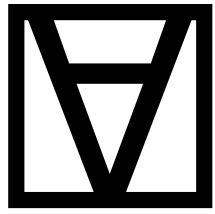
Intermediate Logic

Richard Zach

Philosophy 310
Winter Term 2015
McGill University



Contents

Preface **v**

I Sets, Relations, Functions **1**

1 Sets **2**
1.1 Basics . . . 2
1.2 Some Important Sets . . . 3
1.3 Subsets . . . 4
1.4 Unions and Intersections . . . 5
1.5 Proofs about Sets . . . 6
1.6 Pairs, Tuples, Cartesian Products . . . 7
1.7 Russell's Paradox . . . 8

2 Relations **10**
2.1 Relations as Sets . . . 10
2.2 Special Properties of Relations . . . 11
2.3 Orders . . . 12
2.4 Graphs . . . 14
2.5 Operations on Relations . . . 15

3 Functions **17**
3.1 Basics . . . 17
3.2 Kinds of Functions . . . 18
3.3 Inverses of Functions . . . 19
3.4 Composition of Functions . . . 20
3.5 Isomorphism . . . 20
3.6 Partial Functions . . . 21
3.7 Functions and Relations . . . 21

4 The Size of Sets **22**
4.1 Introduction . . . 22
4.2 Enumerable Sets . . . 22

4.3 Non-enumerable Sets . . . 26
4.4 Reduction . . . 29
4.5 Equinumerous Sets . . . 30
4.6 Comparing Sizes of Sets . . . 31

II First-Order Logic 33

5 Syntax and Semantics 34
5.1 Introduction . . . 34
5.2 First-Order Languages . . . 35
5.3 Terms and Formulas . . . 37
5.4 Unique Readability . . . 39
5.5 Main operator of a Formula . . . 41
5.6 Subformulas . . . 42
5.7 Free Variables and Sentences . . . 43
5.8 Substitution . . . 44
5.9 Structures for First-order Languages . . . 46
5.10 Covered Structures for First-order Languages . . . 47
5.11 Satisfaction of a Formula in a Structure . . . 48
5.12 Extensionality . . . 52
5.13 Semantic Notions . . . 53

6 Theories and Their Models 55
6.1 Introduction . . . 55
6.2 Expressing Properties of Structures . . . 57
6.3 Examples of First-Order Theories . . . 57
6.4 Expressing Relations in a Structure . . . 60
6.5 The Theory of Sets . . . 61
6.6 Expressing the Size of Structures . . . 63

III Proofs and Completeness 65

7 The Sequent Calculus 66
7.1 Rules and Derivations . . . 66
7.2 Examples of Derivations . . . 68
7.3 Proof-Theoretic Notions . . . 72
7.4 Properties of Derivability . . . 73
7.5 Soundness . . . 76
7.6 Derivations with Identity predicate . . . 80

8 The Completeness Theorem 82
8.1 Introduction . . . 82
8.2 Outline of the Proof . . . 82

8.3 Maximally Consistent Sets of Sentences . . . 84
8.4 Henkin Expansion . . . 86
8.5 Lindenbaum's Lemma . . . 87
8.6 Construction of a Model . . . 88
8.7 Identity . . . 89
8.8 The Completeness Theorem . . . 91
8.9 The Compactness Theorem . . . 92
8.10 The Löwenheim-Skolem Theorem . . . 94
8.11 Overspill . . . 95

IV Computability and Incompleteness **96**

9 Recursive Functions **97**
9.1 Introduction . . . 97
9.2 Primitive Recursion . . . 98
9.3 Primitive Recursive Functions are Computable . . . 101
9.4 Examples of Primitive Recursive Functions . . . 102
9.5 Primitive Recursive Relations . . . 103
9.6 Bounded Minimization . . . 105
9.7 Primes . . . 106
9.8 Sequences . . . 107
9.9 Other Recursions . . . 109
9.10 Non-Primitive Recursive Functions . . . 110
9.11 Partial Recursive Functions . . . 112
9.12 The Normal Form Theorem . . . 114
9.13 The Halting Problem . . . 114
9.14 General Recursive Functions . . . 116

10 Arithmetization of Syntax **117**
10.1 Introduction . . . 117
10.2 Coding Symbols . . . 118
10.3 Coding Terms . . . 120
10.4 Coding Formulas . . . 122
10.5 Substitution . . . 123
10.6 Derivations in **LK** . . . 124

11 Representability in Q **128**
11.1 Introduction . . . 128
11.2 Functions Representable in **Q** are Computable . . . 130
11.3 The Beta Function Lemma . . . 131
11.4 Simulating Primitive Recursion . . . 134
11.5 Basic Functions are Representable in **Q** . . . 135
11.6 Composition is Representable in **Q** . . . 137

11.7 Regular Minimization is Representable in **Q** 139
11.8 Computable Functions are Representable in **Q** 142
11.9 Representing Relations 142
11.10 Undecidability 143

12 Incompleteness and Provability **145**
12.1 Introduction 145
12.2 The Fixed-Point Lemma 146
12.3 The First Incompleteness Theorem 148
12.4 Rosser's Theorem 149
12.5 Comparison with Gödel's Original Paper 151
12.6 The Provability Conditions for **PA** 151
12.7 The Second Incompleteness Theorem 152
12.8 Löb's Theorem 155
12.9 The Undefinability of Truth 158

Problems **160**

Preface

Formal logic has many applications both within philosophy and outside (especially in mathematics, computer science, and linguistics). This second course will introduce you to the concepts, results, and methods of formal logic necessary to understand and appreciate these applications as well as the limitations of formal logic. It will be mathematical in that you will be required to master abstract formal concepts and to prove theorems *about* logic (not just *in* logic the way you did in Phil 210); but it does not presuppose any advanced knowledge of mathematics.

We will begin by studying some basic formal concepts: sets, relations, and functions and sizes of infinite sets. We will then consider the language, semantics, and proof theory of first-order logic (FOL), and ways in which we can use first-order logic to formalize facts and reasoning abouts some domains of interest to philosophers and logicians.

In the second part of the course, we will begin to investigate the metatheory of first-order logic. We will concentrate on a few central results: the completeness theorem, which relates the proof theory and semantics of first-order logic, and the compactness theorem and Löwenheim-Skolem theorems, which concern the existence and size of first-order interpretations.

In the third part of the course, we will discuss a particular way of making precise what it means for a function to be computable, namely, when it is recursive. This will enable us to prove important results in the metatheory of logic and of formal systems formulated in first-order logic: Gödel's incompleteness theorem, the Church-Turing undecidability theorem, and Tarski's theorem about the undefinability of truth.

Week 1 (Jan 5, 7). Introduction. Sets and Relations.

Week 2 (Jan 12, 14). Functions. Enumerability.

Week 3 (Jan 19, 21). Syntax and Semantics of FOL.

Week 4 (Jan 26, 28). Structures and Theories.

Week 5 (Feb 2, 5). Sequent Calculus and Proofs in FOL.

Week 6 (Feb 9, 12). The Completeness Theorem.

Week 7 (Feb 16, 18). Compactness and Löwenheim-Skolem Theorems

Week 8 (Mar 23, 25). Recursive Functions

Week 9 (Mar 9, 11). Arithmetization of Syntax

Week 10 (Mar 16, 18). Theories and Computability

Week 11 (Mar 23, 25). Gödel's Incompleteness Theorems

Week 12 (Mar 30, Apr 1). The Undefinability of Truth.

Week 13, 14 (Apr 8, 13). Applications.

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: $\{x : \ldots x \ldots\}$, 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

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

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

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

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 ($\mathbb{Q} = \{z/n : z \in \mathbb{Z}, n \in \mathbb{N}, n \neq 0\}$); 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 "enumerable and infinite" whereas $\mathbb{R}$ is "non-enumerable".

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 $a_1 a_2 a_3 a_4 \ldots$ 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 $\wp(X)$.

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

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

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.

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

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: $\{a, b, c\} \cup \emptyset = \{a, b, c\}$.

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.

$$X \cap Y = \{x : x \in X \wedge x \in Y\}$$

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:

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

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:

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

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, ...

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

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.,

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

1.5 Proofs about Sets

Sets and the notations we've introduced so far provide us with convenient shorthands for specifying sets and expressing relationships between them. Often it will also be necessary to prove claims about such relationships. If you're not familiar with mathematical proofs, this may be new to you. So we'll walk through a simple example. We'll prove that for any sets X and Y, it's always the case that $X \cap (X \cup Y) = X$. How do you prove an identity between sets like this? Recall that sets are determined solely by their elements, i.e., sets are identical iff they have the same elements. So in this case we have to prove that (a) every element of $X \cap (X \cup Y)$ is also an element of X and, conversely, that (b) every element of X is also an element of $X \cap (X \cup Y)$. In other words, we show that both (a) $X \cap (X \cup Y) \subseteq X$ and (b) $X \subseteq X \cap (X \cup Y)$.

A proof of a general claim like "every element z of $X \cap (X \cup Y)$ is also an element of X" is proved by first assuming that an arbitrary $z \in X \cap (X \cup Y)$ is given, and proving from this assumtion that $z \in X$. You may know this pattern as "general conditional proof." In this proof we'll also have to make use of the definitions involved in the assumption and conclusion, e.g., in this case of "$\cap$" and "$\cup$." So case (a) would be argued as follows:

(a) We first want to show that $X \cap (X \cup Y) \subseteq X$, i.e., by definition of $\subseteq$, that if $z \in X \cap (X \cup Y)$ then $z \in X$, for any z. So assume that $z \in X \cap (X \cup Y)$. Since z is an element of the intersection of two sets iff it is an element of both sets, we can conclude that $z \in X$ and also $z \in X \cup Y$. In particular, $z \in X$, which is what we wanted to show.

This completes the first half of the proof. Note that in the last step we used the fact that if a conjunction ($z \in X$ and $z \in X \cup Y$) follows from an assumption, each conjunct follows from that same assumption. You may know this rule as "conjunction elimination," or $\wedge$Elim. Now let's prove (b):

(b) We now prove that $X \subseteq X \cap (X \cup Y)$, i.e., by definition of $\subseteq$, that if $z \in X$ then also $z \in X \cap (X \cup Y)$, for any z. Assume $z \in X$. To show that $z \in X \cap (X \cup Y)$, we have to show (by definition of "$\cap$") that (i) $z \in X$ and also (ii) $z \in X \cup Y$. Here (i) is just our assumption, so there is nothing further to prove. For (ii), recall that z is an element of a union of sets iff it is an element of at least one of those sets. Since $z \in X$, and $X \cup Y$ is the union of X and Y, this is the case here. So $z \in X \cup Y$. We've shown both (i) $z \in X$ and (ii) $z \in X \cup Y$, hence, by definition of "$\cap$," $z \in X \cap (X \cup Y)$.

This was somewhat long-winded, but it illustrates how we reason about sets and their relationships. We usually aren't this explicit; in particular, we might not repeat all the definitions. A proof of our result in a more advanced text would be much more compressed. It might look something like this.

Proposition 1.19 (Absorption)**.** *For all sets X, Y,*

$$X \cap (X \cup Y) = X$$

Proof. (a) Suppose $z \in X \cap (X \cup Y)$. Then $z \in X$, so $X \cap (X \cup Y) \subseteq X$.

(b) Now suppose $z \in X$. Then also $z \in X \cup Y$, and therefore also $z \in X \cap (X \cup Y)$. Thus, $X \subseteq X \cap (X \cup Y)$. □

1.6 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* $\langle x, y \rangle$, or more generally, *ordered n-tuples* $\langle x_1, \ldots, x_n \rangle$.

Definition 1.20 (Cartesian product)**.** Given sets X and Y, their *Cartesian product* $X \times Y$ is $\{\langle x, y \rangle : x \in X \text{ and } y \in Y\}$.

Example 1.21. 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 1.22. 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.

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

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

1.7 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

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

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

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

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.,

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

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:

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

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 $\mathrm{Id}_X = \{\langle x, x\rangle : x \in X\}$ 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 $E = \{\langle n, m\rangle : n > 5 \text{ or } m \times n \geq 34\}$ 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 $\neg Rxy$ or $\neg Ryx$).

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. For instance, the identity relation and the full relation on X are preorders, but they are not partial orders, because they are not anti-symmetric (if X has more than one element). 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, even a connected preorder, but not a partial order.

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 $x \in X$, $\neg Rxx$.

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 $\subsetneq$.

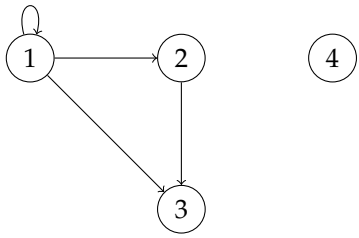
2.4 Graphs

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 descrete 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.

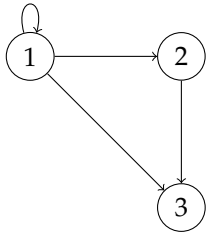
Definition 2.21 (Directed graph)**.** A *directed graph* $G = \langle V, E\rangle$ is a set of *vertices* V and a set of *edges* $E \subseteq V^2$.

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 $R^{-1} = \{\langle y, x\rangle : \langle x, y\rangle \in R\}$.
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 $R^* = R^+ \cup I_X$.

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

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 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

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

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 $x = x'$.

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

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 $g(y) = y - 1$ "undos" 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"

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

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 domains and codomains match, i.e., the codomain 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 $f\colon X \to Y$ 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 $f\colon X \to Y$ 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* $f\colon X \nrightarrow Y$ 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., $\mathrm{dom}(f) = \{x : f(x)\downarrow\}$.

Example 3.16. Every function $f\colon X \to Y$ 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 $f\colon \mathbb{R} \nrightarrow \mathbb{R}$ 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 $f\colon X \nrightarrow Y$ be a partial function. The *graph* of f is the relation $R_f \subseteq X \times Y$ defined by

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

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 $f\colon X \nrightarrow Y$ 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 enumerable. Cantor's surprising result was that some infinite sets are not enumerable.

4.2 Enumerable 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 *enumerable*. If X is enumerable and infinite, we say X is denumerable.

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 enumerable 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 last 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 enumerable 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

$$\begin{aligned} f(n) &= 2n \text{ and} \\ g(n) &= 2n + 1 \end{aligned}$$

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?

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

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?

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

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,

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

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 Non-enumerable 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 enumerable. However, there are also infinite sets which do not have this property. Such sets are called *non-enumerable*.

First of all, it is perhaps already surprising that there are non-enumerable sets. For any enumerable set X there is a surjective function $f\colon \mathbb{Z}^+ \to X$. If a set is non-enumerable 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 non-enumerable? 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 non-enumerable.*

Proof. We proceed by indirect proof. Suppose that $\mathbb{B}^\omega$ were enumerable, i.e., suppose that there is a list $s_1, s_2, s_3, s_4, \ldots$ 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 enumerable by using a similar idea even when no array and no actual diagonal is involved.

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

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 $n \notin Z_n$:

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

$\overline{Z}$ 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 non-enumerable by a diagonalization argument. We already had a proof that $\mathbb{B}^\omega$, the set of all infinite sequences of 0s and 1s, is non-enumerable. Here's another way we can prove that $\wp(\mathbb{Z}^+)$ is non-enumerable: Show that *if* $\wp(\mathbb{Z}^+)$ *is enumerable then* $\mathbb{B}^\omega$ *is also enumerable*. Since we know $\mathbb{B}^\omega$ is not enumerable, $\wp(\mathbb{Z}^+)$ 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 enumerable, 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:

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

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 enumerable, $\mathbb{B}^\omega$ would be enumerable. But $\mathbb{B}^\omega$ is non-enumerable (Theorem 4.8). Hence $\wp(\mathbb{Z}^+)$ is non-enumerable. □

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 non-enumerable. (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 enumerable.

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 enumerable if and only if Y is.*

Proof. Let X and Y be equinumerous. Suppose that X is enumerable. 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 enumerable. □

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

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

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 $Z_1, Z_2, \ldots$, 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 $Z_1, Z_2, \ldots$, $\overline{Y}$ 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 formulas. Terms are formed from variables, constant symbols, and function symbols. Formulas, in turn, are formed from predicate symbols together with terms (these form the smallest, "atomic" formulas), and then from atomic formulas 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 formulas *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 formulas, 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 $\varphi \wedge \psi$ 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, $\models \varphi$, if every structure satisfies it. It is entailed by a set of sentences, $\Gamma \models \varphi$, 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 formulas are inductively defined, and satisfaction is in turn defined by induction on the structure of formulas, 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 *formulas* 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: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (conditional), $\forall$ (universal quantifier), $\exists$ (existential quantifier).
    b) The propositional constant for falsity $\bot$.
    c) The two-place identity predicate $=$.
    d) A denumerable set of variables: v_0, v_1, v_2, ...
2. Non-logical symbols, making up the *standard language* of first-order logic
    a) A denumerable set of n-place predicate symbols for each $n > 0$: A^n_0, A^n_1, A^n_2, ...
    b) A denumerable set of constant symbols: c_0, c_1, c_2, ...

c) A denumerable set of n-place function symbols for each $n > 0$: f^n_0, f^n_1, f^n_2, ...

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: $\leftrightarrow$ (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 $\{\neg, \vee\}$, $\{\neg, \wedge\}$, and $\{\neg, \rightarrow\}$—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 Formulas

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 *formulas*.

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 *formulas* $\mathrm{Frm}(\mathcal{L})$ 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 $\neg\varphi$ is formula.
5. If φ and ψ are formulas, then $(\varphi \wedge \psi)$ is a formula.
6. If φ and ψ are formulas, then $(\varphi \vee \psi)$ is a formula.

7. If φ and ψ are formulas, then $(\varphi \to \psi)$ 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 formulas are *inductive definitions*. Essentially, we construct the set of formulas 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 formulas out of formulas already constructed. At the second stage, we can use them to construct formulas out of atomic formulas. 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, $\neg=(t_1, t_2)$ 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 formulas. 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 $\neg\bot$.
2. $\varphi \leftrightarrow \psi$ abbreviates $(\varphi \to \psi) \wedge (\psi \to \varphi)$.

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 $f^1_1(t)$, 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 formulas 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 formulas and only one way of "interpreting" it. If this were not so, we would have ambiguous formulas, i.e., formulas 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 formulas 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 formulas, then so is $\varphi \to \psi$.

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

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

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

Correspondingly, there are two ways to "read" the formula $\theta \to \theta \to \theta$. It is of the form $\psi \to \chi$ where ψ is θ and χ is $\theta \to \theta$, but *it is also* of the form $\psi \to \chi$ with ψ being $\theta \to \theta$ 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 $\psi \to \chi$, the main operator is the indicated occurrence of $\to$. But if we can match the formula $\theta \to \theta \to \theta$ with $\psi \to \chi$ in the two different ways mentioned above, then in one case we get the first occurrence of $\to$ 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. $\varphi \equiv \neg\psi$: 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* $\neg\psi$.
3. φ *is of the form* $(\psi \wedge \chi)$.
4. φ *is of the form* $(\psi \vee \chi)$.
5. φ *is of the form* $(\psi \to \chi)$.
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* $(\psi \to \chi)$ *and* $(\psi' \to \chi')$.

Proof. The formation rules require that if a formula is not atomic, it must start with an opening parenthesis (, $\neg$, 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 $\chi \not\equiv \chi'$. Since χ and χ' are both substrings of φ that begin at the same place, one must be a 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 $\neg\varphi$ is $\neg$, 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. $\varphi \equiv \neg\psi$: the main operator of φ is $\neg$.

3. $\varphi \equiv (\psi \wedge \chi)$: the main operator of φ is $\wedge$.
4. $\varphi \equiv (\psi \vee \chi)$: the main operator of φ is $\vee$.
5. $\varphi \equiv (\psi \to \chi)$: the main operator of φ is $\to$.
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 $((\theta \to \alpha) \to (\alpha \to \theta))$ is of the form $(\psi \to \chi)$ where ψ is $(\theta \to \alpha)$ and χ is $(\alpha \to \theta)$, the second occurrence of $\to$ is the main operator.

This is a *recursive* definition of a function which maps all non-atomic formulas to their main operator occurrence. Because of the way formulas 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 formulas 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 formulas by the following names depending on which symbol their main operator is:

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

5.6 Subformulas

It is often useful to talk about the formulas that "make up" a given formula. We call these its *subformulas*. 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 subformulas* of φ are defined inductively as follows:

1. Atomic formulas have no immediate subformulas.
2. $\varphi \equiv \neg\psi$: The only immediate subformula of φ is ψ.
3. $\varphi \equiv (\psi * \chi)$: The immediate subformulas 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 subformulas* of φ are recursively as follows:

1. Atomic formulas have no proper subformulas.

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

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

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

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

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

Note the subtle difference in how we have defined immediate subformulas and proper subformulas. In the first case, we have directly defined the immediate subformulas 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 formulas. In the second case, we have also mirrored the way the set of all formulas is defined, but in each case we have also included the proper subformulas of the smaller formulas ψ, χ in addition to these formulas themselves. This makes the definition *recursive*. In general, a definition of a function on an inductively defined set (in our case, formulas) 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 subformulas of the "earlier" formulas 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. $\varphi \equiv \neg\psi$: 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:

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

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 other than 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.

Definition 5.24 (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. $\varphi \equiv P(t_1, \ldots, t_n)$: $\varphi[t/x]$ is $P(t_1[t/x], \ldots, t_n[t/x])$.

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

3. $\varphi \equiv \neg\psi$: $\varphi[t/x]$ is $\neg\psi[t/x]$.

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

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

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

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

8. $\varphi \equiv \exists y\, \psi$: $\varphi[t/x]$ 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 5.25 (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 5.26. 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 $\mathsf{'}^{\mathfrak{M}} \colon |\mathfrak{M}| \to |\mathfrak{M}|$, 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. $\mathsf{'}^{\mathfrak{N}}(n) = n + 1$ 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. $<^{\mathfrak{N}} = \{\langle n, m\rangle : n \in \mathbb{N}, m \in \mathbb{N}, n < m\}$

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

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 $\exists x\,(\varphi(x) \vee \neg\varphi(x))$ 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 5.28 (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 5.29 (Covered structure)**.** A structure is *covered* if every element of the domain is the value of some closed term.

