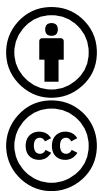


THE OPEN LOGIC TEXT

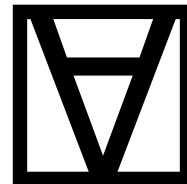
Complete Build

Open Logic Project

Revision: 6612311 (master)
2017-07-17



The Open Logic Text by
the Open Logic Project
is licensed under a Cre-
ative Commons Attribu-
tion 4.0 International Li-
cense.



Chapter 1

About the Open Logic Project

The *Open Logic Text* is an open-source, collaborative textbook of formal meta-logic and formal methods, starting at an intermediate level (i.e., after an introductory formal logic course). Though aimed at a non-mathematical audience (in particular, students of philosophy and computer science), it is rigorous.

The *Open Logic Text* is a collaborative project and is under active development. Coverage of some topics currently included may not yet be complete, and many sections still require substantial revision. We plan to expand the text to cover more topics in the future. We also plan to add features to the text, such as a glossary, a list of further reading, historical notes, pictures, better explanations, sections explaining the relevance of results to philosophy, computer science, and mathematics, and more problems and examples. If you find an error, or have a suggestion, please let the project team know.

The project operates in the spirit of open source. Not only is the text freely available, we provide the LaTeX source under the Creative Commons Attribution license, which gives anyone the right to download, use, modify, rearrange, convert, and re-distribute our work, as long as they give appropriate credit.

Please see the Open Logic Project website at openlogicproject.org for additional information.

Contents

1	About the Open Logic Project	1
I	Sets, Relations, Functions	12
2	Sets	14
2.1	Basics	14
2.2	Some Important Sets	15
2.3	Subsets	16
2.4	Unions and Intersections	17
2.5	Pairs, Tuples, Cartesian Products	19
2.6	Russell's Paradox	21
	Problems	21
3	Relations	23
3.1	Relations as Sets	23
3.2	Special Properties of Relations	24
3.3	Orders	25
3.4	Graphs	27
3.5	Operations on Relations	28
	Problems	29
4	Functions	30
4.1	Basics	30
4.2	Kinds of Functions	32
4.3	Inverses of Functions	33
4.4	Composition of Functions	34
4.5	Isomorphism	35
4.6	Partial Functions	35
4.7	Functions and Relations	36
	Problems	36
5	The Size of Sets	37

CONTENTS

5.1	Introduction	37
5.2	Enumerable Sets	37
5.3	Non-enumerable Sets	41
5.4	Reduction	44
5.5	Equinumerous Sets	45
5.6	Comparing Sizes of Sets	46
	Problems	47
 II First-order Logic		50
6	Syntax and Semantics	52
6.1	Introduction	52
6.2	First-Order Languages	53
6.3	Terms and Formulas	55
6.4	Unique Readability	56
6.5	Main operator of a Formula	59
6.6	Subformulas	60
6.7	Free Variables and Sentences	61
6.8	Substitution	62
6.9	Structures for First-order Languages	64
6.10	Covered Structures for First-order Languages	65
6.11	Satisfaction of a Formula in a Structure	66
6.12	Variable Assignments	70
6.13	Extensionality	73
6.14	Semantic Notions	74
	Problems	76
7	Theories and Their Models	78
7.1	Introduction	78
7.2	Expressing Properties of Structures	80
7.3	Examples of First-Order Theories	80
7.4	Expressing Relations in a Structure	83
7.5	The Theory of Sets	84
7.6	Expressing the Size of Structures	86
	Problems	87
8	The Sequent Calculus	89
8.1	Rules and Derivations	89
8.2	Examples of Derivations	92
8.3	Proof-Theoretic Notions	96
8.4	Properties of Derivability	97
8.5	Soundness	100
8.6	Derivations with Identity predicate	104

Problems	105
9 Natural Deduction	106
9.1 Introduction	106
9.2 Rules and Derivations	108
9.3 Examples of Derivations	110
9.4 Proof-Theoretic Notions	117
9.5 Properties of Derivability	118
9.6 Soundness	122
9.7 Derivations with Identity predicate	125
9.8 Soundness of Identity predicate Rules	127
Problems	127
10 The Completeness Theorem	129
10.1 Introduction	129
10.2 Outline of the Proof	129
10.3 Maximally Consistent Sets of Sentences	131
10.4 Henkin Expansion	133
10.5 Lindenbaum's Lemma	134
10.6 Construction of a Model	135
10.7 Identity	137
10.8 The Completeness Theorem	139
10.9 The Compactness Theorem	140
10.10 The Löwenheim-Skolem Theorem	142
Problems	142
11 Beyond First-order Logic	144
11.1 Overview	144
11.2 Many-Sorted Logic	145
11.3 Second-Order logic	146
11.4 Higher-Order logic	150
11.5 Intuitionistic Logic	152
11.6 Modal Logics	156
11.7 Other Logics	158
III Model Theory	159
12 Basics of Model Theory	161
12.1 Reducts and Expansions	161
12.2 Substructures	162
12.3 Overspill	162
12.4 Isomorphic Structures	163
12.5 The Theory of a Structure	164

CONTENTS

12.6	Partial Isomorphisms	165
12.7	Dense Linear Orders	168
	Problems	169
13	Models of Arithmetic	170
13.1	Introduction	170
13.2	Standard Models of Arithmetic	171
13.3	Non-Standard Models	173
13.4	Models of \mathbf{Q}	174
13.5	Computable Models of Arithmetic	176
	Problems	178
14	The Interpolation Theorem	180
14.1	Introduction	180
14.2	Separation of Sentences	180
14.3	Craig's Interpolation Theorem	182
14.4	The Definability Theorem	184
15	Lindström's Theorem	187
15.1	Introduction	187
15.2	Abstract Logics	187
15.3	Compactness and Löwenheim-Skolem Properties	189
15.4	Lindström's Theorem	190
IV	Computability	193
16	Recursive Functions	195
16.1	Introduction	195
16.2	Primitive Recursion	196
16.3	Primitive Recursive Functions are Computable	199
16.4	Examples of Primitive Recursive Functions	200
16.5	Primitive Recursive Relations	202
16.6	Bounded Minimization	203
16.7	Primes	204
16.8	Sequences	205
16.9	Other Recursions	207
16.10	Non-Primitive Recursive Functions	209
16.11	Partial Recursive Functions	210
16.12	The Normal Form Theorem	212
16.13	The Halting Problem	213
16.14	General Recursive Functions	214
	Problems	214
17	The Lambda Calculus	216

17.1	Introduction	216
17.2	The Syntax of the Lambda Calculus	217
17.3	Reduction of Lambda Terms	218
17.4	The Church-Rosser Property	219
17.5	Representability by Lambda Terms	220
17.6	Lambda Representable Functions are Computable	220
17.7	Computable Functions are Lambda Representable	221
17.8	The Basic Primitive Recursive Functions are Lambda Representable	221
17.9	Lambda Representable Functions Closed under Composition	222
17.10	Lambda Representable Functions Closed under Primitive Recursion	222
17.11	Fixed-Point Combinators	224
17.12	Lambda Representable Functions Closed under Minimization	225
18	Computability Theory	227
18.1	Introduction	227
18.2	Coding Computations	228
18.3	The Normal Form Theorem	229
18.4	The s - m - n Theorem	230
18.5	The Universal Partial Computable Function	230
18.6	No Universal Computable Function	231
18.7	The Halting Problem	231
18.8	Comparison with Russell's Paradox	232
18.9	Computable Sets	234
18.10	Computably Enumerable Sets	234
18.11	Definitions of C. E. Sets	235
18.12	Union and Intersection of C.E. Sets	237
18.13	Computably Enumerable Sets not Closed under Complement	238
18.14	Reducibility	239
18.15	Properties of Reducibility	240
18.16	Complete Computably Enumerable Sets	241
18.17	An Example of Reducibility	242
18.18	Totality is Undecidable	243
18.19	Rice's Theorem	244
18.20	The Fixed-Point Theorem	246
18.21	Applying the Fixed-Point Theorem	249
18.22	Defining Functions using Self-Reference	250
18.23	Minimization with Lambda Terms	251
	Problems	252

CONTENTS

V	Turing Machines	253
19	Turing Machine Computations	255
19.1	Introduction	255
19.2	Representing Turing Machines	257
19.3	Turing Machines	260
19.4	Configurations and Computations	261
19.5	Unary Representation of Numbers	263
19.6	Halting States	263
19.7	Combining Turing Machines	264
19.8	Variants of Turing Machines	266
19.9	The Church-Turing Thesis	267
	Problems	268
20	Undecidability	270
20.1	Introduction	270
20.2	Enumerating Turing Machines	272
20.3	The Halting Problem	272
20.4	The Decision Problem	274
20.5	Representing Turing Machines	275
20.6	Verifying the Representation	278
20.7	The Decision Problem is Unsolvable	283
	Problems	283
VI	Incompleteness	285
21	Introduction to Incompleteness	287
21.1	Historical Background	287
21.2	Definitions	291
21.3	Overview of Incompleteness Results	295
21.4	Undecidability and Incompleteness	297
	Problems	298
22	Arithmetization of Syntax	299
22.1	Introduction	299
22.2	Coding Symbols	300
22.3	Coding Terms	302
22.4	Coding Formulas	304
22.5	Substitution	305
22.6	Derivations in LK	306
22.7	Derivations in Natural Deduction	309
	Problems	314
23	Representability in Q	315

23.1	Introduction	315
23.2	Functions Representable in \mathbf{Q} are Computable	317
23.3	The Beta Function Lemma	318
23.4	Simulating Primitive Recursion	321
23.5	Basic Functions are Representable in \mathbf{Q}	322
23.6	Composition is Representable in \mathbf{Q}	324
23.7	Regular Minimization is Representable in \mathbf{Q}	326
23.8	Computable Functions are Representable in \mathbf{Q}	329
23.9	Representing Relations	329
23.10	Undecidability	330
	Problems	331
24	Theories and Computability	332
24.1	Introduction	332
24.2	\mathbf{Q} is C.e.-Complete	332
24.3	ω -Consistent Extensions of \mathbf{Q} are Undecidable	333
24.4	Consistent Extensions of \mathbf{Q} are Undecidable	334
24.5	Axiomatizable Theories	335
24.6	Axiomatizable Complete Theories are Decidable	335
24.7	\mathbf{Q} has no Complete, Consistent, Axomatizable Extensions	335
24.8	Sentences Provable and Refutable in \mathbf{Q} are Computably Inseparable	336
24.9	Theories Consistent with \mathbf{Q} are Undecidable	337
24.10	Theories in which \mathbf{Q} is Interpretable are Undecidable	337
25	Incompleteness and Provability	339
25.1	Introduction	339
25.2	The Fixed-Point Lemma	340
25.3	The First Incompleteness Theorem	342
25.4	Rosser's Theorem	343
25.5	Comparison with Gödel's Original Paper	345
25.6	The Provability Conditions for \mathbf{PA}	345
25.7	The Second Incompleteness Theorem	346
25.8	Löb's Theorem	349
25.9	The Undefinability of Truth	352
	Problems	353
VII	Second-order Logic	354
26	Syntax and Semantics	356
26.1	Introduction	356
26.2	Terms and Formulas	357
26.3	Satisfaction	358

CONTENTS

26.4	Semantic Notions	359
26.5	Expressive Power	359
26.6	Describing Infinite and Enumerable Domains	360
	Problems	362
27	Metatheory of Second-order Logic	363
27.1	Introduction	363
27.2	Second-order Arithmetic	364
27.3	Second-order Logic is not Axiomatizable	365
27.4	Second-order Logic is not Compact	366
27.5	The Löwenheim-Skolem Theorem Fails for Second-order Logic	366
	Problems	367
28	Second-order Logic and Set Theory	368
28.1	Introduction	368
28.2	Comparing Sets	368
28.3	Cardinalities of Sets	369
28.4	The Power of the Continuum	370
VIII	Methods	373
29	Proofs	375
29.1	Introduction	375
29.2	Starting a Proof	376
29.3	Using Definitions	376
29.4	Inference Patterns	378
29.5	An Example	383
29.6	Another Example	386
29.7	Indirect Proof	387
29.8	Reading Proofs	391
29.9	I can't do it!	392
29.10	Other Resources	393
	Problems	394
30	Induction	395
30.1	Introduction	395
30.2	Induction on \mathbb{N}	396
30.3	Strong Induction	398
30.4	Inductive Definitions	399
30.5	Structural Induction	400

CONTENTS

IX History	402
31 Biographies	403
31.1 Georg Cantor	403
31.2 Alonzo Church	404
31.3 Gerhard Gentzen	405
31.4 Kurt Gödel	406
31.5 Emmy Noether	407
31.6 Rózsa Péter	409
31.7 Julia Robinson	410
31.8 Bertrand Russell	412
31.9 Alfred Tarski	413
31.10 Alan Turing	414
31.11 Ernst Zermelo	415
Photo Credits	417
Bibliography	419

CONTENTS

This file loads all content included in the Open Logic Project. Editorial notes like this, if displayed, indicate that the file was compiled without any thought to how this material will be presented. It is thus *not advisable* to teach or study from a PDF that includes this comment.

The Open Logic Project provides many mechanisms by which a text can be generate which is more appropriate for teaching or self-study. For instance, by default, the text will make all logical operators primitives and carry out all cases for all operators in proofs. But it is much better to leave some of these cases as exercises. The Open Logic Project is also a work in progress. In an effort to stimulate collaboration and improvement, material is included even if it is only in draft form, is missing exercises, etc. A PDF produced for a course will exclude these sections.

To find PDFs more suitable for reading, have a look at the sample courses available on the OLP website.

Part I

Sets, Relations, Functions

CONTENTS

The material in this part is a reasonably complete introduction to basic naive set theory. Unless students can be assumed to have this background, it's probably advisable to start a course with a review of this material, at least the part on sets, functions, and relations. This should ensure that all students have the basic facility with mathematical notation required for any of the other logical sections. NB: This part does not cover induction directly.

The presentation here would benefit from additional examples, especially, "real life" examples of relations of interest to the audience.

It is planned to expand this part to cover naive set theory more extensively.

Chapter 2

Sets

2.1 Basics

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

Definition 2.1 (Set). A *set* is a collection of objects, considered independently of the way it is specified, of the order of the objects in the set, or of their multiplicity. The objects making up the set are called *elements* or *members* of the set. If a is an element of a set X , we write $a \in X$ (otherwise, $a \notin X$). The set which has no elements is called the *empty* set and denoted by the symbol \emptyset .

Example 2.2. Whenever you have a bunch of objects, you can collect them together in a set. The set of Richard's siblings, for instance, is a set that contains one person, and we could write it as $S = \{\text{Ruth}\}$. In general, when we have some objects a_1, \dots, a_n , then the set consisting of exactly those objects is written $\{a_1, \dots, a_n\}$. Frequently we'll specify a set by some property that its elements share—as we just did, for instance, by specifying S as the set of Richard's siblings. We'll use the following shorthand notation for that: $\{x : \dots x \dots\}$, where the $\dots x \dots$ stands for the property that x has to have in order to be counted among the elements of the set. In our example, we could have specified S also as

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

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

2.2. SOME IMPORTANT SETS

that

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

are three ways of specifying one and the same set.

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

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

are two ways of specifying the same set; and that

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

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

Definition 2.3 (Extensionality). If X and Y are sets, then X and Y are *identical*, $X = Y$, iff every element of X is also an element of Y , and vice versa.

