An Open Mathematical Introduction to Logic
Example OLP Text à la Enderton

Open Logic Project

open-logic-enderton rev: b5b5460 (2023-05-22) by OLP / CC–BY

Contents

Part I

Propositional Logic

Chapter 1

Syntax and Semantics

§1.0 Introduction

Propositional logic deals with wffs that are built from propositional variables using the propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, and $\leftrightarrow$. Intuitively, a propositional variable p stands for a sentence or proposition that is true or false. Whenever the "truth value" of the propositional variable in a wff is determined, so is the truth value of any wffs formed from them using propositional connectives. We say that propositional logic is *truth functional*, because its semantics is given by functions of truth values. In particular, in propositional logic we leave out of consideration any further determination of truth and falsity, e.g., whether something is necessarily true rather than just contingently true, or whether something is known to be true, or whether something is true now rather than was true or will be true. We only consider two truth values true ($\mathbb{T}$) and false ($\mathbb{F}$), and so exclude from discussion the possibility that a statement may be neither true nor false, or only half true. We also concentrate only on connectives where the truth value of a wff built from them is completely determined by the truth values of its parts (and not, say, on its meaning). In particular, whether the truth value of conditionals in English is truth functional in this sense is contentious. The material conditional $\rightarrow$ is; other logics deal with conditionals that are not truth functional.

In order to develop the theory and metatheory of truth-functional propositional logic, we must first define the syntax and semantics of its expressions. We will describe one way of constructing wffs from propositional variables using the connectives. Alternative definitions are possible. Other systems will choose different symbols, will select different sets of connectives as primitive, and 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 wffs *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 for propositional logic is that of satisfaction in a valuation. A valuation $\mathfrak{v}$ assigns truth values $\mathbb{T}$, $\mathbb{F}$ to the propositional variables. Any valuation determines a truth value $\overline{\mathfrak{v}}(\alpha)$ for any wff α. A wff is satisfied in a valuation $\mathfrak{v}$ iff $\overline{\mathfrak{v}}(\alpha) = \mathbb{T}$—we write this as $\mathfrak{v} \vDash \alpha$. This relation can also be defined by induction on the structure of α, using the truth functions for the logical connectives to define, say, satisfaction of $\alpha \wedge \beta$ in terms of satisfaction (or not) of α and β.

On the basis of the satisfaction relation $\mathfrak{v} \vDash \alpha$ for sentences we can then define the basic semantic notions of tautology, entailment, and satisfiability. A wff is a tautology, $\vDash \alpha$, if every valuation satisfies it, i.e., $\overline{\mathfrak{v}}(\alpha) = \mathbb{T}$ for any $\mathfrak{v}$. It is entailed by a set of wffs, $\Gamma \vDash \alpha$, if every valuation that satisfies all the wffs in Γ also satisfies α. And a set of wffs is satisfiable if some valuation satisfies all wffs in it at the same time. Because wffs are inductively defined, and satisfaction is in turn defined by induction on the structure of wffs, we can use induction to prove properties of our semantics and to relate the semantic notions defined.

§1.1 Propositional Wffs

Wffs of propositional logic are built up from *propositional variables* using *logical connectives*.

1. A denumerable set At_0 of propositional variables p_0, p_1, ...
2. The logical connectives: $\neg$ (negation), $\rightarrow$ (conditional)
3. Punctuation marks: (,), and the comma.

We denote this language of propositional logic by $\mathcal{L}_0$.

In addition to the primitive connectives introduced above, we also use the following *defined* symbols: $\wedge$ (conjunction), $\vee$ (disjunction), $\leftrightarrow$ (biconditional), $\bot$ (falsity), $\top$ (truth)

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."

Definition 11A (Formula). *The set* $\mathrm{Frm}(\mathcal{L}_0)$ *of* wffs *of propositional logic is defined inductively as follows:*

1. *Every propositional variable* p_i *is an atomic wff.*
2. *If* α *is a wff, then* $\neg\alpha$ *is a wff.*
3. *If* α *and* β *are wffs, then* $(\alpha \to \beta)$ *is a wff.*

The definition of wffs is an *inductive definition*. Essentially, we construct the set of wffs 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 p_i. "Atomic wff" thus means any wff of this form.

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

Definition 11B. *Formulas constructed using the defined operators are to be understood as follows:*

1. $\top$ *abbreviates* $(\alpha \vee \neg\alpha)$ *for some fixed atomic wff* α*.*
2. $\bot$ *abbreviates* $(\alpha \wedge \neg\alpha)$ *for some fixed atomic wff* α*.*
3. $\alpha \vee \beta$ *abbreviates* $\neg\alpha \to \beta$*.*
4. $\alpha \wedge \beta$ *abbreviates* $\neg(\alpha \to \neg\beta)$*.*
5. $\alpha \leftrightarrow \beta$ *abbreviates* $(\alpha \to \beta) \wedge (\beta \to \alpha)$*.*

Definition 11C (Syntactic identity). *The symbol* $\equiv$ *expresses syntactic identity between strings of symbols, i.e.,* $\alpha \equiv \beta$ *iff* α *and* β *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., $\alpha \equiv (\beta \vee \gamma)$ means: the string of symbols α is the same string as the one obtained by concatenating an opening parenthesis, the string β, the $\vee$ symbol, the string γ, and a closing parenthesis, in this order. If this is the case, then we know that the first symbol of α is an opening parenthesis, α contains β as a substring (starting at the second symbol), that substring is followed by $\vee$, etc.

§1.2 Preliminaries

Theorem 12A (*Principle of induction on wffs*). *If some property* P *holds for all the atomic wffs and is such that*

1. *it holds for* $\neg\alpha$ *whenever it holds for* α*;*

2. *it holds for* $(\alpha \to \beta)$ *whenever it holds for* α *and* β*;*

then P *holds for all wffs.*

Proof. Let S be the collection of all wffs with property P. Clearly $S \subseteq \mathrm{Frm}(\mathcal{L}_0)$. S satisfies all the conditions of Definition 11A: it contains all atomic wffs and is closed under the logical operators. $\mathrm{Frm}(\mathcal{L}_0)$ is the smallest such class, so $\mathrm{Frm}(\mathcal{L}_0) \subseteq S$. So $\mathrm{Frm}(\mathcal{L}_0) = S$, and every formula has property P. □

Proposition 12B. *Any wff in* $\mathrm{Frm}(\mathcal{L}_0)$ *is* balanced*, in that it has as many left parentheses as right ones.*

Proposition 12C. *No proper initial segment of a wff is a wff.*

Proposition 12D (Unique Readability). *Any wff* α *in* $\mathrm{Frm}(\mathcal{L}_0)$ *has exactly one parsing as one of the following*

1. p_n *for some* $p_n \in \mathrm{At}_0$.
2. $\neg\beta$ *for some wff* β.
3. $(\beta \to \gamma)$ *for some wffs* β *and* γ.

Moreover, this parsing is unique.

Proof. By induction on α. For instance, suppose that α has two distinct readings as $(\beta \to \gamma)$ and $(\beta' \to \gamma')$. Then β and β' must be the same (or else one would be a proper initial segment of the other); so if the two readings of α are distinct it must be because γ and γ' are distinct readings of the same sequence of symbols, which is impossible by the inductive hypothesis. □

Definition 12E (Uniform Substitution). *If* α *and* β *are wffs, and* p_i *is a propositional variable, then* $\alpha[\beta/p_i]$ *denotes the result of replacing each occurrence of* p_i *by an occurrence of* β *in* α*; similarly, the simultaneous substitution of* $p_1, \ldots, p_n$ *by wffs* $\beta_1, \ldots, \beta_n$ *is denoted by* $\alpha[\beta_1/p_1, \ldots, \beta_n/p_n]$.

§1.3 Formation Sequences

Defining wffs via an inductive definition, and the complementary technique of proving properties of wffs via induction, is an elegant and efficient approach. However, it can also be useful to consider a more bottom-up, step-by-step approach to the construction of wffs, which we do here using the notion of a *formation sequence*.

Definition 13A (Formation sequences for formulas). *A finite sequence* $\langle \alpha_0, \ldots, \alpha_n \rangle$ *of strings of symbols from the language* $\mathcal{L}_0$ *is a* formation sequence *for* α *if* $\alpha \equiv \alpha_n$ *and for all* $i \leq n$*, either* α_i *is an atomic formula or there exist* $j, k < i$ *such that one of the following holds:*

1. $\alpha_i \equiv \neg\alpha_j$.
2. $\alpha_i \equiv (\alpha_j \to \alpha_k)$.

Example 1.3.2.

$$\langle p_0, p_1, (p_1 \wedge p_0), \neg(p_1 \wedge p_0)\rangle$$

is a formation sequence of $\neg(p_1 \wedge p_0)$, as is

$$\langle p_0, p_1, p_0, (p_1 \wedge p_0), (p_0 \to p_1), \neg(p_1 \wedge p_0)\rangle.$$

As can be seen from the second example, formation sequences may contain 'junk': formulas which are redundant or do not contribute to the construction.

Proposition 13C. *Every wff α in* $\mathrm{Frm}(\mathcal{L}_0)$ *has a formation sequence.*

Proof. Suppose α is atomic. Then the sequence $\langle\alpha\rangle$ is a formation sequence for α. Now suppose that β and γ have formation sequences $\langle\beta_0, \ldots, \beta_n\rangle$ and $\langle\gamma_0, \ldots, \gamma_m\rangle$ respectively.

1. If $\alpha \equiv \neg\beta$, then $\langle\beta_0, \ldots, \beta_n, \neg\beta_n\rangle$ is a formation sequence for α.
2. If $\alpha \equiv (\beta \to \gamma)$, then $\langle\beta_0, \ldots, \beta_n, \gamma_0, \ldots, \gamma_m, (\beta_n \to \gamma_m)\rangle$ is a formation sequence for α.

By the principle of induction on wffs, every wff has a formation sequence. □

We can also prove the converse. This is important because it shows that our two ways of defining formulas are equivalent: they give the same results. It also means that we can prove theorems about formulas by using ordinary induction on the length of formation sequences.

Lemma 13D. *Suppose that $\langle\alpha_0, \ldots, \alpha_n\rangle$ is a formation sequence for α_n, and that $k \leq n$. Then $\langle\alpha_0, \ldots, \alpha_k\rangle$ is a formation sequence for α_k.*

Theorem 13E. $\mathrm{Frm}(\mathcal{L}_0)$ *is the set of all expressions (strings of symbols) in the language $\mathcal{L}_0$ with a formation sequence.*

Proof. Let F be the set of all strings of symbols in the language $\mathcal{L}_0$ that have a formation sequence. We have seen in Proposition 13C that $\mathrm{Frm}(\mathcal{L}_0) \subseteq F$, so now we prove the converse.

Suppose α has a formation sequence $\langle\alpha_0, \ldots, \alpha_n\rangle$. We prove that $\alpha \in \mathrm{Frm}(\mathcal{L}_0)$ by strong induction on n. Our induction hypothesis is that every string of symbols with a formation sequence of length $m < n$ is in $\mathrm{Frm}(\mathcal{L}_0)$. By the definition of a formation sequence, either α_n is atomic or there must exist $j, k < n$ such that one of the following is the case:

1. $\alpha_i \equiv \neg\alpha_j$.

2. $\alpha_i \equiv (\alpha_j \to \alpha_k)$.

Now we reason by cases. If α_n is atomic then $\alpha_n \in \mathrm{Frm}(\mathcal{L}_0)$. Suppose instead that $\alpha \equiv (\alpha_j \wedge \alpha_k)$. By Lemma 13D, $\langle \alpha_0, \ldots, \alpha_j \rangle$ and $\langle \alpha_0, \ldots, \alpha_k \rangle$ are formation sequences for α_j and α_k respectively. Since these are proper initial subsequences of the formation sequence for α, they both have length less than n. Therefore by the induction hypothesis, α_j and α_k are in $\mathrm{Frm}(\mathcal{L}_0)$, and so by the definition of a wff, so is $(\alpha_j \wedge \alpha_k)$. The other cases follow by parallel reasoning. □

§1.4 Valuations and Satisfaction

Definition 14A (Valuations). *Let $\{\mathbb{T}, \mathbb{F}\}$ be the set of the two truth values, "true" and "false." A* valuation *for $\mathcal{L}_0$ is a function $\mathfrak{v}$ assigning either $\mathbb{T}$ or $\mathbb{F}$ to the propositional variables of the language, i.e., $\mathfrak{v}\colon \mathrm{At}_0 \to \{\mathbb{T}, \mathbb{F}\}$.*

Definition 14B. *Given a valuation $\mathfrak{v}$, define the evaluation function $\overline{\mathfrak{v}}\colon \mathrm{Frm}(\mathcal{L}_0) \to \{\mathbb{T}, \mathbb{F}\}$ inductively by:*

$$\overline{\mathfrak{v}}(p_n) = \mathfrak{v}(p_n);$$

$$\overline{\mathfrak{v}}(\neg\alpha) = \begin{cases} \mathbb{T} & \text{if } \overline{\mathfrak{v}}(\alpha) = \mathbb{F}; \\ \mathbb{F} & \text{otherwise.} \end{cases}$$

$$\overline{\mathfrak{v}}(\alpha \to \beta) = \begin{cases} \mathbb{T} & \text{if } \overline{\mathfrak{v}}(\alpha) = \mathbb{F} \text{ or } \overline{\mathfrak{v}}(\beta) = \mathbb{T}; \\ \mathbb{F} & \text{if } \overline{\mathfrak{v}}(\alpha) = \mathbb{T} \text{ and } \overline{\mathfrak{v}}(\beta) = \mathbb{F}. \end{cases}$$

The clauses correspond to the following truth tables:

α	$\neg\alpha$
$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$

α	β	$\alpha \to \beta$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$

Theorem 14C (Local Determination). *Suppose that $\mathfrak{v}_1$ and $\mathfrak{v}_2$ are valuations that agree on the propositional letters occurring in α, i.e., $\mathfrak{v}_1(p_n) = \mathfrak{v}_2(p_n)$ whenever p_n occurs in some wff α. Then $\overline{\mathfrak{v}_1}$ and $\overline{\mathfrak{v}_2}$ also agree on α, i.e., $\overline{\mathfrak{v}_1}(\alpha) = \overline{\mathfrak{v}_2}(\alpha)$.*

Proof. By induction on α. □

Definition 14D (Satisfaction). *We can inductively define the notion of* satisfaction *of a wff α by a valuation $\mathfrak{v}$, $\mathfrak{v} \vDash \alpha$, as follows. (We write $\mathfrak{v} \nvDash \alpha$ to mean "not $\mathfrak{v} \vDash \alpha$.")*

1. *$\alpha \equiv p_i$: $\mathfrak{v} \vDash \alpha$ iff $\mathfrak{v}(p_i) = \mathbb{T}$.*

2. $\alpha \equiv \neg\beta$: $\mathfrak{v} \vDash \alpha$ *iff* $\mathfrak{v} \nvDash \beta$.

3. $\alpha \equiv (\beta \to \gamma)$: $\mathfrak{v} \vDash \alpha$ *iff* $\mathfrak{v} \nvDash \beta$ *or* $\mathfrak{v} \vDash \gamma$ *(or both).*

If Γ is a set of wffs, $\mathfrak{v} \vDash \Gamma$ iff $\mathfrak{v} \vDash \alpha$ for every $\alpha \in \Gamma$.

Proposition 14E. *$\mathfrak{v} \vDash \alpha$ iff $\overline{\mathfrak{v}}(\alpha) = \mathbb{T}$.*

Proof. By induction on α. □

§1.5 Semantic Notions

We define the following semantic notions:

Definition 15A.

1. *A wff α is* satisfiable *if for some $\mathfrak{v}$, $\mathfrak{v} \vDash \alpha$; it is* unsatisfiable *if for no $\mathfrak{v}$, $\mathfrak{v} \vDash \alpha$;*

2. *A wff α is a* tautology *if $\mathfrak{v} \vDash \alpha$ for all valuations v;*

3. *A wff α is* contingent *if it is satisfiable but not a tautology;*

4. *If Γ is a set of wffs, $\Gamma \vDash \alpha$ ("Γ entails α") if and only if $\mathfrak{v} \vDash \alpha$ for every valuation $\mathfrak{v}$ for which $\mathfrak{v} \vDash \Gamma$.*

5. *If Γ is a set of wffs, Γ is* satisfiable *if there is a valuation $\mathfrak{v}$ for which $\mathfrak{v} \vDash \Gamma$, and Γ is* unsatisfiable *otherwise.*

Proposition 15B.

1. *α is a tautology if and only if $\emptyset \vDash \alpha$;*

2. *If $\Gamma \vDash \alpha$ and $\Gamma \vDash \alpha \to \beta$ then $\Gamma \vDash \beta$;*

3. *If Γ is satisfiable then every finite subset of Γ is also satisfiable;*

4. *Monotonicity: if $\Gamma \subseteq \Delta$ and $\Gamma \vDash \alpha$ then also $\Delta \vDash \alpha$;*

5. *Transitivity: if $\Gamma \vDash \alpha$ and $\Delta \cup \{\alpha\} \vDash \beta$ then $\Gamma \cup \Delta \vDash \beta$.*

Proof. Exercise. □

Proposition 15C. *$\Gamma \vDash \alpha$ if and only if $\Gamma \cup \{\neg\alpha\}$ is unsatisfiable.*

Proof. Exercise. □

Theorem 15D (Semantic Deduction Theorem). *$\Gamma \vDash \alpha \to \beta$ if and only if $\Gamma \cup \{\alpha\} \vDash \beta$.*

Proof. Exercise. □

Problems

Problem 1. Prove Proposition 12B

Problem 2. Prove Proposition 12C

Problem 3. For each of the five wffs below determine whether the wff can be expressed as a substitution $\alpha[\beta/p_i]$ where α is (i) p_0; (ii) $(\neg p_0 \wedge p_1)$; and (iii) $((\neg p_0 \to p_1) \wedge p_2)$. In each case specify the relevant substitution.

1. p_1
2. $(\neg p_0 \wedge p_0)$
3. $((p_0 \vee p_1) \wedge p_2)$
4. $\neg((p_0 \to p_1) \wedge p_2)$
5. $((\neg(p_0 \to p_1) \to (p_0 \vee p_1)) \wedge \neg(p_0 \wedge p_1))$

Problem 4. Give a mathematically rigorous definition of $\alpha[\beta/p]$ by induction.

Problem 5. Consider adding to $\mathcal{L}_0$ a ternary connective $\diamondsuit$ with evaluation given by

$$\overline{\mathfrak{v}}(\diamondsuit(\alpha,\beta,\gamma)) = \begin{cases} \overline{\mathfrak{v}}(\beta) & \text{if } \overline{\mathfrak{v}}(\alpha) = \mathbb{T}; \\ \overline{\mathfrak{v}}(\gamma) & \text{if } \overline{\mathfrak{v}}(\alpha) = \mathbb{F}. \end{cases}$$

Write down the truth table for this connective.

Problem 6. Prove Proposition 14E

Problem 7. For each of the following four wffs determine whether it is (a) satisfiable, (b) tautology, and (c) contingent.

1. $(p_0 \to (\neg p_1 \to \neg p_0))$.
2. $((p_0 \wedge \neg p_1) \to (\neg p_0 \wedge p_2)) \leftrightarrow ((p_2 \to p_0) \to (p_0 \to p_1))$.
3. $(p_0 \leftrightarrow p_1) \to (p_2 \leftrightarrow \neg p_1)$.
4. $((p_0 \leftrightarrow (\neg p_1 \wedge p_2)) \vee (p_2 \to (p_0 \leftrightarrow p_1)))$.

Problem 8. Prove Proposition 15B

Problem 9. Prove Proposition 15C

Problem 10. Prove Theorem 15D

Chapter 2

Derivation Systems

§2.0 Introduction

Logics commonly have both a semantics and a derivation system. The semantics concerns concepts such as truth, satisfiability, validity, and entailment. The purpose of derivation systems is to provide a purely syntactic method of establishing entailment and validity. They are purely syntactic in the sense that a derivation in such a system is a finite syntactic object, usually a sequence (or other finite arrangement) of sentences or wffs. Good derivation systems have the property that any given sequence or arrangement of sentences or wffs can be verified mechanically to be "correct."

The simplest (and historically first) derivation systems for first-order logic were *axiomatic*. A sequence of wffs counts as a derivation in such a system if each individual wff in it is either among a fixed set of "axioms" or follows from wffs coming before it in the sequence by one of a fixed number of "inference rules"—and it can be mechanically verified if a wff is an axiom and whether it follows correctly from other wffs by one of the inference rules. Axiomatic derivation systems are easy to describe—and also easy to handle metatheoretically—but derivations in them are hard to read and understand, and are also hard to produce.

Other derivation systems have been developed with the aim of making it easier to construct derivations or easier to understand derivations once they are complete. Examples are natural deduction, truth trees, also known as tableaux proofs, and the sequent calculus. Some derivation systems are designed especially with mechanization in mind, e.g., the resolution method is easy to implement in software (but its derivations are essentially impossible to understand). Most of these other derivation systems represent derivations as trees of wffs rather than sequences. This makes it easier to see which parts of a derivation depend on which other parts.

So for a given logic, such as first-order logic, the different derivation systems will give different explications of what it is for a sentence to be a *theorem* and what it means for a sentence to be derivable from some others. However that is

done (via axiomatic derivations, natural deductions, sequent derivations, truth trees, resolution refutations), we want these relations to match the semantic notions of validity and entailment. Let's write $\vdash \alpha$ for "α is a theorem" and "$\Gamma \vdash \alpha$" for "α is derivable from Γ." However $\vdash$ is defined, we want it to match up with $\models$, that is:

1. $\vdash \alpha$ if and only if $\models \alpha$
2. $\Gamma \vdash \alpha$ if and only if $\Gamma \models \alpha$

The "only if" direction of the above is called *soundness*. A derivation system is sound if derivability guarantees entailment (or validity). Every decent derivation system has to be sound; unsound derivation systems are not useful at all. After all, the entire purpose of a derivation is to provide a syntactic guarantee of validity or entailment. We'll prove soundness for the derivation systems we present.

The converse "if" direction is also important: it is called *completeness*. A complete derivation system is strong enough to show that α is a theorem whenever α is valid, and that $\Gamma \vdash \alpha$ whenever $\Gamma \models \alpha$. Completeness is harder to establish, and some logics have no complete derivation systems. First-order logic does. Kurt Gödel was the first one to prove completeness for a derivation system of first-order logic in his 1929 dissertation.

Another concept that is connected to derivation systems is that of *consistency*. A set of sentences is called inconsistent if anything whatsoever can be derived from it, and consistent otherwise. Inconsistency is the syntactic counterpart to unsatisfiablity: like unsatisfiable sets, inconsistent sets of sentences do not make good theories, they are defective in a fundamental way. Consistent sets of sentences may not be true or useful, but at least they pass that minimal threshold of logical usefulness. For different derivation systems the specific definition of consistency of sets of sentences might differ, but like $\vdash$, we want consistency to coincide with its semantic counterpart, satisfiability. We want it to always be the case that Γ is consistent if and only if it is satisfiable. Here, the "if" direction amounts to completeness (consistency guarantees satisfiability), and the "only if" direction amounts to soundness (satisfiability guarantees consistency). In fact, for classical first-order logic, the two versions of soundness and completeness are equivalent.

§2.1 The Sequent Calculus

While many derivation systems operate with arrangements of sentences, the sequent calculus operates with *sequents*. A sequent is an expression of the form

$$\alpha_1, \ldots, \alpha_m \Rightarrow \beta_1, \ldots, \beta_m,$$

that is a pair of sequences of sentences, separated by the sequent symbol $\Rightarrow$. Either sequence may be empty. A derivation in the sequent calculus is a tree of sequents, where the topmost sequents are of a special form (they are called

"initial sequents" or "axioms") and every other sequent follows from the sequents immediately above it by one of the rules of inference. The rules of inference either manipulate the sentences in the sequents (adding, removing, or rearranging them on either the left or the right), or they introduce a complex wff in the conclusion of the rule. For instance, the $\wedge$L rule allows the inference from $\alpha, \Gamma \Rightarrow \Delta$ to $\alpha \wedge \beta, \Gamma \Rightarrow \Delta$, and the $\rightarrow$R allows the inference from $\alpha, \Gamma \Rightarrow \Delta, \beta$ to $\Gamma \Rightarrow \Delta, \alpha \rightarrow \beta$, for any Γ, Δ, α, and β. (In particular, Γ and Δ may be empty.)

The $\vdash$ relation based on the sequent calculus is defined as follows: $\Gamma \vdash \alpha$ iff there is some sequence Γ_0 such that every α in Γ_0 is in Γ and there is a derivation with the sequent $\Gamma_0 \Rightarrow \alpha$ at its root. α is a theorem in the sequent calculus if the sequent $\Rightarrow \alpha$ has a derivation. For instance, here is a derivation that shows that $\vdash (\alpha \wedge \beta) \rightarrow \alpha$:

$$\dfrac{\dfrac{\alpha \Rightarrow \alpha}{\alpha \wedge \beta \Rightarrow \alpha}\,\wedge\text{L}}{\Rightarrow (\alpha \wedge \beta) \rightarrow \alpha}\,\rightarrow\text{R}$$

A set Γ is inconsistent in the sequent calculus if there is a derivation of $\Gamma_0 \Rightarrow$ (where every $\alpha \in \Gamma_0$ is in Γ and the right side of the sequent is empty). Using the rule WR, any sentence can be derived from an inconsistent set.

The sequent calculus was invented in the 1930s by Gerhard Gentzen. Because of its systematic and symmetric design, it is a very useful formalism for developing a theory of derivations. It is relatively easy to find derivations in the sequent calculus, but these derivations are often hard to read and their connection to proofs are sometimes not easy to see. It has proved to be a very elegant approach to derivation systems, however, and many logics have sequent calculus systems.

§2.2 Natural Deduction

Natural deduction is a derivation system intended to mirror actual reasoning (especially the kind of regimented reasoning employed by mathematicians). Actual reasoning proceeds by a number of "natural" patterns. For instance, proof by cases allows us to establish a conclusion on the basis of a disjunctive premise, by establishing that the conclusion follows from either of the disjuncts. Indirect proof allows us to establish a conclusion by showing that its negation leads to a contradiction. Conditional proof establishes a conditional claim "if ... then ..." by showing that the consequent follows from the antecedent. Natural deduction is a formalization of some of these natural inferences. Each of the logical connectives and quantifiers comes with two rules, an introduction and an elimination rule, and they each correspond to one such natural inference pattern. For instance, $\rightarrow$Intro corresponds to conditional proof, and $\vee$Elim to proof by cases. A particularly simple rule is $\wedge$Elim which allows the inference from $\alpha \wedge \beta$ to α (or β).

One feature that distinguishes natural deduction from other derivation systems is its use of assumptions. A derivation in natural deduction is a tree of

wffs. A single wff stands at the root of the tree of wffs, and the "leaves" of the tree are wffs from which the conclusion is derived. In natural deduction, some leaf wffs play a role inside the derivation but are "used up" by the time the derivation reaches the conclusion. This corresponds to the practice, in actual reasoning, of introducing hypotheses which only remain in effect for a short while. For instance, in a proof by cases, we assume the truth of each of the disjuncts; in conditional proof, we assume the truth of the antecedent; in indirect proof, we assume the truth of the negation of the conclusion. This way of introducing hypothetical assumptions and then doing away with them in the service of establishing an intermediate step is a hallmark of natural deduction. The formulas at the leaves of a natural deduction derivation are called assumptions, and some of the rules of inference may "discharge" them. For instance, if we have a derivation of β from some assumptions which include α, then the $\rightarrow$Intro rule allows us to infer $\alpha \rightarrow \beta$ and discharge any assumption of the form α. (To keep track of which assumptions are discharged at which inferences, we label the inference and the assumptions it discharges with a number.) The assumptions that remain undischarged at the end of the derivation are together sufficient for the truth of the conclusion, and so a derivation establishes that its undischarged assumptions entail its conclusion.

The relation $\Gamma \vdash \alpha$ based on natural deduction holds iff there is a derivation in which α is the last sentence in the tree, and every leaf which is undischarged is in Γ. α is a theorem in natural deduction iff there is a derivation in which α is the last sentence and all assumptions are discharged. For instance, here is a derivation that shows that $\vdash (\alpha \wedge \beta) \rightarrow \alpha$:

$$\cfrac{\cfrac{[\alpha \wedge \beta]^1}{\alpha}\,\wedge\text{Elim}}{(\alpha \wedge \beta) \rightarrow \alpha}\,{}^{1}\rightarrow\text{Intro}$$

The label 1 indicates that the assumption $\alpha \wedge \beta$ is discharged at the $\rightarrow$Intro inference.

A set Γ is inconsistent iff $\Gamma \vdash \bot$ in natural deduction. The rule $\bot_I$ makes it so that from an inconsistent set, any sentence can be derived.

Natural deduction systems were developed by Gerhard Gentzen and Stanisław Jaśkowski in the 1930s, and later developed by Dag Prawitz and Frederic Fitch. Because its inferences mirror natural methods of proof, it is favored by philosophers. The versions developed by Fitch are often used in introductory logic textbooks. In the philosophy of logic, the rules of natural deduction have sometimes been taken to give the meanings of the logical operators ("proof-theoretic semantics").

§2.3 Tableaux

While many derivation systems operate with arrangements of sentences, tableaux operate with signed formulas. A signed formula is a pair consisting of a truth

value sign ($\mathbb{T}$ or $\mathbb{F}$) and a sentence

$$\mathbb{T}\,\alpha \text{ or } \mathbb{F}\,\alpha.$$

A tableau consists of signed formulas arranged in a downward-branching tree. It begins with a number of *assumptions* and continues with signed formulas which result from one of the signed formulas above it by applying one of the rules of inference. Each rule allows us to add one or more signed formulas to the end of a branch, or two signed formulas side by side—in this case a branch splits into two, with the two added signed formulas forming the ends of the two branches.

A rule applied to a complex signed formula results in the addition of signed formulas which are immediate sub-wffs. They come in pairs, one rule for each of the two signs. For instance, the $\wedge\mathbb{T}$ rule applies to $\mathbb{T}\,\alpha \wedge \beta$, and allows the addition of both the two signed formulas $\mathbb{T}\,\alpha$ and $\mathbb{T}\,\beta$ to the end of any branch containing $\mathbb{T}\,\alpha \wedge \beta$, and the rule $\alpha \wedge \beta\mathbb{F}$ allows a branch to be split by adding $\mathbb{F}\,\alpha$ and $\mathbb{F}\,\beta$ side-by-side. A tableau is closed if every one of its branches contains a matching pair of signed formulas $\mathbb{T}\,\alpha$ and $\mathbb{F}\,\alpha$.

The $\vdash$ relation based on tableaux is defined as follows: $\Gamma \vdash \alpha$ iff there is some finite set $\Gamma_0 = \{\beta_1, \ldots, \beta_n\} \subseteq \Gamma$ such that there is a closed tableau for the assumptions

$$\{\mathbb{F}\,\alpha, \mathbb{T}\,\beta_1, \ldots, \mathbb{T}\,\beta_n\}$$

For instance, here is a closed tableau that shows that $\vdash (\alpha \wedge \beta) \to \alpha$:

1.	$\mathbb{F}\,(\alpha \wedge \beta) \to \alpha$	Assumption
2.	$\mathbb{T}\,\alpha \wedge \beta$	$\to\mathbb{F}\,1$
3.	$\mathbb{F}\,\alpha$	$\to\mathbb{F}\,1$
4.	$\mathbb{T}\,\alpha$	$\to\mathbb{T}\,2$
5.	$\mathbb{T}\,\beta$	$\to\mathbb{T}\,2$
	$\otimes$	

A set Γ is inconsistent in the tableau calculus if there is a closed tableau for assumptions

$$\{\mathbb{T}\,\beta_1, \ldots, \mathbb{T}\,\beta_n\}$$

for some $\beta_i \in \Gamma$.

Tableaux were invented in the 1950s independently by Evert Beth and Jaakko Hintikka, and simplified and popularized by Raymond Smullyan. They are very easy to use, since constructing a tableau is a very systematic procedure. Because of the systematic nature of tableaux, they also lend themselves to implementation by computer. However, a tableau is often hard to read and their connection to proofs are sometimes not easy to see. The approach is also quite general, and many different logics have tableau systems. Tableaux also help us to find structures that satisfy given (sets of) sentences: if the set is satisfiable, it won't have a closed tableau, i.e., any tableau will have an open branch. The satisfying structure can be "read off" an open branch, provided

every rule it is possible to apply has been applied on that branch. There is also a very close connection to the sequent calculus: essentially, a closed tableau is a condensed derivation in the sequent calculus, written upside-down.

§2.4 Axiomatic Derivations

Axiomatic derivations are the oldest and simplest logical derivation systems. Its derivations are simply sequences of sentences. A sequence of sentences counts as a correct derivation if every sentence α in it satisfies one of the following conditions:

1. α is an axiom, or
2. α is an element of a given set Γ of sentences, or
3. α is justified by a rule of inference.

To be an axiom, α has to have the form of one of a number of fixed sentence schemas. There are many sets of axiom schemas that provide a satisfactory (sound and complete) derivation system for first-order logic. Some are organized according to the connectives they govern, e.g., the schemas

$$\alpha \to (\beta \to \alpha) \qquad \beta \to (\beta \vee \gamma) \qquad (\beta \wedge \gamma) \to \beta$$

are common axioms that govern $\to$, $\vee$ and $\wedge$. Some axiom systems aim at a minimal number of axioms. Depending on the connectives that are taken as primitives, it is even possible to find axiom systems that consist of a single axiom.

A rule of inference is a conditional statement that gives a sufficient condition for a sentence in a derivation to be justified. Modus ponens is one very common such rule: it says that if α and $\alpha \to \beta$ are already justified, then β is justified. This means that a line in a derivation containing the sentence β is justified, provided that both α and $\alpha \to \beta$ (for some sentence α) appear in the derivation before β.

The $\vdash$ relation based on axiomatic derivations is defined as follows: $\Gamma \vdash \alpha$ iff there is a derivation with the sentence α as its last formula (and Γ is taken as the set of sentences in that derivation which are justified by (2) above). α is a theorem if α has a derivation where Γ is empty, i.e., every sentence in the derivation is justified either by (1) or (3). For instance, here is a derivation that shows that $\vdash \alpha \to (\beta \to (\beta \vee \alpha))$:

1. $\beta \to (\beta \vee \alpha)$
2. $(\beta \to (\beta \vee \alpha)) \to (\alpha \to (\beta \to (\beta \vee \alpha)))$
3. $\alpha \to (\beta \to (\beta \vee \alpha))$

The sentence on line 1 is of the form of the axiom $\alpha \to (\alpha \vee \beta)$ (with the roles of α and β reversed). The sentence on line 2 is of the form of the axiom $\alpha \to (\beta \to \alpha)$. Thus, both lines are justified. Line 3 is justified by modus ponens:

if we abbreviate it as δ, then line 2 has the form $\gamma \to \delta$, where γ is $\beta \to (\beta \vee \alpha)$, i.e., line 1.

A set Γ is inconsistent if $\Gamma \vdash \bot$. A complete axiom system will also prove that $\bot \to \alpha$ for any α, and so if Γ is inconsistent, then $\Gamma \vdash \alpha$ for any α.

Systems of axiomatic derivations for logic were first given by Gottlob Frege in his 1879 *Begriffsschrift*, which for this reason is often considered the first work of modern logic. They were perfected in Alfred North Whitehead and Bertrand Russell's *Principia Mathematica* and by David Hilbert and his students in the 1920s. They are thus often called "Frege systems" or "Hilbert systems." They are very versatile in that it is often easy to find an axiomatic system for a logic. Because derivations have a very simple structure and only one or two inference rules, it is also relatively easy to prove things *about* them. However, they are very hard to use in practice, i.e., it is difficult to find and write proofs.

Chapter 3

Axiomatic Derivations

§3.0 Rules and Derivations

Axiomatic derivations are perhaps the simplest derivation system for logic. A derivation is just a sequence of wffs. To count as a derivation, every wff in the sequence must either be an instance of an axiom, or must follow from one or more wffs that precede it in the sequence by a rule of inference. A derivation derives its last wff.

Definition 30A (Derivability). *If Γ is a set of wffs of $\mathcal{L}$ then a* derivation *from Γ is a finite sequence $\alpha_1, \ldots, \alpha_n$ of wffs where for each $i \leq n$ one of the following holds:*

1. *$\alpha_i \in \Gamma$; or*
2. *α_i is an axiom; or*
3. *α_i follows from some α_j (and α_k) with $j < i$ (and $k < i$) by a rule of inference.*

What counts as a correct derivation depends on which inference rules we allow (and of course what we take to be axioms). And an inference rule is an if-then statement that tells us that, under certain conditions, a step A_i in a derivation is a correct inference step.

Definition 30B (Rule of inference). *A* rule of inference *gives a sufficient condition for what counts as a correct inference step in a derivation from Γ.*

For instance, since any one-element sequence α with $\alpha \in \Gamma$ trivially counts as a derivation, the following might be a very simple rule of inference:

> If $\alpha \in \Gamma$, then α is always a correct inference step in any derivation from Γ.

Similarly, if α is one of the axioms, then α by itself is a derivation, and so this is also a rule of inference:

> If α is an axiom, then α is a correct inference step.

It gets more interesting if the rule of inference appeals to wffs that appear before the step considered. The following rule is called *modus ponens:*

> If $\beta \to \alpha$ and β occur higher up in the derivation, then α is a correct inference step.

If this is the only rule of inference, then our definition of derivation above amounts to this: $\alpha_1, \ldots, \alpha_n$ is a derivation iff for each $i \leq n$ one of the following holds:

1. $\alpha_i \in \Gamma$; or
2. α_i is an axiom; or
3. for some $j < i$, α_j is $\beta \to \alpha_i$, and for some $k < i$, α_k is β.

The last clause says that α_i follows from α_j (β) and α_k ($\beta \to \alpha_i$) by modus ponens. If we can go from 1 to n, and each time we find a wff α_i that is either in Γ, an axiom, or which a rule of inference tells us that it is a correct inference step, then the entire sequence counts as a correct derivation.

Definition 30C (Derivability). *A wff α is* derivable *from Γ, written $\Gamma \vdash \alpha$, if there is a derivation from Γ ending in α.*

Definition 30D (Theorems). *A wff α is a* theorem *if there is a derivation of α from the empty set. We write $\vdash \alpha$ if α is a theorem and $\nvdash \alpha$ if it is not.*

§3.1 Axiom and Rules for the Propositional Connectives

Definition 31A (Axioms). *The set of* Ax_0 *of* axioms *for the propositional*

connectives comprises all wffs of the following forms:

$$(\alpha \wedge \beta) \to \alpha \tag{3.1}$$
$$(\alpha \wedge \beta) \to \beta \tag{3.2}$$
$$\alpha \to (\beta \to (\alpha \wedge \beta)) \tag{3.3}$$
$$\alpha \to (\alpha \vee \beta) \tag{3.4}$$
$$\alpha \to (\beta \vee \alpha) \tag{3.5}$$
$$(\alpha \to \gamma) \to ((\beta \to \gamma) \to ((\alpha \vee \beta) \to \gamma)) \tag{3.6}$$
$$\alpha \to (\beta \to \alpha) \tag{3.7}$$
$$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)) \tag{3.8}$$
$$(\alpha \to \beta) \to ((\alpha \to \neg\beta) \to \neg\alpha) \tag{3.9}$$
$$\neg\alpha \to (\alpha \to \beta) \tag{3.10}$$
$$\top \tag{3.11}$$
$$\bot \to \alpha \tag{3.12}$$
$$(\alpha \to \bot) \to \neg\alpha \tag{3.13}$$
$$\neg\neg\alpha \to \alpha \tag{3.14}$$

Definition 31B (Modus ponens). *If β and $\beta \to \alpha$ already occur in a derivation, then α is a correct inference step.*

We'll abbreviate the rule modus ponens as "MP."

§3.2 Examples of Derivations

Example 3.2.1. Suppose we want to prove $(\neg\delta \vee \varphi) \to (\delta \to \varphi)$. Clearly, this is not an instance of any of our axioms, so we have to use the MP rule to derive it. Our only rule is MP, which given α and $\alpha \to \beta$ allows us to justify β. One strategy would be to use eq. (3.6) with α being $\neg\delta$, β being φ, and γ being $\delta \to \varphi$, i.e., the instance

$$(\neg\delta \to (\delta \to \varphi)) \to ((\varphi \to (\delta \to \varphi)) \to ((\neg\delta \vee \varphi) \to (\delta \to \varphi))).$$

Why? Two applications of MP yield the last part, which is what we want. And we easily see that $\neg\delta \to (\delta \to \varphi)$ is an instance of eq. (3.10), and $\varphi \to (\delta \to \varphi)$ is an instance of eq. (3.7). So our derivation is:

1.	$\neg\delta \to (\delta \to \varphi)$	eq. (3.10)
2.	$(\neg\delta \to (\delta \to \varphi)) \to$ $((\varphi \to (\delta \to \varphi)) \to ((\neg\delta \vee \varphi) \to (\delta \to \varphi)))$	eq. (3.6)
3.	$((\varphi \to (\delta \to \varphi)) \to ((\neg\delta \vee \varphi) \to (\delta \to \varphi))$	1, 2, MP
4.	$\varphi \to (\delta \to \varphi)$	eq. (3.7)
5.	$(\neg\delta \vee \varphi) \to (\delta \to \varphi)$	3, 4, MP

Example 3.2.2. Let's try to find a derivation of $\delta \to \delta$. It is not an instance of an axiom, so we have to use MP to derive it. eq. (3.7) is an axiom of the form $\alpha \to \beta$ to which we could apply MP. To be useful, of course, the β which MP would justify as a correct step in this case would have to be $\delta \to \delta$, since this is what we want to derive. That means α would also have to be δ, i.e., we might look at this instance of eq. (3.7):

$$\delta \to (\delta \to \delta)$$

In order to apply MP, we would also need to justify the corresponding second premise, namely α. But in our case, that would be δ, and we won't be able to derive δ by itself. So we need a different strategy.

The other axiom involving just $\to$ is eq. (3.8), i.e.,

$$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$$

We could get to the last nested conditional by applying MP twice. Again, that would mean that we want an instance of eq. (3.8) where $\alpha \to \gamma$ is $\delta \to \delta$, the wff we are aiming for. Then of course, α and γ are both δ. How should we pick β so that both $\alpha \to (\beta \to \gamma)$ and $\alpha \to \beta$, i.e., in our case $\delta \to (\beta \to \delta)$ and $\delta \to \beta$, are also derivable? Well, the first of these is already an instance of eq. (3.7), whatever we decide β to be. And $\delta \to \beta$ would be another instance of eq. (3.7) if β were $(\delta \to \delta)$. So, our derivation is:

1.	$\delta \to ((\delta \to \delta) \to \delta)$	eq. (3.7)
2.	$(\delta \to ((\delta \to \delta) \to \delta)) \to$ $((\delta \to (\delta \to \delta)) \to (\delta \to \delta))$	eq. (3.8)
3.	$(\delta \to (\delta \to \delta)) \to (\delta \to \delta)$	1, 2, MP
4.	$\delta \to (\delta \to \delta)$	eq. (3.7)
5.	$\delta \to \delta$	3, 4, MP

Example 3.2.3. Sometimes we want to show that there is a derivation of some wff from some other wffs Γ. For instance, let's show that we can derive $\alpha \to \gamma$ from $\Gamma = \{\alpha \to \beta, \beta \to \gamma\}$.

1.	$\alpha \to \beta$	Hyp
2.	$\beta \to \gamma$	Hyp
3.	$(\beta \to \gamma) \to (\alpha \to (\beta \to \gamma))$	eq. (3.7)
4.	$\alpha \to (\beta \to \gamma)$	2, 3, MP
5.	$(\alpha \to (\beta \to \gamma)) \to$ $((\alpha \to \beta) \to (\alpha \to \gamma))$	eq. (3.8)
6.	$((\alpha \to \beta) \to (\alpha \to \gamma))$	4, 5, MP
7.	$\alpha \to \gamma$	1, 6, MP

The lines labelled "Hyp" (for "hypothesis") indicate that the wff on that line is an element of Γ.

Proposition 32D. *If $\Gamma \vdash \alpha \to \beta$ and $\Gamma \vdash \beta \to \gamma$, then $\Gamma \vdash \alpha \to \gamma$*

Proof. Suppose $\Gamma \vdash \alpha \to \beta$ and $\Gamma \vdash \beta \to \gamma$. Then there is a derivation of $\alpha \to \beta$ from Γ; and a derivation of $\beta \to \gamma$ from Γ as well. Combine these into a single derivation by concatenating them. Now add lines 3–7 of the derivation in the preceding example. This is a derivation of $\alpha \to \gamma$—which is the last line of the new derivation—from Γ. Note that the justifications of lines 4 and 7 remain valid if the reference to line number 2 is replaced by reference to the last line of the derivation of $\alpha \to \beta$, and reference to line number 1 by reference to the last line of the derivation of $B \to \gamma$. □

§3.3 Proof-Theoretic Notions

Just as we've defined a number of important semantic notions (tautology, entailment, satisfiability), 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 formulas. It was an important discovery that these notions coincide. That they do is the content of the *soundness* and *completeness theorems*.

Definition 33A (Derivability). *A wff α is* derivable *from Γ, written $\Gamma \vdash \alpha$, if there is a derivation from Γ ending in α.*

Definition 33B (Theorems). *A wff α is a* theorem *if there is a derivation of α from the empty set. We write $\vdash \alpha$ if α is a theorem and $\nvdash \alpha$ if it is not.*

Definition 33C (Consistency). *A set Γ of wffs is* consistent *if and only if $\Gamma \nvdash \bot$; it is* inconsistent *otherwise.*

Proposition 33D (Reflexivity). *If $\alpha \in \Gamma$, then $\Gamma \vdash \alpha$.*

Proof. The wff α by itself is a derivation of α from Γ. □

Proposition 33E (Monotonicity). *If $\Gamma \subseteq \Delta$ and $\Gamma \vdash \alpha$, then $\Delta \vdash \alpha$.*

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

Proposition 33F (Transitivity). *If $\Gamma \vdash \alpha$ and $\{\alpha\} \cup \Delta \vdash \beta$, then $\Gamma \cup \Delta \vdash \beta$.*

Proof. Suppose $\{\alpha\} \cup \Delta \vdash \beta$. Then there is a derivation $\beta_1, \ldots, \beta_l = \beta$ from $\{\alpha\} \cup \Delta$. Some of the steps in that derivation will be correct because of a rule which refers to a prior line $\beta_i = \alpha$. By hypothesis, there is a derivation of α from Γ, i.e., a derivation $\alpha_1, \ldots, \alpha_k = \alpha$ where every α_i is an axiom, an element of Γ, or correct by a rule of inference. Now consider the sequence

$$\alpha_1, \ldots, \alpha_k = \alpha, \beta_1, \ldots, \beta_l = \beta.$$

This is a correct derivation of β from $\Gamma \cup \Delta$ since every $B_i = \alpha$ is now justified by the same rule which justifies $\alpha_k = \alpha$. □

Note that this means that in particular if $\Gamma \vdash \alpha$ and $\alpha \vdash \beta$, then $\Gamma \vdash \beta$. It follows also that if $\alpha_1, \ldots, \alpha_n \vdash \beta$ and $\Gamma \vdash \alpha_i$ for each i, then $\Gamma \vdash \beta$.

Proposition 33G. *Γ is inconsistent iff $\Gamma \vdash \alpha$ for every α.*

Proof. Exercise. □

Proposition 33H (Compactness).

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

Proof. 1. If $\Gamma \vdash \alpha$, then there is a finite sequence of wffs $\alpha_1, \ldots, \alpha_n$ so that $\alpha \equiv \alpha_n$ and each α_i is either a logical axiom, an element of Γ or follows from previous wffs by modus ponens. Take Γ_0 to be those α_i which are in Γ. Then the derivation is likewise a derivation from Γ_0, and so $\Gamma_0 \vdash \alpha$.

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

§3.4 The Deduction Theorem

As we've seen, giving derivations in an axiomatic system is cumbersome, and derivations may be hard to find. Rather than actually write out long lists of wffs, it is generally easier to argue that such derivations exist, by making use of a few simple results. We've already established three such results: Proposition 33D says we can always assert that $\Gamma \vdash \alpha$ when we know that $\alpha \in \Gamma$. Proposition 33E says that if $\Gamma \vdash \alpha$ then also $\Gamma \cup \{\beta\} \vdash \alpha$. And Proposition 33F implies that if $\Gamma \vdash \alpha$ and $\alpha \vdash \beta$, then $\Gamma \vdash \beta$. Here's another simple result, a "meta"-version of modus ponens:

Proposition 34A. *If $\Gamma \vdash \alpha$ and $\Gamma \vdash \alpha \to \beta$, then $\Gamma \vdash \beta$.*

Proof. We have that $\{\alpha, \alpha \to \beta\} \vdash \beta$:

1.	α	Hyp.
2.	$\alpha \to \beta$	Hyp.
3.	β	1, 2, MP

By Proposition 33F, $\Gamma \vdash \beta$. □

The most important result we'll use in this context is the deduction theorem:

Theorem 34B (Deduction Theorem). *$\Gamma \cup \{\alpha\} \vdash \beta$ if and only if $\Gamma \vdash \alpha \to \beta$.*

Proof. The "if" direction is immediate. If $\Gamma \vdash \alpha \to \beta$ then also $\Gamma \cup \{\alpha\} \vdash \alpha \to \beta$ by Proposition 33E. Also, $\Gamma \cup \{\alpha\} \vdash \alpha$ by Proposition 33D. So, by Proposition 34A, $\Gamma \cup \{\alpha\} \vdash \beta$.

For the "only if" direction, we proceed by induction on the length of the derivation of β from $\Gamma \cup \{\alpha\}$.

For the induction basis, we prove the claim for every derivation of length 1. A derivation of β from $\Gamma \cup \{\alpha\}$ of length 1 consists of β by itself; and if it is correct β is either $\in \Gamma \cup \{\alpha\}$ or is an axiom. If $\beta \in \Gamma$ or is an axiom, then $\Gamma \vdash \beta$. We also have that $\Gamma \vdash \beta \to (\alpha \to \beta)$ by eq. (3.7), and Proposition 34A gives $\Gamma \vdash \alpha \to \beta$. If $\beta \in \{\alpha\}$ then $\Gamma \vdash \alpha \to \beta$ because then last sentence $\alpha \to \beta$ is the same as $\alpha \to \alpha$, and we have derived that in Example 3.2.2.

For the inductive step, suppose a derivation of β from $\Gamma \cup \{\alpha\}$ ends with a step β which is justified by modus ponens. (If it is not justified by modus ponens, $\beta \in \Gamma$, $\beta \equiv \alpha$, or β is an axiom, and the same reasoning as in the induction basis applies.) Then some previous steps in the derivation are $\gamma \to \beta$ and γ, for some wff γ, i.e., $\Gamma \cup \{\alpha\} \vdash \gamma \to \beta$ and $\Gamma \cup \{\alpha\} \vdash \gamma$, and the respective derivations are shorter, so the inductive hypothesis applies to them. We thus have both:

$$\Gamma \vdash \alpha \to (\gamma \to \beta);$$
$$\Gamma \vdash \alpha \to \gamma.$$

But also

$$\Gamma \vdash (\alpha \to (\gamma \to \beta)) \to ((\alpha \to \gamma) \to (\alpha \to \beta)),$$

by eq. (3.8), and two applications of Proposition 34A give $\Gamma \vdash \alpha \to \beta$, as required. □

Notice how eq. (3.7) and eq. (3.8) were chosen precisely so that the Deduction Theorem would hold.

The following are some useful facts about derivability, which we leave as exercises.

Proposition 34C.

1. $\vdash (\alpha \to \beta) \to ((\beta \to \gamma) \to (\alpha \to \gamma))$;
2. *If* $\Gamma \cup \{\neg\alpha\} \vdash \neg\beta$ *then* $\Gamma \cup \{\beta\} \vdash \alpha$ *(Contraposition);*
3. $\{\alpha, \neg\alpha\} \vdash \beta$ *(Ex Falso Quodlibet, Explosion);*
4. $\{\neg\neg\alpha\} \vdash \alpha$ *(Double Negation Elimination);*
5. *If* $\Gamma \vdash \neg\neg\alpha$ *then* $\Gamma \vdash \alpha$;

§3.5 Derivability and Consistency

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

Proposition 35A. *If $\Gamma \vdash \alpha$ and $\Gamma \cup \{\alpha\}$ is inconsistent, then Γ is inconsistent.*

Proof. If $\Gamma \cup \{\alpha\}$ is inconsistent, then $\Gamma \cup \{\alpha\} \vdash \bot$. By Proposition 33D, $\Gamma \vdash \beta$ for every $\beta \in \Gamma$. Since also $\Gamma \vdash \alpha$ by hypothesis, $\Gamma \vdash \beta$ for every $\beta \in \Gamma \cup \{\alpha\}$. By Proposition 33F, $\Gamma \vdash \bot$, i.e., Γ is inconsistent. □

Proposition 35B. *$\Gamma \vdash \alpha$ iff $\Gamma \cup \{\neg\alpha\}$ is inconsistent.*

Proof. First suppose $\Gamma \vdash \alpha$. Then $\Gamma \cup \{\neg\alpha\} \vdash \alpha$ by Proposition 33E. $\Gamma \cup \{\neg\alpha\} \vdash \neg\alpha$ by Proposition 33D. We also have $\vdash \neg\alpha \to (\alpha \to \bot)$ by eq. (3.10). So by two applications of Proposition 34A, we have $\Gamma \cup \{\neg\alpha\} \vdash \bot$.

Now assume $\Gamma \cup \{\neg\alpha\}$ is inconsistent, i.e., $\Gamma \cup \{\neg\alpha\} \vdash \bot$. By the deduction theorem, $\Gamma \vdash \neg\alpha \to \bot$. $\Gamma \vdash (\neg\alpha \to \bot) \to \neg\neg\alpha$ by eq. (3.13), so $\Gamma \vdash \neg\neg\alpha$ by Proposition 34A. Since $\Gamma \vdash \neg\neg\alpha \to \alpha$ (eq. (3.14)), we have $\Gamma \vdash \alpha$ by Proposition 34A again. □

Proposition 35C. *If $\Gamma \vdash \alpha$ and $\neg\alpha \in \Gamma$, then Γ is inconsistent.*

Proof. $\Gamma \vdash \neg\alpha \to (\alpha \to \bot)$ by eq. (3.10). $\Gamma \vdash \bot$ by two applications of Proposition 34A. □

Proposition 35D. *If $\Gamma \cup \{\alpha\}$ and $\Gamma \cup \{\neg\alpha\}$ are both inconsistent, then Γ is inconsistent.*

Proof. Exercise. □

§3.6 Derivability and the Propositional Connectives

We establish that the derivability relation $\vdash$ of axiomatic deduction is strong enough to establish some basic facts involving the propositional connectives, such as that $\alpha \wedge \beta \vdash \alpha$ and $\alpha, \alpha \to \beta \vdash \beta$ (modus ponens). These facts are needed for the proof of the completeness theorem.

Proposition 36A.

1. *Both $\alpha \wedge \beta \vdash \alpha$ and $\alpha \wedge \beta \vdash \beta$*
2. *$\alpha, \beta \vdash \alpha \wedge \beta$.*

Proof. 1. From eq. (3.1) and eq. (3.1) by modus ponens.

2. From eq. (3.3) by two applications of modus ponens. □

Proposition 36B.

1. $\alpha \vee \beta, \neg\alpha, \neg\beta$ *is inconsistent.*
2. *Both* $\alpha \vdash \alpha \vee \beta$ *and* $\beta \vdash \alpha \vee \beta$.

Proof. 1. From eq. (3.9) we get $\vdash \neg\alpha \to (\alpha \to \bot)$ and $\vdash \neg\beta \to (\beta \to \bot)$. So by the deduction theorem, we have $\{\neg\alpha\} \vdash \alpha \to \bot$ and $\{\neg\beta\} \vdash \beta \to \bot$. From eq. (3.6) we get $\{\neg\alpha, \neg\beta\} \vdash (\alpha \vee \beta) \to \bot$. By the deduction theorem, $\{\alpha \vee \beta, \neg\alpha, \neg\beta\} \vdash \bot$.

2. From eq. (3.4) and eq. (3.5) by modus ponsens. □

Proposition 36C.

1. $\alpha, \alpha \to \beta \vdash \beta$.
2. *Both* $\neg\alpha \vdash \alpha \to \beta$ *and* $\beta \vdash \alpha \to \beta$.

Proof. 1. We can derive:

1.	α	Hyp
2.	$\alpha \to \beta$	Hyp
3.	β	1, 2, MP

2. By eq. (3.10) and eq. (3.7) and the deduction theorem, respectively. □

§3.7 Soundness

A derivation system, such as axiomatic deduction, 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 α is valid;
2. if α is derivable from some others Γ, it is also a consequence of them;
3. if a set of wffs Γ 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.

Proposition 37A. *If* α *is an axiom, then* $\mathfrak{v} \vDash \alpha$ *for each valuation* $\mathfrak{v}$.

Proof. Do truth tables for each axiom to verify that they are tautologies. □

Theorem 37B (Soundness). *If $\Gamma \vdash \alpha$ then $\Gamma \vDash \alpha$.*

Proof. By induction on the length of the derivation of α from Γ. If there are no steps justified by inferences, then all wffs in the derivation are either instances of axioms or are in Γ. By the previous proposition, all the axioms are tautologies, and hence if α is an axiom then $\Gamma \vDash \alpha$. If $\alpha \in \Gamma$, then trivially $\Gamma \vDash \alpha$.

If the last step of the derivation of α is justified by modus ponens, then there are wffs β and $\beta \to \alpha$ in the derivation, and the induction hypothesis applies to the part of the derivation ending in those wffs (since they contain at least one fewer steps justified by an inference). So, by induction hypothesis, $\Gamma \vDash \beta$ and $\Gamma \vDash \beta \to \alpha$. Then $\Gamma \vDash \alpha$ by Theorem 15D.

Corollary 37C. *If $\vdash \alpha$, then α is a tautology.*

Corollary 37D. *If Γ is satisfiable, then it is consistent.*

Proof. We prove the contrapositive. Suppose that Γ is not consistent. Then $\Gamma \vdash \bot$, i.e., there is a derivation of $\bot$ from Γ. By Theorem 37B, any valuation $\mathfrak{v}$ that satisfies Γ must satisfy $\bot$. Since $\mathfrak{v} \nvDash \bot$ for every valuation $\mathfrak{v}$, no $\mathfrak{v}$ can satisfy Γ, i.e., Γ is not satisfiable. □

Problems

Problem 1. Show that the following hold by exhibiting derivations from the axioms:

1. $(\alpha \wedge \beta) \to (\beta \wedge \alpha)$
2. $((\alpha \wedge \beta) \to \gamma) \to (\alpha \to (\beta \to \gamma))$
3. $\neg(\alpha \vee \beta) \to \neg\alpha$

Problem 2. Prove Proposition 33G.

Problem 3. Prove Proposition 34C

Problem 4. Prove that $\Gamma \vdash \neg\alpha$ iff $\Gamma \cup \{\alpha\}$ is inconsistent.

Problem 5. Prove Proposition 35D

Chapter 4

The Completeness Theorem

§4.0 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 derivation system: if a sentence α follows from some sentences Γ, then there is also a derivation that establishes $\Gamma \vdash \alpha$. Thus, the derivation system is as strong as it can possibly be without proving things that don't actually follow.

In its second formulation, it can be stated as a model existence result: every consistent set of sentences is satisfiable. Consistency is a proof-theoretic notion: it says that our derivation system is unable to produce certain derivations. But who's to say that just because there are no derivations of a certain sort from Γ, it's guaranteed that there is valuation $\mathfrak{v}$ with $\mathfrak{v} \vDash \Gamma$? Before the completeness theorem was first proved—in fact before we had the derivation systems we now do—the great German mathematician David Hilbert held the view that consistency of mathematical theories guarantees the existence of the objects they are about. He put it as follows in a letter to Gottlob Frege:

> If the arbitrarily given axioms do not contradict one another with all their consequences, then they are true and the things defined by the axioms exist. This is for me the criterion of truth and existence.

Frege vehemently disagreed. The second formulation of the completeness theorem shows that Hilbert was right in at least the sense that if the axioms are consistent, then *some* valuation exists that makes them all true.

These aren't the only reasons the completeness theorem—or rather, its proof—is important. It has a number of important consequences, some of which we'll discuss separately. For instance, since any derivation that shows $\Gamma \vdash \alpha$ is finite and so can only use finitely many of the sentences in Γ, it follows by the completeness theorem that if α is a consequence of Γ, it is already a

consequence of a finite subset of Γ. This is called *compactness*. Equivalently, if every finite subset of Γ is consistent, then Γ itself must be consistent.

Although the compactness theorem follows from the completeness theorem via the detour through derivations, it is also possible to use the *the proof of* the completeness theorem to establish it directly. For what the proof does is take a set of sentences with a certain property—consistency—and constructs a structure out of this set that has certain properties (in this case, that it satisfies the set). Almost the very same construction can be used to directly establish compactness, by starting from "finitely satisfiable" sets of sentences instead of consistent ones.

§4.1 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 $\Gamma \vDash \alpha$ then $\Gamma \vdash \alpha$," it may be hard to even come up with an idea: for to show that $\Gamma \vdash \alpha$ we have to find a derivation, and it does not look like the hypothesis that $\Gamma \vDash \alpha$ 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 approach. The shift in perspective required is this: completeness can also be formulated as: "if Γ is consistent, it is satisfiable." Perhaps we can use the information in Γ together with the hypothesis that it is consistent to construct a valuation that satisfies every wff in Γ. After all, we know what kind of valuation we are looking for: one that is as Γ describes it!

If Γ contains only propositional variables, it is easy to construct a model for it. All we have to do is come up with a valuation $\mathfrak{v}$ such that $\mathfrak{v} \vDash p$ for all $p \in \Gamma$. Well, let $\mathfrak{v}(p) = \mathbb{T}$ iff $p \in \Gamma$.

Now suppose Γ contains some wff $\neg\beta$, with β atomic. We might worry that the construction of $\mathfrak{v}$ interferes with the possibility of making $\neg\beta$ true. But here's where the consistency of Γ comes in: if $\neg\beta \in \Gamma$, then $\beta \notin \Gamma$, or else Γ would be inconsistent. And if $\beta \notin \Gamma$, then according to our construction of $\mathfrak{v}$, $\mathfrak{v} \nvDash \beta$, so $\mathfrak{v} \vDash \neg\beta$. So far so good.

What if Γ contains complex, non-atomic formulas? Say it contains $\alpha \wedge \beta$. To make that true, we should proceed as if both α and β were in Γ. And if $\alpha \vee \beta \in \Gamma$, 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 wffs to Γ so as to (a) keep the resulting set consistent and (b) make sure that for every possible atomic sentence α, either α is in the resulting set, or $\neg\alpha$ is, and (c) such that, whenever $\alpha \wedge \beta$ is in the set, so are both α and β, if $\alpha \vee \beta$ is in the set, at least one of α or β is also, etc. We keep doing this (potentially forever). Call the set of all wffs so added Γ^*. Then our construction above would provide us with a valuation $\mathfrak{v}$ for which we could prove, by induction, that it satisfies all sentences in Γ^*, and hence also all sentence in Γ since $\Gamma \subseteq \Gamma^*$. It turns

out that guaranteeing (a) and (b) is enough. A set of sentences for which (b) holds is called *complete*. So our task will be to extend the consistent set Γ to a consistent and complete set Γ^*.

So here's what we'll do. First we investigate the properties of complete consistent sets, in particular we prove that a complete consistent set contains $\alpha \wedge \beta$ iff it contains both α and β, $\alpha \vee \beta$ iff it contains at least one of them, etc. (Proposition 42B). We'll then take the consistent set Γ and show that it can be extended to a consistent and complete set Γ^* (Lemma 43A). This set Γ^* is what we'll use to define our valuation $\mathfrak{v}(\Gamma^*)$. The valuation is determined by the propositional variables in Γ^* (Definition 44A). We'll use the properties of complete consistent sets to show that indeed $\mathfrak{v}(\Gamma^*) \vDash \alpha$ iff $\alpha \in \Gamma^*$ (Lemma 44B), and thus in particular, $\mathfrak{v}(\Gamma^*) \vDash \Gamma$.

