Y$ but no bijective $g\colon X \to Y$.

Theorem 4.16 (Cantor)**.** *For all* X, $X \prec \wp(X)$.

Proof. The function $f\colon X \to \wp(X)$ that maps any $x \in X$ to its singleton $\{x\}$ is injective, since if $x \neq y$ then also $f(x) = \{x\} \neq \{y\} = f(y)$.

There cannot be a surjective function $g\colon X \to \wp(X)$, let alone a bijective one. For suppose that $g\colon X \to \wp(X)$. Since g is total, every $x \in X$ is mapped to a subset $g(x) \subseteq X$. We show that g cannot be surjective. To do this, we define a subset $Y \subseteq X$ which by definition cannot be in the range of g. Let

$$\overline{Y} = \{x \in X \mid x \notin g(x)\}.$$

Since $g(x)$ is defined for all $x \in X$, $\overline{Y}$ is clearly a well-defined subset of X. But, it cannot be in the range of g. Let $x \in X$ be arbitrary, we show that $\overline{Y} \neq g(x)$. If $x \in g(x)$, then it does not satisfy $x \notin g(x)$, and so by the definition of $\overline{Y}$, we have $x \notin \overline{Y}$. If $x \in \overline{Y}$, it must satisfy the defining property of $\overline{Y}$, i.e., $x \notin g(x)$. Since x was arbitrary this shows that for each $x \in X$, $x \in g(x)$ iff $x \notin \overline{Y}$, and so $g(x) \neq \overline{Y}$. So $\overline{Y}$ cannot be in the range of g, contradicting the assumption that g is surjective. □

It's instructive to compare the proof of Theorem 4.16 to that of Theorem 4.9. There we showed that for any list Z_1, Z_2, ..., of subsets of $\mathbb{Z}^+$ one can construct a set $\overline{Z}$ of numbers guaranteed not to be on the list. It was guaranteed not to be on the list because, for every $n \in \mathbb{Z}^+$, $n \in Z_n$ iff $n \notin \overline{Z}$. This way, there is always some number that is an element of one of Z_n and $\overline{Z}$ but not the other. We follow the same idea here, except the indices n are now elements of X instead of $\mathbb{Z}^+$. The set $\overline{Y}$ is defined so that it is different from $g(x)$ for each $x \in X$, because $x \in g(x)$ iff $x \notin \overline{Y}$. Again, there is always an element of X which is an element of one of $g(x)$ and $\overline{Y}$ but not the other. And just as $\overline{Z}$ therefore cannot be on the list Z_1, Z_2, ..., $\overline{Y}$ cannot be in the range of g.

Part II

First-order Logic

Chapter 5

Syntax and Semantics

5.1 Introduction

In order to develop the theory and metatheory of first-order logic, we must first define the syntax and semantics of its expressions. The expressions of first-order logic are terms and formulae. Terms are formed from variables, constant symbols, and function symbols. Formulae, in turn, are formed from predicate symbols together with terms (these form the smallest, "atomic" formulae), and then from atomic formulae we can form more complex ones using logical connectives and quantifiers. There are many different ways to set down the formation rules; we give just one possible one. Other systems will chose different symbols, will select different sets of connectives as primitive, will use parentheses differently (or even not at all, as in the case of so-called Polish notation). What all approaches have in common, though, is that the formation rules define the set of terms and formulae *inductively*. If done properly, every expression can result essentially in only one way according to the formation rules. The inductive definition resulting in expressions that are *uniquely readable* means we can give meanings to these expressions using the same method—inductive definition.

Giving the meaning of expressions is the domain of semantics. The central concept in semantics is that of satisfaction in a structure. A structure gives meaning to the building blocks of the language: a domain is a non-empty set of objects. The quantifiers are interpreted as ranging over this domain, constant symbols are assigned elements in the domain, function symbols are assigned functions from the domain to itself, and predicate symbols are assigned relations on the domain. The domain together with assignments to the basic vocabulary constitutes a structure. Variables may appear in formulae, and in order to give a semantics, we also have to assign elements of the domain to them—this is a variable assignment. The satisfaction relation, finally, brings these together. A formula may be satisfied in a structure $\mathfrak{M}$ relative to a variable assignment s, written as $\mathfrak{M}, s \models \varphi$. This relation is also defined by in-

duction on the structure of φ, using the truth tables for the logical connectives to define, say, satisfaction of $\varphi \,\&\, \psi$ in terms of satisfaction (or not) of φ and ψ. It then turns out that the variable assignment is irrelevant if the formula φ is a sentence, i.e., has no free variables, and so we can talk of sentences being simply satisfied (or not) in structures.

On the basis of the satisfaction relation $\mathfrak{M} \models \varphi$ for sentences we can then define the basic semantic notions of validity, entailment, and satisfiability. A sentence is valid, $\vDash \varphi$, if every structure satisfies it. It is entailed by a set of sentences, $\Gamma \vDash \varphi$, if every structure that satisfies all the sentences in Γ also satisfies φ. And a set of sentences is satisfiable if some structure satisfies all sentences in it at the same time. Because formulae are inductively defined, and satisfaction is in turn defined by induction on the structure of formulae, we can use induction to prove properties of our semantics and to relate the semantic notions defined.

5.2 First-Order Languages

Expressions of first-order logic are built up from a basic vocabulary containing *variables*, *constant symbols*, *predicate symbols* and sometimes *function symbols*. From them, together with logical connectives, quantifiers, and punctuation symbols such as parentheses and commas, *terms* and *formulae* are formed.

Informally, predicate symbols are names for properties and relations, constant symbols are names for individual objects, and function symbols are names for mappings. These, except for the identity predicate $=$, are the *non-logical symbols* and together make up a language. Any first-order language $\mathcal{L}$ is determined by its non-logical symbols. In the most general case, $\mathcal{L}$ contains infinitely many symbols of each kind.

In the general case, we make use of the following symbols in first-order logic:

1. Logical symbols
    a) Logical connectives: $\sim$ (negation), & (conjunction), $\vee$ (disjunction), $\supset$ (conditional), $\forall$ (universal quantifier), $\exists$ (existential quantifier).
    b) The propositional constant for falsity $\bot$.
    c) The two-place identity predicate $=$.
    d) A countably infinite set of variables: v_0, v_1, v_2, ...
2. Non-logical symbols, making up the *standard language* of first-order logic
    a) A countably infinite set of n-place predicate symbols for each $n > 0$: A^n_0, A^n_1, A^n_2, ...
    b) A countably infinite set of constant symbols: c_0, c_1, c_2, ...

c) A countably infinite set of n-place function symbols for each $n > 0$: $f^n_0, f^n_1, f^n_2, \ldots$

3. Punctuation marks: (,), and the comma.

Most of our definitions and results will be formulated for the full standard language of first-order logic. However, depending on the application, we may also restrict the language to only a few predicate symbols, constant symbols, and function symbols.

Example 5.1. The language $\mathcal{L}_A$ of arithmetic contains a single two-place predicate symbol $<$, a single constant symbol 0, one one-place function symbol $'$, and two two-place function symbols $+$ and $\times$.

Example 5.2. The language of set theory $\mathcal{L}_Z$ contains only the single two-place predicate symbol $\in$.

Example 5.3. The language of orders $\mathcal{L}_\leq$ contains only the two-place predicate symbol $\leq$.

Again, these are conventions: officially, these are just aliases, e.g., $<$, $\in$, and $\leq$ are aliases for A^2_0, 0 for c_0, $'$ for f^1_0, $+$ for f^2_0, $\times$ for f^2_1.

In addition to the primitive connectives and quantifiers introduced above, we also use the following *defined* symbols: $\equiv$ (biconditional), truth $\top$

A defined symbol is not officially part of the language, but is introduced as an informal abbreviation: it allows us to abbreviate formulas which would, if we only used primitive symbols, get quite long. This is obviously an advantage. The bigger advantage, however, is that proofs become shorter. If a symbol is primitive, it has to be treated separately in proofs. The more primitive symbols, therefore, the longer our proofs.

You may be familiar with different terminology and symbols than the ones we use above. Logic texts (and teachers) commonly use either $\sim$, $\neg$, and ! for "negation", $\wedge$, $\cdot$, and & for "conjunction". Commonly used symbols for the "conditional" or "implication" are $\rightarrow$, $\Rightarrow$, and $\supset$. Symbols for "biconditional," "bi-implication," or "(material) equivalence" are $\leftrightarrow$, $\Leftrightarrow$, and $\equiv$. The $\bot$ symbol is variously called "falsity," "falsum,", "absurdity,", or "bottom." The $\top$ symbol is variously called "truth," "verum,", or "top."

It is conventional to use lower case letters (e.g., a, b, c) from the beginning of the Latin alphabet for constant symbols (sometimes called names), and lower case letters from the end (e.g., x, y, z) for variables. Quantifiers combine with variables, e.g., x; notational variations include $\forall x$, $(\forall x)$, (x), Πx, $\bigwedge_x$ for the universal quantifier and $\exists x$, $(\exists x)$, (Ex), Σx, $\bigvee_x$ for the existential quantifier.

We might treat all the propositional operators and both quantifiers as primitive symbols of the language. We might instead choose a smaller stock of primitive symbols and treat the other logical operators as defined. "Truth

functionally complete" sets of Boolean operators include $\{\sim, \vee\}$, $\{\sim, \&\}$, and $\{\sim, \supset\}$—these can be combined with either quantifier for an expressively complete first-order language.

You may be familiar with two other logical operators: the Sheffer stroke $|$ (named after Henry Sheffer), and Peirce's arrow $\downarrow$, also known as Quine's dagger. When given their usual readings of "nand" and "nor" (respectively), these operators are truth functionally complete by themselves.

5.3 Terms and Formulae

Once a first-order language $\mathcal{L}$ is given, we can define expressions built up from the basic vocabulary of $\mathcal{L}$. These include in particular *terms* and *formulae*.

Definition 5.4 (Terms)**.** The set of *terms* $\mathrm{Trm}(\mathcal{L})$ of $\mathcal{L}$ is defined inductively by:

1. Every variable is a term.
2. Every constant symbol of $\mathcal{L}$ is a term.
3. If f is an n-place function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.
4. Nothing else is a term.

A term containing no variables is a *closed term*.

The constant symbols appear in our specification of the language and the terms as a separate category of symbols, but they could instead have been included as zero-place function symbols. We could then do without the second clause in the definition of terms. We just have to understand $f(t_1, \ldots, t_n)$ as just f by itself if $n = 0$.

Definition 5.5 (Formula)**.** The set of *formulae* $\mathrm{Frm}(\mathcal{L})$ of the language $\mathcal{L}$ is defined inductively as follows:

1. $\bot$ is an atomic formula.
2. If R is an n-place predicate symbol of $\mathcal{L}$ and $t_1, \ldots, t_n$ are terms of $\mathcal{L}$, then $R(t_1, \ldots, t_n)$ is an atomic formula.
3. If t_1 and t_2 are terms of $\mathcal{L}$, then $=(t_1, t_2)$ is an atomic formula.
4. If φ is a formula, then $\sim\varphi$ is formula.
5. If φ and ψ are formulae, then $(\varphi \,\&\, \psi)$ is a formula.
6. If φ and ψ are formulae, then $(\varphi \vee \psi)$ is a formula.

7. If φ and ψ are formulae, then $(\varphi \supset \psi)$ is a formula.
8. If φ is a formula and x is a variable, then $\forall x\, \varphi$ is a formula.
9. If φ is a formula and x is a variable, then $\exists x\, \varphi$ is a formula.
10. Nothing else is a formula.

The definitions of the set of terms and that of formulae are *inductive definitions*. Essentially, we construct the set of formulae in infinitely many stages. In the initial stage, we pronounce all atomic formulas to be formulas; this corresponds to the first few cases of the definition, i.e., the cases for $\bot$, $R(t_1, \ldots, t_n)$ and $=(t_1, t_2)$. "Atomic formula" thus means any formula of this form.

The other cases of the definition give rules for constructing new formulae out of formulae already constructed. At the second stage, we can use them to construct formulae out of atomic formulae. At the third stage, we construct new formulas from the atomic formulas and those obtained in the second stage, and so on. A formula is anything that is eventually constructed at such a stage, and nothing else.

By convention, we write $=$ between its arguments and leave out the parentheses: $t_1 = t_2$ is an abbreviation for $=(t_1, t_2)$. Moreover, $\sim{=}(t_1, t_2)$ is abbreviated as $t_1 \neq t_2$. When writing a formula $(\psi * \chi)$ constructed from ψ, χ using a two-place connective $*$, we will often leave out the outermost pair of parentheses and write simply $\psi * \chi$.

Some logic texts require that the variable x must occur in φ in order for $\exists x\, \varphi$ and $\forall x\, \varphi$ to count as formulae. Nothing bad happens if you don't require this, and it makes things easier.

Definition 5.6. Formulas constructed using the defined operators are to be understood as follows:

1. $\top$ abbreviates $\sim\bot$.
2. $\varphi \equiv \psi$ abbreviates $(\varphi \supset \psi)\ \&\ (\psi \supset \varphi)$.

If we work in a language for a specific application, we will often write two-place predicate symbols and function symbols between the respective terms, e.g., $t_1 < t_2$ and $(t_1 + t_2)$ in the language of arithmetic and $t_1 \in t_2$ in the language of set theory. The successor function in the language of arithmetic is even written conventionally *after* its argument: t'. Officially, however, these are just conventional abbreviations for $A^2_0(t_1, t_2)$, $f^2_0(t_1, t_2)$, $A^2_0(t_1, t_2)$ and $f^1_0(t)$, respectively.

Definition 5.7 (Syntactic identity)**.** The symbol $\equiv$ expresses syntactic identity between strings of symbols, i.e., $\varphi \equiv \psi$ iff φ and ψ are strings of symbols of the same length and which contain the same symbol in each place.

The $\equiv$ symbol may be flanked by strings obtained by concatenation, e.g., $\varphi \equiv (\psi \vee \chi)$ means: the string of symbols φ is the same string as the one obtained by concatenating an opening parenthesis, the string ψ, the $\vee$ symbol, the string χ, and a closing parenthesis, in this order. If this is the case, then we know that the first symbol of φ is an opening parenthesis, φ contains ψ as a substring (starting at the second symbol), that substring is followed by $\vee$, etc.

5.4 Unique Readability

The way we defined formulae guarantees that every formula has a *unique reading*, i.e., there is essentially only one way of constructing it according to our formation rules for formulae and only one way of "interpreting" it. If this were not so, we would have ambiguous formulae, i.e., formulae that have more than one reading or intepretation—and that is clearly something we want to avoid. But more importantly, without this property, most of the definitions and proofs we are going to give will not go through.

Perhaps the best way to make this clear is to see what would happen if we had given bad rules for forming formulae that would not guarantee unique readability. For instance, we could have forgotten the parentheses in the formation rules for connectives, e.g., we might have allowed this:

> If φ and ψ are formulae, then so is $\varphi \supset \psi$.

Starting from an atomic formula θ, this would allow us to form $\theta \supset \theta$. From this, together with θ, we would get $\theta \supset \theta \supset \theta$. But there are two ways to do this:

1. We take θ to be φ and $\theta \supset \theta$ to be ψ.

2. We take φ to be $\theta \supset \theta$ and ψ is θ.

Correspondingly, there are two ways to "read" the formula $\theta \supset \theta \supset \theta$. It is of the form $\psi \supset \chi$ where ψ is θ and χ is $\theta \supset \theta$, but *it is also* of the form $\psi \supset \chi$ with ψ being $\theta \supset \theta$ and χ being θ.

If this happens, our definitions will not always work. For instance, when we define the main operator of a formula, we say: in a formula of the form $\psi \supset \chi$, the main operator is the indicated occurrence of $\supset$. But if we can match the formula $\theta \supset \theta \supset \theta$ with $\psi \supset \chi$ in the two different ways mentioned above, then in one case we get the first occurrence of $\supset$ as the main operator, and in the second case the second occurrence. But we intend the main operator to be a *function* of the formula, i.e., every formula must have exactly one main operator occurrence.

Lemma 5.8. *The number of left and right parentheses in a formula φ are equal.*

Proof. We prove this by induction on the way φ is constructed. This requires two things: (a) We have to prove first that all atomic formulas have the property in question (the induction basis). (b) Then we have to prove that when we construct new formulas out of given formulas, the new formulas have the property provided the old ones do.

Let $l(\varphi)$ be the number of left parentheses, and $r(\varphi)$ the number of right parentheses in φ, and $l(t)$ and $r(t)$ similarly the number of left and right parentheses in a term t. We leave the proof that for any term t, $l(t) = r(t)$ as an exercise.

1. $\varphi \equiv \bot$: φ has 0 left and 0 right parentheses.
2. $\varphi \equiv R(t_1, \ldots, t_n)$: $l(\varphi) = 1 + l(t_1) + \cdots + l(t_n) = 1 + r(t_1) + \cdots + r(t_n) = r(\varphi)$. Here we make use of the fact, left as an exercise, that $l(t) = r(t)$ for any term t.
3. $\varphi \equiv t_1 = t_2$: $l(\varphi) = l(t_1) + l(t_2) = r(t_1) + r(t_2) = r(\varphi)$.
4. $\varphi \equiv \sim\psi$: By induction hypothesis, $l(\psi) = r(\psi)$. Thus $l(\varphi) = l(\psi) = r(\psi) = r(\varphi)$.
5. $\varphi \equiv (\psi * \chi)$: By induction hypothesis, $l(\psi) = r(\psi)$ and $l(\chi) = r(\chi)$. Thus $l(\varphi) = 1 + l(\psi) + l(\chi) = 1 + r(\psi) + r(\chi) = r(\varphi)$.
6. $\varphi \equiv \forall x\, \psi$: By induction hypothesis, $l(\psi) = r(\psi)$. Thus, $l(\varphi) = l(\psi) = r(\psi) = r(\varphi)$.
7. $\varphi \equiv \exists x\, \psi$: Similarly.

□

Definition 5.9 (Proper prefix)**.** A string of symbols ψ is a *proper prefix* of a string of symbols φ if concatenating ψ and a non-empty string of symbols yields φ.

Lemma 5.10. *If φ is a formula, and ψ is a proper prefix of φ, then ψ is not a formula.*

Proof. Exercise. □

Proposition 5.11. *If φ is an atomic formula, then it satisfes one, and only one of the following conditions.*

1. *$\varphi \equiv \bot$.*
2. *$\varphi \equiv R(t_1, \ldots, t_n)$ where R is an n-place predicate symbol, $t_1, \ldots, t_n$ are terms, and each of $R, t_1, \ldots, t_n$ is uniquely determined.*
3. *$\varphi \equiv t_1 = t_2$ where t_1 and t_2 are uniquely determined terms.*

Proof. Exercise. □

Proposition 5.12 (Unique Readability)**.** *Every formula satisfies one, and only one of the following conditions.*

1. *φ is atomic.*
2. *φ is of the form $\sim\psi$.*
3. *φ is of the form $(\psi \mathbin{\&} \chi)$.*
4. *φ is of the form $(\psi \vee \chi)$.*
5. *φ is of the form $(\psi \supset \chi)$.*
6. *φ is of the form $\forall x\, \psi$.*
7. *φ is of the form $\exists x\, \psi$.*

Moreover, in each case ψ, or ψ and χ, are uniquely determined. This means that, e.g., there are no different pairs ψ, χ and ψ', χ' so that φ is both of the form $(\psi \supset \chi)$ and $(\psi' \supset \chi')$.

Proof. The formation rules require that if a formula is not atomic, it must start with an opening parenthesis (, $\sim$, or with a quantifier. On the other hand, every formula that start with one of the following symbols must be atomic: a predicate symbol, a function symbol, a constant symbol, $\bot$.

So we really only have to show that if φ is of the form $(\psi * \chi)$ and also of the form $(\psi' *' \chi')$, then $\psi \equiv \psi'$, $\chi \equiv \chi'$, and $* = *'$.

So suppose both $\varphi \equiv (\psi * \chi)$ and $\varphi \equiv (\psi' *' \chi')$. Then either $\psi \equiv \psi'$ or not. If it is, clearly $* = *'$ and $\chi \equiv \chi'$, since they then are substrings of φ that begin in the same place and are of the same length. The other case is $\psi \not\equiv \psi'$. Since ψ and ψ' are both substrings of φ that begin at the same place, one must be a proper prefix of the other. But this is impossible by Lemma 5.10. □

5.5 Main operator of a Formula

It is often useful to talk about the last operator used in constructing a formula φ. This operator is called the *main operator* of φ. Intuitively, it is the "outermost" operator of φ. For example, the main operator of $\sim\varphi$ is $\sim$, the main operator of $(\varphi \vee \psi)$ is $\vee$, etc.

Definition 5.13 (Main operator)**.** The *main operator* of a formula φ is defined as follows:

1. φ is atomic: φ has no main operator.
2. $\varphi \equiv \sim\psi$: the main operator of φ is $\sim$.

3. $\varphi \equiv (\psi \,\&\, \chi)$: the main operator of φ is &.
4. $\varphi \equiv (\psi \vee \chi)$: the main operator of φ is $\vee$.
5. $\varphi \equiv (\psi \supset \chi)$: the main operator of φ is $\supset$.
6. $\varphi \equiv \forall x\, \psi$: the main operator of φ is $\forall$.
7. $\varphi \equiv \exists x\, \psi$: the main operator of φ is $\exists$.

In each case, we intend the specific indicated *occurrence* of the main operator in the formula. For instance, since the formula $((\theta \supset \alpha) \supset (\alpha \supset \theta))$ is of the form $(\psi \supset \chi)$ where ψ is $(\theta \supset \alpha)$ and χ is $(\alpha \supset \theta)$, the second occurrence of $\supset$ is the main operator.

This is a *recursive* definition of a function which maps all non-atomic formulae to their main operator occurrence. Because of the way formulae are defined inductively, every formula φ satisfies one of the cases in Definition 5.13. This guarantees that for each non-atomic formula φ a main operator exists. Because each formula satisfies only one of these conditions, and because the smaller formulae from which φ is constructed are uniquely determined in each case, the main operator occurrence of φ is unique, and so we have defined a function.

We call formulae by the following names depending on which symbol their main operator is:

Main operator	Type of formula	Example
none	atomic (formula)	$\bot,\ R(t_1, \ldots, t_n), t_1 = t_2$
$\sim$	negation	$\sim\varphi$
&	conjunction	$(\varphi \,\&\, \psi)$
$\vee$	disjunction	$(\varphi \vee \psi)$
$\supset$	conditional	$(\varphi \supset \psi)$
$\forall$	universal (formula)	$\forall x\, \varphi$
$\exists$	existential (formula)	$\exists x\, \varphi$

5.6 Subformulae

It is often useful to talk about the formulae that "make up" a given formula. We call these its *subformulae*. Any formula counts as a subformula of itself; a subformula of φ other than φ itself is a *proper subformula*.

Definition 5.14 (Immediate Subformula)**.** If φ is a formula, the *immediate subformulae* of φ are defined inductively as follows:

1. Atomic formulae have no immediate subformulae.
2. $\varphi \equiv \sim\psi$: The only immediate subformula of φ is ψ.
3. $\varphi \equiv (\psi * \chi)$: The immediate subformulae of φ are ψ and χ ($*$ is any one of the two-place connectives).

4. $\varphi \equiv \forall x\, \psi$: The only immediate subformula of φ is ψ.

5. $\varphi \equiv \exists x\, \psi$: The only immediate subformula of φ is ψ.

Definition 5.15 (Proper Subformula)**.** If φ is a formula, the *proper subformulae* of φ are recursively as follows:

1. Atomic formulae have no proper subformulae.

2. $\varphi \equiv \sim\psi$: The proper subformulae of φ are ψ together with all proper subformulae of ψ.

3. $\varphi \equiv (\psi * \chi)$: The proper subformulae of φ are ψ, χ, together with all proper subformulae of ψ and those of χ.

4. $\varphi \equiv \forall x\, \psi$: The proper subformulae of φ are ψ together with all proper subformulae of ψ.

5. $\varphi \equiv \exists x\, \psi$: The proper subformulae of φ are ψ together with all proper subformulae of ψ.

Definition 5.16 (Subformula)**.** The subformulae of φ are φ itself together with all its proper subformulae.

Note the subtle difference in how we have defined immediate subformulae and proper subformulae. In the first case, we have directly defined the immediate subformulae of a formula φ for each possible form of φ. It is an explicit definition by cases, and the cases mirror the inductive definition of the set of formulae. In the second case, we have also mirrored the way the set of all formulae is defined, but in each case we have also included the proper subformulae of the smaller formulae ψ, χ in addition to these formulae themselves. This makes the definition *recursive*. In general, a definition of a function on an inductively defined set (in our case, formulae) is recursive if the cases in the definition of the function make use of the function itself. To be well defined, we must make sure, however, that we only ever use the values of the function for arguments that come "before" the one we are defining—in our case, when defining "proper subformula" for $(\psi * \chi)$ we only use the proper subformulae of the "earlier" formulae ψ and χ.

5.7 Free Variables and Sentences

Definition 5.17 (Free occurrences of a variable)**.** The *free* occurrences of a variable in a formula are defined inductively as follows:

1. φ is atomic: all variable occurrences in φ are free.

2. $\varphi \equiv \sim\psi$: the free variable occurrences of φ are exactly those of ψ.

3. $\varphi \equiv (\psi * \chi)$: the free variable occurrences of φ are those in ψ together with those in χ.

4. $\varphi \equiv \forall x\, \psi$: the free variable occurrences in φ are all of those in ψ except for occurrences of x.

5. $\varphi \equiv \exists x\, \psi$: the free variable occurrences in φ are all of those in ψ except for occurrences of x.

Definition 5.18 (Bound Variables)**.** An occurrence of a variable in a formula φ is *bound* if it is not free.

Definition 5.19 (Scope)**.** If $\forall x\, \psi$ is an occurrence of a subformula in a formula φ, then the corresponding occurrence of ψ in φ is called the *scope* of the corresponding occurrence of $\forall x$. Similarly for $\exists x$.

If ψ is the scope of a quantifier occurrence $\forall x$ or $\exists x$ in φ, then all occurrences of x which are free in ψ are said to be *bound by* the mentioned quantifier occurrence.

Example 5.20. Consider the following formula:

$$\exists v_0 \underbrace{A_0^2(v_0, v_1)}_{\psi}$$

ψ represents the scope of $\exists v_0$. The quantifier binds the occurence of v_0 in ψ, but does not bind the occurence of v_1. So v_1 is a free variable in this case.

We can now see how this might work in a more complicated formula φ:

$$\forall v_0 \underbrace{(A_0^1(v_0) \supset A_0^2(v_0, v_1))}_{\psi} \supset \exists v_1 \underbrace{(A_1^2(v_0, v_1) \vee \forall v_0 \overbrace{\sim A_1^1(v_0)}^{\theta})}_{\chi}$$

ψ is the scope of the first $\forall v_0$, χ is the scope of $\exists v_1$, and θ is the scope of the second $\forall v_0$. The first $\forall v_0$ binds the occurrences of v_0 in ψ, $\exists v_1$ the occurrence of v_1 in χ, and the second $\forall v_0$ binds the occurrence of v_0 in θ. The first occurrence of v_1 and the fourth occurrence of v_0 are free in φ. The last occurrence of v_0 is free in θ, but bound in χ and φ.

Definition 5.21 (Sentence)**.** A formula φ is a *sentence* iff it contains no free occurrences of variables.

5.8 Substitution

Definition 5.22 (Substitution in a term)**.** We define $s[t/x]$, the result of *substituting* t for every occurrence of x in s, recursively:

1. $s \equiv c$: $s[t/x]$ is just s.

2. $s \equiv y$: $s[t/x]$ is also just s, provided y is a variable and $y \not\equiv x$.

3. $s \equiv x$: $s[t/x]$ is t.

4. $s \equiv f(t_1, \ldots, t_n)$: $s[t/x]$ is $f(t_1[t/x], \ldots, t_n[t/x])$.

Definition 5.23. A term t is *free for* x in φ if none of the free occurrences of x in φ occur in the scope of a quantifier that binds a variable in t.

Example 5.24.

1. v_8 is free for v_1 in $\exists v_3 A_4^2(v_3, v_1)$

2. $f_1^2(v_1, v_2)$ is *not* free for v_o in $\forall v_2 A_4^2(v_0, v_2)$

Definition 5.25 (Substitution in a formula)**.** If φ is a formula, x is a variable, and t is a term free for x in φ, then $\varphi[t/x]$ is the result of substituting t for all free occurrences of x in φ.

1. $\varphi \equiv \bot$: $\varphi[t/x]$ is $\bot$.

2. $\varphi \equiv P(t_1, \ldots, t_n)$: $\varphi[t/x]$ is $P(t_1[t/x], \ldots, t_n[t/x])$.

3. $\varphi \equiv t_1 = t_2$: $\varphi[t/x]$ is $t_1[t/x] = t_2[t/x]$.

4. $\varphi \equiv \sim\psi$: $\varphi[t/x]$ is $\sim\psi[t/x]$.

5. $\varphi \equiv (\psi \,\&\, \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \,\&\, \chi[t/x])$.

6. $\varphi \equiv (\psi \vee \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \vee \chi[t/x])$.

7. $\varphi \equiv (\psi \supset \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \supset \chi[t/x])$.

8. $\varphi \equiv \forall y\, \psi$: $\varphi[t/x]$ is $\forall y\, \psi[t/x]$, provided y is a variable other than x; otherwise $\varphi[t/x]$ is just φ.

9. $\varphi \equiv \exists y\, \psi$: $\varphi[t/x]$ is $\exists y\, \psi[t/x]$, provided y is a variable other than x; otherwise $\varphi[t/x]$ is just φ.

Note that substitution may be vacuous: If x does not occur in φ at all, then $\varphi[t/x]$ is just φ.

The restriction that t must be free for x in φ is necessary to exclude cases like the following. If $\varphi \equiv \exists y\, x < y$ and $t \equiv y$, then $\varphi[t/x]$ would be $\exists y\, y < y$. In this case the free variable y is "captured" by the quantifier $\exists y$ upon substitution, and that is undesirable. For instance, we would like it to be the case that whenever $\forall x\, \psi$ holds, so does $\psi[t/x]$. But consider $\forall x\, \exists y\, x < y$ (here ψ is $\exists y\, x < y$). It is sentence that is true about, e.g., the natural numbers: for every number x there is a number y greater than it. If we allowed y as a possible substitution for x, we would end up with $\psi[y/x] \equiv \exists y\, y < y$, which

is false. We prevent this by requiring that none of the free variables in t would end up being bound by a quantifier in φ.

We often use the following convention to avoid cumbersume notation: If φ is a formula with a free variable x, we write $\varphi(x)$ to indicate this. When it is clear which φ and x we have in mind, and t is a term (assumed to be free for x in $\varphi(x)$), then we write $\varphi(t)$ as short for $\varphi(x)[t/x]$.

5.9 Structures for First-order Languages

First-order languages are, by themselves, *uninterpreted:* the constant symbols, function symbols, and predicate symbols have no specific meaning attached to them. Meanings are given by specifying a *structure*. It specifies the *domain*, i.e., the objects which the constant symbols pick out, the function symbols operate on, and the quantifiers range over. In addition, it specifies which constant symbols pick out which objects, how a function symbol maps objects to objects, and which objects the predicate symbols apply to. Structures are the basis for *semantic* notions in logic, e.g., the notion of consequence, validity, satisfiablity. They are variously called "structures," "interpretations," or "models" in the literature.

Definition 5.26 (Structures)**.** A *structure* $\mathfrak{M}$, for a language $\mathcal{L}$ of first-order logic consists of the following elements:

1. *Domain:* a non-empty set, $|\mathfrak{M}|$
2. *Interpretation of constant symbols:* for each constant symbol c of $\mathcal{L}$, an element $c^{\mathfrak{M}} \in |\mathfrak{M}|$
3. *Interpretation of predicate symbols:* for each n-place predicate symbol R of $\mathcal{L}$ (other than $=$), an n-place relation $R^{\mathfrak{M}} \subseteq |\mathfrak{M}|^n$
4. *Interpretation of function symbols:* for each n-place function symbol f of $\mathcal{L}$, an n-place function $f^{\mathfrak{M}} \colon |\mathfrak{M}|^n \to |\mathfrak{M}|$

Example 5.27. A structure $\mathfrak{M}$ for the language of arithmetic consists of a set, an element of $|\mathfrak{M}|$, $\mathsf{o}^{\mathfrak{M}}$, as interpretation of the constant symbol o, a one-place function $\mathsf{I}^{\mathfrak{M}} \colon |\mathfrak{M}| \to |\mathfrak{M}|$, two two-place functions $+^{\mathfrak{M}}$ and $\times^{\mathfrak{M}}$, both $|\mathfrak{M}|^2 \to |\mathfrak{M}|$, and a two-place relation $<^{\mathfrak{M}} \subseteq |\mathfrak{M}|^2$.

An obvious example of such a structure is the following:

1. $|\mathfrak{N}| = \mathbb{N}$
2. $\mathsf{o}^{\mathfrak{N}} = 0$
3. $\mathsf{I}^{\mathfrak{N}}(n) = n + 1$ for all $n \in \mathbb{N}$
4. $+^{\mathfrak{N}}(n, m) = n + m$ for all $n, m \in \mathbb{N}$

5. $\times^{\mathfrak{N}}(n, m) = n \cdot m$ for all $n, m \in \mathbb{N}$

6. $<^{\mathfrak{N}} = \{\langle n, m\rangle \mid n \in \mathbb{N}, m \in \mathbb{N}, n < m\}$

The structure $\mathfrak{N}$ for $\mathcal{L}_A$ so defined is called the *standard model of arithmetic*, because it interprets the non-logical constants of $\mathcal{L}_A$ exactly how you would expect.

However, there are many other possible structures for $\mathcal{L}_A$. For instance, we might take as the domain the set $\mathbb{Z}$ of integers instead of $\mathbb{N}$, and define the interpretations of 0, $'$, $+$, $\times$, $<$ accordingly. But we can also define structures for $\mathcal{L}_A$ which have nothing even remotely to do with numbers.

Example 5.28. A structure $\mathfrak{M}$ for the language $\mathcal{L}_Z$ of set theory requires just a set and a single-two place relation. So technically, e.g., the set of people plus the relation "x is older than y" could be used as a structure for $\mathcal{L}_Z$, as well as $\mathbb{N}$ together with $n \geq m$ for $n, m \in \mathbb{N}$.

A particularly interesting structure for $\mathcal{L}_Z$ in which the elements of the domain are actually sets, and the interpretation of $\in$ actually is the relation "x is an element of y" is the structure $\mathfrak{HF}$ of *hereditarily finite sets*:

1. $|\mathfrak{HF}| = \emptyset \cup \wp(\emptyset) \cup \wp(\wp(\emptyset)) \cup \wp(\wp(\wp(\emptyset))) \cup \ldots$;

2. $\in^{\mathfrak{HF}} = \{\langle x, y\rangle \mid x, y \in |\mathfrak{HF}|, x \in y\}$.

The stipulations we make as to what counts as a structure impact our logic. For example, the choice to prevent empty domains ensures, given the usual account of satisfaction (or truth) for quantified sentences, that $\exists x\, (\varphi(x) \vee \sim\varphi(x))$ is valid—that is, a logical truth. And the stipulation that all constant symbols must refer to an object in the domain ensures that the existential generalization is a sound pattern of inference: $\varphi(a)$, therefore $\exists x\, \varphi(x)$. If we allowed names to refer outside the domain, or to not refer, then we would be on our way to a *free logic*, in which existential generalization requires an additional premise: $\varphi(a)$ and $\exists x\, x = a$, therefore $\exists x\, \varphi(x)$.

5.10 Covered Structures for First-order Languages

Recall that a term is *closed* if it contains no variables.

Definition 5.29 (Value of closed terms)**.** If t is a closed term of the language $\mathcal{L}$ and $\mathfrak{M}$ is a structure for $\mathcal{L}$, the *value* $\mathrm{Val}^{\mathfrak{M}}(t)$ is defined as follows:

1. If t is just the constant symbol c, then $\mathrm{Val}^{\mathfrak{M}}(c) = c^{\mathfrak{M}}$.

2. If t is of the form $f(t_1, \ldots, t_n)$, then

$$\mathrm{Val}^{\mathfrak{M}}(t) = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}(t_n)).$$

Definition 5.30 (Covered structure)**.** A structure is *covered* if every element of the domain is the value of some closed term.

Example 5.31. Let $\mathcal{L}$ be the language with constant symbols $zero$, one, two, ..., the binary predicate symbol $<$, and the binary function symbols $+$ and $\times$. Then a structure $\mathfrak{M}$ for $\mathcal{L}$ is the one with domain $|\mathfrak{M}| = \{0, 1, 2, \ldots\}$ and assignments $zero^{\mathfrak{M}} = 0$, $one^{\mathfrak{M}} = 1$, $two^{\mathfrak{M}} = 2$, and so forth. For the binary relation symbol $<$, the set $<^{\mathfrak{M}}$ is the set of all pairs $\langle c_1, c_2\rangle \in |\mathfrak{M}|^2$ such that c_1 is less than c_2: for example, $\langle 1, 3\rangle \in <^{\mathfrak{M}}$ but $\langle 2, 2\rangle \notin <^{\mathfrak{M}}$. For the binary function symbol $+$, define $+^{\mathfrak{M}}$ in the usual way—for example, $+^{\mathfrak{M}}(2, 3)$ maps to 5, and similarly for the binary function symbol $\times$. Hence, the value of $four$ is just 4, and the value of $\times(two, +(three, zero))$ (or in infix notation, $two \times (three + zero)$) is

$$\begin{aligned}
\mathrm{Val}^{\mathfrak{M}}(\times(two, +(three, zero)) = \\
&= \times^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}(two), \mathrm{Val}^{\mathfrak{M}}(two, +(three, zero))) \\
&= \times^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}(two), +^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}(three), \mathrm{Val}^{\mathfrak{M}}(zero))) \\
&= \times^{\mathfrak{M}}(two^{\mathfrak{M}}, +^{\mathfrak{M}}(three^{\mathfrak{M}}, zero^{\mathfrak{M}})) \\
&= \times^{\mathfrak{M}}(2, +^{\mathfrak{M}}(3, 0)) \\
&= \times^{\mathfrak{M}}(2, 3) \\
&= 6
\end{aligned}$$

5.11 Satisfaction of a Formula in a Structure

The basic notion that relates expressions such as terms and formulae, on the one hand, and structures on the other, are those of *value* of a term and *satisfaction* of a formula. Informally, the value of a term is an element of a structure—if the term is just a constant, its value is the object assigned to the constant by the structure, and if it is built up using function symbols, the value is computed from the values of constants and the functions assigned to the functions in the term. A formula is *satisfied* in a structure if the interpretation given to the predicates makes the formula true in the domain of the structure. This notion of satisfaction is specified inductively: the specification of the structure directly states when atomic formulae are satisfied, and we define when a complex formula is satisfied depending on the main connective or quantifier and whether or not the immediate subformulae are satisfied. The case of the quantifiers here is a bit tricky, as the immediate subformula of a quantified formula has a free variable, and structures don't specify the values of variables. In order to deal with this difficulty, we also introduce *variable assignments* and define satisfaction not with respect to a structure alone, but with respect to a structure plus a variable assignment.

Definition 5.32 (Variable Assignment)**.** A *variable assignment* s for a structure $\mathfrak{M}$ is a function which maps each variable to an element of $|\mathfrak{M}|$, i.e., $s\colon \mathrm{Var} \to |\mathfrak{M}|$.

A structure assigns a value to each constant symbol, and a variable assignment to each variable. But we want to use terms built up from them to also name elements of the domain. For this we define the value of terms inductively. For constant symbols and variables the value is just as the structure or the variable assignment specifies it; for more complex terms it is computed recursively using the functions the structure assigns to the function symbols.

Definition 5.33 (Value of Terms)**.** If t is a term of the language $\mathcal{L}$, $\mathfrak{M}$ is a structure for $\mathcal{L}$, and s is a variable assignment for $\mathfrak{M}$, the *value* $\mathrm{Val}^{\mathfrak{M}}_s(t)$ is defined as follows:

1. $t \equiv c$: $\mathrm{Val}^{\mathfrak{M}}_s(t) = c^{\mathfrak{M}}$.
2. $t \equiv x$: $\mathrm{Val}^{\mathfrak{M}}_s(t) = s(x)$.
3. $t \equiv f(t_1, \ldots, t_n)$:
$$\mathrm{Val}^{\mathfrak{M}}_s(t) = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_s(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_s(t_n)).$$

Definition 5.34 (x-Variant)**.** If s is a variable assignment for a structure $\mathfrak{M}$, then any variable assignment s' for $\mathfrak{M}$ which differs from s at most in what it assigns to x is called an *x-variant* of s. If s' is an x-variant of s we write $s \sim_x s'$.

Note that an x-variant of an assignment s does not *have* to assign something different to x. In fact, every assignment counts as an x-variant of itself.

Definition 5.35 (Satisfaction)**.** Satisfaction of a formula φ in a structure $\mathfrak{M}$ relative to a variable assignment s, in symbols: $\mathfrak{M}, s \models \varphi$, is defined recursively as follows. (We write $\mathfrak{M}, s \not\models \varphi$ to mean "not $\mathfrak{M}, s \models \varphi$.")

1. $\varphi \equiv \bot$: $\mathfrak{M}, s \not\models \varphi$.
2. $\varphi \equiv R(t_1, \ldots, t_n)$: $\mathfrak{M}, s \models \varphi$ iff $\langle \mathrm{Val}^{\mathfrak{M}}_s(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_s(t_n)\rangle \in R^{\mathfrak{M}}$.
3. $\varphi \equiv t_1 = t_2$: $\mathfrak{M}, s \models \varphi$ iff $\mathrm{Val}^{\mathfrak{M}}_s(t_1) = \mathrm{Val}^{\mathfrak{M}}_s(t_2)$.
4. $\varphi \equiv \sim\psi$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \not\models \psi$.
5. $\varphi \equiv (\psi \,\&\, \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \models \psi$ and $\mathfrak{M}, s \models \chi$.
6. $\varphi \equiv (\psi \vee \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \models \varphi$ or $\mathfrak{M}, s \models \psi$ (or both).
7. $\varphi \equiv (\psi \supset \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \not\models \psi$ or $\mathfrak{M}, s \models \chi$ (or both).
8. $\varphi \equiv \forall x\, \psi$: $\mathfrak{M}, s \models \varphi$ iff for every x-variant s' of s, $\mathfrak{M}, s' \models \psi$.

9. $\varphi \equiv \exists x\, \psi$: $\mathfrak{M}, s \models \varphi$ iff there is an x-variant s' of s so that $\mathfrak{M}, s' \models \psi$.

The variable assignments are important in the last two clauses. We cannot define satisfaction of $\forall x\, \psi(x)$ by "for all $a \in |\mathfrak{M}|$, $\mathfrak{M} \models \psi(a)$." We cannot define satisfaction of $\exists x\, \psi(x)$ by "for at least one $a \in |\mathfrak{M}|$, $\mathfrak{M} \models \psi(a)$." The reason is that a is not symbol of the language, and so $\psi(a)$ is not a formula (that is, $\psi[a/x]$ is undefined). We also cannot assume that we have constant symbols or terms available that name every element of $\mathfrak{M}$, since there is nothing in the definition of structures that requires it. Even in the standard language the set of constant symbols is countably infinite, so if $|\mathfrak{M}|$ is not countable there aren't even enough constant symbols to name every object.

Example 5.36. Let $=\{a, b, f, R\}$ where a and b are constant symbols, f is a two-place function symbol, and A is a two-place predicate symbol. Consider the structure $\mathfrak{M}$ defined by:

1. $|\mathfrak{M}| = \{1, 2, 3, 4\}$
2. $a^{\mathfrak{M}} = 1$
3. $b^{\mathfrak{M}} = 2$
4. $f^{\mathfrak{M}}(x, y) = x + y$ if $x + y \leq 3$ and $= 3$ otherwise.
5. $R^{\mathfrak{M}} = \{\langle 1, 1\rangle, \langle 1, 2\rangle, \langle 2, 3\rangle, \langle 2, 4\rangle\}$

The function $s(x) = 1$ that assigns $1 \in |\mathfrak{M}|$ to every variable is a variable assignment for $\mathfrak{M}$.

Then

$$\mathrm{Val}^{\mathfrak{M}}_s(f(a, b)) = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_s(a), \mathrm{Val}^{\mathfrak{M}}_s(b)).$$

Since a and b are constant symbols, $\mathrm{Val}^{\mathfrak{M}}_s(a) = a^{\mathfrak{M}} = 1$ and $\mathrm{Val}^{\mathfrak{M}}_s(b) = b^{\mathfrak{M}} = 2$. So

$$\mathrm{Val}^{\mathfrak{M}}_s(f(a, b)) = f^{\mathfrak{M}}(1, 2) = 1 + 2 = 3.$$

To compute the value of $f(f(a, b), a)$ we have to consider

$$\mathrm{Val}^{\mathfrak{M}}_s(f(f(a, b), a)) = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_s(f(a, b)), \mathrm{Val}^{\mathfrak{M}}_s(a)) = f^{\mathfrak{M}}(3, 1) = 3,$$

since $3 + 1 > 3$. Since $s(x) = 1$ and $\mathrm{Val}^{\mathfrak{M}}_s(x) = s(x)$, we also have

$$\mathrm{Val}^{\mathfrak{M}}_s(f(f(a, b), x)) = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_s(f(a, b)), \mathrm{Val}^{\mathfrak{M}}_s(x)) = f^{\mathfrak{M}}(3, 1) = 3,$$

An atomic formula $R(t_1, t_2)$ is satisfied if the tuple of values of its arguments, i.e., $\langle \mathrm{Val}^{\mathfrak{M}}_s(t_1), \mathrm{Val}^{\mathfrak{M}}_s(t_2)\rangle$, is an element of $R^{\mathfrak{M}}$. So, e.g., we have

$\mathfrak{M}, s \models R(b, f(a, b))$ since $\langle \mathrm{Val}^{\mathfrak{M}}(b), \mathrm{Val}^{\mathfrak{M}}(f(a, b))\rangle = \langle 2, 3\rangle \in R^{\mathfrak{M}}$, but $\mathfrak{M} \not\models R(x, f(a, b))$ since $\langle 1, 3\rangle \notin R^{\mathfrak{M}}[s]$.

To determine if a non-atomic formula φ is satisfied, you apply the clauses in the inductive definition that applies to the main connective. For instance, the main connective in $R(a, a) \supset (R(b, x) \vee R(x, b)$ is the $\supset$, and

$$\begin{aligned}&\mathfrak{M}, s \models R(a, a) \supset (R(b, x) \vee R(x, b)) \text{ iff}\\ &\quad \mathfrak{M}, s \not\models R(a, a) \text{ or } \mathfrak{M}, s \models R(b, x) \vee R(x, b)\end{aligned}$$

Since $\mathfrak{M}, s \models R(a, a)$ (because $\langle 1, 1\rangle \in R^{\mathfrak{M}}$) we can't yet determine the answer and must first figure out if $\mathfrak{M}, s \models R(b, x) \vee R(x, b)$:

$$\begin{aligned}&\mathfrak{M}, s \models R(b, x) \vee R(x, b) \text{ iff}\\ &\quad \mathfrak{M}, s \models R(b, x) \text{ or } \mathfrak{M}, s \models R(x, b)\end{aligned}$$

And this is the case, since $\mathfrak{M}, s \models R(x, b)$ (because $\langle 1, 2\rangle \in R^{\mathfrak{M}}$).

Recall that an x-variant of s is a variable assignment that differs from s at most in what it assigns to x. For every element of $|\mathfrak{M}|$, there is an x-variant of s: $s_1(x) = 1$, $s_2(x) = 2$, $s_3(x) = 3$, $s_4(s) = 4$, and with $s_i(y) = s(y) = 1$ for all variables y other than x are all the x-variants of s for the structure $\mathfrak{M}$. Note, in particular, that $s_1 = s$ is also an x-variant of s, i.e., s is an x-variant of itself.

To determine if an existentially quantified formula $\exists x\, \varphi(x)$ is satisfied, we have to determine if $\mathfrak{M}, s' \models \varphi(x)$ for at least one x-variant s' of s. So,

$$\mathfrak{M}, s \models \exists x\, (R(b, x) \vee R(x, b)),$$

since $\mathfrak{M}, s_1 \models R(b, x) \vee R(x, b)$ (s_3 would also fit the bill). But,

$$\mathfrak{M}, s \not\models \exists x\, (R(b, x) \,\&\, R(x, b))$$

since for none of the s_i, $\mathfrak{M}, s_i \models R(b, x) \,\&\, R(x, b)$.

To determine if a universally quantified formula $\forall x\, \varphi(x)$ is satisfied, we have to determine if $\mathfrak{M}, s' \models \varphi(x)$ for all x-variants s' of s. So,

$$\mathfrak{M}, s \models \forall x\, (R(x, a) \supset R(a, x)),$$

since $\mathfrak{M}, s_i \models R(x, a) \supset R(a, x)$ for all s_i ($\mathfrak{M}, s_1 \models R(a, x)$ and $\mathfrak{M}, s_j \not\models R(a, x)$ for $j = 2$, 3, and 4). But,

$$\mathfrak{M}, s \not\models \forall x\, (R(a, x) \supset R(x, a))$$

since $\mathfrak{M}, s_2 \not\models R(a, x) \supset R(x, a)$ (because $\mathfrak{M}, s_2 \models R(a, x)$ and $\mathfrak{M}, s_2 \not\models R(x, a)$).

For a more complicated case, consider

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

Since $\mathfrak{M}, s_3 \not\models R(a,x)$ and $\mathfrak{M}, s_4 \not\models R(a,x)$, the interesting cases where we have to worry about the consequent of the conditional are only s_1 and s_2. Does $\mathfrak{M}, s_1 \models \exists y\, R(x,y)$ hold? It does if there is at least one y-variant s_1' of s_1 so that $\mathfrak{M}, s_1' \models R(x,y)$. In fact, s_1 is such a y-variant ($s_1(x) = 1$, $s_1(y) = 1$, and $\langle 1,1\rangle \in R^{\mathfrak{M}}$), so the answer is yes. To determine if $\mathfrak{M}, s_2 \models \exists y\, R(x,y)$ we have to look at the y-variants of s_2. Here, s_2 itself does not satisfy $R(x,y)$ ($s_2(x) = 2$, $s_2(y) = 1$, and $\langle 2,1\rangle \notin R^{\mathfrak{M}}$). However, consider $s_2' \sim_y s_2$ with $s_2'(y) = 3$. $\mathfrak{M}, s_2' \models R(x,y)$ since $\langle 2,3\rangle \in R^{\mathfrak{M}}$, and so $\mathfrak{M}, s_2 \models \exists y\, R(x,y)$. In sum, for every x-variant s_i of s, either $\mathfrak{M}, s_i \not\models R(a,x)$ ($i = 3, 4$) or $\mathfrak{M}, s_i \models \exists y\, R(x,y)$ ($i = 1, 2$), and so

$$\mathfrak{M}, s \models \forall x\,(R(a,x) \supset \exists y\, R(x,y)).$$

On the other hand,

$$\mathfrak{M}, s \not\models \exists x\,(R(a,x)\,\&\,\forall y\, R(x,y)).$$

The only x-variants s_i of s with $\mathfrak{M}, s_i \models R(a,x)$ are s_1 and s_2. But for each, there is in turn a y-variant $s_i' \sim_y s_i$ with $s_i'(y) = 4$ so that $\mathfrak{M}, s_i' \not\models R(x,y)$ and so $\mathfrak{M}, s_i \not\models \forall y\, R(x,y)$ for $i = 1, 2$. In sum, none of the x-variants $s_i \sim_x s$ are such that $\mathfrak{M}, s_i \models R(a,x)\,\&\,\forall y\, R(x,y)$.

5.12 Variable Assignments

A variable assignment s provides a value for *every* variable—and there are infinitely many of them. This is of course not necessary. We require variable assignments to assign values to all variables simply because it makes things a lot easier. The value of a term t, and whether or not a formula φ is satisfied in a structure with respect to s, only depend on the assignments s makes to the variables in t and the free variables of φ. This is the content of the next two propositions. To make the idea of "depends on" precise, we show that any two variable assignments that agree on all the variables in t give the same value, and that φ is satisfied relative to one iff it is satisfied relative to the other if two variable assignments agree on all free variables of φ.

Proposition 5.37. *If the variables in a term t are among $x_1, \ldots, x_n$, and $s_1(x_i) = s_2(x_i)$ for $i = 1, \ldots, n$, then $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t) = \mathrm{Val}^{\mathfrak{M}}_{s_2}(t)$.*

Proof. By induction on the complexity of t. For the base case, t can be a constant symbol or one one of the variables $x_1, \ldots, x_n$. If $t = c$, then $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t) = c^{\mathfrak{M}} = \mathrm{Val}^{\mathfrak{M}}_{s_2}(t)$. If $t = x_i$, $s_1(x_i) = s_2(x_i)$ by the hypothesis of the proposition, and so $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t) = s_1(x_i) = s_2(x_i) = \mathrm{Val}^{\mathfrak{M}}_{s_2}(t)$.

For the inductive step, assume that $t = f(t_1, \ldots, t_k)$ and that the claim holds for $t_1, \ldots, t_k$. Then

$$\begin{aligned}\mathrm{Val}^{\mathfrak{M}}_{s_1}(t) &= \mathrm{Val}^{\mathfrak{M}}_{s_1}(f(t_1, \ldots, t_k)) = \\ &= f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_{s_1}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_k))\end{aligned}$$

For $j = 1, \ldots, k$, the variables of t_j are among $x_1, \ldots, x_n$. So by induction hypothesis, $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t_j) = \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_j)$. So,

$$\begin{aligned}\mathrm{Val}^{\mathfrak{M}}_{s_1}(t) &= \mathrm{Val}^{\mathfrak{M}}_{s_2}(f(t_1, \ldots, t_k)) = \\ &= f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_{s_1}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_k)) = \\ &= f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_{s_2}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_k)) = \\ &= \mathrm{Val}^{\mathfrak{M}}_{s_2}(f(t_1, \ldots, t_k)) = \mathrm{Val}^{\mathfrak{M}}_{s_2}(t).\end{aligned}$$

□

Proposition 5.38. *If the free variables in φ are among $x_1, \ldots, x_n$, and $s_1(x_i) = s_2(x_i)$ for $i = 1, \ldots, n$, then $\mathfrak{M}, s_1 \models \varphi$ iff $\mathfrak{M}, s_2 \models \varphi$.*

Proof. We use induction on the complexity of φ. For the base case, where φ is atomic, φ can be: $\bot$, $R(t_1, \ldots, t_k)$ for a k-place predicate R and terms $t_1, \ldots, t_k$, or $t_1 = t_2$ for terms t_1 and t_2.

1. $\varphi \equiv \bot$: both $\mathfrak{M}, s_1 \nvDash \varphi$ and $\mathfrak{M}, s_2 \nvDash \varphi$.

2. $\varphi \equiv R(t_1, \ldots, t_k)$: let $\mathfrak{M}, s_1 \models \varphi$. Then

   $$\langle \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_k) \rangle \in R^{\mathfrak{M}}.$$

   For $i = 1, \ldots, k$, $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t_i) = \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_i)$ by Proposition 5.37. So we also have $\langle \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_i), \ldots, \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_k) \rangle \in R^{\mathfrak{M}}$.

3. $\varphi \equiv t_1 = t_2$: suppose $\mathfrak{M}, s_1 \models \varphi$. Then $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t_1) = \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_2)$. So,

   $$\begin{aligned}\mathrm{Val}^{\mathfrak{M}}_{s_2}(t_1) &= \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_1) && \text{(by Proposition 5.37)} \\ &= \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_2) && \text{(since } \mathfrak{M}, s_1 \models t_1 = t_2\text{)} \\ &= \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_2) && \text{(by Proposition 5.37),}\end{aligned}$$

   so $\mathfrak{M}, s_2 \models t_1 = t_2$.

Now assume $\mathfrak{M}, s_1 \models \psi$ iff $\mathfrak{M}, s_2 \models \psi$ for all formulae ψ less complex than φ. The induction step proceeds by cases determined by the main operator of φ. In each case, we only demonstrate the forward direction of the biconditional; the proof of the reverse direction is symmetrical. In all cases

except those for the quantifiers, we apply the induction hypothesis to subformulae ψ of φ. The free variables of ψ are among those of φ. Thus, if s_1 and s_2 agree on the free variables of φ, they also agree on those of ψ, and the induction hypothesis applies to ψ.

1. $\varphi \equiv \sim\psi$: if $\mathfrak{M}, s_1 \models \varphi$, then $\mathfrak{M}, s_1 \not\models \psi$, so by the induction hypothesis, $\mathfrak{M}, s_2 \not\models \psi$, hence $\mathfrak{M}, s_2 \models \varphi$.

2. $\varphi \equiv \psi \,\&\, \chi$: if $\mathfrak{M}, s_1 \models \varphi$, then $\mathfrak{M}, s_1 \models \psi$ and $\mathfrak{M}, s_1 \models \chi$, so by induction hypothesis, $\mathfrak{M}, s_2 \models \psi$ and $\mathfrak{M}, s_2 \models \chi$. Hence, $\mathfrak{M}, s_2 \models \varphi$.