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

2.2 Some Important Sets

Example 2.4. Mostly we'll be dealing with sets that have mathematical objects as members. You will remember the various sets of numbers: \mathbb{N} is the set of *natural* numbers $\{0, 1, 2, 3, \dots\}$; \mathbb{Z} the set of *integers*,

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

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

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

Example 2.5 (Strings). Another interesting example is the set A^* of *finite strings* over an alphabet A : any finite sequence of elements of A is a string over A . We include the *empty string* Λ among the strings over A , for every alphabet A . For instance,

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

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

Example 2.6 (Infinite sequences). For any set A we may also consider the set A^ω of infinite sequences of elements of A . An infinite sequence $a_1 a_2 a_3 a_4 \dots$ consists of a one-way infinite list of objects, each one of which is an element of A .

2.3 Subsets

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

Definition 2.7 (Subset). If every element of a set X is also an element of Y , then we say that X is a *subset* of Y , and write $X \subseteq Y$.

Example 2.8. First of all, every set is a subset of itself, and \emptyset is a subset of every set. The set of even numbers is a subset of the set of natural numbers. Also, $\{a, b\} \subseteq \{a, b, c\}$.

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

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

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

Definition 2.9 (Power Set). The set consisting of all subsets of a set X is called the *power set* of X , written $\wp(X)$.

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

2.4. UNIONS AND INTERSECTIONS

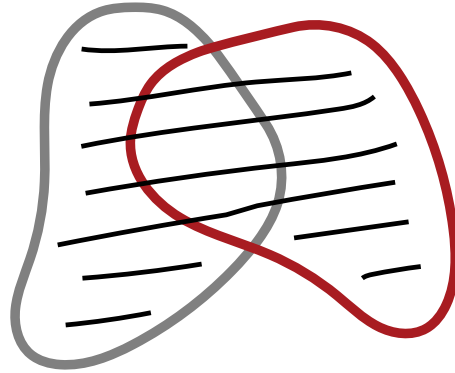


Figure 2.1: The union $X \cup Y$ of two sets is set of elements of X together with those of Y .

Example 2.10. What are all the possible subsets of $\{a, b, c\}$? They are: \emptyset , $\{a\}$, $\{b\}$, $\{c\}$, $\{a, b\}$, $\{a, c\}$, $\{b, c\}$, $\{a, b, c\}$. The set of all these subsets is $\wp(\{a, b, c\})$:

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

2.4 Unions and Intersections

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

Definition 2.11 (Union). The *union* of two sets X and Y , written $X \cup Y$, is the set of all things which are elements of X , Y , or both.

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

Example 2.12. Since the multiplicity of elements doesn't matter, the union of two sets which have an element in common contains that element only once, e.g., $\{a, b, c\} \cup \{a, 0, 1\} = \{a, b, c, 0, 1\}$.

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

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

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

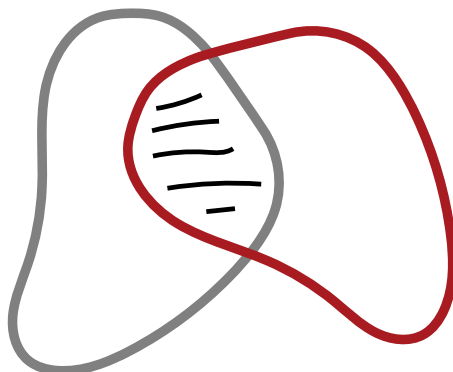


Figure 2.2: The intersection $X \cap Y$ of two sets is the set of elements they have in common.

Definition 2.13 (Intersection). The *intersection* of two sets X and Y , written $X \cap Y$, is the set of all things which are elements of both X and Y .

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

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

Example 2.14. If two sets have no elements in common, their intersection is empty: $\{a, b, c\} \cap \{0, 1\} = \emptyset$.

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

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

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

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

Definition 2.15. If Z is a set of sets, then $\bigcup Z$ is the set of elements of elements of Z :

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

2.5. PAIRS, TUPLES, CARTESIAN PRODUCTS

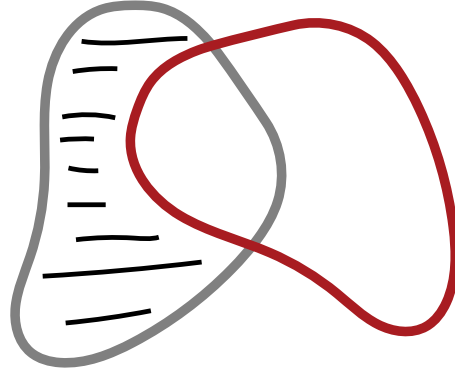


Figure 2.3: The difference $X \setminus Y$ of two sets is the set of those elements of X which are not also elements of Y .

Definition 2.16. If Z is a set of sets, then $\bigcap Z$ is the set of objects which all elements of Z have in common:

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

Example 2.17. Suppose $Z = \{\{a, b\}, \{a, d, e\}, \{a, d\}\}$. Then $\bigcup Z = \{a, b, d, e\}$ and $\bigcap Z = \{a\}$.

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

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

Definition 2.18 (Difference). The *difference* $X \setminus Y$ is the set of all elements of X which are not also elements of Y , i.e.,

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

2.5 Pairs, Tuples, Cartesian Products

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

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

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

Definition 2.19 (Cartesian product). Given sets X and Y , their *Cartesian product* $X \times Y$ is $\{\langle x, y \rangle : x \in X \text{ and } y \in Y\}$.

Example 2.20. If $X = \{0, 1\}$, and $Y = \{1, a, b\}$, then their product is

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

Example 2.21. If X is a set, the product of X with itself, $X \times X$, is also written X^2 . It is the set of *all* pairs $\langle x, y \rangle$ with $x, y \in X$. The set of all triples $\langle x, y, z \rangle$ is X^3 , and so on. We can give an inductive definition:

$$\begin{aligned} X^1 &= X \\ X^{k+1} &= X^k \times X \end{aligned}$$

Proposition 2.22. If X has n elements and Y has m elements, then $X \times Y$ has $n \cdot m$ elements.

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

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

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

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

Example 2.23. If X is a set, a *word* over X is any sequence of elements of X . A sequence can be thought of as an n -tuple of elements of X . For instance, if $X = \{a, b, c\}$, then the sequence “bac” can be thought of as the triple $\langle b, a, c \rangle$. Words, i.e., sequences of symbols, are of crucial importance in computer science, of course. By convention, we count elements of X as sequences of length 1, and \emptyset as the sequence of length 0. The set of *all* words over X then is

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

2.6. RUSSELL'S PARADOX

2.6 Russell's Paradox

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

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

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

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

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

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

exist?

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

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

Problems

Problem 2.1. Show that there is only one empty set, i.e., show that if X and Y are sets without members, then $X = Y$.

Problem 2.2. List all subsets of $\{a, b, c, d\}$.

Problem 2.3. Show that if X has n elements, then $\wp(X)$ has 2^n elements.

Problem 2.4. Prove rigorously that if $X \subseteq Y$, then $X \cup Y = Y$.

Problem 2.5. Prove rigorously that if $X \subseteq Y$, then $X \cap Y = X$.

Problem 2.6. List all elements of $\{1, 2, 3\}^3$.

Problem 2.7. Show, by induction on k , that for all $k \geq 1$, if X has n elements, then X^k has n^k elements.

Chapter 3

Relations

3.1 Relations as Sets

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

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

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

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

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

Definition 3.1 (Binary relation). A *binary relation* on a set X is a subset of X^2 . If $R \subseteq X^2$ is a binary relation on X and $x, y \in X$, we write Rxy (or xRy) for $\langle x, y \rangle \in R$.

Example 3.2. The set \mathbb{N}^2 of pairs of natural numbers can be listed in a 2-dimensional matrix like this:

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

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

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

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

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

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

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

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

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

3.2 Special Properties of Relations

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

3.3. ORDERS

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

Definition 3.3 (Reflexivity). A relation $R \subseteq X^2$ is *reflexive* iff, for every $x \in X$, Rxx .

Definition 3.4 (Transitivity). A relation $R \subseteq X^2$ is *transitive* iff, whenever Rxy and Ryz , then also Rxz .

Definition 3.5 (Symmetry). A relation $R \subseteq X^2$ is *symmetric* iff, whenever Rxy , then also Ryx .

Definition 3.6 (Anti-symmetry). A relation $R \subseteq X^2$ is *anti-symmetric* iff, whenever both Rxy and Ryx , then $x = y$ (or, in other words: if $x \neq y$ then either $\neg Rxy$ or $\neg Ryx$).

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

Definition 3.7 (Connectivity). A relation $R \subseteq X^2$ is *connected* if for all $x, y \in X$, if $x \neq y$, then either Rxy or Ryx .

Definition 3.8 (Partial order). A relation $R \subseteq X^2$ that is reflexive, transitive, and anti-symmetric is called a *partial order*.

Definition 3.9 (Linear order). A partial order that is also connected is called a *linear order*.

Definition 3.10 (Equivalence relation). A relation $R \subseteq X^2$ that is reflexive, symmetric, and transitive is called an *equivalence relation*.

3.3 Orders

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

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

Definition 3.11 (Preorder). A relation which is both reflexive and transitive is called a *preorder*.

Definition 3.12 (Partial order). A preorder which is also anti-symmetric is called a *partial order*.

Definition 3.13 (Linear order). A partial order which is also connected is called a *total order* or *linear order*.

Example 3.14. Every linear order is also a partial order, and every partial order is also a preorder, but the converses don't hold. For instance, the identity relation and the full relation on X are preorders, but they are not partial orders, because they are not anti-symmetric (if X has more than one element). For a somewhat less silly example, consider the *no longer than* relation \preceq on \mathbb{B}^* : $x \preceq y$ iff $\text{len}(x) \leq \text{len}(y)$. This is a preorder, even a connected preorder, but not a partial order.

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

Notice that the examples L and G from [Example 3.2](#), although we said there that they were called "strict orders" are not linear orders even though they are connected (they are not reflexive). But there is a close connection, as we will see momentarily.

Definition 3.15 (Irreflexivity). A relation R on X is called *irreflexive* if, for all $x \in X$, $\neg Rxx$.

Definition 3.16 (Asymmetry). A relation R on X is called *asymmetric* if for no pair $x, y \in X$ we have Rxy and Ryx .

Definition 3.17 (Strict order). A *strict order* is a relation which is irreflexive, asymmetric, and transitive.

Definition 3.18 (Strict linear order). A strict order which is also connected is called a *strict linear order*.

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

3.4. GRAPHS

Proposition 3.19. 1. If R is a strict (linear) order on X , then $R^+ = R \cup \text{Id}_X$ is a partial order (linear order).

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

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

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

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

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

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

2. Exercise.

□

Example 3.20. \leq is the linear order corresponding to the strict linear order $<$. \subseteq is the partial order corresponding to the strict order \subsetneq .

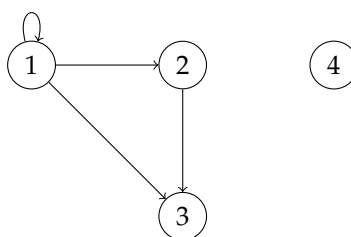
3.4 Graphs

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

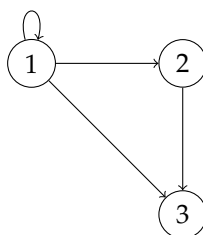
Definition 3.21 (Directed graph). A *directed graph* $G = \langle V, E \rangle$ is a set of *vertices* V and a set of *edges* $E \subseteq V^2$.

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

Example 3.22. The graph $\langle V, E \rangle$ with $V = \{1, 2, 3, 4\}$ and $E = \{\langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 3 \rangle, \langle 2, 3 \rangle\}$ looks like this:



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



3.5 Operations on Relations

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

Definition 3.23. Let $R, S \subseteq X^2$ be relations and Y a set.

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

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

3.5. OPERATIONS ON RELATIONS

3. The *restriction* $R \upharpoonright Y$ of R to Y is $R \cap Y^2$

4. The *application* $R[Y]$ of R to Y is

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

Example 3.24. Let $S \subseteq \mathbb{Z}^2$ be the successor relation on \mathbb{Z} , i.e., the set of pairs $\langle x, y \rangle$ where $x + 1 = y$, for $x, y \in \mathbb{Z}$. Sxy holds iff y is the successor of x .

1. The inverse S^{-1} of S is the predecessor relation, i.e., $S^{-1}xy$ iff $x - 1 = y$.

2. The relative product $S \mid S$ is the relation x bears to y if $x + 2 = y$.

3. The restriction of S to \mathbb{N} is the successor relation on \mathbb{N} .

4. The application of S to a set, e.g., $S[\{1, 2, 3\}]$ is $\{2, 3, 4\}$.

Definition 3.25 (Transitive closure). The *transitive closure* R^+ of a relation $R \subseteq X^2$ is $R^+ = \bigcup_{i=1}^{\infty} R^i$ where $R^1 = R$ and $R^{i+1} = R^i \mid R$.

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

Example 3.26. Take the successor relation $S \subseteq \mathbb{Z}^2$. S^2xy iff $x + 2 = y$, S^3xy iff $x + 3 = y$, etc. So R^*xy iff for some $i \geq 1$, $x + i = y$. In other words, S^+xy iff $x < y$ (and R^*xy iff $x \leq y$).

Problems

Problem 3.1. List the elements of the relation \subseteq on the set $\wp(\{a, b, c\})$.

Problem 3.2. Give examples of relations that are (a) reflexive and symmetric but not transitive, (b) reflexive and anti-symmetric, (c) anti-symmetric, transitive, but not reflexive, and (d) reflexive, symmetric, and transitive. Do not use relations on numbers or sets.

Problem 3.3. Complete the proof of [Proposition 3.19](#), i.e., prove that if R is a partial order on X , then $R^- = R \setminus \text{Id}_X$ is a strict order.

Problem 3.4. Consider the less-than-or-equal-to relation \leq on the set $\{1, 2, 3, 4\}$ as a graph and draw the corresponding diagram.

Problem 3.5. Show that the transitive closure of R is in fact transitive.

Chapter 4

Functions

4.1 Basics

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

Definition 4.1 (Function). A function $f: X \rightarrow Y$ is a mapping of each element of X to an element of Y . We call X the *domain* of f and Y the *codomain* of f . The elements of X are called inputs or *arguments* of f , and the element of Y that is paired with an argument x by f is called the *value* of f for argument x , written $f(x)$.

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

Example 4.2. Multiplication takes pairs of natural numbers as inputs and maps them to natural numbers as outputs, so goes from $\mathbb{N} \times \mathbb{N}$ (the domain) to \mathbb{N} (the codomain). As it turns out, the range is also \mathbb{N} , since every $n \in \mathbb{N}$ is $n \times 1$.

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

4.1. BASICS

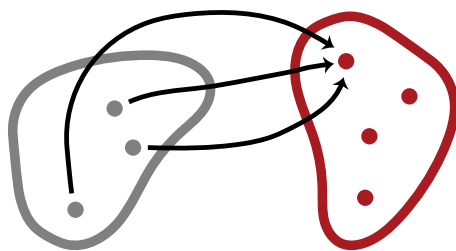


Figure 4.1: A function is a mapping of each element of one set to an element of another. An arrow points from an argument in the domain to the corresponding value in the codomain.

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

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

Example 4.3. Let $f: \mathbb{N} \rightarrow \mathbb{N}$ be defined such that $f(x) = x + 1$. This is a definition that specifies f as a function which takes in natural numbers and outputs natural numbers. It tells us that, given a natural number x , f will output its successor $x + 1$. In this case, the codomain \mathbb{N} is not the range of f , since the natural number 0 is not the successor of any natural number. The range of f is the set of all positive integers, \mathbb{Z}^+ .

Example 4.4. Let $g: \mathbb{N} \rightarrow \mathbb{N}$ be defined such that $g(x) = x + 2 - 1$. This tells us that g is a function which takes in natural numbers and outputs natural numbers. Given a natural number n , g will output the predecessor of the successor of the successor of x , i.e., $x + 1$. Despite their different definitions, g and f are the same function.

