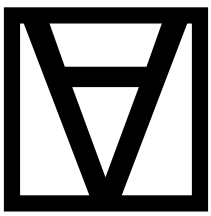
Logical Theory

June 18, 2020



This version of the text was compiled on June 18, 2020. Please check the Canvas activity of the course for the most recent version. If you find typos, errors or have suggestions for improvement please contact your course instructor.

Contents

Part I

Propositional Logic

Chapter 1

Syntax and Semantics

1.1 Introduction

Propositional logic deals with formulas 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 formula is determined, so is the truth value of any formulas 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 formula 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 formulas from propositional variables using the connectives. Alternative definitions are possible. 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 formulas *inductively*. If done properly, every expression can result essentially in only one way according to the formation rules. The inductive definition resulting in expressions that are

uniquely readable means we can give meanings to these expressions using the same method—inductive definition.

Giving the meaning of expressions is the domain of semantics. The central concept in semantics for propositonal logic is that of satisfaction in a valuation. A valuation v assigns truth values $\mathbb{T}$, $\mathbb{F}$ to the propositional variables. Any valuation determines a truth value $\overline{v}(\varphi)$ for any formula φ. A formula is satisfied in a valuation v iff $\overline{v}(\varphi) = \mathbb{T}$—we write this as $v \vDash \varphi$. This relation can also be defined by induction on the structure of φ, using the truth functions for the logical connectives to define, say, satisfaction of $\varphi \wedge \psi$ in terms of satisfaction (or not) of φ and ψ.

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

1.2 Propositional Formulas

Formulas of propositional logic are built up from *propositional variables* and the propositional constant $\bot$ using *logical connectives*.

1. A countably infinite set At_0 of propositional variables $p_0, p_1, \ldots$
2. The propositional constant for falsity $\bot$.
3. The logical connectives: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (conditional)
4. 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: $\leftrightarrow$ (biconditional), $\top$ (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 1.1 (Formula). The set $\mathrm{Frm}(\mathcal{L}_0)$ of *formulas* of propositional logic is defined inductively as follows:

1. $\bot$ is an atomic formula.
2. Every propositional variable p_i is an atomic formula.
3. If φ is a formula, then $\neg\varphi$ is formula.
4. If φ and ψ are formulas, then $(\varphi \wedge \psi)$ is a formula.
5. If φ and ψ are formulas, then $(\varphi \vee \psi)$ is a formula.
6. If φ and ψ are formulas, then $(\varphi \to \psi)$ is a formula.
7. Nothing else is a formula.

The definition of formulas is an *inductive definition*. Essentially, we construct the set of formulas in infinitely many stages. In the initial stage, we pronounce all atomic formulas to be formulas; this corresponds to the first few cases of the definition, i.e., the cases for $\bot$, p_i. "Atomic formula" thus means any formula of this form.

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

Definition 1.2. Formulas constructed using the defined operators are to be understood as follows:

1. $\top$ abbreviates $\neg\bot$.
2. $\varphi \leftrightarrow \psi$ abbreviates $(\varphi \to \psi) \wedge (\psi \to \varphi)$.

Definition 1.3 (Syntactic identity). The symbol $\equiv$ expresses syntactic identity between strings of symbols, i.e., $\varphi \equiv \psi$ iff φ and ψ are strings of symbols of the same length and which contain the same symbol in each place.

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

1.3 Preliminaries

Theorem 1.4 (*Principle of induction on formulas*). *If some property P holds for all the atomic formulas and is such that*

1. *it holds for $\neg\varphi$ whenever it holds for φ;*
2. *it holds for $(\varphi \wedge \psi)$ whenever it holds for φ and ψ;*
3. *it holds for $(\varphi \vee \psi)$ whenever it holds for φ and ψ;*
4. *it holds for $(\varphi \to \psi)$ whenever it holds for φ and ψ;*

then P holds for all formulas.

Proof. Let S be the collection of all formulas with property P. Clearly $S \subseteq \mathrm{Frm}(\mathcal{L}_0)$. S satisfies all the conditions of Definition 1.1: it contains all atomic formulas 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 1.5. *Any formula in* $\mathrm{Frm}(\mathcal{L}_0)$ *is* balanced*, in that it has as many left parentheses as right ones.*

Proposition 1.6. *No proper initial segment of a formula is a formula.*

Proposition 1.7 (Unique Readability). *Any formula φ in* $\mathrm{Frm}(\mathcal{L}_0)$ *has exactly one parsing as one of the following*

1. $\bot$.
2. p_n *for some* $p_n \in \mathrm{At}_0$.
3. $\neg\psi$ *for some formula* ψ.
4. $(\psi \wedge \chi)$ *for some formulas* ψ *and* χ.
5. $(\psi \vee \chi)$ *for some formulas* ψ *and* χ.
6. $(\psi \to \chi)$ *for some formulas* ψ *and* χ.

Moreover, this parsing is unique.

Proof. By induction on φ. For instance, suppose that φ has two distinct readings as $(\psi \to \chi)$ and $(\psi' \to \chi')$. Then ψ and ψ' must be the same (or else one would be a proper initial segment of the other and that's not possible by Proposition 1.6); so if the two readings of φ are distinct it must be because χ and χ' are distinct readings of the same sequence of symbols, which is impossible by the inductive hypothesis. □

It may be worth pointing out that the unique readability is not something we get for free for any inductively defined system. For example, if in the definition of $\mathrm{Frm}(\mathcal{L}_0)$ we hadn't used parantheses the "formula" $\varphi \wedge \psi \vee \chi$ would have two different parsings corresponding to $(\varphi \wedge \psi) \vee \chi$ and $\varphi \wedge (\psi \vee \chi)$.

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

Definition 1.8 (Immediate Subformula). If φ is a formula, the *immediate subformulas* of φ are defined inductively as follows:

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

Definition 1.9 (Proper Subformula). If φ is a formula, the *proper subformulas* of φ are recursively as follows:

1. Atomic formulas have no proper subformulas.

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

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

Definition 1.10 (Subformula). The subformulas of φ are φ itself together with all its proper subformulas.

The *main connective* of a formula is the outermost connective of the formula. We can now define what the *scope* of a connective is.

Definition 1.11 (Scope). The scope of a connective in a formula is the subformula for which the connective is the main connective.

Definition 1.12 (Uniform Substitution). If φ and ψ are formulas, and p_i is a propositional variable, then $\varphi[\psi/p_i]$ denotes the result of replacing each occurrence of p_i by an occurrence of ψ in φ; similarly, the simultaneous substitution of $p_1, \ldots, p_n$ by formulas $\psi_1, \ldots, \psi_n$ is denoted by $\varphi[\psi_1/p_1, \ldots, \psi_n/p_n]$.

1.4 Valuations and Satisfaction

Definition 1.13 (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 v assigning either $\mathbb{T}$ or $\mathbb{F}$ to the propositional variables of the language, i.e., $v\colon \mathrm{At}_0 \to \{\mathbb{T}, \mathbb{F}\}$.

Definition 1.14. Given a valuation v, define the evaluation function $\overline{v}\colon \mathrm{Frm}(\mathcal{L}_0) \to \{\mathbb{T}, \mathbb{F}\}$ inductively by:

$$
\begin{aligned}
\overline{v}(\bot) &= \mathbb{F};\\
\overline{v}(p_n) &= v(p_n);\\
\overline{v}(\neg\varphi) &= \begin{cases} \mathbb{T} & \text{if } \overline{v}(\varphi) = \mathbb{F};\\ \mathbb{F} & \text{otherwise.}\end{cases}\\
\overline{v}(\varphi \wedge \psi) &= \begin{cases} \mathbb{T} & \text{if } \overline{v}(\varphi) = \mathbb{T} \text{ and } \overline{v}(\psi) = \mathbb{T};\\ \mathbb{F} & \text{if } \overline{v}(\varphi) = \mathbb{F} \text{ or } \overline{v}(\psi) = \mathbb{F}.\end{cases}\\
\overline{v}(\varphi \vee \psi) &= \begin{cases} \mathbb{T} & \text{if } \overline{v}(\varphi) = \mathbb{T} \text{ or } \overline{v}(\psi) = \mathbb{T};\\ \mathbb{F} & \text{if } \overline{v}(\varphi) = \mathbb{F} \text{ and } \overline{v}(\psi) = \mathbb{F}.\end{cases}\\
\overline{v}(\varphi \to \psi) &= \begin{cases} \mathbb{T} & \text{if } \overline{v}(\varphi) = \mathbb{F} \text{ or } \overline{v}(\psi) = \mathbb{T};\\ \mathbb{F} & \text{if } \overline{v}(\varphi) = \mathbb{T} \text{ and } \overline{v}(\psi) = \mathbb{F}.\end{cases}
\end{aligned}
$$

The clauses correspond to the following truth tables:

φ	$\neg\varphi$
$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$

φ	ψ	$\varphi \wedge \psi$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$

φ	ψ	$\varphi \vee \psi$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$

φ	ψ	$\varphi \rightarrow \psi$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$

Theorem 1.15 (Local Determination). *Suppose that v_1 and v_2 are valuations that agree on the propositional letters occurring in φ, i.e., $v_1(p_n) = v_2(p_n)$ whenever p_n occurs in some formula φ. Then $\overline{v_1}$ and $\overline{v_2}$ also agree on φ, i.e., $\overline{v_1}(\varphi) = \overline{v_2}(\varphi)$.*

Proof. By induction on φ. □

Definition 1.16 (Satisfaction). Using the evaluation function, we can define the notion of *satisfaction of a formula φ by a valuation v*, $v \vDash \varphi$, inductively as follows. (We write $v \nvDash \varphi$ to mean "not $v \vDash \varphi$.")

1. $\varphi \equiv \bot$: $v \nvDash \varphi$.
2. $\varphi \equiv p_i$: $v \vDash \varphi$ iff $v(p_i) = \mathbb{T}$.
3. $\varphi \equiv \neg\psi$: $v \vDash \varphi$ iff $v \nvDash \psi$.
4. $\varphi \equiv (\psi \wedge \chi)$: $v \vDash \varphi$ iff $v \vDash \psi$ and $v \vDash \chi$.
5. $\varphi \equiv (\psi \vee \chi)$: $v \vDash \varphi$ iff $v \vDash \varphi$ or $v \vDash \psi$ (or both).
6. $\varphi \equiv (\psi \rightarrow \chi)$: $v \vDash \varphi$ iff $v \nvDash \psi$ or $v \vDash \chi$ (or both).

If Γ is a set of formulas, $v \vDash \Gamma$ iff $v \vDash \varphi$ for every $\varphi \in \Gamma$.

Proposition 1.17. *$v \vDash \varphi$ iff $\overline{v}(\varphi) = \mathbb{T}$.*

Proof. By induction on φ. □

1.5 Semantic Notions

We define the following semantic notions:

Definition 1.18.

1. A formula φ is *satisfiable* if for some v, $v \vDash \varphi$; it is *unsatisfiable* if for no v, $v \vDash \varphi$;
2. A formula φ is a *tautology* if $v \vDash \varphi$ for all valuations v;
3. A formula φ is *contingent* if it is satisfiable but not a tautology;
4. If Γ is a set of formulas, $\Gamma \vDash \varphi$ ("Γ entails φ") if and only if $v \vDash \varphi$ for every valuation v for which $v \vDash \Gamma$.

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

Proposition 1.19. *1. φ is a tautology if and only if $\emptyset \vDash \varphi$;*

2. If $\Gamma \vDash \varphi$ and $\Gamma \vDash \varphi \to \psi$ then $\Gamma \vDash \psi$;

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

4. Monotony: if $\Gamma \subseteq \Delta$ and $\Gamma \vDash \varphi$ then also $\Delta \vDash \varphi$;

5. Transitivity: if $\Gamma \vDash \varphi$ and $\Delta \cup \{\varphi\} \vDash \psi$ then $\Gamma \cup \Delta \vDash \psi$;

Proof. Exercise. □

Proposition 1.20. *$\Gamma \vDash \varphi$ if and only if $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable;*

Proof. Exercise. □

Theorem 1.21 (Semantic Deduction Theorem). *$\Gamma \vDash \varphi \to \psi$ if and only if $\Gamma \cup \{\varphi\} \vDash \psi$.*

Proof. Exercise. □

We write $\varphi \vDash \psi$ for $\Gamma \vDash \psi$ when $\Gamma = \{\varphi\}$ is a singleton and say that two formulas are semantically equivalent, $\varphi \approx \psi$, when $\varphi \vDash \psi$ and $\psi \vDash \varphi$, i.e., when $\overline{v}(\varphi) = \overline{v}(\psi)$ for all valuations v.

The following equivalences, known as the De Morgan laws, seem to indicate that the connectives $\wedge$ and $\vee$ behave in a similar, dual, way.

$$(\varphi \wedge \psi) \approx \neg(\neg\varphi \vee \neg\psi)$$
$$(\varphi \vee \psi) \approx \neg(\neg\varphi \wedge \neg\psi)$$

This symmetry, or duality, between conjunction and disjunction can be made precise, but first we define the dual of a formula.

Definition 1.22. The mapping that maps a formula with no occurrences of $\to$ nor $\leftrightarrow$ to its *dual* is defined by the following clauses:

- $\varphi^d \equiv \varphi$ when φ is atomic,
- $(\neg\varphi)^d \equiv \neg\varphi^d$,
- $(\varphi \wedge \psi)^d \equiv \varphi^d \vee \psi^d$,
- $(\varphi \vee \psi)^d \equiv \varphi^d \wedge \psi^d$.

Observe that the dual of the dual of a formula is the formula itself, i.e., that $(\varphi^d)^d \equiv \varphi$.

Proposition 1.23. *$\varphi \approx \psi$ iff $\varphi^d \approx \psi^d$ whenever the dual is defined.*

Proof. Exercise. □

1.6 Normal forms

In this section, we prove two *normal form* theorems for propositional logic. These guarantee that, for any formula, there is a semantically equivalent formula in some canonical normal form. Moreover, we shall give methods for finding these normal-form equivalents.

Say that a formula is in *disjunctive normal form if* it meets all of the following conditions:

- No connectives occur in the formula other than negations, conjunctions and disjunctions;
- Every occurrence of negation has minimal scope (i.e. any '$\neg$' is immediately followed by an atomic formula);
- No disjunction occurs within the scope of any conjunction.

Here are are some formulas in disjunctive normal form:

$$p_0$$
$$(p_0 \wedge p_1) \vee (p_0 \wedge \neg p_1)$$
$$(p_0 \wedge p_1) \vee (p_0 \wedge p_1 \wedge p_2 \wedge \neg p_3 \wedge \neg \alpha)$$
$$p_0 \vee (p_2 \wedge \neg p_7 \wedge p_9 \wedge p_3) \vee \neg p_1$$

Note that we have allowed ourselves to employ the relaxed bracketing-conventions that allow conjunctions and disjunctions to be of arbitrary length. These conventions make it easier to see when a formula is in disjunctive normal form.

To further illustrate the idea of disjunctive normal form, we shall introduce some more notation. We write '$(\neg)p_i$' to indicate that p_i is an atomic formula which may or may not be prefaced with an occurrence of negation. Then a formula in disjunctive normal form has the following shape:

$$\big((\neg)p_{i_1} \wedge \ldots \wedge (\neg)p_{i_j}\big) \vee \big((\neg)p_{i_{j+1}} \wedge \ldots \wedge (\neg)p_{i_k}\big) \vee \ldots \vee \big((\neg)p_{i_l} \wedge \ldots \wedge (\neg)p_{i_n}\big)$$

We now know what it is for a formula to be in disjunctive normal form. The result that we are aiming at is the following.

Proposition 1.24. *For any formula, there is a semantically equivalent formula in disjunctive normal form.*

Henceforth, we shall abbreviate 'Disjunctive Normal Form' by 'DNF'.

The proof of the DNF Theorem employs truth tables. We shall first illustrate the technique for finding an equivalent formula in DNF, and then turn this illustration into a rigorous proof.

Let's suppose we have some formula, φ, which contains three atomic formulas, 'p_0', 'p_1' and 'p_2'. The very first thing to do is fill out a complete truth table for φ. Maybe we end up with this:

φ	p_0	p_1	p_2
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$	$\mathbb{T}$
$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$	$\mathbb{F}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{T}$
$\mathbb{T}$	$\mathbb{F}$	$\mathbb{F}$	$\mathbb{F}$

As it happens, φ is true on four lines of its truth table, namely lines 1, 3, 7 and 8. Corresponding to each of those lines, we shall write down four formulas, whose only connectives are negations and conjunctions, where every negation has minimal scope:

- $p_0 \wedge p_1 \wedge p_2$ which is true on line 1 (and only then)
- $p_0 \wedge \neg p_1 \wedge p_2$ which is true on line 3 (and only then)
- $\neg p_0 \wedge \neg p_1 \wedge p_2$ which is true on line 7 (and only then)
- $\neg p_0 \wedge \neg p_1 \wedge \neg p_2$ which is true on line 8 (and only then)

But if we now disjoin all of these conjunctions, like so:

$$(p_0 \wedge p_1 \wedge p_2) \vee (p_0 \wedge \neg p_1 \wedge p_2) \vee (\neg p_0 \wedge \neg p_1 \wedge p_2) \vee (\neg p_0 \wedge \neg p_1 \wedge \neg p_2)$$

we have a formula in DNF which is true on exactly those lines where one of the disjuncts is true, i.e. it is true on (and only on) lines 1, 3, 7, and 8. So this formula has exactly the same truth table as φ. So we have a formula in DNF that is semantically equivalent to φ. Which is exactly what we wanted.

Now, this strategy did not depend on the specifics of φ; it is perfectly general. Consequently, we can use it to obtain a simple proof of the DNF Theorem.

Proof of DNF Theorem. Pick any arbitrary formula, φ, and let $p_0, \ldots, p_n$ be the atomic formulas that occur in φ. To obtain a formula in DNF that is semantically equivalent to φ, we consider φ's truth table. There are two cases to consider:

1. φ is false on every line of its truth table. Then, φ is a contradiction. In that case, the contradiction $(p_0 \wedge \neg p_0) \approx \varphi$, and $(p_0 \wedge \neg p_0)$ is in DNF.

2. φ is true on at least one line of its truth table. For each line i of the truth table, let ψ_i be a conjunction of the form

   $$((\neg)p_0 \wedge \ldots \wedge (\neg)p_n)$$

   where the following rules determine whether or not to include a negation in front of the atomic formulas:

   $$p_m \text{ is a conjunct of } \psi_i \textit{ iff } p_m \text{ is true on line } i$$
   $$\neg p_m \text{ is a conjunct of } \psi_i \textit{ iff } p_m \text{ is false on line } i$$

   Given these rules, a trivial proof by induction shows that ψ_i is true on (and only on) line i of the truth table which considers all possible valuations of $p_0, \ldots, p_n$ (i.e. φ's truth table).

Next, let $i_1, i_2, \ldots, i_m$ be the numbers of the lines of the truth table where φ is *true*. Now let χ be the formula:

$$\psi_{i_1} \vee \psi_{i_2} \vee \ldots \vee \psi_{i_m}$$

Since φ is true on at least one line of its truth table, χ is indeed well-defined; and in the limiting case where φ is true on exactly one line of its truth table, χ is just ψ_{i_k}, for some i_k.

By construction, χ is in DNF. Moreover, by construction, for each line i of the truth table: φ is true on line i of the truth table *iff* one of χ's disjuncts (namely, ψ_i) is true on, and only on, line i. (Again, this is shown by a trivial proof by induction.) Hence φ and χ have the same truth table, and so are semantically equivalent.

These two cases are exhaustive and, either way, we have a formula in DNF that is semantically equivalent to φ. □

So far we have discussed *disjunctive* normal form. Given the duality of disjunction and conjunction, it may not come as a surprise to hear that there is also such a thing as *conjunctive normal form* (CNF).

The definition of CNF is exactly analogous to the definition of DNF: A formula is in CNF iff it meets all of the following conditions:

- No connectives occur in the formula other than negations, conjunctions and disjunctions;
- Every occurrence of negation has minimal scope;
- No conjunction occurs within the scope of any disjunction.

Generally, then, a formula in CNF looks like this:

$$\big((\neg)p_{i_1} \vee \ldots \vee (\neg)p_{i_j}\big) \wedge \big((\neg)p_{i_{j+1}} \vee \ldots \vee (\neg)p_{i_k}\big) \wedge \ldots \wedge \big((\neg)p_{i_l} \vee \ldots \vee (\neg)p_{i_n}\big)$$

It should be immediate clear that if a formula is in DNF, then its dual is in CNF; and *vice versa*. Armed with this insight, we can immediately prove another normal form theorem:

Proposition 1.25. *For any formula, there is a semantically equivalent formula in conjunctive normal form.*

Proof. Let φ be any formula. Let ψ be a DNF formula semantically equivalent to φ^d by using Proposition 1.24. Now, ψ^d is on CNF by the observation above. Using Proposition 1.23, we have $(\varphi^d)^d \approx \psi^d$, i.e., the CNF formula ψ^d is semantically equivalent to φ. □

This slick proof is a further illustration of the power of duality. However, it might suggest that the DNF Theorem enjoys some kind of 'precedence' over the CNF Theorem. That would be misleading. We can easily prove the CNF Theorem directly, using the same proof techniques that we used to prove the DNF Theorem (whereupon the DNF Theorem could be proved as a consequence of the CNF Theorem and duality).

1.7 Expressive adequacy

We shall now demonstrate the expressive power of propositional logic.

The only primitive connectives we have defined are one-place (i.e. '$\neg$') and two-place (i.e. '$\wedge$', '$\vee$', '$\rightarrow$' and '$\leftrightarrow$'). But nothing stops us from introducing three-, four-, or five-place connectives; or, more generally, n-place connectives, for any number n we like. We might, for example, define a three-place connective, '$\heartsuit$', into existence, by stipulating that it is to have the following characteristic truth table:

φ	ψ	χ	$\heartsuit(\varphi, \psi, \chi)$
T	T	T	F
T	T	F	T
T	F	T	T
T	F	F	F
F	T	T	F
F	T	F	T
F	F	T	F
F	F	F	F

Probably this new connective would not correspond with any natural English expression (in the way that '$\wedge$' corresponds with 'and'). But a question arises: if we wanted to employ a connective with this characteristic truth table, must we add a *new* connective? Or can we get by with the connectives we *already have*?

Let us make this question more precise. Say that some connectives are *jointly expressively adequate iff*, for any possible truth function, there is a scheme containing only those connectives which expresses that truth function. Since we can represent truth functions using characteristic truth tables, we could equivalently say the following: some connectives are jointly expressively adequate *iff*, for any possible truth table, there is a scheme containing only those connectives with that truth table.

We say 'scheme' rather than 'formula', because we are not concerned with something as specific as a formula. To see why, consider the characteristic truth table for conjunction; this schematically encodes the information that a conjunction $(\varphi \wedge \psi)$ is true iff both φ and ψ are true (whatever φ and ψ might be). When we discuss expressive adequacy, we are considering something at the same level of generality.

The general point is, when we are armed with some jointly expressively adequate connectives, no truth function lies beyond our grasp.

Theorem 1.26. *The following pairs of connectives are jointly expressively adequate:*

- '$\neg$' *and* '$\vee$'
- '$\neg$' *and* '$\wedge$'
- '$\neg$' *and* '$\rightarrow$'

Proof. Given any truth table, we can use the method of proving the DNF Theorem (or the CNF Theorem) via truth tables, to write down a scheme which has the same truth table. For example, employing the truth table method for proving the DNF Theorem, I can tell you that the following scheme has the same characteristic truth table as $\heartsuit(\varphi, \psi, \chi)$, above:

$$(\varphi \wedge \psi \wedge \neg\chi) \vee (\varphi \wedge \neg\psi \wedge \chi) \vee (\neg\varphi \wedge \psi \wedge \neg\chi)$$

It follows that the connectives ¬', '∨' and '∧' are jointly expressively adequate.

We now show that there is an equivalent scheme which contains only '¬' and '∨'. To show do this, we simply consider the following equivalence:

$$(\varphi \wedge \psi) \approx \neg(\neg\varphi \vee \neg\psi)$$

(The details are left as an exercise).

For the joint expressive adequacy of '¬' and '∧' we note that:

$$(\varphi \vee \psi) \approx \neg(\neg\varphi \wedge \neg\psi)$$

To get the last result we note that:

$$(\varphi \vee \psi) \approx (\neg\varphi \rightarrow \psi)$$
$$(\varphi \wedge \psi) \approx \neg(\varphi \rightarrow \neg\psi)$$ □

In short, there is never any *need* to add new connectives. Indeed, there is already some redundancy among the connectives we have: we could have made do with just two connectives, if we had been feeling really austere.

In fact, some two-place connectives are *individually* expressively adequate. These connectives are among the standard ones, since they are rather cumbersome to use. But their existence shows that, if we had wanted to, we could have defined a truth-functional language that was expressively adequate, which contained only a single primitive connective.

The first such connective we shall consider is '↑', which has the following characteristic truth table.

φ	ψ	$\varphi \uparrow \psi$
T	T	F
T	F	T
F	T	T
F	F	T

This is often called 'the Sheffer stroke', after Harry Sheffer, who used it to show how to reduce the number of logical connectives in Russell and Whitehead's *Principia Mathematica*. It is quite common, as well, to call it 'nand', since its characteristic truth table is the negation of the truth table for '∧'.

Proposition 1.27. *'↑' is expressively adequate all by itself.*

Proof. Theorem 1.26 tells us that '¬' and '∨' are jointly expressively adequate. So it suffices to show that, given any scheme which contains only those two connectives, we can rewrite it as a semantically equivalent scheme which contains only '↑'. As in the proof of the subsidiary cases of Theorem 1.26, then, we simply apply the following equivalences:

$$\neg\varphi \approx (\varphi \uparrow \varphi)$$
$$(\varphi \vee \psi) \approx ((\varphi \uparrow \varphi) \uparrow (\psi \uparrow \psi))$$ □

Similarly, we can consider the connective '↓':

φ	ψ	$\varphi \downarrow \psi$
T	T	F
T	F	F
F	T	F
F	F	T

This is sometimes called the 'Peirce arrow' (Peirce himself called it 'ampheck'). More often, though, it is called 'nor', since its characteristic truth table is the negation of '$\vee$'.

Proposition 1.28. *'$\downarrow$' is expressively adequate all by itself.*

Proof. As in Proposition 1.27, although invoking the dual equivalences:

$$\neg\varphi \approx (\varphi \downarrow \varphi)$$
$$(\varphi \wedge \psi) \approx ((\varphi \downarrow \varphi) \downarrow (\psi \downarrow \psi))$$

□

1.8 Failures of expressive adequacy

In fact, the *only* two-place connectives which are individually expressively adequate are '$\uparrow$' and '$\downarrow$'. But how would we show this? More generally, how can we show that some connectives are *not* jointly expressively adequate?

The obvious thing to do is to try to find some truth table which we *cannot* express, using just the given connectives. But there is a bit of an art to this. Moreover, in the end, we shall have to rely upon induction; for we shall need to show that *no* scheme – no matter how *long* – is capable of expressing the target truth table.

To make this concrete, let's consider the question of whether '$\vee$' is expressively adequate all by itself. After a little reflection, it should be clear that it is not. In particular, it should be clear that any scheme which only contains disjunctions cannot have the same truth table as negation, i.e.:

φ	$\neg\varphi$
T	F
F	T

The intuitive reason, why this should be so, is simple: the top line of the desired truth table needs to have the value False; but the top line of any truth table for a scheme which *only* contains disjunctions will always be True. But so far, this is just hand-waving. To make it rigorous, we need to reach for induction. Here, then, is our rigorous proof.

Proposition 1.29. *'$\vee$' is not expressively adequate by itself.*

Proof. Let φ by any scheme containing no connective other than disjunctions. Suppose, for induction on length, that every shorter scheme containing only disjunctions is true whenever all its atomic constituents are true. There are two cases to consider:

- φ is atomic. Then there is nothing to prove.
- φ is $(\psi \vee \chi)$, for some schemes ψ and χ containing only disjunctions. Then, since ψ and χ are both shorter than φ, by the induction hypothesis they are both true when all their atomic constituents are true. Now the atomic constituents of φ are just the constituents of both ψ and χ, and φ is true whenever ψ and χ. So φ is true when all of its atomic constituents are true.

It now follows, by induction on length, that any scheme containing no connective other than disjunctions is true whenever all of its atomic constituents are true. Consequently, no scheme containing only disjunctions has the same truth table as that of negation. Hence '$\vee$' is not expressively adequate by itself. □

In fact, we can generalise Proposition 1.29:

Theorem 1.30. *The* only *two-place connectives that are expressively adequate by themselves are '$\uparrow$' and '$\downarrow$'.*

Proof. There are sixteen distinct two-place connectives. We shall run through them all, considering whether or not they are individually expressively adequate, in four groups.

Group 1: the top line of the truth table is True. Consider those connectives where the top line of the truth table is True. There are eight of these, including '$\wedge$', '$\vee$', '$\rightarrow$' and '$\leftrightarrow$', but also the following:

φ	ψ	$\varphi \circ_1 \psi$	$\varphi \circ_2 \psi$	$\varphi \circ_3 \psi$	$\varphi \circ_4 \psi$
T	T	T	T	T	T
T	F	T	T	T	F
F	T	T	F	F	T
F	F	T	T	F	F

(obviously the names for these connectives were chosen arbitrarily). But, exactly as in Proposition Proposition 1.29, none of these connectives can express the truth table for negation. So there is a connective whose truth table they cannot express. So none of them is individually expressively adequate.

Group 2: the bottom line of the truth table is False. Having eliminated eight connectives, eight remain. Of these, four are false on the bottom line of their truth table, namely:

φ	ψ	$\varphi \circ_5 \psi$	$\varphi \circ_6 \psi$	$\varphi \circ_7 \psi$	$\varphi \circ_8 \psi$
T	T	F	F	F	F
T	F	T	T	F	F
F	T	T	F	T	F
F	F	F	F	F	F

As above, though, none of these connectives can express the truth table for negation. To show this we prove that any scheme whose only connective is one of these (perhaps several times) is false whenever all of its atomic constituents are false. We can show this by induction, exactly as in Proposition Proposition 1.29 (I leave the details as an exercise).

Group 3: connectives with redundant positions. Consider two of the remaining four connectives:

φ	ψ	$\varphi \circ_9 \psi$	$\varphi \circ_{10} \psi$
T	T	F	F
T	F	F	T
F	T	T	F
F	F	T	T

These connectives have redundant positions, in the sense that the truth value of the overarching scheme only depends upon the truth value of one of the atomic constituents. More precisely:

$$\varphi \circ_9 \psi \approx \neg\varphi$$
$$\varphi \circ_{10} \psi \approx \neg\psi$$

Consequently, there are many truth functions that they cannot express. In particular, they cannot express either the tautologous truth function (given by '$\circ_1$'), or the contradictory truth function (given by '$\circ_8$'). To show this, it suffices to prove that any scheme whose only connective is either '$\circ_9$' or '$\circ_{10}$' (perhaps several times) is contingent, i.e. it is true on at least one line and false on at least one other line. We leave the details of this proof as an exercise.

Group 4. Only two connectives now remain, namely '$\uparrow$' and '$\downarrow$', and Propositions Proposition 1.27 and Proposition 1.28 show that both are individually expressively adequate. □

Problems

Problem 1.1. Prove Proposition 1.5

Problem 1.2. Prove Proposition 1.6

Problem 1.3. Give a mathematically rigorous definition of $\varphi[\psi/p]$ by induction.

Problem 1.4. Prove Proposition 1.17

Problem 1.5. Prove Proposition 1.19

Problem 1.6. Prove Proposition 1.20

Problem 1.7. Prove Theorem 1.21

Problem 1.8. Prove Proposition 1.23 by introducing an auxiliary mapping φ^n just as φ^d except for atomic formulas where φ^n is defined to be $\neg\varphi$ and proving that $\varphi^n \approx \neg\varphi$.

Problem 1.9. Consider the following formulas:

- $(A \to \neg B)$
- $\neg(A \leftrightarrow B)$
- $(\neg A \vee \neg(A \wedge B))$
- $(\neg(A \to B) \wedge (A \to C))$
- $(\neg(A \vee B) \leftrightarrow ((\neg C \wedge \neg A) \to \neg B))$
- $((\neg(A \wedge \neg B) \to C) \wedge \neg(A \wedge D))$

For each formula:

- write down formulas in DNF that are semantically equivalent to these formulas.

- write down formulas in CNF that are semantically equivalent to these formulas.

Problem 1.10. Where '$\circ_7$' has the characteristic truth table defined in the proof of Theorem 1.30, show that the following are jointly expressively adequate:

1. '$\circ_7$' and '$\neg$'.
2. '$\circ_7$' and '$\rightarrow$'.
3. '$\circ_7$' and '$\leftrightarrow$'.

Problem 1.11. Show that the connectives '$\circ_7$', '$\wedge$' and '$\vee$' are not jointly expressively adequate.

Problem 1.12. Complete the proof of Theorem 1.26.

Chapter 2

Natural Deduction

2.1 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 formulas or formulas. Good derivation systems have the property that any given sequence or arrangement of formulas or formulas can be verified mechanically to be "correct."

The simplest (and historically first) derivation systems for first-order logic were *axiomatic*. A sequence of formulas counts as a derivation in such a system if each individual formula in it is either among a fixed set of "axioms" or follows from formulas coming before it in the sequence by one of a fixed number of "inference rules"—and it can be mechanically verified if a formula is an axiom and whether it follows correctly from other formulas by one of the inference rules. Axiomatic proof systems are easy to describe—and also easy to handle meta-theoretically—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 proof systems represent derivations as trees of formulas 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 formula to be a *theorem* and what it means for a formula 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 $\vdash \varphi$ for "φ is a theorem" and "$\Gamma \vdash \varphi$" for "φ is derivable from Γ." However $\vdash$ is defined, we want it to match up with $\models$, that is:

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

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 φ is a theorem whenever φ is valid, and that $\Gamma \vdash \varphi$ whenever $\Gamma \models \varphi$. 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 formulas 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 formulas do not make good theories, they are defective in a fundamental way. Consistent sets of formulas 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 formulas 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.

2.2 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, $\rightarrow$I corresponds to conditional proof, and $\vee$E to proof by cases. A particularly simple rule is $\wedge$E which allows the inference from $\varphi \wedge \psi$ to φ (or ψ).

One feature that distinguishes natural deduction from other derivation systems is its use of assumptions. A derivation in natural deduction is a tree of formulas. A single formula stands at the root of the tree of formulas, and the "leaves" of the tree are formulas from which the conclusion is derived. In natural deduction, some leaf formulas 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 ψ from some assumptions which include φ, then the $\to$I rule allows us to infer $\varphi \to \psi$ and discharge any assumption of the form φ. (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 $\Gamma \vdash \varphi$ based on natural deduction holds iff there is a derivation in which φ is the last formula in the tree, and every leaf which is undischarged is in Γ. φ is a theorem in natural deduction iff there is a derivation in which φ is the last formula and all assumptions are discharged. For instance, here is a derivation that shows that $\vdash (\varphi \wedge \psi) \to \varphi$:

$$\dfrac{\dfrac{[\varphi \wedge \psi]^1}{\varphi}\wedge\text{E}}{(\varphi \wedge \psi) \to \varphi}\to\text{I}_1$$

The label 1 indicates that the assumption $\varphi \wedge \psi$ is discharged at the $\to$I inference.

A set Γ is inconsistent iff $\Gamma \vdash \bot$ in natural deduction. The rule $\bot$E makes it so that from an inconsistent set, any formula 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 Rules and Derivations

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

Definition 2.1 (Assumption). An *assumption* is any formula in the topmost position of any branch.

Derivations in natural deduction are certain trees of formulas, where the topmost formulas are assumptions, and if a formula stands below one, two, or three other sequents, it must follow correctly by a rule of inference. The formulas at the top of the inference are called the *premises* and the formula below the *conclusion* of the inference. The rules come in pairs, an introduction and an elimination rule for each logical operator. They introduce a logical operator in the conclusion or remove a logical operator from a premise of the rule. Some of the rules allow an assumption of a certain type to be *discharged*. To indicate which assumption is discharged by

which inference, we also assign labels to both the assumption and the inference. This is indicated by writing the assumption as "$[\varphi]^n$."

It is customary to consider rules for all the logical operators $\wedge$, $\vee$, $\rightarrow$, $\neg$, and $\bot$, even if some of those are consider as defined.

2.4 Propositional Rules

Rules for $\wedge$

$$\frac{\varphi \qquad \psi}{\varphi \wedge \psi}\,\wedge\mathrm{I} \qquad\qquad \frac{\varphi \wedge \psi}{\varphi}\,\wedge\mathrm{E} \qquad \frac{\varphi \wedge \psi}{\psi}\,\wedge\mathrm{E}$$

Rules for $\vee$

$$\frac{\varphi}{\varphi \vee \psi}\,\vee\mathrm{I} \qquad \frac{\psi}{\varphi \vee \psi}\,\vee\mathrm{I} \qquad\qquad \frac{\varphi \vee \psi \qquad \begin{array}{c}[\varphi]^n\\ \vdots\\ \chi\end{array} \qquad \begin{array}{c}[\psi]^n\\ \vdots\\ \chi\end{array}}{\chi}\,\vee\mathrm{E}_n$$

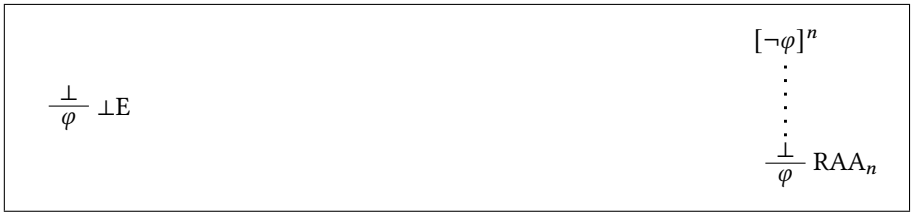
Rules for $\rightarrow$

$$\frac{\begin{array}{c}[\varphi]^n\\ \vdots\\ \psi\end{array}}{\varphi \rightarrow \psi}\,\rightarrow\mathrm{I}_n \qquad\qquad \frac{\varphi \rightarrow \psi \qquad \varphi}{\psi}\,\rightarrow\mathrm{E}$$

Rules for $\neg$

$$\frac{\begin{array}{c}[\varphi]^n\\ \vdots\\ \bot\end{array}}{\neg\varphi}\,\neg\mathrm{I}_n \qquad\qquad \frac{\neg\varphi \qquad \varphi}{\bot}\,\neg\mathrm{E}$$

Rules for $\bot$



Note that ¬I and RAA are very similar: The difference is that ¬I derives a negated formula $\neg\varphi$ but RAA a positive formula φ.

Whenever a rule indicates that some assumption may be discharged, we take this to be a permission, but not a requirement. E.g., in the $\rightarrow$I rule, we may discharge any number of assumptions of the form φ in the derivation of the premise ψ, including zero.

2.5 Derivations

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

Definition 2.2 (Derivation). A *derivation* of a formula φ from assumptions Γ is a tree of formulas satisfying the following conditions:

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

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

Example 2.3. Every assumption on its own is a derivation. So, e.g., χ by itself is a derivation, and so is θ by itself. We can obtain a new derivation from these by applying, say, the $\wedge$I rule,

$$\frac{\varphi \qquad \psi}{\varphi \wedge \psi}\ \wedge\text{I}$$

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

$$\frac{\chi \qquad \theta}{\chi \wedge \theta}\ \wedge\text{I}$$

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

$$\dfrac{\theta \qquad \chi}{\theta \wedge \chi}\,\wedge\mathrm{I}$$

is also a correct derivation.

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

$$\dfrac{\dfrac{[\chi]^1 \qquad \theta}{\chi \wedge \theta}\,\wedge\mathrm{I}}{\chi \rightarrow (\chi \wedge \theta)}\,\rightarrow\mathrm{I}_1 \qquad \dfrac{\dfrac{\chi \qquad [\theta]^1}{\chi \wedge \theta}\,\wedge\mathrm{I}}{\theta \rightarrow (\chi \wedge \theta)}\,\rightarrow\mathrm{I}_1$$

Remember that discharging of assumptions is a permission, not a requirement: we don't have to discharge the assumptions. In particular, we can apply a rule even if the assumptions are not present in the derivation. For instance, the following is legal, even though there is no assumption φ to be discharged:

$$\dfrac{\psi}{\varphi \rightarrow \psi}\,\rightarrow\mathrm{I}_1$$

2.6 Examples of Derivations

Example 2.4. Let's give a derivation of the formula $(\varphi \wedge \psi) \rightarrow \varphi$.

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

$$\dfrac{}{(\varphi \wedge \psi) \rightarrow \varphi}$$

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

$$\dfrac{\begin{array}{c}[\varphi \wedge \psi]^1 \\ \vdots \\ \varphi\end{array}}{(\varphi \wedge \psi) \rightarrow \varphi}\,\rightarrow\mathrm{I}_1$$

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

$$\dfrac{\dfrac{[\varphi \wedge \psi]^1}{\varphi}\,\wedge\mathrm{E}}{(\varphi \wedge \psi) \rightarrow \varphi}\,\rightarrow\mathrm{I}_1$$

We now have a correct derivation of $(\varphi \wedge \psi) \rightarrow \varphi$.

Example 2.5. Now let's give a derivation of $(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)$.

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

$$\dfrac{}{(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)}$$

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

$$\dfrac{\begin{array}{c}[\neg\varphi \vee \psi]^1 \\ \vdots \\ \varphi \rightarrow \psi\end{array}}{(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)} \rightarrow\mathrm{I}_1$$

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

$$\dfrac{\dfrac{\begin{array}{ccc}[\neg\varphi \vee \psi]^1 & \begin{array}{c}[\neg\varphi]^2 \\ \vdots \\ \varphi \rightarrow \psi\end{array} & \begin{array}{c}[\psi]^2 \\ \vdots \\ \varphi \rightarrow \psi\end{array}\end{array}}{\varphi \rightarrow \psi} \vee\mathrm{E}_2}{(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)} \rightarrow\mathrm{I}_1$$

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

$$\dfrac{\dfrac{\begin{array}{ccc}[\neg\varphi \vee \psi]^1 & \dfrac{\begin{array}{c}[\neg\varphi]^2, [\varphi]^3 \\ \vdots \\ \psi\end{array}}{\varphi \rightarrow \psi} \rightarrow\mathrm{I}_3 & \dfrac{\begin{array}{c}[\psi]^2, [\varphi]^4 \\ \vdots \\ \psi\end{array}}{\varphi \rightarrow \psi} \rightarrow\mathrm{I}_4\end{array}}{\varphi \rightarrow \psi} \vee\mathrm{E}_2}{(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)} \rightarrow\mathrm{I}_1$$

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

$$\dfrac{\begin{array}{cc}[\neg\varphi]^2 & [\varphi]^3\end{array}}{\begin{array}{c}\bot \\ \vdots \\ \psi\end{array}} \neg\mathrm{E}$$

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

$$\dfrac{[\neg\varphi \vee \psi]^1 \qquad \dfrac{\dfrac{[\neg\varphi]^2 \qquad [\varphi]^3}{\dfrac{\bot}{\psi}\,\bot\mathrm{E}}\,\bot\mathrm{I}}{\varphi \to \psi}\,{\to}\mathrm{I}_3 \qquad \dfrac{\begin{array}{c}[\psi]^2, [\varphi]^4 \\ \vdots \\ \psi\end{array}}{\varphi \to \psi}\,{\to}\mathrm{I}_4}{\varphi \to \psi}\,\vee\mathrm{E}_2}{(\neg\varphi \vee \psi) \to (\varphi \to \psi)}\,{\to}\mathrm{I}_1$$

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

$$\dfrac{[\neg\varphi \vee \psi]^1 \qquad \dfrac{\dfrac{[\neg\varphi]^2 \qquad [\varphi]^3}{\dfrac{\bot}{\psi}\,\bot\mathrm{E}}\,\neg\mathrm{E}}{\varphi \to \psi}\,{\to}\mathrm{I}_3 \qquad \dfrac{[\psi]^2}{\varphi \to \psi}\,{\to}\mathrm{I}}{\varphi \to \psi}\,\vee\mathrm{E}_2}{(\neg\varphi \vee \psi) \to (\varphi \to \psi)}\,{\to}\mathrm{I}_1$$

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

Example 2.6. So far we have not needed the RAA rule. It is special in that it allows us to discharge an assumption that isn't a sub-formula of the conclusion of the rule. It is closely related to the $\bot$E rule. In fact, the $\bot$E rule is a special case of the RAA rule—there is a logic called "intuitionistic logic" in which only $\bot$E is allowed. The RAA rule is a last resort when nothing else works. For instance, suppose we want to derive $\varphi \vee \neg\varphi$. Our usual strategy would be to attempt to derive $\varphi \vee \neg\varphi$ using $\vee$I. But this would require us to derive either φ or $\neg\varphi$ from no assumptions, and this can't be done. RAA to the rescue!

$$\dfrac{\begin{array}{c}[\neg(\varphi \vee \neg\varphi)]^1 \\ \vdots \\ \bot\end{array}}{\varphi \vee \neg\varphi}\,\mathrm{RAA}_1$$

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

$$\dfrac{\begin{array}{c}[\neg(\varphi \vee \neg\varphi)]^1 \\ \vdots \\ \neg\varphi\end{array} \qquad \begin{array}{c}[\neg(\varphi \vee \neg\varphi)]^1 \\ \vdots \\ \varphi\end{array}}{\dfrac{\bot}{\varphi \vee \neg\varphi}\,\mathrm{RAA}_1}\,\neg\mathrm{E}$$

Our strategy for finding a derivation of $\neg\varphi$ calls for an application of $\neg$I:

$$\dfrac{\dfrac{\dfrac{\dfrac{[\neg(\varphi \vee \neg\varphi)]^1, [\varphi]^2}{\vdots}}{\dfrac{\bot}{\neg\varphi}\,\neg\mathrm{I}_2} \qquad \dfrac{[\neg(\varphi \vee \neg\varphi)]^1}{\vdots}}{\varphi}}{\dfrac{\bot}{\varphi \vee \neg\varphi}\,\mathrm{RAA}_1}\,\neg\mathrm{E}$$

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

$$\dfrac{\dfrac{\dfrac{\dfrac{[\neg(\varphi \vee \neg\varphi)]^1 \qquad \dfrac{[\varphi]^2}{\varphi \vee \neg\varphi}\,\vee\mathrm{I}}{\bot}\,\neg\mathrm{E}}{\neg\varphi}\,\neg\mathrm{I}_2 \qquad \dfrac{[\neg(\varphi \vee \neg\varphi)]^1}{\vdots}}{\varphi}}{\dfrac{\bot}{\varphi \vee \neg\varphi}\,\mathrm{RAA}_1}\,\neg\mathrm{E}$$

On the right side we use the same strategy, except we get φ by RAA:

$$\dfrac{\dfrac{\dfrac{\dfrac{[\neg(\varphi \vee \neg\varphi)]^1 \qquad \dfrac{[\varphi]^2}{\varphi \vee \neg\varphi}\,\vee\mathrm{I}}{\bot}\,\neg\mathrm{E}}{\neg\varphi}\,\neg\mathrm{I}_2 \qquad \dfrac{\dfrac{\dfrac{[\neg(\varphi \vee \neg\varphi)]^1 \qquad \dfrac{[\neg\varphi]^3}{\varphi \vee \neg\varphi}\,\vee\mathrm{I}}{\bot}\,\neg\mathrm{E}}{\varphi}\,\mathrm{RAA}_3}{\bot}\,\neg\mathrm{E}}{\dfrac{\bot}{\varphi \vee \neg\varphi}\,\mathrm{RAA}_1}$$

2.7 Proof-Theoretic Notions

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

Definition 2.7 (Theorems). A formula φ is a *theorem* if there is a derivation of φ in natural deduction in which all assumptions are discharged. We write $\vdash \varphi$ if φ is a theorem and $\nvdash \varphi$ if it is not.

Definition 2.8 (Derivability). A formula φ is *derivable from* a set of formulas Γ, $\Gamma \vdash \varphi$, if there is a derivation with conclusion φ and in which every assumption is either discharged or is in Γ. If φ is not derivable from Γ we write $\Gamma \nvdash \varphi$.

Definition 2.9 (Consistency). A set of formulas Γ is *inconsistent* iff $\Gamma \vdash \bot$. If Γ is not inconsistent, i.e., if $\Gamma \nvdash \bot$, we say it is *consistent*.

Proposition 2.10 (Reflexivity). *If $\varphi \in \Gamma$, then $\Gamma \vdash \varphi$.*

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

Proposition 2.11 (Monotony). *If $\Gamma \subseteq \Delta$ and $\Gamma \vdash \varphi$, then $\Delta \vdash \varphi$.*

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

Proposition 2.12 (Transitivity). *If* $\Gamma \vdash \varphi$ *and* $\{\varphi\} \cup \Delta \vdash \psi$*, then* $\Gamma \cup \Delta \vdash \psi$*.*

Proof. If $\Gamma \vdash \varphi$, there is a derivation δ_0 of φ with all undischarged assumptions in Γ. If $\{\varphi\} \cup \Delta \vdash \psi$, then there is a derivation δ_1 of ψ with all undischarged assumptions in $\{\varphi\} \cup \Delta$. Now consider:

$$\dfrac{\dfrac{\begin{array}{c}\Delta, [\varphi]^1 \\ \vdots\,\delta_1 \\ \psi\end{array}}{\varphi \to \psi}\to\mathrm{I}_1 \qquad \begin{array}{c}\Gamma \\ \vdots\,\delta_0 \\ \varphi\end{array}}{\psi}\to\mathrm{E}$$

The undischarged assumptions are now all among $\Gamma \cup \Delta$, so this shows $\Gamma \cup \Delta \vdash \psi$. □

When $\Gamma = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ is a finite set we may use the simplified notation $\varphi_1, \varphi_2, \ldots, \varphi_k \vdash \psi$ for $\Gamma \vdash \psi$, in particular $\varphi \vdash \psi$ means that $\{\varphi\} \vdash \psi$.

Note that if $\Gamma \vdash \varphi$ and $\varphi \vdash \psi$, then $\Gamma \vdash \psi$. It follows also that if $\varphi_1, \ldots, \varphi_n \vdash \psi$ and $\Gamma \vdash \varphi_i$ for each i, then $\Gamma \vdash \psi$.

Proposition 2.13. Γ *is inconsistent iff* $\Gamma \vdash \varphi$ *for every formula* φ*.*

Proof. Exercise. □

Proposition 2.14 (Compactness).

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

Proof.

1. If $\Gamma \vdash \varphi$, then there is a derivation δ of φ from Γ. Let Γ_0 be the set of undischarged assumptions of δ. Since any derivation is finite, Γ_0 can only contain finitely many formulas. So, δ is a derivation of φ from a finite $\Gamma_0 \subseteq \Gamma$.
2. This is the contrapositive of (1) for the special case $\varphi \equiv \bot$. □

2.8 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 2.15. *If* $\Gamma \vdash \varphi$ *and* $\Gamma \cup \{\varphi\}$ *is inconsistent, then* Γ *is inconsistent.*

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

$$\dfrac{\dfrac{\dfrac{\Gamma, [\varphi]^1}{\vdots\,\delta_2}}{\dfrac{\bot}{\neg\varphi}\,\neg\mathrm{I}_1} \qquad \dfrac{\Gamma}{\vdots\,\delta_1}}{\bot}\,\neg\mathrm{E}$$

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

Proposition 2.16. *$\Gamma \vdash \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is inconsistent.*

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

$$\dfrac{\neg\varphi \qquad \dfrac{\Gamma}{\vdots\,\delta_0}}{\bot}\,\neg\mathrm{E}$$

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

$$\dfrac{\dfrac{\Gamma, [\neg\varphi]^1}{\vdots\,\delta_1}}{\bot}\,\mathrm{RAA}$$

□

Proposition 2.17. *If $\Gamma \vdash \varphi$ and $\neg\varphi \in \Gamma$, then Γ is inconsistent.*

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

$$\dfrac{\neg\varphi \qquad \dfrac{\Gamma}{\vdots\,\delta}}{\bot}\,\neg\mathrm{E}$$

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

Proposition 2.18. *If $\Gamma \cup \{\varphi\}$ and $\Gamma \cup \{\neg\varphi\}$ are both inconsistent, then Γ is inconsistent.*

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

$$\dfrac{\dfrac{\dfrac{\Gamma, [\neg\varphi]^2}{\vdots\,\delta_2}}{\dfrac{\bot}{\neg\neg\varphi}\,\neg\mathrm{I}_2} \qquad \dfrac{\dfrac{\Gamma, [\varphi]^1}{\vdots\,\delta_1}}{\dfrac{\bot}{\neg\varphi}\,\neg\mathrm{I}_1}}{\bot}\,\neg\mathrm{E}$$

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

2.9 Derivability and the Propositional Connectives

Proposition 2.19. *1. Both $\varphi \wedge \psi \vdash \varphi$ and $\varphi \wedge \psi \vdash \psi$*

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

Proof. 1. We can derive both

$$\dfrac{\varphi \wedge \psi}{\varphi}\,\wedge\mathrm{E} \qquad \dfrac{\varphi \wedge \psi}{\psi}\,\wedge\mathrm{E}$$

2. We can derive:

$$\dfrac{\varphi \qquad \psi}{\varphi \wedge \psi}\,\wedge\mathrm{I}$$

□

Proposition 2.20. *1. $\varphi \vee \psi, \neg\varphi, \neg\psi$ is inconsistent.*

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

Proof. 1. Consider the following derivation:

$$\dfrac{\varphi \vee \psi \qquad \dfrac{\neg\varphi \qquad [\varphi]^1}{\bot}\,\neg\mathrm{E} \qquad \dfrac{\neg\psi \qquad [\psi]^1}{\bot}\,\neg\mathrm{E}}{\bot}\,\vee\mathrm{E}_1$$

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

2. We can derive both

$$\dfrac{\varphi}{\varphi \vee \psi}\,\vee\mathrm{I} \qquad \dfrac{\psi}{\varphi \vee \psi}\,\vee\mathrm{I}$$

□

Proposition 2.21. *1. $\varphi, \varphi \to \psi \vdash \psi$.*

2. Both $\neg\varphi \vdash \varphi \to \psi$ and $\psi \vdash \varphi \to \psi$.

Proof. 1. We can derive:

$$\dfrac{\varphi \to \psi \qquad \psi}{\psi}\,\to\mathrm{E}$$

2. This is shown by the following two derivations:

$$\dfrac{\dfrac{\dfrac{\neg\varphi \qquad [\varphi]^1}{\bot}\,\neg\mathrm{E}}{\psi}\,\bot\mathrm{E}}{\varphi \to \psi}\,\to\mathrm{I}_1 \qquad \dfrac{\psi}{\varphi \to \psi}\,\to\mathrm{I}$$

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

2.10 Soundness

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

1. every derivable formula is a tautology;
2. if a formula is derivable from some others, it is also a consequence of them;
3. if a set of formulas is inconsistent, it is unsatisfiable.

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

Theorem 2.22 (Soundness). *If φ is derivable from the undischarged assumptions Γ, then $\Gamma \vDash \varphi$.*

Proof. Let δ be a derivation of φ. We proceed by induction on the number of inferences in δ.

For the induction basis we show the claim if the number of inferences is 0. In this case, δ consists only of a single formula φ, i.e., an assumption. That assumption is undischarged, since assumptions can only be discharged by inferences, and there are no inferences. So, any valuation v that satisfies all of the undischarged assumptions of the proof also satisfies φ.

Now for the inductive step. Suppose that δ contains n inferences. The premise(s) of the lowermost inference are derived using sub-derivations, each of which contains fewer than n inferences. We assume the induction hypothesis: The premises of the lowermost inference follow from the undischarged assumptions of the sub-derivations ending in those premises. We have to show that the conclusion φ follows from the undischarged assumptions of the entire proof.

We distinguish cases according to the type of the lowermost inference. First, we consider the possible inferences with only one premise.

1. Suppose that the last inference is ¬I: The derivation has the form

$$\dfrac{\begin{array}{c}\Gamma, [\varphi]^n \\ \vdots\,\delta_1\,\vdots \\ \bot\end{array}}{\neg\varphi}\ \neg\mathrm{I}_n$$

   By inductive hypothesis, $\bot$ follows from the undischarged assumptions $\Gamma \cup \{\varphi\}$ of δ_1. Consider a valuation v. We need to show that, if $v \vDash \Gamma$, then $v \vDash \neg\varphi$. Suppose for reductio that $v \vDash \Gamma$, but $v \nvDash \neg\varphi$, i.e., $v \vDash \varphi$. This would mean that $v \vDash \Gamma \cup \{\varphi\}$. This is contrary to our inductive hypothesis. So, $v \vDash \neg\varphi$.

2. The last inference is ∧E: There are two variants: φ or ψ may be inferred from the premise $\varphi \wedge \psi$. Consider the first case. The derivation δ looks like this:

$$\dfrac{\begin{array}{c}\Gamma\\ \vdots\\ \delta_1\\ \vdots\\ \varphi \wedge \psi\end{array}}{\varphi}\ \wedge\mathrm{E}$$

By inductive hypothesis, $\varphi \wedge \psi$ follows from the undischarged assumptions Γ of δ_1. Consider a structure v. We need to show that, if $v \vDash \Gamma$, then $v \vDash \varphi$. Suppose $v \vDash \Gamma$. By our inductive hypothesis ($\Gamma \vDash \varphi \vee \psi$), we know that $v \vDash \varphi \wedge \psi$. By definition, $v \vDash \varphi \wedge \psi$ iff $v \vDash \varphi$ and $v \vDash \psi$. (The case where ψ is inferred from $\varphi \wedge \psi$ is handled similarly.)

3. The last inference is $\vee$I: There are two variants: $\varphi \vee \psi$ may be inferred from the premise φ or the premise ψ. Consider the first case. The derivation has the form

$$\dfrac{\begin{array}{c}\Gamma\\ \vdots\\ \delta_1\\ \vdots\\ \varphi\end{array}}{\varphi \vee \psi}\ \vee\mathrm{I}$$

By inductive hypothesis, φ follows from the undischarged assumptions Γ of δ_1. Consider a valuation v. We need to show that, if $v \vDash \Gamma$, then $v \vDash \varphi \vee \psi$. Suppose $v \vDash \Gamma$; then $v \vDash \varphi$ since $\Gamma \vDash \varphi$ (the inductive hypothesis). So it must also be the case that $v \vDash \varphi \vee \psi$. (The case where $\varphi \vee \psi$ is inferred from ψ is handled similarly.)

4. The last inference is $\rightarrow$I: $\varphi \rightarrow \psi$ is inferred from a subproof with assumption φ and conclusion ψ, i.e.,

$$\dfrac{\begin{array}{c}\Gamma, [\varphi]^n\\ \vdots\\ \delta_1\\ \vdots\\ \psi\end{array}}{\varphi \rightarrow \psi}\ \rightarrow\mathrm{I}_n$$

By inductive hypothesis, ψ follows from the undischarged assumptions of δ_1, i.e., $\Gamma \cup \{\varphi\} \vDash \psi$. Consider a valuation v. The undischarged assumptions of δ are just Γ, since φ is discharged at the last inference. So we need to show that $\Gamma \vDash \varphi \rightarrow \psi$. For reductio, suppose that for some valuation v, $v \vDash \Gamma$ but $v \nvDash \varphi \rightarrow \psi$. So, $v \vDash \varphi$ and $v \nvDash \psi$. But by hypothesis, ψ is a consequence of $\Gamma \cup \{\varphi\}$, i.e., $v \vDash \psi$, which is a contradiction. So, $\Gamma \vDash \varphi \rightarrow \psi$.

5. The last inference is $\bot$E: Here, δ ends in

$$\dfrac{\begin{array}{c}\Gamma\\ \vdots\\ \delta_1\\ \vdots\\ \bot\end{array}}{\varphi}\ \bot\mathrm{E}$$

By induction hypothesis, $\Gamma \vDash \bot$. We have to show that $\Gamma \vDash \varphi$. Suppose not; then for some v we have $v \vDash \Gamma$ and $v \nvDash \varphi$. But we always have $v \nvDash \bot$, so this would mean that $\Gamma \nvDash \bot$, contrary to the induction hypothesis.

6. The last inference is RAA: Exercise.

Now let's consider the possible inferences with several premises: $\vee$E, $\wedge$I, and $\rightarrow$E.

1. The last inference is $\wedge$I. $\varphi \wedge \psi$ is inferred from the premises φ and ψ and δ has the form

$$\dfrac{\begin{array}{cc}\Gamma_1 & \Gamma_2 \\ \vdots & \vdots \\ \delta_1 & \delta_2 \\ \vdots & \vdots \\ \varphi & \psi\end{array}}{\varphi \wedge \psi}\ \wedge\text{I}$$

   By induction hypothesis, φ follows from the undischarged assumptions Γ_1 of δ_1 and ψ follows from the undischarged assumptions Γ_2 of δ_2. The undischarged assumptions of δ are $\Gamma_1 \cup \gamma_2$, so we have to show that $\Gamma_1 \cup \Gamma_2 \vDash \varphi \wedge \psi$. Consider a valuation v with $v \vDash \Gamma_1 \cup \Gamma_2$. Since $v \vDash \Gamma_1$, it must be the case that $v \vDash \varphi$ as $\Gamma_1 \vDash \varphi$, and since $v \vDash \Gamma_2$, $v \vDash \psi$ since $\Gamma_2 \vDash \psi$. Together, $v \vDash \varphi \wedge \psi$.

2. The last inference is $\vee$E: Exercise.

3. The last inference is $\rightarrow$E. ψ is inferred from the premises $\varphi \rightarrow \psi$ and φ. The derivation δ looks like this:

$$\dfrac{\begin{array}{cc}\Gamma_1 & \Gamma_2 \\ \vdots & \vdots \\ \delta_1 & \delta_2 \\ \vdots & \vdots \\ \varphi \rightarrow \psi & \varphi\end{array}}{\psi}\ \rightarrow\text{E}$$

   By induction hypothesis, $\varphi \rightarrow \psi$ follows from the undischarged assumptions Γ_1 of δ_1 and φ follows from the undischarged assumptions Γ_2 of δ_2. Consider a valuation v. We need to show that, if $v \vDash \Gamma_1 \cup \Gamma_2$, then $v \vDash \psi$. Suppose $v \vDash \Gamma_1 \cup \Gamma_2$. Since $\Gamma_1 \vDash \varphi \rightarrow \psi$, $v \vDash \varphi \rightarrow \psi$. Since $\Gamma_2 \vDash \varphi$, we have $v \vDash \varphi$. This means that $v \vDash \psi$ (For if $v \nvDash \psi$, since $v \vDash \varphi$, we'd have $v \nvDash \varphi \rightarrow \psi$, contradicting $v \vDash \varphi \rightarrow \psi$).

4. The last inference is $\neg$E: Exercise.

□

Corollary 2.23. *If $\vdash \varphi$, then φ is a tautology.*

Corollary 2.24. *If Γ is satisfiable, then it is consistent.*

Proof. We prove the contrapositive. Suppose that Γ is not consistent. Then $\Gamma \vdash \bot$, i.e., there is a derivation of $\bot$ from undischarged assumptions in Γ. By Theorem 2.22, any valuation v that satisfies Γ must satisfy $\bot$. Since $v \nvDash \bot$ for every valuation v, no v can satisfy Γ, i.e., Γ is not satisfiable. □

Problems

Problem 2.1. Give derivations of the following:

1. $\neg(\varphi \to \psi) \to (\varphi \wedge \neg\psi)$
2. $(\varphi \to \chi) \vee (\psi \to \chi)$ from the assumption $(\varphi \wedge \psi) \to \chi$

Problem 2.2. Prove Proposition 2.13

Problem 2.3. Prove that $\Gamma \vdash \neg\varphi$ iff $\Gamma \cup \{\varphi\}$ is inconsistent.

Problem 2.4. Complete the proof of Theorem 2.22.

Chapter 3

The Completeness Theorem

3.1 Introduction

The completeness theorem is one of the most fundamental results about logic. It comes in two formulations, the equivalence of which we'll prove. In its first formulation it says something fundamental about the relationship between semantic consequence and our proof system: if a formula φ follows from some formulas Γ, then there is also a derivation that establishes $\Gamma \vdash \varphi$. Thus, the proof system is as strong as it can possibly be without proving things that don't actually follow.

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

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

Frege vehemently disagreed. The second formulation of the completeness theorem shows that Hilbert was right in at least the sense that if the axioms are consistent, then *some* 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 $\Gamma \vdash \varphi$ is finite and so can only use finitely many of the formulas in Γ, it follows by the completeness theorem that if φ is a consequence of Γ, it is already a consequence of a finite subset of Γ. This is called *compactness*. Equivalently, if every finite subset of Γ is consistent, then Γ itself must be consistent.

Although the compactness theorem follows from the completeness theorem via the detour through derivations, it is also possible to use the *the proof of* the completeness theorem to establish it directly. For what the proof does is take a set of formulas 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 formulas instead of consistent ones.

3.2 Outline of the Proof

The proof of the completeness theorem is a bit complex, and upon first reading it, it is easy to get lost. So let us outline the proof. The first step is a shift of perspective, that allows us to see a route to a proof. When completeness is thought of as "whenever $\Gamma \models \varphi$ then $\Gamma \vdash \varphi$," it may be hard to even come up with an idea: for to show that $\Gamma \vdash \varphi$ we have to find a derivation, and it does not look like the hypothesis that $\Gamma \models \varphi$ helps us for this in any way. For some proof systems it is possible to directly construct a derivation, but we will take a slightly different 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 formula 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 v such that $v \models p$ for all $p \in \Gamma$. Well, let $v(p) = \mathbb{T}$ iff $p \in \Gamma$.

Now suppose Γ contains some formula $\neg\psi$, with ψ atomic. We might worry that the construction of v interferes with the possibility of making $\neg\psi$ true. But here's where the consistency of Γ comes in: if $\neg\psi \in \Gamma$, then $\psi \notin \Gamma$, or else Γ would be inconsistent. And if $\psi \notin \Gamma$, then according to our construction of v, $v \not\models \psi$, so $v \models \neg\psi$. So far so good.

What if Γ contains complex, non-atomic formulas? Say it contains $\varphi \wedge \psi$. To make that true, we should proceed as if both φ and ψ were in Γ. And if $\varphi \vee \psi \in \Gamma$, then we will have to make at least one of them true, i.e., proceed as if one of them was in Γ.

This suggests the following idea: we add additional formulas to Γ so as to (a) keep the resulting set consistent and (b) make sure that for every possible atomic formula φ, either φ is in the resulting set, or $\neg\varphi$ is, and (c) such that, whenever $\varphi \wedge \psi$ is in the set, so are both φ and ψ, if $\varphi \vee \psi$ is in the set, at least one of φ or ψ is also, etc. We keep doing this (potentially forever). Call the set of all formulas so added Γ^*. Then our construction above would provide us with a valuation 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 $\varphi \wedge \psi$ iff it contains both φ and ψ, $\varphi \vee \psi$ iff it contains at least one of them, etc. (Proposition 3.2). We'll then take the consistent set Γ and show that it can be extended to a consistent and complete set Γ^* (Lemma 3.3). This set Γ^* is what we'll use to define our valuation $v(\Gamma^*)$. The valuation is determined by the propositional variables in Γ^* (Definition 3.4). We'll use the properties of complete consistent sets to show that indeed $v(\Gamma^*) \models \varphi$ iff $\varphi \in \Gamma^*$ (Lemma 3.5), and thus in particular, $v(\Gamma^*) \models \Gamma$.

3.3 Complete Consistent Sets of Formulas

Definition 3.1 (Complete set). A set Γ of formulas is *complete* iff for any formula φ, either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$.

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

Complete consistent sets are important in the completeness proof since we can guarantee that every consistent set of formulas Γ is contained in a complete consistent set Γ^*. A complete consistent set contains, for each formula φ, either φ or its negation $\neg\varphi$, 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 formulas in Γ^* (and hence also all those in Γ) true. The proof of this latter fact requires that $\neg\varphi \in \Gamma^*$ iff $\varphi \notin \Gamma^*$, $(\varphi \vee \psi) \in \Gamma^*$ iff $\varphi \in \Gamma^*$ or $\psi \in \Gamma^*$, etc.

In what follows, we will often tacitly use the properties of reflexivity, monotonicity, and transitivity of $\vdash$ (see section 2.7).

Proposition 3.2. *Suppose Γ is complete and consistent. Then:*

1. *If $\Gamma \vdash \varphi$, then $\varphi \in \Gamma$.*
2. *$\varphi \wedge \psi \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$.*
3. *$\varphi \vee \psi \in \Gamma$ iff either $\varphi \in \Gamma$ or $\psi \in \Gamma$.*
4. *$\varphi \to \psi \in \Gamma$ iff either $\varphi \notin \Gamma$ or $\psi \in \Gamma$.*

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

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

   Suppose that $\Gamma \vdash \varphi$. Suppose to the contrary that $\varphi \notin \Gamma$. Since Γ is complete, $\neg\varphi \in \Gamma$. By Proposition 2.17, Γ is inconsistent. This contradicts the assumption that Γ is consistent. Hence, it cannot be the case that $\varphi \notin \Gamma$, so $\varphi \in \Gamma$.

2. $\varphi \wedge \psi \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$:

   For the forward direction, suppose $\varphi \wedge \psi \in \Gamma$. Then by Proposition 2.19, item (1), $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$. By (1), $\varphi \in \Gamma$ and $\psi \in \Gamma$, as required.

   For the reverse direction, let $\varphi \in \Gamma$ and $\psi \in \Gamma$. By Proposition 2.19, item (2), $\Gamma \vdash \varphi \wedge \psi$. By (1), $\varphi \wedge \psi \in \Gamma$.

3. First we show that if $\varphi \vee \psi \in \Gamma$, then either $\varphi \in \Gamma$ or $\psi \in \Gamma$. Suppose $\varphi \vee \psi \in \Gamma$ but $\varphi \notin \Gamma$ and $\psi \notin \Gamma$. Since Γ is complete, $\neg\varphi \in \Gamma$ and $\neg\psi \in \Gamma$. By Proposition 2.20, item (1), Γ is inconsistent, a contradiction. Hence, either $\varphi \in \Gamma$ or $\psi \in \Gamma$.

   For the reverse direction, suppose that $\varphi \in \Gamma$ or $\psi \in \Gamma$. By Proposition 2.20, item (2), $\Gamma \vdash \varphi \vee \psi$. By (1), $\varphi \vee \psi \in \Gamma$, as required.

4. For the forward direction, suppose $\varphi \to \psi \in \Gamma$, and suppose to the contrary that $\varphi \in \Gamma$ and $\psi \notin \Gamma$. On these assumptions, $\varphi \to \psi \in \Gamma$ and $\varphi \in \Gamma$. By Proposition 2.21, item (1), $\Gamma \vdash \psi$. But then by (1), $\psi \in \Gamma$, contradicting the assumption that $\psi \notin \Gamma$.

   For the reverse direction, first consider the case where $\varphi \notin \Gamma$. Since Γ is complete, $\neg\varphi \in \Gamma$. By Proposition 2.21, item (2), $\Gamma \vdash \varphi \to \psi$. Again by (1), we get that $\varphi \to \psi \in \Gamma$, as required.

   Now consider the case where $\psi \in \Gamma$. By Proposition 2.21, item (2) again, $\Gamma \vdash \varphi \to \psi$. By (1), $\varphi \to \psi \in \Gamma$. □

3.4 Lindenbaum's Lemma

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

Lemma 3.3 (Lindenbaum's Lemma). *Every consistent set Γ in a language $\mathcal{L}$ can be extended to a complete and consistent set Γ^*.*

Proof. Let Γ be consistent. Let $\varphi_0, \varphi_1, \ldots$ be an enumeration of all the formulas of $\mathcal{L}$. Define $\Gamma_0 = \Gamma$, and

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

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

Each Γ_n is consistent: Γ_0 is consistent by definition. If $\Gamma_{n+1} = \Gamma_n \cup \{\varphi_n\}$, this is because the latter is consistent. If it isn't, $\Gamma_{n+1} = \Gamma_n \cup \{\neg\varphi_n\}$. We have to verify that $\Gamma_n \cup \{\neg\varphi_n\}$ is consistent. Suppose it's not. Then *both* $\Gamma_n \cup \{\varphi_n\}$ and $\Gamma_n \cup \{\neg\varphi_n\}$ are inconsistent. This means that Γ_n would be inconsistent by Proposition 2.17, 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 $\Gamma_{n+1} = \Gamma_n \cup \{\varphi_n\}$ or $= \Gamma_n \cup \{\neg\varphi_n\}$ 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 $\psi \in \Gamma^*$ not already in Γ_0 is added at some stage i. If n is the last one of these, then all ψ in the finite subset are in Γ_n. So, every finite subset of Γ^* is consistent. By Proposition 2.14, Γ^* is consistent.