§4.2 Complete Consistent Sets of Sentences

Definition 42A (Complete set). *A set Γ of sentences is* complete *iff for any sentence α, either $\alpha \in \Gamma$ or $\neg\alpha \in \Gamma$.*

Complete sets of sentences leave no questions unanswered. For any sentence α, Γ "says" if α is true or false. The importance of complete sets extends beyond the proof of the completeness theorem. A theory which is complete and axiomatizable, for instance, is always decidable.

Complete consistent sets are important in the completeness proof since we can guarantee that every consistent set of sentences Γ is contained in a complete consistent set Γ^*. A complete consistent set contains, for each sentence α, either α or its negation $\neg\alpha$, but not both. This is true in particular for propositional variables, so from a complete consistent set, we can construct a valuation where the truth value assigned to propositional variables is defined according to which propositional variables are in Γ^*. This valuation can then be shown to make all sentences in Γ^* (and hence also all those in Γ) true. The proof of this latter fact requires that $\neg\alpha \in \Gamma^*$ iff $\alpha \notin \Gamma^*$, $(\alpha \vee \beta) \in \Gamma^*$ iff $\alpha \in \Gamma^*$ or $\beta \in \Gamma^*$, etc.

In what follows, we will often tacitly use the properties of reflexivity, monotonicity, and transitivity of $\vdash$ (see section 3.3).

Proposition 42B. *Suppose Γ is complete and consistent. Then:*

1. *If $\Gamma \vdash \alpha$, then $\alpha \in \Gamma$.*
2. *$\alpha \to \beta \in \Gamma$ iff either $\alpha \notin \Gamma$ or $\beta \in \Gamma$.*

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

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

   Suppose that $\Gamma \vdash \alpha$. Suppose to the contrary that $\alpha \notin \Gamma$. Since Γ is complete, $\neg\alpha \in \Gamma$. By Proposition 35C, Γ is inconsistent. This contradicts the assumption that Γ is consistent. Hence, it cannot be the case that $\alpha \notin \Gamma$, so $\alpha \in \Gamma$.

2. For the forward direction, suppose $\alpha \to \beta \in \Gamma$, and suppose to the contrary that $\alpha \in \Gamma$ and $\beta \notin \Gamma$. On these assumptions, $\alpha \to \beta \in \Gamma$ and $\alpha \in \Gamma$. By Proposition 36C, item (1), $\Gamma \vdash \beta$. But then by (1), $\beta \in \Gamma$, contradicting the assumption that $\beta \notin \Gamma$.

   For the reverse direction, first consider the case where $\alpha \notin \Gamma$. Since Γ is complete, $\neg\alpha \in \Gamma$. By Proposition 36C, item (2), $\Gamma \vdash \alpha \to \beta$. Again by (1), we get that $\alpha \to \beta \in \Gamma$, as required.

   Now consider the case where $\beta \in \Gamma$. By Proposition 36C, item (2) again, $\Gamma \vdash \alpha \to \beta$. By (1), $\alpha \to \beta \in \Gamma$. □

§4.3 Lindenbaum's Lemma

We now prove a lemma that shows that any consistent set of sentences is contained in some set of sentences which is not just consistent, but also complete. The proof works by adding one sentence at a time, guaranteeing at each step that the set remains consistent. We do this so that for every α, either α or $\neg\alpha$ gets added at some stage. The union of all stages in that construction then contains either α or its negation $\neg\alpha$ and is thus complete. It is also consistent, since we made sure at each stage not to introduce an inconsistency.

Lemma 43A (Lindenbaum's Lemma). *Every consistent set Γ in a language $\mathcal{L}$ can be extended to a complete and consistent set Γ^*.*

Proof. Let Γ be consistent. Let α_0, α_1, ... be an enumeration of all the sentences of $\mathcal{L}$. Define $\Gamma_0 = \Gamma$, and

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

Let $\Gamma^* = \bigcup_{n \geq 0} \Gamma_n$.

Each Γ_n is consistent: Γ_0 is consistent by definition. If $\Gamma_{n+1} = \Gamma_n \cup \{\alpha_n\}$, this is because the latter is consistent. If it isn't, $\Gamma_{n+1} = \Gamma_n \cup \{\neg\alpha_n\}$. We have to verify that $\Gamma_n \cup \{\neg\alpha_n\}$ is consistent. Suppose it's not. Then *both* $\Gamma_n \cup \{\alpha_n\}$ and $\Gamma_n \cup \{\neg\alpha_n\}$ are inconsistent. This means that Γ_n would be inconsistent by Proposition 35D, contrary to the induction hypothesis.

For every n and every $i < n$, $\Gamma_i \subseteq \Gamma_n$. This follows by a simple induction on n. For $n = 0$, there are no $i < 0$, so the claim holds automatically. For the inductive step, suppose it is true for n. We have $\Gamma_{n+1} = \Gamma_n \cup \{\alpha_n\}$ or $= \Gamma_n \cup \{\neg\alpha_n\}$ by construction. So $\Gamma_n \subseteq \Gamma_{n+1}$. If $i < n$, then $\Gamma_i \subseteq \Gamma_n$ by inductive hypothesis, and so $\subseteq \Gamma_{n+1}$ by transitivity of $\subseteq$.

From this it follows that every finite subset of Γ^* is a subset of Γ_n for some n, since each $\beta \in \Gamma^*$ not already in Γ_0 is added at some stage i. If n is the last one of these, then all β in the finite subset are in Γ_n. So, every finite subset of Γ^* is consistent. By Proposition 33H, Γ^* is consistent.

Every sentence of $\mathrm{Frm}(\mathcal{L})$ appears on the list used to define Γ^*. If $\alpha_n \notin \Gamma^*$, then that is because $\Gamma_n \cup \{\alpha_n\}$ was inconsistent. But then $\neg\alpha_n \in \Gamma^*$, so Γ^* is complete. □

§4.4 Construction of a Model

We are now ready to define a valuation that makes all $\alpha \in \Gamma$ true. To do this, we first apply Lindenbaum's Lemma: we get a complete consistent $\Gamma^* \supseteq \Gamma$. We let the propositional variables in Γ^* determine $\mathfrak{v}(\Gamma^*)$.

Definition 44A. *Suppose Γ^* is a complete consistent set of wffs. Then we let*

$$\mathfrak{v}(\Gamma^*)(p) = \begin{cases} \mathbb{T} & \text{if } p \in \Gamma^* \\ \mathbb{F} & \text{if } p \notin \Gamma^* \end{cases}$$

Lemma 44B (Truth Lemma). *$\mathfrak{v}(\Gamma^*) \vDash \alpha$ iff $\alpha \in \Gamma^*$.*

Proof. We prove both directions simultaneously, and by induction on α.

1. $\alpha \equiv p$: $\mathfrak{v}(\Gamma^*) \vDash p$ iff $\mathfrak{v}(\Gamma^*)(p) = \mathbb{T}$ (by the definition of satisfaction) iff $p \in \Gamma^*$ (by the construction of $\mathfrak{v}(\Gamma^*)$).

2. $\alpha \equiv \neg\beta$: $\mathfrak{v}(\Gamma^*) \vDash \alpha$ iff $\mathfrak{v}(\Gamma^*) \nvDash \beta$ (by definition of satisfaction). By induction hypothesis, $\mathfrak{v}(\Gamma^*) \nvDash \beta$ iff $\beta \notin \Gamma^*$. Since Γ^* is consistent and complete, $\beta \notin \Gamma^*$ iff $\neg\beta \in \Gamma^*$.

3. $\alpha \equiv \beta \to \gamma$: $\mathfrak{v}(\Gamma^*) \vDash \alpha$ iff $\mathfrak{v}(\Gamma^*) \nvDash \beta$ or $\mathfrak{v}(\Gamma^*) \vDash \gamma$ (by definition of satisfaction) iff $\beta \notin \Gamma^*$ or $\gamma \in \Gamma^*$ (by induction hypothesis). This is the case iff $(\beta \to \gamma) \in \Gamma^*$ (by Proposition 42B(2)).

§4.5 The Completeness Theorem

Let's combine our results: we arrive at the completeness theorem.

Theorem 45A (Completeness Theorem). *Let Γ be a set of sentences. If Γ is consistent, it is satisfiable.*

Proof. Suppose Γ is consistent. By Lemma 43A, there is a $\Gamma^* \supseteq \Gamma$ which is consistent and complete. By Lemma 44B, $\mathfrak{v}(\Gamma^*) \vDash \alpha$ iff $\alpha \in \Gamma^*$. From this it follows in particular that for all $\alpha \in \Gamma$, $\mathfrak{v}(\Gamma^*) \vDash \alpha$, so Γ is satisfiable. □

Corollary 45B (Completeness Theorem, Second Version). *For all Γ and sentences α: if $\Gamma \vDash \alpha$ then $\Gamma \vdash \alpha$.*

Proof. Note that the Γ's in corollary 45B and Theorem 45A are universally quantified. To make sure we do not confuse ourselves, let us restate Theorem 45A using a different variable: for any set of sentences Δ, if Δ is consistent, it is satisfiable. By contraposition, if Δ is not satisfiable, then Δ is inconsistent. We will use this to prove the corollary.

Suppose that $\Gamma \vDash \alpha$. Then $\Gamma \cup \{\neg\alpha\}$ is unsatisfiable by Proposition 15C. Taking $\Gamma \cup \{\neg\alpha\}$ as our Δ, the previous version of Theorem 45A gives us that $\Gamma \cup \{\neg\alpha\}$ is inconsistent. By Proposition 35B, $\Gamma \vdash \alpha$. □

§4.6 The Compactness Theorem

One important consequence of the completeness theorem is the compactness theorem. The compactness theorem states that if each *finite* subset of a set of sentences is satisfiable, the entire set is satisfiable—even if the set itself is infinite. This is far from obvious. There is nothing that seems to rule out, at first glance at least, the possibility of there being infinite sets of sentences which are contradictory, but the contradiction only arises, so to speak, from the infinite number. The compactness theorem says that such a scenario can be ruled out: there are no unsatisfiable infinite sets of sentences each finite subset of which is satisfiable. Like the completeness theorem, it has a version related to entailment: if an infinite set of sentences entails something, already a finite subset does.

Definition 46A. *A set Γ of wffs is* finitely satisfiable *iff every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable.*

Theorem 46B (Compactness Theorem). *The following hold for any sentences Γ and α:*

1. *$\Gamma \vDash \alpha$ iff there is a finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \vDash \alpha$.*
2. *Γ is satisfiable iff it is finitely satisfiable.*

Proof. We prove (2). If Γ is satisfiable, then there is a valuation $\mathfrak{v}$ such that $\mathfrak{v} \vDash \alpha$ for all $\alpha \in \Gamma$. Of course, this $\mathfrak{v}$ also satisfies every finite subset of Γ, so Γ is finitely satisfiable.

Now suppose that Γ is finitely satisfiable. Then every finite subset $\Gamma_0 \subseteq \Gamma$ is satisfiable. By soundness (corollary 37D), every finite subset is consistent. Then Γ itself must be consistent by Proposition 33H. By completeness (Theorem 45A), since Γ is consistent, it is satisfiable. □

§4.7 A Direct Proof of the Compactness Theorem

We can prove the Compactness Theorem directly, without appealing to the Completeness Theorem, using the same ideas as in the proof of the completeness theorem. In the proof of the Completeness Theorem we started with a consistent set Γ of sentences, expanded it to a consistent and complete set Γ^*

of sentences, and then showed that in the valuation $\mathfrak{v}(\Gamma^*)$ constructed from Γ^*, all sentences of Γ are true, so Γ is satisfiable.

We can use the same method to show that a finitely satisfiable set of sentences is satisfiable. We just have to prove the corresponding versions of the results leading to the truth lemma where we replace "consistent" with "finitely satisfiable."

Proposition 47A. *Suppose Γ is complete and finitely satisfiable. Then:*

1. *$(\alpha \to \beta) \in \Gamma$ iff either $\alpha \notin \Gamma$ or $\beta \in \Gamma$.*

Lemma 47B. *Every finitely satisfiable set Γ can be extended to a complete and finitely satisfiable set Γ^*.*

Theorem 47C (Compactness). *Γ is satisfiable if and only if it is finitely satisfiable.*

Proof. If Γ is satisfiable, then there is a valuation $\mathfrak{v}$ such that $\mathfrak{v} \vDash \alpha$ for all $\alpha \in \Gamma$. Of course, this $\mathfrak{v}$ also satisfies every finite subset of Γ, so Γ is finitely satisfiable.

Now suppose that Γ is finitely satisfiable. By Lemma 47B, Γ can be extended to a complete and finitely satisfiable set Γ^*. Construct the valuation $\mathfrak{v}(\Gamma^*)$ as in Definition 44A. The proof of the Truth Lemma (Lemma 44B) goes through if we replace references to Proposition 42B. □

Problems

Problem 1. Complete the proof of Proposition 42B.

Problem 2. Use corollary 45B to prove Theorem 45A, thus showing that the two formulations of the completeness theorem are equivalent.

Problem 3. 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 derivation 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 derivation were used in which results that lead up to the proof of Theorem 45A. Be sure to note any tacit uses of rules in these proofs.

Problem 4. Prove (1) of Theorem 46B.

Problem 5. Prove Proposition 47A. Avoid the use of $\vdash$.

Problem 6. Prove Lemma 47B. (Hint: the crucial step is to show that if Γ_n is finitely satisfiable, then either $\Gamma_n \cup \{\alpha_n\}$ or $\Gamma_n \cup \{\neg\alpha_n\}$ is finitely satisfiable.)

Problem 7. Write out the complete proof of the Truth Lemma (Lemma 44B) in the version required for the proof of Theorem 47C.

Part II

First-order Logic

Chapter 5

Introduction to First-Order Logic

§5.0 First-Order Logic

You are probably familiar with first-order logic from your first introduction to formal logic.[1] You may know it as "quantificational logic" or "predicate logic." First-order logic, first of all, is a formal language. That means, it has a certain vocabulary, and its expressions are strings from this vocabulary. But not every string is permitted. There are different kinds of permitted expressions: terms, wffs, and sentences. We are mainly interested in sentences of first-order logic: they provide us with a formal analogue of sentences of English, and about them we can ask the questions a logician typically is interested in. For instance:

- Does β follow from α logically?
- Is α logically true, logically false, or contingent?
- Are α and β equivalent?

These questions are primarily questions about the "meaning" of sentences of first-order logic. For instance, a philosopher would analyze the question of whether β follows logically from α as asking: is there a case where α is true but β is false (β doesn't follow from α), or does every case that makes α true also make β true (β does follow from α)? But we haven't been told yet what a "case" is—that is the job of *semantics*. The semantics of first-order logic provides a mathematically precise model of the philosopher's intuitive idea of "case," and also—and this is important—of what it is for a sentence α to be *true in* a case. We call the mathematically precise model that we will develop a structure. The relation which makes "true in" precise, is called the relation of *satisfaction*. So what we will define is "α is satisfied in $\mathfrak{A}$" (in symbols: $\models_{\mathfrak{A}} \alpha$) for sentences α and structures $\mathfrak{A}$. Once this is done, we can also give precise definitions of the other semantical terms such as "follows from" or "is logically

[1] In fact, we more or less assume you are! If you're not, you could review a more elementary textbook, such as *forall x* (**?**).

true." These definitions will make it possible to settle, again with mathematical precision, whether, e.g., $\forall x\,(\alpha(x) \to \beta(x)), \exists x\,\alpha(x) \vDash \exists x\,\beta(x)$. The answer will, of course, be "yes." If you've already been trained to symbolize sentences of English in first-order logic, you will recognize this as, e.g., the symbolizations of, say, "All ants are insects, there are ants, therefore there are insects." That is obviously a valid argument, and so our mathematical model of "follows from" for our formal language should give the same answer.

Another topic you probably remember from your first introduction to formal logic is that there are *derivations*. If you have taken a first formal logic course, your instructor will have made you practice finding such derivations, perhaps even a derivation that shows that the above entailment holds. There are many different ways to give derivations: you may have done something called "natural deduction" or "truth trees," but there are many others. The purpose of derivation systems is to provide tools using which the logicians' questions above can be answered: e.g., a natural deduction derivation in which $\forall x\,(\alpha(x) \to \beta(x))$ and $\exists x\,\alpha(x)$ are premises and $\exists x\,\beta(x)$ is the conclusion (last line) *verifies* that $\exists x\,\beta(x)$ logically follows from $\forall x\,(\alpha(x) \to \beta(x))$ and $\exists x\,\alpha(x)$.

But why is that? On the face of it, derivation systems have nothing to do with semantics: giving a formal derivation merely involves arranging symbols in certain rule-governed ways; they don't mention "cases" or "true in" at all. The connection between derivation systems and semantics has to be established by a meta-logical investigation. What's needed is a mathematical proof, e.g., that a formal derivation of $\exists x\,\beta(x)$ from premises $\forall x\,(\alpha(x) \to \beta(x))$ and $\exists x\,\alpha(x)$ is possible, if, and only if, $\forall x\,(\alpha(x) \to \beta(x))$ and $\exists x\,\alpha(x)$ together entail $\exists x\,\beta(x)$. Before this can be done, however, a lot of painstaking work has to be carried out to get the definitions of syntax and semantics correct.

§5.1 Syntax

We first must make precise what strings of symbols count as sentences of first-order logic. We'll do this later; for now we'll just proceed by example. The basic building blocks—the vocabulary—of first-order logic divides into two parts. The first part is the symbols we use to say specific things or to pick out specific things. We pick out things using constant symbols, and we say stuff about the things we pick out using predicate symbols. E.g, we might use a as a constant symbol to pick out a single thing, and then say something about it using the sentence Pa. If you have meanings for "a" and "P" in mind, you can read Pa as a sentence of English (and you probably have done so when you first learned formal logic). Once you have such simple sentences of first-order logic, you can build more complex ones using the second part of the vocabulary: the logical symbols (connectives and quantifiers). So, for instance, we can form expressions like $(Pa \wedge Qb)$ or $\exists x\,Px$.

In order to provide the precise definitions of semantics and the rules of our derivation systems required for rigorous meta-logical study, we first of all have to give a precise definition of what counts as a sentence of first-order

logic. The basic idea is easy enough to understand: there are some simple sentences we can form from just predicate symbols and constant symbols, such as Pa. And then from these we form more complex ones using the connectives and quantifiers. But what exactly are the rules by which we are allowed to form more complex sentences? These must be specified, otherwise we have not defined "sentence of first-order logic" precisely enough. There are a few issues. The first one is to get the right strings to count as sentences. The second one is to do this in such a way that we can give mathematical proofs about *all* sentences. Finally, we'll have to also give precise definitions of some rudimentary operations with sentences, such as "replace every x in α by b." The trouble is that the quantifiers and variables we have in first-order logic make it not entirely obvious how this should be done. E.g., should $\exists x\, Pa$ count as a sentence? What about $\exists x\, \exists x\, Px$? What should the result of "replace x by b in $(Px \wedge \exists x\, Px)$" be?

§5.2 Wffs

Here is the approach we will use to rigorously specify sentences of first-order logic and to deal with the issues arising from the use of variables. We first define a *different* set of expressions: wffs. Once we've done that, we can consider the role variables play in them—and on the basis of some other ideas, namely those of "free" and "bound" variables, we can define what a sentence is (namely, a wff without free variables). We do this not just because it makes the definition of "sentence" more manageable, but also because it will be crucial to the way we define the semantic notion of satisfaction.

Let's define "wff" for a simple first-order language, one containing only a single predicate symbol P and a single constant symbol a, and only the logical symbols $\neg$, $\wedge$, and $\exists$. Our full definitions will be much more general: we'll allow infinitely many predicate symbols and constant symbols. In fact, we will also consider function symbols which can be combined with constant symbols and variables to form "terms." For now, a and the variables will be our only terms. We do need infinitely many variables. We'll officially use the symbols v_0, v_1, ..., as variables.

Definition 52A. *The set of* wffs Frm *is defined as follows:*

1. *Pa and Pv_i are wffs ($i \in \mathbb{N}$).*

2. *If α is a wff, then $\neg\alpha$ is wff.*

(1) tells us that Pa and Pv_i are wffs, for any $i \in \mathbb{N}$. These are the so-called *atomic* wffs. They give us something to start from. The other clauses give us ways of forming new wffs from ones we have already formed. So for instance, by (2), we get that $\neg Pv_2$ is a wff, since Pv_2 is already a wff by (1). Then, by **??**, we get that $\exists v_2\, \neg Pv_2$ is another wff, and so on. **??** tells us that *only* strings we can form in this way count as wffs. In particular, $\exists v_0\, Pa$ and

$\exists v_0\, \exists v_0\, Pa$ *do* count as wffs, and $(\neg Pa)$ does not, because of the extraneous outer parentheses.

This way of defining wffs is called an *inductive definition*, and it allows us to prove things about wffs using a version of proof by induction called *structural induction*. These are discussed in a general way in **??** and **??**, which you should review before delving into the proofs later on. Basically, the idea is that if you want to give a proof that something is true for all wffs, you show first that it is true for the atomic wffs, and then that *if* it's true for any wff α (and β), it's *also* true for $\neg\alpha$, $(\alpha \wedge \beta)$, and $\exists x\, \alpha$. For instance, this proves that it's true for $\exists v_2\, \neg Pv_2$: from the first part you know that it's true for the atomic wff Pv_2. Then you get that it's true for $\neg Pv_2$ by the second part, and then again that it's true for $\exists v_2\, \neg Pv_2$ itself. Since all wffs are inductively generated from atomic wffs, this works for any of them.

§5.3 Satisfaction

We can already skip ahead to the semantics of first-order logic once we know what wffs are: here, the basic definition is that of a structure. For our simple language, a structure $\mathfrak{A}$ has just three components: a non-empty set $|\mathfrak{A}|$ called the *domain*, what a picks out in $\mathfrak{A}$, and what P is true of in $\mathfrak{A}$. The object picked out by a is denoted $a^{\mathfrak{A}}$ and the set of things P is true of by $P^{\mathfrak{A}}$. A structure $\mathfrak{A}$ consists of just these three things: $|\mathfrak{A}|$, $a^{\mathfrak{A}} \in |\mathfrak{A}|$ and $P^{\mathfrak{A}} \subseteq |\mathfrak{A}|$. The general case will be more complicated, since there will be many predicate symbols and constant symbols, the constant symbols can have more than one place, and there will also be function symbols.

This is enough to give a definition of satisfaction for wffs that don't contain variables. The idea is to give an inductive definition that mirrors the way we have defined wffs. We specify when an atomic formula is satisfied in $\mathfrak{A}$, and then when, e.g., $\neg\alpha$ is satisfied in $\mathfrak{A}$ on the basis of whether or not α is satisfied in $\mathfrak{A}$. E.g., we could define:

1. Pa is satisfied in $\mathfrak{A}$ iff $a^{\mathfrak{A}} \in P^{\mathfrak{A}}$.
2. $\neg\alpha$ is satisfied in $\mathfrak{A}$ iff α is not satisfied in $\mathfrak{A}$.
3. $(\alpha \wedge \beta)$ is satisfied in $\mathfrak{A}$ iff α is satisfied in $\mathfrak{A}$, and β is satisfied in $\mathfrak{A}$ as well.

Let's say that $|\mathfrak{A}| = \{0, 1, 2\}$, $a^{\mathfrak{A}} = 1$, and $P^{\mathfrak{A}} = \{1, 2\}$. This definition would tell us that Pa is satisfied in $\mathfrak{A}$ (since $a^{\mathfrak{A}} = 1 \in \{1, 2\} = P^{\mathfrak{A}}$). It tells us further that $\neg Pa$ is not satisfied in $\mathfrak{A}$, and that in turn $\neg\neg Pa$ is and $(\neg Pa \wedge Pa)$ is not satisfied, and so on.

The trouble comes when we want to give a definition for the quantifiers: we'd like to say something like, "$\exists v_0\, Pv_0$ is satisfied iff Pv_0 is satisfied." But the structure $\mathfrak{A}$ doesn't tell us what to do about variables. What we actually want to say is that Pv_0 is satisfied *for some value of* v_0. To make this precise we need a way to assign elements of $|\mathfrak{A}|$ not just to a but also to v_0. To this

end, we introduce variable *assignments*. A variable assignment is simply a function s that maps variables to elements of $|\mathfrak{A}|$ (in our example, to one of 1, 2, or 3). Since we don't know beforehand which variables might appear in a wff we can't limit which variables s assigns values to. The simple solution is to require that s assigns values to *all* variables v_0, v_1, ... We'll just use only the ones we need.

Instead of defining satisfaction of wffs just relative to a structure, we'll define it relative to a structure $\mathfrak{A}$ *and* a variable assignment s, and write $\models_{\mathfrak{A}} \alpha\,[s]$ for short. Our definition will now include an additional clause to deal with atomic wffs containing variables:

1. $\models_{\mathfrak{A}} Pa\,[s]$ iff $a^{\mathfrak{A}} \in P^{\mathfrak{A}}$.
2. $\models_{\mathfrak{A}} Pv_i\,[s]$ iff $s(v_i) \in P^{\mathfrak{A}}$.
3. $\models_{\mathfrak{A}} \neg\alpha\,[s]$ iff not $\models_{\mathfrak{A}} \alpha\,[s]$.
4. $\models_{\mathfrak{A}} (\alpha \wedge \beta)\,[s]$ iff $\models_{\mathfrak{A}} \alpha\,[s]$ and $\models_{\mathfrak{A}} \beta\,[s]$.

Ok, this solves one problem: we can now say when $\mathfrak{A}$ satisfies Pv_0 for the value $s(v_0)$. To get the definition right for $\exists v_0\, Pv_0$ we have to do one more thing: We want to have that $\models_{\mathfrak{A}} \exists v_0\, Pv_0\,[s]$ iff $\models_{\mathfrak{A}} Pv_0\,[s']$ for *some* way s' of assigning a value to v_0. But the value assigned to v_0 does not necessarily have to be the value that $s(v_0)$ picks out. We'll introduce a notation for that: if $m \in |\mathfrak{A}|$, then we let $s[m/v_0]$ be the assignment that is just like s (for all variables other than v_0), except to v_0 it assigns m. Now our definition can be:

5. $\models_{\mathfrak{A}} \exists v_i\, \alpha\,[s]$ iff $\models_{\mathfrak{A}} \alpha\,[s[m/v_i]]$ for some $m \in |\mathfrak{A}|$.

Does it work out? Let's say we let $s(v_i) = 0$ for all $i \in \mathbb{N}$. $\models_{\mathfrak{A}} \exists v_0\, Pv_0\,[s]$ iff there is an $m \in |\mathfrak{A}|$ so that $\models_{\mathfrak{A}} Pv_0\,[s[m/v_0]]$. And there is: we can choose $m = 1$ or $m = 2$. Note that this is true even if the value $s(v_0)$ assigned to v_0 by s itself—in this case, 0—doesn't do the job. We have $\models_{\mathfrak{A}} Pv_0\,[s[1/v_0]]$ but not $\models_{\mathfrak{A}} Pv_0\,[s]$.

If this looks confusing and cumbersome: it is. But the added complexity is required to give a precise, inductive definition of satisfaction for all wffs, and we need something like it to precisely define the semantic notions. There are other ways of doing it, but they are all equally (in)elegant.

§5.4 Sentences

Ok, now we have a (sketch of a) definition of satisfaction ("true in") for structures and wffs. But it needs this additional bit—a variable assignment—and what we wanted is a definition of sentences. How do we get rid of assignments, and what are sentences?

You probably remember a discussion in your first introduction to formal logic about the relation between variables and quantifiers. A quantifier is always followed by a variable, and then in the part of the sentence to which that

quantifier applies (its "scope"), we understand that the variable is "bound" by that quantifier. In wffs it was not required that every variable has a matching quantifier, and variables without matching quantifiers are "free" or "unbound." We will take sentences to be all those wffs that have no free variables.

Again, the intuitive idea of when an occurrence of a variable in a wff α is bound, which quantifier binds it, and when it is free, is not difficult to get. You may have learned a method for testing this, perhaps involving counting parentheses. We have to insist on a precise definition—and because we have defined wffs by induction, we can give a definition of the free and bound occurrences of a variable x in a wff α also by induction. E.g., it might look like this for our simplified language:

1. If α is atomic, all occurrences of x in it are free (that is, the occurrence of x in Px is free).

2. If α is of the form $\neg\beta$, then an occurrence of x in $\neg\beta$ is free iff the corresponding occurrence of x is free in β (that is, the free occurrences of variables in β are exactly the corresponding occurrences in $\neg\beta$).

3. If α is of the form $(\beta \wedge \gamma)$, then an occurrence of x in $(\beta \wedge \gamma)$ is free iff the corresponding occurrence of x is free in β or in γ.

4. If α is of the form $\exists x\, \beta$, then no occurrence of x in α is free; if it is of the form $\exists y\, \beta$ where y is a different variable than x, then an occurrence of x in $\exists y\, \beta$ is free iff the corresponding occurrence of x is free in β.

Once we have a precise definition of free and bound occurrences of variables, we can simply say: a sentence is any wff without free occurrences of variables.

§5.5 Semantic Notions

We mentioned above that when we consider whether $\models_{\mathfrak{A}} \alpha[s]$ holds, we (for convenience) let s assign values to all variables, but only the values it assigns to variables in α are used. In fact, it's only the values of *free* variables in α that matter. Of course, because we're careful, we are going to prove this fact. Since sentences have no free variables, s doesn't matter at all when it comes to whether or not they are satisfied in a structure. So, when α is a sentence we can define $\models_{\mathfrak{A}} \alpha$ to mean "$\models_{\mathfrak{A}} \alpha[s]$ for all s," which as it happens is true iff $\models_{\mathfrak{A}} \alpha[s]$ for at least one s. We need to introduce variable assignments to get a working definition of satisfaction for wffs, but for sentences, satisfaction is independent of the variable assignments.

Once we have a definition of "$\models_{\mathfrak{A}} \alpha$," we know what "case" and "true in" mean as far as sentences of first-order logic are concerned. On the basis of the definition of $\models_{\mathfrak{A}} \alpha$ for sentences we can then define the basic semantic notions of validity, entailment, and satisfiability. A sentence is valid, $\models \alpha$, if every structure satisfies it. It is entailed by a set of sentences, $\Gamma \models \alpha$, if every structure that satisfies all the sentences in Γ also satisfies α. And a set of

sentences is satisfiable if some structure satisfies all sentences in it at the same time.

Because wffs are inductively defined, and satisfaction is in turn defined by induction on the structure of wffs, we can use induction to prove properties of our semantics and to relate the semantic notions defined. We'll collect and prove some of these properties, partly because they are individually interesting, but mainly because many of them will come in handy when we go on to investigate the relation between semantics and derivation systems. In order to do so, we'll also have to define (precisely, i.e., by induction) some syntactic notions and operations we haven't mentioned yet.

§5.6 Substitution

We'll discuss an example to illustrate how things hang together, and how the development of syntax and semantics lays the foundation for our more advanced investigations later. Our derivation systems should let us derive Pa from $\forall v_0\, Pv_0$. Maybe we even want to state this as a rule of inference. However, to do so, we must be able to state it in the most general terms: not just for P, a, and v_0, but for any wff α, and term t, and variable x. (Recall that constant symbols are terms, but we'll consider also more complicated terms built from constant symbols and function symbols.) So we want to be able to say something like, "whenever you have derived $\forall x\, \alpha(x)$ you are justified in inferring $\alpha(t)$—the result of removing $\forall x$ and replacing x by t." But what exactly does "replacing x by t" mean? What is the relation between $\alpha(x)$ and $\alpha(t)$? Does this always work?

To make this precise, we define the operation of *substitution*. Substitution is actually tricky, because we can't just replace all x's in α by t, and not every t can be substituted for any x. We'll deal with this, again, using inductive definitions. But once this is done, specifying an inference rule as "infer $\alpha(t)$ from $\forall x\, \alpha(x)$" becomes a precise definition. Moreover, we'll be able to show that this is a good inference rule in the sense that $\forall x\, \alpha(x)$ entails $\alpha(t)$. But to prove this, we have to again prove something that may at first glance prompt you to ask "why are we doing this?" That $\forall x\, \alpha(x)$ entails $\alpha(t)$ relies on the fact that whether or not $\models_{\mathfrak{A}} \alpha(t)$ holds depends only on the value of the term t, i.e., if we let m be whatever element of $|\mathfrak{A}|$ is picked out by t, then $\models_{\mathfrak{A}} \alpha(t)\,[s]$ iff $\models_{\mathfrak{A}} \alpha(x)\,[s[m/x]]$. This holds even when t contains variables, but we'll have to be careful with how exactly we state the result.

§5.7 Models and Theories

Once we've defined the syntax and semantics of first-order logic, we can get to work investigating the properties of structures and the semantic notions. We can also define derivation systems, and investigate those. For a set of sentences, we can ask: what structures make all the sentences in that set true? Given a set of sentences Γ, a structure $\mathfrak{A}$ that satisfies them is called a *model of* Γ.

We might start from Γ and try to find its models—what do they look like? How big or small do they have to be? But we might also start with a single structure or collection of structures and ask: what sentences are true in them? Are there sentences that *characterize* these structures in the sense that they, and only they, are true in them? These kinds of questions are the domain of *model theory*. They also underlie the *axiomatic method*: describing a collection of structures by a set of sentences, the axioms of a theory. This is made possible by the observation that exactly those sentences entailed in first-order logic by the axioms are true in all models of the axioms.

As a very simple example, consider preorders. A preorder is a relation R on some set A which is both reflexive and transitive. A set A with a two-place relation $R \subseteq A \times A$ on it is exactly what we would need to give a structure for a first-order language with a single two-place relation symbol P: we would set $|\mathfrak{A}| = A$ and $P^{\mathfrak{A}} = R$. Since R is a preorder, it is reflexive and transitive, and we can find a set Γ of sentences of first-order logic that say this:

$$\forall v_0\, Pv_0v_0$$
$$\forall v_0\, \forall v_1\, \forall v_2\, ((Pv_0v_1 \wedge Pv_1v_2) \rightarrow Pv_0v_2)$$

These sentences are just the symbolizations of "for any x, Rxx" (R is reflexive) and "whenever Rxy and Ryz then also Rxz" (R is transitive). We see that a structure $\mathfrak{A}$ is a model of these two sentences Γ iff R (i.e., $P^{\mathfrak{A}}$), is a preorder on A (i.e., $|\mathfrak{A}|$). In other words, the models of Γ are exactly the preorders. Any property of all preorders that can be expressed in the first-order language with just P as predicate symbol (like reflexivity and transitivity above), is entailed by the two sentences in Γ and vice versa. So anything we can prove about models of Γ we have proved about all preorders.

For any particular theory and class of models (such as Γ and all preorders), there will be interesting questions about what can be expressed in the corresponding first-order language, and what cannot be expressed. There are some properties of structures that are interesting for all languages and classes of models, namely those concerning the size of the domain. One can always express, for instance, that the domain contains exactly n elements, for any $n \in \mathbb{Z}^+$. One can also express, using a set of infinitely many sentences, that the domain is infinite. But one cannot express that the domain is finite, or that the domain is non-enumerable. These results about the limitations of first-order languages are consequences of the compactness and Löwenheim-Skolem theorems.

§5.8 Soundness and Completeness

We'll also introduce derivation systems for first-order logic. There are many derivation systems that logicians have developed, but they all define the same derivability relation between sentences. We say that Γ *derives* α, $\Gamma \vdash \alpha$, if there is a derivation of a certain precisely defined sort. Derivations are always finite arrangements of symbols—perhaps a list of sentences, or some more complicated structure. The purpose of derivation systems is to provide

a tool to determine if a sentence is entailed by some set Γ. In order to serve that purpose, it must be true that $\Gamma \vDash \alpha$ if, and only if, $\Gamma \vdash \alpha$.

If $\Gamma \vdash \alpha$ but not $\Gamma \vDash \alpha$, our derivation system would be too strong, prove too much. The property that if $\Gamma \vdash \alpha$ then $\Gamma \vDash \alpha$ is called *soundness*, and it is a minimal requirement on any good derivation system. On the other hand, if $\Gamma \vDash \alpha$ but not $\Gamma \vdash \alpha$, then our derivation system is too weak, it doesn't prove enough. The property that if $\Gamma \vDash \alpha$ then $\Gamma \vdash \alpha$ is called *completeness*. Soundness is usually relatively easy to prove (by induction on the structure of derivations, which are inductively defined). Completeness is harder to prove.

Soundness and completeness have a number of important consequences. If a set of sentences Γ derives a contradiction (such as $\alpha \wedge \neg\alpha$) it is called *inconsistent*. Inconsistent Γs cannot have any models, they are unsatisfiable. From completeness the converse follows: any Γ that is not inconsistent—or, as we will say, *consistent*—has a model. In fact, this is equivalent to completeness, and is the form of completeness we will actually prove. It is a deep and perhaps surprising result: just because you cannot prove $\alpha \wedge \neg\alpha$ from Γ guarantees that there is a structure that is as Γ describes it. So completeness gives an answer to the question: which sets of sentences have models? Answer: all and only consistent sets do.

The soundness and completeness theorems have two important consequences: the compactness and the Löwenheim-Skolem theorem. These are important results in the theory of models, and can be used to establish many interesting results. We've already mentioned two: first-order logic cannot express that the domain of a structure is finite or that it is non-enumerable.

Historically, all of this—how to define syntax and semantics of first-order logic, how to define good derivation systems, how to prove that they are sound and complete, getting clear about what can and cannot be expressed in first-order languages—took a long time to figure out and get right. We now know how to do it, but going through all the details can still be confusing and tedious. But it's also important, because the methods developed here for the formal language of first-order logic are applied all over the place in logic, computer science, and linguistics. So working through the details pays off in the long run.

Chapter 6

Syntax of First-Order Logic

§6.0 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 wffs. Terms are formed from variables, constant symbols, and function symbols. Wffs, in turn, are formed from predicate symbols together with terms (these form the smallest, "atomic" wffs), and then from atomic wffs 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 wffs *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.

§6.1 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 *wffs* 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 equality symbol $=$, 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), $\to$ (conditional), $\forall$ (universal quantifier).
   b) The two-place equality symbol $=$.
   c) 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 6.1.1. 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 6.1.2. The language of set theory $\mathcal{L}_Z$ contains only the single two-place predicate symbol $\in$.

Example 6.1.3. 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 quantifier introduced above, we also use the following *defined* symbols: $\wedge$ (conjunction), $\vee$ (disjunction), $\leftrightarrow$ (biconditional), $\exists$ (existential quantifier), falsity $\bot$, 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 $\sim$, $\neg$, or ! for "negation", $\wedge$, $\cdot$, or & for "conjunction". Commonly used symbols for the "conditional" or "implication" are $\to$, $\Rightarrow$, 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.

§6.2 Terms and Wffs

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 *wffs*.

Definition 62A (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* $ft_1 \ldots t_n$ *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 $ft_1 \ldots t_n$ as just f by itself if $n = 0$.

Definition 62B (Formulas). *The set of* wffs $\mathrm{Frm}(\mathcal{L})$ *of the language* $\mathcal{L}$ *is defined inductively as follows:*

1. *If* R *is an* n*-place predicate symbol of* $\mathcal{L}$ *and* $t_1, \ldots, t_n$ *are terms of* $\mathcal{L}$*, then* $Rt_1 \ldots t_n$ *is an atomic wff.*
2. *If* t_1 *and* t_2 *are terms of* $\mathcal{L}$*, then* $= t_1 t_2$ *is an atomic wff.*

3. *If α is a wff, then $\neg\alpha$ is wff.*

4. *If α and β are wffs, then $(\alpha \to \beta)$ is a wff.*

5. *If α is a wff and x is a variable, then $\forall x\, \alpha$ is a wff.*

The definitions of the set of terms and that of wffs are *inductive definitions*. Essentially, we construct the set of wffs 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 $Rt_1 \dots t_n$ and $= t_1 t_2$. "Atomic wff" thus means any wff of this form.

The other cases of the definition give rules for constructing new wffs out of wffs already constructed. At the second stage, we can use them to construct wffs out of atomic wffs. At the third stage, we construct new formulas from the atomic formulas and those obtained in the second stage, and so on. A wff 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 $(\beta * \gamma)$ constructed from β, γ using a two-place connective $*$, we will often leave out the outermost pair of parentheses and write simply $\beta * \gamma$.

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

Definition 62C. *Formulas constructed using the defined operators are to be understood as follows:*

1. $\top$ *abbreviates* $(\alpha \vee \neg\alpha)$ *for some fixed atomic wff* α.

2. $\bot$ *abbreviates* $(\alpha \wedge \neg\alpha)$ *for some fixed atomic wff* α.

3. $\alpha \vee \beta$ *abbreviates* $\neg\alpha \to \beta$.

4. $\alpha \wedge \beta$ *abbreviates* $\neg(\alpha \to \neg\beta)$.

5. $\alpha \leftrightarrow \beta$ *abbreviates* $(\alpha \to \beta) \wedge (\beta \to \alpha)$.

6. $\exists x\, \alpha$ *abbreviates* $\neg\forall x\, \neg\alpha$.

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

Definition 62D (Syntactic identity). *The symbol $\equiv$ expresses syntactic identity between strings of symbols, i.e., $\alpha \equiv \beta$ iff α and β 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., $\alpha \equiv (\beta \vee \gamma)$ means: the string of symbols α is the same string as the one obtained by concatenating an opening parenthesis, the string β, the $\vee$ symbol, the string γ, and a closing parenthesis, in this order. If this is the case, then we know that the first symbol of α is an opening parenthesis, α contains β as a substring (starting at the second symbol), that substring is followed by $\vee$, etc.

As terms and wffs are built up from basic elements via inductive definitions, we can use the following induction principles to prove things about them.

Lemma 62E (*Principle of induction on terms*). *Let $\mathcal{L}$ be a first-order language. If some property P holds in all of the following cases, then $P(t)$ for every $t \in \mathrm{Trm}(\mathcal{L})$.*

1. *$P(v)$ for every variable v,*
2. *$P(a)$ for every constant symbol a of $\mathcal{L}$,*
3. *If $t_1, \ldots, t_n \in \mathrm{Trm}(\mathcal{L})$, f is an n-place function symbol of $\mathcal{L}$, and $P(t_1), \ldots, P(t_n)$, then $P(f(t_1, \ldots, t_n))$.*

Lemma 62F (*Principle of induction on wffs*). *Let $\mathcal{L}$ be a first-order language. If some property P holds for all the atomic wffs and is such that*

1. *α is an atomic formula.*
2. *it holds for $\neg\alpha$ whenever it holds for α;*
3. *it holds for $(\alpha \to \beta)$ whenever it holds for α and β;*
4. *it holds for $\forall x \alpha$ whenever it holds for α;*

then P holds for all formulas $\alpha \in \mathrm{Frm}(\mathcal{L})$.

§6.3 Unique Readability

The way we defined wffs guarantees that every wff has a *unique reading*, i.e., there is essentially only one way of constructing it according to our formation rules for wffs and only one way of "interpreting" it. If this were not so, we would have ambiguous wffs, i.e., wffs 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 wffs 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 α and β are wffs, then so is $\alpha \to \beta$.

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

1. We take δ to be α and $\delta \to \delta$ to be β.

2. We take α to be $\delta \to \delta$ and β is δ.

Correspondingly, there are two ways to "read" the wff $\delta \to \delta \to \delta$. It is of the form $\beta \to \gamma$ where β is δ and γ is $\delta \to \delta$, but *it is also* of the form $\beta \to \gamma$ with β being $\delta \to \delta$ and γ being δ.

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 $\beta \to \gamma$, the main operator is the indicated occurrence of $\to$. But if we can match the formula $\delta \to \delta \to \delta$ with $\beta \to \gamma$ 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 wff, i.e., every wff must have exactly one main operator occurrence.

Lemma 63A. *The number of left and right parentheses in a wff α are equal.*

Proof. We prove this by induction on the way α 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 $l(\alpha)$ be the number of left parentheses, and $r(\alpha)$ the number of right parentheses in α, and $l(t)$ and $r(t)$ similarly the number of left and right parentheses in a term t.

1. $\alpha \equiv Rt_1 \dots t_n$: $l(\alpha) = 1+l(t_1)+\dots+l(t_n) = 1+r(t_1)+\dots+r(t_n) = r(\alpha)$. Here we make use of the fact, left as an exercise, that $l(t) = r(t)$ for any term t.

2. $\alpha \equiv t_1 = t_2$: $l(\alpha) = l(t_1) + l(t_2) = r(t_1) + r(t_2) = r(\alpha)$.

3. $\alpha \equiv \neg\beta$: By induction hypothesis, $l(\beta) = r(\beta)$. Thus $l(\alpha) = l(\beta) = r(\beta) = r(\alpha)$.

4. $\alpha \equiv (\beta * \gamma)$: By induction hypothesis, $l(\beta) = r(\beta)$ and $l(\gamma) = r(\gamma)$. Thus $l(\alpha) = 1 + l(\beta) + l(\gamma) = 1 + r(\beta) + r(\gamma) = r(\alpha)$.

5. $\alpha \equiv \forall x\, \beta$: By induction hypothesis, $l(\beta) = r(\beta)$. Thus, $l(\alpha) = l(\beta) = r(\beta) = r(\alpha)$.

Definition 63B (Proper prefix). *A string of symbols β is a* proper prefix *of a string of symbols α if concatenating β and a non-empty string of symbols yields α.*

Lemma 63C. *If α is a wff, and β is a proper prefix of α, then β is not a wff.*

Proof. Exercise. □

Proposition 63D. *If α is an atomic wff, then it satisfies one, and only one of the following conditions.*

1. *$\alpha \equiv Rt_1 \dots t_n$ where R is an n-place predicate symbol, t_1, ..., t_n are terms, and each of R, t_1, ..., t_n is uniquely determined.*

2. *$\alpha \equiv t_1 = t_2$ where t_1 and t_2 are uniquely determined terms.*

Proof. Exercise. □

Proposition 63E (Unique Readability). *Every wff satisfies one, and only one of the following conditions.*

1. *α is atomic.*

2. *α is of the form $\neg\beta$.*

3. *α is of the form $(\beta \to \gamma)$.*

4. *α is of the form $\forall x\, \beta$.*

Moreover, in each case β, or β and γ, are uniquely determined. This means that, e.g., there are no different pairs β, γ and β', γ' so that α is both of the form $(\beta \to \gamma)$ and $(\beta' \to \gamma')$.

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

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

So suppose both $\alpha \equiv (\beta * \gamma)$ and $\alpha \equiv (\beta' *' \gamma')$. Then either $\beta \equiv \beta'$ or not. If it is, clearly $* = *'$ and $\gamma \equiv \gamma'$, since they then are substrings of α that begin in the same place and are of the same length. The other case is $\beta \not\equiv \beta'$. Since β and β' are both substrings of α that begin at the same place, one must be a proper prefix of the other. But this is impossible by Lemma 63C. □

§6.4 Main operator of a Formula

It is often useful to talk about the last operator used in constructing a wff α. This operator is called the *main operator* of α. Intuitively, it is the "outermost" operator of α. For example, the main operator of $\neg\alpha$ is $\neg$, the main operator of $(\alpha \vee \beta)$ is $\vee$, etc.

Definition 64A (Main operator). *The* main operator *of a wff α is defined as follows:*

1. *α is atomic: α has no main operator.*
2. *$\alpha \equiv \neg\beta$: the main operator of α is $\neg$.*
3. *$\alpha \equiv (\beta \to \gamma)$: the main operator of α is $\to$.*
4. *$\alpha \equiv \forall x\, \beta$: the main operator of α is $\forall$.*

In each case, we intend the specific indicated *occurrence* of the main operator in the formula. For instance, since the formula $((\delta \to \varphi) \to (\varphi \to \delta))$ is of the form $(\beta \to \gamma)$ where β is $(\delta \to \varphi)$ and γ is $(\varphi \to \delta)$, the second occurrence of $\to$ is the main operator.

This is a *recursive* definition of a function which maps all non-atomic wffs to their main operator occurrence. Because of the way wffs are defined inductively, every wff α satisfies one of the cases in Definition 64A. This guarantees that for each non-atomic wff α a main operator exists. Because each wff satisfies only one of these conditions, and because the smaller wffs from which α is constructed are uniquely determined in each case, the main operator occurrence of α is unique, and so we have defined a function.

We call wffs by the names in table 6.1 depending on which symbol their main operator is.Recall, however, that defined operators do not officially appear in wffs. They are just abbreviations, so officially they cannot be the main operator of a formula. In proofs about all wffs they therefore do not have to be treated separately.

Main operator	Type of wff	Example
none	atomic (wff)	$Rt_1 \dots t_n$, $t_1 = t_2$
$\neg$	negation	$\neg\alpha$
$\wedge$	conjunction	$(\alpha \wedge \beta)$
$\vee$	disjunction	$(\alpha \vee \beta)$
$\to$	conditional	$(\alpha \to \beta)$
$\leftrightarrow$	biconditional	$(\alpha \leftrightarrow \beta)$
$\forall$	universal (wff)	$\forall x\, \alpha$
$\exists$	existential (wff)	$\exists x\, \alpha$

Table 6.1: Main operator and names of wffs

§6.5 Subformulas

It is often useful to talk about the wffs that "make up" a given wff. We call these its *subformulas*. Any wff counts as a subformula of itself; a subformula of α other than α itself is a *proper subformula*.

Definition 65A (Immediate Subformula). *If α is a wff, the* immediate subformulas *of α are defined inductively as follows:*

1. *Atomic wffs have no immediate subformulas.*
2. *$\alpha \equiv \neg\beta$: The only immediate subformula of α is β.*
3. *$\alpha \equiv (\beta * \gamma)$: The immediate subformulas of α are β and γ ($*$ is any one of the two-place connectives).*
4. *$\alpha \equiv \forall x\, \beta$: The only immediate subformula of α is β.*

Definition 65B (Proper Subformula). *If α is a wff, the* proper subformulas *of α are defined recursively as follows:*

1. *Atomic wffs have no proper subformulas.*
2. *$\alpha \equiv \neg\beta$: The proper subformulas of α are β together with all proper subformulas of β.*
3. *$\alpha \equiv (\beta * \gamma)$: The proper subformulas of α are β, γ, together with all proper subformulas of β and those of γ.*
4. *$\alpha \equiv \forall x\, \beta$: The proper subformulas of α are β together with all proper subformulas of β.*

Definition 65C (Subformula). *The subformulas of α are α 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 α for each possible form of α. It is an explicit definition by cases, and the cases mirror the inductive definition of the set of wffs. In the second case, we have also mirrored the way the set of all wffs is defined, but in each case we have also included the proper subformulas of the smaller wffs β, γ in addition to these wffs themselves. This makes the definition *recursive*. In general, a definition of a function on an inductively defined set (in our case, wffs) 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 $(\beta * \gamma)$ we only use the proper subformulas of the "earlier" wffs β and γ.

Proposition 65D. *Suppose β is a subformula of α and γ is a subformula of β. Then γ is a subformula of α. In other words, the subformula relation is transitive.*

Proposition 65E. *Suppose α is a formula with n connectives and quantifiers. Then α has at most $2n + 1$ subformulas.*

§6.6 Formation Sequences

Defining wffs via an inductive definition, and the complementary technique of proving properties of wffs via induction, is an elegant and efficient approach. However, it can also be useful to consider a more bottom-up, step-by-step approach to the construction of wffs, which we do here using the notion of a *formation sequence*. To show how terms and wffs can be introduced in this way without needing to refer to their inductive definitions, we first introduce the notion of an arbitrary string of symbols drawn from some language $\mathcal{L}$.

Definition 66A (Strings). *Suppose $\mathcal{L}$ is a first-order language. An* $\mathcal{L}$-string *is a finite sequence of symbols of $\mathcal{L}$. Where the language $\mathcal{L}$ is clearly fixed by the context, we will often refer to a $\mathcal{L}$-string as a* string *simpliciter.*

Example 6.6.2. For any first-order language $\mathcal{L}$, all $\mathcal{L}$-wffs are $\mathcal{L}$-strings, but not conversely. For example,

$$)(v_0 \to \exists$$

is an $\mathcal{L}$-string but not an $\mathcal{L}$-wff.

Definition 66C (Formation sequences for terms). *A finite sequence of $\mathcal{L}$-strings $\langle t_0, \ldots, t_n\rangle$ is a* formation sequence *for a term t if $t \equiv t_n$ and for all $i \leq n$, either t_i is a variable or a constant symbol, or $\mathcal{L}$ contains a k-ary function symbol f and there exist $m_0, \ldots, m_k < i$ such that $t_i \equiv f(t_{m_0}, \ldots, t_{m_k})$.*

Example 6.6.4. The sequence

$$\langle c_0, v_0, f_0^2 c_0 v_0, f_0^1 f_0^2 c_0 v_0\rangle$$

is a formation sequence for the term $f_0^1 f_0^2 c_0 v_0$, as is

$$\langle v_0, c_0, f_0^2 c_0 v_0, f_0^1 f_0^2 c_0 v_0\rangle.$$

Definition 66E (Formation sequences for formulas). *A finite sequence of $\mathcal{L}$-strings $\langle \alpha_0, \ldots, \alpha_n\rangle$ is a* formation sequence *for α if $\alpha \equiv \alpha_n$ and for all $i \leq n$, either α_i is an atomic wff or there exist $j, k < i$ and a variable x such that one of the following holds:*

1. $\alpha_i \equiv \neg\alpha_j$.
2. $\alpha_i \equiv (\alpha_j \to \alpha_k)$.

3. $\alpha_i \equiv \forall x\, \alpha_j$.

Example 6.6.6.

$$\langle A^1_0 v_0, A^1_1 c_1, (A^1_1 c_1 \land A^1_0 v_0), \exists v_0\, (A^1_1 c_1 \land A^1_0 v_0)\rangle$$

is a formation sequence of $\exists v_0\, (A^1_1 c_1 \land A^1_0 v_0)$, as is

$$\langle A^1_0 v_0, A^1_1 c_1, (A^1_1 c_1 \land A^1_0 v_0), A^1_1 c_1, \forall v_1\, A^1_0 v_0, \exists v_0\, (A^1_1 c_1 \land A^1_0 v_0)\rangle.$$

As can be seen from the second example, formation sequences may contain "junk": wffs which are redundant or do not contribute to the construction.

Proposition 66G. *Every wff α in* $\mathrm{Frm}(\mathcal{L})$ *has a formation sequence.*

Proof. Suppose α is atomic. Then the sequence $\langle\alpha\rangle$ is a formation sequence for α. Now suppose that β and γ have formation sequences $\langle\beta_0, \ldots, \beta_n\rangle$ and $\langle\gamma_0, \ldots, \gamma_m\rangle$ respectively.

1. If $\alpha \equiv \neg\beta$, then $\langle\beta_0, \ldots, \beta_n, \neg\beta_n\rangle$ is a formation sequence for α.
2. If $\alpha \equiv (\beta \to \gamma)$, then $\langle\beta_0, \ldots, \beta_n, \gamma_0, \ldots, \gamma_m, (\beta_n \to \gamma_m)\rangle$ is a formation sequence for α.
3. If $\alpha \equiv \forall x\, \beta$, then $\langle\beta_0, \ldots, \beta_n, \forall x\, \beta_n\rangle$ is a formation sequence for α.

By the principle of induction on wffs, every wff has a formation sequence. □

We can also prove the converse. This is important because it shows that our two ways of defining formulas are equivalent: they give the same results. It also means that we can prove theorems about formulas by using ordinary induction on the length of formation sequences.

Lemma 66H. *Suppose that $\langle\alpha_0, \ldots, \alpha_n\rangle$ is a formation sequence for α_n, and that $k \leq n$. Then $\langle\alpha_0, \ldots, \alpha_k\rangle$ is a formation sequence for α_k.*

Proof. Exercise. □

Theorem 66I. $\mathrm{Frm}(\mathcal{L})$ *is the set of all expressions (strings of symbols) in the language $\mathcal{L}$ with a formation sequence.*

Proof. Let F be the set of all strings of symbols in the language $\mathcal{L}$ that have a formation sequence. We have seen in Proposition 66G that $\mathrm{Frm}(\mathcal{L}) \subseteq F$, so now we prove the converse.

Suppose α has a formation sequence $\langle\alpha_0, \ldots, \alpha_n\rangle$. We prove that $\alpha \in \mathrm{Frm}(\mathcal{L})$ by strong induction on n. Our induction hypothesis is that every string of symbols with a formation sequence of length $m < n$ is in $\mathrm{Frm}(\mathcal{L})$. By the definition of a formation sequence, either α_n is atomic or there must exist $j, k < n$ such that one of the following is the case:

1. $\alpha_i \equiv \neg\alpha_j$.

2. $\alpha_i \equiv (\alpha_j \to \alpha_k)$.

3. $\alpha_i \equiv \forall x\, \alpha_j$.

Now we reason by cases. If α_n is atomic then $\alpha_n \in \mathrm{Frm}(\mathcal{L}_0)$. Suppose instead that $\alpha \equiv (\alpha_j \wedge \alpha_k)$. By Lemma 66H, $\langle \alpha_0, \ldots, \alpha_j \rangle$ and $\langle \alpha_0, \ldots, \alpha_k \rangle$ are formation sequences for α_j and α_k, respectively. Since these are proper initial subsequences of the formation sequence for α, they both have length less than n. Therefore by the induction hypothesis, α_j and α_k are in $\mathrm{Frm}(\mathcal{L}_0)$, and by the definition of a wff, so is $(\alpha_j \wedge \alpha_k)$. The other cases follow by parallel reasoning. □

Formation sequences for terms have similar properties to those for wffs.

Proposition 66J. $\mathrm{Trm}(\mathcal{L})$ *is the set of all expressions t in the language $\mathcal{L}$ such that there exists a (term) formation sequence fo t.*

Proof. Exercise. □

There are two types of "junk" that can appear in formation sequences: repeated elements, and elements that are irrelevant to the construction of the formation or term. We can eliminate both by looking at minimal formation sequences.

Definition 66K (Minimal formation sequences). *A formation sequence $\langle \alpha_0, \ldots, \alpha_n \rangle$ for α is a* minimal formation sequence *for α if for every other formation sequence s for α, the length of s is greater than or equal to $n + 1$.*

Proposition 66L. *The following are equivalent:*

1. *β is a sub-wff of α.*

2. *β occurs in every formation sequence of α.*

3. *β occurs in a minimal formation sequence of α.*

Proof. Exercise. □

Formation sequences were introduced by Raymond Smullyan in his textbook *First-Order Logic* (**?**). Additional properties of formation sequences were established by **?**.

§6.7 Free Variables and Sentences

Definition 67A (Free occurrences of a variable). *The* free *occurrences of a variable in a wff are defined inductively as follows:*

1. *α is atomic: all variable occurrences in α are free.*
2. *$\alpha \equiv \neg\beta$: the free variable occurrences of α are exactly those of β.*
3. *$\alpha \equiv (\beta * \gamma)$: the free variable occurrences of α are those in β together with those in γ.*
4. *$\alpha \equiv \forall x\, \beta$: the free variable occurrences in α are all of those in β except for occurrences of x.*

Definition 67B (Bound Variables). *An occurrence of a variable in a formula α is* bound *if it is not free.*

Definition 67C (Scope). *If $\forall x\, \beta$ is an occurrence of a subformula in a formula α, then the corresponding occurrence of β in α is called the* scope *of the corresponding occurrence of $\forall x$.*

If β is the scope of a quantifier occurrence $\forall x$ in α, then the free occurrences of x in β are bound in $\forall x\, \beta$. We say that these occurrences are bound by *the mentioned quantifier occurrence.*

Example 6.7.4. Consider the following formula:

$$\exists v_0 \underbrace{A_0^2 v_0 v_1}_{\beta}$$

β represents the scope of $\exists v_0$. The quantifier binds the occurrence of v_0 in β, but does not bind the occurrence 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 wff α:

$$\forall v_0 \underbrace{(A_0^1 v_0 \to A_0^2 v_0 v_1)}_{\beta} \to \exists v_1 \underbrace{(A_1^2 v_0 v_1 \vee \forall v_0 \overbrace{\neg A_1^1 v_0}^{\delta})}_{\gamma}$$

β is the scope of the first $\forall v_0$, γ is the scope of $\exists v_1$, and δ is the scope of the second $\forall v_0$. The first $\forall v_0$ binds the occurrences of v_0 in β, $\exists v_1$ binds the occurrence of v_1 in γ, and the second $\forall v_0$ binds the occurrence of v_0 in δ. The first occurrence of v_1 and the fourth occurrence of v_0 are free in α. The last occurrence of v_0 is free in δ, but bound in γ and α.

Definition 67E (Sentence). *A wff α is a* sentence *iff it contains no free occurrences of variables.*

§6.8 Substitution

Definition 68A (Substitution in a term). *We define $s[t/x]$, the result of* substituting *t for every occurrence of x in s, recursively:*

1. *$s \equiv c$: $s[t/x]$ is just s.*
2. *$s \equiv y$: $s[t/x]$ is also just s, provided y is a variable and $y \not\equiv x$.*
3. *$s \equiv x$: $s[t/x]$ is t.*
4. *$s \equiv ft_1 \dots t_n$: $s[t/x]$ is $ft_1[t/x] \dots t_n[t/x]$.*

Definition 68B. *A term t is* free for *x in α if none of the free occurrences of x in α occur in the scope of a quantifier that binds a variable in t.*

Example 6.8.3.

1. v_8 is free for v_1 in $\exists v_3 A_4^2 v_3 v_1$
2. $f_1^2(v_1, v_2)$ is *not* free for v_0 in $\forall v_2 A_4^2 v_0 v_2$

Definition 68D (Substitution in a wff). *If α is a wff, x is a variable, and t is a term free for x in α, then $\alpha[t/x]$ is the result of substituting t for all free occurrences of x in α.*

1. *$\alpha \equiv Pt_1 \dots t_n$: $\alpha[t/x]$ is $Pt_1[t/x] \dots t_n[t/x]$.*
2. *$\alpha \equiv t_1 = t_2$: $\alpha[t/x]$ is $t_1[t/x] = t_2[t/x]$.*
3. *$\alpha \equiv \neg\beta$: $\alpha[t/x]$ is $\neg\beta[t/x]$.*
4. *$\alpha \equiv (\beta \to \gamma)$: $\alpha[t/x]$ is $(\beta[t/x] \to \gamma[t/x])$.*
5. *$\alpha \equiv \forall y\, \beta$: $\alpha[t/x]$ is $\forall y\, \beta[t/x]$, provided y is a variable other than x; otherwise $\alpha[t/x]$ is just α.*

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

The restriction that t must be free for x in α is necessary to exclude cases like the following. If $\alpha \equiv \exists y\, x < y$ and $t \equiv y$, then $\alpha[t/x]$ 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 $\forall x\, \beta$ holds, so does $\beta[t/x]$. But consider $\forall x \exists y\, x < y$ (here β is $\exists y\, x < y$). It is a 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 $\beta[y/x] \equiv \exists y\, y < y$, 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 α.

We often use the following convention to avoid cumbersome notation: If α is a wff which may contain the variable x free, we also write $\alpha(x)$ to indicate

this. When it is clear which α and x we have in mind, and t is a term (assumed to be free for x in $\alpha(x)$), then we write $\alpha(t)$ as short for $\alpha[t/x]$. So for instance, we might say, "we call $\alpha(t)$ an instance of $\forall x\, \alpha(x)$." By this we mean that if α is any wff, x a variable, and t a term that's free for x in α, then $\alpha[t/x]$ is an instance of $\forall x\, \alpha$.

Problems

Problem 1. Prove Lemma 62E.

Problem 2. Prove that for any term t, $l(t) = r(t)$.

Problem 3. Prove Lemma 63C.

Problem 4. Prove Proposition 63D (Hint: Formulate and prove a version of Lemma 63C for terms.)