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

Example 4.5. We can also define functions by cases. For instance, we could define $h: \mathbb{N} \rightarrow \mathbb{N}$ by

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

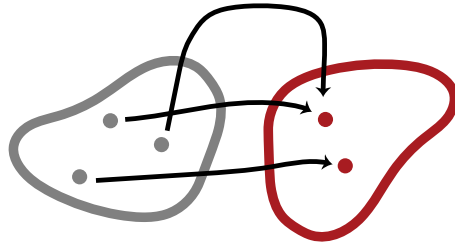


Figure 4.2: A surjective function has every element of the codomain as a value.



Figure 4.3: An injective function never maps two different arguments to the same value.

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

4.2 Kinds of Functions

Definition 4.6 (Surjective function). A function $f: X \rightarrow Y$ is *surjective* iff Y is also the range of f , i.e., for every $y \in Y$ there is at least one $x \in X$ such that $f(x) = y$.

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

Definition 4.7 (Injective function). A function $f: X \rightarrow Y$ is *injective* iff for each $y \in Y$ there is at most one $x \in X$ such that $f(x) = y$.

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

4.3. INVERSES OF FUNCTIONS

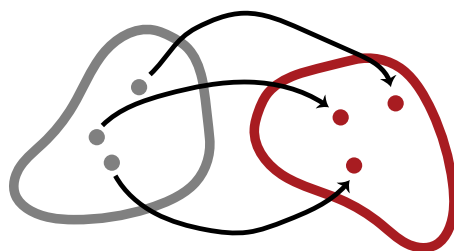


Figure 4.4: A bijective function uniquely pairs the elements of the codomain with those of the domain.

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

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

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

The function

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

is surjective, but not injective.

Definition 4.8 (Bijection). A function $f: X \rightarrow Y$ is *bijective* iff it is both surjective and injective. We call such a function a *bijection* from X to Y (or between X and Y).

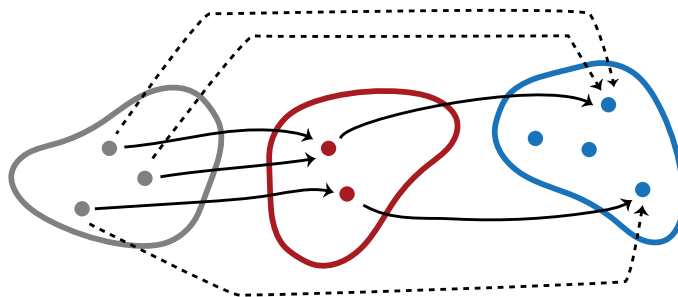
4.3 Inverses of Functions

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

Definition 4.9. A function $g: Y \rightarrow X$ is an *inverse* of a function $f: X \rightarrow Y$ if $f(g(y)) = y$ and $g(f(x)) = x$ for all $x \in X$ and $y \in Y$.

When do functions have inverses? A good candidate for an inverse of $f: X \rightarrow Y$ is $g: Y \rightarrow X$ “defined by”

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

Figure 4.5: The composition $g \circ f$ of two functions f and g .

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

Proposition 4.10. *If $f: X \rightarrow Y$ is bijective, f has a unique inverse $f^{-1}: Y \rightarrow X$.*

Proof. Exercise. □

4.4 Composition of Functions

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

Definition 4.11 (Composition). Let $f: X \rightarrow Y$ and $g: Y \rightarrow Z$. The *composition* of f with g is the function $(g \circ f): X \rightarrow Z$, where $(g \circ f)(x) = g(f(x))$.

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

Example 4.12. Consider the functions $f(x) = x + 1$, and $g(x) = 2x$. What function do you get when you compose these two? $(g \circ f)(x) = g(f(x))$. So that means for every natural number you give this function, you first add one,

4.5. ISOMORPHISM

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

4.5 Isomorphism

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

Definition 4.13 (Isomorphism). Let U be the pair $\langle X, R \rangle$ and V be the pair $\langle Y, S \rangle$ such that X and Y are sets and R and S are relations on X and Y respectively. A bijection f from X to Y is an *isomorphism* from U to V iff it preserves the relational structure, that is, for any x_1 and x_2 in X , $\langle x_1, x_2 \rangle \in R$ iff $\langle f(x_1), f(x_2) \rangle \in S$.

Example 4.14. Consider the following two sets $X = \{1, 2, 3\}$ and $Y = \{4, 5, 6\}$, and the relations less than and greater than. The function $f: X \rightarrow Y$ where $f(x) = 7 - x$ is an isomorphism between $\langle X, < \rangle$ and $\langle Y, > \rangle$.

4.6 Partial Functions

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

Definition 4.15. A *partial function* $f: X \rightarrow Y$ is a mapping which assigns to every element of X at most one element of Y . If f assigns an element of Y to $x \in X$, we say $f(x)$ is *defined*, and otherwise *undefined*. If $f(x)$ is defined, we write $f(x) \downarrow$, otherwise $f(x) \uparrow$. The *domain* of a partial function f is the subset of X where it is defined, i.e., $\text{dom}(f) = \{x : f(x) \downarrow\}$.

Example 4.16. Every function $f: X \rightarrow Y$ is also a partial function. Partial functions that are defined everywhere on X —i.e., what we so far have simply called a function—are also called *total functions*.

Example 4.17. The partial function $f: \mathbb{R} \rightarrow \mathbb{R}$ given by $f(x) = 1/x$ is undefined for $x = 0$, and defined everywhere else.

4.7 Functions and Relations

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

Definition 4.18 (Graph of a function). Let $f: X \rightarrow Y$ be a partial function. The *graph* of f is the relation $R_f \subseteq X \times Y$ defined by

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

Proposition 4.19. Suppose $R \subseteq X \times Y$ has the property that whenever Rxy and Rxy' then $y = y'$. Then R is the graph of the partial function $f: X \rightarrow Y$ defined by: if there is a y such that Rxy , then $f(x) = y$, otherwise $f(x) \uparrow$. If R is also serial, i.e., for each $x \in X$ there is a $y \in Y$ such that Rxy , then f is total.

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

Problems

Problem 4.1. Show that if f is bijective, an inverse g of f exists, i.e., define such a g , show that it is a function, and show that it is an inverse of f , i.e., $f(g(y)) = y$ and $g(f(x)) = x$ for all $x \in X$ and $y \in Y$.

Problem 4.2. Show that if $f: X \rightarrow Y$ has an inverse g , then f is bijective.

Problem 4.3. Show that if $g: Y \rightarrow X$ and $g': Y \rightarrow X$ are inverses of $f: X \rightarrow Y$, then $g = g'$, i.e., for all $y \in Y$, $g(y) = g'(y)$.

Problem 4.4. Show that if $f: X \rightarrow Y$ and $g: Y \rightarrow Z$ are both injective, then $g \circ f: X \rightarrow Z$ is injective.

Problem 4.5. Show that if $f: X \rightarrow Y$ and $g: Y \rightarrow Z$ are both surjective, then $g \circ f: X \rightarrow Z$ is surjective.

Problem 4.6. Given $f: X \rightarrow Y$, define the partial function $g: Y \rightarrow X$ by: for any $y \in Y$, if there is a unique $x \in X$ such that $f(x) = y$, then $g(y) = x$; otherwise $g(y) \uparrow$. Show that if f is injective, then $g(f(x)) = x$ for all $x \in \text{dom}(f)$, and $f(g(y)) = y$ for all $y \in \text{ran}(f)$.

Problem 4.7. Suppose $f: X \rightarrow Y$ and $g: Y \rightarrow Z$. Show that the graph of $(g \circ f)$ is $R_f \mid R_g$.

Chapter 5

The Size of Sets

5.1 Introduction

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

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

5.2 Enumerable Sets

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

Definition 5.1 (Enumeration). Informally, an *enumeration* of a set X is a list (possibly infinite) of elements of X such that every element of X appears on the list at some finite position. If X has an enumeration, then X is said to be *enumerable*. If X is enumerable and infinite, we say X is denumerable.

A couple of points about enumerations:

1. We count as enumerations only lists which have a beginning and in which every element other than the first has a single element immediately preceding it. In other words, there are only finitely many elements between the first element of the list and any other element. In particular, this means that every element of an enumeration has a finite position: the first element has position 1, the second position 2, etc.
2. We can have different enumerations of the same set X which differ by the order in which the elements appear: 4, 1, 25, 16, 9 enumerates the (set of the) first five square numbers just as well as 1, 4, 9, 16, 25 does.
3. Redundant enumerations are still enumerations: 1, 1, 2, 2, 3, 3, ... enumerates the same set as 1, 2, 3, ... does.
4. Order and redundancy *do* matter when we specify an enumeration: we can enumerate the positive integers beginning with 1, 2, 3, 1, ..., but the pattern is easier to see when enumerated in the standard way as 1, 2, 3, 4, ...
5. Enumerations must have a beginning: ..., 3, 2, 1 is not an enumeration of the natural numbers because it has no first element. To see how this follows from the informal definition, ask yourself, "at what position in the list does the number 76 appear?"
6. The following is not an enumeration of the positive integers: 1, 3, 5, ..., 2, 4, 6, ... The problem is that the even numbers occur at places $\infty + 1$, $\infty + 2$, $\infty + 3$, rather than at finite positions.
7. Lists may be gappy: 2, -, 4, -, 6, -, ... enumerates the even positive integers.
8. The empty set is enumerable: it is enumerated by the empty list!

Proposition 5.2. *If X has an enumeration, it has an enumeration without gaps or repetitions.*

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

5.2. ENUMERABLE SETS

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

Definition 5.3 (Enumeration). An *enumeration* of a set X is any surjective function $f: \mathbb{Z}^+ \rightarrow X$.

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

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

Definition 5.4. A set X is enumerable iff it is empty or has an enumeration.

Example 5.5. A function enumerating the positive integers (\mathbb{Z}^+) is simply the identity function given by $f(n) = n$. A function enumerating the natural numbers \mathbb{N} is the function $g(n) = n - 1$.

Example 5.6. The functions $f: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ and $g: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ given by

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

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

Example 5.7. The function $f(n) = (-1)^n \lceil \frac{n-1}{2} \rceil$ (where $\lceil x \rceil$ denotes the *ceiling* function, which rounds x up to the nearest integer) enumerates the set of integers \mathbb{Z} . Notice how f generates the values of \mathbb{Z} by "hopping" back and forth between positive and negative integers:

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

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

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

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

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

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

	1	2	3	4	...
1	$\langle 1, 1 \rangle$	$\langle 1, 2 \rangle$	$\langle 1, 3 \rangle$	$\langle 1, 4 \rangle$...
2	$\langle 2, 1 \rangle$	$\langle 2, 2 \rangle$	$\langle 2, 3 \rangle$	$\langle 2, 4 \rangle$...
3	$\langle 3, 1 \rangle$	$\langle 3, 2 \rangle$	$\langle 3, 3 \rangle$	$\langle 3, 4 \rangle$...
4	$\langle 4, 1 \rangle$	$\langle 4, 2 \rangle$	$\langle 4, 3 \rangle$	$\langle 4, 4 \rangle$...
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

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

	1	2	4	7	...
1	1	2	4	7	...
3	3	5	8
6	6	9
10	10
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

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

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

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

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

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

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

5.3. NON-ENUMERABLE SETS

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

	1	2	3	4	...
$\langle 1, 1 \rangle$	$\langle 1, 1, 1 \rangle$	$\langle 1, 1, 2 \rangle$	$\langle 1, 1, 3 \rangle$	$\langle 1, 1, 4 \rangle$...
$\langle 1, 2 \rangle$	$\langle 1, 2, 1 \rangle$	$\langle 1, 2, 2 \rangle$	$\langle 1, 2, 3 \rangle$	$\langle 1, 2, 4 \rangle$...
$\langle 2, 1 \rangle$	$\langle 2, 1, 1 \rangle$	$\langle 2, 1, 2 \rangle$	$\langle 2, 1, 3 \rangle$	$\langle 2, 1, 4 \rangle$...
$\langle 1, 3 \rangle$	$\langle 1, 3, 1 \rangle$	$\langle 1, 3, 2 \rangle$	$\langle 1, 3, 3 \rangle$	$\langle 1, 3, 4 \rangle$...
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

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

5.3 Non-enumerable Sets

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

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

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

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

Theorem 5.8. \mathbb{B}^ω is non-enumerable.

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

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

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

	1	2	3	4	...
1	$s_1(1)$	$s_1(2)$	$s_1(3)$	$s_1(4)$...
2	$s_2(1)$	$s_2(2)$	$s_2(3)$	$s_2(4)$...
3	$s_3(1)$	$s_3(2)$	$s_3(3)$	$s_3(4)$...
4	$s_4(1)$	$s_4(2)$	$s_4(3)$	$s_4(4)$...
\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

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

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

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

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

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

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

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

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

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

5.3. NON-ENUMERABLE SETS

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

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

Theorem 5.9. $\wp(\mathbb{Z}^+)$ is not enumerable.

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

$$Z_1, Z_2, Z_3, \dots$$

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

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

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

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

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

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

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

5.4 Reduction

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

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

Proof of Theorem 5.9 by reduction. Suppose that $\wp(\mathbb{Z}^+)$ were enumerable, and thus that there is an enumeration of it, Z_1, Z_2, Z_3, \dots

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

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

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

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

Now consider the list

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

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

So if $\wp(\mathbb{Z}^+)$ were enumerable, \mathbb{B}^ω would be enumerable. But \mathbb{B}^ω is non-enumerable (Theorem 5.8). Hence $\wp(\mathbb{Z}^+)$ is non-enumerable. \square

It is easy to be confused about the direction the reduction goes in. For instance, a surjective function $g: \mathbb{B}^\omega \rightarrow X$ does *not* establish that X is non-enumerable. (Consider $g: \mathbb{B}^\omega \rightarrow \mathbb{B}$ defined by $g(s) = s(1)$, the function that

5.5. EQUINUMEROUS SETS

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

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

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

5.5 Equinumerous Sets

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

Reflexivity: Every set is equinumerous with itself.

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

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

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

Definition 5.10. A set X is *equinumerous* with a set Y , $X \approx Y$, if and only if there is a bijective $f: X \rightarrow Y$.

Proposition 5.11. *Equinumerosity defines an equivalence relation.*

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

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

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

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

Therefore, equinumerosity is an equivalence relation. \square

Theorem 5.12. *Suppose X and Y are equinumerous. Then X is enumerable if and only if Y is.*

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

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

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

5.6 Comparing Sizes of Sets

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

Definition 5.13. X is *no larger than* Y , $X \preceq Y$, if and only if there is an injective function $f: X \rightarrow Y$.

Theorem 5.14 (Schröder-Bernstein). *Let X and Y be sets. If $X \preceq Y$ and $Y \preceq X$, then $X \approx Y$.*

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

5.6. COMPARING SIZES OF SETS

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

Definition 5.15. X is *smaller than* Y , $X \prec Y$, if and only if there is an injective function $f: X \rightarrow Y$ but no bijective $g: X \rightarrow Y$.

Theorem 5.16 (Cantor). *For all X , $X \prec \wp(X)$.*

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

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

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

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

It's instructive to compare the proof of [Theorem 5.16](#) to that of [Theorem 5.9](#). There we showed that for any list Z_1, Z_2, \dots , of subsets of \mathbb{Z}^+ one can construct a set \bar{Z} of numbers guaranteed not to be on the list. It was guaranteed not to be on the list because, for every $n \in \mathbb{Z}^+$, $n \in Z_n$ iff $n \notin \bar{Z}$. This way, there is always some number that is an element of one of Z_n and \bar{Z} but not the other. We follow the same idea here, except the indices n are now elements of X instead of \mathbb{Z}^+ . The set \bar{Y} is defined so that it is different from $g(x)$ for each $x \in X$, because $x \in g(x)$ iff $x \notin \bar{Y}$. Again, there is always an element of X which is an element of one of $g(x)$ and \bar{Y} but not the other. And just as \bar{Z} therefore cannot be on the list Z_1, Z_2, \dots , \bar{Y} cannot be in the range of g .