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

3.5 Construction of a Model

We are now ready to define a valuation that makes all $\varphi \in \Gamma$ 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 $v(\Gamma^*)$.

Definition 3.4. Suppose Γ^* is a complete consistent set of formulas. Then we let

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

Lemma 3.5 (Truth Lemma). *$v(\Gamma^*) \vDash \varphi$ iff $\varphi \in \Gamma^*$.*

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

1. $\varphi \equiv \bot$: $v(\Gamma^*) \nvDash \bot$ by definition of satisfaction. On the other hand, $\bot \notin \Gamma^*$ since Γ^* is consistent.

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

3. $\varphi \equiv \neg\psi$: $v(\Gamma^*) \vDash \varphi$ iff $v(\Gamma^*) \vDash \psi$ (by definition of satisfaction). By induction hypothesis, $v(\Gamma^*) \vDash \psi$ iff $\psi \notin \Gamma^*$. Since Γ^* is consistent and complete, $\psi \notin \Gamma^*$ iff $\neg\psi \in \Gamma^*$.

4. $\varphi \equiv \psi \wedge \chi$: $v(\Gamma^*) \vDash \varphi$ iff we have both $v(\Gamma^*) \vDash \psi$ and $v(\Gamma^*) \vDash \chi$ (by definition of satisfaction) iff both $\psi \in \Gamma^*$ and $\chi \in \Gamma^*$ (by the induction hypothesis). By Proposition 3.2(2), this is the case iff $(\psi \wedge \chi) \in \Gamma^*$.

5. $\varphi \equiv \psi \vee \chi$: $v(\Gamma^*) \vDash \varphi$ iff $v(\Gamma^*) \vDash \psi$ or $v(\Gamma^*) \vDash \chi$ (by definition of satisfaction) iff $\psi \in \Gamma^*$ or $\chi \in \Gamma^*$ (by induction hypothesis). This is the case iff $(\psi \vee \chi) \in \Gamma^*$ (by Proposition 3.2(3)).

6. $\varphi \equiv \psi \to \chi$: $v(\Gamma^*) \vDash \varphi$ iff $\mathfrak{M}(\Gamma^*) \nvDash \psi$ or $\mathfrak{M}(\Gamma^*) \vDash \chi$ (by definition of satisfaction) iff $\psi \notin \Gamma^*$ or $\chi \in \Gamma^*$ (by induction hypothesis). This is the case iff $(\psi \to \chi) \in \Gamma^*$ (by Proposition 3.2(4)).

3.6 The Completeness Theorem

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

Theorem 3.6 (Completeness Theorem). *Let Γ be a set of formulas. If Γ is consistent, it is satisfiable.*

Proof. Suppose Γ is consistent. By Lemma 3.3, there is a $\Gamma^* \supseteq \Gamma$ which is consistent and complete. By Lemma 3.5, $v(\Gamma^*) \vDash \varphi$ iff $\varphi \in \Gamma^*$. From this it follows in particular that for all $\varphi \in \Gamma$, $v(\Gamma^*) \vDash \varphi$, so Γ is satisfiable. □

Corollary 3.7 (Completeness Theorem, Second Version). *For all Γ and formulas φ: if $\Gamma \vDash \varphi$ then $\Gamma \vdash \varphi$.*

Proof. Note that the Γ's in Corollary 3.7 and Theorem 3.6 are universally quantified. To make sure we do not confuse ourselves, let us restate Theorem 3.6 using a different variable: for any set of formulas Δ, if Δ is consistent, it is satisfiable. By contraposition, if Δ is not satisfiable, then Δ is inconsistent. We will use this to prove the corollary.

Suppose that $\Gamma \vDash \varphi$. Then $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable by Proposition 1.20. Taking $\Gamma \cup \{\neg\varphi\}$ as our Δ, the previous version of Theorem 3.6 gives us that $\Gamma \cup \{\neg\varphi\}$ is inconsistent. By Proposition 2.16, $\Gamma \vdash \varphi$. □

3.7 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 formulas 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 formulas 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 formulas each finite subset of which is satisfiable. Like the completeness theorem, it has a version related to entailment: if an infinite set of formulas entails something, already a finite subset does.

Definition 3.8. A set Γ of formulas is *finitely satisfiable* if and only if every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable.

Theorem 3.9 (Compactness Theorem). *The following hold for any sentences* Γ *and* φ*:*

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

Proof. We prove (2). If Γ is satisfiable, then there is a valuation v such that $v \vDash \varphi$ for all $\varphi \in \Gamma$. Of course, this 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 2.24), every finite subset is consistent. Then Γ itself must be consistent by Proposition 2.14. By completeness (Theorem 3.6), since Γ is consistent, it is satisfiable. □

3.8 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 formulas, expanded it to a consistent and complete set Γ^* of formulas, and then showed that in the valuation $v(\Gamma^*)$ constructed from Γ^*, all formulas 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 3.10. *Suppose* Γ *is complete and finitely satisfiable. Then:*

1. $(\varphi \wedge \psi) \in \Gamma$ *iff both* $\varphi \in \Gamma$ *and* $\psi \in \Gamma$.
2. $(\varphi \vee \psi) \in \Gamma$ *iff either* $\varphi \in \Gamma$ *or* $\psi \in \Gamma$.
3. $(\varphi \to \psi) \in \Gamma$ *iff either* $\varphi \notin \Gamma$ *or* $\psi \in \Gamma$.

Lemma 3.11. *Every finitely satisfiable set* Γ *can be extended to a complete and finitely satisfiable set* Γ^*.

Theorem 3.12 (Compactness). Γ *is satisfiable if and only if it is finitely satisfiable.*

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

Now suppose that Γ is finitely satisfiable. By Lemma 3.11, Γ can be extended to a complete and finitely satisfiable set Γ^*. Construct the valuation $v(\Gamma^*)$ as in Definition 3.4. The proof of the Truth Lemma (Lemma 3.5) goes through if we replace references to Proposition 3.2. □

Problems

Problem 3.1. Complete the proof of Proposition 3.2.

Problem 3.2. Use Corollary 3.7 to prove Theorem 3.6, thus showing that the two formulations of the completeness theorem are equivalent.

Problem 3.3. 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 3.6. Be sure to note any tacit uses of rules in these proofs.

Problem 3.4. Prove (1) of Theorem 3.9.

Problem 3.5. Prove Proposition 3.10. Avoid the use of $\vdash$.

Problem 3.6. Prove Lemma 3.11. (Hint: the crucial step is to show that if Γ_n is finitely satisfiable, then either $\Gamma_n \cup \{\varphi_n\}$ or $\Gamma_n \cup \{\neg\varphi_n\}$ is finitely satisfiable.)

Problem 3.7. Write out the complete proof of the Truth Lemma (Lemma 3.5) in the version required for the proof of Theorem 3.12.

Part II

First-order Logic

Chapter 4

Syntax and Semantics

4.1 Introduction

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

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

satisfaction of $\varphi \wedge \psi$ in terms of satisfaction (or not) of φ and ψ. It then turns out that the variable assignment is irrelevant if the formula φ is a sentence, i.e., has no free variables, and so we can talk of sentences being simply satisfied (or not) in structures.

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

4.2 First-Order Languages

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

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

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

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

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

Example 4.1. 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 4.2. The language of set theory $\mathcal{L}_Z$ contains only the single two-place predicate symbol $\in$.

Example 4.3. The language of orders $\mathcal{L}_\leq$ contains only the two-place predicate symbol $\leq$.

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

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

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

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

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

We might treat all the propositional operators and both quantifiers as primitive symbols of the language. We might instead choose a smaller stock of primitive symbols and treat the other logical operators as defined. "Truth functionally complete" sets of Boolean operators include $\{\neg, \vee\}$, $\{\neg, \wedge\}$, and $\{\neg, \rightarrow\}$—these can be combined with either quantifier for an expressively complete first-order language.

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

4.3 Terms and Formulas

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

Definition 4.4 (Terms). The set of *terms* $\mathrm{Trm}(\mathcal{L})$ of $\mathcal{L}$ is defined inductively by:

1. Every variable is a term.
2. Every constant symbol of $\mathcal{L}$ is a term.
3. If f is an n-place function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

4. Nothing else is a term.

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

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

Definition 4.5 (Formula). The set of *formulas* $\mathrm{Frm}(\mathcal{L})$ of the language $\mathcal{L}$ is defined inductively as follows:

1. $\bot$ is an atomic formula.
2. If R is an n-place predicate symbol of $\mathcal{L}$ and $t_1, \ldots, t_n$ are terms of $\mathcal{L}$, then $R(t_1, \ldots, t_n)$ is an atomic formula.
3. If t_1 and t_2 are terms of $\mathcal{L}$, then $=(t_1, t_2)$ is an atomic formula.
4. If φ is a formula, then $\neg\varphi$ is formula.
5. If φ and ψ are formulas, then $(\varphi \wedge \psi)$ is a formula.
6. If φ and ψ are formulas, then $(\varphi \vee \psi)$ is a formula.
7. If φ and ψ are formulas, then $(\varphi \to \psi)$ is a formula.
8. If φ is a formula and x is a variable, then $\forall x\, \varphi$ is a formula.
9. If φ is a formula and x is a variable, then $\exists x\, \varphi$ is a formula.
10. Nothing else is a formula.

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

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

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

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

Definition 4.6. Formulas constructed using the defined operators are to be understood as follows:

1. $\top$ abbreviates $\neg\bot$.

2. $\varphi \leftrightarrow \psi$ abbreviates $(\varphi \to \psi) \wedge (\psi \to \varphi)$.

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

Definition 4.7 (Syntactic identity). The symbol $\equiv$ expresses syntactic identity between strings of symbols, i.e., $\varphi \equiv \psi$ iff φ and ψ are strings of symbols of the same length and which contain the same symbol in each place.

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

4.4 Unique Readability

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

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

> If φ and ψ are formulas, then so is $\varphi \to \psi$.

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

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

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

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

Lemma 4.8. *The number of left and right parentheses in a formula φ are equal.*

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

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

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

Definition 4.9 (Proper prefix). A string of symbols ψ is a *proper prefix* of a string of symbols φ if concatenating ψ and a non-empty string of symbols yields φ.

Lemma 4.10. *If φ is a formula, and ψ is a proper prefix of φ, then ψ is not a formula.*

Proof. Exercise. □

Proposition 4.11. *If φ is an atomic formula, then it satisfes one, and only one of the following conditions.*

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

Proof. Exercise. □

Proposition 4.12 (Unique Readability). *Every formula satisfies one, and only one of the following conditions.*

1. *φ is atomic.*
2. *φ is of the form $\neg\psi$.*

3. *φ is of the form $(\psi \wedge \chi)$.*
4. *φ is of the form $(\psi \vee \chi)$.*
5. *φ is of the form $(\psi \to \chi)$.*
6. *φ is of the form $\forall x\, \psi$.*
7. *φ is of the form $\exists x\, \psi$.*

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

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

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

So suppose both $\varphi \equiv (\psi * \chi)$ and $\varphi \equiv (\psi' *' \chi')$. Then either $\psi \equiv \psi'$ or not. If it is, clearly $* = *'$ and $\chi \equiv \chi'$, since they then are substrings of φ that begin in the same place and are of the same length. The other case is $\psi \not\equiv \psi'$. Since ψ and ψ' are both substrings of φ that begin at the same place, one must be a proper prefix of the other. But this is impossible by Lemma 4.10. □

4.5 Main operator of a Formula

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

Definition 4.13 (Main operator). The *main operator* of a formula φ is defined as follows:

1. φ is atomic: φ has no main operator.
2. $\varphi \equiv \neg\psi$: the main operator of φ is $\neg$.
3. $\varphi \equiv (\psi \wedge \chi)$: the main operator of φ is $\wedge$.
4. $\varphi \equiv (\psi \vee \chi)$: the main operator of φ is $\vee$.
5. $\varphi \equiv (\psi \to \chi)$: the main operator of φ is $\to$.
6. $\varphi \equiv \forall x\, \psi$: the main operator of φ is $\forall$.
7. $\varphi \equiv \exists x\, \psi$: the main operator of φ is $\exists$.

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

This is a *recursive* definition of a function which maps all non-atomic formulas to their main operator occurrence. Because of the way formulas are defined inductively,

every formula φ satisfies one of the cases in Definition 4.13. This guarantees that for each non-atomic formula φ a main operator exists. Because each formula satisfies only one of these conditions, and because the smaller formulas from which φ is constructed are uniquely determined in each case, the main operator occurrence of φ is unique, and so we have defined a function.

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

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

4.6 Subformulas

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

Definition 4.14 (Immediate Subformula). If φ is a formula, the *immediate subformulas* of φ are defined inductively as follows:

1. Atomic formulas have no immediate subformulas.
2. $\varphi \equiv \neg\psi$: The only immediate subformula of φ is ψ.
3. $\varphi \equiv (\psi * \chi)$: The immediate subformulas of φ are ψ and χ ($*$ is any one of the two-place connectives).
4. $\varphi \equiv \forall x\, \psi$: The only immediate subformula of φ is ψ.
5. $\varphi \equiv \exists x\, \psi$: The only immediate subformula of φ is ψ.

Definition 4.15 (Proper Subformula). If φ is a formula, the *proper subformulas* of φ are recursively as follows:

1. Atomic formulas have no proper subformulas.
2. $\varphi \equiv \neg\psi$: The proper subformulas of φ are ψ together with all proper subformulas of ψ.
3. $\varphi \equiv (\psi * \chi)$: The proper subformulas of φ are ψ, χ, together with all proper subformulas of ψ and those of χ.
4. $\varphi \equiv \forall x\, \psi$: The proper subformulas of φ are ψ together with all proper subformulas of ψ.
5. $\varphi \equiv \exists x\, \psi$: The proper subformulas of φ are ψ together with all proper subformulas of ψ.

Definition 4.16 (Subformula). The subformulas of φ are φ itself together with all its proper subformulas.

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

4.7 Free Variables and Sentences

Definition 4.17 (Free occurrences of a variable). The *free* occurrences of a variable in a formula are defined inductively as follows:

1. φ is atomic: all variable occurrences in φ are free.
2. $\varphi \equiv \neg\psi$: the free variable occurrences of φ are exactly those of ψ.
3. $\varphi \equiv (\psi * \chi)$: the free variable occurrences of φ are those in ψ together with those in χ.
4. $\varphi \equiv \forall x\, \psi$: the free variable occurrences in φ are all of those in ψ except for occurrences of x.
5. $\varphi \equiv \exists x\, \psi$: the free variable occurrences in φ are all of those in ψ except for occurrences of x.

Definition 4.18 (Bound Variables). An occurrence of a variable in a formula φ is *bound* if it is not free.

Definition 4.19 (Scope). If $\forall x\, \psi$ is an occurrence of a subformula in a formula φ, then the corresponding occurrence of ψ in φ is called the *scope* of the corresponding occurrence of $\forall x$. Similarly for $\exists x$.

If ψ is the scope of a quantifier occurrence $\forall x$ or $\exists x$ in φ, then the free occurrences of x in ψ are bound in $\forall x\, \psi$ and $\exists x\, \psi$. We say that these occurrences are *bound by* the mentioned quantifier occurrence.

Example 4.20. Consider the following formula:

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

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

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

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

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

Definition 4.21 (Sentence). A formula φ is a *sentence* iff it contains no free occurrences of variables.

4.8 Substitution

Definition 4.22 (Substitution in a term). We define $s[t/x]$, the result of *substituting* t for every occurrence of x in s, recursively:

1. $s \equiv c$: $s[t/x]$ is just s.
2. $s \equiv y$: $s[t/x]$ is also just s, provided y is a variable and $y \not\equiv x$.
3. $s \equiv x$: $s[t/x]$ is t.
4. $s \equiv f(t_1, \ldots, t_n)$: $s[t/x]$ is $f(t_1[t/x], \ldots, t_n[t/x])$.

Definition 4.23. A term t is *free for* x in φ if none of the free occurrences of x in φ occur in the scope of a quantifier that binds a variable in t.

Example 4.24.

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

Definition 4.25 (Substitution in a formula). If φ is a formula, x is a variable, and t is a term free for x in φ, then $\varphi[t/x]$ is the result of substituting t for all free occurrences of x in φ.

1. $\varphi \equiv \bot$: $\varphi[t/x]$ is $\bot$.
2. $\varphi \equiv P(t_1, \ldots, t_n)$: $\varphi[t/x]$ is $P(t_1[t/x], \ldots, t_n[t/x])$.
3. $\varphi \equiv t_1 = t_2$: $\varphi[t/x]$ is $t_1[t/x] = t_2[t/x]$.
4. $\varphi \equiv \neg\psi$: $\varphi[t/x]$ is $\neg\psi[t/x]$.
5. $\varphi \equiv (\psi \wedge \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \wedge \chi[t/x])$.
6. $\varphi \equiv (\psi \vee \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \vee \chi[t/x])$.
7. $\varphi \equiv (\psi \to \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \to \chi[t/x])$.

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

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

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

The restriction that t must be free for x in φ is necessary to exclude cases like the following. If $\varphi \equiv \exists y\, x < y$ and $t \equiv y$, then $\varphi[t/x]$ would be $\exists y\, y < y$. In this case the free variable y is "captured" by the quantifier $\exists y$ upon substitution, and that is undesirable. For instance, we would like it to be the case that whenever $\forall x\, \psi$ holds, so does $\psi[t/x]$. But consider $\forall x \exists y\, x < y$ (here ψ is $\exists y\, x < y$). It is sentence that is true about, e.g., the natural numbers: for every number x there is a number y greater than it. If we allowed y as a possible substitution for x, we would end up with $\psi[y/x] \equiv \exists y\, y < y$, which is false. We prevent this by requiring that none of the free variables in t would end up being bound by a quantifier in φ.

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

4.9 Structures for First-order Languages

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

Definition 4.26 (Structures). A *structure* $\mathfrak{M}$, for a language $\mathcal{L}$ of first-order logic consists of the following elements:

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

Example 4.27. A structure $\mathfrak{M}$ for the language of arithmetic consists of a set, an element of $|\mathfrak{M}|$, $0^{\mathfrak{M}}$, as interpretation of the constant symbol 0, a one-place function $'^{\mathfrak{M}} \colon |\mathfrak{M}| \to |\mathfrak{M}|$, two two-place functions $+^{\mathfrak{M}}$ and $\times^{\mathfrak{M}}$, both $|\mathfrak{M}|^2 \to |\mathfrak{M}|$, and a two-place relation $<^{\mathfrak{M}} \subseteq |\mathfrak{M}|^2$.

An obvious example of such a structure is the following:

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

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

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

Example 4.28. A structure $\mathfrak{M}$ for the language $\mathcal{L}_Z$ of set theory requires just a set and a single-two place relation. So technically, e.g., the set of people plus the relation "x is older than y" could be used as a structure for $\mathcal{L}_Z$, as well as $\mathbb{N}$ together with $n \geq m$ for $n, m \in \mathbb{N}$.

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

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

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

4.10 Covered Structures for First-order Languages

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

Definition 4.29 (Value of closed terms). If t is a closed term of the language $\mathcal{L}$ and $\mathfrak{M}$ is a structure for $\mathcal{L}$, the *value* $\mathrm{Val}^{\mathfrak{M}}(t)$ is defined as follows:

1. If t is just the constant symbol c, then $\mathrm{Val}^{\mathfrak{M}}(c) = c^{\mathfrak{M}}$.
2. If t is of the form $f(t_1, \ldots, t_n)$, then

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

Definition 4.30 (Covered structure). A structure is *covered* if every element of the domain is the value of some closed term.

Example 4.31. Let $\mathcal{L}$ be the language with constant symbols $zero$, one, two, ..., the binary predicate symbol $<$, and the binary function symbols $+$ and $\times$. Then a structure $\mathfrak{M}$ for $\mathcal{L}$ is the one with domain $|\mathfrak{M}| = \{0, 1, 2, \ldots\}$ and assignments $zero^{\mathfrak{M}} = 0$, $one^{\mathfrak{M}} = 1$, $two^{\mathfrak{M}} = 2$, and so forth. For the binary relation symbol $<$, the set $<^{\mathfrak{M}}$ is the set of all pairs $\langle c_1, c_2\rangle \in |\mathfrak{M}|^2$ such that c_1 is less than c_2: for example, $\langle 1, 3\rangle \in <^{\mathfrak{M}}$ but $\langle 2, 2\rangle \notin <^{\mathfrak{M}}$. For the binary function symbol $+$, define $+^{\mathfrak{M}}$ in the usual way—for example, $+^{\mathfrak{M}}(2, 3)$ maps to 5, and similarly for the binary function symbol $\times$. Hence, the value of $four$ is just 4, and the value of $\times(two, +(three, zero))$ (or in infix notation, $two \times (three + zero)$) is

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

4.11 Satisfaction of a Formula in a Structure

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

Definition 4.32 (Variable Assignment). A *variable assignment* s for a structure $\mathfrak{M}$ is a function which maps each variable to an element of $|\mathfrak{M}|$, i.e., $s\colon \mathrm{Var} \to |\mathfrak{M}|$.

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

Definition 4.33 (Value of Terms). If t is a term of the language $\mathcal{L}$, $\mathfrak{M}$ is a structure for $\mathcal{L}$, and s is a variable assignment for $\mathfrak{M}$, the *value* $\mathrm{Val}^{\mathfrak{M}}_s(t)$ is defined as follows:

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

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

Definition 4.34 (x-Variant). If s is a variable assignment for a structure $\mathfrak{M}$, then any variable assignment s' for $\mathfrak{M}$ which differs from s at most in what it assigns to x is called an *x-variant* of s. If s' is an x-variant of s we write $s \sim_x s'$.

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

Definition 4.35 (Satisfaction). Satisfaction of a formula φ in a structure $\mathfrak{M}$ relative to a variable assignment s, in symbols: $\mathfrak{M}, s \vDash \varphi$, is defined recursively as follows. (We write $\mathfrak{M}, s \nvDash \varphi$ to mean "not $\mathfrak{M}, s \vDash \varphi$.")

1. $\varphi \equiv \bot$: $\mathfrak{M}, s \nvDash \varphi$.
2. $\varphi \equiv R(t_1, \ldots, t_n)$: $\mathfrak{M}, s \vDash \varphi$ iff $\langle \mathrm{Val}^{\mathfrak{M}}_s(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_s(t_n)\rangle \in R^{\mathfrak{M}}$.
3. $\varphi \equiv t_1 = t_2$: $\mathfrak{M}, s \vDash \varphi$ iff $\mathrm{Val}^{\mathfrak{M}}_s(t_1) = \mathrm{Val}^{\mathfrak{M}}_s(t_2)$.
4. $\varphi \equiv \neg\psi$: $\mathfrak{M}, s \vDash \varphi$ iff $\mathfrak{M}, s \nvDash \psi$.
5. $\varphi \equiv (\psi \wedge \chi)$: $\mathfrak{M}, s \vDash \varphi$ iff $\mathfrak{M}, s \vDash \psi$ and $\mathfrak{M}, s \vDash \chi$.
6. $\varphi \equiv (\psi \vee \chi)$: $\mathfrak{M}, s \vDash \varphi$ iff $\mathfrak{M}, s \vDash \varphi$ or $\mathfrak{M}, s \vDash \psi$ (or both).
7. $\varphi \equiv (\psi \to \chi)$: $\mathfrak{M}, s \vDash \varphi$ iff $\mathfrak{M}, s \nvDash \psi$ or $\mathfrak{M}, s \vDash \chi$ (or both).
8. $\varphi \equiv \forall x\, \psi$: $\mathfrak{M}, s \vDash \varphi$ iff for every x-variant s' of s, $\mathfrak{M}, s' \vDash \psi$.
9. $\varphi \equiv \exists x\, \psi$: $\mathfrak{M}, s \vDash \varphi$ iff there is an x-variant s' of s so that $\mathfrak{M}, s' \vDash \psi$.

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

Example 4.36. 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 $\mathfrak{M}$ defined by:

1. $|\mathfrak{M}| = \{1, 2, 3, 4\}$

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

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

Then

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

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

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

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

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

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

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

An atomic formula $R(t_1, t_2)$ is satisfied if the tuple of values of its arguments, i.e., $\langle \mathrm{Val}^{\mathfrak{M}}_s(t_1), \mathrm{Val}^{\mathfrak{M}}_s(t_2)\rangle$, is an element of $R^{\mathfrak{M}}$. So, e.g., we have $\mathfrak{M}, s \vDash R(b, f(a, b))$ since $\langle \mathrm{Val}^{\mathfrak{M}}(b), \mathrm{Val}^{\mathfrak{M}}(f(a, b))\rangle = \langle 2, 3\rangle \in R^{\mathfrak{M}}$, but $\mathfrak{M}, s \nvDash R(x, f(a, b))$ since $\langle 1, 3\rangle \notin R^{\mathfrak{M}}[s]$.

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

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

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

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

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

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

Note, in particular, that $s_1 = s$ is also an x-variant of s, i.e., s is always an x-variant of itself.

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

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

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

$$\mathfrak{M}, s \nvDash \exists x\, (R(b, x) \wedge R(x, b))$$

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

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

$$\mathfrak{M}, s \vDash \forall x\, (R(x, a) \rightarrow R(a, x)),$$

since $\mathfrak{M}, s_i \vDash R(x, a) \rightarrow R(a, x)$ for all s_i ($\mathfrak{M}, s_1 \vDash R(a, x)$ and $\mathfrak{M}, s_j \nvDash R(x, a)$ for $j = 2$, 3, and 4). But,

$$\mathfrak{M}, s \nvDash \forall x\, (R(a, x) \rightarrow R(x, a))$$

since $\mathfrak{M}, s_2 \nvDash R(a, x) \rightarrow R(x, a)$ (because $\mathfrak{M}, s_2 \vDash R(a, x)$ and $\mathfrak{M}, s_2 \nvDash R(x, a)$).

For a more complicated case, consider

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

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

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

On the other hand,

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

The only x-variants s_i of s with $\mathfrak{M}, s_i \vDash R(a, x)$ are s_1 and s_2. But for each, there is in turn a y-variant $s_i' \sim_y s_i$ with $s_i'(y) = 4$ so that $\mathfrak{M}, s_i' \nvDash R(x, y)$ and so $\mathfrak{M}, s_i \nvDash \forall y\, R(x, y)$ for $i = 1, 2$. In sum, none of the x-variants $s_i \sim_x s$ are such that $\mathfrak{M}, s_i \vDash R(a, x) \wedge \forall y\, R(x, y)$.

4.12 Variable Assignments

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

on" precise, we show that any two variable assignments that agree on all the variables in t give the same value, and that φ is satisfied relative to one iff it is satisfied relative to the other if two variable assignments agree on all free variables of φ.

Proposition 4.37. *If the variables in a term t are among $x_1, \ldots, x_n$, and $s_1(x_i) = s_2(x_i)$ for $i = 1, \ldots, n$, then $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t) = \mathrm{Val}^{\mathfrak{M}}_{s_2}(t)$.*

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

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

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

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

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

□

Proposition 4.38. *If the free variables in φ are among $x_1, \ldots, x_n$, and $s_1(x_i) = s_2(x_i)$ for $i = 1, \ldots, n$, then $\mathfrak{M}, s_1 \vDash \varphi$ iff $\mathfrak{M}, s_2 \vDash \varphi$.*

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

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

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

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

   For $i = 1, \ldots, k$, $\mathrm{Val}^{\mathfrak{M}}_{s_1}(t_i) = \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_i)$ by Proposition 4.37. So we also have $\langle \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_i), \ldots, \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_k) \rangle \in R^{\mathfrak{M}}$.

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

   $$\begin{aligned}\mathrm{Val}^{\mathfrak{M}}_{s_2}(t_1) &= \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_1) && \text{(by Proposition 4.37)} \\ &= \mathrm{Val}^{\mathfrak{M}}_{s_1}(t_2) && \text{(since } \mathfrak{M}, s_1 \vDash t_1 = t_2\text{)} \\ &= \mathrm{Val}^{\mathfrak{M}}_{s_2}(t_2) && \text{(by Proposition 4.37),}\end{aligned}$$

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

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

1. $\varphi \equiv \neg\psi$: if $\mathfrak{M}, s_1 \vDash \varphi$, then $\mathfrak{M}, s_1 \nvDash \psi$, so by the induction hypothesis, $\mathfrak{M}, s_2 \nvDash \psi$, hence $\mathfrak{M}, s_2 \vDash \varphi$.

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

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

4. $\varphi \equiv \psi \to \chi$: if $\mathfrak{M}, s_1 \vDash \varphi$, then $\mathfrak{M}, s_1 \nvDash \psi$ or $\mathfrak{M}, s_1 \vDash \chi$. By the induction hypothesis, $\mathfrak{M}, s_2 \nvDash \psi$ or $\mathfrak{M}, s_2 \vDash \chi$, so $\mathfrak{M}, s_2 \vDash \varphi$.

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

6. $\varphi \equiv \forall x\, \psi$: if $\mathfrak{M}, s_1 \vDash \varphi$, then for every x-variant s_1' of s_1, $\mathfrak{M}, s_1' \vDash \psi$. Take an arbitrary x-variant s_2' of s_2, let s_1' be the x-variant of s_1 which assigns the same thing to x as does s_2'. The free variables of ψ are among $x_1, \ldots, x_n$, and x. $s_1'(x_i) = s_2'(x_i)$, since s_1' and s_2' are x-variants of s_1 and s_2, respectively, and by hypothesis $s_1(x_i) = s_2(x_i)$. $s_1'(x) = s_2'(x)$ by the way we have defined s_1'. Then the induction hypothesis applies to ψ and s_1', s_2', and we have $\mathfrak{M}, s_2' \vDash \psi$. Since s_2' is an arbitrary x-variant of s_2, every x-variant of s_2 satisfies ψ, and so $\mathfrak{M}, s_2 \vDash \varphi$.

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

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

Corollary 4.39. *If φ is a sentence and s a variable assignment, then $\mathfrak{M}, s \vDash \varphi$ iff $\mathfrak{M}, s' \vDash \varphi$ for every variable assignment s'.*

Proof. Let s' be any variable assignment. Since φ is a sentence, it has no free variables, and so every variable assignment s' trivially assigns the same things to all free variables of φ as does s. So the condition of Proposition 4.38 is satisfied, and we have $\mathfrak{M}, s \vDash \varphi$ iff $\mathfrak{M}, s' \vDash \varphi$. □

Definition 4.40. If φ is a sentence, we say that a structure $\mathfrak{M}$ *satisfies* φ, $\mathfrak{M} \vDash \varphi$, iff $\mathfrak{M}, s \vDash \varphi$ for all variable assignments s.

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

Proposition 4.41. *Let $\mathfrak{M}$ be a structure, φ be a sentence, and s a variable assignment. $\mathfrak{M} \vDash \varphi$ iff $\mathfrak{M}, s \vDash \varphi$.*

Proof. Exercise. □

Proposition 4.42. *Suppose $\varphi(x)$ only contains x free, and $\mathfrak{M}$ is a structure. Then:*

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

Proof. Exercise. □

4.13 Extensionality

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

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

Proposition 4.43 (Extensionality). *Let φ be a formula, and $\mathfrak{M}_1$ and $\mathfrak{M}_2$ be structures with $|\mathfrak{M}_1| = |\mathfrak{M}_2|$, and s a variable assignment on $|\mathfrak{M}_1| = |\mathfrak{M}_2|$. If $c^{\mathfrak{M}_1} = c^{\mathfrak{M}_2}$, $R^{\mathfrak{M}_1} = R^{\mathfrak{M}_2}$, and $f^{\mathfrak{M}_1} = f^{\mathfrak{M}_2}$ for every constant symbol c, relation symbol R, and function symbol f occurring in φ, then $\mathfrak{M}_1, s \vDash \varphi$ iff $\mathfrak{M}_2, s \vDash \varphi$.*

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

Corollary 4.44 (Extensionality for Sentences). *Let φ be a sentence and $\mathfrak{M}_1$, $\mathfrak{M}_2$ as in Proposition 4.43. Then $\mathfrak{M}_1 \vDash \varphi$ iff $\mathfrak{M}_2 \vDash \varphi$.*

Proof. Follows from Proposition 4.43 by Corollary 4.39. □

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

Proposition 4.45. *Let $\mathfrak{M}$ be a structure, t and t' terms, and s a variable assignment. Let $s' \sim_x s$ be the x-variant of s given by $s'(x) = \mathrm{Val}_s^{\mathfrak{M}}(t')$. Then $\mathrm{Val}_s^{\mathfrak{M}}(t[t'/x]) = \mathrm{Val}_{s'}^{\mathfrak{M}}(t)$.*

Proof. By induction on t.

1. If t is a constant, say, $t \equiv c$, then $t[t'/x] = c$, and $\mathrm{Val}_s^{\mathfrak{M}}(c) = c^{\mathfrak{M}} = \mathrm{Val}_{s'}^{\mathfrak{M}}(c)$.
2. If t is a variable other than x, say, $t \equiv y$, then $t[t'/x] = y$, and $\mathrm{Val}_s^{\mathfrak{M}}(y) = \mathrm{Val}_{s'}^{\mathfrak{M}}(y)$ since $s' \sim_x s$.

3. If $t \equiv x$, then $t[t'/x] = t'$. But $\mathrm{Val}^{\mathfrak{M}}_{s'}(x) = \mathrm{Val}^{\mathfrak{M}}_{s}(t')$ by definition of s'.

4. If $t \equiv f(t_1, \ldots, t_n)$ then we have:

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

Proposition 4.46. *Let $\mathfrak{M}$ be a structure, φ a formula, t a term, and s a variable assignment. Let $s' \sim_x s$ be the x-variant of s given by $s'(x) = \mathrm{Val}^{\mathfrak{M}}_{s}(t)$. Then $\mathfrak{M}, s \vDash \varphi[t/x]$ iff $\mathfrak{M}, s' \vDash \varphi$.*

Proof. Exercise. □

4.14 Semantic Notions

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

Definition 4.47 (Validity). A sentence φ is *valid*, $\vDash \varphi$, iff $\mathfrak{M} \vDash \varphi$ for every structure $\mathfrak{M}$.

Definition 4.48 (Entailment). A set of sentences Γ *entails* a sentence φ, $\Gamma \vDash \varphi$, iff for every structure $\mathfrak{M}$ with $\mathfrak{M} \vDash \Gamma$, $\mathfrak{M} \vDash \varphi$.

Definition 4.49 (Satisfiability). A set of sentences Γ is *satisfiable* if $\mathfrak{M} \vDash \Gamma$ for some structure $\mathfrak{M}$. If Γ is not satisfiable it is called *unsatisfiable*.

Proposition 4.50. *A sentence φ is valid iff $\Gamma \vDash \varphi$ for every set of sentences Γ.*

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

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

Proposition 4.51. *$\Gamma \vDash \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable.*

Proof. For the forward direction, suppose $\Gamma \vDash \varphi$ and suppose to the contrary that there is a structure $\mathfrak{M}$ so that $\mathfrak{M} \vDash \Gamma \cup \{\neg\varphi\}$. Since $\mathfrak{M} \vDash \Gamma$ and $\Gamma \vDash \varphi$, $\mathfrak{M} \vDash \varphi$. Also, since $\mathfrak{M} \vDash \Gamma \cup \{\neg\varphi\}$, $\mathfrak{M} \vDash \neg\varphi$, so we have both $\mathfrak{M} \vDash \varphi$ and $\mathfrak{M} \nvDash \varphi$, a contradiction. Hence, there can be no such structure $\mathfrak{M}$, so $\Gamma \cup \{\varphi\}$ is unsatisfiable.

For the reverse direction, suppose $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable. So for every structure $\mathfrak{M}$, either $\mathfrak{M} \nvDash \Gamma$ or $\mathfrak{M} \vDash \varphi$. Hence, for every structure $\mathfrak{M}$ with $\mathfrak{M} \vDash \Gamma$, $\mathfrak{M} \vDash \varphi$, so $\Gamma \vDash \varphi$. □

Proposition 4.52. *If $\Gamma \subseteq \Gamma'$ and $\Gamma \vDash \varphi$, then $\Gamma' \vDash \varphi$.*

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

Theorem 4.53 (Semantic Deduction Theorem). *$\Gamma \cup \{\varphi\} \vDash \psi$ iff $\Gamma \vDash \varphi \to \psi$.*

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

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

Proposition 4.54. *Let $\mathfrak{M}$ be a structure, and $\varphi(x)$ a formula with one free variable x, and t a closed term. Then:*

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

Proof.

1. Suppose $\mathfrak{M} \vDash \varphi(t)$. Let s be a variable assignment with $s(x) = \mathrm{Val}^{\mathfrak{M}}(t)$. Then $\mathfrak{M}, s \vDash \varphi(t)$ since $\varphi(t)$ is a sentence. By Proposition 4.46, $\mathfrak{M}, s \vDash \varphi(x)$. By Proposition 4.42, $\mathfrak{M} \vDash \exists x\, \varphi(x)$.

2. Suppose $\mathfrak{M} \vDash \forall x\, \varphi(x)$. Let s be a variable assignment with $s(x) = \mathrm{Val}^{\mathfrak{M}}(t)$. By Proposition 4.42, $\mathfrak{M}, s \vDash \varphi(x)$. By Proposition 4.46, $\mathfrak{M}, s \vDash \varphi(t)$. By Proposition 4.41, $\mathfrak{M} \vDash \varphi(t)$ since $\varphi(t)$ is a sentence. □

Problems

Problem 4.1. Prove Lemma 4.10.

Problem 4.2. Prove Proposition 4.11 (Hint: Formulate and prove a version of Lemma 4.10 for terms.)

Problem 4.3. Give an inductive definition of the bound variable occurrences along the lines of Definition 4.17.

Problem 4.4. Is $\mathfrak{N}$, the standard model of arithmetic, covered? Explain.

Problem 4.5. Let $\mathcal{L} = \{c, f, A\}$ with one constant symbol, one one-place function symbol and one two-place predicate symbol, and let the structure $\mathfrak{M}$ be given by

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

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

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

Explain why or why not.

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

Problem 4.6. Complete the proof of Proposition 4.38.

Problem 4.7. Prove Proposition 4.41

Problem 4.8. Prove Proposition 4.42.

Problem 4.9. Suppose $\mathcal{L}$ is a language without function symbols. Given a structure $\mathfrak{M}$, c a constant symbol and $a \in |\mathfrak{M}|$, define $\mathfrak{M}[a/c]$ to be the structure that is just like $\mathfrak{M}$, except that $c^{\mathfrak{M}[a/c]} = a$. Define $\mathfrak{M} \models \varphi$ for sentences φ by:

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

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

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

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

Problem 4.10. Suppose that f is a function symbol not in $\varphi(x, y)$. Show that there is a structure $\mathfrak{M}$ such that $\mathfrak{M} \vDash \forall x\, \exists y\, \varphi(x, y)$ iff there is an $\mathfrak{M}'$ such that $\mathfrak{M}' \vDash \forall x\, \varphi(x, f(x))$.

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

Problem 4.11. Carry out the proof of Proposition 4.43 in detail.

Problem 4.12. Prove Proposition 4.46

Problem 4.13.

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

Problem 4.14. Complete the proof of Proposition 4.54.

Chapter 5

Theories and Their Models

5.1 Introduction

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

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

Definition 5.1. A set of sentences Γ is *closed* iff, whenever $\Gamma \vDash \varphi$ then $\varphi \in \Gamma$. The *closure* of a set of sentences Γ is $\{\varphi \mid \Gamma \vDash \varphi\}$.

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

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

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

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

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

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

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

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

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

5.2 Expressing Properties of Structures

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

Definition 5.2 (Model of a set). Let Γ be a set of sentences in a language $\mathcal{L}$. We say that a structure $\mathfrak{M}$ *is a model of* Γ if $\mathfrak{M} \vDash \varphi$ for all $\varphi \in \Gamma$.

Example 5.3. The sentence $\forall x\, x \leq x$ is true in $\mathfrak{M}$ iff $\leq^{\mathfrak{M}}$ is a reflexive relation. The sentence $\forall x\, \forall y\, ((x \leq y \wedge y \leq x) \rightarrow x = y)$ is true in $\mathfrak{M}$ iff $\leq^{\mathfrak{M}}$ is anti-symmetric. The sentence $\forall x\, \forall y\, \forall z\, ((x \leq y \wedge y \leq z) \rightarrow x \leq z)$ is true in $\mathfrak{M}$ iff $\leq^{\mathfrak{M}}$ is transitive. Thus, the models of

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

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

5.3 Examples of First-Order Theories

Example 5.4. The theory of strict linear orders in the language $\mathcal{L}_<$ is axiomatized by the set

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

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

Example 5.5. The theory of groups in the language 1 (constant symbol), $\cdot$ (two-place function symbol) is axiomatized by

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

Example 5.6. The theory of Peano arithmetic is axiomatized by the following sentences in the language of arithmetic $\mathcal{L}_A$.

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

plus all sentences of the form

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

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

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

Example 5.7. The theory of pure sets plays an important role in the foundations (and in the philosophy) of mathematics. A set is pure if all its elements are also pure sets. The empty set counts therefore as pure, but a set that has something as an element that is not a set would not be pure. So the pure sets are those that are formed just from the empty set and no "urelements," i.e., objects that are not themselves sets.

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

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

plus all sentences of the form

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

The first axiom says that there is a set with no elements (i.e., $\emptyset$ exists); the second says that sets are extensional; the third that for any sets X and Y, the set $\{X, Y\}$ exists; the fourth that for any 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 $\varphi(x)$, the set $\{x \mid \varphi(x)\}$ exists—so at first glance a true, useful, and perhaps even necessary axiom. It is called "naive" because, as it turns out, it makes this theory unsatisfiable: if you take $\varphi(y)$ to be $\neg y \in y$, 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 5.8. In the area of *mereology*, the relation of *parthood* is a fundamental relation. Just like theories of sets, there are theories of parthood that axiomatize various conceptions (sometimes conflicting) of this relation.

The language of mereology contains a single two-place predicate symbol P, and $P(x, y)$ "means" that x is a part of y. When we have this interpretation in mind, a structure for this language is called a *parthood structure*. Of course, not every structure for a single two-place predicate will really deserve this name. To have a chance of capturing "parthood," $P^{\mathfrak{M}}$ must satisfy some conditions, which we can lay down as axioms for a theory of parthood. For instance, parthood is a partial order on objects: every object is a part (albeit an *improper* part) of itself; no two different objects can be parts of each other; a part of a part of an object is itself part of that

object. Note that in this sense "is a part of" resembles "is a subset of," but does not resemble "is an element of" which is neither reflexive nor transitive.

$$\forall x\, P(x,x),$$
$$\forall x\, \forall y\, ((P(x,y) \wedge P(y,x)) \rightarrow x = y),$$
$$\forall x\, \forall y\, \forall z\, ((P(x,y) \wedge P(y,z)) \rightarrow P(x,z)),$$

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

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

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

5.4 Expressing Relations in a Structure

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

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

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

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

Definition 5.9. Let $\varphi(v_1, \ldots, v_n)$ be a formula of $\mathcal{L}$ in which only $v_1, \ldots, v_n$ occur free, and let $\mathfrak{M}$ be a structure for $\mathcal{L}$. $\varphi(v_1, \ldots, v_n)$ *expresses the relation* $R \subseteq |\mathfrak{M}|^n$ iff

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

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

Example 5.10. In the standard model of arithmetic $\mathfrak{N}$, the formula $v_1 < v_2 \vee v_1 = v_2$ expresses the $\leq$ relation on $\mathbb{N}$. The formula $v_2 = v_1'$ expresses the successor relation, i.e., the relation $R \subseteq \mathbb{N}^2$ where Rnm holds if m is the successor of n. The formula $v_1 = v_2'$ expresses the predecessor relation. The formulas $\exists v_3\, (v_3 \neq 0 \wedge v_2 = (v_1 + v_3))$ and $\exists v_3\, (v_1 + v_3') = v_2$ both express the $<$ relation. This means that the predicate symbol $<$ is actually superfluous in the language of arithmetic; it can be defined.

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

5.5 The Theory of Sets

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5.6 Expressing the Size of Structures

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

Proposition 5.11. *The sentence*

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

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

Proposition 5.12. *The sentence*

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

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

Proposition 5.13. *A structure is infinite iff it is a model of*

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

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

Problems

Problem 5.1. Find formulas in $\mathcal{L}_A$ which define the following relations:

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

Problem 5.2. Suppose the formula $\varphi(v_1, v_2)$ expresses the relation $R \subseteq |\mathfrak{M}|^2$ in a structure $\mathfrak{M}$. Find formulas that express the following relations:

1. the inverse R^{-1} of R;
2. the relative product $R \mid R$;

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

Problem 5.3. Let $\mathcal{L}$ be the language containing a 2-place predicate symbol $<$ only (no other constant symbols, function symbols or predicate symbols— except of course $=$). Let $\mathfrak{N}$ be the structure such that $|\mathfrak{N}| = \mathbb{N}$, and $<^{\mathfrak{N}} = \{\langle n, m\rangle \mid n < m\}$. Prove the following:

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

Problem 5.4. 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 6

Natural Deduction

6.1 Introduction

To define a derivation system for first-order logic we will use what we already have for propositional logic and add rules for the quantifiers.

6.2 Quantifier Rules

Rules for $\forall$

$$\frac{\varphi(a)}{\forall x\, \varphi(x)}\,\forall\mathrm{I} \qquad\qquad \frac{\forall x\, \varphi(x)}{\varphi(t)}\,\forall\mathrm{E}$$

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

Rules for $\exists$

$$\frac{\varphi(t)}{\exists x\, \varphi(x)}\,\exists\mathrm{I} \qquad\qquad \frac{\exists x\, \varphi(x) \qquad \begin{array}{c}[\varphi(a)]^n \\ \vdots \\ \chi\end{array}}{\chi}\,\exists\mathrm{E}_n$$

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

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

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

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

$$\dfrac{\exists x\, \varphi(x) \qquad \dfrac{[\varphi(a)]^1}{\forall x\, \varphi(x)}\,{}^*\forall\mathrm{I}}{\forall x\, \varphi(x)}\,\exists\mathrm{E}$$

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

6.3 Derivations with Quantifiers

Example 6.1. When dealing with quantifiers, we have to make sure not to violate the eigenvariable condition, and sometimes this requires us to play around with the order of carrying out certain inferences. In general, it helps to try and take care of rules subject to the eigenvariable condition first (they will be lower down in the finished proof).

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

$$\dfrac{}{\exists x\, \neg\varphi(x) \to \neg\forall x\, \varphi(x)}$$

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

$$\dfrac{\begin{array}{c}[\exists x\, \neg\varphi(x)]^1 \\ \vdots \\ \neg\forall x\, \varphi(x)\end{array}}{\exists x\, \neg\varphi(x) \to \neg\forall x\, \varphi(x)}\,\to\mathrm{I}_1$$

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

$$\dfrac{\dfrac{[\exists x\, \neg\varphi(x)]^1 \qquad \begin{array}{c}[\neg\varphi(a)]^2 \\ \vdots \\ \neg\forall x\, \varphi(x)\end{array}}{\neg\forall x\, \varphi(x)}\,\exists\mathrm{E}_2}{\exists x\, \neg\varphi(x) \to \neg\forall x\, \varphi(x)}\,\to\mathrm{I}_1$$

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

$$\dfrac{\dfrac{[\exists x\,\neg\varphi(x)]^1 \qquad \dfrac{[\neg\varphi(a)]^2,\ [\forall x\,\varphi(x)]^3}{\vdots}}{\dfrac{\neg\forall x\,\varphi(x)}{\bot}\neg\mathrm{I}_3}\exists\mathrm{E}_2}{\exists x\,\neg\varphi(x) \to \neg\forall x\,\varphi(x)}\to\mathrm{I}_1$$

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

$$\dfrac{\dfrac{[\exists x\,\neg\varphi(x)]^1 \qquad \dfrac{\dfrac{[\neg\varphi(a)]^2 \qquad \dfrac{[\forall x\,\varphi(x)]^3}{\varphi(a)}\forall\mathrm{E}}{\bot}\neg\mathrm{E}}{\neg\forall x\,\varphi(x)}\neg\mathrm{I}_3}{\neg\forall x\,\varphi(x)}\exists\mathrm{E}_2}{\exists x\,\neg\varphi(x) \to \neg\forall x\,\varphi(x)}\to\mathrm{I}_1$$

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

Example 6.2. Sometimes we may derive a formula from other formulas. In these cases, we may have undischarged assumptions. It is important to keep track of our assumptions as well as the end goal.

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

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

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

$$\dfrac{\exists x\,(\varphi(x) \wedge \psi(x)) \qquad \dfrac{[\varphi(a) \wedge \psi(a)]^1}{\vdots}}{\exists x\,\chi(x,b)}\exists\mathrm{E}_1$$

The two assumptions we are working with share ψ. It may be useful at this point to apply $\wedge$E to separate out $\psi(a)$.

$$\dfrac{\exists x\,(\varphi(x) \wedge \psi(x)) \qquad \exists x\,\chi(x,b)}{\exists x\,\chi(x,b)}\,\exists\mathrm{E}_1 \qquad \dfrac{[\varphi(a) \wedge \psi(a)]^1}{\psi(a)}\,\wedge\mathrm{E}$$

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

$$\dfrac{\exists x\,(\varphi(x) \wedge \psi(x)) \qquad \exists x\,\chi(x,b)}{\exists x\,\chi(x,b)}\,\exists\mathrm{E}_1 \qquad \dfrac{\dfrac{\forall x\,(\psi(x) \to \chi(x,b))}{\psi(a) \to \chi(a,b)}\,\forall\mathrm{E} \qquad \dfrac{[\varphi(a) \wedge \psi(a)]^1}{\psi(a)}\,\wedge\mathrm{E}}{\chi(a,b)}\,\to\mathrm{E}$$

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

$$\dfrac{\exists x\,(\varphi(x) \wedge \psi(x)) \qquad \dfrac{\chi(a,b)}{\exists x\,\chi(x,b)}\,\exists\mathrm{I}}{\exists x\,\chi(x,b)}\,\exists\mathrm{E}_1 \qquad \dfrac{\dfrac{\forall x\,(\psi(x) \to \chi(x,b))}{\psi(a) \to \chi(a,b)}\,\forall\mathrm{E} \qquad \dfrac{[\varphi(a) \wedge \psi(a)]^1}{\psi(a)}\,\wedge\mathrm{E}}{\chi(a,b)}\,\to\mathrm{E}$$

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

Example 6.3. Give a derivation of the formula $\neg\forall x\,\varphi(x)$ from the assumptions $\forall x\,\varphi(x) \to \exists y\,\psi(y)$ and $\neg\exists y\,\psi(y)$. Starting as usual, we write the target formula at the bottom.

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

The last line of the derivation is a negation, so let's try using $\neg$I. This will require that we figure out how to derive a contradiction.

$$\dfrac{\begin{array}{c}[\forall x\,\varphi(x)]^1\\ \vdots\\ \bot\end{array}}{\neg\forall x\,\varphi(x)}\,\neg\mathrm{I}_1$$

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

$$\dfrac{\dfrac{\forall x\,\varphi(x) \to \exists y\,\psi(y) \qquad [\forall x\,\varphi(x)]^1}{\exists y\,\psi(y)}\,{\to}\mathrm{E}}{\vdots}$$
$$\dfrac{\bot}{\neg\forall x\,\varphi(x)}\,\neg\mathrm{I}_1$$

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

$$\dfrac{\neg\exists y\,\psi(y) \qquad \dfrac{\dfrac{\forall x\,\varphi(x) \to \exists y\,\psi(y) \qquad [\forall x\,\varphi(x)]^1}{\exists y\,\psi(y)}\,{\to}\mathrm{E}}{\bot}\,\neg\mathrm{E}}{\neg\forall x\,\varphi(x)}\,\neg\mathrm{I}_1$$

6.4 Proof-Theoretic Notions

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

Definition 6.4 (Theorems). A sentence φ is a *theorem* if there is a derivation of φ in natural deduction in which all assumptions are discharged. We write $\vdash \varphi$ if φ is a theorem and $\nvdash \varphi$ if it is not.

Definition 6.5 (Derivability). A sentence φ is *derivable from* a set of sentences Γ, $\Gamma \vdash \varphi$, if there is a derivation with conclusion φ and in which every assumption is either discharged or is in Γ. If φ is not derivable from Γ we write $\Gamma \nvdash \varphi$.

Definition 6.6 (Consistency). A set of sentences Γ is *inconsistent* iff $\Gamma \vdash \bot$. If Γ is not inconsistent, i.e., if $\Gamma \nvdash \bot$, we say it is *consistent*.

Proposition 6.7 (Reflexivity). *If $\varphi \in \Gamma$, then $\Gamma \vdash \varphi$.*

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

Proposition 6.8 (Monotony). *If $\Gamma \subseteq \Delta$ and $\Gamma \vdash \varphi$, then $\Delta \vdash \varphi$.*

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

Proposition 6.9 (Transitivity). *If $\Gamma \vdash \varphi$ and $\{\varphi\} \cup \Delta \vdash \psi$, then $\Gamma \cup \Delta \vdash \psi$.*

Proof. If $\Gamma \vdash \varphi$, there is a derivation δ_0 of φ with all undischarged assumptions in Γ. If $\{\varphi\} \cup \Delta \vdash \psi$, then there is a derivation δ_1 of ψ with all undischarged assumptions in $\{\varphi\} \cup \Delta$. Now consider:

$$\dfrac{\dfrac{\dfrac{\Delta, [\varphi]^1}{\vdots\,\delta_1}}{\dfrac{\psi}{\varphi \to \psi}\to\mathrm{I}_1} \qquad \dfrac{\Gamma}{\vdots\,\delta_0}}{\dfrac{}{\psi}\to\mathrm{E}}$$

The undischarged assumptions are now all among $\Gamma \cup \Delta$, so this shows $\Gamma \cup \Delta \vdash \psi$. □

When $\Gamma = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ is a finite set we may use the simplified notation $\varphi_1, \varphi_2, \ldots, \varphi_k \vdash \psi$ for $\Gamma \vdash \psi$, in particular $\varphi \vdash \psi$ means that $\{\varphi\} \vdash \psi$.

Note that if $\Gamma \vdash \varphi$ and $\varphi \vdash \psi$, then $\Gamma \vdash \psi$. It follows also that if $\varphi_1, \ldots, \varphi_n \vdash \psi$ and $\Gamma \vdash \varphi_i$ for each i, then $\Gamma \vdash \psi$.

Proposition 6.10. *Γ is inconsistent iff $\Gamma \vdash \varphi$ for every sentence φ.*

Proof. Exercise. □

Proposition 6.11 (Compactness). *1. If $\Gamma \vdash \varphi$ then there is a finite subset $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \vdash \varphi$.*

2. If every finite subset of Γ is consistent, then Γ is consistent.

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

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

6.5 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 6.12. *If $\Gamma \vdash \varphi$ and $\Gamma \cup \{\varphi\}$ is inconsistent, then Γ is inconsistent.*

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

$$\dfrac{\dfrac{\dfrac{\Gamma, [\varphi]^1}{\vdots\,\delta_2}}{\dfrac{\bot}{\neg\varphi}\neg\mathrm{I}_1} \qquad \dfrac{\Gamma}{\vdots\,\delta_1}}{\dfrac{}{\bot}\neg\mathrm{E}}$$

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

Proposition 6.13. *$\Gamma \vdash \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is inconsistent.*

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

$$\dfrac{\begin{array}{cc}\neg\varphi & \begin{array}{c}\Gamma\\ \vdots\\ \varphi\end{array}\delta_0\end{array}}{\bot}\,\neg\mathrm{E}$$

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

$$\dfrac{\begin{array}{c}\Gamma, [\neg\varphi]^1\\ \vdots\\ \bot\end{array}\delta_1}{\varphi}\,\mathrm{RAA}$$

□

Proposition 6.14. *If $\Gamma \vdash \varphi$ and $\neg\varphi \in \Gamma$, then Γ is inconsistent.*

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

$$\dfrac{\begin{array}{cc}\neg\varphi & \begin{array}{c}\Gamma\\ \vdots\\ \varphi\end{array}\delta\end{array}}{\bot}\,\neg\mathrm{E}$$

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

Proposition 6.15. *If $\Gamma \cup \{\varphi\}$ and $\Gamma \cup \{\neg\varphi\}$ are both inconsistent, then Γ is inconsistent.*

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

$$\dfrac{\dfrac{\begin{array}{c}\Gamma, [\neg\varphi]^2\\ \vdots\\ \bot\end{array}\delta_2}{\neg\neg\varphi}\,\neg\mathrm{I}_2 \qquad \dfrac{\begin{array}{c}\Gamma, [\varphi]^1\\ \vdots\\ \bot\end{array}\delta_1}{\neg\varphi}\,\neg\mathrm{I}_1}{\bot}\,\neg\mathrm{E}$$

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

6.6 Derivability and the Propositional Connectives

Proposition 6.16. *1. Both $\varphi \wedge \psi \vdash \varphi$ and $\varphi \wedge \psi \vdash \psi$*

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

Proof. 1. We can derive both

$$\dfrac{\varphi \wedge \psi}{\varphi}\,\wedge\mathrm{E} \qquad \dfrac{\varphi \wedge \psi}{\psi}\,\wedge\mathrm{E}$$

2. We can derive:

$$\dfrac{\varphi \qquad \psi}{\varphi \wedge \psi}\,\wedge\mathrm{I}$$

□

Proposition 6.17. *1. $\varphi \vee \psi, \neg\varphi, \neg\psi$ is inconsistent.*

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

Proof. 1. Consider the following derivation:

$$\dfrac{\varphi \vee \psi \qquad \dfrac{\neg\varphi \qquad [\varphi]^1}{\bot}\,\neg\mathrm{E} \qquad \dfrac{\neg\psi \qquad [\psi]^1}{\bot}\,\neg\mathrm{E}}{\bot}\,\vee\mathrm{E}_1$$

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

2. We can derive both

$$\dfrac{\varphi}{\varphi \vee \psi}\,\vee\mathrm{I} \qquad \dfrac{\psi}{\varphi \vee \psi}\,\vee\mathrm{I}$$

□

Proposition 6.18. *1. $\varphi, \varphi \to \psi \vdash \psi$.*

2. Both $\neg\varphi \vdash \varphi \to \psi$ and $\psi \vdash \varphi \to \psi$.

Proof. 1. We can derive:

$$\dfrac{\varphi \to \psi \qquad \psi}{\psi}\,\to\mathrm{E}$$

2. This is shown by the following two derivations:

$$\dfrac{\dfrac{\dfrac{\neg\varphi \qquad [\varphi]^1}{\bot}\,\neg\mathrm{E}}{\psi}\,\bot\mathrm{E}}{\varphi \to \psi}\,\to\mathrm{I}_1 \qquad \dfrac{\psi}{\varphi \to \psi}\,\to\mathrm{I}$$

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

6.7 Derivability and the Quantifiers

Theorem 6.19. *If c is a constant not occurring in Γ or $\varphi(x)$ and $\Gamma \vdash \varphi(c)$, then $\Gamma \vdash \forall x\, \varphi(x)$.*

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

Proposition 6.20. *1.* $\varphi(t) \vdash \exists x\, \varphi(x)$.

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

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

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

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

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

□

6.8 Soundness

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

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

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

Theorem 6.21 (Soundness). *If φ is derivable from the undischarged assumptions Γ, then $\Gamma \vDash \varphi$.*

Proof. Let δ be a derivation of φ. We proceed by induction on the number of inferences in δ.

For the induction basis we show the claim if the number of inferences is 0. In this case, δ consists only of a single sentence φ, i.e., an assumption. That assumption is undischarged, since assumptions can only be discharged by inferences, and there are no inferences. So, any structure $\mathfrak{M}$ that satisfies all of the undischarged assumptions of the proof also satisfies φ.

Now for the inductive step. Suppose that δ contains n inferences. The premise(s) of the lowermost inference are derived using sub-derivations, each of which contains

fewer than n inferences. We assume the induction hypothesis: The premises of the lowermost inference follow from the undischarged assumptions of the sub-derivations ending in those premises. We have to show that the conclusion φ follows from the undischarged assumptions of the entire proof.

We distinguish cases according to the type of the lowermost inference. First, we consider the possible inferences with only one premise.

1. Suppose that the last inference is ¬I: The derivation has the form

$$\dfrac{\begin{array}{c}\Gamma, [\varphi]^n \\ \vdots\,\delta_1 \\ \vdots \\ \bot\end{array}}{\neg\varphi}\ \neg\mathrm{I}_n$$

   By inductive hypothesis, $\bot$ follows from the undischarged assumptions $\Gamma \cup \{\varphi\}$ of δ_1. Consider a structure $\mathfrak{M}$. We need to show that, if $\mathfrak{M} \vDash \Gamma$, then $\mathfrak{M} \vDash \neg\varphi$. Suppose for reductio that $\mathfrak{M} \vDash \Gamma$, but $\mathfrak{M} \nvDash \neg\varphi$, i.e., $\mathfrak{M} \vDash \varphi$. This would mean that $\mathfrak{M} \vDash \Gamma \cup \{\varphi\}$. This is contrary to our inductive hypothesis. So, $\mathfrak{M} \vDash \neg\varphi$.

2. The last inference is ∧E: There are two variants: φ or ψ may be inferred from the premise $\varphi \wedge \psi$. Consider the first case. The derivation δ looks like this:

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots\,\delta_1 \\ \vdots \\ \varphi \wedge \psi\end{array}}{\varphi}\ \wedge\mathrm{E}$$

   By inductive hypothesis, $\varphi \wedge \psi$ follows from the undischarged assumptions Γ of δ_1. Consider a structure $\mathfrak{M}$. We need to show that, if $\mathfrak{M} \vDash \Gamma$, then $\mathfrak{M} \vDash \varphi$. Suppose $\mathfrak{M} \vDash \Gamma$. By our inductive hypothesis ($\Gamma \vDash \varphi \vee \psi$), we know that $\mathfrak{M} \vDash \varphi \wedge \psi$. By definition, $\mathfrak{M} \vDash \varphi \wedge \psi$ iff $\mathfrak{M} \vDash \varphi$ and $\mathfrak{M} \vDash \psi$. (The case where ψ is inferred from $\varphi \wedge \psi$ is handled similarly.)

3. The last inference is ∨I: There are two variants: $\varphi \vee \psi$ may be inferred from the premise φ or the premise ψ. Consider the first case. The derivation has the form

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots\,\delta_1 \\ \vdots \\ \varphi\end{array}}{\varphi \vee \psi}\ \vee\mathrm{I}$$

   By inductive hypothesis, φ follows from the undischarged assumptions Γ of δ_1. Consider a structure $\mathfrak{M}$. We need to show that, if $\mathfrak{M} \vDash \Gamma$, then $\mathfrak{M} \vDash \varphi \vee \psi$. Suppose $\mathfrak{M} \vDash \Gamma$; then $\mathfrak{M} \vDash \varphi$ since $\Gamma \vDash \varphi$ (the inductive hypothesis). So it must also be the case that $\mathfrak{M} \vDash \varphi \vee \psi$. (The case where $\varphi \vee \psi$ is inferred from ψ is handled similarly.)

4. The last inference is →I: $\varphi \to \psi$ is inferred from a subproof with assumption φ and conclusion ψ, i.e.,

$$\dfrac{\begin{array}{c}\Gamma, [\varphi]^n \\ \vdots \\ \delta_1 \\ \vdots \\ \psi\end{array}}{\varphi \to \psi} \to \mathrm{I}_n$$

   By inductive hypothesis, ψ follows from the undischarged assumptions of δ_1, i.e., $\Gamma \cup \{\varphi\} \models \psi$. Consider a structure $\mathfrak{M}$. The undischarged assumptions of δ are just Γ, since φ is discharged at the last inference. So we need to show that $\Gamma \models \varphi \to \psi$. For reductio, suppose that for some structure $\mathfrak{M}$, $\mathfrak{M} \models \Gamma$ but $\mathfrak{M} \not\models \varphi \to \psi$. So, $\mathfrak{M} \models \varphi$ and $\mathfrak{M} \not\models \psi$. But by hypothesis, ψ is a consequence of $\Gamma \cup \{\varphi\}$, i.e., $\mathfrak{M} \models \psi$, which is a contradiction. So, $\Gamma \models \varphi \to \psi$.

5. The last inference is ⊥E: Here, δ ends in

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots \\ \delta_1 \\ \vdots \\ \bot\end{array}}{\varphi} \bot\mathrm{E}$$

   By induction hypothesis, $\Gamma \models \bot$. We have to show that $\Gamma \models \varphi$. Suppose not; then for some $\mathfrak{M}$ we have $\mathfrak{M} \models \Gamma$ and $\mathfrak{M} \not\models \varphi$. But we always have $\mathfrak{M} \not\models \bot$, so this would mean that $\Gamma \not\models \bot$, contrary to the induction hypothesis.

6. The last inference is RAA: Exercise.

7. The last inference is ∀I: Then δ has the form

$$\dfrac{\begin{array}{c}\Gamma \\ \vdots \\ \delta_1 \\ \vdots \\ \varphi(a)\end{array}}{\forall x\, \varphi(x)} \forall\mathrm{I}$$

   The premise $\varphi(a)$ is a consequence of the undischarged assumptions Γ by induction hypothesis. Consider some structure, $\mathfrak{M}$, such that $\mathfrak{M} \models \Gamma$. We need to show that $\mathfrak{M} \models \forall x\, \varphi(x)$. Since $\forall x\, \varphi(x)$ is a sentence, this means we have to show that for every variable assignment s, $\mathfrak{M}, s \models \varphi(x)$ (Proposition 4.42). Since Γ consists entirely of sentences, $\mathfrak{M}, s \models \psi$ for all $\psi \in \Gamma$ by Definition 4.35. Let $\mathfrak{M}'$ be like $\mathfrak{M}$ except that $a^{\mathfrak{M}'} = s(x)$. Since a does not occur in Γ, $\mathfrak{M}' \models \Gamma$ by Corollary 4.44. Since $\Gamma \models \varphi(a)$, $\mathfrak{M}' \models \varphi(a)$. Since $\varphi(a)$ is a sentence, $\mathfrak{M}', s \models \varphi(a)$ by Proposition 4.41. $\mathfrak{M}', s \models \varphi(x)$ iff $\mathfrak{M}' \models \varphi(a)$ by Proposition 4.46 (recall that $\varphi(a)$ is just $\varphi(x)[a/x]$). So, $\mathfrak{M}', s \models \varphi(x)$. Since a does not occur in $\varphi(x)$, by Proposition 4.43, $\mathfrak{M}, s \models \varphi(x)$. But s was an arbitrary variable assignment, so $\mathfrak{M} \models \forall x\, \varphi(x)$.

8. The last inference is ∃I: Exercise.

9. The last inference is ∀E: Exercise.

Now let's consider the possible inferences with several premises: ∨E, ∧I, →E, and ∃E.

1. The last inference is ∧I. $\varphi \wedge \psi$ is inferred from the premises φ and ψ and δ has the form

$$\dfrac{\begin{array}{cc}\Gamma_1 & \Gamma_2 \\ \vdots & \vdots \\ \delta_1 & \delta_2 \\ \vdots & \vdots \\ \varphi & \psi\end{array}}{\varphi \wedge \psi}\,\wedge\text{I}$$

   By induction hypothesis, φ follows from the undischarged assumptions Γ_1 of δ_1 and ψ follows from the undischarged assumptions Γ_2 of δ_2. The undischarged assumptions of δ are $\Gamma_1 \cup \gamma_2$, so we have to show that $\Gamma_1 \cup \Gamma_2 \vDash \varphi \wedge \psi$. Consider a structure $\mathfrak{M}$ with $\mathfrak{M} \vDash \Gamma_1 \cup \Gamma_2$. Since $\mathfrak{M} \vDash \Gamma_1$, it must be the case that $\mathfrak{M} \vDash \varphi$ as $\Gamma_1 \vDash \varphi$, and since $\mathfrak{M} \vDash \Gamma_2$, $\mathfrak{M} \vDash \psi$ since $\Gamma_2 \vDash \psi$. Together, $\mathfrak{M} \vDash \varphi \wedge \psi$.

2. The last inference is ∨E: Exercise.

3. The last inference is →E. ψ is inferred from the premises $\varphi \to \psi$ and φ. The derivation δ looks like this:

$$\dfrac{\begin{array}{cc}\Gamma_1 & \Gamma_2 \\ \vdots & \vdots \\ \delta_1 & \delta_2 \\ \vdots & \vdots \\ \varphi \to \psi & \varphi\end{array}}{\psi}\,\to\text{E}$$

   By induction hypothesis, $\varphi \to \psi$ follows from the undischarged assumptions Γ_1 of δ_1 and φ follows from the undischarged assumptions Γ_2 of δ_2. Consider a structure $\mathfrak{M}$. We need to show that, if $\mathfrak{M} \vDash \Gamma_1 \cup \Gamma_2$, then $\mathfrak{M} \vDash \psi$. Suppose $\mathfrak{M} \vDash \Gamma_1 \cup \Gamma_2$. Since $\Gamma_1 \vDash \varphi \to \psi$, $\mathfrak{M} \vDash \varphi \to \psi$. Since $\Gamma_2 \vDash \varphi$, we have $\mathfrak{M} \vDash \varphi$. This means that $\mathfrak{M} \vDash \psi$ (For if $\mathfrak{M} \nvDash \psi$, since $\mathfrak{M} \vDash \varphi$, we'd have $\mathfrak{M} \nvDash \varphi \to \psi$, contradicting $\mathfrak{M} \vDash \varphi \to \psi$).

4. The last inference is ¬E: Exercise.

5. The last inference is ∃E: Exercise. □

Corollary 6.22. *If $\vdash \varphi$, then φ is valid.*

Corollary 6.23. *If Γ is satisfiable, then it is consistent.*

Proof. We prove the contrapositive. Suppose that Γ is not consistent. Then $\Gamma \vdash \bot$, i.e., there is a derivation of $\bot$ from undischarged assumptions in Γ. By Theorem 6.21, any structure $\mathfrak{M}$ that satisfies Γ must satisfy $\bot$. Since $\mathfrak{M} \nvDash \bot$ for every structure $\mathfrak{M}$, no $\mathfrak{M}$ can satisfy Γ, i.e., Γ is not satisfiable. □

6.9 Derivations with Identity predicate

Derivations with identity predicate require additional inference rules.

$$\frac{}{t = t}\ =\text{I} \qquad \frac{t_1 = t_2 \qquad \varphi(t_1)}{\varphi(t_2)}\ =\text{E} \qquad \frac{t_1 = t_2 \qquad \varphi(t_2)}{\varphi(t_1)}\ =\text{E}$$

In the above rules, t, t_1, and t_2 are closed terms. The =I rule allows us to derive any identity statement of the form $t = t$ outright, from no assumptions.