Example 5.30. Let $\mathcal{L}$ be the language with constant symbols zero, one, two, $\ldots$, 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 $\mathit{zero}^{\mathfrak{M}} = 0$, $\mathit{one}^{\mathfrak{M}} = 1$, $\mathit{two}^{\mathfrak{M}} = 2$, 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 formulas, 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 formulas are satisfied, and we define when a complex formula is satisfied depending on the main connective or quantifier and whether or not the immediate subformulas 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 5.31 (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 5.32 (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 5.33 (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 5.34 (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. $\varphi \equiv \bot$: not $\mathfrak{M}, s \models \varphi$.
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. $\varphi \equiv \neg\psi$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \not\models \psi$.
5. $\varphi \equiv (\psi \wedge \chi)$: $\mathfrak{M}, s \models \varphi$ 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. $\varphi \equiv (\psi \to \chi)$: $\mathfrak{M}, s \models \varphi$ 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 denumerable, so if $|\mathfrak{M}|$ is not enumerable there aren't even enough constant symbols to name every object.

A variable assignment s provides a value for *every* variable in the language. This is of course not necessary: whether or not a formula φ is satisfied in a structure with respect to s only depends on the assignments s makes to the free variables that actually occur in φ. This is the content of the next theorem. We require variable assignments to assign values to all variables simply because it makes things a lot easier.

Proposition 5.35. *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 $t = x_i$, then $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t) = s_1(x_i) = s_2(x_i)$ (by the hypothesis of the proposition) $= \mathrm{Val}^{\mathfrak{M}}_{s_2}(t)$. For the inductive step, assume that $t = f(t_1, \ldots, t_k)$ for some terms $t_1, \ldots, t_k$, and that the claim holds for $t_1, \ldots, t_k$. Then $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t) =$

$$\mathrm{Val}^{\mathfrak{M}}_{s_1}(f(t_1, \ldots, t_k)) = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_{s_1}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_k))$$

For $i = 1, \ldots, t_k$, its variables are among $x_1, \ldots, x_n$. So by induction hypothesis, $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t_i) = \mathrm{Val}^{\mathfrak{M}}_{s_k}(t_i)$. So,

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

□

Proposition 5.36. *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 $\mathfrak{M}, s_1 \not\models \varphi$ and $\mathfrak{M}, s_2 \not\models \varphi$.
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 5.35. 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. $\varphi \equiv t_1 = t_2$: if $\mathfrak{M}, s_1 \models \varphi$, $\mathrm{Val}^{\mathfrak{M}}_{s_2}(t_1) = \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_1)$ (by Proposition 5.35) $= \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_2)$ (since $\mathfrak{M}, s_1 \models t_1 = t_2$) $= \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_2)$ (by Proposition 5.35), 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 formulas 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.

1. $\varphi \equiv \neg\psi$: 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. $\varphi \equiv \psi \wedge \chi$: 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. $\varphi \equiv \psi \to \chi$: exercise.

5. $\varphi \equiv \exists x\, \psi$: if $\mathfrak{M}, s_1 \models \varphi$, there is an x-variant $\bar{s_1}$ of s_1 so that $\mathfrak{M}, \bar{s_1} \models \psi$. Let $\bar{s_2}$ denote the x-variant of s_2 that assigns the same thing to x as does $\bar{s_1}$. The free variables of ψ are among $x_1, \ldots, x_n$, and x. $\bar{s_1}(x_i) = \bar{s_2}(x_i)$, since $\bar{s_1}$ and $\bar{s_2}$ are x-variants of s_1 and s_2, respectively, and by hypothesis $s_1(x_i) = s_2(x_i)$. $\bar{s_1}(x) = \bar{s_2}(x)$ by the way we have defined $\bar{s_2}$. Then the induction hypothesis applies to ψ and $\bar{s_1}$, $\bar{s_2}$, so $\mathfrak{M}, \bar{s_2} \models \psi$. Hence, there is an x-variant of s_2 that satisfies ψ, 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 $s(x_i) = s'(x_i)$ for $i = 1, \ldots, n$. □

Definition 5.37. Definition 5.34 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 say that φ *is true in* $\mathfrak{M}$.

Proposition 5.38. *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. □

5.12 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 φ itself.

Proposition 5.39 (Extensionality)**.** *Let φ be a sentence, and $\mathfrak{M}$ and $\mathfrak{M}'$ be structures. If $c^{\mathfrak{M}} = c^{\mathfrak{M}'}$, $R^{\mathfrak{M}} = R^{\mathfrak{M}'}$, and $f^{\mathfrak{M}} = f^{\mathfrak{M}'}$ for every constant symbol c, relation symbol R, and function symbol f occurring in φ, then $\mathfrak{M} \models \varphi$ iff $\mathfrak{M}' \models \varphi$.*

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

Proposition 5.40. *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 $s'(x) = \mathrm{Val}^{\mathfrak{M}}_s(t')$. Then $\mathrm{Val}^{\mathfrak{M}}_s(t[t'/x]) = \mathrm{Val}^{\mathfrak{M}}_{s'}(t)$.*

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 5.41. *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. □

5.13 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 5.42 (Validity)**.** A sentence φ is *valid*, $\vDash \varphi$, iff $\mathfrak{M} \models \varphi$ for every structure $\mathfrak{M}$.

Definition 5.43 (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 5.44 (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 5.45. *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 5.46. *$\Gamma \vDash \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ 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 $\mathfrak{M} \models \Gamma \cup \{\neg\varphi\}$. Since $\mathfrak{M} \models \Gamma$ and $\Gamma \vDash \varphi$, $\mathfrak{M} \models \varphi$. Also, since $\mathfrak{M} \models \Gamma \cup \{\neg\varphi\}$, $\mathfrak{M} \models \neg\varphi$, 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 $\Gamma \cup \{\neg\varphi\}$ 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 5.47. *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 5.48 (Semantic Deduction Theorem)**.** $\Gamma \cup \{\varphi\} \vDash \psi$ *iff* $\Gamma \vDash \varphi \to \psi$.

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, $\mathfrak{M} \models \varphi \to \psi$, so $\Gamma \vDash \varphi \to \psi$.

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

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 $\{\varphi : \Gamma \vDash \varphi\}$.

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 $(\Gamma \setminus \{\varphi\}) \cup \{\neg\varphi\}$ 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 $\forall x \forall y\, ((x \leq y \wedge y \leq x) \rightarrow x = y)$ is true in $\mathfrak{M}$ iff $\leq^{\mathfrak{M}}$ is anti-symmetric. The sentence $\forall x \forall y \forall z\, ((x \leq y \wedge y \leq z) \rightarrow x \leq z)$ is true in $\mathfrak{M}$ iff $\leq^{\mathfrak{M}}$ is transitive. Thus, the models of

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

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

$$\begin{aligned} & \forall x \neg x < x, \\ & \forall x \forall y\, ((x < y \vee y < x) \vee x = y), \\ & \forall x \forall y \forall z\, ((x < y \wedge y < z) \rightarrow x < z) \end{aligned}$$

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 $\mathcal{L}_A$.

$$\begin{aligned}
&\neg\exists x\, x' = 0\\
&\forall x\,\forall y\,(x' = y' \to x = y)\\
&\forall x\,\forall y\,(x < y \leftrightarrow \exists z\,(x + z' = y))\\
&\forall x\,(x + 0) = x\\
&\forall x\,\forall y\,(x + y') = (x + y)'\\
&\forall x\,(x \times 0) = 0\\
&\forall x\,\forall y\,(x \times y') = ((x \times y) + x)
\end{aligned}$$

plus all sentences of the form

$$(\varphi(0) \wedge \forall x\,(\varphi(x) \to \varphi(x'))) \to \forall x\,\varphi(x)$$

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:

$$\begin{aligned}
&\exists x\,\neg\exists y\, y \in x\\
&\forall x\,\forall y\,(\forall z(z \in x \leftrightarrow z \in y) \to x = y)\\
&\forall x\,\forall y\,\exists z\,\forall u\,(u \in z \leftrightarrow (u = x \vee u = y))\\
&\forall x\,\exists y\,\forall z\,(z \in y \leftrightarrow \exists u\,(z \in u \wedge u \in x))
\end{aligned}$$

plus all sentences of the form

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

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 $\{x : \varphi(x)\}$ 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 $\neg y \in y$, you get the sentence

$$\exists x \, \forall y \, (y \in x \leftrightarrow \neg y \in y)$$

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.

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

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

$$\forall x \, \forall y \, \exists z \, \forall u \, (P(z, u) \leftrightarrow (P(x, u) \wedge P(y, u)))$$

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 formulas 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_0^2 is in $\mathcal{L}$, then $\mathfrak{M}$ assigns to it a relation $R = A_0^{2\mathfrak{M}}$. Then the formula $A_0^2(v_1, v_2)$ *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

$$Rab \quad \text{iff} \quad \mathfrak{M}, s \models A_0^2(v_1, v_2).$$

Note that we have to involve variable assignments here: we can't just say "Rab iff $\mathfrak{M} \models A_0^2(a, b)$" because a and b are not symbols of our language: they are elements of $|\mathfrak{M}|$.

Since we don't just have atomic formulas, but can combine them using the logical connectives and the quantifiers, more complex formulas 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 formulas $\exists v_3\,(v_3 \neq 0 \wedge v_2 = (v_1 + v_3))$ 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

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

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 $\forall x\, \forall y\, ((x \subseteq y \wedge y \subseteq x) \to x = y)$, or, if we replace $\subseteq$ by the above definition, by

$$\forall x\, \forall y\, ((\forall z\, (z \in x \to z \in y) \wedge \forall z\, (z \in y \to z \in x)) \to x = y).$$

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 $\neg \exists y\, y \in x$. Then "$\emptyset$ exists" becomes the sentence $\exists x\, \neg \exists y\, y \in x$. 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

$$\exists x\, (\neg \exists y\, y \in x \wedge \forall z\, x \subseteq z)$$

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

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

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 formulas 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., $\{\{x\}, \{x, y\}\} = z$:

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

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 $X \times Y = Z$:

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

A function $f\colon X \to Y$ can be thought of as the relation $f(x) = y$, i.e., as the set of pairs $\{\langle x, y\rangle : f(x) = y\}$. 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:

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

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

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

$$f\colon X \to Y \wedge \forall x\,\forall x'\,((x \in X \wedge x' \in X \wedge \exists y\,(\mathrm{maps}(f, x, y) \wedge \mathrm{maps}(f, x', y))) \to x = x')$$

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:

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

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

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

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 $\varphi(x) \equiv \neg x \in x$, then the comprehension principle states

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

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:

$$\forall z\, \exists y\, \forall x\, (x \in y \leftrightarrow (x \in z \wedge \varphi(x)).$$

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*

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

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

Proposition 6.12. *The sentence*

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

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*

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

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 non-enumerable structure cannot even be expressed with an infinite set of sentences. These facts follow from the compactness and Löwenheim-Skolem theorems.

Part III

Proofs and Completeness

Chapter 7

The Sequent Calculus

7.1 Rules and Derivations

Let $\mathcal{L}$ be a first-order language with the usual constants, variables, logical symbols, and auxiliary symbols (parentheses and the comma).

Definition 7.1 (sequent)**.** A *sequent* is an expression of the form

$$\Gamma \Rightarrow \Delta$$

where Γ and Δ are finite (possibly empty) sets of sentences of the language $\mathcal{L}$. The formulas in Γ are the *antecedent formulas*, while the formulae in Δ are the *succedent formulas*.

The intuitive idea behind a sequent is: if all of the antecedent formulas hold, then at least one of the succedent formulas holds. That is, if $\Gamma = \{\Gamma_1, \ldots, \Gamma_m\}$ and $\Delta = \{\Delta_1, \ldots, \Delta_n\}$, then $\Gamma \Rightarrow \Delta$ holds iff

$$(\Gamma_1 \wedge \cdots \wedge \Gamma_m) \rightarrow (\Delta_1 \vee \cdots \vee \Delta_n)$$

holds.

When $m = 0$, $\Rightarrow \Delta$ holds iff $\Delta_1 \vee \cdots \vee \Delta_n$ holds. When $n = 0$, $\Gamma \Rightarrow$ holds iff $\Gamma_1 \wedge \cdots \wedge \Gamma_m$ does not.

An empty succedent is sometimes filled with the $\bot$ symbol. he empty sequent $\Rightarrow$ canonically represents a contradiction.

We write Γ, φ (or φ, Γ) for $\Gamma \cup \{\varphi\}$, and Γ, Δ for $\Gamma \cup \Delta$.

Definition 7.2 (Inference)**.** An *inference* is an expression of the form

$$\frac{S_1}{S} \quad \text{or} \quad \frac{S_1 \qquad S_2}{S}$$

where S, S_1, and S_2 are sequents. S_1 and S_2 are called the *upper sequents* and S the *lower sequent* of the inference.

In sequent calculus derivations, a correct inference yields a valid sequent, provided the upper sequents are valid.

For the following, let $\Gamma, \Delta, \Pi, \Lambda$ represent finite sets of sentences.

The rules for **LK** are divided into two main types: *structural* rules and *logical* rules. The logical rules are further divided into *propositional* rules (quantifier-free) and *quantifier* rules.

Structural rules: Weakening:

$$\frac{\Gamma \Rightarrow \Delta}{\varphi, \Gamma \Rightarrow \Delta}\,\mathrm{WL} \quad \text{and} \quad \frac{\Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \varphi}\,\mathrm{WR}$$

where φ is called the *weakening formula*.

A series of weakening inferences will often be indicated by double inference lines.

Cut:

$$\frac{\Gamma \Rightarrow \Delta, \varphi \qquad \varphi, \Pi \Rightarrow \Lambda}{\Gamma, \Pi \Rightarrow \Delta, \Lambda}$$

Logical rules: The rules are named by the main operator of the *principal formula* of the inference (the formula containing φ and/or ψ in the lower sequent). The designations "left" and "right" indicate whether the logical symbol has been introduced in an antecedent formula or a succedent formula (to the left or to the right of the sequent symbol).

Propositional Rules:

$$\frac{\Gamma \Rightarrow \Delta, \varphi}{\neg\varphi, \Gamma \Rightarrow \Delta}\,\neg\mathrm{L} \quad \frac{\varphi, \Gamma \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg\varphi}\,\neg\mathrm{R}$$

$$\frac{\varphi, \Gamma \Rightarrow \Delta}{\varphi \wedge \psi, \Gamma \Rightarrow \Delta}\,\wedge\mathrm{L} \quad \frac{\psi, \Gamma \Rightarrow \Delta}{\varphi \wedge \psi, \Gamma \Rightarrow \Delta}\,\wedge\mathrm{L} \quad \frac{\Gamma \Rightarrow \Delta, \varphi \qquad \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \wedge \psi}\,\wedge\mathrm{R}$$

$$\frac{\varphi, \Gamma \Rightarrow \Delta \qquad \psi, \Gamma \Rightarrow \Delta}{\varphi \vee \psi, \Gamma \Rightarrow \Delta}\,\vee\mathrm{L} \quad \frac{\Gamma \Rightarrow \Delta, \varphi}{\Gamma \Rightarrow \Delta, \varphi \vee \psi}\,\vee\mathrm{R} \quad \frac{\Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \vee \psi}\,\vee\mathrm{R}$$

$$\frac{\Gamma \Rightarrow \Delta, \varphi \qquad \psi, \Pi \Rightarrow \Lambda}{\varphi \to \psi, \Gamma, \Pi \Rightarrow \Delta, \Lambda}\,\to\mathrm{L} \quad \frac{\varphi, \Gamma \Rightarrow \Delta, \psi}{\Gamma \Rightarrow \Delta, \varphi \to \psi}\,\to\mathrm{R}$$

Quantifier Rules:

$$\frac{\varphi(t), \Gamma \Rightarrow \Delta}{\forall x\, \varphi(x), \Gamma \Rightarrow \Delta}\,\forall\mathrm{L} \quad \frac{\Gamma \Rightarrow \Delta, \varphi(a)}{\Gamma \Rightarrow \Delta, \forall x\, \varphi(x)}\,\forall\mathrm{R}$$

where t is a ground term (i.e., one without variables), and a is a constant which does not occur anywhere in the lower sequent of the $\forall$R rule. We call a the *eigenvariable* of the $\forall$R inference.

$$\frac{\varphi(a), \Gamma \Rightarrow \Delta}{\exists x\, \varphi(x), \Gamma \Rightarrow \Delta}\,\exists\text{L} \qquad \frac{\Gamma \Rightarrow \Delta, \varphi(t)}{\Gamma \Rightarrow \Delta, \exists x\, \varphi(x)}\,\exists\text{R}$$

where t is a ground term, and a is a constant which does not occur in the lower sequent of the $\exists$L rule. We call a the *eigenvariable* of the $\exists$L inference.

The condition that an eigenvariable not occur in the upper sequent of the $\forall$R or $\exists$L 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 $\exists$R and $\forall$L there are no restrictions, and the term t can be anything, so we do not have to worry about any conditions. However, because the t may appear elsewhere in the sequent, the values of t for which the sequent is satisfied are constrained. On the other hand, in the $\exists$L and $\forall$ right rules, the eigenvariable condition requires that a does not occur anywhere else in the sequent. Thus, if the upper sequent is valid, the truth values of the formulas other than $\varphi(a)$ are independent of a.

Definition 7.3 (Initial Sequent)**.** An *initial sequent* is a sequent of one of the following forms:

1. $\varphi \Rightarrow \varphi$
2. $\bot \Rightarrow$

for any sentence φ in the language.

Definition 7.4 (LK derivation)**.** An **LK**-*derivation* of a sequent S is a tree of sequents satisfying the following conditions:

1. The topmost sequents of the tree are initial sequents.
2. Every sequent in the tree (except S) is an upper sequent of an inference whose lower sequent stands directly below that sequent in the tree.

We then say that S is the *end-sequent* of the derivation and that S is *derivable in* **LK** (or **LK**-derivable).

Definition 7.5 (LK theorem)**.** A sentence φ is a *theorem* of **LK** if the sequent $\Rightarrow \varphi$ is **LK**-derivable.

7.2 Examples of Derivations

Example 7.6. Give an **LK**-derivation for the sequent $\varphi \wedge \psi \Rightarrow \varphi$.

We begin by writing the desired end-sequent at the bottom of the derivation.

$$\frac{}{\varphi \wedge \psi \Rightarrow \varphi}$$

Next, we need to figure out what kind of inference could have a lower sequent of this form. This could be a structural rule, but it is a good idea to start by looking for a logical rule. The only logical connective occurring in a formula in the lower sequent is $\wedge$, so we're looking for an $\wedge$ rule, and since the $\wedge$ symbol occurs in the antecedent formulas, we're looking at the $\wedge$L rule.

$$\dfrac{}{\varphi \wedge \psi \Rightarrow \varphi}\,\wedge\mathrm{L}$$

There are two options for what could have been the upper sequent of the $\wedge$L inference: we could have an upper sequent of $\varphi \Rightarrow \varphi$, or of $\psi \Rightarrow \varphi$. Clearly, $\varphi \Rightarrow \varphi$ is an initial sequent (which is a good thing), while $\psi \Rightarrow \varphi$ is not derivable in general. We fill in the upper sequent:

$$\dfrac{\varphi \Rightarrow \varphi}{\varphi \wedge \psi \Rightarrow \varphi}\,\wedge\mathrm{L}$$

We now have a correct **LK**-derivation of the sequent $\varphi \wedge \psi \Rightarrow \varphi$.

Example 7.7. Give an **LK**-derivation for the sequent $\neg\varphi \vee \psi \Rightarrow \varphi \to \psi$.