Problem 5. Prove Proposition 65D.

Problem 6. Prove Proposition 65E.

Problem 7. Prove Lemma 66H.

Problem 8. Prove Proposition 66J. Hint: use a similar strategy to that used in the proof of Theorem 66I.

Problem 9. Prove Proposition 66L.

Problem 10. Give an inductive definition of the bound variable occurrences along the lines of Definition 67A.

Chapter 7

Semantics of First-Order Logic

§7.0 Introduction

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 wffs, 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 wff may be satisfied in a structure $\mathfrak{A}$ relative to a variable assignment s, written as $\models_{\mathfrak{A}} \alpha\,[s]$. This relation is also defined by induction on the structure of α, using the truth tables for the logical connectives to define, say, satisfaction of $(\alpha \wedge \beta)$ in terms of satisfaction (or not) of α and β. It then turns out that the variable assignment is irrelevant if the wff α 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 $\models_{\mathfrak{A}} \alpha$ for sentences we can then define the basic semantic notions of validity, entailment, and satisfiability. A sentence is valid, $\models \alpha$, if every structure satisfies it. It is entailed by a set of sentences, $\Gamma \models \alpha$, if every structure that satisfies all the sentences in Γ also satisfies α. And a set of sentences is satisfiable if some structure satisfies all sentences in it at the same time. Because wffs are inductively defined, and satisfaction is in turn defined by induction on the structure of wffs, we can use induction to prove properties of our semantics and to relate the semantic notions defined.

§7.1 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, satisfiability. They are variously called "structures," "interpretations," or "models" in the literature.

Definition 71A (Structures). *A* structure $\mathfrak{A}$*, for a language* $\mathcal{L}$ *of first-order logic consists of the following elements:*

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

Example 7.1.2. A structure $\mathfrak{A}$ for the language of arithmetic consists of a set, an element of $|\mathfrak{A}|$, $0^{\mathfrak{A}}$, as interpretation of the constant symbol 0, a one-place function $\prime^{\mathfrak{A}} \colon |\mathfrak{A}| \to |\mathfrak{A}|$, two two-place functions $+^{\mathfrak{A}}$ and $\times^{\mathfrak{A}}$, both $|\mathfrak{A}|^2 \to |\mathfrak{A}|$, and a two-place relation $<^{\mathfrak{A}} \subseteq |\mathfrak{A}|^2$.

An obvious example of such a structure is the following:

1. $|\mathfrak{B}| = \mathbb{N}$
2. $0^{\mathfrak{B}} = 0$
3. $\prime^{\mathfrak{B}}(n) = n + 1$ for all $n \in \mathbb{N}$
4. $+^{\mathfrak{B}}(n, m) = n + m$ for all $n, m \in \mathbb{N}$
5. $\times^{\mathfrak{B}}(n, m) = n \cdot m$ for all $n, m \in \mathbb{N}$
6. $<^{\mathfrak{B}} = \{\langle n, m\rangle : n \in \mathbb{N}, m \in \mathbb{N}, n < m\}$

The structure $\mathfrak{B}$ 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, $\prime$, $+$, $\times$, $<$ accordingly. But we can also define structures for $\mathcal{L}_A$ which have nothing even remotely to do with numbers.

Example 7.1.3. A structure $\mathfrak{A}$ 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 HF of *hereditarily finite sets*:

1. $|HF| = \emptyset \cup \wp(\emptyset) \cup \wp(\wp(\emptyset)) \cup \wp(\wp(\wp(\emptyset))) \cup \ldots$;
2. $\in^{HF} = \{\langle x, y\rangle : x, y \in |HF|, x \in y\}$.

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 $\exists x\,(\alpha(x) \vee \neg\alpha(x))$ 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: $\alpha(a)$, therefore $\exists x\,\alpha(x)$. 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: $\alpha(a)$ and $\exists x\, x = a$, therefore $\exists x\,\alpha(x)$.

§7.2 Covered Structures for First-order Languages

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

Definition 72A (Value of closed terms). *If t is a closed term of the language $\mathcal{L}$ and $\mathfrak{A}$ is a structure for $\mathcal{L}$, the* value $t^{\mathfrak{A}}$ *is defined as follows:*

1. *If t is just the constant symbol c, then $c^{\mathfrak{A}} = c^{\mathfrak{A}}$.*
2. *If t is of the form $ft_1 \ldots t_n$, then*

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

Definition 72B (Covered structure). *A structure is* covered *if every element of the domain is the value of some closed term.*

Example 7.2.3. Let $\mathcal{L}$ be the language with constant symbols *zero*, *one*, *two*, ..., the binary predicate symbol $<$, and the binary function symbols $+$ and $\times$. Then a structure $\mathfrak{A}$ for $\mathcal{L}$ is the one with domain $|\mathfrak{A}| = \{0, 1, 2, \ldots\}$ and assignments $zero^{\mathfrak{A}} = 0$, $one^{\mathfrak{A}} = 1$, $two^{\mathfrak{A}} = 2$, and so forth. For the binary relation symbol $<$, the set $<^{\mathfrak{A}}$ is the set of all pairs $\langle c_1, c_2\rangle \in |\mathfrak{A}|^2$ such that c_1 is less than c_2: for example, $\langle 1, 3\rangle \in <^{\mathfrak{A}}$ but $\langle 2, 2\rangle \notin <^{\mathfrak{A}}$. For the binary function symbol $+$, define $+^{\mathfrak{A}}$ in the usual way—for example, $+^{\mathfrak{A}}(2, 3)$ 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

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

§7.3 Satisfaction of a Wff in a Structure

The basic notion that relates expressions such as terms and wffs, on the one hand, and structures on the other, are those of *value* of a term and *satisfaction* of a wff. 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 wff is *satisfied* in a structure if the interpretation given to the predicates makes the wff true in the domain of the structure. This notion of satisfaction is specified inductively: the specification of the structure directly states when atomic wffs are satisfied, and we define when a complex wff 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 wff 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 73A (Variable Assignment). *A* variable assignment *s for a structure $\mathfrak{A}$ is a function which maps each variable to an element of $|\mathfrak{A}|$, i.e., $s\colon \mathrm{Var} \to |\mathfrak{A}|$.*

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 73B (Value of Terms). *If t is a term of the language $\mathcal{L}$, $\mathfrak{A}$ is a structure for $\mathcal{L}$, and s is a variable assignment for $\mathfrak{A}$, the* value $\bar{s}(t)$ *is defined as follows:*

1. $t \equiv c$: $\bar{s}(t) = c^{\mathfrak{A}}$.
2. $t \equiv x$: $\bar{s}(t) = s(x)$.
3. $t \equiv ft_1 \dots t_n$:
$$\bar{s}(t) = f^{\mathfrak{A}}(\bar{s}(t_1), \dots, \bar{s}(t_n)).$$

Definition 73C (x-Variant). *If s is a variable assignment for a structure $\mathfrak{A}$, then any variable assignment s' for $\mathfrak{A}$ 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 73D. *If s is a variable assignment for a structure $\mathfrak{A}$ and $m \in |\mathfrak{A}|$, then the assignment $s[m/x]$ is the variable assignment defined by*

$$s[m/x](y) = \begin{cases} m & \text{if } y \equiv x \\ s(y) & \text{otherwise.} \end{cases}$$

In other words, $s[m/x]$ is the particular x-variant of s which assigns the domain element m to x, and assigns the same things to variables other than x that s does.

Definition 73E (Satisfaction). *Satisfaction of a wff α in a structure $\mathfrak{A}$ relative to a variable assignment s, in symbols: $\models_{\mathfrak{A}} \alpha\,[s]$, is defined recursively as follows. (We write $\not\models_{\mathfrak{A}} \alpha\,[s]$ to mean "not $\models_{\mathfrak{A}} \alpha\,[s]$.")*

1. $\alpha \equiv Rt_1 \dots t_n$: $\models_{\mathfrak{A}} \alpha\,[s]$ *iff* $\langle \bar{s}(t_1), \dots, \bar{s}(t_n) \rangle \in R^{\mathfrak{A}}$.
2. $\alpha \equiv t_1 = t_2$: $\models_{\mathfrak{A}} \alpha\,[s]$ *iff* $\bar{s}(t_1) = \bar{s}(t_2)$.
3. $\alpha \equiv \neg\beta$: $\models_{\mathfrak{A}} \alpha\,[s]$ *iff* $\not\models_{\mathfrak{A}} \beta\,[s]$.
4. $\alpha \equiv (\beta \to \gamma)$: $\models_{\mathfrak{A}} \alpha\,[s]$ *iff* $\not\models_{\mathfrak{A}} \beta\,[s]$ *or* $\models_{\mathfrak{A}} \gamma\,[s]$ *(or both).*
5. $\alpha \equiv \forall x\, \beta$: $\models_{\mathfrak{A}} \alpha\,[s]$ *iff for every element* $m \in |\mathfrak{A}|$, $\models_{\mathfrak{A}} \beta\,[s[m/x]]$.

The variable assignments are important in the last clause. We cannot define satisfaction of $\forall x\, \beta(x)$ by "for all $m \in |\mathfrak{A}|$, $\models_{\mathfrak{A}} \beta(m)$." The reason is that if $m \in |\mathfrak{A}|$, it is not a symbol of the language, and so $\beta(m)$ is not a wff (that is, $\beta[m/x]$ is undefined). We also cannot assume that we have constant symbols or terms available that name every element of $\mathfrak{A}$, since there is nothing in the definition of structures that requires it. In the standard language, the set of constant symbols is denumerable, so if $|\mathfrak{A}|$ is not enumerable there aren't even enough constant symbols to name every object.

We solve this problem by introducing variable assignments, which allow us to link variables directly with elements of the domain. Then instead of saying

that, e.g., $\forall x\, \beta(x)$ is satisfied in $\mathfrak{A}$ iff for all $m \in |\mathfrak{A}|$, $\models_{\mathfrak{A}} \beta(m)$, we say it is satisfied in $\mathfrak{A}$ *relative to* s iff $\beta(x)$ is satisfied relative to $s[m/x]$ for every $m \in |\mathfrak{A}|$.

Example 7.3.6. Let $\mathcal{L} = \{a, b, f, R\}$ where a and b are constant symbols, f is a two-place function symbol, and R is a two-place predicate symbol. Consider the structure $\mathfrak{A}$ defined by:

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

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

Then

$$\bar{s}(f(a, b)) = f^{\mathfrak{A}}(\bar{s}(a), \bar{s}(b)).$$

Since a and b are constant symbols, $\bar{s}(a) = a^{\mathfrak{A}} = 1$ and $\bar{s}(b) = b^{\mathfrak{A}} = 2$. So

$$\bar{s}(f(a, b)) = f^{\mathfrak{A}}(1, 2) = 1 + 2 = 3.$$

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

$$\bar{s}(f(f(a, b), a)) = f^{\mathfrak{A}}(\bar{s}(f(a, b)), \bar{s}(a)) = f^{\mathfrak{A}}(3, 1) = 3,$$

since $3 + 1 > 3$. Since $s(x) = 1$ and $\bar{s}(x) = s(x)$, we also have

$$\bar{s}(f(f(a, b), x)) = f^{\mathfrak{A}}(\bar{s}(f(a, b)), \bar{s}(x)) = f^{\mathfrak{A}}(3, 1) = 3,$$

An atomic wff $R(t_1, t_2)$ is satisfied if the tuple of values of its arguments, i.e., $\langle \bar{s}(t_1), \bar{s}(t_2)\rangle$, is an element of $R^{\mathfrak{A}}$. So, e.g., we have $\models_{\mathfrak{A}} R(b, f(a, b))\,[s]$ since $\langle b^{\mathfrak{A}}, f(a, b)^{\mathfrak{A}}\rangle = \langle 2, 3\rangle \in R^{\mathfrak{A}}$, but $\not\models_{\mathfrak{A}} R(x, f(a, b))\,[s]$ since $\langle 1, 3\rangle \notin R^{\mathfrak{A}}[s]$.

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

$$\begin{aligned} &\models_{\mathfrak{A}} R(a, a) \to (R(b, x) \vee R(x, b))\,[s] \text{ iff} \\ &\quad \not\models_{\mathfrak{A}} R(a, a)\,[s] \text{ or } \models_{\mathfrak{A}} R(b, x) \vee R(x, b)\,[s] \end{aligned}$$

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

$$\begin{aligned}&\models_{\mathfrak{A}} R(b,x) \vee R(x,b)\,[s] \text{ iff}\\ &\quad \models_{\mathfrak{A}} R(b,x)\,[s] \text{ or } \models_{\mathfrak{A}} R(x,b)\,[s]\end{aligned}$$

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

Recall that an x-variant of s is a variable assignment that differs from s at most in what it assigns to x. For every element of $|\mathfrak{A}|$, there is an x-variant of s:

$$\begin{aligned} s_1 &= s[1/x], & s_2 &= s[2/x],\\ s_3 &= s[3/x], & s_4 &= s[4/x].\end{aligned}$$

So, e.g., $s_2(x) = 2$ and $s_2(y) = s(y) = 1$ for all variables y other than x. These are all the x-variants of s for the structure $\mathfrak{A}$, since $|\mathfrak{A}| = \{1,2,3,4\}$. Note, in particular, that $s_1 = s$ (s is always an x-variant of itself).

To determine if a universally quantified wff $\forall x\,\alpha(x)$ is satisfied, we have to determine if $\models_{\mathfrak{A}} \alpha(x)\,[s[m/x]]$ for all $m \in |\mathfrak{A}|$. So,

$$\models_{\mathfrak{A}} \forall x\,(R(x,a) \to R(a,x))\,[s],$$

since $\models_{\mathfrak{A}} R(x,a) \to R(a,x)\,[s[m/x]]$ for all $m \in |\mathfrak{A}|$. For $m = 1$, we have $\models_{\mathfrak{A}} R(a,x)\,[s[1/x]]$ so the consequent is true; for $m = 2$, 3, and 4, we have $\not\models_{\mathfrak{A}} R(x,a)\,[s[m/x]]$, so the antecedent is false. But,

$$\not\models_{\mathfrak{A}} \forall x\,(R(a,x) \to R(x,a))\,[s]$$

since $\not\models_{\mathfrak{A}} R(a,x) \to R(x,a)\,[s[2/x]]$ (because $\models_{\mathfrak{A}} R(a,x)\,[s[2/x]]$ and $\not\models_{\mathfrak{A}} R(x,a)\,[s[2/x]]$).

To determine if an existentially quantified wff $\exists x\,\alpha(x)$ is satisfied, we have to determine if $\models_{\mathfrak{A}} \neg\forall x\,\neg\alpha(x)\,[s]$. For instance, we have

$$\models_{\mathfrak{A}} \exists x\,(R(b,x) \vee R(x,b))\,[s].$$

First, $\models_{\mathfrak{A}} R(b,x) \vee R(x,b)\,[s[1/x]]$ ($s[3/x]$ would also fit the bill). So, $\not\models_{\mathfrak{A}} \neg(R(b,x) \vee R(x,b))\,[s[1/x]]$, thus $\not\models_{\mathfrak{A}} \forall x\,\neg((R(b,x) \vee R(x,b))\,[s]$, and therefore $\models_{\mathfrak{A}} \neg\forall x\,\neg((R(b,x) \vee R(x,b))\,[s]$. On the other hand,

$$\not\models_{\mathfrak{A}} \exists x\,(R(b,x) \wedge R(x,b)),\,[s].$$

That's because $\models_{\mathfrak{A}} \forall x\,\neg(R(b,x) \wedge R(x,b))\,[s]$, since for no $m \in |\mathfrak{A}|$, $\models_{\mathfrak{A}} R(b,x) \wedge R(x,b)\,[s[m/x]]$. As you can probably guess from these examples, $\models_{\mathfrak{A}} \exists x\,\alpha(x)\,[s]$ iff $\models_{\mathfrak{A}} \alpha(x)\,[s[m/x]]$ for at least one $m \in |\mathfrak{A}|$.

For a more complicated case, consider

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

Since $\not\models_{\mathfrak{A}} R(a,x)\,[s[3/x]]$ and $\not\models_{\mathfrak{A}} R(a,x)\,[s[4/x]]$, the interesting cases where we have to worry about the consequent of the conditional are only $m = 1$ and $= 2$. Does $\models_{\mathfrak{A}} \exists y\, R(x,y)\,[s[1/x]]$ hold? It does if there is at least one $n \in |\mathfrak{A}|$ so that $\models_{\mathfrak{A}} R(x,y)\,[s[1/x][n/y]]$. In fact, if we take $n = 1$, we have $s[1/x][n/y] = s[1/y] = s$. Since $s(x) = 1$, $s(y) = 1$, and $\langle 1,1\rangle \in R^{\mathfrak{A}}$, the answer is yes.

To determine if $\models_{\mathfrak{A}} \exists y\, R(x,y)\,[s[2/x]]$, we have to look at the variable assignments $s[2/x][n/y]$. Here, for $n = 1$, this assignment is $s_2 = s[2/x]$, which does not satisfy $R(x,y)$ ($s_2(x) = 2$, $s_2(y) = 1$, and $\langle 2,1\rangle \notin R^{\mathfrak{A}}$). However, consider $s[2/x][3/y] = s_2[3/y]$. $\models_{\mathfrak{A}} R(x,y)\,[s_2[3/y]]$ since $\langle 2,3\rangle \in R^{\mathfrak{A}}$, and so $\models_{\mathfrak{A}} \exists y\, R(x,y)\,[s_2]$.

So, for all $n \in |\mathfrak{A}|$, either $\not\models_{\mathfrak{A}} R(a,x)\,[s[m/x]]$ (if $m = 3,\ 4$) or $\models_{\mathfrak{A}} \exists y\, R(x,y)\,[s[m/x]]$ (if $m = 1,\ 2$), and so

$$\models_{\mathfrak{A}} \forall x\,(R(a,x) \to \exists y\, R(x,y))\,[s].$$

On the other hand,

$$\not\models_{\mathfrak{A}} \exists x\,(R(a,x) \wedge \forall y\, R(x,y))\,[s].$$

We have $\models_{\mathfrak{A}} R(a,x)\,[s[m/x]]$ only for $m = 1$ and $m = 2$. But for both of these values of m, there is in turn an $n \in |\mathfrak{A}|$, namely $n = 4$, so that $\not\models_{\mathfrak{A}} R(x,y)\,[s[m/x][n/y]]$ and so $\not\models_{\mathfrak{A}} \forall y\, R(x,y)\,[s[m/x]]$ for $m = 1$ and $m = 2$. In sum, there is no $m \in |\mathfrak{A}|$ such that $\models_{\mathfrak{A}} R(a,x) \wedge \forall y\, R(x,y)\,[s[m/x]]$.

Proposition 73G. *$\models_{\mathfrak{A}} \exists x\, \beta(x)\,[s]$ iff there is an x-variant s' of s so that $\models_{\mathfrak{A}} \beta(x)\,[s']$.*

Proof. Exercise. □

§7.4 Variable Assignments

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

Proposition 74A. *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 $\bar{s_1}(t) = \bar{s_2}(t)$.*

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

For the inductive step, assume that $t = ft_1 \ldots t_k$ and that the claim holds for $t_1, \ldots, t_k$. Then

$$\begin{aligned}\bar{s_1}(t) &= \bar{s_1}(ft_1 \ldots t_k) = \\ &= f^{\mathfrak{A}}(\bar{s_1}(t_1), \ldots, \bar{s_1}(t_k))\end{aligned}$$

For $j = 1, \ldots, k$, the variables of t_j are among $x_1, \ldots, x_n$. By induction hypothesis, $\bar{s_1}(t_j) = \bar{s_2}(t_j)$. So,

$$\begin{aligned}\bar{s_1}(t) &= \bar{s_1}(ft_1 \ldots t_k) = \\ &= f^{\mathfrak{A}}(\bar{s_1}(t_1), \ldots, \bar{s_1}(t_k)) = \\ &= f^{\mathfrak{A}}(\bar{s_2}(t_1), \ldots, \bar{s_2}(t_k)) = \\ &= \bar{s_2}(ft_1 \ldots t_k) = \bar{s_2}(t).\end{aligned}$$

□

Proposition 74B. *If the free variables in α are among $x_1, \ldots, x_n$, and $s_1(x_i) = s_2(x_i)$ for $i = 1, \ldots, n$, then $\models_{\mathfrak{A}} \alpha\,[s_1]$ iff $\models_{\mathfrak{A}} \alpha\,[s_2]$.*

Proof. We use induction on the complexity of α. For the base case, where α is atomic, α can be: $Rt_1 \ldots t_k$ 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. $\alpha \equiv Rt_1 \ldots t_k$: let $\models_{\mathfrak{A}} \alpha\,[s_1]$. Then

   $$\langle \bar{s_1}(t_1), \ldots, \bar{s_1}(t_k) \rangle \in R^{\mathfrak{A}}.$$

   For $i = 1, \ldots, k$, $\bar{s_1}(t_i) = \bar{s_2}(t_i)$ by Proposition 74A. So we also have $\langle \bar{s_2}(t_i), \ldots, \bar{s_2}(t_k) \rangle \in R^{\mathfrak{A}}$.

2. $\alpha \equiv t_1 = t_2$: suppose $\models_{\mathfrak{A}} \alpha\,[s_1]$. Then $\bar{s_1}(t_1) = \bar{s_1}(t_2)$. So,

   $$\begin{aligned}\bar{s_2}(t_1) &= \bar{s_1}(t_1) && \text{(by Proposition 74A)} \\ &= \bar{s_1}(t_2) && (\text{since } \models_{\mathfrak{A}} t_1 = t_2\,[s_1]) \\ &= \bar{s_2}(t_2) && \text{(by Proposition 74A),}\end{aligned}$$

   so $\models_{\mathfrak{A}} t_1 = t_2\,[s_2]$.

Now assume $\models_{\mathfrak{A}} \beta\,[s_1]$ iff $\models_{\mathfrak{A}} \beta\,[s_2]$ for all wffs β less complex than α. The induction step proceeds by cases determined by the main operator of α. In each case, we only demonstrate the forward direction of the biconditional; the proof of the reverse direction is symmetrical. In all cases except those for the quantifiers, we apply the induction hypothesis to sub-wffs β of α. The free variables of β are among those of α. Thus, if s_1 and s_2 agree on the free variables of α, they also agree on those of β, and the induction hypothesis applies to β.

1. $\alpha \equiv \neg\beta$: if $\models_{\mathfrak{A}} \alpha\,[s_1]$, then $\not\models_{\mathfrak{A}} \beta\,[s_1]$, so by the induction hypothesis, $\not\models_{\mathfrak{A}} \beta\,[s_2]$, hence $\models_{\mathfrak{A}} \alpha\,[s_2]$.

2. $\alpha \equiv \beta \to \gamma$: if $\models_{\mathfrak{A}} \alpha\,[s_1]$, then $\not\models_{\mathfrak{A}} \beta\,[s_1]$ or $\models_{\mathfrak{A}} \gamma\,[s_1]$. By the induction hypothesis, $\not\models_{\mathfrak{A}} \beta\,[s_2]$ or $\models_{\mathfrak{A}} \gamma\,[s_2]$, so $\models_{\mathfrak{A}} \alpha\,[s_2]$.

3. $\alpha \equiv \forall x\,\beta$: if $\models_{\mathfrak{A}} \alpha\,[s_1]$, then for every $m \in |\mathfrak{A}|$, $\models_{\mathfrak{A}} \beta\,[s_1[m/x]]$. We want to show that also, for every $m \in |\mathfrak{A}|$, $\models_{\mathfrak{A}} \beta\,[s_2[m/x]]$. So let $m \in |\mathfrak{A}|$ be arbitrary, and consider $s_1' = s[m/x]$ and $s_2' = s[m/x]$. We have that $\models_{\mathfrak{A}} \beta\,[s_1']$. The free variables of β are among $x_1, \ldots, x_n$, and x. $s_1'(x_i) = s_2'(x_i)$, since s_1' and s_2' are x-variants of s_1 and s_2, respectively, and by hypothesis $s_1(x_i) = s_2(x_i)$. $s_1'(x) = s_2'(x) = m$ by the way we have defined s_1' and s_2'. Then the induction hypothesis applies to β and s_1', s_2', and we have $\models_{\mathfrak{A}} \beta\,[s_2']$. This applies to every $m \in |\mathfrak{A}|$, i.e., $\models_{\mathfrak{A}} \beta\,[s_2[m/x]]$ for all $m \in |\mathfrak{A}|$, so $\models_{\mathfrak{A}} \alpha\,[s_2]$.

By induction, we get that $\models_{\mathfrak{A}} \alpha\,[s_1]$ iff $\models_{\mathfrak{A}} \alpha\,[s_2]$ whenever the free variables in α are among $x_1, \ldots, x_n$ and $s_1(x_i) = s_2(x_i)$ for $i = 1, \ldots, n$. □

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

Corollary 74C. *If α is a sentence and s a variable assignment, then $\models_{\mathfrak{A}} \alpha\,[s]$ iff $\models_{\mathfrak{A}} \alpha\,[s']$ for every variable assignment s'.*

Proof. Let s' be any variable assignment. Since α is a sentence, it has no free variables, and so every variable assignment s' trivially assigns the same things to all free variables of α as does s. So the condition of Proposition 74B is satisfied, and we have $\models_{\mathfrak{A}} \alpha\,[s]$ iff $\models_{\mathfrak{A}} \alpha\,[s']$. □

Definition 74D. *If α is a sentence, we say that a structure $\mathfrak{A}$* satisfies *α, $\models_{\mathfrak{A}} \alpha$, iff $\models_{\mathfrak{A}} \alpha\,[s]$ for all variable assignments s.*

If $\models_{\mathfrak{A}} \alpha$, we also simply say that *α is true in $\mathfrak{A}$*.

Proposition 74E. *Let $\mathfrak{A}$ be a structure, α be a sentence, and s a variable assignment. $\models_{\mathfrak{A}} \alpha$ iff $\models_{\mathfrak{A}} \alpha\,[s]$.*

Proof. Exercise. □

Proposition 74F. *Suppose $\alpha(x)$ only contains x free, and $\mathfrak{A}$ is a structure. Then: $\models_{\mathfrak{A}} \forall x\,\alpha(x)$ iff $\models_{\mathfrak{A}} \alpha(x)\,[s]$ for all variable assignments s.*

Proof. Exercise. □

§7.5 Extensionality

Extensionality, sometimes called relevance, can be expressed informally as follows: the only factors that bear upon the satisfaction of wff α in a structure $\mathfrak{A}$ relative to a variable assignment s, are the size of the domain and the assignments made by $\mathfrak{A}$ and s to the elements of the language that actually appear in α.

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

Proposition 75A (Extensionality). *Let α be a wff, and $\mathfrak{A}_1$ and $\mathfrak{A}_2$ be structures with $|\mathfrak{A}_1| = |\mathfrak{A}_2|$, and s a variable assignment on $|\mathfrak{A}_1| = |\mathfrak{A}_2|$. If $c^{\mathfrak{A}_1} = c^{\mathfrak{A}_2}$, $R^{\mathfrak{A}_1} = R^{\mathfrak{A}_2}$, and $f^{\mathfrak{A}_1} = f^{\mathfrak{A}_2}$ for every constant symbol c, relation symbol R, and function symbol f occurring in α, then $\models_{\mathfrak{A}_1} \alpha\,[s]$ iff $\models_{\mathfrak{A}_2} \alpha\,[s]$.*

Proof. First prove (by induction on t) that for every term, $\bar{s}(t) = \bar{s}(t)$. Then prove the proposition by induction on α, making use of the claim just proved for the induction basis (where α is atomic). □

Corollary 75B (Extensionality for Sentences). *Let α be a sentence and $\mathfrak{A}_1$, $\mathfrak{A}_2$ as in Proposition 75A. Then $\models_{\mathfrak{A}_1} \alpha$ iff $\models_{\mathfrak{A}_2} \alpha$.*

Proof. Follows from Proposition 75A by corollary 74C. □

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

Proposition 75C. *Let $\mathfrak{A}$ be a structure, t and t' terms, and s a variable assignment. Then $\bar{s}(t[t'/x]) = s[^{-}(\bar{t'})/x]\bar{s}(t)$.*

Proof. By induction on t.

1. If t is a constant, say, $t \equiv c$, then $t[t'/x] = c$, and $\bar{s}(c) = c^{\mathfrak{A}} = s[^{-}(\bar{t'})/x]\bar{s}(c)$.

2. If t is a variable other than x, say, $t \equiv y$, then $t[t'/x] = y$, and $\bar{s}(y) = s[^{-}(\bar{t'})/x]\bar{s}(y)$ since $s \sim_x s[\bar{s}(t')/x]$.

3. If $t \equiv x$, then $t[t'/x] = t'$. But $s[^{-}(\bar{t'})/x]\bar{s}(x) = \bar{s}(t')$ by definition of $s[\bar{s}(t')/x]$.

4. If $t \equiv ft_1 \dots t_n$ then we have:

$$\begin{aligned}
\bar{s}(t[t'/x]) = & \\
& = \bar{s}(ft_1[t'/x] \dots t_n[t'/x]) \\
& \qquad \text{by definition of } t[t'/x] \\
& = f^{\mathfrak{A}}(\bar{s}(t_1[t'/x]), \dots, \bar{s}(t_n[t'/x])) \\
& \qquad \text{by definition of } \bar{s}(f \dots) \\
& = f^{\mathfrak{A}}(s[^{-}(\bar{t'})/x](t_1), \dots, s[^{-}(\bar{t'})/x](t_n)) \\
& \qquad \text{by induction hypothesis} \\
& = s[^{-}(\bar{t'})/x](t) \text{ by definition of } s[^{-}(\bar{t'})/x](f \dots)
\end{aligned}$$ □

Proposition 75D. *Let $\mathfrak{A}$ be a structure, α a wff, t' a term, and s a variable assignment. Then $\models_{\mathfrak{A}} \alpha[t'/x]\,[s]$ iff $\models_{\mathfrak{A}} \alpha\,[s[\bar{s}(t')/x]]$.*

Proof. Exercise. □

The point of Propositions 75C and 75D is the following. Suppose we have a term t or a wff α and some term t', and we want to know the value of $t[t'/x]$ or whether or not $\alpha[t'/x]$ is satisfied in a structure $\mathfrak{A}$ relative to a variable assignment s. Then we can either perform the substitution first and then consider the value or satisfaction relative to $\mathfrak{A}$ and s, or we can first determine the value $m = \bar{s}(t')$ of t' in $\mathfrak{A}$ relative to s, change the variable assignment to $s[m/x]$ and then consider the value of t in $\mathfrak{A}$ and $s[m/x]$, or whether $\models_{\mathfrak{A}} \alpha\,[s[m/x]]$. Propositions 75C and 75D guarantee that the answer will be the same, whichever way we do it.

§7.6 Semantic Notions

Given 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 76A (Validity). *A sentence α is* valid, *$\vDash \alpha$, iff $\models_{\mathfrak{A}} \alpha$ for every structure $\mathfrak{A}$.*

Definition 76B (Entailment). *A set of sentences Γ* entails *a sentence α, $\Gamma \vDash \alpha$, iff for every structure $\mathfrak{A}$ with $\models_{\mathfrak{A}} \Gamma$, $\models_{\mathfrak{A}} \alpha$.*

Definition 76C (Satisfiability). *A set of sentences Γ is* satisfiable *if $\models_{\mathfrak{A}} \Gamma$ for some structure $\mathfrak{A}$. If Γ is not satisfiable it is called* unsatisfiable.

Proposition 76D. *A sentence α is valid iff $\Gamma \vDash \alpha$ for every set of sentences Γ.*

Proof. For the forward direction, let α be valid, and let Γ be a set of sentences. Let $\mathfrak{A}$ be a structure so that $\models_{\mathfrak{A}} \Gamma$. Since α is valid, $\models_{\mathfrak{A}} \alpha$, hence $\Gamma \vDash \alpha$.

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

Proposition 76E. *$\Gamma \vDash \alpha$ iff $\Gamma \cup \{\neg\alpha\}$ is unsatisfiable.*

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

For the reverse direction, suppose $\Gamma \cup \{\neg\alpha\}$ is unsatisfiable. So for every structure $\mathfrak{A}$, either $\not\models_{\mathfrak{A}} \Gamma$ or $\models_{\mathfrak{A}} \alpha$. Hence, for every structure $\mathfrak{A}$ with $\models_{\mathfrak{A}} \Gamma$, $\models_{\mathfrak{A}} \alpha$, so $\Gamma \vDash \alpha$. □

Proposition 76F. *If $\Gamma \subseteq \Gamma'$ and $\Gamma \vDash \alpha$, then $\Gamma' \vDash \alpha$.*

Proof. Suppose that $\Gamma \subseteq \Gamma'$ and $\Gamma \vDash \alpha$. Let $\mathfrak{A}$ be a structure such that $\models_{\mathfrak{A}} \Gamma'$; then $\models_{\mathfrak{A}} \Gamma$, and since $\Gamma \vDash \alpha$, we get that $\models_{\mathfrak{A}} \alpha$. Hence, whenever $\models_{\mathfrak{A}} \Gamma'$, $\models_{\mathfrak{A}} \alpha$, so $\Gamma' \vDash \alpha$. □

Theorem 76G (Semantic Deduction Theorem). *$\Gamma \cup \{\alpha\} \vDash \beta$ iff $\Gamma \vDash \alpha \to \beta$.*

Proof. For the forward direction, let $\Gamma \cup \{\alpha\} \vDash \beta$ and let $\mathfrak{A}$ be a structure so that $\models_{\mathfrak{A}} \Gamma$. If $\models_{\mathfrak{A}} \alpha$, then $\models_{\mathfrak{A}} \Gamma \cup \{\alpha\}$, so since $\Gamma \cup \{\alpha\}$ entails β, we get $\models_{\mathfrak{A}} \beta$. Therefore, $\models_{\mathfrak{A}} \alpha \to \beta$, so $\Gamma \vDash \alpha \to \beta$.

For the reverse direction, let $\Gamma \vDash \alpha \to \beta$ and $\mathfrak{A}$ be a structure so that $\models_{\mathfrak{A}} \Gamma \cup \{\alpha\}$. Then $\models_{\mathfrak{A}} \Gamma$, so $\models_{\mathfrak{A}} \alpha \to \beta$, and since $\models_{\mathfrak{A}} \alpha$, $\models_{\mathfrak{A}} \beta$. Hence, whenever $\models_{\mathfrak{A}} \Gamma \cup \{\alpha\}$, $\models_{\mathfrak{A}} \beta$, so $\Gamma \cup \{\alpha\} \vDash \beta$. □

Proposition 76H. *Let $\mathfrak{A}$ be a structure, and $\alpha(x)$ a wff with one free variable x, and t a closed term. Then:* $\forall x\, \alpha(x) \vDash \alpha(t)$

Proof. Suppose $\models_{\mathfrak{A}} \forall x\, \alpha(x)$. Let s be a variable assignment with $s(x) = t^{\mathfrak{A}}$. By Proposition 74F, $\models_{\mathfrak{A}} \alpha(x)\,[s]$. By Proposition 75D, $\models_{\mathfrak{A}} \alpha(t)\,[s]$. By Proposition 74E, $\models_{\mathfrak{A}} \alpha(t)$ since $\alpha(t)$ is a sentence. □

Problems

Problem 1. Is $\mathfrak{B}$, the standard model of arithmetic, covered? Explain.

Problem 2. Prove Proposition 73G

Problem 3. 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 $\mathfrak{A}$ be given by

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

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

$$\models_{\mathfrak{A}} \exists x\, (A(f(z), c) \to \forall y\, (A(y, x) \vee A(f(y), x)))\, [s]$$

Explain why or why not.

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

Problem 4. Complete the proof of Proposition 74B.

Problem 5. Prove Proposition 74E

Problem 6. Prove Proposition 74F.

Problem 7. Suppose $\mathcal{L}$ is a language without function symbols. Given a structure $\mathfrak{A}$, c a constant symbol and $a \in |\mathfrak{A}|$, define $\mathfrak{A}[a/c]$ to be the structure that is just like $\mathfrak{A}$, except that $c^{\mathfrak{A}[a/c]} = a$. Define $\mathfrak{A} \mid\!\models \alpha$ for sentences α by:

1. $\alpha \equiv Rd_1 \dots d_n$: $\mathfrak{A} \mid\!\models \alpha$ iff $\langle d_1^{\mathfrak{A}}, \dots, d_n^{\mathfrak{A}}\rangle \in R^{\mathfrak{A}}$.
2. $\alpha \equiv d_1 = d_2$: $\mathfrak{A} \mid\!\models \alpha$ iff $d_1^{\mathfrak{A}} = d_2^{\mathfrak{A}}$.
3. $\alpha \equiv \neg\beta$: $\mathfrak{A} \mid\!\models \alpha$ iff not $\mathfrak{A} \mid\!\models \beta$.
4. $\alpha \equiv (\beta \to \gamma)$: $\mathfrak{A} \mid\!\models \alpha$ iff not $\mathfrak{A} \mid\!\models \beta$ or $\mathfrak{A} \mid\!\models \gamma$ (or both).
5. $\alpha \equiv \forall x\, \beta$: $\mathfrak{A} \mid\!\models \alpha$ iff for all $a \in |\mathfrak{A}|$, $\mathfrak{A}[a/c] \mid\!\models \beta[c/x]$, if c does not occur in β.

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

Show that $\models_{\mathfrak{A}} \alpha\, [s]$ iff $\mathfrak{A}[a_1/c_1, \dots, a_n/c_n] \mid\!\models \alpha[c_1/x_1] \dots [c_n/x_n]$.

(This problem shows that it is possible to give a semantics for first-order logic that makes do without variable assignments.)

Problem 8. Suppose that f is a function symbol not in $\alpha(x, y)$. Show that there is a structure $\mathfrak{A}$ such that $\models_{\mathfrak{A}} \forall x \exists y \, \alpha(x, y)$ iff there is an $\mathfrak{A}'$ such that $\models_{\mathfrak{A}'} \forall x \, \alpha(x, f(x))$.

(This problem is a special case of what's known as Skolem's Theorem; $\forall x \, \alpha(x, f(x))$ is called a *Skolem normal form* of $\forall x \exists y \, \alpha(x, y)$.)

Problem 9. Carry out the proof of Proposition 75A in detail.

Problem 10. Prove Proposition 75D

Problem 11.

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

Problem 12. Complete the proof of Proposition 76H.

Chapter 8

Theories and Their Models

§8.0 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 80A. *A set of sentences Γ is* closed *iff, whenever $\Gamma \vDash \alpha$ then $\alpha \in \Gamma$. The* closure *of a set of sentences Γ is $\{\alpha : \Gamma \vDash \alpha\}$.*

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 struc-

tures, we will successfully capture that class by an axiom system Γ iff the structures are exactly those $\mathfrak{A}$ such that $\models_{\mathfrak{A}} \Gamma$.

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

3. If we are successful at least in the respect that Γ is true in all the intended structures, then a sentence α is true in all intended structures whenever $\Gamma \vDash \alpha$. Thus we can use logical tools (such as derivation 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 α in Γ is redundant by proving $\Gamma \setminus \{\alpha\} \vDash \alpha$. We can also prove that an axiom is not redundant by showing that $(\Gamma \setminus \{\alpha\}) \cup \{\neg\alpha\}$ 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 consist 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 others (and hence must add it to the primitives of the language)?

§8.1 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 $\mathfrak{A}$ can we express by a sentence (or perhaps a set of sentences) in the language of $\mathfrak{A}$? 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 $\mathfrak{A}$ if $|\mathfrak{A}| = \mathbb{N}$. 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 81A (Model of a set). *Let Γ be a set of sentences in a language $\mathcal{L}$. We say that a structure* $\mathfrak{A}$ is a model of Γ *if $\models_{\mathfrak{A}} \alpha$ for all $\alpha \in \Gamma$.*

Example 8.1.2. The sentence $\forall x\, x \leq x$ is true in $\mathfrak{A}$ iff $\leq^{\mathfrak{A}}$ is a reflexive relation. The sentence $\forall x \forall y\, ((x \leq y \wedge y \leq x) \rightarrow x = y)$ is true in $\mathfrak{A}$ iff $\leq^{\mathfrak{A}}$ 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 $\mathfrak{A}$ iff $\leq^{\mathfrak{A}}$ is transitive. Thus, the models of

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

are exactly those structures in which $\leq^{\mathfrak{A}}$ 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*.

§8.2 Examples of First-Order Theories

Example 8.2.1. The theory of strict linear orders in the language $\mathcal{L}_<$ is axiomatized by the set

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

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 $\mathfrak{A}$ with $|\mathfrak{A}| = X$ and $<^{\mathfrak{A}} = R$ is a model of this theory.

Example 8.2.2. 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 8.2.3. The theory of Peano arithmetic is axiomatized by the following sentences in the language of arithmetic $\mathcal{L}_A$.

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

plus all sentences of the form

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

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 last axiom is an *explicit definition* of $<$.

Example 8.2.4. 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

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

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 set X, the set $\cup X$ exists, where $\cup X$ is the union of all the elements of X.

The sentences mentioned last are collectively called the *naive comprehension scheme*. It essentially says that for every $\alpha(x)$, the set $\{x : \alpha(x)\}$ 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 $\alpha(y)$ 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 8.2.5. 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 Pxy "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," $P^{\mathfrak{A}}$ 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.

$$\begin{aligned}
&\forall x \, Pxx \\
&\forall x \, \forall y \, ((Pxy \wedge Pyx) \rightarrow x = y) \\
&\forall x \, \forall y \, \forall z \, ((Pxy \wedge Pyz) \rightarrow Pxz)
\end{aligned}$$

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

$$\forall x \, \forall y \, \exists z \, \forall u \, (Pzu \leftrightarrow (Pxu \wedge Pyu))$$

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 introducing 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.

§8.3 Expressing Relations in a Structure

One main use wffs can be put to is to express properties and relations in a structure $\mathfrak{A}$ in terms of the primitives of the language $\mathcal{L}$ of $\mathfrak{A}$. By this we mean the following: the domain of $\mathfrak{A}$ is a set of objects. The constant symbols, function symbols, and predicate symbols are interpreted in $\mathfrak{A}$ by some objects in$|\mathfrak{A}|$, functions on $|\mathfrak{A}|$, and relations on $|\mathfrak{A}|$. For instance, if A_0^2 is in $\mathcal{L}$, then $\mathfrak{A}$ assigns to it a relation $R = A_0^{2\mathfrak{A}}$. 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 $a \in |\mathfrak{A}|$ and v_2 to $b \in |\mathfrak{A}|$, then

$$Rab \quad \text{iff} \quad \models_{\mathfrak{A}} A_0^2 v_1 v_2 \, [s].$$

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

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

Definition 83A. *Let $\alpha(v_1, \ldots, v_n)$ be a wff of $\mathcal{L}$ in which only $v_1, \ldots, v_n$ occur free, and let $\mathfrak{A}$ be a structure for $\mathcal{L}$. $\alpha(v_1, \ldots, v_n)$* expresses the relation $R \subseteq |\mathfrak{A}|^n$ *iff*

$$Ra_1 \ldots a_n \quad \textit{iff} \quad \models_{\mathfrak{A}} \alpha v_1 \ldots v_n \, [s]$$

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

Example 8.3.2. In the standard model of arithmetic $\mathfrak{B}$, the wff $v_1 < v_2 \vee v_1 = v_2$ expresses the $\leq$ relation on $\mathbb{N}$. The wff $v_2 = v_1'$ 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 wffs $\exists v_3 \, (v_3 \neq 0 \wedge v_2 = (v_1 + v_3))$ 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.

§8.4 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 wff $\alpha(x, y)$ 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 as $X \cup Y$ 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 wffs 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 wff expresses "$f(x) = y$").

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

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

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

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

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

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

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

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

§8.5 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 85A. *The sentence*

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

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

Proposition 85B. *The sentence*

$$\begin{aligned}\alpha_{=n} \equiv \exists x_1\, \exists x_2\, \ldots \exists x_n\ & \\ & (x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge x_1 \neq x_4 \wedge \cdots \wedge x_1 \neq x_n \wedge \\ & \qquad x_2 \neq x_3 \wedge x_2 \neq x_4 \wedge \cdots \wedge x_2 \neq x_n \wedge \\ & \qquad\qquad \vdots \\ & \qquad x_{n-1} \neq x_n \wedge \\ & \qquad \forall y\, (y = x_1 \vee \cdots \vee y = x_n))\end{aligned}$$

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

Proposition 85C. *A structure is infinite iff it is a model of*

$$\{\alpha_{\geq 1}, \alpha_{\geq 2}, \alpha_{\geq 3}, \dots\}.$$

There is no single purely logical sentence which is true in $\mathfrak{A}$ iff $|\mathfrak{A}|$ 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.

Problems

Problem 1. Find wffs 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 2. Suppose the formula $\alpha(v_1, v_2)$ expresses the relation $R \subseteq |\mathfrak{A}|^2$ in a structure $\mathfrak{A}$. 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 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 $\mathfrak{B}$ be the structure such that $|\mathfrak{B}| = \mathbb{N}$, and $<^{\mathfrak{B}} = \{\langle n, m\rangle : n < m\}$. Prove the following:

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

Problem 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 $(A \to \neg A) \wedge (\neg A \to A) \vdash \bot$.

Chapter 9

Derivation Systems

§9.0 Introduction

Logics commonly have both a semantics and a derivation system. The semantics concerns concepts such as truth, satisfiability, validity, and entailment. The purpose of derivation systems is to provide a purely syntactic method of establishing entailment and validity. They are purely syntactic in the sense that a derivation in such a system is a finite syntactic object, usually a sequence (or other finite arrangement) of sentences or wffs. Good derivation systems have the property that any given sequence or arrangement of sentences or wffs can be verified mechanically to be "correct."

The simplest (and historically first) derivation systems for first-order logic were *axiomatic*. A sequence of wffs counts as a derivation in such a system if each individual wff in it is either among a fixed set of "axioms" or follows from wffs coming before it in the sequence by one of a fixed number of "inference rules"—and it can be mechanically verified if a wff is an axiom and whether it follows correctly from other wffs by one of the inference rules. Axiomatic derivation systems are easy to describe—and also easy to handle metatheoretically—but derivations in them are hard to read and understand, and are also hard to produce.

Other derivation systems have been developed with the aim of making it easier to construct derivations or easier to understand derivations once they are complete. Examples are natural deduction, truth trees, also known as tableaux proofs, and the sequent calculus. Some derivation systems are designed especially with mechanization in mind, e.g., the resolution method is easy to implement in software (but its derivations are essentially impossible to understand). Most of these other derivation systems represent derivations as trees of wffs rather than sequences. This makes it easier to see which parts of a derivation depend on which other parts.

So for a given logic, such as first-order logic, the different derivation systems will give different explications of what it is for a sentence to be a *theorem* and what it means for a sentence to be derivable from some others. However that is

done (via axiomatic derivations, natural deductions, sequent derivations, truth trees, resolution refutations), we want these relations to match the semantic notions of validity and entailment. Let's write $\vdash \alpha$ for "α is a theorem" and "$\Gamma \vdash \alpha$" for "α is derivable from Γ." However $\vdash$ is defined, we want it to match up with $\models$, that is:

1. $\vdash \alpha$ if and only if $\models \alpha$
2. $\Gamma \vdash \alpha$ if and only if $\Gamma \models \alpha$

The "only if" direction of the above is called *soundness*. A derivation system is sound if derivability guarantees entailment (or validity). Every decent derivation system has to be sound; unsound derivation systems are not useful at all. After all, the entire purpose of a derivation is to provide a syntactic guarantee of validity or entailment. We'll prove soundness for the derivation systems we present.

The converse "if" direction is also important: it is called *completeness*. A complete derivation system is strong enough to show that α is a theorem whenever α is valid, and that $\Gamma \vdash \alpha$ whenever $\Gamma \models \alpha$. Completeness is harder to establish, and some logics have no complete derivation systems. First-order logic does. Kurt Gödel was the first one to prove completeness for a derivation system of first-order logic in his 1929 dissertation.

Another concept that is connected to derivation systems is that of *consistency*. A set of sentences is called inconsistent if anything whatsoever can be derived from it, and consistent otherwise. Inconsistency is the syntactic counterpart to unsatisfiablity: like unsatisfiable sets, inconsistent sets of sentences do not make good theories, they are defective in a fundamental way. Consistent sets of sentences may not be true or useful, but at least they pass that minimal threshold of logical usefulness. For different derivation systems the specific definition of consistency of sets of sentences might differ, but like $\vdash$, we want consistency to coincide with its semantic counterpart, satisfiability. We want it to always be the case that Γ is consistent if and only if it is satisfiable. Here, the "if" direction amounts to completeness (consistency guarantees satisfiability), and the "only if" direction amounts to soundness (satisfiability guarantees consistency). In fact, for classical first-order logic, the two versions of soundness and completeness are equivalent.

§9.1 The Sequent Calculus

While many derivation systems operate with arrangements of sentences, the sequent calculus operates with *sequents*. A sequent is an expression of the form

$$\alpha_1, \ldots, \alpha_m \Rightarrow \beta_1, \ldots, \beta_m,$$

that is a pair of sequences of sentences, separated by the sequent symbol $\Rightarrow$. Either sequence may be empty. A derivation in the sequent calculus is a tree of sequents, where the topmost sequents are of a special form (they are called

"initial sequents" or "axioms") and every other sequent follows from the sequents immediately above it by one of the rules of inference. The rules of inference either manipulate the sentences in the sequents (adding, removing, or rearranging them on either the left or the right), or they introduce a complex wff in the conclusion of the rule. For instance, the $\wedge$L rule allows the inference from $\alpha, \Gamma \Rightarrow \Delta$ to $\alpha \wedge \beta, \Gamma \Rightarrow \Delta$, and the $\rightarrow$R allows the inference from $\alpha, \Gamma \Rightarrow \Delta, \beta$ to $\Gamma \Rightarrow \Delta, \alpha \rightarrow \beta$, for any Γ, Δ, α, and β. (In particular, Γ and Δ may be empty.)

The $\vdash$ relation based on the sequent calculus is defined as follows: $\Gamma \vdash \alpha$ iff there is some sequence Γ_0 such that every α in Γ_0 is in Γ and there is a derivation with the sequent $\Gamma_0 \Rightarrow \alpha$ at its root. α is a theorem in the sequent calculus if the sequent $\Rightarrow \alpha$ has a derivation. For instance, here is a derivation that shows that $\vdash (\alpha \wedge \beta) \rightarrow \alpha$:

$$\dfrac{\dfrac{\alpha \Rightarrow \alpha}{\alpha \wedge \beta \Rightarrow \alpha}\,\wedge\text{L}}{\Rightarrow (\alpha \wedge \beta) \rightarrow \alpha}\,\rightarrow\text{R}$$

A set Γ is inconsistent in the sequent calculus if there is a derivation of $\Gamma_0 \Rightarrow$ (where every $\alpha \in \Gamma_0$ is in Γ and the right side of the sequent is empty). Using the rule WR, any sentence can be derived from an inconsistent set.

The sequent calculus was invented in the 1930s by Gerhard Gentzen. Because of its systematic and symmetric design, it is a very useful formalism for developing a theory of derivations. It is relatively easy to find derivations in the sequent calculus, but these derivations are often hard to read and their connection to proofs are sometimes not easy to see. It has proved to be a very elegant approach to derivation systems, however, and many logics have sequent calculus systems.

§9.2 Natural Deduction

Natural deduction is a derivation system intended to mirror actual reasoning (especially the kind of regimented reasoning employed by mathematicians). Actual reasoning proceeds by a number of "natural" patterns. For instance, proof by cases allows us to establish a conclusion on the basis of a disjunctive premise, by establishing that the conclusion follows from either of the disjuncts. Indirect proof allows us to establish a conclusion by showing that its negation leads to a contradiction. Conditional proof establishes a conditional claim "if ... then ..." by showing that the consequent follows from the antecedent. Natural deduction is a formalization of some of these natural inferences. Each of the logical connectives and quantifiers comes with two rules, an introduction and an elimination rule, and they each correspond to one such natural inference pattern. For instance, $\rightarrow$Intro corresponds to conditional proof, and $\vee$Elim to proof by cases. A particularly simple rule is $\wedge$Elim which allows the inference from $\alpha \wedge \beta$ to α (or β).

One feature that distinguishes natural deduction from other derivation systems is its use of assumptions. A derivation in natural deduction is a tree of

wffs. A single wff stands at the root of the tree of wffs, and the "leaves" of the tree are wffs from which the conclusion is derived. In natural deduction, some leaf wffs play a role inside the derivation but are "used up" by the time the derivation reaches the conclusion. This corresponds to the practice, in actual reasoning, of introducing hypotheses which only remain in effect for a short while. For instance, in a proof by cases, we assume the truth of each of the disjuncts; in conditional proof, we assume the truth of the antecedent; in indirect proof, we assume the truth of the negation of the conclusion. This way of introducing hypothetical assumptions and then doing away with them in the service of establishing an intermediate step is a hallmark of natural deduction. The formulas at the leaves of a natural deduction derivation are called assumptions, and some of the rules of inference may "discharge" them. For instance, if we have a derivation of β from some assumptions which include α, then the $\rightarrow$Intro rule allows us to infer $\alpha \rightarrow \beta$ and discharge any assumption of the form α. (To keep track of which assumptions are discharged at which inferences, we label the inference and the assumptions it discharges with a number.) The assumptions that remain undischarged at the end of the derivation are together sufficient for the truth of the conclusion, and so a derivation establishes that its undischarged assumptions entail its conclusion.

The relation $\Gamma \vdash \alpha$ based on natural deduction holds iff there is a derivation in which α is the last sentence in the tree, and every leaf which is undischarged is in Γ. α is a theorem in natural deduction iff there is a derivation in which α is the last sentence and all assumptions are discharged. For instance, here is a derivation that shows that $\vdash (\alpha \wedge \beta) \rightarrow \alpha$:

$$\cfrac{\cfrac{[\alpha \wedge \beta]^1}{\alpha}\,\wedge\text{Elim}}{(\alpha \wedge \beta) \rightarrow \alpha}\,{}^{1}\rightarrow\text{Intro}$$

The label 1 indicates that the assumption $\alpha \wedge \beta$ is discharged at the $\rightarrow$Intro inference.

A set Γ is inconsistent iff $\Gamma \vdash \bot$ in natural deduction. The rule $\bot_I$ makes it so that from an inconsistent set, any sentence can be derived.

Natural deduction systems were developed by Gerhard Gentzen and Stanisław Jaśkowski in the 1930s, and later developed by Dag Prawitz and Frederic Fitch. Because its inferences mirror natural methods of proof, it is favored by philosophers. The versions developed by Fitch are often used in introductory logic textbooks. In the philosophy of logic, the rules of natural deduction have sometimes been taken to give the meanings of the logical operators ("proof-theoretic semantics").

§9.3 Tableaux

While many derivation systems operate with arrangements of sentences, tableaux operate with signed formulas. A signed formula is a pair consisting of a truth

value sign ($\mathbb{T}$ or $\mathbb{F}$) and a sentence

$$\mathbb{T}\,\alpha \text{ or } \mathbb{F}\,\alpha.$$

A tableau consists of signed formulas arranged in a downward-branching tree. It begins with a number of *assumptions* and continues with signed formulas which result from one of the signed formulas above it by applying one of the rules of inference. Each rule allows us to add one or more signed formulas to the end of a branch, or two signed formulas side by side—in this case a branch splits into two, with the two added signed formulas forming the ends of the two branches.

A rule applied to a complex signed formula results in the addition of signed formulas which are immediate sub-wffs. They come in pairs, one rule for each of the two signs. For instance, the $\wedge\mathbb{T}$ rule applies to $\mathbb{T}\,\alpha \wedge \beta$, and allows the addition of both the two signed formulas $\mathbb{T}\,\alpha$ and $\mathbb{T}\,\beta$ to the end of any branch containing $\mathbb{T}\,\alpha \wedge \beta$, and the rule $\alpha \wedge \beta\mathbb{F}$ allows a branch to be split by adding $\mathbb{F}\,\alpha$ and $\mathbb{F}\,\beta$ side-by-side. A tableau is closed if every one of its branches contains a matching pair of signed formulas $\mathbb{T}\,\alpha$ and $\mathbb{F}\,\alpha$.

The $\vdash$ relation based on tableaux is defined as follows: $\Gamma \vdash \alpha$ iff there is some finite set $\Gamma_0 = \{\beta_1, \ldots, \beta_n\} \subseteq \Gamma$ such that there is a closed tableau for the assumptions

$$\{\mathbb{F}\,\alpha, \mathbb{T}\,\beta_1, \ldots, \mathbb{T}\,\beta_n\}$$

For instance, here is a closed tableau that shows that $\vdash (\alpha \wedge \beta) \to \alpha$:

1.	$\mathbb{F}\,(\alpha \wedge \beta) \to \alpha$	Assumption
2.	$\mathbb{T}\,\alpha \wedge \beta$	$\to\mathbb{F}\,1$
3.	$\mathbb{F}\,\alpha$	$\to\mathbb{F}\,1$
4.	$\mathbb{T}\,\alpha$	$\to\mathbb{T}\,2$
5.	$\mathbb{T}\,\beta$	$\to\mathbb{T}\,2$
	$\otimes$	

A set Γ is inconsistent in the tableau calculus if there is a closed tableau for assumptions

$$\{\mathbb{T}\,\beta_1, \ldots, \mathbb{T}\,\beta_n\}$$

for some $\beta_i \in \Gamma$.

Tableaux were invented in the 1950s independently by Evert Beth and Jaakko Hintikka, and simplified and popularized by Raymond Smullyan. They are very easy to use, since constructing a tableau is a very systematic procedure. Because of the systematic nature of tableaux, they also lend themselves to implementation by computer. However, a tableau is often hard to read and their connection to proofs are sometimes not easy to see. The approach is also quite general, and many different logics have tableau systems. Tableaux also help us to find structures that satisfy given (sets of) sentences: if the set is satisfiable, it won't have a closed tableau, i.e., any tableau will have an open branch. The satisfying structure can be "read off" an open branch, provided

every rule it is possible to apply has been applied on that branch. There is also a very close connection to the sequent calculus: essentially, a closed tableau is a condensed derivation in the sequent calculus, written upside-down.

§9.4 Axiomatic Derivations

Axiomatic derivations are the oldest and simplest logical derivation systems. Its derivations are simply sequences of sentences. A sequence of sentences counts as a correct derivation if every sentence α in it satisfies one of the following conditions:

1. α is an axiom, or
2. α is an element of a given set Γ of sentences, or
3. α is justified by a rule of inference.

To be an axiom, α has to have the form of one of a number of fixed sentence schemas. There are many sets of axiom schemas that provide a satisfactory (sound and complete) derivation system for first-order logic. Some are organized according to the connectives they govern, e.g., the schemas

$$\alpha \to (\beta \to \alpha) \qquad \beta \to (\beta \vee \gamma) \qquad (\beta \wedge \gamma) \to \beta$$

are common axioms that govern $\to$, $\vee$ and $\wedge$. Some axiom systems aim at a minimal number of axioms. Depending on the connectives that are taken as primitives, it is even possible to find axiom systems that consist of a single axiom.

A rule of inference is a conditional statement that gives a sufficient condition for a sentence in a derivation to be justified. Modus ponens is one very common such rule: it says that if α and $\alpha \to \beta$ are already justified, then β is justified. This means that a line in a derivation containing the sentence β is justified, provided that both α and $\alpha \to \beta$ (for some sentence α) appear in the derivation before β.

The $\vdash$ relation based on axiomatic derivations is defined as follows: $\Gamma \vdash \alpha$ iff there is a derivation with the sentence α as its last formula (and Γ is taken as the set of sentences in that derivation which are justified by (2) above). α is a theorem if α has a derivation where Γ is empty, i.e., every sentence in the derivation is justified either by (1) or (3). For instance, here is a derivation that shows that $\vdash \alpha \to (\beta \to (\beta \vee \alpha))$:

1. $\beta \to (\beta \vee \alpha)$
2. $(\beta \to (\beta \vee \alpha)) \to (\alpha \to (\beta \to (\beta \vee \alpha)))$
3. $\alpha \to (\beta \to (\beta \vee \alpha))$

The sentence on line 1 is of the form of the axiom $\alpha \to (\alpha \vee \beta)$ (with the roles of α and β reversed). The sentence on line 2 is of the form of the axiom $\alpha \to (\beta \to \alpha)$. Thus, both lines are justified. Line 3 is justified by modus ponens:

if we abbreviate it as δ, then line 2 has the form $\gamma \to \delta$, where γ is $\beta \to (\beta \vee \alpha)$, i.e., line 1.

A set Γ is inconsistent if $\Gamma \vdash \bot$. A complete axiom system will also prove that $\bot \to \alpha$ for any α, and so if Γ is inconsistent, then $\Gamma \vdash \alpha$ for any α.

Systems of axiomatic derivations for logic were first given by Gottlob Frege in his 1879 *Begriffsschrift*, which for this reason is often considered the first work of modern logic. They were perfected in Alfred North Whitehead and Bertrand Russell's *Principia Mathematica* and by David Hilbert and his students in the 1920s. They are thus often called "Frege systems" or "Hilbert systems." They are very versatile in that it is often easy to find an axiomatic system for a logic. Because derivations have a very simple structure and only one or two inference rules, it is also relatively easy to prove things *about* them. However, they are very hard to use in practice, i.e., it is difficult to find and write proofs.

Chapter 10

Axiomatic Derivations

§10.0 Rules and Derivations

Axiomatic derivations are perhaps the simplest derivation system for logic. A derivation is just a sequence of wffs. To count as a derivation, every wff in the sequence must either be an instance of an axiom, or must follow from one or more wffs that precede it in the sequence by a rule of inference. A derivation derives its last wff.

Definition 100A (Derivability). *If Γ is a set of wffs of $\mathcal{L}$ then a* derivation *from Γ is a finite sequence $\alpha_1, \ldots, \alpha_n$ of wffs where for each $i \leq n$ one of the following holds:*

1. *$\alpha_i \in \Gamma$; or*
2. *α_i is an axiom; or*
3. *α_i follows from some α_j (and α_k) with $j < i$ (and $k < i$) by a rule of inference.*

What counts as a correct derivation depends on which inference rules we allow (and of course what we take to be axioms). And an inference rule is an if-then statement that tells us that, under certain conditions, a step A_i in a derivation is a correct inference step.

Definition 100B (Rule of inference). *A* rule of inference *gives a sufficient condition for what counts as a correct inference step in a derivation from Γ.*

For instance, since any one-element sequence α with $\alpha \in \Gamma$ trivially counts as a derivation, the following might be a very simple rule of inference:

> If $\alpha \in \Gamma$, then α is always a correct inference step in any derivation from Γ.

Similarly, if α is one of the axioms, then α by itself is a derivation, and so this is also a rule of inference:

> If α is an axiom, then α is a correct inference step.

It gets more interesting if the rule of inference appeals to wffs that appear before the step considered. The following rule is called *modus ponens:*

> If $\beta \to \alpha$ and β occur higher up in the derivation, then α is a correct inference step.

If this is the only rule of inference, then our definition of derivation above amounts to this: $\alpha_1, \ldots, \alpha_n$ is a derivation iff for each $i \leq n$ one of the following holds:

1. $\alpha_i \in \Gamma$; or
2. α_i is an axiom; or
3. for some $j < i$, α_j is $\beta \to \alpha_i$, and for some $k < i$, α_k is β.

The last clause says that α_i follows from α_j (β) and α_k ($\beta \to \alpha_i$) by modus ponens. If we can go from 1 to n, and each time we find a wff α_i that is either in Γ, an axiom, or which a rule of inference tells us that it is a correct inference step, then the entire sequence counts as a correct derivation.