3. $\varphi \equiv \psi \vee \chi$: if $\mathfrak{M}, s_1 \models \varphi$, then $\mathfrak{M}, s_1 \models \psi$ or $\mathfrak{M}, s_1 \models \chi$. By induction hypothesis, $\mathfrak{M}, s_2 \models \psi$ or $\mathfrak{M}, s_2 \models \chi$, so $\mathfrak{M}, s_2 \models \varphi$.

4. $\varphi \equiv \psi \supset \chi$: exercise.

5. $\varphi \equiv \exists x\, \psi$: if $\mathfrak{M}, s_1 \models \varphi$, there is an x-variant s_1' of s_1 so that $\mathfrak{M}, s_1' \models \psi$. Let s_2' be the x-variant of s_2 that assigns the same thing to x as does s_1'. The free variables of ψ are among $x_1, \ldots, x_n$, and x. $s_1'(x_i) = s_2'(x_i)$, since s_1' and s_2' are x-variants of s_1 and s_2, respectively, and by hypothesis $s_1(x_i) = s_2(x_i)$. $s_1'(x) = s_2'(x)$ by the way we have defined s_2'. Then the induction hypothesis applies to ψ and s_1', s_2', so $\mathfrak{M}, s_2' \models \psi$. Hence, there is an x-variant of s_2 that satisfies ψ, and so $\mathfrak{M}, s_2 \models \varphi$.

6. $\varphi \equiv \forall x\, \psi$: exercise.

By induction, we get that $\mathfrak{M}, s_1 \models \varphi$ iff $\mathfrak{M}, s_2 \models \varphi$ whenever the free variables in φ are among $x_1, \ldots, x_n$ and $s_1(x_i) = s_2(x_i)$ for $i = 1, \ldots, n$. □

Sentences have no free variables, so any two variable assignments assign the same things to all the (zero) free variables of any sentence. The proposition just proved then means that whether or not a sentence is satisfied in a structure relative to a variable assignment is completely independent of the assignment. We'll record this fact. It justifies the definition of satisfaction of a sentence in a structure (without mentioning a variable assignment) that follows.

Corollary 5.39. *If φ is a sentence and s a variable assignment, then $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s' \models \varphi$ for every variable assignment s'.*

Proof. Let s' be any variable assignment. Since φ is a sentence, it has no free variables, and so every variable assignment s' trivially assigns the same things to all free variables of φ as does s. So the condition of Proposition 5.38 is satisfied, and we have $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s' \models \varphi$. □

Definition 5.40. If φ is a sentence, we say that a structure $\mathfrak{M}$ *satisfies* φ, $\mathfrak{M} \models \varphi$, iff $\mathfrak{M}, s \models \varphi$ for all variable assignments s.

If $\mathfrak{M} \models \varphi$, we also simply say that φ *is true in* $\mathfrak{M}$.

Proposition 5.41. *Let $\mathfrak{M}$ be a structure, φ be a sentence, and s a variable assignment. $\mathfrak{M} \models \varphi$ iff $\mathfrak{M}, s \models \varphi$.*

Proof. Exercise. □

Proposition 5.42. *Suppose $\varphi(x)$ only contains x free, and $\mathfrak{M}$ is a structure. Then:*

1. *$\mathfrak{M} \models \exists x\, \varphi(x)$ iff $\mathfrak{M}, s \models \varphi(x)$ for at least one variable assignment s.*
2. *$\mathfrak{M} \models \forall x\, \varphi(x)$ iff $\mathfrak{M}, s \models \varphi(x)$ for all variable assignments s.*

Proof. Exercise. □

5.13 Extensionality

Extensionality, sometimes called relevance, can be expressed informally as follows: the only thing that bears upon the satisfaction of formula φ in a structure $\mathfrak{M}$ relative to a variable assignment s, are the assignments made by $\mathfrak{M}$ and s to the elements of the language that actually appear in φ.

One immediate consequence of extensionality is that where two structures $\mathfrak{M}$ and $\mathfrak{M}'$ agree on all the elements of the language appearing in a sentence φ and have the same domain, $\mathfrak{M}$ and $\mathfrak{M}'$ must also agree on whether or not φ itself is true.

Proposition 5.43 (Extensionality)**.** *Let φ be a formula, and $\mathfrak{M}_1$ and $\mathfrak{M}_2$ be structures with $|\mathfrak{M}_1| = |\mathfrak{M}_2|$, and s a variable assignment on $|\mathfrak{M}_1| = |\mathfrak{M}_2|$. If $c^{\mathfrak{M}_1} = c^{\mathfrak{M}_2}$, $R^{\mathfrak{M}_1} = R^{\mathfrak{M}_2}$, and $f^{\mathfrak{M}_1} = f^{\mathfrak{M}_2}$ for every constant symbol c, relation symbol R, and function symbol f occurring in φ, then $\mathfrak{M}_1, s \models \varphi$ iff $\mathfrak{M}_2, s \models \varphi$.*

Proof. First prove (by induction on t) that for every term, $\mathrm{Val}_s^{\mathfrak{M}_1}(t) = \mathrm{Val}_s^{\mathfrak{M}_2}(t)$. Then prove the proposition by induction on φ, making use of the claim just proved for the induction basis (where φ is atomic). □

Corollary 5.44 (Extensionality for Sentences)**.** *Let φ be a sentence and $\mathfrak{M}_1$, $\mathfrak{M}_2$ as in Proposition 5.43. Then $\mathfrak{M}_1 \models \varphi$ iff $\mathfrak{M}_2 \models \varphi$.*

Proof. Follows from Proposition 5.43 by Corollary 5.39. □

Moreover, the value of a term, and whether or not a structure satisfies a formula, only depends on the values of its subterms.

Proposition 5.45. *Let $\mathfrak{M}$ be a structure, t and t' terms, and s a variable assignment. Let $s' \sim_x s$ be the x-variant of s given by $s'(x) = \mathrm{Val}_s^{\mathfrak{M}}(t')$. Then $\mathrm{Val}_s^{\mathfrak{M}}(t[t'/x]) = \mathrm{Val}_{s'}^{\mathfrak{M}}(t)$.*

Proof. By induction on t.

1. If t is a constant, say, $t \equiv c$, then $t[t'/x] = c$, and $\mathrm{Val}^{\mathfrak{M}}_s(c) = c^{\mathfrak{M}} = \mathrm{Val}^{\mathfrak{M}}_{s'}(c)$.
2. If t is a variable other than x, say, $t \equiv y$, then $t[t'/x] = y$, and $\mathrm{Val}^{\mathfrak{M}}_s(y) = \mathrm{Val}^{\mathfrak{M}}_{s'}(y)$ since $s' \sim_x s$.
3. If $t \equiv x$, then $t[t'/x] = t'$. But $\mathrm{Val}^{\mathfrak{M}}_{s'}(x) = \mathrm{Val}^{\mathfrak{M}}_s(t')$ by definition of s'.
4. If $t \equiv f(t_1, \ldots, t_n)$ then we have:

$$\begin{aligned}
\mathrm{Val}^{\mathfrak{M}}_s(t[t'/x]) = & \\
& = \mathrm{Val}^{\mathfrak{M}}_s(f(t_1[t'/x], \ldots, t_n[t'/x])) \\
& \qquad \text{by definition of } t[t'/x] \\
& = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_s(t_1[t'/x]), \ldots, \mathrm{Val}^{\mathfrak{M}}_s(t_n[t'/x])) \\
& \qquad \text{by definition of } \mathrm{Val}^{\mathfrak{M}}_s(f(\ldots)) \\
& = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_{s'}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_{s'}(t_n)) \\
& \qquad \text{by induction hypothesis} \\
& = \mathrm{Val}^{\mathfrak{M}}_{s'}(t) \text{ by definition of } \mathrm{Val}^{\mathfrak{M}}_{s'}(f(\ldots))
\end{aligned}$$

□

Proposition 5.46. *Let $\mathfrak{M}$ be a structure, φ a formula, t a term, and s a variable assignment. Let $s' \sim_x s$ be the x-variant of s given by $s'(x) = \mathrm{Val}^{\mathfrak{M}}_s(t)$. Then $\mathfrak{M}, s \models \varphi[t/x]$ iff $\mathfrak{M}, s' \models \varphi$.*

Proof. Exercise. □

5.14 Semantic Notions

Give the definition of structures for first-order languages, we can define some basic semantic properties of and relationships between sentences. The simplest of these is the notion of *validity* of a sentence. A sentence is valid if it is satisfied in every structure. Valid sentences are those that are satisfied regardless of how the non-logical symbols in it are interpreted. Valid sentences are therefore also called *logical truths*—they are true, i.e., satisfied, in any structure and hence their truth depends only on the logical symbols occurring in them and their syntactic structure, but not on the non-logical symbols or their interpretation.

Definition 5.47 (Validity)**.** A sentence φ is *valid*, $\vDash \varphi$, iff $\mathfrak{M} \models \varphi$ for every structure $\mathfrak{M}$.

Definition 5.48 (Entailment)**.** A set of sentences Γ *entails* a sentence φ, $\Gamma \vDash \varphi$, iff for every structure $\mathfrak{M}$ with $\mathfrak{M} \models \Gamma$, $\mathfrak{M} \models \varphi$.

Definition 5.49 (Satisfiability)**.** A set of sentences Γ is *satisfiable* if $\mathfrak{M} \models \Gamma$ for some structure $\mathfrak{M}$. If Γ is not satisfiable it is called *unsatisfiable*.

Proposition 5.50. *A sentence φ is valid iff $\Gamma \vDash \varphi$ for every set of sentences Γ.*

Proof. For the forward direction, let φ be valid, and let Γ be a set of sentences. Let $\mathfrak{M}$ be a structure so that $\mathfrak{M} \models \Gamma$. Since φ is valid, $\mathfrak{M} \models \varphi$, hence $\Gamma \vDash \varphi$.

For the contrapositive of the reverse direction, let φ be invalid, so there is a structure $\mathfrak{M}$ with $\mathfrak{M} \not\models \varphi$. When $\Gamma = \{\top\}$, since $\top$ is valid, $\mathfrak{M} \models \Gamma$. Hence, there is a structure $\mathfrak{M}$ so that $\mathfrak{M} \models \Gamma$ but $\mathfrak{M} \not\models \varphi$, hence Γ does not entail φ. □

Proposition 5.51. *$\Gamma \vDash \varphi$ iff $\Gamma \cup \{\sim\varphi\}$ is unsatisfiable.*

Proof. For the forward direction, suppose $\Gamma \vDash \varphi$ and suppose to the contrary that there is a structure $\mathfrak{M}$ so that $\mathfrak{M} \models \Gamma \cup \{\sim\varphi\}$. Since $\mathfrak{M} \models \Gamma$ and $\Gamma \vDash \varphi$, $\mathfrak{M} \models \varphi$. Also, since $\mathfrak{M} \models \Gamma \cup \{\sim\varphi\}$, $\mathfrak{M} \models \sim\varphi$, so we have both $\mathfrak{M} \models \varphi$ and $\mathfrak{M} \not\models \varphi$, a contradiction. Hence, there can be no such structure $\mathfrak{M}$, so $\Gamma \cup \{\varphi\}$ is unsatisfiable.

For the reverse direction, suppose $\Gamma \cup \{\sim\varphi\}$ is unsatisfiable. So for every structure $\mathfrak{M}$, either $\mathfrak{M} \not\models \Gamma$ or $\mathfrak{M} \models \varphi$. Hence, for every structure $\mathfrak{M}$ with $\mathfrak{M} \models \Gamma$, $\mathfrak{M} \models \varphi$, so $\Gamma \vDash \varphi$. □

Proposition 5.52. *If $\Gamma \subseteq \Gamma'$ and $\Gamma \vDash \varphi$, then $\Gamma' \vDash \varphi$.*

Proof. Suppose that $\Gamma \subseteq \Gamma'$ and $\Gamma \vDash \varphi$. Let $\mathfrak{M}$ be such that $\mathfrak{M} \models \Gamma'$; then $\mathfrak{M} \models \Gamma$, and since $\Gamma \vDash \varphi$, we get that $\mathfrak{M} \models \varphi$. Hence, whenever $\mathfrak{M} \models \Gamma'$, $\mathfrak{M} \models \varphi$, so $\Gamma' \vDash \varphi$. □

Theorem 5.53 (Semantic Deduction Theorem)**.** *$\Gamma \cup \{\varphi\} \vDash \psi$ iff $\Gamma \vDash \varphi \supset \psi$.*

Proof. For the forward direction, let $\Gamma \cup \{\varphi\} \vDash \psi$ and let $\mathfrak{M}$ be a structure so that $\mathfrak{M} \models \Gamma$. If $\mathfrak{M} \models \varphi$, then $\mathfrak{M} \models \Gamma \cup \{\varphi\}$, so since $\Gamma \cup \{\varphi\}$ entails ψ, we get $\mathfrak{M} \models \psi$. Therefore, $\mathfrak{M} \models \varphi \supset \psi$, so $\Gamma \vDash \varphi \supset \psi$.

For the reverse direction, let $\Gamma \vDash \varphi \supset \psi$ and $\mathfrak{M}$ be a structure so that $\mathfrak{M} \models \Gamma \cup \{\varphi\}$. Then $\mathfrak{M} \models \Gamma$, so $\mathfrak{M} \models \varphi \supset \psi$, and since $\mathfrak{M} \models \varphi$, $\mathfrak{M} \models \psi$. Hence, whenever $\mathfrak{M} \models \Gamma \cup \{\varphi\}$, $\mathfrak{M} \models \psi$, so $\Gamma \cup \{\varphi\} \vDash \psi$. □

Proposition 5.54. *Let $\mathfrak{M}$ be a structure, and $\varphi(x)$ a formula with one free variable x, and t a closed term. Then:*

1. $\varphi(t) \vDash \exists x\, \varphi(x)$
2. $\forall x\, \varphi(x) \vDash \varphi(t)$

Proof. 1. Suppose $\mathfrak{M} \models \varphi(t)$. Let s be a variable assignment with $s(x) = \mathrm{Val}^{\mathfrak{M}}(t)$. Then $\mathfrak{M}, s \models \varphi(t)$ since $\varphi(t)$ is a sentence. By Proposition 5.46, $\mathfrak{M}, s \models \varphi(x)$. By Proposition 5.42, $\mathfrak{M} \models \exists x\, \varphi(x)$.

2. Exercise.

□

Chapter 6

Theories and Their Models

6.1 Introduction

The development of the axiomatic method is a significant achievement in the history of science, and is of special importance in the history of mathematics. An axiomatic development of a field involves the clarification of many questions: What is the field about? What are the most fundamental concepts? How are they related? Can all the concepts of the field be defined in terms of these fundamental concepts? What laws do, and must, these concepts obey?

The axiomatic method and logic were made for each other. Formal logic provides the tools for formulating axiomatic theories, for proving theorems from the axioms of the theory in a precisely specified way, for studying the properties of all systems satisfying the axioms in a systematic way.

Definition 6.1. A set of sentences Γ is *closed* iff, whenever $\Gamma \vDash \varphi$ then $\varphi \in \Gamma$. The *closure* of a set of sentences Γ is $\{\varphi \mid \Gamma \vDash \varphi\}$.

We say that Γ is *axiomatized by* a set of sentences Δ if Γ is the closure of Δ

We can think of an axiomatic theory as the set of sentences that is axiomatized by its set of axioms Δ. In other words, when we have a first-order language which contains non-logical symbols for the primitives of the axiomatically developed science we wish to study, together with a set of sentences that express the fundamental laws of the science, we can think of the theory as represented by all the sentences in this language that are entailed by the axioms. This ranges from simple examples with only a single primitive and simple axioms, such as the theory of partial orders, to complex theories such as Newtonian mechanics.

The important logical facts that make this formal approach to the axiomatic method so important are the following. Suppose Γ is an axiom system for a theory, i.e., a set of sentences.

1. We can state precisely when an axiom system captures an intended class of structures. That is, if we are interested in a certain class of structures, we will successfully capture that class by an axiom system Γ iff the structures are exactly those $\mathfrak{M}$ such that $\mathfrak{M} \models \Gamma$.

2. We may fail in this respect because there are $\mathfrak{M}$ such that $\mathfrak{M} \models \Gamma$, but $\mathfrak{M}$ is not one of the structures we intend. This may lead us to add axioms which are not true in $\mathfrak{M}$.

3. If we are successful at least in the respect that Γ is true in all the intended structures, then a sentence φ is true in all intended structures whenever $\Gamma \vDash \varphi$. Thus we can use logical tools (such as proof methods) to show that sentences are true in all intended structures simply by showing that they are entailed by the axioms.

4. Sometimes we don't have intended structures in mind, but instead start from the axioms themselves: we begin with some primitives that we want to satisfy certain laws which we codify in an axiom system. One thing that we would like to verify right away is that the axioms do not contradict each other: if they do, there can be no concepts that obey these laws, and we have tried to set up an incoherent theory. We can verify that this doesn't happen by finding a model of Γ. And if there are models of our theory, we can use logical methods to investigate them, and we can also use logical methods to construct models.

5. The independence of the axioms is likewise an important question. It may happen that one of the axioms is actually a consequence of the others, and so is redundant. We can prove that an axiom φ in Γ is redundant by proving $\Gamma \setminus \{\varphi\} \vDash \varphi$. We can also prove that an axiom is not redundant by showing that $(\Gamma \setminus \{\varphi\}) \cup \{\sim\varphi\}$ is satisfiable. For instance, this is how it was shown that the parallel postulate is independent of the other axioms of geometry.

6. Another important question is that of definability of concepts in a theory: The choice of the language determines what the models of a theory consists of. But not every aspect of a theory must be represented separately in its models. For instance, every ordering $\leq$ determines a corresponding strict ordering $<$—given one, we can define the other. So it is not necessary that a model of a theory involving such an order must *also* contain the corresponding strict ordering. When is it the case, in general, that one relation can be defined in terms of others? When is it impossible to define a relation in terms of other (and hence must add it to the primitives of the language)?

6.2 Expressing Properties of Structures

It is often useful and important to express conditions on functions and relations, or more generally, that the functions and relations in a structure satisfy these conditions. For instance, we would like to have ways of distinguishing those structures for a language which "capture" what we want the predicate symbols to "mean" from those that do not. Of course we're completely free to specify which structures we "intend," e.g., we can specify that the interpretation of the predicate symbol $\leq$ must be an ordering, or that we are only interested in interpretations of $\mathcal{L}$ in which the domain consists of sets and $\in$ is interpreted by the "is an element of" relation. But can we do this with sentences of the language? In other words, which conditions on a structure $\mathfrak{M}$ can we express by a sentence (or perhaps a set of sentences) in the language of $\mathfrak{M}$? There are some conditions that we will not be able to express. For instance, there is no sentence of $\mathcal{L}_A$ which is only true in a structure $\mathfrak{M}$ if $|\mathfrak{M}| = \mathbb{N}$. We cannot express "the domain contains only natural numbers." But there are "structural properties" of structures that we perhaps can express. Which properties of structures can we express by sentences? Or, to put it another way, which collections of structures can we describe as those making a sentence (or set of sentences) true?

Definition 6.2 (Model of a set)**.** Let Γ be a set of sentences in a language $\mathcal{L}$. We say that a structure $\mathfrak{M}$ *is a model of* Γ if $\mathfrak{M} \models \varphi$ for all $\varphi \in \Gamma$.

Example 6.3. The sentence $\forall x\, x \leq x$ is true in $\mathfrak{M}$ iff $\leq^{\mathfrak{M}}$ is a reflexive relation. The sentence $\forall x \forall y\, ((x \leq y \,\&\, y \leq x) \supset x = y)$ is true in $\mathfrak{M}$ iff $\leq^{\mathfrak{M}}$ is anti-symmetric. The sentence $\forall x \forall y \forall z\, ((x \leq y \,\&\, y \leq z) \supset x \leq z)$ is true in $\mathfrak{M}$ iff $\leq^{\mathfrak{M}}$ is transitive. Thus, the models of

$$\begin{aligned} \{ \quad & \forall x\, x \leq x, \\ & \forall x \forall y\, ((x \leq y \,\&\, y \leq x) \supset x = y), \\ & \forall x \forall y \forall z\, ((x \leq y \,\&\, y \leq z) \supset x \leq z) \quad \} \end{aligned}$$

are exactly those structures in which $\leq^{\mathfrak{M}}$ is reflexive, anti-symmetric, and transitive, i.e., a partial order. Hence, we can take them as axioms for the *first-order theory of partial orders*.

6.3 Examples of First-Order Theories

Example 6.4. The theory of strict linear orders in the language $\mathcal{L}_<$ is axiomatized by the set

$$\begin{aligned} & \forall x \sim x < x, \\ & \forall x \forall y\, ((x < y \vee y < x) \vee x = y), \\ & \forall x \forall y \forall z\, ((x < y \,\&\, y < z) \supset x < z) \end{aligned}$$

It completely captures the intended structures: every strict linear order is a model of this axiom system, and vice versa, if R is a linear order on a set X, then the structure $\mathfrak{M}$ with $|\mathfrak{M}| = X$ and $<^{\mathfrak{M}} = R$ is a model of this theory.

Example 6.5. The theory of groups in the language 1 (constant symbol), $\cdot$ (two-place function symbol) is axiomatized by

$$\begin{aligned}
&\forall x\,(x \cdot 1) = x\\
&\forall x\,\forall y\,\forall z\,(x \cdot (y \cdot z)) = ((x \cdot y) \cdot z)\\
&\forall x\,\exists y\,(x \cdot y) = 1
\end{aligned}$$

Example 6.6. The theory of Peano arithmetic is axiomatized by the following sentences in the language of arithmetic $\mathcal{L}_A$.

$$\begin{aligned}
&\sim\exists x\,x' = 0\\
&\forall x\,\forall y\,(x' = y' \supset x = y)\\
&\forall x\,\forall y\,(x < y \equiv \exists z\,(x + z' = y))\\
&\forall x\,(x + 0) = x\\
&\forall x\,\forall y\,(x + y') = (x + y)'\\
&\forall x\,(x \times 0) = 0\\
&\forall x\,\forall y\,(x \times y') = ((x \times y) + x)
\end{aligned}$$

plus all sentences of the form

$$(\varphi(0)\,\&\,\forall x\,(\varphi(x) \supset \varphi(x'))) \supset \forall x\,\varphi(x)$$

Since there are infinitely many sentences of the latter form, this axiom system is infinite. The latter form is called the *induction schema*. (Actually, the induction schema is a bit more complicated than we let on here.)

The third axiom is an *explicit definition* of $<$.

Example 6.7. The theory of pure sets plays an important role in the foundations (and in the philosophy) of mathematics. A set is pure if all its elements are also pure sets. The empty set counts therefore as pure, but a set that has something as an element that is not a set would not be pure. So the pure sets are those that are formed just from the empty set and no "urelements," i.e., objects that are not themselves sets.

The following might be considered as an axiom system for a theory of pure sets:

$$\begin{aligned}
&\exists x \sim\exists y\,y \in x\\
&\forall x\,\forall y\,(\forall z(z \in x \equiv z \in y) \supset x = y)\\
&\forall x\,\forall y\,\exists z\,\forall u\,(u \in z \equiv (u = x \vee u = y))\\
&\forall x\,\exists y\,\forall z\,(z \in y \equiv \exists u\,(z \in u\,\&\,u \in x))
\end{aligned}$$

plus all sentences of the form

$$\exists x \, \forall y \, (y \in x \equiv \varphi(y))$$

The first axiom says that there is a set with no elements (i.e., $\emptyset$ exists); the second says that sets are extensional; the third that for any sets X and Y, the set $\{X, Y\}$ exists; the fourth that for any sets X and Y, the set $X \cup Y$ exists.

The sentences mentioned last are collectively called the *naive comprehension scheme*. It essentially says that for every $\varphi(x)$, the set $\{x \mid \varphi(x)\}$ exists—so at first glance a true, useful, and perhaps even necessary axiom. It is called "naive" because, as it turns out, it makes this theory unsatisfiable: if you take $\varphi(y)$ to be $\sim y \in y$, you get the sentence

$$\exists x \, \forall y \, (y \in x \equiv \sim y \in y)$$

and this sentence is not satisfied in any structure.

Example 6.8. In the area of *mereology*, the relation of *parthood* is a fundamental relation. Just like theories of sets, there are theories of parthood that axiomatize various conceptions (sometimes conflicting) of this relation.

The language of mereology contains a single two-place predicate symbol P, and $P(x, y)$ "means" that x is a part of y. When we have this interpretation in mind, a structure for this language is called a *parthood structure*. Of course, not every structure for a single two-place predicate will really deserve this name. To have a chance of capturing "parthood," $P^{\mathfrak{M}}$ must satisfy some conditions, which we can lay down as axioms for a theory of parthood. For instance, parthood is a partial order on objects: every object is a part (albeit an *improper* part) of itself; no two different objects can be parts of each other; a part of a part of an object is itself part of that object. Note that in this sense "is a part of" resembles "is a subset of," but does not resemble "is an element of" which is neither reflexive nor transitive.

$$\begin{aligned}
&\forall x \, P(x, x), \\
&\forall x \, \forall y \, ((P(x, y) \mathbin{\&} P(y, x)) \supset x = y), \\
&\forall x \, \forall y \, \forall z \, ((P(x, y) \mathbin{\&} P(y, z)) \supset P(x, z)),
\end{aligned}$$

Moreover, any two objects have a mereological sum (an object that has these two objects as parts, and is minimal in this respect).

$$\forall x \, \forall y \, \exists z \, \forall u \, (P(z, u) \equiv (P(x, u) \mathbin{\&} P(y, u)))$$

These are only some of the basic principles of parthood considered by metaphysicians. Further principles, however, quickly become hard to formulate or write down without first introducting some defined relations. For instance, most metaphysicians interested in mereology also view the following as a valid principle: whenever an object x has a proper part y, it also has a part z that has no parts in common with y, and so that the fusion of y and z is x.

6.4 Expressing Relations in a Structure

One main use formulae can be put to is to express properties and relations in a structure $\mathfrak{M}$ in terms of the primitives of the language $\mathcal{L}$ of $\mathfrak{M}$. By this we mean the following: the domain of $\mathfrak{M}$ is a set of objects. The constant symbols, function symbols, and predicate symbols are interpreted in $\mathfrak{M}$ by some objects in$|\mathfrak{M}|$, functions on $|\mathfrak{M}|$, and relations on $|\mathfrak{M}|$. For instance, if A^2_0 is in $\mathcal{L}$, then $\mathfrak{M}$ assigns to it a relation $R = A^{2\mathfrak{M}}_0$. Then the formula $A^2_0(v_1, v_2)$ *expresses* that very relation, in the following sense: if a variable assignment s maps v_1 to $a \in |\mathfrak{M}|$ and v_2 to $b \in |\mathfrak{M}|$, then

$$Rab \quad \text{iff} \quad \mathfrak{M}, s \models A^2_0(v_1, v_2).$$

Note that we have to involve variable assignments here: we can't just say "Rab iff $\mathfrak{M} \models A^2_0(a, b)$" because a and b are not symbols of our language: they are elements of $|\mathfrak{M}|$.

Since we don't just have atomic formulae, but can combine them using the logical connectives and the quantifiers, more complex formulae can define other relations which aren't directly built into $\mathfrak{M}$. We're interested in how to do that, and specifically, which relations we can define in a structure.

Definition 6.9. Let $\varphi(v_1, \ldots, v_n)$ be a formula of $\mathcal{L}$ in which only $v_1, \ldots, v_n$ occur free, and let $\mathfrak{M}$ be a structure for $\mathcal{L}$. $\varphi(v_1, \ldots, v_n)$ *expresses the relation* $R \subseteq |\mathfrak{M}|^n$ iff

$$Ra_1 \ldots a_n \quad \text{iff} \quad \mathfrak{M}, s \models \varphi(v_1, \ldots, v_n)$$

for any variable assignment s with $s(v_i) = a_i$ $(i = 1, \ldots, n)$.

Example 6.10. In the standard model of arithmetic $\mathfrak{N}$, the formula $v_1 < v_2 \vee v_1 = v_2$ expresses the $\leq$ relation on $\mathbb{N}$. The formula $v_2 = v_1'$ expresses the successor relation, i.e., the relation $R \subseteq \mathbb{N}^2$ where Rnm holds if m is the successor of n. The formula $v_1 = v_2'$ expresses the predecessor relation. The formulae $\exists v_3\,(v_3 \neq 0 \,\&\, v_2 = (v_1 + v_3))$ and $\exists v_3\,(v_1 + v_3') = v_2$ both express the $<$ relation. This means that the predicate symbol $<$ is actually superfluous in the language of arithmetic; it can be defined.

This idea is not just interesting in specific structures, but generally whenever we use a language to describe an intended model or models, i.e., when we consider theories. These theories often only contain a few predicate symbols as basic symbols, but in the domain they are used to describe often many other relations play an important role. If these other relations can be systematically expressed by the relations that interpret the basic predicate symbols of the language, we say we can *define* them in the language.

6.5 The Theory of Sets

Almost all of mathematics can be developed in the theory of sets. Developing mathematics in this theory involves a number of things. First, it requires a set of axioms for the relation $\in$. A number of different axiom systems have been developed, sometimes with conflicting properties of $\in$. The axiom system known as **ZFC**, Zermelo-Fraenkel set theory with the axiom of choice stands out: it is by far the most widely used and studied, because it turns out that its axioms suffice to prove almost all the things mathematicians expect to be able to prove. But before that can be established, it first is necessary to make clear how we can even *express* all the things mathematicians would like to express. For starters, the language contains no constant symbols or function symbols, so it seems at first glance unclear that we can talk about particular sets (such as $\emptyset$ or $\mathbb{N}$), can talk about operations on sets (such as $X \cup Y$ and $\wp(X)$), let alone other constructions which involve things other than sets, such as relations and functions.

To begin with, "is an element of" is not the only relation we are interested in: "is a subset of" seems almost as important. But we can *define* "is a subset of" in terms of "is an element of." To do this, we have to find a formula $\varphi(x, y)$ in the language of set theory which is satisfied by a pair of sets $\langle X, Y \rangle$ iff $X \subseteq Y$. But X is a subset of Y just in case all elements of X are also elements of Y. So we can define $\subseteq$ by the formula

$$\forall z\,(z \in x \supset z \in y)$$

Now, whenever we want to use the relation $\subseteq$ in a formula, we could instead use that formula (with x and y suitably replaced, and the bound variable z renamed if necessary). For instance, extensionality of sets means that if any sets x and y are contained in each other, then x and y must be the same set. This can be expressed by $\forall x\,\forall y\,((x \subseteq y \,\&\, y \subseteq x) \supset x = y)$, or, if we replace $\subseteq$ by the above definition, by

$$\forall x\,\forall y\,((\forall z\,(z \in x \supset z \in y) \,\&\, \forall z\,(z \in y \supset z \in x)) \supset x = y).$$

This is in fact one of the axioms of **ZFC**, the "axiom of extensionality."

There is no constant symbol for $\emptyset$, but we can express "x is empty" by $\sim\exists y\, y \in x$. Then "$\emptyset$ exists" becomes the sentence $\exists x \sim\exists y\, y \in x$. This is another axiom of **ZFC**. (Note that the axiom of extensionality implies that there is only one empty set.) Whenever we want to talk about $\emptyset$ in the language of set theory, we would write this as "there is a set that's empty and ..." As an example, to express the fact that $\emptyset$ is a subset of every set, we could write

$$\exists x\,(\sim\exists y\, y \in x \,\&\, \forall z\, x \subseteq z)$$

where, of course, $x \subseteq z$ would in turn have to be replaced by its definition.

To talk about operations on sets, such has $X \cup Y$ and $\wp(X)$, we have to use a similar trick. There are no function symbols in the language of set theory, but we can express the functional relations $X \cup Y = Z$ and $\wp(X) = Y$ by

$$\forall u\,((u \in x \vee u \in y) \equiv u \in z)$$
$$\forall u\,(u \subseteq x \equiv u \in y)$$

since the elements of $X \cup Y$ are exactly the sets that are either elements of X or elements of Y, and the elements of $\wp(X)$ are exactly the subsets of X. However, this doesn't allow us to use $x \cup y$ or $\wp(x)$ as if they were terms: we can only use the entire formulae that define the relations $X \cup Y = Z$ and $\wp(X) = Y$. In fact, we do not know that these relations are ever satisfied, i.e., we do not know that unions and power sets always exist. For instance, the sentence $\forall x\,\exists y\,\wp(x) = y$ is another axiom of **ZFC** (the power set axiom).

Now what about talk of ordered pairs or functions? Here we have to explain how we can think of ordered pairs and functions as special kinds of sets. One way to define the ordered pair $\langle x, y\rangle$ is as the set $\{\{x\}, \{x, y\}\}$. But like before, we cannot introduce a function symbol that names this set; we can only define the relation $\langle x, y\rangle = z$, i.e., $\{\{x\}, \{x, y\}\} = z$:

$$\forall u\,(u \in z \equiv (\forall v\,(v \in u \equiv v = x) \vee \forall v\,(v \in u \equiv (v = x \vee v = y))))$$

This says that the elements u of z are exactly those sets which either have x as its only element or have x and y as its only elements (in other words, those sets that are either identical to $\{x\}$ or identical to $\{x, y\}$). Once we have this, we can say further things, e.g., that $X \times Y = Z$:

$$\forall z\,(z \in Z \equiv \exists x\,\exists y\,(x \in X \,\&\, y \in Y \,\&\, \langle x, y\rangle = z))$$

A function $f\colon X \to Y$ can be thought of as the relation $f(x) = y$, i.e., as the set of pairs $\{\langle x, y\rangle \mid f(x) = y\}$. We can then say that a set f is a function from X to Y if (a) it is a relation $\subseteq X \times Y$, (b) it is total, i.e., for all $x \in X$ there is some $y \in Y$ such that $\langle x, y\rangle \in f$ and (c) it is functional, i.e., whenever $\langle x, y\rangle, \langle x, y'\rangle \in f$, $y = y'$ (because values of functions must be unique). So "f is a function from X to Y" can be written as:

$$\begin{aligned}&\forall u\,(u \in f \supset \exists x\,\exists y\,(x \in X \,\&\, y \in Y \,\&\, \langle x, y\rangle = u)) \,\&\\ &\forall x\,(x \in X \supset (\exists y\,(y \in Y \,\&\, \mathrm{maps}(f, x, y)) \,\&\\ &\qquad\qquad (\forall y\,\forall y'\,((\mathrm{maps}(f, x, y) \,\&\, \mathrm{maps}(f, x, y')) \supset y = y')))\end{aligned}$$

where $\mathrm{maps}(f, x, y)$ abbreviates $\exists v\,(v \in f \,\&\, \langle x, y\rangle = v)$ (this formula expresses "$f(x) = y$").

It is now also not hard to express that $f\colon X \to Y$ is injective, for instance:

$$\begin{aligned}f\colon X \to Y \,\&\, \forall x\,\forall x'\,((x \in X \,\&\, x' \in X \,\&\,&\\ \exists y\,(\mathrm{maps}(f, x, y) \,\&\, \mathrm{maps}(f, x', y))) \supset x = x')&\end{aligned}$$

A function $f\colon X \to Y$ is injective iff, whenever f maps $x, x' \in X$ to a single y, $x = x'$. If we abbreviate this formula as $\mathrm{inj}(f, X, Y)$, we're already in a position to state in the language of set theory something as non-trivial as Cantor's theorem: there is no injective function from $\wp(X)$ to X:

$$\forall X \forall Y \, (\wp(X) = Y \supset \sim \exists f \, \mathrm{inj}(f, Y, X))$$

One might think that set theory requires another axiom that guarantees the existence of a set for every defining property. If $\varphi(x)$ is a formula of set theory with the variable x free, we can consider the sentence

$$\exists y \forall x \, (x \in y \equiv \varphi(x)).$$

This sentence states that there is a set y whose elements are all and only those x that satisfy $\varphi(x)$. This schema is called the "comprehension principle." It looks very useful; unfortunately it is inconsistent. Take $\varphi(x) \equiv \sim x \in x$, then the comprehension principle states

$$\exists y \forall x \, (x \in y \equiv x \notin x),$$

i.e., it states the existence of a set of all sets that are not elements of themselves. No such set can exist—this is Russell's Paradox. **ZFC**, in fact, contains a restricted—and consistent—version of this principle, the separation principle:

$$\forall z \, \exists y \forall x \, (x \in y \equiv (x \in z \,\&\, \varphi(x)).$$

6.6 Expressing the Size of Structures

There are some properties of structures we can express even without using the non-logical symbols of a language. For instance, there are sentences which are true in a structure iff the domain of the structure has at least, at most, or exactly a certain number n of elements.

Proposition 6.11. *The sentence*

$$\begin{aligned} !A_{\geq n} \equiv \exists x_1 \, \exists x_2 \, \ldots \exists x_n \quad (x_1 \neq x_2 \,\&\, x_1 \neq x_3 \,\&\, x_1 \neq x_4 \,\&\, \cdots \,\&\, x_1 \neq x_n \,\&\\ x_2 \neq x_3 \,\&\, x_2 \neq x_4 \,\&\, \cdots \,\&\, x_2 \neq x_n \,\&\\ \vdots \\ x_{n-1} \neq x_n) \end{aligned}$$

is true in a structure $\mathfrak{M}$ iff $|\mathfrak{M}|$ contains at least n elements. Consequently, $\mathfrak{M} \models \sim \varphi_{\geq n+1}$ iff $|\mathfrak{M}|$ contains at most n elements.

Proposition 6.12. *The sentence*

$$!A_{=n} \equiv \exists x_1 \exists x_2 \dots \exists x_n \quad (x_1 \neq x_2 \,\&\, x_1 \neq x_3 \,\&\, x_1 \neq x_4 \,\&\, \dots \,\&\, x_1 \neq x_n \,\&\, x_2 \neq x_3 \,\&\, x_2 \neq x_4 \,\&\, \dots \,\&\, x_2 \neq x_n \,\&\, \vdots \, x_{n-1} \neq x_n \,\&\, \forall y\,(y = x_1 \vee \dots y = x_n) \dots))$$

is true in a structure $\mathfrak{M}$ iff $|\mathfrak{M}|$ contains exactly n elements.

Proposition 6.13. *A structure is infinite iff it is a model of*

$$\{\varphi_{\geq 1}, \varphi_{\geq 2}, \varphi_{\geq 3}, \dots\}$$

There is no single purely logical sentence which is true in $\mathfrak{M}$ iff $|\mathfrak{M}|$ is infinite. However, one can give sentences with non-logical predicate symbols which only have infinite models (although not every infinite structure is a model of them). The property of being a finite structure, and the property of being a uncountable structure cannot even be expressed with an infinite set of sentences. These facts follow from the compactness and Löwenheim-Skolem theorems.

Chapter 7

Natural Deduction

7.1 Rules and Derivations

Natural deduction systems are meant to closely parallel the informal reasoning used in mathematical proof (hence it is somewhat "natural"). Natural deduction proofs begin with assumptions. Inference rules are then applied. Assumptions are "discharged" by the $\sim$Intro, $\supset$Intro, $\vee$Elim and $\exists$Elim inference rules, and the label of the discharged assumption is placed beside the inference for clarity.

Definition 7.1 (Initial Formula)**.** An *initial formula* or *assumption* is any formula in the topmost position of any branch.

Derivations in natural deduction are certain trees of sentences, where the topmost sentences are assumptions, and if a sentence stands below one, two, or three other sequents, it must follow correctly by a rule of inference. The sentences at the top of the inference are called the *premises* and the sentence below the *conclusion* of the inference. The rules come in pairs, an introduction and an elimination rule for each logical operator. They introduce a logical operator in the conclusion or remove a logical operator from a premise of the rule. Some of the rules allow an assumption of a certain type to be *discharged*. To indicate which assumption is discharged by which inference, we also assign labels to both the assumption and the inference. This is indicated by writing the assumption as "$[\varphi]^n$".

It is customary to consider rules for all logical operators, even for those (if any) that we consider as defined.

7.2 Propositional Rules

Rules for &

$$\frac{\varphi \qquad \psi}{\varphi \,\&\, \psi}\ \&\text{Intro}$$

$$\frac{\varphi \,\&\, \psi}{\varphi}\ \&\text{Elim} \qquad \frac{\varphi \,\&\, \psi}{\psi}\ \&\text{Elim}$$

Rules for $\vee$

$$\frac{\varphi}{\varphi \vee \psi}\ \vee\text{Intro} \qquad \frac{\psi}{\varphi \vee \psi}\ \vee\text{Intro}$$

$$\begin{array}{c} \varphi \vee \psi \qquad \begin{array}{cc} [\varphi]^n & [\psi]^n \\ \vdots & \vdots \\ \chi & \chi \end{array} \\ \hline \chi \end{array}\ n\ \vee\text{Elim}$$

Rules for $\supset$

$$\begin{array}{c} [\varphi]^n \\ \vdots \\ \psi \end{array}\ n\ \frac{}{\varphi \supset \psi}\ \supset\text{Intro}$$

$$\frac{\varphi \supset \psi \qquad \varphi}{\psi}\ \supset\text{Elim}$$

Rules for $\sim$

$$\begin{array}{c} [\varphi]^n \\ \vdots \\ \bot \end{array}\ n\ \frac{}{\sim\varphi}\ \sim\text{Intro}$$

$$\frac{\sim\varphi \qquad \varphi}{\bot}\ \sim\text{Elim}$$

Rules for $\bot$

$$\frac{\bot}{\varphi}\,\bot_I \qquad\qquad \begin{array}{c}[\sim\varphi]^n\\ \vdots\\ \bot\end{array}\;{}^{n}\,\bot_C\;\varphi$$

Note that $\sim$Intro and $\bot_C$ are very similar: The difference is that $\sim$Intro derives a negated sentence $\sim\varphi$ but $\bot_C$ a positive sentence φ.

7.3 Quantifier Rules

Rules for $\forall$

$$\frac{\varphi(a)}{\forall x\,\varphi(x)}\,\forall\text{Intro} \qquad\qquad \frac{\forall x\,\varphi(x)}{\varphi(t)}\,\forall\text{Elim}$$

In the rules for $\forall$, t is a ground term (a term that does not contain any variables), and a is a constant symbol which does not occur in the conclusion $\forall x\,\varphi(x)$, or in any assumption which is undischarged in the derivation ending with the premise $\varphi(a)$. We call a the *eigenvariable* of the $\forall$Intro inference.

Rules for $\exists$

$$\frac{\varphi(t)}{\exists x\,\varphi(x)}\,\exists\text{Intro} \qquad\qquad {}^{n}\,\frac{\exists x\,\varphi(x) \qquad \begin{array}{c}[\varphi(a)]^n\\ \vdots\\ \chi\end{array}}{\chi}\,\exists\text{Elim}$$

Again, t is a ground term, and a is a constant which does not occur in the premise $\exists x\,\varphi(x)$, in the conclusion χ, or any assumption which is undischarged in the derivations ending with the two premises (other than the assumptions $\varphi(a)$). We call a the *eigenvariable* of the $\exists$Elim inference.

The condition that an eigenvariable neither occur in the premises nor in any assumption that is undischarged in the derivations leading to the premises for the $\forall$Intro or $\exists$Elim inference is called the *eigenvariable condition*.

We use the term "eigenvariable" even though a in the above rules is a constant. This has historical reasons.

In $\exists$Intro and $\forall$Elim there are no restrictions, and the term t can be anything, so we do not have to worry about any conditions. On the other hand, in the $\exists$Elim and $\forall$Intro rules, the eigenvariable condition requires that the constant symbol a does not occur anywhere in the conclusion or in an undischarged assumption. The condition is necessary to ensure that the system is sound, i.e., only derives sentences from undischarged assumptions from which the follow. Without this condition, the following would be allowed:

$$\dfrac{\exists x\, \varphi(x) \qquad \dfrac{[\varphi(a)]^1}{\forall x\, \varphi(x)}\,{*\forall\text{Intro}}}{\forall x\, \varphi(x)}\,\exists\text{Elim}$$

However, $\exists x\, \varphi(x) \nvDash \forall x\, \varphi(x)$.

7.4 Derivations

We've said what an assumption is, and we've given the rules of inference. Derivations in the sequent calculus are inductively generated from these: each derivation either is an assumption on its own, or consists of one, two, or three derivations followed by a correct inference.

Definition 7.2 (Derivation)**.** A *derivation* of a sentence φ from assumptions Γ is a tree of sentences satisfying the following conditions:

1. The topmost sentences of the tree are either in Γ or are discharged by an inference in the tree.
2. The bottommost sentence of the tree is φ.
3. Every sentence in the tree except φ is a premise of a correct application of am inference rule whose conclusion stands directly below that sentence in the tree.

We then say that φ is the *conclusion* of the derivation and that φ is *derivable* from Γ.

Example 7.3. Every assumption on its own is a derivation. So, e.g., χ by itself is a derivation, and so is θ by itself. We can obtain a new derivation from these by applying, say, the &Intro rule,

$$\dfrac{\varphi \qquad \psi}{\varphi \,\&\, \psi}\,\&\text{Intro}$$

These rules are meant to be general: we can replace the φ and ψ in it with any sentences, e.g., by χ and θ. Then the conclusion would be $\chi \,\&\, \theta$, and so

$$\dfrac{\chi \qquad \theta}{\chi \,\&\, \theta}\ \&\text{Intro}$$

is a correct derivation. Of course, we can also switch the assumptions, so that θ plays the role of φ and χ that of ψ. Thus,

$$\dfrac{\theta \qquad \chi}{\theta \,\&\, \chi}\ \&\text{Intro}$$

is also a correct derivation.

We can now apply another rule, say, $\supset$Intro, which allows us to conclude a conditional and allows us to discharge any assumption that is identical to the conclusion of that conditional. So both of the following would be correct derivations:

$$1\ \dfrac{\dfrac{[\chi]^1 \qquad \theta}{\chi \,\&\, \theta}\ \&\text{Intro}}{\chi \supset (\chi \,\&\, \theta)}\ \supset\text{Intro} \qquad 1\ \dfrac{\dfrac{\chi \qquad [\theta]^1}{\chi \,\&\, \theta}\ \&\text{Intro}}{\theta \supset (\chi \,\&\, \theta)}\ \supset\text{Intro}$$

7.5 Examples of Derivations

Example 7.4. Let's give a derivation of the sentence $(\varphi \,\&\, \psi) \supset \varphi$.

We begin by writing the desired conclusion at the bottom of the derivation.

$$\dfrac{}{(\varphi \,\&\, \psi) \supset \varphi}$$

Next, we need to figure out what kind of inference could result in a sentence of this form. The main operator of the conclusion is $\supset$, so we'll try to arrive at the conclusion using the $\supset$Intro rule. It is best to write down the assumptions involved and label the inference rules as you progress, so it is easy to see whether all assumptions have been discharged at the end of the proof.

$$1\ \dfrac{\begin{array}{c}[\varphi \,\&\, \psi]^1 \\ \vdots \\ \varphi\end{array}}{(\varphi \,\&\, \psi) \supset \varphi}\ \supset\text{Intro}$$

We now need to fill in the steps from the assumption $\varphi \,\&\, \psi$ to φ. Since we only have one connective to deal with, &, we must use the & elim rule. This gives us the following proof:

$$1\ \dfrac{\dfrac{[\varphi \,\&\, \psi]^1}{\varphi}\ \&\text{Elim}}{(\varphi \,\&\, \psi) \supset \varphi}\ \supset\text{Intro}$$

We now have a correct derivation of $(\varphi \,\&\, \psi) \supset \varphi$.

Example 7.5. Now let's give a derivation of $(\sim\varphi \vee \psi) \supset (\varphi \supset \psi)$.

We begin by writing the desired conclusion at the bottom of the derivation.

$$\frac{}{(\sim\varphi \vee \psi) \supset (\varphi \supset \psi)}$$

To find a logical rule that could give us this conclusion, we look at the logical connectives in the conclusion: $\sim$, $\vee$, and $\supset$. We only care at the moment about the first occurence of $\supset$ because it is the main operator of the sentence in the end-sequent, while $\sim$, $\vee$ and the second occurence of $\supset$ are inside the scope of another connective, so we will take care of those later. We therefore start with the $\supset$Intro rule. A correct application must look as follows:

$$\dfrac{\begin{array}{c}[\sim\varphi \vee \psi]^1 \\ \vdots \\ \varphi \supset \psi\end{array}}{(\sim\varphi \vee \psi) \supset (\varphi \supset \psi)}\,{\scriptstyle 1}\ \supset\text{Intro}$$

This leaves us with two possibilities to continue. Either we can keep working from the bottom up and look for another application of the $\supset$Intro rule, or we can work from the top down and apply a $\vee$Elim rule. Let us apply the latter. We will use the assumption $\sim\varphi \vee \psi$ as the leftmost premise of $\vee$Elim. For a valid application of $\vee$Elim, the other two premises must be identical to the conclusion $\varphi \supset \psi$, but each may be derived in turn from another assumption, namely the two disjuncts of $\sim\varphi \vee \psi$. So our derivation will look like this:

$$\dfrac{\dfrac{\begin{array}{ccc}[\sim\varphi \vee \psi]^1 & \begin{array}{c}[\sim\varphi]^2 \\ \vdots \\ \varphi \supset \psi\end{array} & \begin{array}{c}[\psi]^2 \\ \vdots \\ \varphi \supset \psi\end{array}\end{array}}{\varphi \supset \psi}\,{\scriptstyle 2}\ \vee\text{Elim}}{(\sim\varphi \vee \psi) \supset (\varphi \supset \psi)}\,{\scriptstyle 1}\ \supset\text{Intro}$$

In each of the two branches on the right, we want to derive $\varphi \supset \psi$, which is best done using $\supset$Intro.

$$\dfrac{\dfrac{\begin{array}{ccc}[\sim\varphi \vee \psi]^1 & \dfrac{\begin{array}{c}[\sim\varphi]^2, [\varphi]^3 \\ \vdots \\ \psi\end{array}}{\varphi \supset \psi}\,{\scriptstyle 3}\ \supset\text{Intro} & \dfrac{\begin{array}{c}[\psi]^2, [\varphi]^4 \\ \vdots \\ \psi\end{array}}{\varphi \supset \psi}\,{\scriptstyle 4}\ \supset\text{Intro}\end{array}}{\varphi \supset \psi}\,{\scriptstyle 2}\ \vee\text{Elim}}{(\sim\varphi \vee \psi) \supset (\varphi \supset \psi)}\,{\scriptstyle 1}\ \supset\text{Intro}$$

For the two missing parts of the derivation, we need derivations of ψ from $\sim\varphi$ and φ in the middle, and from φ and ψ on the left. Let's take the former first. $\sim\varphi$ and φ are the two premises of $\sim$Elim:

$$\dfrac{[\sim\varphi]^2 \qquad [\varphi]^3}{\begin{array}{c}\bot\\ \vdots\\ \psi\end{array}}\ \sim\text{Elim}$$

By using $\bot_I$, we can obtain ψ as a conclusion and complete the branch.

$$\dfrac{\dfrac{\dfrac{[\sim\varphi]^2 \qquad [\varphi]^3}{\bot}\ \bot\text{Intro}}{\psi}\ \bot_I}{{}^3\dfrac{}{\varphi\supset\psi}\ \supset\text{Intro}} \qquad \begin{array}{c}[\psi]^2, [\varphi]^4\\ \vdots\\ \psi\end{array}\ {}^4\dfrac{}{\varphi\supset\psi}\ \supset\text{Intro}}{{}^2\dfrac{[\sim\varphi\vee\psi]^1 \qquad \varphi\supset\psi}{{}^1\dfrac{}{(\sim\varphi\vee\psi)\supset(\varphi\supset\psi)}\ \supset\text{Intro}}\ \vee\text{Elim}}$$

Let's now look at the rightmost branch. Here it's important to realize that the definition of derivation *allows assumptions to be discharged* but *does not require* them to be. In other words, if we can derive ψ from one of the assumptions φ and ψ without using the other, that's ok. And to derive ψ from ψ is trivial: ψ by itself is such a derivation, and no inferences are needed. So we can simply delete the assumtion φ.

$$\dfrac{\dfrac{\dfrac{[\sim\varphi]^2 \qquad [\varphi]^3}{\bot}\ \sim\text{Elim}}{\psi}\ \bot_I}{{}^3\dfrac{}{\varphi\supset\psi}\ \supset\text{Intro}} \qquad \dfrac{[\psi]^2}{\varphi\supset\psi}\ \supset\text{Intro}}{{}^2\dfrac{[\sim\varphi\vee\psi]^1 \qquad \varphi\supset\psi}{{}^1\dfrac{}{(\sim\varphi\vee\psi)\supset(\varphi\supset\psi)}\ \supset\text{Intro}}\ \vee\text{Elim}}$$

Note that in the finished derivation, the rightmost $\supset$Intro inference does not actually discharge any assumptions.

Example 7.6. So far we have not needed the $\bot_C$ rule. It is special in that it allows us to discharge an assumption that isn't a sub-formula of the conclusion of the rule. It is closely related to the $\bot_I$ rule. In fact, the $\bot_I$ rule is a special case of the $\bot_C$ rule—there is a logic called "intuitionistic logic" in which only $\bot_I$ is allowed. The $\bot_C$ rule is a last resort when nothing else works. For instance, suppose we want to derive $\varphi\vee\sim\varphi$. Our usual strategy would be to attempt to derive $\varphi\vee\sim\varphi$ using $\vee$Intro. But this would require us to derive either φ or $\sim\varphi$ from no assumptions, and this can't be done. $\bot_C$ to the rescue!

$$\dfrac{\begin{array}{c}[\sim(\varphi \vee \sim\varphi)]^1 \\ \vdots \\ \bot\end{array}}{\varphi \vee \sim\varphi}\,{}^{1}\,\bot_C$$

Now we're looking for a derivation of $\bot$ from $\sim(\varphi \vee \sim\varphi)$. Since $\bot$ is the conclusion of $\sim$Elim we might try that:

$$\dfrac{\dfrac{\begin{array}{c}[\sim(\varphi \vee \sim\varphi)]^1 \\ \vdots \\ \sim\varphi\end{array} \qquad \begin{array}{c}[\sim(\varphi \vee \sim\varphi)]^1 \\ \vdots \\ \varphi\end{array}}{\bot}\,\sim\text{Elim}}{\varphi \vee \sim\varphi}\,{}^{1}\,\bot_C$$

Our strategy for finding a derivation of $\sim\varphi$ calls for an application of $\sim$Intro:

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}[\sim(\varphi \vee \sim\varphi)]^1, [\varphi]^2 \\ \vdots \\ \bot\end{array}}{\sim\varphi}\,\sim\text{Intro} \qquad \begin{array}{c}[\sim(\varphi \vee \sim\varphi)]^1 \\ \vdots \\ \varphi\end{array}}{\bot}\,\sim\text{Elim}}{\varphi \vee \sim\varphi}\,{}^{1}\,\bot_C$$

Here, we can get $\bot$ easily by applying $\sim$Elim to the assumption $\sim(\varphi \vee \sim\varphi)$ and $\varphi \vee \sim\varphi$ which follows from our new assumption φ by $\vee$Intro:

$$\dfrac{\dfrac{\dfrac{[\sim(\varphi \vee \sim\varphi)]^1 \qquad \dfrac{[\varphi]^2}{\varphi \vee \sim\varphi}\,\vee\text{Intro}}{\bot}\,\sim\text{Elim}}{\sim\varphi}\,{}^{2}\,\sim\text{Intro} \qquad \begin{array}{c}[\sim(\varphi \vee \sim\varphi)]^1 \\ \vdots \\ \varphi\end{array}}{\bot}\,\sim\text{Elim}}{\varphi \vee \sim\varphi}\,{}^{1}\,\bot_C$$

On the right side we use the same strategy, except we get φ by $\bot_C$:

$$\dfrac{\dfrac{\dfrac{[\sim(\varphi \vee \sim\varphi)]^1 \qquad \dfrac{[\varphi]^2}{\varphi \vee \sim\varphi}\,\vee\text{Intro}}{\bot}\,\sim\text{Elim}}{\sim\varphi}\,{}^{2}\,\sim\text{Intro} \qquad \dfrac{\dfrac{[\sim(\varphi \vee \sim\varphi)]^1 \qquad \dfrac{[\sim\varphi]^3}{\varphi \vee \sim\varphi}\,\vee\text{Intro}}{\bot}\,\sim\text{Elim}}{\varphi}\,{}^{3}\,\bot_C}{\bot}\,\sim\text{Elim}}{\varphi \vee \sim\varphi}\,{}^{1}\,\bot_C$$

7.6 Derivations with Quantifiers

Example 7.7. When dealing with quantifiers, we have to make sure not to violate the eigenvariable condition, and sometimes this requires us to play around with the order of carrying out certain inferences. In general, it helps to try and take care of rules subject to the eigenvariable condition first (they will be lower down in the finished proof).