Begin by writing the desired end-sequent at the bottom of the derivation.

$$\dfrac{}{\neg\varphi \vee \psi \Rightarrow \varphi \to \psi}$$

To find a logical rule that could give us this end-sequent, we look at the logical connectives in the end-sequent: $\neg$, $\vee$, and $\to$. We only care at the moment about $\vee$ and $\to$ because they are main operators of sentences in the end-sequent, while $\neg$ is inside the scope of another connective, so we will take care of it later. Our options for logical rules for the final inference are therefore the $\vee$L rule and the $\to$ R rule. We could pick either rule, really, but let's pick the $\to$ R rule (if for no reason other than it allows us to put off splitting into two branches). According to the form of $\to$ R inferences which can yield the lower sequent, this must look like:

$$\dfrac{\dfrac{}{\varphi, \neg\varphi \vee \psi \Rightarrow \psi}}{\neg\varphi \vee \psi \Rightarrow \varphi \to \psi}\,\to\mathrm{R}$$

Now we can apply the $\vee$L rule. According to the schema, this must split into two upper sequents as follows:

$$\dfrac{\dfrac{\dfrac{}{\varphi, \neg\varphi \Rightarrow \psi} \qquad \dfrac{}{\varphi, \psi \Rightarrow \psi}}{\varphi, \neg\varphi \vee \psi \Rightarrow \psi}\,\vee\mathrm{L}}{\neg\varphi \vee \psi \Rightarrow \varphi \to \psi}\,\to\mathrm{R}$$

Remember that we are trying to wind our way up to initial sequents; we seem to be pretty close! The right branch is just one weakening away from an initial sequent and then it is done:

$$\dfrac{\dfrac{}{\varphi, \neg\varphi \Rightarrow \psi} \quad \dfrac{\dfrac{}{\psi \Rightarrow \psi}}{\varphi, \psi \Rightarrow \psi}\,\mathrm{WL}}{\dfrac{\varphi, \neg\varphi \vee \psi \Rightarrow \psi}{\neg\varphi \vee \psi \Rightarrow \varphi \to \psi}\,\to \mathrm{R}}\,\vee\mathrm{L}$$

Now looking at the left branch, the only logical connective in any sentence is the $\neg$ symbol in the antecedent sentences, so we're looking at an instance of the $\neg$L rule.

$$\dfrac{\dfrac{\dfrac{}{\varphi \Rightarrow \psi, \varphi}}{\varphi, \neg\varphi \Rightarrow \psi}\,\neg\mathrm{L} \quad \dfrac{\dfrac{}{\psi \Rightarrow \psi}}{\varphi, \psi \Rightarrow \psi}\,\mathrm{WL}}{\dfrac{\varphi, \neg\varphi \vee \psi \Rightarrow \psi}{\neg\varphi \vee \psi \Rightarrow \varphi \to \psi}\,\to \mathrm{R}}\,\vee\mathrm{L}$$

Similarly to how we finished off the right branch, we are just one weakening away from finishing off this left branch as well.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi \Rightarrow \varphi}}{\varphi \Rightarrow \psi, \varphi}\,\mathrm{WR}}{\varphi, \neg\varphi \Rightarrow \psi}\,\neg\mathrm{L} \quad \dfrac{\dfrac{}{\psi \Rightarrow \psi}}{\varphi, \psi \Rightarrow \psi}\,\mathrm{WL}}{\dfrac{\varphi, \neg\varphi \vee \psi \Rightarrow \psi}{\neg\varphi \vee \psi \Rightarrow \varphi \to \psi}\,\to \mathrm{R}}\,\vee\mathrm{L}$$

Example 7.8. Give an **LK**-derivation of the sequent $\neg\varphi \vee \neg\psi \Rightarrow \neg(\varphi \wedge \psi)$

Using the techniques from above, we start by writing the desired end-sequent at the bottom.

$$\dfrac{}{\neg\varphi \vee \neg\psi \Rightarrow \neg(\varphi \wedge \psi)}$$

The available main connectives of sentences in the end-sequent are the $\vee$ symbol and the $\neg$ symbol. It would work to apply either the $\vee$L or the $\neg$R rule here, but we start with the $\neg$R rule because it avoids splitting up into two branches for a moment:

$$\dfrac{\dfrac{}{\varphi \wedge \psi, \neg\varphi \vee \neg\psi \Rightarrow}}{\neg\varphi \vee \neg\psi \Rightarrow \neg(\varphi \wedge \psi)}\,\neg\mathrm{R}$$

Now we have a choice of whether to look at the $\wedge$L or the $\vee$L rule. Let's see what happens when we apply the $\wedge$L rule: we have a choice to start with either the sequent $\varphi, \neg\varphi \vee \psi \Rightarrow$ or the sequent $\psi, \neg\varphi \vee \psi \Rightarrow$. Since the proof is symmetric with regards to φ and ψ, let's go with the former:

$$\dfrac{\dfrac{\dfrac{}{\varphi, \neg\varphi \vee \neg\psi \Rightarrow}}{\varphi \wedge \psi, \neg\varphi \vee \neg\psi \Rightarrow}\,\wedge\mathrm{L}}{\neg\varphi \vee \neg\psi \Rightarrow \neg(\varphi \wedge \psi)}\,\neg\mathrm{R}$$

Continuing to fill in the derivation, we see that we run into a problem:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi \Rightarrow \varphi}}{\varphi, \neg\varphi \Rightarrow}\neg\mathrm{L} \quad \dfrac{\dfrac{}{\varphi \Rightarrow \psi}?}{\varphi, \neg\psi \Rightarrow}\neg\mathrm{L}}{\varphi, \neg\varphi \vee \neg\psi \Rightarrow}\vee\mathrm{L}}{\varphi \wedge \psi, \neg\varphi \vee \neg\psi \Rightarrow}\wedge\mathrm{L}}{\neg\varphi \vee \neg\psi \Rightarrow \neg(\varphi \wedge \psi)}\neg\mathrm{R}$$

The top of the right branch cannot be reduced any further, and it cannot be brought by way of structural inferences to an initial sequent, so this is not the right path to take. So clearly, it was a mistake to apply the $\wedge$L rule above. Going back to what we had before and carrying out the $\vee$L rule instead, we get

$$\dfrac{\dfrac{\dfrac{}{\varphi \wedge \psi, \neg\varphi \Rightarrow} \quad \dfrac{}{\varphi \wedge \psi, \neg\psi \Rightarrow}}{\varphi \wedge \psi, \neg\varphi \vee \neg\psi \Rightarrow}\vee\mathrm{L}}{\neg\varphi \vee \neg\psi \Rightarrow \neg(\varphi \wedge \psi)}\neg\mathrm{R}$$

Completing each branch as we've done before, we get

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\varphi \Rightarrow \varphi}}{\varphi \wedge \psi \Rightarrow \varphi}\wedge\mathrm{L}}{\varphi \wedge \psi, \neg\varphi \Rightarrow}\neg\mathrm{L} \quad \dfrac{\dfrac{\dfrac{}{\psi \Rightarrow \psi}}{\varphi \wedge \psi \Rightarrow \psi}\wedge\mathrm{L}}{\varphi \wedge \psi, \neg\psi \Rightarrow}\neg\mathrm{L}}{\varphi \wedge \psi, \neg\varphi \vee \neg\psi \Rightarrow}\vee\mathrm{L}}{\neg\varphi \vee \neg\psi \Rightarrow \neg(\varphi \wedge \psi)}\neg\mathrm{R}$$

(We could have carried out the $\wedge$ rules lower than the $\neg$ rules in these steps and still obtained a correct derivation).

Example 7.9. Give an **LK**-derivation of the sequent $\exists x\, \neg\varphi(x) \Rightarrow \neg\forall x\, \varphi(x)$.

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). Also, it is a good idea to try and look ahead and try to guess what the initial sequent might look like. In our case, it will have to be something like $\varphi(a) \Rightarrow \varphi(a)$. That means that when we are "reversing" the quantifier rules, we will have to pick the same term—what we will call a—for both the $\forall$ and the $\exists$ rule. If we picked different terms for each rule, we would end up with something like $\varphi(a) \Rightarrow \varphi(b)$, which, of course, is not derivable.

Starting as usual, we write

$$\frac{}{\exists x\, \neg\varphi(x) \;\Rightarrow\; \neg\forall x\, \varphi(x)}$$

We could either carry out the $\exists$L rule or the $\neg$R rule. Since the $\exists$L rule is subject to the eigenvariable condition, it's a good idea to take care of it sooner rather than later, so we'll do that one first.

$$\dfrac{\dfrac{}{\neg\varphi(a) \;\Rightarrow\; \neg\forall x\, \varphi(x)}}{\exists x\, \neg\varphi(x) \;\Rightarrow\; \neg\forall x\, \varphi(x)}\,\exists\text{L}$$

Applying the $\neg$L and $\neg$R rules and then eliminating the double $\neg$ signs on both sides—the reader may do this as an exercise—we get

$$\dfrac{\begin{array}{c}\dfrac{}{\forall x\, \varphi(x) \;\Rightarrow\; \varphi(a)}\\ \vdots \\ \neg\varphi(a) \;\Rightarrow\; \neg\forall x\varphi(x)\end{array}}{\exists x\neg\varphi(x) \;\Rightarrow\; \neg\forall x\varphi(x)}\,\exists\text{L}$$

At this point, our only option is to carry out the $\forall$L rule. Since this rule is not subject to the eigenvariable restriction, we're in the clear. Remember, we want to try and obtain an initial sequent (of the form $\varphi(a) \Rightarrow \varphi(a)$), so we should choose a as our argument for φ when we apply the rule.

$$\dfrac{\begin{array}{c}\dfrac{\varphi(a) \;\Rightarrow\; \varphi(a)}{\forall x\, \varphi(x) \;\Rightarrow\; \varphi(a)}\,\forall\text{L}\\ \vdots \\ \neg\varphi(a) \;\Rightarrow\; \neg\forall x\, \varphi(x)\end{array}}{\exists x\, \neg\varphi(x) \;\Rightarrow\; \neg\forall x\, \varphi(x)}\,\exists\text{L}$$

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 $\exists$L, and the eigenvariable a does not occur in its lower sequent (the end-sequent), this is a correct derivation.

7.3 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 sequents. 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 in **LK** of the sequent $\Rightarrow \varphi$. We write $\vdash_{\mathbf{LK}} \varphi$ if φ is a theorem and $\nvdash_{\mathbf{LK}} \varphi$ if it is not.

Definition 7.11 (Derivability)**.** A sentence φ is *derivable from* a set of sentences Γ, $\Gamma \vdash_{\mathbf{LK}} \varphi$, if there is a finite subset $\Gamma_0 \subseteq \Gamma$ such that **LK** derives $\Gamma_0 \Rightarrow \varphi$. If φ is not derivable from Γ we write $\Gamma \nvdash_{\mathbf{LK}} \varphi$.

Definition 7.12 (Consistency)**.** A set of sentences Γ is *consistent* iff $\Gamma \nvdash_{\mathbf{LK}} \bot$. If Γ is not consistent, i.e., if $\Gamma \vdash_{\mathbf{LK}} \bot$, we say it is *inconsistent*.

Proposition 7.13. *$\Gamma \vdash_{\mathbf{LK}} \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is inconsistent.*

Proof. Exercise. □

Proposition 7.14. *Γ is inconsistent iff $\Gamma \vdash_{\mathbf{LK}} \varphi$ for every sentence φ.*

Proof. Exercise. □

Proposition 7.15. *$\Gamma \vdash_{\mathbf{LK}} \varphi$ iff for some finite $\Gamma_0 \subseteq \Gamma$, $\Gamma_0 \vdash_{\mathbf{LK}} \varphi$.*

Proof. Follows immediately from the definion of $\vdash_{\mathbf{LK}}$. □

7.4 Properties of Derivability

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 7.16 (Monotony)**.** *If $\Gamma \subseteq \Delta$ and $\Gamma \vdash_{\mathbf{LK}} \varphi$, then $\Delta \vdash_{\mathbf{LK}} \varphi$.*

Proof. Any finite $\Gamma_0 \subseteq \Gamma$ is also a finite subset of Δ, so a derivation of $\Gamma_0 \Rightarrow \varphi$ also shows $\Delta \vdash_{\mathbf{LK}} \varphi$. □

Proposition 7.17. *If $\Gamma \vdash_{\mathbf{LK}} \varphi$ and $\Gamma \cup \{\varphi\} \vdash_{\mathbf{LK}} \bot$, then Γ is inconsistent.*

Proof. There are finite Γ_0 and $\Gamma_1 \subseteq \Gamma$ such that **LK** derives $\Gamma_0 \Rightarrow \varphi$ and $\Gamma_1, \varphi \Rightarrow \bot$. Let the **LK**-derivation of $\Gamma_0 \Rightarrow \varphi$ be Π_0 and the **LK**-derivation of $\Gamma_1, \varphi \Rightarrow \bot$ be Π_1. We can then derive

$$\dfrac{\begin{array}{cc}\vdots\,\Pi_0 & \vdots\,\Pi_1\\ \Gamma_0 \Rightarrow \varphi & \Gamma_1, \varphi \Rightarrow \bot\end{array}}{\Gamma_0, \Gamma_1 \Rightarrow \bot}\,\text{cut}$$

Since $\Gamma_0 \subseteq \Gamma$ and $\Gamma_1 \subseteq \Gamma$, $\Gamma_0 \cup \Gamma_1 \subseteq \Gamma$, hence $\Gamma \vdash_{\mathbf{LK}} \bot$. □

Proposition 7.18. *If $\Gamma \cup \{\varphi\} \vdash_{\mathbf{LK}} \bot$, then $\Gamma \vdash_{\mathbf{LK}} \neg\varphi$.*

Proof. Suppose that $\Gamma \cup \{\varphi\} \vdash_{\mathbf{LK}} \bot$. Then there is a finite set $\Gamma_0 \subseteq \Gamma$ such that **LK** derives $\Gamma_0, \varphi \Rightarrow \bot$. Let Π_0 be an **LK**-derivation of $\Gamma_0, \varphi \Rightarrow \bot$, and consider

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Pi_0\\ \vdots\\ \Gamma_0, \varphi \Rightarrow \bot\end{array}}{\Gamma_0 \Rightarrow \bot, \neg\varphi}\,\neg\mathrm{R} \qquad \bot \Rightarrow}{\Gamma_0 \Rightarrow \neg\varphi}\,\mathrm{cut}$$

□

Proposition 7.19. *If* $\Gamma \cup \{\varphi\} \vdash_{\mathbf{LK}} \bot$ *and* $\Gamma \cup \{\neg\varphi\} \vdash_{\mathbf{LK}} \bot$, *then* $\Gamma \vdash_{\mathbf{LK}} \bot$.

Proof. There are finite sets $\Gamma_0 \subseteq \Gamma$ and $\Gamma_1 \subseteq \Gamma$ and **LK**-derivations Π_0 and Π_1 of $\Gamma_0, \varphi \Rightarrow \bot$ and $\Gamma_1, \neg\varphi \Rightarrow \bot$, respectively. We can then derive

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Pi_0\\ \vdots\\ \Gamma_0, \varphi \Rightarrow \bot\end{array}}{\Gamma_0 \Rightarrow \bot, \neg\varphi}\,\neg\mathrm{R} \qquad \begin{array}{c}\vdots\\ \Pi_1\\ \vdots\\ \Gamma_1, \neg\varphi \Rightarrow \bot\end{array}}{\Gamma_0, \Gamma_1 \Rightarrow \bot}\,\mathrm{cut}$$

Since $\Gamma_0 \subseteq \Gamma$ and $\Gamma_1 \subseteq \Gamma$, $\Gamma_0 \cup \Gamma_1 \subseteq \Gamma$. Hence $\Gamma \vdash_{\mathbf{LK}} \bot$. □

Proposition 7.20. *If* $\Gamma \cup \{\varphi\} \vdash_{\mathbf{LK}} \bot$ *and* $\Gamma \cup \{\psi\} \vdash_{\mathbf{LK}} \bot$, *then* $\Gamma \cup \{\varphi \vee \psi\} \vdash_{\mathbf{LK}} \bot$.

Proof. There are finite sets $\Gamma_0, \Gamma_1 \subseteq \Gamma$ and **LK**-derivations Π_0 and Π_1 such that

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Pi_0\\ \vdots\\ \Gamma_0, \varphi \Rightarrow \bot\end{array}}{\Gamma_0, \Gamma_1, \varphi \Rightarrow \bot} \qquad \dfrac{\begin{array}{c}\vdots\\ \Pi_1\\ \vdots\\ \Gamma_1, \psi \Rightarrow \bot\end{array}}{\Gamma_0, \Gamma_1, \psi \Rightarrow \bot}}{\Gamma_0, \Gamma_1, \varphi \vee \psi \Rightarrow \bot}\,\vee\mathrm{L}$$

(Recall that double inference lines indicate several weakening inferences.)

Since $\Gamma_0, \Gamma_1 \subseteq \Gamma$ and $\Gamma \cup \{\varphi \vee \psi\} \vdash_{\mathbf{LK}} \bot$. □

Proposition 7.21. *If* $\Gamma \vdash_{\mathbf{LK}} \varphi$ *or* $\Gamma \vdash_{\mathbf{LK}} \psi$, *then* $\Gamma \vdash_{\mathbf{LK}} \varphi \vee \psi$.

Proof. There is an **LK**-derivation Π_0 and a finite set $\Gamma_0 \subseteq \Gamma$ such that we can derive

$$\dfrac{\begin{array}{c}\vdots\\ \Pi_0\\ \vdots\\ \Gamma_0 \Rightarrow \varphi\end{array}}{\Gamma_0 \Rightarrow \varphi \vee \psi}\,\vee\mathrm{R}$$

Therefore $\Gamma \vdash_{\mathbf{LK}} \varphi \vee \psi$. The proof for when $\Gamma \vdash_{\mathbf{LK}} \psi$ is similar. □

Proposition 7.22. *If $\Gamma \vdash_{\mathbf{LK}} \varphi \wedge \psi$ then $\Gamma \vdash_{\mathbf{LK}} \varphi$ and $\Gamma \vdash_{\mathbf{LK}} \psi$.*

Proof. If $\Gamma \vdash_{\mathbf{LK}} \varphi \wedge \psi$, there is a finite set $\Gamma_0 \subseteq \Gamma$ and an **LK**-derivation Π_0 of $\Gamma_0 \Rightarrow \varphi \wedge \psi$. Consider

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Pi_0\\ \vdots\\ \Gamma_0 \Rightarrow \varphi \wedge \psi\end{array} \qquad \dfrac{\varphi \Rightarrow \varphi}{\varphi \wedge \psi \Rightarrow \varphi}\,\wedge\mathrm{L}}{\Gamma_0 \Rightarrow \varphi}\,\mathrm{cut}$$

Hence, $\Gamma \vdash_{\mathbf{LK}} \varphi$. A similar derivation starting with $\psi \Rightarrow \psi$ on the right side shows that $\Gamma \vdash_{\mathbf{LK}} \psi$. □

Proposition 7.23. *If $\Gamma \vdash_{\mathbf{LK}} \varphi$ and $\Gamma \vdash_{\mathbf{LK}} \psi$, then $\Gamma \vdash_{\mathbf{LK}} \varphi \wedge \psi$.*

Proof. If $\Gamma \vdash_{\mathbf{LK}} \varphi$ as well as $\Gamma \vdash_{\mathbf{LK}} \psi$, there are finite sets $\Gamma_0, \Gamma_1 \subseteq \Gamma$ and an **LK**-derivations Π_0 of $\Gamma_0 \Rightarrow \varphi$ and Π_1 of $\Gamma_1 \Rightarrow \varphi$. Consider

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Pi_0\\ \vdots\\ \Gamma_0 \Rightarrow \varphi\end{array}}{\Gamma_0, \Gamma_1 \Rightarrow \varphi} \qquad \dfrac{\begin{array}{c}\vdots\\ \Pi_1\\ \vdots\\ \Gamma_1 \Rightarrow \psi\end{array}}{\Gamma_0, \Gamma_1 \Rightarrow \psi}}{\Gamma_0, \Gamma_1 \Rightarrow \varphi \wedge \psi}\,\wedge\mathrm{R}$$

Since $\Gamma_0 \cup \Gamma_1 \subseteq \Gamma$, we have $\Gamma \vdash_{\mathbf{LK}} \varphi \wedge \psi$. □

Proposition 7.24. *If $\Gamma \vdash_{\mathbf{LK}} \varphi$ and $\Gamma \vdash_{\mathbf{LK}} \varphi \to \psi$, then $\Gamma \vdash_{\mathbf{LK}} \psi$.*

Proof. Exercise. □

Proposition 7.25. *If $\Gamma \vdash_{\mathbf{LK}} \neg\varphi$ or $\Gamma \vdash_{\mathbf{LK}} \psi$, then $\Gamma \vdash_{\mathbf{LK}} \varphi \to \psi$.*

Proof. Exercise. □

Theorem 7.26. *If c is a constant not occurring in Γ or $\varphi(x)$ and $\Gamma \vdash_{\mathbf{LK}} \varphi(c)$, then $\Gamma \vdash_{\mathbf{LK}} \forall x\, \varphi(x)$.*

Proof. Let Π_0 be an **LK**-derivation of $\Gamma_0 \Rightarrow \varphi(c)$ for some finite $\Gamma_0 \subseteq \Gamma$. By adding a $\forall$R inference, we obtain a proof of $\Gamma \Rightarrow \forall x\, \varphi(x)$, since c does not occur in Γ or $\varphi(x)$ and thus the eigenvariable condition is satisfied. □

Theorem 7.27. *1. If* $\Gamma \vdash_{\mathbf{LK}} \varphi(t)$ *then* $\Gamma \vdash \exists x\, \varphi(x)$.

2. If $\Gamma \vdash_{\mathbf{LK}} \forall x\, \varphi(x)$ *then* $\Gamma \vdash \varphi(t)$.

Proof. 1. Suppose $\Gamma \vdash_{\mathbf{LK}} \varphi(t)$. Then for some finite $\Gamma_0 \subseteq \Gamma$, **LK** derives $\Gamma_0 \Rightarrow \varphi(t)$. Add an $\exists$R inference to get a derivation of $\Gamma_0 \Rightarrow \exists x\, \varphi(x)$.

2. Suppose $\Gamma \vdash_{\mathbf{LK}} \forall x\, \varphi(x)$. Then there is a finite $\Gamma_0 \subseteq \Gamma$ and an **LK**-derivation Π of $\Gamma_0 \Rightarrow \forall x\, \varphi(x)$. Then

$$\dfrac{\begin{array}{c}\vdots\\ \Pi\\ \vdots\end{array}}{\Gamma_0 \Rightarrow \forall x\, \varphi(x)} \qquad \dfrac{\dfrac{}{\varphi(t) \Rightarrow \varphi(t)}}{\forall x\, \varphi(x) \Rightarrow \varphi(t)}\,\forall\mathrm{L}}{\Gamma_0 \Rightarrow \varphi(t)}\,\mathrm{cut}$$

shows that $\Gamma \vdash_{\mathbf{LK}} \varphi(t)$.

□

7.5 Soundness

A derivation system, such as the sequent calculus, is *sound* if it cannot derive things that do not actually hold. 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.

Because all these proof-theoretic properties are defined via derivability in the sequent calculus of certain sequents, proving (1)–(3) above requires proving something about the semantic properties of derivable sequents. We will first define what it means for a sequent to be *valid*, and then show that every derivable sequent is valid. (1)–(3) then follow as corollaries from this result.

Definition 7.28. A structure $\mathfrak{M}$ *satisfies* a sequent $\Gamma \Rightarrow \Delta$ iff either $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma$ or $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta$.

A sequent is *valid* iff every structure $\mathfrak{M}$ satisfies it.

Theorem 7.29 (Soundness)**.** *If* **LK** *derives* $\Gamma \Rightarrow \Delta$, *then* $\Gamma \Rightarrow \Delta$ *is valid.*

Proof. Let Π be a derivation of $\Gamma \Rightarrow \Delta$. We proceed by induction on the number of inferences in Π.

If the number of inferences is 0, then Π consists only of an initial sequent. Every initial sequent $\varphi \Rightarrow \varphi$ is obviously valid, since for every $\mathfrak{M}$, either $\mathfrak{M} \not\models \varphi$ or $\mathfrak{M} \models \varphi$.

If the number of inferences is greater than 0, we distinguish cases according to the type of the lowermost inference. By induction hypothesis, we can assume that the premises of that inference are valid, since the height of the proof of any premise is smaller than n.

First, we consider the possible inferences with only one premise $\Gamma' \Rightarrow \Delta'$.

1. The last inference is a weakening. Then $\Gamma' \subseteq \Gamma$ and $\Delta = \Delta'$ if it's a weakening on the left, or $\Gamma = \Gamma'$ and $\Delta' \subseteq \Delta$ if it's a weaking on the right. In either case, $\Delta' \subseteq \Delta$ and $\Gamma' \subseteq \Gamma$. If $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma'$, then, since $\Gamma' \subseteq \Gamma$, $\alpha \in \Gamma$ as well, and so $\mathfrak{M} \not\models \alpha$ for the same $\alpha \in \Gamma$. Similarly, if $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$, as $\alpha \in \Delta$, $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta$. Since $\Gamma' \Rightarrow \Delta'$ is valid, one of these cases obtains for every $\mathfrak{M}$. Consequently, $\Gamma \Rightarrow \Delta$ is valid.