Example 6.24. If s and t are closed terms, then $\varphi(s), s = t \vdash \varphi(t)$:

$$\frac{s = t \qquad \varphi(s)}{\varphi(t)}\ =\text{E}$$

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

Example 6.25. We derive the sentence

$$\forall x\, \forall y\, ((\varphi(x) \wedge \varphi(y)) \rightarrow x = y)$$

from the sentence

$$\exists x\, \forall y\, (\varphi(y) \rightarrow y = x)$$

We develop the derivation backwards:

$$\dfrac{\dfrac{\dfrac{\exists x\, \forall y\, (\varphi(y) \rightarrow y = x) \quad [\varphi(a) \wedge \varphi(b)]^1}{\vdots}}{\dfrac{a = b}{((\varphi(a) \wedge \varphi(b)) \rightarrow a = b)}\ \rightarrow\text{I}_1}}{\dfrac{\forall y\, ((\varphi(a) \wedge \varphi(y)) \rightarrow a = y)}{\forall x\, \forall y\, ((\varphi(x) \wedge \varphi(y)) \rightarrow x = y)}\ \forall\text{I}}\ \forall\text{I}$$

We'll now have to use the main assumption: since it is an existential formula, we use ∃E to derive the intermediary conclusion $a = b$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\exists x\, \forall y\, (\varphi(y) \rightarrow y = x) \qquad \begin{array}{c}[\forall y\, (\varphi(y) \rightarrow y = c)]^2 \\ [\varphi(a) \wedge \varphi(b)]^1 \\ \vdots \\ a = b\end{array}}{a = b}\ \exists\text{E}_2}{((\varphi(a) \wedge \varphi(b)) \rightarrow a = b)}\ \rightarrow\text{I}_1}{\forall y\, ((\varphi(a) \wedge \varphi(y)) \rightarrow a = y)}\ \forall\text{I}}{\forall x\, \forall y\, ((\varphi(x) \wedge \varphi(y)) \rightarrow x = y)}\ \forall\text{I}$$

The sub-derivation on the top right is completed by using its assumptions to show that $a = c$ and $b = c$. This requies two separate derivations. The derivation for $a = c$ is as follows:

$$\dfrac{\dfrac{[\forall y\,(\varphi(y) \to y = c)]^2}{\varphi(a) \to a = c}\,\forall\text{E} \qquad \dfrac{[\varphi(a) \wedge \varphi(b)]^1}{\varphi(a)}\,\wedge\text{E}}{a = c}\,\to\text{E}$$

From $a = c$ and $b = c$ we derive $a = b$ by =E.

6.10 Soundness with Identity predicate

Proposition 6.26. *Natural deduction with rules for = is sound.*

Proof. Any formula of the form $t = t$ is valid, since for every structure $\mathfrak{M}$, $\mathfrak{M} \vDash t = t$. (Note that we assume the term t to be ground, i.e., it contains no variables, so variable assignments are irrelevant).

Suppose the last inference in a derivation is =E, i.e., the derivation has the following form:

$$\dfrac{\begin{array}{cc}\Gamma_1 & \Gamma_2 \\ \vdots & \vdots \\ \delta_1 & \delta_2 \\ \vdots & \vdots \\ t_1 = t_2 & \varphi(t_1)\end{array}}{\varphi(t_2)}\,=\text{E}$$

The premises $t_1 = t_2$ and $\varphi(t_1)$ are derived from undischarged assumptions Γ_1 and Γ_2, respectively. We want to show that $\varphi(t_2)$ follows from $\Gamma_1 \cup \Gamma_2$. Consider a structure $\mathfrak{M}$ with $\mathfrak{M} \vDash \Gamma_1 \cup \Gamma_2$. By induction hypothesis, $\mathfrak{M} \vDash \varphi(t_1)$ and $\mathfrak{M} \vDash t_1 = t_2$. Therefore, $\mathrm{Val}^{\mathfrak{M}}(t_1) = \mathrm{Val}^{\mathfrak{M}}(t_2)$. Let s be any variable assignment, and s' be the x-variant given by $s'(x) = \mathrm{Val}^{\mathfrak{M}}(t_1) = \mathrm{Val}^{\mathfrak{M}}(t_2)$. By Proposition 4.46, $\mathfrak{M}, s \vDash \varphi(t_1)$ iff $\mathfrak{M}, s' \vDash \varphi(x)$ iff $\mathfrak{M}, s \vDash \varphi(t_2)$. Since $\mathfrak{M} \vDash \varphi(t_1)$, we have $\mathfrak{M} \vDash \varphi(t_2)$. □

Problems

Problem 6.1. Give derivations of the following:

1. $\exists y\,\varphi(y) \to \psi$ from the assumption $\forall x\,(\varphi(x) \to \psi)$
2. $\exists x\,(\varphi(x) \to \forall y\,\varphi(y))$

Problem 6.2. Prove Proposition 6.10

Problem 6.3. Prove that $\Gamma \vdash \neg\varphi$ iff $\Gamma \cup \{\varphi\}$ is inconsistent.

Problem 6.4. Complete the proof of Theorem 6.21.

Problem 6.5. Prove that = is both symmetric and transitive, i.e., give derivations of $\forall x\,\forall y\,(x = y \to y = x)$ and $\forall x\,\forall y\,\forall z((x = y \wedge y = z) \to x = z)$

Problem 6.6. Give derivations of the following formulas:

1. $\forall x\,\forall y\,((x = y \wedge \varphi(x)) \to \varphi(y))$
2. $\exists x\,\varphi(x) \wedge \forall y\,\forall z\,((\varphi(y) \wedge \varphi(z)) \to y = z) \to \exists x\,(\varphi(x) \wedge \forall y\,(\varphi(y) \to y = x))$

Chapter 7

The Completeness Theorem

7.1 Introduction

The completeness theorem is one of the most fundamental results about logic. It comes in two formulations, the equivalence of which we'll prove. In its first formulation it says something fundamental about the relationship between semantic consequence and our proof system: if a sentence φ follows from some sentences Γ, then there is also a derivation that establishes $\Gamma \vdash \varphi$. Thus, the proof system is as strong as it can possibly be without proving things that don't actually follow.

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

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

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

These aren't the only reasons the completeness theorem—or rather, its proof—is important. It has a number of important consequences, some of which we'll discuss separately. For instance, since any derivation that shows $\Gamma \vdash \varphi$ is finite and so can only use finitely many of the sentences in Γ, it follows by the completeness theorem that if φ is a consequence of Γ, it is already a consequence of a finite subset of Γ. This is called *compactness*. Equivalently, if every finite subset of Γ is consistent, then Γ itself must be consistent.

Although the compactness theorem follows from the completeness theorem via the detour through derivations, it is also possible to use the *the proof of* the completeness theorem to establish it directly. For what the proof does is take a set of sentences with a certain property—consistency—and constructs a structure out of this set that

has certain properties (in this case, that it satisfies the set). Almost the very same construction can be used to directly establish compactness, by starting from "finitely satisfiable" sets of sentences instead of consistent ones. The construction also yields other consequences, e.g., that any satisfiable set of sentences has a finite or countably infinite model. (This result is called the Löwenheim-Skolem theorem.) In general, the construction of structures from sets of sentences is used often in logic, and sometimes even in philosophy.

7.2 Outline of the Proof

The proof of the completeness theorem is a bit complex, and upon first reading it, it is easy to get lost. So let us outline the proof. The first step is a shift of perspective, that allows us to see a route to a proof. When completeness is thought of as "whenever $\Gamma \vDash \varphi$ then $\Gamma \vdash \varphi$," it may be hard to even come up with an idea: for to show that $\Gamma \vdash \varphi$ we have to find a derivation, and it does not look like the hypothesis that $\Gamma \vDash \varphi$ helps us for this in any way. For some proof systems it is possible to directly construct a derivation, but we will take a slightly different 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 $P(a_1, \ldots, a_n)$ where the a_i are constant symbols. All we have to do is come up with a domain $|\mathfrak{M}|$ and an assignment for P so that $\mathfrak{M} \vDash P(a_1, \ldots, a_n)$. But that's not very hard: put $|\mathfrak{M}| = \mathbb{N}$, $c_i^{\mathfrak{M}} = i$, and for every $P(a_1, \ldots, a_n) \in \Gamma$, put the tuple $\langle k_1, \ldots, k_n \rangle$ into $P^{\mathfrak{M}}$, where k_i is the index of the constant symbol a_i (i.e., $a_i \equiv c_{k_i}$).

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

What if Γ contains complex, non-atomic formulas? Say it contains $\varphi \wedge \psi$. To make that true, we should proceed as if both φ and ψ were in Γ. And if $\varphi \vee \psi \in \Gamma$, then we will have to make at least one of them true, i.e., proceed as if one of them was in Γ.

This suggests the following idea: we add additional formulas to Γ so as to (a) keep the resulting set consistent and (b) make sure that for every possible atomic sentence φ, either φ is in the resulting set, or $\neg\varphi$ is, and (c) such that, whenever $\varphi \wedge \psi$ is in the set, so are both φ and ψ, if $\varphi \vee \psi$ is in the set, at least one of φ or ψ is also, etc. We keep doing this (potentially forever). Call the set of all formulas so added Γ^*. Then our construction above would provide us with a structure $\mathfrak{M}$ 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 $\exists x\, \varphi(x) \in \Gamma$ we would hope to be able to pick some constant symbol c and add $\varphi(c)$ in this process. But how do we know we can always do that? Perhaps we only have a few constant symbols in our language, and for each one of them we have $\neg\varphi(c) \in \Gamma$. We can't also add $\varphi(c)$, since this would

make the set inconsistent, and we wouldn't know whether $\mathfrak{M}$ has to make $\varphi(c)$ or $\neg\varphi(c)$ true. Moreover, it might happen that Γ contains only sentences in a language that has no constant symbols at all (e.g., the language of set theory).

The solution to this problem is to simply add infinitely many constants at the beginning, plus sentences that connect them with the quantifiers in the right way. (Of course, we have to verify that this cannot introduce an inconsistency.)

Our original construction works well if we only have constant symbols in the atomic sentences. But the language might also contain function symbols. In that case, it might be tricky to find the right functions on $\mathbb{N}$ to assign to these function symbols to make everything work. So here's another trick: instead of using i to interpret c_i, just take the set of constant symbols itself as the domain. Then $\mathfrak{M}$ can assign every constant symbol to itself: $c_i^{\mathfrak{M}} = c_i$. But why not go all the way: let $|\mathfrak{M}|$ be all *terms* of the language! If we do this, there is an obvious assignment of functions (that take terms as arguments and have terms as values) to function symbols: we assign to the function symbol f_i^n the function which, given n terms $t_1, \ldots, t_n$ as input, produces the term $f_i^n(t_1, \ldots, t_n)$ as value.

The last piece of the puzzle is what to do with =. The predicate symbol = has a fixed interpretation: $\mathfrak{M} \vDash t = t'$ iff $\mathrm{Val}^{\mathfrak{M}}(t) = \mathrm{Val}^{\mathfrak{M}}(t')$. Now if we set things up so that the value of a term t is t itself, then this structure will make *no* sentence of the form $t = t'$ true unless t and t' are one and the same term. And of course this is a problem, since basically every interesting theory in a language with function symbols will have as theorems sentences $t = t'$ where t and t' are not the same term (e.g., in theories of arithmetic: $(0 + 0) = 0$). To solve this problem, we change the domain of $\mathfrak{M}$: instead of using terms as the objects in $|\mathfrak{M}|$, we use sets of terms, and each set is so that it contains all those terms which the sentences in Γ require to be equal. So, e.g., if Γ is a theory of arithmetic, one of these sets will contain: 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 $\varphi \wedge \psi$ iff it contains both φ and ψ, $\varphi \vee \psi$ iff it contains at least one of them, etc. (Proposition 7.2). Then we define and investigate "saturated" sets of sentences. A saturated set is one which contains conditionals that link each quantified sentence to instances of it (Definition 7.5). We show that any consistent set Γ can always be extended to a saturated set Γ' (Lemma 7.6). If a set is consistent, saturated, and complete it also has the property that it contains $\exists x\, \varphi(x)$ iff it contains $\varphi(t)$ for some closed term t and $\forall x\, \varphi(x)$ iff it contains $\varphi(t)$ for all closed terms t (Proposition 7.7). We'll then take the saturated consistent set Γ' and show that it can be extended to a saturated, consistent, and complete set Γ^* (Lemma 7.8). This set Γ^* is what we'll use to define our term model $\mathfrak{M}(\Gamma^*)$. The term model has the set of closed terms as its domain, and the interpretation of its predicate symbols is given by the atomic sentences in Γ^* (Definition 7.9). We'll use the properties of saturated, complete consistent sets to show that indeed $\mathfrak{M}(\Gamma^*) \vDash \varphi$ iff $\varphi \in \Gamma^*$ (Lemma 7.11), and thus in particular, $\mathfrak{M}(\Gamma^*) \vDash \Gamma$. Finally, we'll consider how to define a term model if Γ contains = as well (Definition 7.15) and show that it satisfies Γ^* (Lemma 7.17).

7.3 Complete Consistent Sets of Sentences

Definition 7.1 (Complete set). A set Γ of sentences is *complete* iff for any sentence φ, either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$.

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

Complete consistent sets are important in the completeness proof since we can guarantee that every consistent set of sentences Γ is contained in a complete consistent set Γ^*. A complete consistent set contains, for each sentence φ, either φ or its negation $\neg\varphi$, but not both. This is true in particular for atomic sentences, so from a complete consistent set in a language suitably expanded by constant symbols, we can construct a structure where the interpretation of predicate symbols is defined according to which atomic sentences are in Γ^*. This structure can then be shown to make all sentences in Γ^* (and hence also all those in Γ) true. The proof of this latter fact requires that $\neg\varphi \in \Gamma^*$ iff $\varphi \notin \Gamma^*$, $(\varphi \vee \psi) \in \Gamma^*$ iff $\varphi \in \Gamma^*$ or $\psi \in \Gamma^*$, etc.

In what follows, we will often tacitly use the properties of reflexivity, monotonicity, and transitivity of $\vdash$ (see section 6.4).

Proposition 7.2. *Suppose Γ is complete and consistent. Then:*

1. *If $\Gamma \vdash \varphi$, then $\varphi \in \Gamma$.*
2. *$\varphi \wedge \psi \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$.*
3. *$\varphi \vee \psi \in \Gamma$ iff either $\varphi \in \Gamma$ or $\psi \in \Gamma$.*
4. *$\varphi \to \psi \in \Gamma$ iff either $\varphi \notin \Gamma$ or $\psi \in \Gamma$.*

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

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

   Suppose that $\Gamma \vdash \varphi$. Suppose to the contrary that $\varphi \notin \Gamma$. Since Γ is complete, $\neg\varphi \in \Gamma$. By Proposition 6.14, Γ is inconsistent. This contradicts the assumption that Γ is consistent. Hence, it cannot be the case that $\varphi \notin \Gamma$, so $\varphi \in \Gamma$.

2. $\varphi \wedge \psi \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$:

   For the forward direction, suppose $\varphi \wedge \psi \in \Gamma$. Then by Proposition 6.16, item (1), $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$. By (1), $\varphi \in \Gamma$ and $\psi \in \Gamma$, as required.

   For the reverse direction, let $\varphi \in \Gamma$ and $\psi \in \Gamma$. By Proposition 6.16, item (2), $\Gamma \vdash \varphi \wedge \psi$. By (1), $\varphi \wedge \psi \in \Gamma$.

3. First we show that if $\varphi \vee \psi \in \Gamma$, then either $\varphi \in \Gamma$ or $\psi \in \Gamma$. Suppose $\varphi \vee \psi \in \Gamma$ but $\varphi \notin \Gamma$ and $\psi \notin \Gamma$. Since Γ is complete, $\neg\varphi \in \Gamma$ and $\neg\psi \in \Gamma$. By Proposition 6.17, item (1), Γ is inconsistent, a contradiction. Hence, either $\varphi \in \Gamma$ or $\psi \in \Gamma$.

   For the reverse direction, suppose that $\varphi \in \Gamma$ or $\psi \in \Gamma$. By Proposition 6.17, item (2), $\Gamma \vdash \varphi \vee \psi$. By (1), $\varphi \vee \psi \in \Gamma$, as required.

4. For the forward direction, suppose $\varphi \to \psi \in \Gamma$, and suppose to the contrary that $\varphi \in \Gamma$ and $\psi \notin \Gamma$. On these assumptions, $\varphi \to \psi \in \Gamma$ and $\varphi \in \Gamma$. By Proposition 6.18, item (1), $\Gamma \vdash \psi$. But then by (1), $\psi \in \Gamma$, contradicting the assumption that $\psi \notin \Gamma$.

   For the reverse direction, first consider the case where $\varphi \notin \Gamma$. Since Γ is complete, $\neg\varphi \in \Gamma$. By Proposition 6.18, item (2), $\Gamma \vdash \varphi \to \psi$. Again by (1), we get that $\varphi \to \psi \in \Gamma$, as required.

   Now consider the case where $\psi \in \Gamma$. By Proposition 6.18, item (2) again, $\Gamma \vdash \varphi \to \psi$. By (1), $\varphi \to \psi \in \Gamma$. □

7.4 Henkin Expansion

Part of the challenge in proving the completeness theorem is that the model we construct from a complete consistent set Γ must make all the quantified formulas in Γ true. In order to guarantee this, we use a trick due to Leon Henkin. In essence, the trick consists in expanding the language by infinitely many constant symbols and adding, for each formula with one free variable $\varphi(x)$ a formula of the form $\exists x\, \varphi(x) \to \varphi(c)$, where c is one of the new constant symbols. When we construct the structure satisfying Γ, this will guarantee that each true existential sentence has a witness among the new constants.

Proposition 7.3. *If Γ is consistent in $\mathcal{L}$ and $\mathcal{L}'$ is obtained from $\mathcal{L}$ by adding a countably infinite set of new constant symbols $d_0, d_1, \ldots$, then Γ is consistent in $\mathcal{L}'$.*

Definition 7.4 (Saturated set). A set Γ of formulas of a language $\mathcal{L}$ is *saturated* iff for each formula $\varphi(x) \in \mathrm{Frm}(\mathcal{L})$ with one free variable x there is a constant symbol $c \in \mathcal{L}$ such that $\exists x\, \varphi(x) \to \varphi(c) \in \Gamma$.

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

Definition 7.5. Let $\mathcal{L}'$ be as in Proposition 7.3. Fix an enumeration $\varphi_0(x_0)$, $\varphi_1(x_1)$, ...of all formulas $\varphi_i(x_i)$ of $\mathcal{L}'$ in which one variable (x_i) occurs free. We define the sentences θ_n by induction on n.

Let c_0 be the first constant symbol among the d_i we added to $\mathcal{L}$ which does not occur in $\varphi_0(x_0)$. Assuming that $\theta_0, \ldots, \theta_{n-1}$ have already been defined, let c_n be the first among the new constant symbols d_i that occurs neither in $\theta_0, \ldots, \theta_{n-1}$ nor in $\varphi_n(x_n)$.

Now let θ_n be the formula $\exists x_n\, \varphi_n(x_n) \to \varphi_n(c_n)$.

Lemma 7.6. *Every consistent set Γ can be extended to a saturated consistent set Γ'.*

Proof. Given a consistent set of sentences Γ in a language $\mathcal{L}$, expand the language by adding a countably infinite set of new constant symbols to form $\mathcal{L}'$. By Proposition 7.3, Γ is still consistent in the richer language. Further, let θ_i be as in Definition 7.5. Let

$$\Gamma_0 = \Gamma$$
$$\Gamma_{n+1} = \Gamma_n \cup \{\theta_n\}$$

i.e., $\Gamma_{n+1} = \Gamma \cup \{\theta_0, \ldots, \theta_n\}$, and let $\Gamma' = \bigcup_n \Gamma_n$. Γ' is clearly saturated.

If Γ' were inconsistent, then for some n, Γ_n would be inconsistent (Exercise: explain why). So to show that Γ' is consistent it suffices to show, by induction on n, that each set Γ_n is consistent.

The induction basis is simply the claim that $\Gamma_0 = \Gamma$ is consistent, which is the hypothesis of the theorem. For the induction step, suppose that Γ_n is consistent but $\Gamma_{n+1} = \Gamma_n \cup \{\theta_n\}$ is inconsistent. Recall that θ_n is $\exists x_n \varphi_n(x_n) \to \varphi_n(c_n)$, where $\varphi_n(x_n)$ is a formula of $\mathcal{L}'$ with only the variable x_n free. By the way we've chosen the c_n (see Definition 7.5), c_n does not occur in $\varphi_n(x_n)$ nor in Γ_n.

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

$$\Gamma_n \vdash \exists x_n \varphi_n(x_n) \qquad \Gamma_n \vdash \neg\varphi_n(c_n)$$

Since c_n does not occur in Γ_n or in $\varphi_n(x_n)$, Theorem 6.19 applies. From $\Gamma_n \vdash \neg\varphi_n(c_n)$, we obtain $\Gamma_n \vdash \forall x_n \neg\varphi_n(x_n)$. Thus we have that both $\Gamma_n \vdash \exists x_n \varphi_n(x_n)$ and $\Gamma_n \vdash \forall x_n \neg\varphi_n(x_n)$, so Γ_n itself is inconsistent. (Note that $\forall x_n \neg\varphi_n(x_n) \vdash \neg\exists x_n \varphi_n(x_n)$.) Contradiction: Γ_n was supposed to be consistent. Hence $\Gamma_n \cup \{\theta_n\}$ is consistent. □

We'll now show that *complete*, consistent sets which are saturated have the property that it contains a universally quantified sentence iff it contains all its instances and it contains an existentially quantified sentence iff it contains at least one instance. We'll use this to show that the structure we'll generate from a complete, consistent, saturated set makes all its quantified sentences true.

Proposition 7.7. *Suppose Γ is complete, consistent, and saturated.*

1. *$\exists x \varphi(x) \in \Gamma$ iff $\varphi(t) \in \Gamma$ for at least one closed term t.*

2. *$\forall x \varphi(x) \in \Gamma$ iff $\varphi(t) \in \Gamma$ for all closed terms t.*

Proof. 1. First suppose that $\exists x \varphi(x) \in \Gamma$. Because Γ is saturated, $(\exists x \varphi(x) \to \varphi(c)) \in \Gamma$ for some constant symbol c. By Proposition 6.18, item (1), and Proposition 7.2(1), $\varphi(c) \in \Gamma$.

For the other direction, saturation is not necessary: Suppose $\varphi(t) \in \Gamma$. Then $\Gamma \vdash \exists x \varphi(x)$ by Proposition 6.20, item (1). By Proposition 7.2(1), $\exists x \varphi(x) \in \Gamma$.

2. Suppose that $\varphi(t) \in \Gamma$ for all closed terms t. By way of contradiction, assume $\forall x \varphi(x) \notin \Gamma$. Since Γ is complete, $\neg\forall x \varphi(x) \in \Gamma$. By saturation, $(\exists x \neg\varphi(x) \to \neg\varphi(c)) \in \Gamma$ for some constant symbol c. By assumption, since c is a closed term, $\varphi(c) \in \Gamma$. But this would make Γ inconsistent. (Exercise: give the derivation that shows

$$\neg\forall x \varphi(x), \exists x \neg\varphi(x) \to \neg\varphi(c), \varphi(c)$$

is inconsistent.)

For the reverse direction, we do not need saturation: Suppose $\forall x \varphi(x) \in \Gamma$. Then $\Gamma \vdash \varphi(t)$ by Proposition 6.20, item (2). We get $\varphi(t) \in \Gamma$ by Proposition 7.2. □

7.5 Lindenbaum's Lemma

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

Lemma 7.8 (Lindenbaum's Lemma). *Every consistent set Γ in a language $\mathcal{L}$ can be extended to a complete and consistent set Γ^*.*

Proof. Let Γ be consistent. Let $\varphi_0, \varphi_1, \ldots$ be an enumeration of all the sentences of $\mathcal{L}$. Define $\Gamma_0 = \Gamma$, and

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

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

Each Γ_n is consistent: Γ_0 is consistent by definition. If $\Gamma_{n+1} = \Gamma_n \cup \{\varphi_n\}$, this is because the latter is consistent. If it isn't, $\Gamma_{n+1} = \Gamma_n \cup \{\neg\varphi_n\}$. We have to verify that $\Gamma_n \cup \{\neg\varphi_n\}$ is consistent. Suppose it's not. Then *both* $\Gamma_n \cup \{\varphi_n\}$ and $\Gamma_n \cup \{\neg\varphi_n\}$ are inconsistent. This means that Γ_n would be inconsistent by Proposition 6.14, 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 $\Gamma_{n+1} = \Gamma_n \cup \{\varphi_n\}$ or $= \Gamma_n \cup \{\neg\varphi_n\}$ 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 $\psi \in \Gamma^*$ not already in Γ_0 is added at some stage i. If n is the last one of these, then all ψ in the finite subset are in Γ_n. So, every finite subset of Γ^* is consistent. By Proposition 6.11, Γ^* is consistent.

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

7.6 Construction of a Model

Right now we are not concerned about $=$, i.e., we only want to show that a consistent set Γ of sentences not containing $=$ is satisfiable. We first extend Γ to a consistent, complete, and saturated set Γ^*. In this case, the definition of a model $\mathfrak{M}(\Gamma^*)$ is simple: We take the set of closed terms of $\mathcal{L}'$ as the domain. We assign every constant symbol to itself, and make sure that more generally, for every closed term t, $\mathrm{Val}^{\mathfrak{M}(\Gamma^*)}(t) = t$. The predicate symbols are assigned extensions in such a way that an atomic sentence is true in $\mathfrak{M}(\Gamma^*)$ iff it is in Γ^*. This will obviously make all the atomic sentences in Γ^* true in $\mathfrak{M}(\Gamma^*)$. The rest are true provided the Γ^* we start with is consistent, complete, and saturated.

Definition 7.9 (Term model). Let Γ^* be a complete and consistent, saturated set of sentences in a language $\mathcal{L}$. The *term model* $\mathfrak{M}(\Gamma^*)$ of Γ^* is the structure defined as follows:

1. The domain $|\mathfrak{M}(\Gamma^*)|$ is the set of all closed terms of $\mathcal{L}$.

2. The interpretation of a constant symbol c is c itself: $c^{\mathfrak{M}(\Gamma^*)} = c$.

3. The function symbol f is assigned the function which, given as arguments the closed terms $t_1, \ldots, t_n$, has as value the closed term $f(t_1, \ldots, t_n)$:

$$f^{\mathfrak{M}(\Gamma^*)}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$$

4. If R is an n-place predicate symbol, then

$$\langle t_1, \ldots, t_n \rangle \in R^{\mathfrak{M}(\Gamma^*)} \text{ iff } R(t_1, \ldots, t_n) \in \Gamma^*.$$

A structure $\mathfrak{M}$ may make an existentially quantified sentence $\exists x\, \varphi(x)$ true without there being an instance $\varphi(t)$ that it makes true. A structure $\mathfrak{M}$ may make all instances $\varphi(t)$ of a universally quantified sentence $\forall x\, \varphi(x)$ true, without making $\forall x\, \varphi(x)$ true. This is because in general not every element of $|\mathfrak{M}|$ is the value of a closed term ($\mathfrak{M}$ may not be covered). This is the reason the satisfaction relation is defined via variable assignments. However, for our term model $\mathfrak{M}(\Gamma^*)$ this wouldn't be necessary—because it is covered. This is the content of the next result.

Proposition 7.10. *Let $\mathfrak{M}(\Gamma^*)$ be the term model of Definition 7.9.*

1. *$\mathfrak{M}(\Gamma^*) \vDash \exists x\, \varphi(x)$ iff $\mathfrak{M} \vDash \varphi(t)$ for at least one term t.*

2. *$\mathfrak{M}(\Gamma^*) \vDash \forall x\, \varphi(x)$ iff $\mathfrak{M} \vDash \varphi(t)$ for all terms t.*

Proof. 1. By Proposition 4.42, $\mathfrak{M}(\Gamma^*) \vDash \exists x\, \varphi(x)$ iff for at least one variable assignment s, $\mathfrak{M}(\Gamma^*), s \vDash \varphi(x)$. As $|\mathfrak{M}(\Gamma^*)|$ consists of the closed terms of $\mathcal{L}$, this is the case iff there is at least one closed term t such that $s(x) = t$ and $\mathfrak{M}(\Gamma^*), s \vDash \varphi(x)$. By Proposition 4.46, $\mathfrak{M}(\Gamma^*), s \vDash \varphi(x)$ iff $\mathfrak{M}(\Gamma^*), s \vDash \varphi(t)$, where $s(x) = t$. By Proposition 4.41, $\mathfrak{M}(\Gamma^*), s \vDash \varphi(t)$ iff $\mathfrak{M}(\Gamma^*) \vDash \varphi(t)$, since $\varphi(t)$ is a sentence.

2. By Proposition 4.42, $\mathfrak{M}(\Gamma^*) \vDash \forall x\, \varphi(x)$ iff for every variable assignment s, $\mathfrak{M}(\Gamma^*), s \vDash \varphi(x)$. Recall that $|\mathfrak{M}(\Gamma^*)|$ 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 4.46, $\mathfrak{M}(\Gamma^*), s \vDash \varphi(x)$ iff $\mathfrak{M}(\Gamma^*), s \vDash \varphi(t)$, where $s(x) = t$. By Proposition 4.41, $\mathfrak{M}(\Gamma^*), s \vDash \varphi(t)$ iff $\mathfrak{M}(\Gamma^*) \vDash \varphi(t)$, since $\varphi(t)$ is a sentence. □

Lemma 7.11 (Truth Lemma). *Suppose φ does not contain $=$. Then $\mathfrak{M}(\Gamma^*) \vDash \varphi$ iff $\varphi \in \Gamma^*$.*

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

1. $\varphi \equiv \bot$: $\mathfrak{M}(\Gamma^*) \nvDash \bot$ by definition of satisfaction. On the other hand, $\bot \notin \Gamma^*$ since Γ^* is consistent.

2. $\varphi \equiv R(t_1, \ldots, t_n)$: $\mathfrak{M}(\Gamma^*) \vDash R(t_1, \ldots, t_n)$ iff $\langle t_1, \ldots, t_n \rangle \in R^{\mathfrak{M}(\Gamma^*)}$ (by the definition of satisfaction) iff $R(t_1, \ldots, t_n) \in \Gamma^*$ (by the construction of $\mathfrak{M}(\Gamma^*)$).

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

4. $\varphi \equiv \psi \wedge \chi$: $\mathfrak{M}(\Gamma^*) \vDash \varphi$ iff we have both $\mathfrak{M}(\Gamma^*) \vDash \psi$ and $\mathfrak{M}(\Gamma^*) \vDash \chi$ (by definition of satisfaction) iff both $\psi \in \Gamma^*$ and $\chi \in \Gamma^*$ (by the induction hypothesis). By Proposition 7.2(2), this is the case iff $(\psi \wedge \chi) \in \Gamma^*$.

5. $\varphi \equiv \psi \vee \chi$: $\mathfrak{M}(\Gamma^*) \vDash \varphi$ iff $\mathfrak{M}(\Gamma^*) \vDash \psi$ or $\mathfrak{M}(\Gamma^*) \vDash \chi$ (by definition of satisfaction) iff $\psi \in \Gamma^*$ or $\chi \in \Gamma^*$ (by induction hypothesis). This is the case iff $(\psi \vee \chi) \in \Gamma^*$ (by Proposition 7.2(3)).

6. $\varphi \equiv \psi \to \chi$: $\mathfrak{M}(\Gamma^*) \vDash \varphi$ iff $\mathfrak{M}(\Gamma^*) \nvDash \psi$ or $\mathfrak{M}(\Gamma^*) \vDash \chi$ (by definition of satisfaction) iff $\psi \notin \Gamma^*$ or $\chi \in \Gamma^*$ (by induction hypothesis). This is the case iff $(\psi \to \chi) \in \Gamma^*$ (by Proposition 7.2(4)).

7. $\varphi \equiv \forall x\, \psi(x)$: $\mathfrak{M}(\Gamma^*) \vDash \varphi$ iff $\mathfrak{M}(\Gamma^*) \vDash \psi(t)$ for all terms t (Proposition 7.10). By induction hypothesis, this is the case iff $\psi(t) \in \Gamma^*$ for all terms t, by Proposition 7.7, this in turn is the case iff $\forall x\, \varphi(x) \in \Gamma^*$.

8. $\varphi \equiv \exists x\, \psi(x)$: $\mathfrak{M}(\Gamma^*) \vDash \varphi$ iff $\mathfrak{M}(\Gamma^*) \vDash \psi(t)$ for at least one term t (Proposition 7.10). By induction hypothesis, this is the case iff $\psi(t) \in \Gamma^*$ for at least one term t. By Proposition 7.7, this in turn is the case iff $\exists x\, \varphi(x) \in \Gamma^*$. □

7.7 Identity

The construction of the term model given in the preceding section is enough to establish completeness for first-order logic for sets Γ that do not contain $=$. The term model satisfies every $\varphi \in \Gamma^*$ which does not contain $=$ (and hence all $\varphi \in \Gamma$). It does not work, however, if $=$ is present. The reason is that Γ^* then may contain a sentence $t = t'$, but in the term model the value of any term is that term itself. Hence, if t and t' are different terms, their values in the term model—i.e., t and t', respectively—are different, and so $t = t'$ is false. We can fix this, however, using a construction known as "factoring."

Definition 7.12. Let Γ^* be a consistent and complete set of sentences in $\mathcal{L}$. We define the relation $\approx$ on the set of closed terms of $\mathcal{L}$ by

$$t \approx t' \quad \text{iff} \quad t = t' \in \Gamma^*$$

Proposition 7.13. *The relation $\approx$ has the following properties:*

1. *$\approx$ is reflexive.*

2. *$\approx$ is symmetric.*

3. *$\approx$ is transitive.*

4. *If $t \approx t'$, f is a function symbol, and $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$ are terms, then*

$$f(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n) \approx f(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots, t_n).$$

5. *If $t \approx t'$, R is a predicate symbol, and $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$ are terms, then*

$$R(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n) \in \Gamma^* \text{ iff } R(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots, t_n) \in \Gamma^*.$$

Proof. Since Γ^* is consistent and complete, $t = t' \in \Gamma^*$ iff $\Gamma^* \vdash t = t'$. Thus it is enough to show the following:

1. $\Gamma^* \vdash t = t$ for all terms t.
2. If $\Gamma^* \vdash t = t'$ then $\Gamma^* \vdash t' = t$.
3. If $\Gamma^* \vdash t = t'$ and $\Gamma^* \vdash t' = t''$, then $\Gamma^* \vdash t = t''$.
4. If $\Gamma^* \vdash t = t'$, then

$$\Gamma^* \vdash f(t_1, \ldots, t_{i-1}, t, t_{i+1}, , \ldots, t_n) = f(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots, t_n)$$

   for every n-place function symbol f and terms $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$.
5. If $\Gamma^* \vdash t = t'$ and $\Gamma^* \vdash R(t_1, \ldots, t_{i-1}, t, t_{i+1}, \ldots, t_n)$, then $\Gamma^* \vdash R(t_1, \ldots, t_{i-1}, t', t_{i+1}, \ldots, t_n)$ for every n-place predicate symbol R and terms $t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_n$. □

Definition 7.14. Suppose Γ^* is a consistent and complete set in a language $\mathcal{L}$, t is a term, and $\approx$ as in the previous definition. Then:

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

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

Definition 7.15. Let $\mathfrak{M} = \mathfrak{M}(\Gamma^*)$ be the term model for Γ^*. Then $\mathfrak{M}/_\approx$ is the following structure:

1. $|\mathfrak{M}/_\approx| = \mathrm{Trm}(\mathcal{L})/_\approx$.
2. $c^{\mathfrak{M}/_\approx} = [c]_\approx$
3. $f^{\mathfrak{M}/_\approx}([t_1]_\approx, \ldots, [t_n]_\approx) = [f(t_1, \ldots, t_n)]_\approx$
4. $\langle [t_1]_\approx, \ldots, [t_n]_\approx \rangle \in R^{\mathfrak{M}/_\approx}$ iff $\mathfrak{M} \vDash R(t_1, \ldots, t_n)$.

Note that we have defined $f^{\mathfrak{M}/_\approx}$ and $R^{\mathfrak{M}/_\approx}$ for elements of $\mathrm{Trm}(\mathcal{L})/_\approx$ by referring to them as $[t]_\approx$, i.e., via *representatives* $t \in [t]_\approx$. We have to make sure that these definitions do not depend on the choice of these representatives, i.e., that for some other choices t' which determine the same equivalence classes ($[t]_\approx = [t']_\approx$), the definitions yield the same result. For instance, if R is a one-place predicate symbol, the last clause of the definition says that $[t]_\approx \in R^{\mathfrak{M}/_\approx}$ iff $\mathfrak{M} \vDash R(t)$. If for some other term t' with $t \approx t'$, $\mathfrak{M} \nvDash R(t)$, then the definition would require $[t']_\approx \notin R^{\mathfrak{M}/_\approx}$. If $t \approx t'$, then $[t]_\approx = [t']_\approx$, but we can't have both $[t]_\approx \in R^{\mathfrak{M}/_\approx}$ and $[t]_\approx \notin R^{\mathfrak{M}/_\approx}$. However, Proposition 7.13 guarantees that this cannot happen.

Proposition 7.16. *$\mathfrak{M}/_\approx$ is well defined, i.e., if $t_1, \ldots, t_n, t_1', \ldots, t_n'$ are terms, and $t_i \approx t_i'$ then*

1. $[f(t_1, \ldots, t_n)]_\approx = [f(t'_1, \ldots, t'_n)]_\approx$, *i.e.,*

$$f(t_1, \ldots, t_n) \approx f(t'_1, \ldots, t'_n)$$

and

2. $\mathfrak{M} \vDash R(t_1, \ldots, t_n)$ *iff* $\mathfrak{M} \vDash R(t'_1, \ldots, t'_n)$, *i.e.,*

$$R(t_1, \ldots, t_n) \in \Gamma^* \text{ iff } R(t'_1, \ldots, t'_n) \in \Gamma^*.$$

Proof. Follows from Proposition 7.13 by induction on n. □

Lemma 7.17. $\mathfrak{M}/_\approx \vDash \varphi$ *iff* $\varphi \in \Gamma^*$ *for all sentences* φ.

Proof. By induction on φ, just as in the proof of Lemma 7.11. The only case that needs additional attention is when $\varphi \equiv t = t'$.

$$\begin{aligned}\mathfrak{M}/_\approx \vDash t = t' \text{ iff } [t]_\approx = [t']_\approx &\text{ (by definition of } \mathfrak{M}/_\approx)\\ \text{iff } t \approx t' &\text{ (by definition of } [t]_\approx)\\ \text{iff } t = t' \in \Gamma^* &\text{ (by definition of } \approx).\end{aligned}$$ □

Note that while $\mathfrak{M}(\Gamma^*)$ is always countable and infinite, $\mathfrak{M}/_\approx$ may be finite, since it may turn out that there are only finitely many classes $[t]_\approx$. This is to be expected, since Γ may contain sentences which require any structure in which they are true to be finite. For instance, $\forall x\, \forall y\, x = y$ is a consistent sentence, but is satisfied only in structures with a domain that contains exactly one element.

7.8 The Completeness Theorem

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

Theorem 7.18 (Completeness Theorem). *Let* Γ *be a set of sentences. If* Γ *is consistent, it is satisfiable.*

Proof. Suppose Γ is consistent. By Lemma 7.6, there is a saturated consistent set $\Gamma' \supseteq \Gamma$. By Lemma 7.8, there is a $\Gamma^* \supseteq \Gamma'$ which is consistent and complete. Since $\Gamma' \subseteq \Gamma^*$, for each formula $\varphi(x)$, Γ^* contains a sentence of the form $\exists x\, \varphi(x) \to \varphi(c)$ and so Γ^* is saturated. If Γ does not contain $=$, then by Lemma 7.11, $\mathfrak{M}(\Gamma^*) \vDash \varphi$ iff $\varphi \in \Gamma^*$. From this it follows in particular that for all $\varphi \in \Gamma$, $\mathfrak{M}(\Gamma^*) \vDash \varphi$, so Γ is satisfiable. If Γ does contain $=$, then by Lemma 7.17, for all sentences φ, $\mathfrak{M}/_\approx \vDash \varphi$ iff $\varphi \in \Gamma^*$. In particular, $\mathfrak{M}/_\approx \vDash \varphi$ for all $\varphi \in \Gamma$, so Γ is satisfiable. □

Corollary 7.19 (Completeness Theorem, Second Version). *For all* Γ *and sentences* φ*: if* $\Gamma \vDash \varphi$ *then* $\Gamma \vdash \varphi$.

Proof. Note that the Γ's in Corollary 7.19 and Theorem 7.18 are universally quantified. To make sure we do not confuse ourselves, let us restate Theorem 7.18 using a different variable: for any set of sentences Δ, if Δ is consistent, it is satisfiable. By contraposition, if Δ is not satisfiable, then Δ is inconsistent. We will use this to prove the corollary.

Suppose that $\Gamma \vDash \varphi$. Then $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable by Proposition 4.51. Taking $\Gamma \cup \{\neg\varphi\}$ as our Δ, the previous version of Theorem 7.18 gives us that $\Gamma \cup \{\neg\varphi\}$ is inconsistent. By Proposition 6.13, $\Gamma \vdash \varphi$. □

7.9 The Compactness Theorem

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

Definition 7.20. A set Γ of formulas is *finitely satisfiable* if and only if every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable.

Theorem 7.21 (Compactness Theorem). *The following hold for any sentences Γ and φ:*

1. *$\Gamma \models \varphi$ iff there is a finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models \varphi$.*
2. *Γ is satisfiable if and only if it is finitely satisfiable.*

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

Now suppose that Γ is finitely satisfiable. Then every finite subset $\Gamma_0 \subseteq \Gamma$ is satisfiable. By soundness (Corollary 6.23), every finite subset is consistent. Then Γ itself must be consistent by Proposition 6.11. By completeness (Theorem 7.18), since Γ is consistent, it is satisfiable. □

Example 7.22. In every model $\mathfrak{M}$ of a theory Γ, each term t of course picks out an element of $|\mathfrak{M}|$. Can we guarantee that it is also true that every element of $|\mathfrak{M}|$ is picked out by some term or other? In other words, are there theories Γ all models of which are covered? The compactness theorem shows that this is not the case if Γ has infinite models. Here's how to see this: Let $\mathfrak{M}$ be an infinite model of Γ, and let c be a constant symbol not in the language of Γ. Let Δ be the set of all sentences $c \neq t$ for t a term in the language $\mathcal{L}$ of Γ, i.e.,

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

A finite subset of $\Gamma \cup \Delta$ can be written as $\Gamma' \cup \Delta'$, with $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$. Since Δ' is finite, it can contain only finitely many terms. Let $a \in |\mathfrak{M}|$ be an element of $|\mathfrak{M}|$ not picked out by any of them, and let $\mathfrak{M}'$ be the structure that is just like $\mathfrak{M}$, but also $c^{\mathfrak{M}'} = a$. Since $a \neq \mathrm{Val}^{\mathfrak{M}}(t)$ for all t occuring in Δ', $\mathfrak{M}' \models \Delta'$. Since $\mathfrak{M} \models \Gamma$, $\Gamma' \subseteq \Gamma$, and c does not occur in Γ, also $\mathfrak{M}' \models \Gamma'$. Together, $\mathfrak{M}' \models \Gamma' \cup \Delta'$ for every finite subset $\Gamma' \cup \Delta'$ of $\Gamma \cup \Delta$. So every finite subset of $\Gamma \cup \Delta$ is satisfiable. By compactness, $\Gamma \cup \Delta$ itself is satisfiable. So there are models $\mathfrak{M} \models \Gamma \cup \Delta$. Every such $\mathfrak{M}$ is a model of Γ, but is not covered, since $\mathrm{Val}^{\mathfrak{M}}(c) \neq \mathrm{Val}^{\mathfrak{M}}(t)$ for all terms t of $\mathcal{L}$.

Example 7.23. Consider a language $\mathcal{L}$ containing the predicate symbol $<$, constant symbols 0, 1, and function symbols $+$, $\times$, $-$, $\div$. Let Γ be the set of all sentences in this

language true in $\mathfrak{Q}$ with domain $\mathbb{Q}$ and the obvious interpretations. Γ is the set of all sentences of $\mathcal{L}$ true about the rational numbers. Of course, in $\mathbb{Q}$ (and even in $\mathbb{R}$), there are no numbers which are greater than 0 but less than $1/k$ for all $k \in \mathbb{Z}^+$. Such a number, if it existed, would be an *infinitesimal:* non-zero, but infinitely small. The compactness theorem shows that there are models of Γ in which infinitesimals exist: Let Δ be $\{0 < c\} \cup \{c < (1 \div \overline{k}) \mid k \in \mathbb{Z}^+\}$ (where $\overline{k} = (1 + (1 + \cdots + (1 + 1)\ldots))$ with k 1's). For any finite subset Δ_0 of Δ there is a K such that all the sentences $c < (1 \div \overline{k})$ in Δ_0 have $k < K$. If we expand $\mathfrak{Q}$ to $\mathfrak{Q}'$ with $c^{\mathfrak{Q}'} = 1/K$ we have that $\mathfrak{Q}' \models \Gamma \cup \Delta_0$, and so $\Gamma \cup \Delta$ is finitely satisfiable (Exercise: prove this in detail). By compactness, $\Gamma \cup \Delta$ is satisfiable. Any model $\mathfrak{S}$ of $\Gamma \cup \Delta$ contains an infinitesimal, namely $c^{\mathfrak{S}}$.

Example 7.24. We know that first-order logic with identity predicate can express that the size of the domain must have some minimal size: The sentence $\varphi_{\geq n}$ (which says "there are at least n distinct objects") is true only in structures where $|\mathfrak{M}|$ has at least n objects. So if we take

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

then any model of Δ must be infinite. Thus, we can guarantee that a theory only has infinite models by adding Δ to it: the models of $\Gamma \cup \Delta$ are all and only the infinite models of Γ.

So first-order logic can express infinitude. The compactness theorem shows that it cannot express finitude, however. For suppose some set of sentences Λ were satisfied in all and only finite structures. Then $\Delta \cup \Lambda$ is finitely satisfiable. Why? Suppose $\Delta' \cup \Lambda' \subseteq \Delta \cup \Lambda$ is finite with $\Delta' \subseteq \Delta$ and $\Lambda' \subseteq \Lambda$. Let n be the largest number such that $\varphi_{\geq n} \in \Delta'$. Λ, being satisfied in all finite structures, has a model $\mathfrak{M}$ with finitely many but $\geq n$ elements. But then $\mathfrak{M} \models \Delta' \cup \Lambda'$. By compactness, $\Delta \cup \Lambda$ has an infinite model, contradicting the assumption that Λ is satisfied only in finite structures.

7.10 A Direct Proof of the Compactness Theorem

We can prove the Compactness Theorem directly, without appealing to the Completeness Theorem, using the same ideas as in the proof of the completeness theorem. In the proof of the Completeness Theorem we started with a consistent set Γ of sentences, expanded it to a consistent, saturated, and complete set Γ^* of sentences, and then showed that in the term model $\mathfrak{M}(\Gamma^*)$ constructed from Γ^*, all sentences of Γ are true, so Γ is satisfiable.

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

Proposition 7.25. *Suppose Γ is complete and finitely satisfiable. Then:*

1. *$(\varphi \wedge \psi) \in \Gamma$ iff both $\varphi \in \Gamma$ and $\psi \in \Gamma$.*
2. *$(\varphi \vee \psi) \in \Gamma$ iff either $\varphi \in \Gamma$ or $\psi \in \Gamma$.*
3. *$(\varphi \to \psi) \in \Gamma$ iff either $\varphi \notin \Gamma$ or $\psi \in \Gamma$.*

Lemma 7.26. *Every finitely satisfiable set Γ can be extended to a saturated finitely satisfiable set Γ'.*

Proposition 7.27. *Suppose Γ is complete, finitely satisfiable, and saturated.*

1. *$\exists x\, \varphi(x) \in \Gamma$ iff $\varphi(t) \in \Gamma$ for at least one closed term t.*
2. *$\forall x\, \varphi(x) \in \Gamma$ iff $\varphi(t) \in \Gamma$ for all closed terms t.*

Lemma 7.28. *Every finitely satisfiable set Γ can be extended to a complete and finitely satisfiable set Γ^*.*

Theorem 7.29 (Compactness). *Γ is satisfiable if and only if it is finitely satisfiable.*

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

Now suppose that Γ is finitely satisfiable. By Lemma 7.26, there is a finitely satisfiable, saturated set $\Gamma' \supseteq \Gamma$. By Lemma 7.28, Γ' can be extended to a complete and finitely satisfiable set Γ^*, and Γ^* is still saturated. Construct the term model $\mathfrak{M}(\Gamma^*)$ as in Definition 7.9. Note that Proposition 7.10 did not rely on the fact that Γ^* is consistent (or complete or saturated, for that matter), but just on the fact that $\mathfrak{M}(\Gamma^*)$ is covered. The proof of the Truth Lemma (Lemma 7.11) goes through if we replace references to Proposition 7.2 and Proposition 7.7 by references to Proposition 7.25 and Proposition 7.27 □

7.11 The Löwenheim-Skolem Theorem

The Löwenheim-Skolem Theorem says that if a theory has an infinite model, then it also has a model that is at most countably infinite. An immediate consequence of this fact is that first-order logic cannot express that the size of a structure is uncountable: any sentence or set of sentences satisfied in all uncountable structures is also satisfied in some countable structure.

Theorem 7.30. *If Γ is consistent then it has a countable model, i.e., it is satisfiable in a structure whose domain is either finite or countably infinite.*

Proof. If Γ is consistent, the structure $\mathfrak{M}$ delivered by the proof of the completeness theorem has a domain $|\mathfrak{M}|$ that is no larger than the set of the terms of the language $\mathcal{L}$. So $\mathfrak{M}$ is at most countably infinite. □

Theorem 7.31. *If Γ is a consistent set of sentences in the language of first-order logic without identity, then it has a countably infinite model, i.e., it is satisfiable in a structure whose domain is infinite and countable.*

Proof. If Γ is consistent and contains no sentences in which identity appears, then the structure $\mathfrak{M}$ delivered by the proof of the completness theorem has a domain $|\mathfrak{M}|$ identical to the set of terms of the language $\mathcal{L}'$. So $\mathfrak{M}$ is countably infinite, since $\mathrm{Trm}(\mathcal{L}')$ is. □

Example 7.32 (Skolem's Paradox). Zermelo-Fraenkel set theory **ZFC** is a very powerful framework in which practically all mathematical statements can be expressed, including facts about the sizes of sets. So for instance, **ZFC** can prove that the set $\mathbb{R}$ of real numbers is uncountable, it can prove Cantor's Theorem that the power set of any set is larger than the set itself, etc. If **ZFC** is consistent, its models are all infinite, and moreover, they all contain elements about which the theory says that

they are uncountable, such as the element that makes true the theorem of **ZFC** that the power set of the natural numbers exists. By the Löwenheim-Skolem Theorem, **ZFC** also has countable models—models that contain "uncountable" sets but which themselves are countable.

Problems

Problem 7.1. Complete the proof of Proposition 7.2.

Problem 7.2. Complete the proof of Proposition 7.13.

Problem 7.3. Use Corollary 7.19 to prove Theorem 7.18, thus showing that the two formulations of the completeness theorem are equivalent.

Problem 7.4. 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 7.18. Be sure to note any tacit uses of rules in these proofs.

Problem 7.5. Prove (1) of Theorem 7.21.

Problem 7.6. In the standard model of arithmetic $\mathfrak{N}$, there is no element $k \in |\mathfrak{N}|$ which satisfies every formula $\overline{n} < x$ (where $\overline{n}$ is $0'^{\cdots'}$ with n $'$'s). Use the compactness theorem to show that the set of sentences in the language of arithmetic which are true in the standard model of arithmetic $\mathfrak{N}$ are also true in a structure $\mathfrak{N}'$ that contains an element which *does* satisfy every formula $\overline{n} < x$.

Problem 7.7. Prove Proposition 7.25. Avoid the use of $\vdash$.

Problem 7.8. Prove Lemma 7.26. (Hint: The crucial step is to show that if Γ_n is finitely satisfiable, so is $\Gamma_n \cup \{\theta_n\}$, without any appeal to derivations or consistency.)

Problem 7.9. Prove Proposition 7.27.

Problem 7.10. Prove Lemma 7.28. (Hint: the crucial step is to show that if Γ_n is finitely satisfiable, then either $\Gamma_n \cup \{\varphi_n\}$ or $\Gamma_n \cup \{\neg\varphi_n\}$ is finitely satisfiable.)

Problem 7.11. Write out the complete proof of the Truth Lemma (Lemma 7.11) in the version required for the proof of Theorem 7.29.

Chapter 8

Basics of Model Theory

8.1 Reducts and Expansions

Often it is useful or necessary to compare languages which have symbols in common, as well as structures for these languages. The most comon case is when all the symbols in a language $\mathcal{L}$ are also part of a language $\mathcal{L}'$, i.e., $\mathcal{L} \subseteq \mathcal{L}'$. An $\mathcal{L}$-structure $\mathfrak{M}$ can then always be expanded to an $\mathcal{L}'$-structure by adding interpretations of the additional symbols while leaving the interpretations of the common symbols the same. On the other hand, from an $\mathcal{L}'$-structure $\mathfrak{M}'$ we can obtain an $\mathcal{L}$-structure simply by "forgetting" the interpretations of the symbols that do not occur in $\mathcal{L}$.

Definition 8.1. Suppose $\mathcal{L} \subseteq \mathcal{L}'$, $\mathfrak{M}$ is an $\mathcal{L}$-structure and $\mathfrak{M}'$ is an $\mathcal{L}'$-structure. $\mathfrak{M}$ is the *reduct* of $\mathfrak{M}'$ to $\mathcal{L}$, and $\mathfrak{M}'$ is an *expansion* of $\mathfrak{M}$ to $\mathcal{L}'$ iff

1. $|\mathfrak{M}| = |\mathfrak{M}'|$
2. For every constant symbol $c \in \mathcal{L}$, $c^{\mathfrak{M}} = c^{\mathfrak{M}'}$.
3. For every function symbol $f \in \mathcal{L}$, $f^{\mathfrak{M}} = f^{\mathfrak{M}'}$.
4. For every predicate symbol $P \in \mathcal{L}$, $P^{\mathfrak{M}} = P^{\mathfrak{M}'}$.

Proposition 8.2. *If an $\mathcal{L}$-structure $\mathfrak{M}$ is a reduct of an $\mathcal{L}'$-structure $\mathfrak{M}'$, then for all $\mathcal{L}$-sentences φ,*

$$\mathfrak{M} \vDash \varphi \text{ iff } \mathfrak{M}' \vDash \varphi.$$

Proof. Exercise. □

Definition 8.3. When we have an $\mathcal{L}$-structure $\mathfrak{M}$, and $\mathcal{L}' = \mathcal{L} \cup \{P\}$ is the expansion of $\mathcal{L}$ obtained by adding a single n-place predicate symbol P, and $R \subseteq |\mathfrak{M}|^n$ is an n-place relation, then we write $(\mathfrak{M}, R)$ for the expansion $\mathfrak{M}'$ of $\mathfrak{M}$ with $P^{\mathfrak{M}'} = R$.

8.2 Substructures

The domain of a structure $\mathfrak{M}$ may be a subset of another $\mathfrak{M}'$. But we should obviously only consider $\mathfrak{M}$ a "part" of $\mathfrak{M}'$ if not only $|\mathfrak{M}| \subseteq |\mathfrak{M}'|$, but $\mathfrak{M}$ and $\mathfrak{M}'$ "agree" in how they interpret the symbols of the language at least on the shared part $|\mathfrak{M}|$.

Definition 8.4. Given structures $\mathfrak{M}$ and $\mathfrak{M}'$ for the same language $\mathcal{L}$, we say that $\mathfrak{M}$ is a *substructure* of $\mathfrak{M}'$, and $\mathfrak{M}'$ an *extension* of $\mathfrak{M}$, written $\mathfrak{M} \subseteq \mathfrak{M}'$, iff

1. $|\mathfrak{M}| \subseteq |\mathfrak{M}'|$,
2. For each constant $c \in \mathcal{L}$, $c^{\mathfrak{M}} = c^{\mathfrak{M}'}$;
3. For each n-place predicate symbol $f \in \mathcal{L}$ $f^{\mathfrak{M}}(a_1, \ldots, a_n) = f^{\mathfrak{M}'}(a_1, \ldots, a_n)$ for all $a_1, \ldots, a_n \in |\mathfrak{M}|$.
4. For each n-place predicate symbol $R \in \mathcal{L}$, $\langle a_1, \ldots, a_n \rangle \in R^{\mathfrak{M}}$ iff $\langle a_1, \ldots, a_n \rangle \in R^{\mathfrak{M}'}$ for all $a_1, \ldots, a_n \in |\mathfrak{M}|$.

Remark 1. If the language contains no constant or function symbols, then any $N \subseteq |\mathfrak{M}|$ determines a substructure $\mathfrak{N}$ of $\mathfrak{M}$ with domain $|\mathfrak{N}| = N$ by putting $R^{\mathfrak{N}} = R^{\mathfrak{M}} \cap N^n$.

8.3 Overspill

Theorem 8.5. *If a set Γ of sentences has arbitrarily large finite models, then it has an infinite model.*

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

Proposition 8.6. *There is no sentence φ of any first-order language that is true in a structure $\mathfrak{M}$ if and only if the domain $|\mathfrak{M}|$ of the structure is infinite.*

Proof. If there were such a φ, its negation $\neg\varphi$ would be true in all and only the finite structures, and it would therefore have arbitrarily large finite models but it would lack an infinite model, contradicting Theorem 8.5. □

8.4 Isomorphic Structures

First-order structures can be alike in one of two ways. One way in which the can be alike is that they make the same sentences true. We call such structures *elementarily equivalent*. But structures can be very different and still make the same sentences true—for instance, one can be countable and the other not. This is because there are lots of features of a structure that cannot be expressed in first-order languages, either because the language is not rich enough, or because of fundamental limitations of first-order logic such as the Löwenheim-Skolem theorem. So another, stricter, aspect in which structures can be alike is if they are fundamentally the same, in the sense that they only differ in the objects that make them up, but not in their structural features. A way of making this precise is by the notion of an *isomorphism*.

Definition 8.7. Given two structures $\mathfrak{M}$ and $\mathfrak{M}'$ for the same language $\mathcal{L}$, we say that $\mathfrak{M}$ is *elementarily equivalent to* $\mathfrak{M}'$, written $\mathfrak{M} \equiv \mathfrak{M}'$, if and only if for every sentence φ of $\mathcal{L}$, $\mathfrak{M} \vDash \varphi$ iff $\mathfrak{M}' \vDash \varphi$.

Definition 8.8. Given two structures $\mathfrak{M}$ and $\mathfrak{M}'$ for the same language $\mathcal{L}$, we say that $\mathfrak{M}$ is *isomorphic to* $\mathfrak{M}'$, written $\mathfrak{M} \simeq \mathfrak{M}'$, if and only if there is a function $h\colon |\mathfrak{M}| \to |\mathfrak{M}'|$ such that:

1. h is injective: if $h(x) = h(y)$ then $x = y$;
2. h is surjective: for every $y \in |\mathfrak{M}'|$ there is $x \in |\mathfrak{M}|$ such that $h(x) = y$;
3. for every constant symbol c: $h(c^{\mathfrak{M}}) = c^{\mathfrak{M}'}$;
4. for every n-place predicate symbol P:
$$\langle a_1, \ldots, a_n\rangle \in P^{\mathfrak{M}} \quad \text{iff} \quad \langle h(a_1), \ldots, h(a_n)\rangle \in P^{\mathfrak{M}'};$$
5. for every n-place function symbol f:
$$h(f^{\mathfrak{M}}(a_1, \ldots, a_n)) = f^{\mathfrak{M}'}(h(a_1), \ldots, h(a_n)).$$

Theorem 8.9. *If $\mathfrak{M} \simeq \mathfrak{M}'$ then $\mathfrak{M} \equiv \mathfrak{M}'$.*

Proof. Let h be an isomorphism of $\mathfrak{M}$ onto $\mathfrak{M}'$. For any assignment s, $h \circ s$ is the composition of h and s, i.e., the assignment in $\mathfrak{M}'$ such that $(h \circ s)(x) = h(s(x))$. By induction on t and φ one can prove the stronger claims:

a. $h(\mathrm{Val}^{\mathfrak{M}}_s(t)) = \mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t)$.

b. $\mathfrak{M}, s \vDash \varphi$ iff $\mathfrak{M}', h \circ s \vDash \varphi$.

The first is proved by induction on the complexity of t.

1. If $t \equiv c$, then $\mathrm{Val}^{\mathfrak{M}}_s(c) = c^{\mathfrak{M}}$ and $\mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(c) = c^{\mathfrak{M}'}$. Thus, $h(\mathrm{Val}^{\mathfrak{M}}_s(t)) = h(c^{\mathfrak{M}}) = c^{\mathfrak{M}'}$ (by (3) of Definition 8.8) $= \mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t)$.
2. If $t \equiv x$, then $\mathrm{Val}^{\mathfrak{M}}_s(x) = s(x)$ and $\mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(x) = h(s(x))$. Thus, $h(\mathrm{Val}^{\mathfrak{M}}_s(x)) = h(s(x)) = \mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(x)$.
3. If $t \equiv f(t_1, \ldots, t_n)$, then
$$\mathrm{Val}^{\mathfrak{M}}_s(t) = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_s(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_s(t_n)) \quad \text{and}$$
$$\mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t) = f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t_n)).$$
The induction hypothesis is that for each i, $h(\mathrm{Val}^{\mathfrak{M}}_s(t_i)) = \mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t_i)$. So,
$$\begin{aligned} h(\mathrm{Val}^{\mathfrak{M}}_s(t)) &= h(f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}_s(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}}_s(t_n)) \\ &= h(f^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t_n)) && (8.1) \\ &= f^{\mathfrak{M}'}(\mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t_1), \ldots, \mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t_n)) && (8.2) \\ &= \mathrm{Val}^{\mathfrak{M}'}_{h\circ s}(t) \end{aligned}$$
Here, eq. (8.1) follows by induction hypothesis and eq. (8.2) by (5) of Definition 8.8.

Part (b) is left as an exercise.

If φ is a sentence, the assignments s and $h \circ s$ are irrelevant, and we have $\mathfrak{M} \vDash \varphi$ iff $\mathfrak{M}' \vDash \varphi$. □

Definition 8.10. An *automorphism* of a structure $\mathfrak{M}$ is an isomorphism of $\mathfrak{M}$ onto itself.

8.5 The Theory of a Structure

Every structure $\mathfrak{M}$ makes some sentences true, and some false. The set of all the sentences it makes true is called its *theory*. That set is in fact a theory, since anything it entails must be true in all its models, including $\mathfrak{M}$.

Definition 8.11. Given a structure $\mathfrak{M}$, the *theory* of $\mathfrak{M}$ is the set $\mathrm{Th}(\mathfrak{M})$ of sentences that are true in $\mathfrak{M}$, i.e., $\mathrm{Th}(\mathfrak{M}) = \{\varphi \mid \mathfrak{M} \vDash \varphi\}$.

We also use the term "theory" informally to refer to sets of sentences having an intended interpretation, whether deductively closed or not.

Proposition 8.12. *For any $\mathfrak{M}$, $\mathrm{Th}(\mathfrak{M})$ is complete.*

Proof. For any sentence φ either $\mathfrak{M} \vDash \varphi$ or $\mathfrak{M} \vDash \neg\varphi$, so either $\varphi \in \mathrm{Th}(\mathfrak{M})$ or $\neg\varphi \in \mathrm{Th}(\mathfrak{M})$. □

Proposition 8.13. *If $\mathfrak{N} \models \varphi$ for every $\varphi \in \mathrm{Th}(\mathfrak{M})$, then $\mathfrak{M} \equiv \mathfrak{N}$.*

Proof. Since $\mathfrak{N} \vDash \varphi$ for all $\varphi \in \mathrm{Th}(\mathfrak{M})$, $\mathrm{Th}(\mathfrak{M}) \subseteq \mathrm{Th}(\mathfrak{N})$. If $\mathfrak{N} \vDash \varphi$, then $\mathfrak{N} \nvDash \neg\varphi$, so $\neg\varphi \notin \mathrm{Th}(\mathfrak{M})$. Since $\mathrm{Th}(\mathfrak{M})$ is complete, $\varphi \in \mathrm{Th}(\mathfrak{M})$. So, $\mathrm{Th}(\mathfrak{N}) \subseteq \mathrm{Th}(\mathfrak{M})$, and we have $\mathfrak{M} \equiv \mathfrak{N}$. □

Remark 2. Consider $\mathfrak{R} = \langle \mathbb{R}, < \rangle$, the structure whose domain is the set $\mathbb{R}$ of the real numbers, in the language comprising only a 2-place predicate symbol interpreted as the $<$ relation over the reals. Clearly $\mathfrak{R}$ is uncountable; however, since $\mathrm{Th}(\mathfrak{R})$ is obviously consistent, by the Löwenheim-Skolem theorem it has a countable model, say $\mathfrak{S}$, and by Proposition 8.13, $\mathfrak{R} \equiv \mathfrak{S}$. Moreover, since $\mathfrak{R}$ and $\mathfrak{S}$ are not isomorphic, this shows that the converse of Theorem 8.9 fails in general.

8.6 Models of Arithmetic

The *standard model* of aritmetic is the structure $\mathfrak{N}$ with $|\mathfrak{N}| = \mathbb{N}$ in which 0, $'$, $+$, $\times$, and $<$ are interpreted as you would expect. That is, 0 is 0, $'$ is the successor function, $+$ is interpeted as addition and $\times$ as multiplication of the numbers in $\mathbb{N}$. Specifically,

$$\begin{aligned} 0^{\mathfrak{N}} &= 0 \\ {}'^{\mathfrak{N}}(n) &= n + 1 \\ +^{\mathfrak{N}}(n, m) &= n + m \\ \times^{\mathfrak{N}}(n, m) &= nm \end{aligned}$$

Of course, there are structures for $\mathcal{L}_A$ that have domains other than $\mathbb{N}$. For instance, we can take $\mathfrak{M}$ with domain $|\mathfrak{M}| = \{a\}^*$ (the finite sequences of the single symbol a, i.e., $\emptyset$, a, aa, aaa, ...), and interpretations

$$\begin{aligned} 0^{\mathfrak{M}} &= \emptyset \\ {}'^{\mathfrak{M}}(s) &= s \frown a \\ +^{\mathfrak{M}}(n, m) &= a^{n+m} \\ \times^{\mathfrak{M}}(n, m) &= a^{nm} \end{aligned}$$

These two structures are "essentially the same" in the sense that the only difference is the elements of the domains but not how the elements of the domains are related among each other by the interpretation functions. We say that the two structures are *isomorphic*.

It is an easy consequence of the compactness theorem that any theory true in $\mathfrak{N}$ also has models that are not isomorphic to $\mathfrak{N}$. Such structures are called *non-standard*. The interesting thing about them is that while the elements of a standard model (i.e., $\mathfrak{N}$, but also all structures isomorphic to it) are exhausted by the values of the standard numerals $\overline{n}$, i.e.,

$$|\mathfrak{N}| = \{\mathrm{Val}^{\mathfrak{N}}(\overline{n}) \mid n \in \mathbb{N}\}$$

that isn't the case in non-standard models: if $\mathfrak{M}$ is non-standard, then there is at least one $x \in |\mathfrak{M}|$ such that $x \neq \mathrm{Val}^{\mathfrak{M}}(\overline{n})$ for all n.

Definition 8.14. The theory of *true arithmetic* is the set of sentences satisfied in the standard model of arithmetic, i.e.,

$$\mathbf{TA} = \{\varphi \mid \mathfrak{N} \vDash \varphi\}.$$

Definition 8.15. The theory $\mathbf{Q}$ axiomatized by the following sentences is known as "Robinson's $\mathbf{Q}$" and is a very simple theory of arithmetic.

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

The set of sentences $\{Q_1, \ldots, Q_8\}$ are the axioms of $\mathbf{Q}$, so $\mathbf{Q}$ consists of all sentences entailed by them:

$$\mathbf{Q} = \{\varphi \mid \{Q_1, \ldots, Q_8\} \vDash \varphi\}.$$

Definition 8.16. Suppose $\varphi(x)$ is a formula in $\mathcal{L}_A$ with free variables x and $y_1, \ldots, y_n$. Then any sentence of the form

$$\forall y_1\, \ldots \forall y_n\, ((\varphi(0) \wedge \forall x\, (\varphi(x) \to \varphi(x'))) \to \forall x\, \varphi(x))$$

is an instance of the *induction schema*.

Peano arithmetic **PA** is the theory axiomatized by the axioms of $\mathbf{Q}$ together with all instances of the induction schema.

8.7 Standard Models of Arithmetic

The language of arithmetic $\mathcal{L}_A$ is obviously intended to be about numbers, specifically, about natural numbers. So, "the" standard model $\mathfrak{N}$ is special: it is the model we want to talk about. But in logic, we are often just interested in structural properties, and any two structures that are isomorphic share those. So we can be a bit more liberal, and consider any structure that is isomorphic to $\mathfrak{N}$ "standard."

Definition 8.17. A structure for $\mathcal{L}_A$ is *standard* if it is isomorphic to $\mathfrak{N}$.

Proposition 8.18. *If a structure $\mathfrak{M}$ standard, its domain is the set of values of the standard numerals, i.e.,*

$$|\mathfrak{M}| = \{\mathrm{Val}^{\mathfrak{M}}(\overline{n}) \mid n \in \mathbb{N}\}$$

Proof. Clearly, every $\mathrm{Val}^{\mathfrak{M}}(\overline{n}) \in |\mathfrak{M}|$. We just have to show that every $x \in |\mathfrak{M}|$ is equal to $\mathrm{Val}^{\mathfrak{M}}(\overline{n})$ for some n. Since $\mathfrak{M}$ is standard, it is isomorphic to $\mathfrak{N}$. Suppose $g\colon \mathbb{N} \to |\mathfrak{M}|$ is an isomorphism. Then $g(n) = g(\mathrm{Val}^{\mathfrak{N}}(\overline{n})) = \mathrm{Val}^{\mathfrak{M}}(\overline{n})$. But for every $x \in |\mathfrak{M}|$, there is an $n \in \mathbb{N}$ such that $g(n) = x$, since g is surjective. □

If a structure $\mathfrak{M}$ for $\mathcal{L}_A$ is standard, the elements of its domain can all be named by the standard numerals $\overline{0}$, $\overline{1}$, $\overline{2}$, …, i.e., the terms 0, $0'$, $0''$, etc. Of course, this does not mean that the elements of $|\mathfrak{M}|$ *are* the numbers, just that we can pick them out the same way we can pick out the numbers in $|\mathfrak{N}|$.

Proposition 8.19. *If $\mathfrak{M} \models \mathbf{Q}$, and $|\mathfrak{M}| = \{\mathrm{Val}^{\mathfrak{M}}(\overline{n}) \mid n \in \mathbb{N}\}$, then $\mathfrak{M}$ is standard.*

Proof. We have to show that $\mathfrak{M}$ is isomorphic to $\mathfrak{N}$. Consider the function $g\colon \mathbb{N} \to |\mathfrak{M}|$ defined by $g(n) = \mathrm{Val}^{\mathfrak{M}}(\overline{n})$. By the hypothesis, g is surjective. It is also injective: $\mathbf{Q} \vdash \overline{n} \neq \overline{m}$ whenever $n \neq m$. Thus, since $\mathfrak{M} \models \mathbf{Q}$, $\mathfrak{M} \models \overline{n} \neq \overline{m}$, whenever $n \neq m$. Thus, if $n \neq m$, then $\mathrm{Val}^{\mathfrak{M}}(\overline{n}) \neq \mathrm{Val}^{\mathfrak{M}}(\overline{m})$, i.e., $g(n) \neq g(m)$.

We also have to verify that g is an isomorphism.

1. We have $g(0^{\mathfrak{N}}) = g(0)$ since, $0^{\mathfrak{N}} = 0$. By definition of g, $g(0) = \mathrm{Val}^{\mathfrak{M}}(\overline{0})$. But $\overline{0}$ is just 0, and the value of a term which happens to be a constant symbol is given by what the structure assigns to that constant symbol, i.e., $\mathrm{Val}^{\mathfrak{M}}(0) = 0^{\mathfrak{M}}$. So we have $g(0^{\mathfrak{N}}) = 0^{\mathfrak{M}}$ as required.

2. $g(\prime^{\mathfrak{N}}(n)) = g(n+1)$, since $\prime$ in $\mathfrak{N}$ is the successor function on $\mathbb{N}$. Then, $g(n+1) = \mathrm{Val}^{\mathfrak{M}}(\overline{n+1})$ by definition of g. But $\overline{n+1}$ is the same term as $\overline{n}'$, so $\mathrm{Val}^{\mathfrak{M}}(\overline{n+1}) = \mathrm{Val}^{\mathfrak{M}}(\overline{n}')$. By the definition of the value function, this is $= \prime^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}(\overline{n}))$. Since $\mathrm{Val}^{\mathfrak{M}}(\overline{n}) = g(n)$ we get $g(\prime^{\mathfrak{N}}(n)) = \prime^{\mathfrak{M}}(g(n))$.