Problems

Problem 5.1. According to [Definition 5.4](#), a set X is enumerable iff $X = \emptyset$ or there is a surjective $f: \mathbb{Z}^+ \rightarrow X$. It is also possible to define “enumerable set” precisely by: a set is enumerable iff there is an injective function $g: X \rightarrow \mathbb{Z}^+$. Show that the definitions are equivalent, i.e., show that there is an injective function $g: X \rightarrow \mathbb{Z}^+$ iff either $X = \emptyset$ or there is a surjective $f: \mathbb{Z}^+ \rightarrow X$.

Problem 5.2. Define an enumeration of the positive squares 4, 9, 16, ...

Problem 5.3. Show that if X and Y are enumerable, so is $X \cup Y$.

Problem 5.4. Show by induction on n that if X_1, X_2, \dots, X_n are all enumerable, so is $X_1 \cup \dots \cup X_n$.

Problem 5.5. Give an enumeration of the set of all positive rational numbers. (A positive rational number is one that can be written as a fraction n/m with $n, m \in \mathbb{Z}^+$).

Problem 5.6. Show that \mathbb{Q} is enumerable. (A rational number is one that can be written as a fraction z/m with $z \in \mathbb{Z}, m \in \mathbb{Z}^+$).

Problem 5.7. Define an enumeration of \mathbb{B}^* .

Problem 5.8. Recall from your introductory logic course that each possible truth table expresses a truth function. In other words, the truth functions are all functions from $\mathbb{B}^k \rightarrow \mathbb{B}$ for some k . Prove that the set of all truth functions is enumerable.

Problem 5.9. Show that the set of all finite subsets of an arbitrary infinite enumerable set is enumerable.

Problem 5.10. A set of positive integers is said to be *cofinite* iff it is the complement of a finite set of positive integers. Let I be the set that contains all the finite and cofinite sets of positive integers. Show that I is enumerable.

Problem 5.11. Show that the enumerable union of enumerable sets is enumerable. That is, whenever X_1, X_2, \dots are sets, and each X_i is enumerable, then the union $\bigcup_{i=1}^{\infty} X_i$ of all of them is also enumerable.

Problem 5.12. Show that $\wp(\mathbb{N})$ is non-enumerable by a diagonal argument.

Problem 5.13. Show that the set of functions $f: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ is non-enumerable by an explicit diagonal argument. That is, show that if f_1, f_2, \dots , is a list of functions and each $f_i: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$, then there is some $\bar{f}: \mathbb{Z}^+ \rightarrow \mathbb{Z}^+$ not on this list.

Problem 5.14. Show that if there is an injective function $g: Y \rightarrow X$, and Y is non-enumerable, then so is X . Do this by showing how you can use g to turn an enumeration of X into one of Y .

Problem 5.15. Show that the set of all *sets of* pairs of positive integers is non-enumerable by a reduction argument.

Problem 5.16. Show that \mathbb{N}^ω , the set of infinite sequences of natural numbers, is non-enumerable by a reduction argument.

5.6. COMPARING SIZES OF SETS

Problem 5.17. Let P be the set of functions from the set of positive integers to the set $\{0\}$, and let Q be the set of *partial* functions from the set of positive integers to the set $\{0\}$. Show that P is enumerable and Q is not. (Hint: reduce the problem of enumerating \mathbb{B}^ω to enumerating Q).

Problem 5.18. Let S be the set of all surjective functions from the set of positive integers to the set $\{0,1\}$, i.e., S consists of all surjective $f: \mathbb{Z}^+ \rightarrow \mathbb{B}$. Show that S is non-enumerable.

Problem 5.19. Show that the set \mathbb{R} of all real numbers is non-enumerable.

Problem 5.20. Show that if X is equinumerous with U and Y is equinumerous with V , and the intersections $X \cap Y$ and $U \cap V$ are empty, then the unions $X \cup Y$ and $U \cup V$ are equinumerous.

Problem 5.21. Show that if X is infinite and enumerable, then it is equinumerous with the positive integers \mathbb{Z}^+ .

Problem 5.22. Show that there cannot be an injective function $g: \wp(X) \rightarrow X$, for any set X . Hint: Suppose $g: \wp(X) \rightarrow X$ is injective. Then for each $x \in X$ there is at most one $Y \subseteq X$ such that $g(Y) = x$. Define a set \bar{Y} such that for every $x \in X$, $g(\bar{Y}) \neq x$.

Part II

First-order Logic

5.6. COMPARING SIZES OF SETS

This part covers the metatheory of first-order logic through completeness. Currently it does not rely on a separate treatment of propositional logic. It is planned, however, to separate the propositional and quantifier material on semantics and proof theory so that propositional logic can be covered independently. This will become important especially when material on propositional modal logic will be added, since then one might *not* want to cover quantifiers. Currently two different proof systems are offered as alternatives, (a version of) sequent calculus and natural deduction. A third alternative treatment based on Enderton-style axiomatic deduction is available in experimental form in the branch “axiomatic-deduction”. In particular, this part needs an introduction (issue #69).

Chapter 6

Syntax and Semantics

6.1 Introduction

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

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

6.2. FIRST-ORDER LANGUAGES

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

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

6.2 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), \leftrightarrow (biconditional), \forall (universal quantifier), \exists (existential quantifier).
 - b) The propositional constant for falsity \perp .
 - c) The propositional constant for truth \top .
 - d) The two-place identity predicate $=$.
 - e) A denumerable set of variables: v_0, v_1, v_2, \dots
2. Non-logical symbols, making up the *standard language* of first-order logic
 - a) A denumerable set of n -place predicate symbols for each $n > 0$: $A_0^n, A_1^n, A_2^n, \dots$

- b) A denumerable set of constant symbols: c_0, c_1, c_2, \dots
 - c) A denumerable set of n -place function symbols for each $n > 0$: $f_0^n, f_1^n, f_2^n, \dots$
3. Punctuation marks: $(,)$, and the comma.

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

Example 6.1. The language \mathcal{L}_A of arithmetic contains a single two-place predicate symbol $<$, a single constant symbol 0 , one one-place function symbol $!$, and two two-place function symbols $+$ and \times .

Example 6.2. The language of set theory \mathcal{L}_Z contains only the single two-place predicate symbol \in .

Example 6.3. 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_0^2 , 0 for c_0 , $!$ for f_0^1 , $+$ for f_0^2 , \times for f_1^2 .

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 \perp symbol is variously called “falsity,” “falsum,” “absurdity,” or “bottom.” The \top symbol is variously called “truth,” “verum,” or “top.”

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

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

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

6.3 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 6.4 (Terms). The set of *terms* $\text{Trm}(\mathcal{L})$ 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, \dots, t_n are terms, then $f(t_1, \dots, t_n)$ 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 $f(t_1, \dots, t_n)$ as just f by itself if $n = 0$.

Definition 6.5 (Formula). The set of *formulas* $\text{Frm}(\mathcal{L})$ of the language \mathcal{L} is defined inductively as follows:

1. \perp is an atomic formula.
2. \top is an atomic formula.
3. If R is an n -place predicate symbol of \mathcal{L} and t_1, \dots, t_n are terms of \mathcal{L} , then $R(t_1, \dots, t_n)$ is an atomic formula.
4. If t_1 and t_2 are terms of \mathcal{L} , then $=(t_1, t_2)$ is an atomic formula.
5. If φ is a formula, then $\neg\varphi$ is formula.
6. If φ and ψ are formulas, then $(\varphi \wedge \psi)$ is a formula.
7. If φ and ψ are formulas, then $(\varphi \vee \psi)$ is a formula.
8. If φ and ψ are formulas, then $(\varphi \rightarrow \psi)$ is a formula.
9. If φ and ψ are formulas, then $(\varphi \leftrightarrow \psi)$ is a formula.
10. If φ is a formula and x is a variable, then $\forall x \varphi$ is a formula.
11. If φ is a formula and x is a variable, then $\exists x \varphi$ is a formula.

12. 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 \top , \perp , $R(t_1, \dots, t_n)$ 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.

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 $A_0^2(t_1, t_2)$, $f_0^2(t_1, t_2)$, $A_0^2(t_1, t_2)$ and $f_0^1(t)$, respectively.

Definition 6.6 (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.

6.4 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

6.4. UNIQUE READABILITY

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 interpretation—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 $\varphi \rightarrow \psi$.

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

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

Correspondingly, there are two ways to “read” the formula $\theta \rightarrow \theta \rightarrow \theta$. It is of the form $\psi \rightarrow \chi$ where ψ is θ and χ is $\theta \rightarrow \theta$, but *it is also* of the form $\psi \rightarrow \chi$ with ψ being $\theta \rightarrow \theta$ 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 $\psi \rightarrow \chi$, the main operator is the indicated occurrence of \rightarrow . But if we can match the formula $\theta \rightarrow \theta \rightarrow \theta$ with $\psi \rightarrow \chi$ in the two different ways mentioned above, then in one case we get the first occurrence of \rightarrow 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 6.7. *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. $\varphi \equiv \perp$: φ has 0 left and 0 right parentheses.
2. $\varphi \equiv \top$: φ has 0 left and 0 right parentheses.

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

□

Definition 6.8 (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 6.9. *If φ is a formula, and ψ is a proper prefix of φ , then ψ is not a formula.*

Proof. Exercise.

□

Proposition 6.10. *If φ is an atomic formula, then it satisfies one, and only one of the following conditions.*

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

Proof. Exercise.

□

Proposition 6.11 (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 $(\psi \vee \chi)$.

6.5. MAIN OPERATOR OF A FORMULA

5. φ is of the form $(\psi \rightarrow \chi)$.
6. φ is of the form $(\psi \leftrightarrow \chi)$.
7. φ is of the form $\forall x \psi$.
8. 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 $(\psi \rightarrow \chi)$ and $(\psi' \rightarrow \chi')$.

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, \perp , \top .

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 6.9](#). \square

6.5 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 6.12 (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. $\varphi \equiv (\psi \rightarrow \chi)$: the main operator of φ is \rightarrow .
6. $\varphi \equiv (\psi \leftrightarrow \chi)$: the main operator of φ is \leftrightarrow .
7. $\varphi \equiv \forall x \psi$: the main operator of φ is \forall .
8. $\varphi \equiv \exists x \psi$: 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 $((\theta \rightarrow \alpha) \rightarrow (\alpha \rightarrow \theta))$ is of the form $(\psi \rightarrow \chi)$ where ψ is $(\theta \rightarrow \alpha)$ and χ is $(\alpha \rightarrow \theta)$, the second occurrence of \rightarrow 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 6.12. 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)	$\perp, \top, R(t_1, \dots, t_n), t_1 = t_2$
\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)	$\exists x \varphi$

6.6 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 6.13 (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 6.14 (Proper Subformula). If φ is a formula, the *proper subformulas* of φ are recursively as follows:

1. Atomic formulas have no proper subformulas.

6.7. FREE VARIABLES AND SENTENCES

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 6.15 (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 .

6.7 Free Variables and Sentences

Definition 6.16 (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 6.17 (Bound Variables). An occurrence of a variable in a formula φ is *bound* if it is not free.

Definition 6.18 (Scope). If $\forall x \psi$ is an occurrence of a subformula in a formula φ , then the corresponding occurrence of ψ in φ is called the *scope* of the corresponding occurrence of $\forall x$. Similarly for $\exists x$.

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

Example 6.19. 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 occurrence of v_0 in ψ , but does not bind the occurrence of v_1 . So v_1 is a free variable in this case.

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

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

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

6.8 Substitution

Definition 6.21 (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 $y \neq x$.
3. $s \equiv x$: $s[t/x]$ is t .
4. $s \equiv f(t_1, \dots, t_n)$: $s[t/x]$ is $f(t_1[t/x], \dots, t_n[t/x])$.

Definition 6.22. 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 .

6.8. SUBSTITUTION

Example 6.23.

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

1. $\varphi \equiv \perp$: $\varphi[t/x]$ is \perp .
2. $\varphi \equiv \top$: $\varphi[t/x]$ is \top .
3. $\varphi \equiv P(t_1, \dots, t_n)$: $\varphi[t/x]$ is $P(t_1[t/x], \dots, t_n[t/x])$.
4. $\varphi \equiv t_1 = t_2$: $\varphi[t/x]$ is $t_1[t/x] = t_2[t/x]$.
5. $\varphi \equiv \neg\psi$: $\varphi[t/x]$ is $\neg\psi[t/x]$.
6. $\varphi \equiv (\psi \wedge \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \wedge \chi[t/x])$.
7. $\varphi \equiv (\psi \vee \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \vee \chi[t/x])$.
8. $\varphi \equiv (\psi \rightarrow \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \rightarrow \chi[t/x])$.
9. $\varphi \equiv (\psi \leftrightarrow \chi)$: $\varphi[t/x]$ is $(\psi[t/x] \leftrightarrow \chi[t/x])$.
10. $\varphi \equiv \forall y \psi$: $\varphi[t/x]$ is $\forall y \psi[t/x]$, provided y is a variable other than x ; otherwise $\varphi[t/x]$ is just φ .
11. $\varphi \equiv \exists y \psi$: $\varphi[t/x]$ 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 cumbersome 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 $\varphi(x)[t/x]$.

6.9 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, satisfiability. They are variously called “structures,” “interpretations,” or “models” in the literature.

Definition 6.25 (Structures). A *structure* \mathfrak{M} , for a language \mathcal{L} of first-order logic consists of the following elements:

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

Example 6.26. A structure \mathfrak{M} for the language of arithmetic consists of a set, an element of $|\mathfrak{M}|$, $o^{\mathfrak{M}}$, as interpretation of the constant symbol o , a one-place function $r^{\mathfrak{M}}: |\mathfrak{M}| \rightarrow |\mathfrak{M}|$, two two-place functions $+^{\mathfrak{M}}$ and $\times^{\mathfrak{M}}$, both $|\mathfrak{M}|^2 \rightarrow |\mathfrak{M}|$, 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. $o^{\mathfrak{N}} = 0$
3. $r^{\mathfrak{N}}(n) = n + 1$ for all $n \in \mathbb{N}$
4. $+^{\mathfrak{N}}(n, m) = n + m$ for all $n, m \in \mathbb{N}$
5. $\times^{\mathfrak{N}}(n, m) = n \cdot m$ for all $n, m \in \mathbb{N}$
6. $<^{\mathfrak{N}} = \{\langle n, m \rangle : n \in \mathbb{N}, m \in \mathbb{N}, n < m\}$

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

6.10. COVERED STRUCTURES FOR FIRST-ORDER LANGUAGES

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

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

1. $|\mathfrak{H}\mathfrak{F}| = \emptyset \cup \wp(\emptyset) \cup \wp(\wp(\emptyset)) \cup \wp(\wp(\wp(\emptyset))) \cup \dots;$
2. $\in^{\mathfrak{H}\mathfrak{F}} = \{ \langle x, y \rangle : x, y \in |\mathfrak{H}\mathfrak{F}|, x \in y \}.$

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 $\exists x \varphi(x)$.

6.10 Covered Structures for First-order Languages

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

Definition 6.28 (Value of closed terms). If t is a closed term of the language \mathcal{L} and \mathfrak{M} is a structure for \mathcal{L} , the *value* $\text{Val}^{\mathfrak{M}}(t)$ is defined as follows:

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

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

Definition 6.29 (Covered structure). A structure is *covered* if every element of the domain is the value of some closed term.