Let's see how we'd give a derivation of the formula $\exists x \sim\varphi(x) \supset \sim\forall x\, \varphi(x)$. Starting as usual, we write

$$\frac{}{\exists x \sim\varphi(x) \supset \sim\forall x\, \varphi(x)}$$

We start by writing down what it would take to justify that last step using the $\supset$Intro rule.

$$\dfrac{\begin{array}{c}[\exists x \sim\varphi(x)]^1 \\ \vdots \\ \sim\forall x\, \varphi(x)\end{array}}{\exists x \sim\varphi(x) \supset \sim\forall x\, \varphi(x)} \supset\text{Intro}$$

Since there is no obvious rule to apply to $\sim\forall x\, \varphi(x)$, we will proceed by setting up the derivation so we can use the $\exists$Elim rule. Here we must pay attention to the eigenvariable condition, and choose a constant that does not appear in $\exists x\, \varphi(x)$ or any assumptions that it depends on. (Since no constant symbols appear, however, any choice will do fine.)

$$\dfrac{\dfrac{\begin{array}{cc}[\exists x \sim\varphi(x)]^1 & \begin{array}{c}[\sim\varphi(a)]^2 \\ \vdots \\ \sim\forall x\, \varphi(x)\end{array}\end{array}}{\sim\forall x\, \varphi(x)}\ {}^2\exists\text{Elim}}{\exists x \sim\varphi(x) \supset \sim\forall x\, \varphi(x)} \supset\text{Intro}$$

In order to derive $\sim\forall x\, \varphi(x)$, we will attempt to use the $\sim$Intro rule: this requires that we derive a contradiction, possibly using $\forall x\, \varphi(x)$ as an additional assumption. Of coursem, this contradiction may involve the assumption $\sim\varphi(a)$ which will be discharged by the $\supset$Intro inference. We can set it up as follows:

$$\dfrac{\dfrac{\begin{array}{cc}[\exists x \sim\varphi(x)]^1 & \dfrac{\begin{array}{c}[\sim\varphi(a)]^2, [\forall x\, \varphi(x)]^3 \\ \vdots \\ \bot\end{array}}{\sim\forall x\, \varphi(x)}\ {}^3\sim\text{Intro}\end{array}}{\sim\forall x\, \varphi(x)}\ {}^2\exists\text{Elim}}{\exists x \sim\varphi(x) \supset \sim\forall x\, \varphi(x)} \supset\text{Intro}$$

It looks like we are close to getting a contradiction. The easiest rule to apply is the ∀Elim, which has no eigenvariable conditions. Since we can use any term we want to replace the universally quantified x, it makes the most sense to continue using a so we can reach a contradiction.

$$\dfrac{\dfrac{\dfrac{[\exists x \sim\varphi(x)]^1 \qquad \dfrac{\dfrac{\dfrac{[\forall x\, \varphi(x)]^3}{\varphi(a)}\forall\text{Elim} \qquad [\sim\varphi(a)]^2}{\bot}\sim\text{Elim}}{\sim\forall x\, \varphi(x)}\sim\text{Intro}^3}{\sim\forall x\, \varphi(x)}\exists\text{Elim}^2}{\exists x \sim\varphi(x) \supset \sim\forall x\, \varphi(x)}\supset\text{Intro}$$

It is important, especially when dealing with quantifiers, to double check at this point that the eigenvariable condition has not been violated. Since the only rule we applied that is subject to the eigenvariable condition was ∃Elim, and the eigenvariable a does not occur in any assumptions it depends on, this is a correct derivation.

Example 7.8. Sometimes we may derive a formula from other formulae. In these cases, we may have undischarged assumptions. It is important to keep track of our assumptions as well as the end goal.

Let's see how we'd give a derivation of the formula $\exists x\, \chi(x, b)$ from the assumptions $\exists x\, (\varphi(x) \,\&\, \psi(x))$ and $\forall x\, (\psi(x) \supset \chi(x, b))$. Starting as usual, we write the conclusion at the bottom.

$$\dfrac{}{\exists x\, \chi(x, b)}$$

We have two premises to work with. To use the first, i.e., try to find a derivation of $\exists x\, \chi(x, b)$ from $\exists x\, (\varphi(x) \,\&\, \psi(x))$ we would use the ∃Elim rule. Since it has an eigenvariable condition, we will apply that rule first. We get the following:

$$\dfrac{\exists x\, (\varphi(x) \,\&\, \psi(x)) \qquad \begin{array}{c}[\varphi(a) \,\&\, \psi(a)]^1 \\ \vdots \\ \exists x\, \chi(x, b)\end{array}}{\exists x\, \chi(x, b)}\exists\text{Elim}^1$$

The two assumptions we are working with share ψ. It may be useful at this

point to apply &Elim to separate out $\psi(a)$.

$$\dfrac{\exists x\,(\varphi(x)\,\&\,\psi(x)) \qquad \dfrac{\dfrac{[\varphi(a)\,\&\,\psi(a)]^1}{\psi(a)}\,\&\text{Elim}}{\vdots}}{\exists x\,\chi(x,b)}\,{}^1\,\exists\text{Elim}$$

The second assumption we have to work with is $\forall x\,(\psi(x) \supset \chi(x,b))$. Since there is no eigenvariable condition we can instantiate x with the constant symbol a using $\forall$Elim to get $\psi(a) \supset \chi(a,b)$. We now have both $\psi(a) \supset \chi(a,b)$ and $\psi(a)$. Our next move should be a straightforward application of the $\supset$Elim rule.

$$\dfrac{\exists x\,(\varphi(x)\,\&\,\psi(x)) \qquad \dfrac{\dfrac{\dfrac{\forall x\,(\psi(x) \supset \chi(x,b))}{\psi(a) \supset \chi(a,b)}\,\forall\text{Elim} \qquad \dfrac{[\varphi(a)\,\&\,\psi(a)]^1}{\psi(a)}\,\&\text{Elim}}{\chi(a,b)}\,\supset\text{Elim}}{\vdots}}{\exists x\,\chi(x,b)}\,{}^1\,\exists\text{Elim}$$

We are so close! One application of $\exists$Intro and we have reached our goal.

$$\dfrac{\exists x\,(\varphi(x)\,\&\,\psi(x)) \qquad \dfrac{\dfrac{\dfrac{\forall x\,(\psi(x) \supset \chi(x,b))}{\psi(a) \supset \chi(a,b)}\,\forall\text{Elim} \qquad \dfrac{[\varphi(a)\,\&\,\psi(a)]^1}{\psi(a)}\,\&\text{Elim}}{\chi(a,b)}\,\supset\text{Elim}}{\exists x\,\chi(x,b)}\,\exists\text{Intro}}{\exists x\,\chi(x,b)}\,{}^1\,\exists\text{Elim}$$

Since we ensured at each step that the eigenvariable conditions were not violated, we can be confident that this is a correct derivation.

Example 7.9. Give a derivation of the formula $\sim\forall x\,\varphi(x)$ from the assumptions $\forall x\,\varphi(x) \supset \exists y\,\psi(y)$ and $\sim\exists y\,\psi(y)$. Starting as usual, we write the target formula at the bottom.

$$\dfrac{}{\sim\forall x\,\varphi(x)}$$

The last line of the derivation is a negation, so let's try using $\sim$Intro. This will

require that we figure out how to derive a contradiction.

$$\dfrac{\dfrac{\begin{array}{c}[\forall x\, \varphi(x)]^1\\ \vdots\\ \bot\end{array}}{\sim\forall x\, \varphi(x)}\,{\scriptstyle 1}\sim\text{Intro}}$$

So far so good. We can use ∀Elim but it's not obvious if that will help us get to our goal. Instead, let's use one of our assumptions. $\forall x\, \varphi(x) \supset \exists y\, \psi(y)$ together with $\forall x\, \varphi(x)$ will allow us to use the ⊃Elim rule.

$$\dfrac{\dfrac{\dfrac{\forall x\, \varphi(x) \supset \exists y\, \psi(y) \qquad [\forall x\, \varphi(x)]^1}{\exists y\, \psi(y)}\supset\text{Elim}\\ \vdots\\ \bot}{\sim\forall x\, \varphi(x)}\,{\scriptstyle 1}\sim\text{Intro}}$$

We now have one final assumption to work with, and it looks like this will help us reach a contradiction by using ∼Elim.

$$\dfrac{\dfrac{\sim\exists y\, \psi(y) \qquad \dfrac{\forall x\, \varphi(x) \supset \exists y\, \psi(y) \qquad [\forall x\, \varphi(x)]^1}{\exists y\, \psi(y)}\supset\text{Elim}}{\bot}\sim\text{Elim}}{\sim\forall x\, \varphi(x)}\,{\scriptstyle 1}\sim\text{Intro}$$

7.7 Proof-Theoretic Notions

Just as we've defined a number of important semantic notions (validity, entailment, satisfiabilty), we now define corresponding *proof-theoretic notions*. These are not defined by appeal to satisfaction of sentences in structures, but by appeal to the derivability or non-derivability of certain sentences from others. It was an important discovery, due to Gödel, that these notions coincide. That they do is the content of the *completeness theorem*.

Definition 7.10 (Theorems)**.** A sentence φ is a *theorem* if there is a derivation of φ in natural deduction in which all assumptions are discharged. We write $\vdash \varphi$ if φ is a theorem and $\nvdash \varphi$ if it is not.

Definition 7.11 (Derivability)**.** A sentence φ is *derivable from* a set of sentences Γ, $\Gamma \vdash \varphi$, if there is a derivation with conclusion φ and in which every assumption is either discharged or is in Γ. If φ is not derivable from Γ we write $\Gamma \nvdash \varphi$.

Definition 7.12 (Consistency)**.** A set of sentences Γ is *inconsistent* iff $\Gamma \vdash \bot$. If Γ is not inconsistent, i.e., if $\Gamma \nvdash \bot$, we say it is *consistent*.

Proposition 7.13 (Reflexivity)**.** *If $\varphi \in \Gamma$, then $\Gamma \vdash \varphi$.*

Proof. The assumption φ by itself is a derivation of φ where every undischarged assumption (i.e., φ) is in Γ. □

Proposition 7.14 (Monotony)**.** *If $\Gamma \subseteq \Delta$ and $\Gamma \vdash \varphi$, then $\Delta \vdash \varphi$.*

Proof. Any derivation of φ from Γ is also a derivation of φ from Δ. □

Proposition 7.15 (Transitivity)**.** *If $\Gamma \vdash \varphi$ for every $\varphi \in \Delta$ and $\Delta \vdash \psi$, then $\Gamma \vdash \psi$.*

Proof. If $\Delta \vdash \psi$, then there is a derivation δ_0 of ψ with all undischarged assumptions in Δ. We show that $\Gamma \vdash \psi$ by induction on the number n of undischarged assumptions in δ_0.

If $n = 0$, then δ_0 has no undischarged assumptions, and so also counts as a derivation of ψ from Γ.

Otherwise, pick an undischarged assumption φ in δ_0 and let Δ_1 be the remaining undischarged assumptions. We obtain the derivation δ_1:

$$\dfrac{\begin{array}{c}\Delta_1, [\varphi]^1 \\ \vdots\,\delta_0 \\ \psi\end{array}}{\varphi \supset \psi}\ \supset\text{Intro}$$

Since the number of undischarged assumptions in δ_1 is $n - 1$, the inductive hypothesis applies: there is a derivation δ_2 of $\varphi \supset \psi$ from Γ. Since $\Gamma \vdash \varphi$ there is also a derivation δ_3 of φ from Γ. Now consider:

$$\dfrac{\begin{array}{cc}\Gamma & \Gamma \\ \vdots\,\delta_2 & \vdots\,\delta_3 \\ \varphi \supset \psi & \varphi\end{array}}{\psi}\ \supset\text{Elim}$$

This shows $\Gamma \vdash \psi$. □

Proposition 7.16. *Γ is inconsistent iff $\Gamma \vdash \varphi$ for every sentence φ.*

Proof. Exercise. □

Proposition 7.17 (Compactness)**.**

1. *If $\Gamma \vdash \varphi$ then there is a finite subset $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \vdash \varphi$.*
2. *If every finite subset of Γ is consistent, then Γ is consistent.*

Proof. 1. If $\Gamma \vdash \varphi$, then there is a derivation δ of φ from Γ. Let Γ_0 be the set of undischarged assumptions of δ. Since any derivation is finite, Γ_0 can only contain finitely many sentences. So, δ is a derivation of φ from a finite $\Gamma_0 \subseteq \Gamma$.

2. This is the contrapositive of (1) for the special case $\varphi \equiv \bot$.

□

7.8 Derivability and Consistency

We will now establish a number of properties of the derivability relation. They are independently interesting, but each will play a role in the proof of the completeness theorem.

Proposition 7.18. *If $\Gamma \vdash \varphi$ and $\Gamma \cup \{\varphi\}$ is inconsistent, then Γ is inconsistent.*

Proof. Let the derivation of φ from Γ be δ_1 and the derivation of $\bot$ from $\Gamma \cup \{\varphi\}$ be δ_2. We can then derive:

$$\dfrac{\dfrac{\Gamma, [\varphi]^1}{\vdots\ \delta_2}\ \bot}{\sim\varphi}\ {}^1\sim\text{Intro} \qquad \begin{matrix}\Gamma\\ \vdots\ \delta_1\\ \varphi\end{matrix}}{\bot}\ \sim\text{Elim}$$

In the new derivation, the assumption φ is discharged, so it is a derivation from Γ. □

Proposition 7.19. *$\Gamma \vdash \varphi$ iff $\Gamma \cup \{\sim\varphi\}$ is inconsistent.*

Proof. First suppose $\Gamma \vdash \varphi$, i.e., there is a derivation δ_0 of φ from undischarged assumptions Γ. We obtain a derivation of $\bot$ from $\Gamma \cup \{\sim\varphi\}$ as follows:

$$\dfrac{\sim\varphi \qquad \begin{matrix}\Gamma\\ \vdots\ \delta_0\\ \varphi\end{matrix}}{\bot}\ \sim\text{Elim}$$

Now assume $\Gamma \cup \{\sim\varphi\}$ is inconsistent, and let δ_1 be the corresponding derivation of $\bot$ from undischarged assumptions in $\Gamma \cup \{\sim\varphi\}$. We obtain a derivation of φ from Γ alone by using $\bot_C$:

$$\dfrac{\begin{array}{c}\Gamma, [\sim\varphi]^1 \\ \vdots\,\delta_1 \\ \bot\end{array}}{\varphi}\,\bot_C$$

□

Proposition 7.20. *If $\Gamma \vdash \varphi$ and $\sim\varphi \in \Gamma$, then Γ is inconsistent.*

Proof. Suppose $\Gamma \vdash \varphi$ and $\sim\varphi \in \Gamma$. Then there is a derivation δ of φ from Γ. Consider this simple application of the $\sim$Elim rule:

$$\dfrac{\sim\varphi \qquad \begin{array}{c}\Gamma \\ \vdots\,\delta \\ \varphi\end{array}}{\bot}\,\sim\text{Elim}$$

Since $\sim\varphi \in \Gamma$, all undischarged assumptions are in Γ, this shows that $\Gamma \vdash \bot$. □

Proposition 7.21. *If $\Gamma \cup \{\varphi\}$ and $\Gamma \cup \{\sim\varphi\}$ are both inconsistent, then Γ is inconsistent.*

Proof. There are derivations δ_1 and δ_2 of $\bot$ from $\Gamma \cup \{\varphi\}$ and $\bot$ from $\Gamma \cup \{\sim\varphi\}$, respectively. We can then derive

$$\dfrac{\dfrac{\begin{array}{c}\Gamma, [\sim\varphi]^2 \\ \vdots\,\delta_2 \\ \bot\end{array}}{\sim\sim\varphi}\,{}^2\sim\text{Intro} \qquad \dfrac{\begin{array}{c}\Gamma, [\varphi]^1 \\ \vdots\,\delta_1 \\ \bot\end{array}}{\sim\varphi}\,{}^1\sim\text{Intro}}{\bot}\,\sim\text{Elim}$$

Since the assumptions φ and $\sim\varphi$ are discharged, this is a derivation of $\bot$ from Γ alone. Hence Γ is inconsistent. □

7.9 Derivability and the Propositional Connectives

Proposition 7.22. *1. Both $\varphi \,\&\, \psi \vdash \varphi$ and $\varphi \,\&\, \psi \vdash \psi$*

2. $\varphi, \psi \vdash \varphi \,\&\, \psi$.

Proof. 1. We can derive both

$$\dfrac{\varphi \,\&\, \psi}{\varphi}\,\&\text{Elim} \qquad \dfrac{\varphi \,\&\, \psi}{\psi}\,\&\text{Elim}$$

2. We can derive:

$$\dfrac{\varphi \qquad \psi}{\varphi \,\&\, \psi}\ \&\text{Intro}$$

□

Proposition 7.23. *1. $\varphi \vee \psi, \sim\varphi, \sim\psi$ is inconsistent.*

2. Both $\varphi \vdash \varphi \vee \psi$ and $\psi \vdash \varphi \vee \psi$.

Proof. 1. Consider the following derivation:

$$\dfrac{\varphi \vee \psi \qquad \dfrac{\sim\varphi \qquad [\varphi]^1}{\bot}\sim\text{Elim} \qquad \dfrac{\sim\psi \qquad [\psi]^1}{\bot}\sim\text{Elim}}{\bot}\ {}^1\vee\text{Elim}$$

This is a derivation of $\bot$ from undischarged assumptions $\varphi \vee \psi$, $\sim\varphi$, and $\sim\psi$.

2. We can derive both

$$\dfrac{\varphi}{\varphi \vee \psi}\ \vee\text{Intro} \qquad \dfrac{\psi}{\varphi \vee \psi}\ \vee\text{Intro}$$

□

Proposition 7.24. *1. $\varphi, \varphi \supset \psi \vdash \psi$.*

2. Both $\sim\varphi \vdash \varphi \supset \psi$ and $\psi \vdash \varphi \supset \psi$.

Proof. 1. We can derive:

$$\dfrac{\varphi \supset \psi \qquad \psi}{\psi}\ \supset\text{Elim}$$

2. This is shown by the following two derivations:

$$\dfrac{\dfrac{\dfrac{\sim\varphi \qquad [\varphi]^1}{\bot}\sim\text{Elim}}{\psi}\ \bot_I}{\varphi \supset \psi}\ {}^1\supset\text{Intro} \qquad \dfrac{\psi}{\varphi \supset \psi}\ \supset\text{Intro}$$

Note that $\supset$Intro may, but does not have to, discharge the assumption φ.

□

7.10 Derivability and the Quantifiers

Theorem 7.25. *If c is a constant not occurring in Γ or $\varphi(x)$ and $\Gamma \vdash \varphi(c)$, then $\Gamma \vdash \forall x\, \varphi(x)$.*

Proof. Let δ be a derivation of $\varphi(c)$ from Γ. By adding a $\forall$Intro inference, we obtain a proof of $\forall x\, \varphi(x)$. Since c does not occur in Γ or $\varphi(x)$, the eigenvariable condition is satisfied. □

Proposition 7.26. *1. $\varphi(t) \vdash \exists x\, \varphi(x)$.*

2. $\forall x\, \varphi(x) \vdash \varphi(t)$.

Proof. 1. The following is a derivation of $\exists x\, \varphi(x)$ from $\varphi(t)$:

$$\dfrac{\varphi(t)}{\exists x\, \varphi(x)}\ \exists\text{Intro}$$

2. The following is a derivation of $\varphi(t)$ from $\forall x\, \varphi(x)$:

$$\dfrac{\forall x\, \varphi(x)}{\varphi(t)}\ \forall\text{Elim}$$

□

7.11 Soundness

A derivation system, such as natural deduction, is *sound* if it cannot derive things that do not actually follow. Soundness is thus a kind of guaranteed safety property for derivation systems. Depending on which proof theoretic property is in question, we would like to know for instance, that

1. every derivable sentence is valid;
2. if a sentence is derivable from some others, it is also a consequence of them;
3. if a set of sentences is inconsistent, it is unsatisfiable.

These are important properties of a derivation system. If any of them do not hold, the derivation system is deficient—it would derive too much. Consequently, establishing the soundness of a derivation system is of the utmost importance.

Theorem 7.27 (Soundness)**.** *If φ is derivable from the undischarged assumptions Γ, then $\Gamma \vDash \varphi$.*

Proof. Let δ be a derivation of φ. We proceed by induction on the number of inferences in δ.

For the induction basis we show the claim if the number of inferences is 0. In this case, δ consists only of an initial formula. Every initial formula φ is an undischarged assumption, and as such, any structure $\mathfrak{M}$ that satisfies all of the undischarged assumptions of the proof also satisfies φ.

Now for the inductive step. Suppose that δ contains n inferences. The premise(s) of the lowermost inference are derived using sub-derivations, each of which contains fewer than n inferences. We assume the induction hypothesis: The premises of the last inference follow from the undischarged assumptions of the sub-derivations ending in those premises. We have to show that φ follows from the undischarged assumptions of the entire proof.

We distinguish cases according to the type of the lowermost inference. First, we consider the possible inferences with only one premise.

1. Suppose that the last inference is $\sim$Intro: The derivation has the form

$$\dfrac{\begin{array}{c}\Gamma, [\varphi]^n \\ \vdots\,\delta_1\,\vdots \\ \bot\end{array}}{\sim\varphi}\,{}^{n}\,\sim\text{Intro}$$

   By inductive hypothesis, $\bot$ follows from the undischarged assumptions $\Gamma \cup \{\varphi\}$ of δ_1. Consider a structure $\mathfrak{M}$. We need to show that, if $\mathfrak{M} \models \Gamma$, then $\mathfrak{M} \models \sim\varphi$. Suppose for reductio that $\mathfrak{M} \models \Gamma$, but $\mathfrak{M} \not\models \sim\varphi$, i.e., $\mathfrak{M} \models \varphi$. This would mean that $\mathfrak{M} \models \Gamma \cup \{\varphi\}$. This is contrary to our inductive hypothesis. So, $\mathfrak{M} \models \sim\varphi$.

2. The last inference is &Elim: There are two variants: φ or ψ may be inferred from the premise $\varphi \,\&\, \psi$. Consider the first case. The derivation δ looks like this:

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots\,\delta_1\,\vdots \\ \varphi \,\&\, \psi\end{array}}{\varphi}\,\&\text{Elim}$$

   By inductive hypothesis, $\varphi \,\&\, \psi$ follows from the undischarged assumptions Γ of δ_1. Consider a structure $\mathfrak{M}$. We need to show that, if $\mathfrak{M} \models \Gamma$, then $\mathfrak{M} \models \varphi$. Suppose $\mathfrak{M} \models \Gamma$. By our inductive hypothesis ($\Gamma \vDash \varphi \vee \psi$), we know that $\mathfrak{M} \models \varphi \,\&\, \psi$. By definition, $\mathfrak{M} \models \varphi \,\&\, \psi$ iff $\mathfrak{M} \models \varphi$ and $\mathfrak{M} \models \psi$. (The case where ψ is inferred from $\varphi \,\&\, \psi$ is handled similarly.)

3. The last inference is $\vee$Intro: There are two variants: $\varphi \vee \psi$ may be inferred from the premise φ or the premise ψ. Consider the first case. The derivation has the form

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots \\ \delta_1 \\ \vdots \\ \varphi\end{array}}{\varphi \vee \psi}\ \vee\text{Intro}$$

   By inductive hypothesis, φ follows from the undischarged assumptions Γ of δ_1. Consider a structure $\mathfrak{M}$. We need to show that, if $\mathfrak{M} \models \Gamma$, then $\mathfrak{M} \models \varphi \vee \psi$. Suppose $\mathfrak{M} \models \Gamma$; then $\mathfrak{M} \models \varphi$ since $\Gamma \vDash \varphi$ (the inductive hypothesis). So it must also be the case that $\mathfrak{M} \models \varphi \vee \psi$. (The case where $\varphi \vee \psi$ is inferred from ψ is handled similarly.)

4. The last inference is $\supset$Intro: $\varphi \supset \psi$ is inferred from a subproof with assumption φ and conclusion ψ, i.e.,

$$\dfrac{\begin{array}{c}\Gamma, [\varphi]^n \\ \vdots \\ \delta_1 \\ \vdots \\ \psi\end{array}}{\varphi \supset \psi}\ {}^{n}\ \supset\text{Intro}$$

   By inductive hypothesis, ψ follows from the undischarged assumptions of δ_1, i.e., $\Gamma \cup \{\varphi\} \vDash \psi$. Consider a structure $\mathfrak{M}$. The undischarged assumptions of δ are just Γ, since φ is discharged at the last inference. So we need to show that $\Gamma \vDash \varphi \supset \psi$. For reductio, suppose that for some structure $\mathfrak{M}$, $\mathfrak{M} \models \Gamma$ but $\mathfrak{M} \not\models \varphi \supset \psi$. So, $\mathfrak{M} \models \varphi$ and $\mathfrak{M} \not\models \psi$. But by hypothesis, ψ is a consequence of $\Gamma \cup \{\varphi\}$, i.e., $\mathfrak{M} \models \psi$, which is a contradiction. So, $\Gamma \vDash \varphi \supset \psi$.

5. The last inference is $\bot_I$: Here, δ ends in

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots \\ \delta_1 \\ \vdots \\ \bot\end{array}}{\varphi}\ \bot_I$$

   By induction hypothesis, $\Gamma \vDash \bot$. We have to show that $\Gamma \vDash \varphi$. Suppose not; then for some $\mathfrak{M}$ we have $\mathfrak{M} \models \Gamma$ and $\mathfrak{M} \not\models \varphi$. But we always have $\mathfrak{M} \not\models \bot$, so this would mean that $\Gamma \nvDash \bot$, contrary to the induction hypothesis.

6. The last inference is $\bot_C$: Exercise.

7. The last inference is $\forall$Intro: Then δ has the form

$$\dfrac{\begin{array}{c}\Gamma\\ \vdots\\ \delta_1\\ \vdots\\ \varphi(a)\end{array}}{\forall x\, \varphi(x)}\ \forall\text{Intro}$$

   The premise $\varphi(a)$ is a consequence of the undischarged assumptions Γ by induction hypothesis. Consider some structure, $\mathfrak{M}$, such that $\mathfrak{M} \models \Gamma$. We need to show that $\mathfrak{M} \models \forall x\, \varphi(x)$. Since $\forall x\, \varphi(x)$ is a sentence, this means we have to show that for every variable assignment s, $\mathfrak{M}, s \models \varphi(x)$ (Proposition 5.42). Since Γ consists entirely of sentences, $\mathfrak{M}, s \models \psi$ for all $\psi \in \Gamma$ by Definition 5.35. Let $\mathfrak{M}'$ be like $\mathfrak{M}$ except that $a^{\mathfrak{M}'} = s(x)$. Since a does not occur in Γ, $\mathfrak{M}' \models \Gamma$ by Corollary 5.44. Since $\Gamma \vDash A(a)$, $\mathfrak{M}' \models A(a)$. Since $\varphi(a)$ is a sentence, $\mathfrak{M}, s \models \varphi(a)$ by Proposition 5.41. $\mathfrak{M}', s \models \varphi(x)$ iff $\mathfrak{M}' \models \varphi(a)$ by Proposition 5.46 (recall that $\varphi(a)$ is just $\varphi(x)[a/x]$). So, $\mathfrak{M}', s \models \varphi(x)$. Since a does not occur in $\varphi(x)$, by Proposition 5.43, $\mathfrak{M}, s \models \varphi(x)$. But s was an arbitrary variable assignment, so $\mathfrak{M} \models \forall x\, \varphi(x)$.

8. The last inference is $\exists$Intro: Exercise.

9. The last inference is $\forall$Elim: Exercise.

Now let's consider the possible inferences with several premises: $\vee$Elim, &Intro, $\supset$Elim, and $\exists$Elim.

1. The last inference is &Intro. $\varphi \,\&\, \psi$ is inferred from the premises φ and ψ and δ has the form

$$\dfrac{\begin{array}{cc}\Gamma_1 & \Gamma_2\\ \vdots & \vdots\\ \delta_1 & \delta_2\\ \vdots & \vdots\\ \varphi & \psi\end{array}}{\varphi \,\&\, \psi}\ \&\text{Intro}$$

   By induction hypothesis, φ follows from the undischarged assumptions Γ_1 of δ_1 and ψ follows from the undischarged assumptions Γ_2 of δ_2. The undischarged assumptions of δ are $\Gamma_1 \cup \gamma_2$, so we have to show that $\Gamma_1 \cup \Gamma_2 \vDash \varphi \,\&\, \psi$. Consider a structure $\mathfrak{M}$ with $\mathfrak{M} \models \Gamma_1 \cup \Gamma_2$. Since $\mathfrak{M} \models \Gamma_1$, it must be the case that $\mathfrak{M} \models \varphi$ as $\Gamma_1 \vDash \varphi$, and since $\mathfrak{M} \models \Gamma_2$, $\mathfrak{M} \models \psi$ since $\Gamma_2 \vDash \psi$. Together, $\mathfrak{M} \models \varphi \,\&\, \psi$.

2. The last inference is $\vee$Elim: Exercise.

3. The last inference is $\supset$Elim. ψ is inferred from the premises $\varphi \supset \psi$ and φ. The derivation δ looks like this:

$$\dfrac{\begin{array}{cc}\Gamma_1 & \Gamma_2 \\ \vdots & \vdots \\ \delta_1 & \delta_2 \\ \vdots & \vdots \\ \varphi \supset \psi & \varphi\end{array}}{\psi}\ \supset\text{Elim}$$

   By induction hypothesis, $\varphi \supset \psi$ follows from the undischarged assumptions Γ_1 of δ_1 and φ follows from the undischarged assumptions Γ_2 of δ_2. Consider a structure $\mathfrak{M}$. We need to show that, if $\mathfrak{M} \models \Gamma_1 \cup \Gamma_2$, then $\mathfrak{M} \models \psi$. Suppose $\mathfrak{M} \models \Gamma_1 \cup \Gamma_2$. Since $\Gamma_1 \vDash \varphi \supset \psi$, $\mathfrak{M} \models \varphi \supset \psi$. Since $\Gamma_2 \vDash \varphi$, we have $\mathfrak{M} \models \varphi$. This means that $\mathfrak{M} \models \psi$ (For if $\mathfrak{M} \not\models \psi$, since $\mathfrak{M} \models \varphi$, we'd have $\mathfrak{M} \not\models \varphi \supset \psi$, contradicting $\mathfrak{M} \models \varphi \supset \psi$).

4. The last inference is $\sim$Elim: Exercise.

5. The last inference is $\exists$Elim: Exercise.

□

Corollary 7.28. *If $\vdash \varphi$, then φ is valid.*

Corollary 7.29. *If Γ is satisfiable, then it is consistent.*

Proof. We prove the contrapositive. Suppose that Γ is not consistent. Then $\Gamma \vdash \bot$, i.e., there is a derivation of $\bot$ from undischarged assumptions in Γ. By Theorem 7.27, any structure $\mathfrak{M}$ that satisfies Γ must satisfy $\bot$. Since $\mathfrak{M} \not\models \bot$ for every structure $\mathfrak{M}$, no $\mathfrak{M}$ can satisfy Γ, i.e., Γ is not satisfiable. □

7.12 Derivations with Identity predicate

Derivations with identity predicate require additional inference rules.

$$\frac{}{t = t}\ =\text{Intro} \qquad \frac{t_1 = t_2 \qquad \varphi(t_1)}{\varphi(t_2)}\ =\text{Elim} \qquad \frac{t_1 = t_2 \qquad \varphi(t_2)}{\varphi(t_1)}\ =\text{Elim}$$

In the above rules, t, t_1, and t_2 are closed terms. The $=$Intro rule allows us to derive any identity statement of the form $t = t$ outright, from no assumptions.

Example 7.30. If s and t are closed terms, then $\varphi(s), s = t \vdash \varphi(t)$:

$$\dfrac{s = t \qquad \varphi(s)}{\varphi(t)} =\text{Elim}$$

This may be familiar as the "principle of substitutability of identicals," or Leibniz' Law.

Example 7.31. We derive the sentence

$$\forall x\, \forall y\, ((\varphi(x) \mathbin{\&} \varphi(y)) \supset x = y)$$

from the sentence

$$\exists x\, \forall y\, (\varphi(y) \supset y = x)$$

We develop the derivation backwards:

$$\dfrac{\dfrac{\dfrac{\exists x\, \forall y\, (\varphi(y) \supset y = x) \quad [\varphi(a) \mathbin{\&} \varphi(b)]^1}{\vdots}}{\dfrac{\dfrac{\dfrac{a = b}{((\varphi(a) \mathbin{\&} \varphi(b)) \supset a = b)}\,{\scriptstyle 1}\ \supset\text{Intro}}{\forall y\, ((\varphi(a) \mathbin{\&} \varphi(y)) \supset a = y)}\ \forall\text{Intro}}{\forall x\, \forall y\, ((\varphi(x) \mathbin{\&} \varphi(y)) \supset x = y)}\ \forall\text{Intro}}$$

We'll now have to use the main assumption: since it is an existential formula, we use $\exists$Elim to derive the intermediary conclusion $a = b$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\exists x\, \forall y\, (\varphi(y) \supset y = x) \qquad \begin{array}{c}[\forall y\, (\varphi(y) \supset y = c)]^2 \\ [\varphi(a) \mathbin{\&} \varphi(b)]^1 \\ \vdots \\ a = b\end{array}}{a = b}\,{\scriptstyle 2}\ \exists\text{Elim}}{((\varphi(a) \mathbin{\&} \varphi(b)) \supset a = b)}\,{\scriptstyle 1}\ \supset\text{Intro}}{\forall y\, ((\varphi(a) \mathbin{\&} \varphi(y)) \supset a = y)}\ \forall\text{Intro}}{\forall x\, \forall y\, ((\varphi(x) \mathbin{\&} \varphi(y)) \supset x = y)}\ \forall\text{Intro}$$

The sub-derivation on the top right is completed by using its assumptions to show that $a = c$ and $b = c$. This requies two separate derivations. The derivation for $a = c$ is as follows:

$$\dfrac{\dfrac{[\forall y\, (\varphi(y) \supset y = c)]^2}{\varphi(a) \supset a = c}\ \forall\text{Elim} \qquad \dfrac{[\varphi(a) \mathbin{\&} \varphi(b)]^1}{\varphi(a)}\ \&\text{Elim}}{a = c}\ \supset\text{Elim}$$

From $a = c$ and $b = c$ we derive $a = b$ by $=$Elim.

7.13 Soundness with Identity predicate

Proposition 7.32. *Natural deduction with rules for $=$ is sound.*

Proof. Any formula of the form $t = t$ is valid, since for every structure $\mathfrak{M}$, $\mathfrak{M} \models t = t$. (Note that we assume the term t to be ground, i.e., it contains no variables, so variable assignments are irrelevant).

Suppose the last inference in a derivation is $=$Elim, i.e., the derivation has the following form:

$$\dfrac{\begin{array}{cc}\Gamma_1 & \Gamma_2 \\ \vdots & \vdots \\ \delta_1 & \delta_2 \\ \vdots & \vdots \\ t_1 = t_2 & \varphi(t_1)\end{array}}{\varphi(t_2)} =\text{Elim}$$

The premises $t_1 = t_2$ and $\varphi(t_1)$ are derived from undischarged assumptions Γ_1 and Γ_2, respectively. We want to show that $\varphi(t_2)$ follows from $\Gamma_1 \cup \Gamma_2$. Consider a structure $\mathfrak{M}$ with $\mathfrak{M} \models \Gamma_1 \cup \Gamma_2$. By induction hypothesis, $\mathfrak{M} \models \varphi(t_1)$ and $\mathfrak{M} \models t_1 = t_2$. Therefore, $\mathrm{Val}^{\mathfrak{M}}(t_1) = \mathrm{Val}^{\mathfrak{M}}(t_2)$. Let s be any variable assignment, and s' be the x-variant given by $s'(x) = \mathrm{Val}^{\mathfrak{M}}(t_1) = \mathrm{Val}^{\mathfrak{M}}(t_2)$. By Proposition 5.46, $\mathfrak{M}, s \models \varphi(t_1)$ iff $\mathfrak{M}, s' \models \varphi(x)$ iff $\mathfrak{M}, s \models \varphi(t_2)$. Since $\mathfrak{M} \models \varphi(t_1)$, we have $\mathfrak{M} \models \varphi(t_2)$. □

Chapter 8

The Completeness Theorem

8.1 Introduction

The completeness theorem is one of the most fundamental results about logic. It comes in two formulations, the equivalence of which we'll prove. In its first formulation it says something fundamental about the relationship between semantic consequence and our proof system: if a sentence φ follows from some sentences Γ, then there is also a derivation that establishes $\Gamma \vdash \varphi$. Thus, the proof system is as strong as it can possibly be without proving things that don't actually follow.

In its second formulation, it can be stated as a model existence result: every consistent set of sentences is satisfiable. Consistency is a proof-theoretic notion: it says that our proof system is unable to produce certain derivations. But who's to say that just because there are no derivations of a certain sort from Γ, it's guaranteed that there is a structure $\mathfrak{M}$ with $\mathfrak{M} \models \Gamma$? Before the completeness theorem was first proved—in fact before we had the proof systems we now do—the great German mathematician David Hilbert held the view that consistency of mathematical theories guarantees the existence of the objects they are about. He put it as follows in a letter to Gottlob Frege:

> If the arbitrarily given axioms do not contradict one another with all their consequences, then they are true and the things defined by the axioms exist. This is for me the criterion of truth and existence.

Frege vehemently disagreed. The second formulation of the completeness theorem shows that Hilbert was right in at least the sense that if the axioms are consistent, then *some* structure exists that makes them all true.

These aren't the only reasons the completeness theorem—or rather, its proof—is important. It has a number of important consequences, some of which we'll discuss separately. For instance, since any derivation that shows $\Gamma \vdash \varphi$ is finite and so can only use finitely many of the sentences in Γ, it follows by the completeness theorem that if φ is a consequence of Γ, it is already

a consequence of a finite subset of Γ. This is called *compactness*. Equivalently, if every finite subset of Γ is consistent, then Γ itself must be consistent.

Although the compactness theorem follows from the completeness theorem via the detour through derivations, it is also possible to use the *the proof of* the completeness theorem to establish it directly. For what the proof does is take a set of sentences with a certain property—consistency—and constructs a structure out of this set that has certain properties (in this case, that it satisfies the set). Almost the very same construction can be used to directly establish compactness, by starting from "finitely satisfiable" sets of sentences instead of consistent ones. The construction also yields other consequences, e.g., that any satisfiable set of sentences has a finite or countably infinite model. (This result is called the Löwenheim-Skolem theorem.) In general, the construction of structures from sets of sentences is used often in logic, and sometimes even in philosophy.

8.2 Outline of the Proof

The proof of the completeness theorem is a bit complex, and upon first reading it, it is easy to get lost. So let us outline the proof. The first step is a shift of perspective, that allows us to see a route to a proof. When completeness is thought of as "whenever $\Gamma \vDash \varphi$ then $\Gamma \vdash \varphi$," it may be hard to even come up with an idea: for to show that $\Gamma \vdash \varphi$ we have to find a derivation, and it does not look like the hypothesis that $\Gamma \vDash \varphi$ helps us for this in any way. For some proof systems it is possible to directly construct a derivation, but we will take a slightly different tack. The shift in perspective required is this: completeness can also be formulated as: "if Γ is consistent, it has a model." Perhaps we can use the information in Γ together with the hypothesis that it is consistent to construct a model. After all, we know what kind of model we are looking for: one that is as Γ describes it!

If Γ contains only atomic sentences, it is easy to construct a model for it: for atomic sentences are all of the form $P(a_1, \ldots, a_n)$ where the a_i are constant symbols. So all we have to do is come up with a domain $|\mathfrak{M}|$ and an interpretation for P so that $\mathfrak{M} \models P(a_1, \ldots, a_n)$. But nothing's easier than that: put $|\mathfrak{M}| = \mathbb{N}$, $c_i^{\mathfrak{M}} = i$, and for every $P(a_1, \ldots, a_n) \in \Gamma$, put the tuple $\langle k_1, \ldots, k_n \rangle$ into $P^{\mathfrak{M}}$, where k_i is the index of the constant symbol a_i (i.e., $a_i \equiv c_{k_i}$).

Now suppose Γ contains some sentence $\sim\psi$, with ψ atomic. We might worry that the construction of $\mathfrak{M}$ interferes with the possibility of making $\sim\psi$ true. But here's where the consistency of Γ comes in: if $\sim\psi \in \Gamma$, then $\psi \notin \Gamma$, or else Γ would be inconsistent. And if $\psi \notin \Gamma$, then according to our construction of $\mathfrak{M}$, $\mathfrak{M} \not\models \psi$, so $\mathfrak{M} \models \sim\psi$. So far so good.

Now what if Γ contains complex, non-atomic formulas? Say, it contains $\varphi \,\&\, \psi$. Then we should proceed as if both φ and ψ were in Γ. And if $\varphi \vee \psi \in \Gamma$,

then we will have to make at least one of them true, i.e., proceed as if one of them was in Γ.

This suggests the following idea: we add additional sentences to Γ so as to (a) keep the resulting set consistent and (b) make sure that for every possible atomic sentence φ, either φ is in the resulting set, or $\sim\varphi$, and (c) such that, whenever $\varphi \,\&\, \psi$ is in the set, so are both φ and ψ, if $\varphi \vee \psi$ is in the set, at least one of φ or ψ is also, etc. We keep doing this (potentially forever). Call the set of all sentences so added Γ^*. Then our construction above would provide us with a structure for which we could prove, by induction, that all sentences in Γ^* are true in $\mathfrak{M}$, and hence also all sentence in Γ since $\Gamma \subseteq \Gamma^*$. It turns out that guaranteeing (a) is enough. A set of sentences for which (a) holds is called *complete*. So our task will be to extend the consistent set Γ to a consistent and complete set Γ^*.

There is one wrinkle in this plan: if $\exists x\, \varphi(x) \in \Gamma$ we would hope to be able to pick some constant symbol c and add $\varphi(c)$ in this process. But how do we know we can always do that? Perhaps we only have a few constant symbols in our language, and for each one of them we have $\sim\varphi(c) \in \Gamma$. We can't also add $\varphi(c)$, since this would make the set inconsistent, and we wouldn't know whether $\mathfrak{M}$ has to make $\varphi(c)$ or $\sim\varphi(c)$ true. Moreover, it might happen that Γ contains only sentences in a language that has no constant symbols at all (e.g., the language of set theory).

The solution to this problem is to simply add infinitely many constants at the beginning, plus sentences that connect them with the quantifiers in the right way. (Of course, we have to verify that this cannot introduce an inconsistency.)

Our original construction works well if we only have constant symbols in the atomic sentences. But the language might also contain function symbols. In that case, it might be tricky to find the right functions on $\mathbb{N}$ to assign to these function symbols to make everything work. So here's another trick: instead of using i to interpret c_i, just take the set of constant symbols itself as the domain. Then $\mathfrak{M}$ can assign every constant symbol to itself: $c_i^{\mathfrak{M}} = c_i$. But why not go all the way: let $|\mathfrak{M}|$ be all *terms* of the language! If we do this, there is an obvious assignment of functions (that take terms as arguments and have terms as values) to function symbols: we assign to the function symbol f_i^n the function which, given n terms $t_1, \ldots, t_n$ as input, produces the term $f_i^n(t_1, \ldots, t_n)$ as value.

The last piece of the puzzle is what to do with $=$. The predicate symbol $=$ has a fixed interpretation: $\mathfrak{M} \models t = t'$ iff $\mathrm{Val}^{\mathfrak{M}}(t) = \mathrm{Val}^{\mathfrak{M}}(t')$. Now if we set things up so that the value of a term t is t itself, then this structure will make *no* sentence of the form $t = t'$ true unless t and t' are one and the same term. And of course this is a problem, since basically every interesting theory in a language with function symbols will have as theorems sentences $t = t'$ where t and t' are not the same term (e.g., in theories of arithmetic: $(\mathsf{0} + \mathsf{0}) = \mathsf{0}$). To

solve this problem, we change the domain of $\mathfrak{M}$: instead of using terms as the objects in $|\mathfrak{M}|$, we use sets of terms, and each set is so that it contains all those terms which the sentences in Γ require to be equal. So, e.g., if Γ is a theory of arithmetic, one of these sets will contain: o, $(\mathsf{o}+\mathsf{o})$, $(\mathsf{o}\times\mathsf{o})$, etc. This will be the set we assign to o, and it will turn out that this set is also the value of all the terms in it, e.g., also of $(\mathsf{o}+\mathsf{o})$. Therefore, the sentence $(\mathsf{o}+\mathsf{o})=\mathsf{o}$ will be true in this revised structure.

So here's what we'll do. First we investigate the properties of complete consistent sets, in particular we prove that a complete consistent set contains $\varphi \,\&\, \psi$ iff it contains both φ and ψ, $\varphi \vee \psi$ iff it contains at least one of them, etc. (Proposition 8.2). Then we define and investigate "saturated" sets of sentences. A saturated set is one which contains conditionals that link each quantified sentence to instances of it (Definition 8.5). We show that any consistent set Γ can always be extended to a saturated set Γ' (Lemma 8.6). If a set is consistent, saturated, and complete it also has the property that it contains $\exists x\, \varphi(x)$ iff it contains $\varphi(t)$ for some closed term t and $\forall x\, \varphi(x)$ iff it contains $\varphi(t)$ for all closed terms t (Proposition 8.7). We'll then take the saturated consistent set Γ' and show that it can be extended to a saturated, consistent, and complete set Γ^* (Lemma 8.8). This set Γ^* is what we'll use to define our term model $\mathfrak{M}(\Gamma^*)$. The term model has the set of closed terms as its domain, and the interpretation of its predicate symbols is given by the atomic sentences in Γ^* (Definition 8.9). We'll use the properties of consistent, complete, saturated sets to show that indeed $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\varphi \in \Gamma^*$ (Lemma 8.11), and thus in particular, $\mathfrak{M}(\Gamma^*) \models \Gamma$. Finally, we'll consider how to define a term model if Γ contains $=$ as well (Definition 8.15) and show that it satisfies Γ^* (Lemma 8.17).

8.3 Complete Consistent Sets of Sentences

Definition 8.1 (Complete set)**.** A set Γ of sentences is *complete* iff for any sentence φ, either $\varphi \in \Gamma$ or $\sim\varphi \in \Gamma$.

Complete sets of sentences leave no questions unanswered. For any sentence A, Γ "says" if φ is true or false. The importance of complete sets extends beyond the proof of the completeness theorem. A theory which is complete and axiomatizable, for instance, is always decidable.

Complete consistent sets are important in the completeness proof since we can guarantee that every consistent set of sentences Γ is contained in a complete consistent set Γ^*. A complete consistent set contains, for each sentence φ, either φ or its negation $\sim\varphi$, but not both. This is true in particular for atomic sentences, so from a complete consistent set in a language suitably expanded by constant symbols, we can construct a structure where the interpretation of predicate symbols is defined according to which atomic sentences are in Γ^*. This structure can then be shown to make all sentences in Γ^* (and hence also

all those in Γ) true. The proof of this latter fact requires that $\sim\varphi \in \Gamma^*$ iff $\varphi \notin \Gamma^*$, $(\varphi \vee \psi) \in \Gamma^*$ iff $\varphi \in \Gamma^*$ or $\psi \in \Gamma^*$, etc.

In what follows, we will often tacitly use the properties of reflexivity, monotonicity, and transitivity of $\vdash$ (see section 7.7).

Proposition 8.2. *Suppose Γ is complete and consistent. Then:*

1. *If $\Gamma \vdash \varphi$, then $\varphi \in \Gamma$.*
2. *$\varphi \,\&\, \psi \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$.*
3. *$\varphi \vee \psi \in \Gamma$ iff either $\varphi \in \Gamma$ or $\psi \in \Gamma$.*
4. *$\varphi \supset \psi \in \Gamma$ iff either $\varphi \notin \Gamma$ or $\psi \in \Gamma$.*

Proof. Let us suppose for all of the following that Γ is complete and consistent.

1. If $\Gamma \vdash \varphi$, then $\varphi \in \Gamma$.

   Suppose that $\Gamma \vdash \varphi$. Suppose to the contrary that $\varphi \notin \Gamma$. Since Γ is complete, $\sim\varphi \in \Gamma$. By Proposition 7.20, Γ is inconsistent. This contradicts the assumption that Γ is consistent. Hence, it cannot be the case that $\varphi \notin \Gamma$, so $\varphi \in \Gamma$.

2. $\varphi \,\&\, \psi \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$:

   For the forward direction, suppose $\varphi \,\&\, \psi \in \Gamma$. Then by Proposition 7.22, item (1), $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$. By (1), $\varphi \in \Gamma$ and $\psi \in \Gamma$, as required.

   For the reverse direction, let $\varphi \in \Gamma$ and $\psi \in \Gamma$. By Proposition 7.22, item (2), $\Gamma \vdash \varphi \,\&\, \psi$. By (1), $\varphi \,\&\, \psi \in \Gamma$.

3. First we show that if $\varphi \vee \psi \in \Gamma$, then either $\varphi \in \Gamma$ or $\psi \in \Gamma$. Suppose $\varphi \vee \psi \in \Gamma$ but $\varphi \notin \Gamma$ and $\psi \notin \Gamma$. Since Γ is complete, $\sim\varphi \in \Gamma$ and $\sim\psi \in \Gamma$. By Proposition 7.23, item (1), Γ is inconsistent, a contradiction. Hence, either $\varphi \in \Gamma$ or $\psi \in \Gamma$.

   For the reverse direction, suppose that $\varphi \in \Gamma$ or $\psi \in \Gamma$. By Proposition 7.23, item (2), $\Gamma \vdash \varphi \vee \psi$. By (1), $\varphi \vee \psi \in \Gamma$, as required.

4. Exercise.

□

8.4 Henkin Expansion

Part of the challenge in proving the completeness theorem is that the model we construct from a complete consistent set Γ must make all the quantified formulae in Γ true. In order to guarantee this, we use a trick due to Leon Henkin. In essence, the trick consists in expanding the language by infinitely

many constant symbols and adding, for each formula with one free variable $\varphi(x)$ a formula of the form $\exists x\, \varphi \supset \varphi(c)$, where c is one of the new constant symbols. When we construct the structure satisfying Γ, this will guarantee that each true existential sentence has a witness among the new constants.

Proposition 8.3. *If Γ is consistent in $\mathcal{L}$ and $\mathcal{L}'$ is obtained from $\mathcal{L}$ by adding a countably infinite set of new constant symbols d_0, d_1, ..., then Γ is consistent in $\mathcal{L}'$.*

Definition 8.4 (Saturated set)**.** A set Γ of formulae of a language $\mathcal{L}$ is *saturated* iff for each formula $\varphi(x) \in \mathrm{Frm}(\mathcal{L})$ with one free variable x there is a constant symbol $c \in \mathcal{L}$ such that $\exists x\, \varphi(x) \supset \varphi(c) \in \Gamma$.

The following definition will be used in the proof of the next theorem.

Definition 8.5. Let $\mathcal{L}'$ be as in Proposition 8.3. Fix an enumeration $\varphi_0(x_0)$, $\varphi_1(x_1)$, ... of all formulae $\varphi_i(x_i)$ of $\mathcal{L}'$ in which one variable (x_i) occurs free. We define the sentences θ_n by induction on n.

Let c_0 be the first constant symbol among the d_i we added to $\mathcal{L}$ which does not occur in $\varphi_0(x_0)$. Assuming that θ_0, ..., θ_{n-1} have already been defined, let c_n be the first among the new constant symbols d_i that occurs neither in θ_0, ..., θ_{n-1} nor in $\varphi_n(x_n)$.

Now let θ_n be the formula $\exists x_n\, \varphi_n(x_n) \supset \varphi_n(c_n)$.

Lemma 8.6. *Every consistent set Γ can be extended to a saturated consistent set Γ'.*

Proof. Given a consistent set of sentences Γ in a language $\mathcal{L}$, expand the language by adding a countably infinite set of new constant symbols to form $\mathcal{L}'$. By Proposition 8.3, Γ is still consistent in the richer language. Further, let θ_i be as in Definition 8.5. Let

$$\Gamma_0 = \Gamma$$
$$\Gamma_{n+1} = \Gamma_n \cup \{\theta_n\}$$

i.e., $\Gamma_{n+1} = \Gamma \cup \{\theta_0, \ldots, \theta_n\}$, and let $\Gamma' = \bigcup_n \Gamma_n$. Γ' is clearly saturated.

If Γ' were inconsistent, then for some n, Γ_n would be inconsistent (Exercise: explain why). So to show that Γ' is consistent it suffices to show, by induction on n, that each set Γ_n is consistent.

The induction basis is simply the claim that $\Gamma_0 = \Gamma$ is consistent, which is the hypothesis of the theorem. For the induction step, suppose that Γ_n is consistent but $\Gamma_{n+1} = \Gamma_n \cup \{\theta_n\}$ is inconsistent. Recall that θ_n is $\exists x_n\, \varphi_n(x_n) \supset \varphi_n(c_n)$, where $\varphi_n(x_n)$ is a formula of $\mathcal{L}'$ with only the variable x_n free. By the way we've chosen the c_n (see Definition 8.5), c_n does not occur in $A_n(x_n)$ nor in Γ_n.

If $\Gamma_n \cup \{\theta_n\}$ is inconsistent, then $\Gamma_n \vdash \sim\theta_n$, and hence both of the following hold:

$$\Gamma_n \vdash \exists x_n\, \varphi_n(x_n) \qquad \Gamma_n \vdash \sim\varphi_n(c_n)$$

Since c_n does not occur in Γ_n or in $\varphi_n(x_n)$, Theorem 7.25 applies. From $\Gamma_n \vdash \sim\varphi_n(c_n)$, we obtain $\Gamma_n \vdash \forall x_n \sim\varphi_n(x_n)$. Thus we have that both $\Gamma_n \vdash \exists x_n\, \varphi_n$ and $\Gamma_n \vdash \forall x_n \sim\varphi_n(x_n)$, so Γ_n itself is inconsistent. (Note that $\forall x_n \sim\varphi_n(x_n) \vdash \sim\exists x_n\, \varphi_n(x_n)$.) Contradiction: Γ_n was supposed to be consistent. Hence $\Gamma_n \cup \{\theta_n\}$ is consistent. □

We'll now show that *complete*, consistent sets which are saturated have the property that it contains a universally quantified sentence iff it contains all its instances and it contains an existentially quantified sentence iff it contains at least one instance. We'll use this to show that the structure we'll generate from a complete, consistent, saturated set makes all its quantified sentences true.

Proposition 8.7. *Suppose Γ is complete, consistent, and saturated.*

1. *$\exists x\, \varphi(x) \in \Gamma$ iff $\varphi(t) \in \Gamma$ for at least one closed term t.*
2. *$\forall x\, \varphi(x) \in \Gamma$ iff $\varphi(t) \in \Gamma$ for all closed terms t.*

Proof. 1. First suppose that $\exists x\, \varphi(x) \in \Gamma$. Because Γ is saturated, $(\exists x\, \varphi(x) \supset \varphi(c)) \in \Gamma$ for some constant symbol c. By Proposition 7.24, item (1), and Proposition 8.2(1), $\varphi(c) \in \Gamma$.

   For the other direction, saturation is not necessary: Suppose $\varphi(t) \in \Gamma$. Then $\Gamma \vdash \exists x\, \varphi(x)$ by Proposition 7.26, item (1). By Proposition 8.2(1), $\exists x\, \varphi(x) \in \Gamma$.

2. Exercise.

□

8.5 Lindenbaum's Lemma

We now prove a lemma that shows that any consistent set of sentences is contained in some set of sentences which is not just consistent, but also complete. The proof works by adding one sentence at a time, guaranteeing at each step that the set remains consistent. We do this so that for every φ, either φ or $\sim\varphi$ gets added at some stage. The union of all stages in that construction then contains either φ or its negation $\sim\varphi$ and is thus complete. It is also consistent, since we made sure at each stage not to introduce an inconsistency.

Lemma 8.8 (Lindenbaum's Lemma)**.** *Every consistent set Γ' in a language $\mathcal{L}'$ can be extended to a complete and consistent set Γ^*.*

Proof. Let Γ' be consistent. Let $\varphi_0, \varphi_1, \ldots$ be an enumeration of all the formulae of $\mathcal{L}'$. Define $\Gamma_0 = \Gamma'$, and

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

Let $\Gamma^* = \bigcup_{n \geq 0} \Gamma_n$.

Each Γ_n is consistent: Γ_0 is consistent by definition. If $\Gamma_{n+1} = \Gamma_n \cup \{\varphi_n\}$, this is because the latter is consistent. If it isn't, $\Gamma_{n+1} = \Gamma_n \cup \{\sim\varphi_n\}$. We have to verify that $\Gamma_n \cup \{\sim\varphi_n\}$ is consistent. Suppose it's not. Then *both* $\Gamma_n \cup \{\varphi_n\}$ and $\Gamma_n \cup \{\sim\varphi_n\}$ are inconsistent. This means that Γ_n would be inconsistent by Proposition 7.20, contrary to the induction hypothesis.

Every finite subset of Γ^* is a subset of Γ_n for some n, since each $\psi \in \Gamma^*$ not already in Γ' is added at some stage i. If n is the last one of these, then all ψ in the finite subset are in Γ_n. So, every finite subset of Γ^* is consistent. By Proposition 7.17, Γ^* is consistent.