2. The last inference is $\neg$L: Then for some $\varphi \in \Delta'$, $\neg\varphi \in \Gamma$. Also, $\Gamma' \subseteq \Gamma$, and $\Delta' \setminus \{\varphi\} \subseteq \Delta$.

   If $\mathfrak{M} \models \varphi$, then $\mathfrak{M} \not\models \neg\varphi$, and since $\neg\varphi \in \Gamma$, $\mathfrak{M}$ satisfies $\Gamma \Rightarrow \Delta$. Since $\Gamma' \Rightarrow \Delta'$ is valid, if $\mathfrak{M} \not\models \varphi$, then either $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma'$ or $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$ different from φ. Consequently, $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma$ (since $\Gamma' \subseteq \Gamma$) or $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$ different from φ (since $\Delta' \setminus \{\varphi\} \subseteq \Delta$).

3. The last inference is $\neg$R: Exercise.

4. The last inference is $\wedge$L: There are two variants: $\varphi \wedge \psi$ may be inferred on the left from φ or from ψ on the left side of the premise. In the first case, $\varphi \in \Gamma'$. Consider a structure $\mathfrak{M}$. Since $\Gamma' \Rightarrow \Delta'$ is valid, (a) $\mathfrak{M} \not\models \varphi$, (b) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma' \setminus \{\varphi\}$, or (c) $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$. In case (a), $\mathfrak{M} \not\models \varphi \wedge \psi$. In case (b), there is an $\alpha \in \Gamma \setminus \{\varphi \wedge \psi\}$ such that $\mathfrak{M} \not\models \alpha$, since $\Gamma' \setminus \{\varphi\} \subseteq \Gamma \setminus \{\varphi \wedge \psi\}$. In case (c), there is a $\alpha \in \Delta$ such that $\mathfrak{M} \models \alpha$, as $\Delta = \Delta'$. So in each case, $\mathfrak{M}$ satisfies $\varphi \wedge \psi, \Gamma \Rightarrow \Delta$. Since $\mathfrak{M}$ was arbitrary, $\Gamma \Rightarrow \Delta$ is valid. The case where $\varphi \wedge \psi$ is inferred from ψ is handled the same, changing φ to ψ.

5. The last inference is $\vee$R: There are two variants: $\varphi \vee \psi$ may be inferred on the right from φ or from ψ on the right side of the premise. In the first case, $\varphi \in \Delta'$. Consider a structure $\mathfrak{M}$. Since $\Gamma' \Rightarrow \Delta'$ is valid, (a) $\mathfrak{M} \models \varphi$, (b) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma'$, or (c) $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta' \setminus \{\varphi\}$. In case (a), $\mathfrak{M} \models \varphi \vee \psi$. In case (b), there is $\alpha \in \Gamma$ such that $\mathfrak{M} \not\models \alpha$, as $\Gamma = \Gamma'$. In case (c), there is an $\alpha \in \Delta$ such that $\mathfrak{M} \models \alpha$, since $\Delta' \setminus \{\varphi\} \subseteq \Delta$. So

in each case, $\mathfrak{M}$ satisfies $\varphi \wedge \psi, \Gamma \Rightarrow \Delta$. Since $\mathfrak{M}$ was arbitrary, $\Gamma \Rightarrow \Delta$ is valid. The case where $\varphi \vee \psi$ is inferred from ψ is handled the same, changing φ to ψ.

6. The last inference is $\rightarrow$ R: Then $\varphi \in \Gamma'$, $\psi \in \Delta'$, $\Gamma' \setminus \{\varphi\} \subseteq \Gamma$ and $\Delta' \setminus \{\psi\} \subseteq \Delta$. Since $\Gamma' \Rightarrow \Delta'$ is valid, for any structure $\mathfrak{M}$, (a) $\mathfrak{M} \not\models \varphi$, (b) $\mathfrak{M} \models \psi$, (c) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma' \setminus \{\varphi\}$, or $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta' \setminus \{\psi\}$. In cases (a) and (b), $\mathfrak{M} \models \varphi \rightarrow \psi$. In case (c), for some $\alpha \in \Gamma$, $\mathfrak{M} \not\models \alpha$. In case (d), for some $\alpha \in \Delta$, $\mathfrak{M} \models \alpha$. In each case, $\mathfrak{M}$ satisfies $\Gamma \Rightarrow \Delta$. Since $\mathfrak{M}$ was arbitrary, $\Gamma \Rightarrow \Delta$ is valid.

7. The last inference is $\forall$L: Then there is a formula $\varphi(x)$ and a ground term t such that $\varphi(t) \in \Gamma'$, $\forall x\, \varphi(x) \in \Gamma$, and $\Gamma' \setminus \{\varphi(t)\} \subseteq \Gamma$. Consider a structure $\mathfrak{M}$. Since $\Gamma' \Rightarrow \Delta'$ is valid, (a) $\mathfrak{M} \not\models \varphi(t)$, (b) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma' \setminus \{\varphi(t)\}$, or (c) $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$. In case (a), $\mathfrak{M} \not\models \forall x\, \varphi(x)$. In case (b), there is an $\alpha \in \Gamma \setminus \{\varphi(t)\}$ such that $\mathfrak{M} \not\models \alpha$. In case (c), there is a $\alpha \in \Delta$ such that $\mathfrak{M} \models \alpha$, as $\Delta = \Delta'$. So in each case, $\mathfrak{M}$ satisfies $\Gamma \Rightarrow \Delta$. Since $\mathfrak{M}$ was arbitrary, $\Gamma \Rightarrow \Delta$ is valid.

8. The last inference is $\exists$R: Exercise.

9. The last inference is $\forall$R: Then there is a formula $\varphi(x)$ and a constant symbol a such that $\varphi(a) \in \Delta'$, $\forall x\, \varphi(x) \in \Delta$, and $\Delta' \setminus \{\varphi(a)\} \subseteq \Delta$. Furthermore, $a \notin \Gamma \cup \Delta$. Consider a structure $\mathfrak{M}$. Since $\Gamma' \Rightarrow \Delta'$ is valid, (a) $\mathfrak{M} \models \varphi(a)$, (b) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma'$, or (c) $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta' \setminus \{\varphi(a)\}$.

   First, suppose (a) is the case but neither (b) nor (c), i.e., $\mathfrak{M} \models \alpha$ for all $\alpha \in \Gamma'$ and $\mathfrak{M} \not\models \alpha$ for all $\alpha \in \Delta' \setminus \{\varphi(a)\}$. In other words, assume $\mathfrak{M} \models \varphi(a)$ and that $\mathfrak{M}$ does not satisfy $\Gamma' \Rightarrow \Delta' \setminus \{\varphi(a)\}$. Since $a \notin \Gamma \cup \Delta$, also $a \notin \Gamma' \cup (\Delta' \setminus \{\varphi(a)\})$. Thus, if $\mathfrak{M}'$ is like $\mathfrak{M}$ except that $a^{\mathfrak{M}} \neq a^{\mathfrak{M}'}$, $\mathfrak{M}'$ also does not satisfy $\Gamma' \Rightarrow \Delta' \setminus \{\varphi(a)\}$ by extensionality. But since $\Gamma' \Rightarrow \Delta'$ is valid, we must have $\mathfrak{M}' \models \varphi(a)$.

   We now show that $\mathfrak{M} \models \forall x\, \varphi(x)$. To do this, we have to show that for every variable assignment s, $\mathfrak{M}, s \models \forall x\, \varphi(x)$. This in turn means that for every x-variant s' of s, we must have $\mathfrak{M}, s' \models \varphi(x)$. So consider any variable assignment s and let s' be an x-variant of s. Since Γ' and Δ' consist entirely of sentences, $\mathfrak{M}, s \models \alpha$ iff $\mathfrak{M}, s' \models \alpha$ iff $\mathfrak{M} \models \alpha$ for all $\alpha \in \Gamma' \cup \Delta'$. Let $\mathfrak{M}'$ be like $\mathfrak{M}$ except that $a^{\mathfrak{M}'} = s'(x)$. Then $\mathfrak{M}, s' \models \varphi(x)$ iff $\mathfrak{M}' \models \varphi(a)$ (as $\varphi(x)$ does not contain a). Since we've already established that $\mathfrak{M}' \models \varphi(a)$ for all $\mathfrak{M}'$ which differ from $\mathfrak{M}$ at most in what they assign to a, this means that $\mathfrak{M}, s' \models \varphi(x)$. Thus weve shown that $\mathfrak{M}, s \models \forall x\, \varphi(x)$. Since s is an arbitrary variable assignment and $\forall x\, \varphi(x)$ is a sentence, then $\mathfrak{M} \models \forall x\, \varphi(x)$.

If (b) is the case, there is a $\alpha \in \Gamma$ such that $\mathfrak{M} \not\models \alpha$, as $\Gamma = \Gamma'$. If (c) is the case, there is an $\alpha \in \Delta' \setminus \{\varphi(a)\}$ such that $\mathfrak{M} \models \alpha$. So in each case, $\mathfrak{M}$ satisfies $\Gamma \Rightarrow \Delta$. Since $\mathfrak{M}$ was arbitrary, $\Gamma \Rightarrow \Delta$ is valid.

10. The last inference is $\exists$L: Exercise.

Now let's consider the possible inferences with two premises.

1. The last inference is a cut: Suppose the premises are $\Gamma' \Rightarrow \Delta'$ and $\Pi' \Rightarrow \Lambda'$ and the cut formula φ is in both Δ' and Π'. Since each is valid, every structure $\mathfrak{M}$ satisfies both premises. We distinguish two cases: (a) $\mathfrak{M} \not\models \varphi$ and (b) $\mathfrak{M} \models \varphi$. In case (a), in order for $\mathfrak{M}$ to satisfy the left premise, it must satisfy $\Gamma' \Rightarrow \Delta' \setminus \{\varphi\}$. But $\Gamma' \subseteq \Gamma$ and $\Delta' \setminus \{\varphi\} \subseteq \Delta$, so $\mathfrak{M}$ also satisfies $\Gamma \Rightarrow \Delta$. In case (b), in order for $\mathfrak{M}$ to satisfy the right premise, it must satisfy $\Pi' \setminus \{\varphi\} \Rightarrow \Lambda'$. But $\Pi' \setminus \{\varphi\} \subseteq \Gamma$ and $\Lambda' \subseteq \Delta$, so $\mathfrak{M}$ also satisfies $\Gamma \Rightarrow \Delta$.

2. The last inference is $\wedge$R. The premises are $\Gamma \Rightarrow \Delta'$ and $\Gamma \Rightarrow \Delta''$, where $\varphi \in \Delta'$ an $\psi \in \Delta''$. By induction hypothesis, both are valid. Consider a structure $\mathfrak{M}$. We have two cases: (a) $\mathfrak{M} \not\models \varphi \wedge \psi$ or (b) $\mathfrak{M} \models \varphi \wedge \psi$. In case (a), either $\mathfrak{M} \not\models \varphi$ or $\mathfrak{M} \not\models \psi$. In the former case, in order for $\mathfrak{M}$ to satisfy $\Gamma \Rightarrow \Delta'$, it must already satisfy $\Gamma \Rightarrow \Delta' \setminus \{\varphi\}$. In the latter case, it must satisfy $\Gamma \Rightarrow \Delta'' \setminus \{\psi\}$. But since both $\Delta' \setminus \{\varphi\} \subseteq \Delta$ and $\Delta'' \setminus \{\psi\} \subseteq \Delta$, that means $\mathfrak{M}$ satisfies $\Gamma \Rightarrow \Delta$. In case (b), $\mathfrak{M}$ satisfies $\Gamma \Rightarrow \Delta$ since $\varphi \wedge \psi \in \Delta$.

3. The last inference is $\vee$L: Exercise.

4. The last inference is $\rightarrow$ L. The premises are $\Gamma \Rightarrow \Delta'$ and $\Gamma' \Rightarrow \Delta$, where $\varphi \in \Delta'$ and $\psi \in \Gamma'$. By induction hypothesis, both are valid. Consider a structure $\mathfrak{M}$. We have two cases: (a) $\mathfrak{M} \models \varphi \rightarrow \psi$ or (b) $\mathfrak{M} \not\models \varphi \rightarrow \psi$. In case (a), either $\mathfrak{M} \not\models \varphi$ or $\mathfrak{M} \models \psi$. In the former case, in order for $\mathfrak{M}$ to satisfy $\Gamma \Rightarrow \Delta'$, it must already satisfy $\Gamma \Rightarrow \Delta' \setminus \{\varphi\}$. In the latter case, it must satisfy $\Gamma' \setminus \{\psi\} \Rightarrow \Delta$. But since both $\Delta' \setminus \{\varphi\} \subseteq \Delta$ and $\Gamma' \setminus \{\psi\} \subseteq \Gamma$, that means $\mathfrak{M}$ satisfies $\Gamma \Rightarrow \Delta$. In case (b), $\mathfrak{M}$ satisfies $\Gamma \Rightarrow \Delta$ since $\varphi \rightarrow \psi \in \Gamma$.

□

Corollary 7.30. *If $\vdash_{\mathbf{LK}} \varphi$ then φ is valid.*

Corollary 7.31. *If $\Gamma \vdash_{\mathbf{LK}} \varphi$ then $\Gamma \vDash \varphi$.*

Proof. If $\Gamma \vdash_{\mathbf{LK}} \varphi$ then for some finite subset $\Gamma_0 \subseteq \Gamma$, there is a derivation of $\Gamma_0 \Rightarrow \varphi$. By Theorem 7.29, every structure $\mathfrak{M}$ either makes some $\psi \in \Gamma_0$ false or makes φ true. Hence, if $\mathfrak{M} \models \Gamma$ then also $\mathfrak{M} \models \varphi$. □

Corollary 7.32. *If Γ is satisfiable, then it is consistent.*

Proof. We prove the contrapositive. Suppose that Γ is not consistent. Then $\Gamma \vdash_{\mathbf{LK}} \bot$, i.e., there is a finite $\Gamma_0 \subseteq \Gamma$ and a derivation of $\Gamma_0 \Rightarrow \bot$. By Theorem 7.29, $\Gamma_0 \Rightarrow \bot$ is valid. Since $\mathfrak{M} \not\models \bot$ for every structure $\mathfrak{M}$, for $\mathfrak{M}$ to satisfy $\Gamma_0 \Rightarrow \bot$ there must be an $\alpha \in \Gamma_0$ so that $\mathfrak{M} \not\models \alpha$, and since $\Gamma_0 \subseteq \Gamma$, that α is also in Γ. In other words, no $\mathfrak{M}$ satisfies Γ, i.e., Γ is not satisfiable. □

7.6 Derivations with Identity predicate

Derivations with the identity predicate require additional inference rules.

Initial sequents for $=$: If t is a closed term, then $\Rightarrow t = t$ is an initial sequent.

Rules for $=$:

$$\frac{\Gamma, t_1 = t_2 \Rightarrow \Delta, \varphi(t_1)}{\Gamma, t_1 = t_2 \Rightarrow \Delta, \varphi(t_2)} = \quad \text{and} \quad \frac{\Gamma, t_1 = t_2 \Rightarrow \Delta, \varphi(t_2)}{\Gamma, t_1 = t_2 \Rightarrow \Delta, \varphi(t_1)} =$$

where t_1 and t_2 are closed terms.

Example 7.33. If s and t are ground terms, then $\varphi(s), s = t \vdash_{\mathbf{LK}} \varphi(t)$:

$$\dfrac{\dfrac{\varphi(s) \Rightarrow \varphi(s)}{\varphi(s), s = t \Rightarrow \varphi(s)}\,\mathrm{WL}}{\varphi(s), s = t \Rightarrow \varphi(t)}\,=$$

This may be familiar as the principle of substitutability of identicals, or Leibniz' Law.

LK proves that $=$ is symmetric and transitive:

$$\dfrac{\dfrac{\Rightarrow t_1 = t_1}{t_1 = t_2 \Rightarrow t_1 = t_1}\,\mathrm{WL}}{t_1 = t_2 \Rightarrow t_2 = t_1}\,= \qquad \dfrac{\dfrac{t_1 = t_2 \Rightarrow t_1 = t_2}{t_1 = t_2, t_2 = t_3 \Rightarrow t_1 = t_2}\,\mathrm{WL}}{t_1 = t_2, t_2 = t_3 \Rightarrow t_1 = t_3}\,=$$

In the proof on the left, the formula $x = t_1$ is our $\varphi(x)$. On the right, we take $\varphi(x)$ to be $t_1 = x$.

Proposition 7.34. **LK** *with initial sequents and rules for identity is sound.*

Proof. Initial sequents of the form $\Rightarrow t = t$ are 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 $=$. Then the premise $\Gamma' \Rightarrow \Delta'$ contains $t_1 = t_2$ on the left and $\varphi(t_1)$ on the right, and the conclusion is $\Gamma \Rightarrow \Delta$ where $\Gamma = \Gamma'$ and $\Delta = (\Delta' \setminus \{\varphi(t_1)\}) \cup \{\varphi(t_2)\}$. Consider a structure $\mathfrak{M}$. Since, by induction hypothesis, the premise $\Gamma' \Rightarrow \Delta'$ is valid, either (a) for

some $\alpha \in \Gamma'$, $\mathfrak{M} \not\models \alpha$, (b) for some $\alpha \in \Delta' \setminus \{\varphi(s)\}$, $\mathfrak{M} \models \alpha$, or (c) $\mathfrak{M} \models \varphi(t_1)$. In both cases cases (a) and (b), since $\Gamma = \Gamma'$, and $\Delta' \setminus \{\varphi(s)\} \subseteq \Delta$, $\mathfrak{M}$ satisfies $\Gamma \Rightarrow \Delta$. So assume cases (a) and (b) do not apply, but case (c) does. If (a) does not apply, $\mathfrak{M} \models \alpha$ for all $\alpha \in \Gamma'$, in particular, $\mathfrak{M} \models t_1 = t_2$. 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 5.41, $\mathfrak{M}, s \models \varphi(t_2)$ iff $\mathfrak{M}, s' \models \varphi(x)$ iff $\mathfrak{M}, s \models \varphi(t_1)$. Since $\mathfrak{M} \models \varphi(t_1)$ therefore $\mathfrak{M} \models \varphi(t_2)$. □

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.

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. It also follows from *the proof of* the completeness theorem that any satisfiable set of sentences has a finite or denumerable model. This result is called the Löwenheim-Skolem theorem.

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 $\neg\psi$, with ψ atomic. We might worry that the construction of $\mathfrak{M}$ interferes with the possibility of making $\neg\psi$ true. But here's where the consistency of Γ comes in: if $\neg\psi \in \Gamma$, 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 $\mathfrak{M} \models \neg\psi$. So far so good.

Now what if Γ contains complex, non-atomic formulas? Say, it contains $\varphi \wedge \psi$. 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 $\neg\varphi$, and (c) such that, whenever $\varphi \wedge \psi$ 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 $\Gamma \subseteq \Gamma^*$.

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 $\neg\psi(c) \in \Gamma$. We can't also add $\psi(c)$, since this would make the set inconsistent, and we wouldn't know whether $\mathfrak{M}$ has to make $\psi(c)$ or $\neg\psi(c)$ 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: $(\mathsf{o} + \mathsf{o}) = \mathsf{o}$). 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.

8.3 Maximally Consistent Sets of Sentences

Definition 8.1 (Maximally consistent set)**.** A set Γ of sentences is *maximally consistent* iff

1. Γ is consistent, and
2. if $\Gamma \subsetneq \Gamma'$, then Γ' is inconsistent.

An alternate definition equivalent to the above is: a set Γ of sentences is *maximally consistent* iff

1. Γ is consistent, and
2. If $\Gamma \cup \{\varphi\}$ is consistent, then $\varphi \in \Gamma$.

In other words, one cannot add sentences not already in Γ to a maximally consistent set Γ without making the resulting larger set inconsistent.

Maximally consistent sets are important in the completeness proof since we can guarantee that every consistent set of sentences Γ is contained in a maximally consistent set Γ^*, and a maximally consistent set contains, for each sentence φ, either φ or its negation $\neg\varphi$. This is true in particular for atomic sentences, so from a maximally 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 in Γ) true. The proof of this latter fact requires that $\neg\varphi \in \Gamma^*$ iff $\varphi \notin \Gamma^*$, $(\varphi \vee \psi) \in \Gamma^*$ iff $\varphi \in \Gamma^*$ or $\psi \in \Gamma^*$, etc.

Proposition 8.2. *Suppose Γ is maximally consistent. Then:*

1. *If $\Gamma \vdash \varphi$, then $\varphi \in \Gamma$.*
2. *For any φ, either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$.*
3. *$(\varphi \wedge \psi) \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$.*
4. *$(\varphi \vee \psi) \in \Gamma$ iff either $\varphi \in \Gamma$ or $\psi \in \Gamma$.*
5. *$(\varphi \to \psi) \in \Gamma$ iff either $\varphi \notin \Gamma$ or $\psi \in \Gamma$.*

Proof. Let us suppose for all of the following that Γ is maximally consistent.

1. If $\Gamma \vdash \varphi$, then $\varphi \in \Gamma$.

   Suppose that $\Gamma \vdash \varphi$. Suppose to the contrary that $\varphi \notin \Gamma$: then since Γ is maximally consistent, $\Gamma \cup \{\varphi\}$ is inconsistent, hence $\Gamma \cup \{\varphi\} \vdash \bot$. By Proposition 7.17, 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. For any φ, either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$.

   Suppose to the contrary that for some φ both $\varphi \notin \Gamma$ and $\neg\varphi \notin \Gamma$. Since Γ is maximally consistent, $\Gamma \cup \{\varphi\}$ and $\Gamma \cup \{\neg\varphi\}$ are both inconsistent, so $\Gamma \cup \{\varphi\} \vdash \bot$ and $\Gamma \cup \{\neg\varphi\} \vdash \bot$. By Proposition 7.19, Γ is inconsistent, a contradiction. Hence there cannot be such a sentence φ and, for every φ, $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$.

3. $(\varphi \wedge \psi) \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$:

   For the forward direction, suppose $(\varphi \wedge \psi) \in \Gamma$. Then $\Gamma \vdash \varphi \wedge \psi$. By Proposition 7.22, $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$. By (1), $\varphi \in \Gamma$ and $\psi \in \Gamma$, as required.