Example 6.30. Let \mathcal{L} be the language with constant symbols *zero*, *one*, *two*, \dots , the binary predicate symbol $<$, and the binary function symbols $+$ and \times . Then a structure \mathfrak{M} for \mathcal{L} is the one with domain $|\mathfrak{M}| = \{0, 1, 2, \dots\}$ and assignments $\text{zero}^{\mathfrak{M}} = 0$, $\text{one}^{\mathfrak{M}} = 1$, $\text{two}^{\mathfrak{M}} = 2$, 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 $\times(\text{two}, +(\text{three}, \text{zero}))$ (or in infix notation, $\text{two} \times (\text{three} + \text{zero})$) is

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

6.11 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 6.31 (Variable Assignment). A *variable assignment* s for a structure \mathfrak{M} is a function which maps each variable to an element of $|\mathfrak{M}|$, i.e., $s: \text{Var} \rightarrow |\mathfrak{M}|$.

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

6.11. SATISFACTION OF A FORMULA IN A STRUCTURE

the variable assignment specifies it; for more complex terms it is computed recursively using the functions the structure assigns to the function symbols.

Definition 6.32 (Value of Terms). If t is a term of the language \mathcal{L} , \mathfrak{M} is a structure for \mathcal{L} , and s is a variable assignment for \mathfrak{M} , the *value* $\text{Val}_s^{\mathfrak{M}}(t)$ is defined as follows:

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

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

Definition 6.33 (x -Variant). If s is a variable assignment for a structure \mathfrak{M} , then any variable assignment s' for \mathfrak{M} which differs from s at most in what it assigns to x is called an x -variant of s . If s' is an x -variant of s we write $s \sim_x s'$.

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

Definition 6.34 (Satisfaction). Satisfaction of a formula φ in a structure \mathfrak{M} relative to a variable assignment s , in symbols: $\mathfrak{M}, s \models \varphi$, is defined recursively as follows. (We write $\mathfrak{M}, s \not\models \varphi$ to mean “not $\mathfrak{M}, s \models \varphi$.”)

1. $\varphi \equiv \perp$: $\mathfrak{M}, s \not\models \varphi$.
2. $\varphi \equiv \top$: $\mathfrak{M}, s \models \varphi$.
3. $\varphi \equiv R(t_1, \dots, t_n)$: $\mathfrak{M}, s \models \varphi$ iff $\langle \text{Val}_s^{\mathfrak{M}}(t_1), \dots, \text{Val}_s^{\mathfrak{M}}(t_n) \rangle \in R^{\mathfrak{M}}$.
4. $\varphi \equiv t_1 = t_2$: $\mathfrak{M}, s \models \varphi$ iff $\text{Val}_s^{\mathfrak{M}}(t_1) = \text{Val}_s^{\mathfrak{M}}(t_2)$.
5. $\varphi \equiv \neg\psi$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \not\models \psi$.
6. $\varphi \equiv (\psi \wedge \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \models \psi$ and $\mathfrak{M}, s \models \chi$.
7. $\varphi \equiv (\psi \vee \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \models \psi$ or $\mathfrak{M}, s \models \chi$ (or both).
8. $\varphi \equiv (\psi \rightarrow \chi)$: $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s \not\models \psi$ or $\mathfrak{M}, s \models \chi$ (or both).
9. $\varphi \equiv (\psi \leftrightarrow \chi)$: $\mathfrak{M}, s \models \varphi$ iff either both $\mathfrak{M}, s \models \psi$ and $\mathfrak{M}, s \models \chi$, or neither $\mathfrak{M}, s \models \psi$ nor $\mathfrak{M}, s \models \chi$.
10. $\varphi \equiv \forall x \psi$: $\mathfrak{M}, s \models \varphi$ iff for every x -variant s' of s , $\mathfrak{M}, s' \models \psi$.
11. $\varphi \equiv \exists x \psi$: $\mathfrak{M}, s \models \varphi$ iff there is an x -variant s' of s so that $\mathfrak{M}, s' \models \psi$.

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

Example 6.35. Let $=\{a, b, f, R\}$ 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

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

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

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

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

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

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

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

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

6.11. SATISFACTION OF A FORMULA IN A STRUCTURE

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 $R(a, a) \rightarrow (R(b, x) \vee R(x, b))$ is the \rightarrow , and

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

For a more complicated case, consider

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

Since $\mathfrak{M}, s_3 \not\models R(a, x)$ and $\mathfrak{M}, s_4 \not\models R(a, x)$, the interesting cases where we have to worry about the consequent of the conditional are only s_1 and s_2 . Does $\mathfrak{M}, s_1 \models \exists y R(x, y)$ hold? It does if there is at least one y -variant s'_1 of s_1 so that $\mathfrak{M}, s'_1 \models R(x, y)$. In fact, s_1 is such a y -variant ($s_1(x) = 1, s_1(y) = 1$, and $\langle 1, 1 \rangle \in R^{\mathfrak{M}}$), so the answer is yes. To determine if $\mathfrak{M}, s_2 \models \exists y R(x, y)$ we have to look at the y -variants of s_2 . Here, s_2 itself does not satisfy $R(x, y)$ ($s_2(x) = 2, s_2(y) = 1$, and $\langle 2, 1 \rangle \notin R^{\mathfrak{M}}$). However, consider $s'_2 \sim_y s_2$ with $s'_2(y) = 3$. $\mathfrak{M}, s'_2 \models R(x, y)$ since $\langle 2, 3 \rangle \in R^{\mathfrak{M}}$, and so $\mathfrak{M}, s_2 \models \exists y R(x, y)$. In sum, for every x -variant s_i of s , either $\mathfrak{M}, s_i \not\models R(a, x)$ ($i = 3, 4$) or $\mathfrak{M}, s_i \models \exists y R(x, y)$ ($i = 1, 2$), and so

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

On the other hand,

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

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

6.12 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 6.36. *If the variables in a term t are among x_1, \dots, x_n , and $s_1(x_i) = s_2(x_i)$ for $i = 1, \dots, n$, then $\text{Val}_{s_1}^{\mathfrak{M}}(t) = \text{Val}_{s_2}^{\mathfrak{M}}(t)$.*

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, \dots, x_n . If $t = c$, then $\text{Val}_{s_1}^{\mathfrak{M}}(t) = c^{\mathfrak{M}} = \text{Val}_{s_2}^{\mathfrak{M}}(t)$. If $t = x_i$, $s_1(x_i) = s_2(x_i)$ by the hypothesis of the proposition, and so $\text{Val}_{s_1}^{\mathfrak{M}}(t) = s_1(x_i) = s_2(x_i) = \text{Val}_{s_2}^{\mathfrak{M}}(t)$.

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

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

6.12. VARIABLE ASSIGNMENTS

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

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

□

Proposition 6.37. *If the free variables in φ are among x_1, \dots, x_n , and $s_1(x_i) = s_2(x_i)$ for $i = 1, \dots, n$, then $\mathfrak{M}, s_1 \models \varphi$ iff $\mathfrak{M}, s_2 \models \varphi$.*

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

1. $\varphi \equiv \top$: both $\mathfrak{M}, s_1 \models \varphi$ and $\mathfrak{M}, s_2 \models \varphi$.
2. $\varphi \equiv \perp$: both $\mathfrak{M}, s_1 \not\models \varphi$ and $\mathfrak{M}, s_2 \not\models \varphi$.
3. $\varphi \equiv R(t_1, \dots, t_k)$: let $\mathfrak{M}, s_1 \models \varphi$. Then

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

For $i = 1, \dots, k$, $\text{Val}_{s_1}^{\mathfrak{M}}(t_i) = \text{Val}_{s_2}^{\mathfrak{M}}(t_i)$ by [Proposition 6.36](#). So we also have $\langle \text{Val}_{s_2}^{\mathfrak{M}}(t_1), \dots, \text{Val}_{s_2}^{\mathfrak{M}}(t_k) \rangle \in R^{\mathfrak{M}}$.

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

$$\begin{aligned} \text{Val}_{s_2}^{\mathfrak{M}}(t_1) &= \text{Val}_{s_1}^{\mathfrak{M}}(t_1) && \text{(by Proposition 6.36)} \\ &= \text{Val}_{s_1}^{\mathfrak{M}}(t_2) && \text{(since } \mathfrak{M}, s_1 \models t_1 = t_2 \text{)} \\ &= \text{Val}_{s_2}^{\mathfrak{M}}(t_2) && \text{(by Proposition 6.36),} \end{aligned}$$

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

Now assume $\mathfrak{M}, s_1 \models \psi$ iff $\mathfrak{M}, s_2 \models \psi$ for all 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 $\mathfrak{M}, s_1 \models \varphi$, then $\mathfrak{M}, s_1 \not\models \psi$, so by the induction hypothesis, $\mathfrak{M}, s_2 \not\models \psi$, hence $\mathfrak{M}, s_2 \models \varphi$.
2. $\varphi \equiv \psi \wedge \chi$: if $\mathfrak{M}, s_1 \models \varphi$, then $\mathfrak{M}, s_1 \models \psi$ and $\mathfrak{M}, s_1 \models \chi$, so by induction hypothesis, $\mathfrak{M}, s_2 \models \psi$ and $\mathfrak{M}, s_2 \models \chi$. Hence, $\mathfrak{M}, s_2 \models \varphi$.
3. $\varphi \equiv \psi \vee \chi$: if $\mathfrak{M}, s_1 \models \varphi$, then $\mathfrak{M}, s_1 \models \psi$ or $\mathfrak{M}, s_1 \models \chi$. By induction hypothesis, $\mathfrak{M}, s_2 \models \psi$ or $\mathfrak{M}, s_2 \models \chi$, so $\mathfrak{M}, s_2 \models \varphi$.
4. $\varphi \equiv \psi \rightarrow \chi$: if $\mathfrak{M}, s_1 \models \varphi$, then $\mathfrak{M}, s_1 \not\models \psi$ or $\mathfrak{M}, s_1 \models \chi$. By the induction hypothesis, $\mathfrak{M}, s_2 \not\models \psi$ or $\mathfrak{M}, s_2 \models \chi$, so $\mathfrak{M}, s_2 \models \varphi$.
5. $\varphi \equiv \psi \leftrightarrow \chi$: if $\mathfrak{M}, s_1 \models \varphi$, then either $\mathfrak{M}, s_1 \models \psi$ and $\mathfrak{M}, s_1 \models \chi$, or $\mathfrak{M}, s_1 \not\models \psi$ and $\mathfrak{M}, s_1 \not\models \chi$. By the induction hypothesis, either $\mathfrak{M}, s_2 \models \psi$ and $\mathfrak{M}, s_2 \models \chi$ or $\mathfrak{M}, s_2 \not\models \psi$ and $\mathfrak{M}, s_2 \not\models \chi$. In either case, $\mathfrak{M}, s_2 \models \varphi$.
6. $\varphi \equiv \exists x \psi$: if $\mathfrak{M}, s_1 \models \varphi$, there is an x -variant s'_1 of s_1 so that $\mathfrak{M}, s'_1 \models \psi$. Let s'_2 be the x -variant of s_2 that assigns the same thing to x as does s'_1 . The free variables of ψ are among x_1, \dots, x_n , and x . $s'_1(x_i) = s'_2(x_i)$, since s'_1 and s'_2 are x -variants of s_1 and s_2 , respectively, and by hypothesis $s_1(x_i) = s_2(x_i)$. $s'_1(x) = s'_2(x)$ by the way we have defined s'_2 . Then the induction hypothesis applies to ψ and s'_1, s'_2 , so $\mathfrak{M}, s'_2 \models \psi$. Hence, there is an x -variant of s_2 that satisfies ψ , and so $\mathfrak{M}, s_2 \models \varphi$.
7. $\varphi \equiv \forall x \psi$: if $\mathfrak{M}, s_1 \models \varphi$, then for every x -variant s'_1 of s_1 , $\mathfrak{M}, s'_1 \models \psi$. 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, \dots, 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 $\mathfrak{M}, s'_2 \models \psi$. Since s'_2 is an arbitrary x -variant of s_2 , every x -variant of s_2 satisfies ψ , and so $\mathfrak{M}, s_2 \models \varphi$.

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

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

6.13. EXTENSIONALITY

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 6.37](#) is satisfied, and we have $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}, s' \models \varphi$. \square

Definition 6.39. If φ is a sentence, we say that a structure \mathfrak{M} *satisfies* φ , $\mathfrak{M} \models \varphi$, iff $\mathfrak{M}, s \models \varphi$ for all variable assignments s .

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

Proposition 6.40. Suppose $\varphi(x)$ only contains x free, and \mathfrak{M} is a structure. Then:

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

Proof. Exercise. \square

6.13 Extensionality

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

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

Proposition 6.41 (Extensionality). Let φ be a formula, and \mathfrak{M}_1 and \mathfrak{M}_2 be structures with $|\mathfrak{M}_1| = |\mathfrak{M}_2|$, and s a variable assignment on $|\mathfrak{M}_1| = |\mathfrak{M}_2|$. If $c^{\mathfrak{M}_1} = c^{\mathfrak{M}_2}$, $R^{\mathfrak{M}_1} = R^{\mathfrak{M}_2}$, and $f^{\mathfrak{M}_1} = f^{\mathfrak{M}_2}$ for every constant symbol c , relation symbol R , and function symbol f occurring in φ , then $\mathfrak{M}_1, s \models \varphi$ iff $\mathfrak{M}_2, s \models \varphi$.

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

Corollary 6.42 (Extensionality). Let φ be a sentence and $\mathfrak{M}_1, \mathfrak{M}_2$ as in [Proposition 6.41](#). Then $\mathfrak{M}_1 \models \varphi$ iff $\mathfrak{M}_2 \models \varphi$.

Proof. Follows from [Proposition 6.41](#) by [Corollary 6.38](#). \square

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

Proposition 6.43. *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 $s'(x) = \text{Val}_s^{\mathfrak{M}}(t')$. Then $\text{Val}_s^{\mathfrak{M}}(t[t'/x]) = \text{Val}_{s'}^{\mathfrak{M}}(t)$.*

Proof. By induction on t .

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

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

□

Proposition 6.44. *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 $s'(x) = \text{Val}_s^{\mathfrak{M}}(t)$. Then $\mathfrak{M}, s \models \varphi[t/x]$ iff $\mathfrak{M}, s' \models \varphi$.*

Proof. Exercise. □

6.14 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.

6.14. SEMANTIC NOTIONS

Definition 6.45 (Validity). A sentence φ is *valid*, $\models \varphi$, iff $\mathfrak{M} \models \varphi$ for every structure \mathfrak{M} .

Definition 6.46 (Entailment). A set of sentences Γ *entails* a sentence φ , $\Gamma \models \varphi$, iff for every structure \mathfrak{M} with $\mathfrak{M} \models \Gamma$, $\mathfrak{M} \models \varphi$.

Definition 6.47 (Satisfiability). A set of sentences Γ is *satisfiable* if $\mathfrak{M} \models \Gamma$ for some structure \mathfrak{M} . If Γ is not satisfiable it is called *unsatisfiable*.

Proposition 6.48. A sentence φ is valid iff $\Gamma \models \varphi$ for every set of sentences Γ .

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

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

Proposition 6.49. $\Gamma \models \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable.

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

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

Proposition 6.50. If $\Gamma \subseteq \Gamma'$ and $\Gamma \models \varphi$, then $\Gamma' \models \varphi$.

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

Theorem 6.51 (Semantic Deduction Theorem). $\Gamma \cup \{\varphi\} \models \psi$ iff $\Gamma \models \varphi \rightarrow \psi$.

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

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

Problems

Problem 6.1. Prove [Lemma 6.9](#).

Problem 6.2. Prove [Proposition 6.10](#) (Hint: Formulate and prove a version of [Lemma 6.9](#) for terms.)