Definition 100C (Derivability). *A wff α is* derivable *from Γ, written $\Gamma \vdash \alpha$, if there is a derivation from Γ ending in α.*

Definition 100D (Theorems). *A wff α is a* theorem *if there is a derivation of α from the empty set. We write $\vdash \alpha$ if α is a theorem and $\nvdash \alpha$ if it is not.*

§10.1 Axiom and Rules for the Propositional Connectives

Definition 101A (Axioms). *The set of Ax_0 of* axioms *for the propositional*

connectives comprises all wffs of the following forms:

$$(\alpha \wedge \beta) \to \alpha \tag{10.1}$$
$$(\alpha \wedge \beta) \to \beta \tag{10.2}$$
$$\alpha \to (\beta \to (\alpha \wedge \beta)) \tag{10.3}$$
$$\alpha \to (\alpha \vee \beta) \tag{10.4}$$
$$\alpha \to (\beta \vee \alpha) \tag{10.5}$$
$$(\alpha \to \gamma) \to ((\beta \to \gamma) \to ((\alpha \vee \beta) \to \gamma)) \tag{10.6}$$
$$\alpha \to (\beta \to \alpha) \tag{10.7}$$
$$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)) \tag{10.8}$$
$$(\alpha \to \beta) \to ((\alpha \to \neg\beta) \to \neg\alpha) \tag{10.9}$$
$$\neg\alpha \to (\alpha \to \beta) \tag{10.10}$$
$$\top \tag{10.11}$$
$$\bot \to \alpha \tag{10.12}$$
$$(\alpha \to \bot) \to \neg\alpha \tag{10.13}$$
$$\neg\neg\alpha \to \alpha \tag{10.14}$$

Definition 101B (Modus ponens). *If β and $\beta \to \alpha$ already occur in a derivation, then α is a correct inference step.*

We'll abbreviate the rule modus ponens as "MP."

§10.2 Axioms and Rules for Quantifiers

Definition 102A (Axioms for quantifiers). *The* axioms *governing quantifiers are all instances of the following:*

$$\forall x\, \beta \to \beta(t), \tag{10.15}$$
$$\beta(t) \to \exists x\, \beta. \tag{10.16}$$

for any closed term t.

Definition 102B (Rules for quantifiers).

If $\beta \to \alpha(a)$ already occurs in the derivation and a does not occur in Γ or β, then $\beta \to \forall x\, \alpha(x)$ is a correct inference step.

If $\alpha(a) \to \beta$ already occurs in the derivation and a does not occur in Γ or β, then $\exists x\, \alpha(x) \to \beta$ is a correct inference step.

We'll abbreviate either of these by "QR."

§10.3 Examples of Derivations

Example 10.3.1. Suppose we want to prove $(\neg\delta \vee \varphi) \to (\delta \to \varphi)$. Clearly, this is not an instance of any of our axioms, so we have to use the MP rule to derive it. Our only rule is MP, which given α and $\alpha \to \beta$ allows us to justify β. One strategy would be to use eq. (10.6) with α being $\neg\delta$, β being φ, and γ being $\delta \to \varphi$, i.e., the instance

$$(\neg\delta \to (\delta \to \varphi)) \to ((\varphi \to (\delta \to \varphi)) \to ((\neg\delta \vee \varphi) \to (\delta \to \varphi))).$$

Why? Two applications of MP yield the last part, which is what we want. And we easily see that $\neg\delta \to (\delta \to \varphi)$ is an instance of eq. (10.10), and $\varphi \to (\delta \to \varphi)$ is an instance of eq. (10.7). So our derivation is:

1.	$\neg\delta \to (\delta \to \varphi)$	eq. (10.10)
2.	$(\neg\delta \to (\delta \to \varphi)) \to$ $((\varphi \to (\delta \to \varphi)) \to ((\neg\delta \vee \varphi) \to (\delta \to \varphi)))$	eq. (10.6)
3.	$((\varphi \to (\delta \to \varphi)) \to ((\neg\delta \vee \varphi) \to (\delta \to \varphi))$	1, 2, MP
4.	$\varphi \to (\delta \to \varphi)$	eq. (10.7)
5.	$(\neg\delta \vee \varphi) \to (\delta \to \varphi)$	3, 4, MP

Example 10.3.2. Let's try to find a derivation of $\delta \to \delta$. It is not an instance of an axiom, so we have to use MP to derive it. eq. (10.7) is an axiom of the form $\alpha \to \beta$ to which we could apply MP. To be useful, of course, the β which MP would justify as a correct step in this case would have to be $\delta \to \delta$, since this is what we want to derive. That means α would also have to be δ, i.e., we might look at this instance of eq. (10.7):

$$\delta \to (\delta \to \delta)$$

In order to apply MP, we would also need to justify the corresponding second premise, namely α. But in our case, that would be δ, and we won't be able to derive δ by itself. So we need a different strategy.

The other axiom involving just $\to$ is eq. (10.8), i.e.,

$$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$$

We could get to the last nested conditional by applying MP twice. Again, that would mean that we want an instance of eq. (10.8) where $\alpha \to \gamma$ is $\delta \to \delta$, the wff we are aiming for. Then of course, α and γ are both δ. How should we pick β so that both $\alpha \to (\beta \to \gamma)$ and $\alpha \to \beta$, i.e., in our case $\delta \to (\beta \to \delta)$ and $\delta \to \beta$, are also derivable? Well, the first of these is already an instance of eq. (10.7), whatever we decide β to be. And $\delta \to \beta$ would be another instance of eq. (10.7) if β were $(\delta \to \delta)$. So, our derivation is:

1.	$\delta \to ((\delta \to \delta) \to \delta)$	eq. (10.7)
2.	$(\delta \to ((\delta \to \delta) \to \delta)) \to$ $((\delta \to (\delta \to \delta)) \to (\delta \to \delta))$	eq. (10.8)
3.	$(\delta \to (\delta \to \delta)) \to (\delta \to \delta)$	1, 2, MP
4.	$\delta \to (\delta \to \delta)$	eq. (10.7)
5.	$\delta \to \delta$	3, 4, MP

Example 10.3.3. Sometimes we want to show that there is a derivation of some wff from some other wffs Γ. For instance, let's show that we can derive $\alpha \to \gamma$ from $\Gamma = \{\alpha \to \beta, \beta \to \gamma\}$.

1.	$\alpha \to \beta$	Hyp
2.	$\beta \to \gamma$	Hyp
3.	$(\beta \to \gamma) \to (\alpha \to (\beta \to \gamma))$	eq. (10.7)
4.	$\alpha \to (\beta \to \gamma)$	2, 3, MP
5.	$(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$	eq. (10.8)
6.	$((\alpha \to \beta) \to (\alpha \to \gamma))$	4, 5, MP
7.	$\alpha \to \gamma$	1, 6, MP

The lines labelled "Hyp" (for "hypothesis") indicate that the wff on that line is an element of Γ.

Proposition 103D. *If $\Gamma \vdash \alpha \to \beta$ and $\Gamma \vdash \beta \to \gamma$, then $\Gamma \vdash \alpha \to \gamma$*

Proof. Suppose $\Gamma \vdash \alpha \to \beta$ and $\Gamma \vdash \beta \to \gamma$. Then there is a derivation of $\alpha \to \beta$ from Γ; and a derivation of $\beta \to \gamma$ from Γ as well. Combine these into a single derivation by concatenating them. Now add lines 3–7 of the derivation in the preceding example. This is a derivation of $\alpha \to \gamma$—which is the last line of the new derivation—from Γ. Note that the justifications of lines 4 and 7 remain valid if the reference to line number 2 is replaced by reference to the last line of the derivation of $\alpha \to \beta$, and reference to line number 1 by reference to the last line of the derivation of $B \to \gamma$. □

§10.4 Derivations with Quantifiers

Example 10.4.1. Let us give a derivation of $(\forall x\, \alpha(x) \wedge \forall y\, \beta(y)) \to \forall x\, (\alpha(x) \wedge \beta(x))$.

First, note that

$$(\forall x\, \alpha(x) \wedge \forall y\, \beta(y)) \to \forall x\, \alpha(x)$$

is an instance of eq. (10.1), and

$$\forall x\, \alpha(x) \to \alpha(a)$$

of eq. (10.15). So, by Proposition 103D, we know that

$$(\forall x\, \alpha(x) \wedge \forall y\, \beta(y)) \to \alpha(a)$$

is derivable. Likewise, since

$$(\forall x\, \alpha(x) \wedge \forall y\, \beta(y)) \to \forall y\, \beta(y) \qquad \text{and}$$
$$\forall y\, \beta(y) \to \beta(a)$$

are instances of eq. (10.2) and eq. (10.15), respectively,

$$(\forall x\, \alpha(x) \land \forall y\, \beta(y)) \to \beta(a)$$

is derivable by Proposition 103D. Using an appropriate instance of eq. (10.3) and two applications of MP, we see that

$$(\forall x\, \alpha(x) \land \forall y\, \beta(y)) \to (\alpha(a) \land \beta(a))$$

is derivable. We can now apply QR to obtain

$$(\forall x\, \alpha(x) \land \forall y\, \beta(y)) \to \forall x\, (\alpha(x) \land \beta(x)).$$

§10.5 Proof-Theoretic Notions

Just as we've defined a number of important semantic notions (validity, entailment, satisfiability), 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 formulas. It was an important discovery that these notions coincide. That they do is the content of the *soundness* and *completeness theorems*.

Definition 105A (Derivability). *A wff α is* derivable *from Γ, written $\Gamma \vdash \alpha$, if there is a derivation from Γ ending in α.*

Definition 105B (Theorems). *A wff α is a* theorem *if there is a derivation of α from the empty set. We write $\vdash \alpha$ if α is a theorem and $\nvdash \alpha$ if it is not.*

Definition 105C (Consistency). *A set Γ of wffs is* consistent *if and only if $\Gamma \nvdash \bot$; it is* inconsistent *otherwise.*

Proposition 105D (Reflexivity). *If $\alpha \in \Gamma$, then $\Gamma \vdash \alpha$.*

Proof. The wff α by itself is a derivation of α from Γ. □

Proposition 105E (Monotonicity). *If $\Gamma \subseteq \Delta$ and $\Gamma \vdash \alpha$, then $\Delta \vdash \alpha$.*

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

Proposition 105F (Transitivity). *If $\Gamma \vdash \alpha$ and $\{\alpha\} \cup \Delta \vdash \beta$, then $\Gamma \cup \Delta \vdash \beta$.*

Proof. Suppose $\{\alpha\} \cup \Delta \vdash \beta$. Then there is a derivation $\beta_1, \ldots, \beta_l = \beta$ from $\{\alpha\} \cup \Delta$. Some of the steps in that derivation will be correct because of a rule which refers to a prior line $\beta_i = \alpha$. By hypothesis, there is a derivation

of α from Γ, i.e., a derivation $\alpha_1, \ldots, \alpha_k = \alpha$ where every α_i is an axiom, an element of Γ, or correct by a rule of inference. Now consider the sequence

$$\alpha_1, \ldots, \alpha_k = \alpha, \beta_1, \ldots, \beta_l = \beta.$$

This is a correct derivation of β from $\Gamma \cup \Delta$ since every $B_i = \alpha$ is now justified by the same rule which justifies $\alpha_k = \alpha$. □

Note that this means that in particular if $\Gamma \vdash \alpha$ and $\alpha \vdash \beta$, then $\Gamma \vdash \beta$. It follows also that if $\alpha_1, \ldots, \alpha_n \vdash \beta$ and $\Gamma \vdash \alpha_i$ for each i, then $\Gamma \vdash \beta$.

Proposition 105G. *Γ is inconsistent iff $\Gamma \vdash \alpha$ for every α.*

Proof. Exercise. □

Proposition 105H (Compactness).

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

Proof. 1. If $\Gamma \vdash \alpha$, then there is a finite sequence of wffs $\alpha_1, \ldots, \alpha_n$ so that $\alpha \equiv \alpha_n$ and each α_i is either a logical axiom, an element of Γ or follows from previous wffs by modus ponens. Take Γ_0 to be those α_i which are in Γ. Then the derivation is likewise a derivation from Γ_0, and so $\Gamma_0 \vdash \alpha$.

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

§10.6 The Deduction Theorem

As we've seen, giving derivations in an axiomatic system is cumbersome, and derivations may be hard to find. Rather than actually write out long lists of wffs, it is generally easier to argue that such derivations exist, by making use of a few simple results. We've already established three such results: Proposition 105D says we can always assert that $\Gamma \vdash \alpha$ when we know that $\alpha \in \Gamma$. Proposition 105E says that if $\Gamma \vdash \alpha$ then also $\Gamma \cup \{\beta\} \vdash \alpha$. And Proposition 105F implies that if $\Gamma \vdash \alpha$ and $\alpha \vdash \beta$, then $\Gamma \vdash \beta$. Here's another simple result, a "meta"-version of modus ponens:

Proposition 106A. *If $\Gamma \vdash \alpha$ and $\Gamma \vdash \alpha \to \beta$, then $\Gamma \vdash \beta$.*

Proof. We have that $\{\alpha, \alpha \to \beta\} \vdash \beta$:

1.	α	Hyp.
2.	$\alpha \to \beta$	Hyp.
3.	β	1, 2, MP

By Proposition 105F, $\Gamma \vdash \beta$. □

The most important result we'll use in this context is the deduction theorem:

Theorem 106B (Deduction Theorem). *$\Gamma \cup \{\alpha\} \vdash \beta$ if and only if $\Gamma \vdash \alpha \to \beta$.*

Proof. The "if" direction is immediate. If $\Gamma \vdash \alpha \to \beta$ then also $\Gamma \cup \{\alpha\} \vdash \alpha \to \beta$ by Proposition 105E. Also, $\Gamma \cup \{\alpha\} \vdash \alpha$ by Proposition 105D. So, by Proposition 106A, $\Gamma \cup \{\alpha\} \vdash \beta$.

For the "only if" direction, we proceed by induction on the length of the derivation of β from $\Gamma \cup \{\alpha\}$.

For the induction basis, we prove the claim for every derivation of length 1. A derivation of β from $\Gamma \cup \{\alpha\}$ of length 1 consists of β by itself; and if it is correct β is either $\in \Gamma \cup \{\alpha\}$ or is an axiom. If $\beta \in \Gamma$ or is an axiom, then $\Gamma \vdash \beta$. We also have that $\Gamma \vdash \beta \to (\alpha \to \beta)$ by eq. (10.7), and Proposition 106A gives $\Gamma \vdash \alpha \to \beta$. If $\beta \in \{\alpha\}$ then $\Gamma \vdash \alpha \to \beta$ because then last sentence $\alpha \to \beta$ is the same as $\alpha \to \alpha$, and we have derived that in Example 10.3.2.

For the inductive step, suppose a derivation of β from $\Gamma \cup \{\alpha\}$ ends with a step β which is justified by modus ponens. (If it is not justified by modus ponens, $\beta \in \Gamma$, $\beta \equiv \alpha$, or β is an axiom, and the same reasoning as in the induction basis applies.) Then some previous steps in the derivation are $\gamma \to \beta$ and γ, for some wff γ, i.e., $\Gamma \cup \{\alpha\} \vdash \gamma \to \beta$ and $\Gamma \cup \{\alpha\} \vdash \gamma$, and the respective derivations are shorter, so the inductive hypothesis applies to them. We thus have both:

$$\Gamma \vdash \alpha \to (\gamma \to \beta);$$
$$\Gamma \vdash \alpha \to \gamma.$$

But also

$$\Gamma \vdash (\alpha \to (\gamma \to \beta)) \to ((\alpha \to \gamma) \to (\alpha \to \beta)),$$

by eq. (10.8), and two applications of Proposition 106A give $\Gamma \vdash \alpha \to \beta$, as required. □

Notice how eq. (10.7) and eq. (10.8) were chosen precisely so that the Deduction Theorem would hold.

The following are some useful facts about derivability, which we leave as exercises.

Proposition 106C.

1. $\vdash (\alpha \to \beta) \to ((\beta \to \gamma) \to (\alpha \to \gamma))$;
2. *If* $\Gamma \cup \{\neg\alpha\} \vdash \neg\beta$ *then* $\Gamma \cup \{\beta\} \vdash \alpha$ *(Contraposition);*
3. $\{\alpha, \neg\alpha\} \vdash \beta$ *(Ex Falso Quodlibet, Explosion);*
4. $\{\neg\neg\alpha\} \vdash \alpha$ *(Double Negation Elimination);*
5. *If* $\Gamma \vdash \neg\neg\alpha$ *then* $\Gamma \vdash \alpha$;

§10.7 The Deduction Theorem with Quantifiers

Theorem 107A (Deduction Theorem). *If $\Gamma \cup \{\alpha\} \vdash \beta$, then $\Gamma \vdash \alpha \to \beta$.*

Proof. We again proceed by induction on the length of the derivation of β from $\Gamma \cup \{\alpha\}$.

The proof of the induction basis is identical to that in the proof of Theorem 106B.

For the inductive step, suppose again that the derivation of β from $\Gamma \cup \{\alpha\}$ ends with a step β which is justified by an inference rule. If the inference rule is modus ponens, we proceed as in the proof of Theorem 106B. If the inference rule is QR, we know that $\beta \equiv \gamma \to \forall x\, \delta(x)$ and a wff of the form $\gamma \to \delta(a)$ appears earlier in the derivation, where a does not occur in γ, α, or Γ. We thus have that

$$\Gamma \cup \{\alpha\} \vdash \gamma \to \delta(a),$$

and the induction hypothesis applies, i.e., we have that

$$\Gamma \vdash \alpha \to (\gamma \to \delta(a)).$$

By

$$\vdash (\alpha \to (\gamma \to \delta(a))) \to ((\alpha \wedge \gamma) \to \delta(a))$$

and modus ponens we get

$$\Gamma \vdash (\alpha \wedge \gamma) \to \delta(a).$$

Since the eigenvariable condition still applies, we can add a step to this derivation justified by QR, and get

$$\Gamma \vdash (\alpha \wedge \gamma) \to \forall x\, \delta(x).$$

We also have

$$\vdash ((\alpha \wedge \gamma) \to \forall x\, \delta(x)) \to (\alpha \to (\gamma \to \forall x\, \delta(x)),$$

so by modus ponens,

$$\Gamma \vdash \alpha \to (\gamma \to \forall x\, \delta(x)),$$

i.e., $\Gamma \vdash \beta$.

We leave the case where β is justified by the rule QR, but is of the form $\exists x\, \delta(x) \to \gamma$, as an exercise. □

§10.8 Derivability and Consistency

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

Proposition 108A. *If $\Gamma \vdash \alpha$ and $\Gamma \cup \{\alpha\}$ is inconsistent, then Γ is inconsistent.*

Proof. If $\Gamma \cup \{\alpha\}$ is inconsistent, then $\Gamma \cup \{\alpha\} \vdash \bot$. By Proposition 105D, $\Gamma \vdash \beta$ for every $\beta \in \Gamma$. Since also $\Gamma \vdash \alpha$ by hypothesis, $\Gamma \vdash \beta$ for every $\beta \in \Gamma \cup \{\alpha\}$. By Proposition 105F, $\Gamma \vdash \bot$, i.e., Γ is inconsistent. □

Proposition 108B. *$\Gamma \vdash \alpha$ iff $\Gamma \cup \{\neg\alpha\}$ is inconsistent.*

Proof. First suppose $\Gamma \vdash \alpha$. Then $\Gamma \cup \{\neg\alpha\} \vdash \alpha$ by Proposition 105E. $\Gamma \cup \{\neg\alpha\} \vdash \neg\alpha$ by Proposition 105D. We also have $\vdash \neg\alpha \to (\alpha \to \bot)$ by eq. (10.10). So by two applications of Proposition 106A, we have $\Gamma \cup \{\neg\alpha\} \vdash \bot$.

Now assume $\Gamma \cup \{\neg\alpha\}$ is inconsistent, i.e., $\Gamma \cup \{\neg\alpha\} \vdash \bot$. By the deduction theorem, $\Gamma \vdash \neg\alpha \to \bot$. $\Gamma \vdash (\neg\alpha \to \bot) \to \neg\neg\alpha$ by eq. (10.13), so $\Gamma \vdash \neg\neg\alpha$ by Proposition 106A. Since $\Gamma \vdash \neg\neg\alpha \to \alpha$ (eq. (10.14)), we have $\Gamma \vdash \alpha$ by Proposition 106A again. □

Proposition 108C. *If $\Gamma \vdash \alpha$ and $\neg\alpha \in \Gamma$, then Γ is inconsistent.*

Proof. $\Gamma \vdash \neg\alpha \to (\alpha \to \bot)$ by eq. (10.10). $\Gamma \vdash \bot$ by two applications of Proposition 106A. □

Proposition 108D. *If $\Gamma \cup \{\alpha\}$ and $\Gamma \cup \{\neg\alpha\}$ are both inconsistent, then Γ is inconsistent.*

Proof. Exercise. □

§10.9 Derivability and the Propositional Connectives

We establish that the derivability relation $\vdash$ of axiomatic deduction is strong enough to establish some basic facts involving the propositional connectives, such as that $\alpha \wedge \beta \vdash \alpha$ and $\alpha, \alpha \to \beta \vdash \beta$ (modus ponens). These facts are needed for the proof of the completeness theorem.

Proposition 109A.

1. *Both $\alpha \wedge \beta \vdash \alpha$ and $\alpha \wedge \beta \vdash \beta$*
2. *$\alpha, \beta \vdash \alpha \wedge \beta$.*

Proof. 1. From eq. (10.1) and eq. (10.1) by modus ponens.

2. From eq. (10.3) by two applications of modus ponens. □

Proposition 109B.

1. *$\alpha \vee \beta, \neg\alpha, \neg\beta$ is inconsistent.*

2. *Both $\alpha \vdash \alpha \vee \beta$ and $\beta \vdash \alpha \vee \beta$.*

Proof. 1. From eq. (10.9) we get $\vdash \neg\alpha \to (\alpha \to \bot)$ and $\vdash \neg\beta \to (\beta \to \bot)$. So by the deduction theorem, we have $\{\neg\alpha\} \vdash \alpha \to \bot$ and $\{\neg\beta\} \vdash \beta \to \bot$. From eq. (10.6) we get $\{\neg\alpha, \neg\beta\} \vdash (\alpha \vee \beta) \to \bot$. By the deduction theorem, $\{\alpha \vee \beta, \neg\alpha, \neg\beta\} \vdash \bot$.

2. From eq. (10.4) and eq. (10.5) by modus ponsens. □

Proposition 109C.

1. *$\alpha, \alpha \to \beta \vdash \beta$.*

2. *Both $\neg\alpha \vdash \alpha \to \beta$ and $\beta \vdash \alpha \to \beta$.*

Proof. 1. We can derive:

1.	α	Hyp
2.	$\alpha \to \beta$	Hyp
3.	β	1, 2, MP

2. By eq. (10.10) and eq. (10.7) and the deduction theorem, respectively. □

§10.10 Derivability and the Quantifiers

The completeness theorem also requires that axiomatic deductions yield the facts about $\vdash$ established in this section.

Theorem 1010A. *If c is a constant symbol not occurring in Γ or $\alpha(x)$ and $\Gamma \vdash \alpha(c)$, then $\Gamma \vdash \forall x\, \alpha(x)$.*

Proof. By the deduction theorem, $\Gamma \vdash \top \to \alpha(c)$. Since c does not occur in Γ or $\top$, we get $\Gamma \vdash \top \to \alpha(c)$. By the deduction theorem again, $\Gamma \vdash \forall x\, \alpha(x)$. □

Proposition 1010B.
$\forall x\, \alpha(x) \vdash \alpha(t)$.

Proof. By eq. (10.15) and the deduction theorem. □

§10.11 Soundness

A derivation system, such as axiomatic deduction, 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 α is valid;
2. if α is derivable from some others Γ, it is also a consequence of them;
3. if a set of wffs Γ 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.

Proposition 1011A. *If α is an axiom, then $\models_{\mathfrak{A}} \alpha\,[s]$ for each structure $\mathfrak{A}$ and assignment s.*

Proof. We have to verify that all the axioms are valid. For instance, here is the case for eq. (10.15): suppose t is free for x in α, and assume $\models_{\mathfrak{A}} \forall x\, \alpha\,[s]$. Then by definition of satisfaction, for each $s' \sim_x s$, also $\models_{\mathfrak{A}} \alpha\,[s']$, and in particular this holds when $s'(x) = \bar{s}(t)$. By Proposition 75D, $\models_{\mathfrak{A}} \alpha[t/x]\,[s]$. This shows that $\models_{\mathfrak{A}} (\forall x\, \alpha \to \alpha[t/x])\,[s]$. □

Theorem 1011B (Soundness). *If $\Gamma \vdash \alpha$ then $\Gamma \vDash \alpha$.*

Proof. By induction on the length of the derivation of α from Γ. If there are no steps justified by inferences, then all wffs in the derivation are either instances of axioms or are in Γ. By the previous proposition, all the axioms are valid, and hence if α is an axiom then $\Gamma \vDash \alpha$. If $\alpha \in \Gamma$, then trivially $\Gamma \vDash \alpha$.

If the last step of the derivation of α is justified by modus ponens, then there are wffs β and $\beta \to \alpha$ in the derivation, and the induction hypothesis applies to the part of the derivation ending in those wffs (since they contain at least one fewer steps justified by an inference). So, by induction hypothesis, $\Gamma \vDash \beta$ and $\Gamma \vDash \beta \to \alpha$. Then $\Gamma \vDash \alpha$ by Theorem 76G.

Now suppose the last step is justified by QR. Then that step has the form $\gamma \to \forall x\, B(x)$ and there is a preceding step $\gamma \to \beta(c)$ with c not in Γ, γ, or $\forall x\, B(x)$. By induction hypothesis, $\Gamma \vDash \gamma \to \forall x\, B(x)$. By Theorem 76G, $\Gamma \cup \{\gamma\} \vDash \beta(c)$.

Consider some structure $\mathfrak{A}$ such that $\models_{\mathfrak{A}} \Gamma \cup \{\gamma\}$. We need to show that $\models_{\mathfrak{A}} \forall x\, \beta(x)$. Since $\forall x\, \beta(x)$ is a sentence, this means we have to show that for every variable assignment s, $\models_{\mathfrak{A}} \beta(x)\,[s]$ (Proposition 74F). Since $\Gamma \cup \{\gamma\}$ consists entirely of sentences, $\models_{\mathfrak{A}} \delta\,[s]$ for all $\delta \in \Gamma$ by Definition 73E. Let $\mathfrak{A}'$ be like $\mathfrak{A}$ except that $c^{\mathfrak{A}'} = s(x)$. Since c does not occur in Γ or γ, $\models_{\mathfrak{A}'} \Gamma \cup \{\gamma\}$ by corollary 75B. Since $\Gamma \cup \{\gamma\} \vDash \beta(c)$, $\models_{\mathfrak{A}'} B(c)$. Since $\beta(c)$ is a sentence, $\models_{\mathfrak{A}} \beta(c)\,[s]$ by Proposition 74E. $\models_{\mathfrak{A}'} \beta(x)\,[s]$ iff $\models_{\mathfrak{A}'} \beta(c)$ by Proposition 75D

(recall that $\beta(c)$ is just $\beta(x)[c/x]$). So, $\models_{\mathfrak{A}'} \beta(x)\,[s]$. Since c does not occur in $\beta(x)$, by Proposition 75A, $\models_{\mathfrak{A}} \beta(x)\,[s]$. But s was an arbitrary variable assignment, so $\models_{\mathfrak{A}} \forall x\,\beta(x)$. Thus $\Gamma \cup \{\gamma\} \models \forall x\,\beta(x)$. By Theorem 76G, $\Gamma \models \gamma \to \forall x\,\beta(x)$.

The case where α is justified by QR but is of the form $\exists x\,\beta(x) \to \gamma$ is left as an exercise. □

Corollary 1011C. *If $\vdash \alpha$, then α is valid.*

Corollary 1011D. *If Γ is satisfiable, then it is consistent.*

Proof. We prove the contrapositive. Suppose that Γ is not consistent. Then $\Gamma \vdash \bot$, i.e., there is a derivation of $\bot$ from Γ. By Theorem 1011B, any structure $\mathfrak{A}$ that satisfies Γ must satisfy $\bot$. Since $\not\models_{\mathfrak{A}} \bot$ for every structure $\mathfrak{A}$, no $\mathfrak{A}$ can satisfy Γ, i.e., Γ is not satisfiable. □

§10.12 Derivations with Equality symbol

In order to accommodate $=$ in derivations, we simply add new axiom schemas. The definition of derivation and $\vdash$ remains the same, we just also allow the new axioms.

Definition 1012A (Axioms for equality symbol).

$$t = t, \tag{10.17}$$
$$t_1 = t_2 \to (\beta(t_1) \to \beta(t_2)), \tag{10.18}$$

for any closed terms t, t_1, t_2.

Proposition 1012B. *The axioms eq. (10.17) and eq. (10.18) are valid.*

Proof. Exercise. □

Proposition 1012C. *$\Gamma \vdash t = t$, for any term t and set Γ.*

Proposition 1012D. *If $\Gamma \vdash \alpha(t_1)$ and $\Gamma \vdash t_1 = t_2$, then $\Gamma \vdash \alpha(t_2)$.*

Proof. The wff

$$(t_1 = t_2 \to (\alpha(t_1) \to \alpha(t_2)))$$

is an instance of eq. (10.18). The conclusion follows by two applications of MP. □

Problems

Problem 1. Show that the following hold by exhibiting derivations from the axioms:

1. $(\alpha \wedge \beta) \to (\beta \wedge \alpha)$
2. $((\alpha \wedge \beta) \to \gamma) \to (\alpha \to (\beta \to \gamma))$
3. $\neg(\alpha \vee \beta) \to \neg\alpha$

Problem 2. Prove Proposition 105G.

Problem 3. Prove Proposition 106C

Problem 4. Complete the proof of Theorem 107A.

Problem 5. Prove that $\Gamma \vdash \neg\alpha$ iff $\Gamma \cup \{\alpha\}$ is inconsistent.

Problem 6. Prove Proposition 108D

Problem 7. Complete the proof of Theorem 1011B.

Problem 8. Prove Proposition 1012B.

Chapter 11

The Completeness Theorem

§11.0 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 derivation system: if a sentence α follows from some sentences Γ, then there is also a derivation that establishes $\Gamma \vdash \alpha$. Thus, the derivation system is as strong as it can possibly be without proving things that don't actually follow.

In its second formulation, it can be stated as a model existence result: every consistent set of sentences is satisfiable. Consistency is a proof-theoretic notion: it says that our derivation system is unable to produce certain derivations. But who's to say that just because there are no derivations of a certain sort from Γ, it's guaranteed that there is a structure $\mathfrak{A}$? Before the completeness theorem was first proved—in fact before we had the derivation systems we now do—the great German mathematician David Hilbert held the view that consistency of mathematical theories guarantees the existence of the objects they are about. He put it as follows in a letter to Gottlob Frege:

> If the arbitrarily given axioms do not contradict one another with all their consequences, then they are true and the things defined by the axioms exist. This is for me the criterion of truth and existence.

Frege vehemently disagreed. The second formulation of the completeness theorem shows that Hilbert was right in at least the sense that if the axioms are consistent, then *some* structure exists that makes them all true.

These aren't the only reasons the completeness theorem—or rather, its proof—is important. It has a number of important consequences, some of which we'll discuss separately. For instance, since any derivation that shows $\Gamma \vdash \alpha$ is finite and so can only use finitely many of the sentences in Γ, it follows by the completeness theorem that if α is a consequence of Γ, it is already a

consequence of a finite subset of Γ. This is called *compactness*. Equivalently, if every finite subset of Γ is consistent, then Γ itself must be consistent.

Although the compactness theorem follows from the completeness theorem via the detour through derivations, it is also possible to use the *the proof of* the completeness theorem to establish it directly. For what the proof does is take a set of sentences with a certain property—consistency—and constructs a structure out of this set that has certain properties (in this case, that it satisfies the set). Almost the very same construction can be used to directly establish compactness, by starting from "finitely satisfiable" sets of sentences instead of consistent ones. The construction also yields other consequences, e.g., that any satisfiable set of sentences has a finite or denumerable model. (This result is called the Löwenheim-Skolem theorem.) In general, the construction of structures from sets of sentences is used often in logic, and sometimes even in philosophy.

§11.1 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 $\Gamma \vDash \alpha$ then $\Gamma \vdash \alpha$," it may be hard to even come up with an idea: for to show that $\Gamma \vdash \alpha$ we have to find a derivation, and it does not look like the hypothesis that $\Gamma \vDash \alpha$ 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 approach. The shift in perspective required is this: completeness can also be formulated as: "if Γ is consistent, it is satisfiable." Perhaps we can use the information in Γ together with the hypothesis that it is consistent to construct a structure that satisfies every sentence in Γ. After all, we know what kind of structure 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. Suppose the atomic sentences are all of the form $Pa_1 \dots a_n$ where the a_i are constant symbols. All we have to do is come up with a domain $|\mathfrak{A}|$ and an assignment for P so that $\models_{\mathfrak{A}} Pa_1 \dots a_n$. But that's not very hard: put $|\mathfrak{A}| = \mathbb{N}$, $c_i^{\mathfrak{A}} = i$, and for every $Pa_1 \dots a_n \in \Gamma$, put the tuple $\langle k_1, \dots, k_n \rangle$ into $P^{\mathfrak{A}}$, 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 wff $\neg\beta$, with β atomic. We might worry that the construction of $\mathfrak{A}$ interferes with the possibility of making $\neg\beta$ true. But here's where the consistency of Γ comes in: if $\neg\beta \in \Gamma$, then $\beta \notin \Gamma$, or else Γ would be inconsistent. And if $\beta \notin \Gamma$, then according to our construction of $\mathfrak{A}$, $\not\models_{\mathfrak{A}} \beta$, so $\models_{\mathfrak{A}} \neg\beta$. So far so good.

What if Γ contains complex, non-atomic formulas? Say it contains $\alpha \wedge \beta$. To make that true, we should proceed as if both α and β were in Γ. And if $\alpha \vee \beta \in \Gamma$, 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 wffs to Γ so as to (a) keep the resulting set consistent and (b) make sure that for every possible atomic sentence α, either α is in the resulting set, or $\neg\alpha$ is, and (c) such that, whenever $\alpha \wedge \beta$ is in the set, so are both α and β, if $\alpha \vee \beta$ is in the set, at least one of α or β is also, etc. We keep doing this (potentially forever). Call the set of all wffs so added Γ^*. Then our construction above would provide us with a structure $\mathfrak{A}$ for which we could prove, by induction, that it satisfies all sentences in Γ^*, and hence also all sentence in Γ since $\Gamma \subseteq \Gamma^*$. It turns out that guaranteeing (a) and (b) is enough. A set of sentences for which (b) holds is called *complete*. So our task will be to extend the consistent set Γ to a consistent and complete set Γ^*.

There is one wrinkle in this plan: if $\exists x\, \alpha(x) \in \Gamma$ we would hope to be able to pick some constant symbol c and add $\alpha(c)$ 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 $\neg\alpha(c) \in \Gamma$. We can't also add $\alpha(c)$, since this would make the set inconsistent, and we wouldn't know whether $\mathfrak{A}$ has to make $\alpha(c)$ or $\neg\alpha(c)$ 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 $\mathfrak{A}$ can assign every constant symbol to itself: $c_i^{\mathfrak{A}} = c_i$. But why not go all the way: let $|\mathfrak{A}|$ 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: $\models_{\mathfrak{A}} t = t'$ iff $t^{\mathfrak{A}} = t'^{\mathfrak{A}}$. 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: $(0 + 0) = 0$). To solve this problem, we change the domain of $\mathfrak{A}$: instead of using terms as the objects in $|\mathfrak{A}|$, 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: 0, $(0 + 0)$, (0×0), etc. This will be the set we assign to 0, and it will turn out that this set is also the value of all the terms in it, e.g., also of $(0 + 0)$. Therefore, the sentence $(0 + 0) = 0$ will be

true in this revised structure.

So here's what we'll do. First we investigate the properties of complete consistent sets, in particular we prove that a complete consistent set contains $\alpha \wedge \beta$ iff it contains both α and β, $\alpha \vee \beta$ iff it contains at least one of them, etc. (Proposition 112B). Then we define and investigate "saturated" sets of sentences. A saturated set is one which contains conditionals that link each quantified sentence to instances of it (Definition 113C). We show that any consistent set Γ can always be extended to a saturated set Γ' (Lemma 113D). If a set is consistent, saturated, and complete it also has the property that it contains $\forall x\, \alpha(x)$ iff it contains $\alpha(t)$ for all closed terms t (Proposition 113E). We'll then take the saturated consistent set Γ' and show that it can be extended to a saturated, consistent, and complete set Γ^* (Lemma 114A). This set Γ^* is what we'll use to define our term model $\mathfrak{A}(\Gamma^*)$. The term model has the set of closed terms as its domain, and the interpretation of its predicate symbols is given by the atomic sentences in Γ^* (Definition 115A). We'll use the properties of saturated, complete consistent sets to show that indeed $\models_{\mathfrak{A}(\Gamma^*)} \alpha$ iff $\alpha \in \Gamma^*$ (Lemma 115D), and thus in particular, $\models_{\mathfrak{A}(\Gamma^*)} \Gamma$. Finally, we'll consider how to define a term model if Γ contains $=$ as well (Definition 116D) and show that it satisfies Γ^* (Lemma 116G).

§11.2 Complete Consistent Sets of Sentences

Definition 112A (Complete set). *A set Γ of sentences is* complete *iff for any sentence α, either $\alpha \in \Gamma$ or $\neg\alpha \in \Gamma$.*

Complete sets of sentences leave no questions unanswered. For any sentence α, Γ "says" if α is true or false. The importance of complete sets extends beyond the proof of the completeness theorem. A theory which is complete and axiomatizable, for instance, is always decidable.

Complete consistent sets are important in the completeness proof since we can guarantee that every consistent set of sentences Γ is contained in a complete consistent set Γ^*. A complete consistent set contains, for each sentence α, either α or its negation $\neg\alpha$, but not both. This is true in particular for atomic sentences, so from a complete consistent set in a language suitably expanded by constant symbols, we can construct a structure where the interpretation of predicate symbols is defined according to which atomic sentences are in Γ^*. This structure can then be shown to make all sentences in Γ^* (and hence also all those in Γ) true. The proof of this latter fact requires that $\neg\alpha \in \Gamma^*$ iff $\alpha \notin \Gamma^*$, $(\alpha \vee \beta) \in \Gamma^*$ iff $\alpha \in \Gamma^*$ or $\beta \in \Gamma^*$, etc.

In what follows, we will often tacitly use the properties of reflexivity, monotonicity, and transitivity of $\vdash$ (see section 10.5).

Proposition 112B. *Suppose Γ is complete and consistent. Then:*

1. *If $\Gamma \vdash \alpha$, then $\alpha \in \Gamma$.*
2. *$\alpha \to \beta \in \Gamma$ iff either $\alpha \notin \Gamma$ or $\beta \in \Gamma$.*

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

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

   Suppose that $\Gamma \vdash \alpha$. Suppose to the contrary that $\alpha \notin \Gamma$. Since Γ is complete, $\neg\alpha \in \Gamma$. By Proposition 108C, Γ is inconsistent. This contradicts the assumption that Γ is consistent. Hence, it cannot be the case that $\alpha \notin \Gamma$, so $\alpha \in \Gamma$.

2. For the forward direction, suppose $\alpha \to \beta \in \Gamma$, and suppose to the contrary that $\alpha \in \Gamma$ and $\beta \notin \Gamma$. On these assumptions, $\alpha \to \beta \in \Gamma$ and $\alpha \in \Gamma$. By Proposition 109C, item (1), $\Gamma \vdash \beta$. But then by (1), $\beta \in \Gamma$, contradicting the assumption that $\beta \notin \Gamma$.

   For the reverse direction, first consider the case where $\alpha \notin \Gamma$. Since Γ is complete, $\neg\alpha \in \Gamma$. By Proposition 109C, item (2), $\Gamma \vdash \alpha \to \beta$. Again by (1), we get that $\alpha \to \beta \in \Gamma$, as required.

   Now consider the case where $\beta \in \Gamma$. By Proposition 109C, item (2) again, $\Gamma \vdash \alpha \to \beta$. By (1), $\alpha \to \beta \in \Gamma$. □

§11.3 Henkin Expansion

Part of the challenge in proving the completeness theorem is that the model we construct from a complete consistent set Γ must make all the quantified wffs in Γ true. In order to guarantee this, we use a trick due to Leon Henkin. In essence, the trick consists in expanding the language by infinitely many constant symbols and adding, for each wff with one free variable $\alpha(x)$ a formula of the form $\neg\forall x\, \alpha(x) \to \neg\alpha(c)$, where c is one of the new constant symbols. When we construct the structure satisfying Γ, this will guarantee that each false universal sentence has a counterexample among the new constants.

Proposition 113A. *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_0, d_1, ..., then Γ is consistent in $\mathcal{L}'$.*

Definition 113B (Saturated set). *A set Γ of wffs of a language $\mathcal{L}$ is* saturated *iff for each wff $\alpha(x) \in \mathrm{Frm}(\mathcal{L})$ with one free variable x there is a constant symbol $c \in \mathcal{L}$ such that $\neg\forall x\, \alpha(x) \to \neg\alpha(c) \in \Gamma$.*

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

Definition 113C. *Let $\mathcal{L}'$ be as in Proposition 113A. Fix an enumeration $\alpha_0(x_0)$, $\alpha_1(x_1)$, ... of all wffs $\alpha_i(x_i)$ of $\mathcal{L}'$ in which one variable (x_i) occurs free. We define the sentences δ_n by induction on n.*

Let c_0 be the first constant symbol among the d_i we added to $\mathcal{L}$ which does not occur in $\alpha_0(x_0)$. Assuming that δ_0, ..., δ_{n-1} have already been defined, let c_n be the first among the new constant symbols d_i that occurs neither in δ_0, ..., δ_{n-1} nor in $\alpha_n(x_n)$.

Now let δ_n be the wff $\neg\forall x_n\, \alpha_n(x_n) \to \neg\alpha_n(c_n)$.

Lemma 113D. *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 Proposition 113A, Γ is still consistent in the richer language. Further, let δ_i be as in Definition 113C. Let

$$\Gamma_0 = \Gamma$$
$$\Gamma_{n+1} = \Gamma_n \cup \{\delta_n\}$$

i.e., $\Gamma_{n+1} = \Gamma \cup \{\delta_0, \ldots, \delta_n\}$, and let $\Gamma' = \bigcup_n \Gamma_n$. Γ' is clearly saturated.

If Γ' were inconsistent, then for some n, Γ_n would be inconsistent (Exercise: explain why). So to show that Γ' is consistent it suffices to show, by induction on n, that each set Γ_n is consistent.

The induction basis is simply the claim that $\Gamma_0 = \Gamma$ is consistent, which is the hypothesis of the theorem. For the induction step, suppose that Γ_n is consistent but $\Gamma_{n+1} = \Gamma_n \cup \{\delta_n\}$ is inconsistent. Recall that δ_n is $\neg\forall x_n\, \alpha_n(x_n) \to \neg\alpha_n(c_n)$, where $\alpha_n(x_n)$ is a wff of $\mathcal{L}'$ with only the variable x_n free. By the way we've chosen the c_n (see Definition 113C), c_n does not occur in $\alpha_n(x_n)$ nor in Γ_n.

If $\Gamma_n \cup \{\delta_n\}$ is inconsistent, then $\Gamma_n \vdash \neg\delta_n$, and hence both of the following hold:

$$\Gamma_n \vdash \neg\forall x_n\, \alpha_n(x_n) \qquad \Gamma_n \vdash \alpha_n(c_n)$$

Since c_n does not occur in Γ_n or in $\alpha_n(x_n)$, Theorem 1010A applies. From $\Gamma_n \vdash \alpha_n(c_n)$, we obtain $\Gamma_n \vdash \forall x_n\, \alpha_n(x_n)$. Thus we have that both $\Gamma_n \vdash \neg\forall x_n\, \alpha_n(x_n)$ and $\Gamma_n \vdash \forall x_n\, \alpha_n(x_n)$, so Γ_n itself is inconsistent. Contradiction: Γ_n was supposed to be consistent. Hence $\Gamma_n \cup \{\delta_n\}$ is consistent. □

We'll now show that *complete*, consistent sets which are saturated have the property that it contains a universally quantified sentence iff it contains all its instancesit contains an existentially quantified sentence iff it contains at least one instance. We'll use this to show that the structure we'll generate from a complete, consistent, saturated set makes all its quantified sentences true.

Proposition 113E. *Suppose Γ is complete, consistent, and saturated. $\forall x\, \alpha(x) \in \Gamma$ iff $\alpha(t) \in \Gamma$ for all closed terms t.*

Proof. Suppose that $\alpha(t) \in \Gamma$ for all closed terms t. By way of contradiction, assume $\forall x\, \alpha(x) \notin \Gamma$. Since Γ is complete, $\neg\forall x\, \alpha(x) \in \Gamma$. By saturation, $(\neg\forall x\, \alpha(x) \to \neg\alpha(c)) \in \Gamma$ for some constant symbol c. By assumption, since c is a closed term, $\alpha(c) \in \Gamma$. But this would make Γ inconsistent. (Exercise: give the derivation that shows

$$\neg\forall x\, \alpha(x), \neg\forall x\, \alpha(x) \to \neg\alpha(c), \alpha(c)$$

is inconsistent.)

For the reverse direction, we do not need saturation: Suppose $\forall x\, \alpha(x) \in \Gamma$. Then $\Gamma \vdash \alpha(t)$ by Proposition 1010B, item (2). We get $\alpha(t) \in \Gamma$ by Proposition 112B. □

§11.4 Lindenbaum's Lemma

We now prove a lemma that shows that any consistent set of sentences is contained in some set of sentences which is not just consistent, but also complete. The proof works by adding one sentence at a time, guaranteeing at each step that the set remains consistent. We do this so that for every α, either α or $\neg\alpha$ gets added at some stage. The union of all stages in that construction then contains either α or its negation $\neg\alpha$ and is thus complete. It is also consistent, since we made sure at each stage not to introduce an inconsistency.

Lemma 114A (Lindenbaum's Lemma). *Every consistent set Γ in a language $\mathcal{L}$ can be extended to a complete and consistent set Γ^*.*

Proof. Let Γ be consistent. Let α_0, α_1, ... be an enumeration of all the sentences of $\mathcal{L}$. Define $\Gamma_0 = \Gamma$, and

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

Let $\Gamma^* = \bigcup_{n \geq 0} \Gamma_n$.

Each Γ_n is consistent: Γ_0 is consistent by definition. If $\Gamma_{n+1} = \Gamma_n \cup \{\alpha_n\}$, this is because the latter is consistent. If it isn't, $\Gamma_{n+1} = \Gamma_n \cup \{\neg\alpha_n\}$. We have to verify that $\Gamma_n \cup \{\neg\alpha_n\}$ is consistent. Suppose it's not. Then *both* $\Gamma_n \cup \{\alpha_n\}$ and $\Gamma_n \cup \{\neg\alpha_n\}$ are inconsistent. This means that Γ_n would be inconsistent by Proposition 108D, contrary to the induction hypothesis.

For every n and every $i < n$, $\Gamma_i \subseteq \Gamma_n$. This follows by a simple induction on n. For $n = 0$, there are no $i < 0$, so the claim holds automatically. For the inductive step, suppose it is true for n. We have $\Gamma_{n+1} = \Gamma_n \cup \{\alpha_n\}$ or $= \Gamma_n \cup \{\neg\alpha_n\}$ by construction. So $\Gamma_n \subseteq \Gamma_{n+1}$. If $i < n$, then $\Gamma_i \subseteq \Gamma_n$ by inductive hypothesis, and so $\subseteq \Gamma_{n+1}$ by transitivity of $\subseteq$.

From this it follows that every finite subset of Γ^* is a subset of Γ_n for some n, since each $\beta \in \Gamma^*$ not already in Γ_0 is added at some stage i. If n is the last one of these, then all β in the finite subset are in Γ_n. So, every finite subset of Γ^* is consistent. By Proposition 105H, Γ^* is consistent.

Every sentence of $\mathrm{Frm}(\mathcal{L})$ appears on the list used to define Γ^*. If $\alpha_n \notin \Gamma^*$, then that is because $\Gamma_n \cup \{\alpha_n\}$ was inconsistent. But then $\neg\alpha_n \in \Gamma^*$, so Γ^* is complete. □

§11.5 Construction of a Model

Right now we are not concerned about $=$, i.e., we only want to show that a consistent set Γ of sentences not containing $=$ is satisfiable. We first extend Γ

to a consistent, complete, and saturated set Γ^*. In this case, the definition of a model $\mathfrak{A}(\Gamma^*)$ is simple: We take the set of closed terms of $\mathcal{L}'$ as the domain. We assign every constant symbol to itself, and make sure that more generally, for every closed term t, $t^{\mathfrak{A}(\Gamma^*)} = t$. The predicate symbols are assigned extensions in such a way that an atomic sentence is true in $\mathfrak{A}(\Gamma^*)$ iff it is in Γ^*. This will obviously make all the atomic sentences in Γ^* true in $\mathfrak{A}(\Gamma^*)$. The rest are true provided the Γ^* we start with is consistent, complete, and saturated.

Definition 115A (Term model). *Let Γ^* be a complete and consistent, saturated set of sentences in a language $\mathcal{L}$. The* term model $\mathfrak{A}(\Gamma^*)$ *of Γ^* is the structure defined as follows:*

1. *The domain $|\mathfrak{A}(\Gamma^*)|$ is the set of all closed terms of $\mathcal{L}$.*
2. *The interpretation of a constant symbol c is c itself: $c^{\mathfrak{A}(\Gamma^*)} = c$.*
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 $f(t_1, \ldots, t_n)$:*

$$f^{\mathfrak{A}(\Gamma^*)}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

4. *If R is an n-place predicate symbol, then*

$$\langle t_1, \ldots, t_n \rangle \in R^{\mathfrak{A}(\Gamma^*)} \text{ iff } Rt_1 \ldots t_n \in \Gamma^*.$$

We will now check that we indeed have $t^{\mathfrak{A}(\Gamma^*)} = t$.

Lemma 115B. *Let $\mathfrak{A}(\Gamma^*)$ be the term model of Definition 115A, then $t^{\mathfrak{A}(\Gamma^*)} = t$.*

Proof. The proof is by induction on t, where the base case, when t is a constant symbol, follows directly from the definition of the term model. For the induction step assume $t_1, \ldots, t_n$ are closed terms such that ${t_i}^{\mathfrak{A}(\Gamma^*)} = t_i$ and that f is an n-ary function symbol. Then

$$\begin{aligned} f(t_1, \ldots, t_n)^{\mathfrak{A}(\Gamma^*)} &= f^{\mathfrak{A}(\Gamma^*)}({t_1}^{\mathfrak{A}(\Gamma^*)}, \ldots, {t_n}^{\mathfrak{A}(\Gamma^*)}) \\ &= f^{\mathfrak{A}(\Gamma^*)}(t_1, \ldots, t_n) \\ &= f(t_1, \ldots, t_n), \end{aligned}$$

and so by induction this holds for every closed term t. □

A structure $\mathfrak{A}$ may make all instances $\alpha(t)$ of a universally quantified sentence $\forall x\, \alpha(x)$ true, without making $\forall x\, \alpha(x)$ true. This is because in general not every element of $|\mathfrak{A}|$ is the value of a closed term ($\mathfrak{A}$ may not be covered). This is the reason the satisfaction relation is defined via variable assignments. However, for our term model $\mathfrak{A}(\Gamma^*)$ this wouldn't be necessary—because it is covered. This is the content of the next result.

Proposition 115C. *Let $\mathfrak{A}(\Gamma^*)$ be the term model of Definition 115A.* $\models_{\mathfrak{A}(\Gamma^*)} \forall x\, \alpha(x)$ *iff* $\models_{\mathfrak{A}(\Gamma^*)} \alpha(t)$ *for all terms t.*

Proof. By Proposition 74F, $\models_{\mathfrak{A}(\Gamma^*)} \forall x\, \alpha(x)$ iff for every variable assignment s, $\models_{\mathfrak{A}(\Gamma^*)} \alpha(x)\,[s]$. Recall that $|\mathfrak{A}(\Gamma^*)|$ consists of the closed terms of $\mathcal{L}$, so for every closed term t, $s(x) = t$ is such a variable assignment, and for any variable assignment, $s(x)$ is some closed term t. By Proposition 75D, $\models_{\mathfrak{A}(\Gamma^*)} \alpha(x)\,[s]$ iff $\models_{\mathfrak{A}(\Gamma^*)} \alpha(t)\,[s]$, where $s(x) = t$. By Proposition 74E, $\models_{\mathfrak{A}(\Gamma^*)} \alpha(t)\,[s]$ iff $\models_{\mathfrak{A}(\Gamma^*)} \alpha(t)$, since $\alpha(t)$ is a sentence. □

Lemma 115D (Truth Lemma). *Suppose α does not contain $=$. Then* $\models_{\mathfrak{A}(\Gamma^*)} \alpha$ *iff* $\alpha \in \Gamma^*$.

Proof. We prove both directions simultaneously, and by induction on α.

1. $\alpha \equiv R(t_1, \ldots, t_n)$: $\models_{\mathfrak{A}(\Gamma^*)} Rt_1 \ldots t_n$ iff $\langle t_1, \ldots, t_n\rangle \in R^{\mathfrak{A}(\Gamma^*)}$ (by the definition of satisfaction) iff $R(t_1, \ldots, t_n) \in \Gamma^*$ (by the construction of $\mathfrak{A}(\Gamma^*)$).

2. $\alpha \equiv \neg\beta$: $\models_{\mathfrak{A}(\Gamma^*)} \alpha$ iff $\not\models_{\mathfrak{A}(\Gamma^*)} \beta$ (by definition of satisfaction). By induction hypothesis, $\not\models_{\mathfrak{A}(\Gamma^*)} \beta$ iff $\beta \notin \Gamma^*$. Since Γ^* is consistent and complete, $\beta \notin \Gamma^*$ iff $\neg\beta \in \Gamma^*$.

3. $\alpha \equiv \beta \to \gamma$: $\models_{\mathfrak{A}(\Gamma^*)} \alpha$ iff $\not\models_{\mathfrak{A}(\Gamma^*)} \beta$ or $\models_{\mathfrak{A}(\Gamma^*)} \gamma$ (by definition of satisfaction) iff $\beta \notin \Gamma^*$ or $\gamma \in \Gamma^*$ (by induction hypothesis). This is the case iff $(\beta \to \gamma) \in \Gamma^*$ (by Proposition 112B(2)).

4. $\alpha \equiv \forall x\, \beta(x)$: $\models_{\mathfrak{A}(\Gamma^*)} \alpha$ iff $\models_{\mathfrak{A}(\Gamma^*)} \beta(t)$ for all terms t (Proposition 115C). By induction hypothesis, this is the case iff $\beta(t) \in \Gamma^*$ for all terms t, by Proposition 113E, this in turn is the case iff $\forall x\, \alpha(x) \in \Gamma^*$.

§11.6 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 $\alpha \in \Gamma^*$ which does not contain $=$ (and hence all $\alpha \in \Gamma$). 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 116A. *Let Γ^* be a consistent and complete set of sentences in $\mathcal{L}$. We define the relation $\approx$ on the set of closed terms of $\mathcal{L}$ by*

$$t \approx t' \quad \textit{iff} \quad t = t' \in \Gamma^*$$

Proposition 116B. *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*
$$ft_1 \ldots t_{i-1} t t_{i+1} \ldots t_n \approx ft_1 \ldots t_{i-1} t' t_{i+1} \ldots t_n.$$
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*
$$Rt_1 \ldots t_{i-1} t t_{i+1} \ldots t_n \in \Gamma^* \text{ iff } Rt_1 \ldots t_{i-1} t' t_{i+1} \ldots t_n \in \Gamma^*.$$

Proof. Since Γ^* is consistent and complete, $t = t' \in \Gamma^*$ iff $\Gamma^* \vdash t = t'$. Thus it is enough to show the following:

1. $\Gamma^* \vdash t = t$ for all terms t.
2. If $\Gamma^* \vdash t = t'$ then $\Gamma^* \vdash t' = t$.
3. If $\Gamma^* \vdash t = t'$ and $\Gamma^* \vdash t' = t''$, then $\Gamma^* \vdash t = t''$.
4. If $\Gamma^* \vdash t = t'$, then
$$\Gamma^* \vdash ft_1 \ldots t_{i-1} t t_{i+1} \ldots t_n = ft_1 \ldots t_{i-1} t' t_{i+1} \ldots t_n$$
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 $\Gamma^* \vdash Rt_1 \ldots t_{i-1} t t_{i+1} \ldots t_n$, then $\Gamma^* \vdash Rt_1 \ldots t_{i-1} t' t_{i+1} \ldots t_n$ for every n-place predicate symbol R and terms $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$.

□

Definition 116C. *Suppose Γ^* is a consistent and complete set in a language $\mathcal{L}$, t is a term, and $\approx$ as in the previous definition. Then:*

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

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

Definition 116D. *Let $\mathfrak{A} = \mathfrak{A}(\Gamma^*)$ be the term model for Γ^* from Definition 115A. Then $\mathfrak{A}/_\approx$ is the following structure:*

1. $|\mathfrak{A}/_\approx| = \mathrm{Trm}(\mathcal{L})/_\approx$.
2. $c^{\mathfrak{A}/_\approx} = [c]_\approx$

3. $f^{\mathfrak{A}/\approx}([t_1]_\approx, \ldots, [t_n]_\approx) = [ft_1 \ldots t_n]_\approx$

4. $\langle [t_1]_\approx, \ldots, [t_n]_\approx \rangle \in R^{\mathfrak{A}/\approx}$ *iff* $\models_{\mathfrak{A}} Rt_1 \ldots t_n$, *i.e., iff* $Rt_1 \ldots t_n \in \Gamma^*$.

Note that we have defined $f^{\mathfrak{A}/\approx}$ and $R^{\mathfrak{A}/\approx}$ 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 $[t]_\approx \in R^{\mathfrak{A}/\approx}$ iff $\models_{\mathfrak{A}} Rt$. If for some other term t' with $t \approx t'$, $\not\models_{\mathfrak{A}} Rt$, then the definition would require $[t']_\approx \notin R^{\mathfrak{A}/\approx}$. If $t \approx t'$, then $[t]_\approx = [t']_\approx$, but we can't have both $[t]_\approx \in R^{\mathfrak{A}/\approx}$ and $[t]_\approx \notin R^{\mathfrak{A}/\approx}$. However, Proposition 116B guarantees that this cannot happen.

Proposition 116E. $\mathfrak{A}/_\approx$ *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. $[ft_1 \ldots t_n]_\approx = [ft'_1 \ldots t'_n]_\approx$, *i.e.,*

$$ft_1 \ldots t_n \approx ft'_1 \ldots t'_n$$

and

2. $\models_{\mathfrak{A}} Rt_1 \ldots t_n$ *iff* $\models_{\mathfrak{A}} Rt'_1 \ldots t'_n$, *i.e.,*

$$Rt_1 \ldots t_n \in \Gamma^* \text{ iff } Rt'_1 \ldots t'_n \in \Gamma^*.$$

Proof. Follows from Proposition 116B by induction on n. □

As in the case of the term model, before proving the truth lemma we need the following lemma.

Lemma 116F. *Let* $\mathfrak{A} = \mathfrak{A}(\Gamma^*)$, *then* $t^{\mathfrak{A}/\approx} = [t]_\approx$.

Proof. The proof is similar to that of Lemma 115B. □

Lemma 116G. $\models_{\mathfrak{A}/\approx} \alpha$ *iff* $\alpha \in \Gamma^*$ *for all sentences* α.

Proof. By induction on α, just as in the proof of Lemma 115D. The only case that needs additional attention is when $\alpha \equiv t = t'$.

$$\begin{aligned} \models_{\mathfrak{A}/\approx} t = t' &\text{ iff } [t]_\approx = [t']_\approx \text{ (by definition of } \mathfrak{A}/\approx) \\ &\text{iff } t \approx t' \text{ (by definition of } [t]_\approx) \\ &\text{iff } t = t' \in \Gamma^* \text{ (by definition of } \approx). \end{aligned}$$

□

Note that while $\mathfrak{A}(\Gamma^*)$ is always enumerable and infinite, $\mathfrak{A}/_\approx$ 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.

§11.7 The Completeness Theorem

Let's combine our results: we arrive at the completeness theorem.

Theorem 117A (Completeness Theorem). *Let Γ be a set of sentences. If Γ is consistent, it is satisfiable.*

Proof. Suppose Γ is consistent. By Lemma 113D, there is a saturated consistent set $\Gamma' \supseteq \Gamma$. By Lemma 114A, there is a $\Gamma^* \supseteq \Gamma'$ which is consistent and complete. Since $\Gamma' \subseteq \Gamma^*$, for each wff $\alpha(x)$, Γ^* contains a sentence of the form $\neg\forall x\, \alpha(x) \to \neg\alpha(c)$ and so Γ^* is saturated. If Γ does not contain $=$, then by Lemma 115D, $\models_{\mathfrak{A}(\Gamma^*)} \alpha$ iff $\alpha \in \Gamma^*$. From this it follows in particular that for all $\alpha \in \Gamma$, $\models_{\mathfrak{A}(\Gamma^*)} \alpha$, so Γ is satisfiable. If Γ does contain $=$, then by Lemma 116G, for all sentences α, $\models_{\mathfrak{A}/\approx} \alpha$ iff $\alpha \in \Gamma^*$. In particular, $\models_{\mathfrak{A}/\approx} \alpha$ for all $\alpha \in \Gamma$, so Γ is satisfiable. □

Corollary 117B (Completeness Theorem, Second Version). *For all Γ and sentences α: if $\Gamma \vDash \alpha$ then $\Gamma \vdash \alpha$.*

Proof. Note that the Γ's in corollary 117B and Theorem 117A are universally quantified. To make sure we do not confuse ourselves, let us restate Theorem 117A using a different variable: for any set of sentences Δ, if Δ is consistent, it is satisfiable. By contraposition, if Δ is not satisfiable, then Δ is inconsistent. We will use this to prove the corollary.

Suppose that $\Gamma \vDash \alpha$. Then $\Gamma \cup \{\neg\alpha\}$ is unsatisfiable by Proposition 76E. Taking $\Gamma \cup \{\neg\alpha\}$ as our Δ, the previous version of Theorem 117A gives us that $\Gamma \cup \{\neg\alpha\}$ is inconsistent. By Proposition 108B, $\Gamma \vdash \alpha$. □

§11.8 The Compactness Theorem

One important consequence of the completeness theorem is the compactness theorem. The compactness theorem states that if each *finite* subset of a set of sentences is satisfiable, the entire set is satisfiable—even if the set itself is infinite. This is far from obvious. There is nothing that seems to rule out, at first glance at least, the possibility of there being infinite sets of sentences which are contradictory, but the contradiction only arises, so to speak, from the infinite number. The compactness theorem says that such a scenario can be ruled out: there are no unsatisfiable infinite sets of sentences each finite subset of which is satisfiable. Like the completeness theorem, it has a version related to entailment: if an infinite set of sentences entails something, already a finite subset does.

Definition 118A. *A set Γ of wffs is* finitely satisfiable *iff every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable.*

Theorem 118B (Compactness Theorem). *The following hold for any sentences Γ and α:*

1. *$\Gamma \vDash \alpha$ iff there is a finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \vDash \alpha$.*

2. *Γ is satisfiable iff it is finitely satisfiable.*

Proof. We prove (2). If Γ is satisfiable, then there is a structure $\mathfrak{A}$ such that $\models_{\mathfrak{A}} \alpha$ for all $\alpha \in \Gamma$. Of course, this $\mathfrak{A}$ also satisfies every finite subset of Γ, so Γ is finitely satisfiable.