Every sentence of $\mathrm{Frm}(\mathcal{L}')$ appears on the list used to define Γ^*. If $\varphi_n \notin \Gamma^*$, then that is because $\Gamma_n \cup \{\varphi_n\}$ was inconsistent. But then $\sim\varphi_n \in \Gamma^*$, so Γ^* is complete. □

8.6 Construction of a Model

Right now we are not concerned about $=$, i.e., we only want to show that a consistent set Γ of sentences not containing $=$ is satisfiable. We first extend Γ to a consistent, complete, and saturated set Γ^*. In this case, the definition of a model $\mathfrak{M}(\Gamma^*)$ is simple: We take the set of closed terms of $\mathcal{L}'$ as the domain. We assign every constant symbol to itself, and make sure that more generally, for every closed term t, $\mathrm{Val}^{\mathfrak{M}(\Gamma^*)}(t) = t$. The predicate symbols are assigned extensions in such a way that an atomic sentence is true in $\mathfrak{M}(\Gamma^*)$ iff it is in Γ^*. This will obviously make all the atomic sentences in Γ^* true in $\mathfrak{M}(\Gamma^*)$. The rest are true provided the Γ^* we start with is consistent, complete, and saturated.

Definition 8.9 (Term model)**.** Let Γ^* be a complete and consistent, saturated set of sentences in a language $\mathcal{L}$. The *term model* $\mathfrak{M}(\Gamma^*)$ of Γ^* is the structure defined as follows:

1. The domain $|\mathfrak{M}(\Gamma^*)|$ is the set of all closed terms of $\mathcal{L}$.
2. The interpretation of a constant symbol c is c itself: $c^{\mathfrak{M}(\Gamma^*)} = c$.
3. The function symbol f is assigned the function which, given as arguments the closed terms $t_1, \ldots, t_n$, has as value the closed term $f(t_1, \ldots, t_n)$:
$$f^{\mathfrak{M}(\Gamma^*)}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$
4. If R is an n-place predicate symbol, then
$$\langle t_1, \ldots, t_n \rangle \in R^{\mathfrak{M}(\Gamma^*)} \text{ iff } R(t_1, \ldots, t_n) \in \Gamma^*.$$

A structure $\mathfrak{M}$ may make an existentially quantified sentence $\exists x\, \varphi(x)$ true without there being an instance $\varphi(t)$ that it makes true. A structure $\mathfrak{M}$ may

make all instances $\varphi(t)$ of a universally quantified sentence $\forall x\, \varphi(x)$ true, without making $\forall x\, \varphi(x)$ true. This is because in general not every element of $|\mathfrak{M}|$ is the value of a closed term ($\mathfrak{M}$ may not be covered). This is the reason the satisfaction relation is defined via variable assignments. However, for our term model $\mathfrak{M}(\Gamma^*)$ this wouldn't be necessary—because it is covered. This is the content of the next result.

Proposition 8.10. *Let $\mathfrak{M}(\Gamma^*)$ be the term model of Definition 8.9.*

1. *$\mathfrak{M}(\Gamma^*) \models \exists x\, \varphi(x)$ iff $\mathfrak{M} \models \varphi(t)$ for at least one term t.*
2. *$\mathfrak{M}(\Gamma^*) \models \forall x\, \varphi(x)$ iff $\mathfrak{M} \models \varphi(t)$ for all terms t.*

Proof. 1. By Proposition 5.42, $\mathfrak{M}(\Gamma^*) \models \exists x\, \varphi(x)$ iff for at least one variable assignment s, $\mathfrak{M}(\Gamma^*), s \models \varphi(x)$. As $|\mathfrak{M}(\Gamma^*)|$ consists of the closed terms of $\mathcal{L}$, this is the case iff there is at least one closed term t such that $s(x) = t$ and $\mathfrak{M}(\Gamma^*), s \models \varphi(x)$. By Proposition 5.46, $\mathfrak{M}(\Gamma^*), s \models \varphi(x)$ iff $\mathfrak{M}(\Gamma^*), s \models \varphi(t)$, where $s(x) = t$. By Proposition 5.41, $\mathfrak{M}(\Gamma^*), s \models \varphi(t)$ iff $\mathfrak{M}(\Gamma^*) \models \varphi(t)$, since $\varphi(t)$ is a sentence.

2. Exercise.

□

Lemma 8.11 (Truth Lemma)**.** *Suppose φ does not contain $=$. Then $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\varphi \in \Gamma^*$.*

Proof. We prove both directions simultaneously, and by induction on φ.

1. $\varphi \equiv \bot$: $\mathfrak{M}(\Gamma^*) \not\models \bot$ by definition of satisfaction. On the other hand, $\bot \notin \Gamma^*$ since Γ^* is consistent.
2. $\varphi \equiv R(t_1, \ldots, t_n)$: $\mathfrak{M}(\Gamma^*) \models R(t_1, \ldots, t_n)$ iff $\langle t_1, \ldots, t_n \rangle \in R^{\mathfrak{M}(\Gamma^*)}$ (by the definition of satisfaction) iff $R(t_1, \ldots, t_n) \in \Gamma^*$ (by the construction of $\mathfrak{M}(\Gamma^*)$).
3. $\varphi \equiv \sim\psi$: $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\mathfrak{M}(\Gamma^*) \not\models \psi$ (by definition of satisfaction). By induction hypothesis, $\mathfrak{M}(\Gamma^*) \not\models \psi$ iff $\psi \notin \Gamma^*$. Since Γ^* is consistent and complete, $\psi \notin \Gamma^*$ iff $\sim\psi \in \Gamma^*$.
4. $\varphi \equiv \psi \,\&\, \chi$: $\mathfrak{M}(\Gamma^*) \models \varphi$ iff we have both $\mathfrak{M}(\Gamma^*) \models \psi$ and $\mathfrak{M}(\Gamma^*) \models \chi$ (by definition of satisfaction) iff both $\psi \in \Gamma^*$ and $\chi \in \Gamma^*$ (by the induction hypothesis). By Proposition 8.2(2), this is the case iff $(\psi \,\&\, \chi) \in \Gamma^*$.
5. $\varphi \equiv \psi \vee \chi$: $\mathfrak{M}(\Gamma^*) \models \varphi$ iff at $\mathfrak{M}(\Gamma^*) \models \psi$ or $\mathfrak{M}(\Gamma^*) \models \chi$ (by definition of satisfaction) iff $\psi \in \Gamma^*$ or $\chi \in \Gamma^*$ (by induction hypothesis). This is the case iff $(\psi \vee \chi) \in \Gamma^*$ (by Proposition 8.2(3)).
6. $\varphi \equiv \psi \supset \chi$: exercise.

7. $\varphi \equiv \forall x\, \psi(x)$: exercise.

8. $\varphi \equiv \exists x\, \psi(x)$: $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\mathfrak{M}(\Gamma^*) \models \psi(t)$ for at least one term t (Proposition 8.10). By induction hypothesis, this is the case iff $\psi(t) \in \Gamma^*$ for at least one term t. By Proposition 8.7, this in turn is the case iff $\exists x\, \varphi(x) \in \Gamma^*$.

□

8.7 Identity

The construction of the term model given in the preceding section is enough to establish completeness for first-order logic for sets Γ that do not contain $=$. The term model satisfies every $\varphi \in \Gamma^*$ which does not contain $=$ (and hence all $\varphi \in \Gamma$). It does not work, however, if $=$ is present. The reason is that Γ^* then may contain a sentence $t = t'$, but in the term model the value of any term is that term itself. Hence, if t and t' are different terms, their values in the term model—i.e., t and t', respectively—are different, and so $t = t'$ is false. We can fix this, however, using a construction known as "factoring."

Definition 8.12. Let Γ^* be a consistent and complete set of sentences in $\mathcal{L}$. We define the relation $\approx$ on the set of closed terms of $\mathcal{L}$ by

$$t \approx t' \quad \text{iff} \quad t = t' \in \Gamma^*$$

Proposition 8.13. *The relation $\approx$ has the following properties:*

1. *$\approx$ is reflexive.*
2. *$\approx$ is symmetric.*
3. *$\approx$ is transitive.*
4. *If $t \approx t'$, f is a function symbol, and $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$ are terms, then*

$$f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n) \approx f(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots, t_n).$$

5. *If $t \approx t'$, R is a predicate symbol, and $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$ are terms, then*

$$R(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n) \in \Gamma^* \text{ iff } R(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots, t_n) \in \Gamma^*.$$

Proof. Since Γ^* is consistent and complete, $t = t' \in \Gamma^*$ iff $\Gamma^* \vdash t = t'$. Thus it is enough to show the following:

1. $\Gamma^* \vdash t = t$ for all terms t.

2. If $\Gamma^* \vdash t = t'$ then $\Gamma^* \vdash t' = t$.
3. If $\Gamma^* \vdash t = t'$ and $\Gamma^* \vdash t' = t''$, then $\Gamma^* \vdash t = t''$.
4. If $\Gamma^* \vdash t = t'$, then

   $$\Gamma^* \vdash f(t_1, \ldots, t_{i-1}, t, t_{i+1}, , \ldots, t_n) = f(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots, t_n)$$

   for every n-place function symbol f and terms $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$.
5. If $\Gamma^* \vdash t = t'$ and $\Gamma^* \vdash R(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n)$, then $\Gamma^* \vdash R(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots, t_n)$ for every n-place predicate symbol R and terms $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$.

□

Definition 8.14. Suppose Γ^* is a consistent and complete set in a language $\mathcal{L}$, t is a term, and $\approx$ as in the previous definition. Then:

$$[t]_\approx = \{t' \mid t' \in \mathrm{Trm}(\mathcal{L}), t \approx t'\}$$

and $\mathrm{Trm}(\mathcal{L})/_\approx = \{[t]_\approx \mid t \in \mathrm{Trm}(\mathcal{L})\}$.

Definition 8.15. Let $\mathfrak{M} = \mathfrak{M}(\Gamma^*)$ be the term model for Γ^*. Then $\mathfrak{M}/_\approx$ is the following structure:

1. $|\mathfrak{M}/_\approx| = \mathrm{Trm}(\mathcal{L})/_\approx$.
2. $c^{\mathfrak{M}/_\approx} = [c]_\approx$
3. $f^{\mathfrak{M}/_\approx}([t_1]_\approx, \ldots, [t_n]_\approx) = [f(t_1, \ldots, t_n)]_\approx$
4. $\langle [t_1]_\approx, \ldots, [t_n]_\approx \rangle \in R^{\mathfrak{M}/_\approx}$ iff $\mathfrak{M} \models R(t_1, \ldots, t_n)$.

Note that we have defined $f^{\mathfrak{M}/_\approx}$ and $R^{\mathfrak{M}/_\approx}$ for elements of $\mathrm{Trm}(\mathcal{L})/_\approx$ by referring to them as $[t]_\approx$, i.e., via *representatives* $t \in [t]_\approx$. We have to make sure that these definitions do not depend on the choice of these representatives, i.e., that for some other choices t' which determine the same equivalence classes ($[t]_\approx = [t']_\approx$), the definitions yield the same result. For instance, if R is a one-place predicate symbol, the last clause of the definition says that $[t]_\approx \in R^{\mathfrak{M}/_\approx}$ iff $\mathfrak{M} \models R(t)$. If for some other term t' with $t \approx t'$, $\mathfrak{M} \not\models R(t)$, then the definition would require $[t']_\approx \notin R^{\mathfrak{M}/_\approx}$. If $t \approx t'$, then $[t]_\approx = [t']_\approx$, but we can't have both $[t]_\approx \in R^{\mathfrak{M}/_\approx}$ and $[t]_\approx \notin R^{\mathfrak{M}/_\approx}$. However, Proposition 8.13 guarantees that this cannot happen.

Proposition 8.16. *$\mathfrak{M}/_\approx$ is well defined, i.e., if $t_1, \ldots, t_n$, $t'_1, \ldots, t'_n$ are terms, and $t_i \approx t'_i$ then*

1. *$[f(t_1, \ldots, t_n)]_\approx = [f(t'_1, \ldots, t'_n)]_\approx$, i.e.,*

   $$f(t_1, \ldots, t_n) \approx f(t'_1, \ldots, t'_n)$$

   and

2. $\mathfrak{M} \models R(t_1, \ldots, t_n)$ *iff* $\mathfrak{M} \models R(t'_1, \ldots, t'_n)$, *i.e.,*

$$R(t_1, \ldots, t_n) \in \Gamma^* \text{ iff } R(t'_1, \ldots, t'_n) \in \Gamma^*.$$

Proof. Follows from Proposition 8.13 by induction on n. □

Lemma 8.17. $\mathfrak{M}/{\approx} \models \varphi$ *iff* $\varphi \in \Gamma^*$ *for all sentences* φ.

Proof. By induction on φ, just as in the proof of Lemma 8.11. The only case that needs additional attention is when $\varphi \equiv t = t'$.

$$\begin{aligned}\mathfrak{M}/{\approx} \models t = t' &\text{ iff } [t]_\approx = [t']_\approx \text{ (by definition of } \mathfrak{M}/{\approx})\\ &\text{iff } t \approx t' \text{ (by definition of } [t]_\approx)\\ &\text{iff } t = t' \in \Gamma^* \text{ (by definition of } \approx).\end{aligned}$$

□

Note that while $\mathfrak{M}(\Gamma^*)$ is always countable and infinite, $\mathfrak{M}/{\approx}$ may be finite, since it may turn out that there are only finitely many classes $[t]_\approx$. This is to be expected, since Γ may contain sentences which require any structure in which they are true to be finite. For instance, $\forall x \, \forall y \, x = y$ is a consistent sentence, but is satisfied only in structures with a domain that contains exactly one element.

8.8 The Completeness Theorem

Let's combine our results: we arrive at Gödel's completeness theorem.

Theorem 8.18 (Completeness Theorem)**.** *Let* Γ *be a set of sentences. If* Γ *is consistent, it is satisfiable.*

Proof. Suppose Γ is consistent. By Lemma 8.6, there is a saturated consistent set $\Gamma' \supseteq \Gamma$. By Lemma 8.8, there is a $\Gamma^* \supseteq \Gamma'$ which is consistent and complete. Since $\Gamma' \subseteq \Gamma^*$, for each formula φ, Γ^* contains a formula of the form $\exists x \, \varphi \supset \varphi(c)$ and so Γ^* is saturated.

If Γ does not contain $=$, then by Lemma 8.11, $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\varphi \in \Gamma^*$. From this it follows in particular that for all $\varphi \in \Gamma$, $\mathfrak{M}(\Gamma^*) \models \varphi$, so Γ is satisfiable. If Γ does contain $=$, then by Lemma 8.17, $\mathfrak{M}/{\approx} \models \varphi$ iff $\varphi \in \Gamma^*$ for all sentences φ. In particular, $\mathfrak{M}/{\approx} \models \varphi$ for all $\varphi \in \Gamma$, so Γ is satisfiable. □

Corollary 8.19 (Completeness Theorem, Second Version)**.** *For all* Γ *and* φ *sentences: if* $\Gamma \vDash \varphi$ *then* $\Gamma \vdash \varphi$.

Proof. Note that the Γ's in Corollary 8.19 and Theorem 8.18 are universally quantified. To make sure we do not confuse ourselves, let us restate Theorem 8.18 using a different variable: for any set of sentences Δ, if Δ is consistent, it is satisfiable. By contraposition, if Δ is not satisfiable, then Δ is inconsistent. We will use this to prove the corollary.

Suppose that $\Gamma \vDash \varphi$. Then $\Gamma \cup \{\sim\varphi\}$ is unsatisfiable by Proposition 5.51. Taking $\Gamma \cup \{\sim\varphi\}$ as our Δ, the previous version of Theorem 8.18 gives us that $\Gamma \cup \{\sim\varphi\}$ is inconsistent. By Proposition 7.19, $\Gamma \vdash \varphi$. □

8.9 The Compactness Theorem

One important consequence of the completeness theorem is the compactness theorem. The compactness theorem states that if each *finite* subset of a set of sentences is satisfiable, the entire set is satisfiable—even if the set itself is infinite. This is far from obvious. There is nothing that seems to rule out, at first glance at least, the possibility of there being infinite sets of sentences which are contradictory, but the contradiction only arises, so to speak, from the infinite number. The compactness theorem says that such a scenario can be ruled out: there are no unsatisfiable infinite sets of sentences each finite subset of which is satisfiable. Like the completeness theorem, it has a version related to entailment: if an infinite set of sentences entails something, already a finite subset does.

Definition 8.20. A set Γ of formulae is *finitely satisfiable* if and only if every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable.

Theorem 8.21 (Compactness Theorem)**.** *The following hold for any sentences Γ and φ:*

1. *$\Gamma \vDash \varphi$ iff there is a finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \vDash \varphi$.*
2. *Γ is satisfiable if and only if it is finitely satisfiable.*

Proof. We prove (2). If Γ is satisfiable, then there is a structure $\mathfrak{M}$ such that $\mathfrak{M} \models \varphi$ for all $\varphi \in \Gamma$. Of course, this $\mathfrak{M}$ also satisfies every finite subset of Γ, so Γ is finitely satisfiable.

Now suppose that Γ is finitely satisfiable. Then every finite subset $\Gamma_0 \subseteq \Gamma$ is satisfiable. By soundness (Corollary 7.29), every finite subset is consistent. Then Γ itself must be consistent by Proposition 7.17. By completeness (Theorem 8.18), since Γ is consistent, it is satisfiable. □

Example 8.22. In every model $\mathfrak{M}$ of a theory Γ, each term t of course picks out an element of $|\mathfrak{M}|$. Can we guarantee that it is also true that every element of $|\mathfrak{M}|$ is picked out by some term or other? In other words, are there theories Γ all models of which are covered? The compactness theorem shows that this is not the case if Γ has infinite models. Here's how to see this: Let $\mathfrak{M}$ be

an infinite model of Γ, and let c be a constant symbol not in the language of Γ. Let Δ be the set of all sentences $c \neq t$ for t a term in the language $\mathcal{L}$ of Γ, i.e.,

$$\Delta = \{c \neq t \mid t \in \mathrm{Trm}(\mathcal{L})\}.$$

A finite subset of $\Gamma \cup \Delta$ can be written as $\Gamma' \cup \Delta'$, with $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$. Since Δ' is finite, it can contain only finitely many terms. Let $a \in |\mathfrak{M}|$ be an element of $|\mathfrak{M}|$ not picked out by any of them, and let $\mathfrak{M}'$ be the structure that is just like $\mathfrak{M}$, but also $c^{\mathfrak{M}'} = a$. Since $a \neq \mathrm{Val}^{\mathfrak{M}}(t)$ for all t occuring in Δ', $\mathfrak{M}' \models \Delta'$. Since $\mathfrak{M} \models \Gamma$, $\Gamma' \subseteq \Gamma$, and c does not occur in Γ, also $\mathfrak{M}' \models \Gamma'$. Together, $\mathfrak{M}' \models \Gamma' \cup \Delta'$ for every finite subset $\Gamma' \cup \Delta'$ of $\Gamma \cup \Delta$. So every finite subset of $\Gamma \cup \Delta$ is satisfiable. By compactness, $\Gamma \cup \Delta$ itself is satisfiable. So there are models $\mathfrak{M} \models \Gamma \cup \Delta$. Every such $\mathfrak{M}$ is a model of Γ, but is not covered, since $\mathrm{Val}^{\mathfrak{M}}(c) \neq \mathrm{Val}^{\mathfrak{M}}(t)$ for all terms t of $\mathcal{L}$.

Example 8.23. Consider a language $\mathcal{L}$ containing the predicate symbol $<$, constant symbols 0, 1, and function symbols $+$, $\times$, $-$, $\div$. Let Γ be the set of all sentences in this language true in $\mathfrak{Q}$ with domain $\mathbb{Q}$ and the obvious interpretations. Γ is the set of all sentences of $\mathcal{L}$ true about the rational numbers. Of course, in $\mathbb{Q}$ (and even in $\mathbb{R}$), there are no numbers which are greater than 0 but less than $1/k$ for all $k \in \mathbb{Z}^+$. Such a number, if it existed, would be an *infinitesimal:* non-zero, but infinitely small. The compactness theorem shows that there are models of Γ in which infinitesimals exist: Let Δ be $\{0 < c\} \cup \{c < (1 \div \overline{k}) \mid k \in \mathbb{Z}^+\}$ (where $\overline{k} = (1 + (1 + \cdots + (1 + 1)\ldots))$ with k 1's). For any finite subset Δ_0 of Δ there is a K such that all the sentences $c < \overline{k}$ in Δ_0 have $k < K$. If we expand $\mathfrak{Q}$ to $\mathfrak{Q}'$ with $c^{\mathfrak{Q}'} = 1/K$ we have that $\mathfrak{Q}' \models \Gamma \cup \Delta_0$, and so $\Gamma \cup \Delta$ is finitely satisfiable (Exercise: prove this in detail). By compactness, $\Gamma \cup \Delta$ is satisfiable. Any model $\mathfrak{S}$ of $\Gamma \cup \Delta$ contains an infinitesimal, namely $c^{\mathfrak{S}}$.

Example 8.24. We know that first-order logic with identity predicate can express that the size of the domain must have some minimal size: The sentence $\varphi_{\geq n}$ (which says "there are at least n distinct objects") is true only in structures where $|\mathfrak{M}|$ has at least n objects. So if we take

$$\Delta = \{\varphi_{\geq n} \mid n \geq 1\}$$

then any model of Δ must be infinite. Thus, we can guarantee that a theory only has infinite models by adding Δ to it: the models of $\Gamma \cup \Delta$ are all and only the infinite models of Γ.

So first-order logic can express infinitude. The compactness theorem shows that it cannot express finitude, however. For suppose some set of sentences Λ were satisfied in all and only finite structures. Then $\Delta \cup \Lambda$ is finitely satisfiable. Why? Suppose $\Delta' \cup \Lambda' \subseteq \Delta \cup \Lambda$ is finite with $\Delta' \subseteq \Delta$ and $\Lambda' \subseteq \Lambda$. Let n be the largest number such that $A_{\geq n} \in \Delta'$. Λ, being satisfied in all finite structures,

has a model $\mathfrak{M}$ with finitely many but $\geq n$ elements. But then $\mathfrak{M} \models \Delta' \cup \Lambda'$. By compactness, $\Delta \cup \Lambda$ has an infinite model, contradicting the assumption that Λ is satisfied only in finite structures.

8.10 A Direct Proof of the Compactness Theorem

We can prove the Compactness Theorem directly, without appealing to the Completeness Theorem, using the same ideas as in the proof of the completeness theorem. In the proof of the Completeness Theorem we started with a consistent set Γ of sentences, expanded it to a consistent, saturated, and complete set Γ^* of sentences, and then showed that in the term model $\mathfrak{M}(\Gamma^*)$ constructed from Γ^*, all sentences of Γ are true, so Γ is satisfiable.

We can use the same method to show that a finitely satisfiable set of sentences is satisfiable. We just have to prove the corresponding versions of the results leading to the truth lemma where we replace "consistent" with "finitely satisfiable."

Proposition 8.25. *Suppose Γ is complete and finitely satisfiable. Then:*

1. *$(\varphi \mathbin{\&} \psi) \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$.*
2. *$(\varphi \vee \psi) \in \Gamma$ iff either $\varphi \in \Gamma$ or $\psi \in \Gamma$.*
3. *$(\varphi \supset \psi) \in \Gamma$ iff either $\varphi \notin \Gamma$ or $\psi \in \Gamma$.*

Lemma 8.26. *Every finitely satisfiable set Γ can be extended to a saturated finitely satisfiable set Γ'.*

Proposition 8.27. *Suppose Γ is complete, finitely satisfiable, and saturated.*

1. *$\exists x\, \varphi(x) \in \Gamma$ iff $\varphi(t) \in \Gamma$ for at least one closed term t.*
2. *$\forall x\, \varphi(x) \in \Gamma$ iff $\varphi(t) \in \Gamma$ for all closed terms t.*

Lemma 8.28. *Every finitely satisfiable set Γ' can be extended to a complete and finitely satisfiable set Γ^*.*

Theorem 8.29 (Compactness)**.** *Γ is satisfiable if and only if it is finitely satisfiable.*

Proof. If Γ is satisfiable, then there is a structure $\mathfrak{M}$ such that $\mathfrak{M} \models \varphi$ for all $\varphi \in \Gamma$. Of course, this $\mathfrak{M}$ also satisfies every finite subset of Γ, so Γ is finitely satisfiable.

Now suppose that Γ is finitely satisfiable. By Lemma 8.26, there is a finitely satisfiable, saturated set $\Gamma' \supseteq \Gamma$. By Lemma 8.28, Γ' can be extended to a complete and finitely satisfiable set Γ^*, and Γ^* is still saturated. Construct the term model $\mathfrak{M}(\Gamma^*)$ as in Definition 8.9. Note that Proposition 8.10 did not rely on the fact that Γ^* is consistent (or complete or saturated, for that matter),

but just on the fact that $\mathfrak{M}(\Gamma^*)$ is covered. The proof of the Truth Lemma (Lemma 8.11) goes through if we replace references to Proposition 8.2 and Proposition 8.7 by references to Proposition 8.25 and Proposition 8.27. □

8.11 The Löwenheim-Skolem Theorem

The Löwenheim-Skolem Theorem says that if a theory has an infinite model, then it also has a model that is at most countably infinite. An immediate consequene of this fact is that first-order logic cannot express that the size of a structure is uncountable: any sentence or set of sentences satisfied in all uncountable structures is also satisfied in some countable structure.

Theorem 8.30. *If Γ is consistent then it has a countable model, i.e., it is satisfiable in a structure whose domain is either finite or countably infinite.*

Proof. If Γ is consistent, the structure $\mathfrak{M}$ delivered by the proof of the completeness theorem has a domain $|\mathfrak{M}|$ that is no larger than the set of the terms of the language $\mathcal{L}$. So $\mathfrak{M}$ is at most countably infinite. □

Theorem 8.31. *If Γ is consistent set of sentences in the language of first-order logic without identity, then it has a countably infinite model, i.e., it is satisfiable in a structure whose domain is infinite and countable.*

Proof. If Γ is consistent and contains no sentences in which identity appears, then the structure $\mathfrak{M}$ delivered by the proof of the completness theorem has a domain $|\mathfrak{M}|$ identical to the set of terms of the language $\mathcal{L}'$. So $\mathfrak{M}$ is countably infinite, since $\mathrm{Trm}(\mathcal{L}')$ is. □

Example 8.32 (Skolem's Paradox)**.** Zermelo-Fraenkel set theory **ZFC** is a very powerful framework in which practically all mathematical statements can be expressed, including facts about the sizes of sets. So for instance, **ZFC** can prove that the set $\mathbb{R}$ of real numbers is uncountable, it can prove Cantor's Theorem that the power set of any set is larger than the set itself, etc. If **ZFC** is consistent, its models are all infinite, and moreover, they all contain elements about which the theory says that they are uncountable, such as the element that makes true the theorem of **ZFC** that the power set of the natural numbers exists. By the Löwenheim-Skolem Theorem, **ZFC** also has countable models—models that contain "uncountable" sets but which themselves are countable.

Chapter 9

Beyond First-order Logic

9.1 Overview

First-order logic is not the only system of logic of interest: there are many extensions and variations of first-order logic. A logic typically consists of the formal specification of a language, usually, but not always, a deductive system, and usually, but not always, an intended semantics. But the technical use of the term raises an obvious question: what do logics that are not first-order logic have to do with the word "logic," used in the intuitive or philosophical sense? All of the systems described below are designed to model reasoning of some form or another; can we say what makes them logical?

No easy answers are forthcoming. The word "logic" is used in different ways and in different contexts, and the notion, like that of "truth," has been analyzed from numerous philosophical stances. For example, one might take the goal of logical reasoning to be the determination of which statements are necessarily true, true a priori, true independent of the interpretation of the nonlogical terms, true by virtue of their form, or true by linguistic convention; and each of these conceptions requires a good deal of clarification. Even if one restricts one's attention to the kind of logic used in mathematics, there is little agreement as to its scope. For example, in the *Principia Mathematica*, Russell and Whitehead tried to develop mathematics on the basis of logic, in the *logicist* tradition begun by Frege. Their system of logic was a form of higher-type logic similar to the one described below. In the end they were forced to introduce axioms which, by most standards, do not seem purely logical (notably, the axiom of infinity, and the axiom of reducibility), but one might nonetheless hold that some forms of higher-order reasoning should be accepted as logical. In contrast, Quine, whose ontology does not admit "propositions" as legitimate objects of discourse, argues that second-order and higher-order logic are really manifestations of set theory in sheep's clothing; in other words, systems involving quantification over predicates are not purely logical.

For now, it is best to leave such philosophical issues for a rainy day, and

simply think of the systems below as formal idealizations of various kinds of reasoning, logical or otherwise.

9.2 Many-Sorted Logic

In first-order logic, variables and quantifiers range over a single domain. But it is often useful to have multiple (disjoint) domains: for example, you might want to have a domain of numbers, a domain of geometric objects, a domain of functions from numbers to numbers, a domain of abelian groups, and so on.

Many-sorted logic provides this kind of framework. One starts with a list of "sorts"—the "sort" of an object indicates the "domain" it is supposed to inhabit. One then has variables and quantifiers for each sort, and (usually) an identity predicate for each sort. Functions and relations are also "typed" by the sorts of objects they can take as arguments. Otherwise, one keeps the usual rules of first-order logic, with versions of the quantifier-rules repeated for each sort.

For example, to study international relations we might choose a language with two sorts of objects, French citizens and German citizens. We might have a unary relation, "drinks wine," for objects of the first sort; another unary relation, "eats wurst," for objects of the second sort; and a binary relation, "forms a multinational married couple," which takes two arguments, where the first argument is of the first sort and the second argument is of the second sort. If we use variables a, b, c to range over French citizens and x, y, z to range over German citizens, then

$$\forall a \forall x[(MarriedTo(a, x) \supset (DrinksWine(a) \vee \sim EatsWurst(x))]]$$

asserts that if any French person is married to a German, either the French person drinks wine or the German doesn't eat wurst.

Many-sorted logic can be embedded in first-order logic in a natural way, by lumping all the objects of the many-sorted domains together into one first-order domain, using unary predicate symbols to keep track of the sorts, and relativizing quantifiers. For example, the first-order language corresponding to the example above would have unary predicate symbolss "$German$" and "$French$," in addition to the other relations described, with the sort requirements erased. A sorted quantifier $\forall x\, \varphi$, where x is a variable of the German sort, translates to

$$\forall x\, (German(x) \supset \varphi).$$

We need to add axioms that insure that the sorts are separate—e.g., $\forall x \sim(German(x)$ & $French(x))$—as well as axioms that guarantee that "drinks wine" only holds of objects satisfying the predicate $French(x)$, etc. With these conventions and axioms, it is not difficult to show that many-sorted sentences translate to first-order sentences, and many-sorted derivations translate to first-order deriva-

tions. Also, many-sorted structures "translate" to corresponding first-order structures and vice-versa, so we also have a completeness theorem for many-sorted logic.

9.3 Second-Order logic

The language of second-order logic allows one to quantify not just over a domain of individuals, but over relations on that domain as well. Given a first-order language $\mathcal{L}$, for each k one adds variables R which range over k-ary relations, and allows quantification over those variables. If R is a variable for a k-ary relation, and $t_1, \ldots, t_k$ are ordinary (first-order) terms, $R(t_1, \ldots, t_k)$ is an atomic formula. Otherwise, the set of formulae is defined just as in the case of first-order logic, with additional clauses for second-order quantification. Note that we only have the identity predicate for first-order terms: if R and S are relation variables of the same arity k, we can define $R = S$ to be an abbreviation for

$$\forall x_1 \ldots \forall x_k \, (R(x_1, \ldots, x_k) \equiv S(x_1, \ldots, x_k)).$$

The rules for second-order logic simply extend the quantifier rules to the new second order variables. Here, however, one has to be a little bit careful to explain how these variables interact with the predicate symbols of $\mathcal{L}$, and with formulae of $\mathcal{L}$ more generally. At the bare minimum, relation variables count as terms, so one has inferences of the form

$$\varphi(R) \vdash \exists R \, \varphi(R)$$

But if $\mathcal{L}$ is the language of arithmetic with a constant relation symbol $<$, one would also expect the following inference to be valid:

$$x < y \vdash \exists R \, R(x, y)$$

or for a given formula φ,

$$\varphi(x_1, \ldots, x_k) \vdash \exists R \, R(x_1, \ldots, x_k)$$

More generally, we might want to allow inferences of the form

$$\varphi[\lambda \vec{x}. \, \psi(\vec{x}) / R] \vdash \exists R \, \varphi$$

where $\varphi[\lambda \vec{x}. \, \psi(\vec{x}) / R]$ denotes the result of replacing every atomic formula of the form $Rt_1, \ldots, t_k$ in φ by $\psi(t_1, \ldots, t_k)$. This last rule is equivalent to having a *comprehension schema*, i.e., an axiom of the form

$$\exists R \, \forall x_1, \ldots, x_k \, (\varphi(x_1, \ldots, x_k) \equiv R(x_1, \ldots, x_k)),$$

one for each formula φ in the second-order language, in which R is not a free variable. (Exercise: show that if R is allowed to occur in φ, this schema is inconsistent!)

When logicians refer to the "axioms of second-order logic" they usually mean the minimal extension of first-order logic by second-order quantifier rules together with the comprehension schema. But it is often interesting to study weaker subsystems of these axioms and rules. For example, note that in its full generality the axiom schema of comprehension is *impredicative*: it allows one to assert the existence of a relation $R(x_1, \ldots, x_k)$ that is "defined" by a formula with second-order quantifiers; and these quantifiers range over the set of all such relations—a set which includes R itself! Around the turn of the twentieth century, a common reaction to Russell's paradox was to lay the blame on such definitions, and to avoid them in developing the foundations of mathematics. If one prohibits the use of second-order quantifiers in the formula φ, one has a *predicative* form of comprehension, which is somewhat weaker.

From the semantic point of view, one can think of a second-order structure as consisting of a first-order structure for the language, coupled with a set of relations on the domain over which the second-order quantifiers range (more precisely, for each k there is a set of relations of arity k). Of course, if comprehension is included in the proof system, then we have the added requirement that there are enough relations in the "second-order part" to satisfy the comprehension axioms—otherwise the proof system is not sound! One easy way to insure that there are enough relations around is to take the second-order part to consist of *all* the relations on the first-order part. Such a structure is called *full*, and, in a sense, is really the "intended structure" for the language. If we restrict our attention to full structures we have what is known as the *full* second-order semantics. In that case, specifying a structure boils down to specifying the first-order part, since the contents of the second-order part follow from that implicitly.

To summarize, there is some ambiguity when talking about second-order logic. In terms of the proof system, one might have in mind either

1. A "minimal" second-order proof system, together with some comprehension axioms.
2. The "standard" second-order proof system, with full comprehension.

In terms of the semantics, one might be interested in either

1. The "weak" semantics, where a structure consists of a first-order part, together with a second-order part big enough to satisfy the comprehension axioms.
2. The "standard" second-order semantics, in which one considers full structures only.

When logicians do not specify the proof system or the semantics they have in mind, they are usually refering to the second item on each list. The advantage to using this semantics is that, as we will see, it gives us categorical descriptions of many natural mathematical structures; at the same time, the proof system is quite strong, and sound for this semantics. The drawback is that the proof system is *not* complete for the semantics; in fact, *no* effectively given proof system is complete for the full second-order semantics. On the other hand, we will see that the proof system *is* complete for the weakened semantics; this implies that if a sentence is not provable, then there is *some* structure, not necessarily the full one, in which it is false.

The language of second-order logic is quite rich. One can identify unary relations with subsets of the domain, and so in particular you can quantify over these sets; for example, one can express induction for the natural numbers with a single axiom

$$\forall R\,((R(\mathsf{o}) \,\&\, \forall x\,(R(x) \supset R(x'))) \supset \forall x\,R(x)).$$

If one takes the language of arithmetic to have symbols $\mathsf{o}, \prime, +, \times$ and $<$, one can add the following axioms to describe their behavior:

1. $\forall x \sim x' = \mathsf{o}$
2. $\forall x\,\forall y\,(s(x) = s(y) \supset x = y)$
3. $\forall x\,(x + \mathsf{o}) = x$
4. $\forall x\,\forall y\,(x + y') = (x + y)'$
5. $\forall x\,(x \times \mathsf{o}) = \mathsf{o}$
6. $\forall x\,\forall y\,(x \times y') = ((x \times y) + x)$
7. $\forall x\,\forall y\,(x < y \equiv \exists z\,y = (x + z'))$

It is not difficult to show that these axioms, together with the axiom of induction above, provide a categorical description of the structure $\mathfrak{N}$, the standard model of arithmetic, provided we are using the full second-order semantics. Given any structure $\mathfrak{A}$ in which these axioms are true, define a function f from $\mathbb{N}$ to the domain of $\mathfrak{A}$ using ordinary recursion on $\mathbb{N}$, so that $f(0) = \mathsf{o}^{\mathfrak{A}}$ and $f(x+1) = \prime^{\mathfrak{A}}(f(x))$. Using ordinary induction on $\mathbb{N}$ and the fact that axioms (1) and (2) hold in $\mathfrak{A}$, we see that f is injective. To see that f is surjective, let P be the set of elements of $|\mathfrak{A}|$ that are in the range of f. Since $\mathfrak{A}$ is full, P is in the second-order domain. By the construction of f, we know that $\mathsf{o}^{\mathfrak{A}}$ is in P, and that P is closed under $\prime^{\mathfrak{A}}$. The fact that the induction axiom holds in $\mathfrak{A}$ (in particular, for P) guarantees that P is equal to the entire first-order domain of $\mathfrak{A}$. This shows that f is a bijection. Showing that f is a homomorphism is no more difficult, using ordinary induction on $\mathbb{N}$ repeatedly.

In set-theoretic terms, a function is just a special kind of relation; for example, a unary function f can be identified with a binary relation R satisfying $\forall x \exists y\, R(x, y)$. As a result, one can quantify over functions too. Using the full semantics, one can then define the class of infinite structures to be the class of structures $\mathfrak{A}$ for which there is an injective function from the domain of $\mathfrak{A}$ to a proper subset of itself:

$$\exists f\, (\forall x\, \forall y\, (f(x) = f(y) \supset x = y)\, \&\, \exists y\, \forall x\, f(x) \neq y).$$

The negation of this sentence then defines the class of finite structures.

In addition, one can define the class of well-orderings, by adding the following to the definition of a linear ordering:

$$\forall P\, (\exists x\, P(x) \supset \exists x\, (P(x)\, \&\, \forall y\, (y < x \supset \sim P(y)))).$$

This asserts that every non-empty set has a least element, modulo the identification of "set" with "one-place relation". For another example, one can express the notion of connectedness for graphs, by saying that there is no non-trivial separation of the vertices into disconnected parts:

$$\sim \exists A\, (\exists x\, A(x)\, \&\, \exists y \sim A(y)\, \&\, \forall w\, \forall z\, ((A(w)\, \&\, \sim A(z)) \supset \sim R(w, z))).$$

For yet another example, you might try as an exercise to define the class of finite structures whose domain has even size. More strikingly, one can provide a categorical description of the real numbers as a complete ordered field containing the rationals.

In short, second-order logic is much more expressive than first-order logic. That's the good news; now for the bad. We have already mentioned that there is no effective proof system that is complete for the full second-order semantics. For better or for worse, many of the properties of first-order logic are absent, including compactness and the Löwenheim-Skolem theorems.

On the other hand, if one is willing to give up the full second-order semantics in terms of the weaker one, then the minimal second-order proof system is complete for this semantics. In other words, if we read $\vdash$ as "proves in the minimal system" and $\models$ as "logically implies in the weaker semantics", we can show that whenever $\Gamma \models \varphi$ then $\Gamma \vdash \varphi$. If one wants to include specific comprehension axioms in the proof system, one has to restrict the semantics to second-order structures that satisfy these axioms: for example, if Δ consists of a set of comprehension axioms (possibly all of them), we have that if $\Gamma \cup \Delta \models \varphi$, then $\Gamma \cup \Delta \vdash \varphi$. In particular, if φ is not provable using the comprehension axioms we are considering, then there is a model of $\sim\varphi$ in which these comprehension axioms nonetheless hold.

The easiest way to see that the completeness theorem holds for the weaker semantics is to think of second-order logic as a many-sorted logic, as follows. One sort is interpreted as the ordinary "first-order" domain, and then for each

k we have a domain of "relations of arity k." We take the language to have built-in relation symbols "$true_k(R, x_1, \ldots, x_k)$" which is meant to assert that R holds of $x_1, \ldots, x_k$, where R is a variable of the sort "k-ary relation" and x_1, $\ldots$, x_k are objects of the first-order sort.

With this identification, the weak second-order semantics is essentially the usual semantics for many-sorted logic; and we have already observed that many-sorted logic can be embedded in first-order logic. Modulo the translations back and forth, then, the weaker conception of second-order logic is really a form of first-order logic in disguise, where the domain contains both "objects" and "relations" governed by the appropriate axioms.

9.4 Higher-Order logic

Passing from first-order logic to second-order logic enabled us to talk about sets of objects in the first-order domain, within the formal language. Why stop there? For example, third-order logic should enable us to deal with sets of sets of objects, or perhaps even sets which contain both objects and sets of objects. And fourth-order logic will let us talk about sets of objects of that kind. As you may have guessed, one can iterate this idea arbitrarily.

In practice, higher-order logic is often formulated in terms of functions instead of relations. (Modulo the natural identifications, this difference is inessential.) Given some basic "sorts" A, B, C, ...(which we will now call "types"), we can create new ones by stipulating

> If σ and τ are finite types then so is $\sigma \to \tau$.

Think of types as syntactic "labels," which classify the objects we want in our domain; $\sigma \to \tau$ describes those objects that are functions which take objects of type σ to objects of type τ. For example, we might want to have a type Ω of truth values, "true" and "false," and a type $\mathbb{N}$ of natural numbers. In that case, you can think of objects of type $\mathbb{N} \to \Omega$ as unary relations, or subsets of $\mathbb{N}$; objects of type $\mathbb{N} \to \mathbb{N}$ are functions from natural numers to natural numbers; and objects of type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ are "functionals," that is, higher-type functions that take functions to numbers.

As in the case of second-order logic, one can think of higher-order logic as a kind of many-sorted logic, where there is a sort for each type of object we want to consider. But it is usually clearer just to define the syntax of higher-type logic from the ground up. For example, we can define a set of finite types inductively, as follows:

1. $\mathbb{N}$ is a finite type.
2. If σ and τ are finite types, then so is $\sigma \to \tau$.
3. If σ and τ are finite types, so is $\sigma \times \tau$.

Intuitively, $\mathbb{N}$ denotes the type of the natural numbers, $\sigma \to \tau$ denotes the type of functions from σ to τ, and $\sigma \times \tau$ denotes the type of pairs of objects, one from σ and one from τ. We can then define a set of terms inductively, as follows:

1. For each type σ, there is a stock of variables $x, y, z, \ldots$ of type σ
2. 0 is a term of type $\mathbb{N}$
3. S (successor) is a term of type $\mathbb{N} \to \mathbb{N}$
4. If s is a term of type σ, and t is a term of type $\mathbb{N} \to (\sigma \to \sigma)$, then R_{st} is a term of type $\mathbb{N} \to \sigma$
5. If s is a term of type $\tau \to \sigma$ and t is a term of type τ, then $s(t)$ is a term of type σ
6. If s is a term of type σ and x is a variable of type τ, then $\lambda x.\, s$ is a term of type $\tau \to \sigma$.
7. If s is a term of type σ and t is a term of type τ, then $\langle s, t \rangle$ is a term of type $\sigma \times \tau$.
8. If s is a term of type $\sigma \times \tau$ then $p_1(s)$ is a term of type σ and $p_2(s)$ is a term of type τ.

Intuitively, R_{st} denotes the function defined recursively by

$$R_{st}(0) = s$$
$$R_{st}(x+1) = t(x, R_{st}(x)),$$

$\langle s, t \rangle$ denotes the pair whose first component is s and whose second component is t, and $p_1(s)$ and $p_2(s)$ denote the first and second elements ("projections") of s. Finally, $\lambda x.\, s$ denotes the function f defined by

$$f(x) = s$$

for any x of type σ; so item (6) gives us a form of comprehension, enabling us to define functions using terms. Formulae are built up from identity predicate statements $s = t$ between terms of the same type, the usual propositional connectives, and higher-type quantification. One can then take the axioms of the system to be the basic equations governing the terms defined above, together with the usual rules of logic with quantifiers and identity predicate.

If one augments the finite type system with a type Ω of truth values, one has to include axioms which govern its use as well. In fact, if one is clever, one can get rid of complex formulae entirely, replacing them with terms of type Ω! The proof system can then be modified accordingly. The result is essentially the *simple theory of types* set forth by Alonzo Church in the 1930s.

As in the case of second-order logic, there are different versions of higher-type semantics that one might want to use. In the full version, variables of type $\sigma \to \tau$ range over the set of *all* functions from the objects of type σ to objects of type τ. As you might expect, this semantics is too strong to admit a complete, effective proof system. But one can consider a weaker semantics, in which a structure consists of sets of elements T_τ for each type τ, together with appropriate operations for application, projection, etc. If the details are carried out correctly, one can obtain completeness theorems for the kinds of proof systems described above.

Higher-type logic is attractive because it provides a framework in which we can embed a good deal of mathematics in a natural way: starting with $\mathbb{N}$, one can define real numbers, continuous functions, and so on. It is also particularly attractive in the context of intuitionistic logic, since the types have clear "constructive" intepretations. In fact, one can develop constructive versions of higher-type semantics (based on intuitionistic, rather than classical logic) that clarify these constructive interpretations quite nicely, and are, in many ways, more interesting than the classical counterparts.

9.5 Intuitionistic Logic

In constrast to second-order and higher-order logic, intuitionistic first-order logic represents a restriction of the classical version, intended to model a more "constructive" kind of reasoning. The following examples may serve to illustrate some of the underlying motivations.

Suppose someone came up to you one day and announced that they had determined a natural number x, with the property that if x is prime, the Riemann hypothesis is true, and if x is composite, the Riemann hypothesis is false. Great news! Whether the Riemann hypothesis is true or not is one of the big open questions of mathematics, and here they seem to have reduced the problem to one of calculation, that is, to the determination of whether a specific number is prime or not.

What is the magic value of x? They describe it as follows: x is the natural number that is equal to 7 if the Riemann hypothesis is true, and 9 otherwise.

Angrily, you demand your money back. From a classical point of view, the description above does in fact determine a unique value of x; but what you really want is a value of x that is given *explicitly*.

To take another, perhaps less contrived example, consider the following question. We know that it is possible to raise an irrational number to a rational power, and get a rational result. For example, $\sqrt{2}^2 = 2$. What is less clear is whether or not it is possible to raise an irrational number to an *irrational* power, and get a rational result. The following theorem answers this in the affirmative:

Theorem 9.1. *There are irrational numbers a and b such that a^b is rational.*

Proof. Consider $\sqrt{2}^{\sqrt{2}}$. If this is rational, we are done: we can let $a = b = \sqrt{2}$. Otherwise, it is irrational. Then we have

$$(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = \sqrt{2}^2 = 2,$$

which is certainly rational. So, in this case, let a be $\sqrt{2}^{\sqrt{2}}$, and let b be $\sqrt{2}$. □

Does this constitute a valid proof? Most mathematicians feel that it does. But again, there is something a little bit unsatisfying here: we have proved the existence of a pair of real numbers with a certain property, without being able to say *which* pair of numbers it is. It is possible to prove the same result, but in such a way that the pair a, b *is* given in the proof: take $a = \sqrt{3}$ and $b = \log_3 4$. Then

$$a^b = \sqrt{3}^{\log_3 4} = 3^{1/2\cdot\log_3 4} = (3^{\log_3 4})^{1/2} = 4^{1/2} = 2,$$

since $3^{\log_3 x} = x$.

Intuitionistic logic is designed to model a kind of reasoning where moves like the one in the first proof are disallowed. Proving the existence of an x satisfying $\varphi(x)$ means that you have to give a specific x, and a proof that it satisfies φ, like in the second proof. Proving that φ or ψ holds requires that you can prove one or the other.

Formally speaking, intuitionistic first-order logic is what you get if you omit restrict a proof system for first-order logic in a certain way. Similarly, there are intuitionistic versions of second-order or higher-order logic. From the mathematical point of view, these are just formal deductive systems, but, as already noted, they are intended to model a kind of mathematical reasoning. One can take this to be the kind of reasoning that is justified on a certain philosophical view of mathematics (such as Brouwer's intuitionism); one can take it to be a kind of mathematical reasoning which is more "concrete" and satisfying (along the lines of Bishop's constructivism); and one can argue about whether or not the formal description captures the informal motivation. But whatever philosophical positions we may hold, we can study intuitionistic logic as a formally presented logic; and for whatever reasons, many mathematical logicians find it interesting to do so.

There is an informal constructive interpretation of the intuitionist connectives, usually known as the Brouwer-Heyting-Kolmogorov interpretation. It runs as follows: a proof of $\varphi \,\&\, \psi$ consists of a proof of φ paired with a proof of ψ; a proof of $\varphi \vee \psi$ consists of either a proof of φ, or a proof of ψ, where we have explicit information as to which is the case; a proof of $\varphi \supset \psi$ consists of a procedure, which transforms a proof of φ to a proof of ψ; a proof of $\forall x\, \varphi(x)$ consists of a procedure which returns a proof of $\varphi(x)$ for any value of x; and a proof of $\exists x\, \varphi(x)$ consists of a value of x, together with a proof that this value satisfies φ. One can describe the interpretation in computational terms known as the "Curry-Howard isomorphism" or the "formulae-as-types

paradigm": think of a formula as specifying a certain kind of data type, and proofs as computational objects of these data types that enable us to see that the corresponding formula is true.

Intuitionistic logic is often thought of as being classical logic "minus" the law of the excluded middle. This following theorem makes this more precise.

Theorem 9.2. *Intuitionistically, the following axiom schemata are equivalent:*

1. $(\varphi \supset \bot) \supset \sim\varphi$.
2. $\varphi \vee \sim\varphi$
3. $\sim\sim\varphi \supset \varphi$

Obtaining instances of one schema from either of the others is a good exercise in intuitionistic logic.

The first deductive systems for intuitionistic propositional logic, put forth as formalizations of Brouwer's intuitionism, are due, independently, to Kolmogorov, Glivenko, and Heyting. The first formalization of intuitionistic first-order logic (and parts of intuitionist mathematics) is due to Heyting. Though a number of classically valid schemata are not intuitionistically valid, many are.

The *double-negation translation* describes an important relationship between classical and intuitionist logic. It is defined inductively follows (think of φ^N as the "intuitionist" translation of the classical formula φ):

$$\begin{aligned}
!A^N &\equiv \sim\sim\varphi \quad \text{for atomic formulae } \varphi \\
(\varphi \,\&\, \psi)^N &\equiv (\varphi^N \,\&\, \psi^N) \\
(\varphi \vee \psi)^N &\equiv \sim\sim(\varphi^N \vee \psi^N) \\
(\varphi \supset \psi)^N &\equiv (\varphi^N \supset \psi^N) \\
(\forall x\, \varphi)^N &\equiv \forall x\, \varphi^N \\
(\exists x\, \varphi)^N &\equiv \sim\sim\exists x\, \varphi^N
\end{aligned}$$

Kolmogorov and Glivenko had versions of this translation for propositional logic; for predicate logic, it is due to Gödel and Gentzen, independently. We have

Theorem 9.3. 1. *$\varphi \equiv \varphi^N$ is provable classically*

2. *If φ is provable classically, then φ^N is provable intuitionistically.*

We can now envision the following dialogue. Classical mathematician: "I've proved φ!" Intuitionist mathematician: "Your proof isn't valid. What you've really proved is φ^N." Classical mathematician: "Fine by me!" As far as

the classical mathematician is concerned, the intuitionist is just splitting hairs, since the two are equivalent. But the intuitionist insists there is a difference.

Note that the above translation concerns pure logic only; it does not address the question as to what the appropriate *nonlogical* axioms are for classical and intuitionistic mathematics, or what the relationship is between them. But the following slight extension of the theorem above provides some useful information:

Theorem 9.4. *If Γ proves φ classically, Γ^N proves φ^N intuitionistically.*

In other words, if φ is provable from some hypotheses classically, then φ^N is provable from their double-negation translations.

To show that a sentence or propositional formula is intuitionistically valid, all you have to do is provide a proof. But how can you show that it is not valid? For that purpose, we need a semantics that is sound, and preferrably complete. A semantics due to Kripke nicely fits the bill.

We can play the same game we did for classical logic: define the semantics, and prove soundness and completeness. It is worthwhile, however, to note the following distinction. In the case of classical logic, the semantics was the "obvious" one, in a sense implicit in the meaning of the connectives. Though one can provide some intuitive motivation for Kripke semantics, the latter does not offer the same feeling of inevitability. In addition, the notion of a classical structure is a natural mathematical one, so we can either take the notion of a structure to be a tool for studying classical first-order logic, or take classical first-order logic to be a tool for studying mathematical structures. In contrast, Kripke structures can only be viewed as a logical construct; they don't seem to have independent mathematical interest.

A Kripke structure for a propositional langauge consists of a partial order $\mathrm{Mod}(P)$ with a least element, and an "monotone" assignment of propositional variables to the elements of $\mathrm{Mod}(P)$. The intuition is that the elements of $\mathrm{Mod}(P)$ represent "worlds," or "states of knowledge"; an element $p \geq q$ represents a "possible future state" of q; and the propositional variables assigned to p are the propositions that are known to be true in state p. The forcing relation $\mathfrak{P}, p \Vdash \varphi$ then extends this relationship to arbitrary formulae in the language; read $\mathfrak{P}, p \Vdash \varphi$ as "φ is true in state p." The relationship is defined inductively, as follows:

1. $\mathfrak{P}, p \Vdash p_i$ iff p_i is one of the propositional variables assigned to p.
2. $\mathfrak{P}, p \nVdash \bot$.
3. $\mathfrak{P}, p \Vdash (\varphi \,\&\, \psi)$ iff $\mathfrak{P}, p \Vdash \varphi$ and $\mathfrak{P}, p \Vdash \psi$.
4. $\mathfrak{P}, p \Vdash (\varphi \vee \psi)$ iff $\mathfrak{P}, p \Vdash \varphi$ or $\mathfrak{P}, p \Vdash \psi$.
5. $\mathfrak{P}, p \Vdash (\varphi \supset \psi)$ iff, whenever $q \geq p$ and $\mathfrak{P}, q \Vdash \varphi$, then $\mathfrak{P}, q \Vdash \psi$.

It is a good exercise to try to show that $\sim(p \,\&\, q) \supset (\sim p \vee \sim q)$ is not intuitionistically valid, by cooking up a Kripke structure that provides a counterexample.

9.6 Modal Logics

Consider the following example of a conditional sentence:

> If Jeremy is alone in that room, then he is drunk and naked and dancing on the chairs.

This is an example of a conditional assertion that may be materially true but nonetheless misleading, since it seems to suggest that there is a stronger link between the antecedent and conclusion other than simply that either the antecedent is false or the consequent true. That is, the wording suggests that the claim is not only true in this particular world (where it may be trivially true, because Jeremy is not alone in the room), but that, moreover, the conclusion *would have* been true *had* the antecedent been true. In other words, one can take the assertion to mean that the claim is true not just in this world, but in any "possible" world; or that it is *necessarily* true, as opposed to just true in this particular world.

Modal logic was designed to make sense of this kind of necessity. One obtains modal propositional logic from ordinary propositional logic by adding a box operator; which is to say, if φ is a formula, so is $\Box\varphi$. Intuitively, $\Box\varphi$ asserts that φ is *necessarily* true, or true in any possible world. $\Diamond\varphi$ is usually taken to be an abbreviation for $\sim\Box\sim\varphi$, and can be read as asserting that φ is *possibly* true. Of course, modality can be added to predicate logic as well.

Kripke structures can be used to provide a semantics for modal logic; in fact, Kripke first designed this semantics with modal logic in mind. Rather than restricting to partial orders, more generally one has a set of "possible worlds," P, and a binary "accessibility" relation $R(x, y)$ between worlds. Intuitively, $R(p, q)$ asserts that the world q is compatible with p; i.e., if we are "in" world p, we have to entertain the possibility that the world could have been like q.

Modal logic is sometimes called an "intensional" logic, as opposed to an "extensional" one. The intended semantics for an extensional logic, like classical logic, will only refer to a single world, the "actual" one; while the semantics for an "intensional" logic relies on a more elaborate ontology. In addition to structureing necessity, one can use modality to structure other linguistic constructions, reinterpreting $\Box$ and $\Diamond$ according to the application. For example:

1. In provability logic, $\Box\varphi$ is read "φ is provable" and $\Diamond\varphi$ is read "φ is consistent."

2. In epistemic logic, one might read $\Box\varphi$ as "I know φ" or "I believe φ."

3. In temporal logic, one can read $\Box\varphi$ as "φ is always true" and $\Diamond\varphi$ as "φ is sometimes true."

One would like to augment logic with rules and axioms dealing with modality. For example, the system **S4** consists of the ordinary axioms and rules of propositional logic, together with the following axioms:

$$\Box(\varphi \supset \psi) \supset (\Box\varphi \supset \Box\psi)$$
$$\Box\varphi \supset \varphi$$
$$\Box\varphi \supset \Box\Box\varphi$$

as well as a rule, "from φ conclude $\Box\varphi$." **S5** adds the following axiom:

$$\Diamond\varphi \supset \Box\Diamond\varphi$$

Variations of these axioms may be suitable for different applications; for example, S5 is usually taken to characterize the notion of logical necessity. And the nice thing is that one can usually find a semantics for which the proof system is sound and complete by restricting the accessibility relation in the Kripke structures in natural ways. For example, **S4** corresponds to the class of Kripke structures in which the accessibility relation is reflexive and transitive. **S5** corresponds to the class of Kripke structures in which the accessibility relation is *universal*, which is to say that every world is accessible from every other; so $\Box\varphi$ holds if and only if φ holds in every world.