   For the reverse direction, let $\varphi \in \Gamma$ and $\psi \in \Gamma$. Then $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$. By Proposition 7.23, $\Gamma \vdash \varphi \wedge \psi$. By (1), $(\varphi \wedge \psi) \in \Gamma$.

4. $(\varphi \vee \psi) \in \Gamma$ iff either $\varphi \in \Gamma$ or $\psi \in \Gamma$.

   For the contrapositive of the forward direction, suppose that $\varphi \notin \Gamma$ and $\psi \notin \Gamma$. We want to show that $(\varphi \vee \psi) \notin \Gamma$. Since Γ is maximally consistent, $\Gamma \cup \{\varphi\} \vdash \bot$ and $\Gamma \cup \{\psi\} \vdash \bot$. By Proposition 7.20, $\Gamma \cup \{(\varphi \vee \psi)\}$ is inconsistent. Hence, $(\varphi \vee \psi) \notin \Gamma$, as required.

   For the reverse direction, suppose that $\varphi \in \Gamma$ or $\psi \in \Gamma$. Then $\Gamma \vdash \varphi$ or $\Gamma \vdash \psi$. By Proposition 7.21, $\Gamma \vdash \varphi \vee \psi$. By (1), $(\varphi \vee \psi) \in \Gamma$, as required.

5. Exercise.

□

8.4 Henkin Expansion

Part of the challenge in proving the completeness theorem is that the model we construct from a maximally consistent set Γ must make all the quantified formulas 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 constants and adding, for each formula with one free variable $\varphi(x)$ a formula of the form $\exists x\, \varphi \to \varphi(c)$, 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.

Lemma 8.3. *If Γ is consistent in $\mathcal{L}$ and $\mathcal{L}'$ is obtained from $\mathcal{L}$ by adding a denumerable set of new constant symbols $d_1, d_2, \ldots$, then Γ is consistent in $\mathcal{L}'$.*

Definition 8.4 (Saturated set)**.** A set Γ of formulas of a language $\mathcal{L}$ is *saturated* if and only if for each formula $\varphi \in \mathrm{Frm}(\mathcal{L})$ and variable x there is a constant symbol c such that $\exists x\, \varphi \to \varphi(c) \in \Gamma$.

The following definition will be used in the proof of the next theorem.

Definition 8.5. Let $\mathcal{L}'$ be as in Lemma 8.3. Fix an enumeration $\langle \varphi_1, x_1 \rangle, \langle \varphi_2, x_2 \rangle, \ldots$ of all formula-variable pairs of $\mathcal{L}'$. We define the sentences θ_n by recursion on n. Assuming that $\theta_1, \ldots, \theta_n$ have already been defined, let c_{n+1} be the first new constant symbol among the d_i that does not occur in $\theta_1, \ldots, \theta_n$, and let θ_{n+1} be the formula $\exists x_{n+1}\, \varphi_{n+1}(x_{n+1}) \to \varphi_{n+1}(c_{n+1})$. This includes the case where $n = 0$ and the list of previous θ_i's is empty, i.e., θ_1 is $\exists x_1\, \varphi_1 \to \varphi_1(c_1)$.

Theorem 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 denumerable set of new constant symbols to form $\mathcal{L}'$. By the previous Lemma, Γ is still consistent in the richer language. Further, let θ_i be as in the previous definition: then $\Gamma \cup \{\theta_1, \theta_2, \ldots\}$ is saturated by construction. Let

$$\begin{aligned} \Gamma_0 &= \Gamma \\ \Gamma_{n+1} &= \Gamma_n \cup \{\theta_{n+1}\} \end{aligned}$$

i.e., $\Gamma_n = \Gamma \cup \{\theta_1, \ldots, \theta_n\}$, and let $\Gamma' = \bigcup_n \Gamma_n$. 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-1} is

consistent but $\Gamma_n = \Gamma_{n-1} \cup \{\theta_n\}$ is inconsistent. Recall that θ_n is $\exists x_n\, \varphi_n(x_n) \to \varphi_n(c_n)$. where $\varphi(x)$ is a formula of $\mathcal{L}'$ with only the variable x_n free and not containing any constant symbols c_i where $i \geq n$.

If $\Gamma_{n-1} \cup \{\theta_n\}$ is inconsistent, then $\Gamma_{n-1} \vdash \neg\theta_n$, and hence both of the following hold:

$$\Gamma_{n-1} \vdash \exists x_n\, \varphi_n(x_n) \qquad \Gamma_{n-1} \vdash \neg\varphi_n(c_n)$$

Here c_n does not occur in Γ_{n-1} or $\varphi_n(x_n)$ (remember, it was added only with θ_n). By Theorem 7.26, from $\Gamma \vdash \neg\varphi_n(c_n)$, we obtain $\Gamma \vdash \forall x_n\, \neg\varphi_n(x_n)$. Thus we have that both $\Gamma_{n-1} \vdash \exists x_n\, \varphi_n$ and $\Gamma_{n-1} \vdash \forall x_n\, \neg\varphi_n(x_n)$, so Γ itself is inconsistent. (Note that $\forall x_n\, \neg\varphi_n(x_n) \vdash \neg\exists x_n\, \varphi_n(x_n)$.) Contradiction: Γ_{n-1} was supposed to be consistent. Hence $\Gamma_n \cup \{\theta_n\}$ is consistent. □

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 maximally so, and moreover, is saturated. The proof works by first extending the set to a saturated set, and then adding one sentence at a time, guaranteeing at each step that the set remains consistent. The union of all stages in that construction then contains, for each sentence φ, either it or its negation $\neg\varphi$, is saturated, and is also consistent.

Lemma 8.7 (Lindenbaum's Lemma)**.** *Every consistent set Γ can be extended to a maximally consistent saturated set Γ^*.*

Proof. Let Γ be consistent, and let Γ' be as in the proof of Theorem 8.6: we proved there that $\Gamma \cup \Gamma'$ is a consistent saturated set in the richer language $\mathcal{L}'$ (with the denumerable set of new constants). Let φ_0, φ_1, ... be an enumeration of all the formulas of $\mathcal{L}'$. Define $\Gamma_0 = \Gamma \cup \Gamma'$, and

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

Let $\Gamma^* = \bigcup_{n\geq 0} \Gamma_n$. Since $\Gamma' \subseteq \Gamma^*$, for each formula φ, Γ^* contains a formula of the form $\exists x\, \varphi \to \varphi(c)$ and thus is saturated.

Each Γ_n is consistent: Γ_0 is consistent by definition. If $\Gamma_{n+1} = \Gamma_n \cup \{\varphi\}$, this is because the latter is consistent. If it isn't, $\Gamma_{n+1} = \Gamma_n \cup \{\neg\varphi\}$, which must be consistent. If it weren't, i.e., both $\Gamma_n \cup \{\varphi\}$ and $\Gamma_n \cup \{\neg\varphi\}$ are inconsistent, then $\Gamma_n \vdash \neg\varphi$ and $\Gamma_n \vdash \varphi$, so Γ_n would be inconsistent contrary to induction hypothesis.

Every formula 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 that means that Γ^* is maximally consistent. □

8.6 Construction of a Model

We will begin by showing how to construct a structure which satisfies a maximally consistent, saturated set of sentences in a language $\mathcal{L}$ without $=$.

Definition 8.8 (Term model)**.** Let Γ^* be a maximally 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 $\langle t_1, \ldots, t_n \rangle \in R^{\mathfrak{M}(\Gamma^*)}$ iff $R(t_1, \ldots, t_n) \in \Gamma^*$.

Lemma 8.9 (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 $R(t_1, \ldots, t_n) \in \Gamma^*$ (the construction of $\mathfrak{M}(\Gamma^*)$.
3. $\varphi \equiv \neg\psi$: $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\mathfrak{M}(\Gamma^*) \not\models \psi$ (by definition of satisfaction). By induction hypothesis, $\mathfrak{M}(\Gamma^*) \not\models \psi$ iff $\psi \notin \Gamma^*$. By Proposition 8.2(2), $\neg\psi \in \Gamma^*$ if $\psi \notin \Gamma^*$; and $\neg\psi \notin \Gamma^*$ if $\psi \in \Gamma^*$ since Γ^* is consistent.
4. $\varphi \equiv \psi \wedge \chi$: $\mathfrak{M}(\Gamma^*) \models \varphi$ 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(3), this is the case iff $(\psi \wedge \chi) \in \Gamma^*$.
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(4)).
6. $\varphi \equiv \psi \to \chi$: exercise.
7. $\varphi \equiv \forall x\, \psi(x)$: exercise.

8. $\varphi \equiv \exists x\, \psi(x)$: First suppose that $\mathfrak{M}(\Gamma^*) \models \varphi$. By the definition of satisfaction, for some variable assignment s, $\mathfrak{M}(\Gamma^*), s \models \psi(x)$. The value $s(x)$ is some term $t \in |\mathfrak{M}(\Gamma^*)|$. Thus, $\mathfrak{M}(\Gamma^*) \models \psi(t)$, and by our induction hypothesis, $\psi(t) \in \Gamma^*$. By Theorem 7.27 we have $\Gamma^* \vdash \exists x\, \psi(x)$. Then, by Proposition 8.2(1), we can conclude that $\varphi \in \Gamma^*$.

   Conversely, suppose that $\exists x\, \psi(x) \in \Gamma^*$. Because Γ^* is saturated, $(\exists x\, \psi(x) \to \psi(c)) \in \Gamma^*$. By Proposition 7.24 together with Proposition 8.2(1), $\psi(c) \in \Gamma^*$. By inductive hypothesis, $\mathfrak{M}(\Gamma^*) \models \psi(c)$. Now consider the variable assignment with $s(x) = c^{\mathfrak{M}(\Gamma^*)}$. Then $\mathfrak{M}(\Gamma^*), s \models \psi(x)$. By definition of satisfaction, $\mathfrak{M}(\Gamma^*) \models \exists x\, \psi(x)$.

□

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 8.10. Let Γ^* be a maximally consistent 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 8.11. *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 maximally consistent, $t = t' \in \Gamma^*$ 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 $\Gamma^* \vdash R(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots,$
   for every n-place predicate symbol R and terms $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$.

□

Definition 8.12. Suppose Γ^* is a maximally consistent set in a language $\mathcal{L}$, t is a term, and $\approx$ as in the previous definition. Then:

$$[t]_\approx = \{t' : t' \in \mathrm{Trm}(\mathcal{L}), t \approx t'\}$$

and $\mathrm{Trm}(\mathcal{L})/_\approx = \{[t]_\approx : t \in \mathrm{Trm}(\mathcal{L})\}$.

Definition 8.13. 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 8.11 guarantees that this cannot happen.

Proposition 8.14. *$\mathfrak{M}/{\approx}$ 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 8.11 by induction on n. □

Lemma 8.15. *$\mathfrak{M}/{\approx} \models \varphi$ iff $\varphi \in \Gamma^*$ for all sentences φ.*

Proof. By induction on φ, just as in the proof of Lemma 8.9. 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 enumerable 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 the Gödel's completeness theorem.

Theorem 8.16 (Completeness Theorem)**.** *Let Γ be a set of sentences. If Γ is consistent, it is satisfiable.*

Proof. Suppose Γ is consistent. By Lemma 8.7, there is a $\Gamma^* \supseteq \Gamma$ which is maximally consistent and saturated. If Γ does not contain $=$, then by Lemma 8.9, $\mathfrak{M}(\Gamma^*) \models \varphi$ 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 8.15, $\mathfrak{M}/{\approx} \models \varphi$ 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 8.17 (Completeness Theorem, Second Version)**.** *For all* Γ *and* φ *sentences: if* $\Gamma \models \varphi$ *then* $\Gamma \vdash \varphi$*.*

Proof. Note that the Γ's in Corollary 8.17 and Theorem 8.16 are universally quantified. To make sure we do not confuse ourselves, let us restate Theorem 8.16 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 $\Gamma \models \varphi$. Then $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable by Proposition 5.46. Taking $\Gamma \cup \{\neg\varphi\}$ as our Δ, the previous version of Theorem 8.16 gives us that $\Gamma \cup \{\neg\varphi\}$ is inconsistent. By Proposition 7.13, $\Gamma \vdash \varphi$. □

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 copmpleteness theorem, it has a version related to entailment: if an infinite set of sentences entails something, already a finite subset does.

Definition 8.18. A set Γ of formulas is *finitely satisfiable* if and only if every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable.

Theorem 8.19 (Compactness Theorem)**.** *The following hold for any sentences* Γ *and* φ*:*

1. $\Gamma \models \varphi$ *iff there is a finite* $\Gamma_0 \subseteq \Gamma$ *such that* $\Gamma_0 \models \varphi$*.*

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, every finite subset is consistent. Then Γ itself must be consistent. For assume it is not, i.e., $\Gamma \vdash \bot$. But derivations are finite, and so already some finite subset $\Gamma_0 \subseteq \Gamma$ must be inconsistent (cf. Proposition 7.15). But we just showed they are all consistent, a contradiction. Now by completeness, since Γ is consistent, it is satisfiable. □

Example 8.20. 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.,

$$\Delta = \{c \neq t : t \in \mathrm{Trm}(\mathcal{L})\}.$$

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 8.21. Consider a language $\mathcal{L}$ containing the predicate symbol $<$, constant symbols $\mathtt{0}$, $\mathtt{1}$, 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 $\{0 < c\} \cup \{c < (\mathtt{1} \div \overline{k}) : k \in \mathbb{Z}^+\}$ (where $\overline{k} = (\mathtt{1} + (\mathtt{1} + \cdots + (\mathtt{1} + \mathtt{1}) \ldots))$ with k $\mathtt{1}$'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 8.22. 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

$$\Delta = \{\varphi_{\geq n} : n \geq 1\}$$

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 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 denumerable. An immediate consequene of this fact is that first-order logic cannot express that the size of a structure is non-enumerable: any sentence or set of sentences satisfied in all non-enumerable structures is also satisfied in some denumerable structure.

Theorem 8.23. *If Γ is consistent then it has a denumerable model, i.e., it is satisfiable in a structure whose domain is either finite or infinite but enumerable.*

Proof. If Γ is consistent, the structure $\mathfrak{M}$ delivered by the proof of the completeness theorem has a domain $|\mathfrak{M}|$ whose cardinality is bounded by that of the set of the terms of the language $\mathcal{L}$. So $\mathfrak{M}$ is at most denumerable. □

Theorem 8.24. *If Γ is consistent set of sentences in the language of first-order logic without identity, then it has a denumerable model, i.e., it is satisfiable in a structure whose domain is infinite and enumerable.*

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 $|\mathfrak{M}|$ whose cardinality is identical to that of the set of the terms of the language $\mathcal{L}$. So $\mathfrak{M}$ is denumerably infinite. □

Example 8.25 (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 non-enumerable, 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 non-enumerable, 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 enumerable models—models that contain "non-enumerable" sets but which themselves are enumerable.

8.11 Overspill

Theorem 8.26. *If a set Γ of sentences has arbitrarily large finite models, then it has an infinite model.*

Proof. Expand the language of Γ by adding countably many new constants c_0, c_1, ... and consider the set $\Gamma \cup \{c_i \neq c_j : i \neq j\}$. To say that Γ has arbitrarily large finite models means that for every $m > 0$ there is $n \geq m$ such that Γ has a model of cardinality n. This implies that $\Gamma \cup \{c_i \neq c_j : i \neq j\}$ is finitely satisfiable. By compactness, $\Gamma \cup \{c_i \neq c_j : i \neq j\}$ has a model $\mathfrak{M}$ whose domain must be infinite, since it satisfies all inequalities $c_i \neq c_j$. □

Proposition 8.27. *There is no sentence φ of any first-order language that is true in a structure $\mathfrak{M}$ if and only if the domain $|\mathfrak{M}|$ of the structure is infinite.*

Proof. If there were such a φ, its negation $\neg\varphi$ would be true in all and only the finite structures, and it would therefore have arbitrarily large finite models but it would lack an infinite model, contradicting Theorem 8.26. □

Part IV

Computability and Incompleteness

Chapter 9

Recursive Functions

9.1 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.

9.2 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 9.1. 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 9.2. 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 9.3. 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.

9.3 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.

9.4 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

$$\begin{aligned}\text{pred}'(0,y) &= \text{zero}(y) = 0\\ \text{pred}'(x+1,y) &= P_0^3(x,\text{pred}'(x,y),y) = x\end{aligned}$$

and then define pred from it by composition, e.g., as $\text{pred}(x) = \text{pred}'(P_0^1(x),\text{zero}(x))$.

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.

9.5 Primitive Recursive Relations

Definition 9.4. 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, $\neg P$
2. Conjunction, $P \wedge Q$
3. Disjunction, $P \vee Q$
4. If ... then, $P \to Q$

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, $(\exists x < y)\ \varphi(x) \leftrightarrow \neg(\forall x < y)\ \neg\varphi(x)$. 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, $\mathrm{cond}(x, y, z)$, defined by

$$\mathrm{cond}(x, y, z) = \begin{cases} y & \text{if } x = 0 \\ z & \text{otherwise} \end{cases}$$

This is defined recursively by

$$\mathrm{cond}(0, y, z) = y, \quad \mathrm{cond}(x + 1, y, z) = z.$$

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

$$f(\vec{x}) = \mathrm{cond}(\chi_{\neg R_0}(\vec{x}), g_0(\vec{x}), g_1(\vec{x})).$$

For m greater than 1, one can just compose definitions of this form.

9.6 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 9.5. *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 $(\min x < y + 1) \ldots$.

9.7 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

$$\text{Prime}(x) \Leftrightarrow x \geq 2 \wedge (\forall y \leq x)\ (y \mid x \rightarrow y = 1 \vee y = x)$$

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 $x! + 1$.

$$\text{nextPrime(x)} = (\min y \leq x! + 1)\ (y > x \wedge \text{Prime}(y)).$$

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 $\leq x! + 1$.)

9.8 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

$$R(i,s) \text{ iff } p_i \mid s \wedge (\forall j < s)\ (j > i \rightarrow p_j \not\mid s)$$

   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:

$$\begin{aligned}\text{concat}(s,t) = (\min v < \text{sequenceBound}(s+t, \text{len}(s)+\text{len}(t)))\\ (\text{len}(v) = \text{len}(s)+\text{len}(t) \wedge\\ (\forall i < \text{len}(s))\ ((v)_i = (s)_i) \wedge\\ (\forall j < \text{len}(t))\ ((v)_{\text{len}(s)+j} = (t)_j))\end{aligned}$$

We will write $s \frown t$ instead of concat(s,t).

9.9 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.

9.10 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.

9.11 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 9.6. 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 9.7. 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.

9.12 The Normal Form Theorem

Theorem 9.8 (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.

9.13 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 9.9. *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. □

9.14 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 9.10. 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 9.10 and Definition 9.7 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 9.10 is *less* general than Definition 9.7. 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 10

Arithmetization of Syntax

10.1 Introduction

In order to connect computability and logic, we need a way to talk about the objects of logic (symbols, terms, formulas, 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 an enumerable 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 formulas) 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 formulas, 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 functions. 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, formulas, 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).

10.2 Coding Symbols

The basic language $\mathcal{L}$ of first order logic makes use of the symbols

$$\bot \quad \neg \quad \vee \quad \wedge \quad \to \quad \forall \quad \exists \quad = \quad (\quad) \quad ,$$

together with enumerable sets of variables and constant symbols, and enumerable 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 enumerable 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 10.1. 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:

$\bot$	$\neg$	$\vee$	$\wedge$	$\to$	$\forall$
$\langle 0,0 \rangle$	$\langle 0,1 \rangle$	$\langle 0,2 \rangle$	$\langle 0,3 \rangle$	$\langle 0,4 \rangle$	$\langle 0,5 \rangle$
$\exists$	$=$	$($	$)$	$,$	
$\langle 0,6 \rangle$	$\langle 0,7 \rangle$	$\langle 0,8 \rangle$	$\langle 0,9 \rangle$	$\langle 0,10 \rangle$	

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 10.2. *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 10.3. 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 10.4. 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

$$\langle \mathsf{c}_=, \mathsf{c}_(, \mathsf{c}_{v_0}, \mathsf{c}_,, \mathsf{c}_{c_0}, \mathsf{c}_) \rangle.$$

Here, $\mathsf{c}_=$ 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

$$\begin{aligned} &2^{\mathsf{c}_=+1} \cdot 3^{\mathsf{c}_(+1} \cdot 5^{\mathsf{c}_{v_0}+1} \cdot 7^{\mathsf{c}_,+1} \cdot 11^{\mathsf{c}_{c_0}+1} \cdot 13^{\mathsf{c}_)+1} = \\ &\qquad 2^{2^1 \cdot 3^8+1} \cdot 3^{2^1 \cdot 3^9+1} \cdot 5^{2^2 \cdot 3^1+1} \cdot 7^{2^1 \cdot 3^{11}+1} \cdot 11^{2^3 \cdot 3^1+1} \cdot 13^{2^1 \cdot 3^{10}+1} = \\ &\qquad\qquad 2^{13\,123} \cdot 3^{39\,367} \cdot 5^{13} \cdot 7^{354\,295} \cdot 11^{25} \cdot 13^{118\,099}. \end{aligned}$$

10.3 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 10.5. *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 $(y)_i = {}^{\ulcorner}c_j{}^{\urcorner}$, 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

$$(y)_i = {}^{\ulcorner}f_j^n({}^{\urcorner} \frown \text{flatten}(z) \frown {}^{\ulcorner})^{\urcorner},$$

and moreover $(y)_{k-1} = x$. The function $\text{flatten}(z)$ turns the sequence $\langle {}^{\ulcorner}t_1{}^{\urcorner}, \ldots, {}^{\ulcorner}t_n{}^{\urcorner} \rangle$ into ${}^{\ulcorner}t_1, \ldots, t_n{}^{\urcorner}$ 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 10.5. 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 $x = {}^{\ulcorner}v_j{}^{\urcorner}$, or

2. there is a j such that $x = {}^{\ulcorner}c_j{}^{\urcorner}$, 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:

$$\begin{aligned}
\text{lTerm}(x, 0) &= \text{Var}(x) \vee \text{Const}(x) \\
\text{lTerm}(x, l+1) &= \text{Var}(x) \vee \text{Const}(x) \vee \\
&\qquad (\exists z < x)\ ((\forall i < \text{len}(z))\ \text{lTerm}((z)_i, l) \wedge \\
&\qquad\quad (\exists j < x)\ x = ({}^{\ulcorner}f_j^{\text{len}(z)}({}^{\ulcorner} \frown \text{flatten}(z) \frown {}^{\urcorner})^{\urcorner}))
\end{aligned}$$

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 10.6. *The function* $\text{num}(n) = {}^{\ulcorner}\overline{n}^{\urcorner}$ *is primitive recursive.*

Proof. We define $\text{num}(n)$ by primitive recursion:

$$\begin{aligned}
\text{num}(0) &= {}^{\ulcorner}0^{\urcorner} \\
\text{num}(n+1) &= {}^{\ulcorner}\prime({}^{\ulcorner} \frown \text{num}(n) \frown {}^{\urcorner})^{\urcorner}.
\end{aligned}$$

□

10.4 Coding Formulas

Proposition 10.7. *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 10.8. *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 formulas 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 10.9. *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 10.10. *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

$$(\forall i < \mathrm{len}(x))\ (\forall z < x)\ ((\exists j < z)\ z = {}^{\#}v_j{}^{\#} \to \neg \mathrm{FreeOcc}(x, z, i)).$$

□

10.5 Substitution

Proposition 10.11. *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 10.12. *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. □

10.6 Derivations in LK

In order to arithmetize derivations, we must represent derivations as numbers. Since derivations are trees of sequents where each inference carries also a label, a recursive representation is the most obvious approach: we represent a derivation as a tuple, the components of which are the end-sequent, the label, and the representations of the sub-derivations leading to the premises of the last inference.