3. $g(+^{\mathfrak{N}}(n, m)) = g(n + m)$, since $+$ in $\mathfrak{N}$ is the addition function on $\mathbb{N}$. Then, $g(n + m) = \mathrm{Val}^{\mathfrak{M}}(\overline{n+m})$ by definition of g. But $\mathbf{Q} \vdash \overline{n+m} = (\overline{n} + \overline{m})$, so $\mathrm{Val}^{\mathfrak{M}}(\overline{n+m}) = \mathrm{Val}^{\mathfrak{M}}(\overline{n} + \overline{m})$. By the definition of the value function, this is $= +^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}(\overline{n}), \mathrm{Val}^{\mathfrak{M}}(\overline{m}))$. Since $\mathrm{Val}^{\mathfrak{M}}(\overline{n}) = g(n)$ and $\mathrm{Val}^{\mathfrak{M}}(\overline{m}) = g(m)$, we get $g(+^{\mathfrak{N}}(n, m)) = +^{\mathfrak{M}}(g(n), g(m))$.

4. $g(\times^{\mathfrak{N}}(n, m)) = \times^{\mathfrak{M}}(g(n), g(m))$: Exercise.

5. $\langle n, m\rangle \in <^{\mathfrak{N}}$ iff $n < m$. If $n < m$, then $\mathbf{Q} \vdash \overline{n} < \overline{m}$, and also $\mathfrak{M} \models \overline{n} < \overline{m}$. Thus $\langle \mathrm{Val}^{\mathfrak{M}}(\overline{n}), \mathrm{Val}^{\mathfrak{M}}(\overline{m})\rangle \in <^{\mathfrak{M}}$, i.e., $\langle g(n), g(m)\rangle \in <^{\mathfrak{M}}$. If $n \not< m$, then $\mathbf{Q} \vdash \neg\overline{n} < \overline{m}$, and consequently $\mathfrak{M} \nvDash \overline{n} < \overline{m}$. Thus, as before, $\langle g(n), g(m)\rangle \notin <^{\mathfrak{M}}$. Together, we get: $\langle n, m\rangle \in <^{\mathfrak{N}}$ iff $\langle g(n), g(m)\rangle \in <^{\mathfrak{M}}$. □

The function g is the most obvious way of defining a mapping from $\mathbb{N}$ to the domain of any other structure $\mathfrak{M}$ for $\mathcal{L}_A$, since every such $\mathfrak{M}$ contains elements named by $\overline{0}$, $\overline{1}$, $\overline{2}$, etc. So it isn't surprising that if $\mathfrak{M}$ makes at least some basic statements about the $\overline{n}$'s true in the same way that $\mathfrak{N}$ does, and g is also bijective, then g will turn into an isomorphism. In fact, if $|\mathfrak{M}|$ contains no elements other than what the $\overline{n}$'s name, it's the only one.

Proposition 8.20. *If $\mathfrak{M}$ is standard, then g from the proof of Proposition 8.19 is the only isomorphism from $\mathfrak{N}$ to $\mathfrak{M}$.*

Proof. Suppose $h\colon \mathbb{N} \to |\mathfrak{M}|$ is an isomorphism between $\mathfrak{N}$ and $\mathfrak{M}$. We show that $g = h$ by induction on n. If $n = 0$, then $g(0) = 0^{\mathfrak{M}}$ by definition of g. But since h is an isomorphism, $h(0) = h(0^{\mathfrak{N}}) = 0^{\mathfrak{M}}$, so $g(0) = h(0)$.

Now consider the case for $n + 1$. We have

$$\begin{aligned}
g(n+1) &= \mathrm{Val}^{\mathfrak{M}}(\overline{n+1}) && \text{by definition of } g\\
&= \mathrm{Val}^{\mathfrak{M}}(\overline{n}') && \text{since } \overline{n+1} \equiv \overline{n}'\\
&= '^{\mathfrak{M}}(\mathrm{Val}^{\mathfrak{M}}(\overline{n})) && \text{by definition of } \mathrm{Val}^{\mathfrak{M}}(t')\\
&= '^{\mathfrak{M}}(g(n)) && \text{by definition of } g\\
&= '^{\mathfrak{M}}(h(n)) && \text{by induction hypothesis}\\
&= h('^{\mathfrak{N}}(n)) && \text{since } h \text{ is an isomorphism}\\
&= h(n+1)
\end{aligned}$$

□

For any countably infinite set M, there's a bijection between $\mathbb{N}$ and M, so every such set M is potentially the domain of a standard model $\mathfrak{M}$. In fact, once you pick an object $z \in M$ and a suitable function s as $0^{\mathfrak{M}}$ and $'^{\mathfrak{M}}$, the interpretations of $+$, $\times$, and $<$ is already fixed. Only functions $s\colon M \to M \setminus \{z\}$ that are both injective and surjective are suitable in a standard model as $'^{\mathfrak{M}}$. The range of s cannot contain z, since otherwise $\forall x\, 0 \neq x'$ would be false. That sentence is true in $\mathfrak{N}$, and so $\mathfrak{M}$ also has to make it true. The function s has to be injective, since the successor function $'^{\mathfrak{N}}$ in $\mathfrak{N}$ is, and that $'^{\mathfrak{N}}$ is injective is expressed by a sentence true in $\mathfrak{N}$. It has to be surjective because otherwise there would be some $x \in M \setminus \{z\}$ not in the domain of s, i.e., the sentence $\forall x\, (x = 0 \vee \exists y\, y' = x)$ would be false in $\mathfrak{M}$—but it is true in $\mathfrak{N}$.

8.8 Non-Standard Models

We call a structure for $\mathcal{L}_A$ standard if it is isomorphic to $\mathfrak{N}$. If a structure isn't isomorphic to $\mathfrak{N}$, it is called non-standard.

Definition 8.21. A structure $\mathfrak{M}$ for $\mathcal{L}_A$ is *non-standard* if it is not isomorphic to $\mathfrak{N}$. The elements $x \in |\mathfrak{M}|$ which are equal to $\mathrm{Val}^{\mathfrak{M}}(\overline{n})$ for some $n \in \mathbb{N}$ are called *standard numbers* (of $\mathfrak{M}$), and those not, *non-standard numbers*.

By Proposition 8.18, any standard structure for $\mathcal{L}_A$ contains only standard elements. Consequently, a non-standard structure must contain at least one non-standard element. In fact, the existence of a non-standard element guarantees that the structure is non-standard.

Proposition 8.22. *If a structure $\mathfrak{M}$ for $\mathcal{L}_A$ contains a non-standard number, $\mathfrak{M}$ is non-standard.*

Proof. Suppose not, i.e., suppose $\mathfrak{M}$ standard but contains a non-standard number x. Let $g\colon \mathbb{N} \to |\mathfrak{M}|$ be an isomorphism. It is easy to see (by induction on n) that $g(\mathrm{Val}^{\mathfrak{N}}(\overline{n})) = \mathrm{Val}^{\mathfrak{M}}(\overline{n})$. In other words, g maps standard numbers of $\mathfrak{N}$ to standard numbers of $\mathfrak{M}$. If $\mathfrak{M}$ contains a non-standard number, g cannot be surjective, contrary to hypothesis. □

It is easy enough to specify non-standard structures for $\mathcal{L}_A$. For instance, take the structure with domain $\mathbb{Z}$ and interpret all non-logical symbols as usual. Since negative numbers are not values of $\overline{n}$ for any n, this structure is non-standard. Of course, it will not be a *model* of arithmetic in the sense that it makes the same sentences true as $\mathfrak{N}$. For instance, $\forall x\, x' \neq 0$ is false. However, we can prove that non-standard models of arithmetic exist easily enough, using the compactness theorem.

Proposition 8.23. *Let* $\mathbf{TA} = \{\varphi \mid \mathfrak{N} \vDash \varphi\}$ *be the theory of* $\mathfrak{N}$. $\mathbf{TA}$ *has a countable non-standard model.*

Proof. Expand $\mathcal{L}_A$ by a new constant symbol c and consider the set of sentences

$$\Gamma = \mathbf{TA} \cup \{c \neq \overline{0}, c \neq \overline{1}, c \neq \overline{2}, \dots\}$$

Any model $\mathfrak{M}^c$ of Γ would contain an element $x = c^{\mathfrak{M}}$ which is non-standard, since $x \neq \mathrm{Val}^{\mathfrak{M}}(\overline{n})$ for all $n \in \mathbb{N}$. Also, obviously, $\mathfrak{M}^c \vDash \mathbf{TA}$, since $\mathbf{TA} \subseteq \Gamma$. If we turn $\mathfrak{M}^c$ into a structure $\mathfrak{M}$ for $\mathcal{L}_A$ simply by forgetting about c, its domain still contains the non-standard x, and also $\mathfrak{M} \vDash \mathbf{TA}$. The latter is guaranteed since c does not occur in $\mathbf{TA}$. So, it suffices to show that Γ has a model.

We use the compactness theorem to show that Γ has a model. If every finite subset of Γ is satisfiable, so is Γ. Consider any finite subset $\Gamma_0 \subseteq \Gamma$. Γ_0 includes some sentences of $\mathbf{TA}$ and some of the form $c \neq \overline{n}$, but only finitely many. Suppose k is the largest number so that $c \neq \overline{k} \in \Gamma_0$. Define $\mathfrak{N}_k$ by expanding $\mathfrak{N}$ to include the interpretation $c^{\mathfrak{N}_k} = k + 1$. $\mathfrak{N}_k \vDash \Gamma_0$: if $\varphi \in \mathbf{TA}$, $\mathfrak{N}_k \vDash \varphi$ since $\mathfrak{N}_k$ is just like $\mathfrak{N}$ in all respects except c, and c does not occur in φ. And $\mathfrak{N}_k \vDash c \neq \overline{n}$, since $n \leq k$, and $\mathrm{Val}^{\mathfrak{N}_k}(c) = k + 1$. Thus, every finite subset of Γ is satisfiable. □

Problems

Problem 8.1. Prove Proposition 8.2.

Problem 8.2. Carry out the proof of (b) of Theorem 8.9 in detail. Make sure to note where each of the five properties characterizing isomorphisms of Definition 8.8 is used.

Problem 8.3. Show that for any structure $\mathfrak{M}$, if X is a definable subset of $\mathfrak{M}$, and h is an automorphism of $\mathfrak{M}$, then $X = \{h(x) \mid x \in X\}$ (i.e., X is fixed under h).

Problem 8.4. Show that the converse of Proposition 8.18 is false, i.e., give an example of a structure $\mathfrak{M}$ with $|\mathfrak{M}| = \{\mathrm{Val}^{\mathfrak{M}}(\overline{n}) \mid n \in \mathbb{N}\}$ that is not isomorphic to $\mathfrak{N}$.

Problem 8.5. Recall that $\mathbf{Q}$ contains the axioms

$$\forall x\, \forall y\, (x' = y' \to x = y) \tag{Q_1}$$
$$\forall x\, 0 \neq x' \tag{Q_2}$$
$$\forall x\, (x = 0 \vee \exists y\, x = y') \tag{Q_3}$$

Give structures $\mathfrak{M}_1, \mathfrak{M}_2, \mathfrak{M}_3$ such that

1. $\mathfrak{M}_1 \vDash Q_1$, $\mathfrak{M}_1 \vDash Q_2$, $\mathfrak{M}_1 \nvDash Q_3$;
2. $\mathfrak{M}_2 \vDash Q_1$, $\mathfrak{M}_2 \nvDash Q_2$, $\mathfrak{M}_2 \vDash Q_3$; and

3. $\mathfrak{M}_3 \nvDash Q_1, \mathfrak{M}_3 \vDash Q_2, \mathfrak{M}_3 \vDash Q_3$;

Obviously, you just have to specify $0^{\mathfrak{M}_i}$ and $'^{\mathfrak{M}_i}$ for each.

Part III

Second-order Logic

Chapter 9

Syntax and Semantics

9.1 Introduction

In first-order logic, we combine the non-logical symbols of a given language, i.e., its constant symbols, function symbols, and predicate symbols, with the logical symbols to express things about first-order structures. This is done using the notion of satisfaction, which relates a structure $\mathfrak{M}$, together with a variable assignment s, and a formula φ: $\mathfrak{M}, s \vDash \varphi$ holds iff what φ expresses when its constant symbols, function symbols, and predicate symbols are interpreted as $\mathfrak{M}$ says, and its free variables are interpreted as s says, is true. The interpretation of the identity predicate = is built into the definition of $\mathfrak{M}, s \vDash \varphi$, as is the interpretation of $\forall$ and $\exists$. The former is always interpreted as the identity relation on the domain $|\mathfrak{M}|$ of the structure, and the quantifiers are always interpreted as ranging over the entire domain. But, crucially, quantification is only allowed over elements of the domain, and so only object variables are allowed to follow a quantifier.

In second-order logic, both the language and the definition of satisfaction are extended to include free and bound function and predicate variables, and quantification over them. These variables are related to function symbols and predicate symbols the same way that object variables are related to constant symbols. They play the same role in the formation of terms and formulas of second-order logic, and quantification over them is handled in a similar way. In the *standard* semantics, the second-order quantifiers range over all possible objects of the right type (n-place functions from $|\mathfrak{M}|$ to $|\mathfrak{M}|$ for function variables, n-place relations for predicate variables). For instance, while $\forall v_0\, (P^1_0(v_0) \vee \neg P^1_0(v_0))$ is a formula in both first- and second-order logic, in the latter we can also consider $\forall V^1_0\, \forall v_0\, (V^1_0(v_0) \vee \neg V^1_0(v_0))$ and $\exists V^1_0\, \forall v_0\, (V^1_0(v_0) \vee \neg V^1_0(v_0))$. Since these contain no free varaibles, they are sentences of second-order logic. Here, V^1_0 is a second-order 1-place predicate variable. The allowable interpretations of V^1_0 are the same that we can assign to a 1-place predicate symbol like P^1_0, i.e., subsets of $|\mathfrak{M}|$. Quantification over them then amounts to saying that $\forall v_0\, (V^1_0(v_0) \vee \neg V^1_0(v_0))$ holds for all ways of assigning a subset of $|\mathfrak{M}|$ as the

value of V_0^1, or for at least one. Since every set either contains or fails to contain a given object, both are true in any structure.

9.2 Terms and Formulas

Like in first-order logic, expressions of second-order logic are built up from a basic vocabulary containing *variables*, *constant symbols*, *predicate symbols* and sometimes *function symbols*. From them, together with logical connectives, quantifiers, and punctuation symbols such as parentheses and commas, *terms* and *formulas* are formed. The difference is that in addition to variables for objects, second-order logic also contains variables for relations and functions, and allows quantification over them. So the logical symbols of second-order logic are those of first-order logic, plus:

1. A countably infinite set of second-order relation variables of every arity n: V_0^n, V_1^n, V_2^n, $\dots$
2. A countably infinite set of second-order function variables: u_0^n, u_1^n, u_2^n, $\dots$

Just as we use x, y, z as meta-variables for first-order variables v_i, we'll use X, Y, Z, etc., as metavariables for V_i^n and u, v, etc., as meta-variables for u_i^n.

The non-logical symbols of a second-order language are specified the same way a first-order language is: by listing its constant symbols, function symbols, and predicate symbols.

In first-order logic, the identity predicate = is usually included. In first-order logic, the non-logical symbols of a language $\mathcal{L}$ are crucial to allow us to express anything interesting. There are of course sentences that use no non-logical symbols, but with only = it is hard to say anything interesting. In second-order logic, since we have an unlimited supply of relation and function variables, we can say anything we can say in a first-order language even without a special supply of non-logical symbols.

Definition 9.1 (Second-order Terms). The set of *second-order terms* of $\mathcal{L}$, $\mathrm{Trm}^2(\mathcal{L})$, is defined by adding to Definition 4.4 the clause

1. If u is an n-place function variable and $t_1, \dots, t_n$ are terms, then $u(t_1, \dots, t_n)$ is a term.

So, a second-order term looks just like a first-order term, except that where a first-order term contains a function symbol f_i^n, a second-order term may contain a function variable u_i^n in its place.

Definition 9.2 (Second-order formula). The set of *second-order formulas* $\mathrm{Frm}^2(\mathcal{L})$ of the language $\mathcal{L}$ is defined by adding to Definition 4.4 the clauses

1. If X is an n-place predicate variable and $t_1, \dots, t_n$ are second-order terms of $\mathcal{L}$, then $X(t_1, \dots, t_n)$ is an atomic formula.
2. If φ is a formula and u is a function variable, then $\forall u\, \varphi$ is a formula.
3. If φ is a formula and X is a predicate variable, then $\forall X\, \varphi$ is a formula.
4. If φ is a formula and u is a function variable, then $\exists u\, \varphi$ is a formula.
5. If φ is a formula and X is a predicate variable, then $\exists X\, \varphi$ is a formula.

9.3 Satisfaction

To define the satisfaction relation $\mathfrak{M}, s \vDash \varphi$ for second-order formulas, we have to extend the definitions to cover second-order variables. The notion of a structure is the same for second-order logic as it is for first-order logic. There is only a diffence for variable assignments s: these now must not just provide values for the first-order variables, but also for the second-order variables.

Definition 9.3 (Variable Assignment). A *variable assignment* s for a structure $\mathfrak{M}$ is a function which maps each

1. object variable v_i to an element of $|\mathfrak{M}|$, i.e., $s(v_i) \in |\mathfrak{M}|$
2. n-place relation variable V_i^n to an n-place relation on $|\mathfrak{M}|$, i.e., $s(V_i^n) \subseteq |\mathfrak{M}|^n$;
3. n-place function variable u_i^n to an n-place function from $|\mathfrak{M}|$ to $|\mathfrak{M}|$, i.e., $s(u_i^n)\colon |\mathfrak{M}|^n \to |\mathfrak{M}|$;

A structure assigns a value to each constant symbol and function symbol, and a second-order variable assigns objects and functions to each object and function variable. Together, they let us assign a value to every term.

Definition 9.4 (Value of a Term). If t is a term of the language $\mathcal{L}$, $\mathfrak{M}$ is a structure for $\mathcal{L}$, and s is a variable assignment for $\mathfrak{M}$, the *value* $\mathrm{Val}_s^{\mathfrak{M}}(t)$ is defined as for first-order terms, plus the following clause:

$t \equiv u(t_1, \ldots, t_n)$:

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

Definition 9.5 (x-Variant). If s is a variable assignment for a structure $\mathfrak{M}$, then any variable assignment s' for $\mathfrak{M}$ which differs from s at most in what it assigns to x is called an *x-variant* of s. If s' is an x-variant of s we write $s \sim_x s'$. (Similarly for second-order variables X or u.)

Definition 9.6 (Satisfaction). For second-order formulas φ, the definition of satisfaction is like Definition 4.35 with the addition of:

1. $\varphi \equiv X^n(t_1, \ldots, t_n)$: $\mathfrak{M}, s \vDash \varphi$ iff $\langle \mathrm{Val}_s^{\mathfrak{M}}(t_1), \ldots, \mathrm{Val}_s^{\mathfrak{M}}(t_n)\rangle \in s(X^n)$.
2. $\varphi \equiv \forall X\, \psi$: $\mathfrak{M}, s \vDash \varphi$ iff for every X-variant s' of s, $\mathfrak{M}, s' \vDash \psi$.
3. $\varphi \equiv \exists X\, \psi$: $\mathfrak{M}, s \vDash \varphi$ iff there is an X-variant s' of s so that $\mathfrak{M}, s' \vDash \psi$.
4. $\varphi \equiv \forall u\, \psi$: $\mathfrak{M}, s \vDash \varphi$ iff for every u-variant s' of s, $\mathfrak{M}, s' \vDash \psi$.
5. $\varphi \equiv \exists u\, \psi$: $\mathfrak{M}, s \vDash \varphi$ iff there is an u-variant s' of s so that $\mathfrak{M}, s' \vDash \psi$.

Example 9.7. Consider the formula $\forall z\, (X(z) \leftrightarrow \neg Y(z))$. It contains no second-order quantifiers, but does contain the second-order variables X and Y (here understood to be one-place). The corresponding first-order sentence $\forall z\, (P(z) \leftrightarrow \neg R(z))$ says that whatever falls under the interpretation of P does not fall under the interpretation of R and vice versa. In a structure, the interpretation of a predicate symbol P is given by the interpretation $P^{\mathfrak{M}}$. But for second-order variables like X and Y, the

interpretation is provided, not by the structure itself, but by a variable assignment. Since the second-order formula is not a sentence (in includes free variables X and Y), it is only satisfied relative to a structure $\mathfrak{M}$ together with a variable assignment s.

$\mathfrak{M}, s \vDash \forall z\,(X(z) \leftrightarrow \neg Y(z))$ whenever the elements of $s(X)$ are not elements of $s(Y)$, and vice versa, i.e., iff $s(Y) = |\mathfrak{M}| \setminus s(X)$. So for instance, take $|\mathfrak{M}| = \{1, 2, 3\}$. Since no predicate symbols, function symbols, or constant symbols are involved, the domain of $\mathfrak{M}$ is all that is relevant. Now for $s_1(X) = \{1, 2\}$ and $s_1(Y) = \{3\}$, we have $\mathfrak{M}, s_1 \vDash \forall z\,(X(z) \leftrightarrow \neg Y(z))$.

By contrast, if we have $s_2(X) = \{1, 2\}$ and $s_2(Y) = \{2, 3\}$, $\mathfrak{M}, s_2 \nvDash \forall z\,(X(z) \leftrightarrow \neg Y(z))$. That's because there is a z-variant s_2' of s_2 with $s_2'(z) = 2$ where $\mathfrak{M}, s_2' \vDash X(z)$ (since $2 \in s_2'(X)$) but $\mathfrak{M}, s_2' \nvDash \neg Y(z)$ (since also $s_2'(z) \in s_2'(Y)$).

Example 9.8. $\mathfrak{M}, s \vDash \exists Y\,(\exists y\, Y(y) \wedge \forall z\,(X(z) \leftrightarrow \neg Y(z)))$ if there is an $s' \sim_Y s$ such that $\mathfrak{M}, s' \vDash (\exists y\, Y(y) \wedge \forall z\,(X(z) \leftrightarrow \neg Y(z)))$. And that is the case iff $s'(Y) \neq \emptyset$ (so that $\mathfrak{M}, s' \vDash \exists y\, Y(y)$) and, as in the previous example, $s'(Y) = |\mathfrak{M}| \setminus s'(X)$. In other words, $\mathfrak{M}, s \vDash \exists Y\,(\exists y\, Y(y) \wedge \forall z\,(X(z) \leftrightarrow \neg Y(z)))$ iff $|\mathfrak{M}| \setminus s(X)$ is non-empty, i.e., $s(X) \neq |\mathfrak{M}|$. So, the formula is satisfied, e.g., if $|\mathfrak{M}| = \{1, 2, 3\}$ and $s(X) = \{1, 2\}$, but not if $s(X) = \{1, 2, 3\} = |\mathfrak{M}|$.

Since the formula is not satisfied whenever $s(X) = |\mathfrak{M}|$, the sentence

$$\forall X\, \exists Y\,(\exists y\, Y(y) \wedge \forall z\,(X(z) \leftrightarrow \neg Y(z)))$$

is never satisfied: For any structure $\mathfrak{M}$, the assignment $s(X) = |\mathfrak{M}|$ will make the sentence false. On the other hand, the sentence

$$\exists X\, \exists Y\,(\exists y\, Y(y) \wedge \forall z\,(X(z) \leftrightarrow \neg Y(z)))$$

is satisfied relative to any assignment s, since we can always find an X-variant s' of s with $s'(X) \neq |\mathfrak{M}|$.

9.4 Semantic Notions

The central logical notions of *validity*, *entailment*, and *satisfiability* are defined the same way for second-order logic as they are for first-order logic, except that the underlying satisfaction relation is now that for second-order formulas. A second-order sentence, of course, is a formula in which all variables, including predicate and function variables, are bound.

Definition 9.9 (Validity). A sentence φ is *valid*, $\vDash \varphi$, iff $\mathfrak{M} \vDash \varphi$ for every structure $\mathfrak{M}$.

Definition 9.10 (Entailment). A set of sentences Γ *entails* a sentence φ, $\Gamma \vDash \varphi$, iff for every structure $\mathfrak{M}$ with $\mathfrak{M} \vDash \Gamma$, $\mathfrak{M} \vDash \varphi$.

Definition 9.11 (Satisfiability). A set of sentences Γ is *satisfiable* if $\mathfrak{M} \vDash \Gamma$ for some structure $\mathfrak{M}$. If Γ is not satisfiable it is called *unsatisfiable*.

9.5 Expressive Power

Quantification over second-order variables is responsible for an immense increase in the expressive power of the language over that of first-order logic. Second-order existential quantification lets us say that functions or relations with certain properties

exists. In first-order logic, the only way to do that is to specify a non-logical symbol (i.e., a function symbol or predicate symbol) for this purpose. Second-order universal quantification lets us say that all subsets of, relations on, or functions from the domain to the domain have a property. In first-order logic, we can only say that the subsets, relations, or functions assigned to one of the non-logical symbols of the language have a property. And when we say that subsets, relations, functions exist that have a property, or that all of them have it, we can use second-order quantification in specifying this property as well. This lets us define relations not definable in first-order logic, and express properties of the domain not expressible in first-order logic.

Definition 9.12. If $\mathfrak{M}$ is a structure for a language $\mathcal{L}$, a relation $R \subseteq |\mathfrak{M}|^2$ is *definable* in $\mathcal{L}$ if there is some formula $\varphi_R(x, y)$ with only the variables x and y free, such that $R(a, b)$ holds (i.e., $\langle a, b\rangle \in R$) iff $\mathfrak{M}, s \vDash \varphi_R(x, y)$ for $s(x) = a$ and $s(y) = b$.

Example 9.13. In first-order logic we can define the identity relation $\mathrm{Id}_{|\mathfrak{M}|}$ (i.e., $\{\langle a, a\rangle \mid a \in |\mathfrak{M}|\}$) by the formula $x = y$. In second-order logic, we can define this relation *without* $=$. For if a and b are the same element of $|\mathfrak{M}|$, then they are elements of the same subsets of $|\mathfrak{M}|$ (since sets are determined by their elements). Conversely, if a and b are different, then they are not elements of the same subsets: e.g., $a \in \{a\}$ but $b \notin \{a\}$ if $a \neq b$. So "being elements of the same subsets of $|\mathfrak{M}|$" is a relation that holds of a and b iff $a = b$. It is a relation that can be expressed in second-order logic, since we can quantify over all subsets of $|\mathfrak{M}|$. Hence, the following formula defines $\mathrm{Id}_{|\mathfrak{M}|}$:

$$\forall X\, (X(x) \leftrightarrow X(y))$$

Example 9.14. If R is a two-place predicate symbol, $R^{\mathfrak{M}}$ is a two-place relation on $|\mathfrak{M}|$. Perhaps somewhat confusingly, we'll use R as the predicate symbol for R and for the relation $R^{\mathfrak{M}}$ itself. The *transitive closure* R^* of R is the relation that holds between a and b iff for some $c_1, \ldots, c_k$, $R(a, c_1)$, $R(c_1, c_2)$, $\ldots$, $R(c_k, b)$ holds. This includes the case if $k = 0$, i.e., if $R(a, b)$ holds, so does $R^*(a, b)$. This means that $R \subseteq R^*$. In fact, R^* is the smallest relation that includes R and that is transitive. We can say in second-order logic that X is a transitive relation that includes R:

$$\psi_R(X) \equiv \forall x\, \forall y\, (R(x, y) \rightarrow X(x, y)) \wedge \forall x\, \forall y\, \forall z\, ((X(x, y) \wedge X(y, z)) \rightarrow X(x, z)).$$

The first conjunct says that $R \subseteq X$ and the second that X is transitive.

To say that X is the smallest such relation is to say that it is itself included in every relation that includes R and is transitive. So we can define the transitive closure of R by the formula

$$R^*(X) \equiv \psi_R(X) \wedge \forall Y\, (\psi_R(Y) \rightarrow \forall x\, \forall y\, (X(x, y) \rightarrow Y(x, y))).$$

We have $\mathfrak{M}, s \vDash R^*(X)$ iff $s(X) = R^*$. The transitive closure of R cannot be expressed in first-order logic.

9.6 Describing Infinite and Countable Domains

A set M is (Dedekind) infinite iff there is an injective function $f : M \rightarrow M$ which is not surjective, i.e., with $\mathrm{dom}(f) \neq M$. In first-order logic, we can consider a one-place

function symbol f and say that the function $f^{\mathfrak{M}}$ assigned to it in a structure $\mathfrak{M}$ is injective and $\mathrm{ran}(f) \neq |\mathfrak{M}|$:

$$\forall x\, \forall y\, (f(x) = f(y) \to x = y) \wedge \exists y\, \forall x\, y \neq f(x).$$

If $\mathfrak{M}$ satisfies this sentence, $f^{\mathfrak{M}} : |\mathfrak{M}| \to |\mathfrak{M}|$ is injective, and so $|\mathfrak{M}|$ must be infinite. If $|\mathfrak{M}|$ is infinite, and hence such a function exists, we can let $f^{\mathfrak{M}}$ be that function and $\mathfrak{M}$ will satisfy the sentence. However, this requires that our language contains the non-logical symbol f we use for this purpose. In second-order logic, we can simply say that such a function *exists*. This no-longer requires f, and we obtain the sentence in pure second-order logic

$$\mathrm{Inf} \equiv \exists u\, (\forall x\, \forall y\, (u(x) = u(y) \to x = y) \wedge \exists y\, \forall x\, y \neq u(x)).$$

$\mathfrak{M} \vDash \mathrm{Inf}$ iff $|\mathfrak{M}|$ is infinite. We can then define $\mathrm{Fin} \equiv \neg\mathrm{Inf}$; $\mathfrak{M} \vDash \mathrm{Fin}$ iff $|\mathfrak{M}|$ is finite. No single sentence of pure first-order logic can express that the domain is infinite although an infinite set of them can. There is no set of sentences of pure first-order logic that is satisfied in a structure iff its domain is finite.

Proposition 9.15. $\mathfrak{M} \vDash \mathrm{Inf}$ *iff* $|\mathfrak{M}|$ *is infinite.*

Proof. $\mathfrak{M} \vDash \mathrm{Inf}$ iff $\mathfrak{M}, s \vDash \forall x\, \forall y\, (u(x) = u(y) \to x = y) \wedge \exists y\, \forall x\, y \neq u(x)$ for some s. If it does, $s(u)$ is an injective function, and some $y \in |\mathfrak{M}|$ is not in the domain of $s(u)$. Conversely, if there is an injective $f\colon |\mathfrak{M}| \to |\mathfrak{M}|$ with $\mathrm{dom}(f) \neq |\mathfrak{M}|$, then $s(u) = f$ is such a variable assignment. □

A set M is countable if there is an enumeration

$$m_0, m_1, m_2, \ldots$$

of its elements (without repetitions but possibly finite). Such an enumeration exists iff there is an element $z \in M$ and a function $f\colon M \to M$ such that $z, f(z), f(f(z)), \ldots$, are all the elements of M. For if the enumeration exists, $z = m_0$ and $f(m_k) = m_{k+1}$ (or $f(m_k) = m_k$ if m_k is the last element of the enumeration) are the requisite element and function. On the other hand, if such a z and f exist, then $z, f(z), f(f(z)), \ldots$, is an enumeration of M, and M is countable. We can express the existence of z and f in second-order logic to produce a sentence true in a structure iff the structure is countable:

$$\mathrm{Count} \equiv \exists z\, \exists u\, \forall X\, ((X(z) \wedge \forall x\, (X(x) \to X(u(x)))) \to \forall x\, X(x))$$

Proposition 9.16. $\mathfrak{M} \vDash \mathrm{Count}$ *iff* $|\mathfrak{M}|$ *is countable.*

Proof. Suppose $|\mathfrak{M}|$ is countable, and let $m_0, m_1, \ldots$, be an enumeration. By removing repetions we can guarantee that no m_k appears twice. Define $f(m_k) = m_{k+1}$ and let $s(z) = m_0$ and $s(u) = f$. We show that

$$\mathfrak{M}, s \vDash \forall X\, ((X(z) \wedge \forall x\, (X(x) \to X(u(x)))) \to \forall x\, X(x))$$

Suppose $s' \sim_X s$ is arbitrary, and let $M = s'(X)$. Suppose further that $\mathfrak{M}, s' \vDash (X(z) \wedge \forall x\, (X(x) \to X(u(x))))$. Then $s'(z) \in M$ and whenever $x \in M$, also $s'(u)(x) \in M$. In other words, since $s' \sim_X s$, $m_0 \in M$ and if $x \in M$ then $f(x) \in M$, so $m_0 \in M$,

$m_1 = f(m_0) \in M$, $m_2 = f(f(m_0)) \in M$, etc. Thus, $M = |\mathfrak{M}|$, and so $\mathfrak{M}, s' \vDash \forall x\, X(x)$. Since s' was an arbitrary X-variant of s, we are done: $\mathfrak{M} \vDash \mathrm{Count}$.

Now assume that $\mathfrak{M} \vDash \mathrm{Count}$, i.e.,

$$\mathfrak{M}, s \vDash \forall X\, ((X(z) \wedge \forall x\, (X(x) \to X(u(x)))) \to \forall x\, X(x))$$

for some s. Let $m = s(z)$ and $f = s(u)$ and consider $M = \{m, f(m), f(f(m)), \dots\}$. Let s' be the X-variant of s with $s(X) = M$. Then

$$\mathfrak{M}, s' \vDash (X(z) \wedge \forall x\, (X(x) \to X(u(x)))) \to \forall x\, X(x)$$

by assumption. Also, $\mathfrak{M}, s' \vDash X(z)$ since $s'(X) = M \ni m = s'(z)$, and also $\mathfrak{M}, s' \vDash \forall x\, (X(x) \to X(u(x)))$ since whenever $x \in M$ also $f(x) \in M$. So, since both antecedent and conditional are satisfied, the consequent must also be: $\mathfrak{M}, s' \vDash \forall x\, X(x)$. But that means that $M = |\mathfrak{M}|$, and so $|\mathfrak{M}|$ is countable since M is, by definition. □

Problems

Problem 9.1. Show that $\forall X\, (X(x) \to X(y))$ (note: $\to$ not $\leftrightarrow$!) defines $\mathrm{Id}_{|\mathfrak{M}|}$.

Problem 9.2. The sentence Inf $\wedge$ Count is true in all and only countably infinite domains. Adjust the definition of Count so that it becomes a different sentence that directly expresses that the domain is countably infinite, and prove that it does.

Chapter 10

Metatheory of Second-order Logic

10.1 Introduction

First-order logic has a number of nice properties. We know it is not decidable, but at least it is axiomatizable. That is, there are proof systems for first-order logic which are sound and complete, i.e., they give rise to a derivability relation $\vdash$ with the property that for any set of sentences Γ and sentence Q, $\Gamma \models \varphi$ iff $\Gamma \vdash \varphi$. This means in particular that the validities of first-order logic are computably enumerable. There is a computable function $f : \mathbb{N} \to \text{Sent}(\mathcal{L})$ such that the values of f are all and only the valid sentences of $\mathcal{L}$. This is so because derivations can be enumerated, and those that derive a single sentence are then mapped to that sentence. Second-order logic is more expressive than first-order logic, and so it is in general more complicated to capture its validities. In fact, we'll show that second-order logic is not only undecidable, but its validities are not even computably enumerable. This means there can be no sound and complete proof system for second-order logic (although sound, but incomplete proof systems are available and in fact are important objects of research).

First-order logic also has two more properties: it is compact (if every finite subset of a set Γ of sentences is satisfiable, Γ itself is satisfiable) and the Löwenheim-Skolem Theorem holds for it (if Γ has an infinite model it has a countably infinite model). Both of these results fail for second-order logic. Again, the reason is that second-order logic can express facts about the size of domains that first-order logic cannot.

10.2 Second-order Arithmetic

Recall that the theory **PA** of Peano arithmetic includes the eight axioms of **Q**,

$$\forall x\, x' \neq 0$$
$$\forall x \forall y\, (x' = y' \to x = y)$$
$$\forall x\, (x = 0 \vee \exists y\, x = y')$$
$$\forall x\, (x + 0) = x$$
$$\forall x \forall y\, (x + y') = (x + y)'$$
$$\forall x\, (x \times 0) = 0$$
$$\forall x \forall y\, (x \times y') = ((x \times y) + x)$$
$$\forall x \forall y\, (x < y \leftrightarrow \exists z\, (z' + x) = y)$$

plus all sentences of the form

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

The latter is a "schema," i.e., a pattern that generates infinitely many sentences of the language of arithmetic, one for each formula $\varphi(x)$. We call this schema the (first-order) *axiom schema of induction*. In *second-order* Peano arithmetic $\mathbf{PA}^2$, induction can be stated as a single sentence. $\mathbf{PA}^2$ consists of the first eight axioms above plus the (second-order) *induction axiom*:

$$\forall X\,(X(0) \wedge \forall x\,(X(x) \to X(x'))) \to \forall x\,X(x).$$

It says that if a subset X of the domain contains $0^{\mathfrak{M}}$ and with any $x \in |\mathfrak{M}|$ also contains $'^{\mathfrak{M}}(x)$ (i.e., it is "closed under successor") it contains everything in the domain (i.e., $X = |\mathfrak{M}|$).

The induction axiom guarantees that any structure satisfying it contains only those elements of $|\mathfrak{M}|$ the axioms require to be there, i.e., the values of $\overline{n}$ for $n \in \mathbb{N}$. A model of $\mathbf{PA}^2$ contains no non-standard numbers.

Theorem 10.1. *If* $\mathfrak{M} \models \mathbf{PA}^2$ *then* $|\mathfrak{M}| = \{\mathrm{Val}^{\mathfrak{M}}(\overline{n}) \mid n \in \mathbb{N}\}$.

Proof. Let $N = \{\mathrm{Val}^{\mathfrak{M}}(\overline{n}) \mid n \in \mathbb{N}\}$, and suppose $\mathfrak{M} \models \mathbf{PA}^2$. Of course, for any $n \in \mathbb{N}$, $\mathrm{Val}^{\mathfrak{M}}(\overline{n}) \in |\mathfrak{M}|$, so $N \subseteq |\mathfrak{M}|$.

Now for inclusion in the other direction. Consider a variable assignment s with $s(X) = N$. By assumption,

$$\mathfrak{M} \models \forall X\,(X(0) \wedge \forall x\,(X(x) \to X(x'))) \to \forall x\,X(x), \text{ thus}$$
$$\mathfrak{M}, s \models (X(0) \wedge \forall x\,(X(x) \to X(x'))) \to \forall x\,X(x).$$

Consider the antecedent of this conditional. $\mathrm{Val}^{\mathfrak{M}}(0) \in N$, and so $\mathfrak{M}, s \models X(0)$. The second conjunct, $\forall x\,(X(x) \to X(x'))$ is also satisfied. For suppose $x \in N$. By definition of N, $x = \mathrm{Val}^{\mathfrak{M}}(\overline{n})$ for some n. That gives $'^{\mathfrak{M}}(x) = \mathrm{Val}^{\mathfrak{M}}(\overline{n+1}) \in N$. So, $'^{\mathfrak{M}}(x) \in N$.

We have that $\mathfrak{M}, s \models X(0) \wedge \forall x\,(X(x) \to X(x'))$. Consequently, $\mathfrak{M}, s \models \forall x\,X(x)$. But that means that for every $x \in |\mathfrak{M}|$ we have $x \in s(X) = N$. So, $|\mathfrak{M}| \subseteq N$. □

Corollary 10.2. *Any two models of* $\mathbf{PA}^2$ *are isomorphic.*

Proof. By Theorem 10.1, the domain of any model of $\mathbf{PA}^2$ is exhausted by $\mathrm{Val}^{\mathfrak{M}}(\overline{n})$. Any such model is also a model of $\mathbf{Q}$. By Proposition 8.19, any such model is standard, i.e., isomorphic to $\mathfrak{N}$. □

Above we defined $\mathbf{PA}^2$ as the theory that contains the first eight arithmetical axioms plus the second-order induction axiom. In fact, thanks to the expressive power of second-order logic, only the *first two* of the arithmetical axioms plus induction are needed for second-order Peano arithmetic.

Proposition 10.3. *Let* $\mathbf{PA}^{2\dagger}$ *be the second-order theory containing the first two arithmetical axioms (the successor axioms) and the second-order induction axiom. Then* $\leq$, $+$, *and* $\times$ *are definable in* $\mathbf{PA}^{2\dagger}$.

Proof. To show that $\leq$ is definable, we have to find a formula $\varphi_{\leq}(x, y)$ such that $\mathfrak{N} \vDash \varphi_{\leq}(\overline{n}, \overline{m})$ iff $n \leq m$. Consider the formula

$$\psi(x, Y) \equiv Y(x) \wedge \forall y\,(Y(y) \rightarrow Y(y'))$$

Clearly, $\psi(\overline{n}, Y)$ is satisfied by a set $Y \subseteq \mathbb{N}$ iff $\{m \mid n \leq m\} \subseteq Y$, so we can take $\varphi_{\leq}(x, y) \equiv \forall Y\,(\psi(x, Y) \rightarrow Y(y))$. □

Corollary 10.4. $\mathfrak{M} \vDash \mathbf{PA}^2$ *iff* $\mathfrak{M} \vDash \mathbf{PA}^{2\dagger}$.

Proof. Immediate from Proposition 10.3. □

10.3 Second-order Logic is not Axiomatizable

Theorem 10.5. *Second-order logic is undecidable.*

Proof. A first-order sentence is valid in first-order logic iff it is valid in second-order logic, and first-order logic is undecidable. □

Theorem 10.6. *There is no sound and complete proof system for second-order logic.*

Proof. Let φ be a sentence in the language of arithmetic. $\mathfrak{N} \vDash \varphi$ iff $\mathbf{PA}^2 \vDash \varphi$. Let P be the conjunction of the nine axioms of $\mathbf{PA}^2$. $\mathbf{PA}^2 \vDash \varphi$ iff $\vDash P \rightarrow \varphi$, i.e., $\mathfrak{M} \vDash P \rightarrow \varphi$. Now consider the sentence $\forall z\,\forall u\,\forall u'\,\forall u''\,\forall L\,(P' \rightarrow \varphi')$ resulting by replacing 0 by z, $'$ by the one-place function variable u, $+$ and $\times$ by the two-place function-variables u' and u'', respectively, and $<$ by the two-place relation variable L and universally quantifying. It is a valid sentence of pure second-order logic iff the original sentence was valid iff $\mathbf{PA}^2 \vDash \varphi$ iff $\mathfrak{N} \vDash \varphi$. Thus if there were a sound and complete proof system for second-order logic, we could use it to define a computable enumeration $f\colon \mathbb{N} \rightarrow \mathrm{Sent}(\mathcal{L}_A)$ of the sentences true in $\mathfrak{N}$. This function would be representable in $\mathbf{Q}$ by some first-order formula $\psi_f(x, y)$. Then the formula $\exists x\,\psi_f(x, y)$ would define the set of true first-order sentences of $\mathfrak{N}$, contradicting Tarski's Theorem. □

10.4 Second-order Logic is not Compact

Call a set of sentences Γ *finitely satisfiable* if every one of its finite subsets is satisfiable. First-order logic has the property that if a set of sentences Γ is finitely satisfiable, it is satisfiable. This property is called *compactness*. It has an equivalent version involving entailment: if $\Gamma \vDash \varphi$, then already $\Gamma_0 \vDash \varphi$ for some finite subset $\Gamma_0 \subseteq \Gamma$. In this version it is an immediate corollary of the completeness theorem: for if $\Gamma \vDash \varphi$, by completeness $\Gamma \vdash \varphi$. But a derivation can only make use of finitely many sentences of Γ.

Compactness is not true for second-order logic. There are sets of second-order sentences that are finitely satisfiable but not satisfiable, and that entail some φ without a finite subset entailing φ.

Theorem 10.7. *Second-order logic is not compact.*

Proof. Recall that

$$\mathrm{Inf} \equiv \exists u\,(\forall x\,\forall y\,(u(x) = u(y) \rightarrow x = y) \wedge \exists y\,\forall x\; y \neq u(x))$$

is satisfied in a structure iff its domain is infinite. Let $\varphi^{\geq n}$ be a sentence that asserts that the domain has at least n elements, e.g.,

$$\varphi^{\geq n} \equiv \exists x_1 \ldots \exists x_n \, (x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \cdots \wedge x_{n-1} \neq x_n).$$

Consider the set of sentences

$$\Gamma = \{\neg \mathrm{Inf}, \varphi^{\geq 1}, \varphi^{\geq 2}, \varphi^{\geq 3}, \ldots\}.$$

It is finitely satisfiable, since for any finite subset $\Gamma_0 \subseteq \Gamma$ there is some k so that $\varphi^{\geq k} \in \Gamma$ but no $\varphi^{\geq n} \in \Gamma$ for $n > k$. If $|\mathfrak{M}|$ has k elements, $\mathfrak{M} \models \Gamma_0$. But, Γ is not satisfiable: if $\mathfrak{M} \models \neg \mathrm{Inf}$, $|\mathfrak{M}|$ must be finite, say, of size k. Then $\mathfrak{M} \not\models \varphi^{\geq k+1}$. □

10.5 The Löwenheim-Skolem Theorem Fails for Second-order Logic

The (Downward) Löwenheim-Skolem Theorem states that every set of sentences with an infinite model has a countable model. It, too, is a consequence of the completeness theorem: the proof of completeness generates a model for any consistent set of sentences, and that model is countable. There is also an Upward Löwenheim-Skolem Theorem, which guarantees that if a set of sentences has a countably infinite model it also has an uncountable model. Both theorems fail in second-order logic.

Theorem 10.8. *The Löwenheim-Skolem Theorem fails for second-order logic: There are sentences with infinite models but no countable models.*

Proof. Recall that

$$\mathrm{Count} \equiv \exists z \, \exists u \, \forall X \, ((X(z) \wedge \forall x \, (X(x) \rightarrow X(u(x)))) \rightarrow \forall x \, X(x))$$

is true in a structure $\mathfrak{M}$ iff $|\mathfrak{M}|$ is countable, so $\neg\mathrm{Count}$ is true in $\mathfrak{M}$ iff $|\mathfrak{M}|$ is uncountable. There are such structures—take any uncountable set as the domain, e.g., $\wp(\mathbb{N})$ or $\mathbb{R}$. So $\neg\mathrm{Count}$ has infinite models but no countable models. □

Theorem 10.9. *There are sentences with countably infinite but no uncountable models.*

Proof. $\mathrm{Count} \wedge \mathrm{Inf}$ is true in $\mathbb{N}$ but not in any structure $\mathfrak{M}$ with $|\mathfrak{M}|$ uncountable. □

Problems

Problem 10.1. Complete the proof of Proposition 10.3.

Problem 10.2. Give an example of a set Γ and a sentence φ so that $\Gamma \models \varphi$ but for every finite subset $\Gamma_0 \subseteq \Gamma$, $\Gamma_0 \not\models \varphi$.

Part IV

Intuitionistic Logic

Chapter 11

Introduction

11.1 Constructive Reasoning

In constrast to extensions of classical logic by modal operators or second-order quantifiers, intuitionistic logic is "non-classical" in that it restricts classical logic. Classical logic is *non-constructive* in various ways. Intuitionistic logic is intended to capture a more "constructive" kind of reasoning characteristic of a kind of constructive mathematics. The following examples may serve to illustrate some of the underlying motivations.

Suppose someone claimed that they had determined a natural number n with the property that if n is even, the Riemann hypothesis is true, and if n is odd, 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 they seem to have reduced the problem to one of calculation, that is, to the determination of whether a specific number is even or not.

What is the magic value of n? They describe it as follows: n is the natural number that is equal to 2 if the Riemann hypothesis is true, and 3 otherwise.

Angrily, you demand your money back. From a classical point of view, the description above does in fact determine a unique value of n; but what you really want is a value of n 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 11.1. *There are irrational numbers a and b such that a^b is rational.*

Proof. Consider $\sqrt{2}^{\sqrt{2}}$. If this is rational, we are done: we can let $a = b = \sqrt{2}$. Otherwise, it is irrational. Then we have

$$(\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = \sqrt{2}^{2} = 2,$$

which is 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 capture a kind of reasoning where moves like the one in the first proof are disallowed. Proving the existence of an x satisfying $\varphi(x)$ means that you have to give a specific x, and a proof that it satisfies φ, like in the second proof. Proving that φ or ψ holds requires that you can prove one or the other.

Formally speaking, intuitionistic logic is what you get if you restrict a proof system for classical logic in a certain way. From the mathematical point of view, these are just formal deductive systems, but, as already noted, they are intended to capture 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.

11.2 The Brouwer-Heyting-Kolmogorov Interpretation

There is an informal constructive interpretation of the intuitionist connectives, usually known as the Brouwer-Heyting-Kolmogorov interpretation. It uses the notion of a "construction," which you may think of as a constructive proof. (We don't use "proof" in the BHK interpretation so as not to get confused with the notion of a derivation in a formal proof system.) Based on this intuitive notion, the BHK interpretation explains the meanings of the intuitionistic connectives.

1. We assume that we know what constitutes a construction of an atomic statement.

2. A construction of $\varphi_1 \wedge \varphi_2$ is a pair $\langle M_1, M_2\rangle$ where M_1 is a construction of φ_1 and M_2 is a construction of A_2.

3. A construction of $\varphi_1 \vee \varphi_2$ is a pair $\langle s, M\rangle$ where s is 1 and M is a construction of φ_1, or s is 2 and M is a construction of φ_2.

4. A construction of $\varphi \to \psi$ is a function that converts a construction of φ into a construction of ψ.
5. There is no construction for $\bot$ (absurdity).
6. $\neg\varphi$ is defined as synonym for $\varphi \to \bot$. That is, a construction of $\neg\varphi$ is a function converting a construction of φ into a construction of $\bot$.

Example 11.2. Take $\neg\bot$ for example. A construction of it is a function which, given any construction of $\bot$ as input, provides a construction of $\bot$ as output. Obviously, the identity function Id is such a construction: given a construction M of $\bot$, $\mathrm{Id}(M) = M$ yields a construction of $\bot$.

Generally speaking, $\neg\varphi$ means "A construction of φ is impossible".

Example 11.3. Let us prove $\varphi \to \neg\neg\varphi$ for any proposition φ, which is $\varphi \to ((\varphi \to \bot) \to \bot)$. The construction should be a function f that, given a construction M of φ, returns a construction $f(M)$ of $(\varphi \to \bot) \to \bot$. Here is how f constructs the construction of $(\varphi \to \bot) \to \bot$: We have to define a function g which, when given a construction h of $\varphi \to \bot$ as input, outputs a construction of $\bot$. We can define g as follows: apply the input h to the construction M of φ (that we received earlier). Since the output $h(M)$ of h is a construction of $\bot$, $f(M)(h) = h(M)$ is a construction of $\bot$ if M is a construction of φ.

Example 11.4. Let us give a construction for $\neg(\varphi \wedge \neg\varphi)$, i.e., $(\varphi \wedge (\varphi \to \bot)) \to \bot$. This is a function f which, given as input a construction M of $\varphi \wedge (\varphi \to \bot)$, yields a construction of $\bot$. A construction of a conjunction $\psi_1 \wedge \psi_2$ is a pair $\langle N_1, N_2\rangle$ where N_1 is a construction of ψ_1 and N_2 is a construction of ψ_2. We can define functions p_1 and p_2 which recover from a construction of $\psi_1 \wedge \psi_2$ the constructions of ψ_1 and ψ_2, respectively:

$$p_1(\langle N_1, N_2\rangle) = N_1$$
$$p_2(\langle N_1, N_2\rangle) = N_2$$

Here is what f does: First it applies p_1 to its input M. That yields a construction of φ. Then it applies p_2 to M, yielding a construction of $\varphi \to \bot$. Such a construction, in turn, is a function $p_2(M)$ which, if given as input a construction of φ, yields a construction of $\bot$. In other words, if we apply $p_2(M)$ to $p_1(M)$, we get a construction of $\bot$. Thus, we can define $f(M) = p_2(M)(p_1(M))$.

Example 11.5. Let us give a construction of $((\varphi \wedge \psi) \to \chi) \to (\varphi \to (\psi \to \chi))$, i.e., a function f which turns a construction g of $(\varphi \wedge \psi) \to \chi$ into a construction of $(\varphi \to (\psi \to \chi))$. The construction g is itself a function (from constructions of $\varphi \wedge \psi$ to constructions of C). And the output $f(g)$ is a function h_g from constructions of φ to functions from constructions of ψ to constructions of χ.

Ok, this is confusing. We have to construct a certain function h_g, which will be the output of f for input g. The input of h_g is a construction M of φ. The output of $h_g(M)$ should be a function k_M from constructions N of ψ to constructions of χ. Let $k_{g,M}(N) = g(\langle M, N\rangle)$. Remember that $\langle M, N\rangle$ is a construction of $\varphi \wedge \psi$. So $k_{g,M}$ is a construction of $\psi \to \chi$: it maps constructions N of ψ to constructions of χ. Now let $h_g(M) = k_{g,M}$. That's a function that maps constructions M of φ to constructions $k_{g,M}$ of $\psi \to \chi$. Now let $f(g) = h_g$. That's a function that maps constructions g of $(\varphi \wedge \psi) \to \chi$ to constructions of $\varphi \to (\psi \to \chi)$. Whew!

The statement $\varphi \vee \neg\varphi$ is called the Law of Excluded Middle. We can prove it for some specific φ (e.g., $\bot \vee \neg\bot$), but not in general. This is because the intuitionistic disjunction requires a construction of one of the disjuncts, but there are statements which currently can neither be proved nor refuted (say, Goldbach's conjecture). However, you can't refute the law of excluded middle either: that is, $\neg\neg(\varphi \vee \neg\varphi)$ holds.

Example 11.6. To prove $\neg\neg(\varphi \vee \neg\varphi)$, we need a function f that transforms a construction of $\neg(\varphi \vee \neg\varphi)$, i.e., of $(\varphi \vee (\varphi \to \bot)) \to \bot$, into a construction of $\bot$. In other words, we need a function f such that $f(g)$ is a construction of $\bot$ if g is a construction of $\neg(\varphi \vee \neg\varphi)$.

Suppose g is a construction of $\neg(\varphi \vee \neg\varphi)$, i.e., a function that transforms a construction of $\varphi \vee \neg\varphi$ into a construction of $\bot$. A construction of $\varphi \vee \neg\varphi$ is a pair $\langle s, M\rangle$ where either $s = 1$ and M is a construction of φ, or $s = 2$ and M is a construction of $\neg\varphi$. Let h_1 be the function mapping a construction M_1 of φ to a construction of $\varphi \vee \neg\varphi$: it maps M_1 to $\langle 1, M_2\rangle$. And let h_2 be the function mapping a construction M_2 of $\neg\varphi$ to a construction of $\varphi \vee \neg\varphi$: it maps M_2 to $\langle 2, M_2\rangle$.

Let k be $g \circ h_1$: it is a function which, if given a construction of φ, returns a construction of $\bot$, i.e., it is a construction of $\varphi \to \bot$ or $\neg\varphi$. Now let l be $g \circ h_2$. It is a function which, given a construction of $\neg\varphi$, provides a construction of $\bot$. Since k is a construction of $\neg\varphi$, $l(k)$ is a construction of $\bot$.

Together, what we've done is describe how we can turn a construction g of $\neg(\varphi \vee \neg\varphi)$ into a construction of $\bot$, i.e., the function f mapping a construction g of $\neg(\varphi \vee \neg\varphi)$ to the construction $l(k)$ of $\bot$ is a construction of $\neg\neg(\varphi \vee \neg\varphi)$.

As you can see, using the BHK interpretation to show the intuitionistic validity of formulas quickly becomes cumbersome and confusing. Luckily, there are better derivation systems for intuitionistic logic, and more precise semantic interpretations.

11.3 Natural Deduction

Natural deduction without the RAA rules is a standard derivation system for intuitionistic logic. We repeat the rules here and indicate the motivation using the BHK interpretation. In each case, we can think of a rule which allows us to conclude that if the premises have constructions, so does the conclusion.

Since natural deduction derivations have undischarged assumptions, we should consider such a derivation, say, of φ from undischarged assumptions Γ, as a function that turns constructions of all $\psi \in \Gamma$ into a construction of φ. If there is a derivation of φ from no undischarged assumptions, then there is a construction of φ in the sense of the BHK interpretation. For the purpose of the discussion, however, we'll suppress the Γ when not needed.

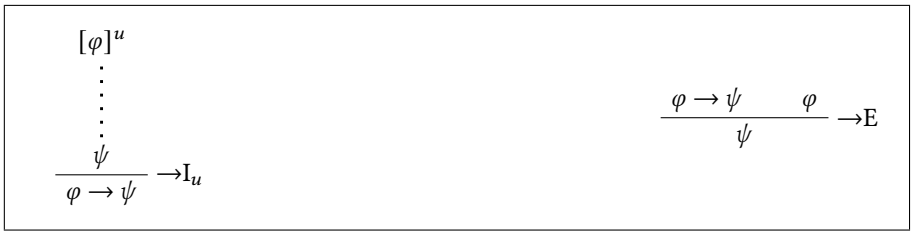
An assumption φ by itself is a derivation of φ from the undischarged assumption φ. This agrees with the BHK-interpretation: the identity function on constructions turns any construction of φ into a construction of φ.

Conjunction

Suppose we have constructions N_1, N_2 of φ_1 and φ_2, respectively. Then we also have a construction $\varphi_1 \wedge \varphi_2$, namely the pair $\langle N_1, N_2 \rangle$.

A construction of $\varphi_1 \wedge \varphi_1$ on the BHK interpretation is a pair $\langle N_1, N_2 \rangle$. So assume we have such a pair. Then we also have a construction of each conjunct: N_1 is a construction of φ_1 and N_2 is a construction of φ_2.

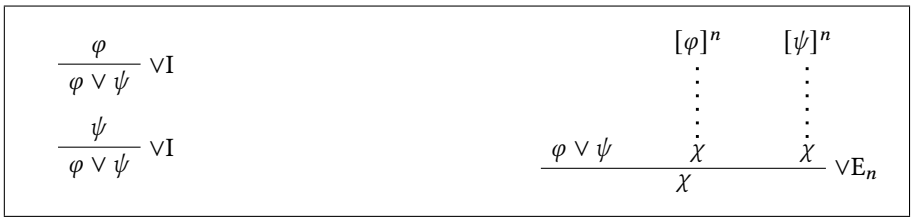
Conditional



If we have a derivation of ψ from undischarged assumption φ, then there is a function f that turns constructions of φ into constructions of ψ. That same function is a construction of $\varphi \to \psi$. So, if the premise of $\to$I has a construction conditional on a construction of φ, the conclusion $\varphi \to \psi$ has a construction.

On the other hand, suppose there are constructions N of φ and f of $\varphi \to \psi$. A construction of $\varphi \to \psi$ is a function that turns constructions of φ into constructions of ψ. So, $f(N)$ is a construction of ψ, i.e., the conclusion of $\to$E has a construction.

Disjunction



If we have a construction N_i of φ_i we can turn it into a construction $\langle i, N_i \rangle$ of $\varphi_1 \vee \varphi_2$. On the other hand, suppose we have a construction of $\varphi_1 \vee \varphi_2$, i.e., a pair $\langle i, N_i \rangle$ where N_i is a construction of φ_i, and also functions f_1, f_2, which turn constructions of φ_1, φ_2, respectively, into constructions of χ. Then $f_i(N_i)$ is a construction of χ, the conclusion of $\vee$E.

Absurdity

$$\frac{\bot}{\varphi}\,\bot\mathrm{E}$$

If we have a derivation of $\bot$ from undischarged assumptions $\psi_1, \ldots, \psi_n$, then there is a function $f(M_1, \ldots, M_n)$ that turns constructions of $\psi_1, \ldots, \psi_n$ into a construction of $\bot$. Since $\bot$ has no construction, there cannot be any constructions of all of $\psi_1, \ldots, \psi_n$ either. Hence, f also has the property that *if* $M_1, \ldots, M_n$ are constructions of $\psi_1, \ldots, \psi_n$, respectively, *then* $f(M_1, \ldots, M_n)$ is a construction of φ.

Rules for ¬

Since $\neg\varphi$ is defined as $\varphi \to \bot$, we strictly speaking do not need rules for ¬. But if we did, this is what they'd look like:

$$\frac{\begin{array}{c}[\varphi]^n\\ \vdots\\ \bot\end{array}}{\neg\varphi}\,\neg\mathrm{I}_n \qquad\qquad \frac{\neg\varphi \qquad \varphi}{\bot}\,\neg\mathrm{E}$$

Examples of Derivations

1. $\vdash \varphi \to (\neg\varphi \to \bot)$, i.e., $\vdash \varphi \to ((\varphi \to \bot) \to \bot)$

$$\dfrac{\dfrac{\dfrac{[\varphi]^2 \qquad [\varphi \to \bot]^1}{\bot}\,\to\mathrm{E}}{(\varphi \to \bot) \to \bot}\,\to\mathrm{I}_1}{\varphi \to (\varphi \to \bot) \to \bot}\,\to\mathrm{I}_2$$

2. $\vdash ((\varphi \wedge \psi) \to \chi) \to (\varphi \to (\psi \to \chi))$

$$\dfrac{\dfrac{\dfrac{\dfrac{[(\varphi \wedge \psi) \to \chi]^3 \qquad \dfrac{[\varphi]^2 \qquad [\psi]^1}{\varphi \wedge \psi}\,\wedge\mathrm{I}}{\chi}\,\to\mathrm{E}}{\psi \to \chi}\,\to\mathrm{I}_1}{\varphi \to (\psi \to \chi)}\,\to\mathrm{I}_2}{((\varphi \wedge \psi) \to \chi) \to (\varphi \to (\psi \to \chi))}\,\to\mathrm{I}_3$$

3. $\vdash \neg(\varphi \wedge \neg\varphi)$, i.e., $\vdash (\varphi \wedge (\varphi \to \bot)) \to \bot$

$$\dfrac{\dfrac{\dfrac{[\varphi \wedge (\varphi \to \bot)]^1}{\varphi \to \bot}\,\wedge\mathrm{E} \qquad \dfrac{[\varphi \wedge (\varphi \to \bot)]^1}{\varphi}\,\wedge\mathrm{E}}{\bot}\,\to\mathrm{E}}{(\varphi \wedge (\varphi \to \bot)) \to \bot}\,\to\mathrm{I}_1$$

4. $\vdash \neg\neg(\varphi \vee \neg\varphi)$, i.e., $\vdash ((\varphi \vee (\varphi \to \bot)) \to \bot) \to \bot$

$$\dfrac{\dfrac{\dfrac{[((\varphi \vee (\varphi \to \bot)) \to \bot]^2 \qquad \dfrac{[\varphi]^1}{\varphi \vee (\varphi \to \bot)}\vee\mathrm{I}}{\bot}\to\mathrm{E}}{\varphi \to \bot}\to\mathrm{I}_1}{\dfrac{[((\varphi \vee (\varphi \to \bot)) \to \bot]^2 \qquad \dfrac{\varphi \to \bot}{\varphi \vee (\varphi \to \bot)}\vee\mathrm{I}}{\bot}\to\mathrm{E}}{((\varphi \vee (\varphi \to \bot)) \to \bot) \to \bot}\to\mathrm{I}_2$$

Proposition 11.7. *If $\Gamma \vdash \varphi$ in intuitionistic logic, $\Gamma \vdash \varphi$ in classical logic. In particular, if φ is an intuitionistic theorem, it is also a classical theorem.*

Proof. Every natural deduction rule is also a rule in classical natural deduction, so every derivation in intuitionistic logic is also a derivation in classical logic. □

Problems

Problem 11.1. Give derivations in intutionistic logic of the following.

1. $(\neg\varphi \vee \psi) \to (\varphi \to \psi)$
2. $\neg\neg\neg\varphi \to \neg\varphi$
3. $\neg\neg(\varphi \wedge \psi) \leftrightarrow (\neg\neg\varphi \wedge \neg\neg\psi)$

Chapter 12

Semantics

12.1 Introduction

No logic is satisfactorily described without a semantics, and intuitionistic logic is no exception. Whereas for classical logic, the semantics based on valuations is canonical, there are several competing semantics for intuitionistic logic. None of them are completely satisfactory in the sense that they give an intuitionistically acceptable account of the meanings of the connectives.

The semantics based on relational models, similar to the semantics for modal logics, is perhaps the most popular one. In this semantics, propositional variables are assigned to worlds, and these worlds are related by an accessibility relation. That relation is always a partial order, i.e., it is reflexive, antisymmetric, and transitive.

Intuitively, you might think of these worlds as states of knowledge or "evidentiary situations." A state w' is accessible from w iff, for all we know, w' is a possible (future) state of knowledge, i.e., one that is compatible with what's known at w. Once a proposition is known, it can't become un-known, i.e., whenever φ is known at w and Rww', φ is known at w' as well. So "knowledge" is monotonic with respect to the accessibility relation.

If we define "φ is known" as in epistemic logic as "true in all epistemic alternatives," then $\varphi \wedge \psi$ is known at w if in all epistemic alternatives, both φ and ψ are known. But since knowledge is monotonic and R is reflexive, that means that $\varphi \wedge \psi$ is known at w iff φ and ψ are known at w. For the same reason, $\varphi \vee \psi$ is known at w iff at least one of them is known. So for $\wedge$ and $\vee$, the truth conditions of the connectives coincide with those in classical logic.

The truth conditions for the conditional, however, differ from classical logic. $\varphi \to \psi$ is known at w iff at no w' with Rww', φ is known without ψ also being known. This is not the same as the condition that φ is unknown or ψ is known at w. For if we know neither φ nor ψ at w, there might be a future epistemic state w' with Rww' such that at w', φ is known without also coming to know ψ.

We know $\neg\varphi$ only if there is no possible future epistemic state in which we know φ. Here the idea is that if φ were knowable, then in some possible future epistemic state φ becomes known. Since we can't know $\bot$, in that future epistemic state, we would know φ but not know $\bot$.

On this interpretation the principle of excluded middle fails. For there are some φ which we don't yet know, but which we might come to know. For such an φ, both φ and $\neg\varphi$ are unknown, so $\varphi \vee \neg\varphi$ is not known. But we do know, e.g., that $\neg(\varphi \wedge \neg\varphi)$.

For no future state in which we know both φ and $\neg\varphi$ is possible, and we know this independently of whether or not we know φ or $\neg\varphi$.

Relational models are not the only available semantics for intuitionistic logic. The topological semantics is another: here propositions are interpreted as open sets in a topological space, and the connectives are interpreted as operations on these sets (e.g., $\wedge$ corresponds to intersection).

12.2 Relational models

In order to give a precise semantics for intuitionistic propositional logic, we have to give a definition of what counts as a model relative to which we can evaluate formulas. On the basis of such a definition it is then also possible to define semantics notions such as validity and entailment. One such semantics is given by relational models.

Definition 12.1. A relational model for intuitionistic propositional logic is a triple $\mathfrak{M} = \langle W, R, V\rangle$, where

1. W is a non-empty set,
2. R is a partial order (i.e., a reflexive, antisymmetric, and transitive binary relation) on W, and
3. V is a function assigning to each propositional variable p a subset of W, such that
4. V is monotone with respect to R, i.e., if $w \in V(p)$ and Rww', then $w' \in V(p)$.

Definition 12.2. We define the notion of φ *being true at* w *in* $\mathfrak{M}$, $\mathfrak{M}, w \Vdash \varphi$, inductively as follows:

1. $\varphi \equiv p$: $\mathfrak{M}, w \Vdash \varphi$ iff $w \in V(p)$.
2. $\varphi \equiv \bot$: not $\mathfrak{M}, w \Vdash \varphi$.
3. $\varphi \equiv \neg\psi$: $\mathfrak{M}, w \Vdash \varphi$ iff for no w' such that Rww', $\mathfrak{M}, w' \Vdash \psi$.
4. $\varphi \equiv \psi \wedge \chi$: $\mathfrak{M}, w \Vdash \varphi$ iff $\mathfrak{M}, w \Vdash \psi$ and $\mathfrak{M}, w \Vdash \chi$.
5. $\varphi \equiv \psi \vee \chi$: $\mathfrak{M}, w \Vdash \varphi$ iff $\mathfrak{M}, w \Vdash \psi$ or $\mathfrak{M}, w \Vdash \chi$ (or both).
6. $\varphi \equiv \psi \to \chi$: $\mathfrak{M}, w \Vdash \varphi$ iff for every w' such that Rww', not $\mathfrak{M}, w' \Vdash \psi$ or $\mathfrak{M}, w' \Vdash \chi$ (or both).

We write $\mathfrak{M}, w \nVdash \varphi$ if not $\mathfrak{M}, w \Vdash \varphi$. If Γ is a set of formulas, $\mathfrak{M}, w \Vdash \Gamma$ means $\mathfrak{M}, w \Vdash \psi$ for all $\psi \in \Gamma$.

Proposition 12.3. *Truth at worlds is monotonic with respect to* R*, i.e., if* $\mathfrak{M}, w \Vdash \varphi$ *and* Rww'*, then* $\mathfrak{M}, w' \Vdash \varphi$*.*

Proof. Exercise. □

12.3 Semantic Notions

Definition 12.4. We say φ is *true in the model* $\mathfrak{M} = \langle W, R, V\rangle$, $\mathfrak{M} \Vdash \varphi$, iff $\mathfrak{M}, w \Vdash \varphi$ for all $w \in W$. φ is *valid*, $\vDash \varphi$, iff it is true in all models. We say a set of formulas Γ *entails* φ, $\Gamma \vDash \varphi$, iff for every model $\mathfrak{M}$ and every w such that $\mathfrak{M}, w \Vdash \Gamma$, $\mathfrak{M}, w \Vdash \varphi$.

Proposition 12.5. *1. If* $\mathfrak{M}, w \Vdash \Gamma$ *and* $\Gamma \vDash \varphi$*, then* $\mathfrak{M}, w \Vdash \varphi$.

2. If $\mathfrak{M} \Vdash \Gamma$ *and* $\Gamma \vDash \varphi$*, then* $\mathfrak{M} \Vdash \varphi$.

Proof. 1. Suppose $\mathfrak{M} \Vdash \Gamma$. Since $\Gamma \vDash \varphi$, we know that if $\mathfrak{M}, w \Vdash \Gamma$, then $\mathfrak{M}, w \Vdash \varphi$. Since $\mathfrak{M}, u \Vdash \Gamma$ for all every $u \in W$, $\mathfrak{M}, w \Vdash \Gamma$. Hence $\mathfrak{M}, w \Vdash \varphi$.

2. Follows immediately from (1). □

Problems

Problem 12.1. Show that according to Definition 12.2, $\mathfrak{M}, w \Vdash \neg\varphi$ iff $\mathfrak{M}, w \Vdash \varphi \to \bot$.

Problem 12.2. Prove Proposition 12.3.

Chapter 13

Soundness and Completeness

13.1 Soundness of Natural Deduction

Theorem 13.1 (Soundness). *If* $\Gamma \vdash \varphi$, *then* $\Gamma \vDash \varphi$.

Proof. We prove that if $\Gamma \vdash \varphi$, then $\Gamma \vDash \varphi$. The proof is by induction on the derivation of φ from Γ.

1. If the derivation consists of just the assumption φ, we have $\varphi \vdash \varphi$, and want to show that $\varphi \vDash \varphi$. Consider any model $\mathfrak{M}$ such that $\mathfrak{M} \Vdash \varphi$. Then trivially $\mathfrak{M} \Vdash \varphi$.

2. The derivation ends in $\wedge$I: The derivations of the premises ψ from undischarged assumptions Γ and of χ from undischarged assumptions Δ show that $\Gamma \vdash \psi$ and $\Delta \vdash \chi$. By induction hypothesis we have that $\Gamma \vDash \psi$ and $\Delta \vDash \chi$. We have to show that $\Gamma \cup \Delta \vDash \varphi \wedge \psi$, since the undischarged assumptions of the entire derivation are Γ together with Δ. So suppose $\mathfrak{M} \Vdash \Gamma \cup \Delta$. Then also $\mathfrak{M} \Vdash \Gamma$. Since $\Gamma \vDash \psi$, $\mathfrak{M} \Vdash \psi$. Similarly, $\mathfrak{M} \Vdash \chi$. So $\mathfrak{M} \Vdash \psi \wedge \chi$.