9.7 Other Logics

As you may have gathered by now, it is not hard to design a new logic. You too can create your own a syntax, make up a deductive system, and fashion a semantics to go with it. You might have to be a bit clever if you want the proof system to be complete for the semantics, and it might take some effort to convince the world at large that your logic is truly interesting. But, in return, you can enjoy hours of good, clean fun, exploring your logic's mathematical and computational properties.

Recent decades have witnessed a veritable explosion of formal logics. Fuzzy logic is designed to model reasoning about vague properties. Probabilistic logic is designed to model reasoning about uncertainty. Default logics and nonmonotonic logics are designed to model defeasible forms of reasoning, which is to say, "reasonable" inferences that can later be overturned in the face of new information. There are epistemic logics, designed to model reasoning about knowledge; causal logics, designed to model reasoning about causal relationships; and even "deontic" logics, which are designed to model reasoning about moral and ethical obligations. Depending on whether the primary

motivation for introducing these systems is philosophical, mathematical, or computational, you may find such creatures studies under the rubric of mathematical logic, philosophical logic, artificial intelligence, cognitive science, or elsewhere.

The list goes on and on, and the possibilities seem endless. We may never attain Leibniz' dream of reducing all of human reason to calculation—but that can't stop us from trying.

Part III

Turing Machines

Chapter 10

Turing Machine Computations

10.1 Introduction

What does it mean for a function, say, from $\mathbb{N}$ to $\mathbb{N}$ to be *computable*? Among the first answers, and the most well known one, is that a function is computable if it can be computed by a Turing machine. This notion was set out by Alan Turing in 1936. Turing machines are an example of *a model of computation*—they are a mathematically precise way of defining the idea of a "computational procedure." What exactly that means is debated, but it is widely agreed that Turing machines are one way of specifying computational procedures. Even though the term "Turing machine" evokes the image of a physical machine with moving parts, strictly speaking a Turing machine is a purely mathematical construct, and as such it idealizes the idea of a computational procedure. For instance, we place no restriction on either the time or memory requirements of a Turing machine: Turing machines can compute something even if the computation would require more storage space or more steps than there are atoms in the universe.

It is perhaps best to think of a Turing machine as a program for a special kind of imaginary mechanism. This mechanism consists of a *tape* and a *read-write head*. In our version of Turing machines, the tape is infinite in one direction (to the right), and it is divided into *squares*, each of which may contain a symbol from a finite *alphabet*. Such alphabets can contain any number of different symbols, say, but we will mainly make do with three: $\triangleright$, 0, and 1. When the mechanism is started, the tape is empty (i.e., each square contains the symbol 0) except for the leftmost square, which contains $\triangleright$, and a finite number of squares which contain the *input*. At any time, the mechanism is in one of a finite number of *states*. At the outset, the head scans the leftmost square and in a specified *initial state*. At each step of the mechanism's run, the content of the square currently scanned together with the state the mechanism is in and the Turing machine program determine what happens next. The Turing machine program is given by a partial function which takes as input a state q

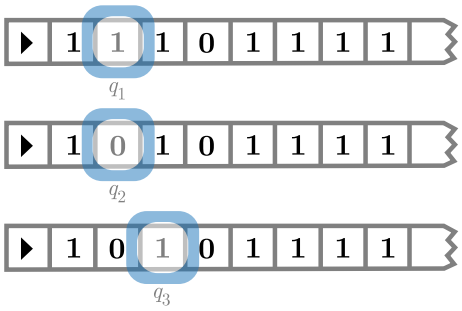


Figure 10.1: A Turing machine executing its program.

and a symbol σ and outputs a triple $\langle q', \sigma', D \rangle$. Whenever the mechanism is in state q and reads symbol σ, it replaces the symbol on the current square with σ', the head moves left, right, or stays put according to whether D is L, R, or N, and the mechanism goes into state q'.

For instance, consider the situation in section 10.1. The visible part of the tape of the Turing machine contains the end-of-tape symbol $\triangleright$ on the leftmost square, followed by three 1's, a 0, and four more 1's. The head is reading the third square from the left, which contains a 1, and is in state q_1—we say "the machine is reading a 1 in state q_1." If the program of the Turing machine returns, for input $\langle q_1, 1 \rangle$, the triple $\langle q_2, 0, N \rangle$, the machine would now replace the 1 on the third square with a 0, leave the read/write head where it is, and switch to state q_2. If then the program returns $\langle q_3, 0, R \rangle$ for input $\langle q_2, 0 \rangle$, the machine would now overwrite the 0 with another 0 (effectively, leaving the content of the tape under the read/write head unchanged), move one square to the right, and enter state q_3. And so on.

We say that the machine *halts* when it encounters some state, q_n, and symbol, σ such that there is no instruction for $\langle q_n, \sigma \rangle$, i.e., the transition function for input $\langle q_n, \sigma \rangle$ is undefined. In other words, the machine has no instruction to carry out, and at that point, it ceases operation. Halting is sometimes represented by a specific halt state h. This will be demonstrated in more detail later on.

The beauty of Turing's paper, "On computable numbers," is that he presents not only a formal definition, but also an argument that the definition captures the intuitive notion of computability. From the definition, it should be clear that any function computable by a Turing machine is computable in the intuitive sense. Turing offers three types of argument that the converse is true, i.e., that any function that we would naturally regard as computable is computable by such a machine. They are (in Turing's words):

1. A direct appeal to intuition.

2. A proof of the equivalence of two definitions (in case the new definition has a greater intuitive appeal).

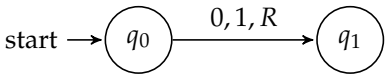
3. Giving examples of large classes of numbers which are computable.

Our goal is to try to define the notion of computability "in principle," i.e., without taking into account practical limitations of time and space. Of course, with the broadest definition of computability in place, one can then go on to consider computation with bounded resources; this forms the heart of the subject known as "computational complexity."

Historical Remarks Alan Turing invented Turing machines in 1936. While his interest at the time was the decidability of first-order logic, the paper has been described as a definitive paper on the foundations of computer design. In the paper, Turing focuses on computable real numbers, i.e., real numbers whose decimal expansions are computable; but he notes that it is not hard to adapt his notions to computable functions on the natural numbers, and so on. Notice that this was a full five years before the first working general purpose computer was built in 1941 (by the German Konrad Zuse in his parent's living room), seven years before Turing and his colleagues at Bletchley Park built the code-breaking Colossus (1943), nine years before the American ENIAC (1945), twelve years before the first British general purpose computer—the Manchester Small-Scale Experimental Machine—was built in Manchester (1948), and thirteen years before the Americans first tested the BINAC (1949). The Manchester SSEM has the distinction of being the first stored-program computer—previous machines had to be rewired by hand for each new task.

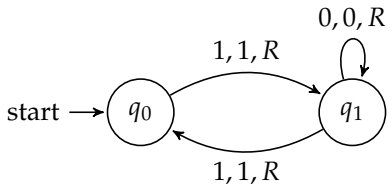
10.2 Representing Turing Machines

Turing machines can be represented visually by *state diagrams*. The diagrams are composed of state cells connected by arrows. Unsurprisingly, each state cell represents a state of the machine. Each arrow represents an instruction that can be carried out from that state, with the specifics of the instruction written above or below the appropriate arrow. Consider the following machine, which has only two internal states, q_0 and q_1, and one instruction:



Recall that the Turing machine has a read/write head and a tape with the input written on it. The instruction can be read as *if reading a blank in state q_0, write a stroke, move right, and move to state q_1*. This is equivalent to the transition function mapping $\langle q_0, 0\rangle$ to $\langle q_1, 1, R\rangle$.

Example 10.1. *Even Machine*: The following Turing machine halts if, and only if, there are an even number of strokes on the tape.



The state diagram corresponds to the following transition function:

$$\delta(q_0, 1) = \langle q_1, 1, R\rangle,$$
$$\delta(q_1, 1) = \langle q_0, 1, R\rangle,$$
$$\delta(q_1, 0) = \langle q_1, 0, R\rangle$$

The above machine halts only when the input is an even number of strokes. Otherwise, the machine (theoretically) continues to operate indefinitely. For any machine and input, it is possible to trace through the *configurations* of the machine in order to determine the output. We will give a formal definition of configurations later. For now, we can intuitively think of configurations as a series of diagrams showing the state of the machine at any point in time during operation. Configurations show the content of the tape, the state of the machine and the location of the read/write head.

Let us trace through the configurations of the even machine if it is started with an input of 4 1s. In this case, we expect that the machine will halt. We will then run the machine on an input of 3 1s, where the machine will run forever.

The machine starts in state q_0, scanning the leftmost 1. We can represent the initial state of the machine as follows:

$$\triangleright 1_0 1110\ldots$$

The above configuration is straightforward. As can be seen, the machine starts in state one, scanning the leftmost 1. This is represented by a subscript of the state name on the first 1. The applicable instruction at this point is $\delta(q_0, 1) = \langle q_1, 1, R\rangle$, and so the machine moves right on the tape and changes to state q_1.

$$\triangleright 11_1 110\ldots$$

Since the machine is now in state q_1 scanning a stroke, we have to "follow" the instruction $\delta(q_1, 1) = \langle q_0, 1, R\rangle$. This results in the configuration

$$\triangleright 111_0 10\ldots$$

As the machine continues, the rules are applied again in the same order, resulting in the following two configurations:

$$\triangleright 1111_1 0 \ldots$$

$$\triangleright 11110_0 \ldots$$

The machine is now in state q_0 scanning a blank. Based on the transition diagram, we can easily see that there is no instruction to be carried out, and thus the machine has halted. This means that the input has been accepted.

Suppose next we start the machine with an input of three strokes. The first few configurations are similar, as the same instructions are carried out, with only a small difference of the tape input:

$$\triangleright 1_0 110 \ldots$$

$$\triangleright 11_1 10 \ldots$$

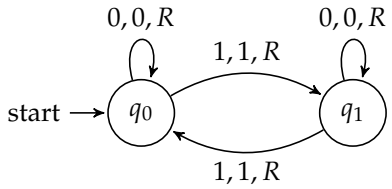
$$\triangleright 111_0 0 \ldots$$

$$\triangleright 1110_1 \ldots$$

The machine has now traversed past all the strokes, and is reading a blank in state q_1. As shown in the diagram, there is an instruction of the form $\delta(q_1, 0) = \langle q_1, 0, R \rangle$. Since the tape is infinitely blank to the right, the machine will continue to execute this instruction *forever*, staying in state q_1 and moving ever further to the right. The machine will never halt, and does not accept the input.

It is important to note that not all machines will halt. If halting means that the machine runs out of instructions to execute, then we can create a machine that never halts simply by ensuring that there is an outgoing arrow for each symbol at each state. The even machine can be modified to run infinitely by adding an instruction for scanning a blank at q_0.

Example 10.2.



Machine tables are another way of representing Turing machines. Machine tables have the tape alphabet displayed on the x-axis, and the set of machine states across the y-axis. Inside the table, at the intersection of each state and symbol, is written the rest of the instruction—the new state, new symbol, and direction of movement. Machine tables make it easy to determine in what

state, and for what symbol, the machine halts. Whenever there is a gap in the table is a possible point for the machine to halt. Unlike state diagrams and instruction sets, where the points at which the machine halts are not always immediately obvious, any halting points are quickly identified by finding the gaps in the machine table.

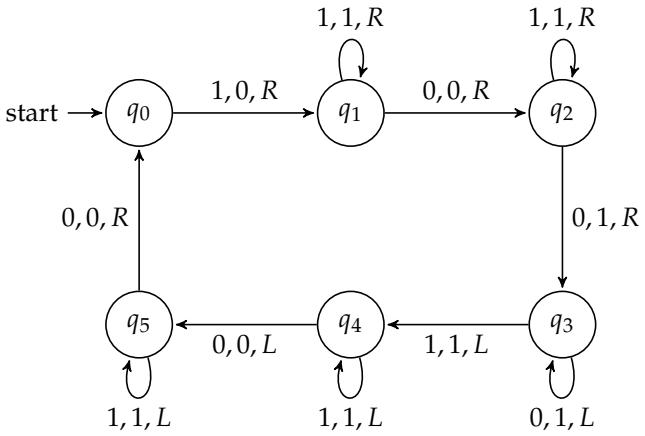
Example 10.3. The machine table for the even machine is:

	0	1
q_0		$1, q_1, R$
q_1	$0, q_1, 0$	$1, q_0, R$

As we can see, the machine halts when scanning a blank in state q_0.

So far we have only considered machines that read and accept input. However, Turing machines have the capacity to both read and write. An example of such a machine (although there are many, many examples) is a *doubler*. A doubler, when started with a block of n strokes on the tape, outputs a block of $2n$ strokes.

Example 10.4. Before building a doubler machine, it is important to come up with a *strategy* for solving the problem. Since the machine (as we have formulated it) cannot remember how many strokes it has read, we need to come up with a way to keep track of all the strokes on the tape. One such way is to separate the output from the input with a blank. The machine can then erase the first stroke from the input, traverse over the rest of the input, leave a blank, and write two new strokes. The machine will then go back and find the second stroke in the input, and double that one as well. For each one stroke of input, it will write two strokes of output. By erasing the input as the machine goes, we can guarantee that no stroke is missed or doubled twice. When the entire input is erased, there will be $2n$ strokes left on the tape.



10.3 Turing Machines

The formal definition of what constitutes a Turing machine looks abstract, but is actually simple: it merely packs into one mathematical structure all the information needed to specify the workings of a Turing machine. This includes (1) which states the machine can be in, (2) which symbols are allowed to be on the tape, (3) which state the machine should start in, and (4) what the instruction set of the machine is.

Definition 10.5 (Turing machine)**.** A *Turing machine* $T = \langle Q, \Sigma, q_0, \delta \rangle$ consists of

1. a finite set of *states* Q,
2. a finite *alphabet* Σ which includes $\triangleright$ and 0,
3. an *initial state* $q_0 \in Q$,
4. a finite *instruction set* $\delta \colon Q \times \Sigma \nrightarrow Q \times \Sigma \times \{L, R, N\}$.

The partial function δ is also called the *transition function* of T.

We assume that the tape is infinite in one direction only. For this reason it is useful to designate a special symbol $\triangleright$ as a marker for the left end of the tape. This makes it easier for Turing machine programs to tell when they're "in danger" of running off the tape.

Example 10.6. *Even Machine:* The even machine is formally the quadruple $\langle Q, \Sigma, q_0, \delta \rangle$ where

$$\begin{aligned} Q &= \{q_0, q_1\} \\ \Sigma &= \{\triangleright, 0, 1\}, \\ \delta(q_0, 1) &= \langle q_1, 1, R \rangle, \\ \delta(q_1, 1) &= \langle q_0, 1, R \rangle, \\ \delta(q_1, 0) &= \langle q_1, 0, R \rangle. \end{aligned}$$

10.4 Configurations and Computations

Recall tracing through the configurations of the even machine earlier. The imaginary mechanism consisting of tape, read/write head, and Turing machine program is really just in intuitive way of visualizing what a Turing machine computation is. Formally, we can define the computation of a Turing machine on a given input as a sequence of *configurations*—and a configuration in turn is a sequence of symbols (corresponding to the contents of the tape at a given point in the computation), a number indicating the position of the read/write head, and a state. Using these, we can define what the Turing machine M computes on a given input.

Definition 10.7 (Configuration)**.** A *configuration* of Turing machine $M = \langle Q, \Sigma, q_0, \delta \rangle$ is a triple $\langle C, n, q \rangle$ where

1. $C \in \Sigma^*$ is a finite sequence of symbols from Σ,
2. $n \in \mathbb{N}$ is a number $< \text{len}(C)$, and
3. $q \in Q$

Intuitively, the sequence C is the content of the tape (symbols of all squares from the leftmost square to the last non-blank or previously visited square), n is the number of the square the read/write head is scanning (beginning with 0 being the number of the leftmost square), and q is the current state of the machine.

The potential input for a Turing machine is a sequence of symbols, usually a sequence that encodes a number in some form. The initial configuration of the Turing machine is that configuration in which we start the Turing machine to work on that input: the tape contains the tape end marker immediately followed by the input written on the squares to the right, the read/write head is scanning the leftmost square of the input (i.e., the square to the right of the left end marker), and the mechanism is in the designated start state q_0.

Definition 10.8 (Initial configuration)**.** The *initial configuration* of M for input $I \in \Sigma^*$ is

$$\langle \triangleright \frown I, 1, q_0 \rangle$$

The $\frown$ symbol is for *concatenation*—we want to ensure that there are no blanks between the left end marker and the beginning of the input.

Definition 10.9. We say that a configuration $\langle C, n, q \rangle$ *yields* $\langle C', n', q' \rangle$ *in one step* (according to M), iff

1. the n-th symbol of C is σ,
2. the instruction set of M specifies $\delta(q, \sigma) = \langle q', \sigma', D \rangle$,
3. the n-th symbol of C' is σ', and
4. a) $D = L$ and $n' = n - 1$ if $n > 0$, otherwise $n' = 0$, or
   b) $D = R$ and $n' = n + 1$, or
   c) $D = N$ and $n' = n$,
5. if $n' > \text{len}(C)$, then $\text{len}(C') = \text{len}(C) + 1$ and the n'-th symbol of C' is 0.
6. for all i such that $i < \text{len}(C')$ and $i \neq n$, $C'(i) = C(i)$,

Definition 10.10. A *run of M on input I* is a sequence C_i of configurations of M, where C_0 is the initial configuration of M for input I, and each C_i yields C_{i+1} in one step.

We say that *M halts on input I after k steps* if $C_k = \langle C, n, q\rangle$, the nth symbol of C is σ, and $\delta(q, \sigma)$ is undefined. In that case, the *output* of M for input I is O, where O is a string of symbols not beginning or ending in 0 such that $C = \triangleright \frown 0^i \frown O \frown 0^j$ for some $i, j \in \mathbb{N}$.

According to this definition, the output O of M always begins and ends in a symbol other than 0, or, if at time k the entire tape is filled with 0 (except for the leftmost $\triangleright$), O is the empty string.

10.5 Unary Representation of Numbers

Turing machines work on sequences of symbols written on their tape. Depending on the alphabet a Turing machine uses, these sequences of symbols can represent various inputs and outputs. Of particular interest, of course, are Turing machines which compute *arithmetical* functions, i.e., functions of natural numbers. A simple way to represent positive integers is by coding them as sequences of a single symbol 1. If $n \in \mathbb{N}$, let 1^n be the empty sequence if $n = 0$, and otherwise the sequence consisting of exactly n 1's.

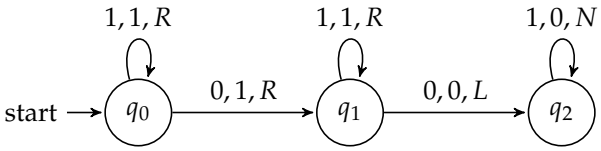
Definition 10.11 (Computation)**.** A Turing machine M *computes* the function $f\colon \mathbb{N}^n \to \mathbb{N}$ iff M halts on input

$$1^{k_1}01^{k_2}0\ldots01^{k_n}$$

with output $1^{f(k_1,\ldots,k_n)}$.

Example 10.12. *Addition:* Build a machine that, when given an input of two non-empty strings of 1's of length n and m, computes the function $f(n, m) = n + m$.

We want to come up with a machine that starts with two blocks of strokes on the tape and halts with one block of strokes. We first need a method to carry out. The input strokes are separated by a blank, so one method would be to write a stroke on the square containing the blank, and erase the first (or last) stroke. This would result in a block of $n + m$ 1's. Alternatively, we could proceed in a similar way to the doubler machine, by erasing a stroke from the first block, and adding one to the second block of strokes until the first block has been removed completely. We will proceed with the former example.

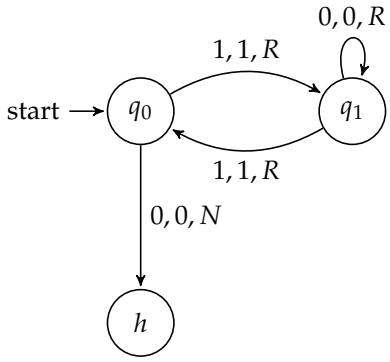


10.6 Halting States

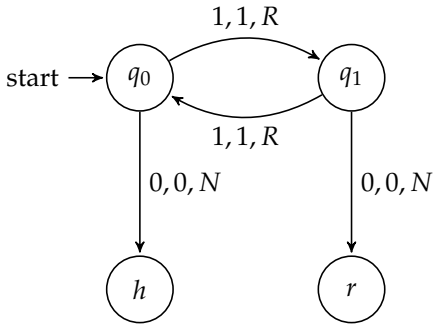
Although we have defined our machines to halt only when there is no instruction to carry out, common representations of Turing machines have a dedicated *halting state*, h, such that $h \in Q$.

The idea behind a halting state is simple: when the machine has finished operation (it is ready to accept input, or has finished writing the output), it goes into a state h where it halts. Some machines have two halting states, one that accepts input and one that rejects input.

Example 10.13. *Halting States*. To elucidate this concept, let us begin with an alteration of the even machine. Instead of having the machine halt in state q_0 if the input is even, we can add an instruction to send the machine into a halt state.



Let us further expand the example. When the machine determines that the input is odd, it never halts. We can alter the machine to include a *reject* state by replacing the looping instruction with an instruction to go to a reject state r.



Adding a dedicated halting state can be advantageous in cases like this, where it makes explicit when the machine accepts/rejects certain inputs. However, it is important to note that no computing power is gained by adding a dedicated halting state. Similarly, a less formal notion of halting has its own

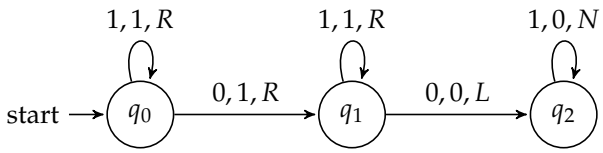
advantages. The definition of halting used so far in this chapter makes the proof of the *Halting Problem* intuitive and easy to demonstrate. For this reason, we continue with our original definition.

10.7 Combining Turing Machines

The examples of Turing machines we have seen so far have been fairly simple in nature. But in fact, any problem that can be solved with any modern programming language can als o be solved with Turing machines. To build more complex Turing machines, it is important to convince ourselves that we can combine them, so we can build machines to solve more complex problems by breaking the procedure into simpler parts. If we can find a natural way to break a complex problem down into constituent parts, we can tackle the problem in several stages, creating several simple Turing machines and combining then into one machine that can solve the problem. This point is especially important when tackling the Halting Problem in the next section.

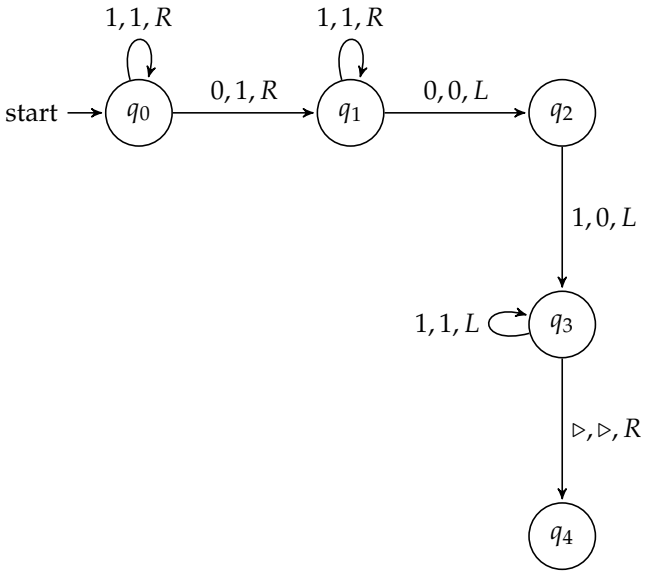
Example 10.14. *Combining Machines:* Design a machine that computes the function $f(m, n) = 2(m + n)$.

In order to build this machine, we can combine two machines we are already familiar with: the addition machine, and the doubler. We begin by drawing a state diagram for the addition machine.



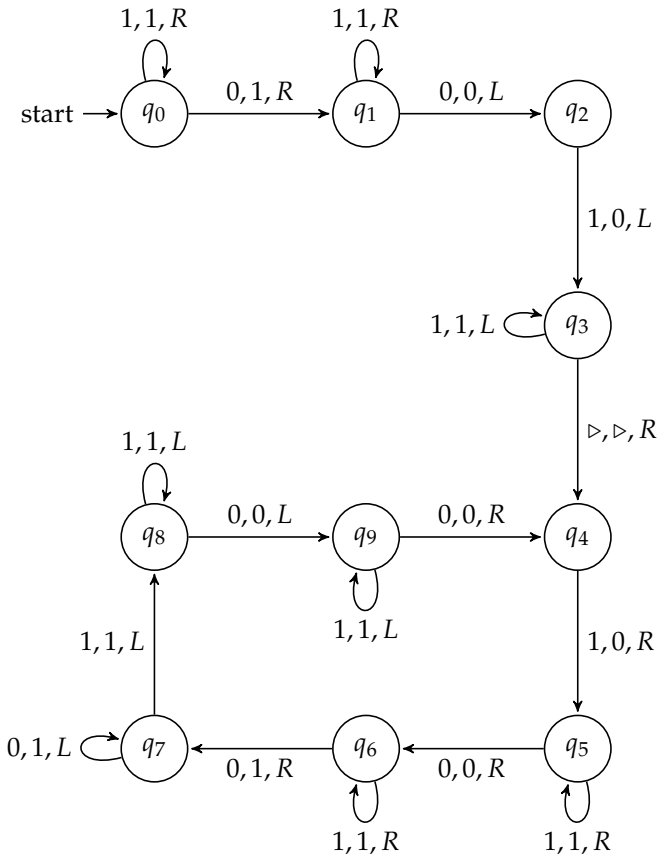
Instead of halting at state q_2, we want to continue operation in order to double the output. Recall that the doubler machine erases the first stroke in the input and writes two strokes in a separate output. Let's add an instruction to make sure the tape head is reading the first stroke of the output of the addition

machine.



It is now easy to double the input—all we have to do is connect the doubler machine onto state q_4. This requires renaming the states of the doubler machine so that they start at q_4 instead of q_0—this way we don't end up with two

starting states. The final diagram should look like:



10.8 Variants of Turing Machines

There are in fact many possible ways to define Turing machines, of which ours is only one. In some ways, our definition is more liberal than others. We allow arbitrary finite alphabets, a more restricted definition might allow only two tape symbols, 1 and 0. We allow the machine to write a symbol to the tape and move at the same time, other definitions allow either writing or moving. We allow the possibility of writing without moving the tape head, other definitions leave out the N "instruction." In other ways, our definition is more restrictive. We assumed that the tape is infinite in one direction only, other definitions allow the tape to be infinite both to the left and the right. In fact, one can even even allow any number of separate tapes, or even an infinite grid of squares. We represent the instruction set of the Turing machine by a transition function; other definitions use a transition relation where the machine has more than one possible instruction in any given situation.

This last relaxation of the definition is particularly interesting. In our definition, when the machine is in state q reading symbol σ, $\delta(q, \sigma)$ determines what the new symbol, state, and tape head position is. But if we allow the instruction set to be a relation between current state-symbol pairs $\langle q, \sigma \rangle$ and new state-symbol-direction triples $\langle q', \sigma', D \rangle$, the action of the Turing machine may not be uniquely determined—the instruction relation may contain both $\langle q, \sigma, q', \sigma', D \rangle$ and $\langle q, \sigma, q'', \sigma'', D' \rangle$. In this case we have a *non-deterministic* Turing machine. These play an important role in computational complexity theory.

There are also different conventions for when a Turing machine halts: we say it halts when the transition function is undefined, other definitions require the machine to be in a special designated halting state. Since the tapes of our turing machines are infinite in one direction only, there ae cases where a Turing machine can't properly carry out an instruction: if it reads the leftmost square and is supposed to move left. According to our definition, it just stays put instead, but we could have defined it so that it halts when that happens. There are also different ways of representing numbers (and hence the input-output function computed by a Turing machine): we use unary representation, but you can also use binary representation (this requires two symbols in addition to 0).

Now here is an interesting fact: none of these variations matters as to which functions are Turing computable. *If a function is Turing computable according to one definition, it is Turing computable according to all of them.*

10.9 The Church-Turing Thesis

Turing machines are supposed to be a precise replacement for the concept of an effective procedure. Turing took it that anyone who grasped the concept of an effective procedure and the concept of a Turing machine would have the intuition that anything that could be done via an effective procedure could be done by Turing machine. This claim is given support by the fact that all the other proposed precise replacements for the concept of an effective procedure turn out to be extensionally equivalent to the concept of a Turing machine—that is, they can compute exactly the same set of functions. This claim is called the *Church-Turing thesis*.

Definition 10.15 (Church-Turing thesis)**.** The *Church-Turing Thesis* states that anything computable via an effective procedure is Turing computable.

The Church-Turing thesis is appealed to in two ways. The first kind of use of the Church-Turing thesis is an excuse for laziness. Suppose we have a description of an effective procedure to compute something, say, in "pseudo-code." Then we can invoke the Church-Turing thesis to justify the claim that

the same function is computed by some Turing machine, eve if we have not in fact constructed it.

The other use of the Church-Turing thesis is more philosophically interesting. It can be shown that there are functions whch cannot be computed by a Turing machines. From this, using the Church-Turing thesis, one can conclude that it cannot be effectively computed, using any procedure whatsoever. For if there were such a procedure, by the Church-Turing thesis, it would follow that there would be a Turing machine. So if we can prove that there is no Turing machine that computes it, there also can't be an effective procedure. In particular, the Church-Turing thesis is invoked to claim that the so-called halting problem not only cannot be solved by Turing machines, it cannot be effectively solved at all.

Chapter 11

Undecidability

11.1 Introduction

It might seem obvious that not every function, even every arithmetical function, can be computable. There are just too many, whose behavior is too complicated. Functions defined from the decay of radioactive particles, for instance, or other chaotic or random behavior. Suppose we start counting 1-second intervals from a given time, and define the function $f(n)$ as the number of particles in the universe that decay in the n-th 1-second interval after that initial moment. This seems like a candidate for a function we cannot ever hope to compute.

But it is one thing to not be able to imagine how one would compute such functions, and quite another to actually prove that they are uncomputable. In fact, even functions that seem hopelessly complicated may, in an abstract sense, be computable. For instance, suppose the universe is finite in time—some day, in the very distant future the universe will contract into a single point, as some cosmological theories predict. Then there is only a finite (but incredibly large) number of seconds from that initial moment for which $f(n)$ is defined. And any function which is defined for only finitely many inputs is computable: we could list the outputs in one big table, or code it in one very big Turing machine state transition diagram.

We are often interested in special cases of functions whose values give the answers to yes/no questions. For instance, the question "is n a prime number?" is associated with the function

$$\text{isprime}(n) = \begin{cases} 1 & \text{if } n \text{ is prime} \\ 0 & \text{otherwise.} \end{cases}$$

We say that a yes/no question can be *effectively decided*, if the associated 1/0-valued function is effectively computable.

To prove mathematically that there are functions which cannot be effectively computed, or problems that cannot effectively decided, it is essential to

fix a specific model of computation, and show about it that there are functions it cannot compute or problems it cannot decide. We can show, for instance, that not every function can be computed by Turing machines, and not every problem can be decided by Turing machines. We can then appeal to the Church-Turing thesis to conclude that not only are Turing machines not powerful enough to compute every function, but no effective procedure can.

The key to proving such negative results is the fact that we can assign numbers to Turing machines themselves. The easiest way to do this is to enumerate them, perhaps by fixing a specific way to write down Turing machines and their programs, and then listing them in a systematic fashion. Once we see that this can be done, then the existence of Turing-uncomputable functions follows by simple cardinality considerations: the set of functions from $\mathbb{N}$ to $\mathbb{N}$ (in fact, even just from $\mathbb{N}$ to $\{0, 1\}$) are uncountable, but since we can enumerate all the Turing machines, the set of Turing-computable functions is only countably infinite.

We can also define *specific* functions and problems which we can prove to be uncomputable and undecidable, respectively. One such problem is the so-called *Halting Problem*. Turing machines can be finitely described by listing their instructions. Such a description of a Turing machine, i.e., a Turing machine program, can of course be used as input to another Turing machine. So we can consider Turing machines that decide questions about other Turing machines. One particularly interesting question is this: "Does the given Turing machine eventually halt when started on input n?" It would be nice if there were a Turing machine that could decide this question: think of it as a quality-control Turing machine which ensures that Turing machines don't get caught in infinite loops and such. The interestign fact, which Turing proved, is that there cannot be such a Turing machine. There cannot be a single Turing machine which, when started on input consisting of a description of a Turing machine M and some number n, will always halt with either output 1 or 0 according to whether M machine would have halted when started on input n or not.

Once we have examples of specific undecidable problems we can use them to show that other problems are undecidable, too. For instance, one celebrated undecidable problem is the question, "Is the first-order formula φ valid?". There is no Turing machine which, given as input a first-order formula φ, is guaranteed to halt with output 1 or 0 according to whether φ is valid or not. Historically, the question of finding a procedure to effectively solve this problem was called simply "the" decision problem; and so we say that the decision problem is unsolvable. Turing and Church proved this result independently at around the same time, so it is also called the Church-Turing Theorem.

11.2 Enumerating Turing Machines

We can show that the set of all Turing-machines is countable. This follows from the fact that each Turing machine can be finitely described. The set of states and the tape vocabulary are finite sets. The transition function is a partial function from $Q \times \Sigma$ to $Q \times \Sigma \times \{L, R, N\}$, and so likewise can be specified by listing its values for the finitely many argument pairs for which it is defined. Of course, strictly speaking, the states and vocabulary can be anything; but the *behavior* of the Turing machine is independent of which objects serve as states and vocabulary. So we may assume, for instance, that the states and vocabulary symbols are natural numbers, or that the states and vocabulary are all strings of letters and digits.

Suppose we fix a countably infinite vocabulary for specifying Turing machines: $\sigma_0 = \triangleright$, $\sigma_1 = 0$, $\sigma_2 = 1$, σ_3, ..., R, L, N, q_0, q_1, ... Then any Turing machine can be specified by some finite string of symbols from this alphabet (though not every finite string of symbols specifies a Turing machine). For instance, suppose we have a Turing machine $M = \langle Q, \Sigma, q, \delta \rangle$ where

$$Q = \{q'_0, \ldots, q'_n\} \subseteq \{q_0, q_1, \ldots\} \text{ and}$$
$$\Sigma = \{\triangleright, \sigma'_1, \sigma'_2, \ldots, \sigma'_m\} \subseteq \{\sigma_0, \sigma_1, \ldots\}.$$

We could specify it by the string

$$q'_0 q'_1 \ldots q'_n \triangleright \sigma'_1 \ldots \sigma'_m \triangleright q \triangleright S(\sigma'_0, q'_0) \triangleright \ldots \triangleright S(\sigma'_m, q'_n)$$

where $S(\sigma'_i, q'_j)$ is the string $\sigma'_i q'_j \delta(\sigma'_i, q'_j)$ if $\delta(\sigma'_i, q'_j)$ is defined, and $\sigma'_i q'_j$ otherwise.

Theorem 11.1. *There are functions from $\mathbb{N}$ to $\mathbb{N}$ which are not Turing computable.*

Proof. We know that the set of finite strings of symbols from a countably infinite alphabet is countable. This gives us that the set of descriptions of Turing machines, as a subset of the finite strings from the countable vocabulary $\{q_0, q_1, \ldots, \triangleright, \sigma_1, \sigma_2, \ldots\}$, is itself enumerable. Since every Turing computable function is computed by some (in fact, many) Turing machines, this means that the set of all Turing computable functions from $\mathbb{N}$ to $\mathbb{N}$ is also enumerable.

On the other hand, the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not countable. This follows immediately from the fact that not even the set of all functions of one argument from $\mathbb{N}$ to the set $\{0, 1\}$ is countable. If all functions were computable by some Turing machine we could enumerate the set of all functions. So there are some functions that are not Turing-computable. □

11.3 The Halting Problem

Assume we have fixed some finite descriptions of Turing machines. Using these, we can enumerate Turing machines via their descriptions, say, ordered

by the lexicographic ordering. Each Turing machine thus receives an *index*: its place in the enumeration M_1, M_2, M_3, ... of Turing machine descriptions.

We know that there must be non-Turing-computable functions: the set of Turing machine descriptions—and hence the set of Turing machines—is enumerable, but the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not. But we can find specific examples of non-computable function as well. One such function is the halting function.

Definition 11.2 (Halting function)**.** The *halting function* h is defined as

$$h(e,n) = \begin{cases} 0 & \text{if machine } M_e \text{ does not halt for input } n \\ 1 & \text{if machine } M_e \text{ halts for input } n \end{cases}$$

Definition 11.3 (Halting problem)**.** The *Halting Problem* is the problem of determining (for any m, w) whether the Turing machine M_e halts for an input of n strokes.

We show that h is not Turing-computable by showing that a related function, s, is not Turing-computable. This proof relies on the fact that anything that can be computed by a Turing machine can be computed using just two symbols: 0 and 1, and the fact that two Turing machines can be hooked together to create a single machine.

Definition 11.4. The function s is defined as

$$s(e) = \begin{cases} 0 & \text{if machine } M_e \text{ does not halt for input } e \\ 1 & \text{if machine } M_e \text{ halts for input } e \end{cases}$$

Lemma 11.5. *The function s is not Turing computable.*

Proof. We suppose, for contradiction, that the function s is Turing-computable. Then there would be a Turing machine S that computes s. We may assume, without loss of generality, that when S halts, it does so while scanning the first square. This machine can be "hooked up" to another machine J, which halts if it is started on a blank tape (i.e., if it reads 0 in the initial state while scanning the square to the right of the end-of-tape symbol), and otherwise wanders off to the right, never halting. $S \frown J$, the machine created by hooking S to J, is a Turing machine, so it is M_e for some e (i.e., it appears somewhere in the enumeration). Start M_e on an input of e 1s. There are two possibilities: either M_e halts or it does not halt.

1. Suppose M_e halts for an input of e 1s. Then $s(e) = 1$. So S, when started on e, halts with a single 1 as output on the tape. Then J starts with a 1 on the tape. In that case J does not halt. But M_e is the machine $S \frown J$, so it should do exactly what S followed by J would do. So M_e cannot halt for an input of e 1's.

2. Now suppose M_e does not halt for an input of e 1s. Then $s(e) = 0$, and S, when started on input e, halts with a blank tape. J, when started on a blank tape, immediately halts. Again, M_e does what S followed by J would do, so M_e must halt for an input of e 1's.

This shows there cannot be a Turing machine S: s is not Turing computable. □

Theorem 11.6 (Unsolvability of the Halting Problem)**.** *The halting problem is unsolvable, i.e., the function h is not Turing computable.*

Proof. Suppose h were Turing computable, say, by a Turing machine H. We could use H to build a Turing machine that computes s: First, make a copy of the input (separated by a blank). Then move back to the beginning, and run H. We can clearly make a machine that does the former, and if H existed, we would be able to "hook it up" to such a modified doubling machine to get a new machine which would determine if M_e halts on input e, i.e., computes s. But we've already shown that no such machine can exist. Hence, h is also not Turing computable. □

11.4 The Decision Problem

We say that first-order logic is *decidable* iff there is an effective method for determining whether or not a given sentence is valid. As it turns out, there is no such method: the problem of deciding validity of first-order sentences is unsolvable.

In order to establish this important negative result, we prove that the decision problem cannot be solved by a Turing machine. That is, we show that there is no Turing machine which, whenever it is started on a tape that contains a first-order sentence, eventually halts and outputs either 1 or 0 depending on whether the sentence is valid or not. By the Church-Turing thesis, every function which is computable is Turing computable. So if if this "validity function" were effectively computable at all, it would be Turing computable. If it isn't Turing computable, then, it also cannot be effectively computable.

Our strategy for proving that the decision problem is unsolvable is to reduce the halting problem to it. This means the following: We have proved that the function $h(e, w)$ that halts with output 1 if the Turing-machine described by e halts on input w and outputs 0 otherwise, is not Turing-computable. We will show that if there were a Turing machine that decides validity of first-order sentences, then there is also Turing machine that computes h. Since h cannot be computed by a Turing machine, there cannot be a Turing machine that decides validity either.

The first step in this strategy is to show that for every input w and a Turing machine M, we can effectively describe a sentence $\tau(M, w)$ representing the

instruction set of M and the input w and a sentence $\alpha(M, w)$ expressing "M eventually halts" such that:

$\vDash \tau(M, w) \supset \alpha(M, w)$ iff M halts for input w.

The bulk of our proof will consist in describing these sentences $\tau(M, w)$ and $\alpha(M, w)$ and verifying that $\tau(M, w) \supset \alpha(M, w)$ is valid iff M halts on input w.

11.5 Representing Turing Machines

In order to represent Turing machines and their behavior by a sentence of first-order logic, we have to define a suitable language. The language consists of two parts: predicate symbols for describing configurations of the machine, and expressions for numbering execution steps ("moments") and positions on the tape.

We introduce two kinds of predicate symbols, both of them 2-place: For each state q, a predicate symbol Q_q, and for each tape symbol σ, a predicate symbol S_σ. The former allow us to describe the state of M and the position of its tape head, the latter allow us to describe the contents of the tape.

In order to express the positions of the tape head and the number of steps executed, we need a way to express numbers. This is done using a constant symbol o, and a 1-place function $\prime$, the successor function. By convention it is written *after* its argument (and we leave out the parentheses). So o names the leftmost position on the tape as well as the time before the first execution step (the initial configuration), o' names the square to the right of the leftmost square, and the time after the first execution step, and so on. We also introduce a predicate symbol $<$ to express both the ordering of tape positions (when it means "to the left of") and execution steps (then it means "before").

Once we have the language in place, we list the "axioms" of $\tau(M, w)$, i.e., the sentences which, taken together, describe the behavior of M when run on input w. There will be sentences which lay down conditions on o, $\prime$, and $<$, sentences that describes the input configuration, and sentences that describe what the configuration of M is after it executes a particular instruction.

Definition 11.7. Given a Turing machine $M = \langle Q, \Sigma, q_0, \delta \rangle$, the language $\mathcal{L}_M$ consists of:

1. A two-place predicate symbol $Q_q(x, y)$ for every state $q \in Q$. Intuitively, $Q_q(\overline{m}, \overline{n})$ expresses "after n steps, M is in state q scanning the mth square."

2. A two-place predicate symbol $S_\sigma(x, y)$ for every symbol $\sigma \in \Sigma$. Intuitively, $S_\sigma(\overline{m}, \overline{n})$ expresses "after n steps, the mth square contains symbol σ."

3. A constant symbol o

4. A one-place function symbol $'$

5. A two-place predicate symbol $<$

For each number n there is a canonical term $\overline{n}$, the *numeral* for n, which represents it in $\mathcal{L}_M$. $\overline{0}$ is o, $\overline{1}$ is o', $\overline{2}$ is o'', and so on. More formally:

$$\overline{0} = \mathsf{o}$$
$$\overline{n+1} = \overline{n}'$$

The sentences describing the operation of the Turing machine M on input $w = \sigma_{i_1} \dots \sigma_{i_k}$ are the following:

1. Axioms describing numbers:

   a) A sentence that says that the successor function is injective:

   $$\forall x \, \forall y \, (x' = y' \supset x = y)$$

   b) A sentence that says that every number is less than its successor:

   $$\forall x \, x < x'$$

   c) A sentence that ensures that $<$ is transitive:

   $$\forall x \, \forall y \, \forall z \, ((x < y \,\&\, y < z) \supset x < z)$$

   d) A sentence that connects $<$ and $=$:

   $$\forall x \, \forall y \, (x < y \supset x \neq y)$$

2. Axioms describing the input configuration:

   a) After after 0 steps—before the machine starts—M is in the inital state q_0, scanning square 1:

   $$Q_{q_0}(\overline{1}, \overline{0})$$

   b) The first $k+1$ squares contain the symbols $\triangleright, \sigma_{i_1}, \dots, \sigma_{i_k}$:

   $$S_{\triangleright}(\overline{0}, \overline{0}) \,\&\, S_{\sigma_{i_1}}(\overline{1}, \overline{0}) \,\&\, \cdots \,\&\, S_{\sigma_{i_k}}(\overline{n}, \overline{0})$$

   c) Otherwise, the tape is empty:

   $$\forall x \, (\overline{k} < x \supset S_0(x, \overline{0}))$$

3. Axioms describing the transition from one configuration to the next:

   For the following, let $\varphi(x, y)$ be the conjunction of all sentences of the form

   $$\forall z\,(((z < x \vee x < z)\ \&\ S_\sigma(z, y)) \supset S_\sigma(z, y'))$$

   where $\sigma \in \Sigma$. We use $\varphi(\overline{m}, \overline{n})$ to express "other than at square m, the tape after $n + 1$ steps is the same as after n steps."

   a) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', R\rangle$, the sentence:

   $$\begin{aligned}\forall x\,\forall y\,((&Q_{q_i}(x, y)\ \&\ S_\sigma(x, y)) \supset\\ &(Q_{q_j}(x', y')\ \&\ S_{\sigma'}(x, y')\ \&\ \varphi(x, y)))\end{aligned}$$

   This says that if, after y steps, the machine is in state q_i scanning square x which contains symbol σ, then after $y + 1$ steps it is scanning square $x + 1$, is in state q_j, square x now contains σ', and every square other than x contains the same symbol as it did after y steps.

   b) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', L\rangle$, the sentence:

   $$\begin{aligned}\forall x\,\forall y\,((&Q_{q_i}(x', y)\ \&\ S_\sigma(x', y)) \supset\\ &(Q_{q_j}(x, y')\ \&\ S_{\sigma'}(x', y')\ \&\ \varphi(x, y)))\ \&\\ \forall y\,((&Q_{q_i}(\mathsf{o}, y)\ \&\ S_\sigma(\mathsf{o}, y)) \supset\\ &(Q_{q_j}(\mathsf{o}, y')\ \&\ S_{\sigma'}(\mathsf{o}, y')\ \&\ \varphi(\mathsf{o}, y)))\end{aligned}$$

   Take a moment to think about how this works: now we don't start with "if scanning square $x \ldots$" but: "if scanning square $x + 1 \ldots$" A move to the left means that in the next step the machine is scanning square x. But the square that is written on is $x + 1$. We do it this way since we don't have subtraction or a predecessor function.

   Note that numbers of the form $x + 1$ are 1, 2, ..., i.e., this doesn't cover the case where the machine is scanning square 0 and is supposed to move left (which of course it can't—it just stays put). That special case is covered by the second conjunction: it says that if, after y steps, the machine is scanning square 0 in state q_i and square 0 contains symbol σ, then after $y + 1$ steps it's still scanning square 0, is now in state q_j, the symbol on square 0 is σ', and the squares other than square 0 contain the same symbols they contained ofter y steps.

   c) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', N\rangle$, the sentence:

   $$\begin{aligned}\forall x\,\forall y\,((&Q_{q_i}(x, y)\ \&\ S_\sigma(x, y)) \supset\\ &(Q_{q_j}(x, y')\ \&\ S_{\sigma'}(x, y')\ \&\ \varphi(x, y)))\end{aligned}$$

Let $\tau(M, w)$ be the conjunction of all the above sentences for Turing machine M and input w

In order to express that M eventually halts, we have to find a sentence that says "after some number of steps, the transition function will be undefined." Let X be the set of all pairs $\langle q, \sigma \rangle$ such that $\delta(q, \sigma)$ is undefined. Let $\alpha(M, w)$ then be the sentence

$$\exists x \, \exists y \, (\bigvee_{\langle q,\sigma \rangle \in X} (Q_q(x, y) \,\&\, S_\sigma(x, y)))$$

If we use a Turing machine with a designated halting state h, it is even easier: then the sentence $\alpha(M, w)$

$$\exists x \, \exists y \, Q_h(x, y)$$

expresses that the machine eventually halts.

Proposition 11.8. *If $m < k$, then $\tau(M, w) \vDash \overline{m} < \overline{k}$*

Proof. Exercise. □

11.6 Verifying the Representation

In order to verify that our representation works, we have to prove two things. First, we have to show that if M halts on input w, then $\tau(M, w) \supset \alpha(M, w)$ is valid. Then, we have to show the converse, i.e., that if $\tau(M, w) \supset \alpha(M, w)$ is valid, then M does in fact eventually halt when run on input w.

The strategy for proving these is very different. For the first result, we have to show that a sentence of first-order logic (namely, $\tau(M, w) \supset \alpha(M, w)$) is valid. The easiest way to do this is to give a derivation. Our proof is supposed to work for all M and w, though, so there isn't really a single sentence for which we have to give a derivation, but infinitely many. So the best we can do is to prove by induction that, whatever M and w look like, and however many steps it takes M to halt on input w, there will be a derivation of $\tau(M, w) \supset \alpha(M, w)$.

Naturally, our induction will proceed on the number of steps M takes before it reaches a halting configuration. In our inductive proof, we'll establish that for each step n of the run of M on input w, $\tau(M, w) \vDash \chi(M, w, n)$, where $\chi(M, w, n)$ correctly describes the configuration of M run on w after n steps. Now if M halts on input w after, say, n steps, $\chi(M, w, n)$ will describe a halting configuration. We'll also show that $\chi(M, w, n) \vDash \alpha(M, w)$, whenever $\chi(M, w, n)$ describes a halting configuration. So, if M halts on input w, then for some n, M will be in a halting configuration after n steps. Hence, $\tau(M, w) \vDash \chi(M, w, n)$ where $\chi(M, w, n)$ describes a halting configuration, and since in that case $\chi(M, w, n) \vDash \alpha(M, w)$, we get that $T(M, w) \vDash \alpha(M, w)$, i.e., that $\vDash \tau(M, w) \supset \alpha(M, w)$.

The strategy for the converse is very different. Here we assume that $\vDash \tau(M,w) \supset \alpha(M,w)$ and have to prove that M halts on input w. From the hypothesis we get that $\tau(M,w) \vDash \alpha(M,w)$, i.e., $\alpha(M,w)$ is true in every structure in which $\tau(M,w)$ is true. So we'll describe a structure $\mathfrak{M}$ in which $\tau(M,w)$ is true: its domain will be $\mathbb{N}$, and the interpretation of all the Q_q and S_σ will be given by the configurations of M during a run on input w. So, e.g., $\mathfrak{M} \models Q_q(\overline{m},\overline{n})$ iff T, when run on input w for n steps, is in state q and scanning square m. Now since $\tau(M,w) \vDash \alpha(M,w)$ by hypothesis, and since $\mathfrak{M} \models \tau(M,w)$ by construction, $\mathfrak{M} \models \alpha(M,w)$. But $\mathfrak{M} \models \alpha(M,w)$ iff there is some $n \in |\mathfrak{M}| = \mathbb{N}$ so that M, run on input w, is in a halting configuration after n steps.

Definition 11.9. Let $\chi(M,w,n)$ be the sentence

$$Q_q(\overline{m},\overline{n}) \,\&\, S_{\sigma_0}(\overline{0},\overline{n}) \,\&\, \cdots \,\&\, S_{\sigma_k}(\overline{k},\overline{n}) \,\&\, \forall x\,(\overline{k} < x \supset S_0(x,\overline{n}))$$

where q is the state of M at time n, M is scanning square m at time n, square i contains symbol σ_i at time n for $0 \leq i \leq k$ and k is the right-most non-blank square of the tape at time 0, or the right-most square the tape head has visited after n steps, whichever is greater.

Lemma 11.10. *If M run on input w is in a halting configuration after n steps, then $\chi(M,w,n) \vDash \alpha(M,w)$.*

Proof. Suppose that M halts for input w after n steps. There is some state q, square m, and symbol σ such that:

1. After n steps, M is in state q scanning square m on which σ appears.
2. The transition function $\delta(q,\sigma)$ is undefined.

$\chi(M,w,n)$ is the description of this configuration and will include the clauses $Q_q(\overline{m},\overline{n})$ and $S_\sigma(\overline{m},\overline{n})$. These clauses together imply $\alpha(M,w)$:

$$\exists x\,\exists y\,(\bigvee_{\langle q,\sigma\rangle \in X} (Q_q(x,y) \,\&\, S_\sigma(x,y)))$$

since $Q_{q'}(\overline{m},\overline{n}) \,\&\, S_{\sigma'}(\overline{m},\overline{n}) \vDash \bigvee_{\langle q,\sigma\rangle \in X}(Q_q(\overline{m},\overline{n}) \,\&\, S_\sigma(\overline{m},\overline{n}))$, as $\langle q',\sigma'\rangle \in X$. □

So if M halts for input w, then there is some n such that $\chi(M,w,n) \vDash \alpha(M,w)$. We will now show that for any time n, $\tau(M,w) \vDash \chi(M,w,n)$.

Lemma 11.11. *For each n, if M has not halted after n steps, $\tau(M,w) \vDash \chi(M,w,n)$.*

Proof. Induction basis: If $n = 0$, then the conjuncts of $\chi(M,w,0)$ are also conjuncts of $\tau(M,w)$, so entailed by it.

Inductive hypothesis: If M has not halted before the nth step, then $\tau(M, w) \vDash \chi(M, w, n)$. We have to show that (unless $\chi(M, w, n)$ describes a halting configuration), $\tau(M, w) \vDash \chi(M, w, n+1)$.

Suppose $n > 0$ and after n steps, M started on w is in state q scanning square m. Since M does not halt after n steps, there must be an instruction of one of the following three forms in the program of M:

1. $\delta(q, \sigma) = \langle q', \sigma', R \rangle$
2. $\delta(q, \sigma) = \langle q', \sigma', L \rangle$
3. $\delta(q, \sigma) = \langle q', \sigma', N \rangle$

We will consider each of these three cases in turn.

1. Suppose there is an instruction of the form (1). By Definition 11.7, (3a), this means that

$$\forall x \, \forall y \, ((Q_q(x, y) \mathbin{\&} S_\sigma(x, y)) \supset (Q_{q'}(x', y') \mathbin{\&} S_{\sigma'}(x, y') \mathbin{\&} \varphi(x, y)))$$

is a conjunct of $\tau(M, w)$. This entails the following sentence (universal instantiation, $\overline{m}$ for x and $\overline{n}$ for y):

$$(Q_q(\overline{m}, \overline{n}) \mathbin{\&} S_\sigma(\overline{m}, \overline{n})) \supset (Q_{q'}(\overline{m}', \overline{n}') \mathbin{\&} S_{\sigma'}(\overline{m}, \overline{n}') \mathbin{\&} \varphi(\overline{m}, \overline{n})).$$

By induction hypothesis, $\tau(M, w) \vDash \chi(M, w, n)$, i.e.,

$$Q_q(\overline{m}, \overline{n}) \mathbin{\&} S_{\sigma_0}(\overline{0}, \overline{n}) \mathbin{\&} \cdots \mathbin{\&} S_{\sigma_k}(\overline{k}, \overline{n}) \mathbin{\&} \forall x \, (\overline{k} < x \supset S_0(x, \overline{n}))$$

Since after n steps, tape square m contains σ, the corresponding conjunct is $S_\sigma(\overline{m}, \overline{n})$, so this entails:

$$Q_q(\overline{m}, \overline{n}) \mathbin{\&} S_\sigma(\overline{m}, \overline{n}))$$

We now get

$$Q_{q'}(\overline{m}', \overline{n}') \mathbin{\&} S_{\sigma'}(\overline{m}, \overline{n}') \mathbin{\&} S_{\sigma_0}(\overline{0}, \overline{n}') \mathbin{\&} \cdots \mathbin{\&} S_{\sigma_k}(\overline{k}, \overline{n}') \mathbin{\&} \forall x \, (\overline{k} < x \supset S_0(x, \overline{n}'))$$

as follows: The first line comes directly from the consequent of the preceding conditional, by modus ponens. Each conjunct in the middle

line—which excludes $S_{\sigma_m}(\overline{m}, \overline{n}')$—follows from the corresponding conjunct in $\chi(M, w, n)$ together with $\varphi(\overline{m}, \overline{n})$.

If $m < k$, $\tau(M, w) \vdash \overline{m} < \overline{k}$ (Proposition 11.8) and by transitivity of $<$, we have $\forall x\,(\overline{k} < x \supset \overline{m} < x)$. If $m = k$, then $\forall x\,(\overline{k} < x \supset \overline{m} < x)$ by logic alone. The last line then follows from the corresponding conjunct in $\chi(M, w, n)$, $\forall x\,(\overline{k} < x \supset \overline{m} < x)$, and $\varphi(\overline{m}, \overline{n})$. If $m < k$, this already is $\chi(M, w, n+1)$.