Problem 6.3. Give an inductive definition of the bound variable occurrences along the lines of [Definition 6.16](#).

Problem 6.4. Is \mathfrak{N} , the standard model of arithmetic, covered? Explain.

Problem 6.5. Let $\mathcal{L} = \{c, f, A\}$ with one constant symbol, one one-place function symbol and one two-place predicate symbol, and let the structure \mathfrak{M} be given by

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

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

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

Explain why or why not.

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

Problem 6.6. Complete the proof of [Proposition 6.37](#).

Problem 6.7. Show that if φ is a sentence, $\mathfrak{M} \models \varphi$ iff *there is* a variable assignment s so that $\mathfrak{M}, s \models \varphi$.

Problem 6.8. Prove [Proposition 6.40](#).

Problem 6.9. 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. $\varphi \equiv \perp$: $\mathfrak{M} \not\models \varphi$.
2. $\varphi \equiv \top$: $\mathfrak{M} \models \varphi$.
3. $\varphi \equiv R(d_1, \dots, d_n)$: $\mathfrak{M} \models \varphi$ iff $\langle d_1^{\mathfrak{M}}, \dots, d_n^{\mathfrak{M}} \rangle \in R^{\mathfrak{M}}$.
4. $\varphi \equiv d_1 = d_2$: $\mathfrak{M} \models \varphi$ iff $d_1^{\mathfrak{M}} = d_2^{\mathfrak{M}}$.

6.14. SEMANTIC NOTIONS

5. $\varphi \equiv \neg\psi$: $\mathfrak{M} \models \varphi$ iff not $\mathfrak{M} \models \psi$.
6. $\varphi \equiv (\psi \wedge \chi)$: $\mathfrak{M} \models \varphi$ iff $\mathfrak{M} \models \psi$ and $\mathfrak{M} \models \chi$.
7. $\varphi \equiv (\psi \vee \chi)$: $\mathfrak{M} \models \varphi$ iff $\mathfrak{M} \models \psi$ or $\mathfrak{M} \models \chi$ (or both).
8. $\varphi \equiv (\psi \rightarrow \chi)$: $\mathfrak{M} \models \varphi$ iff not $\mathfrak{M} \models \psi$ or $\mathfrak{M} \models \chi$ (or both).
9. $\varphi \equiv (\psi \leftrightarrow \chi)$: $\mathfrak{M} \models \varphi$ iff either both $\mathfrak{M} \models \psi$ and $\mathfrak{M} \models \chi$, or neither $\mathfrak{M} \models \psi$ nor $\mathfrak{M} \models \chi$.
10. $\varphi \equiv \forall x \psi$: $\mathfrak{M} \models \varphi$ iff for all $a \in |\mathfrak{M}|$, $\mathfrak{M}[a/c] \models \psi[c/x]$, if c does not occur in ψ .
11. $\varphi \equiv \exists x \psi$: $\mathfrak{M} \models \varphi$ 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, \dots, x_n be all free variables in φ , c_1, \dots, c_n constant symbols not in φ , $a_1, \dots, a_n \in |\mathfrak{M}|$, and $s(x_i) = a_i$.

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

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

Problem 6.10. Suppose that f is a function symbol not in $\varphi(x, y)$. Show that there is a structure \mathfrak{M} such that $\mathfrak{M} \models \forall x \exists y \varphi(x, y)$ iff there is an \mathfrak{M}' such that $\mathfrak{M}' \models \forall x \varphi(x, f(x))$.

(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 6.11. Carry out the proof of [Proposition 6.41](#) in detail.

Problem 6.12. Prove [Proposition 6.44](#)

Problem 6.13. 1. Show that $\Gamma \models \perp$ iff Γ is unsatisfiable.

2. Show that $\Gamma \cup \{\varphi\} \models \perp$ iff $\Gamma \models \neg\varphi$.

3. Suppose c does not occur in φ or Γ . Show that $\Gamma \models \forall x \varphi$ iff $\Gamma \models \varphi[c/x]$.

Chapter 7

Theories and Their Models

7.1 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 7.1. A set of sentences Γ is *closed* iff, whenever $\Gamma \models \varphi$ then $\varphi \in \Gamma$. The *closure* of a set of sentences Γ is $\{\varphi : \Gamma \models \varphi\}$.

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.

7.1. INTRODUCTION

1. We can state precisely when an axiom system captures an intended class of structures. That is, if we are interested in a certain class of structures, we will successfully capture that class by an axiom system Γ iff the structures are exactly those \mathfrak{M} such that $\mathfrak{M} \models \Gamma$.
2. We may fail in this respect because there are \mathfrak{M} such that $\mathfrak{M} \models \Gamma$, but \mathfrak{M} is not one of the structures we intend. This may lead us to add axioms which are not true in \mathfrak{M} .
3. If we are successful at least in the respect that Γ is true in all the intended structures, then a sentence φ is true in all intended structures whenever $\Gamma \models \varphi$. 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 $\Gamma \setminus \{\varphi\} \models \varphi$. 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)?

7.2 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 7.2 (Model of a set). Let Γ be a set of sentences in a language \mathcal{L} . We say that a structure \mathfrak{M} is a *model of Γ* if $\mathfrak{M} \models \varphi$ for all $\varphi \in \Gamma$.

Example 7.3. 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

$$\left\{ \begin{array}{l} \forall x x \leq x, \\ \forall x \forall y ((x \leq y \wedge y \leq x) \rightarrow x = y), \\ \forall x \forall y \forall z ((x \leq y \wedge y \leq z) \rightarrow x \leq z) \end{array} \right\}$$

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*.

7.3 Examples of First-Order Theories

Example 7.4. The theory of strict linear orders in the language $\mathcal{L}_{<}$ is axiomatized by the set

$$\left\{ \begin{array}{l} \forall x \neg x < x, \\ \forall x \forall y ((x < y \vee y < x) \vee x = y), \\ \forall x \forall y \forall z ((x < y \wedge y < z) \rightarrow x < z) \end{array} \right\}$$

7.3. EXAMPLES OF FIRST-ORDER THEORIES

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 7.5. The theory of groups in the language \mathcal{L}_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 7.6. The theory of Peano arithmetic is axiomatized by the following sentences in the language of arithmetic \mathcal{L}_A .

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

plus all sentences of the form

$$(\varphi(0) \wedge \forall x (\varphi(x) \rightarrow \varphi(x'))) \rightarrow \forall x \varphi(x)$$

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

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

Example 7.7. 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:

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

plus all sentences of the form

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

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

The sentences mentioned last are collectively called the *naïve comprehension scheme*. It essentially says that for every $\varphi(x)$, the set $\{x : \varphi(x)\}$ exists—so at first glance a true, useful, and perhaps even necessary axiom. It is called “naïve” 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 7.8. 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^M must satisfy some conditions, which we can lay down as axioms for a theory of parthood. For instance, parthood is a partial order on objects: every object is a part (albeit an *improper* part) of itself; no two different objects can be parts of each other; a part of a part of an object is itself part of that object. Note that in this sense “is a part of” resembles “is a subset of,” but does not resemble “is an element of” which is neither reflexive nor transitive.

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

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

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

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

7.4 Expressing Relations in a Structure

One main use formulas can be put to is to express properties and relations in a structure \mathfrak{M} in terms of the primitives of the language \mathcal{L} of \mathfrak{M} . By this we mean the following: the domain of \mathfrak{M} is a set of objects. The constant symbols, function symbols, and predicate symbols are interpreted in \mathfrak{M} by some objects in $|\mathfrak{M}|$, functions on $|\mathfrak{M}|$, and relations on $|\mathfrak{M}|$. For instance, if A_0^2 is in \mathcal{L} , then \mathfrak{M} assigns to it a relation $R = A_0^{2\mathfrak{M}}$. Then the formula $A_0^2(v_1, v_2)$ expresses that very relation, in the following sense: if a variable assignment s maps v_1 to $a \in |\mathfrak{M}|$ and v_2 to $b \in |\mathfrak{M}|$, then

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

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

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

Definition 7.9. Let $\varphi(v_1, \dots, v_n)$ be a formula of \mathcal{L} in which only v_1, \dots, v_n occur free, and let \mathfrak{M} be a structure for \mathcal{L} . $\varphi(v_1, \dots, v_n)$ expresses the relation $R \subseteq |\mathfrak{M}|^n$ iff

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

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

Example 7.10. 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.

7.5 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

$$\forall z (z \in x \rightarrow z \in y)$$

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

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

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.

7.5. THE THEORY OF SETS

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

$$\begin{aligned}\forall u ((u \in x \vee u \in y) \leftrightarrow u \in z) \\ \forall u (u \subseteq x \leftrightarrow u \in y)\end{aligned}$$

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

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

where $\text{maps}(f, x, y)$ 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 $f: X \rightarrow Y$ is injective, for instance:

$$\begin{aligned}f: X \rightarrow Y \wedge \forall x \forall x' ((x \in X \wedge x' \in X \wedge \\ \exists y (\text{maps}(f, x, y) \wedge \text{maps}(f, x', y))) \rightarrow x = x')\end{aligned}$$

A function $f: X \rightarrow Y$ is injective iff, whenever f maps $x, x' \in X$ to a single y , $x = x'$. If we abbreviate this formula as $\text{inj}(f, X, Y)$, 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 :

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

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:

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

7.6 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 7.11. *The sentence*

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

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

7.6. EXPRESSING THE SIZE OF STRUCTURES

Proposition 7.12. *The sentence*

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

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

Proposition 7.13. *A structure is infinite iff it is a model of*

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

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

Problems

Problem 7.1. 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 7.2. 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 7.3. 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 $<^{\mathfrak{N}} = \{(n, m) : n < m\}$. 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 7.4. Show that the comprehension principle is inconsistent by giving a derivation that shows

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

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

Chapter 8

The Sequent Calculus

This chapter presents a sequent calculus (using sets of formulas instead of sequences) for first-order logic. It will mainly be useful for instructors transitioning from Boolos, Burgess & Jeffrey, who use the same system. For a self-contained introduction to the sequent calculus, it would probably be best to use a standard presentation including structural rules.

To include or exclude material relevant to the sequent calculus as a proof system, use the “prfLK” tag.

8.1 Rules and Derivations

This section collects all the rules propositional connectives and quantifiers, but not for identity. It is planned to divide this into separate sections on connectives and quantifiers so that proofs for propositional logic can be treated separately (issue #77).

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

Definition 8.1 (sequent). A *sequent* is an expression of the form

$$\Gamma \Rightarrow \Delta$$

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

The intuitive idea behind a sequent is: if all of the antecedent formulas hold, then at least one of the succedent formulas holds. That is, if $\Gamma =$

$\{\Gamma_1, \dots, \Gamma_m\}$ and $\Delta = \{\Delta_1, \dots, \Delta_n\}$, then $\Gamma \Rightarrow \Delta$ holds iff

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

holds.

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

An empty succedent is sometimes filled with the \perp symbol. The empty sequent \Rightarrow canonically represents a contradiction.

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

Definition 8.2 (Inference). An *inference* is an expression of the form

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

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

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

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

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

Structural rules: Weakening:

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

where φ is called the *weakening formula*.

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

Cut:

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

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

Propositional Rules:

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

8.1. RULES AND DERIVATIONS

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

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

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

Quantifier Rules:

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

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

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

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

The condition that an eigenvariable not occur in the upper sequent of the $\forall R$ or $\exists L$ inference is called the *eigenvariable condition*.

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

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

Definition 8.3 (Initial Sequent). An *initial sequent* is a sequent of one of the following forms:

1. $\varphi \Rightarrow \varphi$
2. $\Rightarrow \top$
3. $\perp \Rightarrow$

for any sentence φ in the language.

Definition 8.4 (LK derivation). An **LK-derivation** of a sequent S is a tree of sequents satisfying the following conditions:

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

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

Definition 8.5 (LK theorem). A sentence φ is a *theorem* of **LK** if the sequent $\Rightarrow \varphi$ is **LK**-derivable.

8.2 Examples of Derivations

Example 8.6. Give an **LK**-derivation for the sequent $\varphi \wedge \psi \Rightarrow \varphi$.

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

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

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

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

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

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

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

Example 8.7. Give an **LK**-derivation for the sequent $\neg\varphi \vee \psi \Rightarrow \varphi \rightarrow \psi$.

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

$$\frac{}{\neg\varphi \vee \psi \Rightarrow \varphi \rightarrow \psi}$$

To find a logical rule that could give us this end-sequent, we look at the logical connectives in the end-sequent: \neg , \vee , and \rightarrow . We only care at the moment about \vee and \rightarrow because they are main operators of sentences in the end-sequent, while \neg is inside the scope of another connective, so we will take care of it later. Our options for logical rules for the final inference are therefore the

8.2. EXAMPLES OF DERIVATIONS

\vee L rule and the \rightarrow R rule. We could pick either rule, really, but let's pick the \rightarrow R rule (if for no reason other than it allows us to put off splitting into two branches). According to the form of \rightarrow R inferences which can yield the lower sequent, this must look like:

$$\frac{\overline{\varphi, \neg\varphi \vee \psi \Rightarrow \psi}}{\neg\varphi \vee \psi \Rightarrow \varphi \rightarrow \psi} \rightarrow R$$

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

$$\frac{\frac{\overline{\varphi, \neg\varphi \Rightarrow \psi} \quad \overline{\varphi, \psi \Rightarrow \psi}}{\varphi, \neg\varphi \vee \psi \Rightarrow \psi} \vee L}{\neg\varphi \vee \psi \Rightarrow \varphi \rightarrow \psi} \rightarrow R$$

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

$$\frac{\frac{\overline{\varphi, \neg\varphi \Rightarrow \psi} \quad \frac{\psi \Rightarrow \psi}{\varphi, \psi \Rightarrow \psi} WL}{\varphi, \neg\varphi \vee \psi \Rightarrow \psi} \vee L}{\neg\varphi \vee \psi \Rightarrow \varphi \rightarrow \psi} \rightarrow R$$

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

$$\frac{\frac{\overline{\varphi \Rightarrow \psi, \varphi}}{\varphi, \neg\varphi \Rightarrow \psi} \neg L \quad \frac{\psi \Rightarrow \psi}{\varphi, \psi \Rightarrow \psi} WL}{\varphi, \neg\varphi \vee \psi \Rightarrow \psi} \vee L}{\neg\varphi \vee \psi \Rightarrow \varphi \rightarrow \psi} \rightarrow R$$

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

$$\frac{\frac{\frac{\varphi \Rightarrow \varphi}{\varphi \Rightarrow \psi, \varphi} WR}{\varphi, \neg\varphi \Rightarrow \psi} \neg L \quad \frac{\psi \Rightarrow \psi}{\varphi, \psi \Rightarrow \psi} WL}{\varphi, \neg\varphi \vee \psi \Rightarrow \psi} \vee L}{\neg\varphi \vee \psi \Rightarrow \varphi \rightarrow \psi} \rightarrow R$$

Example 8.8. Give an **LK**-derivation of the sequent $\neg\varphi \vee \neg\psi \Rightarrow \neg(\varphi \wedge \psi)$

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

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

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

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

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

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

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

$$\frac{\frac{\frac{}{\varphi \Rightarrow \varphi}}{\varphi, \neg\varphi \Rightarrow} \neg L \quad \frac{\frac{}{\varphi \Rightarrow \psi} ?}{\varphi, \neg\psi \Rightarrow} \neg L}{\frac{\frac{}{\varphi, \neg\varphi \vee \neg\psi \Rightarrow}}{\varphi \wedge \psi, \neg\varphi \vee \neg\psi \Rightarrow} \wedge L} \vee L \neg R$$

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

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

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

$$\frac{\frac{\frac{}{\varphi \Rightarrow \varphi}}{\varphi \wedge \psi \Rightarrow \varphi} \wedge L \quad \frac{\frac{}{\psi \Rightarrow \psi}}{\varphi \wedge \psi \Rightarrow \psi} \wedge L}{\frac{\frac{}{\varphi \wedge \psi, \neg\varphi \Rightarrow} \neg L \quad \frac{\frac{}{\varphi \wedge \psi, \neg\psi \Rightarrow} \neg L}{\varphi \wedge \psi, \neg\varphi \vee \neg\psi \Rightarrow} \vee L} \neg R$$

8.2. EXAMPLES OF DERIVATIONS

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

Example 8.9. Give an LK-derivation of the sequent $\exists x \neg \varphi(x) \Rightarrow \neg \forall x \varphi(x)$.