3. The derivation ends in $\wedge$E: The derivation of the premise $\psi \wedge \chi$ from undischarged assumptions Γ shows that $\Gamma \vdash \psi \wedge \chi$. By induction hypothesis, $\Gamma \vDash \psi \wedge \chi$. We have to show that $\Gamma \vDash \psi$. So suppose $\mathfrak{M} \Vdash \Gamma$. Since $\Gamma \vDash \psi \wedge \chi$, $\mathfrak{M} \Vdash \psi \wedge \chi$. Then also $\mathfrak{M} \Vdash \psi$. Similarly if $\wedge$E ends in χ, then $\Gamma \vDash \chi$.

4. The derivation ends in $\vee$I: Suppose the premise is ψ, and the undischarged assumptions of the derivation ending in ψ are Γ. Then we have $\Gamma \vdash \psi$ and by inductive hypothesis, $\Gamma \vDash \psi$. We have to show that $\Gamma \vDash \psi \vee \chi$. Suppose $\mathfrak{M} \Vdash \Gamma$. Since $\Gamma \vDash \psi$, $\mathfrak{M} \Vdash \psi$. But then also $\mathfrak{M} \Vdash \psi \vee \chi$. Similarly, if the premise is χ, we have that $\Gamma \vDash \chi$.

5. The derivation ends in $\vee$E: The derivations ending in the premises are of $\psi \vee \chi$ from undischarged assumptions Γ, of θ from undischarged assumptions $\Delta_1 \cup \{\psi\}$, and of θ from undischarged assumptions $\Delta_2 \cup \{\chi\}$. So we have $\Gamma \vdash \psi \vee \chi$, $\Delta_1 \cup \{\psi\} \vdash \theta$, and $\Delta_2 \cup \{\chi\} \vdash \theta$. By induction hypothesis, $\Gamma \vDash \psi \vee \chi$, $\Delta_1 \cup \{\psi\} \vDash \theta$, and $\Delta_2 \cup \{\chi\} \vDash \theta$. We have to prove that $\Gamma \cup \Delta_1 \cup \Delta_2 \vDash \theta$.

   Suppose $\mathfrak{M} \Vdash \Gamma \cup \Delta_1 \cup \Delta_2$. Then $\mathfrak{M} \Vdash \Gamma$ and since $\Gamma \vDash \psi \vee \chi$, $\mathfrak{M} \Vdash \psi \vee \chi$. By definition of $\mathfrak{M} \Vdash$, either $\mathfrak{M} \Vdash \psi$ or $\mathfrak{M} \Vdash \chi$. So we distinguish cases: (a) $\mathfrak{M} \Vdash \psi$. Then $\mathfrak{M} \Vdash \Delta_1 \cup \{\psi\}$. Since $\Delta_1 \cup \psi \vDash \theta$, we have $\mathfrak{M} \Vdash \theta$. (b) $\mathfrak{M} \Vdash \chi$. Then

$\mathfrak{M} \Vdash \Delta_2 \cup \{\chi\}$. Since $\Delta_2 \cup \chi \vDash \theta$, we have $\mathfrak{M} \Vdash \theta$. So in either case, $\mathfrak{M} \Vdash \theta$, as we wanted to show.

6. The derivation ends with $\to$I concluding $\psi \to \chi$. Then the premise is χ, and the derivation ending in the premise has undischarged assumptions $\Gamma \cup \{\psi\}$. So we have that $\Gamma \cup \{\psi\} \vdash \chi$, and by induction hypothesis that $\Gamma \cup \{\psi\} \vDash \chi$. We have to show that $\Gamma \vDash \psi \to \chi$.

   Suppose $\mathfrak{M}, w \Vdash \Gamma$. We want to show that for all w' such that Rww', if $\mathfrak{M}, w' \Vdash \psi$, then $\mathfrak{M}, w' \Vdash \chi$. So assume that Rww' and $\mathfrak{M}, w' \Vdash \psi$. By Proposition 12.3, $\mathfrak{M}, w' \Vdash \Gamma$. Since $\Gamma \cup \{\psi\} \vDash \chi$, $\mathfrak{M}, w' \Vdash \chi$, which is what we wanted to show.

7. The derivation ends in $\to$E and conclusion χ. The premises are $\psi \to \chi$ and ψ, with derivations from undischarged assumptions Γ, Δ. So we have $\Gamma \vdash \psi \to \chi$ and $\Delta \vdash \psi$. By inductive hypothesis, $\Gamma \vDash \psi \to \chi$ and $\Delta \vDash \psi$. We have to show that $\Gamma \cup \Delta \vDash \chi$.

   Suppose $\mathfrak{M}, w \Vdash \Gamma \cup \Delta$. Since $\mathfrak{M}, w \Vdash \Gamma$ and $\Gamma \vDash \psi \to \chi$, $\mathfrak{M}, w \Vdash \psi \to \chi$. By definition, this means that for all w' such that Rww', if $\mathfrak{M}, w' \Vdash \psi$ then $\mathfrak{M}, w' \Vdash \chi$. Since R is reflexive, w is among the w' such that Rww', i.e., we have that if $\mathfrak{M}, w \Vdash \psi$ then $\mathfrak{M}, w \Vdash \chi$. Since $\mathfrak{M}, w \Vdash \Delta$ and $\Delta \vDash \psi$, $\mathfrak{M}, w \Vdash \psi$. So, $\mathfrak{M}, w \Vdash \chi$, as we wanted to show.

8. The derivation ends in $\bot$E, concluding φ. The premise is $\bot$ and the undischarged assumptions of the derivation of the premise are Γ. Then $\Gamma \vdash \bot$. By inductive hypothesis, $\Gamma \vDash \bot$. We have to show $\Gamma \vDash \varphi$.

   We proceed indirectly. If $\Gamma \nvDash \varphi$ there is a model $\mathfrak{M}$ and world w such that $\mathfrak{M}, w \Vdash \Gamma$ and $\mathfrak{M}, w \nVdash \varphi$. Since $\Gamma \vDash \bot$, $\mathfrak{M}, w \Vdash \bot$. But that's impossible, since by definition, $\mathfrak{M}, w \nVdash \bot$. So $\Gamma \vDash \varphi$.

9. The derivation ends in $\neg$I: Exercise.

10. The derivation ends in $\neg$E: Exercise. □

13.2 Lindenbaum's Lemma

Definition 13.2. A set of formulas Γ is *prime* iff

1. Γ is consistent.
2. If $\Gamma \vdash \varphi$ then $\varphi \in \Gamma$, and
3. If $\varphi \vee \psi \in \Gamma$ then $\varphi \in \Gamma$ or $\psi \in \Gamma$.

Lemma 13.3 (Lindenbaum's Lemma). *If $\Gamma \nvDash \varphi$, there is a $\Gamma^* \supseteq \Gamma$ such that Γ^* is prime and $\Gamma^* \nvDash \varphi$.*

Proof. Let $\psi_1 \vee \chi_1, \psi_2 \vee \chi_2, \ldots$, be an enumeration of all formulas of the form $\psi \vee \chi$. We'll define an increasing sequence of sets of formulas Γ_n, where each Γ_{n+1} is defined as Γ_n together with one new formula. Γ^* will be the union of all Γ_n. The new formulas are selected so as to ensure that Γ^* is prime and still $\Gamma^* \nvDash \varphi$. This means that at each step we should find the first disjunction $\psi_i \vee \chi_i$ such that:

1. $\Gamma_n \vdash \psi_i \vee \chi_i$
2. $\psi_i \notin \Gamma_n$ and $\chi_i \notin \Gamma_n$

We add to Γ_n either ψ_i if $\Gamma_n \cup \{\psi_i\} \nvdash \varphi$, or χ_i otherwise. We'll have to show that this works. For now, let's define $i(n)$ as the least i such that (1) and (2) hold.

Define $\Gamma_0 = \Gamma$ and

$$\Gamma_{n+1} = \begin{cases} \Gamma_n \cup \{\psi_{i(n)}\} & \text{if } \Gamma_n \cup \{\psi_{i(n)}\} \nvdash \varphi \\ \Gamma_n \cup \{\chi_{i(n)}\} & \text{otherwise} \end{cases}$$

If $i(n)$ is undefined, i.e., whenever $\Gamma_n \vdash \psi \vee \chi$, either $\psi \in \Gamma_n$ or $\chi \in \Gamma_n$, we let $\Gamma_{n+1} = \Gamma_n$. Now let $\Gamma^* = \bigcup_{n=0}^{\infty} \Gamma_n$

First we show that for all n, $\Gamma_n \nvdash \varphi$. We proceed by induction on n. For $n = 0$ the claim holds by the hypothesis of the theorem, i.e., $\Gamma \nvdash \varphi$. If $n > 0$, we have to show that if $\Gamma_n \nvdash \varphi$ then $\Gamma_{n+1} \nvdash \varphi$. If $i(n)$ is undefined, $\Gamma_{n+1} = \Gamma_n$ and there is nothing to prove. So suppose $i(n)$ is defined. For simplicity, let $i = i(n)$.

We'll prove the contrapositive of the claim. Suppose $\Gamma_{n+1} \vdash \varphi$. By construction, $\Gamma_{n+1} = \Gamma_n \cup \{\psi_i\}$ if $\Gamma_n \cup \{\psi_i\} \nvdash \varphi$, or else $\Gamma_{n+1} = \Gamma_n \cup \{\chi_i\}$. It clearly can't be the first, since then $\Gamma_{n+1} \nvdash \varphi$. Hence, $\Gamma_n \cup \{\psi_i\} \vdash \varphi$ and $\Gamma_{n+1} = \Gamma_n \cup \{\chi_i\}$. By definition of $i(n)$, we have that $\Gamma_n \vdash \psi_i \vee \chi_i$. We have $\Gamma_n \cup \{\psi_i\} \vdash \varphi$. We also have $\Gamma_{n+1} = \Gamma_n \cup \{\chi_i\} \vdash \varphi$. Hence, $\Gamma_n \vdash \varphi$, which is what we wanted to show.

If $\Gamma^* \vdash \varphi$, there would be some finite subset $\Gamma' \subseteq \Gamma^*$ such that $\Gamma' \vdash \varphi$. Each $\theta \in \Gamma'$ must be in Γ_i for some i. Let n be the largest of these. Since $\Gamma_i \subseteq \Gamma_n$ if $i \leq n$, $\Gamma' \subseteq \Gamma_n$. But then $\Gamma_n \vdash \varphi$, contrary to our proof above that $\Gamma_n \nvdash \varphi$.

Lastly, we show that Γ^* is prime, i.e., satisfies conditions (1), (2), and (3) of Definition 13.2.

First, $\Gamma^* \nvdash \varphi$, so Γ^* is consistent, so (1) holds.

We now show that if $\Gamma^* \vdash \psi \vee \chi$, then either $\psi \in \Gamma^*$ or $\chi \in \Gamma^*$. This proves (3), since if $\psi \in \Gamma^*$ then also $\Gamma^* \vdash \psi$, and similarly for χ. So assume $\Gamma^* \vdash \psi \vee \chi$ but $\psi \notin \Gamma^*$ and $\chi \notin \Gamma^*$. Since $\Gamma^* \vdash \psi \vee \chi$, $\Gamma_n \vdash \psi \vee \chi$ for some n. $\psi \vee \chi$ appears on the enumeration of all disjunctions, say as $\psi_j \vee \chi_j$. $\psi_j \vee \chi_j$ satisfies the properties in the definition of $i(n)$, namely we have $\Gamma_n \vdash \psi_j \vee \chi_j$, while $\psi_j \notin \Gamma_n$ and $\chi_j \notin \Gamma_n$. At each stage, at least one fewer disjunction $\psi_i \vee \chi_i$ satisfies the conditions (since at each stage we add either ψ_i or χ_i), so at some stage m we will have $j = i(\Gamma_m)$. But then either $\psi \in \Gamma_{m+1}$ or $\chi \in \Gamma_{m+1}$, contrary to the assumption that $\psi \notin \Gamma^*$ and $\chi \notin \Gamma^*$.

Now suppose $\Gamma^* \vdash \varphi$. Then $\Gamma^* \vdash \varphi \vee \varphi$. But we've just proved that if $\Gamma^* \vdash \varphi \vee \varphi$ then $\varphi \in \Gamma^*$. Hence, Γ^* satisfies (2) of Definition 13.2. □

13.3 The Canonical Model

The worlds in our model will be finite sequences σ of natural numbers, i.e., $\sigma \in \mathbb{N}^*$. Note that $\mathbb{N}^*$ is inductively defined by:

1. $\Lambda \in \mathbb{N}^*$.
2. If $\sigma \in \mathbb{N}^*$ and $n \in \mathbb{N}$, then $\sigma.n \in \mathbb{N}^*$ (where $\sigma.n$ is $\sigma \frown \langle n \rangle$ and $\sigma \frown \sigma'$ is the concatenation if σ and σ').
3. Nothing else is in $\mathbb{N}^*$.

So we can use $\mathbb{N}^*$ to give inductive definitions.

Let $\langle \psi_1, \chi_1 \rangle$, $\langle \psi_2, \chi_s \rangle$, ..., be an enumeration of all pairs of formulas. Given a set of formulas Δ, define $\Delta(\sigma)$ by induction as follows:

1. $\Delta(\Lambda) = \Delta$
2. $\Delta(\sigma.n) =$
$$\begin{cases} (\Delta(\sigma) \cup \{\psi_n\})^* & \text{if } \Delta(\sigma) \cup \{\psi_n\} \nvdash \chi_n \\ \Delta(\sigma) & \text{otherwise} \end{cases}$$

Here by $(\Delta(\sigma) \cup \{\psi_n\})^*$ we mean the prime set of formulas which exists by Lemma 13.3 applied to the set $\Delta(\sigma) \cup \{\psi_n\}$ and the formula χ_n. Note that by this definition, if $\Delta(\sigma) \cup \{\psi_n\} \nvdash \chi_n$, then $\Delta(\sigma.n) \vdash \psi_n$ and $\Delta(\sigma.n) \nvdash \chi_n$. Note also that $\Delta(\sigma) \subseteq \Delta(\sigma.n)$ for any n. If Δ is prime, then $\Delta(\sigma)$ is prime for all σ.

Definition 13.4. Suppose Δ is prime. Then the *canonical model* $\mathfrak{M}(\Delta)$ for Δ is defined by:

1. $W = \mathbb{N}^*$, the set of finite sequences of natural numbers.
2. R is the partial order according to which $R\sigma\sigma'$ iff σ is an initial segment of σ' (i.e., $\sigma' = \sigma \frown \sigma''$ for some sequence σ'').
3. $V(p) = \{\sigma \mid p \in \Delta(\sigma)\}$.

It is easy to verify that R is indeed a partial order. Also, the monotonicity condition on V is satisfied. Since $\Delta(\sigma) \subseteq \Delta(\sigma.n)$ we get $\Delta(\sigma) \subseteq \Delta(\sigma')$ whenever $R\sigma\sigma'$ by induction on σ.

13.4 The Truth Lemma

Lemma 13.5. *If Δ is prime, then $\mathfrak{M}(\Delta), \sigma \Vdash \varphi$ iff $\Delta(\sigma) \vdash \varphi$.*

Proof. By induction on φ.

1. $\varphi \equiv \bot$: Since $\Delta(\sigma)$ is prime, it is consistent, so $\Delta(\sigma) \nvdash \varphi$. By definition, $\mathfrak{M}(\Delta), \sigma \nVdash \varphi$.
2. $\varphi \equiv p$: By definition of $\Vdash$, $\mathfrak{M}(\Delta), \sigma \Vdash \varphi$ iff $\sigma \in V(p)$, i.e., $\Delta(\sigma) \vdash \varphi$.
3. $\varphi \equiv \neg\psi$: exercise.
4. $\varphi \equiv \psi \wedge \chi$: $\mathfrak{M}(\Delta), \sigma \Vdash \varphi$ iff $\mathfrak{M}(\Delta), \sigma \Vdash \psi$ and $\mathfrak{M}(\Delta), \sigma \Vdash \chi$. By induction hypothesis, $\mathfrak{M}(\Delta), \sigma \Vdash \psi$ iff $\Delta(\sigma) \vdash \psi$, and similarly for χ. But $\Delta(\sigma) \vdash \psi$ and $\Delta(\sigma) \vdash \chi$ iff $\Delta(\sigma) \vdash \varphi$.
5. $\varphi \equiv \psi \vee \chi$: $\mathfrak{M}(\Delta), \sigma \Vdash \varphi$ iff $\mathfrak{M}(\Delta), \sigma \Vdash \psi$ or $\mathfrak{M}(\Delta), \sigma \Vdash \chi$. By induction hypothesis, this holds iff $\Delta(\sigma) \vdash \psi$ of $\Delta(\sigma) \vdash \chi$. We have to show that this in turn holds iff $\Delta(\sigma) \vdash \varphi$. The left-to-right direction is clear. The right-to-left direction follows since $\Delta(\sigma)$ is prime.

6. $\varphi \equiv \psi \to \chi$: First the contrapositive of the left-to-right direction: Assume $\Delta(\sigma) \nvdash \psi \to \chi$. Then also $\Delta(\sigma) \cup \{\psi\} \nvdash \chi$. Since $\langle \psi, \chi \rangle$ is $\langle \psi_n, \chi_n \rangle$ for some n, we have $\Delta(\sigma.n) = (\Delta(\sigma) \cup \{\psi\})^*$, and $\Delta(\sigma.n) \vdash \psi$ but $\Delta(\sigma.n) \nvdash \chi$. By inductive hypothesis, $\mathfrak{M}(\Delta), \sigma.n \Vdash \psi$ and $\mathfrak{M}(\Delta), \sigma.n \nVdash \chi$. Since $R\sigma(\sigma.n)$, this means that $\mathfrak{M}(\Delta), \sigma \nVdash \varphi$.

   Now assume $\Delta(\sigma) \vdash \psi \to \chi$, and let $R\sigma\sigma'$. Since $\Delta(\sigma) \subseteq \Delta(\sigma')$, we have: if $\Delta(\sigma') \vdash \psi$, then $\Delta(\sigma') \vdash \chi$. In other words, for every σ' such that $R\sigma\sigma'$, either $\Delta(\sigma') \nvdash \psi$ or $\Delta(\sigma') \vdash \chi$. By induction hypothesis, this means that whenever $R\sigma\sigma'$, either $\mathfrak{M}(\Delta), \sigma' \nVdash \psi$ or $\mathfrak{M}(\Delta), \sigma' \Vdash \chi$, i.e., $\mathfrak{M}(\Delta), \sigma \Vdash \varphi$. □

13.5 The Completeness Theorem

Theorem 13.6. *If* $\Gamma \vDash \varphi$ *then* $\Gamma \vdash \varphi$.

Proof. We prove the contrapositive: Suppose $\Gamma \nvdash \varphi$. Then by Lemma 13.3, there is a prime set $\Gamma^* \supseteq \Gamma$ such that $\Gamma^* \nvdash \varphi$. Consider the canonical model $\mathfrak{M}(\Gamma^*)$ for Γ^* as defined in Definition 13.4. For any $\psi \in \Gamma$, $\Gamma^* \vdash \psi$. Note that $\Gamma^*(\Lambda) = \Gamma^*$. By the Truth Lemma (Lemma 13.5), we have $\mathfrak{M}(\Gamma^*), \Lambda \Vdash \psi$ for all $\psi \in \Gamma$ and $\mathfrak{M}(\Gamma^*), \Lambda \nVdash \varphi$. This shows that $\Gamma \nvDash \varphi$. □

Problems

Problem 13.1. Complete the proof of Theorem 13.1. For the cases for ¬I and ¬E, use the definition of $\mathfrak{M}, w \Vdash \neg\varphi$ in Definition 12.2, i.e., don't treat $\neg\varphi$ as defined by $\varphi \to \bot$.

Part V

Computability and Incompleteness

Chapter 14

Turing Machine Computations

14.1 Introduction

What does it mean for a function, say, from $\mathbb{N}$ to $\mathbb{N}$ to be *computable*? Among the first answers, and the most well known one, is that a function is computable if it can be computed by a Turing machine. This notion was set out by Alan Turing in 1936. Turing machines are an example of *a model of computation*—they are a mathematically precise way of defining the idea of a "computational procedure." What exactly that means is debated, but it is widely agreed that Turing machines are one way of specifying computational procedures. Even though the term "Turing machine" evokes the image of a physical machine with moving parts, strictly speaking a Turing machine is a purely mathematical construct, and as such it idealizes the idea of a computational procedure. For instance, we place no restriction on either the time or memory requirements of a Turing machine: Turing machines can compute something even if the computation would require more storage space or more steps than there are atoms in the universe.

It is perhaps best to think of a Turing machine as a program for a special kind of imaginary mechanism. This mechanism consists of a *tape* and a *read-write head*. In our version of Turing machines, the tape is infinite in one direction (to the right), and it is divided into *squares*, each of which may contain a symbol from a finite *alphabet*. Such alphabets can contain any number of different symbols, say, but we will mainly make do with three: ▷, 0, and 1. When the mechanism is started, the tape is empty (i.e., each square contains the symbol 0) except for the leftmost square, which contains ▷, and a finite number of squares which contain the *input*. At any time, the mechanism is in one of a finite number of *states*. At the outset, the head scans the leftmost square and in a specified *initial state*. At each step of the mechanism's run, the content of the square currently scanned together with the state the mechanism is in and the Turing machine program determine what happens next. The Turing machine program is given by a partial function which takes as input a state q and a symbol σ and outputs a triple $\langle q', \sigma', D\rangle$. Whenever the mechanism is in state q and reads symbol σ, it

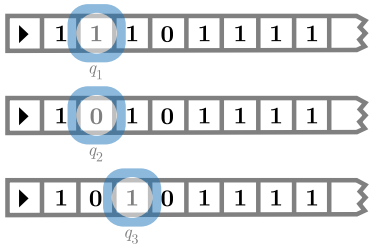


Figure 14.1: A Turing machine executing its program.

replaces the symbol on the current square with σ', the head moves left, right, or stays put according to whether D is L, R, or N, and the mechanism goes into state q'.

For instance, consider the situation in Figure 14.1. The visible part of the tape of the Turing machine contains the end-of-tape symbol ▷ on the leftmost square, followed by three 1's, a 0, and four more 1's. The head is reading the third square from the left, which contains a 1, and is in state q_1—we say "the machine is reading a 1 in state q_1." If the program of the Turing machine returns, for input $\langle q_1, 1\rangle$, the triple $\langle q_2, 0, N\rangle$, the machine would now replace the 1 on the third square with a 0, leave the read/write head where it is, and switch to state q_2. If then the program returns $\langle q_3, 0, R\rangle$ for input $\langle q_2, 0\rangle$, the machine would now overwrite the 0 with another 0 (effectively, leaving the content of the tape under the read/write head unchanged), move one square to the right, and enter state q_3. And so on.

We say that the machine *halts* when it encounters some state, q_n, and symbol, σ such that there is no instruction for $\langle q_n, \sigma\rangle$, i.e., the transition function for input $\langle q_n, \sigma\rangle$ is undefined. In other words, the machine has no instruction to carry out, and at that point, it ceases operation. Halting is sometimes represented by a specific halt state h. This will be demonstrated in more detail later on.

The beauty of Turing's paper, "On computable numbers," is that he presents not only a formal definition, but also an argument that the definition captures the intuitive notion of computability. From the definition, it should be clear that any function computable by a Turing machine is computable in the intuitive sense. Turing offers three types of argument that the converse is true, i.e., that any function that we would naturally regard as computable is computable by such a machine. They are (in Turing's words):

1. A direct appeal to intuition.
2. A proof of the equivalence of two definitions (in case the new definition has a greater intuitive appeal).
3. Giving examples of large classes of numbers which are computable.

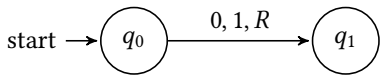
Our goal is to try to define the notion of computability "in principle," i.e., without taking into account practical limitations of time and space. Of course, with the broadest definition of computability in place, one can then go on to consider computation with bounded resources; this forms the heart of the subject known as "computational complexity."

Historical Remarks Alan Turing invented Turing machines in 1936. While his interest at the time was the decidability of first-order logic, the paper has been de-

scribed as a definitive paper on the foundations of computer design. In the paper, Turing focuses on computable real numbers, i.e., real numbers whose decimal expansions are computable; but he notes that it is not hard to adapt his notions to computable functions on the natural numbers, and so on. Notice that this was a full five years before the first working general purpose computer was built in 1941 (by the German Konrad Zuse in his parent's living room), seven years before Turing and his colleagues at Bletchley Park built the code-breaking Colossus (1943), nine years before the American ENIAC (1945), twelve years before the first British general purpose computer—the Manchester Small-Scale Experimental Machine—was built in Manchester (1948), and thirteen years before the Americans first tested the BINAC (1949). The Manchester SSEM has the distinction of being the first stored-program computer—previous machines had to be rewired by hand for each new task.

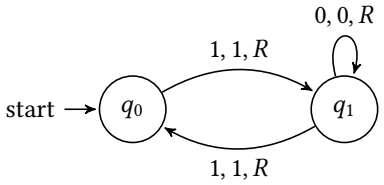
14.2 Representing Turing Machines

Turing machines can be represented visually by *state diagrams*. The diagrams are composed of state cells connected by arrows. Unsurprisingly, each state cell represents a state of the machine. Each arrow represents an instruction that can be carried out from that state, with the specifics of the instruction written above or below the appropriate arrow. Consider the following machine, which has only two internal states, q_0 and q_1, and one instruction:



Recall that the Turing machine has a read/write head and a tape with the input written on it. The instruction can be read as *if reading a* 0 *in state* q_0*, write a* 1*, move right, and move to state* q_1. This is equivalent to the transition function mapping $\langle q_0, 0\rangle$ to $\langle q_1, 1, R\rangle$.

Example 14.1. *Even Machine*: The following Turing machine halts if, and only if, there are an even number of 1's on the tape (under the assumption that all 1's come before the first 0 on the tape).



The state diagram corresponds to the following transition function:

$$\delta(q_0, 1) = \langle q_1, 1, R\rangle,$$
$$\delta(q_1, 1) = \langle q_0, 1, R\rangle,$$
$$\delta(q_1, 0) = \langle q_1, 0, R\rangle$$

The above machine halts only when the input is an even number of strokes. Otherwise, the machine (theoretically) continues to operate indefinitely. For any machine and input, it is possible to trace through the *configurations* of the machine in

order to determine the output. We will give a formal definition of configurations later. For now, we can intuitively think of configurations as a series of diagrams showing the state of the machine at any point in time during operation. Configurations show the content of the tape, the state of the machine and the location of the read/write head.

Let us trace through the configurations of the even machine if it is started with an input of four 1's. In this case, we expect that the machine will halt. We will then run the machine on an input of three 1's, where the machine will run forever.

The machine starts in state q_0, scanning the leftmost 1. We can represent the initial state of the machine as follows:

$$\triangleright 1_0 1110 \ldots$$

The above configuration is straightforward. As can be seen, the machine starts in state one, scanning the leftmost 1. This is represented by a subscript of the state name on the first 1. The applicable instruction at this point is $\delta(q_0, 1) = \langle q_1, 1, R \rangle$, and so the machine moves right on the tape and changes to state q_1.

$$\triangleright 11_1 110 \ldots$$

Since the machine is now in state q_1 scanning a 1, we have to "follow" the instruction $\delta(q_1, 1) = \langle q_0, 1, R \rangle$. This results in the configuration

$$\triangleright 111_0 10 \ldots$$

As the machine continues, the rules are applied again in the same order, resulting in the following two configurations:

$$\triangleright 1111_1 0 \ldots$$

$$\triangleright 11110_0 \ldots$$

The machine is now in state q_0 scanning a 0. Based on the transition diagram, we can easily see that there is no instruction to be carried out, and thus the machine has halted. This means that the input has been accepted.

Suppose next we start the machine with an input of three 1's. The first few configurations are similar, as the same instructions are carried out, with only a small difference of the tape input:

$$\triangleright 1_0 110 \ldots$$

$$\triangleright 11_1 10 \ldots$$

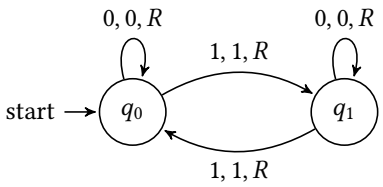
$$\triangleright 111_0 0 \ldots$$

$$\triangleright 1110_1 \ldots$$

The machine has now traversed past all the 1's, and is reading a 0 in state q_1. As shown in the diagram, there is an instruction of the form $\delta(q_1, 0) = \langle q_1, 0, R \rangle$. Since the tape is filled with 0 indefinitely to the right, the machine will continue to execute this instruction *forever*, staying in state q_1 and moving ever further to the right. The machine will never halt, and does not accept the input.

It is important to note that not all machines will halt. If halting means that the machine runs out of instructions to execute, then we can create a machine that never halts simply by ensuring that there is an outgoing arrow for each symbol at each state. The even machine can be modified to run indefinitely by adding an instruction for scanning a 0 at q_0.

Example 14.2.



Machine tables are another way of representing Turing machines. Machine tables have the tape alphabet displayed on the x-axis, and the set of machine states across the y-axis. Inside the table, at the intersection of each state and symbol, is written the rest of the instruction—the new state, new symbol, and direction of movement. Machine tables make it easy to determine in what state, and for what symbol, the machine halts. Whenever there is a gap in the table is a possible point for the machine to halt. Unlike state diagrams and instruction sets, where the points at which the machine halts are not always immediately obvious, any halting points are quickly identified by finding the gaps in the machine table.

Example 14.3. The machine table for the even machine is:

	0	1
q_0		$1, q_1, R$
q_1	$0, q_1, 0$	$1, q_0, R$

As we can see, the machine halts when scanning a blank in state q_0.

So far we have only considered machines that read and accept input. However, Turing machines have the capacity to both read and write. An example of such a machine (although there are many, many examples) is a *doubler*. A doubler, when started with a block of n 1's on the tape, outputs a block of $2n$ 1's.

Example 14.4. Before building a doubler machine, it is important to come up with a *strategy* for solving the problem. Since the machine (as we have formulated it) cannot remember how many 1's it has read, we need to come up with a way to keep track of all the 1's on the tape. One such way is to separate the output from the input with a 0. The machine can then erase the first 1 from the input, traverse over the rest of the input, leave a 0, and write two new 1's. The machine will then go back and find the second 1 in the input, and double that one as well. For each one 1 of input, it will write two 1's of output. By erasing the input as the machine goes, we can guarantee that no 1 is missed or doubled twice. When the entire input is erased, there will be $2n$ 1's left on the tape. The state diagram of the resulting Turing machine is depicted in Figure 14.2.

14.3 Turing Machines

The formal definition of what constitutes a Turing machine looks abstract, but is actually simple: it merely packs into one mathematical structure all the information needed to specify the workings of a Turing machine. This includes (1) which states the machine can be in, (2) which symbols are allowed to be on the tape, (3) which state the machine should start in, and (4) what the instruction set of the machine is.

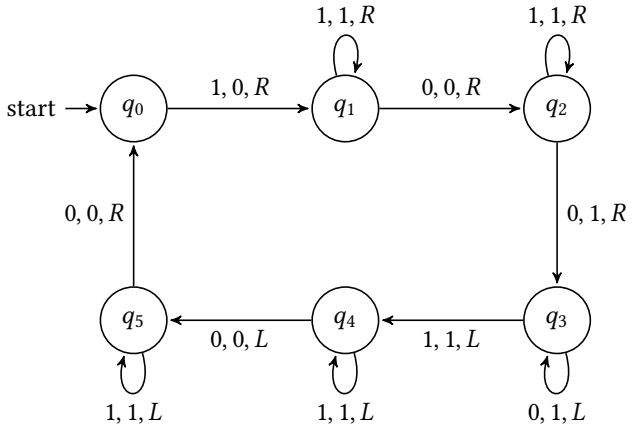


Figure 14.2: A doubler machine

Definition 14.5 (Turing machine). A *Turing machine* M is a tuple $\langle Q, \Sigma, q_0, \delta \rangle$ consisting of

1. a finite set of *states* Q,
2. a finite *alphabet* Σ which includes $\triangleright$ and 0,
3. an *initial state* $q_0 \in Q$,
4. a finite *instruction set* $\delta \colon Q \times \Sigma \nrightarrow Q \times \Sigma \times \{L, R, N\}$.

The partial function δ is also called the *transition function* of M.

We assume that the tape is infinite in one direction only. For this reason it is useful to designate a special symbol $\triangleright$ as a marker for the left end of the tape. This makes it easier for Turing machine programs to tell when they're "in danger" of running off the tape.

Example 14.6. *Even Machine:* The even machine is formally the quadruple $\langle Q, \Sigma, q_0, \delta \rangle$ where

$$\begin{aligned} Q &= \{q_0, q_1\} \\ \Sigma &= \{\triangleright, 0, 1\}, \\ \delta(q_0, 1) &= \langle q_1, 1, R \rangle, \\ \delta(q_1, 1) &= \langle q_0, 1, R \rangle, \\ \delta(q_1, 0) &= \langle q_1, 0, R \rangle. \end{aligned}$$

14.4 Configurations and Computations

Recall tracing through the configurations of the even machine earlier. The imaginary mechanism consisting of tape, read/write head, and Turing machine program is really just an intuitive way of visualizing what a Turing machine computation is. Formally,

we can define the computation of a Turing machine on a given input as a sequence of *configurations*—and a configuration in turn is a sequence of symbols (corresponding to the contents of the tape at a given point in the computation), a number indicating the position of the read/write head, and a state. Using these, we can define what the Turing machine M computes on a given input.

Definition 14.7 (Configuration). A *configuration* of Turing machine $M = \langle Q, \Sigma, q_0, \delta\rangle$ is a triple $\langle C, m, q\rangle$ where

1. $C \in \Sigma^*$ is a finite sequence of symbols from Σ,
2. $m \in \mathbb{N}$ is a number $< \mathrm{len}(C)$, and
3. $q \in Q$

Intuitively, the sequence C is the content of the tape (symbols of all squares from the leftmost square to the last non-blank or previously visited square), m is the number of the square the read/write head is scanning (beginning with 0 being the number of the leftmost square), and q is the current state of the machine.

The potential input for a Turing machine is a sequence of symbols, usually a sequence that encodes a number in some form. The initial configuration of the Turing machine is that configuration in which we start the Turing machine to work on that input: the tape contains the tape end marker immediately followed by the input written on the squares to the right, the read/write head is scanning the leftmost square of the input (i.e., the square to the right of the left end marker), and the mechanism is in the designated start state q_0.

Definition 14.8 (Initial configuration). The *initial configuration* of M for input $I \in \Sigma^*$ is

$$\langle \triangleright \frown I, 1, q_0\rangle.$$

The $\frown$ symbol is for *concatenation*—we want to ensure that there are no blanks between the left end marker and the beginning of the input.

Definition 14.9. We say that a configuration $\langle C, m, q\rangle$ *yields the configuration* $\langle C', m', q'\rangle$ *in one step* (according to M), iff

1. the m-th symbol of C is σ,
2. the instruction set of M specifies $\delta(q, \sigma) = \langle q', \sigma', D\rangle$,
3. the m-th symbol of C' is σ', and
4. a) $D = L$ and $m' = m - 1$ if $m > 0$, otherwise $m' = 0$, or
   b) $D = R$ and $m' = m + 1$, or
   c) $D = N$ and $m' = m$,
5. if $m' = \mathrm{len}(C)$, then $\mathrm{len}(C') = \mathrm{len}(C) + 1$ and the m'-th symbol of C' is 0.
6. for all i such that $i < \mathrm{len}(C')$ and $i \neq m$, $C'(i) = C(i)$,

Definition 14.10. A *run of M on input I* is a sequence C_i of configurations of M, where C_0 is the initial configuration of M for input I, and each C_i yields C_{i+1} in one step.

We say that M *halts on input I after k steps* if $C_k = \langle C, m, q\rangle$, the mth symbol of C is σ, and $\delta(q, \sigma)$ is undefined. In that case, the *output* of M for input I is O, where O is a string of symbols not beginning or ending in 0 such that $C = \triangleright \frown 0^i \frown O \frown 0^j$ for some $i, j \in \mathbb{N}$.

According to this definition, the output O of M always begins and ends in a symbol other than 0, or, if at time k the entire tape is filled with 0 (except for the leftmost $\triangleright$), O is the empty string.

14.5 Unary Representation of Numbers

Turing machines work on sequences of symbols written on their tape. Depending on the alphabet a Turing machine uses, these sequences of symbols can represent various inputs and outputs. Of particular interest, of course, are Turing machines which compute *arithmetical* functions, i.e., functions of natural numbers. A simple way to represent positive integers is by coding them as sequences of a single symbol 1. If $n \in \mathbb{N}$, let 1^n be the empty sequence if $n = 0$, and otherwise the sequence consisting of exactly n 1's.

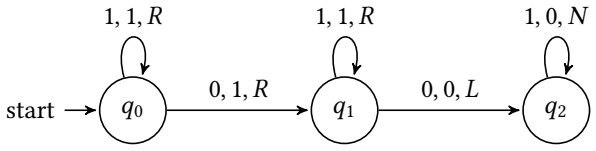
Definition 14.11 (Computation). A Turing machine M *computes* the function $f\colon \mathbb{N}^n \to \mathbb{N}$ iff M halts on input

$$1^{k_1}01^{k_2}0\ldots01^{k_n}$$

with output $1^{f(k_1,\ldots,k_n)}$.

Example 14.12. *Addition:* Build a machine that, when given an input of two non-empty strings of 1's of length n and m, computes the function $f(n, m) = n + m$.

We want to come up with a machine that starts with two blocks of strokes on the tape and halts with one block of strokes. We first need a method to carry out. The input strokes are separated by a blank, so one method would be to write a stroke on the square containing the blank, and erase the first (or last) stroke. This would result in a block of $n + m$ 1's. Alternatively, we could proceed in a similar way to the doubler machine, by erasing a stroke from the first block, and adding one to the second block of strokes until the first block has been removed completely. We will proceed with the former example.

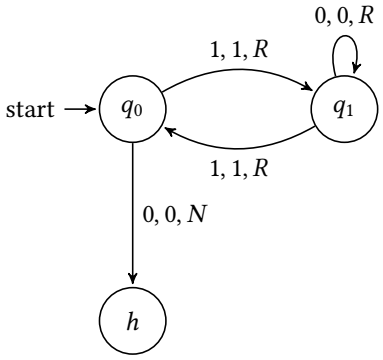


14.6 Halting States

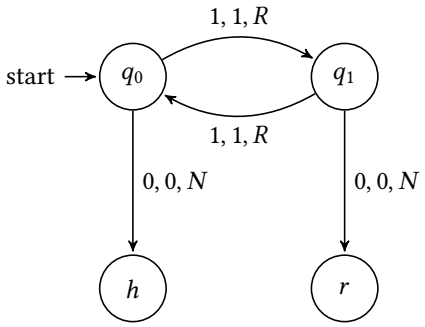
Although we have defined our machines to halt only when there is no instruction to carry out, common representations of Turing machines have a dedicated *halting state*, h, such that $h \in Q$.

The idea behind a halting state is simple: when the machine has finished operation (it is ready to accept input, or has finished writing the output), it goes into a state h where it halts. Some machines have two halting states, one that accepts input and one that rejects input.

Example 14.13. *Halting States.* To elucidate this concept, let us begin with an alteration of the even machine. Instead of having the machine halt in state q_0 if the input is even, we can add an instruction to send the machine into a halt state.



Let us further expand the example. When the machine determines that the input is odd, it never halts. We can alter the machine to include a *reject* state by replacing the looping instruction with an instruction to go to a reject state r.



Adding a dedicated halting state can be advantageous in cases like this, where it makes explicit when the machine accepts/rejects certain inputs. However, it is important to note that no computing power is gained by adding a dedicated halting state. Similarly, a less formal notion of halting has its own advantages. The definition of halting used so far in this chapter makes the proof of the *Halting Problem* intuitive and easy to demonstrate. For this reason, we continue with our original definition.

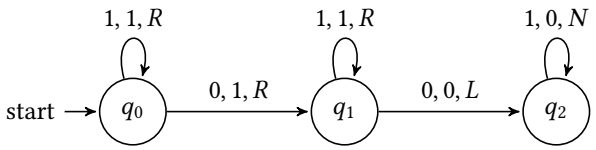
14.7 Combining Turing Machines

The examples of Turing machines we have seen so far have been fairly simple in nature. But in fact, any problem that can be solved with any modern programming language can als o be solved with Turing machines. To build more complex Turing machines, it is important to convince ourselves that we can combine them, so we can build machines to solve more complex problems by breaking the procedure into

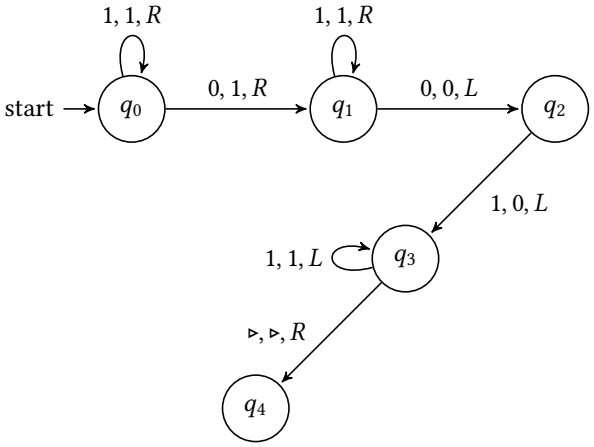
simpler parts. If we can find a natural way to break a complex problem down into constituent parts, we can tackle the problem in several stages, creating several simple Turing machines and combining then into one machine that can solve the problem. This point is especially important when tackling the Halting Problem in the next section.

Example 14.14. *Combining Machines:* Design a machine that computes the function $f(m, n) = 2(m + n)$.

In order to build this machine, we can combine two machines we are already familiar with: the addition machine, and the doubler. We begin by drawing a state diagram for the addition machine.



Instead of halting at state q_2, we want to continue operation in order to double the output. Recall that the doubler machine erases the first stroke in the input and writes two strokes in a separate output. Let's add an instruction to make sure the tape head is reading the first stroke of the output of the addition machine.



It is now easy to double the input—all we have to do is connect the doubler machine onto state q_4. This requires renaming the states of the doubler machine so that they start at q_4 instead of q_0—this way we don't end up with two starting states. The final diagram should look as in Figure 14.3.

14.8 Variants of Turing Machines

There are in fact many possible ways to define Turing machines, of which ours is only one. In some ways, our definition is more liberal than others. We allow arbitrary finite alphabets, a more restricted definition might allow only two tape symbols, 1 and 0. We allow the machine to write a symbol to the tape and move at the same time, other definitions allow either writing or moving. We allow the possibility of writing without moving the tape head, other definitions leave out the N "instruction."

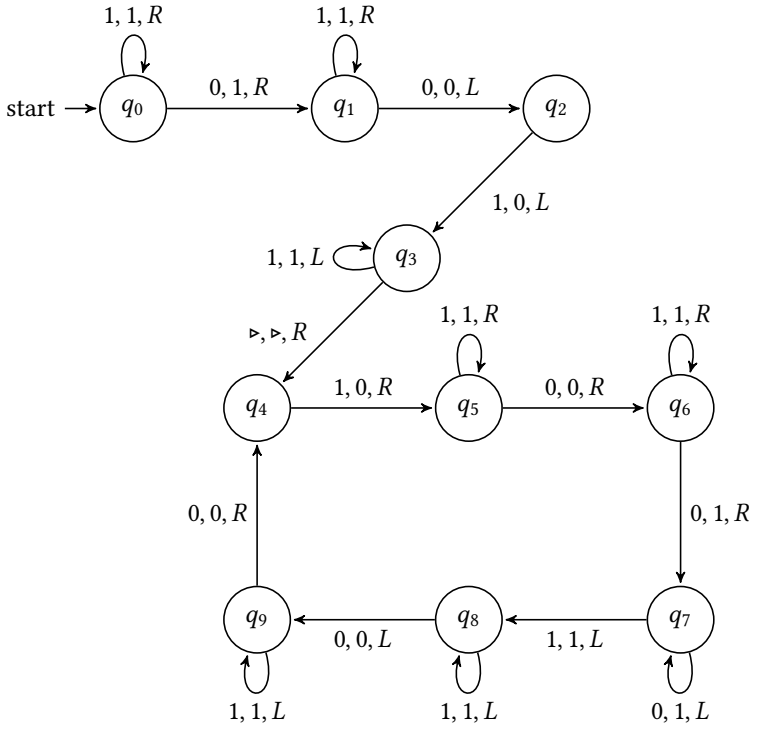


Figure 14.3: Combining adder and doubler machines

In other ways, our definition is more restrictive. We assumed that the tape is infinite in one direction only, other definitions allow the tape to be infinite both to the left and the right. In fact, one can even allow any number of separate tapes, or even an infinite grid of squares. We represent the instruction set of the Turing machine by a transition function; other definitions use a transition relation where the machine has more than one possible instruction in any given situation.

This last relaxation of the definition is particularly interesting. In our definition, when the machine is in state q reading symbol σ, $\delta(q, \sigma)$ determines what the new symbol, state, and tape head position is. But if we allow the instruction set to be a relation between current state-symbol pairs $\langle q, \sigma \rangle$ and new state-symbol-direction triples $\langle q', \sigma', D \rangle$, the action of the Turing machine may not be uniquely determined—the instruction relation may contain both $\langle q, \sigma, q', \sigma', D \rangle$ and $\langle q, \sigma, q'', \sigma'', D' \rangle$. In this case we have a *non-deterministic* Turing machine. These play an important role in computational complexity theory.

There are also different conventions for when a Turing machine halts: we say it halts when the transition function is undefined, other definitions require the machine to be in a special designated halting state. Since the tapes of our turing machines are infinite in one direction only, there are cases where a Turing machine can't properly carry out an instruction: if it reads the leftmost square and is supposed to move left. According to our definition, it just stays put instead, but we could have defined it so that it halts when that happens.

There are also different ways of representing numbers (and hence the input-output function computed by a Turing machine): we use unary representation, but you can also use binary representation. This requires two symbols in addition to 0 and ▷.

Now here is an interesting fact: none of these variations matters as to which functions are Turing computable. *If a function is Turing computable according to one definition, it is Turing computable according to all of them.*

14.9 The Church-Turing Thesis

Turing machines are supposed to be a precise replacement for the concept of an effective procedure. Turing thought that anyone who grasped both the concept of an effective procedure and the concept of a Turing machine would have the intuition that anything that could be done via an effective procedure could be done by Turing machine. This claim is given support by the fact that all the other proposed precise replacements for the concept of an effective procedure turn out to be extensionally equivalent to the concept of a Turing machine —that is, they can compute exactly the same set of functions. This claim is called the *Church-Turing thesis*.

Definition 14.15 (Church-Turing thesis). The *Church-Turing Thesis* states that anything computable via an effective procedure is Turing computable.

The Church-Turing thesis is appealed to in two ways. The first kind of use of the Church-Turing thesis is an excuse for laziness. Suppose we have a description of an effective procedure to compute something, say, in "pseudo-code." Then we can invoke the Church-Turing thesis to justify the claim that the same function is computed by some Turing machine, even if we have not in fact constructed it.

The other use of the Church-Turing thesis is more philosophically interesting. It can be shown that there are functions whch cannot be computed by a Turing machines. From this, using the Church-Turing thesis, one can conclude that it cannot be effectively computed, using any procedure whatsoever. For if there were such a procedure, by the Church-Turing thesis, it would follow that there would be a Turing machine. So if we can prove that there is no Turing machine that computes it, there also can't be an effective procedure. In particular, the Church-Turing thesis is invoked to claim that the so-called halting problem not only cannot be solved by Turing machines, it cannot be effectively solved at all.

Problems

Problem 14.1. Choose an arbitary input and trace through the configurations of the doubler machine in Example 14.4.

Problem 14.2. The double machine in Example 14.4 writes its output to the right of the input. Come up with a new method for solving the doubler problem which generates its output immediately to the right of the end-of-tape marker. Build a machine that executes your method. Check that your machine works by tracing through the configurations.

Problem 14.3. Design a Turing-machine with alphabet $\{\triangleright, 0, A, B\}$ that accepts, i.e., halts on, any string of A's and B's where the number of A's is the same as the number of B's *and* all the A's precede all the B's, and rejects, i.e., does not halt on, any string

where the number of A's is not equal to the number of B's or the A's do not precede all the B's. (E.g., the machine should accept $AABB$, and $AAABBB$, but reject both AAB and $AABBAABB$.)

Problem 14.4. Design a Turing-machine with alphabet $\{\triangleright, 0, A, B\}$ that takes as input any string α of A's and B's and duplicates them to produce an output of the form $\alpha\alpha$. (E.g. input $ABBA$ should result in output $ABBAABBA$).

Problem 14.5. *Alphabetical?:* Design a Turing-machine with alphabet $\{\triangleright, 0, A, B\}$ that when given as input a finite sequence of A's and B's checks to see if all the A's appear to the left of all the B's or not. The machine should leave the input string on the tape, and either halt if the string is "alphabetical", or loop forever if the string is not.

Problem 14.6. *Alphabetizer:* Design a Turing-machine with alphabet $\{\triangleright, 0, A, B\}$ that takes as input a finite sequence of A's and B's rearranges them so that all the A's are to the left of all the B's. (e.g., the sequence $BABAA$ should become the sequence $AAABB$, and the sequence $ABBABB$ should become the sequence $AABBBB$).

Problem 14.7. Trace through the configurations of the machine for input $\langle 3, 5\rangle$.

Problem 14.8. *Subtraction:* Design a Turing machine that when given an input of two non-empty strings of strokes of length n and m, where $n > m$, computes the function $f(n, m) = n - m$.

Problem 14.9. *Equality:* Design a Turing machine to compute the following function:

$$\text{equality}(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$$

where x and y are integers greater than 0.

Problem 14.10. Design a Turing machine to compute the function $\min(x, y)$ where x and y are positive integers represented on the tape by strings of 1's separated by a 0. You may use additional symbols in the alphabet of the machine.

The function min selects the smallest value from its arguments, so $\min(3, 5) = 3$, $\min(20, 16) = 16$, and $\min(4, 4) = 4$, and so on.

Chapter 15

Undecidability

15.1 Introduction

It might seem obvious that not every function, even every arithmetical function, can be computable. There are just too many, whose behavior is too complicated. Functions defined from the decay of radioactive particles, for instance, or other chaotic or random behavior. Suppose we start counting 1-second intervals from a given time, and define the function $f(n)$ as the number of particles in the universe that decay in the n-th 1-second interval after that initial moment. This seems like a candidate for a function we cannot ever hope to compute.

But it is one thing to not be able to imagine how one would compute such functions, and quite another to actually prove that they are uncomputable. In fact, even functions that seem hopelessly complicated may, in an abstract sense, be computable. For instance, suppose the universe is finite in time—some day, in the very distant future the universe will contract into a single point, as some cosmological theories predict. Then there is only a finite (but incredibly large) number of seconds from that initial moment for which $f(n)$ is defined. And any function which is defined for only finitely many inputs is computable: we could list the outputs in one big table, or code it in one very big Turing machine state transition diagram.

We are often interested in special cases of functions whose values give the answers to yes/no questions. For instance, the question "is n a prime number?" is associated with the function

$$\text{isprime}(n) = \begin{cases} 1 & \text{if } n \text{ is prime} \\ 0 & \text{otherwise.} \end{cases}$$

We say that a yes/no question can be *effectively decided*, if the associated 1/0-valued function is effectively computable.

To prove mathematically that there are functions which cannot be effectively computed, or problems that cannot effectively decided, it is essential to fix a specific model of computation, and show about it that there are functions it cannot compute or problems it cannot decide. We can show, for instance, that not every function can be computed by Turing machines, and not every problem can be decided by Turing machines. We can then appeal to the Church-Turing thesis to conclude that not only are Turing machines not powerful enough to compute every function, but no effective procedure can.

The key to proving such negative results is the fact that we can assign numbers to Turing machines themselves. The easiest way to do this is to enumerate them,

perhaps by fixing a specific way to write down Turing machines and their programs, and then listing them in a systematic fashion. Once we see that this can be done, then the existence of Turing-uncomputable functions follows by simple cardinality considerations: the set of functions from $\mathbb{N}$ to $\mathbb{N}$ (in fact, even just from $\mathbb{N}$ to $\{0, 1\}$) are uncountable, but since we can enumerate all the Turing machines, the set of Turing-computable functions is only countably infinite.

We can also define *specific* functions and problems which we can prove to be uncomputable and undecidable, respectively. One such problem is the so-called *Halting Problem.* Turing machines can be finitely described by listing their instructions. Such a description of a Turing machine, i.e., a Turing machine program, can of course be used as input to another Turing machine. So we can consider Turing machines that decide questions about other Turing machines. One particularly interesting question is this: "Does the given Turing machine eventually halt when started on input n?" It would be nice if there were a Turing machine that could decide this question: think of it as a quality-control Turing machine which ensures that Turing machines don't get caught in infinite loops and such. The interesting fact, which Turing proved, is that there cannot be such a Turing machine. There cannot be a single Turing machine which, when started on input consisting of a description of a Turing machine M and some number n, will always halt with either output 1 or 0 according to whether M machine would have halted when started on input n or not.

Once we have examples of specific undecidable problems we can use them to show that other problems are undecidable, too. For instance, one celebrated undecidable problem is the question, "Is the first-order formula φ valid?". There is no Turing machine which, given as input a first-order formula φ, is guaranteed to halt with output 1 or 0 according to whether φ is valid or not. Historically, the question of finding a procedure to effectively solve this problem was called simply "the" decision problem; and so we say that the decision problem is unsolvable. Turing and Church proved this result independently at around the same time, so it is also called the Church-Turing Theorem.

15.2 Enumerating Turing Machines

We can show that the set of all Turing-machines is countable. This follows from the fact that each Turing machine can be finitely described. The set of states and the tape vocabulary are finite sets. The transition function is a partial function from $Q \times \Sigma$ to $Q \times \Sigma \times \{L, R, N\}$, and so likewise can be specified by listing its values for the finitely many argument pairs for which it is defined. Of course, strictly speaking, the states and vocabulary can be anything; but the *behavior* of the Turing machine is independent of which objects serve as states and vocabulary. So we may assume, for instance, that the states and vocabulary symbols are natural numbers, or that the states and vocabulary are all strings of letters and digits.

Suppose we fix a countably infinite vocabulary for specifying Turing machines: $\sigma_0 = \triangleright$, $\sigma_1 = 0$, $\sigma_2 = 1$, $\sigma_3, \ldots, R, L, N, q_0, q_1, \ldots$. Then any Turing machine can be specified by some finite string of symbols from this alphabet (though not every finite string of symbols specifies a Turing machine). For instance, suppose we have a Turing machine $M = \langle Q, \Sigma, q, \delta \rangle$ where

$$Q = \{q'_0, \ldots, q'_n\} \subseteq \{q_0, q_1, \ldots\} \text{ and}$$
$$\Sigma = \{\triangleright, \sigma'_1, \sigma'_2, \ldots, \sigma'_m\} \subseteq \{\sigma_0, \sigma_1, \ldots\}.$$

We could specify it by the string

$$q'_0 q'_1 \dots q'_n \triangleright \sigma'_1 \dots \sigma'_m \triangleright q \triangleright S(\sigma'_0, q'_0) \triangleright \dots \triangleright S(\sigma'_m, q'_n)$$

where $S(\sigma'_i, q'_j)$ is the string $\sigma'_i q'_j \delta(\sigma'_i, q'_j)$ if $\delta(\sigma'_i, q'_j)$ is defined, and $\sigma'_i q'_j$ otherwise.

Theorem 15.1. *There are functions from $\mathbb{N}$ to $\mathbb{N}$ which are not Turing computable.*

Proof. We know that the set of finite strings of symbols from a countably infinite alphabet is countable. This gives us that the set of descriptions of Turing machines, as a subset of the finite strings from the countable vocabulary $\{q_0, q_1, \dots, \triangleright, \sigma_1, \sigma_2, \dots\}$, is itself enumerable. Since every Turing computable function is computed by some (in fact, many) Turing machines, this means that the set of all Turing computable functions from $\mathbb{N}$ to $\mathbb{N}$ is also enumerable.

On the other hand, the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not countable. This follows immediately from the fact that not even the set of all functions of one argument from $\mathbb{N}$ to the set $\{0, 1\}$ is countable. If all functions were computable by some Turing machine we could enumerate the set of all functions. So there are some functions that are not Turing-computable. □

15.3 The Halting Problem

Assume we have fixed some finite descriptions of Turing machines. Using these, we can enumerate Turing machines via their descriptions, say, ordered by the lexicographic ordering. Each Turing machine thus receives an *index*: its place in the enumeration $M_1, M_2, M_3, \dots$ of Turing machine descriptions.

We know that there must be non-Turing-computable functions: the set of Turing machine descriptions—and hence the set of Turing machines—is enumerable, but the set of all functions from $\mathbb{N}$ to $\mathbb{N}$ is not. But we can find specific examples of non-computable function as well. One such function is the halting function.

Definition 15.2 (Halting function). The *halting function h* is defined as

$$h(e, n) = \begin{cases} 0 & \text{if machine } M_e \text{ does not halt for input } n \\ 1 & \text{if machine } M_e \text{ halts for input } n \end{cases}$$

Definition 15.3 (Halting problem). The *Halting Problem* is the problem of determining (for any e, n) whether the Turing machine M_e halts for an input of n strokes.

We show that h is not Turing-computable by showing that a related function, s, is not Turing-computable. This proof relies on the fact that anything that can be computed by a Turing machine can be computed using just two symbols: 0 and 1, and the fact that two Turing machines can be hooked together to create a single machine.

Definition 15.4. The function s is defined as

$$s(e) = \begin{cases} 0 & \text{if machine } M_e \text{ does not halt for input } e \\ 1 & \text{if machine } M_e \text{ halts for input } e \end{cases}$$

Lemma 15.5. *The function s is not Turing computable.*

Proof. We suppose, for contradiction, that the function s is Turing-computable. Then there would be a Turing machine S that computes s. We may assume, without loss of generality, that when S halts, it does so while scanning the first square. This machine can be "hooked up" to another machine J, which halts if it is started on a blank tape (i.e., if it reads 0 in the initial state while scanning the square to the right of the end-of-tape symbol), and otherwise wanders off to the right, never halting. $S \frown J$, the machine created by hooking S to J, is a Turing machine, so it is M_e for some e (i.e., it appears somewhere in the enumeration). Start M_e on an input of e 1s. There are two possibilities: either M_e halts or it does not halt.

1. Suppose M_e halts for an input of e 1s. Then $s(e) = 1$. So S, when started on e, halts with a single 1 as output on the tape. Then J starts with a 1 on the tape. In that case J does not halt. But M_e is the machine $S \frown J$, so it should do exactly what S followed by J would do. So M_e cannot halt for an input of e 1's.

2. Now suppose M_e does not halt for an input of e 1s. Then $s(e) = 0$, and S, when started on input e, halts with a blank tape. J, when started on a blank tape, immediately halts. Again, M_e does what S followed by J would do, so M_e must halt for an input of e 1's.

This shows there cannot be a Turing machine S: s is not Turing computable. □

Theorem 15.6 (Unsolvability of the Halting Problem). *The halting problem is unsolvable, i.e., the function h is not Turing computable.*

Proof. Suppose h were Turing computable, say, by a Turing machine H. We could use H to build a Turing machine that computes s: First, make a copy of the input (separated by a blank). Then move back to the beginning, and run H. We can clearly make a machine that does the former, and if H existed, we would be able to "hook it up" to such a modified doubling machine to get a new machine which would determine if M_e halts on input e, i.e., computes s. But we've already shown that no such machine can exist. Hence, h is also not Turing computable. □

15.4 The Decision Problem

We say that first-order logic is *decidable* iff there is an effective method for determining whether or not a given sentence is valid. As it turns out, there is no such method: the problem of deciding validity of first-order sentences is unsolvable.

In order to establish this important negative result, we prove that the decision problem cannot be solved by a Turing machine. That is, we show that there is no Turing machine which, whenever it is started on a tape that contains a first-order sentence, eventually halts and outputs either 1 or 0 depending on whether the sentence is valid or not. By the Church-Turing thesis, every function which is computable is Turing computable. So if this "validity function" were effectively computable at all, it would be Turing computable. If it isn't Turing computable, then, it also cannot be effectively computable.

Our strategy for proving that the decision problem is unsolvable is to reduce the halting problem to it. This means the following: We have proved that the function $h(e, w)$ that halts with output 1 if the Turing-machine described by e halts on input w and outputs 0 otherwise, is not Turing-computable. We will show that if there were a Turing machine that decides validity of first-order sentences, then there

is also Turing machine that computes h. Since h cannot be computed by a Turing machine, there cannot be a Turing machine that decides validity either.

The first step in this strategy is to show that for every input w and a Turing machine M, we can effectively describe a sentence $\tau(M, w)$ representing the instruction set of M and the input w and a sentence $\alpha(M, w)$ expressing "M eventually halts" such that:

$\vDash \tau(M, w) \to \alpha(M, w)$ iff M halts for input w.

The bulk of our proof will consist in describing these sentences $\tau(M, w)$ and $\alpha(M, w)$ and verifying that $\tau(M, w) \to \alpha(M, w)$ is valid iff M halts on input w.

15.5 Representing Turing Machines

In order to represent Turing machines and their behavior by a sentence of first-order logic, we have to define a suitable language. The language consists of two parts: predicate symbols for describing configurations of the machine, and expressions for numbering execution steps ("moments") and positions on the tape.

We introduce two kinds of predicate symbols, both of them 2-place: For each state q, a predicate symbol Q_q, and for each tape symbol σ, a predicate symbol S_σ. The former allow us to describe the state of M and the position of its tape head, the latter allow us to describe the contents of the tape.

In order to express the positions of the tape head and the number of steps executed, we need a way to express numbers. This is done using a constant symbol 0, and a 1-place function $'$, the successor function. By convention it is written *after* its argument (and we leave out the parentheses). So 0 names the leftmost position on the tape as well as the time before the first execution step (the initial configuration), $0'$ names the square to the right of the leftmost square, and the time after the first execution step, and so on. We also introduce a predicate symbol $<$ to express both the ordering of tape positions (when it means "to the left of") and execution steps (then it means "before").

Once we have the language in place, we list the "axioms" of $\tau(M, w)$, i.e., the sentences which, taken together, describe the behavior of M when run on input w. There will be sentences which lay down conditions on 0, $'$, and $<$, sentences that describes the input configuration, and sentences that describe what the configuration of M is after it executes a particular instruction.

Definition 15.7. Given a Turing machine $M = \langle Q, \Sigma, q_0, \delta \rangle$, the language $\mathcal{L}_M$ consists of:

1. A two-place predicate symbol $Q_q(x, y)$ for every state $q \in Q$. Intuitively, $Q_q(\overline{m}, \overline{n})$ expresses "after n steps, M is in state q scanning the mth square."

2. A two-place predicate symbol $S_\sigma(x, y)$ for every symbol $\sigma \in \Sigma$. Intuitively, $S_\sigma(\overline{m}, \overline{n})$ expresses "after n steps, the mth square contains symbol σ."

3. A constant symbol 0

4. A one-place function symbol $'$

5. A two-place predicate symbol $<$

For each number n there is a canonical term $\overline{n}$, the *numeral* for n, which represents it in $\mathcal{L}_M$. $\overline{0}$ is 0, $\overline{1}$ is $0'$, $\overline{2}$ is $0''$, and so on. More formally:

$$\overline{0} = 0$$
$$\overline{n+1} = \overline{n}'$$

The sentences describing the operation of the Turing machine M on input $w = \sigma_{i_1} \dots \sigma_{i_k}$ are the following:

1. Axioms describing numbers:
   a) A sentence that says that the successor function is injective:
   $$\forall x \, \forall y \, (x' = y' \to x = y)$$
   b) A sentence that says that every number is less than its successor:
   $$\forall x \, x < x'$$
   c) A sentence that ensures that $<$ is transitive:
   $$\forall x \, \forall y \, \forall z \, ((x < y \wedge y < z) \to x < z)$$
   d) A sentence that connects $<$ and $=$:
   $$\forall x \, \forall y \, (x < y \to x \neq y)$$
2. Axioms describing the input configuration:
   a) After 0 steps—before the machine starts—M is in the inital state q_0, scanning square 1:
   $$Q_{q_0}(\overline{1}, \overline{0})$$
   b) The first $k+1$ squares contain the symbols $\triangleright, \sigma_{i_1}, \dots, \sigma_{i_k}$:
   $$S_{\triangleright}(\overline{0}, \overline{0}) \wedge S_{\sigma_{i_1}}(\overline{1}, \overline{0}) \wedge \dots \wedge S_{\sigma_{i_k}}(\overline{k}, \overline{0})$$
   c) Otherwise, the tape is empty:
   $$\forall x \, (\overline{k} < x \to S_0(x, \overline{0}))$$
3. Axioms describing the transition from one configuration to the next:
   For the following, let $\varphi(x, y)$ be the conjunction of all sentences of the form
   $$\forall z \, (((z < x \vee x < z) \wedge S_\sigma(z, y)) \to S_\sigma(z, y'))$$
   where $\sigma \in \Sigma$. We use $\varphi(\overline{m}, \overline{n})$ to express "other than at square m, the tape after $n+1$ steps is the same as after n steps."
   a) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', R \rangle$, the sentence:
   $$\begin{aligned} \forall x \, \forall y \, ((&Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \to \\ &(Q_{q_j}(x', y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y))) \end{aligned}$$
   This says that if, after y steps, the machine is in state q_i scanning square x which contains symbol σ, then after $y+1$ steps it is scanning square $x+1$, is in state q_j, square x now contains σ', and every square other than x contains the same symbol as it did after y steps.

b) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', L\rangle$, the sentence:

$$\begin{aligned}&\forall x\, \forall y\, ((Q_{q_i}(x', y) \wedge S_\sigma(x', y)) \rightarrow \\ &\qquad (Q_{q_j}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\ &\forall y\, ((Q_{q_i}(0, y) \wedge S_\sigma(0, y)) \rightarrow \\ &\qquad (Q_{q_j}(0, y') \wedge S_{\sigma'}(0, y') \wedge \varphi(0, y)))\end{aligned}$$

Take a moment to think about how this works: now we don't start with "if scanning square x ..." but: "if scanning square $x + 1$ ..." A move to the left means that in the next step the machine is scanning square x. But the square that is written on is $x + 1$. We do it this way since we don't have subtraction or a predecessor function.

Note that numbers of the form $x + 1$ are 1, 2, ..., i.e., this doesn't cover the case where the machine is scanning square 0 and is supposed to move left (which of course it can't—it just stays put). That special case is covered by the second conjunction: it says that if, after y steps, the machine is scanning square 0 in state q_i and square 0 contains symbol σ, then after $y + 1$ steps it's still scanning square 0, is now in state q_j, the symbol on square 0 is σ', and the squares other than square 0 contain the same symbols they contained ofter y steps.

c) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', N\rangle$, the sentence:

$$\begin{aligned}&\forall x\, \forall y\, ((Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \rightarrow \\ &\qquad (Q_{q_j}(x, y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y)))\end{aligned}$$

Let $\tau(M, w)$ be the conjunction of all the above sentences for Turing machine M and input w.

In order to express that M eventually halts, we have to find a sentence that says "after some number of steps, the transition function will be undefined." Let X be the set of all pairs $\langle q, \sigma\rangle$ such that $\delta(q, \sigma)$ is undefined. Let $\alpha(M, w)$ then be the sentence

$$\exists x\, \exists y\, (\bigvee_{\langle q,\sigma\rangle \in X} (Q_q(x, y) \wedge S_\sigma(x, y)))$$

If we use a Turing machine with a designated halting state h, it is even easier: then the sentence $\alpha(M, w)$

$$\exists x\, \exists y\, Q_h(x, y)$$

expresses that the machine eventually halts.

Proposition 15.8. *If $m < k$, then $\tau(M, w) \vDash \overline{m} < \overline{k}$*

Proof. Exercise. □

15.6 Verifying the Representation

In order to verify that our representation works, we have to prove two things. First, we have to show that if M halts on input w, then $\tau(M, w) \rightarrow \alpha(M, w)$ is valid. Then, we have to show the converse, i.e., that if $\tau(M, w) \rightarrow \alpha(M, w)$ is valid, then M does in fact eventually halt when run on input w.

The strategy for proving these is very different. For the first result, we have to show that a sentence of first-order logic (namely, $\tau(M, w) \to \alpha(M, w)$) is valid. The easiest way to do this is to give a derivation. Our proof is supposed to work for all M and w, though, so there isn't really a single sentence for which we have to give a derivation, but infinitely many. So the best we can do is to prove by induction that, whatever M and w look like, and however many steps it takes M to halt on input w, there will be a derivation of $\tau(M, w) \to \alpha(M, w)$.

Naturally, our induction will proceed on the number of steps M takes before it reaches a halting configuration. In our inductive proof, we'll establish that for each step n of the run of M on input w, $\tau(M, w) \vDash \chi(M, w, n)$, where $\chi(M, w, n)$ correctly describes the configuration of M run on w after n steps. Now if M halts on input w after, say, n steps, $\chi(M, w, n)$ will describe a halting configuration. We'll also show that $\chi(M, w, n) \vDash \alpha(M, w)$, whenever $\chi(M, w, n)$ describes a halting configuration. So, if M halts on input w, then for some n, M will be in a halting configuration after n steps. Hence, $\tau(M, w) \vDash \chi(M, w, n)$ where $\chi(M, w, n)$ describes a halting configuration, and since in that case $\chi(M, w, n) \vDash \alpha(M, w)$, we get that $T(M, w) \vDash \alpha(M, w)$, i.e., that $\vDash \tau(M, w) \to \alpha(M, w)$.

The strategy for the converse is very different. Here we assume that $\vDash \tau(M, w) \to \alpha(M, w)$ and have to prove that M halts on input w. From the hypothesis we get that $\tau(M, w) \vDash \alpha(M, w)$, i.e., $\alpha(M, w)$ is true in every structure in which $\tau(M, w)$ is true. So we'll describe a structure $\mathfrak{M}$ in which $\tau(M, w)$ is true: its domain will be $\mathbb{N}$, and the interpretation of all the Q_q and S_σ will be given by the configurations of M during a run on input w. So, e.g., $\mathfrak{M} \vDash Q_q(\overline{m}, \overline{n})$ iff T, when run on input w for n steps, is in state q and scanning square m. Now since $\tau(M, w) \vDash \alpha(M, w)$ by hypothesis, and since $\mathfrak{M} \vDash \tau(M, w)$ by construction, $\mathfrak{M} \vDash \alpha(M, w)$. But $\mathfrak{M} \vDash \alpha(M, w)$ iff there is some $n \in |\mathfrak{M}| = \mathbb{N}$ so that M, run on input w, is in a halting configuration after n steps.

Definition 15.9. Let $\chi(M, w, n)$ be the sentence

$$Q_q(\overline{m}, \overline{n}) \wedge S_{\sigma_0}(\overline{0}, \overline{n}) \wedge \cdots \wedge S_{\sigma_k}(\overline{k}, \overline{n}) \wedge \forall x\, (\overline{k} < x \to S_0(x, \overline{n}))$$

where q is the state of M at time n, M is scanning square m at time n, square i contains symbol σ_i at time n for $0 \leq i \leq k$ and k is the right-most non-blank square of the tape at time 0, or the right-most square the tape head has visited after n steps, whichever is greater.

Lemma 15.10. *If M run on input w is in a halting configuration after n steps, then $\chi(M, w, n) \vDash \alpha(M, w)$.*

Proof. Suppose that M halts for input w after n steps. There is some state q, square m, and symbol σ such that:

1. After n steps, M is in state q scanning square m on which σ appears.
2. The transition function $\delta(q, \sigma)$ is undefined.

$\chi(M, w, n)$ is the description of this configuration and will include the clauses $Q_q(\overline{m}, \overline{n})$ and $S_\sigma(\overline{m}, \overline{n})$. These clauses together imply $\alpha(M, w)$:

$$\exists x\, \exists y\, \Big(\bigvee_{\langle q, \sigma \rangle \in X} (Q_q(x, y) \wedge S_\sigma(x, y)) \Big)$$

since $Q_{q'}(\overline{m}, \overline{n}) \wedge S_{\sigma'}(\overline{m}, \overline{n}) \vDash \bigvee_{\langle q, \sigma \rangle \in X} (Q_q(\overline{m}, \overline{n}) \wedge S_\sigma(\overline{m}, \overline{n}))$, as $\langle q', \sigma' \rangle \in X$. □

So if M halts for input w, then there is some n such that $\chi(M, w, n) \vDash \alpha(M, w)$. We will now show that for any time n, $\tau(M, w) \vDash \chi(M, w, n)$.