Now suppose $m = k$. In that case, after $n+1$ steps, the tape head has also visited square $k+1$, which now is the right-most square visited. So $\chi(M, w, n+1)$ has a new conjunct, $S_0(\overline{k}', \overline{n}')$, and the last conjuct is $\forall x\,(\overline{k}' < x \supset S_0(x, \overline{n}'))$. We have to verify that these two sentences are also implied.

We already have $\forall x\,(\overline{k} < x \supset S_0(x, \overline{n}'))$. In particular, this gives us $\overline{k} < \overline{k}' \supset S_0(\overline{k}', \overline{n}')$. From the axiom $\forall x\, x < x'$ we get $\overline{k} < \overline{k}'$. By modus ponens, $S_0(\overline{k}', \overline{n}')$ follows.

Also, since $\tau(M, w) \vdash \overline{k} < \overline{k}'$, the axiom for transitivity of $<$ gives us $\forall x\,(\overline{k}' < x \supset S_0(x, \overline{n}'))$. (We leave the verification of this as an exercise.)

2. Suppose there is an instruction of the form (2). Then, by Definition 11.7, (3b),

$$\begin{aligned}
&\forall x\,\forall y\,((Q_q(x', y) \mathbin{\&} S_\sigma(x', y)) \supset \\
&\qquad (Q_{q'}(x, y') \mathbin{\&} S_{\sigma'}(x', y') \mathbin{\&} \varphi(x, y))) \mathbin{\&} \\
&\forall y\,((Q_{q_i}(\mathsf{o}, y) \mathbin{\&} S_\sigma(\mathsf{o}, y)) \supset \\
&\qquad (Q_{q_j}(\mathsf{o}, y') \mathbin{\&} S_{\sigma'}(\mathsf{o}, y') \mathbin{\&} \varphi(\mathsf{o}, y)))
\end{aligned}$$

is a conjunct of $\tau(M, w)$. If $m > 0$, then let $l = m - 1$ (i.e., $m = l + 1$). The first conjunct of the above sentence entails the following:

$$\begin{aligned}
&(Q_q(\overline{l}', \overline{n}) \mathbin{\&} S_\sigma(\overline{l}', \overline{n})) \supset \\
&\qquad (Q_{q'}(\overline{l}, \overline{n}') \mathbin{\&} S_{\sigma'}(\overline{l}', \overline{n}') \mathbin{\&} \varphi(\overline{l}, \overline{n}))
\end{aligned}$$

Otherwise, let $l = m = 0$ and consider the following sentence entailed by the second conjunct:

$$\begin{aligned}
&((Q_{q_i}(\mathsf{o}, \overline{n}) \mathbin{\&} S_\sigma(\mathsf{o}, \overline{n})) \supset \\
&\qquad (Q_{q_j}(\mathsf{o}, \overline{n}') \mathbin{\&} S_{\sigma'}(\mathsf{o}, \overline{n}') \mathbin{\&} \varphi(\mathsf{o}, \overline{n})))
\end{aligned}$$

Either sentence implies

$$\begin{aligned} &Q_{q'}(\bar{l}, \overline{n}') \mathbin{\&} S_{\sigma'}(\overline{m}, \overline{n}') \mathbin{\&} \\ &\quad S_{\sigma_0}(\overline{0}, \overline{n}') \mathbin{\&} \cdots \mathbin{\&} S_{\sigma_k}(\overline{k}, \overline{n}') \mathbin{\&} \\ &\quad \forall x\, (\overline{k} < x \supset S_0(x, \overline{n}')) \end{aligned}$$

as before. (Note that in the first case, $\bar{l}' = \overline{m}$ and in the second case $\bar{l} = 0$.) But this just is $\chi(M, w, n+1)$.

3. Case (3) is left as an exercise.

We have shown that for any n, $\tau(M, w) \vDash \chi(M, w, n)$. □

Lemma 11.12. *If M halts on input w, then $\tau(M, w) \supset \alpha(M, w)$ is valid.*

Proof. By Lemma 11.11, we know that, for any time n, the description $\chi(M, w, n)$ of the configuration of M at time n is entailed by $\tau(M, w)$. Suppose M halts after k steps. It will be scanning square m, say. Then $\chi(M, w, k)$ describes a halting configuration of M, i.e., it contains as conjuncts both $Q_q(\overline{m}, \overline{k})$ and $S_\sigma(\overline{m}, \overline{k})$ with $\delta(q, \sigma)$ undefined. By Lemma 11.10 Thus, $\chi(M, w, k) \vDash \alpha(M, w)$. But since $(M, w) \vDash \chi(M, w, k)$, we have $\tau(M, w) \vDash \alpha(M, w)$ and therefore $\tau(M, w) \supset \alpha(M, w)$ is valid. □

To complete the verification of our claim, we also have to establish the reverse direction: if $\tau(M, w) \supset \alpha(M, w)$ is valid, then M does in fact halt when started on input m.

Lemma 11.13. *If $\vDash \tau(M, w) \supset \alpha(M, w)$, then M halts on input w.*

Proof. Consider the $\mathcal{L}_M$-structure $\mathfrak{M}$ with domain $\mathbb{N}$ which interprets 0 as 0, $'$ as the successor function, and $<$ as the less-than relation, and the predicates Q_q and S_σ as follows:

$$Q_q^{\mathfrak{M}} = \{\langle m, n\rangle \mid \begin{array}{l}\text{started on } w, \text{ after } n \text{ steps,} \\ M \text{ is in state } q \text{ scanning square } m\end{array}\}$$

$$S_\sigma^{\mathfrak{M}} = \{\langle m, n\rangle \mid \begin{array}{l}\text{started on } w, \text{ after } n \text{ steps,} \\ \text{square } m \text{ of } M \text{ contains symbol } \sigma\end{array}\}$$

In other words, we construct the structure $\mathfrak{M}$ so that it describes what M started on input w actually does, step by step. Clearly, $\mathfrak{M} \models \tau(M, w)$. If $\vDash \tau(M, w) \supset \alpha(M, w)$, then also $\mathfrak{M} \models \alpha(M, w)$, i.e.,

$$\mathfrak{M} \models \exists x\, \exists y\, (\bigvee_{\langle q,\sigma\rangle \in X} (Q_q(x, y) \mathbin{\&} S_\sigma(x, y))).$$

As $|\mathfrak{M}| = \mathbb{N}$, there must be $m, n \in \mathbb{N}$ so that $\mathfrak{M} \models Q_q(\overline{m}, \overline{n}) \mathbin{\&} S_\sigma(\overline{m}, \overline{n})$ for some q and σ such that $\delta(q, \sigma)$ is undefined. By the definition of $\mathfrak{M}$, this means that M started on input w after n steps is in state q and reading symbol σ, and the transition function is undefined, i.e., M has halted. □

11.7 The Decision Problem is Unsolvable

Theorem 11.14. *The decision problem is unsolvable.*

Proof. Suppose the decision problem were solvable, i.e., suppose there were a Turing machine D of the following sort. Whenever D is started on a tape that contains a sentence ψ of first-order logic as input, D eventually halts, and outputs 1 iff ψ is valid and 0 otherwise. Then we could solve the halting problem as follows. We construct a Turing machine E that, given as input the number e of Turing machine M_e and input w, computes the corresponding sentence $\tau(M_e, w) \supset \alpha(M_e, w)$ and halts, scanning the leftmost square on the tape. The machine $E \frown D$ would then, given input e and w, first compute $\tau(M_e, w) \supset \alpha(M_e, w)$ and then run the decision problem machine D on that input. D halts with output 1 iff $\tau(M_e, w) \supset \alpha(M_e, w)$ is valid and outputs 0 otherwise. By Lemma 11.13 and Lemma 11.12, $\tau(M_e, w) \supset \alpha(M_e, w)$ is valid iff M_e halts on input w. Thus, $E \frown D$, given input e and w halts with output 1 iff M_e halts on input w and halts with output 0 otherwise. In other words, $E \frown D$ would solve the halting problem. But we know, by Theorem 11.6, that no such Turing machine can exist. □

Part IV

Computability and Incompleteness

Chapter 12

Recursive Functions

12.1 Introduction

In order to develop a mathematical theory of computability, one has to first of all develop a *model* of computability. We now think of computability as the kind of thing that computers do, and computers work with symbols. But at the beginning of the development of theories of computability, the paradigmatic example of computation was *numerical* computation. Mathematicians were always interested in number-theoretic functions, i.e., functions $f : \mathbb{N}^n \to \mathbb{N}$ that can be computed. So it is not surprising that at the beginning of the theory of computability, it was such functions that were studied. The most familiar examples of computable numerical functions, such as addition, multiplication, exponentiation (of natural numbers) share an interesting feature: they can be defined *recursively*. It is thus quite natural to attempt a general definition of *computable function* on the basis of recursive definitions. Among the many possible ways to define number-theoretic functions recursively, one particulalry simple pattern of definition here becomes central: so-called *primitive recursion*.

In addition to computable functions, we might be interested in computable sets and relations. A set is computable if we can compute the answer to whether or not a given number is an element of the set, and a relation is computable iff we can compute whether or not a tuple $\langle n_1, \ldots, n_k \rangle$ is an element of the relation. By considering the *characteristic function* of a set or relation, discussion of computable sets and relations can be subsumed under that of computable functions. Thus we can define primitive recursive relations as well, e.g., the relation "n evenly divides m" is a primitive recursive relation.

Primitive recursive functions—those that can be defined using just primitive recursion—are not, however, the only computable number-theoretic functions. Many generalizations of primitive recursion have been considered, but the most powerful and widely-accepted additional way of computing functions is by unbounded search. This leads to the definition of *partial recur-*

sive functions, and a related definition to *general recursive functions*. General recursive functions are computable and total, and the definition characterizes exactly the partial recursive functions that happen to be total. Recursive functions can simulate every other model of computation (Turing machines, lambda calculus, etc.) and so represent one of the many accepted models of computation.

12.2 Primitive Recursion

Suppose we specify that a certain function l from $\mathbb{N}$ to $\mathbb{N}$ satisfies the following two clauses:

$$\begin{aligned} l(0) &= 1 \\ l(x+1) &= 2 \cdot l(x). \end{aligned}$$

It is pretty clear that there is only one function, l, that meets these two criteria. This is an instance of a *definition by primitive recursion*. We can define even more fundamental functions like addition and multiplication by

$$\begin{aligned} f(x,0) &= x \\ f(x,y+1) &= f(x,y)+1 \end{aligned}$$

and

$$\begin{aligned} g(x,0) &= 0 \\ g(x,y+1) &= f(g(x,y),x). \end{aligned}$$

Exponentiation can also be defined recursively, by

$$\begin{aligned} h(x,0) &= 1 \\ h(x,y+1) &= g(h(x,y),x). \end{aligned}$$

We can also compose functions to build more complex ones; for example,

$$\begin{aligned} k(x) &= x^x + (x+3) \cdot x \\ &= f(h(x,x), g(f(x,3),x)). \end{aligned}$$

Let $\text{zero}(x)$ be the function that always returns 0, regardless of what x is, and let $\text{succ}(x) = x+1$ be the successor function. The set of *primitive recursive functions* is the set of functions from $\mathbb{N}^n$ to $\mathbb{N}$ that you get if you start with zero and succ by iterating the two operations above, primitive recursion and composition. The idea is that primitive recursive functions are defined in a straightforward and explicit way, so that it is intuitively clear that each one can be computed using finite means.

Definition 12.1. If f is a k-place function and $g_0, \ldots, g_{k-1}$ are l-place functions on the natural numbers, the *composition* of f with $g_0, \ldots, g_{k-1}$ is the l-place function h defined by

$$h(x_0, \ldots, x_{l-1}) = f(g_0(x_0, \ldots, x_{l-1}), \ldots, g_{k-1}(x_0, \ldots, x_{l-1})).$$

Definition 12.2. If f is a k-place function and g is a $(k+2)$-place function, then the function defined by *primitive recursion from f and g* is the $(k+1)$-place function h defined by the equations

$$\begin{aligned} h(0, z_0, \ldots, z_{k-1}) &= f(z_0, \ldots, z_{k-1}) \\ h(x+1, z_0, \ldots, z_{k-1}) &= g(x, h(x, z_0, \ldots, z_{k-1}), z_0, \ldots, z_{k-1}) \end{aligned}$$

In addition to zero and succ, we will include among primitive recursive functions the projection functions,

$$P_i^n(x_0, \ldots, x_{n-1}) = x_i,$$

for each natural number n and $i < n$. These are not terribly exciting in themselves: P_i^n is simply the k-place function that always returns its ith argument. But the allow us to define new functions by disregarding arguments or switching arguments, as we'll see later.

In the end, we have the following:

Definition 12.3. The set of primitive recursive functions is the set of functions from $\mathbb{N}^n$ to $\mathbb{N}$, defined inductively by the following clauses:

1. zero is primitive recursive.
2. succ is primitive recursive.
3. Each projection function P_i^n is primitive recursive.
4. If f is a k-place primitive recursive function and $g_0, \ldots, g_{k-1}$ are l-place primitive recursive functions, then the composition of f with $g_0, \ldots, g_{k-1}$ is primitive recursive.
5. If f is a k-place primitive recursive function and g is a $k+2$-place primitive recursive function, then the function defined by primitive recursion from f and g is primitive recursive.

Put more concisely, the set of primitive recursive functions is the smallest set containing zero, succ, and the projection functions P_j^n, and which is closed under composition and primitive recursion.

Another way of describing the set of primitive recursive functions keeps track of the "stage" at which a function enters the set. Let S_0 denote the set of starting functions: zero, succ, and the projections. Once S_i has been defined,

let S_{i+1} be the set of all functions you get by applying a single instance of composition or primitive recursion to functions in S_i. Then

$$S = \bigcup_{i \in \mathbb{N}} S_i$$

is the set of all primitive recursive functions

Our definition of composition may seem too rigid, since $g_0, \ldots, g_{k-1}$ are all required to have the same arity l. (Remember that the *arity* of a function is the number of arguments; an l-place function has arity l.) But adding the projection functions provides the desired flexibility. For example, suppose f and g are 3-place functions and h is the 2-place function defined by

$$h(x, y) = f(x, g(x, x, y), y).$$

The definition of h can be rewritten with the projection functions, as

$$h(x, y) = f(P_0^2(x, y), g(P_0^2(x, y), P_0^2(x, y), P_1^2(x, y)), P_1^2(x, y)).$$

Then h is the composition of f with P_0^2, l, and P_1^2, where

$$l(x, y) = g(P_0^2(x, y), P_0^2(x, y), P_1^2(x, y)),$$

i.e., l is the composition of g with P_0^2, P_0^2, and P_1^2.

For another example, let us again consider addition. This is described recursively by the following two equations:

$$\begin{aligned} x + 0 &= x \\ x + (y + 1) &= \text{succ}(x + y). \end{aligned}$$

In other words, addition is the function add defined recursively by the equations

$$\begin{aligned} \text{add}(0, x) &= x \\ \text{add}(y + 1, x) &= \text{succ}(\text{add}(y, x)). \end{aligned}$$

But even this is not a strict primitive recursive definition; we need to put it in the form

$$\begin{aligned} \text{add}(0, x) &= f(x) \\ \text{add}(y + 1, x) &= g(y, \text{add}(y, x), x) \end{aligned}$$

for some 1-place primitive recursive function f and some 3-place primitive recursive function g. We can take f to be P_0^1, and we can define g using composition,

$$g(y, w, x) = \text{succ}(P_1^3(y, w, x)).$$

The function g, being the composition of basic primitive recursive functions, is primitive recursive; and hence so is h. (Note that, strictly speaking, we have defined the function $g(y, x)$ meeting the recursive specification of $x + y$; in other words, the variables are in a different order. Luckily, addition is commutative, so here the difference is not important; otherwise, we could define the function g' by

$$g'(x, y) = g(P_1^2(y, x)), P_0^2(y, x)) = g(y, x),$$

using composition.

One advantage to having the precise description of the primitive recursive functions is that we can be systematic in describing them. For example, we can assign a "notation" to each such function, as follows. Use symbols zero, succ, and P_i^n for zero, successor, and the projections. Now suppose f is defined by composition from a k-place function h and l-place functions g_0, $\ldots$, g_{k-1}, and we have assigned notations H, G_0, $\ldots$, G_{k-1} to the latter functions. Then, using a new symbol $\text{Comp}_{k,l}$, we can denote the function f by $\text{Comp}_{k,l}[H, G_0, \ldots, G_{k-1}]$. For the functions defined by primitive recursion, we can use analogous notations of the form $\text{Rec}_k[G, H]$, where k denotes that arity of the function being defined. With this setup, we can denote the addition function by

$$\text{Rec}_2[P_0^1, \text{Comp}_{1,3}[\text{succ}, P_1^3]].$$

Having these notations sometimes proves useful.

12.3 Primitive Recursive Functions are Computable

Suppose a function h is defined by primitive recursion

$$\begin{aligned} h(0, \vec{z}) &= f(\vec{z}) \\ h(x + 1, \vec{z}) &= g(x, h(x, \vec{z}), \vec{z}) \end{aligned}$$

and suppose the functions f and g are computable. Then $h(0, \vec{z})$ can obviously be computed, since it is just $f(\vec{z})$ which we assume is computable. $h(1, \vec{z})$ can then also be computed, since $1 = 0 + 1$ and so $h(1, \vec{z})$ is just

$$g(0, h(0, \vec{z}), \vec{z}) = g(0, f(\vec{z}), \vec{z}).$$

We can go on in this way and compute

$$\begin{aligned} h(2, \vec{z}) &= g(1, g(0, f(\vec{z}), \vec{z}), \vec{z}) \\ h(3, \vec{z}) &= g(2, g(1, g(0, f(\vec{z}), \vec{z}), \vec{z}), \vec{z}) \\ h(4, \vec{z}) &= g(3, g(2, g(1, g(0, f(\vec{z}), \vec{z}), \vec{z}), \vec{z}), \vec{z}) \\ &\vdots \end{aligned}$$

Thus, to compute $h(x, \vec{z})$ in general, successively compute $h(0, \vec{z})$, $h(1, \vec{z})$, ..., until we reach $h(x, \vec{z})$.

Thus, primitive recursion yields a new computable function if the functions f and g are computable. Composition of functions also results in a computable function if the functions f and g_i are computable.

Since the basic functions zero, succ, and P_i^n are computable, and composition and primitive recursion yield computable functions from computable functions, his means that every primitive recursive function is computable.

12.4 Examples of Primitive Recursive Functions

Here are some examples of primitive recursive functions:

1. Constants: for each natural number n, the function that always returns n primitive recursive function, since it is equal to $\text{succ}(\text{succ}(\ldots \text{succ}(\text{zero}(x))))$.
2. The identity function: $\text{id}(x) = x$, i.e. P_0^1
3. Addition, $x + y$
4. Multiplication, $x \cdot y$
5. Exponentiation, x^y (with 0^0 defined to be 1)
6. Factorial, $x! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot x$
7. The predecessor function, $\text{pred}(x)$, defined by
$$\text{pred}(0) = 0, \quad \text{pred}(x+1) = x$$
8. Truncated subtraction, $x \dot{-} y$, defined by
$$x \dot{-} 0 = x, \quad x \dot{-} (y+1) = \text{pred}(x \dot{-} y)$$
9. Maximum, $\max(x, y)$, defined by
$$\max(x, y) = x + (y \dot{-} x)$$
10. Minimum, $\min(x, y)$
11. Distance between x and y, $|x - y|$

In our definitions, we'll often use constants n. This is ok because the constant function $\text{const}_n(x)$ is primitive recursive (defined from zero and succ). So if, e.g., we want to define the function $f(x) = 2 \cdot x$ can obtain it by composition from $\text{const}_n(x)$ and multiplication as $f(x) = \text{const}_2(x) \cdot P_0^1(x)$. We'll make use of this trick from now on.

You'll also have noticed that the definition of pred does not, strictly speaking, fit into the pattern of definition by primitive recursion, since that pattern requires an extra argument. It is also odd in that it does not actually $\text{pred}(x)$ in the definition of $\text{pred}(x+1)$. But we can define $\text{pred}'(x,y)$ by

$$\begin{aligned}\text{pred}'(0,y) &= \text{zero}(y) = 0\\ \text{pred}'(x+1,y) &= P^3_0(x,\text{pred}'(x,y),y) = x\end{aligned}$$

and then define pred from it by composition, e.g., as $\text{pred}(x) = \text{pred}'(P^1_0(x),\text{zero}(x))$.

The set of primitive recursive functions is further closed under the following two operations:

1. Finite sums: if $f(x,\vec{z})$ is primitive recursive, then so is the function

$$g(y,\vec{z}) = \sum_{x=0}^{y} f(x,\vec{z}).$$

2. Finite products: if $f(x,\vec{z})$ is primitive recursive, then so is the function

$$h(y,\vec{z}) = \prod_{x=0}^{y} f(x,\vec{z}).$$

For example, finite sums are defined recursively by the equations

$$g(0,\vec{z}) = f(0,\vec{z}), \quad g(y+1,\vec{z}) = g(y,\vec{z}) + f(y+1,\vec{z}).$$

We can also define boolean operations, where 1 stands for true, and 0 for false:

1. Negation, $\text{not}(x) = 1 \dot{-} x$
2. Conjunction, $\text{and}(x,y) = x \cdot y$

Other classical boolean operations like $\text{or}(x,y)$ and $\text{ifthen}(x,y)$ can be defined from these in the usual way.

12.5 Primitive Recursive Relations

Definition 12.4. A relation $R(\vec{x})$ is said to be primitive recursive if its characteristic function,

$$\chi_R(\vec{x}) = \begin{cases} 1 & \text{if } R(\vec{x}) \\ 0 & \text{otherwise} \end{cases}$$

is primitive recursive.

In other words, when one speaks of a primitive recursive relation $R(\vec{x})$, one is referring to a relation of the form $\chi_R(\vec{x}) = 1$, where χ_R is a primitive recursive function which, on any input, returns either 1 or 0. For example, the relation IsZero(x), which holds if and only if $x = 0$, corresponds to the function χ_{IsZero}, defined using primitive recursion by

$$\chi_{\text{IsZero}}(0) = 1, \quad \chi_{\text{IsZero}}(x+1) = 0.$$

It should be clear that one can compose relations with other primitive recursive functions. So the following are also primitive recursive:

1. The equality relation, $x = y$, defined by IsZero$(|x - y|)$
2. The less-than relation, $x \leq y$, defined by IsZero$(x \dot{-} y)$

Furthermore, the set of primitive recursive relations is closed under boolean operations:

1. Negation, $\sim P$
2. Conjunction, P & Q
3. Disjunction, $P \vee Q$
4. If ... then, $P \supset Q$

are all primitive recursive, if P and Q are. For suppose $\chi_P(\vec{z})$ an $\chi_Q(\vec{z})$ are primitive recursive. Then the relation $R(\vec{z})$ that holds iff both $P(\vec{z})$ and $Q(\vec{z})$ hold has the characteristic function $\chi_R(\vec{z}) = \text{and}(\chi_P(\vec{z}), \chi_Q(\vec{z}))$.

One can also define relations using bounded quantification:

1. Bounded universal quantification: if $R(x, \vec{z})$ is a primitive recursive relation, then so is the relation

   $$(\forall x < y)\ R(x, \vec{z})$$

   which holds if and only if $R(x, \vec{z})$ holds for every x less than y.

2. Bounded existential quantification: if $R(x, \vec{z})$ is a primitive recursive relation, then so is

   $$(\exists x < y)\ R(x, \vec{z}).$$

By convention, we take $(\forall x < 0)\ R(x, \vec{z})$ to be true (for the trivial reason that there *are* no x less than 0) and $(\exists x < 0)\ R(x, \vec{z})$ to be false. A universal quantifier functions just like a finite product; it can also be defined directly by

$$g(0, \vec{z}) = 1, \quad g(y+1, \vec{z}) = \text{and}(g(y, \vec{z}), \chi_R(y, \vec{z})).$$

Bounded existential quantification can similarly be defined using or. Alternatively, it can be defined from bounded universal quantification, using the

equivalence, $(\exists x < y)\ \varphi(x) \equiv \sim(\forall x < y)\ \sim\varphi(x)$. Note that, for example, a bounded quantifier of the form $(\exists x \leq y)\ \ldots x \ldots$ is equivalent to $(\exists x < y + 1)\ \ldots x \ldots$.

Another useful primitive recursive function is:

1. The conditional function, $\text{cond}(x, y, z)$, defined by

$$\text{cond}(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{otherwise} \end{cases}$$

This is defined recursively by

$$\text{cond}(0, y, z) = y, \quad \text{cond}(x + 1, y, z) = z.$$

One can use this to justify:

1. Definition by cases: if $g_0(\vec{x}), \ldots, g_m(\vec{x})$ are functions, and $R_1(\vec{x}), \ldots, R_{m-1}(\vec{x})$ are relations, then the function f defined by

$$f(\vec{x}) = \begin{cases} g_0(\vec{x}) & \text{if } R_0(\vec{x}) \\ g_1(\vec{x}) & \text{if } R_1(\vec{x}) \text{ and not } R_0(\vec{x}) \\ \vdots & \\ g_{m-1}(\vec{x}) & \text{if } R_{m-1}(\vec{x}) \text{ and none of the previous hold} \\ g_m(\vec{x}) & \text{otherwise} \end{cases}$$

is also primitive recursive.

When $m = 1$, this is just the function defined by

$$f(\vec{x}) = \text{cond}(\chi_{\sim R_0}(\vec{x}), g_0(\vec{x}), g_1(\vec{x})).$$

For m greater than 1, one can just compose definitions of this form.

12.6 Bounded Minimization

It is often useful to define a function as the least number satisfying some property or relation P. If P is decidable, we can compute this function simply by trying out all the possible numbers, $0, 1, 2, \ldots$, until we find the least one satisfying P. This kind of unbounded search takes us out of the realm of primitive recursive functions. However, if we're only interested in the least number *less than some indipendently given bound*, we stay primitive recursive. In other words, and a bit more generally, suppose we have a primitive recursive relation $R(x, z)$. Consider the function that maps y and z to the least $x < y$ such that $R(x, z)$. It, too, can be computed, by testing whether $R(0, z)$, $R(1, z)$, $\ldots$, $R(y - 1, z)$. But why is it primitive recursive?

Proposition 12.5. *If $R(x, \vec{z})$ is primitive recursive, so is the function $m_R(y, \vec{z})$ which returns the least x less than y such that $R(x, \vec{z})$ holds, if there is one, and 0 otherwise. We will write the function m_R as*

$$(\min x < y)\, R(x, \vec{z}),$$

Proof. Note than there can be no $x < 0$ such that $R(x, \vec{z})$ since there is no $x < 0$ at all. So $m_R(x, 0) = 0$.

In case the bound is $y + 1$ we have three cases: (a) There is an $x < y$ such that $R(x, \vec{z})$, in which case $m_R(y + 1, \vec{z}) = m_R(y, \vec{z})$. (b) There is no such x but $R(y, \vec{z})$ holds, then $m_R(y + 1, \vec{z}) = y$. (c) There is no $x < y + 1$ such that $R(x, \vec{z})$, then $m_R(y + 1, \vec{z}) = 0$. So,

$$m_R(0, \vec{z}) = 0$$

$$m_R(y + 1, \vec{z}) = \begin{cases} m_R(y, \vec{z}) & \text{if } (\exists x < y)\ R(x, \vec{z}) \\ y & \text{otherwise, provided } R(y, \vec{z}) \\ 0 & \text{otherwise.} \end{cases}$$

□

The choice of "0 otherwise" is somewhat arbitrary. It is in fact even easier to recursively define the function m'_R which returns the least x less than y such that $R(x, \vec{z})$ holds, and $y + 1$ otherwise. When we use min, however, we will always know that the least x such that $R(x, \vec{z})$ exists and is less than y. Thus, in practice, we will not have to worry about the possibility that if $(\min x < y)\, R(x, \vec{z}) = 0$ we do not know if that value indicates that $R(0, \vec{z})$ or that for no $x < y$, $R(x, \vec{z})$. As with bounded quantification, $(\min x \leq y) \ldots$ can be understood as $(\min x < y + 1) \ldots$.

12.7 Primes

Bounded quantification and bounded minimization provide us with a good deal of machinery to show that natural functions and relations are primitive recursive. For example, consider the relation relation "x divides y", written $x \mid y$. $x \mid y$ holds if division of x by y is possible without remainder, i.e., if y is an integer multiple of x. (If it doesn't hold, i.e., the remainder when dividing x by y is > 0, we write $x \nmid y$.) In other words, $x \mid y$ iff for some z, $x \cdot z = y$. Obviously, any such z, if it exists, must be $\leq y$. So, we have that $x \mid y$ iff for some $z \leq y$, $x \cdot z = y$. We can define the relation $x \mid y$ by bounded existential quantification from $=$ and multiplication by

$$x \mid y \Leftrightarrow (\exists z \leq y)\ (x \cdot z) = y.$$

We've thus shown that $x \mid y$ is primitive recursive.

A natural number x is *prime* if it is neither 0 nor 1 and is only divisible by 1 and itself. In other words, prime numbers are such that, whenever $y \mid x$, either $y = 1$ or $y = x$. To test if x is prime, we only have to check if $y \mid x$ for all $y \leq x$, since if $y > x$, then automatically $y \nmid x$. So, the relation $\text{Prime}(x)$, which holds iff x is prime, can be defined by

$$\text{Prime}(x) \Leftrightarrow x \geq 2 \,\&\, (\forall y \leq x)\ (y \mid x \supset y = 1 \vee y = x)$$

and is thus primitive recursive.

The primes are 2, 3, 5, 7, 11, etc. Consider the function $p(x)$ which returns the xth prime in that sequence, i.e., $p(0) = 2$, $p(1) = 3$, $p(2) = 5$, etc. (For convenience we will often write $p(x)$ as p_x ($p_0 = 2$, $p_1 = 3$, etc.)

If we had a function nextPrime(x), which returns the first prime number larger than x, p can be easily defined using primitive recursion:

$$\begin{aligned} p(0) &= 2 \\ p(x+1) &= \text{nextPrime}(p(x)) \end{aligned}$$

Since $\text{nextPrime}(x)$ is the least y such that $y > x$ and y is prime, it can be easily computed by unbounded search. But it can also be defined by bounded minimization, thanks to a result due to Euclid: there is always a prime number between x and $x! + 1$.

$$\text{nextPrime(x)} = (\min y \leq x! + 1)\ (y > x \,\&\, \text{Prime}(y)).$$

This shows, that $\text{nextPrime}(x)$ and hence $p(x)$ are (not just computable but) primitive recursive.

(If you're curious, here's a quick proof of Euclid's theorem. Suppose p_n is the largest prime $\leq x$ and consider the product $p = p_0 \cdot p_1 \cdot \cdots \cdot p_n$ of all primes $\leq x$. Either $p + 1$ is prime or there is a prime between x and $p + 1$. Why? Suppose $p + 1$ is not prime. Then some prime number $q \mid p + 1$ where $q < p + 1$. None of the primes $\leq x$ divide $p + 1$. (By definition of p, each of the primes $p_i \leq x$ divides p, i.e., with remainder 0. So, each of the primes $p_i \leq x$ divides $p + 1$ with remainder 1, and so $p_i \nmid p + 1$.) Hence, q is a prime $> x$ and $< p + 1$. And $p \leq x!$, so there is a prime $> x$ and $\leq x! + 1$.)

12.8 Sequences

The set of primitive recursive functions is remarkably robust. But we will be able to do even more once we have developed an adequate means of handling *sequences*. We will identify finite sequences of natural numbers with natural numbers in the following way: the sequence $\langle a_0, a_1, a_2, \ldots, a_k \rangle$ corresponds to the number

$$p_0^{a_0+1} \cdot p_1^{a_1+1} \cdot p_2^{a_2+1} \cdot \cdots \cdot p_k^{a_k+1}.$$

We add one to the exponents to guarantee that, for example, the sequences $\langle 2, 7, 3\rangle$ and $\langle 2, 7, 3, 0, 0\rangle$ have distinct numeric codes. We can take both 0 and 1 to code the empty sequence; for concreteness, let $\emptyset$ denote 0.

Let us define the following functions:

1. $\text{len}(s)$, which returns the length of the sequence s: Let $R(i, s)$ be the relation defined by

$$R(i, s) \text{ iff } p_i \mid s \,\&\, (\forall j < s)\ (j > i \supset p_j \not\mid s)$$

   R is primitive recursive. Now let

$$\text{len}(s) = \begin{cases} 0 & \text{if } s = 0 \text{ or } s = 1 \\ 1 + (\min i < s)\, R(i, s) & \text{otherwise} \end{cases}$$

   Note that we need to bound the search on i; clearly s provides an acceptable bound.

2. $\text{append}(s, a)$, which returns the result of appending a to the sequence s:

$$\text{append}(s, a) = \begin{cases} 2^{a+1} & \text{if } s = 0 \text{ or } s = 1 \\ s \cdot p_{\text{len}(s)}^{a+1} & \text{otherwise} \end{cases}$$

3. $\text{element}(s, i)$, which returns the ith element of s (where the initial element is called the 0th), or 0 if i is greater than or equal to the length of s:

$$\text{element}(s, i) = \begin{cases} 0 & \text{if } i \geq \text{len}(s) \\ \min j < s\ (p_i^{j+2} \not\mid s) - 1 & \text{otherwise} \end{cases}$$

Instead of using the official names for the functions defined above, we introduce a more compact notation. We will use $(s)_i$ instead of $\text{element}(s, i)$, and $\langle s_0, \ldots, s_k\rangle$ to abbreviate

$$\text{append}(\text{append}(\ldots \text{append}(\emptyset, s_0) \ldots), s_k).$$

Note that if s has length k, the elements of s are $(s)_0, \ldots, (s)_{k-1}$.

It will be useful for us to be able to bound the numeric code of a sequence in terms of its length and its largest element. Suppose s is a sequence of length k, each element of which is less than equal to some number x. Then s has at most k prime factors, each at most p_{k-1}, and each raised to at most $x + 1$ in the prime factorization of s. In other words, if we define

$$\text{sequenceBound}(x, k) = p_{k-1}^{k\cdot(x+1)},$$

then the numeric code of the sequence s described above is at most $\text{sequenceBound}(x, k)$.

Having such a bound on sequences gives us a way of defining new functions using bounded search. For example, suppose we want to define the function concat(s,t), which concatenates two sequences. One first option is to define a "helper" function hconcat(s,t,n) which concatenates the first n symbols of t to s. This function can be defined by primitive recursion, as follows:

$$\begin{aligned}\text{hconcat}(s,t,0) &= s\\ \text{hconcat}(s,t,n+1) &= \text{append}(\text{hconcat}(s,t,n),(t)_n)\end{aligned}$$

Then we can define concat by

$$\text{concat}(s,t) = \text{hconcat}(s,t,\text{len}(t)).$$

But using bounded search, we can be lazy. All we need to do is write down a primitive recursive *specification* of the object (number) we are looking for, and a bound on how far to look. The following works:

$$\begin{aligned}\text{concat}(s,t) = (&\min v < \text{sequenceBound}(s+t, \text{len}(s)+\text{len}(t)))\\ &(\text{len}(v) = \text{len}(s)+\text{len}(t)\ \&\\ &\quad(\forall i < \text{len}(s))\ ((v)_i = (s)_i)\ \&\\ &\quad(\forall j < \text{len}(t))\ ((v)_{\text{len}(s)+j} = (t)_j))\end{aligned}$$

We will write $s \frown t$ instead of concat(s,t).

12.9 Other Recursions

Using pairing and sequencing, we can justify more exotic (and useful) forms of primitive recursion. For example, it is often useful to define two functions simultaneously, such as in the following definition:

$$\begin{aligned}f_0(0,\vec{z}) &= k_0(\vec{z})\\ f_1(0,\vec{z}) &= k_1(\vec{z})\\ f_0(x+1,\vec{z}) &= h_0(x, f_0(x,\vec{z}), f_1(x,\vec{z}),\vec{z})\\ f_1(x+1,\vec{z}) &= h_1(x, f_0(x,\vec{z}), f_1(x,\vec{z}),\vec{z})\end{aligned}$$

This is an instance of *simultaneous recursion*. Another useful way of defining functions is to give the value of $f(x+1,\vec{z})$ in terms of *all* the values $f(0,\vec{z})$, $\ldots$, $f(x,\vec{z})$, as in the following definition:

$$\begin{aligned}f(0,\vec{z}) &= g(\vec{z})\\ f(x+1,\vec{z}) &= h(x,\langle f(0,\vec{z}),\ldots,f(x,\vec{z})\rangle,\vec{z}).\end{aligned}$$

The following schema captures this idea more succinctly:

$$f(x,\vec{z}) = h(x,\langle f(0,\vec{z}),\ldots,f(x-1,\vec{z})\rangle)$$

with the understanding that the second argument to h is just the empty sequence when x is 0. In either formulation, the idea is that in computing the "successor step," the function f can make use of the entire sequence of values computed so far. This is known as a *course-of-values* recursion. For a particular example, it can be used to justify the following type of definition:

$$f(x, \vec{z}) = \begin{cases} h(x, f(k(x, \vec{z}), \vec{z}), \vec{z}) & \text{if } k(x, \vec{z}) < x \\ g(x, \vec{z}) & \text{otherwise} \end{cases}$$

In other words, the value of f at x can be computed in terms of the value of f at *any* previous value, given by k.

You should think about how to obtain these functions using ordinary primitive recursion. One final version of primitive recursion is more flexible in that one is allowed to change the *parameters* (side values) along the way:

$$\begin{aligned} f(0, \vec{z}) &= g(\vec{z}) \\ f(x+1, \vec{z}) &= h(x, f(x, k(\vec{z})), \vec{z}) \end{aligned}$$

This, too, can be simulated with ordinary primitive recursion. (Doing so is tricky. For a hint, try unwinding the computation by hand.)

Finally, notice that we can always extend our "universe" by defining additional objects in terms of the natural numbers, and defining primitive recursive functions that operate on them. For example, we can take an integer to be given by a pair $\langle m, n \rangle$ of natural numbers, which, intuitively, represents the integer $m - n$. In other words, we say

$$\text{Integer}(x) \Leftrightarrow \text{length}(x) = 2$$

and then we define the following:

1. $\text{iequal}(x, y)$
2. $\text{iplus}(x, y)$
3. $\text{iminus}(x, y)$
4. $\text{itimes}(x, y)$

Similarly, we can define a rational number to be a pair $\langle x, y \rangle$ of integers with $y \neq 0$, representing the value x/y. And we can define qequal, qplus, qminus, qtimes, qdivides, and so on.

12.10 Non-Primitive Recursive Functions

The primitive recursive functions do not exhaust the intuitively computable functions. It should be intuitively clear that we can make a list of all the unary

primitive recursive functions, f_0, f_1, f_2, ... such that we can effectively compute the value of f_x on input y; in other words, the function $g(x, y)$, defined by

$$g(x, y) = f_x(y)$$

is computable. But then so is the function

$$\begin{aligned} h(x) &= g(x, x) + 1 \\ &= f_x(x) + 1. \end{aligned}$$

For each primitive recursive function f_i, the value of h and f_i differ at i. So h is computable, but not primitive recursive; and one can say the same about g. This is a an "effective" version of Cantor's diagonalization argument.

One can provide more explicit examples of computable functions that are not primitive recursive. For example, let the notation $g^n(x)$ denote $g(g(\ldots g(x)))$, with n g's in all; and define a sequence $g_0, g_1, \ldots$ of functions by

$$\begin{aligned} g_0(x) &= x + 1 \\ g_{n+1}(x) &= g_n^x(x) \end{aligned}$$

You can confirm that each function g_n is primitive recursive. Each successive function grows much faster than the one before; $g_1(x)$ is equal to $2x$, $g_2(x)$ is equal to $2^x \cdot x$, and $g_3(x)$ grows roughly like an exponential stack of x 2's. Ackermann's function is essentially the function $G(x) = g_x(x)$, and one can show that this grows faster than any primitive recursive function.

Let us return to the issue of enumerating the primitive recursive functions. Remember that we have assigned symbolic notations to each primitive recursive function; so it suffices to enumerate notations. We can assign a natural number $\#(F)$ to each notation F, recursively, as follows:

$$\begin{aligned} \#(0) &= \langle 0 \rangle \\ \#(S) &= \langle 1 \rangle \\ \#(P_i^n) &= \langle 2, n, i \rangle \\ \#(\mathrm{Comp}_{k,l}[H, G_0, \ldots, G_{k-1}]) &= \langle 3, k, l, \#(H), \#(G_0), \ldots, \#(G_{k-1}) \rangle \\ \#(\mathrm{Rec}_l[G, H]) &= \langle 4, l, \#(G), \#(H) \rangle \end{aligned}$$

Here I am using the fact that every sequence of numbers can be viewed as a natural number, using the codes from the last section. The upshot is that every code is assigned a natural number. Of course, some sequences (and hence some numbers) do not correspond to notations; but we can let f_i be the unary primitive recursive function with notation coded as i, if i codes such a notation; and the constant 0 function otherwise. The net result is that we have an explicit way of enumerating the unary primitive recursive functions.

(In fact, some functions, like the constant zero function, will appear more than once on the list. This is not just an artifact of our coding, but also a result

of the fact that the constant zero function has more than one notation. We will later see that one can not computably avoid these repetitions; for example, there is no computable function that decides whether or not a given notation represents the constant zero function.)

We can now take the function $g(x, y)$ to be given by $f_x(y)$, where f_x refers to the enumeration we have just described. How do we know that $g(x, y)$ is computable? Intuitively, this is clear: to compute $g(x, y)$, first "unpack" x, and see if it a notation for a unary function; if it is, compute the value of that function on input y.

You may already be convinced that (with some work!) one can write a program (say, in Java or C++) that does this; and now we can appeal to the Church-Turing thesis, which says that anything that, intuitively, is computable can be computed by a Turing machine.

Of course, a more direct way to show that $g(x, y)$ is computable is to describe a Turing machine that computes it, explicitly. This would, in particular, avoid the Church-Turing thesis and appeals to intuition. But, as noted above, working with Turing machines directly is unpleasant. Soon we will have built up enough machinery to show that $g(x, y)$ is computable, appealing to a model of computation that can be *simulated* on a Turing machine: namely, the recursive functions.

12.11 Partial Recursive Functions

To motivate the definition of the recursive functions, note that our proof that there are computable functions that are not primitive recursive actually establishes much more. The argument was simple: all we used was the fact was that it is possible to enumerate functions $f_0, f_1, \ldots$ such that, as a function of x and y, $f_x(y)$ is computable. So the argument applies to *any class of functions that can be enumerated in such a way*. This puts us in a bind: we would like to describe the computable functions explicitly; but any explicit description of a collection of computable functions cannot be exhaustive!

The way out is to allow *partial* functions to come into play. We will see that it *is* possible to enumerate the partial computable functions. In fact, we already pretty much know that this is the case, since it is possible to enumerate Turing machines in a systematic way. We will come back to our diagonal argument later, and explore why it does not go through when partial functions are included.

The question is now this: what do we need to add to the primitive recursive functions to obtain all the partial recursive functions? We need to do two things:

1. Modify our definition of the primitive recursive functions to allow for partial functions as well.

2. *Add* something to the definition, so that some new partial functions are included.

The first is easy. As before, we will start with zero, successor, and projections, and close under composition and primitive recursion. The only difference is that we have to modify the definitions of composition and primitive recursion to allow for the possibility that some of the terms in the definition are not defined. If f and g are partial functions, we will write $f(x) \downarrow$ to mean that f is defined at x, i.e., x is in the domain of f; and $f(x) \uparrow$ to mean the opposite, i.e., that f is not defined at x. We will use $f(x) \simeq g(x)$ to mean that either $f(x)$ and $g(x)$ are both undefined, or they are both defined and equal. We will use these notations for more complicated terms as well. We will adopt the convention that if h and $g_0, \ldots, g_k$ all are partial functions, then

$$h(g_0(\vec{x}), \ldots, g_k(\vec{x}))$$

is defined if and only if each g_i is defined at $\vec{x}$, and h is defined at $g_0(\vec{x})$, $\ldots$, $g_k(\vec{x})$. With this understanding, the definitions of composition and primitive recursion for partial functions is just as above, except that we have to replace "$=$" by "$\simeq$".

What we will add to the definition of the primitive recursive functions to obtain partial functions is the *unbounded search operator*. If $f(x, \vec{z})$ is any partial function on the natural numbers, define $\mu x\ f(x, \vec{z})$ to be

> the least x such that $f(0, \vec{z}), f(1, \vec{z}), \ldots, f(x, \vec{z})$ are all defined, and $f(x, \vec{z}) = 0$, if such an x exists

with the understanding that $\mu x\ f(x, \vec{z})$ is undefined otherwise. This defines $\mu x\ f(x, \vec{z})$ uniquely.

Note that our definition makes no reference to Turing machines, or algorithms, or any specific computational model. But like composition and primitive recursion, there is an operational, computational intuition behind unbounded search. When it comes to the computability of a partial function, arguments where the function is undefined correspond to inputs for which the computation does not halt. The procedure for computing $\mu x\ f(x, \vec{z})$ will amount to this: compute $f(0, \vec{z}), f(1, \vec{z}), f(2, \vec{z})$ until a value of 0 is returned. If any of the intermediate computations do not halt, however, neither does the computation of $\mu x\ f(x, \vec{z})$.

If $R(x, \vec{z})$ is any relation, $\mu x\ R(x, \vec{z})$ is defined to be $\mu x\ (1 \dot{-} \chi_R(x, \vec{z}))$. In other words, $\mu x\ R(x, \vec{z})$ returns the least value of x such that $R(x, \vec{z})$ holds. So, if $f(x, \vec{z})$ is a total function, $\mu x\ f(x, \vec{z})$ is the same as $\mu x\ (f(x, \vec{z}) = 0)$. But note that our original definition is more general, since it allows for the possibility that $f(x, \vec{z})$ is not everywhere defined (whereas, in contrast, the characteristic function of a relation is always total).

Definition 12.6. The set of *partial recursive functions* is the smallest set of partial functions from the natural numbers to the natural numbers (of various arities) containing zero, successor, and projections, and closed under composition, primitive recursion, and unbounded search.

Of course, some of the partial recursive functions will happen to be total, i.e., defined for every argument.

Definition 12.7. The set of *recursive functions* is the set of partial recursive functions that are total.

A recursive function is sometimes called "total recursive" to emphasize that it is defined everywhere.

12.12 The Normal Form Theorem

Theorem 12.8 (Kleene's Normal Form Theorem)**.** *There is a primitive recursive relation $T(e, x, s)$ and a primitive recursive function $U(s)$, with the following property: if f is any partial recursive function, then for some e,*

$$f(x) \simeq U(\mu s\ T(e, x, s))$$

for every x.

The proof of the normal form theorem is involved, but the basic idea is simple. Every partial recursive function has an *index* e, intuitively, a number coding its program or definition. If $f(x) \downarrow$, the computation can be recorded systematically and coded by some number s, and that s codes the computation of f on input x can be checked primitive recursively using only x and the definition e. This means that T is primitive recursive. Given the full record of the computation s, the "upshot" of s is the value of $f(x)$, and it can be obtained from s primitive recursively as well.

The normal form theorem shows that only a single unbounded search is required for the definition of any partial recursive function. We can use the numbers e as "names" of partial recursive functions, and write φ_e for the function f defined by the equation in the theorem. Note that any partial recursive function can have more than one index—in fact, every partial recursive function has infinitely many indices.

12.13 The Halting Problem

The *halting problem* in general is the problem of deciding, given the specification e (e.g., program) of a computable function and a number n, whether the computation of the function on input n halts, i.e., produces a result. Famously,

Alan Turing proved that this problem itself cannot be solved by a computable function, i.e., the function

$$h(e,n) = \begin{cases} 1 & \text{if computation } e \text{ halts on input } n \\ 0 & \text{otherwise,} \end{cases}$$

is not computable.

In the context of partial recursive functions, the role of the specification of a program may be played by the index e given in Kleene's normal form theorem. If f is a partial recursive function, any e for which the equation in the normal form theorem holds, is an index of f. Given a number e, the normal form theorem states that

$$\varphi_e(x) \simeq U(\mu s\ T(e,x,s))$$

is partial recursive, and for every partial recursive $f\colon \mathbb{N} \to \mathbb{N}$, there is an $e \in \mathbb{N}$ such that $\varphi_e(x) \simeq f(x)$ for all $x \in \mathbb{N}$. In fact, for each such f there is not just one, but infinitely many such e. The *halting function* h is defined by

$$h(e,x) = \begin{cases} 1 & \text{if } \varphi_e(x) \downarrow \\ 0 & \text{otherwise.} \end{cases}$$

Note that $h(e,x) = 0$ if $\varphi_e(x) \uparrow$, but also when e is not the index of a partial recursive function at all.

Theorem 12.9. *The halting function h is not partial recursive.*

Proof. If h were partial recursive, we could define

$$d(y) = \begin{cases} 1 & \text{if } h(y,y) = 0 \\ \mu x\ x \neq x & \text{otherwise.} \end{cases}$$

From this definition it follows that

1. $d(y) \downarrow$ iff $\varphi_y(y) \uparrow$ or y is not the index of a partial recursive function.
2. $d(y) \uparrow$ iff $\varphi_y(y) \downarrow$.

If h were partial recursive, then d would be partial recursive as well. Thus, by the Kleene normal form theorem, it has an index e_d. Consider the value of $h(e_d, e_d)$. There are two possible cases, 0 and 1.

1. If $h(e_d, e_d) = 1$ then $\varphi_{e_d}(e_d) \downarrow$. But $\varphi_{e_d} \simeq d$, and $d(e_d)$ is defined iff $h(e_d, e_d) = 0$. So $h(e_d, e_d) \neq 1$.
2. If $h(e_d, e_d) = 0$ then either e_d is not the index of a partial recursive function, or it is and $\varphi_{e_d}(e_d) \uparrow$. But again, $\varphi_{e_d} \simeq d$, and $d(e_d)$ is undefined iff $\varphi_{e_d}(e_d) \downarrow$.

The upshot is that e_d cannot, after all, be the index of a partial recursive function. But if h were partial recursive, d would be too, and so our definition of e_d as an index of it would be admissible. We must conclude that h cannot be partial recursive. □

12.14 General Recursive Functions

There is another way to obtain a set of total functions. Say a total function $f(x, \vec{z})$ is *regular* if for every sequence of natural numbers $\vec{z}$, there is an x such that $f(x, \vec{z}) = 0$. In other words, the regular functions are exactly those functions to which one can apply unbounded search, and end up with a total function. One can, conservatively, restrict unbounded search to regular functions:

Definition 12.10. The set of *general recursive functions* is the smallest set of functions from the natural numbers to the natural numbers (of various arities) containing zero, successor, and projections, and closed under composition, primitive recursion, and unbounded search applied to *regular* functions.

Clearly every general recursive function is total. The difference between Definition 12.10 and Definition 12.7 is that in the latter one is allowed to use partial recursive functions along the way; the only requirement is that the function you end up with at the end is total. So the word "general," a historic relic, is a misnomer; on the surface, Definition 12.10 is *less* general than Definition 12.7. But, fortunately, the difference is illusory; though the definitions are different, the set of general recursive functions and the set of recursive functions are one and the same.

Chapter 13

Arithmetization of Syntax

13.1 Introduction

In order to connect computability and logic, we need a way to talk about the objects of logic (symbols, terms, formulae, derivations), operations on them, and their properties and relations, in a way amenable to computational treatment. We can do this directly, by considering computable functions and relations on symbols, sequences of symbols, and other objects built from them. Since the objects of logical syntax are all finite and built from a countable sets of symbols, this is possible for some models of computation. But other models of computation—such as the recursive functions—-are restricted to numbers, their relations and functions. Moreover, ultimately we also want to be able to deal with syntax within certain theories, specifically, in theories formulated in the language of arithmetic. In these cases it is necessary to *arithmetize* syntax, i.e., to represent syntactic objects, operations on them, and their relations, as numbers, arithmetical functions, and arithmetical relations, respectively. The idea, which goes back to Leibniz, is to assign numbers to syntactic objects.

It is relatively straightforward to assign numbers to symbols as their "codes." Some symbols pose a bit of a challenge, since, e.g., there are infinitely many variables, and even infinitely many function symbols of each arity n. But of course it's possible to assign numbers to symbols systematically in such a way that, say, v_2 and v_3 are assigned different codes. Sequences of symbols (such as terms and formulae) are a bigger challenge. But if can deal with sequences of numbers purely arithmetically (e.g., by the powers-of-primes coding of sequences), we can extend the coding of individual symbols to coding of sequences of symbols, and then further to sequences or other arrangements of formulae, such as derivations. This extended coding is called "Gödel numbering." Every term, formula, and derivation is assigned a Gödel number.

By coding sequences of symbols as sequences of their codes, and by chosing a system of coding sequences that can be dealt with using computable functions, we can then also deal with Gödel numbers usign computable func-

tions. In practice, all the relevant functions will be primitive recursive. For instance, computing the length of a sequence and computing the i-th element of a sequence from the code of the sequence are both primitive recursive. If the number coding the sequence is, e.g., the Gödel number of a formula φ, we immediately see that the length of a formula and the (code of the) i-th symbol in a formula can also be computed from the Gödel number of φ. It is a bit harder to prove that, e.g., the property of being the Gödel number of a correctly formed term, of being the Gödel number of a corret derivation is primitive recursive. It is nevertheless possible, because the sequences of interest (terms, formulae, derivations) are inductively defined.

As an example, consider the operation of substitution. If φ is a formula, x a variable, and t a term, then $\varphi[t/x]$ is the result of replacing every free occurrence of x in φ by t. Now suppose we have assigned Gödel numbers to φ, x, t—say, k, l, and m, respectively. The same scheme assignes a Gödel number to $[t/x]$, say, n. This mapping—of k, l, m to n—is the arithmetical analog of the substitution operation. When the substitution operation maps φ, x, t to $\varphi[t/x]$, the arithmetized substitution functions maps the Gödel numbers k, l, m to the Gödel number n. We will see that this function is primitive recursive.

Arithmetization of syntax is not just of abstract interest, although it was originally a non-trivial insight that languages like the language of arithmetic, which do not come with mechanisms for "talking about" languages can, after all, formalize complex properties of expressions. It is then just a small step to ask what a theory in this language, such as Peano arithmetic, can *prove* about its own language (including, e.g., whether sentences are provable or true). This leads us to the famous limitative theorems of Gödel (about unprovability) and Tarski (the undefinability of truth). But the trick of arithmetizing syntax is also important in order to prove some important results in computability theory, e.g., about the computational prower of theories or the relationship between different models of computability. The arithmetization of syntax serves as a model for arithmetizing other objects and properties. For instance, it is similarly possible to arithmetize configurations and computations (say, of Turing machines). This makes it possible to simulate computations in one model (e.g., Turing machines) in another (e.g., recursive functions).

13.2 Coding Symbols

The basic language $\mathcal{L}$ of first order logic makes use of the symbols

$$\bot \quad \sim \quad \vee \quad \& \quad \supset \quad \forall \quad \exists \quad = \quad (\quad) \quad ,$$

together with countable sets of variables and constant symbols, and countable sets of function symbols and predicate symbols of arbitrary arity. We can assign *codes* to each of these symbols in such a way that every symbol is assigned a unique number as its code, and no two different symbols are assigned the

same number. We know that this is possible since the set of all symbols is countable and so there is a bijection between it and the set of natural numbers. But we want to make sure that we can recover the symbol (as well as some information about it, e.g., the arity of a function symbol) from its code in a computable way. There are many possible ways of doing this, of course. Here is one such way, which uses primitive recursive functions. (Recall that $\langle n_0, \ldots, n_k \rangle$ is the number coding the sequence of numbers $n_0, \ldots, n_k$.)

Definition 13.1. If s is a symbol of $\mathcal{L}$, let the *symbol code* c_s be defined as follows:

1. If s is among the logical symbols, c_s is given by the following table:

$$\begin{array}{cccccc} \bot & \sim & \vee & \& & \supset & \forall \\ \langle 0,0\rangle & \langle 0,1\rangle & \langle 0,2\rangle & \langle 0,3\rangle & \langle 0,4\rangle & \langle 0,5\rangle \\ \exists & = & (&) & , & \\ \langle 0,6\rangle & \langle 0,7\rangle & \langle 0,8\rangle & \langle 0,9\rangle & \langle 0,10\rangle & \end{array}$$

2. If s is the i-th variable v_i, then $\mathrm{c}_s = \langle 1, i\rangle$.

3. If s is the i-th constant symbol c_i^n, then $\mathrm{c}_s = \langle 2, i\rangle$.

4. If s is the i-th n-ary function symbol f_i^n, then $\mathrm{c}_s = \langle 3, n, i\rangle$.

5. If s is the i-th n-ary predicate symbol P_i^n, then $\mathrm{c}_s = \langle 4, n, i\rangle$.

Proposition 13.2. *The following relations are primitive recursive:*

1. $\mathrm{Fn}(x, n)$ *iff* x *is the code of* f_i^n *for some* i*, i.e.,* x *is the code of an* n*-ary function symbol.*

2. $\mathrm{Pred}(x, n)$ *iff* x *is the code of* P_i^n *for some* i *or* x *is the code of* $=$ *and* $n = 2$*, i.e.,* x *is the code of an* n*-ary predicate symbol.*

Definition 13.3. If $s_0, \ldots, s_{n-1}$ is a sequence of symbols, its *Gödel number* is $\langle \mathrm{c}_{s_0}, \ldots, \mathrm{c}_{s_{n-1}} \rangle$.