Now suppose that Γ is finitely satisfiable. Then every finite subset $\Gamma_0 \subseteq \Gamma$ is satisfiable. By soundness (corollary 1011D), every finite subset is consistent. Then Γ itself must be consistent by Proposition 105H. By completeness (Theorem 117A), since Γ is consistent, it is satisfiable. □

Example 11.8.3. In every model $\mathfrak{A}$ of a theory Γ, each term t of course picks out an element of $|\mathfrak{A}|$. Can we guarantee that it is also true that every element of $|\mathfrak{A}|$ 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 $\mathfrak{A}$ 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 $a \in |\mathfrak{A}|$ be an element of $|\mathfrak{A}|$ not picked out by any of them, and let $\mathfrak{A}'$ be the structure that is just like $\mathfrak{A}$, but also $c^{\mathfrak{A}'} = a$. Since $a \neq t^{\mathfrak{A}}$ for all t occurring in Δ', $\models_{\mathfrak{A}'} \Delta'$. Since $\models_{\mathfrak{A}} \Gamma$, $\Gamma' \subseteq \Gamma$, and c does not occur in Γ, also $\models_{\mathfrak{A}'} \Gamma'$. Together, $\models_{\mathfrak{A}'} \Gamma' \cup \Delta'$ 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 $\models_{\mathfrak{A}} \Gamma \cup \Delta$. Every such $\mathfrak{A}$ is a model of Γ, but is not covered, since $c^{\mathfrak{A}} \neq t^{\mathfrak{A}}$ for all terms t of $\mathcal{L}$.

Example 11.8.4. Consider a language $\mathcal{L}$ containing the predicate symbol $<$, constant symbols 0, 1, and function symbols $+$, $\times$, $-$, $\div$. Let Γ be the set of all sentences in this language true in $\mathfrak{Q}$ with domain $\mathbb{Q}$ and the obvious interpretations. Γ is the set of all sentences of $\mathcal{L}$ true about the rational numbers. Of course, in $\mathbb{Q}$ (and even in $\mathbb{R}$), there are no numbers which are greater than 0 but less than $1/k$ for all $k \in \mathbb{Z}^+$. Such a number, if it existed, would be an *infinitesimal:* non-zero, but infinitely small. The compactness theorem shows that there are models of Γ in which infinitesimals exist: Let Δ be $\{0 < c\} \cup \{c < (1 \div \overline{k}) : k \in \mathbb{Z}^+\}$ (where $\overline{k} = (1 + (1 + \cdots + (1 + 1)\ldots))$ with k 1's). For any finite subset Δ_0 of Δ there is a K such that all the sentences $c < (1 \div \overline{k})$ in Δ_0 have $k < K$. If we expand $\mathfrak{Q}$ to $\mathfrak{Q}'$ with $c^{\mathfrak{Q}'} = 1/K$ we have that $\models_{\mathfrak{Q}'} \Gamma \cup \Delta_0$, and so $\Gamma \cup \Delta$ is finitely satisfiable (Exercise: prove this in detail). By compactness, $\Gamma \cup \Delta$ is satisfiable. Any model $\mathfrak{C}$ of $\Gamma \cup \Delta$ contains an infinitesimal, namely $c^{\mathfrak{C}}$.

Example 11.8.5. We know that first-order logic with equality symbol can express that the size of the domain must have some minimal size: The sentence $\alpha_{\geq n}$ (which says "there are at least n distinct objects") is true only in structures where $|\mathfrak{A}|$ has at least n objects. So if we take

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

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 $\alpha_{\geq n} \in \Delta'$. Λ, being satisfied in all finite structures, has a model $\mathfrak{A}$ with finitely many but $\geq n$ elements. But then $\models_{\mathfrak{A}} \Delta' \cup \Lambda'$. By compactness, $\Delta \cup \Lambda$ has an infinite model, contradicting the assumption that Λ is satisfied only in finite structures.

§11.9 A Direct Proof of the Compactness Theorem

We can prove the Compactness Theorem directly, without appealing to the Completeness Theorem, using the same ideas as in the proof of the completeness theorem. In the proof of the Completeness Theorem we started with a consistent set Γ of sentences, expanded it to a consistent, saturated, and complete set Γ^* of sentences, and then showed that in the term model $\mathfrak{A}(\Gamma^*)$ constructed from Γ^*, all sentences of Γ are true, so Γ is satisfiable.

We can use the same method to show that a finitely satisfiable set of sentences is satisfiable. We just have to prove the corresponding versions of the results leading to the truth lemma where we replace "consistent" with "finitely satisfiable."

Proposition 119A. *Suppose Γ is complete and finitely satisfiable. Then:*

1. *$(\alpha \to \beta) \in \Gamma$ iff either $\alpha \notin \Gamma$ or $\beta \in \Gamma$.*

Lemma 119B. *Every finitely satisfiable set Γ can be extended to a saturated finitely satisfiable set Γ'.*

Proposition 119C. *Suppose Γ is complete, finitely satisfiable, and saturated. $\forall x\, \alpha(x) \in \Gamma$ iff $\alpha(t) \in \Gamma$ for all closed terms t.*

Lemma 119D. *Every finitely satisfiable set Γ can be extended to a complete and finitely satisfiable set Γ^*.*

Theorem 119E (Compactness). *Γ is satisfiable if and only if it is finitely satisfiable.*

Proof. If Γ is satisfiable, then there is a structure $\mathfrak{A}$ such that $\models_{\mathfrak{A}} \alpha$ for all $\alpha \in \Gamma$. Of course, this $\mathfrak{A}$ also satisfies every finite subset of Γ, so Γ is finitely satisfiable.

Now suppose that Γ is finitely satisfiable. By Lemma 119B, there is a finitely satisfiable, saturated set $\Gamma' \supseteq \Gamma$. By Lemma 119D, Γ' can be extended to a complete and finitely satisfiable set Γ^*, and Γ^* is still saturated. Construct the term model $\mathfrak{A}(\Gamma^*)$ as in Definition 115A. Note that Proposition 115C did not rely on the fact that Γ^* is consistent (or complete or saturated, for that matter), but just on the fact that $\mathfrak{A}(\Gamma^*)$ is covered. The proof of the Truth Lemma (Lemma 115D) goes through if we replace references to Proposition 112B and Proposition 113E by references to Proposition 119A and Proposition 119C □

§11.10 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 consequence 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 enumerable structure.

Theorem 1110A. *If Γ is consistent then it has an enumerable model, i.e., it is satisfiable in a structure whose domain is either finite or denumerable.*

Proof. If Γ is consistent, the structure $\mathfrak{A}$ delivered by the proof of the completeness theorem has a domain $|\mathfrak{A}|$ that is no larger than the set of the terms of the language $\mathcal{L}$. So $\mathfrak{A}$ is at most denumerable. □

Theorem 1110B. *If Γ is a 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 $\mathfrak{A}$ delivered by the proof of the completeness theorem has a domain $|\mathfrak{A}|$ identical to the set of terms of the language $\mathcal{L}'$. So $\mathfrak{A}$ is denumerable, since $\mathrm{Trm}(\mathcal{L}')$ is. □

Example 11.10.3 (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.

Problems

Problem 1. Complete the proof of Proposition 112B.

Problem 2. Complete the proof of Proposition 116B.

Problem 3. Complete the proof of Lemma 116F.

Problem 4. Use corollary 117B to prove Theorem 117A, thus showing that the two formulations of the completeness theorem are equivalent.

Problem 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 derivation 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 derivation were used in which results that lead up to the proof of Theorem 117A. Be sure to note any tacit uses of rules in these proofs.

Problem 6. Prove (1) of Theorem 118B.

Problem 7. In the standard model of arithmetic $\mathfrak{B}$, there is no element $k \in |\mathfrak{B}|$ 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 $\mathfrak{B}$ are also true in a structure $\mathfrak{B}'$ that contains an element which *does* satisfy every formula $\overline{n} < x$.

Problem 8. Prove Proposition 119A. Avoid the use of $\vdash$.

Problem 9. Prove Lemma 119B. (Hint: The crucial step is to show that if Γ_n is finitely satisfiable, so is $\Gamma_n \cup \{\delta_n\}$, without any appeal to derivations or consistency.)

Problem 10. Prove Proposition 119C.

Problem 11. Prove Lemma 119D. (Hint: the crucial step is to show that if Γ_n is finitely satisfiable, then either $\Gamma_n \cup \{\alpha_n\}$ or $\Gamma_n \cup \{\neg\alpha_n\}$ is finitely satisfiable.)

Problem 12. Write out the complete proof of the Truth Lemma (Lemma 115D) in the version required for the proof of Theorem 119E.

Chapter 12

Beyond First-order Logic

§12.0 Overview

First-order logic is not the only system of logic of interest: there are many extensions and variations of first-order logic. A logic typically consists of the formal specification of a language, usually, but not always, a deductive system, and usually, but not always, an intended semantics. But the technical use of the term raises an obvious question: what do logics that are not first-order logic have to do with the word "logic," used in the intuitive or philosophical sense? All of the systems described below are designed to model reasoning of some form or another; can we say what makes them logical?

No easy answers are forthcoming. The word "logic" is used in different ways and in different contexts, and the notion, like that of "truth," has been analyzed from numerous philosophical stances. For example, one might take the goal of logical reasoning to be the determination of which statements are necessarily true, true a priori, true independent of the interpretation of the nonlogical terms, true by virtue of their form, or true by linguistic convention; and each of these conceptions requires a good deal of clarification. Even if one restricts one's attention to the kind of logic used in mathematics, there is little agreement as to its scope. For example, in the *Principia Mathematica*, Russell and Whitehead tried to develop mathematics on the basis of logic, in the *logicist* tradition begun by Frege. Their system of logic was a form of higher-type logic similar to the one described below. In the end they were forced to introduce axioms which, by most standards, do not seem purely logical (notably, the axiom of infinity, and the axiom of reducibility), but one might nonetheless hold that some forms of higher-order reasoning should be accepted as logical. In contrast, Quine, whose ontology does not admit "propositions" as legitimate objects of discourse, argues that second-order and higher-order logic are really manifestations of set theory in sheep's clothing; in other words, systems involving quantification over predicates are not purely logical.

For now, it is best to leave such philosophical issues for a rainy day, and simply think of the systems below as formal idealizations of various kinds of

reasoning, logical or otherwise.

§12.1 Many-Sorted Logic

In first-order logic, variables and quantifiers range over a single domain. But it is often useful to have multiple (disjoint) domains: for example, you might want to have a domain of numbers, a domain of geometric objects, a domain of functions from numbers to numbers, a domain of abelian groups, and so on.

Many-sorted logic provides this kind of framework. One starts with a list of "sorts"—the "sort" of an object indicates the "domain" it is supposed to inhabit. One then has variables and quantifiers for each sort, and (usually) an equality symbol for each sort. Functions and relations are also "typed" by the sorts of objects they can take as arguments. Otherwise, one keeps the usual rules of first-order logic, with versions of the quantifier-rules repeated for each sort.

For example, to study international relations we might choose a language with two sorts of objects, French citizens and German citizens. We might have a unary relation, "drinks wine," for objects of the first sort; another unary relation, "eats wurst," for objects of the second sort; and a binary relation, "forms a multinational married couple," which takes two arguments, where the first argument is of the first sort and the second argument is of the second sort. If we use variables a, b, c to range over French citizens and x, y, z to range over German citizens, then

$$\forall a\, \forall x[(MarriedToax \to (DrinksWinea \vee \neg EatsWurstx)]]$$

asserts that if any French person is married to a German, either the French person drinks wine or the German doesn't eat wurst.

Many-sorted logic can be embedded in first-order logic in a natural way, by lumping all the objects of the many-sorted domains together into one first-order domain, using unary predicate symbols to keep track of the sorts, and relativizing quantifiers. For example, the first-order language corresponding to the example above would have unary predicate symbols "$German$" and "$French$," in addition to the other relations described, with the sort requirements erased. A sorted quantifier $\forall x\, \alpha$, where x is a variable of the German sort, translates to

$$\forall x\, (Germanx \to \alpha).$$

We need to add axioms that insure that the sorts are separate—e.g., $\forall x\, \neg(Germanx \wedge Frenchx)$—as well as axioms that guarantee that "drinks wine" only holds of objects satisfying the predicate $Frenchx$, etc. With these conventions and axioms, it is not difficult to show that many-sorted sentences translate to first-order sentences, and many-sorted derivations translate to first-order derivations. Also, many-sorted structures "translate" to corresponding first-order structures and vice-versa, so we also have a completeness theorem for many-sorted logic.

§12.2 Second-Order logic

The language of second-order logic allows one to quantify not just over a domain of individuals, but over relations on that domain as well. Given a first-order language $\mathcal{L}$, for each k one adds variables R which range over k-ary relations, and allows quantification over those variables. If R is a variable for a k-ary relation, and $t_1, \ldots, t_k$ are ordinary (first-order) terms, $Rt_1 \ldots t_k$ is an atomic wff. Otherwise, the set of wffs is defined just as in the case of first-order logic, with additional clauses for second-order quantification. Note that we only have the equality symbol for first-order terms: if R and S are relation variables of the same arity k, we can define $R = S$ to be an abbreviation for

$$\forall x_1 \ldots \forall x_k \, (Rx_1 \ldots x_k \leftrightarrow Sx_1 \ldots x_k).$$

The rules for second-order logic simply extend the quantifier rules to the new second order variables. Here, however, one has to be a little bit careful to explain how these variables interact with the predicate symbols of $\mathcal{L}$, and with wffs of $\mathcal{L}$ more generally. At the bare minimum, relation variables count as terms, so one has inferences of the form

$$\alpha(R) \vdash \exists R \, \alpha(R)$$

But if $\mathcal{L}$ is the language of arithmetic with a constant relation symbol $<$, one would also expect the following inference to be valid:

$$x < y \vdash \exists R \, Rxy$$

or for a given wff α,

$$\alpha(x_1, \ldots, x_k) \vdash \exists R \, Rx_1 \ldots x_k$$

More generally, we might want to allow inferences of the form

$$\alpha[\lambda \vec{x}. \, \beta(\vec{x})/R] \vdash \exists R \, \alpha$$

where $\alpha[\lambda \vec{x}. \, \beta(\vec{x})/R]$ denotes the result of replacing every atomic wff of the form $Rt_1, \ldots, t_k$ in α by $\beta(t_1, \ldots, t_k)$. This last rule is equivalent to having a *comprehension schema*, i.e., an axiom of the form

$$\exists R \, \forall x_1, \ldots, x_k \, (\alpha(x_1, \ldots, x_k) \leftrightarrow Rx_1 \ldots x_k),$$

one for each wff α in the second-order language, in which R is not a free variable. (Exercise: show that if R is allowed to occur in α, this schema is inconsistent!)

When logicians refer to the "axioms of second-order logic" they usually mean the minimal extension of first-order logic by second-order quantifier rules together with the comprehension schema. But it is often interesting to study weaker subsystems of these axioms and rules. For example, note that in its full generality the axiom schema of comprehension is *impredicative*: it allows

one to assert the existence of a relation $Rx_1 \dots x_k$ that is "defined" by a wff with second-order quantifiers; and these quantifiers range over the set of all such relations—a set which includes R itself! Around the turn of the twentieth century, a common reaction to Russell's paradox was to lay the blame on such definitions, and to avoid them in developing the foundations of mathematics. If one prohibits the use of second-order quantifiers in the wff α, one has a *predicative* form of comprehension, which is somewhat weaker.

From the semantic point of view, one can think of a second-order structure as consisting of a first-order structure for the language, coupled with a set of relations on the domain over which the second-order quantifiers range (more precisely, for each k there is a set of relations of arity k). Of course, if comprehension is included in the derivation system, then we have the added requirement that there are enough relations in the "second-order part" to satisfy the comprehension axioms—otherwise the derivation system is not sound! One easy way to insure that there are enough relations around is to take the second-order part to consist of *all* the relations on the first-order part. Such a structure is called *full*, and, in a sense, is really the "intended structure" for the language. If we restrict our attention to full structures we have what is known as the *full* second-order semantics. In that case, specifying a structure boils down to specifying the first-order part, since the contents of the second-order part follow from that implicitly.

To summarize, there is some ambiguity when talking about second-order logic. In terms of the derivation system, one might have in mind either

1. A "minimal" second-order derivation system, together with some comprehension axioms.

2. The "standard" second-order derivation system, with full comprehension.

In terms of the semantics, one might be interested in either

1. The "weak" semantics, where a structure consists of a first-order part, together with a second-order part big enough to satisfy the comprehension axioms.

2. The "standard" second-order semantics, in which one considers full structures only.

When logicians do not specify the derivation system or the semantics they have in mind, they are usually referring to the second item on each list. The advantage to using this semantics is that, as we will see, it gives us categorical descriptions of many natural mathematical structures; at the same time, the derivation system is quite strong, and sound for this semantics. The drawback is that the derivation system is *not* complete for the semantics; in fact, *no* effectively given derivation system is complete for the full second-order semantics. On the other hand, we will see that the derivation system *is* complete for the weakened semantics; this implies that if a sentence is not provable, then there is *some* structure, not necessarily the full one, in which it is false.

The language of second-order logic is quite rich. One can identify unary relations with subsets of the domain, and so in particular you can quantify over these sets; for example, one can express induction for the natural numbers with a single axiom

$$\forall R\,((R0 \wedge \forall x\,(Rx \to Rx')) \to \forall x\,Rx).$$

If one takes the language of arithmetic to have symbols $0, \prime, +, \times$ and $<$, one can add the following axioms to describe their behavior:

1. $\forall x\,\neg x' = 0$
2. $\forall x\,\forall y\,(s(x) = s(y) \to x = y)$
3. $\forall x\,(x + 0) = x$
4. $\forall x\,\forall y\,(x + y') = (x + y)'$
5. $\forall x\,(x \times 0) = 0$
6. $\forall x\,\forall y\,(x \times y') = ((x \times y) + x)$
7. $\forall x\,\forall y\,(x < y \leftrightarrow \exists z\,y = (x + z'))$

It is not difficult to show that these axioms, together with the axiom of induction above, provide a categorical description of the structure $\mathfrak{B}$, the standard model of arithmetic, provided we are using the full second-order semantics. Given any structure $\mathfrak{A}$ in which these axioms are true, define a function f from $\mathbb{N}$ to the domain of $\mathfrak{A}$ using ordinary recursion on $\mathbb{N}$, so that $f(0) = 0^{\mathfrak{A}}$ and $f(x + 1) = \prime^{\mathfrak{A}}(f(x))$. Using ordinary induction on $\mathbb{N}$ and the fact that axioms (1) and (2) hold in $\mathfrak{A}$, we see that f is injective. To see that f is surjective, let P be the set of elements of $|\mathfrak{A}|$ that are in the range of f. Since $\mathfrak{A}$ is full, P is in the second-order domain. By the construction of f, we know that $0^{\mathfrak{A}}$ is in P, and that P is closed under $\prime^{\mathfrak{A}}$. The fact that the induction axiom holds in $\mathfrak{A}$ (in particular, for P) guarantees that P is equal to the entire first-order domain of $\mathfrak{A}$. This shows that f is a bijection. Showing that f is a homomorphism is no more difficult, using ordinary induction on $\mathbb{N}$ repeatedly.

In set-theoretic terms, a function is just a special kind of relation; for example, a unary function f can be identified with a binary relation R satisfying $\forall x\,\exists! y\,R(x, y)$. As a result, one can quantify over functions too. Using the full semantics, one can then define the class of infinite structures to be the class of structures $\mathfrak{A}$ for which there is an injective function from the domain of $\mathfrak{A}$ to a proper subset of itself:

$$\exists f\,(\forall x\,\forall y\,(f(x) = f(y) \to x = y) \wedge \exists y\,\forall x\,f(x) \neq y).$$

The negation of this sentence then defines the class of finite structures.

In addition, one can define the class of well-orderings, by adding the following to the definition of a linear ordering:

$$\forall P\,(\exists x\,Px \to \exists x\,(Px \wedge \forall y\,(y < x \to \neg Py))).$$

This asserts that every non-empty set has a least element, modulo the identification of "set" with "one-place relation". For another example, one can express the notion of connectedness for graphs, by saying that there is no nontrivial separation of the vertices into disconnected parts:

$$\neg\exists A\,(\exists x\,A(x) \wedge \exists y\,\neg A(y) \wedge \forall w\,\forall z\,((Aw \wedge \neg Az) \rightarrow \neg Rwz)).$$

For yet another example, you might try as an exercise to define the class of finite structures whose domain has even size. More strikingly, one can provide a categorical description of the real numbers as a complete ordered field containing the rationals.

In short, second-order logic is much more expressive than first-order logic. That's the good news; now for the bad. We have already mentioned that there is no effective derivation system that is complete for the full second-order semantics. For better or for worse, many of the properties of first-order logic are absent, including compactness and the Löwenheim-Skolem theorems.

On the other hand, if one is willing to give up the full second-order semantics in terms of the weaker one, then the minimal second-order derivation system is complete for this semantics. In other words, if we read $\vdash$ as "proves in the minimal system" and $\models$ as "logically implies in the weaker semantics", we can show that whenever $\Gamma \models \alpha$ then $\Gamma \vdash \alpha$. If one wants to include specific comprehension axioms in the derivation system, one has to restrict the semantics to second-order structures that satisfy these axioms: for example, if Δ consists of a set of comprehension axioms (possibly all of them), we have that if $\Gamma \cup \Delta \models \alpha$, then $\Gamma \cup \Delta \vdash \alpha$. In particular, if α is not provable using the comprehension axioms we are considering, then there is a model of $\neg\alpha$ in which these comprehension axioms nonetheless hold.

The easiest way to see that the completeness theorem holds for the weaker semantics is to think of second-order logic as a many-sorted logic, as follows. One sort is interpreted as the ordinary "first-order" domain, and then for each k we have a domain of "relations of arity k." We take the language to have built-in relation symbols "$true_k R x_1 \ldots x_k$" which is meant to assert that R holds of $x_1, \ldots, x_k$, where R is a variable of the sort "k-ary relation" and $x_1, \ldots, x_k$ are objects of the first-order sort.

With this identification, the weak second-order semantics is essentially the usual semantics for many-sorted logic; and we have already observed that many-sorted logic can be embedded in first-order logic. Modulo the translations back and forth, then, the weaker conception of second-order logic is really a form of first-order logic in disguise, where the domain contains both "objects" and "relations" governed by the appropriate axioms.

§12.3 Higher-Order logic

Passing from first-order logic to second-order logic enabled us to talk about sets of objects in the first-order domain, within the formal language. Why stop there? For example, third-order logic should enable us to deal with sets of sets

of objects, or perhaps even sets which contain both objects and sets of objects. And fourth-order logic will let us talk about sets of objects of that kind. As you may have guessed, one can iterate this idea arbitrarily.

In practice, higher-order logic is often wffted in terms of functions instead of relations. (Modulo the natural identifications, this difference is inessential.) Given some basic "sorts" A, B, C, ... (which we will now call "types"), we can create new ones by stipulating

> If σ and τ are finite types then so is $\sigma \to \tau$.

Think of types as syntactic "labels," which classify the objects we want in our domain; $\sigma \to \tau$ describes those objects that are functions which take objects of type σ to objects of type τ. For example, we might want to have a type Ω of truth values, "true" and "false," and a type $\mathbb{N}$ of natural numbers. In that case, you can think of objects of type $\mathbb{N} \to \Omega$ as unary relations, or subsets of $\mathbb{N}$; objects of type $\mathbb{N} \to \mathbb{N}$ are functions from natural numbers to natural numbers; and objects of type $(\mathbb{N} \to \mathbb{N}) \to \mathbb{N}$ are "functionals," that is, higher-type functions that take functions to numbers.

As in the case of second-order logic, one can think of higher-order logic as a kind of many-sorted logic, where there is a sort for each type of object we want to consider. But it is usually clearer just to define the syntax of higher-type logic from the ground up. For example, we can define a set of finite types inductively, as follows:

1. $\mathbb{N}$ is a finite type.
2. If σ and τ are finite types, then so is $\sigma \to \tau$.
3. If σ and τ are finite types, so is $\sigma \times \tau$.

Intuitively, $\mathbb{N}$ denotes the type of the natural numbers, $\sigma \to \tau$ denotes the type of functions from σ to τ, and $\sigma \times \tau$ denotes the type of pairs of objects, one from σ and one from τ. We can then define a set of terms inductively, as follows:

1. For each type σ, there is a stock of variables x, y, z, ... of type σ
2. 0 is a term of type $\mathbb{N}$
3. S (successor) is a term of type $\mathbb{N} \to \mathbb{N}$
4. If s is a term of type σ, and t is a term of type $\mathbb{N} \to (\sigma \to \sigma)$, then R_{st} is a term of type $\mathbb{N} \to \sigma$
5. If s is a term of type $\tau \to \sigma$ and t is a term of type τ, then $s(t)$ is a term of type σ
6. If s is a term of type σ and x is a variable of type τ, then $\lambda x.\, s$ is a term of type $\tau \to \sigma$.

7. If s is a term of type σ and t is a term of type τ, then $\langle s, t\rangle$ is a term of type $\sigma \times \tau$.

8. If s is a term of type $\sigma \times \tau$ then $p_1(s)$ is a term of type σ and $p_2(s)$ is a term of type τ.

Intuitively, R_{st} denotes the function defined recursively by

$$R_{st}(0) = s$$
$$R_{st}(x+1) = t(x, R_{st}(x)),$$

$\langle s, t\rangle$ denotes the pair whose first component is s and whose second component is t, and $p_1(s)$ and $p_2(s)$ denote the first and second elements ("projections") of s. Finally, $\lambda x.\, s$ denotes the function f defined by

$$f(x) = s$$

for any x of type σ; so item (6) gives us a form of comprehension, enabling us to define functions using terms. Wffs are built up from equality symbol statements $s = t$ between terms of the same type, the usual propositional connectives, and higher-type quantification. One can then take the axioms of the system to be the basic equations governing the terms defined above, together with the usual rules of logic with quantifiers and equality symbol.

If one augments the finite type system with a type Ω of truth values, one has to include axioms which govern its use as well. In fact, if one is clever, one can get rid of complex wffs entirely, replacing them with terms of type Ω! The proof system can then be modified accordingly. The result is essentially the *simple theory of types* set forth by Alonzo Church in the 1930s.

As in the case of second-order logic, there are different versions of higher-type semantics that one might want to use. In the full version, variables of type $\sigma \to \tau$ range over the set of *all* functions from the objects of type σ to objects of type τ. As you might expect, this semantics is too strong to admit a complete, effective derivation system. But one can consider a weaker semantics, in which a structure consists of sets of elements T_τ for each type τ, together with appropriate operations for application, projection, etc. If the details are carried out correctly, one can obtain completeness theorems for the kinds of derivation systems described above.

Higher-type logic is attractive because it provides a framework in which we can embed a good deal of mathematics in a natural way: starting with $\mathbb{N}$, one can define real numbers, continuous functions, and so on. It is also particularly attractive in the context of intuitionistic logic, since the types have clear "constructive" interpretations. In fact, one can develop constructive versions of higher-type semantics (based on intuitionistic, rather than classical logic) that clarify these constructive interpretations quite nicely, and are, in many ways, more interesting than the classical counterparts.

§12.4 Intuitionistic Logic

In contrast to second-order and higher-order logic, intuitionistic first-order logic represents a restriction of the classical version, intended to model a more "constructive" kind of reasoning. The following examples may serve to illustrate some of the underlying motivations.

Suppose someone came up to you one day and announced that they had determined a natural number x, with the property that if x is prime, the Riemann hypothesis is true, and if x is composite, the Riemann hypothesis is false. Great news! Whether the Riemann hypothesis is true or not is one of the big open questions of mathematics, and here they seem to have reduced the problem to one of calculation, that is, to the determination of whether a specific number is prime or not.

What is the magic value of x? They describe it as follows: x is the natural number that is equal to 7 if the Riemann hypothesis is true, and 9 otherwise.

Angrily, you demand your money back. From a classical point of view, the description above does in fact determine a unique value of x; but what you really want is a value of x that is given *explicitly*.

To take another, perhaps less contrived example, consider the following question. We know that it is possible to raise an irrational number to a rational power, and get a rational result. For example, $\sqrt{2}^2 = 2$. What is less clear is whether or not it is possible to raise an irrational number to an *irrational* power, and get a rational result. The following theorem answers this in the affirmative:

Theorem 124A. *There are irrational numbers a and b such that a^b is rational.*

Proof. Consider $\sqrt{2}^{\sqrt{2}}$. If this is rational, we are done: we can let $a = b = \sqrt{2}$. Otherwise, it is irrational. Then we have

$$(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = \sqrt{2}^2 = 2,$$

which is certainly rational. So, in this case, let a be $\sqrt{2}^{\sqrt{2}}$, and let b be $\sqrt{2}$. □

Does this constitute a valid proof? Most mathematicians feel that it does. But again, there is something a little bit unsatisfying here: we have proved the existence of a pair of real numbers with a certain property, without being able to say *which* pair of numbers it is. It is possible to prove the same result, but in such a way that the pair a, b *is* given in the proof: take $a = \sqrt{3}$ and $b = \log_3 4$. Then

$$a^b = \sqrt{3}^{\log_3 4} = 3^{1/2\cdot\log_3 4} = (3^{\log_3 4})^{1/2} = 4^{1/2} = 2,$$

since $3^{\log_3 x} = x$.

Intuitionistic logic is designed to model a kind of reasoning where moves like the one in the first proof are disallowed. Proving the existence of an x

satisfying $\alpha(x)$ means that you have to give a specific x, and a proof that it satisfies α, like in the second proof. Proving that α or β holds requires that you can prove one or the other.

Formally speaking, intuitionistic first-order logic is what you get if you restrict a derivation system for first-order logic in a certain way. Similarly, there are intuitionistic versions of second-order or higher-order logic. From the mathematical point of view, these are just formal deductive systems, but, as already noted, they are intended to model a kind of mathematical reasoning. One can take this to be the kind of reasoning that is justified on a certain philosophical view of mathematics (such as Brouwer's intuitionism); one can take it to be a kind of mathematical reasoning which is more "concrete" and satisfying (along the lines of Bishop's constructivism); and one can argue about whether or not the formal description captures the informal motivation. But whatever philosophical positions we may hold, we can study intuitionistic logic as a formally presented logic; and for whatever reasons, many mathematical logicians find it interesting to do so.

There is an informal constructive interpretation of the intuitionist connectives, usually known as the BHK interpretation (named after Brouwer, Heyting, and Kolmogorov). It runs as follows: a proof of $\alpha \wedge \beta$ consists of a proof of α paired with a proof of β; a proof of $\alpha \vee \beta$ consists of either a proof of α, or a proof of β, where we have explicit information as to which is the case; a proof of $\alpha \to \beta$ consists of a procedure, which transforms a proof of α to a proof of β; a proof of $\forall x\, \alpha(x)$ consists of a procedure which returns a proof of $\alpha(x)$ for any value of x; and a proof of $\exists x\, \alpha(x)$ consists of a value of x, together with a proof that this value satisfies α. One can describe the interpretation in computational terms known as the "Curry-Howard isomorphism" or the "wffs-as-types paradigm": think of a wff as specifying a certain kind of data type, and proofs as computational objects of these data types that enable us to see that the corresponding wff is true.

Intuitionistic logic is often thought of as being classical logic "minus" the law of the excluded middle. This following theorem makes this more precise.

Theorem 124B. *Intuitionistically, the following axiom schemata are equivalent:*

1. $(\alpha \to \bot) \to \neg\alpha$.
2. $\alpha \vee \neg\alpha$
3. $\neg\neg\alpha \to \alpha$

Obtaining instances of one schema from either of the others is a good exercise in intuitionistic logic.

The first deductive systems for intuitionistic propositional logic, put forth as formalizations of Brouwer's intuitionism, are due, independently, to Kolmogorov, Glivenko, and Heyting. The first formalization of intuitionistic first-order logic (and parts of intuitionist mathematics) is due to Heyting. Though

a number of classically valid schemata are not intuitionistically valid, many are.

The *double-negation translation* describes an important relationship between classical and intuitionist logic. It is defined inductively follows (think of α^N as the "intuitionist" translation of the classical wff α):

$$\begin{aligned}
\alpha^N &\equiv \neg\neg\alpha \quad \text{for atomic wffs } \alpha \\
(\alpha \wedge \beta)^N &\equiv (\alpha^N \wedge \beta^N) \\
(\alpha \vee \beta)^N &\equiv \neg\neg(\alpha^N \vee \beta^N) \\
(\alpha \to \beta)^N &\equiv (\alpha^N \to \beta^N) \\
(\forall x\, \alpha)^N &\equiv \forall x\, \alpha^N \\
(\exists x\, \alpha)^N &\equiv \neg\neg\exists x\, \alpha^N
\end{aligned}$$

Kolmogorov and Glivenko had versions of this translation for propositional logic; for predicate logic, it is due to Gödel and Gentzen, independently. We have

Theorem 124C.

1. *$\alpha \leftrightarrow \alpha^N$ is provable classically*
2. *If α is provable classically, then α^N is provable intuitionistically.*

We can now envision the following dialogue. Classical mathematician: "I've proved α!" Intuitionist mathematician: "Your proof isn't valid. What you've really proved is α^N." Classical mathematician: "Fine by me!" As far as the classical mathematician is concerned, the intuitionist is just splitting hairs, since the two are equivalent. But the intuitionist insists there is a difference.

Note that the above translation concerns pure logic only; it does not address the question as to what the appropriate *nonlogical* axioms are for classical and intuitionistic mathematics, or what the relationship is between them. But the following slight extension of the theorem above provides some useful information:

Theorem 124D. *If Γ proves α classically, Γ^N proves α^N intuitionistically.*

In other words, if α is provable from some hypotheses classically, then α^N is provable from their double-negation translations.

To show that a sentence or propositional wff is intuitionistically valid, all you have to do is provide a proof. But how can you show that it is not valid? For that purpose, we need a semantics that is sound, and preferably complete. A semantics due to Kripke nicely fits the bill.

We can play the same game we did for classical logic: define the semantics, and prove soundness and completeness. It is worthwhile, however, to note the following distinction. In the case of classical logic, the semantics was the "obvious" one, in a sense implicit in the meaning of the connectives. Though

one can provide some intuitive motivation for Kripke semantics, the latter does not offer the same feeling of inevitability. In addition, the notion of a classical structure is a natural mathematical one, so we can either take the notion of a structure to be a tool for studying classical first-order logic, or take classical first-order logic to be a tool for studying mathematical structures. In contrast, Kripke structures can only be viewed as a logical construct; they don't seem to have independent mathematical interest.

A Kripke structure $\mathfrak{M} = \langle W, R, V \rangle$ for a propositional language consists of a set W, partial order R on W with a least element, and an "monotone" assignment of propositional variables to the elements of W. The intuition is that the elements of W represent "worlds," or "states of knowledge"; an element $v \geq u$ represents a "possible future state" of u; and the propositional variables assigned to u are the propositions that are known to be true in state u. The forcing relation $\mathfrak{M}, w \Vdash \alpha$ then extends this relationship to arbitrary wffs in the language; read $\mathfrak{M}, w \Vdash \alpha$ as "α is true in state w." The relationship is defined inductively, as follows:

1. $\mathfrak{M}, w \Vdash p_i$ iff p_i is one of the propositional variables assigned to w.
2. $\mathfrak{M}, w \nVdash \bot$.
3. $\mathfrak{M}, w \Vdash (\alpha \wedge \beta)$ iff $\mathfrak{M}, w \Vdash \alpha$ and $\mathfrak{M}, w \Vdash \beta$.
4. $\mathfrak{M}, w \Vdash (\alpha \vee \beta)$ iff $\mathfrak{M}, w \Vdash \alpha$ or $\mathfrak{M}, w \Vdash \beta$.
5. $\mathfrak{M}, w \Vdash (\alpha \to \beta)$ iff, whenever $w' \geq w$ and $\mathfrak{M}, w' \Vdash \alpha$, then $\mathfrak{M}, w' \Vdash \beta$.

It is a good exercise to try to show that $\neg(p \wedge q) \to (\neg p \vee \neg q)$ is not intuitionistically valid, by cooking up a Kripke structure that provides a counterexample.

§12.5 Modal Logics

Consider the following example of a conditional sentence:

> If Jeremy is alone in that room, then he is drunk and naked and dancing on the chairs.

This is an example of a conditional assertion that may be materially true but nonetheless misleading, since it seems to suggest that there is a stronger link between the antecedent and conclusion other than simply that either the antecedent is false or the consequent true. That is, the wording suggests that the claim is not only true in this particular world (where it may be trivially true, because Jeremy is not alone in the room), but that, moreover, the conclusion *would have* been true *had* the antecedent been true. In other words, one can take the assertion to mean that the claim is true not just in this world, but in any "possible" world; or that it is *necessarily* true, as opposed to just true in this particular world.

Modal logic was designed to make sense of this kind of necessity. One obtains modal propositional logic from ordinary propositional logic by adding a box operator; which is to say, if α is a wff, so is $\Box\alpha$. Intuitively, $\Box\alpha$ asserts that α is *necessarily* true, or true in any possible world. $\Diamond\alpha$ is usually taken to be an abbreviation for $\neg\Box\neg\alpha$, and can be read as asserting that α is *possibly* true. Of course, modality can be added to predicate logic as well.

Kripke structures can be used to provide a semantics for modal logic; in fact, Kripke first designed this semantics with modal logic in mind. Rather than restricting to partial orders, more generally one has a set of "possible worlds," P, and a binary "accessibility" relation Rxy between worlds. Intuitively, Rpq asserts that the world q is compatible with p; i.e., if we are "in" world p, we have to entertain the possibility that the world could have been like q.

Modal logic is sometimes called an "intensional" logic, as opposed to an "extensional" one. The intended semantics for an extensional logic, like classical logic, will only refer to a single world, the "actual" one; while the semantics for an "intensional" logic relies on a more elaborate ontology. In addition to structureing necessity, one can use modality to structure other linguistic constructions, reinterpreting $\Box$ and $\Diamond$ according to the application. For example:

1. In provability logic, $\Box\alpha$ is read "α is provable" and $\Diamond\alpha$ is read "α is consistent."
2. In epistemic logic, one might read $\Box\alpha$ as "I know α" or "I believe α."
3. In temporal logic, one can read $\Box\alpha$ as "α is always true" and $\Diamond\alpha$ as "α is sometimes true."

One would like to augment logic with rules and axioms dealing with modality. For example, the system **S4** consists of the ordinary axioms and rules of propositional logic, together with the following axioms:

$$\Box(\alpha \to \beta) \to (\Box\alpha \to \Box\beta)$$
$$\Box\alpha \to \alpha$$
$$\Box\alpha \to \Box\Box\alpha$$

as well as a rule, "from α conclude $\Box\alpha$." **S5** adds the following axiom:

$$\Diamond\alpha \to \Box\Diamond\alpha$$

Variations of these axioms may be suitable for different applications; for example, S5 is usually taken to characterize the notion of logical necessity. And the nice thing is that one can usually find a semantics for which the derivation system is sound and complete by restricting the accessibility relation in the Kripke structures in natural ways. For example, **S4** corresponds to the class of Kripke structures in which the accessibility relation is reflexive and transitive. **S5** corresponds to the class of Kripke structures in which the accessibility relation is *universal*, which is to say that every world is accessible from every other; so $\Box\alpha$ holds if and only if α holds in every world.

§12.6 Other Logics

As you may have gathered by now, it is not hard to design a new logic. You too can create your own a syntax, make up a deductive system, and fashion a semantics to go with it. You might have to be a bit clever if you want the derivation system to be complete for the semantics, and it might take some effort to convince the world at large that your logic is truly interesting. But, in return, you can enjoy hours of good, clean fun, exploring your logic's mathematical and computational properties.

Recent decades have witnessed a veritable explosion of formal logics. Fuzzy logic is designed to model reasoning about vague properties. Probabilistic logic is designed to model reasoning about uncertainty. Default logics and nonmonotonic logics are designed to model defeasible forms of reasoning, which is to say, "reasonable" inferences that can later be overturned in the face of new information. There are epistemic logics, designed to model reasoning about knowledge; causal logics, designed to model reasoning about causal relationships; and even "deontic" logics, which are designed to model reasoning about moral and ethical obligations. Depending on whether the primary motivation for introducing these systems is philosophical, mathematical, or computational, you may find such creatures studies under the rubric of mathematical logic, philosophical logic, artificial intelligence, cognitive science, or elsewhere.

The list goes on and on, and the possibilities seem endless. We may never attain Leibniz' dream of reducing all of human reason to calculation—but that can't stop us from trying.

Part III

Computability

Chapter 13

Recursive Functions

§13.0 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 $f\colon \mathbb{N}^n \to \mathbb{N}$ 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 particularly 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 recursive 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.

§13.1 Primitive Recursion

A characteristic of the natural numbers is that every natural number can be reached from 0 by applying the successor operation +1 finitely many times—any natural number is either 0 or the successor of ... the successor of 0. One way to specify a function $h\colon \mathbb{N} \to \mathbb{N}$ that makes use of this fact is this: (a) specify what the value of h is for argument 0, and (b) also specify how to, given the value of $h(x)$, compute the value of $h(x+1)$. For (a) tells us directly what $h(0)$ is, so h is defined for 0. Now, using the instruction given by (b) for $x = 0$, we can compute $h(1) = h(0+1)$ from $h(0)$. Using the same instructions for $x = 1$, we compute $h(2) = h(1+1)$ from $h(1)$, and so on. For every natural number x, we'll eventually reach the step where we define $h(x)$ from $h(x+1)$, and so $h(x)$ is defined for all $x \in \mathbb{N}$.

For instance, suppose we specify $h\colon \mathbb{N} \to \mathbb{N}$ by the following two equations:

$$\begin{aligned} h(0) &= 1 \\ h(x+1) &= 2 \cdot h(x) \end{aligned}$$

If we already know how to multiply, then these equations give us the information required for (a) and (b) above. By successively applying the second equation, we get that

$$\begin{aligned} h(1) &= 2 \cdot h(0) = 2, \\ h(2) &= 2 \cdot h(1) = 2 \cdot 2, \\ h(3) &= 2 \cdot h(2) = 2 \cdot 2 \cdot 2, \\ &\vdots \end{aligned}$$

We see that the function h we have specified is $h(x) = 2^x$.

The characteristic feature of the natural numbers guarantees that there is only one function h that meets these two criteria. A pair of equations like these is called a *definition by primitive recursion* of the function h. It is so-called because we define h "recursively," i.e., the definition, specifically the second equation, involves h itself on the right-hand-side. It is "primitive" because in defining $h(x+1)$ we only use the value $h(x)$, i.e., the immediately preceding value. This is the simplest way of defining a function on $\mathbb{N}$ recursively.

We can define even more fundamental functions like addition and multiplication by primitive recursion. In these cases, however, the functions in question are 2-place. We fix one of the argument places, and use the other for the recursion. E.g, to define $\mathrm{add}(x, y)$ we can fix x and define the value first for $y = 0$ and then for $y+1$ in terms of y. Since x is fixed, it will appear on the left and

on the right side of the defining equations.

$$\begin{aligned}\mathrm{add}(x, 0) &= x\\ \mathrm{add}(x, y+1) &= \mathrm{add}(x, y) + 1\end{aligned}$$

These equations specify the value of add for all x *and* y. To find $\mathrm{add}(2, 3)$, for instance, we apply the defining equations for $x = 2$, using the first to find $\mathrm{add}(2, 0) = 2$, then using the second to successively find $\mathrm{add}(2, 1) = 2 + 1 = 3$, $\mathrm{add}(2, 2) = 3 + 1 = 4$, $\mathrm{add}(2, 3) = 4 + 1 = 5$.

In the definition of add we used $+$ on the right-hand-side of the second equation, but only to add 1. In other words, we used the successor function $\mathrm{succ}(z) = z+1$ and applied it to the previous value $\mathrm{add}(x, y)$ to define $\mathrm{add}(x, y+1)$. So we can think of the recursive definition as given in terms of a single function which we apply to the previous value. However, it doesn't hurt—and sometimes is necessary—to allow the function to depend not just on the previous value but also on x and y. Consider:

$$\begin{aligned}\mathrm{mult}(x, 0) &= 0\\ \mathrm{mult}(x, y+1) &= \mathrm{add}(\mathrm{mult}(x, y), x)\end{aligned}$$

This is a primitive recursive definition of a function mult by applying the function add to both the preceding value $\mathrm{mult}(x, y)$ and the first argument x. It also defines the function $\mathrm{mult}(x, y)$ for all arguments x and y. For instance, $\mathrm{mult}(2, 3)$ is determined by successively computing $\mathrm{mult}(2, 0)$, $\mathrm{mult}(2, 1)$, $\mathrm{mult}(2, 2)$, and $\mathrm{mult}(2, 3)$:

$$\begin{aligned}&\mathrm{mult}(2, 0) = 0\\ &\mathrm{mult}(2, 1) = \mathrm{mult}(2, 0+1) = \mathrm{add}(\mathrm{mult}(2, 0), 2) = \mathrm{add}(0, 2) = 2\\ &\mathrm{mult}(2, 2) = \mathrm{mult}(2, 1+1) = \mathrm{add}(\mathrm{mult}(2, 1), 2) = \mathrm{add}(2, 2) = 4\\ &\mathrm{mult}(2, 3) = \mathrm{mult}(2, 2+1) = \mathrm{add}(\mathrm{mult}(2, 2), 2) = \mathrm{add}(4, 2) = 6\end{aligned}$$

The general pattern then is this: to give a primitive recursive definition of a function $h(x_0, \ldots, x_{k-1}, y)$, we provide two equations. The first defines the value of $h(x_0, \ldots, x_{k-1}, 0)$ without reference to h. The second defines the value of $h(x_0, \ldots, x_{k-1}, y+1)$ in terms of $h(x_0, \ldots, x_{k-1}, y)$, the other arguments x_0, $\ldots$, x_{k-1}, and y. Only the immediately preceding value of h may be used in that second equation. If we think of the operations given by the right-hand-sides of these two equations as themselves being functions f and g, then the general pattern to define a new function h by primitive recursion is this:

$$\begin{aligned}h(x_0, \ldots, x_{k-1}, 0) &= f(x_0, \ldots, x_{k-1})\\ h(x_0, \ldots, x_{k-1}, y+1) &= g(x_0, \ldots, x_{k-1}, y, h(x_0, \ldots, x_{k-1}, y))\end{aligned}$$

In the case of add, we have $k = 1$ and $f(x_0) = x_0$ (the identity function), and $g(x_0, y, z) = z + 1$ (the 3-place function that returns the successor of its third

argument):

$$\begin{aligned}\text{add}(x_0, 0) &= f(x_0) = x_0\\ \text{add}(x_0, y+1) &= g(x_0, y, \text{add}(x_0, y)) = \text{succ}(\text{add}(x_0, y))\end{aligned}$$

In the case of mult, we have $f(x_0) = 0$ (the constant function always returning 0) and $g(x_0, y, z) = \text{add}(z, x_0)$ (the 3-place function that returns the sum of its last and first argument):

$$\begin{aligned}\text{mult}(x_0, 0) &= f(x_0) = 0\\ \text{mult}(x_0, y+1) &= g(x_0, y, \text{mult}(x_0, y)) = \text{add}(\text{mult}(x_0, y), x_0)\end{aligned}$$

§13.2 Composition

If f and g are two one-place functions of natural numbers, we can compose them: $h(x) = g(f(x))$. The new function $h(x)$ is then defined by *composition* from the functions f and g. We'd like to generalize this to functions of more than one argument.

Here's one way of doing this: suppose f is a k-place function, and $g_0, \ldots, g_{k-1}$ are k functions which are all n-place. Then we can define a new n-place function h as follows:

$$h(x_0, \ldots, x_{n-1}) = f(g_0(x_0, \ldots, x_{n-1}), \ldots, g_{k-1}(x_0, \ldots, x_{n-1}))$$

If f and all g_i are computable, so is h: To compute $h(x_0, \ldots, x_{n-1})$, first compute the values $y_i = g_i(x_0, \ldots, x_{n-1})$ for each $i = 0, \ldots, k-1$. Then feed these values into f to compute $h(x_0, \ldots, x_{k-1}) = f(y_0, \ldots, y_{k-1})$.

This may seem like an overly restrictive characterization of what happens when we compute a new function using some existing ones. For one thing, sometimes we do not use all the arguments of a function, as when we defined $g(x, y, z) = \text{succ}(z)$ for use in the primitive recursive definition of add. Suppose we are allowed use of the following functions:

$$P_i^n(x_0, \ldots, x_{n-1}) = x_i$$

The functions P_i^k are called *projection* functions: P_i^n is an n-place function. Then g can be defined by

$$g(x, y, z) = \text{succ}(P_2^3(x, y, z)).$$

Here the role of f is played by the 1-place function succ, so $k = 1$. And we have one 3-place function P_2^3 which plays the role of g_0. The result is a 3-place function that returns the successor of the third argument.

The projection functions also allow us to define new functions by reordering or identifying arguments. For instance, the function $h(x) = \text{add}(x, x)$ can be defined by

$$h(x_0) = \text{add}(P_0^1(x_0), P_0^1(x_0)).$$

Here $k = 2$, $n = 1$, the role of $f(y_0, y_1)$ is played by add, and the roles of $g_0(x_0)$ and $g_1(x_0)$ are both played by $P^1_0(x_0)$, the one-place projection function (aka the identity function).

If $f(y_0, y_1)$ is a function we already have, we can define the function $h(x_0, x_1) = f(x_1, x_0)$ by

$$h(x_0, x_1) = f(P^2_1(x_0, x_1), P^2_0(x_0, x_1)).$$

Here $k = 2$, $n = 2$, and the roles of g_0 and g_1 are played by P^2_1 and P^2_0, respectively.

You may also worry that $g_0, \ldots, g_{k-1}$ are all required to have the same arity n. (Remember that the *arity* of a function is the number of arguments; an n-place function has arity n.) 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

$$h(x, y) = f(P^2_0(x, y), g(P^2_0(x, y), P^2_0(x, y), P^2_1(x, y)), P^2_1(x, y)).$$

Then h is the composition of f with P^2_0, l, and P^2_1, where

$$l(x, y) = g(P^2_0(x, y), P^2_0(x, y), P^2_1(x, y)),$$

i.e., l is the composition of g with P^2_0, P^2_0, and P^2_1.

§13.3 Primitive Recursion Functions

Let us record again how we can define new functions from existing ones using primitive recursion and composition.

Definition 133A. *Suppose f is a k-place function ($k \geq 1$) and g is a $(k+2)$-place function. The function defined by* primitive recursion from f and g *is the $(k+1)$-place function h defined by the equations*

$$h(x_0, \ldots, x_{k-1}, 0) = f(x_0, \ldots, x_{k-1})$$
$$h(x_0, \ldots, x_{k-1}, y+1) = g(x_0, \ldots, x_{k-1}, y, h(x_0, \ldots, x_{k-1}, y))$$

Definition 133B. *Suppose f is a k-place function, and $g_0, \ldots, g_{k-1}$ are k functions which are all n-place. The function defined by* composition from f and $g_0, \ldots, g_{k-1}$ *is the n-place function h defined by*

$$h(x_0, \ldots, x_{n-1}) = f(g_0(x_0, \ldots, x_{n-1}), \ldots, g_{k-1}(x_0, \ldots, x_{n-1})).$$

In addition to succ and the projection functions

$$P^n_i(x_0, \ldots, x_{n-1}) = x_i,$$

for each natural number n and $i < n$, we will include among the primitive recursive functions the function $\text{zero}(x) = 0$.

Definition 133C. *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 n-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 is by defining it in terms of "stages." Let S_0 denote the set of starting functions: zero, succ, and the projections. These are the primitive recursive functions of stage 0. Once a stage 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 already in S_i. Then

$$S = \bigcup_{i \in \mathbb{N}} S_i$$

is the set of all primitive recursive functions

Let us verify that add is a primitive recursive function.

Proposition 133D. *The addition function* $\mathrm{add}(x, y) = x + y$ *is primitive recursive.*

Proof. We already have a primitive recursive definition of add in terms of two functions f and g which matches the format of Definition 133A:

$$\begin{aligned} \mathrm{add}(x_0, 0) &= f(x_0) = x_0 \\ \mathrm{add}(x_0, y+1) &= g(x_0, y, \mathrm{add}(x_0, y)) = \mathrm{succ}(\mathrm{add}(x_0, y)) \end{aligned}$$

So add is primitive recursive provided f and g are as well. $f(x_0) = x_0 = P_0^1(x_0)$, and the projection functions count as primitive recursive, so f is primitive recursive. The function g is the three-place function $g(x_0, y, z)$ defined by

$$g(x_0, y, z) = \mathrm{succ}(z).$$

This does not yet tell us that g is primitive recursive, since g and succ are not quite the same function: succ is one-place, and g has to be three-place. But we can define g "officially" by composition as

$$g(x_0, y, z) = \text{succ}(P_2^3(x_0, y, z))$$

Since succ and P_2^3 count as primitive recursive functions, g does as well, since it can be defined by composition from primitive recursive functions. □

Proposition 133E. *The multiplication function* $\text{mult}(x, y) = x \cdot y$ *is primitive recursive.*

Proof. Exercise. □

Example 13.3.6. Here's our very first example of a primitive recursive definition:

$$\begin{aligned} h(0) &= 1 \\ h(y+1) &= 2 \cdot h(y). \end{aligned}$$

This function cannot fit into the form required by Definition 133A, since $k = 0$. The definition also involves the constants 1 and 2. To get around the first problem, let's introduce a dummy argument and define the function h':

$$\begin{aligned} h'(x_0, 0) &= f(x_0) = 1 \\ h'(x_0, y+1) &= g(x_0, y, h'(x_0, y)) = 2 \cdot h'(x_0, y). \end{aligned}$$

The function $f(x_0) = 1$ can be defined from succ and zero by composition: $f(x_0) = \text{succ}(\text{zero}(x_0))$. The function g can be defined by composition from $g'(z) = 2 \cdot z$ and projections:

$$g(x_0, y, z) = g'(P_2^3(x_0, y, z))$$

and g' in turn can be defined by composition as

$$g'(z) = \text{mult}(g''(z), P_0^1(z))$$

and

$$g''(z) = \text{succ}(f(z)),$$

where f is as above: $f(z) = \text{succ}(\text{zero}(z))$. Now that we have h', we can use composition again to let $h(y) = h'(P_0^1(y), P_0^1(y))$. This shows that h can be defined from the basic functions using a sequence of compositions and primitive recursions, so h is primitive recursive.

§13.4 Primitive Recursion Notations

One advantage to having the precise inductive 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 h is defined by composition from a k-place function f and n-place functions g_0, $\ldots$, g_{k-1}, and we have assigned notations F, G_0, $\ldots$, G_{k-1} to the latter functions. Then, using a new symbol $\mathrm{Comp}_{k,n}$, we can denote the function h by $\mathrm{Comp}_{k,n}[F, G_0, \ldots, G_{k-1}]$.

For functions defined by primitive recursion, we can use analogous notations. Suppose the $(k+1)$-ary function h is defined by primitive recursion from the k-ary function f and the $(k+2)$-ary function g, and the notations assigned to f and g are F and G, respectively. Then the notation assigned to h is $\mathrm{Rec}_k[F, G]$.

Recall that the addition function is defined by primitive recursion as

$$\begin{aligned} \mathrm{add}(x_0, 0) &= P_0^1(x_0) = x_0 \\ \mathrm{add}(x_0, y+1) &= \mathrm{succ}(P_2^3(x_0, y, \mathrm{add}(x_0, y))) = \mathrm{add}(x_0, y) + 1 \end{aligned}$$

Here the role of f is played by P_0^1, and the role of g is played by $\mathrm{succ}(P_2^3(x_0, y, z))$, which is assigned the notation $\mathrm{Comp}_{1,3}[\mathrm{succ}, P_2^3]$ as it is the result of defining a function by composition from the 1-ary function succ and the 3-ary function P_2^3. With this setup, we can denote the addition function by

$$\mathrm{Rec}_1[P_0^1, \mathrm{Comp}_{1,3}[\mathrm{succ}, P_2^3]].$$

Having these notations sometimes proves useful, e.g., when enumerating primitive recursive functions.

§13.5 Primitive Recursive Functions are Computable

Suppose a function h is defined by primitive recursion

$$\begin{aligned} h(\vec{x}, 0) &= f(\vec{x}) \\ h(\vec{x}, y+1) &= g(\vec{x}, y, h(\vec{x}, y)) \end{aligned}$$

and suppose the functions f and g are computable. (We use $\vec{x}$ to abbreviate x_0, $\ldots$, x_{k-1}.) Then $h(\vec{x}, 0)$ can obviously be computed, since it is just $f(\vec{x})$ which we assume is computable. $h(\vec{x}, 1)$ can then also be computed, since $1 = 0 + 1$ and so $h(\vec{x}, 1)$ is just

$$h(\vec{x}, 1) = g(\vec{x}, 0, h(\vec{x}, 0)) = g(\vec{x}, 0, f(\vec{x})).$$

We can go on in this way and compute

$$\begin{aligned}
h(\vec{x}, 2) &= g(\vec{x}, 1, h(\vec{x}, 1)) = g(\vec{x}, 1, g(\vec{x}, 0, f(\vec{x}))) \\
h(\vec{x}, 3) &= g(\vec{x}, 2, h(\vec{x}, 2)) = g(\vec{x}, 2, g(\vec{x}, 1, g(\vec{x}, 0, f(\vec{x})))) \\
h(\vec{x}, 4) &= g(\vec{x}, 3, h(\vec{x}, 3)) = g(\vec{x}, 3, g(\vec{x}, 2, g(\vec{x}, 1, g(\vec{x}, 0, f(\vec{x}))))) \\
&\vdots
\end{aligned}$$

Thus, to compute $h(\vec{x}, y)$ in general, successively compute $h(\vec{x}, 0)$, $h(\vec{x}, 1)$, ..., until we reach $h(\vec{x}, y)$.

Thus, a primitive recursive definition 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, this means that every primitive recursive function is computable.

§13.6 Examples of Primitive Recursive Functions

We already have some examples of primitive recursive functions: the addition and multiplication functions add and mult. The identity function $\mathrm{id}(x) = x$ is primitive recursive, since it is just P_0^1. The constant functions $\mathrm{const}_n(x) = n$ are primitive recursive since they can be defined from zero and succ by successive composition. This is useful when we want to use constants in primitive recursive definitions, e.g., if we want to define the function $f(x) = 2 \cdot x$ can obtain it by composition from $\mathrm{const}_n(x)$ and multiplication as $f(x) = \mathrm{mult}(\mathrm{const}_2(x), P_0^1(x))$. We'll make use of this trick from now on.

Proposition 136A. *The exponentiation function* $\exp(x, y) = x^y$ *is primitive recursive.*

Proof. We can define exp primitive recursively as

$$\begin{aligned}
\exp(x, 0) &= 1 \\
\exp(x, y + 1) &= \mathrm{mult}(x, \exp(x, y)).
\end{aligned}$$

Strictly speaking, this is not a recursive definition from primitive recursive functions. Officially, though, we have:

$$\begin{aligned}
\exp(x, 0) &= f(x) \\
\exp(x, y + 1) &= g(x, y, \exp(x, y)).
\end{aligned}$$

where

$$\begin{aligned}
f(x) &= \mathrm{succ}(\mathrm{zero}(x)) = 1 \\
g(x, y, z) &= \mathrm{mult}(P_0^3(x, y, z), P_2^3(x, y, z)) = x \cdot z
\end{aligned}$$

and so f and g are defined from primitive recursive functions by composition. □

Proposition 136B. *The predecessor function* $\mathrm{pred}(y)$ *defined by*

$$\mathrm{pred}(y) = \begin{cases} 0 & \text{if } y = 0 \\ y - 1 & \text{otherwise} \end{cases}$$

is primitive recursive.

Proof. Note that

$$\begin{aligned} \mathrm{pred}(0) &= 0 \text{ and} \\ \mathrm{pred}(y+1) &= y. \end{aligned}$$

This is almost a primitive recursive definition. It does not, strictly speaking, fit into the pattern of definition by primitive recursion, since that pattern requires at least one extra argument x. It is also odd in that it does not actually use $\mathrm{pred}(y)$ in the definition of $\mathrm{pred}(y+1)$. But we can first define $\mathrm{pred}'(x, y)$ by

$$\begin{aligned} \mathrm{pred}'(x, 0) &= \mathrm{zero}(x) = 0, \\ \mathrm{pred}'(x, y+1) &= P^3_1(x, y, \mathrm{pred}'(x, y)) = y. \end{aligned}$$

and then define pred from it by composition, e.g., as $\mathrm{pred}(x) = \mathrm{pred}'(\mathrm{zero}(x), P^1_0(x))$.□

Proposition 136C. *The factorial function* $\mathrm{fac}(x) = x! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot x$ *is primitive recursive.*

Proof. The obvious primitive recursive definition is

$$\begin{aligned} \mathrm{fac}(0) &= 1 \\ \mathrm{fac}(y+1) &= \mathrm{fac}(y) \cdot (y+1). \end{aligned}$$

Officially, we have to first define a two-place function h

$$\begin{aligned} h(x, 0) &= \mathrm{const}_1(x) \\ h(x, y+1) &= g(x, y, h(x, y)) \end{aligned}$$

where $g(x, y, z) = \mathrm{mult}(P^3_2(x, y, z), \mathrm{succ}(P^3_1(x, y, z)))$ and then let

$$\mathrm{fac}(y) = h(P^1_0(y), P^1_0(y)) = h(y, y).$$

From now on we'll be a bit more laissez-faire and not give the official definitions by composition and primitive recursion. □

Proposition 136D. *Truncated subtraction,* $x \dot{-} y$*, defined by*

$$x \dot{-} y = \begin{cases} 0 & \text{if } x < y \\ x - y & \text{otherwise} \end{cases}$$

is primitive recursive.

Proof. We have:

$$\begin{aligned} x \dot{-} 0 &= x \\ x \dot{-} (y+1) &= \operatorname{pred}(x \dot{-} y) \end{aligned}$$ □

Proposition 136E. *The distance between x and y, $|x-y|$, is primitive recursive.*

Proof. We have $|x-y| = (x \dot{-} y) + (y \dot{-} x)$, so the distance can be defined by composition from $+$ and $\dot{-}$, which are primitive recursive. □

Proposition 136F. *The maximum of x and y, $\max(x,y)$, is primitive recursive.*

Proof. We can define $\max(x,y)$ by composition from $+$ and $\dot{-}$ by

$$\max(x,y) = x + (y \dot{-} x).$$

If x is the maximum, i.e., $x \geq y$, then $y \dot{-} x = 0$, so $x + (y \dot{-} x) = x + 0 = x$. If y is the maximum, then $y \dot{-} x = y - x$, and so $x + (y \dot{-} x) = x + (y - x) = y$.□

Proposition 136G. *The minimum of x and y, $\min(x,y)$, is primitive recursive.*

Proof. Exercise. □

Proposition 136H. *The set of primitive recursive functions is closed under the following two operations:*

1. *Finite sums: if $f(\vec{x}, z)$ is primitive recursive, then so is the function*

$$g(\vec{x}, y) = \sum_{z=0}^{y} f(\vec{x}, z).$$

2. *Finite products: if $f(\vec{x}, z)$ is primitive recursive, then so is the function*

$$h(\vec{x}, y) = \prod_{z=0}^{y} f(\vec{x}, z).$$

Proof. For example, finite sums are defined recursively by the equations

$$\begin{aligned} g(\vec{x}, 0) &= f(\vec{x}, 0) \\ g(\vec{x}, y+1) &= g(\vec{x}, y) + f(\vec{x}, y+1). \end{aligned}$$ □

§13.7 Primitive Recursive Relations

Definition 137A. *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

$$\begin{aligned} \chi_{\text{IsZero}}(0) &= 1, \\ \chi_{\text{IsZero}}(x+1) &= 0. \end{aligned}$$

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$(x \dot{-} y)$

Proposition 137B. *The set of primitive recursive relations is closed under Boolean operations, that is, if $P(\vec{x})$ and $Q(\vec{x})$ are primitive recursive, so are*

1. $\neg P(\vec{x})$
2. $P(\vec{x}) \wedge Q(\vec{x})$
3. $P(\vec{x}) \vee Q(\vec{x})$
4. $P(\vec{x}) \to Q(\vec{x})$

Proof. Suppose $P(\vec{x})$ and $Q(\vec{x})$ are primitive recursive, i.e., their characteristic functions χ_P and χ_Q are. We have to show that the characteristic functions of $\neg P(\vec{x})$, etc., are also primitive recursive.

$$\chi_{\neg P}(\vec{x}) = \begin{cases} 0 & \text{if } \chi_P(\vec{x}) = 1 \\ 1 & \text{otherwise} \end{cases}$$

We can define $\chi_{\neg P}(\vec{x})$ as $1 \dot{-} \chi_P(\vec{x})$.

$$\chi_{P \wedge Q}(\vec{x}) = \begin{cases} 1 & \text{if } \chi_P(\vec{x}) = \chi_Q(\vec{x}) = 1 \\ 0 & \text{otherwise} \end{cases}$$

We can define $\chi_{P \wedge Q}(\vec{x})$ as $\chi_P(\vec{x}) \cdot \chi_Q(\vec{x})$ or as $\min(\chi_P(\vec{x}), \chi_Q(\vec{x}))$. Similarly,

$$\begin{aligned} \chi_{P \vee Q}(\vec{x}) &= \max(\chi_P(\vec{x}), \chi_Q(\vec{x}))) \text{ and} \\ \chi_{P \to Q}(\vec{x}) &= \max(1 \dot{-} \chi_P(\vec{x}), \chi_Q(\vec{x})). \end{aligned}$$

□

Proposition 137C. *The set of primitive recursive relations is closed under bounded quantification, i.e., if $R(\vec{x}, z)$ is a primitive recursive relation, then so are the relations*

$$(\forall z < y)\ R(\vec{x}, z) \text{ and}$$
$$(\exists z < y)\ R(\vec{x}, z).$$

$(\forall z < y)\ R(\vec{x}, z)$ holds of $\vec{x}$ and y if and only if $R(\vec{x}, z)$ holds for every z less than y, and similarly for $(\exists z < y)\ R(\vec{x}, z)$.