Lemma 15.11. *For each n, if M has not halted after n steps, $\tau(M, w) \vDash \chi(M, w, n)$.*

Proof. Induction basis: If $n = 0$, then the conjuncts of $\chi(M, w, 0)$ are also conjuncts of $\tau(M, w)$, so entailed by it.

Inductive hypothesis: If M has not halted before the nth step, then $\tau(M, w) \vDash \chi(M, w, n)$. We have to show that (unless $\chi(M, w, n)$ describes a halting configuration), $\tau(M, w) \vDash \chi(M, w, n + 1)$.

Suppose $n > 0$ and after n steps, M started on w is in state q scanning square m. Since M does not halt after n steps, there must be an instruction of one of the following three forms in the program of M:

1. $\delta(q, \sigma) = \langle q', \sigma', R\rangle$
2. $\delta(q, \sigma) = \langle q', \sigma', L\rangle$
3. $\delta(q, \sigma) = \langle q', \sigma', N\rangle$

We will consider each of these three cases in turn.

1. Suppose there is an instruction of the form (1). By Definition 15.7(3a), this means that

$$\forall x\, \forall y\, ((Q_q(x, y) \wedge S_\sigma(x, y)) \rightarrow (Q_{q'}(x', y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y)))$$

   is a conjunct of $\tau(M, w)$. This entails the following sentence (universal instantiation, $\overline{m}$ for x and $\overline{n}$ for y):

$$(Q_q(\overline{m}, \overline{n}) \wedge S_\sigma(\overline{m}, \overline{n})) \rightarrow (Q_{q'}(\overline{m}', \overline{n}') \wedge S_{\sigma'}(\overline{m}, \overline{n}') \wedge \varphi(\overline{m}, \overline{n})).$$

   By induction hypothesis, $\tau(M, w) \vDash \chi(M, w, n)$, i.e.,

$$Q_q(\overline{m}, \overline{n}) \wedge S_{\sigma_0}(\overline{0}, \overline{n}) \wedge \cdots \wedge S_{\sigma_k}(\overline{k}, \overline{n}) \wedge \forall x\, (\overline{k} < x \rightarrow S_0(x, \overline{n}))$$

   Since after n steps, tape square m contains σ, the corresponding conjunct is $S_\sigma(\overline{m}, \overline{n})$, so this entails:

$$Q_q(\overline{m}, \overline{n}) \wedge S_\sigma(\overline{m}, \overline{n}))$$

   We now get

$$Q_{q'}(\overline{m}', \overline{n}') \wedge S_{\sigma'}(\overline{m}, \overline{n}') \wedge S_{\sigma_0}(\overline{0}, \overline{n}') \wedge \cdots \wedge S_{\sigma_k}(\overline{k}, \overline{n}') \wedge \forall x\, (\overline{k} < x \rightarrow S_0(x, \overline{n}'))$$

   as follows: The first line comes directly from the consequent of the preceding conditional, by modus ponens. Each conjunct in the middle line—which

excludes $S_{\sigma_m}(\overline{m}, \overline{n}')$—follows from the corresponding conjunct in $\chi(M, w, n)$ together with $\varphi(\overline{m}, \overline{n})$.

If $m < k$, $\tau(M, w) \vdash \overline{m} < \overline{k}$ (Proposition 15.8) and by transitivity of $<$, we have $\forall x\,(\overline{k} < x \to \overline{m} < x)$. If $m = k$, then $\forall x\,(\overline{k} < x \to \overline{m} < x)$ by logic alone. The last line then follows from the corresponding conjunct in $\chi(M, w, n)$, $\forall x\,(\overline{k} < x \to \overline{m} < x)$, and $\varphi(\overline{m}, \overline{n})$. If $m < k$, this already is $\chi(M, w, n+1)$.

Now suppose $m = k$. In that case, after $n+1$ steps, the tape head has also visited square $k+1$, which now is the right-most square visited. So $\chi(M, w, n+1)$ has a new conjunct, $S_0(\overline{k}', \overline{n}')$, and the last conjuct is $\forall x\,(\overline{k}' < x \to S_0(x, \overline{n}'))$. We have to verify that these two sentences are also implied.

We already have $\forall x\,(\overline{k} < x \to S_0(x, \overline{n}'))$. In particular, this gives us $\overline{k} < \overline{k}' \to S_0(\overline{k}', \overline{n}')$. From the axiom $\forall x\, x < x'$ we get $\overline{k} < \overline{k}'$. By modus ponens, $S_0(\overline{k}', \overline{n}')$ follows.

Also, since $\tau(M, w) \vdash \overline{k} < \overline{k}'$, the axiom for transitivity of $<$ gives us $\forall x\,(\overline{k}' < x \to S_0(x, \overline{n}'))$. (We leave the verification of this as an exercise.)

2. Suppose there is an instruction of the form (2). Then, by Definition 15.7(3b),

$$\begin{aligned}
&\forall x\,\forall y\,((Q_q(x', y) \wedge S_\sigma(x', y)) \to \\
&\qquad (Q_{q'}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\
&\forall y\,((Q_{q_i}(0, y) \wedge S_\sigma(0, y)) \to \\
&\qquad (Q_{q_j}(0, y') \wedge S_{\sigma'}(0, y') \wedge \varphi(0, y)))
\end{aligned}$$

is a conjunct of $\tau(M, w)$. If $m > 0$, then let $l = m - 1$ (i.e., $m = l + 1$). The first conjunct of the above sentence entails the following:

$$\begin{aligned}
&(Q_q(\overline{l}', \overline{n}) \wedge S_\sigma(\overline{l}', \overline{n})) \to \\
&\qquad (Q_{q'}(\overline{l}, \overline{n}') \wedge S_{\sigma'}(\overline{l}', \overline{n}') \wedge \varphi(\overline{l}, \overline{n}))
\end{aligned}$$

Otherwise, let $l = m = 0$ and consider the following sentence entailed by the second conjunct:

$$\begin{aligned}
&((Q_{q_i}(0, \overline{n}) \wedge S_\sigma(0, \overline{n})) \to \\
&\qquad (Q_{q_j}(0, \overline{n}') \wedge S_{\sigma'}(0, \overline{n}') \wedge \varphi(0, \overline{n})))
\end{aligned}$$

Either sentence implies

$$\begin{aligned}
&Q_{q'}(\overline{l}, \overline{n}') \wedge S_{\sigma'}(\overline{m}, \overline{n}') \wedge \\
&\qquad S_{\sigma_0}(\overline{0}, \overline{n}') \wedge \cdots \wedge S_{\sigma_k}(\overline{k}, \overline{n}') \wedge \\
&\qquad \forall x\,(\overline{k} < x \to S_0(x, \overline{n}'))
\end{aligned}$$

as before. (Note that in the first case, $\overline{l}' \equiv \overline{l+1} \equiv \overline{m}$ and in the second case $\overline{l} \equiv 0$.) But this just is $\chi(M, w, n+1)$.

3. Case (3) is left as an exercise.

We have shown that for any n, $\tau(M, w) \vDash \chi(M, w, n)$. □

Lemma 15.12. *If M halts on input w, then $\tau(M, w) \to \alpha(M, w)$ is valid.*

Proof. By Lemma 15.11, we know that, for any time n, the description $\chi(M, w, n)$ of the configuration of M at time n is entailed by $\tau(M, w)$. Suppose M halts after k steps. It will be scanning square m, say. Then $\chi(M, w, k)$ describes a halting configuration of M, i.e., it contains as conjuncts both $Q_q(\overline{m}, \overline{k})$ and $S_\sigma(\overline{m}, \overline{k})$ with $\delta(q, \sigma)$ undefined. Thus, by Lemma 15.10, $\chi(M, w, k) \vDash \alpha(M, w)$. But since $\tau(M, w) \vDash \chi(M, w, k)$, we have $\tau(M, w) \vDash \alpha(M, w)$ and therefore $\tau(M, w) \to \alpha(M, w)$ is valid. □

To complete the verification of our claim, we also have to establish the reverse direction: if $\tau(M, w) \to \alpha(M, w)$ is valid, then M does in fact halt when started on input m.

Lemma 15.13. *If $\vDash \tau(M, w) \to \alpha(M, w)$, then M halts on input w.*

Proof. Consider the $\mathcal{L}_M$-structure $\mathfrak{M}$ with domain $\mathbb{N}$ which interprets 0 as 0, $'$ as the successor function, and $<$ as the less-than relation, and the predicates Q_q and S_σ as follows:

$$Q_q^{\mathfrak{M}} = \{\langle m, n\rangle \mid \begin{array}{l}\text{started on } w, \text{ after } n \text{ steps,}\\ M \text{ is in state } q \text{ scanning square } m\end{array}\}$$

$$S_\sigma^{\mathfrak{M}} = \{\langle m, n\rangle \mid \begin{array}{l}\text{started on } w, \text{ after } n \text{ steps,}\\ \text{square } m \text{ of } M \text{ contains symbol } \sigma\end{array}\}$$

In other words, we construct the structure $\mathfrak{M}$ so that it describes what M started on input w actually does, step by step. Clearly, $\mathfrak{M} \vDash \tau(M, w)$. If $\vDash \tau(M, w) \to \alpha(M, w)$, then also $\mathfrak{M} \vDash \alpha(M, w)$, i.e.,

$$\mathfrak{M} \vDash \exists x\, \exists y\, (\bigvee_{\langle q,\sigma\rangle \in X} (Q_q(x, y) \wedge S_\sigma(x, y))).$$

As $|\mathfrak{M}| = \mathbb{N}$, there must be $m, n \in \mathbb{N}$ so that $\mathfrak{M} \vDash Q_q(\overline{m}, \overline{n}) \wedge S_\sigma(\overline{m}, \overline{n})$ for some q and σ such that $\delta(q, \sigma)$ is undefined. By the definition of $\mathfrak{M}$, this means that M started on input w after n steps is in state q and reading symbol σ, and the transition function is undefined, i.e., M has halted. □

15.7 The Decision Problem is Unsolvable

Theorem 15.14. *The decision problem is unsolvable.*

Proof. Suppose the decision problem were solvable, i.e., suppose there were a Turing machine D of the following sort. Whenever D is started on a tape that contains a sentence ψ of first-order logic as input, D eventually halts, and outputs 1 iff ψ is valid and 0 otherwise. Then we could solve the halting problem as follows. We construct a Turing machine E that, given as input the number e of Turing machine M_e and input w, computes the corresponding sentence $\tau(M_e, w) \to \alpha(M_e, w)$ and halts, scanning the leftmost square on the tape. The machine $E \frown D$ would then, given input e and w, first compute $\tau(M_e, w) \to \alpha(M_e, w)$ and then run the decision problem machine D on that input. D halts with output 1 iff $\tau(M_e, w) \to \alpha(M_e, w)$ is valid and outputs 0

otherwise. By Lemma 15.13 and Lemma 15.12, $\tau(M_e, w) \to \alpha(M_e, w)$ is valid iff M_e halts on input w. Thus, $E \frown D$, given input e and w halts with output 1 iff M_e halts on input w and halts with output 0 otherwise. In other words, $E \frown D$ would solve the halting problem. But we know, by Theorem 15.6, that no such Turing machine can exist. □

Problems

Problem 15.1. The Three Halting (3-Halt) problem is the problem of giving a decision procedure to determine whether or not an arbitrarily chosen Turing Machine halts for an input of three strokes on an otherwise blank tape. Prove that the 3-Halt problem is unsolvable.

Problem 15.2. Show that if the halting problem is solvable for Turing machine and input pairs M_e and n where $e \neq n$, then it is also solvable for the cases where $e = n$.

Problem 15.3. We proved that the halting problem is unsolvable if the input is a number e, which identifies a Turing machine M_e via an enumaration of all Turing machines. What if we allow the description of Turing machines from section 15.2 directly as input? (This would require a larger alphabet of course.) Can there be a Turing machine which decides the halting problem but takes as input descriptions of Turing machines rather than indices? Explain why or why not.

Problem 15.4. Prove Proposition 15.8. (Hint: use induction on $k - m$).

Problem 15.5. Complete case (3) of the proof of Lemma 15.11.

Problem 15.6. Give a derivation of $S_{\sigma_i}(\bar{i}, \overline{n}')$ from $S_{\sigma_i}(\bar{i}, \overline{n})$ and $\varphi(m, n)$ (assuming $i \neq m$, i.e., either $i < m$ or $m < i$).

Problem 15.7. Give a derivation of $\forall x\,(\overline{k}' < x \to S_0(x, \overline{n}'))$ from $\forall x\,(\overline{k} < x \to S_0(x, \overline{n}'))$, $\forall x\, x < x'$, and $\forall x \forall y \forall z\,((x < y \wedge y < z) \to x < z)$.)

Chapter 16

Recursive Functions

16.1 Introduction

In order to develop a mathematical theory of computability, one has to, first of all, develop a *model* of computability. We now think of computability as the kind of thing that computers do, and computers work with symbols. But at the beginning of the development of theories of computability, the paradigmatic example of computation was *numerical* computation. Mathematicians were always interested in number-theoretic functions, i.e., functions $f : \mathbb{N}^n \to \mathbb{N}$ that can be computed. So it is not surprising that at the beginning of the theory of computability, it was such functions that were studied. The most familiar examples of computable numerical functions, such as addition, multiplication, exponentiation (of natural numbers) share an interesting feature: they can be defined *recursively*. It is thus quite natural to attempt a general definition of *computable function* on the basis of recursive definitions. Among the many possible ways to define number-theoretic functions recursively, one particulalry simple pattern of definition here becomes central: so-called *primitive recursion*.

In addition to computable functions, we might be interested in computable sets and relations. A set is computable if we can compute the answer to whether or not a given number is an element of the set, and a relation is computable iff we can compute whether or not a tuple $\langle n_1, \ldots, n_k \rangle$ is an element of the relation. By considering the *characteristic function* of a set or relation, discussion of computable sets and relations can be subsumed under that of computable functions. Thus we can define primitive recursive relations as well, e.g., the relation "n evenly divides m" is a primitive recursive relation.

Primitive recursive functions—those that can be defined using just primitive recursion—are not, however, the only computable number-theoretic functions. Many generalizations of primitive recursion have been considered, but the most powerful and widely-accepted additional way of computing functions is by unbounded search. This leads to the definition of *partial 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.

16.2 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 $f\colon \mathbb{N} \to \mathbb{N}$ that makes use of this fact is this: (a) specify what the value of f is for argument 0, and (b) also specify how to, given the value of $f(x)$, compute the value of $f(x+1)$. For (a) tells us directly what $f(0)$ is, so f is defined for 0. Now, using the instruction given by (b) for $x = 0$, we can compute $f(1) = f(0+1)$ from $f(0)$. Using the same instructions for $x = 1$, we compute $f(2) = f(1+1)$ from $f(1)$, and so on. For every natural number x, we'll eventually reach the step where we define $f(x)$ from $f(x+1)$, and so $f(x)$ 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. Successively 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 d that meets these two criteria. A pair of equations like these is called a *definition by primitive recursion* of the function d. It is so-called because we define f "recursively," i.e., the definition, specifically the second equation, involves f itself on the right-hand-side. It is "primitive" because in defining $f(x+1)$ we only use the value $f(x)$, 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 $\text{add}(x, y)$ to define $\text{add}(x, y + 1)$. 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:

$$\begin{aligned}\text{mult}(x, 0) &= 0\\ \text{mult}(x, y + 1) &= \text{add}(\text{mult}(x, y), x)\end{aligned}$$

This is a primitive recursive definition of a function mult by applying the function add to both the preceding value $\text{mult}(x, y)$ and the first argument x. It also defines the function $\text{mult}(x, y)$ for all arguments x and y. For instance, $\text{mult}(2, 3)$ is determined by successively computing $\text{mult}(2, 0)$, $\text{mult}(2, 1)$, $\text{mult}(2, 2)$, and $\text{mult}(2, 3)$:

$$\begin{aligned}&\text{mult}(2, 0) = 0\\ &\text{mult}(2, 1) = \text{mult}(2, 0 + 1) = \text{add}(\text{mult}(2, 0), 2) = \text{add}(0, 2) = 2\\ &\text{mult}(2, 2) = \text{mult}(2, 1 + 1) = \text{add}(\text{mult}(2, 1), 2) = \text{add}(2, 2) = 4\\ &\text{mult}(2, 3) = \text{mult}(2, 2 + 1) = \text{add}(\text{mult}(2, 2), 2) = \text{add}(4, 2) = 6\end{aligned}$$

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 f. 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, …, 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 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 $k = 0$ 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):

$$\begin{aligned}\text{mult}(x_0, 0) &= f(x_0) = 0\\ \text{mult}(x_0, y + 1) &= g(x_0, y, \text{mult}(x_0, y)) = \text{add}(\text{mult}(x_0, y), x_0)\end{aligned}$$

16.3 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:

$$P^n_i(x_0, \ldots, x_{n-1}) = x_i$$

The functions P^k_i are called *projection* functions: P^n_i is an n-place function. Then g can be defined by

$$g(x, y, z) = \text{succ}(P^3_2).$$

Here the role of f is played by the 1-place function succ, so $k = 1$. And we have one 3-place function P^3_2 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

$$h(x_0) = \text{add}(P^1_0(x_0), P^1_0(x_0)).$$

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.

16.4 Primitive Recursion Functions

Let us record again how we can define new functions from existing ones using primitive recursion and composition.

Definition 16.1. 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 16.2. 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 $\mathrm{zero}(x) = 0$.

Definition 16.3. The set of primitive recursive functions is the set of functions from $\mathbb{N}^n$ to $\mathbb{N}$, defined inductively by the following clauses:

1. zero is primitive recursive.
2. succ is primitive recursive.
3. Each projection function P^n_i 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^n_j, 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 16.4. *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 16.1:

$$\begin{aligned}\mathrm{add}(x_0, 0) &= f(x_0) = x_0\\ \mathrm{add}(x_0, y+1) &= g(x_0, y, \mathrm{add}(x_0, y)) = \mathrm{succ}(\mathrm{add}(x_0, y))\end{aligned}$$

So add is primitive recursive provided f and g are as well. $f(x_0) = x_0 = P^1_0(x_0)$, 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

$$g(x_0, y, z) = \mathrm{succ}(P^3_2(x_0, y, z))$$

Since succ and P^3_2 count as primitive recursive functions, g does as well, since it can be defined by composition from primitive recursive functions. □

Proposition 16.5. *The multiplication function* $\mathrm{mult}(x, y) = x \cdot y$ *is primitive recursive.*

Proof. Exercise. □

Example 16.6. 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 16.1, 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: $f(x_0) = \mathrm{succ}(\mathrm{zero}(x_0))$. The function g can be defined by composition from $g'(z) = 2 \cdot z$ and projections:

$$g(x_0, y, z) = g'(P^3_2(x_0, y, z))$$

and g' in turn can be defined by composition as

$$g'(z) = \mathrm{mult}(g''(z), P^1_0(z))$$

and

$$g''(z) = \mathrm{succ}(f(z)),$$

where f is as above: $f(z) = \mathrm{succ}(\mathrm{zero}(z))$. Now that we have h' we can use composition again to let $h(y) = h'(P^1_0(y), P^1_0(y))$. This shows that h can be defined from the basic functions using a sequence of compositions and primitive recursions, so h is primitive recursive.

16.5 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 f is defined by composition from a k-place function h and n-place functions $g_0, \ldots, g_{k-1}$, and we have assigned notations $H, G_0, \ldots, G_{k-1}$ to the latter functions. Then, using a new symbol $\mathrm{Comp}_{k,n}$, we can denote the function f by $\mathrm{Comp}_{k,n}[H, G_0, \ldots, G_{k-1}]$. For the functions defined by primitive recursion, we can use analogous notations of the form $\mathrm{Rec}_k[G, H]$, where $k+1$ is the arity of the function being defined. With this setup, we can denote the addition function by

$$\mathrm{Rec}_2[P_0^1, \mathrm{Comp}_{1,3}[\mathrm{succ}, P_2^3]].$$

Having these notations sometimes proves useful.

16.6 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.

16.7 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^1_0. The constant functions $\text{const}_n(x) = n$ 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 $\text{const}_n(x)$ and multiplication as $f(x) = \text{mult}(\text{const}_2(x), P^1_0(x))$. We'll make use of this trick from now on.

Proposition 16.7. *The exponentiation function* $\exp(x, y) = x^y$ *is primitive recursive.*

Proof. We can define exp primitive recursively as

$$\begin{aligned} \exp(x, 0) &= 1 \\ \exp(x, y + 1) &= \text{mult}(x, \exp(x, y)). \end{aligned}$$

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

$$\begin{aligned} f(x) &= \text{succ}(\text{zero}(x)) = 1 \\ g(x, y, z) &= \text{mult}(P^3_0(x, y, z), P^3_2(x, y, z)) = x \cdot z \end{aligned}$$

and so f and g are defined from primitive recursive functions by composition. □

Proposition 16.8. *The predecessor function* $\text{pred}(y)$ *defined by*

$$\text{pred}(y) = \begin{cases} 0 & \text{if } y = 0 \\ y - 1 & \text{otherwise} \end{cases}$$

is primitive recursive.

Proof. Note that

$$\begin{aligned} \text{pred}(0) &= 0 \text{ and} \\ \text{pred}(y + 1) &= y. \end{aligned}$$

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 $\text{pred}(y)$ in the definition of $\text{pred}(y + 1)$. But we can first define $\text{pred}'(x, y)$ by

$$\begin{aligned} \text{pred}'(x, 0) &= \text{zero}(x) = 0, \\ \text{pred}'(x, y + 1) &= P^3_1(x, y, \text{pred}'(x, y)) = y. \end{aligned}$$

and then define pred from it by composition, e.g., as $\text{pred}(x) = \text{pred}'(\text{zero}(x), P^1_0(x))$.□

Proposition 16.9. *The factorial function* $\text{fac}(x) = x! = 1 \cdot 2 \cdot 3 \cdot \cdots \cdot x$ *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

$$\begin{aligned}h(x,0) &= \mathrm{const}_1(x)\\ h(x,y) &= g(x,y,h(x,y))\end{aligned}$$

where $g(x,y,z) = \mathrm{mult}(P^3_2(x,y,z), \mathrm{succ}(P^3_1(x,y,z)))$ and then let

$$\mathrm{fac}(y) = h(P^1_0(y), P^1_0(y))$$

From now on we'll be a bit more laissez-faire and not give the official definitions by composition and primitive recursion. □

Proposition 16.10. *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:

$$\begin{aligned}x \dot{-} 0 &= x\\ x \dot{-} (y+1) &= \mathrm{pred}(x \dot{-} y)\end{aligned}$$

□

Proposition 16.11. *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 16.12. *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 16.13. *The minimum of x and y, $\min(x,y)$, is primitive recursive.*

Proof. Exercise. □

Proposition 16.14. *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

$$\begin{aligned} g(\vec{x}, 0) &= f(\vec{x}, 0) \\ g(\vec{x}, y+1) &= g(\vec{x}, y) + f(\vec{x}, y+1). \end{aligned}$$

□

16.8 Primitive Recursive Relations

Definition 16.15. 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 $\text{IsZero}(x)$, which holds if and only if $x = 0$, corresponds to the function χ_{IsZero}, defined using primitive recursion by

$$\chi_{\text{IsZero}}(0) = 1, \quad \chi_{\text{IsZero}}(x+1) = 0.$$

It should be clear that one can compose relations with other primitive recursive functions. So the following are also primitive recursive:

1. The equality relation, $x = y$, defined by $\text{IsZero}(|x - y|)$
2. The less-than relation, $x \leq y$, defined by $\text{IsZero}(x \dot{-} y)$

Proposition 16.16. *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, $\chi_{P \vee Q}(\vec{x}) = \max(\chi_P(\vec{x}), \chi_Q(\vec{x}))$ and $\chi_{P \to Q}(\vec{x}) = \max(1 \dot{-} \chi_P(\vec{x}), \chi_Q(\vec{x}))$.□

Proposition 16.17. *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 $(\forall z < y)\ R(\vec{x}, z)$ 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 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

$$\begin{aligned} \chi_P(\vec{x}, 0) &= 1 \\ \chi_P(\vec{x}, y+1) &= \min(\chi_P(\vec{x}, y), \chi_R(\vec{x}, y))). \end{aligned}$$

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 $(\exists x \leq y)\ \ldots x \ldots$ is equivalent to $(\exists x < y + 1)\ \ldots x \ldots$. □

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

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

One can use this to justify definitions of primitive recursive functions by cases from primitive recursive relations:

Proposition 18. *If $g_0(\vec{x}), \ldots, g_m(\vec{x})$ are primitive recursive functions, and $R_0(\vec{x}), \ldots, R_{m-1}(\vec{x})$ 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. □

16.9 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 16.19. *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: (a) There is a $z < y$ such that $R(\vec{x}, z)$, in which case $m_R(\vec{x}, y + 1) = m_R(\vec{x}, y)$. (b) There is no such $z < y$ but $R(\vec{x}, y)$ holds, then $m_R(\vec{x}, y + 1) = y$. (c) There is no $z < y + 1$ such that $R(\vec{x}, z)$, then $m_R(\vec{z}, y + 1) = y + 1$. So,

$$m_R(\vec{x}, 0) = 0$$

$$m_R(\vec{x}, y + 1) = \begin{cases} m_R(\vec{x}, y) & \text{if } m_R(\vec{x}, y) \neq y \\ y & \text{if } m_R(\vec{x}, y) = y \text{ and } R(\vec{x}, y) \\ y + 1 & \text{otherwise.} \end{cases}$$

Note that there is a $z < y$ such that $R(\vec{x}, z)$ iff $m_R(\vec{x}, y) \neq y$. □

16.10 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 Prime(x), which holds iff x is prime, can be defined by

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

and is thus primitive recursive.

The primes are 2, 3, 5, 7, 11, etc. Consider the function $p(x)$ which returns the xth prime in that sequence, i.e., $p(0) = 2$, $p(1) = 3$, $p(2) = 5$, etc. (For convenience we will often write $p(x)$ as p_x ($p_0 = 2$, $p_1 = 3$, etc.)

If we had a function nextPrime(x), which returns the first prime number larger than x, p can be easily defined using primitive recursion:

$$\begin{aligned} p(0) &= 2 \\ p(x+1) &= \text{nextPrime}(p(x)) \end{aligned}$$

Since nextPrime(x) is the least y such that $y > x$ and y is prime, it can be easily computed by unbounded search. But it can also be defined by bounded minimization, thanks to a result due to Euclid: there is always a prime number between x and $x! + 1$.

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

This shows, that 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 $\leq x! + 1$.)

16.11 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 16.20. *The function* $\text{len}(s)$*, 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

$$\text{len}(s) = \begin{cases} 0 & \text{if } s = 0 \text{ or } s = 1 \\ 1 + (\min\, i < s)\, R(i, s) & \text{otherwise} \end{cases}$$

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 16.21. *The function* $\text{append}(s, a)$*, which returns the result of appending* a *to the sequence* s*, is primitive recursive.*

Proof. append can be defined by:

$$\text{append}(s, a) = \begin{cases} 2^{a+1} & \text{if } s = 0 \text{ or } s = 1 \\ s \cdot p_{\text{len}(s)}^{a+1} & \text{otherwise.} \end{cases}$$ fl

Proposition 16.22. *The function* $\text{element}(s, i)$*, 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:

$$\text{element}(s, i) = \begin{cases} 0 & \text{if } i \geq \text{len}(s) \\ (\min\, a < s)\, (p_i^{a+2} \nmid s) & \text{otherwise.} \end{cases}$$ fl

Instead of using the official names for the functions defined above, we introduce a more compact notation. We will use $(s)_i$ instead of $\text{element}(s, i)$, and $\langle s_0, \ldots, s_k\rangle$ to abbreviate

$$\text{append}(\text{append}(\ldots \text{append}(\Lambda, s_0) \ldots), s_k).$$

Note that if s has length k, the elements of s are $(s)_0, \ldots, (s)_{k-1}$.

Proposition 16.23. *The function* $\text{concat}(s, t)$*, 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 $\text{hconcat}(s, t, n)$ 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 $\text{concat}(s, t)$.

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:

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

Proposition 16.24. *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. □

16.12 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 16.25. *The function* $\mathrm{SubtreeSeq}(t)$*, 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 $\mathrm{ISubtrees}(t) = \mathrm{subseq}(t, 1, (t)_0)$ is primitive recursive and returns the codes of the immediate subtrees of a tree t. Now we can define a helper function $\mathrm{hSubtreeSeq}(t, n)$ 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

$$\begin{aligned} \mathrm{hSubtreeSeq}(t, 0) &= \langle t \rangle \\ \mathrm{hSubtreeSeq}(t, n+1) &= \mathrm{hSubtreeSeq}(t, n) \frown h(\mathrm{hSubtree}(t, n)). \end{aligned}$$

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 $\mathrm{hSubtreeSeq}(t, t)$. □

16.13 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.)

16.14 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. Ackermann's function is essentially the function $G(x) = g_x(x)$, and one can show that this grows faster than any primitive recursive function.

Let us return to the issue of enumerating the primitive recursive functions. Remember that we have assigned symbolic notations to each primitive recursive function; so it suffices to enumerate notations. We can assign a natural number $\#(F)$ to each notation F, recursively, as follows:

$$\begin{aligned}
\#(0) &= \langle 0 \rangle \\
\#(S) &= \langle 1 \rangle \\
\#(P_i^n) &= \langle 2, n, i \rangle \\
\#(\mathrm{Comp}_{k,l}[H, G_0, \ldots, G_{k-1}]) &= \langle 3, k, l, \#(H), \#(G_0), \ldots, \#(G_{k-1}) \rangle \\
\#(\mathrm{Rec}_l[G, H]) &= \langle 4, l, \#(G), \#(H) \rangle
\end{aligned}$$

Here 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.

16.15 Partial Recursive Functions

To motivate the definition of the recursive functions, note that our proof that there are computable functions that are not primitive recursive actually establishes much more. The argument was simple: all we used was the fact was that it is possible to enumerate functions $f_0, f_1, \ldots$ such that, as a function of x and y, $f_x(y)$ is computable.

So the argument applies to *any class of functions that can be enumerated in such a way*. This puts us in a bind: we would like to describe the computable functions explicitly; but any explicit description of a collection of computable functions cannot be exhaustive!

The way out is to allow *partial* functions to come into play. We will see that it *is* possible to enumerate the partial computable functions. In fact, we already pretty much know that this is the case, since it is possible to enumerate Turing machines in a systematic way. We will come back to our diagonal argument later, and explore why it does not go through when partial functions are included.

The question is now this: what do we need to add to the primitive recursive functions to obtain all the partial recursive functions? We need to do two things:

1. Modify our definition of the primitive recursive functions to allow for partial functions as well.

2. *Add* something to the definition, so that some new partial functions are included.

The first is easy. As before, we will start with zero, successor, and projections, and close under composition and primitive recursion. The only difference is that we have to modify the definitions of composition and primitive recursion to allow for the possibility that some of the terms in the definition are not defined. If f and g are partial functions, we will write $f(x) \downarrow$ to mean that f is defined at x, i.e., x is in the domain of f; and $f(x) \uparrow$ to mean the opposite, i.e., that f is not defined at x. We will use $f(x) \simeq g(x)$ to mean that either $f(x)$ and $g(x)$ are both undefined, or they are both defined and equal. We will use these notations for more complicated terms as well. We will adopt the convention that if h and $g_0, \ldots, g_k$ all are partial functions, then

$$h(g_0(\vec{x}), \ldots, g_k(\vec{x}))$$

is defined if and only if each g_i is defined at $\vec{x}$, and h is defined at $g_0(\vec{x}), \ldots, g_k(\vec{x})$. With this understanding, the definitions of composition and primitive recursion for partial functions is just as above, except that we have to replace "=" by "$\simeq$".

What we will add to the definition of the primitive recursive functions to obtain partial functions is the *unbounded search operator*. If $f(x, \vec{z})$ is any partial function on the natural numbers, define $\mu x\ f(x, \vec{z})$ to be

> the least x such that $f(0, \vec{z}), f(1, \vec{z}), \ldots, f(x, \vec{z})$ are all defined, and $f(x, \vec{z}) = 0$, if such an x exists

with the understanding that $\mu x\ f(x, \vec{z})$ is undefined otherwise. This defines $\mu x\ f(x, \vec{z})$ uniquely.

Note that our definition makes no reference to Turing machines, or algorithms, or any specific computational model. But like composition and primitive recursion, there is an operational, computational intuition behind unbounded search. When it comes to the computability of a partial function, arguments where the function is undefined correspond to inputs for which the computation does not halt. The procedure for computing $\mu x\ f(x, \vec{z})$ will amount to this: compute $f(0, \vec{z}), f(1, \vec{z}), f(2, \vec{z})$ until a value of 0 is returned. If any of the intermediate computations do not halt, however, neither does the computation of $\mu x\ f(x, \vec{z})$.

If $R(x, \vec{z})$ is any relation, $\mu x\ R(x, \vec{z})$ is defined to be $\mu x\ (1 \dot{-} \chi_R(x, \vec{z}))$. In other words, $\mu x\ R(x, \vec{z})$ returns the least value of x such that $R(x, \vec{z})$ holds. So, if $f(x, \vec{z})$ is a total function, $\mu x\ f(x, \vec{z})$ is the same as $\mu x\ (f(x, \vec{z}) = 0)$. But note that our

original definition is more general, since it allows for the possibility that $f(x, \vec{z})$ is not everywhere defined (whereas, in contrast, the characteristic function of a relation is always total).

Definition 16.26. 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 16.27. 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.

16.16 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 16.28. 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 16.28 and Definition 16.27 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 16.28 is *less* general than Definition 16.27. 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 16.1. Prove Proposition 16.5 by showing that the primitive recursive definition of mult is can be put into the form required by Definition 16.1 and showing that the corresponding functions f and g are primitive recursive.

Problem 16.2. Give the complete primitive recursive notation for mult.

Problem 16.3. Prove Proposition 16.13.

Problem 16.4. Show that

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

is primitive recursive.

Problem 16.5. 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 16.6. 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 16.7. Define integer division $d(x,y)$ using bounded minimization.

Problem 16.8. Show that there is a primitive recursive function $\mathrm{sconcat}(s)$ with the property that

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

Problem 16.9. Show that there is a primitive recursive function $\mathrm{tail}(s)$ with the property that

$$\mathrm{tail}(\Lambda) = 0 \text{ and}$$
$$\mathrm{tail}(\langle s_0, \ldots, s_k \rangle) = \langle s_1, \ldots, s_k \rangle.$$

Problem 16.10. Prove Proposition 16.24.

Problem 16.11. The definition of hSubtreeSeq in the proof of Proposition 16.25 in general includes repetitions. Give an alternative definition which guarantees that the code of a subtree occurs only once in the resulting list.

Chapter 17

Arithmetization of Syntax

17.1 Introduction

In order to connect computability and logic, we need a way to talk about the objects of logic (symbols, terms, formulas, derivations), operations on them, and their properties and relations, in a way amenable to computational treatment. We can do this directly, by considering computable functions and relations on symbols, sequences of symbols, and other objects built from them. Since the objects of logical syntax are all finite and built from a countable sets of symbols, this is possible for some models of computation. But other models of computation—such as the recursive functions—-are restricted to numbers, their relations and functions. Moreover, ultimately we also want to be able to deal with syntax within certain theories, specifically, in theories formulated in the language of arithmetic. In these cases it is necessary to *arithmetize* syntax, i.e., to represent syntactic objects, operations on them, and their relations, as numbers, arithmetical functions, and arithmetical relations, respectively. The idea, which goes back to Leibniz, is to assign numbers to syntactic objects.

It is relatively straightforward to assign numbers to symbols as their "codes." Some symbols pose a bit of a challenge, since, e.g., there are infinitely many variables, and even infinitely many function symbols of each arity n. But of course it's possible to assign numbers to symbols systematically in such a way that, say, v_2 and v_3 are assigned different codes. Sequences of symbols (such as terms and formulas) 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 formulas, such as derivations. This extended coding is called "Gödel numbering." Every term, formula, and derivation is assigned a Gödel number.

By coding sequences of symbols as sequences of their codes, and by chosing a system of coding sequences that can be dealt with using computable functions, we can then also deal with Gödel numbers 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 formula φ, we immediately see that the length of a formula and the (code of the) i-th symbol in a formula can also be computed from the Gödel number of φ. It is a bit harder to prove that, e.g., the property of being the Gödel number of a correctly formed term or of a correct derivation is primitive recursive. It is

nevertheless possible, because the sequences of interest (terms, formulas, derivations) are inductively defined.

As an example, consider the operation of substitution. If φ is a formula, x a variable, and t a term, then $\varphi[t/x]$ is the result of replacing every free occurrence of x in φ by t. Now suppose we have assigned Gödel numbers to φ, x, t—say, k, l, and m, respectively. The same scheme assigns a Gödel number to $\varphi[t/x]$, say, n. This mapping—of k, l, and m to n—is the arithmetical analog of the substitution operation. When the substitution operation maps φ, x, t to $\varphi[t/x]$, the arithmetized substitution functions maps the Gödel numbers k, l, m to the Gödel number n. We will see that this function is primitive recursive.

Arithmetization of syntax is not just of abstract interest, although it was originally a non-trivial insight that languages like the language of arithmetic, which do not come with mechanisms for "talking about" languages can, after all, formalize complex properties of expressions. It is then just a small step to ask what a theory in this language, such as Peano arithmetic, can *prove* about its own language (including, e.g., whether sentences are provable or true). This leads us to the famous limitative theorems of Gödel (about unprovability) and Tarski (the undefinability of truth). But the trick of arithmetizing syntax is also important in order to prove some important results in computability theory, e.g., about the computational prower of theories or the relationship between different models of computability. The arithmetization of syntax serves as a model for arithmetizing other objects and properties. For instance, it is similarly possible to arithmetize configurations and computations (say, of Turing machines). This makes it possible to simulate computations in one model (e.g., Turing machines) in another (e.g., recursive functions).

17.2 Coding Symbols

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

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

together with countable sets of variables and constant symbols, and countable sets of function symbols and predicate symbols of arbitrary arity. We can assign *codes* to each of these symbols in such a way that every symbol is assigned a unique number as its code, and no two different symbols are assigned the same number. We know that this is possible since the set of all symbols is countable and so there is a bijection between it and the set of natural numbers. But we want to make sure that we can recover the symbol (as well as some information about it, e.g., the arity of a function symbol) from its code in a computable way. There are many possible ways of doing this, of course. Here is one such way, which uses primitive recursive functions. (Recall that $\langle n_0, \ldots, n_k \rangle$ is the number coding the sequence of numbers $n_0, \ldots, n_k$.)

Definition 17.1. If s is a symbol of $\mathcal{L}$, let the *symbol code* c_s be defined as follows:

1. If s is among the logical symbols, c_s is given by the following table:

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

2. If s is the i-th variable v_i, then $\mathrm{c}_s = \langle 1, i\rangle$.
3. If s is the i-th constant symbol c_i, 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 17.2. *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 17.3. If $s_0, \ldots, s_{n-1}$ is a sequence of symbols, its *Gödel number* is $\langle \mathrm{c}_{s_0}, \ldots, \mathrm{c}_{s_{n-1}}\rangle$.

Note that *codes* and *Gödel numbers* are different things. For instance, the variable v_5 has a code $\mathrm{c}_{v_5} = \langle 1, 5\rangle = 2^2 \cdot 3^6$. But the variable v_5 considered as a term is also a sequence of symbols (of length 1). The *Gödel number* ${}^{\ulcorner}v_5{}^{\urcorner}$ of the *term* v_5 is $\langle \mathrm{c}_{v_5}\rangle = 2^{\mathrm{c}_{v_5}+1} = 2^{2^2\cdot 3^6+1}$.

Example 17.4. Recall that if $k_0, \ldots, k_{n-1}$ is a sequence of numbers, then the code of the sequence $\langle k_0, \ldots, k_{n-1}\rangle$ in the power-of-primes coding is

$$2^{k_0+1} \cdot 3^{k_1+1} \cdot \ldots \cdot p_{n-1}^{k_{n-1}},$$

where p_i is the i-th prime (starting with $p_0 = 2$). So for instance, the formula $v_0 = 0$, or, more explicitly, $=(v_0, c_0)$, has the Gödel number

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

Here, $\mathrm{c}_=$ is $\langle 0, 7\rangle = 2^{0+1} \cdot 3^{7=1}$, c_{v_0} is $\langle 1, 0\rangle = 2^{1+1} \cdot 3^{0+1}$, etc. So ${}^{\ulcorner}=(v_0, c_0){}^{\urcorner}$ is

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

17.3 Coding Terms

A term is simply a certain kind of sequence of symbols: it is built up inductively from constants and variables according to the formation rules for terms. Since sequences of symbols can be coded as numbers—using a coding scheme for the symbols plus a way to code sequences of numbers—assigning Gödel numbers to terms is not difficult. The challenge is rather to show that the property a number has if it is the Gödel number of a correctly formed term is computable, or in fact primitive recursive.

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 $< x$:

$$\begin{aligned} \mathrm{Var}(x) &\Leftrightarrow (\exists i < x)\ x = \langle\langle 1, i\rangle\rangle \\ \mathrm{Const}(x) &\Leftrightarrow (\exists i < x)\ x = \langle\langle 2, i\rangle\rangle \end{aligned}$$

Proposition 17.5. *The relations* $\mathrm{Term}(x)$ *and* $\mathrm{ClTerm}(x)$ *which hold iff* x *is the Gödel number of a term or a closed term, respectively, are primitive recursive.*

Proof. A sequence of symbols s is a term iff there is a sequence $s_0, \ldots, s_{k-1} = s$ of terms which records how the term s was formed from constant symbols and variables according to the formation rules for terms. To express that such a putative formation sequence follows the formation rules it has to be the case that, for each $i < k$, either

1. s_i is a variable v_j, or
2. s_i is a constant symbol c_j, or
3. s_i is built from n terms $t_1, \ldots, t_n$ occurring prior to place i using an n-place function symbol f^n_j.

To show that the corresponding relation on Gödel numbers is primitive recursive, we have to express this condition primitive recursively, i.e., using primitive recursive functions, relations, and bounded quantification.

Suppose y is the number that codes the sequence $s_0, \ldots, s_{k-1}$, i.e., $y = \langle {}^{\ulcorner}s_0{}^{\urcorner}, \ldots, {}^{\ulcorner}s_{k-1}{}^{\urcorner}\rangle$. 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
$$(y)_i = {}^{\ulcorner}f^n_j({}^{\urcorner} \frown \mathrm{flatten}(z) \frown {}^{\ulcorner})^{\urcorner},$$

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 $\leq x^{\mathrm{len}(x)}$.

For ClTerm, simply leave out the clause for variables. □

Proposition 17.6. *The function* $\mathrm{num}(n) = {}^{\ulcorner}\overline{n}{}^{\urcorner}$ *is primitive recursive.*

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

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

□

17.4 Coding Formulas

Proposition 17.7. *The relation* $\mathrm{Atom}(x)$ *which holds iff* x *is the Gödel number of an atomic formula, is primitive recursive.*

Proof. The number x is the Gödel number of an atomic formula iff one of the following holds:

1. There are n, $j < x$, and $z < x$ such that for each $i < n$, $\mathrm{Term}((z)_i)$ and $x =$
$$^{\ulcorner}P^n_j(^{\ulcorner} \frown \mathrm{flatten}(z) \frown {}^{\urcorner})^{\urcorner}.$$
2. There are $z_1, z_2 < x$ such that $\mathrm{Term}(z_1)$, $\mathrm{Term}(z_2)$, and $x =$
$$^{\ulcorner}{=}(^{\ulcorner} \frown z_1 \frown {}^{\urcorner},^{\ulcorner} \frown z_2 \frown {}^{\urcorner})^{\urcorner}.$$
3. $x = {}^{\ulcorner}\bot^{\urcorner}$. □

Proposition 17.8. *The relation* $\mathrm{Frm}(x)$ *which holds iff* x *is the Gödel number of a formula is primitive recursive.*

Proof. A sequence of symbols s is a formula iff there is formation sequence $s_0, \ldots, s_{k-1} = s$ of formula which records how s was formed from atomic formulas according to the formation rules. The code for each s_i (and indeed of the code of the sequence $\langle s_0, \ldots, s_{k-1}\rangle$) is less than the code x of s. □

Proposition 17.9. *The relation* $\mathrm{FreeOcc}(x, z, i)$*, which holds iff the* i*-th symbol of the formula with Gödel number* x *is a free occurrence of the variable with Gödel number* z*, is primitive recursive.*

Proof. Exercise. □

Proposition 17.10. *The property* $\mathrm{Sent}(x)$ *which holds iff* x *is the Gödel number of a sentence is primitive recursive.*

Proof. A sentence is a formula without free occurrences of variables. So $\mathrm{Sent}(x)$ holds iff

$$(\forall i < \mathrm{len}(x))\ (\forall z < x)$$
$$((\exists j < z)\ z = {}^{\ulcorner}v_j{}^{\urcorner} \rightarrow \neg\mathrm{FreeOcc}(x, z, i)).$$ □

17.5 Substitution

Recall that substitution is the operation of replacing all free occurrences of a variable u in a formula φ by a term t, written $\varphi[t/u]$. This operation, when carried out on Gödel numbers of variables, formulas, and terms, is primitive recursive.

Proposition 17.11. *There is a primitive recursive function* $\mathrm{Subst}(x, y, z)$ *with the property that*
$$\mathrm{Subst}({}^{\ulcorner}\varphi^{\urcorner}, {}^{\ulcorner}t^{\urcorner}, {}^{\ulcorner}u^{\urcorner}) = {}^{\ulcorner}\varphi[t/u]^{\urcorner}$$

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 17.12. *The relation* $\mathrm{FreeFor}(x, y, z)$*, which holds iff the term with Gödel number* y *is free for the variable with Gödel number* z *in the formula with Gödel number* x*, is primitive recursive.*

Proof. Exercise. □

17.6 Derivations in Natural Deduction

In order to arithmetize derivations, we must represent derivations as numbers. Since derivations are trees of formulas where each inference carries one or two labels, a recursive representation is the most obvious approach: we represent a derivation as a tuple, the components of which are the number of immediate sub-derivations leading to the premises of the last inference, the representations of these sub-derivations, and the end-formula, the discharge label of the last inference, and a number indicating the type of the last inference.

Definition 17.13. If δ is a derivation in natural deduction, then ${}^{\ulcorner}\delta^{\urcorner}$ is defined inductively as follows:

1. If δ consists only of the assumption φ, then ${}^{\ulcorner}\delta^{\urcorner}$ is $\langle 0, {}^{\ulcorner}\varphi^{\urcorner}, n\rangle$. The number n is 0 if it is an undischarged assumption, and the numerical label otherwise.
2. If δ ends in an inference with one, two, or three premises, then ${}^{\ulcorner}\delta^{\urcorner}$ is

   $$\begin{aligned}&\langle 1, {}^{\ulcorner}\delta_1{}^{\urcorner}, {}^{\ulcorner}\varphi^{\urcorner}, n, k\rangle, \\ &\langle 2, {}^{\ulcorner}\delta_1{}^{\urcorner}, {}^{\ulcorner}\delta_2{}^{\urcorner}, {}^{\ulcorner}\varphi^{\urcorner}, n, k\rangle, \text{ or} \\ &\langle 3, {}^{\ulcorner}\delta_1{}^{\urcorner}, {}^{\ulcorner}\delta_2{}^{\urcorner}, {}^{\ulcorner}\delta_3{}^{\urcorner}, {}^{\ulcorner}\varphi^{\urcorner}, n, k\rangle,\end{aligned}$$

   respectively. Here δ_1, δ_2, δ_3 are the sub-derivations ending in the premise(s) of the last inference in δ, φ is the conclusion of the last inference in δ, n is the discharge label of the last inference (0 if the inference does not discharge any assumptions), and k is given by the following table according to which rule was used in the last inference.

Rule:	$\wedge$I	$\wedge$E	$\vee$I	$\vee$E
k:	1	2	3	4

Rule:	$\rightarrow$I	$\rightarrow$E	$\neg$I	$\neg$E
k:	5	6	7	8

Rule:	$\bot$E	RAA	$\forall$I	$\forall$E
k:	9	10	11	12

Rule:	$\exists$I	$\exists$E	=I	=E
k:	13	14	15	16

Example 17.14. Consider the very simple derivation

$$\dfrac{\dfrac{[\varphi \wedge \psi]^1}{\varphi}\,\wedge\mathrm{E}}{(\varphi \wedge \psi) \to \varphi}\,{\to}\mathrm{I}_1$$

The Gödel number of the assumption would be $d_0 = \langle 0, {}^{\#}\varphi \wedge \psi^{\#}, 1\rangle$. The Gödel number of the derivation ending in the conclusion of $\wedge$E would be $d_1 = \langle 1, d_0, {}^{\#}\varphi^{\#}, 0, 2\rangle$ (1 since $\wedge$E has one premise, the Gödel number of conclusion φ, 0 because no assumption is discharged, and 2 is the number coding $\wedge$E). The Gödel number of the entire derivation then is $\langle 1, d_1, {}^{\#}((\varphi \wedge \psi) \to \varphi)^{\#}, 1, 5\rangle$, i.e.,

$$\langle 1, \langle 1, \langle 0, {}^{\#}(\varphi \wedge \psi)^{\#}, 1\rangle, {}^{\#}\varphi^{\#}, 0, 2\rangle, {}^{\#}((\varphi \wedge \psi) \to \varphi)^{\#}, 1, 5\rangle.$$

Having settled on a representation of derivations, we must also show that we can manipulate Gödel numbers of such derivations primitive recursively, and express their essential properties and relations. Some operations are simple: e.g., given a Gödel number d of a derivation, $\mathrm{EndFmla}(d) = (d)_{(d)_0+1}$ gives us the Gödel number of its end-formula, $\mathrm{DischargeLabel}(d) = (d)_{(d)_0+2}$ gives us the discharge label and $\mathrm{LastRule}(d) = (d)_{(d)_0+3}$ the number indicating the type of the last inference. Some are much harder. We'll at least sketch how to do this. The goal is to show that the relation "δ is a derivation of φ from Γ" is a primitive recursive relation of the Gödel numbers of δ and φ.

Proposition 17.15. *The following relations are primitive recursive:*

1. *φ occurs as an assumption in δ with label n.*
2. *All assumptions in δ with label n are of the form φ (i.e., we can discharge the assumption φ using label n in δ).*

Proof. We have to show that the corresponding relations between Gödel numbers of formulas and Gödel numbers of derivations are primitive recursive.

1. We want to show that $\mathrm{Assum}(x, d, n)$, which holds if x is the Gödel number of an assumption of the derivation with Gödel number d labelled n, is primitive recursive. This is the case if the derivation with Gödel number $\langle 0, x, n\rangle$ is a sub-derivation of d. Note that the way we code derivations is a special case of the coding of trees introduced in section 16.12, so the primitive recursive function $\mathrm{SubtreeSeq}(d)$ gives a sequence of Gödel numbers of all sub-derivations of d (of length a most d). So we can define

$$\mathrm{Assum}(x, d, n) \Leftrightarrow (\exists i < d)\ (\mathrm{SubtreeSeq}(d))_i = \langle 0, x, n\rangle.$$

2. We want to show that $\mathrm{Discharge}(x, d, n)$, which holds if all assumptions with label n in the derivation with Gödel number d all are the formula with Gödel number x. But this relation holds iff $(\forall y < d)\ (\mathrm{Assum}(y, d, n) \to y = x)$. □

Proposition 17.16. *The property* $\mathrm{Correct}(d)$ *which holds iff the last inference in the derivation δ with Gödel number d is correct, is primitive recursive.*

Proof. Here we have to show that for each rule of inference R the relation $\mathrm{FollowsBy}_R(d)$ is primitive recursive, where $\mathrm{FollowsBy}_R(d)$ holds iff d is the Gödel number of derivation δ, and the end-formula of δ follows by a correct application of R from the immediate sub-derivations of δ.

A simple case is that of the $\wedge$I rule. If δ ends in a correct $\wedge$I inference, it looks like this:

$$\dfrac{\begin{array}{cc}\vdots & \vdots \\ \delta_1 & \delta_2 \\ \vdots & \vdots \\ \varphi & \psi\end{array}}{\varphi \wedge \psi}\ \wedge\mathrm{I}$$

Then the Gödel number d of δ is $\langle 2, d_1, d_2, {}^{\ulcorner}(\varphi \wedge \psi)^{\urcorner}, 0, k\rangle$ where $\mathrm{EndFmla}(d_1) = {}^{\ulcorner}\varphi^{\urcorner}$, $\mathrm{EndFmla}(d_2) = {}^{\ulcorner}B^{\urcorner}$, $n = 0$, and $k = 1$. So we can define $\mathrm{FollowsBy}_{\wedge\mathrm{I}}(d)$ as

$$(d)_0 = 2 \wedge \mathrm{DischargeLabel}(d) = 0 \wedge \mathrm{LastRule}(d) = 1 \wedge \\ \mathrm{EndFmla}(d) = {}^{\ulcorner}({}^{\urcorner} \frown \mathrm{EndFmla}((d)_1) \frown {}^{\ulcorner}\wedge{}^{\urcorner} \frown \mathrm{EndFmla}((d)_2) \frown {}^{\ulcorner})^{\urcorner}.$$

Another simple example if the =I rule. Here the premise is an empty derivation, i.e., $(d)_1 = 0$, and no discharge label, i.e., $n = 0$. However, φ must be of the form $t = t$, for a closed term t. Here, a primitive recursive definition is

$$(d)_0 = 1 \wedge (d)_1 = 0 \wedge \mathrm{DischargeLabel}(d) = 0 \wedge \\ (\exists t < d)\ (\mathrm{ClTerm}(t) \wedge \mathrm{EndFmla}(d) = {}^{\ulcorner}=({}^{\urcorner} \frown t \frown {}^{\ulcorner},{}^{\urcorner} \frown t \frown {}^{\ulcorner})^{\urcorner})$$

For a more complicated example, $\mathrm{FollowsBy}_{\to\mathrm{I}}(d)$ holds iff the end-formula of δ is of the form $(\varphi \to \psi)$, where the end-formula of δ_1 is ψ, and any assumption in δ labelled n is of the form φ. We can express this primitive recursively by

$$(d)_0 = 1 \wedge \\ (\exists a < d)\ (\mathrm{Discharge}(a, (d)_1, \mathrm{DischargeLabel}(d)) \wedge \\ \mathrm{EndFmla}(d) = ({}^{\ulcorner}({}^{\urcorner} \frown a \frown {}^{\ulcorner}\to{}^{\urcorner} \frown \mathrm{EndFmla}((d)_1) \frown {}^{\ulcorner})^{\urcorner}))$$

(Think of a as the Gödel number of φ).

For another example, consider $\exists$I. Here, the last inference in δ is correct iff there is a formula φ, a closed term t and a variable x such that $\varphi[t/x]$ is the end-formula of the derivation δ_1 and $\exists x\, \varphi$ is the conclusion of the last inference. So, $\mathrm{FollowsBy}_{\exists\mathrm{I}}(d)$ holds iff

$$(d)_0 = 1 \wedge \mathrm{DischargeLabel}(d) = 0 \wedge \\ (\exists a < d)\ (\exists x < d)\ (\exists t < d)\ (\mathrm{ClTerm}(t) \wedge \mathrm{Var}(x) \wedge \\ \mathrm{Subst}(a, t, x) = \mathrm{EndFmla}((d)_1) \wedge \mathrm{EndFmla}(d) = ({}^{\ulcorner}\exists{}^{\urcorner} \frown x \frown a)).$$

We then define $\mathrm{Correct}(d)$ as

$$\mathrm{Sent}(\mathrm{EndFmla}(d)) \wedge \\ (\mathrm{LastRule}(d) = 1 \wedge \mathrm{FollowsBy}_{\wedge\mathrm{I}}(d)) \vee \cdots \vee \\ (\mathrm{LastRule}(d) = 16 \wedge \mathrm{FollowsBy}_{=\mathrm{E}}(d)) \vee \\ (\exists n < d)\ (\exists x < d)\ (d = \langle 0, x, n\rangle).$$

The first line ensures that the end-formula of d is a sentence. The last line covers the case where d is just an assumption. □

Proposition 17.17. *The relation* $\mathrm{Deriv}(d)$ *which holds if* d *is the Gödel number of a correct derivation* δ*, is primitive recursive.*

Proof. A derivation δ is correct if every one of its inferences is a correct application of a rule, i.e., if every one of its sub-derivations ends in a correct inference. So, $\mathrm{Deriv}(d)$ iff

$$(\forall i < \mathrm{len}(\mathrm{SubtreeSeq}(d)))\ \mathrm{Correct}((\mathrm{SubtreeSeq}(d))_i) \qquad \text{fl}$$

Proposition 17.18. *The relation* $\mathrm{OpenAssum}(z, d)$ *that holds if* z *is the Gödel number of an undischarged assumption* φ *of the derivation* δ *with Gödel number* d*, is primitive recursive.*

Proof. An occurrence of an assumption is discharged if it occurs with label n in a sub-derivation of δ that ends in a rule with discharge label n. So φ is an undischarged assumption of δ if at least one of its occurrences is not discharged in δ. We must be careful: δ may contain both discharged and undischarged occurrences of φ.

Consider a sequence $\delta_0, \ldots, \delta_k$ where $\delta_0 = d$, δ_k is the assumption $[\varphi]^n$ (for some n), and δ_i is an immediate sub-derivation of δ_{i+1}. If such a sequence exists in which no δ_i ends in an inference with discharge label n, then φ is an undischarged assumption of δ.

The primitive recursive function $\mathrm{SubtreeSeq}(d)$ provides us with a sequence of Gödel numbers of all sub-derivations of δ. Any sequence of Gödel numbers of sub-derivations of δ is a subsequence of it. Being a subsequence of is a primitive recursive relation: $\mathrm{Subseq}(s, s')$ holds iff $(\forall i < \mathrm{len}(s))\ \exists j < \mathrm{len}(s')\ (s)_i = (s)_j$. Being an immediate sub-derivation is as well: $\mathrm{Subderiv}(d, d')$ iff $(\exists j < (d')_0)\ d = (d')_j$. So we can define $\mathrm{OpenAssum}(z, d)$ by

$$\begin{aligned}(\exists s < \mathrm{SubtreeSeq}(d))\ (\mathrm{Subseq}(s, \mathrm{SubtreeSeq}(d)) \wedge (s)_0 = d \wedge \\ (\exists n < d)\ ((s)_{\mathrm{len}(s) \dot{-} 1} = \langle 0, z, n\rangle \wedge \\ (\forall i < (\mathrm{len}(s) \dot{-} 1))\ (\mathrm{Subderiv}((s)_i, (s)_{i+1})] \wedge \\ \mathrm{DischargeLabel}((s)_{i+1}) \neq n))).\end{aligned}$$ □

Proposition 17.19. *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* φ *from undischarged assumptions in* Γ *and* y *is the Gödel number of* φ*" is primitive recursive.*

Proof. Suppose "$y \in \Gamma$" is given by the primitive recursive predicate $R_\Gamma(y)$. We have to show that $\mathrm{Prf}_\Gamma(x, y)$ which holds iff y is the Gödel number of a sentence φ and x is the code of a natural deduction derivation with end formula φ and all undischarged assumptions in Γ is primitive recursive.

By Proposition 17.17, the property $\mathrm{Deriv}(x)$ which holds iff x is the Gödel number of a correct derivation δ in natural deduction is primitive recursive. Thus we can define $\mathrm{Prf}_\Gamma(x, y)$ by

$$\begin{aligned}\mathrm{Prf}_\Gamma(x, y) \Leftrightarrow \mathrm{Deriv}(x) \wedge \mathrm{EndFmla}(x) = y \wedge \\ (\forall z < x)\ (\mathrm{OpenAssum}(z, x) \rightarrow R_\Gamma(z)).\end{aligned}$$ □

Problems

Problem 17.1. 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 17.2. Give a detailed proof of Proposition 17.8 along the lines of the first proof of Proposition 17.5

Problem 17.3. Give a detailed proof of Proposition 17.8 along the lines of the alternate proof of Proposition 17.5

Problem 17.4. Prove Proposition 17.9. You may make use of the fact that any substring of a formula which is a formula is a sub-formula of it.

Problem 17.5. Prove Proposition 17.12

Problem 17.6. Define the following properties as in Proposition 17.16:

1. $\mathrm{FollowsBy}_{\to \mathrm{E}}(d)$,
2. $\mathrm{FollowsBy}_{= \mathrm{E}}(d)$,
3. $\mathrm{FollowsBy}_{\vee \mathrm{E}}(d)$,
4. $\mathrm{FollowsBy}_{\forall \mathrm{I}}(d)$.

For the last one, you will have to also show that you can test primitive recursively if the last inference of the derivation with Gödel number d satisfies the eigenvariable condition, i.e., the eigenvariable a of the $\forall$I inference occurs neither in the end-formula of d nor in an open assumption of d. You may use the primitive recursive predicate OpenAssum from Proposition 17.18 for this.

Chapter 18

Representability in Q

18.1 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 $\{\varphi \mid \mathbf{Q} \vdash \varphi\}$ is not decidable, by reducing the halting problem to it. By the time we are done, we will have proved much stronger things than this.

The language of **Q** is the language of arithmetic; **Q** consists of the following axioms (to be used in conjunction with the other axioms and rules of first-order logic with identity predicate):

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

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**:

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

where $\varphi(x)$ is any formula. If $\varphi(x)$ contains free variables other than x, we add universal quantifiers to the front to bind all of them (so that the corresponding instance of the induction schema is a sentence). For instance, if $\varphi(x, y)$ also contains the variable y free, the corresponding instance is

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

Using instances of the induction schema, one can prove much more from the axioms of **PA** than from those of **Q**. In fact, it takes a good deal of work to find "natural" statements about the natural numbers that can't be proved in Peano arithmetic!

Definition 18.1. A function $f(x_0, \ldots, x_k)$ from the natural numbers to the natural numbers is said to be *representable in* **Q** if there is a formula $\varphi_f(x_0, \ldots, x_k, y)$ such that whenever $f(n_0, \ldots, n_k) = m$, **Q** proves

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

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

Theorem 18.2. *A function is representable in* **Q** *if and only if it is computable.*

There are two directions to proving the theorem. The left-to-right direction is fairly straightforward once arithmetization of syntax is in place. The other direction requires more work. Here is the basic idea: we pick "general recursive" as a way of making "computable" precise, and show that every general recursive function is representable in **Q**. Recall that a function is general recursive if it can be defined from zero, the successor function succ, and the projection functions P^n_i, using composition, primitive recursion, and regular minimization. So one way of showing that every general recursive function is representable in **Q** is to show that the basic functions are representable, and whenever some functions are representable, then so are the functions defined from them using composition, primitive recursion, and regular minimization. In other words, we might show that the basic functions are representable, and that the representable functions are "closed under" composition, primitive recursion, and regular minimization. This guarantees that every general recursive function is representable.

It turns out that the step where we would show that representable functions are closed under primitive recursion is hard. In order to avoid this step, we show first that in fact we can do without primitive recursion. That is, we show that every general recursive function can be defined from basic functions using composition and regular minimization alone. To do this, we show that primitive recursion can actually be done by a specific regular minimization. However, for this to work, we have to add some additional basic functions: addition, multiplication, and the characteristic function of the identity relation $\chi_=$. Then, we can prove the theorem by showing that all of *these* basic functions are representable in **Q**, and the representable functions are closed under composition and regular minimization.

18.2 Functions Representable in Q are Computable

Lemma 18.3. *Every function that is representable in* Q *is computable.*

Proof. Let's first give the intuitive idea for why this is true. If $f(x_0, \ldots, x_k)$ is representable in Q, there is a formula $\varphi(x_0, \ldots, x_k, y)$ such that

$$\mathbf{Q} \vdash \varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m}) \quad \text{iff} \quad m = f(n_0, \ldots, n_k).$$

To compute f, we do the following. List all the possible derivations δ in the language of arithmetic. This is possible to do mechanically. For each one, check if it is a derivation of a formula of the form $\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})$. If it is, m must be $= f(n_0, \ldots, n_k)$ and we've found the value of f. The search terminates because $\mathbf{Q} \vdash \varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{f(n_0, \ldots, n_k)})$, so eventually we find a δ of the right sort.

This is not quite precise because our procedure operates on derivations and formulas instead of just on numbers, and we haven't explained exactly why "listing all possible derivations" is mechanically possible. But as we've seen, it is possible to code terms, formulas, and derivations by Gödel numbers. We've also introduced a precise model of computation, the general recursive functions. And we've seen that the relation $\mathrm{Prf}_{\mathbf{Q}}(d, y)$, which holds iff d is the Gödel number of a derivation of the formula with Gödel number x from the axioms of Q, is (primitive) recursive. Other primitive recursive functions we'll need are num (Proposition 17.6) and Subst (Proposition 17.11). From these, it is possible to define f by minimization; thus, f is recursive.

First, define

$$\begin{aligned} A(n_0, \ldots, n_k, m) = \\ \mathrm{Subst}(\mathrm{Subst}(\ldots \mathrm{Subst}(&{}^{\#}\varphi_f{}^{\#}, \mathrm{num}(n_0), {}^{\#}x_0{}^{\#}), \\ &\ldots), \mathrm{num}(n_k), {}^{\#}x_k{}^{\#}), \mathrm{num}(m), {}^{\#}y{}^{\#}) \end{aligned}$$

This looks complicated, but it's just the function $A(n_0, \ldots, n_k, m) = {}^{\#}\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{m})^{\#}$.

Now, consider the relation $R(n_0, \ldots, n_k, s)$ which holds if $(s)_0$ is the Gödel number of a derivation from Q of $\varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{(s)_1})$:

$$R(n_0, \ldots, n_k, s) \quad \text{iff} \quad \mathrm{Prf}_{\mathbf{Q}}((s)_0, A(n_0, \ldots, n_k, (s)_1))$$

If we can find an s such that $R(n_0, \ldots, n_k, s)$ hold, we have found a pair of numbers—$(s)_0$ and (s_1)—such that $(s)_0$ is the Gödel number of a derivation of $A_f(\overline{n_0}, \ldots, \overline{n_k}, (s)_1)$. So looking for s is like looking for the pair d and m in the informal proof. And a computable function that "looks for" such an s can be defined by regular minimization. Note that R is regular: for every n_0, …, n_k, there is a derivation δ of $\mathbf{Q} \vdash \varphi_f(\overline{n_0}, \ldots, \overline{n_k}, \overline{f(n_0, \ldots, n_k)})$, so $R(n_0, \ldots, n_k, s)$ holds for $s = \langle {}^{\#}\delta^{\#}, f(n_0, \ldots, n_k)\rangle$. So, we can write f as

$$f(n_0, \ldots, n_k) = (\mu s\ R(n_0, \ldots, n_k, s))_1. \qquad \text{fl}$$

18.3 The Beta Function Lemma

In order to show that we can carry out primitive recursion if addition, multiplication, and $\chi_=$ are available, we need to develop functions that handle sequences. (If we had

exponentiation as well, our task would be easier.) When we had primitive recursion, we could define things like the "n-th prime," and pick a fairly straightforward coding. But here we do not have primitive recursion—in fact we want to show that we can do primitive recursion using minimization—so we need to be more clever.

Lemma 18.4. *There is a function $\beta(d, i)$ such that for every sequence $a_0, \ldots, a_n$ there is a number d, such that for every $i \leq n$, $\beta(d, i) = a_i$. Moreover, β can be defined from the basic functions using just composition and regular minimization.*

Think of d as coding the sequence $\langle a_0, \ldots, a_n \rangle$, and $\beta(d, i)$ returning the i-th element. (Note that this "coding" does *not* use the prower-of-primes coding we're already familiar with!). The lemma is fairly minimal; it doesn't say we can concatenate sequences or append elements, or even that we can *compute* d from $a_0, \ldots, a_n$ using functions definable by composition and regular minimization. All it says is that there is a "decoding" function such that every sequence is "coded."

The use of the notation β is Gödel's. To repeat, the hard part of proving the lemma is defining a suitable β using the seemingly restricted resources, i.e., using just composition and minimization—however, we're allowed to use addition, multiplication, and $\chi_=$. There are various ways to prove this lemma, but one of the cleanest is still Gödel's original method, which used a number-theoretic fact called the Chinese Remainder theorem.

Definition 18.5. Two natural numbers a and b are *relatively prime* if their greatest common divisor is 1; in other words, they have no other divisors in common.

Definition 18.6. $a \equiv b \mod c$ means $c \mid (a-b)$, i.e., a and b have the same remainder when divided by c.

Here is the *Chinese Remainder theorem*:

Theorem 18.7. *Suppose $x_0, \ldots, x_n$ are (pairwise) relatively prime. Let $y_0, \ldots, y_n$ be any numbers. Then there is a number z such that*

$$\begin{aligned} z &\equiv y_0 \mod x_0 \\ z &\equiv y_1 \mod x_1 \\ &\vdots \\ z &\equiv y_n \mod x_n. \end{aligned}$$

Here is how we will use the Chinese Remainder theorem: if $x_0, \ldots, x_n$ are bigger than $y_0, \ldots, y_n$ respectively, then we can take z to code the sequence $\langle y_0, \ldots, y_n \rangle$. To recover y_i, we need only divide z by x_i and take the remainder. To use this coding, we will need to find suitable values for $x_0, \ldots, x_n$.

A couple of observations will help us in this regard. Given $y_0, \ldots, y_n$, let

$$j = \max(n, y_0, \ldots, y_n) + 1,$$

and let

$$\begin{aligned} x_0 &= 1 + j\,! \\ x_1 &= 1 + 2 \cdot j\,! \\ x_2 &= 1 + 3 \cdot j\,! \\ &\vdots \\ x_n &= 1 + (n+1) \cdot j\,! \end{aligned}$$

Then two things are true:

1. $x_0, \ldots, x_n$ are relatively prime.
2. For each i, $y_i < x_i$.

To see that (1) is true, note that if p is a prime number and $p \mid x_i$ and $p \mid x_k$, then $p \mid 1 + (i+1)j\,!$ and $p \mid 1 + (k+1)j\,!$. But then p divides their difference,

$$(1 + (i+1)j\,!) - (1 + (k+1)j\,!) = (i-k)j\,!.$$

Since p divides $1 + (i+1)j\,!$, it can't divide $j\,!$ as well (otherwise, the first division would leave a remainder of 1). So p divides $i-k$, since p divides $(i-k)j\,!$. But $|i-k|$ is at most n, and we have chosen $j > n$, so this implies that $p \mid j\,!$, again a contradiction. So there is no prime number dividing both x_i and x_k. Clause (2) is easy: we have $y_i < j < j\,! < x_i$.

Now let us prove the β function lemma. Remember that we can use 0, successor, plus, times, $\chi_=$, projections, and any function defined from them using composition and minimization applied to regular functions. We can also use a relation if its characteristic function is so definable. As before we can show that these relations are closed under boolean combinations and bounded quantification; for example:

1. $\text{not}(x) = \chi_=(x, 0)$
2. $(\min x \leq z)\, R(x, y) = \mu x\ (R(x, y) \vee x = z)$
3. $(\exists x \leq z)\ R(x, y) \Leftrightarrow R((\min x \leq z)\, R(x, y), y)$

We can then show that all of the following are also definable without primitive recursion:

1. The pairing function, $J(x, y) = \frac{1}{2}[(x + y)(x + y + 1)] + x$
2. Projections
$$K(z) = (\min x \leq q)\ (\exists y \leq z\ [z = J(x, y)])$$
and
$$L(z) = (\min y \leq q)\ (\exists x \leq z\ [z = J(x, y)]).$$
3. $x < y$
4. $x \mid y$
5. The function $\text{rem}(x, y)$ which returns the remainder when y is divided by x

Now define

$$\beta^*(d_0, d_1, i) = \text{rem}(1 + (i+1)d_1, d_0)$$

and

$$\beta(d, i) = \beta^*(K(d), L(d), i).$$

This is the function we need. Given $a_0, \ldots, a_n$, as above, let

$$j = \max(n, a_0, \ldots, a_n) + 1,$$

and let $d_1 = j!$. By the observations above, we know that $1+d_1, 1+2d_1, \ldots, 1+(n+1)d_1$ are relatively prime and all are bigger than $a_0, \ldots, a_n$. By the Chinese Remainder theorem there is a value d_0 such that for each i,

$$d_0 \equiv a_i \mod (1 + (i+1)d_1)$$

and so (because d_1 is greater than a_i),

$$a_i = \text{rem}(1 + (i+1)d_1, d_0).$$

Let $d = J(d_0, d_1)$. Then for each $i \leq n$, we have

$$\begin{aligned} \beta(d, i) &= \beta^*(d_0, d_1, i) \\ &= \text{rem}(1 + (i+1)d_1, d_0) \\ &= a_i \end{aligned}$$

which is what we need. This completes the proof of the β-function lemma.