Note that *codes* and *Gödel numbers* are different things. For instance, the variable v_5 has a code $\mathrm{c}_{v_5} = \langle 1, 5\rangle = 2^2 \cdot 3^6$. But the variable v_5 considered as a term is also a sequence of symbols (of length 1). The *Gödel number* $\ulcorner v_5 \urcorner$ of the *term* v_5 is $\langle \mathrm{c}_{v_5} \rangle = 2^{\mathrm{c}_{v_5}+1} = 2^{2^2 \cdot 3^6 + 1}$.

Example 13.4. Recall that if $k_0, \ldots, k_{n-1}$ is a sequence of numbers, then the code of the sequence $\langle k_0, \ldots, k_{n-1}\rangle$ in the power-of-primes coding is

$$2^{k_0+1} \cdot 3^{k_1+1} \cdot \ldots \cdot p_{n-1}^{k_{n-1}},$$

where p_i is the i-th prime (starting with $p_0 = 2$). So for instance, the formula $v_0 = 0$, or, more explicitly, $=(v_0, c_0)$, has the Gödel number

$$\langle \mathrm{c}_=, \mathrm{c}_(, \mathrm{c}_{v_0}, \mathrm{c}_,, \mathrm{c}_{c_0}, \mathrm{c}_) \rangle.$$

Here, $\mathrm{c}_=$ is $\langle 0, 7 \rangle = 2^{0+1} \cdot 3^{7=1}$, c_{v_0} is $\langle 1, 0 \rangle = 2^{1+1} \cdot 3^{0+1}$, etc. So ${}^\#{=}(v_0, c_0)^\#$ is

$$\begin{aligned} &2^{\mathrm{c}_=+1} \cdot 3^{\mathrm{c}_(+1} \cdot 5^{\mathrm{c}_{v_0}+1} \cdot 7^{\mathrm{c}_,+1} \cdot 11^{\mathrm{c}_{c_0}+1} \cdot 13^{\mathrm{c}_)+1} = \\ &\qquad 2^{2^1 \cdot 3^8+1} \cdot 3^{2^1 \cdot 3^9+1} \cdot 5^{2^2 \cdot 3^1+1} \cdot 7^{2^1 \cdot 3^{11}+1} \cdot 11^{2^3 \cdot 3^1+1} \cdot 13^{2^1 \cdot 3^{10}+1} = \\ &\qquad\qquad 2^{13\,123} \cdot 3^{39\,367} \cdot 5^{13} \cdot 7^{354\,295} \cdot 11^{25} \cdot 13^{118\,099}. \end{aligned}$$

13.3 Coding Terms

A term is simply a certain kind of sequence of symbols: it is built up inductively from constants and variables according to the formation rules for terms. Since sequences of symbols can be coded as numbers—using a coding scheme for the symbols plus a way to code sequences of numbers—assigning Gödel numbers to terms is not difficult. The challenge is rather to show that the property a number has if it is the Gödel number of a correctly formed term is computable, or in fact primitive recursive.

Proposition 13.5. *The relations* $\mathrm{Term}(x)$ *and* $\mathrm{ClTerm}(x)$ *which hold iff x is the Gödel number of a term or a closed term, respectively, are primitive recursive.*

Proof. A sequence of symbols s is a term iff there is a sequence $s_0, \ldots, s_{k-1} = s$ of terms which records how the term s was formed from constant symbols and variables according to the formation rules for terms. To express that such a putative formation sequence follows the formation rules it has to be the case that, for each $i < k$, either

1. s_i is a variable v_j, or
2. s_i is a constant symbol c_j, or
3. s_i is built from n terms $t_1, \ldots, t_n$ occurring prior to place i using an n-place function symbol f^n_j.

To show that the corresponding relation on Gödel numbers is primitive recursive, we have to express this condition primitive recursively, i.e., using primitive recursive functions, relations, and bounded quantification.

Suppose y is the number that codes the sequence $s_0, \ldots, s_{k-1}$, i.e., $y = \langle {}^\#s_0{}^\#, \ldots, {}^\#s_k{}^\# \rangle$. It codes a formation sequence for the term with Gödel number x iff for all $i < k$:

1. there is a j such that $(y)_i = {}^\#v_j{}^\#$, or

2. there is a j such that $(y)_i = {}^{\#}c_j{}^{\#}$, or

3. there is an n and a number $z = \langle z_1, \ldots, z_n \rangle$ such that each z_l is equal to some $(y)_{i'}$ for $i' < i$ and

$$(y)_i = {}^{\#}f_j^n({}^{\#} \frown \text{flatten}(z) \frown {}^{\#})^{\#},$$

and moreover $(y)_{k-1} = x$. The function flatten(z) turns the sequence $\langle {}^{\#}t_1{}^{\#}, \ldots, {}^{\#}t_n{}^{\#} \rangle$ into ${}^{\#}t_1, \ldots, t_n{}^{\#}$ and is primitive recursive.

The indices j, n, the Gödel numbers z_l of the terms t_l, and the code z of the sequence $\langle z_1, \ldots, z_n \rangle$, in (3) are all less than y. We can replace k above with $\text{len}(y)$. Hence we can express "y is the code of a formation sequence of the term with Gödel number x" in a way that shows that this relation is primitive recursive.

We now just have to convince ourselves that there is a primitive recursive bound on y. But if x is the Gödel number of a term, it must have a formation sequence with at most $\text{len}(x)$ terms (since every term in the formation sequence of s must start at some place in s, and no two subterms can start at the same place). The Gödel number of each subterm of s is of course $\leq x$. Hence, there always is a formation sequence with code $\leq x^{\text{len}(x)}$.

For ClTerm, simply leave out the clause for variables. □

Alternative proof of Proposition 13.5. The inductive definition says that constant symbols and variables are terms, and if $t_1, \ldots, t_n$ are terms, then so is $f_j^n(t_1, \ldots, t_n)$, for any n and j. So terms are formed in stages: constant symbols and variables at stage 0, terms involving one function symbol at stage 1, those involving at least two nested function symbols at stage 2, etc. Let's say that a sequence of symbols s is a term of level l iff s can be formed by applying the inductive definition of terms l (or fewer) times, i.e., it "becomes" a term by stage l or before. So s is a term of level $l + 1$ iff

1. s is a variable v_j, or

2. s is a constant symbol c_j, or

3. s is built from n terms $t_1, \ldots, t_n$ of level l and an n-place function symbol f_j^n.

To show that the corresponding relation on Gödel numbers is primitive recursive, we have to express this condition primitive recursively, i.e., using primitive recursive functions, relations, and bounded quantification.

The number x is the Gödel number of a term s of level $l + 1$ iff

1. there is a j such that $x = {}^{\#}v_j{}^{\#}$, or

2. there is a j such that $x = {}^{\#}c_j{}^{\#}$, or

3. there is an n, a j, and a number $z = \langle z_1, \ldots, z_n \rangle$ such that each z_i is the Gödel number of a term of level l and

$$x = {}^{\ulcorner}f_j^n({}^{\ulcorner} \frown \text{flatten}(z) \frown {}^{\urcorner})^{\urcorner},$$

and moreover $(y)_{k-1} = x$.

The indices j, n, the Gödel numbers z_i of the terms t_i, and the code z of the sequence $\langle z_1, \ldots, z_n \rangle$, in (3) are all less than x. So we get a primitive recursive definition by:

$$\begin{aligned}
\text{lTerm}(x, 0) &= \text{Var}(x) \vee \text{Const}(x) \\
\text{lTerm}(x, l+1) &= \text{Var}(x) \vee \text{Const}(x) \vee \\
&\quad (\exists z < x)\ ((\forall i < \text{len}(z))\ \text{lTerm}((z)_i, l)\ \&\ \\
&\quad\quad (\exists j < x)\ x = ({}^{\ulcorner}f_j^{\text{len}(z)}({}^{\ulcorner} \frown \text{flatten}(z) \frown {}^{\urcorner})^{\urcorner}))
\end{aligned}$$

We can now define $\text{Term}(x)$ by $\text{lTerm}(x, x)$, since the level of a term is always less than the Gödel number of the term. □

Proposition 13.6. *The function* $\text{num}(n) = {}^{\ulcorner}\overline{n}^{\urcorner}$ *is primitive recursive.*

Proof. We define $\text{num}(n)$ by primitive recursion:

$$\begin{aligned}
\text{num}(0) &= {}^{\ulcorner}\mathsf{o}^{\urcorner} \\
\text{num}(n+1) &= {}^{\ulcorner}\prime({}^{\ulcorner} \frown \text{num}(n) \frown {}^{\urcorner})^{\urcorner}.
\end{aligned}$$

□

13.4 Coding Formulae

Proposition 13.7. *The relation* $\text{Atom}(x)$ *which holds iff x is the Gödel number of an atomic formula, is primitive recursive.*

Proof. The number x is the Gödel number of an atomic formula iff one of the following holds:

1. There are n, $j < x$, and $z < x$ such that for each $i < n$, $\text{Term}((z)_i)$ and $x =$
$$^{\ulcorner}P_j^n({}^{\ulcorner} \frown \text{flatten}(z) \frown {}^{\urcorner})^{\urcorner}.$$

2. There are $z_1, z_2 < x$ such that $\text{Term}(z_1)$, $\text{Term}(z_2)$, and $x =$
$$^{\ulcorner}{=}({}^{\ulcorner} \frown z_1 \frown {}^{\ulcorner},{}^{\ulcorner} \frown z_2 \frown {}^{\urcorner})^{\urcorner}.$$

3. $x = {}^{\ulcorner}\bot^{\urcorner}$.

□

Proposition 13.8. *The relation* $\mathrm{Frm}(x)$ *which holds iff* x *is the Gödel number of a formula is primitive recursive.*

Proof. A sequence of symbols s is a formula iff there is formation sequence s_0, $\ldots$, $s_{k-1} = s$ of formula which records how s was formed from atomic formulae according to the formation rules. The code for each s_i (and indeed of the code of the sequence $\langle s_0, \ldots, s_{k-1} \rangle$ is less than the code x of s. □

Proposition 13.9. *The relation* $\mathrm{FreeOcc}(x, z, i)$*, which holds iff the* i*-th symbol of the formula with Gödel number* x *is a free occurrence of the variable with Gödel number* z*, is primitive recursive.*

Proof. Exercise. □

Proposition 13.10. *The property* $\mathrm{Sent}(x)$ *which holds iff* x *is the Gödel number of a sentence is primitive recursive.*

Proof. A sentence is a formula without free occurrences of variables. So $\mathrm{Sent}(x)$ holds iff

$$(\forall i < \mathrm{len}(x))\ (\forall z < x)\ ((\exists j < z)\ z = {}^{\#}v_j^{\#} \supset {\sim}\mathrm{FreeOcc}(x, z, i)).$$

□

13.5 Substitution

Proposition 13.11. *There is a primitive recursive function* $\mathrm{Subst}(x, y, z)$ *with the property that*

$$\mathrm{Subst}({}^{\#}\varphi^{\#}, {}^{\#}t^{\#}, {}^{\#}u^{\#}) = {}^{\#}\varphi[t/u]^{\#}$$

Proof. We can then define a function hSubst by primitive recursion as follows:

$$\begin{aligned} \mathrm{hSubst}(x, y, z, 0) &= \emptyset \\ \mathrm{hSubst}(x, y, z, i+1) &= \\ &\begin{cases} \mathrm{hSubst}(x, y, z, i) \frown y & \text{if } \mathrm{FreeOcc}(x, z, i+1) \\ \mathrm{append}(\mathrm{hSubst}(x, y, z, i), (x)_{i+1}) & \text{otherwise.} \end{cases} \end{aligned}$$

$\mathrm{Subst}(x, y, z)$ can now be defined as $\mathrm{hSubst}(x, y, z, \mathrm{len}(x))$. □

Proposition 13.12. *The relation* $\mathrm{FreeFor}(x, y, z)$*, which holds iff the term with Gödel number* y *is free for the variable with Gödel number* z *in the formula with Gödel number* x*, is primitive recursive.*

Proof. Exercise. □

13.6 Derivations in Natural Deduction

In order to arithmetize derivations, we must represent derivations as numbers. Since derivations are trees of formulae where each inference carries one or two labels, a recursive representation is the most obvious approach: we represent a derivation as a tuple, the components of which are the end-formula, the labels, and the representations of the sub-derivations leading to the premises of the last inference.

Definition 13.13. If δ is a derivation in natural deduction, then ${}^{\natural}\delta^{\#}$ is

1. $\langle 0, {}^{\natural}\varphi^{\#}, n\rangle$ if δ consists only of the assumption φ. The number n is 0 if it is an undischarged assumption, and the numerical label otherwise.

2. $\langle 1, {}^{\natural}\varphi^{\#}, n, k, {}^{\natural}\delta_1{}^{\#}\rangle$ if δ ends in an inference with one premise, k is given by the following table according to which rule was used in the last inference, and δ_1 is the immediate subproof ending in the premise of the last inference. n is the label of the inference, or 0 if the inference does not discharge any assumptions.

Rule:	&Elim	$\vee$Intro	$\supset$Intro	$\sim$Intro	$\bot_I$
k:	1	2	3	4	5

Rule:	$\bot_C$	$\forall$Intro	$\forall$Elim	$\exists$Intro	=Intro
k:	6	7	8	9	10

3. $\langle 2, {}^{\natural}\varphi^{\#}, n, k, {}^{\natural}\delta_1{}^{\#}, {}^{\natural}\delta_2{}^{\#}\rangle$ if δ ends in an inference with two premises, k is given by the following table according to which rule was used in the last inference, and δ_1, δ_2 are the immediate subderivations ending in the left and right premise of the last inference, respectively. n is the label of the inference, or 0 if the inference does not discharge any assumptions.

Rule:	&Intro	$\supset$Elim	$\sim$Elim
k:	1	2	3

4. $\langle 3, {}^{\natural}\varphi^{\#}, n, {}^{\natural}\delta_1{}^{\#}, {}^{\natural}\delta_2{}^{\#}, {}^{\natural}\delta_3{}^{\#}\rangle$ if δ ends in an $\vee$Elim inference. δ_1, δ_2, δ_3 are the immediate subderivations ending in the left, middle, and right premise of the last inference, respectively, and n is the label of the inference.

Example 13.14. Consider the very simple derivation

$$\cfrac{\cfrac{[(\varphi \,\&\, \psi)]^1}{\varphi}\,\&\text{Elim}}{(\varphi \supset \psi)}\,{}^{1}\supset\text{Intro}$$

The Gödel number of the assumption would be $d_0 = \langle 0, {}^{\natural}(\varphi \,\&\, \psi)^{\#}, 1\rangle$. The Gödel number of the derivation ending in the conclusion of &Elim would be

$d_1 = \langle 1, {}^{\#}\varphi^{\#}, 0, 1, d_0\rangle$ (1 since &Elim has one premise, Gödel number of conclusion φ, 0 because no assumption is discharged, 1 is the number coding &Elim). The Gödel number of the entire derivation then is $\langle 1, {}^{\#}(\varphi \supset \psi)^{\#}, 1, 3, d_1\rangle$, i.e.,

$$2^2 \cdot 3^{{}^{\#}(\varphi\supset\psi)^{\#}+1} \cdot 5^2 \cdot 7^4 \cdot 11^{(2^2\cdot 3^{{}^{\#}\varphi^{\#}+1}\cdot 5^1\cdot 7^2\cdot 11^{(2^1\cdot 3^{{}^{\#}(\varphi\&\psi)^{\#}+1}\cdot 5^2)})}.$$

Having settled on a representation of derivations, we must also show that we can manipulate such derivations primite recursively, and express their essential properties and relations so. Some operations are simple: e.g., given a Gödel number d of a derivation, $(d)_1$ gives us the Gödel number of its end-formula. Some are much harder. We'll at least sketch how to do this. The goal is to show that the relation "δ is a derivation of φ from Γ" is primitive recursive on the Gödel numbers of δ and φ.

Proposition 13.15. *The following relations are primitive recursive:*

1. *φ occurs as an assumption in δ with label n.*
2. *All assumption in δ with label n are of the form φ (i.e., we can discharge the assumption φ using label n in δ).*
3. *φ is an undischarged assumption of δ.*
4. *An inference with conclusion φ, upper derivations δ_1 (and δ_2, δ_3), and discharge label n is correct.*
5. *δ is a correct natural deduction derivation.*

Proof. We have to show that the corresponding relations between Gödel numbers of formulae, sequences of Gödel numbers of formulae (which code sets of formulae), and Gödel numbers of derivations are primitive recursive.

1. We want to show that $\mathrm{Assum}(x, d, n)$, which holds if x is the Gödel number of an assumption of the derivation with Gödel number d labelled n, is primitive recursive. For this we need a helper relation $\mathrm{hAssum}(x, d, n, i)$ which holds if the formula φ with Gödel number x occurs as an initial formula with label n in the derivation with Gödel number d within i inferences up from the end-formula.

$$\begin{aligned}
\mathrm{hAssum}(x, d, n, 0) &\Leftrightarrow 1\\
\mathrm{hAssum}(x, d, n, i+1) &\Leftrightarrow\\
&\mathrm{Sent}(x)\ \&\ (d = \langle 0, x, n\rangle \vee\\
&((d)_0 = 1\ \&\ \mathrm{hAssum}(x, (d)_4, n, i)) \vee\\
&((d)_0 = 2\ \&\ (\mathrm{hAssum}(x, (d)_4, n, i) \vee\\
&\qquad \mathrm{hAssum}(x, (d)_5, n, i))) \vee\\
&((d)_0 = 3\ \&\ (\mathrm{hAssum}(x, (d)_3, n, i) \vee\\
&\qquad \mathrm{hAssum}(x, (d)_2, n, i)) \vee \mathrm{hAssum}(x, (d)_3, n, i))
\end{aligned}$$

If the number i is large enough, e.g., larger than the maximum number of inferences between an initial formula and the end-formula of δ, it holds of x, d, n, and i iff φ is an initial formula in δ labelled n. The number d itself is larger than that maximum number of inferences. So we can define

$$\mathrm{Assum}(x, d, n) = \mathrm{hAssum}(x, d, n, d).$$

2. We want to show that $\mathrm{Discharge}(x, d, n)$, which holds if all assumptions with label n in the derivation with Gödel number d all are the formula with Gödel number x. But this relation holds iff $(\forall y < d)\ (\mathrm{Assum}(y, d, n) \supset y = x)$.

3. An occurrence of an assumption is not open if it occurs with label n in a subderivation that ends in a rule with discharge label n. Define the helper relation $\mathrm{hNotOpen}(x, d, n, i)$ as

$$\begin{aligned}
\mathrm{hNotOpen}(x, d, n, 0) &\Leftrightarrow 1\\
\mathrm{hNotOpen}(x, d, n, i+1) &\Leftrightarrow\\
&\quad (d)_2 = n \vee\\
&\quad ((d)_0 = 1 \,\&\, \mathrm{hNotOpen}(x, (d)_4, n, i)) \vee\\
&\quad ((d)_0 = 2 \,\&\, \mathrm{hNotOpen}(x, (d)_4, n, i) \,\&\,\\
&\qquad \mathrm{hNotOpen}(x, (d)_5, n, i))) \vee\\
&\quad ((d)_0 = 3 \,\&\, \mathrm{hNotOpen}(x, (d)_3, n, i) \,\&\,\\
&\qquad \mathrm{hNotOpen}(x, (d)_4, n, i) \,\&\, \mathrm{hNotOpen}(x, (d)_5, n, i))
\end{aligned}$$

Note that all assumptions of the form φ labelled n are discharged in δ iff either the last inference of δ discharges them (i.e., the last inference has label n), or if it is discharged in all of the immediate subderivations.

A formula φ is an open assumption of δ iff it is an initial formula of δ (with label n) and is not discharged in δ (by a rule with lable n). We can then define $\mathrm{OpenAssum}(x, d)$ as

$$(\exists n < d)\ (\mathrm{Assum}(x, d, n, d) \,\&\, {\sim}\mathrm{hNotOpen}(x, d, n, d)).$$

4. Here we have to show that for each rule of inference R the relation $\mathrm{FollowsBy}_R(x, d_1, n)$ which holds if x is the Gödel number of the conclusion and d_1 is the Gödel number of a derivation ending in the premise of a correct application of R with label n is primitive recursive, and similarly for rules with two or three premises.

The simplest case is that of the $=$Intro rule. Here there is no premise, i.e., $d_1 = 0$. However, φ must be of the form $t = t$, for a closed term t. Here, a primitive recursive definition is

$$(\exists t < x)\ (\mathrm{ClTerm}(t) \,\&\, x = ({}^{\#}{=}({}^{\#} \frown t \frown {}^{\#}{,}^{\#} \frown t \frown {}^{\#}){}^{\#})) \,\&\, d_1 = 0).$$

For a more complicated example, $\text{FollowsBy}_{\supset\text{Intro}}(x, d_1, n)$ holds iff φ is of the form $(\psi \supset \chi)$, the end-formula of δ is χ, and any initial formula in δ labelled n is of the form ψ. We can express this primitive recursively by

$$\begin{aligned}(\exists y < x)\ (&\text{Sent}(y)\ \&\ \text{Discharge}(y, d_1)\ \&\\ &(\exists z < x)\ (\text{Sent}(y)\ \&\ (d)_1 = z)\ \&\\ &\qquad x = ({}^{\#}({}^{\#} \frown y \frown {}^{\#}\supset^{\#} \frown z \frown {}^{\#})^{\#}))\end{aligned}$$

(Think of y as the Gödel number of ψ and z as that of χ.)

For another example, consider $\exists$Intro. Here, φ is the conclusion of a correct inference with one upper derivation iff there is a formula ψ, a closed term t and a variable x such that $\psi[t/x]$ is the end-formula of the upper derivation and $\exists x\, \psi$ is the conclusion φ, i.e., the formula with Gödel number x. So $\text{FollowsBy}_{\exists\text{Intro}}(x, d_1, n)$ holds iff

$$\begin{aligned}&\text{Sent}(x)\ \&\ (\exists y < x)\ (\exists v < x)\ (\exists t < d)\ (\text{Frm}(y)\ \&\ \text{Term}(t)\ \&\ \text{Var}(v)\ \&\\ &\quad \text{FreeFor}(y, t, v)\ \&\ \text{Subst}(y, t, v) = (d_1)_1\ \&\ x = ({}^{\#}\exists^{\#} \frown v \frown z))\end{aligned}$$

5. We first define a helper relation $\text{hDeriv}(d, i)$ which holds if d codes a correct derivation at least to i inferences up from the end sequent. $\text{hDeriv}(d, 0)$ holds always. Otherwise, $\text{hDeriv}(d, i+1)$ iff either d just codes an assumption or d ends in a correct inference and the codes of the immediate

sub-derivations satisfy $\text{hDeriv}(d', i)$.

$$
\begin{aligned}
&\text{hDeriv}(d, 0) \Leftrightarrow 1\\
&\text{hDeriv}(d, i+1) \Leftrightarrow\\
&\qquad (\exists x < d)\ (\exists n < d)\ (\text{Sent}(x)\ \&\ d = \langle 0, x, n\rangle) \vee\\
&\qquad ((d)_0 = 1\ \&\\
&\qquad\quad ((d)_3 = 1\ \&\ \text{FollowsBy}_{\&\text{Elim}}((d)_1, (d)_4, (d)_2) \vee\\
&\qquad\qquad \vdots\\
&\qquad\quad ((d)_3 = 10\ \&\ \text{FollowsBy}_{=\text{Intro}}((d)_1, (d)_4, (d)_2))\ \&\\
&\qquad\quad \text{nDeriv}((d)_4, i)) \vee\\
&\qquad ((d)_0 = 2\ \&\\
&\qquad\quad ((d)_3 = 1\ \&\ \text{FollowsBy}_{\&\text{Intro}}((d)_1, (d)_4, (d)_5, (d)_2)) \vee\\
&\qquad\qquad \vdots\\
&\qquad\quad ((d)_3 = 3\ \&\ \text{FollowsBy}_{\sim\text{Elim}}((d)_1, (d)_4, (d)_5, (d)_2))\ \&\\
&\qquad\quad \text{hDeriv}((d)_4, i)\ \&\ \text{hDeriv}((d)_5, i)) \vee\\
&\qquad ((d)_0 = 3\ \&\\
&\qquad\quad \text{FollowsBy}_{\vee\text{Elim}}((d)_1, (d)_3, (d)_4, (d)_5, (d)_2)\ \&\\
&\qquad\quad \text{hDeriv}((d)_3, i)\ \&\ \text{hDeriv}((d)_4, i))\ \&\ \text{hDeriv}((d)_5, i)
\end{aligned}
$$

This is a primitive recursive definition. Again we can define $\text{Deriv}(d)$ as $\text{hDeriv}(d, d)$.

□

Proposition 13.16. *Suppose Γ is a primitive recursive set of sentences. Then the relation $\text{Prf}_\Gamma(x, y)$ expressing "x is the code of a derivation δ of φ from undischarged assumptions in Γ and y is the Gödel number of φ" is primitive recursive.*

Proof. Suppose "$y \in \Gamma$" is given by the primitive recursive predicate $R_\Gamma(y)$. We have to show that $\text{Prf}_\Gamma(x, y)$ which holds iff y is the Gödel number of a sentence φ and x is the code of a natural deduction derivation with end formula φ and all undischarged assumptions in Γ is primitive recursive.

By the previous proposition, the property $\text{Deriv}(x)$ which holds iff x is the code of a correct derivation δ in natural deduction is primitive recursive. If x is such a code, then $(x)_1$ is the code of the end-formula of δ. Thus we can define $\text{Prf}_\Gamma(x, y)$ by

$$
\begin{aligned}
\text{Prf}_\Gamma(x, y) \Leftrightarrow\ &\text{Deriv}(x)\ \&\ (x)_1 = y\ \&\\
&(\forall z < x)\ (\text{OpenAssum}(z, x) \supset R_\Gamma(z))
\end{aligned}
$$

□

Chapter 14

Representability in Q

14.1 Introduction

We will describe a very minimal such theory called "**Q**" (or, sometimes, "Robinson's Q," after Raphael Robinson). We will say what it means for a function to be *representable* in **Q**, and then we will prove the following:

> A function is representable in **Q** if and only if it is computable.

For one thing, this provides us with another model of computability. But we will also use it to show that the set $\{\varphi \mid \mathbf{Q} \vdash \varphi\}$ is not decidable, by reducing the halting problem to it. By the time we are done, we will have proved much stronger things than this.

The language of **Q** is the language of arithmetic; **Q** consists of the following axioms (to be used in conjunction with the other axioms and rules of first-order logic with identity predicate):

$$\forall x \, \forall y \, (x' = y' \supset x = y) \tag{Q_1}$$
$$\forall x \, \mathsf{o} \neq x' \tag{Q_2}$$
$$\forall x \, (x \neq \mathsf{o} \supset \exists y \, x = y') \tag{Q_3}$$
$$\forall x \, (x + \mathsf{o}) = x \tag{Q_4}$$
$$\forall x \, \forall y \, (x + y') = (x + y)' \tag{Q_5}$$
$$\forall x \, (x \times \mathsf{o}) = \mathsf{o} \tag{Q_6}$$
$$\forall x \, \forall y \, (x \times y') = ((x \times y) + x) \tag{Q_7}$$
$$\forall x \, \forall y \, (x < y \equiv \exists z \, (z' + x) = y) \tag{Q_8}$$

For each natural number n, define the numeral $\overline{n}$ to be the term $0''^{\cdots'}$ where there are n tick marks in all. So, $\overline{0}$ is the constant symbol o by itself, $\overline{1}$ is o', $\overline{2}$ is o'', etc.

As a theory of arithmetic, **Q** is *extremely* weak; for example, you can't even prove very simple facts like $\forall x \, x \neq x'$ or $\forall x \, \forall y \, (x + y) = (y + x)$. But we will

see that much of the reason that **Q** is so interesting is *because* it is so weak. In fact, it is just barely strong enough for the incompleteness theorem to hold. Another reason **Q** is interesting is because it has a *finite* set of axioms.

A stronger theory than **Q** (called *Peano arithmetic* **PA**) is obtained by adding a schema of induction to **Q**:

$$(\varphi(0) \mathbin{\&} \forall x\,(\varphi(x) \supset \varphi(x'))) \supset \forall x\,\varphi(x)$$

where $\varphi(x)$ is any formula. If $\varphi(x)$ contains free variables other than x, we add universal quantifiers to the front to bind all of them (so that the corresponding instance of the induction schema is a sentence). For instance, if $\varphi(x, y)$ also contains the variable y free, the corresponding instance is

$$\forall y\,((\varphi(0) \mathbin{\&} \forall x\,(\varphi(x) \supset \varphi(x'))) \supset \forall x\,\varphi(x))$$

Using instances of the induction schema, one can prove much more from the axioms of **PA** than from those of **Q**. In fact, it takes a good deal of work to find "natural" statements about the natural numbers that can't be proved in Peano arithmetic!

Definition 14.1. A function $f(x_0, \ldots, x_k)$ from the natural numbers to the natural numbers is said to be *representable in* **Q** if there is a formula $\varphi_f(x_0, \ldots, x_k, y)$ such that whenever $f(n_0, \ldots, n_k) = m$, **Q** proves

1. $\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})$
2. $\forall y\,(\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, y) \supset \overline{m} = y)$.

There are other ways of stating the definition; for example, we could equivalently require that **Q** proves $\forall y\,(\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, y) \equiv y = \overline{m})$.

Theorem 14.2. *A function is representable in* **Q** *if and only if it is computable.*

There are two directions to proving the theorem. The left-to-right direction is fairly straightforward once arithmetization of syntax is in place. The other direction requires more work. Here is the basic idea: we pick "general recursive" as a way of making "computable" precise, and show that every general recursive function is representable in **Q**. Recall that a function is general recursive if it can be defined from zero, the successor function succ, and the projection functions P^n_i, using composition, primitive recursion, and regular minimization. So one way of showing that every general recursive function is representable in **Q** is to show that the basic functions are representable, and whenever some functions are representable, then so are the functions defined from them using composition, primitive recursion, and regular minimization. In other words, we might show that the basic functions are representable, and that the representable functions are "closed under" composition, primitive

recursion, and regular minimization. This guarantees that every general recursive function is representable.

It turns out that the step where we would show that representable functions are closed under primitive recursion is hard. In order to avoid this step, we show first that in fact we can do without primitive recursion. That is, we show that every general recursive function can be defined from basic functions using composition and regular minimization alone. To do this, we show that primitive recursion can actually be done by a specific regular minimization. However, for this to work, we have to add some additional basic functions: addition, multiplication, and the characteristic function of the identity relation $\chi_=$. Then, we can prove the theorem by showing that all of *these* basic functions are representable in **Q**, and the representable functions are closed under composition and regular minimization.

14.2 Functions Representable in Q are Computable

Lemma 14.3. *Every function that is representable in* **Q** *is computable.*

Proof. Let's first give the intuitive idea for why this is true. If $f(x_0, \ldots, x_k)$ is representable in **Q**, there is a formula $\varphi(x_0, \ldots, x_k, y)$ such that

$$\mathbf{Q} \vdash \varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m}) \quad \text{iff} \quad m = f(n_0, \ldots, n_k).$$

To compute f, we do the following. List all the possible derivations δ in the language of arithmetic. This is possible to do mechanically. For each one, check if it is a derivation of a formula of the form $\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})$. If it is, m must be $= f(n_0, \ldots, n_k)$ and we've found the value of f. The search terminates because $\mathbf{Q} \vdash \varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{f(n_0, \ldots, n_k)})$, so eventually we find a δ of the right sort.

This is not quite precise because our procedure operates on derivations and formulae instead of just on numbers, and we haven't explained exactly why "listing all possible derivations" is mechanically possible. But as we've seen, it is possible to code terms, formulae, and derivations by Gödel numbers. We've also introduced a precise model of computation, the general recursive functions. And we've seen that the relation $\mathrm{Prf}_{\mathbf{Q}}(d, y)$, which holds iff d is the Gödel number of a derivation of the formula with Gödel number x from the axioms of **Q**, is (primitive) recursive. Other primitive recursive functions we'll need are num (Proposition 13.6) and Subst (Proposition 13.11). From these, it is possible to define f by minimization; thus, f is recursive.

First, define

$$\begin{aligned} A(n_0, \ldots, n_k, m) = \\ \mathrm{Subst}(\mathrm{Subst}(\ldots \mathrm{Subst}(&{}^{\#}\varphi_f{}^{\#}, \mathrm{num}(n_0), {}^{\#}x_0{}^{\#}), \\ &\ldots), \mathrm{num}(n_k), {}^{\#}x_k{}^{\#}), \mathrm{num}(m), {}^{\#}y^{\#}) \end{aligned}$$

This looks complicated, but it's just the function $A(n_0, \ldots, n_k, m) = {}^{\#}\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})^{\#}$.

Now, consider the relation $R(n_0, \ldots, n_k, s)$ which holds if $(s)_0$ is the Gödel number of a derivation from $\mathbf{Q}$ of $\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{(s)_1})$:

$$R(n_0, \ldots, n_k, s) \quad \text{iff} \quad \mathrm{Prf}_{\mathbf{Q}}((s)_0, A(n_0, \ldots, n_k, (s)_1)$$

If we can find an s such that $R(n_0, \ldots, n_k, s)$ hold, we have found a pair of numbers—$(s)_0$ and (s_1)—such that $(s)_0$ is the Gödel number of a derivation of $A_f(\overline{n_0}, \ldots, \overline{n_k}, (s)_1)$. So looking for s is like looking for the pair d and m in the informal proof. And a computable function that "looks for" such an s can be defined by regular minimization. Note that R is regular: for every n_0, ..., n_k, there is a derivation δ of $\mathbf{Q} \vdash \varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{f(n_0, \ldots, n_k)})$, so $R(n_0, \ldots, n_k, s)$ holds for $s = \langle {}^{\#}\delta^{\#}, f(n_0, \ldots, n_k)\rangle$. So, we can write f as

$$f(n_0, \ldots, n_k) = (\mu s\ R(n_0, \ldots, n_k, s))_1.$$

□

14.3 The Beta Function Lemma

In order to show that we can carry out primitive recursion if addition, multiplication, and $\chi_=$ are available, we need to develop functions that handle sequences. (If we had exponentiation as well, our task would be easier.) When we had primitive recursion, we could define things like the "n-th prime," and pick a fairly straightforward coding. But here we do not have primitive recursion—in fact we want to show that we can do primitive recursion using minimization—so we need to be more clever.

Lemma 14.4. *There is a function $\beta(d, i)$ such that for every sequence $a_0, \ldots, a_n$ there is a number d, such that for every $i \leq n$, $\beta(d, i) = a_i$. Moreover, β can be defined from the basic functions using just composition and regular minimization.*

Think of d as coding the sequence $\langle a_0, \ldots, a_n\rangle$, and $\beta(d, i)$ returning the i-th element. (Note that this "coding" does *not* use the prower-of-primes coding we're already familiar with!). The lemma is fairly minimal; it doesn't say we can concatenate sequences or append elements, or even that we can *compute* d from a_0, ..., a_n using functions definable by composition and regular minimization. All it says is that there is a "decoding" function such that every sequence is "coded."

The use of the notation β is Gödel's. To repeat, the hard part of proving the lemma is defining a suitable β using the seemingly restricted resources, i.e., using just composition and minimization—however, we're allowed to use addition, multiplication, and $\chi_=$. There are various ways to prove this lemma, but one of the cleanest is still Gödel's original method, which used a number-theoretic fact called the Chinese Remainder theorem.

Definition 14.5. Two natural numbers a and b are *relatively prime* if their greatest common divisor is 1; in other words, they have no other divisors in common.

Definition 14.6. $a \equiv b \mod c$ means $c \mid (a - b)$, i.e., a and b have the same remainder when divided by c.

Here is the *Chinese Remainder theorem*:

Theorem 14.7. *Suppose $x_0, \ldots, x_n$ are (pairwise) relatively prime. Let $y_0, \ldots, y_n$ be any numbers. Then there is a number z such that*

$$\begin{aligned} z &\equiv y_0 \mod x_0 \\ z &\equiv y_1 \mod x_1 \\ &\vdots \\ z &\equiv y_n \mod x_n. \end{aligned}$$

Here is how we will use the Chinese Remainder theorem: if $x_0, \ldots, x_n$ are bigger than $y_0, \ldots, y_n$ respectively, then we can take z to code the sequence $\langle y_0, \ldots, y_n \rangle$. To recover y_i, we need only divide z by x_i and take the remainder. To use this coding, we will need to find suitable values for $x_0, \ldots, x_n$.

A couple of observations will help us in this regard. Given $y_0, \ldots, y_n$, let

$$j = \max(n, y_0, \ldots, y_n) + 1,$$

and let

$$\begin{aligned} x_0 &= 1 + j! \\ x_1 &= 1 + 2 \cdot j! \\ x_2 &= 1 + 3 \cdot j! \\ &\vdots \\ x_n &= 1 + (n+1) \cdot j! \end{aligned}$$

Then two things are true:

1. $x_0, \ldots, x_n$ are relatively prime.
2. For each i, $y_i < x_i$.

To see that (1) is true, note that if p is a prime number and $p \mid x_i$ and $p \mid x_k$, then $p \mid 1 + (i+1)j!$ and $p \mid 1 + (k+1)j!$. But then p divides their difference,

$$(1 + (i+1)j!) - (1 + (k+1)j!) = (i - k)j!.$$

Since p divides $1 + (i+1)j!$, it can't divide $j!$ as well (otherwise, the first division would leave a remainder of 1). So p divides $i - k$, since p divides $(i - k)j!$.

But $|i - k|$ is at most n, and we have chosen $j > n$, so this implies that $p \mid j!$, again a contradiction. So there is no prime number dividing both x_i and x_k. Clause (2) is easy: we have $y_i < j < j! < x_i$.

Now let us prove the β function lemma. Remember that we can use 0, successor, plus, times, $\chi_=$, projections, and any function defined from them using composition and minimization applied to regular functions. We can also use a relation if its characteristic function is so definable. As before we can show that these relations are closed under boolean combinations and bounded quantification; for example:

1. $\text{not}(x) = \chi_=(x, 0)$
2. $(\min x \leq z)\, R(x, y) = \mu x\ (R(x, y) \vee x = z)$
3. $(\exists x \leq z)\ R(x, y) \Leftrightarrow R((\min x \leq z)\, R(x, y), y)$

We can then show that all of the following are also definable without primitive recursion:

1. The pairing function, $J(x, y) = \frac{1}{2}[(x + y)(x + y + 1)] + x$
2. Projections
$$K(z) = (\min x \leq q)\, (\exists y \leq z\, [z = J(x, y)])$$
and
$$L(z) = (\min y \leq q)\, (\exists x \leq z\, [z = J(x, y)]).$$
3. $x < y$
4. $x \mid y$
5. The function $\text{rem}(x, y)$ which returns the remainder when y is divided by x

Now define
$$\beta^*(d_0, d_1, i) = \text{rem}(1 + (i + 1)d_1, d_0)$$
and
$$\beta(d, i) = \beta^*(K(d), L(d), i).$$
This is the function we need. Given $a_0, \ldots, a_n$, as above, let
$$j = \max(n, a_0, \ldots, a_n) + 1,$$
and let $d_1 = j!$. By the observations above, we know that $1 + d_1, 1 + 2d_1, \ldots, 1 + (n + 1)d_1$ are relatively prime and all are bigger than $a_0, \ldots, a_n$. By the Chinese Remainder theorem there is a value d_0 such that for each i,
$$d_0 \equiv a_i \mod (1 + (i + 1)d_1)$$

and so (because d_1 is greater than a_i),

$$a_i = \text{rem}(1 + (i+1)d_1, d_0).$$

Let $d = J(d_0, d_1)$. Then for each $i \leq n$, we have

$$\begin{aligned} \beta(d, i) &= \beta^*(d_0, d_1, i) \\ &= \text{rem}(1 + (i+1)d_1, d_0) \\ &= a_i \end{aligned}$$

which is what we need. This completes the proof of the β-function lemma.

14.4 Simulating Primitive Recursion

Now we can show that definition by primitive recursion can be "simulated" by regular minimization using the beta function. Suppose we have $f(\vec{z})$ and $g(u, v, \vec{z})$. Then the function $h(x, \vec{z})$ defined from f and g by primitive recursion is

$$\begin{aligned} h(0, \vec{z}) &= f(\vec{z}) \\ h(x+1, \vec{z}) &= g(x, h(x, \vec{z}), \vec{z}). \end{aligned}$$

We need to show that h can be defined from f and g using just composition and regular minimization, using the basic functions and functions defined from them using composition and regular minimization (such as β).

Lemma 14.8. *If h can be defined from f and g using primitive recursion, it can be defined from f, g, the functions* zero, succ, P^n_i, add, mult, $\chi_=$, *using composition and regular minimization.*

Proof. First, define an auxiliary function $\hat{h}(x, \vec{z})$ which returns the least number d such that d codes a sequence which satisfies

1. $(d)_0 = f(\vec{z})$, and
2. for each $i < x$, $(d)_{i+1} = g(i, (d)_i, \vec{z})$,

where now $(d)_i$ is short for $\beta(d, i)$. In other words, $\hat{h}$ returns the sequence $\langle h(0, \vec{z}), h(1, \vec{z}), \ldots, h(x, \vec{z})\rangle$. We can write $\hat{h}$ as

$$\hat{h}(x, z) = \mu d\ (\beta(d, 0) = f(\vec{z}) \mathbin{\&} \forall i < x\, \beta(d, i+1) = g(i, \beta(d, i), \vec{z})).$$

Note: no primitive recursion is needed here, just minimization. The function we minimize is regular because of the beta function lemma Lemma 14.4.

But now we have

$$h(x, \vec{z}) = \beta(\hat{h}(x, \vec{z}), x),$$

so h can be defined from the basic functions using just composition and regular minimization. □

14.5 Basic Functions are Representable in Q

First we have to show that all the basic functions are representable in **Q**. In the end, we need to show how to assign to each k-ary basic function $f(x_0, \ldots, x_{k-1})$ a formula $\varphi_f(x_0, \ldots, x_{k-1}, y)$ that represents it.

We will be able to represent zero, successor, plus, times, the characteristic function for equality, and projections. In each case, the appropriate representing function is entirely straightforward; for example, zero is represented by the formula $y = 0$, successor is represented by the formula $x_0' = y$, and addition is represented by the formula $(x_0 + x_1) = y$. The work involves showing that **Q** can prove the relevant sentences; for example, saying that addition is represented by the formula above involves showing that for every pair of natural numbers m and n, **Q** proves

$$\overline{n} + \overline{m} = \overline{n+m} \text{ and}$$
$$\forall y\, ((\overline{n} + \overline{m}) = y \supset y = \overline{n+m}).$$

Proposition 14.9. *The zero function* $\mathrm{zero}(x) = 0$ *is represented in* **Q** *by* $y = 0$.

Proposition 14.10. *The successor function* $\mathrm{succ}(x) = x + 1$ *is represented in* **Q** *by* $y = x'$.

Proposition 14.11. *The projection function* $P_i^n(x_0, \ldots, x_{n-1}) = x_i$ *is represented in* **Q** *by* $y = x_i$.

Proposition 14.12. *The characteristic function of* $=$,

$$\chi_=(x_0, x_1) = \begin{cases} 1 & \text{if } x_0 = x_1 \\ 0 & \text{otherwise} \end{cases}$$

is represented in **Q** *by*

$$(x_0 = x_1 \,\&\, y = \overline{1}) \vee (x_0 \neq x_1 \,\&\, y = \overline{0}).$$

The proof requires the following lemma.

Lemma 14.13. *Given natural numbers n and m, if $n \neq m$, then* $\mathbf{Q} \vdash \overline{n} \neq \overline{m}$.

Proof. Use induction on n to show that for every m, if $n \neq m$, then $Q \vdash \overline{n} \neq \overline{m}$.

In the base case, $n = 0$. If m is not equal to 0, then $m = k + 1$ for some natural number k. We have an axiom that says $\forall x\, 0 \neq x'$. By a quantifier axiom, replacing x by $\overline{k}$, we can conclude $0 \neq \overline{k}'$. But $\overline{k}'$ is just $\overline{m}$.

In the induction step, we can assume the claim is true for n, and consider $n + 1$. Let m be any natural number. There are two possibilities: either $m = 0$ or for some k we have $m = k + 1$. The first case is handled as above. In the second case, suppose $n + 1 \neq k + 1$. Then $n \neq k$. By the induction hypothesis

for n we have $\mathbf{Q} \vdash \overline{n} \neq \overline{k}$. We have an axiom that says $\forall x \forall y\, x' = y' \supset x = y$. Using a quantifier axiom, we have $\overline{n}' = \overline{k}' \supset \overline{n} = \overline{k}$. Using propositional logic, we can conclude, in $\mathbf{Q}$, $\overline{n} \neq \overline{k} \supset \overline{n}' \neq \overline{k}'$. Using modus ponens, we can conclude $\overline{n}' \neq \overline{k}'$, which is what we want, since $\overline{k}'$ is $\overline{m}$. □

Note that the lemma does not say much: in essence it says that **Q** can prove that different numerals denote different objects. For example, **Q** proves $0'' \neq 0'''$. But showing that this holds in general requires some care. Note also that although we are using induction, it is induction *outside* of **Q**.

Proof of Proposition 14.12. If $n = m$, then $\overline{n}$ and $\overline{m}$ are the same term, and $\chi_=(n, m) = 1$. But $\mathbf{Q} \vdash (\overline{n} = \overline{m} \,\&\, \overline{1} = \overline{1})$, so it proves $\varphi_=(\overline{n}, \overline{m}, \overline{1})$. If $n \neq m$, then $\chi_=(n, m) = 0$. By Lemma 14.13, $\mathbf{Q} \vdash \overline{n} \neq \overline{m}$ and so also $(\overline{n} \neq \overline{m} \,\&\, 0 = 0)$. Thus $\mathbf{Q} \vdash \varphi_=(\overline{n}, \overline{m}, \overline{0})$.

For the second part, we also have two cases. If $n = m$, we have to show that that $\mathbf{Q} \vdash \forall(\varphi_=(\overline{n}, \overline{m}, y) \supset y = \overline{1})$. Arguing informally, suppose $\varphi_=(\overline{n}, \overline{m}, y)$, i.e.,

$$(\overline{n} = \overline{n} \,\&\, y = \overline{1}) \vee (\overline{n} \neq \overline{n} \,\&\, y = \overline{0})$$

The left disjunct implies $y = \overline{1}$ by logic; the right contradicts $\overline{n} = \overline{n}$ which is provable by logic.

Suppose, on the other hand, that $n \neq m$. Then $\varphi_=(\overline{n}, \overline{m}, y)$ is

$$(\overline{n} = \overline{m} \,\&\, y = \overline{1}) \vee (\overline{n} \neq \overline{m} \,\&\, y = \overline{0})$$

Here, the left disjunct contradicts $\overline{n} \neq \overline{m}$, which is provable in **Q** by Lemma 14.13; the right disjunct entails $y = \overline{0}$. □

Proposition 14.14. *The addition function* $\mathrm{add}(x_0, x_1) = x_0 + x_1$ *is is represented in* **Q** *by*

$$y = (x_0 + x_1).$$

Lemma 14.15. $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$

Proof. We prove this by induction on m. If $m = 0$, the claim is that $\mathbf{Q} \vdash (\overline{n} + 0) = \overline{n}$. This follows by axiom Q_4. Now suppose the claim for m; let's prove the claim for $m + 1$, i.e., prove that $\mathbf{Q} \vdash (\overline{n} + \overline{m + 1}) = \overline{n + m + 1}$. Note that $\overline{m + 1}$ is just $\overline{m}'$, and $\overline{n + m + 1}$ is just $\overline{n + m}'$. By axiom Q_5, $\mathbf{Q} \vdash (\overline{n} + \overline{m}') = (\overline{n} + \overline{m})'$. By induction hypothesis, $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$. So $\mathbf{Q} \vdash (\overline{n} + \overline{m}') = \overline{n + m}'$. □

Proof of Proposition 14.14. The formula $\varphi_{\mathrm{add}}(x_0, x_1, y)$ representing add is $y = (x_0 + x_1)$. First we show that if $\mathrm{add}(n, m) = k$, then $\mathbf{Q} \vdash \varphi_{\mathrm{add}}(\overline{n}, \overline{m}, \overline{k})$, i.e., $\mathbf{Q} \vdash \overline{k} = (\overline{n} + \overline{m})$. But since $k = n + m$, $\overline{k}$ just is $\overline{n + m}$, and we've shown in Lemma 14.15 that $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$.

We also have to show that if $\text{add}(n, m) = k$, then

$$\mathbf{Q} \vdash \forall y\,(\varphi_{\text{add}}(\overline{n}, \overline{m}, y) \supset y = \overline{k}).$$

Suppose we have $\overline{n} + \overline{m} = y$. Since

$$\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m},$$

we can replace the left side with $\overline{n + m}$ and get $\overline{n + m} = y$, for arbitrary y. □

Proposition 14.16. *The multiplication function* $\text{mult}(x_0, x_1) = x_0 \cdot x_1$ *is represented in* **Q** *by*

$$y = (x_0 \times x_1).$$

Proof. Exercise. □

Lemma 14.17. $\mathbf{Q} \vdash (\overline{n} \times \overline{m}) = \overline{n \cdot m}$

Proof. Exercise. □

14.6 Composition is Representable in Q

Suppose h is defined by

$$h(x_0, \ldots, x_{l-1}) = f(g_0(x_0, \ldots, x_{l-1}), \ldots, g_{k-1}(x_0, \ldots, x_{l-1})).$$

where we have already found formulae $\varphi_f, \varphi_{g_0}, \ldots, \varphi_{g_{k-1}}$ representing the functions f, and $g_0, \ldots, g_{k-1}$, respectively. We have to find a formula φ_h representing h.

Let's start with a simple case, where all functions are 1-place, i.e., consider $h(x) = f(g(x))$. If $\varphi_f(y, z)$ represents f, and $\varphi_g(x, y)$ represents g, we need a formula $\varphi_h(x, z)$ that represents h. Note that $h(x) = z$ iff there is a y such that both $z = f(y)$ and $y = g(x)$. (If $h(x) = z$, then $g(x)$ is such a y; if such a y exists, then since $y = g(x)$ and $z = f(y)$, $z = f(g(x))$.) This suggests that $\exists y\,(\varphi_g(x, y)\,\&\,\varphi_f(y, z))$ is a good candidate for $\varphi_h(x, z)$. We just have to verify that **Q** proves the relevant formulae.

Proposition 14.18. *If* $h(n) = m$, *then* $\mathbf{Q} \vdash \varphi_h(\overline{n}, \overline{m})$.

Proof. Suppose $h(n) = m$, i.e., $f(g(n)) = m$. Let $k = g(n)$. Then

$$\mathbf{Q} \vdash \varphi_g(\overline{n}, \overline{k})$$

since φ_g represents g, and

$$\mathbf{Q} \vdash \varphi_f(\overline{k}, \overline{m})$$

since φ_f represents f. Thus,

$$\mathbf{Q} \vdash \varphi_g(\overline{n}, \overline{k}) \,\&\, \varphi_f(\overline{k}, \overline{m})$$

and consequently also

$$\mathbf{Q} \vdash \exists y\, (\varphi_g(\overline{n}, y) \,\&\, \varphi_f(y, \overline{m})),$$

i.e., $\mathbf{Q} \vdash \varphi_h(n, m)$. □

Proposition 14.19. *If $h(n) = m$, then $\mathbf{Q} \vdash \forall z\, (\varphi_h(\overline{n}, z) \supset z = \overline{m})$.*

Proof. Suppose $h(n) = m$, i.e., $f(g(n)) = m$. Let $k = g(n)$. Then

$$\mathbf{Q} \vdash \forall y\, (\varphi_g(\overline{n}, y) \supset y = \overline{k})$$

since φ_g represents g, and

$$\mathbf{Q} \vdash \forall z\, (\varphi_f(\overline{k}, z) \supset z = \overline{m})$$

since φ_f represents f. Using just a little bit of logic, we can show that also

$$\mathbf{Q} \vdash \forall z\, (\exists y\, (\varphi_g(\overline{n}, y) \,\&\, \varphi_f(y, z)) \supset z = \overline{m}).$$

i.e., $\mathbf{Q} \vdash \forall y\, (\varphi_h(\overline{n}, y) \supset y = \overline{m})$. □

The same idea works in the more complex case where f and g_i have arity greater than 1.

Proposition 14.20. *If $\varphi_f(y_0, \ldots, y_{k-1}, z)$ represents $f(y_0, \ldots, y_{k-1})$ in $\mathbf{Q}$, and $\varphi_{g_i}(x_0, \ldots, x_{l-1}, y)$ represents $g_i(x_0, \ldots, x_{l-1})$ in $\mathbf{Q}$, then*

$$\begin{aligned}\exists y_0, \ldots \exists y_{k-1}\, (&\varphi_{g_0}(x_0, \ldots, x_{l-1}, y_0) \,\&\, \cdots \,\&\\ &!A_{g_{k-1}}(x_0, \ldots, x_{l-1}, y_{k-1}) \,\&\, \varphi_f(y_0, \ldots, y_{k-1}, z))\end{aligned}$$

represents

$$h(x_0, \ldots, x_{k-1}) = f(g_0(x_0, \ldots, x_{k-1}), \ldots, g_0(x_0, \ldots, x_{k-1})).$$

Proof. Exercise. □

14.7 Regular Minimization is Representable in Q

Let's consider unbounded search. Suppose $g(x, z)$ is regular and representable in $\mathbf{Q}$, say by the formula $\varphi_g(x, z, y)$. Let f be defined by $f(z) = \mu x\,[g(x, z) = 0]$. We would like to find a formula $\varphi_f(z, y)$ representing f. The value of $f(z)$ is that number x which (a) satisfies $g(x, z) = 0$ and (b) is the least such, i.e., for any $w < x$, $g(w, z) \neq 0$. So the following is a natural choice:

$$\varphi_f(z, y) \equiv \varphi_g(y, z, 0) \,\&\, \forall w\,(w < y \supset \sim\varphi_g(w, z, 0)).$$

In the general case, of course, we would have to replace z with $z_0, \ldots, z_k$.

The proof, again, will involve some lemmas about things $\mathbf{Q}$ is strong enough to prove.

Lemma 14.21. *For every variable x and every natural number n,*

$$\mathbf{Q} \vdash (x' + \overline{n}) = (x + \overline{n})'.$$

Proof. The proof is, as usual, by induction on n. In the base case, $n = 0$, we need to show that $\mathbf{Q}$ proves $(x' + 0) = (x + 0)'$. But we have:

$$\mathbf{Q} \vdash (x' + 0) = x' \quad \text{by axiom } Q_4 \tag{14.1}$$
$$\mathbf{Q} \vdash (x + 0) = x \quad \text{by axiom } Q_4 \tag{14.2}$$
$$\mathbf{Q} \vdash (x + 0)' = x' \quad \text{by eq. (14.2)} \tag{14.3}$$
$$\mathbf{Q} \vdash (x' + 0) = (x + 0)' \quad \text{by eq. (14.1) and eq. (14.3)}$$

In the induction step, we can assume that we have shown that $\mathbf{Q} \vdash (x' + \overline{n}) = (x + \overline{n})'$. Since $\overline{n+1}$ is $\overline{n}'$, we need to show that $\mathbf{Q}$ proves $(x' + \overline{n}') = (x + \overline{n}')'$. We have:

$$\mathbf{Q} \vdash (x' + \overline{n}') = (x' + \overline{n})' \quad \text{by axiom } Q_5 \tag{14.4}$$
$$\mathbf{Q} \vdash (x' + \overline{n}') = (x + \overline{n}')' \quad \text{inductive hypothesis} \tag{14.5}$$
$$\mathbf{Q} \vdash (x' + \overline{n})' = (x + \overline{n}')' \quad \text{by eq. (14.4) and eq. (14.5).}$$

□

It is again worth mentioning that this is weaker than saying that $\mathbf{Q}$ proves $\forall x\,\forall y\,(x' + y) = (x + y)'$. Although this sentence is true in $\mathfrak{N}$, $\mathbf{Q}$ does not prove it.

Lemma 14.22. *1.* $\mathbf{Q} \vdash \forall x \sim x < 0$.

2. For every natural number n,

$$\mathbf{Q} \vdash \forall x\,(x < \overline{n+1} \supset (x = 0 \vee \cdots \vee x = \overline{n})).$$

Proof. Let us do 1 and part of 2, informally (i.e., only giving hints as to how to construct the formal derivation).