Definition 10.13. If Γ is a finite set of sentences, $\Gamma = \{\varphi_1, \ldots, \varphi_n\}$, then ${}^\#\Gamma^\# = \langle {}^\#\varphi_1{}^\#, \ldots, {}^\#\varphi_n{}^\# \rangle$.

If $\Gamma \Rightarrow \Delta$ is a sequent, then a Gödel number of $\Gamma \Rightarrow \Delta$ is

$${}^\#\Gamma \Rightarrow \Delta^\# = \langle {}^\#\Gamma^\#, {}^\#\Delta^\# \rangle$$

If π is a derivation in **LK**, then ${}^\#\pi^\#$ is

1. $\langle 0, {}^\#\Gamma \Rightarrow \Delta^\# \rangle$ if π consists only of the initial sequent $\Gamma \Rightarrow \Delta$.
2. $\langle 1, {}^\#\Gamma \Rightarrow \Delta^\#, k, {}^\#\pi'^\# \rangle$ if π 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 π' is the immediate subproof ending in the premise of the last inference.

Rule:	Contr	$\neg$ left	$\neg$ right	$\wedge$ left	$\vee$ right	$\rightarrow$ right
k:	1	2	3	4	5	6

Rule:	$\forall$ left	$\forall$ right	$\exists$ left	$\exists$ right	=
k:	7	8	9	10	11

3. $\langle 2, {}^\#\Gamma \Rightarrow \Delta^\#, k, {}^\#\pi'^\#, {}^\#\pi''^\# \rangle$ if π 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 π', π'' are the immediate subproof ending in the left and right premise of the last inference, respectively.

Rule:	Cut	$\wedge$ right	$\vee$ left	$\rightarrow$ left
k:	1	2	3	4

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, $(s)_1$ gives us the Gödel number of its end-sequent. Some are much harder. We'll at least sketch how to do this. The goal is to show that the relation "π is a derivation of φ from Γ" is primitive recursive on the Gödel numbers of π and φ.

Proposition 10.14. *The following relations are primitive recursive:*

1. $\varphi \in \Gamma$.

2. $\Gamma \subseteq \Delta$.

3. $\Gamma \Rightarrow \Delta$ *is an initial sequent.*

4. $\Gamma \Rightarrow \Delta$ *follows from* $\Gamma' \Rightarrow \Delta'$ *(and* $\Gamma'' \Rightarrow \Delta''$*) by a rule of* **LK**.

5. π *is a correct* **LK***-derivation.*

Proof. We have to show that the corresponding relations between Gödel numbers of formulas, sequences of Gödel numbers of formulas (which code sets of formulas), and Gödel numbers of sequents, are primitive recursive.

1. $\varphi \in \Gamma$ iff ${}^{\ulcorner}\varphi^{\urcorner}$ occurs in the sequence ${}^{\ulcorner}\Gamma^{\urcorner}$, i.e, $\mathrm{IsIn}(x, g) \Leftrightarrow (\exists i < \mathrm{len}(g))\ (g)_i = x$. We'll abbreviate this as $x \in g$.

2. $\Gamma \subseteq \Delta$ iff every element of ${}^{\ulcorner}\Gamma^{\urcorner}$ is also an an element of ${}^{\ulcorner}\Delta^{\urcorner}$, i.e., $\mathrm{SubSet}(g, d) \Leftrightarrow (\forall i < \mathrm{len}(g))\ (g)_i \in d$. We'll abbreviate this as $g \subseteq d$.

3. $\Gamma \Rightarrow \Delta$ is an initial sequent if either there is a sentence φ such that $\Gamma \Rightarrow \Delta$ is $\varphi \Rightarrow \varphi$, or there is a term t such that $\Gamma \Rightarrow \Delta$ is $\emptyset \Rightarrow t = t$. In terms of Gödel numbers, $\mathrm{InitSeq}(s)$ holds iff

$$\begin{aligned}&(\exists x < s)\ (\mathrm{Sent}(x) \wedge s = \langle\langle x\rangle, \langle x\rangle\rangle) \vee \\ &(\exists t < s)\ (\mathrm{Term}(t) \wedge s = \langle 0, \langle {}^{\ulcorner}{=}{}^{\urcorner}({}^{\ulcorner} \frown t \frown {}^{\urcorner}, {}^{\ulcorner} \frown t \frown {}^{\urcorner})^{\urcorner}\rangle\rangle).\end{aligned}$$

4. Here we have to show that for each rule of inference R the relation $\mathrm{FollowsBy}_R(s, s')$ which holds if s and s' are the Gödel numbers of conclusion and premise of a correct application of R is primitive recursive. If R has two premises, $\mathrm{FollowsBy}_R$ of course has three arguments.

   For instance, $\Gamma \Rightarrow \Delta$ follows correctly from $\Gamma' \Rightarrow \Delta'$ by $\exists$right iff $\Gamma = \Gamma'$ and there is a formula φ, a variable x and a closed term t such that $\varphi[t/x] \in \Delta'$ and $\exists x\, \varphi \in \Delta$, for every $\psi \in \Delta$, either $\psi = \exists x\, \varphi$ or $\psi \in \Delta'$, and for every $\psi \in \Delta'$, $\psi = \varphi[t/x]$ or $\psi \in \Delta$. We just have to translate this into Gödel numbers. If $s = {}^{\ulcorner}\Gamma \Rightarrow \Delta^{\urcorner}$ then $(s)_0 = {}^{\ulcorner}\Gamma^{\urcorner}$ and $(s)_1 = {}^{\ulcorner}\Delta^{\urcorner}$. So, $\mathrm{FollowsBy}_{\exists\mathrm{right}}(s, s')$ holds iff

$$\begin{aligned}&(s)_0 \subseteq (s')_0 \wedge (s')_0 \subseteq (s)_0 \wedge \\ &(\exists f < s)\ (\exists x < s)\ (\exists t < s')\ (\mathrm{Frm}(f) \wedge \mathrm{Var}(x) \wedge \mathrm{Term}(t) \wedge \\ &\quad \mathrm{Subst}(f, t, x) \in (s')_1 \wedge \#(\exists) \frown x \frown f \in (s)_1 \wedge \\ &\quad (\forall i < \mathrm{len}((s)_1))\ (((s)_1)_i = \#(\exists) \frown x \frown f \vee ((s)_1)_i \in (s')_1) \wedge \\ &\quad (\forall i < \mathrm{len}((s')_1))\ (((s)'_1)_i = \mathrm{Subst}(f, t, x) \vee ((s')_1)_i \in (s)_1))\end{aligned}$$

   The individual lines express, respectively, "$\Gamma \subseteq \Gamma' \wedge \Gamma' \subseteq \Gamma$," "there is a formula with Gödel number f, a variable with Gödel number x, and a term with Gödel number t," "$\varphi[t/x] \in \Delta' \wedge \exists x\, \varphi \in \Delta$," "for all $\psi \in \Delta$,

either $\psi = \exists x\, \varphi$ or $\psi \in \Delta'$," "for all $\psi \in \Delta'$, either $\psi = \varphi[t/x]$ or $\psi \in \Delta$. Note that in the last two lines, we quantify over the elements ψ of Δ and Δ' not directly, but via their place i in the Gödel numbers of Δ and Δ'. (Remember that ${}^{\#}\Delta^{\#}$ is the number of a sequence of Gödel numbers of formulas in Δ.)

5. We first define a helper relation $\mathrm{hDeriv}(s, n)$ which holds if s codes a correct derivation at least to n inferences up from the end sequent. If $n = 0$ we let the relation be satisfied by default. Otherwise, $\mathrm{hDeriv}(s, n+1)$ iff either s consists just of an initial sequent, or it ends in a correct inference and the codes of the immediate subderivations satisfy $\mathrm{hDeriv}(s, n)$.

$$
\begin{aligned}
&\mathrm{hDeriv}(s, 0) \Leftrightarrow \mathrm{true}\\
&\mathrm{hDeriv}(s, n+1) \Leftrightarrow\\
&\qquad ((s)_0 = 0 \wedge \mathrm{InitialSeq}((s)_1)) \vee\\
&\qquad ((s)_0 = 1 \wedge\\
&\qquad\quad ((s)_2 = 1 \wedge \mathrm{FollowsBy}_{\mathrm{Contr}}((s)_1, ((s)_3)_1)) \vee\\
&\qquad\qquad \vdots\\
&\qquad\quad ((s)_2 = 11 \wedge \mathrm{FollowsBy}_{=}((s)_1, ((s)_3)_1)) \wedge\\
&\qquad\quad \mathrm{hDeriv}((s)_3, n)) \vee\\
&\qquad ((s)_0 = 2 \wedge\\
&\qquad\quad ((s)_2 = 1 \wedge \mathrm{FollowsBy}_{\mathrm{Cut}}((s)_1, ((s)_3)_1), ((s)_4)_1)) \vee\\
&\qquad\qquad \vdots\\
&\qquad\quad ((s)_2 = 4 \wedge \mathrm{FollowsBy}_{\to\mathrm{left}}((s)_1, ((s)_3)_1), ((s)_4)_1)) \wedge\\
&\qquad\quad \mathrm{hDeriv}((s)_3, n) \wedge \mathrm{hDeriv}((s)_4, n))
\end{aligned}
$$

This is a primitive recursive definition. If the number n is large enough, e.g., larger than the maximum number of inferences between an initial sequent and the end sequent in s, it holds of s iff s is the Gödel number of a correct derivation. The number s itself is larger than that maximum number of inferences. So we can now define $\mathrm{Deriv}(s)$ by $\mathrm{hDeriv}(s, s)$.

□

Proposition 10.15. *Suppose Γ is a primitive recursive set of sentences. Then the relation $\mathrm{Prf}_\Gamma(x, y)$ expressing "x is the code of a derivation π of $\Gamma_0 \Rightarrow \varphi$ for some finite $\Gamma_0 \subseteq \Gamma$ and x 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 $\mathrm{Prf}_\Gamma(x, y)$ which holds iff y is the Gödel number of a sentence φ and x is the code of an **LK**-derivation with end sequent $\Gamma_0 \Rightarrow \varphi$ is primitive recursive.

By the previous proposition, the property $\mathrm{Deriv}(x)$ which holds iff x is the code of a correct derivation π in **LK** is primitive recursive. If x is such a code, then $(x)_1$ is the code of the end sequent of π, and so $((x)_1)_0$ is the code of the left side of the end sequent and $((x)_1)_1$ the right side. So we can express "the right side of the end sequent of π is φ" as $\mathrm{len}(((x)_1)_1) = 1 \wedge (((x)_1)_1)_0 = x$. The left side of the end sequent of π is of course automatically finite, we just have to express that every sentence in it is in Γ. Thus we can define $\mathrm{Prf}_\Gamma(x, y)$ by

$$\begin{aligned}\mathrm{Prf}_\Gamma(x, y) \Leftrightarrow\ & \mathrm{Sent}(y) \wedge \mathrm{Deriv}(x) \wedge \\ & (\forall i < \mathrm{len}(((x)_1)_0))\ R_\Gamma((((x)_1)_0)_i) \wedge \\ & \mathrm{len}(((x)_1)_1) = 1 \wedge (((x)_1)_1)_0 = x\end{aligned}$$

□

Chapter 11

Representability in Q

11.1 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 $\{\varphi : \mathbf{Q} \vdash \varphi\}$ 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):

$$\forall x \, \forall y \, (x' = y' \to x = y) \tag{Q_1}$$
$$\forall x \, 0 \neq x' \tag{Q_2}$$
$$\forall x \, (x \neq 0 \to \exists y \, x = y') \tag{Q_3}$$
$$\forall x \, (x + 0) = x \tag{Q_4}$$
$$\forall x \, \forall y \, (x + y') = (x + y)' \tag{Q_5}$$
$$\forall x \, (x \times 0) = 0 \tag{Q_6}$$
$$\forall x \, \forall y \, (x \times y') = ((x \times y) + x) \tag{Q_7}$$
$$\forall x \, \forall y \, (x < y \leftrightarrow \exists z \, (z' + x) = y) \tag{Q_8}$$

For each natural number n, define the numeral $\overline{n}$ to be the term $0^{\prime\prime\ldots\prime}$ where there are n tick marks in all. So, $\overline{0}$ is the constant symbol 0 by itself, $\overline{1}$ is $0'$, $\overline{2}$ is $0''$, 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**:

$$(\varphi(0) \wedge \forall x\,(\varphi(x) \to \varphi(x'))) \to \forall x\,\varphi(x)$$

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

$$\forall y\,((\varphi(0) \wedge \forall x\,(\varphi(x) \to \varphi(x'))) \to \forall x\,\varphi(x))$$

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 11.1. 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. $\forall y\,(\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, y) \to \overline{m} = y)$.

There are other ways of stating the definition; for example, we could equivalently require that **Q** proves $\forall y\,(\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, y) \leftrightarrow y = \overline{m})$.

Theorem 11.2. *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.

11.2 Functions Representable in Q are Computable

Lemma 11.3. *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 formulas 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, formulas, 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 10.6) and Subst (Proposition 10.11). 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 $n_0, \ldots, n_k$, 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

$$f(n_0, \ldots, n_k) = (\mu s\, R(n_0, \ldots, n_k, s))_1.$$

□

11.3 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 11.4. *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 $a_0, \ldots, a_n$ 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 11.5. 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 11.6. $a \equiv b \mod c$ 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 11.7. *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.

11.4 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 11.8. *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

$$\hat{h}(x, z) = \mu d\ (\beta(d, 0) = f(\vec{z}) \wedge \forall i < x\, \beta(d, i+1) = g(i, \beta(d, i), \vec{z})).$$

Note: no primitive recursion is needed here, just minimization. The function we minimize is regular because of the beta function lemma Lemma 11.4.

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. □

11.5 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

$$\overline{n} + \overline{m} = \overline{n + m} \text{ and}$$
$$\forall y\, ((\overline{n} + \overline{m}) = y \rightarrow y = \overline{n + m}).$$

Proposition 11.9. *The zero function* $\mathrm{zero}(x) = 0$ *is represented in* **Q** *by* $y = 0$.

Proposition 11.10. *The successor function* $\mathrm{succ}(x) = x + 1$ *is represented in* **Q** *by* $y = x'$.

Proposition 11.11. *The projection function* $P_i^n(x_0, \ldots, x_{n-1}) = x_i$ *is represented in* **Q** *by* $y = x_i$.

Proposition 11.12. *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*

$$(x_0 = x_1 \wedge y = \overline{1}) \vee (x_0 \neq x_1 \wedge y = \overline{0}).$$

The proof requires the following lemma.

Lemma 11.13. *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 $\forall x \forall y\, x' = y' \to x = y$. Using a quantifier axiom, we have $\overline{n}' = \overline{k}' \to \overline{n} = \overline{k}$. Using propositional logic, we can conclude, in $\mathbf{Q}$, $\overline{n} \neq \overline{k} \to \overline{n}' \neq \overline{k}'$. 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 $\mathbf{Q}$ can prove that different numerals denote different objects. For example, $\mathbf{Q}$ 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 $\mathbf{Q}$.

Proof of Proposition 11.12. If $n = m$, then $\overline{n}$ and $\overline{m}$ are the same term, and $\chi_=(n, m) = 1$. But $\mathbf{Q} \vdash (\overline{n} = \overline{m} \wedge \overline{1} = \overline{1})$, so it proves $\varphi_=(\overline{n}, \overline{m}, \overline{1})$. If $n \neq m$, then $\chi_=(n, m) = 0$. By Lemma 11.13, $\mathbf{Q} \vdash \overline{n} \neq \overline{m}$ and so also $(\overline{n} \neq \overline{m} \wedge \mathsf{o} = \mathsf{o})$. 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 $\mathbf{Q} \vdash \forall(\varphi_=(\overline{n}, \overline{m}, y) \to y = \overline{1})$. Arguing informally, suppose $\varphi_=(\overline{n}, \overline{m}, y)$, i.e.,

$$(\overline{n} = \overline{n} \wedge y = \overline{1}) \vee (\overline{n} \neq \overline{n} \wedge y = \overline{0})$$

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

$$(\overline{n} = \overline{m} \wedge y = \overline{1}) \vee (\overline{n} \neq \overline{m} \wedge y = \overline{0})$$

Here, the left disjunct contradicts $\overline{n} \neq \overline{m}$, which is provable in $\mathbf{Q}$ by Lemma 11.13; the right disjunct entails $y = \overline{0}$. □

Proposition 11.14. *The addition function* $\mathrm{add}(x_0, x_1) = x_0 + x_1$ *is is represented in* $\mathbf{Q}$ *by*

$$y = (x_0 + x_1).$$

Lemma 11.15. $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$

Proof. We prove this by induction on m. If $m = 0$, the claim is that $\mathbf{Q} \vdash (\overline{n} + \mathsf{o}) = \overline{n}$. 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 11.14. 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 11.15 that $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$.

We also have to show that if $\text{add}(n, m) = k$, then

$$\mathbf{Q} \vdash \forall y\, (\varphi_{\text{add}}(\overline{n}, \overline{m}, y) \to y = \overline{k}).$$

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 11.16. *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 11.17. $\mathbf{Q} \vdash (\overline{n} \times \overline{m}) = \overline{n \cdot m}$

Proof. Exercise. □

11.6 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 formulas $\varphi_f, \varphi_{g_0}, \ldots, \varphi_{g_{k-1}}$ 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 $\exists y\, (\varphi_g(x, y) \wedge \varphi_f(y, z))$ is a good candidate for $\varphi_h(x, z)$. We just have to verify that **Q** proves the relevant formulas.

Proposition 11.18. *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,

$$\mathbf{Q} \vdash \varphi_g(\overline{n}, \overline{k}) \wedge \varphi_f(\overline{k}, \overline{m})$$

and consequently also

$$\mathbf{Q} \vdash \exists y\, (\varphi_g(\overline{n}, y) \wedge \varphi_f(y, \overline{m})),$$

i.e., $\mathbf{Q} \vdash \varphi_h(n, m)$. □

Proposition 11.19. *If $h(n) = m$, then $\mathbf{Q} \vdash \forall z\, (\varphi_h(\overline{n}, z) \to z = \overline{m})$.*

Proof. Suppose $h(n) = m$, i.e., $f(g(n)) = m$. Let $k = g(n)$. Then

$$\mathbf{Q} \vdash \forall y\, (\varphi_g(\overline{n}, y) \to y = \overline{k})$$

since φ_g represents g, and

$$\mathbf{Q} \vdash \forall z\, (\varphi_f(\overline{k}, z) \to z = \overline{m})$$

since φ_f represents f. Using just a little bit of logic, we can show that also

$$\mathbf{Q} \vdash \forall z\, (\exists y\, (\varphi_g(\overline{n}, y) \wedge \varphi_f(y, z)) \to z = \overline{m}).$$

i.e., $\mathbf{Q} \vdash \forall y\, (\varphi_h(\overline{n}, y) \to y = \overline{m})$. □

The same idea works in the more complex case where f and g_i have arity greater than 1.

Proposition 11.20. *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*

$$\exists y_0, \ldots\, \exists y_{k-1}\, (\varphi_{g_0}(x_0, \ldots, x_{l-1}, y_0) \wedge \cdots \wedge \\ !A_{g_{k-1}}(x_0, \ldots, x_{l-1}, y_{k-1}) \wedge \varphi_f(y_0, \ldots, y_{k-1}, z))$$

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. □

11.7 Regular Minimization is Representable in Q

Let's consider unbounded search. Suppose $g(x, z)$ is regular and representable in **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:

$$\varphi_f(z, y) \equiv \varphi_g(y, z, 0) \wedge \forall w\,(w < y \to \neg\varphi_g(w, z, 0)).$$

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 **Q** is strong enough to prove.

Lemma 11.21. *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 **Q** proves $(x' + 0) = (x + 0)'$. But we have:

$$\mathbf{Q} \vdash (x' + 0) = x' \quad \text{by axiom } Q_4 \tag{11.1}$$
$$\mathbf{Q} \vdash (x + 0) = x \quad \text{by axiom } Q_4 \tag{11.2}$$
$$\mathbf{Q} \vdash (x + 0)' = x' \quad \text{by eq. (11.2)} \tag{11.3}$$
$$\mathbf{Q} \vdash (x' + 0) = (x + 0)' \quad \text{by eq. (11.1) and eq. (11.3)}$$

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 **Q** proves $(x' + \overline{n}') = (x + \overline{n}')'$. We have:

$$\mathbf{Q} \vdash (x' + \overline{n}') = (x' + \overline{n})' \quad \text{by axiom } Q_5 \tag{11.4}$$
$$\mathbf{Q} \vdash (x' + \overline{n}') = (x + \overline{n}')' \quad \text{inductive hypothesis} \tag{11.5}$$
$$\mathbf{Q} \vdash (x' + \overline{n})' = (x + \overline{n}')' \quad \text{by eq. (11.4) and eq. (11.5).}$$

□

It is again worth mentioning that this is weaker than saying that **Q** proves $\forall x\,\forall y\,(x' + y) = (x + y)'$. Although this sentence is true in $\mathfrak{N}$, **Q** does not prove it.