When dealing with quantifiers, we have to make sure not to violate the eigenvariable condition, and sometimes this requires us to play around with the order of carrying out certain inferences. In general, it helps to try and take care of rules subject to the eigenvariable condition first (they will be lower down in the finished proof). Also, it is a good idea to try and look ahead and try to guess what the initial sequent might look like. In our case, it will have to be something like $\varphi(a) \Rightarrow \varphi(a)$. That means that when we are “reversing” the quantifier rules, we will have to pick the same term—what we will call a —for both the \forall and the \exists rule. If we picked different terms for each rule, we would end up with something like $\varphi(a) \Rightarrow \varphi(b)$, which, of course, is not derivable.

Starting as usual, we write

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

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

$$\frac{\frac{}{\neg \varphi(a) \Rightarrow \neg \forall x \varphi(x)}}{\exists x \neg \varphi(x) \Rightarrow \neg \forall x \varphi(x)} \exists L$$

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

$$\frac{\frac{\frac{}{\forall x \varphi(x) \Rightarrow \varphi(a)}}{\vdots} \neg \varphi(a) \Rightarrow \neg \forall x \varphi(x)}{\exists x \neg \varphi(x) \Rightarrow \neg \forall x \varphi(x)} \exists L$$

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

$$\frac{\frac{\frac{}{\varphi(a) \Rightarrow \varphi(a)}}{\forall x \varphi(x) \Rightarrow \varphi(a)} \forall L}{\frac{\vdots}{\neg \varphi(a) \Rightarrow \neg \forall x \varphi(x)} \exists L} \exists L$$

It is important, especially when dealing with quantifiers, to double check at this point that the eigenvariable condition has not been violated. Since the only rule we applied that is subject to the eigenvariable condition was $\exists L$, and the eigenvariable a does not occur in its lower sequent (the end-sequent), this is a correct derivation.

This section collects the properties of the provability relation required for the completeness theorem. If you find the location unmotivated, include it instead in the chapter on completeness.

8.3 Proof-Theoretic Notions

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

Definition 8.10 (Theorems). A sentence φ is a *theorem* if there is a derivation in **LK** of the sequent $\Rightarrow \varphi$. We write $\vdash_{\mathbf{LK}} \varphi$ if φ is a theorem and $\nvdash_{\mathbf{LK}} \varphi$ if it is not.

Definition 8.11 (Derivability). A sentence φ is *derivable from* a set of sentences Γ , $\Gamma \vdash_{\mathbf{LK}} \varphi$, if there is a finite subset $\Gamma_0 \subseteq \Gamma$ such that **LK** derives $\Gamma_0 \Rightarrow \varphi$. If φ is not derivable from Γ we write $\Gamma \nvdash_{\mathbf{LK}} \varphi$.

Definition 8.12 (Consistency). A set of sentences Γ is *consistent* iff $\Gamma \nvdash_{\mathbf{LK}} \perp$. If Γ is not consistent, i.e., if $\Gamma \vdash_{\mathbf{LK}} \perp$, we say it is *inconsistent*.

Proposition 8.13. $\Gamma \vdash_{\mathbf{LK}} \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is inconsistent.

Proof. Exercise. □

Proposition 8.14. Γ is inconsistent iff $\Gamma \vdash_{\mathbf{LK}} \varphi$ for every sentence φ .

Proof. Exercise. □

Proposition 8.15. $\Gamma \vdash_{\mathbf{LK}} \varphi$ iff for some finite $\Gamma_0 \subseteq \Gamma$, $\Gamma_0 \vdash_{\mathbf{LK}} \varphi$.

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

8.4. PROPERTIES OF DERIVABILITY

8.4 Properties of Derivability

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

Proposition 8.16 (Monotony). *If $\Gamma \subseteq \Delta$ and $\Gamma \vdash_{\mathbf{LK}} \varphi$, then $\Delta \vdash_{\mathbf{LK}} \varphi$.*

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

Proposition 8.17. *If $\Gamma \vdash_{\mathbf{LK}} \varphi$ and $\Gamma \cup \{\varphi\} \vdash_{\mathbf{LK}} \perp$, then Γ is inconsistent.*

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

$$\frac{\begin{array}{c} \vdots \\ \vdots \Pi_0 \\ \vdots \\ \Gamma_0 \Rightarrow \varphi \end{array} \quad \begin{array}{c} \vdots \\ \vdots \Pi_1 \\ \vdots \\ \Gamma_1, \varphi \Rightarrow \perp \end{array}}{\Gamma_0, \Gamma_1 \Rightarrow \perp} \text{ cut}$$

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

Proposition 8.18. *If $\Gamma \cup \{\varphi\} \vdash_{\mathbf{LK}} \perp$, then $\Gamma \vdash_{\mathbf{LK}} \neg\varphi$.*

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

$$\frac{\begin{array}{c} \vdots \\ \vdots \Pi_0 \\ \vdots \\ \Gamma_0, \varphi \Rightarrow \perp \end{array}}{\Gamma_0 \Rightarrow \perp, \neg\varphi} \neg\mathbf{R} \quad \frac{\Gamma_0 \Rightarrow \perp, \neg\varphi \quad \perp \Rightarrow}{\Gamma_0 \Rightarrow \neg\varphi} \text{ cut}$$

\square

Proposition 8.19. *If $\Gamma \cup \{\varphi\} \vdash_{\mathbf{LK}} \perp$ and $\Gamma \cup \{\neg\varphi\} \vdash_{\mathbf{LK}} \perp$, then $\Gamma \vdash_{\mathbf{LK}} \perp$.*

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

$$\frac{\begin{array}{c} \vdots \\ \vdots \Pi_0 \\ \vdots \\ \Gamma_0, \varphi \Rightarrow \perp \end{array} \quad \frac{\Gamma_0 \Rightarrow \perp, \neg\varphi}{\Gamma_0 \Rightarrow \perp, \neg\varphi} \neg\mathbf{R} \quad \begin{array}{c} \vdots \\ \vdots \Pi_1 \\ \vdots \\ \Gamma_1, \neg\varphi \Rightarrow \perp \end{array}}{\Gamma_0, \Gamma_1 \Rightarrow \perp} \text{ cut}$$

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

Proposition 8.20. *If $\Gamma \cup \{\varphi\} \vdash_{\mathbf{LK}} \perp$ and $\Gamma \cup \{\psi\} \vdash_{\mathbf{LK}} \perp$, then $\Gamma \cup \{\varphi \vee \psi\} \vdash_{\mathbf{LK}} \perp$.*

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

$$\frac{\frac{\frac{\vdots \Pi_0}{\Gamma_0, \varphi \Rightarrow \perp}}{\Gamma_0, \Gamma_1, \varphi \Rightarrow \perp} \quad \frac{\frac{\vdots \Pi_1}{\Gamma_1, \psi \Rightarrow \perp}}{\Gamma_0, \Gamma_1, \psi \Rightarrow \perp}}{\Gamma_0, \Gamma_1, \varphi \vee \psi \Rightarrow \perp} \vee L$$

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

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

Proposition 8.21. *If $\Gamma \vdash_{\mathbf{LK}} \varphi$ or $\Gamma \vdash_{\mathbf{LK}} \psi$, then $\Gamma \vdash_{\mathbf{LK}} \varphi \vee \psi$.*

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

$$\frac{\frac{\vdots \Pi_0}{\Gamma_0 \Rightarrow \varphi}}{\Gamma_0 \Rightarrow \varphi \vee \psi} \vee R$$

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

Proposition 8.22. *If $\Gamma \vdash_{\mathbf{LK}} \varphi \wedge \psi$ then $\Gamma \vdash_{\mathbf{LK}} \varphi$ and $\Gamma \vdash_{\mathbf{LK}} \psi$.*

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

$$\frac{\frac{\vdots \Pi_0}{\Gamma_0 \Rightarrow \varphi \wedge \psi} \quad \frac{\varphi \Rightarrow \varphi}{\varphi \wedge \psi \Rightarrow \varphi} \wedge L}{\Gamma_0 \Rightarrow \varphi} \text{cut}$$

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

Proposition 8.23. *If $\Gamma \vdash_{\mathbf{LK}} \varphi$ and $\Gamma \vdash_{\mathbf{LK}} \psi$, then $\Gamma \vdash_{\mathbf{LK}} \varphi \wedge \psi$.*

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

8.4. PROPERTIES OF DERIVABILITY

$$\frac{\frac{\frac{\vdots \Pi_0}{\Gamma_0 \Rightarrow \varphi}}{\Gamma_0, \Gamma_1 \Rightarrow \varphi} \quad \frac{\frac{\vdots \Pi_1}{\Gamma_1 \Rightarrow \psi}}{\Gamma_0, \Gamma_1 \Rightarrow \psi}}{\Gamma_0, \Gamma_1 \Rightarrow \varphi \wedge \psi} \wedge R$$

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

Proposition 8.24. *If $\Gamma \vdash_{\mathbf{LK}} \varphi$ and $\Gamma \vdash_{\mathbf{LK}} \varphi \rightarrow \psi$, then $\Gamma \vdash_{\mathbf{LK}} \psi$.*

Proof. Suppose that $\Gamma \vdash_{\mathbf{LK}} \varphi$ and $\Gamma \vdash_{\mathbf{LK}} \varphi \rightarrow \psi$. There are finite sets $\Gamma_0, \Gamma_1 \subseteq \Gamma$ such that there are \mathbf{LK} -derivations Π_0 of $\Gamma_0 \Rightarrow \varphi$ and Π_1 of $\Gamma_1 \Rightarrow \varphi \rightarrow \psi$. Consider:

$$\frac{\frac{\frac{\vdots \Pi_0}{\Gamma_1 \Rightarrow \varphi \rightarrow \psi} \quad \frac{\frac{\frac{\vdots \Pi_1}{\Gamma_0 \Rightarrow \varphi} \quad \psi \Rightarrow \psi}{\Gamma_0, \varphi \rightarrow \psi \Rightarrow \psi} \rightarrow L}{\Gamma_0, \Gamma_1 \Rightarrow \varphi \rightarrow \psi} \text{cut}}{\Gamma_0, \Gamma_1 \Rightarrow \psi}$$

Since $\Gamma_0 \cup \Gamma_1 \subseteq \Gamma$, this means that $\Gamma \vdash_{\mathbf{LK}} \psi$. \square

Proposition 8.25. *If $\Gamma \vdash_{\mathbf{LK}} \neg \varphi$ or $\Gamma \vdash_{\mathbf{LK}} \psi$, then $\Gamma \vdash_{\mathbf{LK}} \varphi \rightarrow \psi$.*

Proof. First suppose $\Gamma \vdash_{\mathbf{LK}} \neg \varphi$. Then for some finite $\Gamma_0 \subseteq \Gamma$ there is a \mathbf{LK} -derivation of $\Gamma_0 \Rightarrow \neg \varphi$. The following derivation shows that $\Gamma \vdash_{\mathbf{LK}} \varphi \rightarrow \psi$:

$$\frac{\frac{\frac{\vdots \Pi_0}{\Gamma_0 \Rightarrow \neg \varphi} \quad \frac{\frac{\frac{\varphi \Rightarrow \varphi}{\neg \varphi, \varphi \Rightarrow} \neg R}{\varphi, \neg \varphi \Rightarrow \psi} \text{WR}}{\neg \varphi \Rightarrow \varphi \rightarrow \psi} \rightarrow R}{\Gamma_0 \Rightarrow \varphi \rightarrow \psi} \text{cut}$$

Now suppose $\Gamma \vdash_{\mathbf{LK}} \psi$. Then for some finite $\Gamma_0 \subseteq \Gamma$ there is a \mathbf{LK} -derivation of $\Gamma_0 \Rightarrow \psi$. The following derivation shows that $\Gamma \vdash_{\mathbf{LK}} \varphi \rightarrow \psi$:

$$\frac{\frac{\frac{\vdots \Pi_0}{\Gamma_0 \Rightarrow \psi} \quad \frac{\frac{\psi \Rightarrow \psi}{\varphi, \psi \Rightarrow \psi} \text{WL}}{\psi \Rightarrow \varphi \rightarrow \psi} \rightarrow R}{\Gamma_0 \Rightarrow \varphi \rightarrow \psi} \text{cut}$$

\square

Theorem 8.26. *If c is a constant not occurring in Γ or $\varphi(x)$ and $\Gamma \vdash_{\mathbf{LK}} \varphi(c)$, then $\Gamma \vdash_{\mathbf{LK}} \forall x \varphi(x)$.*

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

Theorem 8.27. 1. If $\Gamma \vdash_{\mathbf{LK}} \varphi(t)$ then $\Gamma \vdash \exists x \varphi(x)$.

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

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

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

$$\frac{\Gamma_0 \Rightarrow \forall x \varphi(x) \quad \frac{\varphi(t) \Rightarrow \varphi(t)}{\forall x \varphi(x) \Rightarrow \varphi(t)} \forall\text{L}}{\Gamma_0 \Rightarrow \varphi(t)} \text{cut}$$

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

8.5 Soundness

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

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

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

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

8.5. SOUNDNESS

Definition 8.28. A structure \mathfrak{M} *satisfies* a sequent $\Gamma \Rightarrow \Delta$ iff either $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma$ or $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta$.

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

Theorem 8.29 (Soundness). *If LK derives $\Gamma \Rightarrow \Delta$, then $\Gamma \Rightarrow \Delta$ is valid.*

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

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

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

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

1. The last inference is a weakening. Then $\Gamma' \subseteq \Gamma$ and $\Delta = \Delta'$ if it's a weakening on the left, or $\Gamma = \Gamma'$ and $\Delta' \subseteq \Delta$ if it's a weakening on the right. In either case, $\Delta' \subseteq \Delta$ and $\Gamma' \subseteq \Gamma$. If $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma'$, then, since $\Gamma' \subseteq \Gamma$, $\alpha \in \Gamma$ as well, and so $\mathfrak{M} \not\models \alpha$ for the same $\alpha \in \Gamma$. Similarly, if $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$, as $\alpha \in \Delta$, $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta$. Since $\Gamma' \Rightarrow \Delta'$ is valid, one of these cases obtains for every \mathfrak{M} . Consequently, $\Gamma \Rightarrow \Delta$ is valid.
2. The last inference is \neg L: Then for some $\varphi \in \Delta'$, $\neg\varphi \in \Gamma$. Also, $\Gamma' \subseteq \Gamma$, and $\Delta' \setminus \{\varphi\} \subseteq \Delta$.
If $\mathfrak{M} \models \varphi$, then $\mathfrak{M} \not\models \neg\varphi$, and since $\neg\varphi \in \Gamma$, \mathfrak{M} satisfies $\Gamma \Rightarrow \Delta$. Since $\Gamma' \Rightarrow \Delta'$ is valid, if $\mathfrak{M} \not\models \varphi$, then either $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma'$ or $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$ different from φ . Consequently, $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma$ (since $\Gamma' \subseteq \Gamma$) or $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$ different from φ (since $\Delta' \setminus \{\varphi\} \subseteq \Delta$).
3. The last inference is \neg R: Exercise.
4. The last inference is \wedge L: There are two variants: $\varphi \wedge \psi$ may be inferred on the left from φ or from ψ on the left side of the premise. In the first case, $\varphi \in \Gamma'$. Consider a structure \mathfrak{M} . Since $\Gamma' \Rightarrow \Delta'$ is valid, (a) $\mathfrak{M} \not\models \varphi$, (b) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma' \setminus \{\varphi\}$, or (c) $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$. In case (a), $\mathfrak{M} \not\models \varphi \wedge \psi$. In case (b), there is an $\alpha \in \Gamma' \setminus \{\varphi\}$ such that $\mathfrak{M} \not\models \alpha$, since $\Gamma' \setminus \{\varphi\} \subseteq \Gamma \setminus \{\varphi \wedge \psi\}$. In case (c), there is a $\alpha \in \Delta$ such that $\mathfrak{M} \models \alpha$, as $\Delta = \Delta'$. So in each case, \mathfrak{M} satisfies $\varphi \wedge \psi, \Gamma \Rightarrow \Delta$. Since \mathfrak{M} was arbitrary, $\Gamma \Rightarrow \Delta$ is valid. The case where $\varphi \wedge \psi$ is inferred from ψ is handled the same, changing φ to ψ .