Proof. By convention, we take $(\forall z < 0)\ R(\vec{x}, z)$ to be true (for the trivial reason that there are no z less than 0) and $(\exists z < 0)\ R(\vec{x}, z)$ to be false. A bounded universal quantifier functions just like a finite product or iterated minimum, i.e., if $P(\vec{x}, y) \Leftrightarrow (\forall z < y)\ R(\vec{x}, z)$ then $\chi_P(\vec{x}, y)$ can be defined by

$$\chi_P(\vec{x}, 0) = 1$$
$$\chi_P(\vec{x}, y+1) = \min(\chi_P(\vec{x}, y), \chi_R(\vec{x}, y))).$$

Bounded existential quantification can similarly be defined using max. Alternatively, it can be defined from bounded universal quantification, using the equivalence $(\exists z < y)\ R(\vec{x}, z) \leftrightarrow \neg(\forall z < y)\ \neg R(\vec{x}, z)$. Note that, for example, a bounded quantifier of the form $(\exists x \leq y)\ \dots x \dots$ is equivalent to $(\exists x < y + 1)\ \dots x \dots$. □

Another useful primitive recursive function is 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

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

One can use this to justify definitions of primitive recursive functions by cases from primitive recursive relations:

Proposition 137D. *If $g_0(\vec{x}), \dots, g_m(\vec{x})$ are primitive recursive functions, and $R_0(\vec{x}), \dots, R_{m-1}(\vec{x})$ are primitive recursive 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.

Proof. When $m = 1$, this is just the function defined by

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

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

§13.8 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, ..., 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 independently 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 x and y to the least $z < y$ such that $R(x, z)$. It, too, can be computed, by testing whether $R(x, 0)$, $R(x, 1)$, ..., $R(x, y - 1)$. But why is it primitive recursive?

Proposition 138A. *If $R(\vec{x}, z)$ is primitive recursive, so is the function $m_R(\vec{x}, y)$ which returns the least z less than y such that $R(\vec{x}, z)$ holds, if there is one, and y otherwise. We will write the function m_R as*

$$(\min z < y)\, R(\vec{x}, z),$$

Proof. Note than there can be no $z < 0$ such that $R(\vec{x}, z)$ since there is no $z < 0$ at all. So $m_R(\vec{x}, 0) = 0$.

In case the bound is of the form $y + 1$ we have three cases:

1. There is a $z < y$ such that $R(\vec{x}, z)$, in which case $m_R(\vec{x}, y+1) = m_R(\vec{x}, y)$.
2. There is no such $z < y$ but $R(\vec{x}, y)$ holds, then $m_R(\vec{x}, y + 1) = y$.
3. There is no $z < y + 1$ such that $R(\vec{x}, z)$, then $m_R(\vec{z}, y + 1) = y + 1$.

So we can define $m_R(\vec{x}, 0)$ by primitive recursion as follows:

$$m_R(\vec{x}, 0) = 0$$
$$m_R(\vec{x}, y + 1) = \begin{cases} m_R(\vec{x}, y) & \text{if } m_R(\vec{x}, y) \neq y \\ y & \text{if } m_R(\vec{x}, y) = y \text{ and } R(\vec{x}, y) \\ y + 1 & \text{otherwise.} \end{cases}$$

Note that there is a $z < y$ such that $R(\vec{x}, z)$ iff $m_R(\vec{x}, y) \neq y$. □

§13.9 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 "x divides y", written $x \mid y$. The relation $x \mid y$ holds if division of y by x 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 $\mathrm{Prime}(x)$, which holds iff x is prime, can be defined by

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

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:

$$\begin{aligned} p(0) &= 2 \\ p(x+1) &= \mathrm{nextPrime}(p(x)) \end{aligned}$$

Since $\mathrm{nextPrime}(x)$ 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 $x\,! + 1$.

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

This shows, that $\mathrm{nextPrime}(x)$ 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 $\leq x\,! + 1$.)

§13.10 Sequences

The set of primitive recursive functions is remarkably robust. But we will be able to do even more once we have developed a 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 Λ denote 0.

The reason that this coding of sequences works is the so-called Fundamental Theorem of Arithmetic: every natural number $n \geq 2$ can be written in one and only one way in the form

$$n = p_0^{a_0} \cdot p_1^{a_1} \cdot \cdots \cdot p_k^{a_k}$$

with $a_k \geq 1$. This guarantees that the mapping $\langle\rangle(a_0, \ldots, a_k) = \langle a_0, \ldots, a_k\rangle$ is injective: different sequences are mapped to different numbers; to each number only at most one sequence corresponds.

We'll now show that the operations of determining the length of a sequence, determining its ith element, appending an element to a sequence, and concatenating two sequences, are all primitive recursive.

Proposition 1310A. *The function* $\mathrm{len}(s)$*, which returns the length of the sequence* s*, is primitive recursive.*

Proof. Let $R(i, s)$ be the relation defined by

$$R(i, s) \text{ iff } p_i \mid s \wedge p_{i+1} \nmid s.$$

R is clearly primitive recursive. Whenever s is the code of a non-empty sequence, i.e.,

$$s = p_0^{a_0+1} \cdot \cdots \cdot p_k^{a_k+1},$$

$R(i, s)$ holds if p_i is the largest prime such that $p_i \mid s$, i.e., $i = k$. The length of s thus is $i + 1$ iff p_i is the largest prime that divides s, so we can let

$$\mathrm{len}(s) = \begin{cases} 0 & \text{if } s = 0 \text{ or } s = 1 \\ 1 + (\min i < s)\, R(i, s) & \text{otherwise} \end{cases}$$

We can use bounded minimization, since there is only one i that satisfies $R(s, i)$ when s is a code of a sequence, and if i exists it is less than s itself. □

Proposition 1310B. *The function* $\mathrm{append}(s, a)$*, which returns the result of appending* a *to the sequence* s*, is primitive recursive.*

Proof. append can be defined by:

$$\text{append}(s,a) = \begin{cases} 2^{a+1} & \text{if } s = 0 \text{ or } s = 1 \\ s \cdot p_{\text{len}(s)}^{a+1} & \text{otherwise.} \end{cases}$$

□

Proposition 1310C. *The function* element(s,i)*, which returns the* i*th element of* s *(where the initial element is called the* 0*th), or* 0 *if* i *is greater than or equal to the length of* s*, is primitive recursive.*

Proof. Note that a is the ith element of s iff p_i^{a+1} is the largest power of p_i that divides s, i.e., $p_i^{a+1} \mid s$ but $p_i^{a+2} \nmid s$. So:

$$\text{element}(s,i) = \begin{cases} 0 & \text{if } i \geq \text{len}(s) \\ (\min a < s)\,(p_i^{a+2} \nmid s) & \text{otherwise.} \end{cases}$$

□

Instead of using the official names for the functions defined above, we introduce a more compact notation. We will use $(s)_i$ instead of element(s,i), and $\langle s_0, \ldots, s_k \rangle$ to abbreviate

$$\text{append}(\text{append}(\ldots \text{append}(\Lambda, s_0) \ldots), s_k).$$

Note that if s has length k, the elements of s are $(s)_0$, …, $(s)_{k-1}$.

Proposition 1310D. *The function* concat(s,t)*, which concatenates two sequences, is primitive recursive.*

Proof. We want a function concat with the property that

$$\text{concat}(\langle a_0, \ldots, a_k \rangle, \langle b_0, \ldots, b_l \rangle) = \langle a_0, \ldots, a_k, b_0, \ldots, b_l \rangle.$$

We'll use 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

$$\text{concat}(s,t) = \text{hconcat}(s,t,\text{len}(t)).$$

□

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

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 or 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, we can define concat using bounded search. All we need to do is write down a primitive recursive *specification* of the object (number of the concatenated sequence) we are looking for, and a bound on how far to look. The following works:

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

Proposition 1310E. *The function* $\text{subseq}(s,i,n)$ *which returns the subsequence of* s *of length* n *beginning at the* i*th element, is primitive recursive.*

Proof. Exercise. □

§13.11 Trees

Sometimes it is useful to represent trees as natural numbers, just like we can represent sequences by numbers and properties of and operations on them by primitive recursive relations and functions on their codes. We'll use sequences and their codes to do this. A tree can be either a single node (possibly with a label) or else a node (possibly with a label) connected to a number of subtrees. The node is called the *root* of the tree, and the subtrees it is connected to its *immediate subtrees*.

We code trees recursively as a sequence $\langle k, d_1, \ldots, d_k\rangle$, where k is the number of immediate subtrees and $d_1, \ldots, d_k$ the codes of the immediate subtrees. If the nodes have labels, they can be included after the immediate subtrees. So a tree consisting just of a single node with label l would be coded by $\langle 0, l\rangle$, and a tree consisting of a root (labelled l_1) connected to two single nodes (labelled l_2, l_3) would be coded by $\langle 2, \langle 0, l_2\rangle, \langle 0, l_3\rangle, l_1\rangle$.

Proposition 1311A. *The function* $\text{SubtreeSeq}(t)$*, which returns the code of a sequence the elements of which are the codes of all subtrees of the tree with code* t*, is primitive recursive.*

Proof. First note that $\text{ISubtrees}(t) = \text{subseq}(t, 1, (t)_0)$ is primitive recursive and returns the codes of the immediate subtrees of a tree t. Now we can define a helper function $\text{hSubtreeSeq}(t,n)$ which computes the sequence of all subtrees which are n nodes removed from the root. The sequence of subtrees

of t which is 0 nodes removed from the root—in other words, begins at the root of t—is the sequence consisting just of t. To obtain a sequence of all level $n+1$ subtrees of t, we concatenate the level n subtrees with a sequence consisting of all immediate subtrees of the level n subtrees. To get a list of all these, note that if $f(x)$ is a primitive recursive function returning codes of sequences, then $g_f(s,k) = f((s)_0) \frown \ldots \frown f((s)_k)$ is also primitive recursive:

$$\begin{aligned} g(s,0) &= f((s)_0) \\ g(s,k+1) &= g(s,k) \frown f((s)_{k+1}) \end{aligned}$$

For instance, if s is a sequence of trees, then $h(s) = g_{\mathrm{ISubtrees}}(s, \mathrm{len}(s))$ gives the sequence of the immediate subtrees of the elements of s. We can use it to define hSubtreeSeq by

$$\begin{aligned} \mathrm{hSubtreeSeq}(t,0) &= \langle t \rangle \\ \mathrm{hSubtreeSeq}(t,n+1) &= \mathrm{hSubtreeSeq}(t,n) \frown h(\mathrm{hSubtreeSeq}(t,n)). \end{aligned}$$

The maximum level of subtrees in a tree coded by t, i.e., the maximum distance between the root and a leaf node, is bounded by the code t. So a sequence of codes of all subtrees of the tree coded by t is given by $\mathrm{hSubtreeSeq}(t,t)$. □

§13.12 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:

$$\begin{aligned} h_0(\vec{x},0) &= f_0(\vec{x}) \\ h_1(\vec{x},0) &= f_1(\vec{x}) \\ h_0(\vec{x},y+1) &= g_0(\vec{x},y,h_0(\vec{x},y),h_1(\vec{x},y)) \\ h_1(\vec{x},y+1) &= g_1(\vec{x},y,h_0(\vec{x},y),h_1(\vec{x},y)) \end{aligned}$$

This is an instance of *simultaneous recursion*. Another useful way of defining functions is to give the value of $h(\vec{x},y+1)$ in terms of *all* the values $h(\vec{x},0)$, $\ldots$, $h(\vec{x},y)$, as in the following definition:

$$\begin{aligned} h(\vec{x},0) &= f(\vec{x}) \\ h(\vec{x},y+1) &= g(\vec{x},y,\langle h(\vec{x},0),\ldots,h(\vec{x},y)\rangle). \end{aligned}$$

The following schema captures this idea more succinctly:

$$h(\vec{x},y) = g(\vec{x},y,\langle h(\vec{x},0),\ldots,h(\vec{x},y-1)\rangle)$$

with the understanding that the last argument to g is just the empty sequence when y is 0. In either formulation, the idea is that in computing the "successor step," the function h 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:

$$h(\vec{x}, y) = \begin{cases} g(\vec{x}, y, h(\vec{x}, k(\vec{x}, y))) & \text{if } k(\vec{x}, y) < y \\ f(\vec{x}) & \text{otherwise} \end{cases}$$

In other words, the value of h at y can be computed in terms of the value of h 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:

$$\begin{aligned} h(\vec{x}, 0) &= f(\vec{x}) \\ h(\vec{x}, y+1) &= g(\vec{x}, y, h(k(\vec{x}), y)) \end{aligned}$$

This, too, can be simulated with ordinary primitive recursion. (Doing so is tricky. For a hint, try unwinding the computation by hand.)

§13.13 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 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. The Ackermann–Péter 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 we are 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 is 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. 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.

§13.14 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 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, ..., 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 $g_0(\vec{x})$, ..., $g_k(\vec{x})$. 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 1314A. *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 1314B. *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.

§13.15 The Normal Form Theorem

Theorem 1315A (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 the fact that s codes the computation of f on input x can be checked primitive recursively using only x and the definition e. Consequently, the relation T, "the function with index e has a computation for input x, and s codes this computation," 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. Basically, we can search through all numbers until we find one that codes a computation of the function with index e for input x. 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.

§13.16 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 1316A. *The halting function h is not partial recursive.*

Proof. If h were partial recursive, we could define

$$d(y) = \begin{cases} 1 & \text{if } h(y,y) = 0 \\ \mu x\ x \neq x & \text{otherwise.} \end{cases}$$

Since no number x satisfies $x \neq x$, there is no $\mu x\ x \neq x$, and so $d(y)\uparrow$ iff $h(y,y) \neq 0$. 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. □

§13.17 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 1317A. *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 1317A and Definition 1314B 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 1317A is *less* general than Definition 1314B. 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.

Problems

Problem 1. Prove Proposition 133E by showing that the primitive recursive definition of mult can be put into the form required by Definition 133A and showing that the corresponding functions f and g are primitive recursive.

Problem 2. Give the complete primitive recursive notation for mult.

Problem 3. Prove Proposition 136G.

Problem 4. Show that

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

is primitive recursive.

Problem 5. Show that integer division $d(x,y) = \lfloor x/y \rfloor$ (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 definition of d using primitive recursion and composition.

Problem 6. Show that the three place relation $x \equiv y \mod n$ (congruence modulo n) is primitive recursive.

Problem 7. Suppose $R(\vec{x}, z)$ is primitive recursive. Define the function $m'_R(\vec{x}, y)$ which returns the least z less than y such that $R(\vec{x}, z)$ holds, if there is one, and 0 otherwise, by primitive recursion from χ_R.

Problem 8. Define integer division $d(x,y)$ using bounded minimization.

Problem 9. Show that there is a primitive recursive function sconcat(s) with the property that

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

Problem 10. Show that there is a primitive recursive function tail(s) with the property that

$$\text{tail}(\Lambda) = 0 \text{ and}$$
$$\text{tail}(\langle s_0, \ldots, s_k \rangle) = \langle s_1, \ldots, s_k \rangle.$$

Problem 11. Prove Proposition 1310E.

Problem 12. The definition of hSubtreeSeq in the proof of Proposition 1311A in general includes repetitions. Give an alternative definition which guarantees that the code of a subtree occurs only once in the resulting list.

Problem 13. Define the remainder function $r(x,y)$ by course-of-values recursion. (If x, y are natural numbers and $y > 0$, $r(x,y)$ is the number less than y such that $x = z \times y + r(x,y)$ for some z. For definiteness, let's say that if $y = 0$, $r(x,0) = 0$.)

Chapter 14

Computability Theory

§14.0 Introduction

The branch of logic known as *Computability Theory* deals with issues having to do with the computability, or relative computability, of functions and sets. It is a evidence of Kleene's influence that the subject used to be known as *Recursion Theory*, and today, both names are commonly used.

Let us call a function $f\colon \mathbb{N} \nrightarrow \mathbb{N}$ *partial computable* if it can be computed in some model of computation. If f is total we will simply say that f is *computable*. A relation R with computable characteristic function χ_R is also called computable. 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.

One can explore the subject without having to refer to a specific model of computation. To do this, one shows that there is a universal partial computable function, $\mathrm{Un}(k, x)$. This allows us to enumerate the partial computable functions. We will adopt the notation φ_k to denote the k-th unary partial computable function, defined by $\varphi_k(x) \simeq \mathrm{Un}(k, x)$. (Kleene used $\{k\}$ for this purpose, but this notation has not been used as much recently.) Slightly more generally, we can uniformly enumerate the partial computable functions of arbitrary arities, and we will use φ_k^n to denote the k-th n-ary partial recursive function.

Recall that if $f(\vec{x}, y)$ is a total or partial function, then $\mu y\ f(\vec{x}, y)$ is the function of $\vec{x}$ that returns the least y such that $f(\vec{x}, y) = 0$, assuming that all of $f(\vec{x}, 0), \ldots, f(\vec{x}, y-1)$ are defined; if there is no such y, $\mu y\ f(\vec{x}, y)$ is undefined. If $R(\vec{x}, y)$ is a relation, $\mu y\ R(\vec{x}, y)$ is defined to be the least y such that $R(\vec{x}, y)$ is true; in other words, the least y such that *one minus* the characteristic function of R is equal to zero at $\vec{x}, y$.

To show that a function is computable, there are two ways one can proceed:

1. Rigorously: describe a Turing machine or partial recursive function ex-

plicitly, and show that it computes the function you have in mind;

2. Informally: describe an algorithm that computes it, and appeal to Church's thesis.

There is no fine line between the two; a detailed description of an algorithm should provide enough information so that it is relatively clear how one could, in principle, design the right Turing machine or sequence of partial recursive definitions. Fully rigorous definitions are unlikely to be informative, and we will try to find a happy medium between these two approaches; in short, we will try to find intuitive yet rigorous proofs that the precise definitions could be obtained.

§14.1 Coding Computations

In every model of computation, it is possible to do the following:

1. Describe the *definitions* of computable functions in a systematic way. For instance, you can think of Turing machine specifications, recursive definitions, or programs in a programming language as providing these definitions.

2. Describe the complete record of the computation of a function given by some definition for a given input. For instance, a Turing machine computation can be described by the sequence of configurations (state of the machine, contents of the tape) for each step of computation.

3. Test whether a putative record of a computation is in fact the record of how a computable function with a given definition would be computed for a given input.

4. Extract from such a description of the complete record of a computation the value of the function for a given input. For instance, the contents of the tape in the very last step of a halting Turing machine computation is the value.

Using coding, it is possible to assign to each description of a computable function a numerical *index* in such a way that the instructions can be recovered from the index in a computable way. Similarly, the complete record of a computation can be coded by a single number as well. The resulting arithmetical relation "s codes the record of computation of the function with index e for input x" and the function "output of computation sequence with code s" are then computable; in fact, they are primitive recursive.

This fundamental fact is very powerful, and allows us to prove a number of striking and important results about computability, independently of the model of computation chosen.

§14.2 The Normal Form Theorem

Theorem 142A (Kleene's Normal Form Theorem). *There are a primitive recursive relation $T(k, x, s)$ and a primitive recursive function $U(s)$, with the following property: if f is any partial computable function, then for some k,*

$$f(x) \simeq U(\mu s\, T(k, x, s))$$

for every x.

Proof Sketch. For any model of computation one can rigorously define a description of the computable function f and code such description using a natural number k. One can also rigorously define a notion of "computation sequence" which records the process of computing the function with index k for input x. These computation sequences can likewise be coded as numbers s. This can be done in such a way that (a) it is decidable whether a number s codes the computation sequence of the function with index k on input x and (b) what the end result of the computation sequence coded by s is. In fact, the relation in (a) and the function in (b) are primitive recursive. □

In order to give a rigorous proof of the Normal Form Theorem, we would have to fix a model of computation and carry out the coding of descriptions of computable functions and of computation sequences in detail, and verify that the relation T and function U are primitive recursive. For most applications, it suffices that T and U are computable and that U is total.

It is probably best to remember the proof of the normal form theorem in slogan form: $\mu s\, T(k, x, s)$ searches for a computation sequence of the function with index k on input x, and U returns the output of the computation sequence if one can be found.

T and U can be used to define the enumeration φ_0, φ_1, φ_2, ... From now on, we will assume that we have fixed a suitable choice of T and U, and take the equation

$$\varphi_e(x) \simeq U(\mu s\, T(e, x, s))$$

to be the *definition* of φ_e.

Here is another useful fact:

Theorem 142B. *Every partial computable function has infinitely many indices.*

Again, this is intuitively clear. Given any (description of) a computable function, one can come up with a different description which computes the same function (input-output pair) but does so, e.g., by first doing something that has no effect on the computation (say, test if $0 = 0$, or count to 5, etc.). The index of the altered description will always be different from the original index. Both are indices of the same function, just computed slightly differently.

§14.3 The *s*-*m*-*n* Theorem

The next theorem is known as the "*s*-*m*-*n* theorem," for a reason that will be clear in a moment. The hard part is understanding just what the theorem says; once you understand the statement, it will seem fairly obvious.

Theorem 143A. *For each pair of natural numbers n and m, there is a primitive recursive function s_n^m such that for every sequence x, a_0, ..., a_{m-1}, y_0 ,..., y_{n-1}, we have*

$$\varphi^n_{s^m_n(x,a_0,\ldots,a_{m-1})}(y_0,\ldots,y_{n-1}) \simeq \varphi^{m+n}_x(a_0,\ldots,a_{m-1},y_0,\ldots,y_{n-1}).$$

It is helpful to think of s_n^m as acting on *programs*. That is, s_n^m takes a program, x, for an $(m+n)$-ary function, as well as fixed inputs a_0, ..., a_{m-1}; and it returns a program, $s_n^m(x, a_0, \ldots, a_{m-1})$, for the n-ary function of the remaining arguments. It you think of x as the description of a Turing machine, then $s_n^m(x, a_0, \ldots, a_{m-1})$ is the Turing machine that, on input y_0, ..., y_{n-1}, prepends a_0, ..., a_{m-1} to the input string, and runs x. Each s_n^m is then just a primitive recursive function that finds a code for the appropriate Turing machine.

§14.4 The Universal Partial Computable Function

Theorem 144A. *There is a universal partial computable function* $\mathrm{Un}(k,x)$. *In other words, there is a function* $\mathrm{Un}(k,x)$ *such that:*

1. $\mathrm{Un}(k,x)$ *is partial computable.*
2. *If $f(x)$ is any partial computable function, then there is a natural number k such that $f(x) \simeq \mathrm{Un}(k,x)$ for every x.*

Proof. Let $\mathrm{Un}(k,x) \simeq U(\mu s\, T(k,x,s))$ in Kleene's normal form theorem. □

This is just a precise way of saying that we have an effective enumeration of the partial computable functions; the idea is that if we write f_k for the function defined by $f_k(x) = \mathrm{Un}(k,x)$, then the sequence f_0, f_1, f_2, ... includes all the partial computable functions, with the property that $f_k(x)$ can be computed "uniformly" in k and x. For simplicity, we are using a binary function that is universal for unary functions, but by coding sequences of numbers we can easily generalize this to more arguments. For example, note that if $f(x,y,z)$ is a 3-place partial recursive function, then the function $g(x) \simeq f((x)_0, (x)_1, (x)_2)$ is a unary recursive function.

§14.5 No Universal Computable Function

Theorem 145A. *There is no universal computable function. In other words, the universal function* $\mathrm{Un}'(k,x) = \varphi_k(x)$ *is not computable.*

Proof. This theorem says that there is no *total* computable function that is universal for the total computable functions. The proof is a simple diagonalization: if $\mathrm{Un}'(k, x)$ were total and computable, then

$$d(x) = \mathrm{Un}'(x, x) + 1$$

would also be total and computable. However, for every k, $d(k)$ is not equal to $\mathrm{Un}'(k, k)$. □

Theorem Theorem 144A above shows that we can get around this diagonalization argument, but only at the expense of allowing partial functions. It is worth trying to understand what goes wrong with the diagonalization argument, when we try to apply it in the partial case. In particular, the function $h(x) = \mathrm{Un}(x, x) + 1$ *is* partial recursive. Suppose h is the k-th function in the enumeration; what can we say about $h(k)$?

§14.6 The Halting Problem

Since, in our construction, $\mathrm{Un}(k, x)$ is defined if and only if the computation of the function coded by k produces a value for input x, it is natural to ask if we can decide whether this is the case. And in fact, it is not. For the Turing machine model of computation, this means that whether a given Turing machine halts on a given input is computationally undecidable. The following theorem is therefore known as the "undecidability of the halting problem." We will provide two proofs below. The first continues the thread of our previous discussion, while the second is more direct.

Theorem 146A. *Let*

$$h(k, x) = \begin{cases} 1 & \textit{if } \mathrm{Un}(k, x) \textit{ is defined} \\ 0 & \textit{otherwise.} \end{cases}$$

Then h is not computable.

Proof. If h were computable, we would have a universal computable function, as follows. Suppose h is computable, and define

$$\mathrm{Un}'(k, x) = \begin{cases} fnUn(k, x) & \text{if } h(k, x) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

But now $\mathrm{Un}'(k, x)$ is a total function, and is computable if h is. For instance, we could define g using primitive recursion, by

$$g(0, k, x) \simeq 0$$
$$g(y + 1, k, x) \simeq \mathrm{Un}(k, x);$$

then

$$\mathrm{Un}'(k, x) \simeq g(h(k, x), k, x).$$

And since $\mathrm{Un}'(k, x)$ agrees with $\mathrm{Un}(k, x)$ wherever the latter is defined, Un' is universal for those partial computable functions that happen to be total. But this contradicts Theorem 145A. □

Proof. Suppose $h(k, x)$ were computable. Define the function g by

$$g(x) = \begin{cases} 0 & \text{if } h(x, x) = 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The function g is partial computable; for example, one can define it as $\mu y\ h(x, x) = 0$. So, for some k, $g(x) \simeq \mathrm{Un}(k, x)$ for every x. Is g defined at k? If it is, then, by the definition of g, $h(k, k) = 0$. By the definition of f, this means that $\mathrm{Un}(k, k)$ is undefined; but by our assumption that $g(k) \simeq \mathrm{Un}(k, x)$ for every x, this means that $g(k)$ is undefined, a contradiction. On the other hand, if $g(k)$ is undefined, then $h(k, k) \neq 0$, and so $h(k, k) = 1$. But this means that $\mathrm{Un}(k, k)$ is defined, i.e., that $g(k)$ is defined. □

We can describe this argument in terms of Turing machines. Suppose there were a Turing machine H that took as input a description of a Turing machine K and an input x, and decided whether or not K halts on input x. Then we could build another Turing machine G which takes a single input x, calls H to decide if machine x halts on input x, and does the opposite. In other words, if H reports that x halts on input x, G goes into an infinite loop, and if H reports that x doesn't halt on input x, then G just halts. Does G halt on input G? The argument above shows that it does if and only if it doesn't—a contradiction. So our supposition that there is a such Turing machine H, is false.

§14.7 Comparison with Russell's Paradox

It is instructive to compare and contrast the arguments in this section with Russell's paradox:

1. Russell's paradox: let $S = \{x : x \notin x\}$. Then $x \in S$ if and only if $x \notin S$, a contradiction.

   Conclusion: There is no such set S. Assuming the existence of a "set of all sets" is inconsistent with the other axioms of set theory.

2. A modification of Russell's paradox: let F be the "function" from the set of all functions to $\{0, 1\}$, defined by

   $$F(f) = \begin{cases} 1 & \text{if } f \text{ is in the domain of } f, \text{ and } f(f) = 0 \\ 0 & \text{otherwise} \end{cases}$$

   A similar argument shows that $F(F) = 0$ if and only if $F(F) = 1$, a contradiction.

Conclusion: F is not a function. The "set of all functions" is too big to be the domain of a function.

3. The diagonalization argument: let f_0, f_1, ... be the enumeration of the partial computable functions, and let $G\colon \mathbb{N} \to \{0,1\}$ be defined by

$$G(x) = \begin{cases} 1 & \text{if } f_x(x) \downarrow = 0 \\ 0 & \text{otherwise} \end{cases}$$

If G is computable, then it is the function f_k for some k. But then $G(k) = 1$ if and only if $G(k) = 0$, a contradiction.

Conclusion: G is not computable. Note that according to the axioms of set theory, G is still a function; there is no paradox here, just a clarification.

That talk of partial functions, computable functions, partial computable functions, and so on can be confusing. The set of all partial functions from $\mathbb{N}$ to $\mathbb{N}$ is a big collection of objects. Some of them are total, some of them are computable, some are both total and computable, and some are neither. Keep in mind that when we say "function," by default, we mean a total function. Thus we have:

1. computable functions
2. partial computable functions that are not total
3. functions that are not computable
4. partial functions that are neither total nor computable

To sort this out, it might help to draw a big square representing all the partial functions from $\mathbb{N}$ to $\mathbb{N}$, and then mark off two overlapping regions, corresponding to the total functions and the computable partial functions, respectively. It is a good exercise to see if you can describe an object in each of the resulting regions in the diagram.

§14.8 Computable Sets

We can extend the notion of computability from computable functions to computable sets:

Definition 148A. *Let S be a set of natural numbers. Then S is* computable *iff its characteristic function is. In other words, S is computable iff the function*

$$\chi_S(x) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{otherwise} \end{cases}$$

is computable. Similarly, a relation $Rx_0 \dots x_{k-1}$ is computable if and only if its characteristic function is.

Computable sets are also called *decidable*.

Notice that we now have a number of notions of computability: for partial functions, for functions, and for sets. Do not get them confused! The Turing machine computing a partial function returns the output of the function, for input values at which the function is defined; the Turing machine computing a set returns either 1 or 0, after deciding whether or not the input value is in the set or not.

§14.9 Computably Enumerable Sets

Definition 149A. *A set is* computably enumerable *if it is empty or the range of a computable function.*

Computably enumerable sets are also called *recursively enumerable* instead. This is the original terminology, and today both are commonly used, as well as the abbreviations "c.e." and "r.e."

You should think about what the definition means, and why the terminology is appropriate. The idea is that if S is the range of the computable function f, then

$$S = \{f(0), f(1), f(2), \dots\},$$

and so f can be seen as "enumerating" the elements of S. Note that according to the definition, f need not be an increasing function, i.e., the enumeration need not be in increasing order. In fact, f need not even be injective, so that the constant function $f(x) = 0$ enumerates the set $\{0\}$.

Any computable set is computably enumerable. To see this, suppose S is computable. If S is empty, then by definition it is computably enumerable. Otherwise, let a be any element of S. Define f by

$$f(x) = \begin{cases} x & \text{if } \chi_S(x) = 1 \\ a & \text{otherwise.} \end{cases}$$

Then f is a computable function, and S is the range of f.

§14.10 Equivalent Defininitions of Computably Enumerable Sets

The following gives a number of important equivalent statements of what it means to be computably enumerable.

Theorem 1410A. *Let S be a set of natural numbers. Then the following are equivalent:*

1. *S is computably enumerable.*
2. *S is the range of a* partial *computable function.*

3. S is empty or the range of a primitive recursive function.

4. S is the domain *of a partial computable function.*

The first three clauses say that we can equivalently take any non-empty computably enumerable set to be enumerated by either a computable function, a partial computable function, or a primitive recursive function. The fourth clause tells us that if S is computably enumerable, then for some index e,

$$S = \{x : \varphi_e(x) \downarrow\}.$$

In other words, S is the set of inputs on for which the computation of φ_e halts. For that reason, computably enumerable sets are sometimes called *semi-decidable*: if a number is in the set, you eventually get a "yes," but if it isn't, you never get a "no"!

Proof. Since every primitive recursive function is computable and every computable function is partial computable, (3) implies (1) and (1) implies (2). (Note that if S is empty, S is the range of the partial computable function that is nowhere defined.) If we show that (2) implies (3), we will have shown the first three clauses equivalent.

So, suppose S is the range of the partial computable function φ_e. If S is empty, we are done. Otherwise, let a be any element of S. By Kleene's normal form theorem, we can write

$$\varphi_e(x) = U(\mu s\, T(e, x, s)).$$

In particular, $\varphi_e(x) \downarrow$ and $= y$ if and only if there is an s such that $T(e, x, s)$ and $U(s) = y$. Define $f(z)$ by

$$f(z) = \begin{cases} U((z)_1) & \text{if } T(e, (z)_0, (z)_1) \\ a & \text{otherwise.} \end{cases}$$

Then f is primitive recursive, because T and U are. Expressed in terms of Turing machines, if z codes a pair $\langle (z)_0, (z)_1 \rangle$ such that $(z)_1$ is a halting computation of machine e on input $(z)_0$, then f returns the output of the computation; otherwise, it returns a.We need to show that S is the range of f, i.e., for any natural number y, $y \in S$ if and only if it is in the range of f. In the forwards direction, suppose $y \in S$. Then y is in the range of φ_e, so for some x and s, $T(e, x, s)$ and $U(s) = y$; but then $y = f(\langle x, s \rangle)$. Conversely, suppose y is in the range of f. Then either $y = a$, or for some z, $T(e, (z)_0, (z)_1)$ and $U((z)_1) = y$. Since, in the latter case, $\varphi_e(x) \downarrow = y$, either way, y is in S.

(The notation $\varphi_e(x) \downarrow = y$ means "$\varphi_e(x)$ is defined and equal to y." We could just as well use $\varphi_e(x) = y$, but the extra arrow is sometimes helpful in reminding us that we are dealing with a partial function.)

To finish up the proof of Theorem 1410A, it suffices to show that (1) and (4) are equivalent. First, let us show that (1) implies (4). Suppose S is the range of a computable function f, i.e.,

$$S = \{y : \text{for some } x, f(x) = y\}.$$

Let

$$g(y) = \mu x\ f(x) = y.$$

Then g is a partial computable function, and $g(y)$ is defined if and only if for some x, $f(x) = y$. In other words, the domain of g is the range of f. Expressed in terms of Turing machines: given a Turing machine F that enumerates the elements of S, let G be the Turing machine that semi-decides S by searching through the outputs of F to see if a given element is in the set.

Finally, to show (4) implies (1), suppose that S is the domain of the partial computable function φ_e, i.e.,

$$S = \{x : \varphi_e(x)\downarrow\}.$$

If S is empty, we are done; otherwise, let a be any element of S. Define f by

$$f(z) = \begin{cases} (z)_0 & \text{if } T(e, (z)_0, (z)_1) \\ a & \text{otherwise.} \end{cases}$$

Then, as above, a number x is in the range of f if and only if $\varphi_e(x)\downarrow$, i.e., if and only if $x \in S$. Expressed in terms of Turing machines: given a machine M_e that semi-decides S, enumerate the elements of S by running through all possible Turing machine computations, and returning the inputs that correspond to halting computations. □

The fourth clause of Theorem 1410A provides us with a convenient way of enumerating the computably enumerable sets: for each e, let W_e denote the domain of φ_e. Then if A is any computably enumerable set, $A = W_e$, for some e.

The following provides yet another characterization of the computably enumerable sets.

Theorem 1410B. *A set S is computably enumerable if and only if there is a computable relation Rxy such that*

$$S = \{x : \exists y\, Rxy\}.$$

Proof. In the forward direction, suppose S is computably enumerable. Then for some e, $S = W_e$. For this value of e we can write S as

$$S = \{x : \exists y\, T(e, x, y)\}.$$

In the reverse direction, suppose $S = \{x : \exists y\, Rxy\}$. Define f by

$$f(x) \simeq \mu y\ AtomRx, y.$$

Then f is partial computable, and S is the domain of f. □

§14.11 Computably Enumerable Sets are Closed under Union and Intersection

The following theorem gives some closure properties on the set of computably enumerable sets.

Theorem 1411A. *Suppose A and B are computably enumerable. Then so are $A \cap B$ and $A \cup B$.*

Proof. Theorem 1410A allows us to use various characterizations of the computably enumerable sets. By way of illustration, we will provide a few different proofs.

For the first proof, suppose A is enumerated by a computable function f, and B is enumerated by a computable function g. Let

$$h(x) = \mu y \ (f(y) = x \vee g(y) = x) \text{ and}$$
$$j(x) = \mu y \ (f((y)_0) = x \wedge g((y)_1) = x).$$

Then $A \cup B$ is the domain of h, and $A \cap B$ is the domain of j.

Here is what is going on, in computational terms: given procedures that enumerate A and B, we can semi-decide if an element x is in $A \cup B$ by looking for x in either enumeration; and we can semi-decide if an element x is in $A \cap B$ for looking for x in both enumerations at the same time.

For the second proof, suppose again that A is enumerated by f and B is enumerated by g. Let

$$k(x) = \begin{cases} f(x/2) & \text{if } x \text{ is even} \\ g((x-1)/2) & \text{if } x \text{ is odd.} \end{cases}$$

Then k enumerates $A \cup B$; the idea is that k just alternates between the enumerations offered by f and g. Enumerating $A \cap B$ is tricker. If $A \cap B$ is empty, it is trivially computably enumerable. Otherwise, let c be any element of $A \cap B$, and define l by

$$l(x) = \begin{cases} f((x)_0) & \text{if } f((x)_0) = g((x)_1) \\ c & \text{otherwise.} \end{cases}$$

In computational terms, l runs through pairs of elements in the enumerations of f and g, and outputs every match it finds; otherwise, it just stalls by outputting c.

For the last proof, suppose A is the *domain* of the partial function $m(x)$ and B is the domain of the partial function $n(x)$. Then $A \cap B$ is the domain of the partial function $m(x) + n(x)$.

In computational terms, if A is the set of values for which m halts and B is the set of values for which n halts, $A \cap B$ is the set of values for which both procedures halt.

Expressing $A \cup B$ as a set of halting values is more difficult, because one has to simulate m and n in parallel. Let d be an index for m and let e be an

index for n; in other words, $m = \varphi_d$ and $n = \varphi_e$. Then $A \cup B$ is the domain of the function

$$p(x) = \mu y\ (T(d, x, y) \vee T(e, x, y)).$$

In computational terms, on input x, p searches for either a halting computation for m or a halting computation for n, and halts if it finds either one. □

§14.12 Computably Enumerable Sets not Closed under Complement

Suppose A is computably enumerable. Is the complement of A, $\overline{A} = \mathbb{N} \setminus A$, necessarily computably enumerable as well? The following theorem and corollary show that the answer is "no."

Theorem 1412A. *Let A be any set of natural numbers. Then A is computable if and only if both A and $\overline{A}$ are computably enumerable.*

Proof. The forwards direction is easy: if A is computable, then $\overline{A}$ is computable as well ($\chi_A = 1 \dot{-} \chi_{\overline{A}}$), and so both are computably enumerable.

In the other direction, suppose A and $\overline{A}$ are both computably enumerable. Let A be the domain of φ_d, and let $\overline{A}$ be the domain of φ_e. Define h by

$$h(x) = \mu s\ (T(d, x, s) \vee T(e, x, s)).$$

In other words, on input x, h searches for either a halting computation of φ_d or a halting computation of φ_e. Now, if $x \in A$, it will succeed in the first case, and if $x \in \overline{A}$, it will succeed in the second case. So, h is a total computable function. But now we have that for every x, $x \in A$ if and only if $T(e, x, h(x))$, i.e., if φ_e is the one that is defined. Since $T(e, x, h(x))$ is a computable relation, A is computable. □

It is easier to understand what is going on in informal computational terms: to decide A, on input x search for halting computations of φ_e and φ_f. One of them is bound to halt; if it is φ_e, then x is in A, and otherwise, x is in $\overline{A}$.

Corollary 1412B. *$\overline{K_0}$ is not computably enumerable.*

Proof. We know that K_0 is computably enumerable, but not computable. If $\overline{K_0}$ were computably enumerable, then K_0 would be computable by Theorem 1412A. □

§14.13 Reducibility

We now know that there is at least one set, K_0, that is computably enumerable but not computable. It should be clear that there are others. The method of reducibility provides a powerful method of showing that other sets have these properties, without constantly having to return to first principles.

Generally speaking, a "reduction" of a set A to a set B is a method of transforming answers to whether or not elements are in B into answers as to whether or not elements are in A. We will focus on a notion called "many-one reducibility," but there are many other notions of reducibility available, with varying properties. Notions of reducibility are also central to the study of computational complexity, where efficiency issues have to be considered as well. For example, a set is said to be "NP-complete" if it is in NP and every NP problem can be reduced to it, using a notion of reduction that is similar to the one described below, only with the added requirement that the reduction can be computed in polynomial time.

We have already used this notion implicitly. Define the set K by

$$K = \{x : \varphi_x(x)\downarrow\},$$

i.e., $K = \{x : x \in W_x\}$. Our proof that the halting problem in unsolvable, Theorem 146A, shows most directly that K is not computable. Recall that K_0 is the set

$$K_0 = \{\langle e, x\rangle : \varphi_e(x)\downarrow\}.$$

i.e. $K_0 = \{\langle x, e\rangle : x \in W_e\}$. It is easy to extend any proof of the uncomputability of K to the uncomputability of K_0: if K_0 were computable, we could decide whether or not an element x is in K simply by asking whether or not the pair $\langle x, x\rangle$ is in K_0. The function f which maps x to $\langle x, x\rangle$ is an example of a *reduction* of K to K_0.

Definition 1413A. *Let A and B be sets. Then A is said to be* many-one reducible *to B, written $A \leq_m B$, if there is a computable function f such that for every natural number x,*

$$x \in A \quad \textit{if and only if} \quad f(x) \in B.$$

If A is many-one reducible to B and vice-versa, then A and B are said to be many-one equivalent*, written $A \equiv_m B$.*

If the function f in the definition above happens to be injective, A is said to be *one-one reducible* to B. Most of the reductions described below meet this stronger requirement, but we will not use this fact.

It is true, but by no means obvious, that one-one reducibility really is a stronger requirement than many-one reducibility. In other words, there are infinite sets A and B such that A is many-one reducible to B but not one-one reducible to B.

§14.14 Properties of Reducibility

The intuition behind writing $A \leq_m B$ is that A is "no harder than" B. The following two propositions support this intuition.

Proposition 1414A. *If $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$.*

Proof. Composing a reduction of A to B with a reduction of B to C yields a reduction of A to C. (You should check the details!) □

Proposition 1414B. *Let A and B be any sets, and suppose A is many-one reducible to B.*

1. *If B is computably enumerable, so is A.*

2. *If B is computable, so is A.*

Proof. Let f be a many-one reduction from A to B. For the first claim, just check that if B is the domain of a partial function g, then A is the domain of $g \circ f$:

$$\begin{aligned} x \in A &\text{iff } f(x) \in B \\ &\text{iff } g(f(x)) \downarrow . \end{aligned}$$

For the second claim, remember that if B is computable then B and $\overline{B}$ are computably enumerable. It is not hard to check that f is also a many-one reduction of $\overline{A}$ to $\overline{B}$, so, by the first part of this proof, A and $\overline{A}$ are computably enumerable. So A is computable as well. (Alternatively, you can check that $\chi_A = \chi_B \circ f$; so if χ_B is computable, then so is χ_A.) □

A more general notion of reducibility called *Turing reducibility* is useful in other contexts, especially for proving undecidability results. Note that by corollary 1412B, the complement of K_0 is not reducible to K_0, since it is not computably enumerable. But, intuitively, if you knew the answers to questions about K_0, you would know the answer to questions about its complement as well. A set A is said to be Turing reducible to B if one can determine answers to questions in A using a computable procedure that can ask questions about B. This is more liberal than many-one reducibility, in which (1) you are only allowed to ask one question about B, and (2) a "yes" answer has to translate to a "yes" answer to the question about A, and similarly for "no." It is still the case that if A is Turing reducible to B and B is computable then A is computable as well (though, as we have seen, the analogous statement does not hold for computable enumerability).

You should think about the various notions of reducibility we have discussed, and understand the distinctions between them. We will, however, only deal with many-one reducibility in this chapter. Incidentally, both types of reducibility discussed in the last paragraph have analogues in computational complexity, with the added requirement that the Turing machines run in polynomial time: the complexity version of many-one reducibility is known as *Karp reducibility*, while the complexity version of Turing reducibility is known as *Cook reducibility*.

§14.15 Complete Computably Enumerable Sets

Definition 1415A. *A set A is a* complete computably enumerable set *(under many-one reducibility) if*

1. *A is computably enumerable, and*
2. *for any other computably enumerable set B, $B \leq_m A$.*

In other words, complete computably enumerable sets are the "hardest" computably enumerable sets possible; they allow one to answer questions about *any* computably enumerable set.

Theorem 1415B. *K, K_0, and K_1 are all complete computably enumerable sets.*

Proof. To see that K_0 is complete, let B be any computably enumerable set. Then for some index e,

$$B = W_e = \{x : \varphi_e(x)\downarrow\}.$$

Let f be the function $f(x) = \langle e, x\rangle$. Then for every natural number x, $x \in B$ if and only if $f(x) \in K_0$. In other words, f reduces B to K_0.

To see that K_1 is complete, note that in the proof of Proposition 1416A we reduced K_0 to it. So, by Proposition 1414A, any computably enumerable set can be reduced to K_1 as well.

K can be reduced to K_0 in much the same way. □

So, it turns out that all the examples of computably enumerable sets that we have considered so far are either computable, or complete. This should seem strange! Are there any examples of computably enumerable sets that are neither computable nor complete? The answer is yes, but it wasn't until the middle of the 1950s that this was established by Friedberg and Muchnik, independently.

§14.16 An Example of Reducibility

Let us consider an application of Proposition 1414B.

Proposition 1416A. *Let*

$$K_1 = \{e : \varphi_e(0)\downarrow\}.$$

Then K_1 is computably enumerable but not computable.

Proof. Since $K_1 = \{e : \exists s\, T(e, 0, s)\}$, K_1 is computably enumerable by Theorem 1410B.

To show that K_1 is not computable, let us show that K_0 is reducible to it.

This is a little bit tricky, since using K_1 we can only ask questions about computations that start with a particular input, 0. Suppose you have a smart friend who can answer questions of this type (friends like this are known as "oracles"). Then suppose someone comes up to you and asks you whether or not $\langle e, x\rangle$ is in K_0, that is, whether or not machine e halts on input x. One thing you can do is build another machine, e_x, that, for *any* input, ignores that input and instead runs e on input x. Then clearly the question as to whether machine e halts on input x is equivalent to the question as to whether machine e_x halts on input 0 (or any other input). So, then you ask your friend whether this new machine, e_x, halts on input 0; your friend's answer to the modified question provides the answer to the original one. This provides the desired reduction of K_0 to K_1.

Using the universal partial computable function, let f be the 3-ary function defined by

$$f(x, y, z) \simeq \varphi_x(y).$$

Note that f ignores its third input entirely. Pick an index e such that $f = \varphi^3_e$; so we have

$$\varphi^3_e(x, y, z) \simeq \varphi_x(y).$$

By the *s-m-n* theorem, there is a function $s(e, x, y)$ such that, for every z,

$$\begin{aligned}\varphi_{s(e,x,y)}(z) &\simeq \varphi^3_e(x, y, z)\\ &\simeq \varphi_x(y).\end{aligned}$$

In terms of the informal argument above, $s(e, x, y)$ is an index for the machine that, for any input z, ignores that input and computes $\varphi_x(y)$.

In particular, we have

$$\varphi_{s(e,x,y)}(0)\downarrow \quad \text{if and only if} \quad \varphi_x(y)\downarrow .$$

In other words, $\langle x, y\rangle \in K_0$ if and only if $s(e, x, y) \in K_1$. So the function g defined by

$$g(w) = s(e, (w)_0, (w)_1)$$

is a reduction of K_0 to K_1. □

§14.17 Totality is Undecidable

Let us consider one more example of using the *s-m-n* theorem to show that something is noncomputable. Let Tot be the set of indices of total computable functions, i.e.

$$\text{Tot} = \{x : \text{for every } y,\ \varphi_x(y)\downarrow\}.$$

Proposition 1417A. Tot *is not computable.*

Proof. To see that Tot is not computable, it suffices to show that K is reducible to it. Let $h(x, y)$ be defined by

$$h(x,y) \simeq \begin{cases} 0 & \text{if } x \in K \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that $h(x, y)$ does not depend on y at all. It should not be hard to see that h is partial computable: on input x, y, the we compute h by first simulating the function φ_x on input x; if this computation halts, $h(x, y)$ outputs 0 and halts. So $h(x, y)$ is just $Z(\mu s\, T(x, x, s))$, where Z is the constant zero function.

Using the *s-m-n* theorem, there is a primitive recursive function $k(x)$ such that for every x and y,

$$\varphi_{k(x)}(y) = \begin{cases} 0 & \text{if } x \in K \\ \text{undefined} & \text{otherwise} \end{cases}$$

So $\varphi_{k(x)}$ is total if $x \in K$, and undefined otherwise. Thus, k is a reduction of K to Tot. □

It turns out that Tot is not even computably enumerable—its complexity lies further up on the "arithmetic hierarchy." But we will not worry about this strengthening here.

§14.18 Rice's Theorem

If you think about it, you will see that the specifics of Tot do not play into the proof of Proposition 1417A. We designed $h(x, y)$ to act like the constant function $j(y) = 0$ exactly when x is in K; but we could just as well have made it act like any other partial computable function under those circumstances. This observation lets us state a more general theorem, which says, roughly, that no nontrivial property of computable functions is decidable.

Keep in mind that $\varphi_0, \varphi_1, \varphi_2, \ldots$ is our standard enumeration of the partial computable functions.

Theorem 1418A (Rice's Theorem). *Let C be any set of partial computable functions, and let $A = \{n : \varphi_n \in C\}$. If A is computable, then either C is $\emptyset$ or C is the set of all the partial computable functions.*

An *index set* is a set A with the property that if n and m are indices which "compute" the same function, then either both n and m are in A, or neither is. It is not hard to see that the set A in the theorem has this property. Conversely, if A is an index set and C is the set of functions computed by these indices, then $A = \{n : \varphi_n \in C\}$.

With this terminology, Rice's theorem is equivalent to saying that no nontrivial index set is decidable. To understand what the theorem says, it is helpful to emphasize the distinction between *programs* (say, in your favorite programming language) and the functions they compute. There are certainly questions

about programs (indices), which are syntactic objects, that are computable: does this program have more than 150 symbols? Does it have more than 22 lines? Does it have a "while" statement? Does the string "hello world" every appear in the argument to a "print" statement? Rice's theorem says that no nontrivial question about the program's *behavior* is computable. This includes questions like these: does the program halt on input 0? Does it ever halt? Does it ever output an even number?

Proof of Rice's theorem. Suppose C is neither $\emptyset$ nor the set of all the partial computable functions, and let A be the set of indices of functions in C. We will show that if A were computable, we could solve the halting problem; so A is not computable.

Without loss of generality, we can assume that the function f which is nowhere defined is not in C (otherwise, switch C and its complement in the argument below). Let g be any function in C. The idea is that if we could decide A, we could tell the difference between indices computing f, and indices computing g; and then we could use that capability to solve the halting problem.

Here's how. Using the universal computation predicate, we can define a function

$$h(x,y) \simeq \begin{cases} \text{undefined} & \text{if } \varphi_x(x) \uparrow \\ g(y) & \text{otherwise.} \end{cases}$$

To compute h, first we try to compute $\varphi_x(x)$; if that computation halts, we go on to compute $g(y)$; and if *that* computation halts, we return the output. More formally, we can write

$$h(x,y) \simeq P^2_0(g(y), \mathrm{Un}(x,x)).$$

where $P^2_0(z_0, z_1) = z_0$ is the 2-place projection function returning the 0-th argument, which is computable.

Then h is a composition of partial computable functions, and the right side is defined and equal to $g(y)$ just when $\mathrm{Un}(x,x)$ and $g(y)$ are both defined.

Notice that for a fixed x, if $\varphi_x(x)$ is undefined, then $h(x,y)$ is undefined for every y; and if $\varphi_x(x)$ is defined, then $h(x,y) \simeq g(y)$. So, for any fixed value of x, either $h(x,y)$ acts just like f or it acts just like g, and deciding whether or not $\varphi_x(x)$ is defined amounts to deciding which of these two cases holds. But this amounts to deciding whether or not $h_x(y) \simeq h(x,y)$ is in C or not, and if A were computable, we could do just that.

More formally, since h is partial computable, it is equal to the function φ_k for some index k. By the *s-m-n* theorem there is a primitive recursive function s such that for each x, $\varphi_{s(k,x)}(y) = h_x(y)$. Now we have that for each x, if $\varphi_x(x) \downarrow$, then $\varphi_{s(k,x)}$ is the same function as g, and so $s(k,x)$ is in A. On the other hand, if $\varphi_x(x) \uparrow$, then $\varphi_{s(k,x)}$ is the same function as f, and so $s(k,x)$ is not in A. In other words we have that for every x, $x \in K$ if and only if $s(k,x) \in A$. If A were computable, K would be also, which is a contradiction. So A is not computable. □

Rice's theorem is very powerful. The following immediate corollary shows some sample applications.

Corollary 1418B. *The following sets are undecidable.*

1. $\{x : 17 \text{ is in the range of } \varphi_x\}$
2. $\{x : \varphi_x \text{ is constant}\}$
3. $\{x : \varphi_x \text{ is total}\}$
4. $\{x : \text{whenever } y < y', \varphi_x(y)\downarrow, \text{ and if } \varphi_x(y')\downarrow, \text{ then } \varphi_x(y) < \varphi_x(y')\}$

Proof. These are all nontrivial index sets. □

§14.19 The Fixed-Point Theorem

Let's consider the halting problem again. As temporary notation, let us write $\ulcorner \varphi_x(y) \urcorner$ for $\langle x, y \rangle$; think of this as representing a "name" for the value $\varphi_x(y)$. With this notation, we can reword one of our proofs that the halting problem is undecidable.

Question: is there a computable function h, with the following property? For every x and y,

$$h(\ulcorner \varphi_x(y) \urcorner) = \begin{cases} 1 & \text{if } \varphi_x(y)\downarrow \\ 0 & \text{otherwise.} \end{cases}$$

Answer: No; otherwise, the partial function

$$g(x) \simeq \begin{cases} 0 & \text{if } h(\ulcorner \varphi_x(x) \urcorner) = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

would be computable, and so have some index e. But then we have

$$\varphi_e(e) \simeq \begin{cases} 0 & \text{if } h(\ulcorner \varphi_e(e) \urcorner) = 0 \\ \text{undefined} & \text{otherwise,} \end{cases}$$

in which case $\varphi_e(e)$ is defined if and only if it isn't, a contradiction.

Now, take a look at the equation with φ_e. There is an instance of self-reference there, in a sense: we have arranged for the value of $\varphi_e(e)$ to depend on $\ulcorner \varphi_e(e) \urcorner$, in a certain way. The fixed-point theorem says that we *can* do this, in general—not just for the sake of proving contradictions.

Lemma 1419A gives two equivalent ways of stating the fixed-point theorem. Logically speaking, the fact that the statements are equivalent follows from the fact that they are both true; but what we really mean is that each one follows straightforwardly from the other, so that they can be taken as alternative statements of the same theorem.

Lemma 1419A. *The following statements are equivalent:*

1. *For every partial computable function $g(x, y)$, there is an index e such that for every y,*
$$\varphi_e(y) \simeq g(e, y).$$
2. *For every computable function $f(x)$, there is an index e such that for every y,*
$$\varphi_e(y) \simeq \varphi_{f(e)}(y).$$

Proof. (1) $\Rightarrow$ (2): Given f, define g by $g(x, y) \simeq \mathrm{Un}(f(x), y)$. Use (1) to get an index e such that for every y,
$$\begin{aligned}\varphi_e(y) &= \mathrm{Un}(f(e), y)\\ &= \varphi_{f(e)}(y).\end{aligned}$$

(2) $\Rightarrow$ (1): Given g, use the *s-m-n* theorem to get f such that for every x and y, $\varphi_{f(x)}(y) \simeq g(x, y)$. Use (2) to get an index e such that
$$\begin{aligned}\varphi_e(y) &= \varphi_{f(e)}(y)\\ &= g(e, y).\end{aligned}$$

This concludes the proof. □

Before showing that statement (1) is true (and hence (2) as well), consider how bizarre it is. Think of e as being a computer program; statement (1) says that given any partial computable $g(x, y)$, you can find a computer program e that computes $g_e(y) \simeq g(e, y)$. In other words, you can find a computer program that computes a function that references the program itself.

Theorem 1419B. *The two statements in Lemma 1419A are true. Specifically, for every partial computable function $g(x, y)$, there is an index e such that for every y,*
$$\varphi_e(y) \simeq g(e, y).$$

Proof. The ingredients are already implicit in the discussion of the halting problem above. Let $\mathrm{diag}(x)$ be a computable function which for each x returns an index for the function $f_x(y) \simeq \varphi_x(x, y)$, i.e.
$$\varphi_{\mathrm{diag}(x)}(y) \simeq \varphi_x(x, y).$$

Think of diag as a function that transforms a program for a 2-ary function into a program for a 1-ary function, obtained by fixing the original program as its first argument. The function diag can be defined formally as follows: first define s by
$$s(x, y) \simeq \mathrm{Un}^2(x, x, y),$$
where Un^2 is a 3-ary function that is universal for partial computable 2-ary functions. Then, by the *s-m-n* theorem, we can find a primitive recursive function diag satisfying
$$\varphi_{\mathrm{diag}(x)}(y) \simeq s(x, y).$$

Now, define the function l by

$$l(x, y) \simeq g(\text{diag}(x), y).$$

and let $\ulcorner l \urcorner$ be an index for l. Finally, let $e = \text{diag}(\ulcorner l \urcorner)$. Then for every y, we have

$$\begin{aligned}
\varphi_e(y) &\simeq \varphi_{\text{diag}(\ulcorner l \urcorner)}(y) \\
&\simeq \varphi_{\ulcorner l \urcorner}(\ulcorner l \urcorner, y) \\
&\simeq l(\ulcorner l \urcorner, y) \\
&\simeq g(\text{diag}(\ulcorner l \urcorner), y) \\
&\simeq g(e, y),
\end{aligned}$$

as required. □

What's going on? Suppose you are given the task of writing a computer program that prints itself out. Suppose further, however, that you are working with a programming language with a rich and bizarre library of string functions. In particular, suppose your programming language has a function diag which works as follows: given an input string s, diag locates each instance of the symbol 'x' occurring in s, and replaces it by a quoted version of the original string. For example, given the string

```
hello x world
```

as input, the function returns

```
hello 'hello x world' world
```

as output. In that case, it is easy to write the desired program; you can check that

```
print(diag('print(diag(x))'))
```

does the trick. For more common programming languages like C++ and Java, the same idea (with a more involved implementation) still works.

We are only a couple of steps away from the proof of the fixed-point theorem. Suppose a variant of the print function print(x, y) accepts a string x and another numeric argument y, and prints the string x repeatedly, y times. Then the "program"

```
getinput(y); print(diag('getinput(y); print(diag(x), y)'), y)
```

prints itself out y times, on input y. Replacing the getinput—print—diag skeleton by an arbitrary function $g(x, y)$ yields

```
g(diag('g(diag(x), y)'), y)
```

which is a program that, on input y, runs g on the program itself and y. Thinking of "quoting" with "using an index for," we have the proof above.

For now, it is o.k. if you want to think of the proof as formal trickery, or black magic. But you should be able to reconstruct the details of the argument given above. When we prove the incompleteness theorems (and the related "fixed-point theorem") we will discuss other ways of understanding why it works.

The same idea can be used to get a "fixed point" combinator. Suppose you have a lambda term g, and you want another term k with the property that k is β-equivalent to gk. Define terms

$$\mathrm{diag}(x) = xx$$

and

$$l(x) = g(\mathrm{diag}(x))$$

using our notational conventions; in other words, l is the term $\lambda x.\, g(xx)$. Let k be the term ll. Then we have

$$\begin{aligned} k &= (\lambda x.\, g(xx))(\lambda x.\, g(xx)) \\ &\twoheadrightarrow g((\lambda x.\, g(xx))(\lambda x.\, g(xx))) \\ &= gk. \end{aligned}$$

If one takes

$$Y = \lambda g.\, ((\lambda x.\, g(xx))(\lambda x.\, g(xx)))$$

then Yg and $g(Yg)$ reduce to a common term; so $Yg \equiv_\beta g(Yg)$. This is known as "Curry's combinator." If instead one takes

$$Y = (\lambda xg.\, g(xxg))(\lambda xg.\, g(xxg))$$

then in fact Yg reduces to $g(Yg)$, which is a stronger statement. This latter version of Y is known as "Turing's combinator."

§14.20 Applying the Fixed-Point Theorem

The fixed-point theorem essentially lets us define partial computable functions in terms of their indices. For example, we can find an index e such that for every y,

$$\varphi_e(y) = e + y.$$

As another example, one can use the proof of the fixed-point theorem to design a program in Java or C++ that prints itself out.

Remember that if for each e, we let W_e be the domain of φ_e, then the sequence W_0, W_1, W_2, ... enumerates the computably enumerable sets. Some of these sets are computable. One can ask if there is an algorithm which takes as input a value x, and, if W_x happens to be computable, returns an index for its characteristic function. The answer is "no," there is no such algorithm:

Theorem 1420A. *There is no partial computable function f with the following property: whenever W_e is computable, then $f(e)$ is defined and $\varphi_{f(e)}$ is its characteristic function.*

Proof. Let f be any computable function; we will construct an e such that W_e is computable, but $\varphi_{f(e)}$ is not its characteristic function. Using the fixed point theorem, we can find an index e such that

$$\varphi_e(y) \simeq \begin{cases} 0 & \text{if } y = 0 \text{ and } \varphi_{f(e)}(0) \downarrow = 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

That is, e is obtained by applying the fixed-point theorem to the function defined by

$$g(x, y) \simeq \begin{cases} 0 & \text{if } y = 0 \text{ and } \varphi_{f(x)}(0) \downarrow = 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Informally, we can see that g is partial computable, as follows: on input x and y, the algorithm first checks to see if y is equal to 0. If it is, the algorithm computes $f(x)$, and then uses the universal machine to compute $\varphi_{f(x)}(0)$. If this last computation halts and returns 0, the algorithm returns 0; otherwise, the algorithm doesn't halt.