Lemma 11.22. *1.* $\mathbf{Q} \vdash \forall x\,\neg x < 0$.

2. For every natural number n,

$$\mathbf{Q} \vdash \forall x\,(x < \overline{n + 1} \to (x = 0 \vee \cdots \vee x = \overline{n})).$$

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 $\neg\exists y\,(y' + x) = \mathsf{o}$ 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 $x < \overline{1} \to x = \mathsf{o}$. 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 11.23. *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 $m = 0$. $\mathbf{Q} \vdash \forall y\,(y \neq \mathsf{o} \to \exists z\, y = z')$ 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: $y \neq \mathsf{o} \to (\mathsf{o} < y \vee y < \mathsf{o})$, 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:

$$\begin{aligned} y &= (z' + \overline{m}) && \text{by assumption} \\ (z' + \overline{m}) &= (u'' + \overline{m}) && \text{from } z = u' \\ (u'' + \overline{m}) &= (u' + \overline{m})' && \text{by Lemma 11.21} \\ (u' + \overline{m})' &= (u' + \overline{m}') && \text{by } Q_5\text{, so} \\ y &= (u' + \overline{m+1}) \end{aligned}$$

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 11.24. *If $\varphi_g(x, z, y)$ represents $g(x, y)$ in $\mathbf{Q}$, then*

$$\varphi_f(z, y) \equiv \varphi_g(y, z, 0) \wedge \forall w\, (w < y \to \neg\varphi_g(w, z, 0)).$$

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.,

$$\mathbf{Q} \vdash \varphi_g(\overline{m}, \overline{n}, 0) \wedge \forall w\, (w < \overline{m} \to \neg\varphi_g(w, \overline{n}, 0)).$$

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

$$\mathbf{Q} \vdash \neg\varphi_g(\overline{k}, \overline{n}, 0).$$

We get that

$$\mathbf{Q} \vdash \forall w\, (w < \overline{m} \to \neg\varphi_g(w, \overline{n}, 0)). \tag{11.6}$$

by Lemma 11.22 (by (1) in case $m = 0$ and by (2) otherwise).

Now let's show that if $f(n) = m$, then $\mathbf{Q} \vdash \forall y\, (\varphi_f(\overline{n}, y) \to y = \overline{m})$. 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) $\forall w\, (w < y \to \neg\varphi_g(w, \overline{n}, 0))$. By Lemma 11.23, $(y < \overline{m} \vee \overline{m} < y) \vee y = \overline{m}$. We'll show that both $y < \overline{m}$ and $\overline{m} < y$ leads to a contradiction.

If $\overline{m} < y$, then $\neg\varphi_g(\overline{m}, \overline{n}, 0)$ 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 $\mathbf{Q} \vdash \forall w\, (w < \overline{m} \to \neg\varphi_g(w, \overline{n}, 0))$ by eq. (11.6), we get $\neg\varphi_g(y, \overline{n}, 0)$. This again contradicts (a). □

11.8 Computable Functions are Representable in Q

Theorem 11.25. *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 11.8, 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 11.9 to 11.12, 11.14 and 11.16), and that any function defined from representable functions by composition or regular minimization (Proposition 11.20, Proposition 11.24) 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.

11.9 Representing Relations

Let us say what it means for a *relation* to be representable.

Definition 11.26. 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 $\neg\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$.

Theorem 11.27. *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 $\neg\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$. 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 11.2, 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

$$\forall y\, (\varphi_{\chi_R}(\overline{n_0}, \ldots, \overline{n_k}, y) \to y = \overline{0}).$$

Since $\mathbf{Q}$ proves $\overline{0} \neq \overline{1}$, $\mathbf{Q}$ proves $\neg\varphi_{\chi_R}(\overline{n_0}, \ldots, \overline{n_k}, \overline{1})$, and so it proves $\neg\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$. □

11.10 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 11.28. $\mathbf{Q}$ *is undecidable, i.e., the relation*

$$\mathrm{Prov}_{\mathbf{Q}}(y) \Leftrightarrow \mathrm{Sent}(y) \wedge \exists x\, \mathrm{Prf}_{\mathbf{Q}}(x, y)$$

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 9.8. 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 $\mathbf{Q} \vdash \neg\psi_T(\overline{e}, \overline{n}, \overline{s})$ for every s. But $\mathbf{Q}$ is ω-consistent, i.e., if $\mathbf{Q} \vdash \neg\varphi(\overline{n})$ 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 9.9, so $\mathrm{Prov}_{\mathbf{Q}}$ cannot be either. □

Corollary 11.29. *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 $\vdash \omega \to \varphi$, where ω is the conjunction of the axioms of **Q**. □

Chapter 12

Incompleteness and Provability

12.1 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 12.1. *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* $\varphi \leftrightarrow \psi(\ulcorner\varphi\urcorner)$.

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.

12.2 The Fixed-Point Lemma

The fixed-point lemma says that for any formula $\psi(x)$, there is a sentence φ such that $\mathbf{T} \vdash \varphi \leftrightarrow \psi(\ulcorner\varphi\urcorner)$, 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 **T**." Then $\alpha(x)$ would be "yields a sentence unprovable in **T** when applied to the Gödel number of its diagonalization."

To formalize this in **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 **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 **T** can prove

$$diag(\ulcorner\psi(diag(x))\urcorner) = \ulcorner\varphi\urcorner,$$

it can prove $\psi(diag(\ulcorner\psi(diag(x))\urcorner)) \leftrightarrow \psi(\ulcorner\varphi\urcorner)$. But the left hand side is, by definition, φ.

Of course, $diag$ will in general not be a function symbol of **T**, and certainly is not one of **Q**. But, since diag is computable, it is *representable* in **Q** by some formula $\theta_{\mathrm{diag}}(x, y)$. So instead of writing $\psi(diag(x))$ we can write $\exists y\,(\theta_{\mathrm{diag}}(x, y) \wedge \psi(y))$. Otherwise, the proof sketched above goes through, and in fact, it goes through already in **Q**.

Lemma 12.2. *Let $\psi(x)$ be any formula with one free variable x. Then there is a sentence φ such that $\mathbf{Q} \vdash \varphi \leftrightarrow \psi(\ulcorner\varphi\urcorner)$.*

Proof. Given $\psi(x)$, let $\alpha(x)$ be the formula $\exists y\,(\theta_{\mathrm{diag}}(x, y) \wedge \psi(y))$ 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

$$!D_{\text{diag}}(\ulcorner \alpha(x) \urcorner, \ulcorner \varphi \urcorner) \tag{12.1}$$

$$\forall y\, (\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y) \to y = \ulcorner \varphi \urcorner). \tag{12.2}$$

Now we show that $\mathbf{Q} \vdash \varphi \leftrightarrow \psi(\ulcorner \varphi \urcorner)$. 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

$$\exists y\, (\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y) \wedge \psi(y)).$$

Consider such a y. Since $\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y)$, by eq. (12.2), $y = \ulcorner \varphi \urcorner$. So, from $\psi(y)$ we have $\psi(\ulcorner \varphi \urcorner)$.

Now suppose $\psi(\ulcorner \varphi \urcorner)$. By eq. (12.1), we have $\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, \ulcorner \varphi \urcorner) \wedge \psi(\ulcorner \varphi \urcorner)$. It follows that $\exists y\, (\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y) \wedge \psi(y))$. 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.

12.3 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 12.3. A theory **T** is ω-consistent if the following holds: if $\exists x\, \varphi(x)$ is any sentence and **T** proves $\neg\varphi(\overline{0})$, $\neg\varphi(\overline{1})$, $\neg\varphi(\overline{2})$, ... then **T** does not prove $\exists x\, \varphi(x)$.

We can now prove the following.

Theorem 12.4. *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

$$\gamma_{\mathbf{T}} \leftrightarrow \neg\mathsf{Prov}_T(\ulcorner \gamma_{\mathbf{T}} \urcorner). \tag{12.3}$$

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 $\neg\gamma_{\mathbf{T}}$.

This means that if **T** is ω-consistent, it is incomplete, since it proves neither $\gamma_{\mathbf{T}}$ nor $\neg\gamma_{\mathbf{T}}$. 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. (12.3), **Q** proves $\neg\gamma_{\mathbf{T}}$, and since **T** extends **Q**, so does **T**. We have shown that if **T** proves $\gamma_{\mathbf{T}}$, then it also proves $\neg\gamma_{\mathbf{T}}$, and hence it would be inconsistent.

For the second claim, let us show that if **T** proves $\neg\gamma_{\mathbf{T}}$, then it is ω-inconsistent. Suppose **T** proves $\neg\gamma_{\mathbf{T}}$. 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

$$\neg\mathsf{Prf}_T(\overline{0}, \ulcorner \gamma_{\mathbf{T}} \urcorner), \neg\mathsf{Prf}_T(\overline{1}, \ulcorner \gamma_{\mathbf{T}} \urcorner), \neg\mathsf{Prf}_T(\overline{2}, \ulcorner \gamma_{\mathbf{T}} \urcorner), \ldots$$

and so does **T**. On the other hand, by eq. (12.3), $\neg\gamma_{\mathbf{T}}$ is equivalent to $\exists x\, \mathsf{Prf}_T(x, \ulcorner \gamma_{\mathbf{T}} \urcorner)$. So **T** is ω-inconsistent. □

12.4 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 12.5. *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 $\neg\varphi$. 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 $\mathbf{T} \vdash \neg\varphi$ 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

$$\exists x\, (\mathsf{Prf}_T(x, y) \wedge \forall z\, (z < x \to \neg\mathsf{Ref}_T(z, y))).$$

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

$$\mathbf{Q} \vdash \rho_{\mathbf{T}} \leftrightarrow \neg\mathsf{RProv}_T(\ulcorner\rho_{\mathbf{T}}\urcorner). \tag{12.4}$$

In contrast to the proof of Theorem 12.4, here we claim that if $\mathbf{T}$ is consistent, $\mathbf{T}$ doesn't prove $\rho_{\mathbf{T}}$, and $\mathbf{T}$ also doesn't prove $\neg\rho_{\mathbf{T}}$. (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 $\neg\rho_T$, since $\mathbf{T}$ is consistent. So for each $k < n$, $\mathbf{Q} \vdash \neg\mathsf{Ref}_T(\overline{k}, \ulcorner\rho_T\urcorner)$. By Lemma 11.22(2), $\mathbf{Q} \vdash \forall z\, (z < \overline{n} \to \neg\mathsf{Ref}_T(z, \ulcorner\rho_T\urcorner))$. Thus,

$$\mathbf{Q} \vdash \exists x\, (\mathsf{Prf}_T(x, \ulcorner\rho_T\urcorner) \wedge \forall z\, (z < x \to \neg\mathsf{Ref}_T(z, \ulcorner\rho_T\urcorner))),$$

but that's just $\mathsf{RProv}_T(\ulcorner\rho_T\urcorner)$. By eq. (12.4), $\mathbf{Q} \vdash \neg\rho_T$. Since $\mathbf{T}$ extends $\mathbf{Q}$, also $\mathbf{T} \vdash \neg\rho_T$. 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 $\mathbf{T} \nvdash \neg\rho_T$. Again, suppose it did, and suppose n is the Gödel number of a derivation of $\neg\rho_T$. Then $\mathrm{Ref}_T(n, {}^{\#}\rho_T{}^{\#})$ 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 $\mathbf{Q} \vdash \rho_T \leftrightarrow \neg\mathsf{RProv}_T(\ulcorner\rho_T\urcorner)$, and since $\mathbf{T}$ extends $\mathbf{Q}$, it suffices to show that $\mathbf{Q} \vdash \neg\mathsf{RProv}_T(\ulcorner\rho_T\urcorner)$. The sentence $\neg\mathsf{RProv}_T(\ulcorner\rho_T\urcorner)$, i.e.,

$$\neg\exists x\, (\mathsf{Prf}_T(x, \ulcorner\rho_T\urcorner) \wedge \forall z\, (z < x \to \neg\mathsf{Ref}_T(z, \ulcorner\rho_T\urcorner)))$$

is logically equivalent to

$$\forall x\,(\mathsf{Prf}_T(x, \ulcorner \rho_T \urcorner) \to \exists z\,(z < x \wedge \mathsf{Ref}_T(z, \ulcorner \rho_T \urcorner)))$$

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, $\mathbf{Q} \vdash \neg \mathsf{Prf}_T(\overline{k}, \ulcorner \rho_T \urcorner)$. Thus, for every k it follows that $x \neq \overline{k}$. In particular, we have (a) that $x \neq \overline{n}$. We also have $\neg(x = \overline{0} \vee x = \overline{1} \vee \cdots \vee x = \overline{n-1})$ and so by Lemma 11.22(2), (b) $\neg(x < \overline{n})$. By Lemma 11.23, $\overline{n} < x$. Since $\mathbf{Q} \vdash \mathsf{Ref}_T(\overline{n}, \ulcorner \rho_T \urcorner)$, we have $\overline{n} < x \wedge \mathsf{Ref}_T(\overline{n}, \ulcorner \rho_T \urcorner)$, and from that $\exists z\,(z < x \wedge \mathsf{Ref}_T(z, \ulcorner \rho_T \urcorner))$. Since x was arbitrary we get

$$\forall x\,(\mathsf{Prf}_T(x, \ulcorner \rho_T \urcorner) \to \exists z\,(z < x \wedge \mathsf{Ref}_T(z, \ulcorner \rho_T \urcorner)))$$

as required. □

12.5 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.

12.6 The Provability Conditions for PA

Peano arithmetic, or **PA**, is the theory extending **Q** with induction axioms for all formulas. In other words, one adds to **Q** axioms of the form

$$(\varphi(0) \wedge \forall x\,(\varphi(x) \to \varphi(x'))) \to \forall x\,\varphi(x)$$

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 $\mathsf{Prf}_{\mathbf{PA}}(x, y)$ 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 formulas φ and ψ,

$$\mathbf{PA} \vdash \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \to \psi \urcorner) \to (\mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner) \to \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \psi \urcorner))$$

P3. For every formula φ,

$$\mathbf{PA} \vdash \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner) \to \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \mathsf{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner) \urcorner).$$

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.)

12.7 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 $\neg \mathsf{Prov}_{\mathbf{PA}}(\ulcorner 0 = 1 \urcorner)$, and then the following theorem does the job:

Theorem 12.6. *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 12.4. 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

$$\mathsf{Con}_{\mathbf{PA}} \to \neg\mathsf{Prov}_{\mathbf{PA}}(\ulcorner\gamma_{\mathbf{PA}}\urcorner).$$

Now suppose **PA** proves $\mathsf{Con}_{\mathbf{PA}}$. Then it proves $\neg\mathsf{Prov}_{\mathbf{PA}}(\ulcorner\gamma_{\mathbf{PA}}\urcorner)$. 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 $\mathsf{Con}_{\mathbf{PA}} \to \gamma_{\mathbf{PA}}$. 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.)

$$!G \leftrightarrow \neg \mathsf{Prov}(\ulcorner \gamma \urcorner) \tag{12.5}$$

γ is a Gödel sentence

$$!G \rightarrow \neg \mathsf{Prov}(\ulcorner \gamma \urcorner) \tag{12.6}$$

from eq. (12.5)

$$!G \rightarrow (\mathsf{Prov}(\ulcorner \gamma \urcorner) \rightarrow \bot) \tag{12.7}$$

from eq. (12.6) by logic

$$\mathsf{Prov}(\ulcorner \gamma \rightarrow (\mathsf{Prov}(\ulcorner \gamma \urcorner) \rightarrow \bot) \urcorner) \tag{12.8}$$

by from eq. (12.7) by condition P1

$$\mathsf{Prov}(\ulcorner \gamma \urcorner) \rightarrow \mathsf{Prov}(\ulcorner (\mathsf{Prov}(\ulcorner \gamma \urcorner) \rightarrow \bot) \urcorner) \tag{12.9}$$

from eq. (12.8) by condition P2

$$\mathsf{Prov}(\ulcorner \gamma \urcorner) \rightarrow (\mathsf{Prov}(\ulcorner \mathsf{Prov}(\ulcorner \gamma \urcorner) \urcorner) \rightarrow \mathsf{Prov}(\ulcorner \bot \urcorner)) \tag{12.10}$$

from eq. (12.9) by condition P2 and logic

$$\mathsf{Prov}(\ulcorner \gamma \urcorner) \rightarrow \mathsf{Prov}(\ulcorner \mathsf{Prov}(\ulcorner \gamma \urcorner) \urcorner) \tag{12.11}$$

by P3

$$\mathsf{Prov}(\ulcorner \gamma \urcorner) \rightarrow \mathsf{Prov}(\ulcorner \bot \urcorner) \tag{12.12}$$

from eq. (12.10) and eq. (12.11) by logic

$$\mathsf{Con} \rightarrow \neg \mathsf{Prov}(\ulcorner \gamma \urcorner) \tag{12.13}$$

contraposition of eq. (12.12) and $\mathsf{Con} \equiv \neg \mathsf{Prov}(\ulcorner \bot \urcorner)$

$\mathsf{Con} \rightarrow \gamma$

from eq. (12.5) and eq. (12.13) by logic

The use of logic in the above just elementary facts from propositional logic, e.g., eq. (12.7) uses $\vdash \neg\varphi \leftrightarrow (\varphi \rightarrow \bot)$ and eq. (12.12) uses $\varphi \rightarrow (\psi \rightarrow \chi), \varphi \rightarrow \psi \vdash \varphi \rightarrow \chi$. The use of condition P2 in eq. (12.9) and eq. (12.10) relies on instances of P2, $\mathsf{Prov}(\ulcorner \varphi \rightarrow \psi \urcorner) \rightarrow (\mathsf{Prov}(\ulcorner \varphi \urcorner) \rightarrow \mathsf{Prov}(\ulcorner \psi \urcorner))$. In the first one, $\varphi \equiv \gamma$ and $\psi \equiv \mathsf{Prov}(\ulcorner \gamma \urcorner) \rightarrow \bot$; 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 12.7. *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**.

12.8 Löb's Theorem

The Gödel sentence for a theory $\mathbf{T}$ is a fixed point of $\neg\mathsf{Prov}_T(x)$, i.e., a sentence γ such that

$$\mathbf{T} \vdash \neg\mathsf{Prov}_T(\ulcorner\gamma\urcorner) \leftrightarrow \gamma.$$

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 $\mathbf{T} \vdash \neg\mathsf{Prov}_T(\ulcorner\gamma\urcorner) \leftrightarrow \gamma$ gives $\mathbf{T} \vdash \neg\mathsf{Prov}_T(\ulcorner\gamma\urcorner)$, 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

$$\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\delta\urcorner) \leftrightarrow \delta.$$

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, $\mathsf{Prov}_T(\ulcorner\delta\urcorner) \to \delta$, is an instance of a general schema called a *reflection principle*: $\mathsf{Prov}_T(\ulcorner\varphi\urcorner) \to \varphi$. 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 12.8. *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 12.7. If* $\mathbf{T}$ *proves* $\mathsf{Prov}_T(\ulcorner\varphi\urcorner) \to \varphi$*, then in fact* $\mathbf{T}$ *proves* φ*.*

Put differently, if $\mathbf{T} \nvdash \varphi$, then $\mathbf{T} \nvdash \mathsf{Prov}_T(\ulcorner\varphi\urcorner) \to \varphi$. 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 $\mathsf{Prov}_T(y) \to \varphi$. 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 $\mathsf{Prov}_T(\ulcorner \varphi \urcorner) \to \varphi$. Let $\psi(y)$ be the formula $\mathsf{Prov}_T(y) \to \varphi$, and use the fixed-point lemma to find a sentence θ such that $\mathbf{T}$ proves $\theta \leftrightarrow \psi(\ulcorner \theta \urcorner)$. Then each of the following is provable

in **T**:

$$!D \leftrightarrow (\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \varphi) \tag{12.14}$$

θ is a fixed point of $\psi(y)$

$$!D \to (\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \varphi) \tag{12.15}$$

from eq. (12.14)

$$\mathsf{Prov}_T(\ulcorner\theta \to (\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \varphi)\urcorner) \tag{12.16}$$

from eq. (12.15) by condition P1

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \mathsf{Prov}_T(\ulcorner\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \varphi\urcorner) \tag{12.17}$$

from eq. (12.16) using condition P2

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to (\mathsf{Prov}_T(\ulcorner\mathsf{Prov}_T(\ulcorner\theta\urcorner)\urcorner) \to \mathsf{Prov}_T(\ulcorner\varphi\urcorner)) \tag{12.18}$$

from eq. (12.17) using P2 again

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \mathsf{Prov}_T(\ulcorner\mathsf{Prov}_T(\ulcorner\theta\urcorner)\urcorner) \tag{12.19}$$

by provability condition P3

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \mathsf{Prov}_T(\ulcorner\varphi\urcorner) \tag{12.20}$$

from eq. (12.18) and eq. (12.19)

$$\mathsf{Prov}_T(\ulcorner\varphi\urcorner) \to \varphi \tag{12.21}$$

by assumption of the theorem

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \varphi \tag{12.22}$$

from eq. (12.20) and eq. (12.21)

$$(\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \varphi) \to \theta \tag{12.23}$$

from eq. (12.14)

$$!D \tag{12.24}$$

from eq. (12.22) and eq. (12.23)

$$\mathsf{Prov}_T(\ulcorner\theta\urcorner) \tag{12.25}$$

from eq. (12.24) by condition P1

$!A$ from eq. (12.21) and eq. (12.25)

□

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 $\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\bot\urcorner) \to \bot$, then $\mathbf{T} \vdash \bot$. If **T** is consistent, $\mathbf{T} \nvdash \bot$. So, $\mathbf{T} \nvdash \mathsf{Prov}_T(\ulcorner\bot\urcorner) \to \bot$, 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

$$\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\delta\urcorner) \leftrightarrow \delta$$

in particular

$$\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \delta$$

and so by Löb's theorem, $\mathbf{T} \vdash \delta$.

12.9 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 12.9. 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 $\mathbf{TA} = \{\varphi : \mathfrak{N} \models \varphi\}$ 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 12.10. *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 12.11. *The halting relation is definable in* $\mathfrak{N}$.