18.4 Simulating Primitive Recursion

Now we can show that definition by primitive recursion can be "simulated" by regular minimization using the beta function. Suppose we have $f(\vec{x})$ and $g(\vec{x}, y, z)$. Then the function $h(x, \vec{z})$ defined from f and g by primitive recursion is

$$\begin{aligned} h(\vec{x}, y) &= f(\vec{z}) \\ h(\vec{x}, y+1) &= g(\vec{x}, y, h(\vec{x}, y)). \end{aligned}$$

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 18.8. *If h can be defined from f and g using primitive recursion, it can be defined from f, g, the functions* zero, succ, P^n_i, add, mult, $\chi_=$, *using composition and regular minimization.*

Proof. First, define an auxiliary function $\hat{h}(\vec{x}, y)$ 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 18.4.

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. □

18.5 Basic Functions are Representable in Q

First we have to show that all the basic functions are representable in Q. In the end, we need to show how to assign to each k-ary basic function $f(x_0, \ldots, x_{k-1})$ a formula $\varphi_f(x_0, \ldots, x_{k-1}, y)$ that represents it.

We will be able to represent zero, successor, plus, times, the characteristic function for equality, and projections. In each case, the appropriate representing function is entirely straightforward; for example, zero is represented by the formula $y = 0$, successor is represented by the formula $x_0' = y$, and addition is represented by the formula $(x_0 + x_1) = y$. The work involves showing that Q can prove the relevant sentences; for example, saying that addition is represented by the formula above involves showing that for every pair of natural numbers m and n, Q proves

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

Proposition 18.9. *The zero function* $\mathrm{zero}(x) = 0$ *is represented in* Q *by* $y = 0$.

Proposition 18.10. *The successor function* $\mathrm{succ}(x) = x + 1$ *is represented in* Q *by* $y = x'$.

Proposition 18.11. *The projection function* $P^n_i(x_0, \ldots, x_{n-1}) = x_i$ *is represented in* Q *by* $y = x_i$.

Proposition 18.12. *The characteristic function of* $=$,

$$\chi_=(x_0, x_1) = \begin{cases} 1 & \text{if } x_0 = x_1 \\ 0 & \text{otherwise} \end{cases}$$

is represented in Q *by*

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

The proof requires the following lemma.

Lemma 18.13. *Given natural numbers* n *and* m, *if* $n \neq m$, *then* $\mathbf{Q} \vdash \overline{n} \neq \overline{m}$.

Proof. Use induction on n to show that for every m, if $n \neq m$, then $Q \vdash \overline{n} \neq \overline{m}$.

In the base case, $n = 0$. If m is not equal to 0, then $m = k + 1$ for some natural number k. We have an axiom that says $\forall x\, 0 \neq x'$. By a quantifier axiom, replacing x by $\overline{k}$, we can conclude $0 \neq \overline{k}'$. But $\overline{k}'$ is just $\overline{m}$.

In the induction step, we can assume the claim is true for n, and consider $n + 1$. Let m be any natural number. There are two possibilities: either $m = 0$ or for some k we have $m = k + 1$. The first case is handled as above. In the second case, suppose $n + 1 \neq k + 1$. Then $n \neq k$. By the induction hypothesis for n we have $\mathbf{Q} \vdash \overline{n} \neq \overline{k}$. We have an axiom that says $\forall x\, \forall y\, x' = y' \to x = y$. Using a quantifier axiom, we have $\overline{n}' = \overline{k}' \to \overline{n} = \overline{k}$. Using propositional logic, we can conclude, in $\mathbf{Q}$, $\overline{n} \neq \overline{k} \to \overline{n}' \neq \overline{k}'$. Using modus ponens, we can conclude $\overline{n}' \neq \overline{k}'$, which is what we want, since $\overline{k}'$ is $\overline{m}$. □

Note that the lemma does not say much: in essence it says that $\mathbf{Q}$ can prove that different numerals denote different objects. For example, $\mathbf{Q}$ proves $0'' \neq 0'''$. But showing that this holds in general requires some care. Note also that although we are using induction, it is induction *outside* of $\mathbf{Q}$.

Proof of Proposition 18.12. If $n = m$, then $\overline{n}$ and $\overline{m}$ are the same term, and $\chi_=(n, m) = 1$. But $\mathbf{Q} \vdash (\overline{n} = \overline{m} \wedge \overline{1} = \overline{1})$, so it proves $\varphi_=(\overline{n}, \overline{m}, \overline{1})$. If $n \neq m$, then $\chi_=(n, m) = 0$. By Lemma 18.13, $\mathbf{Q} \vdash \overline{n} \neq \overline{m}$ and so also $(\overline{n} \neq \overline{m} \wedge 0 = 0)$. Thus $\mathbf{Q} \vdash \varphi_=(\overline{n}, \overline{m}, \overline{0})$.

For the second part, we also have two cases. If $n = m$, we have to show that $\mathbf{Q} \vdash \forall y\, (\varphi_=(\overline{n}, \overline{m}, y) \to y = \overline{1})$. Arguing informally, suppose $\varphi_=(\overline{n}, \overline{m}, y)$, i.e.,

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

The left disjunct implies $y = \overline{1}$ by logic; the right contradicts $\overline{n} = \overline{n}$ which is provable by logic.

Suppose, on the other hand, that $n \neq m$. Then $\varphi_=(\overline{n}, \overline{m}, y)$ is

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

Here, the left disjunct contradicts $\overline{n} \neq \overline{m}$, which is provable in $\mathbf{Q}$ by Lemma 18.13; the right disjunct entails $y = \overline{0}$. □

Proposition 18.14. *The addition function* $\mathrm{add}(x_0, x_1) = x_0 + x_1$ *is represented in* $\mathbf{Q}$ *by*

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

Lemma 18.15. $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$

Proof. We prove this by induction on m. If $m = 0$, the claim is that $\mathbf{Q} \vdash (\overline{n} + 0) = \overline{n}$. This follows by axiom Q_4. Now suppose the claim for m; let's prove the claim for $m+1$, i.e., prove that $\mathbf{Q} \vdash (\overline{n} + \overline{m + 1}) = \overline{n + m + 1}$. Note that $\overline{m + 1}$ is just $\overline{m}'$, and $\overline{n + m + 1}$ is just $\overline{n + m}'$. By axiom Q_5, $\mathbf{Q} \vdash (\overline{n} + \overline{m}') = (\overline{n} + \overline{m})'$. By induction hypothesis, $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$. So $\mathbf{Q} \vdash (\overline{n} + \overline{m}') = \overline{n + m}'$. □

Proof of Proposition 18.14. The formula $\varphi_{\mathrm{add}}(x_0, x_1, y)$ representing add is $y = (x_0 + x_1)$. First we show that if $\mathrm{add}(n, m) = k$, then $\mathbf{Q} \vdash \varphi_{\mathrm{add}}(\overline{n}, \overline{m}, \overline{k})$, i.e., $\mathbf{Q} \vdash \overline{k} = (\overline{n} + \overline{m})$. But since $k = n + m$, $\overline{k}$ just is $\overline{n + m}$, and we've shown in Lemma 18.15 that $\mathbf{Q} \vdash (\overline{n} + \overline{m}) = \overline{n + m}$.

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

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

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 18.16. *The multiplication function* $\mathrm{mult}(x_0, x_1) = x_0 \cdot x_1$ *is represented in* Q *by*

$$y = (x_0 \times x_1).$$

Proof. Exercise. □

Lemma 18.17. $\mathbf{Q} \vdash (\overline{n} \times \overline{m}) = \overline{n \cdot m}$

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 $n + m$.

18.6 Composition is Representable in Q

Suppose h is defined by

$$h(x_0, \ldots, x_{l-1}) = f(g_0(x_0, \ldots, x_{l-1}), \ldots, g_{k-1}(x_0, \ldots, x_{l-1})).$$

where we have already found formulas $\varphi_f, \varphi_{g_0}, \ldots, \varphi_{g_{k-1}}$ representing the functions f, and $g_0, \ldots, g_{k-1}$, respectively. We have to find a formula φ_h representing h.

Let's start with a simple case, where all functions are 1-place, i.e., consider $h(x) = f(g(x))$. If $\varphi_f(y, z)$ represents f, and $\varphi_g(x, y)$ represents g, we need a formula $\varphi_h(x, z)$ that represents h. Note that $h(x) = z$ iff there is a y such that both $z = f(y)$ and $y = g(x)$. (If $h(x) = z$, then $g(x)$ is such a y; if such a y exists, then since $y = g(x)$ and $z = f(y)$, $z = f(g(x))$.) This suggests that $\exists y\, (\varphi_g(x, y) \wedge \varphi_f(y, z))$ is a good candidate for $\varphi_h(x, z)$. We just have to verify that Q proves the relevant formulas.

Proposition 18.18. *If* $h(n) = m$, *then* $\mathbf{Q} \vdash \varphi_h(\overline{n}, \overline{m})$.

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

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

since φ_g represents g, and

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

since φ_f represents f. Thus,

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

and consequently also

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

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

Proposition 18.19. *If $h(n) = m$, then $\mathbf{Q} \vdash \forall z\, (\varphi_h(\overline{n}, z) \to z = \overline{m})$.*

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

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

since φ_g represents g, and

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

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

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

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

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

Proposition 18.20. *If $\varphi_f(y_0, \ldots, y_{k-1}, z)$ represents $f(y_0, \ldots, y_{k-1})$ in $\mathbf{Q}$, and $\varphi_{g_i}(x_0, \ldots, x_{l-1}, y)$ represents $g_i(x_0, \ldots, x_{l-1})$ in $\mathbf{Q}$, then*

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

represents

$$h(x_0, \ldots, x_{l-1}) = f(g_0(x_0, \ldots, x_{l-1}), \ldots, g_{k-1}(x_0, \ldots, x_{l-1})).$$

Proof. Exercise. □

18.7 Regular Minimization is Representable in Q

Let's consider unbounded search. Suppose $g(x, z)$ is regular and representable in $\mathbf{Q}$, say by the formula $\varphi_g(x, z, y)$. Let f be defined by $f(z) = \mu x\, [g(x, z) = 0]$. We would like to find a formula $\varphi_f(z, y)$ representing f. The value of $f(z)$ is that number x which (a) satisfies $g(x, z) = 0$ and (b) is the least such, i.e., for any $w < x$, $g(w, z) \neq 0$. So the following is a natural choice:

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

In the general case, of course, we would have to replace z with $z_0, \ldots, z_k$.

The proof, again, will involve some lemmas about things $\mathbf{Q}$ is strong enough to prove.

Lemma 18.21. *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:

$$\mathbf{Q} \vdash (a' + 0) = a' \quad \text{by axiom } Q_4 \tag{18.1}$$
$$\mathbf{Q} \vdash (a + 0) = a \quad \text{by axiom } Q_4 \tag{18.2}$$
$$\mathbf{Q} \vdash (a + 0)' = a' \quad \text{by eq. (18.2)} \tag{18.3}$$
$$\mathbf{Q} \vdash (a' + 0) = (a + 0)' \quad \text{by eq. (18.1) and eq. (18.3)}$$

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:

$$\mathbf{Q} \vdash (a' + \overline{n}') = (a' + \overline{n})' \quad \text{by axiom } Q_5 \tag{18.4}$$
$$\mathbf{Q} \vdash (a' + \overline{n}') = (a + \overline{n}')' \quad \text{inductive hypothesis} \tag{18.5}$$
$$\mathbf{Q} \vdash (a' + \overline{n})' = (a + \overline{n}')' \quad \text{by eq. (18.4) and eq. (18.5).}$$ □

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

Lemma 18.22. $\mathbf{Q} \vdash \forall x \neg x < 0$.

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 $\mathbf{Q}$, 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 18.23. *For every natural number n,*

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

Proof. We use induction on n. Let us consider the base case, when $n = 0$. In that case, we need to show $a < \overline{1} \rightarrow a = 0$, 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 $\mathbf{Q}$, 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 18.22.

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 18.24. *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 18.25. *If $\varphi_g(x, z, y)$ represents $g(x, z)$ in $\mathbf{Q}$, then*

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

represents $f(z) = \mu x\ [g(x, z) = 0]$.

Proof. First we show that if $f(n) = m$, then $\mathbf{Q} \vdash \varphi_f(\overline{n}, \overline{m})$, i.e.,

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

Since $\varphi_g(x, z, y)$ represents $g(x, z)$ and $g(m, n) = 0$ if $f(n) = m$, we have

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

If $f(n) = m$, then for every $k < m$, $g(k, n) \neq 0$. So

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

We get that

$$\mathbf{Q} \vdash \forall w\,(w < \overline{m} \rightarrow \neg\varphi_g(w, \overline{n}, 0)). \tag{18.6}$$

by Lemma 18.22 in case $m = 0$ and by Lemma 18.23 otherwise.

Now let's show that if $f(n) = m$, then $\mathbf{Q} \vdash \forall y\,(\varphi_f(\overline{n}, y) \rightarrow y = \overline{m})$. We again sketch the argument informally, leaving the formalization to the reader.

Suppose $\varphi_f(\overline{n}, b)$. From this we get (a) $\varphi_g(b, \overline{n}, 0)$ and (b) $\forall w\,(w < b \rightarrow \neg\varphi_g(w, \overline{n}, 0))$. By Lemma 18.24, $(b < \overline{m} \vee \overline{m} < b) \vee b = \overline{m}$. We'll show that both $b < \overline{m}$ and $\overline{m} < b$ leads to a contradiction.

If $\overline{m} < b$, then $\neg\varphi_g(\overline{m}, \overline{n}, 0)$ from (b). But $m = f(n)$, so $g(m, n) = 0$, and so $\mathbf{Q} \vdash \varphi_g(\overline{m}, \overline{n}, 0)$ since φ_g represents g. So we have a contradiction.

Now suppose $b < \overline{m}$. Then since $\mathbf{Q} \vdash \forall w\,(w < \overline{m} \rightarrow \neg\varphi_g(w, \overline{n}, 0))$ by eq. (18.6), we get $\neg\varphi_g(b, \overline{n}, 0)$. This again contradicts (a). □

18.8 Computable Functions are Representable in Q

Theorem 18.26. *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^n_i using composition, primitive recursion, and regular minimization. By Lemma 18.8, any function h that can be defined from f and g can also be defined using composition and regular minimization from f, g, and zero, succ, P^n_i, add, mult, $\chi_{=}$. Consequently, a function is general recursive iff it can be defined from zero, succ, P^n_i, add, mult, $\chi_{=}$ using composition and regular minimization.

We've furthermore shown that the basic functions in question are representable in Q (Propositions 18.9 to 18.12, 18.14 and 18.16), and that any function defined from representable functions by composition or regular minimization (Proposition 18.20, Proposition 18.25) is also representable. Thus every general recursive function is representable in Q. □

We have shown that the set of computable functions can be characterized as the set of functions representable in Q. In fact, the proof is more general. From the definition of representability, it is not hard to see that any theory extending Q (or in which one can interpret Q) can represent the computable functions. But, conversely, in any proof system in which the notion of proof is computable, every representable function is computable. So, for example, the set of computable functions can be characterized as the set of functions representable in Peano arithmetic, or even Zermelo-Fraenkel set theory. As Gödel noted, this is somewhat surprising. We will see that when it comes to provability, questions are very sensitive to which theory you consider; roughly, the stronger the axioms, the more you can prove. But across a wide range of axiomatic theories, the representable functions are exactly the computable ones; stronger theories do not represent more functions as long as they are axiomatizable.

18.9 Representing Relations

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

Definition 18.27. A relation $R(x_0, \ldots, x_k)$ on the natural numbers is *representable in* Q if there is a formula $\varphi_R(x_0, \ldots, x_k)$ such that whenever $R(n_0, \ldots, n_k)$ is true, Q proves $\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$, and whenever $R(n_0, \ldots, n_k)$ is false, Q proves $\neg\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$.

Theorem 18.28. *A relation is representable in* Q *if and only if it is computable.*

Proof. For the forwards direction, suppose $R(x_0, \ldots, x_k)$ is represented by the formula $\varphi_R(x_0, \ldots, x_k)$. Here is an algorithm for computing R: on input n_0, …, n_k, simultaneously search for a proof of $\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$ and a proof of $\neg\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$. By our hypothesis, the search is bound to find one or the other; if it is the first, report "yes," and otherwise, report "no."

In the other direction, suppose $R(x_0, \ldots, x_k)$ is computable. By definition, this means that the function $\chi_R(x_0, \ldots, x_k)$ is computable. By Theorem 18.2, χ_R is represented by a formula, say $\varphi_{\chi_R}(x_0, \ldots, x_k, y)$. Let $\varphi_R(x_0, \ldots, x_k)$ be the formula $\varphi_{\chi_R}(x_0, \ldots, x_k, \overline{1})$. Then for any $n_0, \ldots, n_k$, if $R(n_0, \ldots, n_k)$ is true, then $\chi_R(n_0, \ldots, n_k) = 1$, in which case Q proves $\varphi_{\chi_R}(\overline{n_0}, \ldots, \overline{n_k}, \overline{1})$, and so Q proves $\varphi_R(\overline{n_0}, \ldots, \overline{n_k})$. On the other hand, if $R(n_0, \ldots, n_k)$ is false, then $\chi_R(n_0, \ldots, n_k) = 0$. This means that Q proves

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

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

18.10 Undecidability

We call a theory T *undecidable* if there is no computational procedure which, after finitely many steps and unfailingly, provides a correct answer to the question "does T prove φ?" for any sentence φ in the language of T. So Q would be decidable iff there were a computational procedure which decides, given a sentence φ in the language of arithmetic, whether $\mathbf{Q} \vdash \varphi$ or not. We can make this more precise by asking: Is the relation $\mathrm{Prov}_{\mathbf{Q}}(y)$, which holds of y iff y is the Gödel number of a sentence provable in Q, recursive? The answer is: no.

Theorem 18.29. 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 **??**. Since T is primitive recursive it is representable in Q by a formula ψ_T, that is, $\mathbf{Q} \vdash \psi_T(\overline{e}, \overline{n}, \overline{s})$ iff $T(e, n, s)$. If $\mathbf{Q} \vdash \psi_T(\overline{e}, \overline{n}, \overline{s})$ then also $\mathbf{Q} \vdash \exists y\, \psi_T(\overline{e}, \overline{n}, y)$. If no such s exists, then $\mathbf{Q} \vdash \neg\psi_T(\overline{e}, \overline{n}, \overline{s})$ for every s. But Q is ω-consistent, i.e., if $\mathbf{Q} \vdash \neg\varphi(\overline{n})$ for every $n \in \mathbb{N}$, then $\mathbf{Q} \nvdash \exists y\, \varphi(y)$. We know this because the axioms of Q are true in the standard model $\mathfrak{N}$. So, $\mathbf{Q} \nvdash \exists y\, \psi_T(\overline{e}, \overline{n}, y)$. In other words, $\mathbf{Q} \vdash \exists y\, \psi_T(\overline{e}, \overline{n}, y)$ iff there is an s such that $T(e, n, s)$, i.e., iff $\varphi_e(n) \downarrow$. From e and n

we can compute ${}^{\#}\exists y\, \psi_T(\overline{e}, \overline{n}, y)^{\#}$, let $g(e, n)$ be the primitive recursive function which does that. So

$$h(e, n) = \begin{cases} 1 & \text{if } \mathrm{Prov}_{\mathbf{Q}}(g(e, n)) \\ 0 & \text{otherwise.} \end{cases}$$

This would show that h is recursive if $\mathrm{Prov}_{\mathbf{Q}}$ is. But h is not recursive, by **??**, so $\mathrm{Prov}_{\mathbf{Q}}$ cannot be either. □

Corollary 18.30. *First-order logic is undecidable.*

Proof. If first-order logic were decidable, provability in **Q** would be as well, since $\mathbf{Q} \vdash \varphi$ iff $\vdash \omega \to \varphi$, where ω is the conjunction of the axioms of **Q**. □

Problems

Problem 18.1. Prove that $y = 0$, $y = x'$, and $y = x_i$ represent zero, succ, and P^n_i, respectively.

Problem 18.2. Prove Lemma 18.17.

Problem 18.3. Use Lemma 18.17 to prove Proposition 18.16.

Problem 18.4. Using the proofs of Proposition 18.19 and Proposition 18.19 as a guide, carry out the proof of Proposition 18.20 in detail.

Problem 18.5. Show that if R is representable in **Q**, so is χ_R.

Chapter 19

Incompleteness and Provability

19.1 Introduction

Hilbert thought that a system of axioms for a mathematical structure, such as the natural numbers, is inadequate unless it allows one to derive all true statements about the structure. Combined with his later interest in formal systems of deduction, this suggests that he thought that we should guarantee that, say, the formal systems we are using to reason about the natural numbers is not only consistent, but also *complete*, i.e., every statement in its language is either 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. A few years after Gödel, 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 formula φ in the language of **Q**, let $\ulcorner\varphi\urcorner$ denote the numeral corresponding to ${}^{\#}\varphi^{\#}$. Think about what this means: φ is a formula in the language of **Q**, ${}^{\#}\varphi^{\#}$ is a natural number, and $\ulcorner\varphi\urcorner$ is a *term* in the language of **Q**. So every formula φ in the language of **Q** has a *name*, $\ulcorner\varphi\urcorner$, which is a term in the language of **Q**; this provides us with a conceptual framework in which formulas in the language of **Q** can

"say" things about other formulas. The following lemma is known as the fixed-point lemma.

Lemma 19.1. *Let* $\mathbf{T}$ *be any theory extending* $\mathbf{Q}$*, and let* $\psi(x)$ *be any formula with only the variable* x *free. Then there is a sentence* φ *such that* $\mathbf{T} \vdash \varphi \leftrightarrow \psi(\ulcorner \varphi \urcorner)$.

The lemma asserts that given any property $\psi(x)$, there is a sentence φ that asserts "$\psi(x)$ is true of me," and $\mathbf{T}$ "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.

19.2 The Fixed-Point Lemma

The fixed-point lemma says that for any formula $\psi(x)$, there is a sentence φ such that $\mathbf{T} \vdash \varphi \leftrightarrow \psi(\ulcorner \varphi \urcorner)$, provided $\mathbf{T}$ extends $\mathbf{Q}$. In the case of the liar sentence, we'd want φ to be equivalent (provably in $\mathbf{T}$) to "$\ulcorner \varphi \urcorner$ is false," i.e., the statement that ${}^{\#}\varphi^{\#}$ is the Gödel number of a false sentence. To understand the idea of the proof, it will be useful to compare it with Quine's informal gloss of φ as, "'yields a falsehood when preceded by its own quotation' yields a falsehood when preceded by its own quotation." The operation of taking an expression, and then forming a sentence by preceding this expression by its own quotation may be called *diagonalizing* the expression, and the result its diagonalization. So, the diagonalization of 'yields a falsehood when preceded by its own quotation' is "'yields a falsehood when preceded by its own quotation' yields a falsehood when preceded by its own quotation." Now note that Quine's liar sentence is not the diagonalization of 'yields a falsehood' but of 'yields a falsehood when preceded by its own quotation.' So the property being diagonalized to yield the liar sentence itself involves diagonalization!

In the language of arithmetic, we form quotations of a formula with one free variable by computing its Gödel numbers and then substituting the standard numeral for that Gödel number into the free variable. The diagonalization of $\alpha(x)$ is $\alpha(\overline{n})$, where $n = {}^{\#}\alpha(x)^{\#}$. (From now on, let's abbreviate $\overline{{}^{\#}\alpha(x)^{\#}}$ as $\ulcorner \alpha(x) \urcorner$.) So if $\psi(x)$ is "is a falsehood," then "yields a falsehood if preceded by its own quotation," would be "yields a falsehood when applied to the Gödel number of its diagonalization." If we had a symbol *diag* for the function $\mathrm{diag}(n)$ which computes the Gödel number of the diagonalization of the formula with Gödel number n, we could write $\alpha(x)$ as $\psi(\mathit{diag}(x))$. And Quine's version of the liar sentence would then be the diagonalization of it,

i.e., $\alpha(\ulcorner\alpha(x)\urcorner)$ or $\psi(diag(\ulcorner\psi(diag(x))\urcorner))$. Of course, $\psi(x)$ could now be any other property, and the same construction would work. For the incompleteness theorem, we'll take $\psi(x)$ to be "x is not derivable in **T**." Then $\alpha(x)$ would be "yields a sentence not derivable in **T** when applied to the Gödel number of its diagonalization."

To formalize this in **T**, we have to find a way to formalize diag. The function $\mathrm{diag}(n)$ is computable, in fact, it is primitive recursive: if n is the Gödel number of a formula $\alpha(x)$, $\mathrm{diag}(n)$ returns the Gödel number of $\alpha(\ulcorner\alpha(x)\urcorner)$. (Recall, $\ulcorner\alpha(x)\urcorner$ is the standard numeral of the Gödel number of $\alpha(x)$, i.e., $\overline{{}^{\#}\alpha(x)^{\#}}$). If *diag* were a function symbol in **T** representing the function diag, we could take φ to be the formula $\psi(diag(\ulcorner\psi(diag(x))\urcorner))$. Notice that

$$\begin{aligned}\mathrm{diag}({}^{\#}\psi(diag(x))^{\#}) &= {}^{\#}\psi(diag(\ulcorner\psi(diag(x))\urcorner))^{\#}\\ &= {}^{\#}\varphi^{\#}.\end{aligned}$$

Assuming **T** can derive

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

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

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

Lemma 19.2. *Let $\psi(x)$ be any formula with one free variable x. Then there is a sentence φ such that $\mathbf{Q} \vdash \varphi \leftrightarrow \psi(\ulcorner\varphi\urcorner)$.*

Proof. Given $\psi(x)$, let $\alpha(x)$ be the formula $\exists y\,(\theta_{\mathrm{diag}}(x, y) \wedge \psi(y))$ and let φ be its diagonalization, i.e., the formula $\alpha(\ulcorner\alpha(x)\urcorner)$.

Since θ_{diag} represents diag, and $\mathrm{diag}({}^{\#}\alpha(x)^{\#}) = {}^{\#}\varphi^{\#}$, **Q** can derive

$$\theta_{\mathrm{diag}}(\ulcorner\alpha(x)\urcorner, \ulcorner\varphi\urcorner) \tag{19.1}$$

$$\forall y\,(\theta_{\mathrm{diag}}(\ulcorner\alpha(x)\urcorner, y) \to y = \ulcorner\varphi\urcorner). \tag{19.2}$$

Now we show that $\mathbf{Q} \vdash \varphi \leftrightarrow \psi(\ulcorner\varphi\urcorner)$. We argue informally, using just logic and facts derivable in **Q**.

First, suppose φ, i.e., $\alpha(\ulcorner\alpha(x)\urcorner)$. Going back to the definition of $\alpha(x)$, we see that $\alpha(\ulcorner\alpha(x)\urcorner)$ just is

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

Consider such a y. Since $\theta_{\mathrm{diag}}(\ulcorner\alpha(x)\urcorner, y)$, by eq. (19.2), $y = \ulcorner\varphi\urcorner$. So, from $\psi(y)$ we have $\psi(\ulcorner\varphi\urcorner)$.

Now suppose $\psi(\ulcorner\varphi\urcorner)$. By eq. (19.1), we have $\theta_{\mathrm{diag}}(\ulcorner\alpha(x)\urcorner, \ulcorner\varphi\urcorner) \wedge \psi(\ulcorner\varphi\urcorner)$. It follows that $\exists y\,(\theta_{\mathrm{diag}}(\ulcorner\alpha(x)\urcorner, y) \wedge \psi(y))$. But that's just $\alpha(\ulcorner\alpha\urcorner)$, i.e., φ. □

You should compare this to the proof of the fixed-point lemma in computability theory. The difference is that here we want to define a *statement* in terms of itself, whereas there we wanted to define a *function* in terms of itself; this difference aside, it is really the same idea.

19.3 The First Incompleteness Theorem

We can now describe Gödel's original proof of the first incompleteness theorem. Let **T** be any computably axiomatized theory in a language extending the language of arithmetic, such that **T** includes the axioms of **Q**. This means that, in particular, **T** represents computable functions and relations.

We have argued that, given a reasonable coding of formulas and proofs as numbers, the relation $\mathrm{Prf}_T(x, y)$ is computable, where $\mathrm{Prf}_T(x, y)$ holds if and only if x is the Gödel number of a derivation of the formula with Gödel number y in **T**. In fact, for the particular theory that Gödel had in mind, Gödel was able to show that this relation is primitive recursive, using the list of 45 functions and relations in his paper. The 45th relation, xBy, is just $\mathrm{Prf}_T(x, y)$ for his particular choice of **T**. Remember that where Gödel uses the word "recursive" in his paper, we would now use the phrase "primitive recursive."

Since $\mathrm{Prf}_T(x, y)$ is computable, it is representable in **T**. We will use $\mathrm{Prf}_T(x, y)$ to refer to the formula that represents it. Let $\mathrm{Prov}_T(y)$ be the formula $\exists x\, \mathrm{Prf}_T(x, y)$. This describes the 46th relation, $\mathrm{Bew}(y)$, 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 **T** be an axiomatizable theory containing **Q**. Then $\mathrm{Prf}_T(x, y)$ is decidable, hence representable in **Q** by a formula $\mathrm{Prf}_T(x, y)$. Let $\mathrm{Prov}_T(y)$ be the formula we described above. By the fixed-point lemma, there is a formula $\gamma_{\mathbf{T}}$ such that **Q** (and hence **T**) derives

$$\gamma_{\mathbf{T}} \leftrightarrow \neg \mathrm{Prov}_T(\ulcorner \gamma_{\mathbf{T}} \urcorner). \tag{19.3}$$

Note that $\gamma_{\mathbf{T}}$ says, in essence, "$\gamma_{\mathbf{T}}$ is not derivable in **T**."

Lemma 19.3. *If* **T** *is a consistent, axiomatizable theory extending* **Q**, *then* $\mathbf{T} \nvdash \gamma_{\mathbf{T}}$.

Proof. Suppose **T** derives $\gamma_{\mathbf{T}}$. Then there *is* a derivation, and so, for some number m, the relation $\mathrm{Prf}_T(m, {}^{\#}\gamma_{\mathbf{T}}{}^{\#})$ holds. But then **Q** derives the sentence $\mathrm{Prf}_T(\overline{m}, \ulcorner \gamma_{\mathbf{T}} \urcorner)$. So **Q** derives $\exists x\, \mathrm{Prf}_T(x, \ulcorner \gamma_{\mathbf{T}} \urcorner)$, which is, by definition, $\mathrm{Prov}_T(\ulcorner \gamma_{\mathbf{T}} \urcorner)$. By eq. (19.3), **Q** derives $\neg\gamma_{\mathbf{T}}$, and since **T** extends **Q**, so does **T**. We have shown that if **T** derives $\gamma_{\mathbf{T}}$, then it also derives $\neg\gamma_{\mathbf{T}}$, and hence it would be inconsistent. □

Definition 19.4. A theory **T** is *ω-consistent* if the following holds: if $\exists x\, \varphi(x)$ is any sentence and **T** derives $\neg\varphi(\overline{0})$, $\neg\varphi(\overline{1})$, $\neg\varphi(\overline{2})$, ... then **T** does not prove $\exists x\, \varphi(x)$.

Note that every ω-consistent theory is also consistent. This follows simply from the fact that if **T** is inconsistent, then $\mathbf{T} \vdash \varphi$ for every φ. In particular, if **T** is inconsistent, it derives both $\neg\varphi(\overline{n})$ for every n and also derives $\exists x\, \varphi(x)$. So, if **T** is inconsistent, it is ω-inconsistent. By contraposition, if **T** is ω-consistent, it must be consistent.

Lemma 19.5. *If* **T** *is an ω-consistent, axiomatizable theory extending* **Q**, *then* $\mathbf{T} \nvdash \neg\gamma_{\mathbf{T}}$.

Proof. We show that if **T** derives $\neg\gamma_{\mathbf{T}}$, then it is ω-inconsistent. Suppose **T** derives $\neg\gamma_{\mathbf{T}}$. If **T** is inconsistent, it is ω-inconsistent, and we are done. Otherwise, **T** is consistent, so it does not derive $\gamma_{\mathbf{T}}$ by Lemma 19.3. Since there is no derivation of $\gamma_{\mathbf{T}}$ in **T**, **Q** derives

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

and so does **T**. On the other hand, by eq. (19.3), $\neg\gamma_{\mathbf{T}}$ is equivalent to $\exists x\, \mathrm{Prf}_T(x, \ulcorner \gamma_{\mathbf{T}} \urcorner)$. So **T** is ω-inconsistent. □

Theorem 19.6. *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 $\mathbf{T} \nvdash \gamma_{\mathbf{T}}$ by Lemma 19.3. By Lemma 19.5, $\mathbf{T} \nvdash \neg\gamma_{\mathbf{T}}$. This means that $\mathbf{T}$ is incomplete, since it derives neither $\gamma_{\mathbf{T}}$ nor $\neg\gamma_{\mathbf{T}}$. □

19.4 Rosser's Theorem

Can we modify Gödel's proof to get a stronger result, replacing "ω-consistent" with simply "consistent"? The answer is "yes," using a trick discovered by Rosser. Rosser's trick is to use a "modified" derivability predicate $\mathsf{RProv}_T(y)$ instead of $\mathsf{Prov}_T(y)$.

Theorem 19.7. *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 φ, $\mathrm{not}(x)$ is a code of $\neg\varphi$. Then $\mathsf{Ref}_T(x, y)$ holds iff $\mathsf{Prf}_T(x, \mathrm{not}(y))$. Let $\mathsf{Ref}_T(x, y)$ represent it. Then, if $\mathbf{T} \vdash \neg\varphi$ and δ is a corresponding derivation, $\mathbf{Q} \vdash \mathsf{Ref}_T(\ulcorner\delta\urcorner, \ulcorner\varphi\urcorner)$. We define $\mathsf{RProv}_T(y)$ as

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

Roughly, $\mathsf{RProv}_T(y)$ says "there is a proof of y in $\mathbf{T}$, and there is no shorter refutation of y." 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 19.6, 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 $\neg\rho_T$, 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 18.23, $\mathbf{Q} \vdash \forall z\, (z < \overline{n} \to \neg\mathsf{Ref}_T(z, \ulcorner\rho_T\urcorner))$. 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 $\mathsf{Ref}_T(n, {}^{\#}\rho_T{}^{\#})$ holds, and since Ref_T represents Ref_T in $\mathbf{Q}$, $\mathbf{Q} \vdash \mathsf{Ref}_T(\overline{n}, \ulcorner\rho_T\urcorner)$. We'll again show that $\mathbf{T}$ would then be inconsistent because it would also 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 18.23, (b) $\neg(x < \overline{n})$. By Lemma 18.24, $\overline{n} < x$. 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

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

19.5 Comparison with Gödel's Original Paper

It is worthwhile to spend some time with Gödel's 1931 paper. The introduction sketches the ideas we have just discussed. Even if you just skim through the paper, it is easy to see what is going on at each stage: first Gödel describes the formal system P (syntax, axioms, proof rules); then he defines the primitive recursive functions and relations; then he shows that xBy is primitive recursive, and argues that the primitive recursive functions and relations are represented in $\mathbf{P}$. 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.

Problems

Problem 19.1. 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 $\mathbf{Q} \cup \{\neg\gamma_{\mathbf{Q}}\}$ is consistent but ω-inconsistent.

Part VI

Appendices

Appendix A

Proofs

A.1 Introduction

Based on your experiences in introductory logic, you might be comfortable with a proof system—probably a natural deduction or Fitch style proof system, or perhaps a proof-tree system. You probably remember doing proofs in these systems, either proving a formula or show that a given argument is valid. In order to do this, you applied the rules of the system until you got the desired end result. In reasoning *about* logic, we also prove things, but in most cases we are not using a proof system. In fact, most of the proofs we consider are done in English (perhaps, with some symbolic language thrown in) rather than entirely in the language of first-order logic. When constructing such proofs, you might at first be at a loss—how do I prove something without a proof system? How do I start? How do I know if my proof is correct?

Before attempting a proof, it's important to know what a proof is and how to construct one. As implied by the name, a *proof* is meant to show that something is true. You might think of this in terms of a dialogue—someone asks you if something is true, say, if every prime other than two is an odd number. To answer "yes" is not enough; they might want to know *why*. In this case, you'd give them a proof.

In everyday discourse, it might be enough to gesture at an answer, or give an incomplete answer. In logic and mathematics, however, we want rigorous proof—we want to show that something is true beyond *any* doubt. This means that every step in our proof must be justified, and the justification must be cogent (i.e., the assumption you're using is actually assumed in the statement of the theorem you're proving, the definitions you apply must be correctly applied, the justifications appealed to must be correct inferences, etc.).

Usually, we're proving some statement. We call the statements we're proving by various names: propositions, theorems, lemmas, or corollaries. A proposition is a basic proof-worthy statement: important enough to record, but perhaps not particularly deep nor applied often. A theorem is a significant, important proposition. Its proof often is broken into several steps, and sometimes it is named after the person who first proved it (e.g., Cantor's Theorem, the Löwenheim-Skolem theorem) or after the fact it concerns (e.g., the completeness theorem). A lemma is a proposition or theorem that is used to in the proof of a more important result. Confusingly, sometimes lemmas are important results in themselves, and also named after the person who introduced them (e.g., Zorn's Lemma). A corollary is a result that easily follows from another one.

A statement to be proved often contains some assumption that clarifies about which kinds of things we're proving something. It might begin with "Let φ be a formula of the form $\psi \to \chi$" or "Suppose $\Gamma \vdash \varphi$" or something of the sort. These are *hypotheses* of the proposition, theorem, or lemma, and you may assume these to be true in your proof. They restrict what we're proving about, and also introduce some names for the objects we're talking about. For instance, if your proposition begins with "Let φ be a formula of the form $\psi \to \chi$," you're proving something about all formulas of a certain sort only (namely, conditionals), and it's understood that $\psi \to \chi$ is an arbitrary conditional that your proof will talk about.

A.2 Starting a Proof

But where do you even start?

You've been given something to prove, so this should be the last thing that is mentioned in the proof (you can, obviously, *announce* that you're going to prove it at the beginning, but you don't want to use it as an assumption). Write what you are trying to prove at the bottom of a fresh sheet of paper—this way you don't lose sight of your goal.

Next, you may have some assumptions that you are able to use (this will be made clearer when we talk about the *type* of proof you are doing in the next section). Write these at the top of the page and make sure to flag that they are assumptions (i.e., if you are assuming p, write "assume that p," or "suppose that p"). Finally, there might be some definitions in the question that you need to know. You might be told to use a specific definition, or there might be various definitions in the assumptions or conclusion that you are working towards. *Write these down and ensure that you understand what they mean.*

How you set up your proof will also be dependent upon the form of the question. The next section provides details on how to set up your proof based on the type of sentence.

A.3 Using Definitions

We mentioned that you must be familiar with all definitions that may be used in the proof, and that you can properly apply them. This is a really important point, and it is worth looking at in a bit more detail. Definitions are used to abbreviate properties and relations so we can talk about them more succinctly. The introduced abbreviation is called the *definiendum*, and what it abbreviates is the *definiens*. In proofs, we often have to go back to how the definiendum was introduced, because we have to exploit the logical structure of the definiens (the long version of which the defined term is the abbreviation) to get through our proof. By unpacking definitions, you're ensuring that you're getting to the heart of where the logical action is.

We'll start with an example. Suppose you want to prove the following:

Proposition A.1. *For any sets A and B, $A \cup B = B \cup A$.*

In order to even start the proof, we need to know what it means for two sets to be identical; i.e., we need to know what the "=" in that equation means for sets. Sets are defined to be identical whenever they have the same elements. So the definition we have to unpack is:

Definition A.2. Sets A and B are *identical*, $A = B$, iff every element of A is an element of B, and vice versa.

This definition uses A and B as placeholders for arbitrary sets. What it defines—the *definiendum*—is the expression "$A = B$" by giving the condition under which $A = B$ is true. This condition—"every element of A is an element of B, and vice versa"—is the *definiens*.[1] The definition specifies that $A = B$ is true if, and only if (we abbreviate this to "iff") the condition holds.

When you apply the definition, you have to match the A and B in the definition to the case you're dealing with. In our case, it means that in order for $A \cup B = B \cup A$ to be true, each $z \in A \cup B$ must also be in $B \cup A$, and vice versa. The expression $A \cup B$ in the proposition plays the role of A in the definition, and $B \cup A$ that of B. Since A and B are used both in the definition and in the statement of the proposition we're proving, but in different uses, you have to be careful to make sure you don't mix up the two. For instance, it would be a mistake to think that you could prove the proposition by showing that every element of A is an element of B, and vice versa—that would show that $A = B$, not that $A \cup B = B \cup A$. (Also, since A and B may be any two sets, you won't get very far, because if nothing is assumed about A and B they may well be different sets.)

Within the proof we are dealing with set-theoretic notions such as union, and so we must also know the meanings of the symbol $\cup$ in order to understand how the proof should proceed. And sometimes, unpacking the definition gives rise to further definitions to unpack. For instance, $A \cup B$ is defined as $\{z \mid z \in A \text{ or } z \in B\}$. So if you want to prove that $x \in A \cup B$, unpacking the definition of $\cup$ tells you that you have to prove $x \in \{z \mid z \in A \text{ or } z \in B\}$. Now you also have to remember that $x \in \{z \mid \ldots z \ldots\}$ iff $\ldots x \ldots$. So, further unpacking the definition of the $\{z \mid \ldots z \ldots\}$ notation, what you have to show is: $x \in A$ or $x \in B$. So, "every element of $A \cup B$ is also an element of $B \cup A$" really means: "for every x, if $x \in A$ or $x \in B$, then $x \in B$ or $x \in A$." If we fully unpack the definitions in the proposition, we see that what we have to show is this:

Proposition A.3. *For any sets A and B: (a) for every x, if $x \in A$ or $x \in B$, then $x \in B$ or $x \in A$, and (b) for every x, if $x \in B$ or $x \in A$, then $x \in A$ or $x \in B$.*

What's important is that unpacking definitions is a necessary part of constructing a proof. Properly doing it is sometimes difficult: you must be careful to distinguish and match the variables in the definition and the terms in the claim you're proving. In order to be successful, you must know what the question is asking and what all the terms used in the question mean—you will often need to unpack more than one definition. In simple proofs such as the ones below, the solution follows almost immediately from the definitions themselves. Of course, it won't always be this simple.

A.4 Inference Patterns

Proofs are composed of individual inferences. When we make an inference, we typically indicate that by using a word like "so," "thus," or "therefore." The inference

[1] In this particular case—and very confusingly!—when $A = B$, the sets A and B are just one and the same set, even though we use different letters for it on the left and the right side. But the ways in which that set is picked out may be different, and that makes the definition non-trivial.

often relies on one or two facts we already have available in our proof—it may be something we have assumed, or something that we've concluded by an inference already. To be clear, we may label these things, and in the inference we indicate what other statements we're using in the inference. An inference will often also contain an explanation of *why* our new conclusion follows from the things that come before it. There are some common patterns of inference that are used very often in proofs; we'll go through some below. Some patterns of inference, like proofs by induction, are more involved (and will be discussed later).

We've already discussed one pattern of inference: unpacking, or applying, a definition. When we unpack a definition, we just restate something that involves the definiendum by using the definiens. For instance, suppose that we have already established in the course of a proof that $D = E$ (a). Then we may apply the definition of = for sets and infer: "Thus, by definition from (a), every element of D is an element of E and vice versa."

Somewhat confusingly, we often do not write the justification of an inference when we actually make it, but before. Suppose we haven't already proved that $D = E$, but we want to. If $D = E$ is the conclusion we aim for, then we can restate this aim also by applying the definition: to prove $D = E$ we have to prove that every element of D is an element of E and vice versa. So our proof will have the form: (a) prove that every element of D is an element of E; (b) every element of E is an element of D; (c) therefore, from (a) and (b) by definition of =, $D = E$. But we would usually not write it this way. Instead we might write something like,

> We want to show $D = E$. By definition of =, this amounts to showing that every element of D is an element of E and vice versa.
>
> (a) …(a proof that every element of D is an element of E) …
>
> (b) …(a proof that every element of E is an element of D) …

Using a Conjunction

Perhaps the simplest inference pattern is that of drawing as conclusion one of the conjuncts of a conjunction. In other words: if we have assumed or already proved that p and q, then we're entitled to infer that p (and also that q). This is such a basic inference that it is often not mentioned. For instance, once we've unpacked the definition of $D = E$ we've established that every element of D is an element of E and vice versa. From this we can conclude that every element of E is an element of D (that's the "vice versa" part).

Proving a Conjunction

Sometimes what you'll be asked to prove will have the form of a conjunction; you will be asked to "prove p and q." In this case, you simply have to do two things: prove p, and then prove q. You could divide your proof into two sections, and for clarity, label them. When you're making your first notes, you might write "(1) Prove p" at the top of the page, and "(2) Prove q" in the middle of the page. (Of course, you might not be explicitly asked to prove a conjunction but find that your proof requires that you prove a conjunction. For instance, if you're asked to prove that $D = E$ you will find that, after unpacking the definition of =, you have to prove: every element of D is an element of E *and* every element of E is an element of D).

Proving a Disjunction

When what you are proving takes the form of a disjunction (i.e., it is an statement of the form "p or q"), it is enough to show that one of the disjuncts is true. However, it basically never happens that either disjunct just follows from the assumptions of your theorem. More often, the assumptions of your theorem are themselves disjunctive, or you're showing that all things of a certain kind have one of two properties, but some of the things have the one and others have the other property. This is where proof by cases is useful (see below).

Conditional Proof

Many theorems you will encounter are in conditional form (i.e., show that if p holds, then q is also true). These cases are nice and easy to set up—simply assume the antecedent of the conditional (in this case, p) and prove the conclusion q from it. So if your theorem reads, "If p then q," you start your proof with "assume p" and at the end you should have proved q.

Conditionals may be stated in different ways. So instead of "If p then q," a theorem may state that "p only if q," "q if p," or "q, provided p." These all mean the same and require assuming p and proving q from that assumption. Recall that a biconditional ("p if and only if (iff) q") is really two conditionals put together: if p then q, and if q then p. All you have to do, then, is two instances of conditional proof: one for the first conditional and another one for the second. Sometimes, however, it is possible to prove an "iff" statement by chaining together a bunch of other "iff" statements so that you start with "p" an end with "q"—but in that case you have to make sure that each step really is an "iff."

Universal Claims

Using a universal claim is simple: if something is true for anything, it's true for each particular thing. So if, say, the hypothesis of your proof is $A \subseteq B$, that means (unpacking the definition of $\subseteq$), that, for every $x \in A$, $x \in B$. Thus, if you already know that $z \in A$, you can conclude $z \in B$.

Proving a universal claim may seem a little bit tricky. Usually these statements take the following form: "If x has P, then it has Q" or "All Ps are Qs." Of course, it might not fit this form perfectly, and it takes a bit of practice to figure out what you're asked to prove exactly. But: we often have to prove that all objects with some property have a certain other property.

The way to prove a universal claim is to introduce names or variables, for the things that have the one property and then show that they also have the other property. We might put this by saying that to prove something for *all* Ps you have to prove it for an *arbitrary* P. And the name introduced is a name for an arbitrary P. We typically use single letters as these names for arbitrary things, and the letters usually follow conventions: e.g., we use n for natural numbers, φ for formulas, A for sets, f for functions, etc.

The trick is to maintain generality throughout the proof. You start by assuming that an arbitrary object ("x") has the property P, and show (based only on definitions or what you are allowed to assume) that x has the property Q. Because you have not stipulated what x is specifically, other that it has the property P, then you can

assert that all every P has the property Q. In short, x is a stand-in for *all* things with property P.

Proposition A.4. *For all sets A and B, $A \subseteq A \cup B$.*

Proof. Let A and B be arbitrary sets. We want to show that $A \subseteq A \cup B$. By definition of $\subseteq$, this amounts to: for every x, if $x \in A$ then $x \in A \cup B$. So let $x \in A$ be an arbitrary element of A. We have to show that $x \in A \cup B$. Since $x \in A$, $x \in A$ or $x \in B$. Thus, $x \in \{x \mid x \in A \vee x \in B\}$. But that, by definition of $\cup$, means $x \in A \cup B$. □

Proof by Cases

Suppose you have a disjunction as an assumption or as an already established conclusion—you have assumed or proved that p or q is true. You want to prove r. You do this in two steps: first you assume that p is true, and prove r, then you assume that q is true and prove r again. This works because we assume or know that one of the two alternatives holds. The two steps establish that either one is sufficient for the truth of r. (If both are true, we have not one but two reasons for why r is true. It is not necessary to separately prove that r is true assuming both p and q.) To indicate what we're doing, we announce that we "distinguish cases." For instance, suppose we know that $x \in B \cup C$. $B \cup C$ is defined as $\{x \mid x \in B \text{ or } x \in C\}$. In other words, by definition, $x \in B$ or $x \in C$. We would prove that $x \in A$ from this by first assuming that $x \in B$, and proving $x \in A$ from this assumption, and then assume $x \in C$, and again prove $x \in A$ from this. You would write "We distinguish cases" under the assumption, then "Case (1): $x \in B$" underneath, and "Case (2): $x \in C$ halfway down the page. Then you'd proceed to fill in the top half and the bottom half of the page.

Proof by cases is especially useful if what you're proving is itself disjunctive. Here's a simple example:

Proposition A.5. *Suppose $B \subseteq D$ and $C \subseteq E$. Then $B \cup C \subseteq D \cup E$.*

Proof. Assume (a) that $B \subseteq D$ and (b) $C \subseteq E$. By definition, any $x \in B$ is also $\in D$ (c) and any $x \in C$ is also $\in E$ (d). To show that $B \cup C \subseteq D \cup E$, we have to show that if $x \in B \cup C$ then $x \in D \cup E$ (by definition of $\subseteq$). $x \in B \cup C$ iff $x \in B$ or $x \in C$ (by definition of $\cup$). Similarly, $x \in D \cup E$ iff $x \in D$ or $x \in E$. So, we have to show: for any x, if $x \in B$ or $x \in C$, then $x \in D$ or $x \in E$.

> So far we've only unpacked definitions! We've reformulated our proposition without $\subseteq$ and $\cup$ and are left with trying to prove a universal conditional claim. By what we've discussed above, this is done by assuming that x is something about which we assume the "if" part is true, and we'll go on to show that the "then" part is true as well. In other words, we'll assume that $x \in B$ or $x \in C$ and show that $x \in D$ or $x \in E$.[2]

Suppose that $x \in B$ or $x \in C$. We have to show that $x \in D$ or $x \in E$. We distinguish cases.

Case 1: $x \in B$. By (c), $x \in D$. Thus, $x \in D$ or $x \in E$. (Here we've made the inference discussed in the preceding subsection!)

Case 2: $x \in C$. By (d), $x \in E$. Thus, $x \in D$ or $x \in E$. □

[2] This paragraph just explains what we're doing—it's not part of the proof, and you don't have to go into all this detail when you write down your own proofs.

Proving an Existence Claim

When asked to prove an existence claim, the question will usually be of the form "prove that there is an x such that $\ldots x \ldots$", i.e., that some object that has the property described by "$\ldots x \ldots$". In this case you'll have to identify a suitable object show that is has the required property. This sounds straightforward, but a proof of this kind can be tricky. Typically it involves *constructing* or *defining* an object and proving that the object so defined has the required property. Finding the right object may be hard, proving that it has the required property may be hard, and sometimes it's even tricky to show that you've succeeded in defining an object at all!

Generally, you'd write this out by specifying the object, e.g., "let x be …" (where … specifies which object you have in mind), possibly proving that … in fact describes an object that exists, and then go on to show that x has the property Q. Here's a simple example.

Proposition A.6. *Suppose that $x \in B$. Then there is an A such that $A \subseteq B$ and $A \neq \emptyset$.*

Proof. Assume $x \in B$. Let $A = \{x\}$.

> Here we've defined the set A by enumerating its elements. Since we assume that x is an object, and we can always form a set by enumerating its elements, we don't have to show that we've succeeded in defining a set A here. However, we still have to show that A has the properties required by the proposition. The proof isn't complete without that!

Since $x \in A$, $A \neq \emptyset$.

> This relies on the definition of A as $\{x\}$ and the obvious facts that $x \in \{x\}$ and $x \notin \emptyset$.

Since x is the only element of $\{x\}$, and $x \in B$, every element of A is also an element of B. By definition of $\subseteq$, $A \subseteq B$. □

Using Existence Claims

Suppose you know that some existence claim is true (you've proved it, or it's a hypothesis you can use), say, "for some x, $x \in A$" or "there is an $x \in A$." If you want to use it in your proof, you can just pretend that you have a name for one of the things which your hypothesis says exist. Since A contains at least one thing, there are things to which that name might refer. You might of course not be able to pick one out or describe it further (other than that it is $\in A$). But for the purpose of the proof, you can pretend that you have picked it out and give a name to it. It's important to pick a name that you haven't already used (or that appears in your hypotheses), otherwise things can go wrong. In your proof, you indicate this by going from "for some x, $x \in A$" to "Let $a \in A$." Now you can reason about a, use some other hypotheses, etc., until you come to a conclusion, p. If p no longer mentions a, p is independent of the asusmption that $a \in A$, and you've shown that it follows just from the assumption "for some x, $x \in A$."

Proposition A.7. *If $A \neq \emptyset$, then $A \cup B \neq \emptyset$.*

Proof. Suppose $A \neq \emptyset$. So for some x, $x \in A$.

> Here we first just restated the hypothesis of the proposition. This hypothesis, i.e., $A \neq \emptyset$, hides an existential claim, which you get to only by unpacking a few definitions. The definition of = tells us that $A = \emptyset$ iff every $x \in A$ is also $\in \emptyset$ and every $x \in \emptyset$ is also $\in A$. Negating both sides, we get: $A \neq \emptyset$ iff either some $x \in A$ is $\notin \emptyset$ or some $x \in \emptyset$ is $\notin A$. Since nothing is $\in \emptyset$, the second disjunct can never be true, and "$x \in A$ and $x \notin \emptyset$" reduces to just $x \in A$. So $x \neq \emptyset$ iff for some x, $x \in A$. That's an existence claim. Now we use that existence claim by introducing a name for one of the elements of A:

Let $a \in A$.

> Now we've introduced a name for one of the things $\in A$. We'll continue to argue about a, but we'll be careful to only assume that $a \in A$ and nothing else:

Since $a \in A$, $a \in A \cup B$, by definition of $\cup$. So for some x, $x \in A \cup B$, i.e., $A \cup B \neq \emptyset$.

> In that last step, we went from "$a \in A \cup B$" to "for some x, $x \in A \cup B$." That doesn't mention a anymore, so we know that "for some x, $x \in A \cup B$" follows from "for some x, $x \in A$ alone." But that means that $A \cup B \neq \emptyset$. □

It's maybe good practice to keep bound variables like "x" separate from hypothtical names like a, like we did. In practice, however, we often don't and just use x, like so:

> Suppose $A \neq \emptyset$, i.e., there is an $x \in A$. By definition of $\cup$, $x \in A \cup B$. So $A \cup B \neq \emptyset$.

However, when you do this, you have to be extra careful that you use different x's and y's for different existential claims. For instance, the following is *not* a correct proof of "If $A \neq \emptyset$ and $B \neq \emptyset$ then $A \cap B \neq \emptyset$" (which is not true).

> Suppose $A \neq \emptyset$ and $B \neq \emptyset$. So for some x, $x \in A$ and also for some x, $x \in B$. Since $x \in A$ and $x \in B$, $x \in A \cap B$, by definition of $\cap$. So $A \cap B \neq \emptyset$.

Can you spot where the incorrect step occurs and explain why the result does not hold?

A.5 An Example

Our first example is the following simple fact about unions and intersections of sets. It will illustrate unpacking definitions, proofs of conjunctions, of universal claims, and proof by cases.

Proposition A.8. *For any sets A, B, and C,* $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

> Let's prove it!

Proof. We want to show that for any sets A, B, and C, $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$

> First we unpack the definition of "=" in the statement of the proposition. Recall that proving sets identical means showing that the sets have the same elements. That is, all elements of $A \cup (B \cap C)$ are also elements of $(A \cup B) \cap (A \cup C)$, and vice versa. The "vice versa" means that also every element of $(A \cup B) \cap (A \cup C)$ must be an element of $A \cup (B \cap C)$. So in unpacking the definition, we see that we have to prove a conjunction. Let's record this:

By definition, $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ iff every element of $A \cup (B \cap C)$ is also an element of $(A \cup B) \cap (A \cup C)$, and every element of $(A \cup B) \cap (A \cup C)$ is an element of $A \cup (B \cap C)$.

> Since this is a conjunction, we must prove each conjunct separately. Lets start with the first: let's prove that every element of $A \cup (B \cap C)$ is also an element of $(A \cup B) \cap (A \cup C)$.
>
> This is a universal claim, and so we consider an arbitrary element of $A \cup (B \cap C)$ and show that it must also be an element of $(A \cup B) \cap (A \cup C)$. We'll pick a variable to call this arbitrary element by, say, z. Our proof continues:

First, we prove that every element of $A \cup (B \cap C)$ is also an element of $(A \cup B) \cap (A \cup C)$. Let $z \in A \cup (B \cap C)$. We have to show that $z \in (A \cup B) \cap (A \cup C)$.

> Now it is time to unpack the definition of $\cup$ and $\cap$. For instance, the definition of $\cup$ is: $A \cup B = \{z \mid z \in A \text{ or } z \in B\}$. When we apply the definition to "$A \cup (B \cap C)$," the role of the "B" in the definition is now played by "$B \cap C$," so $A \cup (B \cap C) = \{z \mid z \in A \text{ or } z \in B \cap C\}$. So our assumption that $z \in A \cup (B \cap C)$ amounts to: $z \in \{z \mid z \in A \text{ or } z \in B \cap C\}$. And $z \in \{z \mid \ldots z \ldots\}$ iff $\ldots z \ldots$, i.e., in this case, $z \in A$ or $z \in B \cap C$.

By the definition of $\cup$, either $z \in A$ or $z \in B \cap C$.

> Since this is a disjunction, it will be useful to apply proof by cases. We take the two cases, and show that in each one, the conclusion we're aiming for (namely, "$z \in (A \cup B) \cap (A \cup C)$") obtains.

Case 1: Suppose that $z \in A$.

> There's not much more to work from based on our assumptions. So let's look at what we have to work with in the conclusion. We want to show that $z \in (A \cup B) \cap (A \cup C)$. Based on the definition of $\cap$, if we want to show that $z \in (A \cup B) \cap (A \cup C)$, we have to show that it's in both $(A \cup B)$ and $(A \cup C)$. But $z \in A \cup B$ iff $z \in A$ or $z \in B$, and we already have (as the assumption of case 1) that $z \in A$. By the same reasoning—switching C for B—$z \in A \cup C$. This argument went in the reverse direction, so let's record our reasoning in the direction needed in our proof.

Since $z \in A$, $z \in A$ or $z \in B$, and hence, by definition of $\cup$, $z \in A \cup B$. Similarly, $z \in A \cup C$. But this means that $z \in (A \cup B) \cap (A \cup C)$, by definition of $\cap$.

> This completes the first case of the proof by cases. Now we want to derive the conclusion in the second case, where $z \in B \cap C$.

Case 2: Suppose that $z \in B \cap C$.

> Again, we are working with the intersection of two sets. Let's apply the definition of $\cap$:

Since $z \in B \cap C$, z must be an element of both B and C, by definition of $\cap$.

> It's time to look at our conclusion again. We have to show that z is in both $(A \cup B)$ and $(A \cup C)$. And again, the solution is immediate.

Since $z \in B$, $z \in (A \cup B)$. Since $z \in C$, also $z \in (A \cup C)$. So, $z \in (A \cup B) \cap (A \cup C)$.

> Here we applied the definitions of $\cup$ and $\cap$ again, but since we've already recalled those definitions, and already showed that if z is in one of two sets it is in their union, we don't have to be as explicit in what we've done.
>
> We've completed the second case of the proof by cases, so now we can assert our first conclusion.

So, if $z \in A \cup (B \cap C)$ then $z \in (A \cup B) \cap (A \cup C)$.

> Now we just want to show the other direction, that every element of $(A \cup B) \cap (A \cup C)$ is an element of $A \cup (B \cap C)$. As before, we prove this universal claim by assuming we have an arbitrary element of the first set and show it must be in the second set. Let's state what we're about to do.

Now, assume that $z \in (A \cup B) \cap (A \cup C)$. We want to show that $z \in A \cup (B \cap C)$.

> We are now working from the hypothesis that $z \in (A \cup B) \cap (A \cup C)$. It hopefully isn't too confusing that we're using the same z here as in the first part of the proof. When we finished that part, all the assumptions we've made there are no longer in effect, so now we can make new assumptions about what z is. If that is confusing to you, just replace z with a different variable in what follows.
>
> We know that z is in both $A \cup B$ and $A \cup C$, by definition of $\cap$. And by the definition of $\cup$, we can further unpack this to: either $z \in A$ or $z \in B$, and also either $z \in A$ or $z \in C$. This looks like a proof by cases again—except the "and" makes it confusing. You might think that this amounts to there being three possibilities: z is either in A, B or C. But that would be a mistake. We have to be careful, so let's consider each disjunction in turn.

By definition of $\cap$, $z \in A \cup B$ and $z \in A \cup C$. By definition of $\cup$, $z \in A$ or $z \in B$. We distinguish cases.

> Since we're focusing on the first disjunction, we haven't gotten our second disjunction (from unpacking $A \cup C$) yet. In fact, we don't need it yet. The first case is $z \in A$, and an element of a set is also an element of the union of that set with any other. So case 1 is easy:

Case 1: Suppose that $z \in A$. It follows that $z \in A \cup (B \cap C)$.

> Now for the second case, $z \in B$. Here we'll unpack the second $\cup$ and do another proof-by-cases:

Case 2: Suppose that $z \in B$. Since $z \in A \cup C$, either $z \in A$ or $z \in C$. We distinguish cases further:

Case 2a: $z \in A$. Then, again, $z \in A \cup (B \cap C)$.

> Ok, this was a bit weird. We didn't actually need the assumption that $z \in B$ for this case, but that's ok.

Case 2b: $z \in C$. Then $z \in B$ and $z \in C$, so $z \in B \cap C$, and consequently, $z \in A \cup (B \cap C)$.

> This concludes both proofs-by-cases and so we're done with the second half.

So, if $z \in (A \cup B) \cap (A \cup C)$ then $z \in A \cup (B \cap C)$. □

A.6 Another Example

Proposition A.9. *If $A \subseteq C$, then $A \cup (C \setminus A) = C$.*

Proof. Suppose that $A \subseteq C$. We want to show that $A \cup (C \setminus A) = C$.

> We begin by observing that this is a conditional statement. It is tacitly universally quantified: the proposition holds for all sets A and C. So A and C are variables for arbitrary sets. To prove such a statement, we assume the antecedent and prove the consequent.
>
> We continue by using the assumption that $A \subseteq C$. Let's unpack the definition of $\subseteq$: the assumption means that all elements of A are also elements of C. Let's write this down—it's an important fact that we'll use throughout the proof.

By the definition of $\subseteq$, since $A \subseteq C$, for all z, if $z \in A$, then $z \in C$.

> We've unpacked all the definitions that are given to us in the assumption. Now we can move onto the conclusion. We want to show that $A \cup (C \setminus A) = C$, and so we set up a proof similarly to the last example: we show that every element of $A \cup (C \setminus A)$ is also an element of C and, conversely, every element of C is an element of $A \cup (C \setminus A)$. We can shorten this to: $A \cup (C \setminus A) \subseteq C$ and $C \subseteq A \cup (C \setminus A)$. (Here we're doing the opposite of unpacking a definition, but it makes the proof a bit easier to read.) Since this is a conjunction, we have to prove both parts. To show the first part, i.e., that every element of $A \cup (C \setminus A)$ is also an element of C, we assume that $z \in A \cup (C \setminus A)$ for an arbitrary z and show that $z \in C$. By the definition of $\cup$, we can conclude that $z \in A$ or $z \in C \setminus A$ from $z \in A \cup (C \setminus A)$. You should now be getting the hang of this.

$A \cup (C \setminus A) = C$ iff $A \cup (C \setminus A) \subseteq C$ and $C \subseteq (A \cup (C \setminus A)$. First we prove that $A \cup (C \setminus A) \subseteq C$. Let $z \in A \cup (C \setminus A)$. So, either $z \in A$ or $z \in (C \setminus A)$.

> We've arrived at a disjunction, and from it we want to prove that $z \in C$. We do this using proof by cases.

Case 1: $z \in A$. Since for all z, if $z \in A$, $z \in C$, we have that $z \in C$.

Here we've used the fact recorded earlier which followed from the hypothesis of the proposition that $A \subseteq C$. The first case is complete, and we turn to the second case, $z \in (C \setminus A)$. Recall that $C \setminus A$ denotes the *difference* of the two sets, i.e., the set of all elements of C which are not elements of A. But any element of C not in A is in particular an element of C.

Case 2: $z \in (C \setminus A)$. This means that $z \in C$ and $z \notin A$. So, in particular, $z \in C$.

Great, we've proved the first direction. Now for the second direction. Here we prove that $C \subseteq A \cup (C \setminus A)$. So we assume that $z \in C$ and prove that $z \in A \cup (C \setminus A)$.

Now let $z \in C$. We want to show that $z \in A$ or $z \in C \setminus A$.

Since all elements of A are also elements of C, and $C \setminus A$ is the set of all things that are elements of C but not A, it follows that z is either in A or in $C \setminus A$. This may be a bit unclear if you don't already know why the result is true. It would be better to prove it step-by-step. It will help to use a simple fact which we can state without proof: $z \in A$ or $z \notin A$. This is called the "principle of excluded middle:" for any statement p, either p is true or its negation is true. (Here, p is the statement that $z \in A$.) Since this is a disjunction, we can again use proof-by-cases.

Either $z \in A$ or $z \notin A$. In the former case, $z \in A \cup (C \setminus A)$. In the latter case, $z \in C$ and $z \notin A$, so $z \in C \setminus A$. But then $z \in A \cup (C \setminus A)$.

Our proof is complete: we have shown that $A \cup (C \setminus A) = C$. □

A.7 Proof by Contradiction

In the first instance, proof by contradiction is an inference pattern that is used to prove negative claims. Suppose you want to show that some claim p is *false*, i.e., you want to show $\neg p$. The most promising strategy is to (a) suppose that p is true, and (b) show that this assumption leads to something you know to be false. "Something known to be false" may be a result that conflicts with—contradicts—p itself, or some other hypothesis of the overall claim you are considering. For instance, a proof of "if q then $\neg p$" involves assuming that q is true and proving $\neg p$ from it. If you prove $\neg p$ by contradiction, that means assuming p in addition to q. If you can prove $\neg q$ from p, you have shown that the assumption p leads to something that contradicts your other assumption q, since q and $\neg q$ cannot both be true. Of course, you have to use other inference patterns in your proof of the contradiction, as well as unpacking definitions. Let's consider an example.

Proposition A.10. *If $A \subseteq B$ and $B = \emptyset$, then A has no elements.*

Proof. Suppose $A \subseteq B$ and $B = \emptyset$. We want to show that A has no elements.

Since this is a conditional claim, we assume the antecedent and want to prove the consequent. The consequent is: A has no elements. We can make that a bit more explicit: it's not the case that there is an $x \in A$.

A has no elements iff it's not the case that there is an x such that $x \in A$.

> So we've determined that what we want to prove is really a negative claim $\neg p$, namely: it's not the case that there is an $x \in A$. To use proof by contradiction, we have to assume the corresponding positive claim p, i.e., there is an $x \in A$, and prove a contradiction from it. We indicate that we're doing a proof by contradiction by writing "by way of contradiction, assume" or even just "suppose not," and then state the assumption p.

Suppose not: there is an $x \in A$.

> This is now the new assumption we'll use to obtain a contradiction. We have two more assumptions: that $A \subseteq B$ and that $B = \emptyset$. The first gives us that $x \in B$:

Since $A \subseteq B$, $x \in B$.

> But since $B = \emptyset$, every element of B (e.g., x) must also be an element of $\emptyset$.

Since $B = \emptyset$, $x \in \emptyset$. This is a contradiction, since by definition $\emptyset$ has no elements.

> This already completes the proof: we've arrived at what we need (a contradiction) from the assumptions we've set up, and this means that the assumptions can't all be true. Since the first two assumptions ($A \subseteq B$ and $B = \emptyset$) are not contested, it must be the last assumption introduced (there is an $x \in A$) that must be false. But if we want to be thorough, we can spell this out.

Thus, our assumption that there is an $x \in A$ must be false, hence, A has no elements by proof by contradiction. □

Every positive claim is trivially equivalent to a negative claim: p iff $\neg\neg p$. So proofs by contradiction can also be used to establish positive claims "indirectly," as follows: To prove p, read it as the negative claim $\neg\neg p$. If we can prove a contradiction from $\neg p$, we've established $\neg\neg p$ by proof by contradiction, and hence p.

In the last example, we aimed to prove a negative claim, namely that A has no elements, and so the assumption we made for the purpose of proof by contradiction (i.e., that there is an $x \in A$) was a positive claim. It gave us something to work with, namely the hypothetical $x \in A$ about which we continued to reason until we got to $x \in \emptyset$.

When proving a positive claim indirectly, the assumption you'd make for the purpose of proof by contradiction would be negative. But very often you can easily reformulate a positive claim as a negative claim, and a negative claim as a positive claim. Our previous proof would have been essentially the same had we proved "$A = \emptyset$" instead of the negative consequent "A has no elements." (By definition of $=$, "$A = \emptyset$" is a general claim, since it unpacks to "every element of A is an element of $\emptyset$ and vice versa".) But it is easily seen to be equivalent to the negative claim "not: there is an $x \in A$."

So it is sometimes easier to work with $\neg p$ as an assumption than it is to prove p directly. Even when a direct proof is just as simple or even simpler (as in the next example), some people prefer to proceed indirectly. If the double negation confuses you, think of a proof by contradiction of some claim as a proof of a contradiction from the *opposite* claim. So, a proof by contradiction of $\neg p$ is a proof of a contradiction

from the assumption p; and proof by contradiction of p is a proof of a contradiction from $\neg p$.

Proposition A.11. *$A \subseteq A \cup B$.*

Proof. We want to show that $A \subseteq A \cup B$.

> On the face of it, this is a positive claim: every $x \in A$ is also in $A \cup B$. The negation of that is: some $x \in A$ is $\notin A \cup B$. So we can prove the claim indirectly by assuming this negated claim, and showing that it leads to a contradiction.

Suppose not, i.e., $A \not\subseteq A \cup B$.

> We have a definition of $A \subseteq A \cup B$: every $x \in A$ is also $\in A \cup B$. To understand what $A \not\subseteq A \cup B$ means, we have to use some elementary logical manipulation on the unpacked definition: it's false that every $x \in A$ is also $\in A \cup B$ iff there is *some* $x \in A$ that is $\notin C$. (This is a place where you want to be very careful: many students' attempted proofs by contradiction fail because they analyze the negation of a claim like "all As are Bs" incorrectly.) In other words, $A \not\subseteq A \cup B$ iff there is an x such that $x \in A$ and $x \notin A \cup B$. From then on, it's easy.

So, there is an $x \in A$ such that $x \notin A \cup B$. By definition of $\cup$, $x \in A \cup B$ iff $x \in A$ or $x \in B$. Since $x \in A$, we have $x \in A \cup B$. This contradicts the assumption that $x \notin A \cup B$. □

Proposition A.12. *If $A \subseteq B$ and $B \subseteq C$ then $A \subseteq C$.*

Proof. Suppose $A \subseteq B$ and $B \subseteq C$. We want to show $A \subseteq C$.