But now notice that if $\varphi_{f(e)}(0)$ is defined and equal to 0, then $\varphi_e(y)$ is defined exactly when y is equal to 0, so $W_e = \{0\}$. If $\varphi_{f(e)}(0)$ is not defined, or is defined but not equal to 0, then $W_e = \emptyset$. Either way, $\varphi_{f(e)}$ is not the characteristic function of W_e, since it gives the wrong answer on input 0. □

§14.21 Defining Functions using Self-Reference

It is generally useful to be able to define functions in terms of themselves. For example, given computable functions k, l, and m, the fixed-point lemma tells us that there is a partial computable function f satisfying the following equation for every y:

$$f(y) \simeq \begin{cases} k(y) & \text{if } l(y) = 0 \\ f(m(y)) & \text{otherwise.} \end{cases}$$

Again, more specifically, f is obtained by letting

$$g(x, y) \simeq \begin{cases} k(y) & \text{if } l(y) = 0 \\ \varphi_x(m(y)) & \text{otherwise} \end{cases}$$

and then using the fixed-point lemma to find an index e such that $\varphi_e(y) = g(e, y)$.

For a concrete example, the "greatest common divisor" function $\gcd(u, v)$ can be defined by

$$\gcd(u, v) \simeq \begin{cases} v & \text{if } 0 = 0 \\ \gcd(\text{mod}(v, u), u) & \text{otherwise} \end{cases}$$

where $\mathrm{mod}(v, u)$ denotes the remainder of dividing v by u. An appeal to the fixed-point lemma shows that gcd is partial computable. (In fact, this can be put in the format above, letting y code the pair $\langle u, v\rangle$.) A subsequent induction on u then shows that, in fact, gcd is total.

Of course, one can cook up self-referential definitions that are much fancier than the examples just discussed. Most programming languages support definitions of functions in terms of themselves, one way or another. Note that this is a little bit less dramatic than being able to define a function in terms of an *index* for an algorithm computing the functions, which is what, in full generality, the fixed-point theorem lets you do.

§14.22 Minimization with Lambda Terms

When it comes to the lambda calculus, we've shown the following:

1. Every primitive recursive function is represented by a lambda term.
2. There is a lambda term Y such that for any lambda term G, $YG \twoheadrightarrow G(YG)$.

To show that every partial computable function is represented by some lambda term, we only need to show the following.

Lemma 1422A. *Suppose $f(x, y)$ is primitive recursive. Let g be defined by*

$$g(x) \simeq \mu y\ f(x, y) = 0.$$

Then g is represented by a lambda term.

Proof. The idea is roughly as follows. Given x, we will use the fixed-point lambda term Y to define a function $h_x(n)$ which searches for a y starting at n; then $g(x)$ is just $h_x(0)$. The function h_x can be expressed as the solution of a fixed-point equation:

$$h_x(n) \simeq \begin{cases} n & \text{if } f(x, n) = 0 \\ h_x(n+1) & \text{otherwise.} \end{cases}$$

Here are the details. Since f is primitive recursive, it is represented by some term F. Remember that we also have a lambda term D such that $D(M, N, \overline{0}) \twoheadrightarrow M$ and $D(M, N, \overline{1}) \twoheadrightarrow N$. Fixing x for the moment, to represent h_x we want to find a term H (depending on x) satisfying

$$H(\overline{n}) \equiv D(\overline{n}, H(S(\overline{n})), F(x, \overline{n})).$$

We can do this using the fixed-point term Y. First, let U be the term

$$\lambda h.\, \lambda z.\, D(z, (h(Sz)), F(x, z)),$$

and then let H be the term YU. Notice that the only free variable in H is x. Let us show that H satisfies the equation above.

By the definition of Y, we have

$$H = YU \equiv U(YU) = U(H).$$

In particular, for each natural number n, we have

$$\begin{aligned} H(\overline{n}) &\equiv U(H, \overline{n}) \\ &\twoheadrightarrow D(\overline{n}, H(S(\overline{n})), F(x, \overline{n})), \end{aligned}$$

as required. Notice that if you substitute a numeral $\overline{m}$ for x in the last line, the expression reduces to $\overline{n}$ if $F(\overline{m}, \overline{n})$ reduces to $\overline{0}$, and it reduces to $H(S(\overline{n}))$ if $F(\overline{m}, \overline{n})$ reduces to any other numeral.

To finish off the proof, let G be $\lambda x.\, H(\overline{0})$. Then G represents g; in other words, for every m, $G(\overline{m})$ reduces to reduces to $\overline{g(m)}$, if $g(m)$ is defined, and has no normal form otherwise. □

Problems

Problem 1. Give a reduction of K to K_0.

Part IV

Incompleteness

Chapter 15

Introduction to Incompleteness

§15.0 Historical Background

In this section, we will briefly discuss historical developments that will help put the incompleteness theorems in context. In particular, we will give a very sketchy overview of the history of mathematical logic; and then say a few words about the history of the foundations of mathematics.

The phrase "mathematical logic" is ambiguous. One can interpret the word "mathematical" as describing the subject matter, as in, "the logic of mathematics," denoting the principles of mathematical reasoning; or as describing the methods, as in "the mathematics of logic," denoting a mathematical study of the principles of reasoning. The account that follows involves mathematical logic in both senses, often at the same time.

The study of logic began, essentially, with Aristotle, who lived approximately 384–322 BCE. His *Categories*, *Prior analytics*, and *Posterior analytics* include systematic studies of the principles of scientific reasoning, including a thorough and systematic study of the syllogism.

Aristotle's logic dominated scholastic philosophy through the middle ages; indeed, as late as the eighteenth century, Kant maintained that Aristotle's logic was perfect and in no need of revision. But the theory of the syllogism is far too limited to model anything but the most superficial aspects of mathematical reasoning. A century earlier, Leibniz, a contemporary of Newton's, imagined a complete "calculus" for logical reasoning, and made some rudimentary steps towards designing such a calculus, essentially describing a version of propositional logic.

The nineteenth century was a watershed for logic. In 1854 George Boole wrote *The Laws of Thought*, with a thorough algebraic study of propositional logic that is not far from modern presentations. In 1879 Gottlob Frege published his *Begriffsschrift* (Concept writing) which extends propositional logic with quantifiers and relations, and thus includes first-order logic. In fact, Frege's logical systems included higher-order logic as well, and more. In his *Basic Laws of Arithmetic*, Frege set out to show that all of arithmetic could

be derived in his Begriffsschrift from purely logical assumption. Unfortunately, these assumptions turned out to be inconsistent, as Russell showed in 1902. But setting aside the inconsistent axiom, Frege more or less invented modern logic singlehandedly, a startling achievement. Quantificational logic was also developed independently by algebraically-minded thinkers after Boole, including Peirce and Schröder.

Let us now turn to developments in the foundations of mathematics. Of course, since logic plays an important role in mathematics, there is a good deal of interaction with the developments just described. For example, Frege developed his logic with the explicit purpose of showing that all of mathematics could be based solely on his logical framework; in particular, he wished to show that mathematics consists of a priori *analytic* truths instead of, as Kant had maintained, a priori *synthetic* ones.

Many take the birth of mathematics proper to have occurred with the Greeks. Euclid's *Elements*, written around 300 B.C., is already a mature representative of Greek mathematics, with its emphasis on rigor and precision. The definitions and proofs in Euclid's *Elements* survive more or less intact in high school geometry textbooks today (to the extent that geometry is still taught in high schools). This model of mathematical reasoning has been held to be a paradigm for rigorous argumentation not only in mathematics but in branches of philosophy as well. (Spinoza even presented moral and religious arguments in the Euclidean style, which is strange to see!)

Calculus was invented by Newton and Leibniz in the seventeenth century. (A fierce priority dispute raged for centuries, but most scholars today hold that the two developments were for the most part independent.) Calculus involves reasoning about, for example, infinite sums of infinitely small quantities; these features fueled criticism by Bishop Berkeley, who argued that belief in God was no less rational than the mathematics of his time. The methods of calculus were widely used in the eighteenth century, for example by Leonhard Euler, who used calculations involving infinite sums with dramatic results.

In the nineteenth century, mathematicians tried to address Berkeley's criticisms by putting calculus on a firmer foundation. Efforts by Cauchy, Weierstrass, Bolzano, and others led to our contemporary definitions of limits, continuity, differentiation, and integration in terms of "epsilons and deltas," in other words, devoid of any reference to infinitesimals. Later in the century, mathematicians tried to push further, and explain all aspects of calculus, including the real numbers themselves, in terms of the natural numbers. (Kronecker: "God created the whole numbers, all else is the work of man.") In 1872, Dedekind wrote "Continuity and the irrational numbers," where he showed how to "construct" the real numbers as sets of rational numbers (which, as you know, can be viewed as pairs of natural numbers); in 1888 he wrote "Was sind und was sollen die Zahlen" (roughly, "What are the natural numbers, and what should they be?") which aimed to explain the natural numbers in purely "logical" terms. In 1887 Kronecker wrote "Über den Zahlbegriff" ("On the concept of number") where he spoke of representing all mathematical object in terms of the integers; in 1889 Giuseppe Peano gave formal, symbolic axioms

for the natural numbers.

The end of the nineteenth century also brought a new boldness in dealing with the infinite. Before then, infinitary objects and structures (like the set of natural numbers) were treated gingerly; "infinitely many" was understood as "as many as you want," and "approaches in the limit" was understood as "gets as close as you want." But Georg Cantor showed that it was possible to take the infinite at face value. Work by Cantor, Dedekind, and others help to introduce the general set-theoretic understanding of mathematics that is now widely accepted.

This brings us to twentieth century developments in logic and foundations. In 1902 Russell discovered the paradox in Frege's logical system. In 1904 Zermelo proved Cantor's well-ordering principle, using the so-called "axiom of choice"; the legitimacy of this axiom prompted a good deal of debate. Between 1910 and 1913 the three volumes of Russell and Whitehead's *Principia Mathematica* appeared, extending the Fregean program of establishing mathematics on logical grounds. Unfortunately, Russell and Whitehead were forced to adopt two principles that seemed hard to justify as purely logical: an axiom of infinity and an axiom of "reducibility." In the 1900's Poincaré criticized the use of "impredicative definitions" in mathematics, and in the 1910's Brouwer began proposing to refound all of mathematics in an "intuitionistic" basis, which avoided the use of the law of the excluded middle ($\alpha \vee \neg\alpha$).

Strange days indeed! The program of reducing all of mathematics to logic is now referred to as "logicism," and is commonly viewed as having failed, due to the difficulties mentioned above. The program of developing mathematics in terms of intuitionistic mental constructions is called "intuitionism," and is viewed as posing overly severe restrictions on everyday mathematics. Around the turn of the century, David Hilbert, one of the most influential mathematicians of all time, was a strong supporter of the new, abstract methods introduced by Cantor and Dedekind: "no one will drive us from the paradise that Cantor has created for us." At the same time, he was sensitive to foundational criticisms of these new methods (oddly enough, now called "classical"). He proposed a way of having one's cake and eating it too:

1. Represent classical methods with formal axioms and rules; represent mathematical questions as wffs in an axiomatic system.

2. Use safe, "finitary" methods to prove that these formal deductive systems are consistent.

Hilbert's work went a long way toward accomplishing the first goal. In 1899, he had done this for geometry in his celebrated book *Foundations of geometry*. In subsequent years, he and a number of his students and collaborators worked on other areas of mathematics to do what Hilbert had done for geometry. Hilbert himself gave axiom systems for arithmetic and analysis. Zermelo gave an axiomatization of set theory, which was expanded on by Fraenkel, Skolem, von Neumann, and others. By the mid-1920s, there were two approaches that

laid claim to the title of an axiomatization of "all" of mathematics, the *Principia mathematica* of Russell and Whitehead, and what came to be known as Zermelo-Fraenkel set theory.

In 1921, Hilbert set out on a research project to establish the goal of proving these systems to be consistent. He was aided in this project by several of his students, in particular Bernays, Ackermann, and later Gentzen. The basic idea for accomplishing this goal was to cast the question of the possibility of a derivation of an inconsistency in mathematics as a combinatorial problem about possible sequences of symbols, namely possible sequences of sentences which meet the criterion of being a correct derivation of, say, $\alpha \wedge \neg\alpha$ from the axioms of an axiom system for arithmetic, analysis, or set theory. A proof of the impossibility of such a sequence of symbols would—since it is itself a mathematical proof—be formalizable in these axiomatic systems. In other words, there would be some sentence Con which states that, say, arithmetic is consistent. Moreover, this sentence should be provable in the systems in question, especially if its proof requires only very restricted, "finitary" means.

The second aim, that the axiom systems developed would settle every mathematical question, can be made precise in two ways. In one way, we can formulate it as follows: For any sentence α in the language of an axiom system for mathematics, either α or $\neg\alpha$ is provable from the axioms. If this were true, then there would be no sentences which can neither be proved nor refuted on the basis of the axioms, no questions which the axioms do not settle. An axiom system with this property is called *complete*. Of course, for any given sentence it might still be a difficult task to determine which of the two alternatives holds. But in principle there should be a method to do so. In fact, for the axiom and derivation systems considered by Hilbert, completeness would imply that such a method exists—although Hilbert did not realize this. The second way to interpret the question would be this stronger requirement: that there be a mechanical, computational method which would determine, for a given sentence α, whether it is derivable from the axioms or not.

In 1931, Gödel proved the two "incompleteness theorems," which showed that this program could not succeed. There is no axiom system for mathematics which is complete, specifically, the sentence that expresses the consistency of the axioms is a sentence which can neither be proved nor refuted.

This struck a lethal blow to Hilbert's original program. However, as is so often the case in mathematics, it also opened up exciting new avenues for research. If there is no one, all-encompassing formal system of mathematics, it makes sense to develop more circumscribed systems and investigate what can be proved in them. It also makes sense to develop less restricted methods of proof for establishing the consistency of these systems, and to find ways to measure how hard it is to prove their consistency. Since Gödel showed that (almost) every formal system has questions it cannot settle, it makes sense to look for "interesting" questions a given formal system cannot settle, and to figure out how strong a formal system has to be to settle them. To the present day, logicians have been pursuing these questions in a new mathematical discipline, the theory of proofs.

§15.1 Definitions

In order to carry out Hilbert's project of formalizing mathematics and showing that such a formalization is consistent and complete, the first order of business would be that of picking a language, logical framework, and a system of axioms. For our purposes, let us suppose that mathematics can be formalized in a first-order language, i.e., that there is some set of constant symbols, function symbols, and predicate symbols which, together with the connectives and quantifiers of first-order logic, allow us to express the claims of mathematics. Most people agree that such a language exists: the language of set theory, in which $\in$ is the only non-logical symbol. That such a simple language is so expressive is of course a very implausible claim at first sight, and it took a lot of work to establish that practically of all mathematics can be expressed in this very austere vocabulary. To keep things simple, for now, let's restrict our discussion to arithmetic, so the part of mathematics that just deals with the natural numbers $\mathbb{N}$. The natural language in which to express facts of arithmetic is $\mathcal{L}_A$. $\mathcal{L}_A$ contains a single two-place predicate symbol $<$, a single constant symbol 0, one one-place function symbol $'$, and two two-place function symbols $+$ and $\times$.

Definition 151A. *A set of sentences Γ is a* theory *if it is closed under entailment, i.e., if $\Gamma = \{\alpha : \Gamma \vDash \alpha\}$.*

There are two easy ways to specify theories. One is as the set of sentences true in some structure. For instance, consider the structure for $\mathcal{L}_A$ in which the domain is $\mathbb{N}$ and all non-logical symbols are interpreted as you would expect.

Definition 151B. *The* standard model of arithmetic *is the structure $\mathfrak{B}$ defined as follows:*

1. $|\mathfrak{B}| = \mathbb{N}$
2. $0^{\mathfrak{B}} = 0$
3. $'^{\mathfrak{B}}(n) = n + 1$ *for all* $n \in \mathbb{N}$
4. $+^{\mathfrak{B}}(n, m) = n + m$ *for all* $n, m \in \mathbb{N}$
5. $\times^{\mathfrak{B}}(n, m) = n \cdot m$ *for all* $n, m \in \mathbb{N}$
6. $<^{\mathfrak{B}} = \{\langle n, m\rangle : n \in \mathbb{N}, m \in \mathbb{N}, n < m\}$

Note the difference between $\times$ and $\cdot$: $\times$ is a symbol in the language of arithmetic. Of course, we've chosen it to remind us of multiplication, but $\times$ is not the multiplication operation but a two-place function symbol (officially, f^2_1). By contrast, $\cdot$ *is* the ordinary multiplication function. When you see something like $n \cdot m$, we mean the product of the numbers n and m; when you see something like $x \times y$ we are talking about a term in the language of arithmetic. In the standard model, the function symbol times is interpreted

as the function $\cdot$ on the natural numbers. For addition, we use $+$ as both the function symbol of the language of arithmetic, and the addition function on the natural numbers. Here you have to use the context to determine what is meant.

Definition 151C. *The theory of* true arithmetic *is the set of sentences satisfied in the standard model of arithmetic, i.e.,*

$$\mathbf{TA} = \{\alpha : \models_{\mathfrak{B}} \alpha\}.$$

TA is a theory, for whenever $\mathbf{TA} \vDash \alpha$, α is satisfied in every structure which satisfies **TA**. Since $\models_{\mathfrak{A}} \mathbf{TA}$, $\models_{\mathfrak{A}} \alpha$, and so $\alpha \in \mathbf{TA}$.

The other way to specify a theory Γ is as the set of sentences entailed by some set of sentences Γ_0. In that case, Γ is the "closure" of Γ_0 under entailment. Specifying a theory this way is only interesting if Γ_0 is explicitly specified, e.g., if the elements of Γ_0 are listed. At the very least, Γ_0 has to be decidable, i.e., there has to be a computable test for when a sentence counts as an element of Γ_0 or not. We call the sentences in Γ_0 *axioms* for Γ, and Γ *axiomatized* by Γ_0.

Definition 151D. *A theory Γ is* axiomatized *by Γ_0 iff*

$$\Gamma = \{\alpha : \Gamma_0 \vDash \alpha\}$$

Definition 151E. *The theory* **Q** *axiomatized by the following sentences is known as "Robinson's* **Q***" and is a very simple theory of arithmetic.*

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

The set of sentences $\{Q_1, \ldots, Q_8\}$ are the axioms of **Q***, so* **Q** *consists of all sentences entailed by them:*

$$\mathbf{Q} = \{\alpha : \{Q_1, \ldots, Q_8\} \vDash \alpha\}.$$

Definition 151F. *Suppose $\alpha(x)$ is a wff in $\mathcal{L}_A$ with free variables x and y_1, …, y_n. Then any sentence of the form*

$$\forall y_1 \ldots \forall y_n \, ((\alpha(0) \wedge \forall x \, (\alpha(x) \to \alpha(x'))) \to \forall x \, \alpha(x))$$

is an instance of the induction schema.

Peano arithmetic **PA** *is the theory axiomatized by the axioms of* **Q** *together with all instances of the induction schema.*

Every instance of the induction schema is true in $\mathfrak{B}$. This is easiest to see if the wff α only has one free variable x. Then $\alpha(x)$ defines a subset X_α of $\mathbb{N}$ in $\mathfrak{B}$. X_α is the set of all $n \in \mathbb{N}$ such that $\models_{\mathfrak{B}} \alpha(x)\,[s]$ when $s(x) = n$. The corresponding instance of the induction schema is

$$((\alpha(0) \wedge \forall x\,(\alpha(x) \to \alpha(x'))) \to \forall x\,\alpha(x)).$$

If its antecedent is true in $\mathfrak{B}$, then $0 \in X_\alpha$ and, whenever $n \in X_\alpha$, so is $n+1$. Since $0 \in X_\alpha$, we get $1 \in X_\alpha$. With $1 \in X_\alpha$ we get $2 \in X_\alpha$. And so on. So for every $n \in \mathbb{N}$, $n \in X_\alpha$. But this means that $\forall x\,\alpha(x)$ is satisfied in $\mathfrak{B}$.

Both **Q** and **PA** are axiomatized theories. The big question is, how strong are they? For instance, can **PA** prove all the truths about $\mathbb{N}$ that can be expressed in $\mathcal{L}_A$? Specifically, do the axioms of **PA** settle all the questions that can be formulated in $\mathcal{L}_A$?

Another way to put this is to ask: Is **PA** = **TA**? **TA** obviously does prove (i.e., it includes) all the truths about $\mathbb{N}$, and it settles all the questions that can be formulated in $\mathcal{L}_A$, since if α is a sentence in $\mathcal{L}_A$, then either $\models_{\mathfrak{B}} \alpha$ or $\models_{\mathfrak{B}} \neg\alpha$, and so either **TA** $\vDash \alpha$ or **TA** $\vDash \neg\alpha$. Call such a theory *complete*.

Definition 151G. *A theory Γ is* complete *iff for every sentence α in its language, either $\Gamma \vDash \alpha$ or $\Gamma \vDash \neg\alpha$.*

By the Completeness Theorem, $\Gamma \vDash \alpha$ iff $\Gamma \vdash \alpha$, so Γ is complete iff for every sentence α in its language, either $\Gamma \vdash \alpha$ or $\Gamma \vdash \neg\alpha$.

Another question we are led to ask is this: Is there a computational procedure we can use to test if a sentence is in **TA**, in **PA**, or even just in **Q**? We can make this more precise by defining when a set (e.g., a set of sentences) is decidable.

Definition 151H. *A set X is* decidable *iff there is a computational procedure which on input x returns 1 if $x \in X$ and 0 otherwise.*

So our question becomes: Is **TA** (**PA**, **Q**) decidable?

The answer to all these questions will be: no. None of these theories are decidable. However, this phenomenon is not specific to these particular theories. In fact, *any* theory that satisfies certain conditions is subject to the same results. One of these conditions, which **Q** and **PA** satisfy, is that they are axiomatized by a decidable set of axioms.

Definition 151I. *A theory is* axiomatizable *if it is axiomatized by a decidable set of axioms.*

Example 15.1.10. Any theory axiomatized by a finite set of sentences is axiomatizable, since any finite set is decidable. Thus, **Q**, for instance, is axiomatizable.

Schematically axiomatized theories like **PA** are also axiomatizable. For to test if β is among the axioms of **PA**, i.e., to compute the function χ_X where $\chi_X(\beta) = 1$ if β is an axiom of **PA** and $= 0$ otherwise, we can do the following:

First, check if β is one of the axioms of **Q**. If it is, the answer is "yes" and the value of $\chi_X(\beta) = 1$. If not, test if it is an instance of the induction schema. This can be done systematically; in this case, perhaps it's easiest to see that it can be done as follows: Any instance of the induction schema begins with a number of universal quantifiers, and then a sub-wff that is a conditional. The consequent of that conditional is $\forall x\, \alpha(x, y_1, \ldots, y_n)$ where x and $y_1, \ldots, y_n$ are all the free variables of α and the initial quantifiers of β bind the variables $y_1, \ldots, y_n$. Once we have extracted this α and checked that its free variables match the variables bound by the universal quantifiers at the front and $\forall x$, we go on to check that the antecedent of the conditional matches

$$\alpha(0, y_1, \ldots, y_n) \wedge \forall x\, (\alpha(x, y_1, \ldots, y_n) \to \alpha(x', y_1, \ldots, y_n))$$

Again, if it does, β is an instance of the induction schema, and if it doesn't, β isn't.

In answering this question—and the more general question of which theories are complete or decidable—it will be useful to consider also the following definition. Recall that a set X is enumerable iff it is empty or if there is a surjective function $f\colon \mathbb{N} \to X$. Such a function is called an enumeration of X.

Definition 151K. *A set X is called* computably enumerable *(c.e. for short) iff it is empty or it has a computable enumeration.*

In addition to axiomatizability, another condition on theories to which the incompleteness theorems apply will be that they are strong enough to prove basic facts about computable functions and decidable relations. By "basic facts," we mean sentences which express what the values of computable functions are for each of their arguments. And by "strong enough" we mean that the theories in question count these sentences among its theorems. For instance, consider a prototypical computable function: addition. The value of $+$ for arguments 2 and 3 is 5, i.e., $2 + 3 = 5$. A sentence in the language of arithmetic that expresses that the value of $+$ for arguments 2 and 3 is 5 is: $(\overline{2} + \overline{3}) = \overline{5}$. And, e.g., **Q** proves this sentence. More generally, we would like there to be, for each computable function $f(x_1, x_2)$ a wff $\alpha_f(x_1, x_2, y)$ in $\mathcal{L}_A$ such that $\mathbf{Q} \vdash \alpha_f(\overline{n_1}, \overline{n_2}, \overline{m})$ whenever $f(n_1, n_2) = m$. In this way, **Q** proves that the value of f for arguments n_1, n_2 is m. In fact, we require that it proves a bit more, namely that no other number is the value of f for arguments n_1, n_2. And the same goes for decidable relations. This is made precise in the following two definitions.

Definition 151L. *A wff $\alpha(x_1, \ldots, x_k, y)$* represents *the function $f\colon \mathbb{N}^k \to \mathbb{N}$ in Γ iff whenever $f(n_1, \ldots, n_k) = m$, then*

1. *$\Gamma \vdash \alpha(\overline{n_1}, \ldots, \overline{n_k}, \overline{m})$, and*
2. *$\Gamma \vdash \forall y(\alpha(\overline{n_1}, \ldots, \overline{n_k}, y) \to y = \overline{m})$.*

Definition 151M. *A wff* $\alpha(x_1, \ldots, x_k)$ represents *the relation* $R \subseteq \mathbb{N}^k$ *iff,*

1. *whenever* $R(n_1, \ldots, n_k)$, $\Gamma \vdash \alpha(\overline{n_1}, \ldots, \overline{n_k})$, *and*
2. *whenever not* $R(n_1, \ldots, n_k)$, $\Gamma \vdash \neg\alpha(\overline{n_1}, \ldots, \overline{n_k})$.

A theory is "strong enough" for the incompleteness theorems to apply if it represents all computable functions and all decidable relations. **Q** and its extensions satisfy this condition, but it will take us a while to establish this—it's a non-trivial fact about the kinds of things **Q** can prove, and it's hard to show because **Q** has only a few axioms from which we'll have to prove all these facts. However, **Q** is a very weak theory. So although it's hard to prove that **Q** represents all computable functions, most interesting theories are stronger than **Q**, i.e., prove more than **Q** does. And if **Q** proves something, any stronger theory does; since **Q** represents all computable functions, every stronger theory does. This means that many interesting theories meet this condition of the incompleteness theorems. So our hard work will pay off, since it shows that the incompleteness theorems apply to a wide range of theories. Certainly, any theory aiming to formalize "all of mathematics" must prove everything that **Q** proves, since it should at the very least be able to capture the results of elementary computations. So any theory that is a candidate for a theory of "all of mathematics" will be one to which the incompleteness theorems apply.

§15.2 Overview of Incompleteness Results

Hilbert expected that mathematics could be formalized in an axiomatizable theory which it would be possible to prove complete and decidable. Moreover, he aimed to prove the consistency of this theory with very weak, "finitary," means, which would defend classical mathematics against the challenges of intuitionism. Gödel's incompleteness theorems showed that these goals cannot be achieved.

Gödel's first incompleteness theorem showed that a version of Russell and Whitehead's *Principia Mathematica* is not complete. But the proof was actually very general and applies to a wide variety of theories. This means that it wasn't just that *Principia Mathematica* did not manage to completely capture mathematics, but that *no* acceptable theory does. It took a while to isolate the features of theories that suffice for the incompleteness theorems to apply, and to generalize Gödel's proof to apply make it depend only on these features. But we are now in a position to state a very general version of the first incompleteness theorem for theories in the language $\mathcal{L}_A$ of arithmetic.

Theorem 152A. *If* Γ *is a consistent and axiomatizable theory in* $\mathcal{L}_A$ *which represents all computable functions and decidable relations, then* Γ *is not complete.*

To say that Γ is not complete is to say that for at least one sentence α, $\Gamma \nvdash \alpha$ and $\Gamma \nvdash \neg\alpha$. Such a sentence is called *independent* (of Γ). We can in

fact relatively quickly prove that there must be independent sentences. But the power of Gödel's proof of the theorem lies in the fact that it exhibits a *specific example* of such an independent sentence. The intriguing construction produces a sentence χ_Γ, called a *Gödel sentence* for Γ, which is unprovable because in Γ, χ_Γ is equivalent to the claim that χ_Γ is unprovable in Γ. It does so *constructively*, i.e., given an axiomatization of Γ and a description of the derivation system, the proof gives a method for actually writing down χ_Γ.

The construction in Gödel's proof requires that we find a way to express in $\mathcal{L}_A$ the properties of and operations on terms and wffs of $\mathcal{L}_A$ itself. These include properties such as "α is a sentence," "δ is a derivation of α," and operations such as $\alpha[t/x]$. This way must (a) express these properties and relations via a "coding" of symbols and sequences thereof (which is what terms, wffs, derivations, etc. are) as natural numbers (which is what $\mathcal{L}_A$ can talk about). It must (b) do this in such a way that Γ will prove the relevant facts, so we must show that these properties are coded by decidable properties of natural numbers and the operations correspond to computable functions on natural numbers. This is called "arithmetization of syntax."

Before we investigate how syntax can be arithmetized, however, we will consider the condition that Γ is "strong enough," i.e., represents all computable functions and decidable relations. This requires that we give a precise definition of "computable." This can be done in a number of ways, e.g., via the model of Turing machines, or as those functions computable by programs in some general-purpose programming language. Since our aim is to represent these functions and relations in a theory in the language $\mathcal{L}_A$, however, it is best to pick a simple definition of computability of just numerical functions. This is the notion of *recursive function*. So we will first discuss the recursive functions. We will then show that **Q** already represents all recursive functions and relations. This will allow us to apply the incompleteness theorem to specific theories such as **Q** and **PA**, since we will have established that these are examples of theories that are "strong enough."

The end result of the arithmetization of syntax is a wff $\mathsf{Prov}_\Gamma(x)$ which, via the coding of wffs as numbers, expresses provability from the axioms of Γ. Specifically, if α is coded by the number n, and $\Gamma \vdash \alpha$, then $\Gamma \vdash \mathrm{Prov}_\Gamma(\overline{n})$. This "provability predicate" for Γ allows us also to express, in a certain sense, the consistency of Γ as a sentence of $\mathcal{L}_A$: let the "consistency statement" for Γ be the sentence $\neg\mathrm{Prov}_\Gamma(\overline{n})$, where we take n to be the code of a contradiction, e.g., of $\bot$. The second incompleteness theorem states that consistent axiomatizable theories also do not prove their own consistency statements. The conditions required for this theorem to apply are a bit more stringent than just that the theory represents all computable functions and decidable relations, but we will show that **PA** satisfies them.

§15.3 Undecidability and Incompleteness

Gödel's proof of the incompleteness theorems require arithmetization of syntax. But even without that we can obtain some nice results just on the assumption that a theory represents all decidable relations. The proof is a diagonal argument similar to the proof of the undecidability of the halting problem.

Theorem 153A. *If Γ is a consistent theory that represents every decidable relation, then Γ is not decidable.*

Proof. Suppose Γ were decidable. We show that if Γ represents every decidable relation, it must be inconsistent.

Decidable properties (one-place relations) are represented by wffs with one free variable. Let $\alpha_0(x)$, $\alpha_1(x)$, ..., be a computable enumeration of all such wffs. Now consider the following set $D \subseteq \mathbb{N}$:

$$D = \{n : \Gamma \vdash \neg\alpha_n(\overline{n})\}$$

The set D is decidable, since we can test if $n \in D$ by first computing $\alpha_n(x)$, and from this $\neg\alpha_n(\overline{n})$. Obviously, substituting the term $\overline{n}$ for every free occurrence of x in $\alpha_n(x)$ and prefixing $\alpha(\overline{n})$ by $\neg$ is a mechanical matter. By assumption, Γ is decidable, so we can test if $\neg\alpha(\overline{n}) \in \Gamma$. If it is, $n \in D$, and if it isn't, $n \notin D$. So D is likewise decidable.

Since Γ represents all decidable properties, it represents D. And the wffs which represent D in Γ are all among $\alpha_0(x)$, $\alpha_1(x)$, ... So let d be a number such that $\alpha_d(x)$ represents D in Γ. If $d \notin D$, then, since $\alpha_d(x)$ represents D, $\Gamma \vdash \neg\alpha_d(\overline{d})$. But that means that d meets the defining condition of D, and so $d \in D$. This contradicts $d \notin D$. So by indirect proof, $d \in D$.

Since $d \in D$, by the definition of D, $\Gamma \vdash \neg\alpha_d(\overline{d})$. On the other hand, since $\alpha_d(x)$ represents D in Γ, $\Gamma \vdash \alpha_d(\overline{d})$. Hence, Γ is inconsistent. □

The preceding theorem shows that no consistent theory that represents all decidable relations can be decidable. We will show that **Q** does represent all decidable relations; this means that all theories that include **Q**, such as **PA** and **TA**, also do, and hence also are not decidable. (Since all these theories are true in the standard model, they are all consistent.)

We can also use this result to obtain a weak version of the first incompleteness theorem. Any theory that is axiomatizable and complete is decidable. Consistent theories that are axiomatizable and represent all decidable properties then cannot be complete.

Theorem 153B. *If Γ is axiomatizable and complete it is decidable.*

Proof. Any inconsistent theory is decidable, since inconsistent theories contain all sentences, so the answer to the question "is $\alpha \in \Gamma$" is always "yes," i.e., can be decided.

So suppose Γ is consistent, and furthermore is axiomatizable, and complete. Since Γ is axiomatizable, it is computably enumerable. For we can enumerate

all the correct derivations from the axioms of Γ by a computable function. From a correct derivation we can compute the sentence it derives, and so together there is a computable function that enumerates all theorems of Γ. A sentence is a theorem of Γ iff $\neg\alpha$ is not a theorem, since Γ is consistent and complete. We can therefore decide if $\alpha \in \Gamma$ as follows. Enumerate all theorems of Γ. When α appears on this list, we know that $\Gamma \vdash \alpha$. When $\neg\alpha$ appears on this list, we know that $\Gamma \nvdash \alpha$. Since Γ is complete, one of these cases eventually obtains, so the procedure eventually produces an answer. □

Corollary 153C. *If Γ is consistent, axiomatizable, and represents every decidable property, it is not complete.*

Proof. If Γ were complete, it would be decidable by the previous theorem (since it is axiomatizable and consistent). But since Γ represents every decidable property, it is not decidable, by the first theorem. □

Once we have established that, e.g., **Q**, represents all decidable properties, the corollary tells us that **Q** must be incomplete. However, its proof does not provide an example of an independent sentence; it merely shows that such a sentence must exist. For this, we have to arithmetize syntax and follow Gödel's original proof idea. And of course, we still have to show the first claim, namely that **Q** does, in fact, represent all decidable properties.

It should be noted that not every *interesting* theory is incomplete or undecidable. There are many theories that are sufficiently strong to describe interesting mathematical facts that do not satisify the conditions of Gödel's result. For instance, $\mathbf{Pres} = \{\alpha \in \mathcal{L}_{A^+} : \models_{\mathfrak{B}} \alpha\}$, the set of sentences of the language of arithmetic without $\times$ true in the standard model, is both complete and decidable. This theory is called Presburger arithmetic, and proves all the truths about natural numbers that can be formulated just with 0, $'$, and $+$.

Problems

Problem 1. Show that $\mathbf{TA} = \{\alpha : \models_{\mathfrak{B}} \alpha\}$ is not axiomatizable. You may assume that **TA** represents all decidable properties.

Chapter 16

Arithmetization of Syntax

§16.0 Introduction

In order to connect computability and logic, we need a way to talk about the objects of logic (symbols, terms, wffs, 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 wffs) are a bigger challenge. But if we 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 wffs, such as derivations. This extended coding is called "Gödel numbering." Every term, wff, 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 using 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 wff α, we immediately see that the length of a wff and the (code of the) i-th symbol in a wff can also be computed from the Gödel number of α. It is a bit harder to prove that, e.g., the property of being the Gödel number of a correctly formed term or of a correct derivation is primitive recursive. It is nevertheless possible, because the sequences of interest (terms, wffs, derivations) are inductively defined.

As an example, consider the operation of substitution. If α is a formula, x a variable, and t a term, then $\alpha[t/x]$ is the result of replacing every free occurrence of x in α by t. Now suppose we have assigned Gödel numbers to α, x, t—say, k, l, and m, respectively. The same scheme assigns a Gödel number to $\alpha[t/x]$, say, n. This mapping—of k, l, and m to n—is the arithmetical analog of the substitution operation. When the substitution operation maps α, x, t to $\alpha[t/x]$, 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 power 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).

§16.1 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 $\langle n_0, \dots, n_k \rangle$ is the number coding the sequence of numbers $n_0, \dots, n_k$.)

Definition 161A. *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, then $\mathrm{c}_s = \langle 2, i\rangle$.*
4. *If s is the i-th n-ary function symbol f^n_i, then $\mathrm{c}_s = \langle 3, n, i\rangle$.*
5. *If s is the i-th n-ary predicate symbol P^n_i, then $\mathrm{c}_s = \langle 4, n, i\rangle$.*

Proposition 161B. *The following relations are primitive recursive:*

1. $\mathrm{Fn}(x, n)$ *iff x is the code of f^n_i 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^n_i 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 161C. *If $s_0, \dots, s_{n-1}$ is a sequence of symbols, its* Gödel number *is $\langle \mathrm{c}_{s_0}, \dots, \mathrm{c}_{s_{n-1}}\rangle$.*

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 16.1.4. Recall that if $k_0, \dots, k_{n-1}$ is a sequence of numbers, then the code of the sequence $\langle k_0, \dots, k_{n-1}\rangle$ in the power-of-primes coding is

$$2^{k_0+1} \cdot 3^{k_1+1} \cdot \dots \cdot p_{n-1}^{k_{n-1}},$$

where p_i is the i-th prime (starting with $p_0 = 2$). So for instance, the formula $v_0 = 0$, or, more explicitly, $=(v_0, c_0)$, has the Gödel number

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

Here, $\mathrm{c}_=$ is $\langle 0, 7\rangle = 2^{0+1} \cdot 3^{7+1}$, c_{v_0} is $\langle 1, 0\rangle = 2^{1+1} \cdot 3^{0+1}$, etc. So ${}^{\ulcorner}{=}{}^{\urcorner} = (v_0, c_0)^{\#}$ is

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

§16.2 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.

Variables and constant symbols are the simplest terms, and testing whether x is the Gödel number of such a term is easy: $\mathrm{Var}(x)$ holds if x is ${}^{\ulcorner}v_i{}^{\urcorner}$ for some i. In other words, x is a sequence of length 1 and its single element $(x)_0$ is the code of some variable v_i, i.e., x is $\langle\langle 1, i\rangle\rangle$ for some i. Similarly, $\mathrm{Const}(x)$ holds if x is ${}^{\ulcorner}c_i{}^{\urcorner}$ for some i. Both of these relations are primitive recursive, since if such an i exists, it must be $< x$:

$$\mathrm{Var}(x) \Leftrightarrow (\exists i < x)\ x = \langle\langle 1, i\rangle\rangle$$
$$\mathrm{Const}(x) \Leftrightarrow (\exists i < x)\ x = \langle\langle 2, i\rangle\rangle$$

Proposition 162A. *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., $y = \langle {}^{\ulcorner}s_0{}^{\urcorner}, \ldots, {}^{\ulcorner}s_{k-1}{}^{\urcorner}\rangle$. It codes a formation sequence for the term with Gödel number x iff for all $i < k$:

1. $\mathrm{Var}((y)_i)$, or
2. $\mathrm{Const}((y)_i)$, 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

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

and moreover $(y)_{k-1} = x$. (The function $\mathrm{flatten}(z)$ 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 $\mathrm{len}(y)$. 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 $\mathrm{len}(x)$ 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 $\leq p_{k-1}^{k(x+1)}$, where $k = \mathrm{len}(x)$.

For ClTerm, simply leave out the clause for variables. □

Proposition 162B. *The function* $\mathrm{num}(n) = {}^{\ulcorner}\overline{n}{}^{\urcorner}$ *is primitive recursive.*

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

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

□

§16.3 Coding Wffs

Proposition 163A. *The relation* $\mathrm{Atom}(x)$ *which holds iff* x *is the Gödel number of an atomic wff, is primitive recursive.*

Proof. The number x is the Gödel number of an atomic wff iff one of the following holds:

1. There are n, $j < x$, and $z < x$ such that for each $i < n$, $\mathrm{Term}((z)_i)$ and $x =$

$${}^{\ulcorner}P_j^n({}^{\urcorner} \frown \mathrm{flatten}(z) \frown {}^{\ulcorner})^{\urcorner}.$$

2. There are $z_1, z_2 < x$ such that $\mathrm{Term}(z_1)$, $\mathrm{Term}(z_2)$, and $x =$
$$\ulcorner = \urcorner(\ulcorner \frown z_1 \frown \ulcorner, \ulcorner \frown z_2 \frown \ulcorner)\urcorner.$$

□

Proposition 163B. *The relation* $\mathrm{Frm}(x)$ *which holds iff* x *is the Gödel number of a wff is primitive recursive.*

Proof. A sequence of symbols s is a wff iff there is formation sequence $s_0, \ldots, s_{k-1} = s$ of wff which records how s was formed from atomic wffs 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 163C. *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 163D. *The property* $\mathrm{Sent}(x)$ *which holds iff* x *is the Gödel number of a sentence is primitive recursive.*

Proof. A sentence is a wff without free occurrences of variables. So $\mathrm{Sent}(x)$ holds iff

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

§16.4 Substitution

Recall that substitution is the operation of replacing all free occurrences of a variable u in a wff α by a term t, written $\alpha[t/u]$. This operation, when carried out on Gödel numbers of variables, wffs, and terms, is primitive recursive.

Proposition 164A. *There is a primitive recursive function* $\mathrm{Subst}(x, y, z)$ *with the property that*

$$\mathrm{Subst}(\ulcorner \alpha \urcorner, \ulcorner t \urcorner, \ulcorner u \urcorner) = \ulcorner \alpha[t/u] \urcorner.$$

Proof. We can then define a function hSubst by primitive recursion as follows:

$$\begin{aligned}\mathrm{hSubst}(x, y, z, 0) &= \Lambda\\ \mathrm{hSubst}(x, y, z, i+1) &= \\ &\begin{cases}\mathrm{hSubst}(x, y, z, i) \frown y & \text{if } \mathrm{FreeOcc}(x, z, i)\\ \mathrm{append}(\mathrm{hSubst}(x, y, z, i), (x)_i) & \text{otherwise.}\end{cases}\end{aligned}$$

$\mathrm{Subst}(x, y, z)$ can now be defined as $\mathrm{hSubst}(x, y, z, \mathrm{len}(x))$. □

Proposition 164B. *The relation* 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. □

§16.5 Axiomatic Derivations

In order to arithmetize axiomatic derivations, we must represent derivations as numbers. Since derivations are simply sequences of wffs, the obvious approach is to code every derivation as the code of the sequence of codes of wffs in it.

Definition 165A. *If δ is an axiomatic derivation consisting of wffs $\alpha_1, \ldots, \alpha_n$, then ${}^{\ulcorner}\delta^{\urcorner}$ is*

$$\langle {}^{\ulcorner}\alpha_1{}^{\urcorner}, \ldots, {}^{\ulcorner}\alpha_n{}^{\urcorner} \rangle.$$

Example 16.5.2. Consider the very simple derivation:

1. $\beta \to (\beta \vee \alpha)$
2. $(\beta \to (\beta \vee \alpha)) \to (\alpha \to (\beta \to (\beta \vee \alpha)))$
3. $\alpha \to (\beta \to (\beta \vee \alpha))$

The Gödel number of this derivation would be

$$\begin{aligned} \langle\, &{}^{\ulcorner}\beta \to (\beta \vee \alpha)^{\urcorner}, \\ &{}^{\ulcorner}(\beta \to (\beta \vee \alpha)) \to (\alpha \to (\beta \to (\beta \vee \alpha)))^{\urcorner}, \\ &{}^{\ulcorner}\alpha \to (\beta \to (\beta \vee \alpha))^{\urcorner} \,\rangle. \end{aligned}$$

Having settled on a representation of derivations, we must also show that we can manipulate such derivations primitive recursively, and express their essential properties and relations so. Some operations are simple: e.g., given a Gödel number d of a derivation, $(d)_{\mathrm{len}(d)-1}$ gives us the Gödel number of its end-wff. Some are much harder. We'll at least sketch how to do this. The goal is to show that the relation "δ is a derivation of α from Γ" is primitive recursive in the Gödel numbers of δ and α.

Proposition 165C. *The following relations are primitive recursive:*

1. *α is an axiom.*
2. *The i-th line in δ is justified by modus ponens*
3. *The i-th line in δ is justified by* QR*.*
4. *δ is a correct derivation.*

Proof. We have to show that the corresponding relations between Gödel numbers of wffs and Gödel numbers of derivations are primitive recursive.

1. We have a given list of axiom schemas, and α is an axiom if it is of the form given by one of these schemas. Since the list of schemas is finite, it suffices to show that we can test primitive recursively, for each axiom schema, if α is of that form. For instance, consider the axiom schema

$$\beta \to (\gamma \to \beta).$$

α is an instance of this axiom schema if there are wffs β and γ such that we obtain α when we concatenate '(' with β with '$\to$' with '(' with γ with '$\to$' with β and with '))'. We can test the corresponding property of the Gödel number n of α, since concatenation of sequences is primitive recursive and the Gödel numbers of β and γ must be smaller than the Gödel number of α, since when the relation holds, both β and γ are sub-wffs of α. Hence, we can define:

$$\begin{aligned}\mathrm{IsAx}_{\beta\to(\gamma\to\beta)}(n) \Leftrightarrow (\exists b < n)\ (\exists c < n)\ (\mathrm{Sent}(b) \wedge \mathrm{Sent}(c) \wedge \\ n = {}^{\#}(^{\#} \frown b \frown {}^{\#}\to^{\#} \frown {}^{\#}(^{\#} \frown c \frown {}^{\#}\to^{\#} \frown b \frown {}^{\#}))^{\#}).\end{aligned}$$

If we have such a definition for each axiom schema, their disjunction defines the property $\mathrm{IsAx}(n)$, "n is the Gödel number of an axiom."

2. The i-th line in δ is justified by modus ponens iff there are lines j and $k < i$ where the sentence on line j is some formula α, the sentence on line k is $\alpha \to \beta$, and the sentence on line i is β.

$$\begin{aligned}\mathrm{MP}(d, i) \Leftrightarrow (\exists j < i)\ (\exists k < i) \\ (d)_k = {}^{\#}(^{\#} \frown (d)_j \frown {}^{\#}\to^{\#} \frown (d)_i \frown {}^{\#})^{\#}\end{aligned}$$

Since bounded quantification, concatenation, and $=$ are primitive recursive, this defines a primitive recursive relation.

3. A line in δ is justified by QR if it is of the form $\beta \to \forall x\, \alpha(x)$, a preceding line is $\beta \to \alpha(c)$ for some constant symbol c, and c does on occur in β. This is the case iff

   a) there is a sentence β and
   b) a wff $\alpha(x)$ with a single variable x free so that
   c) line i contains $\beta \to \forall x\, \alpha(x)$
   d) some line $j < i$ contains $\beta \to \alpha[c/x]$ for a constant c
   e) which does not occur in β.

   All of these can be tested primitive recursively, since the Gödel numbers of β, $\alpha(x)$, and x are less than the Gödel number of the formula on line i,

and that of a less than the Gödel number of the formula on line j:

$$\begin{aligned}\mathrm{QR}_1(d,i) \Leftrightarrow (\exists b < (d)_i)\ (\exists x < (d)_i)\ (\exists a < (d)_i)\ (\exists c < (d)_j)\ (&\\ \mathrm{Var}(x) \wedge \mathrm{Const}(c) \wedge&\\ (d)_i = {}^{\#}(^{\#} \frown b \frown {}^{\#}{\to}^{\#} \frown {}^{\#}{\forall}^{\#} \frown x \frown a \frown {}^{\#})^{\#} \wedge&\\ (d)_j = {}^{\#}(^{\#} \frown b \frown {}^{\#}{\to}^{\#} \frown \mathrm{Subst}(a,c,x) \frown {}^{\#})^{\#} \wedge&\\ \mathrm{Sent}(b) \wedge \mathrm{Sent}(\mathrm{Subst}(a,c,x)) \wedge (\forall k < \mathrm{len}(b))\ (b)_k \neq (c)_0)&\end{aligned}$$

Here we assume that c and x are the Gödel numbers of the variable and constant considered as terms (i.e., not their symbol codes). We test that x is the only free variable of $\alpha(x)$ by testing if $\alpha(x)[c/x]$ is a sentence, and ensure that c does not occur in β by requiring that every symbol of β is different from c.

We leave the other version of QR as an exercise.

4. d is the Gödel number of a correct derivation iff every line in it is an axiom, or justified by modus ponens or QR. Hence:

$$\mathrm{Deriv}(d) \Leftrightarrow (\forall i < \mathrm{len}(d))\ (\mathrm{IsAx}((d)_i) \vee \mathrm{MP}(d,i) \vee \mathrm{QR}(d,i)) \qquad \square$$

Proposition 165D. *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 δ of α from Γ and y is the Gödel number of α" is primitive recursive.*

Proof. Suppose "$y \in \Gamma$" is given by the primitive recursive predicate $R_\Gamma(y)$. We have to show that the relation $\mathrm{Prf}_\Gamma(x,y)$ is primitive recursive, where $\mathrm{Prf}_\Gamma(x,y)$ holds iff y is the Gödel number of a sentence α and x is the code of a derivation of α from Γ.

By the previous proposition, the property $\mathrm{Deriv}(x)$ which holds iff x is the code of a correct derivation δ is primitive recursive. However, that definition did not take into account the set Γ as an additional way to justify lines in the derivation. Our primitive recursive test of whether a line is justified by QR also left out of consideration the requirement that the constant c is not allowed to occur in Γ. It is possible to amend our definition so that it takes into account Γ directly, but it is easier to use Deriv and the deduction theorem. $\Gamma \vdash \alpha$ iff there is some finite list of sentences $\beta_1, \ldots, \beta_n \in \Gamma$ such that $\{\beta_1, \ldots, \beta_n\} \vdash \alpha$. And by the deduction theorem, this is the case if $\vdash (\beta_1 \to (\beta_2 \to \cdots (\beta_n \to \alpha) \cdots))$. Whether a sentence with Gödel number z is of this form can be tested primitive recursively. So, instead of considering x as the Gödel number of a derivation of the sentence with Gödel number y *from* Γ, we consider x as the Gödel number of a derivation of a nested conditional of the above form from $\emptyset$.

First, if we have a sequence of sentences, we can primitive recursively form the conditional with all these sentences as antecedents and given sentence as

consequent:

$$\begin{aligned}
\text{hCond}(s, y, 0) &= y \\
\text{hCond}(s, y, n+1) &= {}^{\ulcorner}(^{\ulcorner} \frown (s)_n \frown {}^{\ulcorner}\!\to{}^{\urcorner} \frown \text{Cond}(s, y, n) \frown {}^{\ulcorner})^{\urcorner} \\
\text{Cond}(s, y) &= \text{hCond}(s, y, \text{len}(s))
\end{aligned}$$

So we can define $\text{Prf}_\Gamma(x, y)$ by

$$\begin{aligned}
\text{Prf}_\Gamma(x, y) \Leftrightarrow (\exists s < \text{sequenceBound}(x, x))\ (& \\
& (x)_{\text{len}(x)-1} = \text{Cond}(s, y) \wedge \\
& (\forall i < \text{len}(s))\ (s)_i \in \Gamma \wedge \\
& \text{Deriv}(x)).
\end{aligned}$$

The bound on s is given by considering that each $(s)_i$ is the Gödel number of a sub-wff of the last line of the derivation, i.e., is less than $(x)_{\text{len}(x)-1}$. The number of antecedents $\beta \in \Gamma$, i.e., the length of s, is less than the length of the last line of x. □

Problems

Problem 1. Show that the function flatten(z), which turns the sequence $\langle {}^{\ulcorner}t_1{}^{\urcorner}, \ldots, {}^{\ulcorner}t_n{}^{\urcorner}\rangle$ into ${}^{\ulcorner}t_1, \ldots, t_n{}^{\urcorner}$, is primitive recursive.

Problem 2. Give a detailed proof of Proposition 163B along the lines of the first proof of Proposition 162A.

Problem 3. Prove Proposition 163C. You may make use of the fact that any substring of a wff which is a wff is a sub-wff of it.

Problem 4. Prove Proposition 164B

Problem 5. Define the following relations as in Proposition 165C:

1. $\text{IsAx}_{\alpha\to(\beta\to(\alpha\wedge\beta))}(n)$,
2. $\text{IsAx}_{\forall x\,\alpha(x)\to\alpha(t)}(n)$,
3. $\text{QR}_2(d, i)$ (for the other version of QR).

Chapter 17

Representability in Q

§17.0 Introduction

The incompleteness theorems apply to theories in which basic facts about computable functions can be expressed and proved. 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 $\{\alpha : \mathbf{Q} \vdash \alpha\}$ 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 equality symbol):

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

For each natural number n, define the numeral $\overline{n}$ to be the term $0''^{\cdots'}$ 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**:

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

where $\alpha(x)$ is any formula. If $\alpha(x)$ 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 $\alpha(x, y)$ also contains the variable y free, the corresponding instance is

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

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 170A. *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 $\alpha_f(x_0, \ldots, x_k, y)$ such that whenever $f(n_0, \ldots, n_k) = m$, **Q** proves*

1. $\alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})$
2. $\forall y\,(\alpha_f(\overline{n_0}, \ldots, \overline{n_k}, y) \to \overline{m} = y)$.

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

Theorem 170B. *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_i^n, 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.

§17.1 Functions Representable in Q are Computable

We'll prove that every function that is representable in **Q** is computable. We first have to establish a lemma about functions representable in **Q**.

Lemma 171A. *If $f(x_0, \ldots, x_k)$ is representable in* **Q**, *there is a wff $\alpha(x_0, \ldots, x_k, y)$ such that*

$$\mathbf{Q} \vdash \alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m}) \quad \textit{iff} \quad m = f(n_0, \ldots, n_k).$$

Proof. The "if" part is Definition 170A(1). The "only if" part is seen as follows: Suppose $\mathbf{Q} \vdash \alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})$ but $m \neq f(n_0, \ldots, n_k)$. Let $l = f(n_0, \ldots, n_k)$. By Definition 170A(1), $\mathbf{Q} \vdash \alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{l})$. By Definition 170A(2), $\forall y\,(\alpha_f(\overline{n_0}, \ldots, \overline{n_k}, y) \rightarrow \overline{l} = y)$. Using logic and the assumption that $\mathbf{Q} \vdash \alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})$, we get that $\mathbf{Q} \vdash \overline{l} = \overline{m}$. On the other hand, by Lemma 174E, $\mathbf{Q} \vdash \overline{l} \neq \overline{m}$. So **Q** is inconsistent. But that is impossible, since **Q** is satisfied by the standard model (see Definition 151B), $\models_{\mathfrak{B}} \mathbf{Q}$, and satisfiable theories are always consistent by the Soundness Theorem (corollary 1011D). □

Lemma 171B. *Every function that is representable in* **Q** *is computable.*

Proof. Let's first give the intuitive idea for why this is true. 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 wff of the form $\alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})$ (the wff representing f in **Q** from Lemma 171A). If it is, $m = f(n_0, \ldots, n_k)$ by Lemma 171A, and we've found the value of f. The search terminates because $\mathbf{Q} \vdash \alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{f(n_0, \ldots, n_k)})$, so eventually we find a δ of the right sort.

This is not quite precise because our procedure operates on derivations and wffs 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, wffs, 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 wff with Gödel number y from the axioms of **Q**,

is (primitive) recursive. Other primitive recursive functions we'll need are num (Proposition 162B) and Subst (Proposition 164A). From these, it is possible to define f by minimization; thus, f is recursive.

First, define

$$\begin{aligned} A(n_0, \ldots, n_k, m) = \\ \text{Subst}(\text{Subst}(\ldots \text{Subst}(&{}^{\ulcorner}\alpha_f{}^{\urcorner}, \text{num}(n_0), {}^{\ulcorner}x_0{}^{\urcorner}), \\ &\ldots), \text{num}(n_k), {}^{\ulcorner}x_k{}^{\urcorner}), \text{num}(m), {}^{\ulcorner}y{}^{\urcorner}) \end{aligned}$$

This looks complicated, but it's just the function $A(n_0, \ldots, n_k, m) = {}^{\ulcorner}\alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})^{\urcorner}$.

Now, consider the relation $R(n_0, \ldots, n_k, s)$ which holds if $(s)_0$ is the Gödel number of a derivation from **Q** of $\alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{(s)_1})$:

$$R(n_0, \ldots, n_k, s) \quad \text{iff} \quad \text{Prf}_{\mathbf{Q}}((s)_0, A(n_0, \ldots, n_k, (s)_1))$$

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 $\mathbf{Q} \vdash \alpha_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{f(n_0, \ldots, n_k)})$, so $R(n_0, \ldots, n_k, s)$ holds for $s = \langle {}^{\ulcorner}\delta^{\urcorner}, f(n_0, \ldots, n_k)\rangle$. So, we can write f as

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

§17.2 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 172A. *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 power-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 Sunzi's Theorem (traditionally, the "Chinese Remainder Theorem").

Definition 172B. *Two natural numbers a and b are* relatively prime *iff their greatest common divisor is 1; in other words, they have no other divisors in common.*

Definition 172C. *Natural numbers a and b are* congruent modulo c, $a \equiv b \mod c$, *iff $c \mid (a - b)$, i.e., a and b have the same remainder when divided by c.*

Here is Sunzi's Theorem:

Theorem 172D. *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 Sunzi's 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:

$$\begin{aligned} \mathrm{not}(x) &= \chi_=(x, 0) \\ (\min x \le z)\, R(x, y) &= \mu x\, (R(x, y) \vee x = z) \\ (\exists x \le z)\, R(x, y) &\Leftrightarrow R((\min x \le z)\, R(x, y), y) \end{aligned}$$

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. the projection functions

$$\begin{aligned} K(z) &= (\min x \le z)\, (\exists y \le z)\, z = J(x, y), \\ L(z) &= (\min y \le z)\, (\exists x \le z)\, z = J(x, y); \end{aligned}$$

3. the less-than relation $x < y$;
4. the divisibility relation $x \mid y$;
5. the function $\mathrm{rem}(x, y)$ which returns the remainder when y is divided by x.

Now define

$$\begin{aligned} \beta^*(d_0, d_1, i) &= \mathrm{rem}(1 + (i+1)d_1, d_0) \text{ and} \\ \beta(d, i) &= \beta^*(K(d), L(d), i). \end{aligned}$$

This is the function we want. Given $a_0, \ldots, a_n$ as above, let

$$j = \max(n, a_0, \ldots, a_n) + 1,$$

and let $d_1 = j!$. By (1) above, we know that $1 + d_1$, $1 + 2d_1$, ..., $1 + (n+1)d_1$ are relatively prime, and by (2) that all are greater than $a_0, \ldots, a_n$. By Sunzi's 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.

§17.3 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 $f(\vec{x})$ and $g(\vec{x}, y, z)$. Then the function $h(x, \vec{z})$ defined from f and g by primitive recursion is

$$\begin{aligned}h(\vec{x}, 0) &= f(\vec{x})\\ h(\vec{x}, y+1) &= g(\vec{x}, y, h(\vec{x}, y)).\end{aligned}$$

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 173A. *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 $\hat{h}(\vec{x}, y)$ which returns the least number d such that d codes a sequence which satisfies

1. $(d)_0 = f(\vec{x})$, and
2. for each $i < y$, $(d)_{i+1} = g(\vec{x}, i, (d)_i)$,

where now $(d)_i$ is short for $\beta(d, i)$. In other words, $\hat{h}$ returns the sequence $\langle h(\vec{x}, 0), h(\vec{x}, 1), \ldots, h(\vec{x}, y)\rangle$. We can write $\hat{h}$ as

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

Note: no primitive recursion is needed here, just minimization. The function we minimize is regular because of the beta function lemma Lemma 172A.

But now we have

$$h(\vec{x}, y) = \beta(\hat{h}(\vec{x}, y), y),$$

so h can be defined from the basic functions using just composition and regular minimization. □

§17.4 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 wff $\alpha_f(x_0, \ldots, x_{k-1}, y)$ 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 wff $x_0' = y$, and addition is represented by the wff $(x_0 + x_1) = y$. The work involves showing that **Q** can prove the relevant sentences; for example, saying that addition is represented by the wff above involves showing that for every pair of natural numbers m and n, **Q** proves

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

Proposition 174A. *The zero function* $\mathrm{zero}(x) = 0$ *is represented in* **Q** *by* $\alpha_{\mathrm{zero}}(x, y) \equiv y = 0$.

Proposition 174B. *The successor function* $\mathrm{succ}(x) = x + 1$ *is represented in* **Q** *by* $\alpha_{\mathrm{succ}}(x, y) \equiv y = x'$.

Proposition 174C. *The projection function* $P_i^n(x_0, \ldots, x_{n-1}) = x_i$ *is represented in* **Q** *by*

$$\alpha_{P_i^n}(x_0, \ldots, x_{n-1}, y) \equiv y = x_i.$$

Proposition 174D. *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*

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

The proof requires the following lemma.

Lemma 174E. *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 **Q** can prove that different numerals denote different objects. For example, **Q** proves $0'' \neq 0'''$. But showing that this holds in general requires some care. Note also that although we are using induction, it is induction *outside* of **Q**.

Proof of Proposition 174D. 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 $\alpha_=(\overline{n}, \overline{m}, \overline{1})$. If $n \neq m$, then $\chi_=(n, m) = 0$. By Lemma 174E, $\mathbf{Q} \vdash \overline{n} \neq \overline{m}$ and so also $(\overline{n} \neq \overline{m} \wedge 0 = 0)$. Thus $\mathbf{Q} \vdash \alpha_=(\overline{n}, \overline{m}, \overline{0})$.

For the second part, we also have two cases. If $n = m$, we have to show that $\mathbf{Q} \vdash \forall y\, (\alpha_=(\overline{n}, \overline{m}, y) \to y = \overline{1})$. Arguing informally, suppose $\alpha_=(\overline{n}, \overline{m}, y)$, 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 $\alpha_=(\overline{n}, \overline{m}, y)$ 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 **Q** by Lemma 174E; the right disjunct entails $y = \overline{0}$. □

Proposition 174F. *The addition function* $\mathrm{add}(x_0, x_1) = x_0 + x_1$ *is represented in* **Q** *by*

$$\alpha_{\mathrm{add}}(x_0, x_1, y) \equiv y = (x_0 + x_1).$$

Lemma 174G. $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$

Proof. We prove this by induction on m. If $m = 0$, the claim is that $\mathbf{Q} \vdash (\overline{n} + 0) = \overline{n}$. This follows by axiom Q_4. Now suppose the claim for m; let's prove the claim for $m + 1$, i.e., prove that $\mathbf{Q} \vdash (\overline{n} + \overline{m+1}) = \overline{n + m + 1}$. Note that $\overline{m+1}$ is just $\overline{m}'$, and $\overline{n + m + 1}$ is just $\overline{n + m}'$. By axiom Q_5, $\mathbf{Q} \vdash (\overline{n} + \overline{m}') = (\overline{n} + \overline{m})'$. By induction hypothesis, $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$. So $\mathbf{Q} \vdash (\overline{n} + \overline{m}') = \overline{n + m}'$. □

Proof of Proposition 174F. The wff $\alpha_{\text{add}}(x_0, x_1, y)$ representing add is $y = (x_0 + x_1)$. First we show that if $\text{add}(n, m) = k$, then $\mathbf{Q} \vdash \alpha_{\text{add}}(\overline{n}, \overline{m}, \overline{k})$, 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 174G that $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n+m}$.

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

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

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 174H. *The multiplication function* $\text{mult}(x_0, x_1) = x_0 \cdot x_1$ *is represented in* **Q** *by*

$$\alpha_{\text{mult}}(x_0, x_1, y) \equiv y = (x_0 \times x_1).$$

Proof. Exercise. □

Lemma 174I. $\mathbf{Q} \vdash (\overline{n} \times \overline{m}) = \overline{n \cdot m}$

Proof. Exercise. □

Recall that we use $\times$ for the function symbol of the language of arithmetic, and $\cdot$ for the ordinary multiplication operation on numbers. So $\cdot$ can appear between expressions for numbers (such as in $m \cdot n$) while $\times$ appears only between terms of the language of arithmetic (such as in $(\overline{m} \times \overline{n})$). Even more confusingly, $+$ is used for both the function symbol and the addition operation. When it appears between terms—e.g., in $(\overline{n} + \overline{m})$—it is the 2-place function symbol of the language of arithmetic, and when it appears between numbers—e.g., in $n + m$—it is the addition operation. This includes the case $\overline{n+m}$: this is the standard numeral corresponding to the number $n + m$.