For part 1, by the definition of $<$, we need to prove $\sim\exists y\,(y' + x) = \mathsf{o}$ in **Q**, which is equivalent (using the axioms and rules of first-order logic) to $\forall y\,(y' + x) \neq 0$. Here is the idea: suppose $(y' + x) = \mathsf{o}$. If $x = \mathsf{o}$, we have $(y' + \mathsf{o}) = \mathsf{o}$. But by axiom Q_4 of **Q**, we have $(y' + \mathsf{o}) = y'$, and by axiom Q_2 we have $y' \neq \mathsf{o}$, a contradiction. So $\forall y\,(y' + x) \neq \mathsf{o}$. If $x \neq \mathsf{o}$, by axiom Q_3, there is a z such that $x = z'$. But then we have $(y' + z') = 0$. By axiom Q_5, we have $(y' + z)' = \mathsf{o}$, again contradicting axiom Q_2.

For part 2, use induction on n. Let us consider the base case, when $n = 0$. In that case, we need to show $x < \overline{1} \supset x = \mathsf{o}$. Suppose $x < \overline{1}$. Then by the defining axiom for $<$, we have $\exists y\,(y' + x) = \mathsf{o}'$. Suppose y has that property; so we have $y' + x = \mathsf{o}'$.

We need to show $x = \mathsf{o}$. By axiom Q_3, if $x \neq \mathsf{o}$, we get $x = z'$ for some z. Then we have $(y' + z') = \mathsf{o}'$. By axiom Q_5 of **Q**, we have $(y' + z)' = \mathsf{o}'$. By axiom Q_1, we have $(y' + z) = \mathsf{o}$. But this means, by definition, $z < \mathsf{o}$, contradicting part 1. □

Lemma 14.23. *For every $m \in \mathbb{N}$,*

$$\mathbf{Q} \vdash \forall y\,((y < \overline{m} \vee \overline{m} < y) \vee y = \overline{m}).$$

Proof. By induction on m. First, consider the case $m = 0$. $\mathbf{Q} \vdash \forall y\,(y \neq \mathsf{o} \supset \exists z\, y = z')$ by Q_3. But if $y = z'$, then $(z' + \mathsf{o}) = (y + \mathsf{o})$ by the logic of $=$. By Q_4, $(y + \mathsf{o}) = y$, so we have $(z' + \mathsf{o}) = y$, and hence $\exists z\,(z' + \mathsf{o}) = y$. By the definition of $<$ in Q_8, $\mathsf{o} < y$. If $\mathsf{o} < y$, then also $\mathsf{o} < y \vee y < \mathsf{o}$. We obtain: $y \neq \mathsf{o} \supset (\mathsf{o} < y \vee y < \mathsf{o})$, which is equivalent to $(\mathsf{o} < y \vee y < \mathsf{o}) \vee y = \mathsf{o}$.

Now suppose we have

$$\mathbf{Q} \vdash \forall y\,((y < \overline{m} \vee \overline{m} < y) \vee y = \overline{m})$$

and we want to show

$$\mathbf{Q} \vdash \forall y\,((y < \overline{m+1} \vee \overline{m+1} < y) \vee y = \overline{m+1})$$

The first disjunct $y < \overline{m}$ is equivalent (by Q_8) to $\exists z\,(z' + y) = \overline{m}$. If $(z' + y) = \overline{m}$, then also $(z' + y)' = \overline{m}'$. By Q_4, $(z' + y)' = (z'' + y)$. Hence, $(z'' + y) = \overline{m}'$. We get $\exists u\,(u' + y) = \overline{m+1}$ by existentially generalizing on z' and keeping in mind that $\overline{m}'$ is $\overline{m+1}$. Hence, if $y < \overline{m}$ then $y < \overline{m+1}$.

Now suppose $\overline{m} < y$, i.e., $\exists z\,(z' + \overline{m}) = y$. By Q_3 and some logic, we have $z = \mathsf{o} \vee \exists u\, z = u'$. If $z = \mathsf{o}$, we have $(\mathsf{o}' + \overline{m}) = y$. Since $\mathbf{Q} \vdash (\mathsf{o}' + \overline{m}) = \overline{m+1}$,

we have $y = \overline{m+1}$. Now suppose $\exists u\, z = u'$. Then:

$$\begin{aligned} y &= (z' + \overline{m}) && \text{by assumption} \\ (z' + \overline{m}) &= (u'' + \overline{m}) && \text{from } z = u' \\ (u'' + \overline{m}) &= (u' + \overline{m})' && \text{by Lemma 14.21} \\ (u' + \overline{m})' &= (u' + \overline{m}') && \text{by } Q_5\text{, so} \\ y &= (u' + \overline{m+1}) \end{aligned}$$

By existential generalization, $\exists u\, (u' + \overline{m+1}) = y$, i.e., $\overline{m+1} < y$. So, if $\overline{m} < y$, then $\overline{m+1} < y \vee y = \overline{m+1}$.

Finally, assume $y = \overline{m}$. Then, since $\mathbf{Q} \vdash (0' + \overline{m}) = \overline{m+1}$, $(0' + y) = \overline{m+1}$. From this we get $\exists z\, (z' + y) = \overline{m+1}$, or $y < \overline{m+1}$.

Hence, from each disjunct of the case for m, we can obtain the case for $m+1$. □

Proposition 14.24. *If $\varphi_g(x, z, y)$ represents $g(x, y)$ in $\mathbf{Q}$, then*

$$\varphi_f(z, y) \equiv \varphi_g(y, z, 0)\ \&\ \forall w\, (w < y \supset {\sim}\varphi_g(w, z, 0)).$$

represents $f(z) = \mu x\, [g(x, z) = 0]$.

Proof. First we show that if $f(n) = m$, then $\mathbf{Q} \vdash \varphi_f(\overline{n}, \overline{m})$, i.e.,

$$\mathbf{Q} \vdash \varphi_g(\overline{m}, \overline{n}, 0)\ \&\ \forall w\, (w < \overline{m} \supset {\sim}\varphi_g(w, \overline{n}, 0)).$$

Since $\varphi_g(x, z, y)$ represents $g(x, z)$ and $g(m, n) = 0$ if $f(n) = m$, we have

$$\mathbf{Q} \vdash \varphi_g(\overline{m}, \overline{n}, 0).$$

If $f(n) = m$, then for every $k < m$, $g(k, n) \neq 0$. So

$$\mathbf{Q} \vdash {\sim}\varphi_g(\overline{k}, \overline{n}, 0).$$

We get that

$$\mathbf{Q} \vdash \forall w\, (w < \overline{m} \supset {\sim}\varphi_g(w, \overline{n}, 0)). \tag{14.6}$$

by Lemma 14.22 (by (1) in case $m = 0$ and by (2) otherwise).

Now let's show that if $f(n) = m$, then $\mathbf{Q} \vdash \forall y\, (\varphi_f(\overline{n}, y) \supset y = \overline{m})$. We again sketch the argument informally, leaving the formalization to the reader.

Suppose $\varphi_f(\overline{n}, y)$. From this we get (a) $\varphi_g(y, \overline{n}, 0)$ and (b) $\forall w\, (w < y \supset {\sim}\varphi_g(w, \overline{n}, 0))$. By Lemma 14.23, $(y < \overline{m} \vee \overline{m} < y) \vee y = \overline{m}$. We'll show that both $y < \overline{m}$ and $\overline{m} < y$ leads to a contradiction.

If $\overline{m} < y$, then ${\sim}\varphi_g(\overline{m}, \overline{n}, 0)$ from (b). But $m = f(n)$, so $g(m, n) = 0$, and so $\mathbf{Q} \vdash \varphi_g(\overline{m}, \overline{n}, 0)$ since φ_g represents g. So we have a contradiction.

Now suppose $y < \overline{m}$. Then since $\mathbf{Q} \vdash \forall w\, (w < \overline{m} \supset {\sim}\varphi_g(w, \overline{n}, 0))$ by eq. (14.6), we get ${\sim}\varphi_g(y, \overline{n}, 0)$. This again contradicts (a). □

14.8 Computable Functions are Representable in Q

Theorem 14.25. *Every computable function is representable in* **Q**.

Proof. For definiteness, and using the Church-Turing Thesis, let's say that a function is computable iff it is general recursive. The general recursive functions are those which can be defined from the zero function zero, the successor function succ, and the projection function P_i^n using composition, primitive recursion, and regular minimization. By Lemma 14.8, any function h that can be defined from f and g can also be defined using composition and regular minimization from f, g, and zero, succ, P_i^n, add, mult, $\chi_=$. Consequently, a function is general recursive iff it can be defined from zero, succ, P_i^n, add, mult, $\chi_=$ using composition and regular minimization.

We've furthermore shown that the basic functions in question are representable in **Q** (Propositions 14.9 to 14.12, 14.14 and 14.16), and that any function defined from representable functions by composition or regular minimization (Proposition 14.20, Proposition 14.24) is also representable. Thus every general recursive function is representable in **Q**. □

We have shown that the set of computable functions can be characterized as the set of functions representable in **Q**. In fact, the proof is more general. From the definition of representability, it is not hard to see that any theory extending **Q** (or in which one can interpret **Q**) can represent the computable functions. But, conversely, in any proof system in which the notion of proof is computable, every representable function is computable. So, for example, the set of computable functions can be characterized as the set of functions representable in Peano arithmetic, or even Zermelo-Fraenkel set theory. As Gödel noted, this is somewhat surprising. We will see that when it comes to provability, questions are very sensitive to which theory you consider; roughly, the stronger the axioms, the more you can prove. But across a wide range of axiomatic theories, the representable functions are exactly the computable ones; stronger theories do not represent more functions as long as they are axiomatizable.

14.9 Representing Relations

Let us say what it means for a *relation* to be representable.

Definition 14.26. A relation $R(x_0, \ldots, x_k)$ on the natural numbers is *representable in* **Q** if there is a formula $\varphi_R(x_0, \ldots, x_k)$ such that whenever $R(n_0, \ldots, n_k)$ is true, **Q** proves $\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$, and whenever $R(n_0, \ldots, n_k)$ is false, **Q** proves $\sim\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$.

Theorem 14.27. *A relation is representable in* **Q** *if and only if it is computable.*

Proof. For the forwards direction, suppose $R(x_0, \ldots, x_k)$ is represented by the formula $\varphi_R(x_0, \ldots, x_k)$. Here is an algorithm for computing R: on input n_0, $\ldots$, n_k, simultaneously search for a proof of $\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$ and a proof of $\sim\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$. By our hypothesis, the search is bound to find one or the other; if it is the first, report "yes," and otherwise, report "no."

In the other direction, suppose $R(x_0, \ldots, x_k)$ is computable. By definition, this means that the function $\chi_R(x_0, \ldots, x_k)$ is computable. By Theorem 14.2, χ_R is represented by a formula, say $\varphi_{\chi_R}(x_0, \ldots, x_k, y)$. Let $\varphi_R(x_0, \ldots, x_k)$ be the formula $\varphi_{\chi_R}(x_0, \ldots, x_k, \overline{1})$. Then for any $n_0, \ldots, n_k$, if $R(n_0, \ldots, n_k)$ is true, then $\chi_R(n_0, \ldots, n_k) = 1$, in which case $\mathbf{Q}$ proves $\varphi_{\chi_R}(\overline{n_0}, \ldots, \overline{n_k}, \overline{1})$, and so $\mathbf{Q}$ proves $\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$. On the other hand, if $R(n_0, \ldots, n_k)$ is false, then $\chi_R(n_0, \ldots, n_k) = 0$. This means that $\mathbf{Q}$ proves

$$\forall y\, (\varphi_{\chi_R}(\overline{n_0}, \ldots, \overline{n_k}, y) \supset y = \overline{0}).$$

Since $\mathbf{Q}$ proves $\overline{0} \neq \overline{1}$, $\mathbf{Q}$ proves $\sim\varphi_{\chi_R}(\overline{n_0}, \ldots, \overline{n_k}, \overline{1})$, and so it proves $\sim\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$. □

14.10 Undecidability

We call a theory $\mathbf{T}$ *undecidable* if there is no computational procedure which, after finitely many steps and unfailingly, provides a correct answer to the question "does $\mathbf{T}$ prove φ?" for any sentence φ in the language of $\mathbf{T}$. So $\mathbf{Q}$ would be decidable iff there were a computational procedure which decides, given a sentence φ in the language of arithmetic, whether $\mathbf{Q} \vdash \varphi$ or not. We can make this more precise by asking: Is the relation $\mathrm{Prov}_{\mathbf{Q}}(y)$, which holds of y iff y is the Gödel number of a sentence provable in $\mathbf{Q}$, recursive? The answer is: no.

Theorem 14.28. $\mathbf{Q}$ *is undecidable, i.e., the relation*

$$\mathrm{Prov}_{\mathbf{Q}}(y) \Leftrightarrow \mathrm{Sent}(y)\ \&\ \exists x\, \mathrm{Prf}_{\mathbf{Q}}(x, y)$$

is not recursive.

Proof. Suppose it were. Then we could solve the halting problem as follows: Given e and n, we know that $\varphi_e(n) \downarrow$ iff there is an s such that $T(e, n, s)$, where T is Kleene's predicate from Theorem 12.8. Since T is primitive recursive it is representable in $\mathbf{Q}$ by a formula ψ_T, that is, $\mathbf{Q} \vdash \psi_T(\overline{e}, \overline{n}, \overline{s})$ iff $T(e, n, s)$. If $\mathbf{Q} \vdash \psi_T(\overline{e}, \overline{n}, \overline{s})$ then also $\mathbf{Q} \vdash \exists y\, \psi_T(\overline{e}, \overline{n}, y)$. If no such s exists, then $\mathbf{Q} \vdash \sim\psi_T(\overline{e}, \overline{n}, \overline{s})$ for every s. But $\mathbf{Q}$ is ω-consistent, i.e., if $\mathbf{Q} \vdash \sim\varphi(\overline{n})$ for every $n \in \mathbb{N}$, then $\mathbf{Q} \nvdash \exists y\, \varphi(y)$. We know this because the axioms of $\mathbf{Q}$ are true in the standard model $\mathfrak{N}$. So, $\mathbf{Q} \nvdash \exists y\, \psi_T(\overline{e}, \overline{n}, y)$. In other words, $\mathbf{Q} \vdash \exists y\, \psi_T(\overline{e}, \overline{n}, y)$ iff there is an s such that $T(e, n, s)$, i.e., iff $\varphi_e(n) \downarrow$. From e and n we can compute ${}^{\#}\exists y\, \psi_T(\overline{e}, \overline{n}, y)^{\#}$, let $g(e, n)$ be the primitive recursive function which

does that. So

$$h(e,n) = \begin{cases} 1 & \text{if } \mathrm{Prov}_{\mathbf{Q}}(g(e,n)) \\ 0 & \text{otherwise.} \end{cases}$$

This would show that h is recursive if $\mathrm{Prov}_{\mathbf{Q}}$ is. But h is not recursive, by Theorem 12.9, so $\mathrm{Prov}_{\mathbf{Q}}$ cannot be either. □

Corollary 14.29. *First-order logic is undecidable.*

Proof. If first-order logic were decidable, provability in **Q** would be as well, since $\mathbf{Q} \vdash \varphi$ iff $\vdash \omega \supset \varphi$, where ω is the conjunction of the axioms of **Q**. □

Chapter 15

Incompleteness and Provability

15.1 Introduction

Hilbert thought that a system of axioms for a mathematical structure, such as the natural numbers, is inadequate unless it allows one to derive all true statements about the structure. Combined with his later interest in formal systems of deduction, this suggests that he thought that we should guarantee that, say, the formal systems we are using to reason about the natural numbers is not only consistent, but also *complete*, i.e., every statement in its language is either provable or its negation is. Gödel's first incompleteness theorem shows that no such system of axioms exists: there is no complete, consistent, axiomatizable formal system for arithmetic. In fact, no "sufficiently strong," consistent, axiomatizable mathematical theory is complete.

A more important goal of Hilbert's, the centerpiece of his program for the justification of modern ("classical") mathematics, was to find finitary consistency proofs for formal systems representing classical reasoning. With regard to Hilbert's program, then, Gödel's second incompleteness theorem was a much bigger blow. The second incompleteness theorem can be stated in vague terms, like the first incompleteness theorem. Roughly speaking, it says that no sufficiently strong theory of arithmetic can prove its own consistency. We will have to take "sufficiently strong" to include a little bit more than **Q**.

The idea behind Gödel's original proof of the incompleteness theorem can be found in the Epimenides paradox. Epimenides, a Cretan, asserted that all Cretans are liars; a more direct form of the paradox is the assertion "this sentence is false." Essentially, by replacing truth with provability, Gödel was able to formalize a sentence which, in a roundabout way, asserts that it itself is not provable. If that sentence were provable, the theory would then be inconsistent. Assuming ω-consistency—a property stronger than consistency—Gödel was able to show that this sentence is also not refutable from the system of axioms he was considering.

The first challenge is to understand how one can construct a sentence that

refers to itself. For every formula φ in the language of **Q**, let $\ulcorner\varphi\urcorner$ denote the numeral corresponding to ${}^{\#}\varphi^{\#}$. Think about what this means: φ is a formula in the language of **Q**, ${}^{\#}\varphi^{\#}$ is a natural number, and $\ulcorner\varphi\urcorner$ is a *term* in the language of **Q**. So every formula φ in the language of **Q** has a *name*, $\ulcorner\varphi\urcorner$, which is a term in the language of **Q**; this provides us with a conceptual framework in which formulas in the language of **Q** can "say" things about other formulas. The following lemma is known as the fixed-point lemma.

Lemma 15.1. *Let* **T** *be any theory extending* **Q**, *and let* $\psi(x)$ *be any formula with only the variable* x *free. Then there is a sentence* φ *such that* **T** *proves* $\varphi \equiv \psi(\ulcorner\varphi\urcorner)$.

The lemma asserts that given any property $\psi(x)$, there is a sentence φ that asserts "$\psi(x)$ is true of me."

How can we construct such a sentence? Consider the following version of the Epimenides paradox, due to Quine:

> "Yields falsehood when preceded by its quotation" yields falsehood when preceded by its quotation.

This sentence is not directly self-referential. It simply makes an assertion about the syntactic objects between quotes, and, in doing so, it is on par with sentences like

1. "Robert" is a nice name.
2. "I ran." is a short sentence.
3. "Has three words" has three words.

But what happens when one takes the phrase "yields falsehood when preceded by its quotation," and precedes it with a quoted version of itself? Then one has the original sentence! In short, the sentence asserts that it is false.

15.2 The Fixed-Point Lemma

The fixed-point lemma says that for any formula $\psi(x)$, there is a sentence φ such that $\mathbf{T} \vdash \varphi \equiv \psi(\ulcorner\varphi\urcorner)$, provided **T** extends **Q**. In the case of the liar sentence, we'd want φ to be equivalent (provably in **T**) to "$\ulcorner\varphi\urcorner$ is false," i.e., the statement that ${}^{\#}\varphi^{\#}$ is the Gödel number of a false sentence. To understand the idea of the proof, it will be useful to compare it with Quine's informal gloss of φ as, "'yields a falsehood when preceded by its own quotation' yields a falsehood when preceded by its own quotation." The operation of taking an expression, and then forming a sentence by preceding this expression by its own quotation may be called *diagonalizing* the expression, and the result its diagonalization. So, the diagonalization of 'yields a falsehood when preceded by its own quotation' is "'yields a falsehood when preceded by its own quotation' yields a falsehood when preceded by its own quotation." Now note

that Quine's liar sentence is not the diagonalization of 'yields a falsehood' but of 'yields a falsehood when preceded by its own quotation.' So the property being diagonalized to yield the liar sentence itself involves diagonalization!

In the language of arithmetic, we form quotations of a formula with one free variable by computing its Gödel numbers and then substituting the standard numeral for that Gödel number into the free variable. The diagonalization of $\alpha(x)$ is $\alpha(\overline{n})$, where $n = {}^{\#}\alpha(x)^{\#}$. (From now on, let's abbreviate $\overline{{}^{\#}\alpha(x)^{\#}}$ as $\ulcorner\alpha(x)\urcorner$.) So if $\psi(x)$ is "is a falsehood," then "yields a falsehood if preceded by its own quotation," would be "yields a falsehood when applied to the Gödel number of its diagonalization." If we had a symbol $diag$ for the function $\mathrm{diag}(n)$ which computes the Gödel number of the diagonalization of the formula with Gödel number n, we could write $\alpha(x)$ as $\psi(diag(x))$. And Quine's version of the liar sentence would then be the diagonalization of it, i.e., $\alpha(\ulcorner\alpha\urcorner)$ or $\psi(diag(\ulcorner\psi(diag(x))\urcorner))$. Of course, $\psi(x)$ could now be any other property, and the same construction would work. For the incompleteness theorem, we'll take $\psi(x)$ to be "x is unprovable in $\mathbf{T}$." Then $\alpha(x)$ would be "yields a sentence unprovable in $\mathbf{T}$ when applied to the Gödel number of its diagonalization."

To formalize this in $\mathbf{T}$, we have to find a way to formalize diag. The function $\mathrm{diag}(n)$ is computable, in fact, it is primitive recursive: if n is the Gödel number of a formula $\alpha(x)$, $\mathrm{diag}(n)$ returns the Gödel number of $\alpha(\ulcorner\alpha(x)\urcorner)$. (Recall, $\ulcorner\alpha(x)\urcorner$ is the standard numeral of the Gödel number of $\alpha(x)$, i.e., $\overline{{}^{\#}\alpha(x)^{\#}}$). If $diag$ were a function symbol in $\mathbf{T}$ repqresenting the function diag, we could take φ to be the formula $\psi(diag(\ulcorner\psi(diag(x))\urcorner))$. Notice that

$$\begin{aligned}\mathrm{diag}({}^{\#}\psi(diag(x))^{\#}) &= {}^{\#}\psi(diag(\ulcorner\psi(diag(x))\urcorner)^{\#}\\ &= {}^{\#}\varphi^{\#}.\end{aligned}$$

Assuming $\mathbf{T}$ can prove

$$diag(\ulcorner\psi(diag(x))\urcorner) = \ulcorner\varphi\urcorner,$$

it can prove $\psi(diag(\ulcorner\psi(diag(x))\urcorner)) \equiv \psi(\ulcorner\varphi\urcorner)$. But the left hand side is, by definition, φ.

Of course, $diag$ will in general not be a function symbol of $\mathbf{T}$, and certainly is not one of $\mathbf{Q}$. But, since diag is computable, it is *representable* in $\mathbf{Q}$ by some formula $\theta_{\mathrm{diag}}(x,y)$. So instead of writing $\psi(diag(x))$ we can write $\exists y\,(\theta_{\mathrm{diag}}(x,y) \mathbin{\&} \psi(y))$. Otherwise, the proof sketched above goes through, and in fact, it goes through already in $\mathbf{Q}$.

Lemma 15.2. *Let $\psi(x)$ be any formula with one free variable x. Then there is a sentence φ such that $\mathbf{Q} \vdash \varphi \equiv \psi(\ulcorner\varphi\urcorner)$.*

Proof. Given $\psi(x)$, let $\alpha(x)$ be the formula $\exists y\,(\theta_{\mathrm{diag}}(x,y) \mathbin{\&} \psi(y))$ and let φ be its diagonalization, i.e., the formula $\alpha(\ulcorner\alpha(x)\urcorner)$.

Since θ_{diag} represents diag, and $\text{diag}(\ulcorner \alpha(x) \urcorner) = \ulcorner \varphi \urcorner$, **Q** can prove

$$!D_{\text{diag}}(\ulcorner \alpha(x) \urcorner, \ulcorner \varphi \urcorner) \tag{15.1}$$

$$\forall y\, (\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y) \supset y = \ulcorner \varphi \urcorner). \tag{15.2}$$

Now we show that $\mathbf{Q} \vdash \varphi \equiv \psi(\ulcorner \varphi \urcorner)$. We argue informally, using just logic and facts provable in **Q**.

First, suppose φ, i.e., $\alpha(\ulcorner \alpha(x) \urcorner)$. Going back to the definition of $\alpha(x)$, we see that $\alpha(\ulcorner \alpha(x) \urcorner)$ just is

$$\exists y\, (\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y) \mathbin{\&} \psi(y)).$$

Consider such a y. Since $\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y)$, by eq. (15.2), $y = \ulcorner \varphi \urcorner$. So, from $\psi(y)$ we have $\psi(\ulcorner \varphi \urcorner)$.

Now suppose $\psi(\ulcorner \varphi \urcorner)$. By eq. (15.1), we have $\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, \ulcorner \varphi \urcorner) \mathbin{\&} \psi(\ulcorner \varphi \urcorner)$. It follows that $\exists y\, (\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y) \mathbin{\&} \psi(y))$. But that's just $\alpha(\ulcorner \alpha \urcorner)$, i.e., φ. □

You should compare this to the proof of the fixed-point lemma in computability theory. The difference is that here we want to define a *statement* in terms of itself, whereas there we wanted to define a *function* in terms of itself; this difference aside, it is really the same idea.

15.3 The First Incompleteness Theorem

We can now describe Gödel's original proof of the first incompleteness theorem. Let **T** be any computably axiomatized theory in a language extending the language of arithmetic, such that **T** includes the axioms of **Q**. This means that, in particular, **T** represents computable functions and relations.

We have argued that, given a reasonable coding of formulas and proofs as numbers, the relation $\text{Prf}_T(x, y)$ is computable, where $\text{Prf}_T(x, y)$ holds if and only if x is the Gödel number of a derivation of the formula with Gödel number y in **T**. In fact, for the particular theory that Gödel had in mind, Gödel was able to show that this relation is primitive recursive, using the list of 45 functions and relations in his paper. The 45th relation, xBy, is just $\text{Prf}_T(x, y)$ for his particular choice of **T**. Remember that where Gödel uses the word "recursive" in his paper, we would now use the phrase "primitive recursive."

Since $\text{Prf}_T(x, y)$ is computable, it is representable in **T**. We will use $\mathsf{Prf}_T(x, y)$ to refer to the formula that represents it. Let $\mathsf{Prov}_T(y)$ be the formula $\exists x\, \mathsf{Prf}_T(x, y)$. This describes the 46th relation, $\text{Bew}(y)$, on Gödel's list. As Gödel notes, this is the only relation that "cannot be asserted to be recursive." What he probably meant is this: from the definition, it is not clear that it is computable; and later developments, in fact, show that it isn't.

Definition 15.3. A theory **T** is ω-consistent if the following holds: if $\exists x\, \varphi(x)$ is any sentence and **T** proves $\sim\varphi(\overline{0})$, $\sim\varphi(\overline{1})$, $\sim\varphi(\overline{2})$, ... then **T** does not prove $\exists x\, \varphi(x)$.

We can now prove the following.

Theorem 15.4. *Let* **T** *be any ω-consistent, axiomatizable theory extending* **Q**. *Then* **T** *is not complete.*

Proof. Let **T** be an axiomatizable theory containing **Q** Then $\mathsf{Prf}_T(x, y)$ is decidable, hence representable in **Q** by a formula $\mathsf{Prf}_T(x, y)$. Let $\mathsf{Prov}_T(y)$ be the formula we described above. By the fixed-point lemma, there is a formula $\gamma_{\mathbf{T}}$ such that **Q** (and hence **T**) proves

$$\gamma_{\mathbf{T}} \equiv \sim \mathsf{Prov}_T(\ulcorner \gamma_{\mathbf{T}} \urcorner). \tag{15.3}$$

Note that φ says, in essence, "φ is not provable."

We claim that

1. If **T** is consistent, **T** doesn't prove $\gamma_{\mathbf{T}}$
2. If **T** is ω-consistent, **T** doesn't prove $\sim\gamma_{\mathbf{T}}$.

This means that if **T** is ω-consistent, it is incomplete, since it proves neither $\gamma_{\mathbf{T}}$ nor $\sim\gamma_{\mathbf{T}}$. Let us take each claim in turn.

Suppose **T** proves $\gamma_{\mathbf{T}}$. Then there *is* a derivation, and so, for some number m, the relation $\mathsf{Prf}_T(m, {}^{\#}\gamma_{\mathbf{T}}{}^{\#})$ holds. But then **Q** proves the sentence $\mathsf{Prf}_T(\overline{m}, \ulcorner \gamma_{\mathbf{T}} \urcorner)$. So **Q** proves $\exists x\, \mathsf{Prf}_T(x, \ulcorner \gamma_{\mathbf{T}} \urcorner)$, which is, by definition, $\mathsf{Prov}_T(\ulcorner \gamma_{\mathbf{T}} \urcorner)$. By eq. (15.3), **Q** proves $\sim\gamma_{\mathbf{T}}$, and since **T** extends **Q**, so does **T**. We have shown that if **T** proves $\gamma_{\mathbf{T}}$, then it also proves $\sim\gamma_{\mathbf{T}}$, and hence it would be inconsistent.

For the second claim, let us show that if **T** proves $\sim\gamma_{\mathbf{T}}$, then it is ω-inconsistent. Suppose **T** proves $\sim\gamma_{\mathbf{T}}$. If **T** is inconsistent, it is ω-inconsistent, and we are done. Otherwise, **T** is consistent, so it does not prove $\gamma_{\mathbf{T}}$. Since there is no proof of $\gamma_{\mathbf{T}}$ in **T**, **Q** proves

$$\sim\mathsf{Prf}_T(\overline{0}, \ulcorner \gamma_{\mathbf{T}} \urcorner), \sim\mathsf{Prf}_T(\overline{1}, \ulcorner \gamma_{\mathbf{T}} \urcorner), \sim\mathsf{Prf}_T(\overline{2}, \ulcorner \gamma_{\mathbf{T}} \urcorner), \ldots$$

and so does **T**. On the other hand, by eq. (15.3), $\sim\gamma_{\mathbf{T}}$ is equivalent to $\exists x\, \mathsf{Prf}_T(x, \ulcorner \gamma_{\mathbf{T}} \urcorner)$. So **T** is ω-inconsistent. □

15.4 Rosser's Theorem

Can we modify Gödel's proof to get a stronger result, replacing "ω-consistent" with simply "consistent"? The answer is "yes," using a trick discovered by Rosser. Rosser's trick is to use a "modified" provability predicate $\mathsf{RProv}_T(y)$ instead of $\mathsf{Prov}_T(y)$.

Theorem 15.5. *Let* **T** *be any consistent, axiomatizable theory extending* **Q**. *Then* **T** *is not complete.*

Proof. Recall that $\mathsf{Prov}_T(y)$ is defined as $\exists x\, \mathsf{Prf}_T(x,y)$, where $\mathsf{Prf}_T(x,y)$ represents the decidable relation which holds iff x is the Gödel number of a derivation of the sentence with Gödel number y. The relation that holds between x and y if x is the Gödel number of a *refutation* of the sentence with Gödel number y is also decidable. Let $\mathrm{not}(x)$ be the primitive recursive function which does the following: if x is the code of a formula φ, $\mathrm{not}(x)$ is a code of $\sim\varphi$. Then $\mathsf{Ref}_T(x,y)$ holds iff $\mathsf{Prf}_T(x,\mathrm{not}(y))$. Let $\mathsf{Ref}_T(x,y)$ represent it. Then, if $\mathbf{T} \vdash \sim\varphi$ and δ is a corresponding derivation, $\mathbf{Q} \vdash \mathsf{Ref}_T(\ulcorner\delta\urcorner, \ulcorner\varphi\urcorner)$. We define $\mathsf{RProv}_T(y)$ as

$$\exists x\, (\mathsf{Prf}_T(x,y) \mathbin{\&} \forall z\, (z < x \supset \sim\mathsf{Ref}_T(z,y))).$$

Roughly, $\mathsf{RProv}_T(y)$ says "there is a proof of y in $\mathbf{T}$, and there is no shorter refutation of y." (You might find it convenient to read $\mathsf{RProv}_T(y)$ as "y is shmovable.") Assuming $\mathbf{T}$ is consistent, $\mathsf{RProv}_T(y)$ is true of the same numbers as $\mathsf{Prov}_T(y)$; but from the point of view of *provability* in $\mathbf{T}$ (and we now know that there is a difference between truth and provability!) the two have different properties. (If $\mathbf{T}$ is *in*consistent, then the two do *not* hold of the same numbers!)

By the fixed-point lemma, there is a formula $\rho_{\mathbf{T}}$ such that

$$\mathbf{Q} \vdash \rho_{\mathbf{T}} \equiv \sim\mathsf{RProv}_T(\ulcorner\rho_{\mathbf{T}}\urcorner). \tag{15.4}$$

In contrast to the proof of Theorem 15.4, here we claim that if $\mathbf{T}$ is consistent, $\mathbf{T}$ doesn't prove $\rho_{\mathbf{T}}$, and $\mathbf{T}$ also doesn't prove $\sim\rho_{\mathbf{T}}$. (In other words, we don't need the assumption of ω-consistency.)

First, let's show that $\mathbf{T} \nvdash \rho_T$. Suppose it did, so there is a derivation of ρ_T from T; let n be its Gödel number. Then $\mathbf{Q} \vdash \mathsf{Prf}_T(\overline{n}, \ulcorner\rho_T\urcorner)$, since Prf_T represents Prf_T in $\mathbf{Q}$. Also, for each $k < n$, k is not the Gödel number of $\sim\rho_T$, since $\mathbf{T}$ is consistent. So for each $k < n$, $\mathbf{Q} \vdash \sim\mathsf{Ref}_T(\overline{k}, \ulcorner\rho_T\urcorner)$. By Lemma 14.22(2), $\mathbf{Q} \vdash \forall z\, (z < \overline{n} \supset \sim\mathsf{Ref}_T(z, \ulcorner\rho_T\urcorner))$. Thus,

$$\mathbf{Q} \vdash \exists x\, (\mathsf{Prf}_T(x, \ulcorner\rho_T\urcorner) \mathbin{\&} \forall z\, (z < x \supset \sim\mathsf{Ref}_T(z, \ulcorner\rho_T\urcorner))),$$

but that's just $\mathsf{RProv}_T(\ulcorner\rho_T\urcorner)$. By eq. (15.4), $\mathbf{Q} \vdash \sim\rho_T$. Since $\mathbf{T}$ extends $\mathbf{Q}$, also $\mathbf{T} \vdash \sim\rho_T$. We've assumed that $\mathbf{T} \vdash \rho_T$, so $\mathbf{T}$ would be inconsistent, contrary to the assumption of the theorem.

Now, let's show that $\mathbf{T} \nvdash \sim\rho_T$. Again, suppose it did, and suppose n is the Gödel number of a derivation of $\sim\rho_T$. Then $\mathsf{Ref}_T(n, {}^{\#}\rho_T{}^{\#})$ holds, and since Ref_T represents Ref_T in $\mathbf{Q}$, $\mathbf{Q} \vdash \mathsf{Ref}_T(\overline{n}, \ulcorner\rho_T\urcorner)$. We'll again show that $\mathbf{T}$ would then be inconsistent because it would also prove ρ_T. Since $\mathbf{Q} \vdash \rho_T \equiv \sim\mathsf{RProv}_T(\ulcorner\rho_T\urcorner)$, and since $\mathbf{T}$ extends $\mathbf{Q}$, it suffices to show that $\mathbf{Q} \vdash \sim\mathsf{RProv}_T(\ulcorner\rho_T\urcorner)$. The sentence $\sim\mathsf{RProv}_T(\ulcorner\rho_T\urcorner)$, i.e.,

$$\sim\exists x\, (\mathsf{Prf}_T(x, \ulcorner\rho_T\urcorner) \mathbin{\&} \forall z\, (z < x \supset \sim\mathsf{Ref}_T(z, \ulcorner\rho_T\urcorner)))$$

is logically equivalent to

$$\forall x\,(\mathsf{Prf}_T(x,\ulcorner\rho_T\urcorner) \supset \exists z\,(z < x \,\&\, \mathsf{Ref}_T(z,\ulcorner\rho_T\urcorner)))$$

We argue informally using logic, making use of facts about what **Q** proves. Suppose x is arbitrary and $\mathsf{Prf}_T(x,\ulcorner\rho_T\urcorner)$. We already know that $\mathbf{T} \nvdash \rho_T$, and so for every k, $\mathbf{Q} \vdash \sim\mathsf{Prf}_T(\overline{k},\ulcorner\rho_T\urcorner)$. Thus, for every k it follows that $x \neq \overline{k}$. In particular, we have (a) that $x \neq \overline{n}$. We also have $\sim(x = \overline{0} \vee x = \overline{1} \vee \cdots \vee x = \overline{n-1})$ and so by Lemma 14.22(2), (b) $\sim(x < \overline{n})$. By Lemma 14.23, $\overline{n} < x$. Since $\mathbf{Q} \vdash \mathsf{Ref}_T(\overline{n},\ulcorner\rho_T\urcorner)$, we have $\overline{n} < x \,\&\, \mathsf{Ref}_T(\overline{n},\ulcorner\rho_T\urcorner)$, and from that $\exists z\,(z < x \,\&\, \mathsf{Ref}_T(z,\ulcorner\rho_T\urcorner))$. Since x was arbitrary we get

$$\forall x\,(\mathsf{Prf}_T(x,\ulcorner\rho_T\urcorner) \supset \exists z\,(z < x \,\&\, \mathsf{Ref}_T(z,\ulcorner\rho_T\urcorner)))$$

as required. □

15.5 Comparison with Gödel's Original Paper

It is worthwhile to spend some time with Gödel's 1931 paper. The introduction sketches the ideas we have just discussed. Even if you just skim through the paper, it is easy to see what is going on at each stage: first Gödel describes the formal system P (syntax, axioms, proof rules); then he defines the primitive recursive functions and relations; then he shows that xBy is primitive recursive, and argues that the primitive recursive functions and relations are represented in **P**. He then goes on to prove the incompleteness theorem, as above. In section 3, he shows that one can take the unprovable assertion to be a sentence in the language of arithmetic. This is the origin of the β-lemma, which is what we also used to handle sequences in showing that the recursive functions are representable in **Q**. Gödel doesn't go so far to isolate a minimal set of axioms that suffice, but we now know that **Q** will do the trick. Finally, in Section 4, he sketches a proof of the second incompleteness theorem.

15.6 The Provability Conditions for PA

Peano arithmetic, or **PA**, is the theory extending **Q** with induction axioms for all formulae. In other words, one adds to **Q** axioms of the form

$$(\varphi(0) \,\&\, \forall x\,(\varphi(x) \supset \varphi(x'))) \supset \forall x\,\varphi(x)$$

for every formula φ. Notice that this is really a *schema*, which is to say, infinitely many axioms (and it turns out that **PA** is *not* finitely axiomatizable). But since one can effectively determine whether or not a string of symbols is an instance of an induction axiom, the set of axioms for **PA** is computable. **PA** is a much more robust theory than **Q**. For example, one can easily prove that addition and multiplication are commutative, using induction in the usual

way. In fact, most finitary number-theoretic and combinatorial arguments can be carried out in **PA**.

Since **PA** is computably axiomatized, the provability predicate $\mathrm{Prf}_{\mathbf{PA}}(x, y)$ is computable and hence represented in **Q** (and so, in **PA**). As before, I will take $\mathsf{Prf}_{\mathbf{PA}}(x, y)$ to denote the formula representing the relation. Let $\mathsf{Prov}_{\mathbf{PA}}(y)$ be the formula $\exists x\, \mathsf{Prf}_{\mathbf{PA}}(x, y)$, which, intuitively says, "y is provable from the axioms of **PA**." The reason we need a little bit more than the axioms of **Q** is we need to know that the theory we are using is strong enough to prove a few basic facts about this provability predicate. In fact, what we need are the following facts:

P1. If $\mathbf{PA} \vdash \varphi$, then $\mathbf{PA} \vdash \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner)$

P2. For all formulae φ and ψ,

$$\mathbf{PA} \vdash \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \supset \psi \urcorner) \supset (\mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner) \supset \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \psi \urcorner))$$

P3. For every formula φ,

$$\mathbf{PA} \vdash \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner) \supset \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner) \urcorner).$$

The only way to verify that these three properties hold is to describe the formula $\mathsf{Prov}_{\mathbf{PA}}(y)$ carefully and use the axioms of **PA** to describe the relevant formal proofs. Conditions (1) and (2) are easy; it is really condition (3) that requires work. (Think about what kind of work it entails...) Carrying out the details would be tedious and uninteresting, so here we will ask you to take it on faith that **PA** has the three properties listed above. A reasonable choice of $\mathsf{Prov}_{\mathbf{PA}}(y)$ will also satisfy

P4. If $\mathbf{PA} \vdash \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner)$, then $\mathbf{PA} \vdash \varphi$.

But we will not need this fact.

Incidentally, Gödel was lazy in the same way we are being now. At the end of the 1931 paper, he sketches the proof of the second incompleteness theorem, and promises the details in a later paper. He never got around to it; since everyone who understood the argument believed that it could be carried out (he did not need to fill in the details.)

15.7 The Second Incompleteness Theorem

How can we express the assertion that **PA** doesn't prove its own consistency? Saying **PA** is inconsistent amounts to saying that **PA** proves $0 = 1$. So we can take $\mathsf{Con}_{\mathbf{PA}}$ to be the formula $\sim\mathsf{Prov}_{\mathbf{PA}}(\ulcorner 0 = 1 \urcorner)$, and then the following theorem does the job:

Theorem 15.6. *Assuming* **PA** *is consistent, then* **PA** *does not prove* $\mathsf{Con}_{\mathbf{PA}}$.

It is important to note that the theorem depends on the particular representation of $\mathsf{Con}_{\mathbf{PA}}$ (i.e., the particular representation of $\mathsf{Prov}_{\mathbf{PA}}(y)$). All we will use is that the representation of $\mathsf{Prov}_{\mathbf{PA}}(y)$ has the three properties above, so the theorem generalizes to any theory with a provability predicate having these properties.

It is informative to read Gödel's sketch of an argument, since the theorem follows like a good punch line. It goes like this. Let $\gamma_{\mathbf{PA}}$ be the Gödel sentence that we constructed in the proof of Theorem 15.4. We have shown "If **PA** is consistent, then **PA** does not prove $\gamma_{\mathbf{PA}}$." If we formalize this *in* **PA**, we have a proof of

$$\mathsf{Con}_{\mathbf{PA}} \supset \sim \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \gamma_{\mathbf{PA}} \urcorner).$$

Now suppose **PA** proves $\mathsf{Con}_{\mathbf{PA}}$. Then it proves $\sim \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \gamma_{\mathbf{PA}} \urcorner)$. But since $\gamma_{\mathbf{PA}}$ is a Gödel sentence, this is equivalent to $\gamma_{\mathbf{PA}}$. So **PA** proves $\gamma_{\mathbf{PA}}$.

But: we know that if **PA** is consistent, it doesn't prove $\gamma_{\mathbf{PA}}$! So if **PA** is consistent, it can't prove $\mathsf{Con}_{\mathbf{PA}}$.

To make the argument more precise, we will let $\gamma_{\mathbf{PA}}$ be the Gödel sentence for **PA** and use the provability conditions (1)–(3) above to show that **PA** proves $\mathsf{Con}_{\mathbf{PA}} \supset \gamma_{\mathbf{PA}}$. This will show that **PA** doesn't prove $\mathsf{Con}_{\mathbf{PA}}$. Here is a sketch

of the proof, in **PA**. (For simplicity, we drop the **PA** subscripts.)

$$!G \equiv \sim \mathsf{Prov}(\ulcorner \gamma \urcorner) \tag{15.5}$$

γ is a Gödel sentence

$$!G \supset \sim \mathsf{Prov}(\ulcorner \gamma \urcorner) \tag{15.6}$$

from eq. (15.5)

$$!G \supset (\mathsf{Prov}(\ulcorner \gamma \urcorner) \supset \bot) \tag{15.7}$$

from eq. (15.6) by logic

$$\mathsf{Prov}(\ulcorner \gamma \supset (\mathsf{Prov}(\ulcorner \gamma \urcorner) \supset \bot) \urcorner) \tag{15.8}$$

by from eq. (15.7) by condition P1

$$\mathsf{Prov}(\ulcorner \gamma \urcorner) \supset \mathsf{Prov}(\ulcorner (\mathsf{Prov}(\ulcorner \gamma \urcorner) \supset \bot) \urcorner) \tag{15.9}$$

from eq. (15.8) by condition P2

$$\mathsf{Prov}(\ulcorner \gamma \urcorner) \supset (\mathsf{Prov}(\ulcorner \mathsf{Prov}(\ulcorner \gamma \urcorner) \urcorner) \supset \mathsf{Prov}(\ulcorner \bot \urcorner)) \tag{15.10}$$

from eq. (15.9) by condition P2 and logic

$$\mathsf{Prov}(\ulcorner \gamma \urcorner) \supset \mathsf{Prov}(\ulcorner \mathsf{Prov}(\ulcorner \gamma \urcorner) \urcorner) \tag{15.11}$$

by P3

$$\mathsf{Prov}(\ulcorner \gamma \urcorner) \supset \mathsf{Prov}(\ulcorner \bot \urcorner) \tag{15.12}$$

from eq. (15.10) and eq. (15.11) by logic

$$\mathsf{Con} \supset \sim \mathsf{Prov}(\ulcorner \gamma \urcorner) \tag{15.13}$$

contraposition of eq. (15.12) and $\mathsf{Con} \equiv \sim \mathsf{Prov}(\ulcorner \bot \urcorner)$

$$\mathsf{Con} \supset \gamma$$

from eq. (15.5) and eq. (15.13) by logic

The use of logic in the above just elementary facts from propositional logic, e.g., eq. (15.7) uses $\vdash \sim\varphi \equiv (\varphi \supset \bot)$ and eq. (15.12) uses $\varphi \supset (\psi \supset \chi), \varphi \supset \psi \vdash \varphi \supset \chi$. The use of condition P2 in eq. (15.9) and eq. (15.10) relies on instances of P2, $\mathsf{Prov}(\ulcorner \varphi \supset \psi \urcorner) \supset (\mathsf{Prov}(\ulcorner \varphi \urcorner) \supset \mathsf{Prov}(\ulcorner \psi \urcorner))$. In the first one, $\varphi \equiv \gamma$ and $\psi \equiv \mathsf{Prov}(\ulcorner \gamma \urcorner) \supset \bot$; in the second, $\varphi \equiv \mathsf{Prov}(\ulcorner G \urcorner)$ and $\psi \equiv \bot$.

The more abstract version of the incompleteness theorem is as follows:

Theorem 15.7. *Let* **T** *be any axiomatized theory extending* **Q** *and let* $\mathsf{Prov}_T(y)$ *be any formula satisfying provability conditions P1–P3 for* **T**. *Then if* **T** *is consistent, then* **T** *does not prove* Con_T.

The moral of the story is that no "reasonable" consistent theory for mathematics can prove its own consistency. Suppose **T** is a theory of mathematics that includes **Q** and Hilbert's "finitary" reasoning (whatever that may be). Then, the whole of **T** cannot prove the consistency of **T**, and so, a fortiori, the finitary fragment can't prove the consistency of **T** either. In that sense, there cannot be a finitary consistency proof for "all of mathematics."

There is some leeway in interpreting the term "finitary," and Gödel, in the 1931 paper, grants the possibility that something we may consider "finitary" may lie outside the kinds of mathematics Hilbert wanted to formalize. But Gödel was being charitable; today, it is hard to see how we might find something that can reasonably be called finitary but is not formalizable in, say, **ZFC**.

15.8 Löb's Theorem

The Gödel sentence for a theory $\mathbf{T}$ is a fixed point of $\sim\mathsf{Prov}_T(x)$, i.e., a sentence γ such that

$$\mathbf{T} \vdash \sim\mathsf{Prov}_T(\ulcorner\gamma\urcorner) \equiv \gamma.$$

It is not provable, because if $\mathbf{T} \vdash \gamma$, (a) by provability condition (1), $\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\gamma\urcorner)$, and (b) $\mathbf{T} \vdash \gamma$ together with $\mathbf{T} \vdash \sim\mathsf{Prov}_T(\ulcorner\gamma\urcorner) \equiv \gamma$ gives $\mathbf{T} \vdash \sim\mathsf{Prov}_T(\ulcorner\gamma\urcorner)$, and so $\mathbf{T}$ would be inconsistent. Now it is natural to ask about the status of a fixed point of $\mathsf{Prov}_T(x)$, i.e., a sentence δ such that

$$\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\delta\urcorner) \equiv \delta.$$

If it were provable, $\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\delta\urcorner)$ by condition (1), but the same conclusion follows if we apply modus ponens to the equivalence above. Hence, we don't get that $\mathbf{T}$ is inconsistent, at least not by the same argument as in the case of the Gödel sentence. This of course does not show that $\mathbf{T}$ *does* prove δ.

We can make headway on this question if we generalize it a bit. The left-to-right direction of the fixed point equivalence, $\mathsf{Prov}_T(\ulcorner\delta\urcorner) \supset \delta$, is an instance of a general schema called a *reflection principle*: $\mathsf{Prov}_T(\ulcorner\varphi\urcorner) \supset \varphi$. It is called that because it expresses, in a sense, that $\mathbf{T}$ can "reflect" about what it can prove; basically it says, "If $\mathbf{T}$ can prove φ, then φ is true," for any φ. This is true for sound theories only, of course, and this suggests that theories will in general not prove every instance of it. So which instances can a theory (strong enough, and satisfying the provability conditions) prove? Certainly all those where φ itself is provable. And that's it, as the next result shows.

Theorem 15.8. *Let $\mathbf{T}$ be an axiomatizable theory extending $\mathbf{Q}$, and suppose $\mathsf{Prov}_T(y)$ is a formula satisfying conditions P1–P3 from section 15.7. If $\mathbf{T}$ proves $\mathsf{Prov}_T(\ulcorner\varphi\urcorner) \supset \varphi$, then in fact $\mathbf{T}$ proves φ.*

Put differently, if $\mathbf{T} \nvdash \varphi$, then $\mathbf{T} \nvdash \mathsf{Prov}_T(\ulcorner\varphi\urcorner) \supset \varphi$. This result is known as Löb's theorem.

The heuristic for the proof of Löb's theorem is a clever proof that Santa Claus exists. (If you don't like that conclusion, you are free to substitute any other conclusion you would like.) Here it is:

1. Let X be the sentence, "If X is true, then Santa Claus exists."

2. Suppose X is true.

3. Then what it says holds; i.e., we have: if X is true, then Santa Claus exists.

4. Since we are assuming X is true, we can conclude that Santa Claus exists, by modus ponens from (2) and (3).

5. We have succeeded in deriving (4), "Santa Claus exists," from the assumption (2), "X is true." By conditional proof, we have shown: "If X is true, then Santa Claus exists."

6. But this is just the sentence X. So we have shown that X is true.

7. But then, by the argument (2)–(4) above, Santa Claus exists.

A formalization of this idea, replacing "is true" with "is provable," and "Santa Claus exists" with φ, yields the proof of Löb's theorem. The trick is to apply the fixed-point lemma to the formula $\mathsf{Prov}_T(y) \supset \varphi$. The fixed point of that corresponds to the sentence X in the preceding sketch.

Proof. Suppose φ is a sentence such that $\mathbf{T}$ proves $\mathsf{Prov}_T(\ulcorner \varphi \urcorner) \supset \varphi$. Let $\psi(y)$ be the formula $\mathsf{Prov}_T(y) \supset \varphi$, and use the fixed-point lemma to find a sentence θ

such that $\mathbf{T}$ proves $\theta \equiv \psi(\ulcorner\theta\urcorner)$. Then each of the following is provable in $\mathbf{T}$:

$$!D \equiv (\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset \varphi) \tag{15.14}$$

θ is a fixed point of $\psi(y)$

$$!D \supset (\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset \varphi) \tag{15.15}$$

from eq. (15.14)

$$\mathsf{Prov}_T(\ulcorner\theta \supset (\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset \varphi)\urcorner) \tag{15.16}$$

from eq. (15.15) by condition P1

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset \mathsf{Prov}_T(\ulcorner\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset \varphi\urcorner) \tag{15.17}$$

from eq. (15.16) using condition P2

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset (\mathsf{Prov}_T(\ulcorner\mathsf{Prov}_T(\ulcorner\theta\urcorner)\urcorner) \supset \mathsf{Prov}_T(\ulcorner\varphi\urcorner)) \tag{15.18}$$

from eq. (15.17) using P2 again

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset \mathsf{Prov}_T(\ulcorner\mathsf{Prov}_T(\ulcorner\theta\urcorner)\urcorner) \tag{15.19}$$

by provability condition P3

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset \mathsf{Prov}_T(\ulcorner\varphi\urcorner) \tag{15.20}$$

from eq. (15.18) and eq. (15.19)

$$\mathsf{Prov}_T(\ulcorner\varphi\urcorner) \supset \varphi \tag{15.21}$$

by assumption of the theorem

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset \varphi \tag{15.22}$$

from eq. (15.20) and eq. (15.21)

$$(\mathsf{Prov}_T(\ulcorner\theta\urcorner) \supset \varphi) \supset \theta \tag{15.23}$$

from eq. (15.14)

$$!D \tag{15.24}$$

from eq. (15.22) and eq. (15.23)

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \tag{15.25}$$

from eq. (15.24) by condition P1

$!A$ from eq. (15.21) and eq. (15.25)

□

With Löb's theorem in hand, there is a short proof of the first incompleteness theorem (for theories having a provability predicate satisfying conditions P1–P3: if $\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\bot\urcorner) \supset \bot$, then $\mathbf{T} \vdash \bot$. If $\mathbf{T}$ is consistent, $\mathbf{T} \nvdash \bot$. So, $\mathbf{T} \nvdash \mathsf{Prov}_T(\ulcorner\bot\urcorner) \supset \bot$, i.e., $\mathbf{T} \nvdash \mathsf{Con}_{\mathbf{T}}$. We can also apply it to show that δ, the fixed point of $\mathsf{Prov}_T(x)$, is provable. For since

$$\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\delta\urcorner) \equiv \delta$$

in particular

$$\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner \delta \urcorner) \supset \delta$$

and so by Löb's theorem, $\mathbf{T} \vdash \delta$.

15.9 The Undefinability of Truth

The notion of *definability* depends on having a formal semantics for the language of arithmetic. We have described a set of formulas and sentences in the language of arithmetic. The "intended interpretation" is to read such sentences as making assertions about the natural numbers, and such an assertion can be true or false. Let $\mathfrak{N}$ be the structure with domain $\mathbb{N}$ and the standard interpretation for the symbols in the language of arithmetic. Then $\mathfrak{N} \models \varphi$ means "φ is true in the standard interpretation."

Definition 15.9. A relation $R(x_1, \ldots, x_k)$ of natural numbers is *definable* in $\mathfrak{N}$ if and only if there is a formula $\varphi(x_1, \ldots, x_k)$ in the language of arithmetic such that for every $n_1, \ldots, n_k$, $R(n_1, \ldots, n_k)$ if and only if $\mathfrak{N} \models \varphi(\overline{n}_1, \ldots, \overline{n}_k)$.

Put differently, a relation is definable in in $\mathfrak{N}$ if and only if it is representable in the theory **TA**, where $\mathbf{TA} = \{\varphi \mid \mathfrak{N} \models \varphi\}$ is the set of true sentences of arithmetic. (If this is not immediately clear to you, you should go back and check the definitions and convince yourself that this is the case.)

Lemma 15.10. *Every computable relation is definable in* $\mathfrak{N}$.

Proof. It is easy to check that the formula representing a relation in **Q** defines the same relation in $\mathfrak{N}$. □

Now one can ask, is the converse also true? That is, is every relation definable in $\mathfrak{N}$ computable? The answer is no. For example:

Lemma 15.11. *The halting relation is definable in* $\mathfrak{N}$.

Proof. Let H be the halting relation, i.e.,

$$H = \{\langle e, x\rangle \mid \exists s\, T(e, x, s)\}.$$

Let θ_T define T in $\mathfrak{N}$. Then

$$H = \{\langle e, x\rangle \mid \mathfrak{N} \models \exists s\, \theta_T(\overline{e}, \overline{x}, s)\},$$

so $\exists s\, \theta_T(z, x, s)$ defines H in $\mathfrak{N}$. □

What about **TA** itself? Is it definable in arithmetic? That is: is the set $\{{}^{\#}\varphi^{\#} \mid \mathfrak{N} \models \varphi\}$ definable in arithmetic? Tarski's theorem answers this in the negative.

Theorem 15.12. *The set of true statements of arithmetic is not definable in arithmetic.*

Proof. Suppose $\theta(x)$ defined it. By the fixed-point lemma, there is a formula φ such that **Q** proves $\varphi \equiv \sim\theta(\ulcorner\varphi\urcorner)$, and hence $\mathfrak{N} \models \varphi \equiv \sim\theta(\ulcorner\varphi\urcorner)$. But then $\mathfrak{N} \models \varphi$ if and only if $\mathfrak{N} \models \sim\theta(\ulcorner\varphi\urcorner)$, which contradicts the fact that $\theta(y)$ is supposed to define the set of true statements of arithmetic. □

Tarski applied this analysis to a more general philosophical notion of truth. Given any language L, Tarski argued that an adequate notion of truth for L would have to satisfy, for each sentence X,

> 'X' is true if and only if X.

Tarski's oft-quoted example, for English, is the sentence

> 'Snow is white' is true if and only if snow is white.

However, for any language strong enough to represent the diagonal function, and any linguistic predicate $T(x)$, we can construct a sentence X satisfying "X if and only if not $T(\text{'}X\text{'})$." Given that we do not want a truth predicate to declare some sentences to be both true and false, Tarski concluded that one cannot specify a truth predicate for all sentences in a language without, somehow, stepping outside the bounds of the language. In other words, a the truth predicate for a language cannot be defined in the language itself.