> Let's proceed indirectly: we assume the negation of what we want to etablish.

Suppose not, i.e., $A \not\subseteq C$.

> As before, we reason that $A \not\subseteq C$ iff not every $x \in A$ is also $\in C$, i.e., some $x \in A$ is $\notin C$. Don't worry, with practice you won't have to think hard anymore to unpack negations like this.

In other words, there is an x such that $x \in A$ and $x \notin C$.

> Now we can use this to get to our contradiction. Of course, we'll have to use the other two assumptions to do it.

Since $A \subseteq B$, $x \in B$. Since $B \subseteq C$, $x \in C$. But this contradicts $x \notin C$. □

Proposition A.13. *If $A \cup B = A \cap B$ then $A = B$.*

Proof. Suppose $A \cup B = A \cap B$. We want to show that $A = B$.

> The beginning is now routine:

Assume, by way of contradiction, that $A \neq B$.

> Our assumption for the proof by contradiction is that $A \neq B$. Since $A = B$ iff $A \subseteq B$ an $B \subseteq A$, we get that $A \neq B$ iff $A \not\subseteq B$ *or* $B \not\subseteq A$. (Note how important it is to be careful when manipulating negations!) To prove a contradiction from this disjunction, we use a proof by cases and show that in each case, a contradiction follows.

$A \neq B$ iff $A \not\subseteq B$ or $B \not\subseteq A$. We distinguish cases.

> In the first case, we assume $A \not\subseteq B$, i.e., for some x, $x \in A$ but $\notin B$. $A \cap B$ is defined as those elements that A and B have in common, so if something isn't in one of them, it's not in the intersection. $A \cup B$ is A together with B, so anything in either is also in the union. This tells us that $x \in A \cup B$ but $x \notin A \cap B$, and hence that $A \cap B \neq B \cap A$.

Case 1: $A \not\subseteq B$. Then for some x, $x \in A$ but $x \notin B$. Since $x \notin B$, then $x \notin A \cap B$. Since $x \in A$, $x \in A \cup B$. So, $A \cap B \neq B \cap A$, contradicting the assumption that $A \cap B = A \cup B$.

Case 2: $B \not\subseteq A$. Then for some y, $y \in B$ but $y \notin A$. As before, we have $y \in A \cup B$ but $y \notin A \cap B$, and so $A \cap B \neq A \cup B$, again contradicting $A \cap B = A \cup B$. □

A.8 Reading Proofs

Proofs you find in textbooks and articles very seldom give all the details we have so far included in our examples. Authors ofen do not draw attention to when they distinguish cases, when they give an indirect proof, or don't mention that they use a definition. So when you read a proof in a textbook, you will often have to fill in those details for yourself in order to understand the proof. Doing this is also good practice to get the hang of the various moves you have to make in a proof. Let's look at an example.

Proposition A.14 (Absorption). *For all sets A, B,*

$$A \cap (A \cup B) = A$$

Proof. If $z \in A \cap (A \cup B)$, then $z \in A$, so $A \cap (A \cup B) \subseteq A$. Now suppose $z \in A$. Then also $z \in A \cup B$, and therefore also $z \in A \cap (A \cup B)$. □

The preceding proof of the absorption law is very condensed. There is no mention of any definitions used, no "we have to prove that" before we prove it, etc. Let's unpack it. The proposition proved is a general claim about any sets A and B, and when the proof mentions A or B, these are variables for arbitrary sets. The general claims the proof establishes is what's required to prove identity of sets, i.e., that every element of the left side of the identity is an element of the right and vice versa.

> "If $z \in A \cap (A \cup B)$, then $z \in A$, so $A \cap (A \cup B) \subseteq A$."

This is the first half of the proof of the identity: it estabishes that if an arbitrary z is an element of the left side, it is also an element of the right, i.e., $A \cap (A \cup B) \subseteq A$. Assume that $z \in A \cap (A \cup B)$. Since z is an element of the intersection of two sets iff it is an element of both sets, we can conclude that $z \in A$ and also $z \in A \cup B$. In particular, $z \in A$, which is what we wanted to show. Since that's all that has to be done for the first half, we know that the rest of the proof must be a proof of the second half, i.e., a proof that $A \subseteq A \cap (A \cup B)$.

> "Now suppose $z \in A$. Then also $z \in A \cup B$, and therefore also $z \in A \cap (A \cup B)$."

We start by assuming that $z \in A$, since we are showing that, for any z, if $z \in A$ then $z \in A \cap (A \cup B)$. To show that $z \in A \cap (A \cup B)$, we have to show (by definition of "$\cap$") that (i) $z \in A$ and also (ii) $z \in A \cup B$. Here (i) is just our assumption, so there is nothing further to prove, and that's why the proof does not mention it again. For (ii), recall that z is an element of a union of sets iff it is an element of at least one of those sets. Since $z \in A$, and $A \cup B$ is the union of A and B, this is the case here. So $z \in A \cup B$. We've shown both (i) $z \in A$ and (ii) $z \in A \cup B$, hence, by definition of "$\cap$," $z \in A \cap (A \cup B)$. The proof doesn't mention those definitions; it's assumed the reader has already internalized them. If you haven't, you'll have to go back and remind yourself what they are. Then you'll also have to recognize why it follows from $z \in A$ that $z \in A \cup B$, and from $z \in A$ and $z \in A \cup B$ that $z \in A \cap (A \cup B)$.

Here's another version of the proof above, with everything made explicit:

Proof. [By definition of = for sets, $A \cap (A \cup B) = A$ we have to show (a) $A \cap (A \cup B) \subseteq A$ and (b) $A \cap (A \cup B) \subseteq A$. (a): By definition of $\subseteq$, we have to show that if $z \in A \cap (A \cup B)$, then $z \in A$.] If $z \in A \cap (A \cup B)$, then $z \in A$ [since by definition of $\cap$, $z \in A \cap (A \cup B)$ iff $z \in A$ and $z \in A \cup B$], so $A \cap (A \cup B) \subseteq A$. [(b): By definition of $\subseteq$, we have to show that if $z \in A$, then $z \in A \cap (A \cup B)$.] Now suppose [(1)] $z \in A$. Then also [(2)] $z \in A \cup B$ [since by (1) $z \in A$ or $z \in B$, which by definition of $\cup$ means $z \in A \cup B$], and therefore also $z \in A \cap (A \cup B)$ [since the definition of $\cap$ requires that $z \in A$, i.e., (1), and $z \in A \cup B$), i.e., (2)]. □

A.9 I Can't Do It!

We all get to a point where we feel like giving up. But you *can* do it. Your instructor and teaching assistant, as well as your fellow students, can help. Ask them for help! Here are a few tips to help you avoid a crisis, and what to do if you feel like giving up.

To make sure you can solve problems successfully, do the following:

1. *Start as far in advance as possible.* We get busy throughout the semester and many of us struggle with procrastination, one of the best things you can do is to start your homework assignments early. That way, if you're stuck, you have time to look for a solution (that isn't crying).

2. *Talk to your classmates.* You are not alone. Others in the class may also struggle—but the may struggle with different things. Talking it out with your peers can give you a different perspective on the problem that might lead to a breakthrough. Of course, don't just copy their solution: ask them for a hint, or explain where you get stuck and ask them for the next step. And when you do get it, reciprocate. Helping someone else along, and explaining things will help you understand better, too.

3. *Ask for help.* You have many resources available to you—your instructor and teaching assistant are there for you and *want* you to succeed. They should be able to help you work out a problem and identify where in the process you're struggling.

4. *Take a break.* If you're stuck, it *might* be because you've been staring at the problem for too long. Take a short break, have a cup of tea, or work on a different problem for a while, then return to the problem with a fresh mind. Sleep on it.

Notice how these strategies require that you've started to work on the proof well in advance? If you've started the proof at 2am the day before it's due, these might not be so helpful.

This might sound like doom and gloom, but solving a proof is a challenge that pays off in the end. Some people do this as a career—so there must be something to enjoy about it. Like basically everything, solving problems and doing proofs is something that requires practice. You might see classmates who find this easy: they've probably just had lots of practice already. Try not to give in too easily.

If you do run out of time (or patience) on a particular problem: that's ok. It doesn't mean you're stupid or that you will never get it. Find out (from your instructor or another student) how it is done, and identify where you went wrong or got stuck, so you can avoid doing that the next time you encounter a similar issue. Then try to do it without looking at the solution. And next time, start (and ask for help) earlier.

A.10 Other Resources

There are many books on how to do proofs in mathematics which may be useful. Check out *How to Read and do Proofs: An Introduction to Mathematical Thought Processes* (Solow, 2013) and *How to Prove It: A Structured Approach* (Velleman, 2019) in particular. The *Book of Proof* (Hammack, 2013) and *Mathematical Reasoning* (Sandstrum, 2019) are books on proof that are freely available online. Philosophers might find *More Precisely: The Math you need to do Philosophy* (Steinhart, 2018) to be a good primer on mathematical reasoning.

There are also various shorter guides to proofs available on the internet; e.g., "Introduction to Mathematical Arguments" (Hutchings, 2003) and "How to write proofs" (Cheng, 2004).

Motivational Videos

Feel like you have no motivation to do your homework? Feeling down? These videos might help!

- `https://www.youtube.com/watch?v=ZXsQAXxao0`
- `https://www.youtube.com/watch?v=BQ4yd2W50No`
- `https://www.youtube.com/watch?v=StTqXEQ2l-Y`

Problems

Problem A.1. Suppose you are asked to prove that $A \cap B \neq \emptyset$. Unpack all the definitions occuring here, i.e., restate this in a way that does not mention "$\cap$", "=", or "$\emptyset$.

Problem A.2. Prove *indirectly* that $A \cap B \subseteq A$.

Problem A.3. Expand the following proof of $A \cup (A \cap B) = A$, where you mention all the inference patterns used, why each step follows from assumptions or claims established before it, and where we have to appeal to which definitions.

Proof. If $z \in A \cup (A \cap B)$ then $z \in A$ or $z \in A \cap B$. If $z \in A \cap B$, $z \in A$. Any $z \in A$ is also $\in A \cup (A \cap B)$. □

Appendix B

Induction

B.1 Introduction

Induction is an important proof technique which is used, in different forms, in almost all areas of logic, theoretical computer science, and mathematics. It is needed to prove many of the results in logic.

Induction is often contrasted with deduction, and characterized as the inference from the particular to the general. For instance, if we observe many green emeralds, and nothing that we would call an emerald that's not green, we might conclude that all emeralds are green. This is an inductive inference, in that it proceeds from many particlar cases (this emerald is green, that emerald is green, etc.) to a general claim (all emeralds are green). *Mathematical* induction is also an inference that concludes a general claim, but it is of a very different kind that this "simple induction."

Very roughly, an inductive proof in mathematics concludes that all mathematical objects of a certain sort have a certain property. In the simplest case, the mathematical objects an inductive proof is concerned with are natural numbers. In that case an inductive proof is used to establish that all natural numbers have some property, and it does this by showing that

1. 0 has the property, and (2)
2. whenever a number k has the property, so does $k + 1$.

Induction on natural numbers can then also often be used to prove general about mathematical objects that can be assigned numbers. For instance, finite sets each have a finite number n of elements, and if we can use induction to show that every number n has the property "all finite sets of size n are ..." then we will have shown something about all finite sets.

Induction can also be generalized to mathematical objects that are *inductively defined*. For instance, expressions of a formal language suchh as those of first-order logic are defined inductively. *Structural induction* is a way to prove results about all such expressions. Structural induction, in particular, is very useful—and widely used—in logic.

B.2 Induction on $\mathbb{N}$

In its simplest form, induction is a technique used to prove results for all natural numbers. It uses the fact that by starting from 0 and repeatedly adding 1 we eventually

reach every natural number. So to prove that something is true for every number, we can (1) establish that it is true for 0 and (2) show that whenever it is true for a number n, it is also true for the next number $n + 1$. If we abbreviate "number n has property P" by $P(n)$ (and "number k has property P" by $P(k)$, etc.), then a proof by induction that $P(n)$ for all $n \in \mathbb{N}$ consists of:

1. a proof of $P(0)$, and

2. a proof that, for any k, if $P(k)$ then $P(k + 1)$.

To make this crystal clear, suppose we have both (1) and (2). Then (1) tells us that $P(0)$ is true. If we also have (2), we know in particular that if $P(0)$ then $P(0 + 1)$, i.e., $P(1)$. This follows from the general statement "for any k, if $P(k)$ then $P(k + 1)$" by putting 0 for k. So by modus ponens, we have that $P(1)$. From (2) again, now taking 1 for n, we have: if $P(1)$ then $P(2)$. Since we've just established $P(1)$, by modus ponens, we have $P(2)$. And so on. For any number n, after doing this n times, we eventually arrive at $P(n)$. So (1) and (2) together establish $P(n)$ for any $n \in \mathbb{N}$.

Let's look at an example. Suppose we want to find out how many different sums we can throw with n dice. Although it might seem silly, let's start with 0 dice. If you have no dice there's only one possible sum you can "throw": no dots at all, which sums to 0. So the number of different possible throws is 1. If you have only one die, i.e., $n = 1$, there are six possible values, 1 through 6. With two dice, we can throw any sum from 2 through 12, that's 11 possibilities. With three dice, we can throw any number from 3 to 18, i.e., 16 different possibilities. 1, 6, 11, 16: looks like a pattern: maybe the answer is $5n + 1$? Of course, $5n + 1$ is the maximum possible, because there are only $5n + 1$ numbers between n, the lowest value you can throw with n dice (all 1's) and $6n$, the highest you can throw (all 6's).

Theorem B.1. *With n dice one can throw all $5n + 1$ possible values between n and $6n$.*

Proof. Let $P(n)$ be the claim: "It is possible to throw any number between n and $6n$ using n dice." To use induction, we prove:

1. The *induction basis* $P(1)$, i.e., with just one die, you can throw any number between 1 and 6.

2. The *induction step*, for all k, if $P(k)$ then $P(k + 1)$.

(1) Is proved by inspecting a 6-sided die. It has all 6 sides, and every number between 1 and 6 shows up one on of the sides. So it is possible to throw any number between 1 and 6 using a single die.

To prove (2), we assume the antecedent of the conditional, i.e., $P(k)$. This assumption is called the *inductive hypothesis*. We use it to prove $P(k + 1)$. The hard part is to find a way of thinking about the possible values of a throw of $k + 1$ dice in terms of the possible values of throws of k dice plus of throws of the extra $k + 1$-st die—this is what we have to do, though, if we want to use the inductive hypothesis.

The inductive hypothesis says we can get any number between k and $6k$ using k dice. If we throw a 1 with our $(k + 1)$-st die, this adds 1 to the total. So we can throw any value between $k + 1$ and $6k + 1$ by throwing 5 dice and then rolling a 1 with the $(k + 1)$-st die. What's left? The values $6k + 2$ through $6k + 6$. We can get these by rolling k 6s and then a number between 2 and 6 with our $(k + 1)$-st die. Together, this means that with $k + 1$ dice we can throw any of the numbers between $k + 1$

and $6(k+1)$, i.e., we've proved $P(k+1)$ using the assumption $P(k)$, the inductive hypothesis. □

Very often we use induction when we want to prove something about a series of objects (numbers, sets, etc.) that is itself defined "inductively," i.e., by defining the $(n+1)$-st object in terms of the n-th. For instance, we can define the sum s_n of the natural numbers up to n by

$$\begin{aligned} s_0 &= 0 \\ s_{n+1} &= s_n + (n+1) \end{aligned}$$

This definition gives:

$$\begin{aligned} s_0 &= 0, \\ s_1 &= s_0 + 1 && = 1, \\ s_2 &= s_1 + 2 && = 1 + 2 = 3 \\ s_3 &= s_2 + 3 && = 1 + 2 + 3 = 6, \text{ etc.} \end{aligned}$$

Now we can prove, by induction, that $s_n = n(n+1)/2$.

Proposition B.2. *$s_n = n(n+1)/2$.*

Proof. We have to prove (1) that $s_0 = 0 \cdot (0+1)/2$ and (2) if $s_k = k(k+1)/2$ then $s_{k+1} = (k+1)(k+2)/2$. (1) is obvious. To prove (2), we assume the inductive hypothesis: $s_k = k(k+1)/2$. Using it, we have to show that $s_{k+1} = (k+1)(k+2)/2$.

What is s_{k+1}? By the definition, $s_{k+1} = s_k + (k+1)$. By inductive hypothesis, $s_k = k(k+1)/2$. We can substitute this into the previous equation, and then just need a bit of arithmetic of fractions:

$$\begin{aligned} s_{k+1} &= \frac{k(k+1)}{2} + (k+1) = \\ &= \frac{k(k+1)}{2} + \frac{2(k+1)}{2} = \\ &= \frac{n(k+1) + 2(k+1)}{2} = \\ &= \frac{(k+2)(k+1)}{2}. \end{aligned}$$

□

The important lesson here is that if you're proving something about some inductively defined sequence a_n, induction is the obvious way to go. And even if it isn't (as in the case of the possibilities of dice throws), you can use induction if you can somehow relate the case for $k+1$ to the case for k.

B.3 Strong Induction

In the principle of induction discussed above, we prove $P(0)$ and also if $P(k)$, then $P(k+1)$. In the second part, we assume that $P(k)$ is true and use this assumption to prove $P(k+1)$. Equivalently, of course, we could assume $P(k-1)$ and use it to prove $P(k)$—the important part is that we be able to carry out the inference from any number to its successor; that we can prove the claim in question for any number under the assumption it holds for its predecessor.

There is a variant of the principle of induction in which we don't just assume that the claim holds for the predecessor $k-1$ of k, but for all numbers smaller than k, and use this assumption to establish the claim for k. This also gives us the claim $P(n)$ for all $n \in \mathbb{N}$. For once we have established $P(0)$, we have thereby established that P holds for all numbers less than 1. And if we know that if $P(l)$ for all $l < k$, then $P(k)$, we know this in particular for $k = 1$. So we can conclude $P(1)$. With this we have proved $P(0)$ and $P(1)$, i.e., $P(l)$ for all $l < 2$, and since we have also the conditional, if $P(l)$ for all $l < 2$, then $P(2)$, we can conclude $P(2)$, and so on.

In fact, if we can establish the general conditional "for all k, if $P(l)$ for all $l < k$, then $P(k)$," we do not have to establish $P(0)$ anymore, since it follows from it. For remember that a general claim like "for all $l < k$, $P(l)$" is true if there are no $l < k$. This is a case of vacuous quantification: "all As are Bs" is true if there are no As, $\forall x\,(\varphi(x) \to \psi(x))$ is true if no x satisfies $\varphi(x)$. In this case, the formalized version would be "$\forall l\,(l < k \to P(l))$"—and that is true if there are no $l < k$. And if $k = 0$ that's exactly the case: no $l < 0$, hence "for all $l < 0$, $P(0)$" is true, whatever P is. A proof of "if $P(l)$ for all $l < k$, then $P(k)$" thus automatically establishes $P(0)$.

This variant is useful if establishing the claim for k can't be made to just rely on the claim for $k-1$ but may require the assumption that it is true for one or more $l < k$.

B.4 Inductive Definitions

In logic we very often define kinds of objects *inductively*, i.e., by specifying rules for what counts as an object of the kind to be defined which explain how to get new objects of that kind from old objects of that kind. For instance, we often define special kinds of sequences of symbols, such as the terms and formulas of a language, by induction. For a simple example, consider strings of consisting of letters a, b, c, d, the symbol ∘, and brackets [and], such as "[[c ∘ d][", "[a[]∘]", "a" or "[[a ∘ b] ∘ d]". You probably feel that there's something "wrong" with the first two strings: the brackets don't "balance" at all in the first, and you might feel that the "∘" should "connect" expressions that themselves make sense. The third and fourth string look better: for every "[" there's a closing "]" (if there are any at all), and for any ∘ we can find "nice" expressions on either side, surrounded by a pair of parenteses.

We would like to precisely specify what counts as a "nice term." First of all, every letter by itself is nice. Anything that's not just a letter by itself should be of the form "$[t \circ s]$" where s and t are themselves nice. Conversely, if t and s are nice, then we can form a new nice term by putting a ∘ between them and surround them by a pair of brackets. We might use these operations to *define* the set of nice terms. This is an *inductive definition*.

Definition B.3 (Nice terms). The set of *nice terms* is inductively defined as follows:

1. Any letter a, b, c, d is a nice term.
2. If s_1 and s_2 are nice terms, then so is $[s_1 \circ s_2]$.
3. Nothing else is a nice term.

This definition tells us that something counts as a nice term iff it can be constructed according to the two conditions (1) and (2) in some finite number of steps. In the first

step, we construct all nice terms just consisting of letters by themselves, i.e.,

$$\text{a, b, c, d}$$

In the second step, we apply (2) to the terms we've constructed. We'll get

$$[\text{a} \circ \text{a}], [\text{a} \circ \text{b}], [\text{b} \circ \text{a}], \ldots, [\text{d} \circ \text{d}]$$

for all combinations of two letters. In the third step, we apply (2) again, to any two nice terms we've constructed so far. We get new nice term such as $[\text{a} \circ [\text{a} \circ \text{a}]]$—where t is a from step 1 and s is $[\text{a} \circ \text{a}]$ from step 2—and $[[\text{b} \circ \text{c}] \circ [\text{d} \circ \text{b}]]$ constructed out of the two terms $[\text{b} \circ \text{c}]$ and $[\text{d} \circ \text{b}]$ from step 2. And so on. Clause (3) rules out that anything not constructed in this way sneaks into the set of nice terms.

Note that we have not yet proved that every sequence of symbols that "feels" nice is nice according to this definition. However, it should be clear that everything we can construct does in fact "feel nice:" brackets are balanced, and $\circ$ connects parts that are themselves nice.

The key feature of inductive definitions is that if you want to prove something about all nice terms, the definition tells you which cases you must consider. For instance, if you are told that t is a nice term, the inductive definition tells you what t can look like: t can be a letter, or it can be $[s_1 \circ s_2]$ for some pair of nice terms s_1 and s_2. Because of clause (3), those are the only possibilities.

When proving claims about all of an inductively defined set, the strong form of induction becomes particularly important. For instance, suppose we want to prove that for every nice term of length n, the number of [in it is $< n/2$. This can be seen as a claim about all n: for every n, the number of [in any nice term of length n is $< n/2$.

Proposition B.4. *For any n, the number of [in a nice term of length n is $< n/2$.*

Proof. To prove this result by (strong) induction, we have to show that the following conditional claim is true:

> If for every $l < k$, any nice term of length l has $l/2$ ['s, then any nice term of length k has $k/2$ ['s.

To show this conditional, assume that its antecedent is true, i.e., assume that for any $l < k$, nice terms of length l contain $< l/2$ ['s. We call this assumption the inductive hypothesis. We want to show the same is true for nice terms of length k.

So suppose t is a nice term of length k. Because nice terms are inductively defined, we have two cases: (1) t is a letter by itself, or (2) t is $[s_1 \circ s_2]$ for some nice terms s_1 and s_2.

1. t is a letter. Then $k = 1$, and the number of [in t is 0. Since $0 < 1/2$, the claim holds.

2. t is $[s_1 \circ s_2]$ for some nice terms s_1 and s_2. Let's let l_1 be the length of s_1 and l_2 be the length of s_2. Then the length k of t is $l_1 + l_2 + 3$ (the lengths of s_1 and s_2 plus three symbols [, $\circ$,]). Since $l_1 + l_2 + 3$ is always greater than l_1, $l_1 < k$. Similarly, $l_2 < n$. That means that the induction hypothesis applies to the terms s_1 and s_2: the number m_1 of [in s_1 is $< l_1/2$, and the number m_2 of [in s_2 is $< l_2/2$.

The number of [in t is the number of [in s_1, plus the number of [in s_2, plus 1, i.e., it is $m_1 + m_2 + 1$. Since $m_1 < l_1/2$ and $m_2 < l_2/2$ we have:

$$m_1 + m_2 + 1 < \frac{l_1}{2} + \frac{l_2}{2} + 1 = \frac{l_1 + l_2 + 2}{2} < \frac{l_1 + l - 2 + 3}{2} = k/2.$$

In each case, we've shown that the number of [in t is $< k/2$ (on the basis of the inductive hypothesis). By strong induction, the proposition follows. □

B.5 Structural Induction

So far we have used induction to establish results about all natural numbers. But a corresponding principle can be used directly to prove results about all elements of an inductively defined set. This often called *structural* induction, because it depends on the structure of the inductively defined objects.

Generally, an inductive definition is given by (a) a list of "initial" elements of the set and (b) a list of operations which produce new elements of the set from old ones. In the case of nice terms, for instance, the initial objects are the letters. We only have one operation: the operations are

$$o(s_1, s_2) = [s_1 \circ s_2]$$

You can even think of the natural numbers $\mathbb{N}$ themselves as being given be an inductive definition: the initial object is 0, and the operation is the successor function $x + 1$.

In order to prove something about all elements of an inductively defined set, i.e., that every element of the set has a property P, we must:

1. Prove that the initial objects have P
2. Prove that for each operation o, if the arguments have P, so does the result.

For instance, in order to prove something about all nice terms, we would prove that it is true about all letters, and that it is true about $[s_1 \circ s_2]$ provided it is true of s_1 and s_2 individually.

Proposition B.5. *The number of* [*equals the number of*] *in any nice term* t.

Proof. We use structural induction. Nice terms are inductively defined, with letters as initial objects and the operations o for constructing new nice terms out of old ones.

1. The claim is true for every letter, since the number of [in a letter by itself is 0 and the number of] in it is also 0.
2. Suppose the number of [in s_1 equals the number of], and the same is true for s_2. The number of [in $o(s_1, s_2)$, i.e., in $[s_1 \circ s_2]$, is the sum of the number of [in s_1 and s_2. The number of] in $o(s_1, s_2)$ is the sum of the number of] in s_1 and s_2. Thus, the number of [in $o(s_1, s_2)$ equals the number of] in $o(s_1, s_2)$. □

Let's give another proof by structural induction: a proper initial segment of a string t of symbols is any string s that agrees with t symbol by symbol, read from the left, but t is longer. So, e.g., $[a \circ$ is a proper initial segment of $[a \circ b]$, but neither are $[b \circ$ (they disagree at the second symbol) nor $[a \circ b]$ (they are the same length).

Proposition B.6. *Every proper initial segment of a nice term t has more ['s than] 's.*

Proof. By induction on t:

1. t is a letter by itself: Then t has no proper initial segments.
2. $t = [s_1 \circ s_2]$ for some nice terms s_1 and s_2. If r is a proper initial segment of t, there are a number of possibilities:
   a) r is just [: Then r has one more [than it does].
   b) r is $[r_1$ where r_1 is a proper initial segment of s_1: Since s_1 is a nice term, by induction hypothesis, r_1 has more [than] and the same is true for $[r_1$.
   c) r is $[s_1$ or $[s_1 \circ$: By the previous result, the number of [and] in s_1 are equal; so the number of [in $[s_1$ or $[s_1 \circ$ is one more than the number of].
   d) r is $[s_1 \circ r_2$ where r_2 is a proper initial segment of s_2: By induction hypothesis, r_2 contains more [than]. By the previous result, the number of [and of] in s_1 are equal. So the number of [in $[s_1 \circ r_2$ is greater than the number of].
   e) r is $[s_1 \circ s_2$: By the previous result, the number of [and] in s_1 are equal, and the same for s_2. So there is one more [in $[s_1 \circ s_2$ than there are]. □

B.6 Relations and Functions

When we have defined a set of objects (such as the natural numbers or the nice terms) inductively, we can also define *relations on* these objects by induction. For instance, consider the following idea: a nice term t_1 is a subterm of a nice term t_2 if it occurs as a part of it. Let's use a symbol for it: $t_1 \sqsubseteq t_2$. Every nice term is a subterm of itself, of course: $t \sqsubseteq t$. We can give an inductive definition of this relation as follows:

Definition B.7. The relation of a nice term t_1 being a subterm of t_2, $t_1 \sqsubseteq t_2$, is defined by induction on t_2 as follows:

1. If t_2 is a letter, then $t_1 \sqsubseteq t_2$ iff $t_1 = t_2$.
2. If t_2 is $[s_1 \circ s_2]$, then $t_1 \sqsubseteq t_2$ iff $t = t_2$, $t_1 \sqsubseteq s_1$, or $t_1 \sqsubseteq s_2$.

This definition, for instance, will tell us that a $\sqsubseteq$ [b∘a]. For (2) says that a $\sqsubseteq$ [b∘a] iff a = [b ∘ a], or a $\sqsubseteq$ *b*, or a $\sqsubseteq$ a. The first two are false: a clearly isn't identical to [b ∘ a], and by (1), a $\sqsubseteq$ b iff a = b, which is also false. However, also by (1), a $\sqsubseteq$ a iff a = a, which is true.

It's important to note that the success of this definition depends on a fact that we haven't proved yet: every nice term t is either a letter by itself, or there are *uniquely determined* nice terms s_1 and s_2 such that $t = [s_1 \circ s_2]$. "Uniquely determined" here means that if $t = [s_1 \circ s_2]$ it isn't *also* $= [r_1 \circ r_2]$ with $s_1 \neq r_1$ or $s_2 \neq r_2$. If this were the case, then clause (2) may come in conflict with itself: reading t_2 as $[s_1 \circ s_2]$ we might get $t_1 \sqsubseteq t_2$, but if we read t_2 as $[r_1 \circ r_2]$ we might get not $t_1 \sqsubseteq t_2$. Before we prove that this can't happen, let's look at an example where it *can* happen.

Definition B.8. Define *bracketless terms* inductively by

1. Every letter is a bracketles term.

2. If s_1 and s_2 are bracketless terms, then $s_1 \circ s_2$ is a bracketless term.

3. Nothing else is a bracketless term.

Bracketless terms are, e.g., a, b ∘ d, b ∘ a ∘ b. Now if we defined "subterm" for bracketless terms the way we did above, the second clause would read

$$\text{If } t_2 = s_1 \circ s_2, \text{ then } t_1 \sqsubseteq t_2 \text{ iff } t_1 = t_2,\ t_1 \sqsubseteq s_1, \text{ or } t_1 \sqsubseteq s_2.$$

Now b ∘ a ∘ b is of the form $s_1 \circ s_2$ with

$$s_1 = \text{b and} \qquad\qquad s_2 = \text{a} \circ \text{b}.$$

It is also of the form $r_1 \circ r_2$ with

$$r_1 = \text{b} \circ \text{a and} \qquad\qquad r_2 = \text{b}.$$

Now is a ∘ b a subterm of b ∘ a ∘ b? The answer is yes if we go by the first reading, and no if we go by the second.

The property that the way a nice term is built up from other nice terms is unique is called *unique readability*. Since inductive definitions of relations for such inductively defined objects are important, we have to prove that it holds.

Proposition B.9. *Suppose t is a nice term. Then either t is a letter by itself, or there are uniquely determined nice terms s_1, s_2 such that $t = [s_1 \circ s_2]$.*

Proof. If t is a letter by itself, the condition is satisfied. So assume t isn't a letter by itself. We can tell from the inductive definition that then t must be of the form $[s_1 \circ s_2]$ for some nice terms s_1 and s_2. It remains to show that these are uniquely determined, i.e., if $t = [r_1 \circ r_2]$, then $s_1 = r_1$ and $s_2 = r_2$.

So suppose $t = [s_1 \circ s_2]$ and also $t = [r_1 \circ r_2]$ for nice terms s_1, s_2, r_1, r_2. We have to show that $s_1 = r_1$ and $s_2 = r_2$. First, s_1 and r_1 must be identical, for otherwise one is a proper initial segment of the other. But by Proposition B.6, that is impossible if s_1 and r_1 are both nice terms. But if $s_1 = r_1$, then clearly also $s_2 = r_2$. □

We can also define functions inductively: e.g., we can define the function f that maps any nice term to the maximum depth of nested [...] in it as follows:

Definition B.10. The *depth* of a nice term, $f(t)$, is defined inductively as follows:

$$f(t) = \begin{cases} 0 & \text{if } t \text{ is a letter} \\ \max(f(s), f(s')) + 1 & \text{if } t = [s_1 \circ s_2]. \end{cases}$$

For instance

$$\begin{aligned} f([\text{a} \circ \text{b}]) &= \max(f(\text{a}), f(\text{b})) + 1 = \\ &= \max(0, 0) + 1 = 1, \text{ and} \\ f([[\text{a} \circ \text{b}] \circ \text{c}]) &= \max(f([\text{a} \circ \text{b}]), f(\text{c})) + 1 = \\ &= \max(1, 0) + 1 = 2. \end{aligned}$$

Here, of course, we assume that s_1 an s_2 are nice terms, and make use of the fact that every nice term is either a letter or of the form $[s_1 \circ s_2]$. It is again important

that it can be of this form in only one way. To see why, consider again the bracketless terms we defined earlier. The corresponding "definition" would be:

$$g(t) = \begin{cases} 0 & \text{if } t \text{ is a letter} \\ \max(g(s), g(s')) + 1 & \text{if } t = [s_1 \circ s_2]. \end{cases}$$

Now consider the bracketless term a ∘ b ∘ c ∘ d. It can be read in more than one way, e.g., as $s_1 \circ s_2$ with

$$s_1 = \text{a and} \qquad s_2 = \text{b} \circ \text{c} \circ \text{d},$$

or as $r_1 \circ r_2$ with

$$r_1 = \text{a} \circ b \text{ and} \qquad r_2 = \text{c} \circ \text{d}.$$

Calculating g according to the first way of reading it would give

$$\begin{aligned} g(s_1 \circ s_2) &= \max(g(\text{a}), g(\text{b} \circ \text{c} \circ \text{d})) + 1 = \\ &= \max(0, 2) + 1 = 3 \end{aligned}$$

while according to the other reading we get

$$\begin{aligned} g(r_1 \circ r_2) &= \max(g(\text{a} \circ \text{b}), g(\text{c} \circ \text{d})) + 1 = \\ &= \max(1, 1) + 1 = 2 \end{aligned}$$

But a function must always yield a unique value; so our "definition" of g doesn't define a function at all.

Problems

Problem B.1. Define the set of supernice terms by

1. Any letter a, b, c, d is a supernice term.
2. If s is a supernice term, then so is $[s]$.
3. If s_1 and s_2 are supernice terms, then so is $[s_1 \circ s_2]$.
4. Nothing else is a supernice term.

Show that the number of [in a supernice term t of length n is $\leq n/2 + 1$.

Problem B.2. Prove by structural induction that no nice term starts with].

Problem B.3. Give an inductive definition of the function l, where $l(t)$ is the number of symbols in the nice term t.

Problem B.4. Prove by structural induction on nice terms t that $f(t) < l(t)$ (where $l(t)$ is the number of symbols in t and $f(t)$ is the depth of t as defined in Definition B.10).

Appendix C

Biographies

C.1 Georg Cantor

An early biography of Georg Cantor (GAY-org KAHN-tor) claimed that he was born and found on a ship that was sailing for Saint Petersburg, Russia, and that his parents were unknown. This, however, is not true; although he was born in Saint Petersburg in 1845.

Cantor received his doctorate in mathematics at the University of Berlin in 1867. He is known for his work in set theory, and is credited with founding set theory as a distinctive research discipline. He was the first to prove that there are infinite sets of different sizes. His theories, and especially his theory of infinities, caused much debate among mathematicians at the time, and his work was controversial.

Cantor's religious beliefs and his mathematical work were inextricably tied; he even claimed that the theory of transfinite numbers had been communicated to him directly by God. In later life, Cantor suffered from mental illness. Beginning in 1984, and more frequently towards his later years, Cantor was hospitalized. The heavy criticism of his work, including a falling out with the mathematician Leopold Kronecker, led to depression and a lack of interest in mathematics. During depressive episodes, Cantor would turn to philosophy and literature, and even published a theory that Francis Bacon was the author of Shakespeare's plays.

Cantor died on January 6, 1918, in a sanatorium in Halle.

Further Reading For full biographies of Cantor, see Dauben (1990) and Grattan-Guinness (1971). Cantor's radical views are also described in the BBC Radio 4 program *A Brief History of Mathematics* (du Sautoy, 2014). If you'd like to hear about Cantor's theories in rap form, see Rose (2012).

C.2 Alonzo Church

Alonzo Church was born in Washington, DC on June 14, 1903. In early childhood, an air gun incident left Church blind in one eye. He finished preparatory school in Connecticut in 1920 and began his university education at Princeton that same year. He completed his doctoral studies in 1927. After a couple years abroad, Church returned to Princeton. Church was known exceedingly polite and careful. His blackboard writing was immaculate, and he would preserve important papers by carefully covering them in Duco cement (a clear glue). Outside of his academic pursuits, he

enjoyed reading science fiction magazines and was not afraid to write to the editors if he spotted any inaccuracies in the writing.

Church's academic achievements were great. Together with his students Stephen Kleene and Barkley Rosser, he developed a theory of effective calculability, the lambda calculus, independently of Alan Turing's development of the Turing machine. The two definitions of computability are equivalent, and give rise to what is now known as the *Church-Turing Thesis*, that a function of the natural numbers is effectively computable if and only if it is computable via Turing machine (or lambda calculus). He also proved what is now known as *Church's Theorem*: The decision problem for the validity of first-order formulas is unsolvable.

Church continued his work into old age. In 1967 he left Princeton for UCLA, where he was professor until his retirement in 1990. Church passed away on August 1, 1995 at the age of 92.

Further Reading For a brief biography of Church, see Enderton (2019). Church's original writings on the lambda calculus and the Entscheidungsproblem (Church's Thesis) are Church (1936a,b). Aspray (1984) records an interview with Church about the Princeton mathematics community in the 1930s. Church wrote a series of book reviews of the *Journal of Symbolic Logic* from 1936 until 1979. They are all archived on John MacFarlane's website (MacFarlane, 2015).

C.3 Gerhard Gentzen

Gerhard Gentzen is known primarily as the creator of structural proof theory, and specifically the creation of the natural deduction and sequent calculus proof systems. He was born on November 24, 1909 in Greifswald, Germany. Gerhard was home-schooled for three years before attending preparatory school, where he was behind most of his classmates in terms of education. Despite this, he was a brilliant student and showed a strong aptitude for mathematics. His interests were varied, and he, for instance, also write poems for his mother and plays for the school theatre.

Gentzen began his university studies at the University of Greifswald, but moved around to Göttingen, Munich, and Berlin. He received his doctorate in 1933 from the University of Göttingen under Hermann Weyl. (Paul Bernays supervised most of his work, but was dismissed from the university by the Nazis.) In 1934, Gentzen began work as an assistant to David Hilbert. That same year he developed the sequent calculus and natural deduction proof systems, in his papers *Untersuchungen über das logische Schließen I–II [Investigations Into Logical Deduction I–II]*. He proved the consistency of the Peano axioms in 1936.

Gentzen's relationship with the Nazis is complicated. At the same time his mentor Bernays was forced to leave Germany, Gentzen joined the university branch of the SA, the Nazi paramilitary organization. Like many Germans, he was a member of the Nazi party. During the war, he served as a telecommunications officer for the air intelligence unit. However, in 1942 he was released from duty due to a nervous breakdown. It is unclear whether or not Gentzen's loyalties lay with the Nazi party, or whether he joined the party in order to ensure academic success.

In 1943, Gentzen was offered an academic position at the Mathematical Institute of the German University of Prague, which he accepted. However, in 1945 the citizens of Prague revolted against German occupation. Soviet forces arrived in the city and arrested all the professors at the university. Because of his membership in Nazi

organizations, Gentzen was taken to a forced labour camp. He died of malnutrition while in his cell on August 4, 1945 at the age of 35.

Further Reading For a full biography of Gentzen, see Menzler-Trott (2007). An interesting read about mathematicians under Nazi rule, which gives a brief note about Gentzen's life, is given by Segal (2014). Gentzen's papers on logical deduction are available in the original german (Gentzen, 1935a,b). English translations of Gentzen's papers have been collected in a single volume by Szabo (1969), which also includes a biographical sketch.

C.4 Kurt Gödel

Kurt Gödel (GER-dle) was born on April 28, 1906 in Brünn in the Austro-Hungarian empire (now Brno in the Czech Republic). Due to his inquisitive and bright nature, young Kurtele was often called "Der kleine Herr Warum" (Little Mr. Why) by his family. He excelled in academics from primary school onward, where he got less than the highest grade only in mathematics. Gödel was often absent from school due to poor health and was exempt from physical education. He was diagnosed with rheumatic fever during his childhood. Throughout his life, he believed this permanently affected his heart despite medical assessment saying otherwise.

Gödel began studying at the University of Vienna in 1924 and completed his doctoral studies in 1929. He first intended to study physics, but his interests soon moved to mathematics and especially logic, in part due to the influence of the philosopher Rudolf Carnap. His dissertation, written under the supervision of Hans Hahn, proved the completeness theorem of first-order predicate logic with identity (Gödel, 1929). Only a year later, he obtained his most famous results—the first and second incompleteness theorems (published in Gödel 1931). During his time in Vienna, Gödel was heavily involved with the Vienna Circle, a group of scientifically-minded philosophers that included Carnap, whose work was especially influenced by Gödel's results.

In 1938, Gödel married Adele Nimbursky. His parents were not pleased: not only was she six years older than him and already divorced, but she worked as a dancer in a nightclub. Social pressures did not affect Gödel, however, and they remained happily married until his death.

After Nazi Germany annexed Austria in 1938, Gödel and Adele emigrated to the United States, where he took up a position at the Institute for Advanced Study in Princeton, New Jersey. Despite his introversion and eccentric nature, Gödel's time at Princeton was collaborative and fruitful. He published essays in set theory, philosophy and physics. Notably, he struck up a particularly strong friendship with his colleague at the IAS, Albert Einstein.

In his later years, Gödel's mental health deteriorated. His wife's hospitalization in 1977 meant she was no longer able to cook his meals for him. Having suffered from mental health issues throughout his life, he succumbed to paranoia. Deathly afraid of being poisoned, Gödel refused to eat. He died of starvation on January 14, 1978, in Princeton.

Further Reading For a complete biography of Gödel's life is available, see John Dawson (1997). For further biographical pieces, as well as essays about Gödel's contributions to logic and philosophy, see Wang (1990), Baaz et al. (2011), Takeuti et al. (2003), and Sigmund et al. (2007).

Gödel's PhD thesis is available in the original German (Gödel, 1929). The original text of the incompleteness theorems is (Gödel, 1931). All of Gödel's published and unpublished writings, as well as a selection of correspondence, are available in English in his *Collected Papers* Feferman et al. (1986, 1990).

For a detailed treatment of Gödel's incompleteness theorems, see Smith (2013). For an informal, philosophical discussion of Gödel's theorems, see Mark Linsenmayer's podcast (Linsenmayer, 2014).

C.5 Emmy Noether

Emmy Noether (NER-ter) was born in Erlangen, Germany, on March 23, 1882, to an upper-middle class scholarly family. Hailed as the "mother of modern algebra," Noether made groundbreaking contributions to both mathematics and physics, despite significant barriers to women's education. In Germany at the time, young girls were meant to be educated in arts and were not allowed to attend college preparatory schools. However, after auditing classes at the Universities of Göttingen and Erlangen (where her father was professor of mathematics), Noether was eventually able to enrol as a student at Erlangen in 1904, when their policy was updated to allow female students. She received her doctorate in mathematics in 1907.

Despite her qualifications, Noether experienced much resistance during her career. From 1908–1915, she taught at Erlangen without pay. During this time, she caught the attention of David Hilbert, one of the world's foremost mathematicians of the time, who invited her to Göttingen. However, women were prohibited from obtaining professorships, and she was only able to lecture under Hilbert's name, again without pay. During this time she proved what is now known as Noether's theorem, which is still used in theoretical physics today. Noether was finally granted the right to teach in 1919. Hilbert's response to continued resistance of his university colleagues reportedly was: "Gentlemen, the faculty senate is not a bathhouse."

In the later 1920s, she concentrated on work in abstract algebra, and her contributions revolutionized the field. In her proofs she often made use of the so-called ascending chain condition, which states that there is no infinite strictly increasing chain of certain sets. For instance, certain algebraic structures now known as Noetherian rings have the property that there are no infinite sequences of ideals $I_1 \subsetneq I_2 \subsetneq \dots$. The condition can be generalized to any partial order (in algebra, it concerns the special case of ideals ordered by the subset relation), and we can also consider the dual descending chain condition, where every strictly *de*creasing sequence in a partial order eventually ends. If a partial order satisfies the descending chain condition, it is possible to use induction along this order in a similar way in which we can use induction along the $<$ order on $\mathbb{N}$. Such orders are called *well-founded* or *Noetherian*, and the corresponding proof principle *Noetherian induction*.

Noether was Jewish, and when the Nazis came to power in 1933, she was dismissed from her position. Luckily, Noether was able to emigrate to the United States for a temporary position at Bryn Mawr, Pennsylvania. During her time there she also lectured at Princeton, although she found the university to be unwelcoming to women (Dick, 1981, 81). In 1935, Noether underwent an operation to remove a uterine tumour. She died from an infection as a result of the surgery, and was buried at Bryn Mawr.

Further Reading For a biography of Noether, see Dick (1981). The Perimeter Institute for Theoretical Physics has their lectures on Noether's life and influence

available online (Institute, 2015). If you're tired of reading, *Stuff You Missed in History Class* has a podcast on Noether's life and influence (Frey and Wilson, 2015). The collected works of Noether are available in the original German (Jacobson, 1983).

C.6 Rózsa Péter

Rózsa Péter was born Rósza Politzer, in Budapest, Hungary, on February 17, 1905. She is best known for her work on recursive functions, which was essential for the creation of the field of recursion theory.

Péter was raised during harsh political times—WWI raged when she was a teenager—but was able to attend the affluent Maria Terezia Girls' School in Budapest, from where she graduated in 1922. She then studied at Pázmány Péter University (later renamed Loránd Eötvös University) in Budapest. She began studying chemistry at the insistence of her father, but later switched to mathematics, and graduated in 1927. Although she had the credentials to teach high school mathematics, the economic situation at the time was dire as the Great Depression affected the world economy. During this time, Péter took odd jobs as a tutor and private teacher of mathematics. She eventually returned to university to take up graduate studies in mathematics. She had originally planned to work in number theory, but after finding out that her results had already been proven, she almost gave up on mathematics altogether. She was encouraged to work on Gödel's incompleteness theorems, and unknowingly proved several of his results in different ways. This restored her confidence, and Péter went on to write her first papers on recursion theory, inspired by David Hilbert's foundational program. She received her PhD in 1935, and in 1937 she became an editor for the *Journal of Symbolic Logic*.

Péter's early papers are widely credited as founding contributions to the field of recursive function theory. In Péter (1935a), she investigated the relationship between different kinds of recursion. In Péter (1935b), she showed that a certain recursively defined function is not primitive recursive. This simplified an earlier result due to Wilhelm Ackermann. Péter's simplified function is what's now often called the Ackermann function—and sometimes, more properly, the Ackermann-Péter function. She wrote the first book on recursive function theory (Péter, 1951).

Despite the importance and influence of her work, Péter did not obtain a full-time teaching position until 1945. During the Nazi occupation of Hungary during World War II, Péter was not allowed to teach due to anti-Semitic laws. In 1944 the government created a Jewish ghetto in Budapest; the ghetto was cut off from the rest of the city and attended by armed guards. Péter was forced to live in the ghetto until 1945 when it was liberated. She then went on to teach at the Budapest Teachers Training College, and from 1955 onward at Eötvös Loránd University. She was the first female Hungarian mathematician to become an Academic Doctor of Mathematics, and the first woman to be elected to the Hungarian Academy of Sciences.

Péter was known as a passionate teacher of mathematics, who preferred to explore the nature and beauty of mathematical problems with her students rather than to merely lecture. As a result, she was affectionately called "Aunt Rosa" by her students. Péter died in 1977 at the age of 71.

Further Reading For more biographical reading, see (O'Connor and Robertson, 2014) and (Andrásfai, 1986). Tamassy (1994) conducted a brief interview with Péter. For a fun read about mathematics, see Péter's book *Playing With Infinity* (Péter, 2010).

C.7 Julia Robinson

Julia Bowman Robinson was an American mathematician. She is known mainly for her work on decision problems, and most famously for her contributions to the solution of Hilbert's tenth problem. Robinson was born in St. Louis, Missouri, on December 8, 1919. Robinson recalls being intrigued by numbers already as a child (Reid, 1986, 4). At age nine she contracted scarlet fever and suffered from several recurrent bouts of rheumatic fever. This forced her to spend much of her time in bed, putting her behind in her education. Although she was able to catch up with the help of private tutors, the physical effects of her illness had a lasting impact on her life.

Despite her childhood struggles, Robinson graduated high school with several awards in mathematics and the sciences. She started her university career at San Diego State College, and transferred to the University of California, Berkeley, as a senior. There she was influenced by the mathematician Raphael Robinson. They became good friends, and married in 1941. As a spouse of a faculty member, Robinson was barred from teaching in the mathematics department at Berkeley. Although she continued to audit mathematics classes, she hoped to leave university and start a family. Not long after her wedding, however, Robinson contracted pneumonia. She was told that there was substantial scar tissue build up on her heart due to the rheumatic fever she suffered as a child. Due to the severity of the scar tissue, the doctor predicted that she would not live past forty and she was advised not to have children (Reid, 1986, 13).

Robinson was depressed for a long time, but eventually decided to continue studying mathematics. She returned to Berkeley and completed her PhD in 1948 under the supervision of Alfred Tarski. The first-order theory of the real numbers had been shown to be decidable by Tarski, and from Gödel's work it followed that the first-order theory of the natural numbers is undecidable. It was a major open problem whether the first-order theory of the rationals is decidable or not. In her thesis (1949), Robinson proved that it was not.

Interested in decision problems, Robinson next attempted to find a solution to Hilbert's tenth problem. This problem was one of a famous list of 23 mathematical problems posed by David Hilbert in 1900. The tenth problem asks whether there is an algorithm that will answer, in a finite amount of time, whether or not a polynomial equation with integer coefficients, such as $3x^2 - 2y + 3 = 0$, has a solution in the integers. Such questions are known as *Diophantine problems*. After some initial successes, Robinson joined forces with Martin Davis and Hilary Putnam, who were also working on the problem. They succeeded in showing that exponential Diophantine problems (where the unknowns may also appear as exponents) are undecidable, and showed that a certain conjecture (later called "J.R.") implies that Hilbert's tenth problem is undecidable (Davis et al., 1961). Robinson continued to work on the problem throughout the 1960s. In 1970, the young Russian mathematician Yuri Matijasevich finally proved the J.R. hypothesis. The combined result is now called the Matijasevich–Robinson–Davis–Putnam theorem, or MDRP theorem for short. Matijasevich and Robinson became friends and collaborated on several papers. In a letter to Matijasevich, Robinson once wrote that "actually I am very pleased that working together (thousands of miles apart) we are obviously making more progress than either one of us could alone" (Matijasevich, 1992, 45).

Robinson was the first female president of the American Mathematical Society, and the first woman to be elected to the National Academy of Science. She died on July 30, 1985 at the age of 65 after being diagnosed with leukemia.

Further Reading Robinson's mathematical papers are available in her *Collected Works* (Robinson, 1996), which also includes a reprint of her National Academy of Sciences biographical memoir (Feferman, 1994). Robinson's older sister Constance Reid published an "Autobiography of Julia," based on interviews (Reid, 1986), as well as a full memoir (Reid, 1996). A short documentary about Robinson and Hilbert's tenth problem was directed by George Csicsery (Csicsery, 2016). For a brief memoir about Yuri Matijasevich's collaborations with Robinson, and her influence on his work, see (Matijasevich, 1992).

C.8 Bertrand Russell

Bertrand Russell is hailed as one of the founders of modern analytic philosophy. Born May 18, 1872, Russell was not only known for his work in philosophy and logic, but wrote many popular books in various subject areas. He was also an ardent political activist throughout his life.

Russell was born in Trellech, Monmouthshire, Wales. His parents were members of the British nobility. They were free-thinkers, and even made friends with the radicals in Boston at the time. Unfortunately, Russell's parents died when he was young, and Russell was sent to live with his grandparents. There, he was given a religious upbringing (something his parents had wanted to avoid at all costs). His grandmother was very strict in all matters of morality. During adolescence he was mostly homeschooled by private tutors.

Russell's influence in analytic philosophy, and especially logic, is tremendous. He studied mathematics and philosophy at Trinity College, Cambridge, where he was influenced by the mathematician and philosopher Alfred North Whitehead. In 1910, Russell and Whitehead published the first volume of *Principia Mathematica*, where they championed the view that mathematics is reducible to logic. He went on to publish hundreds of books, essays and political pamphlets. In 1950, he won the Nobel Prize for literature.

Russell's was deeply entrenched in politics and social activism. During World War I he was arrested and sent to prison for six months due to pacifist activities and protest. While in prison, he was able to write and read, and claims to have found the experience "quite agreeable." He remained a pacifist throughout his life, and was again incarcerated for attending a nuclear disarmament rally in 1961. He also survived a plane crash in 1948, where the only survivors were those sitting in the smoking section. As such, Russell claimed that he owed his life to smoking. Russell was married four times, but had a reputation for carrying on extra-marital affairs. He died on February 2, 1970 at the age of 97 in Penrhyndeudraeth, Wales.

Further Reading Russell wrote an autobiography in three parts, spanning his life from 1872–1967 (Russell, 1967, 1968, 1969). The Bertrand Russell Research Centre at McMaster University is home of the Bertrand Russell archives. See their website at Duncan (2015), for information on the volumes of his collected works (including searchable indexes), and archival projects. Russell's paper *On Denoting* (Russell, 1905) is a classic of 20th century analytic philosophy.

The Stanford Encyclopedia of Philosophy entry on Russell (Irvine, 2015) has sound clips of Russell speaking on Desire and Political theory. Many video interviews with Russell are available online. To see him talk about smoking and being involved in a

plane crash, e.g., see Russell (n.d.). Some of Russell's works, including his *Introduction to Mathematical Philosophy* are available as free audiobooks on LibriVox (n.d.).

C.9 Alfred Tarski

Alfred Tarski was born on January 14, 1901 in Warsaw, Poland (then part of the Russian Empire). Described as "Napoleonic," Tarski was boisterous, talkative, and intense. His energy was often reflected in his lectures—he once set fire to a wastebasket while disposing of a cigarette during a lecture, and was forbidden from lecturing in that building again.

Tarski had a thirst for knowledge from a young age. Although later in life he would tell students that he studied logic because it was the only class in which he got a B, his high school records show that he got A's across the board—even in logic. He studied at the University of Warsaw from 1918 to 1924. Tarski first intended to study biology, but became interested in mathematics, philosophy, and logic, as the university was the center of the Warsaw School of Logic and Philosophy. Tarski earned his doctorate in 1924 under the supervision of Stanisław Leśniewski.

Before emigrating to the United States in 1939, Tarski completed some of his most important work while working as a secondary school teacher in Warsaw. His work on logical consequence and logical truth were written during this time. In 1939, Tarski was visiting the United States for a lecture tour. During his visit, Germany invaded Poland, and because of his Jewish heritage, Tarski could not return. His wife and children remained in Poland until the end of the war, but were then able to emigrate to the United States as well. Tarski taught at Harvard, the College of the City of New York, and the Institute for Advanced Study at Princeton, and finally the University of California, Berkeley. There he founded the multidisciplinary program in Logic and the Methodology of Science. Tarski died on October 26, 1983 at the age of 82.

Further Reading For more on Tarski's life, see the biography *Alfred Tarski: Life and Logic* (Feferman and Feferman, 2004). Tarski's seminal works on logical consequence and truth are available in English in (Corcoran, 1983). All of Tarski's original works have been collected into a four volume series, (Tarski, 1981).

C.10 Alan Turing

Alan Turing was born in Mailda Vale, London, on June 23, 1912. He is considered the father of theoretical computer science. Turing's interest in the physical sciences and mathematics started at a young age. However, as a boy his interests were not represented well in his schools, where emphasis was placed on literature and classics. Consequently, he did poorly in school and was reprimanded by many of his teachers.

Turing attended King's College, Cambridge as an undergraduate, where he studied mathematics. In 1936 Turing developed (what is now called) the Turing machine as an attempt to precisely define the notion of a computable function and to prove the undecidability of the decision problem. He was beaten to the result by Alonzo Church, who proved the result via his own lambda calculus. Turing's paper was still published with reference to Church's result. Church invited Turing to Princeton, where he spent 1936–1938, and obtained a doctorate under Church.

Despite his interest in logic, Turing's earlier interests in physical sciences remained prevalent. His practical skills were put to work during his service with the British

cryptanalytic department at Bletchley Park during World War II. Turing was a central figure in cracking the cypher used by German Naval communications—the Enigma code. Turing's expertise in statistics and cryptography, together with the introduction of electronic machinery, gave the team the ability to crack the code by creating a de-crypting machine called a "bombe." His ideas also helped in the creation of the world's first programmable electronic computer, the Colossus, also used at Bletchley park to break the German Lorenz cypher.

Turing was gay. Nevertheless, in 1942 he proposed to Joan Clarke, one of his teammates at Bletchley Park, but later broke off the engagement and confessed to her that he was homosexual. He had several lovers throughout his lifetime, although homosexual acts were then criminal offences in the UK. In 1952, Turing's house was burgled by a friend of his lover at the time, and when filing a police report, Turing admitted to having a homosexual relationship, under the impression that the government was on their way to legalizing homosexual acts. This was not true, and he was charged with gross indecency. Instead of going to prison, Turing opted for a hormone treatment that reduced libido. Turing was found dead on June 8, 1954, of a cyanide overdose—most likely suicide. He was given a royal pardon by Queen Elizabeth II in 2013.

Further Reading For a comprehensive biography of Alan Turing, see Hodges (2014). Turing's life and work inspired a play, *Breaking the Code*, which was produced in 1996 for TV starring Derek Jacobi as Turing. *The Imitation Game*, an Academy Award nominated film starring Bendict Cumberbatch and Kiera Knightley, is also loosely based on Alan Turing's life and time at Bletchley Park (Tyldum, 2014).

Radiolab (2012) has several podcasts on Turing's life and work. BBC Horizon's documentary *The Strange Life and Death of Dr. Turing* is available to watch online (Sykes, 1992). (Theelen, 2012) is a short video of a working LEGO Turing Machine—made to honour Turing's centenary in 2012.

Turing's original paper on Turing machines and the decision problem is Turing (1937).

C.11 Ernst Zermelo

Ernst Zermelo was born on July 27, 1871 in Berlin, Germany. He had five sisters, though his family suffered from poor health and only three survived to adulthood. His parents also passed away when he was young, leaving him and his siblings orphans when he was seventeen. Zermelo had a deep interest in the arts, and especially in poetry. He was known for being sharp, witty, and critical. His most celebrated mathematical achievements include the introduction of the axiom of choice (in 1904), and his axiomatization of set theory (in 1908).

Zermelo's interests at university were varied. He took courses in physics, mathematics, and philosophy. Under the supervision of Hermann Schwarz, Zermelo completed his dissertation *Investigations in the Calculus of Variations* in 1894 at the University of Berlin. In 1897, he decided to pursue more studies at the University of Göttigen, where he was heavily influenced by the foundational work of David Hilbert. In 1899 he became eligible for professorship, but did not get one until eleven years later—possibly due to his strange demeanour and "nervous haste."

Zermelo finally received a paid professorship at the University of Zurich in 1910, but was forced to retire in 1916 due to tuberculosis. After his recovery, he was given

an honourary professorship at the University of Freiburg in 1921. During this time he worked on foundational mathematics. He became irritated with the works of Thoralf Skolem and Kurt Gödel, and publicly criticized their approaches in his papers. He was dismissed from his position at Freiburg in 1935, due to his unpopularity and his opposition to Hitler's rise to power in Germany.

The later years of Zermelo's life were marked by isolation. After his dismissal in 1935, he abandoned mathematics. He moved to the country where he lived modestly. He married in 1944, and became completely dependent on his wife as he was going blind. Zermelo lost his sight completely by 1951. He passed away in Günterstal, Germany, on May 21, 1953.

Further Reading For a full biography of Zermelo, see Ebbinghaus (2015). Zermelo's seminal 1904 and 1908 papers are available to read in the original German (Zermelo, 1904, 1908). Zermelo's collected works, including his writing on physics, are available in English translation in (Ebbinghaus et al., 2010; Ebbinghaus and Kanamori, 2013).

Photo Credits

Bibliography

Andrásfai, Béla. 1986. Rózsa (Rosa) Péter. *Periodica Polytechnica Electrical Engineering* 30(2-3): 139–145. URL http://www.pp.bme.hu/ee/article/view/4651.

Aspray, William. 1984. The Princeton mathematics community in the 1930s: Alonzo Church. URL http://www.princeton.edu/mudd/findingaids/mathoral/pmc05.htm. Interview.

Baaz, Matthias, Christos H. Papadimitriou, Hilary W. Putnam, Dana S. Scott, and Charles L. Harper Jr. 2011. *Kurt Gödel and the Foundations of Mathematics: Horizons of Truth*. Cambridge: Cambridge University Press.

Cheng, Eugenia. 2004. How to write proofs: A quick quide. URL http://cheng.staff.shef.ac.uk/proofguide/proofguide.pdf.

Church, Alonzo. 1936a. A note on the Entscheidungsproblem. *Journal of Symbolic Logic* 1: 40–41.

Church, Alonzo. 1936b. An unsolvable problem of elementary number theory. *American Journal of Mathematics* 58: 345–363.

Corcoran, John. 1983. *Logic, Semantics, Metamathematics*. Indianapolis: Hackett, 2nd ed.

Csicsery, George. 2016. Zala films: Julia Robinson and Hilbert's tenth problem. URL http://www.zalafilms.com/films/juliarobinson.html.

Dauben, Joseph. 1990. *Georg Cantor: His Mathematics and Philosophy of the Infinite*. Princeton: Princeton University Press.

Davis, Martin, Hilary Putnam, and Julia Robinson. 1961. The decision problem for exponential Diophantine equations. *Annals of Mathematics* 74(3): 425–436. URL http://www.jstor.org/stable/1970289.

Dick, Auguste. 1981. *Emmy Noether 1882–1935*. Boston: Birkhäuser.

du Sautoy, Marcus. 2014. A brief history of mathematics: Georg Cantor. URL http://www.bbc.co.uk/programmes/b00ss1j0. Audio Recording.

Duncan, Arlene. 2015. The Bertrand Russell Research Centre. URL http://russell.mcmaster.ca/.

Ebbinghaus, Heinz-Dieter. 2015. *Ernst Zermelo: An Approach to his Life and Work*. Berlin: Springer-Verlag.

Ebbinghaus, Heinz-Dieter, Craig G. Fraser, and Akihiro Kanamori. 2010. *Ernst Zermelo. Collected Works*, vol. 1. Berlin: Springer-Verlag.

Ebbinghaus, Heinz-Dieter and Akihiro Kanamori. 2013. *Ernst Zermelo: Collected Works*, vol. 2. Berlin: Springer-Verlag.

Enderton, Herbert B. 2019. Alonzo Church: Life and Work. In *The Collected Works of Alonzo Church*, eds. Tyler Burge and Herbert B. Enderton. Cambridge, MA: MIT Press.

Feferman, Anita and Solomon Feferman. 2004. *Alfred Tarski: Life and Logic*. Cambridge: Cambridge University Press.

Feferman, Solomon. 1994. Julia Bowman Robinson 1919–1985. *Biographical Memoirs of the National Academy of Sciences* 63: 1–28. URL `http://www.nasonline.org/publications/biographical-memoirs/memoir-pdfs/robinson-julia.pdf`.

Feferman, Solomon, John W. Dawson Jr., Stephen C. Kleene, Gregory H. Moore, Robert M. Solovay, and Jean van Heijenoort. 1986. *Kurt Gödel: Collected Works. Vol. 1: Publications 1929–1936*. Oxford: Oxford University Press.

Feferman, Solomon, John W. Dawson Jr., Stephen C. Kleene, Gregory H. Moore, Robert M. Solovay, and Jean van Heijenoort. 1990. *Kurt Gödel: Collected Works. Vol. 2: Publications 1938–1974*. Oxford: Oxford University Press.

Frey, Holly and Tracy V. Wilson. 2015. Stuff you missed in history class: Emmy Noether, mathematics trailblazer. URL `http://www.missedinhistory.com/podcasts/emmy-noether-mathematics-trailblazer/`. Podcast audio.

Gentzen, Gerhard. 1935a. Untersuchungen über das logische Schließen I. *Mathematische Zeitschrift* 39: 176–210. English translation in Szabo (1969), pp. 68–131.

Gentzen, Gerhard. 1935b. Untersuchungen über das logische Schließen II. *Mathematische Zeitschrift* 39: 176–210, 405–431. English translation in Szabo (1969), pp. 68–131.

Gödel, Kurt. 1929. Über die Vollständigkeit des Logikkalküls [On the completeness of the calculus of logic]. Dissertation, Universität Wien. Reprinted and translated in Feferman et al. (1986), pp. 60–101.

Gödel, Kurt. 1931. über formal unentscheidbare Sätze der *Principia Mathematica* und verwandter Systeme I [On formally undecidable propositions of *Principia Mathematica* and related systems I]. *Monatshefte für Mathematik und Physik* 38: 173–198. Reprinted and translated in Feferman et al. (1986), pp. 144–195.

Grattan-Guinness, Ivor. 1971. Towards a biography of Georg Cantor. *Annals of Science* 27(4): 345–391.

Hammack, Richard. 2013. *Book of Proof*. Richmond, VA: Virginia Commonwealth University. URL `http://www.people.vcu.edu/rhammack/BookOfProof/BookOfProof.pdf`.

Hodges, Andrew. 2014. *Alan Turing: The Enigma*. London: Vintage.

Hutchings, Michael. 2003. Introduction to mathematical arguments. URL https://math.berkeley.edu/hutching/teach/proofs.pdf.

Institute, Perimeter. 2015. Emmy Noether: Her life, work, and influence. URL https://www.youtube.com/watch?v=tNNyAyMRsgE. Video Lecture.

Irvine, Andrew David. 2015. Sound clips of Bertrand Russell speaking. URL http://plato.stanford.edu/entries/russell/russell-soundclips.html.

Jacobson, Nathan. 1983. *Emmy Noether: Gesammelte Abhandlungen—Collected Papers*. Berlin: Springer-Verlag.

John Dawson, Jr. 1997. *Logical Dilemmas: The Life and Work of Kurt Gödel*. Boca Raton: CRC Press.

LibriVox. n.d. Bertrand Russell. URL https://librivox.org/author/1508?primarykey=1508&searchcategory=author&searchpage=1&searchform=getresults. Collection of public domain audiobooks.

Linsenmayer, Mark. 2014. The partially examined life: Gödel on math. URL http://www.partiallyexaminedlife.com/2014/06/16/ep95-godel/. Podcast audio.

MacFarlane, John. 2015. Alonzo Church's JSL reviews. URL http://johnmacfarlane.net/church.html.

Matijasevich, Yuri. 1992. My collaboration with Julia Robinson. *The Mathematical Intelligencer* 14(4): 38–45.

Menzler-Trott, Eckart. 2007. *Logic's Lost Genius: The Life of Gerhard Gentzen*. Providence: American Mathematical Society.

O'Connor, John J. and Edmund F. Robertson. 2014. Rózsa Péter. URL http://www-groups.dcs.st-and.ac.uk/history/Biographies/Peter.html.

Péter, Rózsa. 1935a. Über den Zusammenhang der verschiedenen Begriffe der rekursiven Funktion. *Mathematische Annalen* 110: 612–632.

Péter, Rózsa. 1935b. Konstruktion nichtrekursiver Funktionen. *Mathematische Annalen* 111: 42–60.

Péter, Rózsa. 1951. *Rekursive Funktionen*. Budapest: Akademiai Kiado. English translation in (Péter, 1967).

Péter, Rózsa. 1967. *Recursive Functions*. New York: Academic Press.

Péter, Rózsa. 2010. *Playing with Infinity*. New York: Dover. URL https://books.google.ca/books?id=6V3wNs4uv4C&lpg=PP1&ots=BkQZaHcR99&lr&pg=PP1#v=onepage&q&f=false.

Radiolab. 2012. The Turing problem. URL http://www.radiolab.org/story/193037-turing-problem/. Podcast audio.

Reid, Constance. 1986. The autobiography of Julia Robinson. *The College Mathematics Journal* 17: 3–21.

Reid, Constance. 1996. *Julia: A Life in Mathematics*. Cambridge: Cambridge University Press. URL https://books.google.ca/books?id=lRtSzQyHf9UC&lpg=PP1&pg=PP1#v=onepage&q&f=false.

Robinson, Julia. 1949. Definability and decision problems in arithmetic. *Journal of Symbolic Logic* 14(2): 98–114. URL http://www.jstor.org/stable/2266510.

Robinson, Julia. 1996. *The Collected Works of Julia Robinson*. Providence: American Mathematical Society.

Rose, Daniel. 2012. A song about Georg Cantor. URL https://www.youtube.com/watch?v=QUP5Z4Fb5k4. Audio Recording.

Russell, Bertrand. 1905. On denoting. *Mind* 14: 479–493.

Russell, Bertrand. 1967. *The Autobiography of Bertrand Russell*, vol. 1. London: Allen and Unwin.

Russell, Bertrand. 1968. *The Autobiography of Bertrand Russell*, vol. 2. London: Allen and Unwin.

Russell, Bertrand. 1969. *The Autobiography of Bertrand Russell*, vol. 3. London: Allen and Unwin.

Russell, Bertrand. n.d. Bertrand Russell on smoking. URL https://www.youtube.com/watch?v=80oLTiVWlc. Video Interview.

Sandstrum, Ted. 2019. *Mathematical Reasoning: Writing and Proof*. Allendale, MI: Grand Valley State University. URL https://scholarworks.gvsu.edu/books/7/.

Segal, Sanford L. 2014. *Mathematicians under the Nazis*. Princeton: Princeton University Press.

Sigmund, Karl, John Dawson, Kurt Mühlberger, Hans Magnus Enzensberger, and Juliette Kennedy. 2007. Kurt Gödel: Das Album–The Album. *The Mathematical Intelligencer* 29(3): 73–76.

Smith, Peter. 2013. *An Introduction to Gödel's Theorems*. Cambridge: Cambridge University Press.

Solow, Daniel. 2013. *How to Read and Do Proofs*. Hoboken, NJ: Wiley.

Steinhart, Eric. 2018. *More Precisely: The Math You Need to Do Philosophy*. Peterborough, ON: Broadview, 2nd ed.

Sykes, Christopher. 1992. BBC Horizon: The strange life and death of Dr. Turing. URL https://www.youtube.com/watch?v=gyusnGbBSHE.

Szabo, Manfred E. 1969. *The Collected Papers of Gerhard Gentzen*. Amsterdam: North-Holland.

Takeuti, Gaisi, Nicholas Passell, and Mariko Yasugi. 2003. *Memoirs of a Proof Theorist: Gödel and Other Logicians*. Singapore: World Scientific.

Tamassy, Istvan. 1994. Interview with Róza Péter. *Modern Logic* 4(3): 277–280.

Tarski, Alfred. 1981. *The Collected Works of Alfred Tarski*, vol. I–IV. Basel: Birkhäuser.

Theelen, Andre. 2012. Lego turing machine. URL `https://www.youtube.com/watch?v=FTSAiF9AHN4`.

Turing, Alan M. 1937. On computable numbers, with an application to the "Entscheidungsproblem". *Proceedings of the London Mathematical Society, 2nd Series* 42: 230–265.

Tyldum, Morten. 2014. The imitation game. Motion picture.

Velleman, Daniel J. 2019. *How to Prove It: A Structured Approach*. Cambridge: Cambridge University Press, 3rd ed.

Wang, Hao. 1990. *Reflections on Kurt Gödel*. Cambridge: MIT Press.

Zermelo, Ernst. 1904. Beweis, daß jede Menge wohlgeordnet werden kann. *Mathematische Annalen* 59: 514–516. English translation in (Ebbinghaus et al., 2010, pp. 115–119).

Zermelo, Ernst. 1908. Untersuchungen über die Grundlagen der Mengenlehre I. *Mathematische Annalen* 65(2): 261–281. English translation in (Ebbinghaus et al., 2010, pp. 189-229).