§17.5 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 wffs $\alpha_f, \alpha_{g_0}, \ldots, \alpha_{g_{k-1}}$ representing the functions f, and $g_0, \ldots, g_{k-1}$, respectively. We have to find a wff α_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 $\alpha_f(y, z)$ represents f, and $\alpha_g(x, y)$ represents g, we need a wff $\alpha_h(x, z)$ 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 $\exists y\, (\alpha_g(x, y) \wedge \alpha_f(y, z))$ is a good candidate for $\alpha_h(x, z)$. We just have to verify that **Q** proves the relevant wffs.

Proposition 175A. *If $h(n) = m$, then $\mathbf{Q} \vdash \alpha_h(\overline{n}, \overline{m})$.*

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

$$\mathbf{Q} \vdash \alpha_g(\overline{n}, \overline{k})$$

since α_g represents g, and

$$\mathbf{Q} \vdash \alpha_f(\overline{k}, \overline{m})$$

since α_f represents f. Thus,

$$\mathbf{Q} \vdash \alpha_g(\overline{n}, \overline{k}) \wedge \alpha_f(\overline{k}, \overline{m})$$

and consequently also

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

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

Proposition 175B. *If $h(n) = m$, then $\mathbf{Q} \vdash \forall z\, (\alpha_h(\overline{n}, z) \to z = \overline{m})$.*

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

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

since α_g represents g, and

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

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

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

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

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

Proposition 175C. *If $\alpha_f(y_0, \ldots, y_{k-1}, z)$ represents $f(y_0, \ldots, y_{k-1})$ in $\mathbf{Q}$, and $\alpha_{g_i}(x_0, \ldots, x_{l-1}, y)$ represents $g_i(x_0, \ldots, x_{l-1})$ in $\mathbf{Q}$, then*

$$\begin{aligned}\exists y_0 \ldots \exists y_{k-1}\, (&\alpha_{g_0}(x_0, \ldots, x_{l-1}, y_0) \wedge \cdots \wedge \\ &\alpha_{g_{k-1}}(x_0, \ldots, x_{l-1}, y_{k-1}) \wedge \alpha_f(y_0, \ldots, y_{k-1}, z))\end{aligned}$$

represents

$$h(x_0, \ldots, x_{l-1}) = f(g_0(x_0, \ldots, x_{l-1}), \ldots, g_{k-1}(x_0, \ldots, x_{l-1})).$$

Proof. Exercise. □

§17.6 Regular Minimization is Representable in Q

Let's consider unbounded search. Suppose $g(x, z)$ is regular and representable in $\mathbf{Q}$, say by the wff $\alpha_g(x, z, y)$. Let f be defined by $f(z) = \mu x\,[g(x, z) = 0]$. We would like to find a wff $\alpha_f(z, y)$ 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:

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

In the general case, of course, we would have to replace z with $z_0, \ldots, z_k$.

The proof, again, will involve some lemmas about things $\mathbf{Q}$ is strong enough to prove.

Lemma 176A. *For every constant symbol a and every natural number n,*

$$\mathbf{Q} \vdash (a' + \overline{n}) = (a + \overline{n})'.$$

Proof. The proof is, as usual, by induction on n. In the base case, $n = 0$, we need to show that $\mathbf{Q}$ proves $(a' + 0) = (a + 0)'$. But we have:

$$\mathbf{Q} \vdash (a' + 0) = a' \quad \text{by axiom } Q_4 \tag{17.1}$$
$$\mathbf{Q} \vdash (a + 0) = a \quad \text{by axiom } Q_4 \tag{17.2}$$
$$\mathbf{Q} \vdash (a + 0)' = a' \quad \text{by eq. (17.2)} \tag{17.3}$$
$$\mathbf{Q} \vdash (a' + 0) = (a + 0)' \quad \text{by eq. (17.1) and eq. (17.3)}$$

In the induction step, we can assume that we have shown that $\mathbf{Q} \vdash (a' + \overline{n}) = (a + \overline{n})'$. Since $\overline{n+1}$ is $\overline{n}'$, we need to show that $\mathbf{Q}$ proves $(a' + \overline{n}') = (a + \overline{n}')'$. We have:

$$\mathbf{Q} \vdash (a' + \overline{n}') = (a' + \overline{n})' \quad \text{by axiom } Q_5 \tag{17.4}$$
$$\mathbf{Q} \vdash (a' + \overline{n}') = (a + \overline{n}')' \quad \text{inductive hypothesis} \tag{17.5}$$
$$\mathbf{Q} \vdash (a' + \overline{n})' = (a + \overline{n}')' \quad \text{by eq. (17.4) and eq. (17.5).}$$ □

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

Lemma 176B. $\mathbf{Q} \vdash \forall x\,\neg x < 0$.

Proof. We give the proof informally (i.e., only giving hints as to how to construct the formal derivation).

We have to prove $\neg a < 0$ for an arbitrary a. By the definition of $<$, we need to prove $\neg\exists y\,(y' + a) = 0$ in $\mathbf{Q}$. We'll assume $\exists y\,(y' + a) = 0$ and prove a contradiction. Suppose $(b' + a) = 0$. Using Q_3, we have that $a = 0 \vee \exists y\,a = y'$. We distinguish cases.

Case 1: $a = 0$ holds. From $(b' + a) = 0$, we have $(b' + 0) = 0$. By axiom Q_4 of **Q**, we have $(b' + 0) = b'$, and hence $b' = 0$. But by axiom Q_2 we also have $b' \neq 0$, a contradiction.

Case 2: For some c, $a = c'$. But then we have $(b' + c') = 0$. By axiom Q_5, we have $(b' + c)' = 0$, again contradicting axiom Q_2. □

Lemma 176C. *For every natural number n,*

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

Proof. We use induction on n. Let us consider the base case, when $n = 0$. In that case, we need to show $a < \overline{1} \to a = 0$, for arbitrary a. Suppose $a < \overline{1}$. Then by the defining axiom for $<$, we have $\exists y\,(y' + a) = 0'$ (since $\overline{1} \equiv 0'$).

Suppose b has that property, i.e., we have $(b' + a) = 0'$. We need to show $a = 0$. By axiom Q_3, we have either $a = 0$ or that there is a c such that $a = c'$. In the former case, there is nothing to show. So suppose $a = c'$. Then we have $(b' + c') = 0'$. By axiom Q_5 of **Q**, we have $(b' + c)' = 0'$. By axiom Q_1, we have $(b' + c) = 0$. But this means, by axiom Q_8, that $c < 0$, contradicting Lemma 176B.

Now for the inductive step. We prove the case for $n + 1$, assuming the case for n. So suppose $a < \overline{n+2}$. Again using Q_3 we can distinguish two cases: $a = 0$ and for some b, $a = c'$. In the first case, $a = 0 \vee \cdots \vee a = \overline{n+1}$ follows trivially. In the second case, we have $c' < \overline{n+2}$, i.e., $c' < \overline{n+1}'$. By axiom Q_8, for some d, $(d' + c') = \overline{n+1}'$. By axiom Q_5, $(d' + c)' = \overline{n+1}'$. By axiom Q_1, $(d' + c) = \overline{n+1}$, and so $c < \overline{n+1}$ by axiom Q_8. By inductive hypothesis, $c = 0 \vee \cdots \vee c = \overline{n}$. From this, we get $c' = 0' \vee \cdots \vee c' = \overline{n}'$ by logic, and so $a = \overline{1} \vee \cdots \vee a = \overline{n+1}$ since $a = c'$. □

Lemma 176D. *For every natural number m,*

$$\mathbf{Q} \vdash \forall y\,((y < \overline{m} \vee \overline{m} < y) \vee y = \overline{m}).$$

Proof. By induction on m. First, consider the case $m = 0$. $\mathbf{Q} \vdash \forall y\,(y = 0 \vee \exists z\, y = z')$ by Q_3. Let a be arbitrary. Then either $a = 0$ or for some b, $a = b'$. In the former case, we also have $(a < 0 \vee 0 < a) \vee a = 0$. But if $a = b'$, then $(b' + 0) = (a + 0)$ by the logic of $=$. By Q_4, $(a + 0) = a$, so we have $(b' + 0) = a$, and hence $\exists z\,(z' + 0) = a$. By the definition of $<$ in Q_8, $0 < a$. If $0 < a$, then also $(0 < a \vee a < 0) \vee a = 0$.

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

$$\mathbf{Q} \vdash \forall y\,((y < \overline{m+1} \vee \overline{m+1} < y) \vee y = \overline{m+1})$$

Let a be arbitrary. By Q_3, either $a = 0$ or for some b, $a = b'$. In the first case, we have $\overline{m}' + a = \overline{m+1}$ by Q_4, and so $a < \overline{m+1}$ by Q_8.

Now consider the second case, $a = b'$. By the induction hypothesis, $(b < \overline{m} \vee \overline{m} < b) \vee b = \overline{m}$.

The first disjunct $b < \overline{m}$ is equivalent (by Q_8) to $\exists z\,(z' + b) = \overline{m}$. Suppose c has this property. If $(c' + b) = \overline{m}$, then also $(c' + b)' = \overline{m}'$. By Q_5, $(c' + b)' = (c' + b')$. Hence, $(c' + b') = \overline{m}'$. We get $\exists u\,(u' + b') = \overline{m+1}$ by existentially generalizing on c' and keeping in mind that $\overline{m}' \equiv \overline{m+1}$. Hence, if $b < \overline{m}$ then $b' < \overline{m+1}$ and so $a < \overline{m+1}$.

Now suppose $\overline{m} < b$, i.e., $\exists z\,(z' + \overline{m}) = b$. Suppose c is such a z, i.e., $(c' + \overline{m}) = b$. By logic, $(c' + \overline{m})' = b'$. By Q_5, $(c' + \overline{m}') = b'$. Since $a = b'$ and $\overline{m}' \equiv \overline{m+1}$, $(c' + \overline{m+1}) = a$. By Q_8, $\overline{m+1} < a$.

Finally, assume $b = \overline{m}$. Then, by logic, $b' = \overline{m}'$, and so $a = \overline{m+1}$.

Hence, from each disjunct of the case for m and b, we can obtain the corresponding disjunct for for $m + 1$ and a. □

Proposition 176E. *If $\alpha_g(x, z, y)$ represents $g(x, z)$ in **Q**, then*

$$\alpha_f(z, y) \equiv \alpha_g(y, z, 0) \wedge \forall w\,(w < y \to \neg\alpha_g(w, z, 0))$$

represents $f(z) = \mu x\,[g(x, z) = 0]$.

Proof. First we show that if $f(n) = m$, then $\mathbf{Q} \vdash \alpha_f(\overline{n}, \overline{m})$, i.e.,

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

Since $\alpha_g(x, z, y)$ represents $g(x, z)$ and $g(m, n) = 0$ if $f(n) = m$, we have

$$\mathbf{Q} \vdash \alpha_g(\overline{m}, \overline{n}, 0).$$

If $f(n) = m$, then for every $k < m$, $g(k, n) \neq 0$. So

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

We get that

$$\mathbf{Q} \vdash \forall w\,(w < \overline{m} \to \neg\alpha_g(w, \overline{n}, 0)). \tag{17.6}$$

by Lemma 176B in case $m = 0$ and by Lemma 176C otherwise.

Now let's show that if $f(n) = m$, then $\mathbf{Q} \vdash \forall y\,(\alpha_f(\overline{n}, y) \to y = \overline{m})$. We again sketch the argument informally, leaving the formalization to the reader.

Suppose $\alpha_f(\overline{n}, b)$. From this we get (a) $\alpha_g(b, \overline{n}, 0)$ and (b) $\forall w\,(w < b \to \neg\alpha_g(w, \overline{n}, 0))$. By Lemma 176D, $(b < \overline{m} \vee \overline{m} < b) \vee b = \overline{m}$. We'll show that both $b < \overline{m}$ and $\overline{m} < b$ leads to a contradiction.

If $\overline{m} < b$, then $\neg\alpha_g(\overline{m}, \overline{n}, 0)$ from (b). But $m = f(n)$, so $g(m, n) = 0$, and so $\mathbf{Q} \vdash \alpha_g(\overline{m}, \overline{n}, 0)$ since α_g represents g. So we have a contradiction.

Now suppose $b < \overline{m}$. Then since $\mathbf{Q} \vdash \forall w\,(w < \overline{m} \to \neg\alpha_g(w, \overline{n}, 0))$ by eq. (17.6), we get $\neg\alpha_g(b, \overline{n}, 0)$. This again contradicts (a). □

§17.7 Computable Functions are Representable in Q

Theorem 177A. *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 173A, 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 174A to 174D, 174F and 174H), and that any function defined from representable functions by composition or regular minimization (Proposition 175C, Proposition 176E) 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 derivation system in which the notion of derivation 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.

§17.8 Representing Relations

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

Definition 178A. *A relation $R(x_0, \ldots, x_k)$ on the natural numbers is* representable in **Q** *if there is a formula $\alpha_R(x_0, \ldots, x_k)$ such that whenever $R(n_0, \ldots, n_k)$ is true,* **Q** *proves $\alpha_R(\overline{n_0}, \ldots, \overline{n_k})$, and whenever $R(n_0, \ldots, n_k)$ is false,* **Q** *proves $\neg\alpha_R(\overline{n_0}, \ldots, \overline{n_k})$.*

Theorem 178B. *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 $\alpha_R(x_0, \ldots, x_k)$. Here is an algorithm for computing R: on input n_0, $\ldots$, n_k, simultaneously search for a proof of $\alpha_R(\overline{n_0}, \ldots, \overline{n_k})$ and a proof of $\neg\alpha_R(\overline{n_0}, \ldots, \overline{n_k})$. 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 170B, χ_R is represented by a formula, say $\alpha_{\chi_R}(x_0, \ldots, x_k, y)$. Let $\alpha_R(x_0, \ldots, x_k)$ be the formula $\alpha_{\chi_R}(x_0, \ldots, x_k, \overline{1})$. 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 $\alpha_{\chi_R}(\overline{n_0}, \ldots, \overline{n_k}, \overline{1})$, and so $\mathbf{Q}$ proves $\alpha_R(\overline{n_0}, \ldots, \overline{n_k})$. 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

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

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

§17.9 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 α?" for any sentence α in the language of $\mathbf{T}$. So $\mathbf{Q}$ would be decidable iff there were a computational procedure which decides, given a sentence α in the language of arithmetic, whether $\mathbf{Q} \vdash \alpha$ 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 179A. *$\mathbf{Q}$ 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 1315A. Since T is primitive recursive it is representable in $\mathbf{Q}$ by a formula β_T, that is, $\mathbf{Q} \vdash \beta_T(\overline{e}, \overline{n}, \overline{s})$ iff $T(e, n, s)$. If $\mathbf{Q} \vdash \beta_T(\overline{e}, \overline{n}, \overline{s})$ then also $\mathbf{Q} \vdash \exists y\, \beta_T(\overline{e}, \overline{n}, y)$. If no such s exists, then $\mathbf{Q} \vdash \neg\beta_T(\overline{e}, \overline{n}, \overline{s})$ for every s. But $\mathbf{Q}$ is ω-consistent, i.e., if $\mathbf{Q} \vdash \neg\alpha(\overline{n})$ for every $n \in \mathbb{N}$, then $\mathbf{Q} \nvdash \exists y\, \alpha(y)$. We know this because the axioms of $\mathbf{Q}$ are true in the standard model $\mathfrak{B}$. So, $\mathbf{Q} \nvdash \exists y\, \beta_T(\overline{e}, \overline{n}, y)$. In other words, $\mathbf{Q} \vdash \exists y\, \beta_T(\overline{e}, \overline{n}, y)$ 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 ${}^{\#}\exists y\, \beta_T(\overline{e}, \overline{n}, y)^{\#}$, 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 1316A, so $\mathrm{Prov}_{\mathbf{Q}}$ cannot be either. □

Corollary 179B. *First-order logic is undecidable.*

Proof. If first-order logic were decidable, provability in **Q** would be as well, since $\mathbf{Q} \vdash \alpha$ iff $\vdash \omega \to \alpha$, where ω is the conjunction of the axioms of **Q**. □

Problems

Problem 1. Show that the relations $x < y$, $x \mid y$, and the function $\mathrm{rem}(x, y)$ can be defined without primitive recursion. You may use 0, successor, plus, times, $\chi_=$, projections, and bounded minimization and quantification.

Problem 2. Prove that $y = 0$, $y = x'$, and $y = x_i$ represent zero, succ, and P_i^n, respectively.

Problem 3. Prove Lemma 174I.

Problem 4. Use Lemma 174I to prove Proposition 174H.

Problem 5. Using the proofs of Proposition 175B and Proposition 175B as a guide, carry out the proof of Proposition 175C in detail.

Problem 6. Show that if R is representable in **Q**, so is χ_R.

Chapter 18

Theories and Computability

§18.0 Introduction

We have the following:

1. A definition of what it means for a function to be representable in **Q** (Definition 170A)
2. a definition of what it means for a relation to be representable in **Q** (Definition 178A)
3. a theorem asserting that the representable functions of **Q** are exactly the computable ones (Theorem 170B)
4. a theorem asserting that the representable relations of **Q** are exactly the computable ones Theorem 178B)

A *theory* is a set of sentences that is deductively closed, that is, with the property that whenever T proves α then α is in T. It is probably best to think of a theory as being a collection of sentences, together with all the things that these sentences imply. From now on, we will use **Q** to refer to the *theory* consisting of the set of sentences derivable from the eight axioms in section 17.0. Remember that we can code formula of **Q** as numbers; if α is such a formula, let ${}^{\#}\alpha^{\#}$ denote the number coding α. Modulo this coding, we can now ask whether various sets of formulas are computable or not.

§18.1 Q is C.e.-Complete

Theorem 181A. **Q** *is c.e. but not decidable. In fact, it is a complete c.e. set.*

Proof. It is not hard to see that **Q** is c.e., since it is the set of (codes for) sentences y such that there is a proof x of y in **Q**:

$$Q = \{y : \exists x \, \mathrm{Prf}_{\mathbf{Q}}(x, y)\}.$$

But we know that $\mathrm{Prf}_{\mathbf{Q}}(x, y)$ is computable (in fact, primitive recursive), and any set that can be written in the above form is c.e.

Saying that it is a complete c.e. set is equivalent to saying that $K \leq_m Q$, where $K = \{x : \alpha_x(x) \downarrow\}$. So let us show that K is reducible to $\mathbf{Q}$. Since Kleene's predicate $T(e, x, s)$ is primitive recursive, it is representable in $\mathbf{Q}$, say, by α_T. Then for every x, we have

$$\begin{aligned} x \in K &\to \exists s\, T(x, x, s) \\ &\to \exists s\, (\mathbf{Q} \vdash \alpha_T(\overline{x}, \overline{x}, \overline{s})) \\ &\to \mathbf{Q} \vdash \exists s\, \alpha_T(\overline{x}, \overline{x}, s). \end{aligned}$$

Conversely, if $\mathbf{Q} \vdash \exists s\, \alpha_T(\overline{x}, \overline{x}, s)$, then, in fact, for some natural number n the formula $\alpha_T(\overline{x}, \overline{x}, \overline{n})$ must be true. Now, if $T(x, x, n)$ were false, $\mathbf{Q}$ would prove $\neg\alpha_T(\overline{x}, \overline{x}, \overline{n})$, since α_T represents T. But then $\mathbf{Q}$ proves a false formula, which is a contradiction. So $T(x, x, n)$ must be true, which implies $\alpha_x(x) \downarrow$.

In short, we have that for every x, x is in K if and only if $\mathbf{Q}$ proves $\exists s\, T(\overline{x}, \overline{x}, s)$. So the function f which takes x to (a code for) the sentence $\exists s\, T(\overline{x}, \overline{x}, s)$ is a reduction of K to $\mathbf{Q}$. □

§18.2 ω-Consistent Extensions of Q are Undecidable

The proof that $\mathbf{Q}$ is c.e.-complete relied on the fact that any sentence provable in $\mathbf{Q}$ is "true" of the natural numbers. The next definition and theorem strengthen this theorem, by pinpointing just those aspects of "truth" that were needed in the proof above. Don't dwell on this theorem too long, though, because we will soon strengthen it even further. We include it mainly for historical purposes: Gödel's original paper used the notion of ω-consistency, but his result was strengthened by replacing ω-consistency with ordinary consistency soon after.

Definition 182A. *A theory $\mathbf{T}$ is ω-consistent if the following holds: if $\exists x\, \alpha(x)$ is any sentence and $\mathbf{T}$ proves $\neg\alpha(\overline{0})$, $\neg\alpha(\overline{1})$, $\neg\alpha(\overline{2})$, ... then $\mathbf{T}$ does not prove $\exists x\, \alpha(x)$.*

Theorem 182B. *Let $\mathbf{T}$ be any ω-consistent theory that includes $\mathbf{Q}$. Then $\mathbf{T}$ is not decidable.*

Proof. If $\mathbf{T}$ includes $\mathbf{Q}$, then $\mathbf{T}$ represents the computable functions and relations. We need only modify the previous proof. As above, if $x \in K$, then $\mathbf{T}$ proves $\exists s\, \alpha_T(\overline{x}, \overline{x}, s)$. Conversely, suppose $\mathbf{T}$ proves $\exists s\, \alpha_T(\overline{x}, \overline{x}, s)$. Then x must be in K: otherwise, there is no halting computation of machine x on input x; since α_T represents Kleene's T relation, $\mathbf{T}$ proves $\neg\alpha_T(\overline{x}, \overline{x}, \overline{0})$, $\neg\alpha_T(\overline{x}, \overline{x}, \overline{1})$, ..., making $\mathbf{T}$ ω-inconsistent. □

§18.3 Consistent Extensions of Q are Undecidable

Remember that a theory is *consistent* if it does not prove both α and $\neg\alpha$ for any formula α. Since anything follows from a contradiction, an inconsistent theory is trivial: every sentence is provable. Clearly, if a theory if ω-consistent, then it is consistent. But being consistent is a weaker requirement (i.e., there are theories that are consistent but not ω-consistent.). We can weaken the assumption in Definition 182A to simple consistency to obtain a stronger theorem.

Lemma 183A. *There is no "universal computable relation." That is, there is no binary computable relation $R(x, y)$, with the following property: whenever $S(y)$ is a unary computable relation, there is some k such that for every y, $S(y)$ is true if and only if $R(k, y)$ is true.*

Proof. Suppose $R(x, y)$ is a universal computable relation. Let $S(y)$ be the relation $\neg R(y, y)$. Since $S(y)$ is computable, for some k, $S(y)$ is equivalent to $R(k, y)$. But then we have that $S(k)$ is equivalent to both $R(k, k)$ and $\neg R(k, k)$, which is a contradiction. □

Theorem 183B. *Let* **T** *be any consistent theory that includes* **Q**. *Then* **T** *is not decidable.*

Proof. Suppose **T** is a consistent, decidable extension of **Q**. We will obtain a contradiction by using **T** to define a universal computable relation.

Let $R(x, y)$ hold if and only if

> x codes a formula $\delta(u)$, and **T** proves $\delta(\overline{y})$.

Since we are assuming that **T** is decidable, R is computable. Let us show that R is universal. If $S(y)$ is any computable relation, then it is representable in **Q** (and hence **T**) by a formula $\delta_S(u)$. Then for every n, we have

$$\begin{aligned} S(\overline{n}) &\rightarrow T \vdash \delta_S(\overline{n}) \\ &\rightarrow R(^{\#}\delta_S(u)^{\#}, n) \end{aligned}$$

and

$$\begin{aligned} \neg S(\overline{n}) &\rightarrow T \vdash \neg\delta_S(\overline{n}) \\ &\rightarrow T \nvdash \delta_S(\overline{n}) \quad \text{(since } \mathbf{T} \text{ is consistent)} \\ &\rightarrow \neg R(^{\#}\delta_S(u)^{\#}, n). \end{aligned}$$

That is, for every y, $S(y)$ is true if and only if $R(^{\#}\delta_S(u)^{\#}, y)$ is. So R is universal, and we have the contradiction we were looking for. □

Let "true arithmetic" be the theory $\{\alpha : \models_{\mathbb{MNSQ}\mathfrak{Q}N} \alpha\}$, that is, the set of sentences in the language of arithmetic that are true in the standard interpretation.

Corollary 183C. *True arithmetic is not decidable.*

§18.4 Axiomatizable Theories

A theory **T** is said to be *axiomatizable* if it has a computable set of axioms A. (Saying that A is a set of axioms for **T** means $T = \{\alpha : A \vdash \alpha\}$.) Any "reasonable" axiomatization of the natural numbers will have this property. In particular, any theory with a finite set of axioms is axiomatizable.

Lemma 184A. *Suppose* **T** *is axiomatizable. Then* **T** *is computably enumerable.*

Proof. Suppose A is a computable set of axioms for **T**. To determine if $\alpha \in T$, just search for a derivation of α from the axioms.

Put slightly differently, α is in **T** if and only if there is a finite list of axioms $\beta_1, \ldots, \beta_k$ in A and a derivation of $(\beta_1 \wedge \cdots \wedge \beta_k) \to \alpha$ in first-order logic. But we already know that any set with a definition of the form "there exists ... such that ..." is c.e., provided the second "..." is computable. □

§18.5 Axiomatizable Complete Theories are Decidable

A theory is said to be *complete* if for every sentence α, either α or $\neg\alpha$ is provable.

Lemma 185A. *Suppose a theory* **T** *is complete and axiomatizable. Then* **T** *is decidable.*

Proof. Suppose **T** is complete and A is a computable set of axioms. If **T** is inconsistent, it is clearly computable. (Algorithm: "just say yes.") So we can assume that **T** is also consistent.

To decide whether or not a sentence α is in **T**, simultaneously search for a derivation of α from **T** and a derivation of $\neg\alpha$. Since **T** is complete, you are bound to find one or the other; and since **T** is consistent, if you find a derivation of $\neg\alpha$, there is no derivation of α.

Put in different terms, we already know that **T** is c.e.; so by a theorem we proved before, it suffices to show that the complement of **T** is c.e. also. But a wff α is in $\bar{\mathbf{T}}$ if and only if $\neg\alpha$ is in **T**; so $\bar{\mathbf{T}} \leq_m \mathbf{T}$. □

§18.6 Q has no Complete, Consistent, Axiomatizable Extensions

Theorem 186A. *There is no complete, consistent, axiomatizable extension of* **Q**.

Proof. We already know that there is no consistent, decidable extension of **Q**. But if **T** is complete and axiomatized, then it is decidable. □

This theorems is not that far from Gödel's original 1931 formulation of the First Incompleteness Theorem. Aside from the more modern terminology, the key differences are this: Gödel has "ω-consistent" instead of "consistent"; and he could not say "axiomatizable" in full generality, since the formal notion of computability was not in place yet. (The formal models of computability were developed over the following decade, including by Gödel, and in large part to be able to characterize the kinds of theories that are susceptible to the Gödel phenomenon.)

The theorem says you can't have it all, namely, completeness, consistency, and axiomatizability. If you give up any one of these, though, you can have the other two: **Q** is consistent and computably axiomatized, but not complete; the inconsistent theory is complete, and computably axiomatized (say, by $\{0 \neq 0\}$), but not consistent; and the set of true sentence of arithmetic is complete and consistent, but it is not computably axiomatized.

§18.7 Sentences Provable and Refutable in Q are Computably Inseparable

Let $\bar{\mathbf{Q}}$ be the set of sentences whose *negations* are provable in **Q**, i.e., $\bar{\mathbf{Q}} = \{\alpha : \mathbf{Q} \vdash \neg\alpha\}$. Remember that disjoint sets A and B are said to be computably inseparable if there is no computable set C such that $A \subseteq C$ and $B \subseteq \overline{C}$.

Lemma 187A. **Q** *and* $\bar{\mathbf{Q}}$ *are computably inseparable.*

Proof. Suppose C is a computable set such that $\mathbf{Q} \subseteq C$ and $\bar{\mathbf{Q}} \subseteq \overline{C}$. Let $R(x, y)$ be the relation

> x codes a formula $\delta(u)$ and $\delta(\overline{y})$ is in C.

We will show that $R(x, y)$ is a universal computable relation, yielding a contradiction.

Suppose $S(y)$ is computable, represented by $\delta_S(u)$ in **Q**. Then

$$\begin{aligned} S(\overline{n}) &\rightarrow \mathbf{Q} \vdash \delta_S(\overline{n}) \\ &\rightarrow \delta_S(\overline{n}) \in C \end{aligned}$$

and

$$\begin{aligned} \neg S(\overline{n}) &\rightarrow \mathbf{Q} \vdash \neg\delta_S(\overline{n}) \\ &\rightarrow \delta_S(\overline{n}) \in \bar{\mathbf{Q}} \\ &\rightarrow \delta_S(\overline{n}) \notin C \end{aligned}$$

So $S(y)$ is equivalent to $R(\#(\delta_S(\overline{u})), y)$. □

§18.8 Theories Consistent with Q are Undecidable

The following theorem says that not only is **Q** undecidable, but, in fact, any theory that does not disagree with **Q** is undecidable.

Theorem 188A. *Let* **T** *be any theory in the language of arithmetic that is consistent with* **Q** *(i.e.,* $\mathbf{T} \cup \mathbf{Q}$ *is consistent). Then* **T** *is undecidable.*

Proof. Remember that **Q** has a finite set of axioms, $Q_1, \ldots, Q_8$. We can even replace these by a single axiom, $\varphi = Q_1 \wedge \cdots \wedge Q_8$.

Suppose **T** is a decidable theory consistent with **Q**. Let

$$C = \{\alpha : \mathbf{T} \vdash \varphi \to \alpha\}.$$

We show that C would be a computable separation of **Q** and $\bar{\mathbf{Q}}$, a contradiction. First, if α is in **Q**, then α is provable from the axioms of **Q**; by the deduction theorem, there is a derivation of $\varphi \to \alpha$ in first-order logic. So α is in C.

On the other hand, if α is in $\bar{\mathbf{Q}}$, then there is a proof of $\varphi \to \neg\alpha$ in first-order logic. If **T** also proves $\varphi \to \alpha$, then **T** proves $\neg\varphi$, in which case $\mathbf{T} \cup \mathbf{Q}$ is inconsistent. But we are assuming $\mathbf{T} \cup \mathbf{Q}$ is consistent, so **T** does not prove $\varphi \to \alpha$, and so α is not in C.

We've shown that if α is in **Q**, then it is in C, and if α is in $\bar{\mathbf{Q}}$, then it is in $\overline{C}$. So C is a computable separation, which is the contradiction we were looking for. □

This theorem is very powerful. For example, it implies:

Corollary 188B. *First-order logic for the language of arithmetic (that is, the set* $\{\alpha : \alpha$ *is provable in first-order logic*$\}$*) is undecidable.*

Proof. First-order logic is the set of consequences of $\emptyset$, which is consistent with **Q**. □

§18.9 Theories in which Q is Interpretable are Undecidable

We can strengthen these results even more. Informally, an interpretation of a language $\mathcal{L}_1$ in another language $\mathcal{L}_2$ involves defining the universe, relation symbols, and function symbols of $\mathcal{L}_1$ with wffs in $\mathcal{L}_2$. Though we won't take the time to do this, one can make this definition precise.

Theorem 189A. *Suppose* **T** *is a theory in a language in which one can interpret the language of arithmetic, in such a way that* **T** *is consistent with the interpretation of* **Q**. *Then* **T** *is undecidable. If* **T** *proves the interpretation of the axioms of* **Q**, *then no consistent extension of* **T** *is decidable.*

The proof is just a small modification of the proof of the last theorem; one could use a counterexample to get a separation of **Q** and $\bar{\mathbf{Q}}$. One can take **ZFC**, Zermelo-Fraenkel set theory with the axiom of choice, to be an axiomatic foundation that is powerful enough to carry out a good deal of ordinary mathematics. In **ZFC** one can define the natural numbers, and via this interpretation, the axioms of **Q** are true. So we have

Corollary 189B. *There is no decidable extension of* **ZFC**.

Corollary 189C. *There is no complete, consistent, computably axiomatizable extension of* **ZFC**.

The language of **ZFC** has only a single binary relation, $\in$. (In fact, you don't even need equality.) So we have

Corollary 189D. *First-order logic for any language with a binary relation symbol is undecidable.*

This result extends to any language with two unary function symbols, since one can use these to simulate a binary relation symbol. The results just cited are tight: it turns out that first-order logic for a language with only *unary* relation symbols and at most one *unary* function symbol is decidable.

One more bit of trivia. We know that the set of sentences in the language 0, $'$, $+$, $\times$, $<$ true in the standard model is undecidable. In fact, one can define $<$ in terms of the other symbols, and then one can define $+$ in terms of $\times$ and $'$. So the set of true sentences in the language 0, $'$, $\times$ is undecidable. On the other hand, Presburger has shown that the set of sentences in the language 0, $'$, $+$ true in the language of arithmetic is decidable. The procedure is computationally infeasible, however.

Chapter 19

Incompleteness and Provability

§19.0 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 derivable 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 derivability, Gödel was able to formalize a sentence which, in a roundabout way, asserts that it itself is not derivable. If that sentence were derivable, the theory would then be inconsistent. Gödel showed that the negation of that sentence is also not derivable from the system of axioms he was considering. (For this second part, Gödel had to assume that the theory **T** is what's called "ω-consistent." ω-Consistency is related to consistency, but is a stronger property.[1] A few years after Gödel,

[1] That is, any ω-consistent theory is consistent, but not vice versa.

Rosser showed that assuming simple consistency of **T** is enough.)

The first challenge is to understand how one can construct a sentence that refers to itself. For every wff α in the language of **Q**, let $\ulcorner\alpha\urcorner$ denote the numeral corresponding to ${}^{\#}\alpha^{\#}$. Think about what this means: α is a wff in the language of **Q**, ${}^{\#}\alpha^{\#}$ is a natural number, and $\ulcorner\alpha\urcorner$ is a *term* in the language of **Q**. So every wff α in the language of **Q** has a *name*, $\ulcorner\alpha\urcorner$, which is a term in the language of **Q**; this provides us with a conceptual framework in which wffs in the language of **Q** can "say" things about other wffs. The following lemma is known as the fixed-point lemma.

Lemma 190A. *Let* **T** *be any theory extending* **Q**, *and let* $\beta(x)$ *be any wff with only the variable* x *free. Then there is a sentence* α *such that* $\mathbf{T} \vdash \alpha \leftrightarrow \beta(\ulcorner\alpha\urcorner)$.

The lemma asserts that given any property $\beta(x)$, there is a sentence α that asserts "$\beta(x)$ is true of me," and **T** "knows" this.

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.

§19.1 The Fixed-Point Lemma

The fixed-point lemma says that for any wff $\beta(x)$, there is a sentence α such that $\mathbf{T} \vdash \alpha \leftrightarrow \beta(\ulcorner\alpha\urcorner)$, provided **T** extends **Q**. In the case of the liar sentence, we'd want α to be equivalent (provably in **T**) to "$\ulcorner\alpha\urcorner$ is false," i.e., the statement that ${}^{\#}\alpha^{\#}$ 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 α 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 wff 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 $\varphi(x)$ is $\varphi(\overline{n})$, where $n = {}^{\#}\varphi(x)^{\#}$. (From now on, let's abbreviate $\overline{{}^{\#}\varphi(x)^{\#}}$ as $\ulcorner\varphi(x)\urcorner$.) So if $\beta(x)$ 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 wff with Gödel number n, we could write $\varphi(x)$ as $\beta(\mathit{diag}(x))$. And Quine's version of the liar sentence would then be the diagonalization of it, i.e., $\varphi(\ulcorner\varphi(x)\urcorner)$ or $\beta(\mathit{diag}(\ulcorner\beta(\mathit{diag}(x))\urcorner))$. Of course, $\beta(x)$ could now be any other property, and the same construction would work. For the incompleteness theorem, we'll take $\beta(x)$ to be "x is not derivable in $\mathbf{T}$." Then $\varphi(x)$ would be "yields a sentence not derivable in $\mathbf{T}$ when applied to the Gödel number of its diagonalization."

To formalize this in $\mathbf{T}$, we have to find a way to formalize diag. The function $\mathrm{diag}(n)$ is computable, in fact, it is primitive recursive: if n is the Gödel number of a formula $\varphi(x)$, $\mathrm{diag}(n)$ returns the Gödel number of $\varphi(\ulcorner\varphi(x)\urcorner)$. (Recall, $\ulcorner\varphi(x)\urcorner$ is the standard numeral of the Gödel number of $\varphi(x)$, i.e., $\overline{{}^{\#}\varphi(x)^{\#}}$). If *diag* were a function symbol in $\mathbf{T}$ representing the function diag, we could take α to be the formula $\beta(\mathit{diag}(\ulcorner\beta(\mathit{diag}(x))\urcorner))$. Notice that

$$\begin{aligned}\mathrm{diag}({}^{\#}\beta(\mathit{diag}(x))^{\#}) &= {}^{\#}\beta(\mathit{diag}(\ulcorner\beta(\mathit{diag}(x))\urcorner))^{\#}\\ &= {}^{\#}\alpha^{\#}.\end{aligned}$$

Assuming $\mathbf{T}$ can derive

$$\mathit{diag}(\ulcorner\beta(\mathit{diag}(x))\urcorner) = \ulcorner\alpha\urcorner,$$

it can derive $\beta(\mathit{diag}(\ulcorner\beta(\mathit{diag}(x))\urcorner)) \leftrightarrow \beta(\ulcorner\alpha\urcorner)$. But the left hand side is, by definition, α.

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

Lemma 191A. *Let $\beta(x)$ be any formula with one free variable x. Then there is a sentence α such that $\mathbf{Q} \vdash \alpha \leftrightarrow \beta(\ulcorner\alpha\urcorner)$.*

Proof. Given $\beta(x)$, let $\varphi(x)$ be the formula $\exists y\,(\delta_{\mathrm{diag}}(x, y) \wedge \beta(y))$ and let α be its diagonalization, i.e., the formula $\varphi(\ulcorner\varphi(x)\urcorner)$.

Since δ_{diag} represents diag, and $\mathrm{diag}({}^{\#}\varphi(x)^{\#}) = {}^{\#}\alpha^{\#}$, $\mathbf{Q}$ can derive

$$\delta_{\mathrm{diag}}(\ulcorner\varphi(x)\urcorner, \ulcorner\alpha\urcorner) \tag{19.1}$$

$$\forall y\,(\delta_{\mathrm{diag}}(\ulcorner\varphi(x)\urcorner, y) \to y = \ulcorner\alpha\urcorner). \tag{19.2}$$

Now we show that $\mathbf{Q} \vdash \alpha \leftrightarrow \beta(\ulcorner\alpha\urcorner)$. We argue informally, using just logic and facts derivable in $\mathbf{Q}$.

First, suppose α, i.e., $\varphi(\ulcorner\varphi(x)\urcorner)$. Going back to the definition of $\varphi(x)$, we see that $\varphi(\ulcorner\varphi(x)\urcorner)$ just is

$$\exists y\,(\delta_{\text{diag}}(\ulcorner\varphi(x)\urcorner, y) \wedge \beta(y)).$$

Consider such a y. Since $\delta_{\text{diag}}(\ulcorner\varphi(x)\urcorner, y)$, by eq. (19.2), $y = \ulcorner\alpha\urcorner$. So, from $\beta(y)$ we have $\beta(\ulcorner\alpha\urcorner)$.

Now suppose $\beta(\ulcorner\alpha\urcorner)$. By eq. (19.1), we have

$$\delta_{\text{diag}}(\ulcorner\varphi(x)\urcorner, \ulcorner\alpha\urcorner) \wedge \beta(\ulcorner\alpha\urcorner).$$

It follows that

$$\exists y\,(\delta_{\text{diag}}(\ulcorner\varphi(x)\urcorner, y) \wedge \beta(y)).$$

But that's just $\varphi(\ulcorner\varphi(x)\urcorner)$, i.e., α. □

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.

§19.2 The First Incompleteness Theorem

We can now describe Gödel's original proof of the first incompleteness theorem. Let $\mathbf{T}$ be any computably axiomatized theory in a language extending the language of arithmetic, such that $\mathbf{T}$ includes the axioms of $\mathbf{Q}$. This means that, in particular, $\mathbf{T}$ 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 wff with Gödel number y in $\mathbf{T}$. 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 $\mathbf{T}$. 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 $\mathbf{T}$. 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.

Let $\mathbf{T}$ be an axiomatizable theory containing $\mathbf{Q}$. Then $\text{Prf}_T(x, y)$ is decidable, hence representable in $\mathbf{Q}$ by a wff $\mathsf{Prf}_T(x, y)$. Let $\mathsf{Prov}_T(y)$ be the formula

we described above. By the fixed-point lemma, there is a formula $\chi_{\mathbf{T}}$ such that $\mathbf{Q}$ (and hence $\mathbf{T}$) derives

$$\chi_{\mathbf{T}} \leftrightarrow \neg \mathsf{Prov}_T(\ulcorner \chi_{\mathbf{T}} \urcorner). \tag{19.3}$$

Note that $\chi_{\mathbf{T}}$ says, in essence, "$\chi_{\mathbf{T}}$ is not derivable in $\mathbf{T}$."

Lemma 192A. *If $\mathbf{T}$ is a consistent, axiomatizable theory extending $\mathbf{Q}$, then $\mathbf{T} \nvdash \chi_{\mathbf{T}}$.*

Proof. Suppose $\mathbf{T}$ derives $\chi_{\mathbf{T}}$. Then there *is* a derivation, and so, for some number m, the relation $\mathrm{Prf}_T(m, {}^{\#}\chi_{\mathbf{T}}{}^{\#})$ holds. But then $\mathbf{Q}$ derives the sentence $\mathsf{Prf}_T(\overline{m}, \ulcorner \chi_{\mathbf{T}} \urcorner)$. So $\mathbf{Q}$ derives $\exists x\, \mathsf{Prf}_T(x, \ulcorner \chi_{\mathbf{T}} \urcorner)$, which is, by definition, $\mathsf{Prov}_T(\ulcorner \chi_{\mathbf{T}} \urcorner)$. By eq. (19.3), $\mathbf{Q}$ derives $\neg \chi_{\mathbf{T}}$, and since $\mathbf{T}$ extends $\mathbf{Q}$, so does $\mathbf{T}$. We have shown that if $\mathbf{T}$ derives $\chi_{\mathbf{T}}$, then it also derives $\neg \chi_{\mathbf{T}}$, and hence it would be inconsistent. □

Definition 192B. *A theory $\mathbf{T}$ is* ω-consistent *if the following holds: if $\exists x\, \alpha(x)$ is any sentence and $\mathbf{T}$ derives $\neg\alpha(\overline{0})$, $\neg\alpha(\overline{1})$, $\neg\alpha(\overline{2})$, ... then $\mathbf{T}$ does not prove $\exists x\, \alpha(x)$.*

Note that every ω-consistent theory is also consistent. This follows simply from the fact that if $\mathbf{T}$ is inconsistent, then $\mathbf{T} \vdash \alpha$ for every α. In particular, if $\mathbf{T}$ is inconsistent, it derives both $\neg\alpha(\overline{n})$ for every n and also derives $\exists x\, \alpha(x)$. So, if $\mathbf{T}$ is inconsistent, it is ω-inconsistent. By contraposition, if $\mathbf{T}$ is ω-consistent, it must be consistent.

Lemma 192C. *If $\mathbf{T}$ is an ω-consistent, axiomatizable theory extending $\mathbf{Q}$, then $\mathbf{T} \nvdash \neg \chi_{\mathbf{T}}$.*

Proof. We show that if $\mathbf{T}$ derives $\neg \chi_{\mathbf{T}}$, then it is ω-inconsistent. Suppose $\mathbf{T}$ derives $\neg \chi_{\mathbf{T}}$. If $\mathbf{T}$ is inconsistent, it is ω-inconsistent, and we are done. Otherwise, $\mathbf{T}$ is consistent, so it does not derive $\chi_{\mathbf{T}}$ by Lemma 192A. Since there is no derivation of $\chi_{\mathbf{T}}$ in $\mathbf{T}$, $\mathbf{Q}$ derives

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

and so does $\mathbf{T}$. On the other hand, by eq. (19.3), $\neg \chi_{\mathbf{T}}$ is equivalent to $\exists x\, \mathsf{Prf}_T(x, \ulcorner \chi_{\mathbf{T}} \urcorner)$. So $\mathbf{T}$ is ω-inconsistent. □

Theorem 192D. *Let $\mathbf{T}$ be any ω-consistent, axiomatizable theory extending $\mathbf{Q}$. Then $\mathbf{T}$ is not complete.*

Proof. If $\mathbf{T}$ is ω-consistent, it is consistent, so $\mathbf{T} \nvdash \chi_{\mathbf{T}}$ by Lemma 192A. By Lemma 192C, $\mathbf{T} \nvdash \neg \chi_{\mathbf{T}}$. This means that $\mathbf{T}$ is incomplete, since it derives neither $\chi_{\mathbf{T}}$ nor $\neg \chi_{\mathbf{T}}$. □

§19.3 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" derivability predicate $\mathsf{RProv}_T(y)$ instead of $\mathsf{Prov}_T(y)$.

Theorem 193A. *Let* $\mathbf{T}$ *be any consistent, axiomatizable theory extending* $\mathbf{Q}$. *Then* $\mathbf{T}$ *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 α, $\mathrm{not}(x)$ is a code of $\neg\alpha$. Then $\mathrm{Ref}_T(x, y)$ holds iff $\mathrm{Prf}_T(x, \mathrm{not}(y))$. Let $\mathsf{Ref}_T(x, y)$ represent it. Then, if $\mathbf{T} \vdash \neg\alpha$ and δ is a corresponding derivation, $\mathbf{Q} \vdash \mathsf{Ref}_T(\ulcorner\delta\urcorner, \ulcorner\alpha\urcorner)$. 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." 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! ($\mathsf{RProv}_T(y)$ is often read as "y is Rosser provable." Since, as just discussed, Rosser provability is not some special kind of provability—in inconsistent theories, there are sentences that are provable but not Rosser provable—this may be confusing. To avoid the confusion, you could instead read it as "y is shmovable.")

By the fixed-point lemma, there is a formula $\rho_{\mathbf{T}}$ such that

$$\mathbf{Q} \vdash \rho_{\mathbf{T}} \leftrightarrow \neg\mathsf{RProv}_T(\ulcorner\rho_{\mathbf{T}}\urcorner). \tag{19.4}$$

In contrast to the proof of Theorem 192D, here we claim that if $\mathbf{T}$ is consistent, $\mathbf{T}$ doesn't derive $\rho_{\mathbf{T}}$, and $\mathbf{T}$ also doesn't derive $\neg\rho_{\mathbf{T}}$. (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 a derivation 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 176C, $\mathbf{Q} \vdash \forall z\, (z < \overline{n} \to \neg\mathsf{Ref}_T(z, \ulcorner\rho_T\urcorner))$. 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. (19.4), $\mathbf{Q} \vdash \neg\rho_T$. 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 derive 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 $\mathbf{Q}$ derives. 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 176C, (b) $\neg(x < \overline{n})$. By Lemma 176D, $\overline{n} < x$. 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, as required, that

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

§19.4 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 $\mathbf{P}$. 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 $\mathbf{Q}$. Gödel doesn't go so far to isolate a minimal set of axioms that suffice, but we now know that $\mathbf{Q}$ will do the trick. Finally, in Section 4, he sketches a proof of the second incompleteness theorem.

§19.5 The Derivability Conditions for PA

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

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

for every wff α. 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 derivability predicate $\mathrm{Prf}_{\mathbf{PA}}(x, y)$ is computable and hence represented in **Q** (and so, in **PA**). As before, we will take $\mathsf{Prf}_{\mathbf{PA}}(x, y)$ to denote the formula representing the relation. Let $\mathsf{Prov}_{\mathbf{PA}}(y)$ be the formula $\exists x\,\mathrm{Prf}_{\mathbf{PA}}(x, y)$, which, intuitively says, "y is derivable 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 derive a few basic facts about this derivability predicate. In fact, what we need are the following facts:

P1. If $\mathbf{PA} \vdash \alpha$, then $\mathbf{PA} \vdash \mathsf{Prov}_{\mathbf{PA}}(\ulcorner\alpha\urcorner)$.

P2. For all wffs α and β,

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

P3. For every wff α,

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

The only way to verify that these three properties hold is to describe the wff $\mathsf{Prov}_{\mathbf{PA}}(y)$ carefully and use the axioms of **PA** to describe the relevant formal derivations. 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 $\mathbf{PA} \vdash \mathsf{Prov}_{\mathbf{PA}}(\ulcorner\alpha\urcorner)$, then $\mathbf{PA} \vdash \alpha$.

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.)

§19.6 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 $\mathbf{PA} \vdash 0 = 1$. So we can take the consistency statement $\mathsf{Con}_{\mathbf{PA}}$ to be the sentence $\neg\mathsf{Prov}_{\mathbf{PA}}(\ulcorner 0 = 1 \urcorner)$, and then the following theorem does the job:

Theorem 196A. *Assuming* **PA** *is consistent, then* **PA** *does not derive* $\mathsf{Con}_{\mathbf{PA}}$.

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 $\mathsf{Prov}_{\mathbf{PA}}(y)$ satisfies the three derivability conditions, so the theorem generalizes to any theory with a derivability 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 $\chi_{\mathbf{PA}}$ be the Gödel sentence that we constructed in the proof of Theorem 192D. We have shown "If **PA** is consistent, then **PA** does not derive $\chi_{\mathbf{PA}}$." If we formalize this *in* **PA**, we have a proof of

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

Now suppose **PA** derives $\mathsf{Con}_{\mathbf{PA}}$. Then it derives $\neg\mathrm{Prov}_{\mathbf{PA}}(\ulcorner \chi_{\mathbf{PA}} \urcorner)$. But since $\chi_{\mathbf{PA}}$ is a Gödel sentence, this is equivalent to $\chi_{\mathbf{PA}}$. So **PA** derives $\chi_{\mathbf{PA}}$.

But: we know that if **PA** is consistent, it doesn't derive $\chi_{\mathbf{PA}}$! So if **PA** is consistent, it can't derive $\mathsf{Con}_{\mathbf{PA}}$.

To make the argument more precise, we will let $\chi_{\mathbf{PA}}$ be the Gödel sentence for **PA** and use the derivability conditions (P1)–(P3) to show that **PA** derives $\mathsf{Con}_{\mathbf{PA}} \to \chi_{\mathbf{PA}}$. This will show that **PA** doesn't derive $\mathsf{Con}_{\mathbf{PA}}$. Here is a sketch

of the proof, in **PA**. (For simplicity, we drop the **PA** subscripts.)

$$\chi \leftrightarrow \neg \mathsf{Prov}(\ulcorner \chi \urcorner) \tag{19.5}$$

χ is a Gödel sentence

$$\chi \rightarrow \neg \mathsf{Prov}(\ulcorner \chi \urcorner) \tag{19.6}$$

from eq. (19.5)

$$\chi \rightarrow (\mathsf{Prov}(\ulcorner \chi \urcorner) \rightarrow \bot) \tag{19.7}$$

from eq. (19.6) by logic

$$\mathsf{Prov}(\ulcorner \chi \rightarrow (\mathsf{Prov}(\ulcorner \chi \urcorner) \rightarrow \bot) \urcorner) \tag{19.8}$$

by from eq. (19.7) by condition P1

$$\mathsf{Prov}(\ulcorner \chi \urcorner) \rightarrow \mathsf{Prov}(\ulcorner (\mathsf{Prov}(\ulcorner \chi \urcorner) \rightarrow \bot) \urcorner) \tag{19.9}$$

from eq. (19.8) by condition P2

$$\mathsf{Prov}(\ulcorner \chi \urcorner) \rightarrow (\mathsf{Prov}(\ulcorner \mathsf{Prov}(\ulcorner \chi \urcorner) \urcorner) \rightarrow \mathsf{Prov}(\ulcorner \bot \urcorner)) \tag{19.10}$$

from eq. (19.9) by condition P2 and logic

$$\mathsf{Prov}(\ulcorner \chi \urcorner) \rightarrow \mathsf{Prov}(\ulcorner \mathsf{Prov}(\ulcorner \chi \urcorner) \urcorner) \tag{19.11}$$

by P3

$$\mathsf{Prov}(\ulcorner \chi \urcorner) \rightarrow \mathsf{Prov}(\ulcorner \bot \urcorner) \tag{19.12}$$

from eq. (19.10) and eq. (19.11) by logic

$$\mathsf{Con} \rightarrow \neg \mathsf{Prov}(\ulcorner \chi \urcorner) \tag{19.13}$$

contraposition of eq. (19.12) and $\mathsf{Con} \equiv \neg \mathsf{Prov}(\ulcorner \bot \urcorner)$

$\mathsf{Con} \rightarrow \chi$

from eq. (19.5) and eq. (19.13) by logic

The use of logic in the above just elementary facts from propositional logic, e.g., eq. (19.7) uses $\vdash \neg\alpha \leftrightarrow (\alpha \rightarrow \bot)$ and eq. (19.12) uses $\alpha \rightarrow (\beta \rightarrow \gamma), \alpha \rightarrow \beta \vdash \alpha \rightarrow \gamma$. The use of condition P2 in eq. (19.9) and eq. (19.10) relies on instances of P2, $\mathsf{Prov}(\ulcorner \alpha \rightarrow \beta \urcorner) \rightarrow (\mathsf{Prov}(\ulcorner \alpha \urcorner) \rightarrow \mathsf{Prov}(\ulcorner \beta \urcorner))$. In the first one, $\alpha \equiv \chi$ and $\beta \equiv \mathsf{Prov}(\ulcorner \chi \urcorner) \rightarrow \bot$; in the second, $\alpha \equiv \mathsf{Prov}(\ulcorner G \urcorner)$ and $\beta \equiv \bot$.

The more abstract version of the second incompleteness theorem is as follows:

Theorem 196B. *Let* **T** *be any consistent, axiomatized theory extending* **Q** *and let* $\mathsf{Prov}_T(y)$ *be any formula satisfying derivability conditions P1–P3 for* **T**. *Then* **T** *does not derive* Con_T.

The moral of the story is that no "reasonable" consistent theory for mathematics can derive its own consistency statement. 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 derive the consistency statement of **T**, and so, a fortiori, the finitary fragment can't derive the consistency statement 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**, Zermelo-Fraenkel set theory with the axiom of choice.

§19.7 Löb's Theorem

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

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

It is not derivable, because if $\mathbf{T} \vdash \chi$, (a) by derivability condition (1), $\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\chi\urcorner)$, and (b) $\mathbf{T} \vdash \chi$ together with $\mathbf{T} \vdash \neg\mathsf{Prov}_T(\ulcorner\chi\urcorner) \leftrightarrow \chi$ gives $\mathbf{T} \vdash \neg\mathsf{Prov}_T(\ulcorner\chi\urcorner)$, and so $\mathbf{T}$ would be inconsistent. Now it is natural to ask about the status of a fixed point of $\mathsf{Prov}_T(y)$, i.e., a sentence θ such that

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

If it were derivable, $\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner\theta\urcorner)$ 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* derive θ.

We can make headway on this question if we generalize it a bit. The left-to-right direction of the fixed point equivalence, $\mathsf{Prov}_T(\ulcorner\theta\urcorner) \to \theta$, is an instance of a general schema called a *reflection principle*: $\mathsf{Prov}_T(\ulcorner\alpha\urcorner) \to \alpha$. It is called that because it expresses, in a sense, that $\mathbf{T}$ can "reflect" about what it can derive; basically it says, "If $\mathbf{T}$ can derive α, then α is true," for any α. This is true for sound theories only, of course, and this suggests that theories will in general not derive every instance of it. So which instances can a theory (strong enough, and satisfying the derivability conditions) derive? Certainly all those where α itself is derivable. And that's it, as the next result shows.

Theorem 197A. *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 19.6. If $\mathbf{T}$ derives $\mathsf{Prov}_T(\ulcorner\alpha\urcorner) \to \alpha$, then in fact $\mathbf{T}$ derives α.*

Put differently, if $\mathbf{T} \nvdash \alpha$, then $\mathbf{T} \nvdash \mathsf{Prov}_T(\ulcorner\alpha\urcorner) \to \alpha$. 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 derivable," and "Santa Claus exists" with α, yields the proof of Löb's theorem. The trick is to apply the fixed-point lemma to the wff $\mathsf{Prov}_T(y) \to \alpha$. The fixed point of that corresponds to the sentence X in the preceding sketch.

Proof of Theorem 197A. Suppose α is a sentence such that $\mathbf{T}$ derives $\mathsf{Prov}_T(\ulcorner \alpha \urcorner) \to \alpha$. Let $\beta(y)$ be the wff $\mathsf{Prov}_T(y) \to \alpha$, and use the fixed-point lemma to find a sentence δ such that $\mathbf{T}$ derives $\delta \leftrightarrow \beta(\ulcorner \delta \urcorner)$. Then each of the following is

derivable in $\mathbf{T}$:

$$\delta \leftrightarrow (\mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \alpha) \tag{19.14}$$

δ is a fixed point of $\beta(y)$

$$\delta \to (\mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \alpha) \tag{19.15}$$

from eq. (19.14)

$$\mathsf{Prov}_T(\ulcorner \delta \to (\mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \alpha) \urcorner) \tag{19.16}$$

from eq. (19.15) by condition P1

$$\mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \mathsf{Prov}_T(\ulcorner \mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \alpha \urcorner) \tag{19.17}$$

from eq. (19.16) using condition P2

$$\mathsf{Prov}_T(\ulcorner \delta \urcorner) \to (\mathsf{Prov}_T(\ulcorner \mathsf{Prov}_T(\ulcorner \delta \urcorner) \urcorner) \to \mathsf{Prov}_T(\ulcorner \alpha \urcorner)) \tag{19.18}$$

from eq. (19.17) using P2 again

$$\mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \mathsf{Prov}_T(\ulcorner \mathsf{Prov}_T(\ulcorner \delta \urcorner) \urcorner) \tag{19.19}$$

by derivability condition P3

$$\mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \mathsf{Prov}_T(\ulcorner \alpha \urcorner) \tag{19.20}$$

from eq. (19.18) and eq. (19.19)

$$\mathsf{Prov}_T(\ulcorner \alpha \urcorner) \to \alpha \tag{19.21}$$

by assumption of the theorem

$$\mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \alpha \tag{19.22}$$

from eq. (19.20) and eq. (19.21)

$$(\mathsf{Prov}_T(\ulcorner \delta \urcorner) \to \alpha) \to \delta \tag{19.23}$$

from eq. (19.14)

$$\delta \tag{19.24}$$

from eq. (19.22) and eq. (19.23)

$$\mathsf{Prov}_T(\ulcorner \delta \urcorner) \tag{19.25}$$

from eq. (19.24) by condition P1

α from eq. (19.21) and eq. (19.25) □

With Löb's theorem in hand, there is a short proof of the second incompleteness theorem (for theories having a derivability predicate satisfying conditions P1–P3): if $\mathbf{T} \vdash \mathsf{Prov}_T(\ulcorner \bot \urcorner) \to \bot$, then $\mathbf{T} \vdash \bot$. If $\mathbf{T}$ is consistent, $\mathbf{T} \nvdash \bot$. So, $\mathbf{T} \nvdash \mathsf{Prov}_T(\ulcorner \bot \urcorner) \to \bot$, i.e., $\mathbf{T} \nvdash \mathsf{Con}_{\mathbf{T}}$. We can also apply it to show that θ, the fixed point of $\mathsf{Prov}_T(x)$, is derivable. For since

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

in particular

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

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

§19.8 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 $\mathfrak{B}$ be the structure with domain $\mathbb{N}$ and the standard interpretation for the symbols in the language of arithmetic. Then $\models_{\mathfrak{B}} \alpha$ means "α is true in the standard interpretation."

Definition 198A. *A relation $R(x_1, \ldots, x_k)$ of natural numbers is* definable *in $\mathfrak{B}$ if and only if there is a formula $\alpha(x_1, \ldots, x_k)$ in the language of arithmetic such that for every $n_1, \ldots, n_k$, $R(n_1, \ldots, n_k)$ if and only if $\models_{\mathfrak{B}} \alpha(\overline{n}_1, \ldots, \overline{n}_k)$.*

Put differently, a relation is definable in $\mathfrak{B}$ if and only if it is representable in the theory **TA**, where $\mathbf{TA} = \{\alpha : \models_{\mathfrak{B}} \alpha\}$ 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 198B. *Every computable relation is definable in $\mathfrak{B}$.*

Proof. It is easy to check that the formula representing a relation in **Q** defines the same relation in $\mathfrak{B}$. □

Now one can ask, is the converse also true? That is, is every relation definable in $\mathfrak{B}$ computable? The answer is no. For example:

Lemma 198C. *The halting relation is definable in $\mathfrak{B}$.*

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

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

Let δ_T define T in $\mathfrak{B}$. Then

$$H = \{\langle e, x\rangle : \models_{\mathfrak{B}} \exists s\, \delta_T(\overline{e}, \overline{x}, s)\},$$

so $\exists s\, \delta_T(z, x, s)$ defines H in $\mathfrak{B}$. □

What about **TA** itself? Is it definable in arithmetic? That is: is the set $\{{}^{\ulcorner}\alpha^{\urcorner} : \models_{\mathfrak{B}} \alpha\}$ definable in arithmetic? Tarski's theorem answers this in the negative.

Theorem 198D. *The set of true sentences of arithmetic is not definable in arithmetic.*

Proof. Suppose $\delta(x)$ defined it, i.e., $\models_{\mathfrak{B}} \alpha$ iff $\models_{\mathfrak{B}} \delta(\ulcorner\alpha\urcorner)$. By the fixed-point lemma, there is a formula α such that $\mathbf{Q} \vdash \alpha \leftrightarrow \neg\delta(\ulcorner\alpha\urcorner)$, and hence $\models_{\mathfrak{B}} \alpha \leftrightarrow \neg\delta(\ulcorner\alpha\urcorner)$. But then $\models_{\mathfrak{B}} \alpha$ if and only if $\models_{\mathfrak{B}} \neg\delta(\ulcorner\alpha\urcorner)$, which contradicts the fact that $\delta(y)$ 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

Problem 1. A wff $\alpha(x)$ is a *truth definition* if $\mathbf{Q} \vdash \beta \leftrightarrow \alpha(\ulcorner\beta\urcorner)$ for all sentences β. Show that no wff is a truth definition by using the fixed-point lemma.

Problem 2. Every ω-consistent theory is consistent. Show that the converse does not hold, i.e., that there are consistent but ω-inconsistent theories. Do this by showing that $\mathbf{Q} \cup \{\neg\chi_{\mathbf{Q}}\}$ is consistent but ω-inconsistent.

Problem 3. Two sets A and B of natural numbers are said to be *computably inseparable* if there is no decidable set X such that $A \subseteq X$ and $B \subseteq \overline{X}$ ($\overline{X}$ is the complement, $\mathbb{N} \setminus X$, of X). Let $\mathbf{T}$ be a consistent axiomatizable extension of $\mathbf{Q}$. Suppose A is the set of Gödel numbers of sentences provable in $\mathbf{T}$ and B the set of Gödel numbers of sentences refutable in $\mathbf{T}$. Prove that A and B are computably inseparable.

Problem 4. Show that $\mathbf{PA}$ derives $\chi_{\mathbf{PA}} \to \mathsf{Con}_{\mathbf{PA}}$.

Problem 5. Let $\mathbf{T}$ be a computably axiomatized theory, and let Prov_T be a derivability predicate for $\mathbf{T}$. Consider the following four statements:

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

Under what conditions are each of these statements true?

Problem 6. Show that $Q(n) \Leftrightarrow n \in \{^{\#}\alpha^{\#} : \mathbf{Q} \vdash \alpha\}$ is definable in arithmetic.