Proof. Let H be the halting relation, i.e.,

$$H = \{\langle e, x\rangle : \exists s\, T(e, x, s)\}.$$

Let θ_T define T in $\mathfrak{N}$. Then

$$H = \{\langle e, x\rangle : \mathfrak{N} \models \exists s\, \theta_T(\overline{e}, \overline{x}, s)\},$$

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 $\{^{\#}\varphi^{\#} : \mathfrak{N} \models \varphi\}$ definable in arithmetic? Tarski's theorem answers this in the negative.

Theorem 12.12. *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 $\varphi \leftrightarrow \neg\theta(\ulcorner\varphi\urcorner)$, and hence $\mathfrak{N} \models \varphi \leftrightarrow \neg\theta(\ulcorner\varphi\urcorner)$. But then $\mathfrak{N} \models \varphi$ if and only if $\mathfrak{N} \models \neg\theta(\ulcorner\varphi\urcorner)$, 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.

Problems

Problems for Chapter 1

Problem 1.1. Show that there is only one empty set, i.e., show that if X and Y are sets without members, then $X = Y$.

Problem 1.2. List all subsets of $\{a, b, c, d\}$.

Problem 1.3. Show that if X has n elements, then $\wp(X)$ has 2^n elements.

Problem 1.4. Prove rigorously that if $X \subseteq Y$, then $X \cup Y = Y$.

Problem 1.5. Prove rigorously that if $X \subseteq Y$, then $X \cap Y = X$.

Problem 1.6. Prove in detail that $X \cup (X \cap Y) = X$. Then give a shortened, compressed proof. (Hint: for the $X \cup (X \cap Y) \subseteq X$ direction you will need proof by cases, aka $\vee$Elim.)

Problem 1.7. List all elements of $\{1, 2, 3\}^3$.

Problem 1.8. Show that if X has n elements, then X^k has n^k elements.

Problems for Chapter 2

Problem 2.1. List the elements of the relation $\subseteq$ on the set $\wp(\{a, b, c\})$.

Problem 2.2. Give examples of relations that are (a) reflexive and symmetric but not transitive, (b) reflexive and anti-symmetric, (c) anti-symmetric, transitive, but not reflexive, and (d) reflexive, symmetric, and transitive. Do not use relations on numbers or sets.

Problem 2.3. Complete the proof of Proposition 2.19, i.e., prove that if R is a partial order on X, then $R^- = R \setminus \mathrm{Id}_X$ is a strict order.

Problem 2.4. Consider the less-than-or-equal-to relation $\leq$ on the set $\{1, 2, 3, 4\}$ as a graph and draw the corresponding diagram.

Problem 2.5. Show that the transitive closure of R is in fact transitive.

Problems for Chapter 3

Problem 3.1. Show that if f is bijective, an inverse g of f exists, i.e., define such a g, show that it is a function, and show that it is an inverse of f, i.e., $f(g(y)) = y$ and $g(f(x)) = x$ for all $x \in X$ and $y \in Y$.

Problem 3.2. Show that if $f\colon X \to Y$ has an inverse g, then f is bijective.

Problem 3.3. Show that if $g\colon Y \to X$ and $g'\colon Y \to X$ are inverses of $f\colon X \to Y$, then $g = g'$, i.e., for all $y \in Y$, $g(y) = g'(y)$.

Problem 3.4. Show that if $f\colon X \to Y$ and $g\colon Y \to Z$ are both injective, then $g \circ f\colon X \to Z$ is injective.

Problem 3.5. Show that if $f\colon X \to Y$ and $g\colon Y \to Z$ are both surjective, then $g \circ f\colon X \to Z$ is surjective.

Problem 3.6. Given $f\colon X \pfun Y$, define the partial function $g\colon Y \pfun X$ by: for any $y \in Y$, if there is a unique $x \in X$ such that $f(x) = y$, then $g(y) = x$; otherwise $g(y) \uparrow$. Show that if f is injective, then $g(f(x)) = x$ for all $x \in \mathrm{dom}(f)$, and $f(g(y)) = y$ for all $y \in \mathrm{ran}(f)$.

Problem 3.7. Suppose $f\colon X \to Y$ and $g\colon Y \to Z$. Show that the graph of $(g \circ f)$ is $R_f \mid R_g$.

Problems for Chapter 4

Problem 4.1. According to Definition 4.4, a set X is enumerable iff $X = \emptyset$ or there is a surjective $f\colon \mathbb{Z}^+ \to X$. It is also possible to define "enumerable set" precisely by: a set is enumerable iff there is an injective function $g\colon X \to \mathbb{Z}^+$. Show that the definitions are equivalent, i.e., show that there is an injective function $g\colon X \to \mathbb{Z}^+$ iff either $X = \emptyset$ or there is a surjective $f\colon \mathbb{Z}^+ \to X$.

Problem 4.2. Define an enumeration of the positive squares $4, 9, 16, \ldots$

Problem 4.3. Show that if X and Y are enumerable, so is $X \cup Y$.

Problem 4.4. Show by induction on n that if $X_1, X_2, \ldots, X_n$ are all enumerable, so is $X_1 \cup \cdots \cup X_n$.

Problem 4.5. Give an enumeration of the set of all positive rational numbers. (A positive rational number is one that can be written as a fraction n/m with $n, m \in \mathbb{Z}^+$).

Problem 4.6. Show that $\mathbb{Q}$ is enumerable. (A rational number is one that can be written as a fraction z/m with $z \in \mathbb{Z}$, $m \in \mathbb{Z}^+$).

Problem 4.7. Define an enumeration of $\mathbb{B}^*$.

Problem 4.8. Recall from your introductory logic course that each possible truth table expresses a truth function. In other words, the truth functions are all functions from $\mathbb{B}^k \to \mathbb{B}$ for some k. Prove that the set of all truth functions is enumerable.

Problem 4.9. Show that the set of all finite subsets of an arbitrary infinite enumerable set is enumerable.

Problem 4.10. A set of positive integers is said to be *cofinite* iff it is the complement of a finite set of positive integers. Let I be the set that contains all the finite and cofinite sets of positive integers. Show that I is enumerable.

Problem 4.11. Show that the enumerable union of enumerable sets is enumerable. That is, whenever $X_1, X_2, \ldots$ are sets, and each X_i is enumerable, then the union $\bigcup_{i=1}^{\infty} X_i$ of all of them is also enumerable.

Problem 4.12. Show that $\wp(\mathbb{N})$ is non-enumerable by a diagonal argument.

Problem 4.13. Show that the set of functions $f\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ is non-enumerable by an explicit diagonal argument. That is, show that if $f_1, f_2, \ldots$, is a list of functions and each $f_i\colon \mathbb{Z}^+ \to \mathbb{Z}^+$, then there is some $\overline{f}\colon \mathbb{Z}^+ \to \mathbb{Z}^+$ not on this list.

Problem 4.14. Show that if there is an injective function $g\colon Y \to X$, and Y is non-enumerable, then so is X. Do this by showing how you can use g to turn an enumeration of X into one of Y.

Problem 4.15. Show that the set of all *sets of* pairs of positive integers is non-enumerable by a reduction argument.

Problem 4.16. Show that $\mathbb{N}^\omega$, the set of infinite sequences of natural numbers, is non-enumerable by a reduction argument.

Problem 4.17. Let P be the set of functions from the set of positive integers to the set $\{0\}$, and let Q be the set of *partial* functions from the set of positive integers to the set $\{0\}$. Show that P is enumerable and Q is not. (Hint: reduce the problem of enumerating $\mathbb{B}^\omega$ to enumerating Q).

Problem 4.18. Let S be the set of all surjective functions from the set of positive integers to the set $\{0,1\}$, i.e., S consists of all surjective $f\colon \mathbb{Z}^+ \to \mathbb{B}$. Show that S is non-enumerable.

Problem 4.19. Show that the set $\mathbb{R}$ of all real numbers is non-enumerable.

Problem 4.20. Show that if X is equinumerous with U and and Y is equinumerous with V, and the intersections $X \cap Y$ and $U \cap V$ are empty, then the unions $X \cup Y$ and $U \cup V$ are equinumerous.

Problem 4.21. Show that if X is infinite and enumerable, then it is equinumerous with the positive integers $\mathbb{Z}^+$.

Problem 4.22. Show that there cannot be an injective function $g\colon \wp(X) \to X$, for any set X. Hint: Suppose $g\colon \wp(X) \to X$ is injective. Then for each $x \in X$ there is at most one $Y \subseteq X$ such that $g(Y) = x$. Define a set $\overline{Y}$ such that for every $x \in X$, $g(\overline{Y}) \neq x$.

Problems for Chapter 5

Problem 5.1. Prove Lemma 5.10.

Problem 5.2. Prove Proposition 5.11 (Hint: Formulate and prove a version of Lemma 5.10 for terms.)

Problem 5.3. Give an inductive definition of the bound variable occurrences along the lines of Definition 5.17.

Problem 5.4. Is $\mathfrak{N}$, the standard model of arithmetic, covered? Explain.

Problem 5.5. Let $\mathcal{L} = \{c, f, A\}$ with one constant symbol, one one-place function symbol and one two-place predicate symbol, and let the structure $\mathfrak{M}$ be given by

1. $|\mathfrak{M}| = \{1, 2, 3\}$
2. $c^{\mathfrak{M}} = 3$
3. $f^{\mathfrak{M}}(1) = 2, f^{\mathfrak{M}}(2) = 3, f^{\mathfrak{M}}(3) = 2$
4. $A^{\mathfrak{M}} = \{\langle 1, 2\rangle, \langle 2, 3\rangle, \langle 3, 3\rangle\}$

(a) Let $s(v) = 1$ for all variables v. Find out whether

$$\mathfrak{M}, s \models \exists x\,(A(f(z), c) \to \forall y\,(A(y, x) \vee A(f(y), x)))$$

Explain why or why not.

(b) Give a different structure and variable assignment in which the formula is not satisfied.

Problem 5.6. Complete the proof of Proposition 5.36.

Problem 5.7. Show that if φ is a sentence, $\mathfrak{M} \models \varphi$ iff *there is* a variable assignment s so that $\mathfrak{M}, s \models \varphi$.

Problem 5.8. Prove Proposition 5.38.

Problem 5.9. Suppose $\mathcal{L}$ is a language without function symbols. Given a structure $\mathfrak{M}$ and $a \in |\mathfrak{M}|$, define $\mathfrak{M}[a/c]$ to be the structure that is just like $\mathfrak{M}$, except that $c^{\mathfrak{M}[a/c]} = a$. Define $\mathfrak{M} \mid\!\models \varphi$ for sentences φ by:

1. $\varphi \equiv \bot$: not $\mathfrak{M} \mid\!\models \varphi$.
2. $\varphi \equiv R(d_1, \ldots, d_n)$: $\mathfrak{M} \mid\!\models \varphi$ iff $\langle d_1^{\mathfrak{M}}, \ldots, d_n^{\mathfrak{M}} \rangle \in R^{\mathfrak{M}}$.
3. $\varphi \equiv d_1 = d_2$: $\mathfrak{M} \mid\!\models \varphi$ iff $d_1^{\mathfrak{M}} = d_2^{\mathfrak{M}}$.
4. $\varphi \equiv \neg\psi$: $\mathfrak{M} \mid\!\models \varphi$ iff not $\mathfrak{M} \mid\!\models \psi$.
5. $\varphi \equiv (\psi \wedge \chi)$: $\mathfrak{M} \mid\!\models \varphi$ iff $\mathfrak{M} \mid\!\models \psi$ and $\mathfrak{M} \mid\!\models \chi$.
6. $\varphi \equiv (\psi \vee \chi)$: $\mathfrak{M} \mid\!\models \varphi$ iff $\mathfrak{M} \mid\!\models \psi$ or $\mathfrak{M} \mid\!\models \chi$ (or both).
7. $\varphi \equiv (\psi \to \chi)$: $\mathfrak{M} \mid\!\models \varphi$ iff not $\mathfrak{M} \mid\!\models \psi$ or $\mathfrak{M} \mid\!\models \chi$ (or both).
8. $\varphi \equiv \forall x\, \psi$: $\mathfrak{M} \mid\!\models \varphi$ iff for all $a \in |\mathfrak{M}|$, $\mathfrak{M}[a/c] \mid\!\models \psi[c/x]$, if c does not occur in ψ.
9. $\varphi \equiv \exists x\, \psi$: $\mathfrak{M} \mid\!\models \varphi$ iff there is an $a \in |\mathfrak{M}|$ such that $\mathfrak{M}[a/c] \mid\!\models \psi[c/x]$, if c does not occur in ψ.

Let $x_1, \ldots, x_n$ be all free variables in φ, $c_1, \ldots, c_n$ constant symbols not in φ, $a_1, \ldots, a_n \in |\mathfrak{M}|$, and $s(x_i) = a_i$.

Show that $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}[a_1/c_1, \ldots, a_n/c_n] \mid\!\models \varphi[c_1/x_1] \ldots [c_n/x_n]$.

Problem 5.10. Suppose that f is a function symbol not in $\varphi(x, y)$. Show that there is a $\mathfrak{M}$ such that $\mathfrak{M} \models \forall x\, \exists y\, \varphi(x, y)$ iff there is a $\mathfrak{M}'$ such that $\mathfrak{M}' \models \forall x\, \varphi(x, f(x))$.

Problem 5.11. Prove Proposition 5.41

Problem 5.12.

1. Show that $\Gamma \vDash \bot$ iff Γ is unsatisfiable.
2. Show that $\Gamma \cup \{\varphi\} \vDash \bot$ iff $\Gamma \vDash \neg\varphi$.
3. Suppose c does not occur in φ or Γ. Show that $\Gamma \vDash \forall x\, \varphi$ iff $\Gamma \vDash \varphi[c/x]$.

Problems for Chapter 6

Problem 6.1. Find formulas in $\mathcal{L}_A$ which define the following relations:

1. n is between i and j;
2. n evenly divides m (i.e., m is a multiple of n);
3. n is a prime number (i.e., no number other than 1 and n evenly divides n).

Problem 6.2. Suppose the formula $\varphi(v_1, v_2)$ expresses the relation $R \subseteq |\mathfrak{M}|^2$ in a structure $\mathfrak{M}$. Find formulas that express the following relations:

1. the inverse R^{-1} of R;

2. the relative product $R \mid R$;

Can you find a way to express R^+, the transitive closure of R?

Problem 6.3. Let $\mathcal{L}$ be the language containing a 2-place predicate symbol $<$ only (no other constant symbols, function symbols or predicate symbols—except of course $=$). Let $\mathfrak{N}$ be the structure such that $|\mathfrak{N}| = \mathbb{N}$, and $<^{\mathfrak{N}} = \{\langle n, m\rangle : n < m\}$. Prove the following:

1. $\{0\}$ is definable in $\mathfrak{N}$;
2. $\{1\}$ is definable in $\mathfrak{N}$;
3. $\{2\}$ is definable in $\mathfrak{N}$;
4. for each $n \in \mathbb{N}$, the set $\{n\}$ is definable in $\mathfrak{N}$;
5. every finite subset of $|\mathfrak{N}|$ is definable in $\mathfrak{N}$;
6. every co-finite subset of $|\mathfrak{N}|$ is definable in $\mathfrak{N}$ (where $X \subseteq \mathbb{N}$ is co-finite iff $\mathbb{N} \setminus X$ is finite).

Problem 6.4. Show that the comprehension principle is inconsistent by giving a derivation that shows

$$\exists y \forall x\, (x \in y \leftrightarrow x \notin x) \vdash \bot.$$

It may help to first show $(A \to \neg A) \wedge (\neg A \to A) \vdash \bot$.

Problems for Chapter 7

Problem 7.1. Give derivations of the following sequents:

1. $\Rightarrow \neg(\varphi \to \psi) \to (\varphi \wedge \neg\psi)$
2. $\forall x\, (\varphi(x) \to \psi) \Rightarrow (\exists y\, \varphi(y) \to \psi)$

Problem 7.2. Prove Proposition 7.13

Problem 7.3. Prove Proposition 7.14

Problem 7.4. Prove Proposition 7.20.

Problem 7.5. Prove Proposition 7.21.

Problem 7.6. Prove Proposition 7.22.

Problem 7.7. Prove Proposition 7.23.

Problem 7.8. Prove Proposition 7.24.

Problem 7.9. Prove Proposition 7.25.

Problem 7.10. Complete the proof of Theorem 7.29.

Problem 7.11. Give derivations of the following sequents:

1. $\Rightarrow \forall x \forall y ((x = y \wedge \varphi(x)) \rightarrow \varphi(y))$
2. $\exists x \varphi(x) \wedge \forall y \forall z ((\varphi(y) \wedge \varphi(z)) \rightarrow y = z) \Rightarrow$
   $\exists x (\varphi(x) \wedge \forall y (\varphi(y) \rightarrow y = x))$

Problems for Chapter 8

Problem 8.1. Complete the proof of Proposition 8.2.

Problem 8.2. Complete the proof of Lemma 8.9.

Problem 8.3. Complete the proof of Proposition 8.11.

Problem 8.4. Use Corollary 8.17 to prove Theorem 8.16, thus showing that the two formulations of the completeness theorem are equivalent.

Problem 8.5. In order for a derivation system to be complete, its rules must be strong enough to prove every unsatisfiable set inconsistent. Which of the rules of **LK** were necessary to prove completeness? Are any of these rules not used anywhere in the proof? In order to answer these questions, make a list or diagram that shows which of the rules of **LK** were used in which results that lead up to the proof of Theorem 8.16. Be sure to note any tacit uses of rules in these proofs.

Problem 8.6. Prove (1) of Theorem 8.19.

Problem 8.7. In the standard model of arithmetic $\mathfrak{N}$, there is no element $k \in |\mathfrak{N}|$ which satisfies every formula $\overline{n} < x$ (where $\overline{n}$ is $0'^{\cdots'}$ with n $'$'s). Use the compactness theorem to show that the set of sentences in the language of arithmetic which are true in the standard model of arithmetic $\mathfrak{N}$ are also true in a structure $\mathfrak{N}'$ that contains an element which *does* satisfy every formula $\overline{n} < x$.

Problem 8.8. Let Γ be the set of all sentences φ in the language of arithmetic such that $\mathfrak{N} \models \varphi$, i.e., Γ contains all sentences true in the "standard model." Show that there is a model $\mathfrak{M}$ of Γ which is not covered, i.e., some $a \in |\mathfrak{M}|$ is such that $a \neq \mathrm{Val}^{\mathfrak{M}}(t)$ for all closed terms t.

Problems for Chapter 9

Problem 9.1. Multiplication satisfies the recursive equations

$$0 \cdot y = y$$
$$(x+1) \cdot y = (x \cdot y) + x$$

Give the explicit precise definition of the function $\text{mult}(x, y) = x \cdot y$, assuming that $\text{add}(x, y) = x + y$ is already defined. Give the complete notation for mult.

Problem 9.2. Show that

$$f(x, y) = \underbrace{2^{2^{\cdot^{\cdot^{2^x}}}}}_{y \text{ 2's}}$$

is primitive recursive.

Problem 9.3. Show that $d(x, y) = \lfloor x/y \rfloor$ (i.e., division, where you disregard everything after the decimal point) is primitive recursive. When $y = 0$, we stipulate $d(x, y) = 0$. Give an explicit definifion of d using primitive recursion and composition. You will have detour through an axiliary function—you cannot use recursion on the arguments x or y themselves.

Problem 9.4. Suppose $R(x, \vec{z})$ is primitive recursive. Define 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 $y + 1$ otherwise, by primitive recursion from χ_R.

Problem 9.5. Define integer division $d(x, y)$ using bounded minimization.

Problem 9.6. Show that there is a primitive recursive function $\text{sconcat}(s)$ with the property that

$$\text{sconcat}(\langle s_0, \ldots, s_k \rangle) = s_0 \frown \ldots\ldots \frown s_k.$$

Problems for Chapter 10

Problem 10.1. Show that the function $\text{flatten}(z)$, which turns the sequence $\langle {}^{\#}t_1{}^{\#}, \ldots, {}^{\#}t_n{}^{\#} \rangle$ into ${}^{\#}t_1, \ldots, t_n{}^{\#}$, is primitive recursive.

Problem 10.2. Give a detailed proof of Proposition 10.8 along the lines of the first proof of Proposition 10.5

Problem 10.3. Give a detailed proof of Proposition 10.8 along the lines of the alternate proof of Proposition 10.5

Problem 10.4. Prove Proposition 10.9. You may make use of the fact that any substring of a formula which is a formula is a sub-formula of it.

Problem 10.5. Prove Proposition 10.12

Problem 10.6. Define the following relations as in Proposition 10.14:

1. $\text{FollowsBy}_{\wedge\text{right}}(s, s', s'')$,
2. $\text{FollowsBy}_{=}(s, s')$,
3. $\text{FollowsBy}_{\forall\text{right}}(s, s')$.

Problems for Chapter 11

Problem 11.1. Prove that $y = \mathsf{o}$, $y = x'$, and $y = x_i$ represent zero, succ, and P^n_i, respectively.

Problem 11.2. Prove Lemma 11.17.

Problem 11.3. Use Lemma 11.17 to prove Proposition 11.16.

Problem 11.4. Using the proofs of Proposition 11.19 and Proposition 11.19 as a guide, carry out the proof of Proposition 11.20 in detail.

Problem 11.5. Show that if R is representable in $\mathbf{Q}$, so is χ_R.

Problems for Chapter 12

Problem 12.1. Show that **PA** proves $\gamma_{\mathbf{PA}} \to \mathsf{Con}_{\mathbf{PA}}$.

Problem 12.2. Let **T** be a computably axiomatized theory, and let Prov_T be a provability predicate for **T**. Consider the following four statements:

1. If $T \vdash \varphi$, then $T \vdash \mathsf{Prov}_T(\ulcorner\varphi\urcorner)$.
2. $T \vdash \varphi \to \mathsf{Prov}_T(\ulcorner\varphi\urcorner)$.
3. If $T \vdash \mathsf{Prov}_T(\ulcorner\varphi\urcorner)$, then $T \vdash \varphi$.
4. $T \vdash \mathsf{Prov}_T(\ulcorner\varphi\urcorner) \to \varphi$

Under what conditions are each of these statements true?

Problem 12.3. Show that $Q(n) \Leftrightarrow n \in \{{}^{\#}\varphi^{\#} : \mathbf{Q} \vdash \varphi\}$ is definable in arithmetic.