5. The last inference is $\vee R$: There are two variants: $\varphi \vee \psi$ may be inferred on the right from φ or from ψ on the right side of the premise. In the first case, $\varphi \in \Delta'$. Consider a structure \mathfrak{M} . Since $\Gamma' \Rightarrow \Delta'$ is valid, (a) $\mathfrak{M} \models \varphi$, (b) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma'$, or (c) $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta' \setminus \{\varphi\}$. In case (a), $\mathfrak{M} \models \varphi \vee \psi$. In case (b), there is $\alpha \in \Gamma$ such that $\mathfrak{M} \not\models \alpha$, as $\Gamma = \Gamma'$. In case (c), there is an $\alpha \in \Delta$ such that $\mathfrak{M} \models \alpha$, since $\Delta' \setminus \{\varphi\} \subseteq \Delta$. So in each case, \mathfrak{M} satisfies $\varphi \wedge \psi, \Gamma \Rightarrow \Delta$. Since \mathfrak{M} was arbitrary, $\Gamma \Rightarrow \Delta$ is valid. The case where $\varphi \vee \psi$ is inferred from ψ is handled the same, changing φ to ψ .
6. The last inference is $\rightarrow R$: Then $\varphi \in \Gamma', \psi \in \Delta', \Gamma' \setminus \{\varphi\} \subseteq \Gamma$ and $\Delta' \setminus \{\psi\} \subseteq \Delta$. Since $\Gamma' \Rightarrow \Delta'$ is valid, for any structure \mathfrak{M} , (a) $\mathfrak{M} \not\models \varphi$, (b) $\mathfrak{M} \models \psi$, (c) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma' \setminus \{\varphi\}$, or $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta' \setminus \{\psi\}$. In cases (a) and (b), $\mathfrak{M} \models \varphi \rightarrow \psi$. In case (c), for some $\alpha \in \Gamma$, $\mathfrak{M} \not\models \alpha$. In case (d), for some $\alpha \in \Delta$, $\mathfrak{M} \models \alpha$. In each case, \mathfrak{M} satisfies $\Gamma \Rightarrow \Delta$. Since \mathfrak{M} was arbitrary, $\Gamma \Rightarrow \Delta$ is valid.
7. The last inference is $\forall L$: Then there is a formula $\varphi(x)$ and a ground term t such that $\varphi(t) \in \Gamma', \forall x \varphi(x) \in \Gamma$, and $\Gamma' \setminus \{\varphi(t)\} \subseteq \Gamma$. Consider a structure \mathfrak{M} . Since $\Gamma' \Rightarrow \Delta'$ is valid, (a) $\mathfrak{M} \not\models \varphi(t)$, (b) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma' \setminus \{\varphi(t)\}$, or (c) $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta'$. In case (a), $\mathfrak{M} \not\models \forall x \varphi(x)$. In case (b), there is an $\alpha \in \Gamma \setminus \{\varphi(t)\}$ such that $\mathfrak{M} \not\models \alpha$. In case (c), there is a $\alpha \in \Delta$ such that $\mathfrak{M} \models \alpha$, as $\Delta = \Delta'$. So in each case, \mathfrak{M} satisfies $\Gamma \Rightarrow \Delta$. Since \mathfrak{M} was arbitrary, $\Gamma \Rightarrow \Delta$ is valid.
8. The last inference is $\exists R$: Exercise.
9. The last inference is $\forall R$: Then there is a formula $\varphi(x)$ and a constant symbol a such that $\varphi(a) \in \Delta', \forall x \varphi(x) \in \Delta$, and $\Delta' \setminus \{\varphi(a)\} \subseteq \Delta$. Furthermore, $a \notin \Gamma \cup \Delta$. Consider a structure \mathfrak{M} . Since $\Gamma' \Rightarrow \Delta'$ is valid, (a) $\mathfrak{M} \models \varphi(a)$, (b) $\mathfrak{M} \not\models \alpha$ for some $\alpha \in \Gamma'$, or (c) $\mathfrak{M} \models \alpha$ for some $\alpha \in \Delta' \setminus \{\varphi(a)\}$.

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

We now show that $\mathfrak{M} \models \forall x \varphi(x)$. To do this, we have to show that for every variable assignment s , $\mathfrak{M}, s \models \forall x \varphi(x)$. This in turn means that for every x -variant s' of s , we must have $\mathfrak{M}, s' \models \varphi(x)$. So consider any variable assignment s and let s' be an x -variant of s . Since Γ' and Δ' consist entirely of sentences, $\mathfrak{M}, s \models \alpha$ iff $\mathfrak{M}, s' \models \alpha$ iff $\mathfrak{M} \models \alpha$ for all $\alpha \in \Gamma' \cup \Delta'$. Let \mathfrak{M}' be like \mathfrak{M} except that $a^{\mathfrak{M}'} = s'(x)$. Then $\mathfrak{M}, s' \models$

8.5. SOUNDNESS

$\varphi(x)$ iff $\mathfrak{M}' \models \varphi(a)$ (as $\varphi(x)$ does not contain a). Since we've already established that $\mathfrak{M}' \models \varphi(a)$ for all \mathfrak{M}' which differ from \mathfrak{M} at most in what they assign to a , this means that $\mathfrak{M}, s' \models \varphi(x)$. Thus we've shown that $\mathfrak{M}, s \models \forall x \varphi(x)$. Since s is an arbitrary variable assignment and $\forall x \varphi(x)$ is a sentence, then $\mathfrak{M} \models \forall x \varphi(x)$.

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

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

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

1. The last inference is a cut: Suppose the premises are $\Gamma' \Rightarrow \Delta'$ and $\Pi' \Rightarrow \Delta'$ and the cut formula φ is in both Δ' and Π' . Since each is valid, every structure \mathfrak{M} satisfies both premises. We distinguish two cases: (a) $\mathfrak{M} \not\models \varphi$ and (b) $\mathfrak{M} \models \varphi$. In case (a), in order for \mathfrak{M} to satisfy the left premise, it must satisfy $\Gamma' \Rightarrow \Delta' \setminus \{\varphi\}$. But $\Gamma' \subseteq \Gamma$ and $\Delta' \setminus \{\varphi\} \subseteq \Delta$, so \mathfrak{M} also satisfies $\Gamma \Rightarrow \Delta$. In case (b), in order for \mathfrak{M} to satisfy the right premise, it must satisfy $\Pi' \setminus \{\varphi\} \Rightarrow \Delta'$. But $\Pi' \setminus \{\varphi\} \subseteq \Gamma$ and $\Delta' \subseteq \Delta$, so \mathfrak{M} also satisfies $\Gamma \Rightarrow \Delta$.
2. The last inference is \wedge R. The premises are $\Gamma \Rightarrow \Delta'$ and $\Gamma \Rightarrow \Delta''$, where $\varphi \in \Delta'$ and $\psi \in \Delta''$. By induction hypothesis, both are valid. Consider a structure \mathfrak{M} . We have two cases: (a) $\mathfrak{M} \not\models \varphi \wedge \psi$ or (b) $\mathfrak{M} \models \varphi \wedge \psi$. In case (a), either $\mathfrak{M} \not\models \varphi$ or $\mathfrak{M} \not\models \psi$. In the former case, in order for \mathfrak{M} to satisfy $\Gamma \Rightarrow \Delta'$, it must already satisfy $\Gamma \Rightarrow \Delta' \setminus \{\varphi\}$. In the latter case, it must satisfy $\Gamma \Rightarrow \Delta'' \setminus \{\psi\}$. But since both $\Delta' \setminus \{\varphi\} \subseteq \Delta$ and $\Delta'' \setminus \{\psi\} \subseteq \Delta$, that means \mathfrak{M} satisfies $\Gamma \Rightarrow \Delta$. In case (b), \mathfrak{M} satisfies $\Gamma \Rightarrow \Delta$ since $\varphi \wedge \psi \in \Delta$.
3. The last inference is \vee L: Exercise.
4. The last inference is \rightarrow L. The premises are $\Gamma \Rightarrow \Delta'$ and $\Gamma' \Rightarrow \Delta$, where $\varphi \in \Delta'$ and $\psi \in \Gamma'$. By induction hypothesis, both are valid. Consider a structure \mathfrak{M} . We have two cases: (a) $\mathfrak{M} \models \varphi \rightarrow \psi$ or (b) $\mathfrak{M} \not\models \varphi \rightarrow \psi$. In case (a), either $\mathfrak{M} \not\models \varphi$ or $\mathfrak{M} \models \psi$. In the former case, in order for \mathfrak{M} to satisfy $\Gamma \Rightarrow \Delta'$, it must already satisfy $\Gamma \Rightarrow \Delta' \setminus \{\varphi\}$. In the latter case, it must satisfy $\Gamma' \setminus \{\psi\} \Rightarrow \Delta$. But since both $\Delta' \setminus \{\varphi\} \subseteq \Delta$ and $\Gamma' \setminus \{\psi\} \subseteq \Gamma$, that means \mathfrak{M} satisfies $\Gamma \Rightarrow \Delta$. In case (b), \mathfrak{M} satisfies $\Gamma \Rightarrow \Delta$ since $\varphi \rightarrow \psi \in \Gamma$.

□

Corollary 8.30. *If $\vdash_{\text{LK}} \varphi$ then φ is valid.*

Corollary 8.31. *If $\Gamma \vdash_{\text{LK}} \varphi$ then $\Gamma \models \varphi$.*

Proof. If $\Gamma \vdash_{\mathbf{LK}} \varphi$ then for some finite subset $\Gamma_0 \subseteq \Gamma$, there is a derivation of $\Gamma_0 \Rightarrow \varphi$. By [Theorem 8.29](#), every structure \mathfrak{M} either makes some $\psi \in \Gamma_0$ false or makes φ true. Hence, if $\mathfrak{M} \models \Gamma$ then also $\mathfrak{M} \models \varphi$. \square

Corollary 8.32. *If Γ is satisfiable, then it is consistent.*

Proof. We prove the contrapositive. Suppose that Γ is not consistent. Then $\Gamma \vdash_{\mathbf{LK}} \perp$, i.e., there is a finite $\Gamma_0 \subseteq \Gamma$ and a derivation of $\Gamma_0 \Rightarrow \perp$. By [Theorem 8.29](#), $\Gamma_0 \Rightarrow \perp$ is valid. Since $\mathfrak{M} \not\models \perp$ for every structure \mathfrak{M} , for \mathfrak{M} to satisfy $\Gamma_0 \Rightarrow \perp$ there must be an $\alpha \in \Gamma_0$ so that $\mathfrak{M} \not\models \alpha$, and since $\Gamma_0 \subseteq \Gamma$, that α is also in Γ . In other words, no \mathfrak{M} satisfies Γ , i.e., Γ is not satisfiable. \square

8.6 Derivations with Identity predicate

Derivations with the identity predicate require additional inference rules.

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

Rules for $=$:

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

where t_1 and t_2 are closed terms.

Example 8.33. If s and t are ground terms, then $\varphi(s), s = t \vdash_{\mathbf{LK}} \varphi(t)$:

$$\frac{\frac{\varphi(s) \Rightarrow \varphi(s)}{\varphi(s), s = t \Rightarrow \varphi(s)} \text{WL}}{\varphi(s), s = t \Rightarrow \varphi(t)} =$$

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

LK proves that $=$ is symmetric and transitive:

$$\frac{\frac{\Rightarrow t_1 = t_1}{t_1 = t_2 \Rightarrow t_1 = t_1} \text{WL}}{t_1 = t_2 \Rightarrow t_2 = t_1} = \quad \frac{\frac{t_1 = t_2 \Rightarrow t_1 = t_2}{t_1 = t_2, t_2 = t_3 \Rightarrow t_1 = t_2} \text{WL}}{t_1 = t_2, t_2 = t_3 \Rightarrow t_1 = t_3} =$$

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

Proposition 8.34. *LK with initial sequents and rules for identity is sound.*

8.6. DERIVATIONS WITH IDENTITY PREDICATE

Proof. Initial sequents of the form $\Rightarrow t = t$ are valid, since for every structure \mathfrak{M} , $\mathfrak{M} \models t = t$. (Note that we assume the term t to be ground, i.e., it contains no variables, so variable assignments are irrelevant).

Suppose the last inference in a derivation is $=$. Then the premise $\Gamma' \Rightarrow \Delta'$ contains $t_1 = t_2$ on the left and $\varphi(t_1)$ on the right, and the conclusion is $\Gamma \Rightarrow \Delta$ where $\Gamma = \Gamma'$ and $\Delta = (\Delta' \setminus \{\varphi(t_1)\}) \cup \{\varphi(t_2)\}$. Consider a structure \mathfrak{M} . Since, by induction hypothesis, the premise $\Gamma' \Rightarrow \Delta'$ is valid, either (a) for some $\alpha \in \Gamma'$, $\mathfrak{M} \not\models \alpha$, (b) for some $\alpha \in \Delta' \setminus \{\varphi(s)\}$, $\mathfrak{M} \models \alpha$, or (c) $\mathfrak{M} \models \varphi(t_1)$. In both cases (a) and (b), since $\Gamma = \Gamma'$, and $\Delta' \setminus \{\varphi(s)\} \subseteq \Delta$, \mathfrak{M} satisfies $\Gamma \Rightarrow \Delta$. So assume cases (a) and (b) do not apply, but case (c) does. If (a) does not apply, $\mathfrak{M} \models \alpha$ for all $\alpha \in \Gamma'$, in particular, $\mathfrak{M} \models t_1 = t_2$. Therefore, $\text{Val}^{\mathfrak{M}}(t_1) = \text{Val}^{\mathfrak{M}}(t_2)$. Let s be any variable assignment, and s' be the x -variant given by $s'(x) = \text{Val}^{\mathfrak{M}}(t_1) = \text{Val}^{\mathfrak{M}}(t_2)$. By [Proposition 6.44](#), $\mathfrak{M}, s \models \varphi(t_2)$ iff $\mathfrak{M}, s' \models \varphi(x)$ iff $\mathfrak{M}, s \models \varphi(t_1)$. Since $\mathfrak{M} \models \varphi(t_1)$ therefore $\mathfrak{M} \models \varphi(t_2)$. \square

Problems

Problem 8.1. Give derivations of the following sequents:

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

Problem 8.2. Prove [Proposition 8.13](#)

Problem 8.3. Prove [Proposition 8.14](#)

Problem 8.4. Prove [Proposition 8.20](#).

Problem 8.5. Prove [Proposition 8.21](#).

Problem 8.6. Prove [Proposition 8.22](#).

Problem 8.7. Prove [Proposition 8.23](#).

Problem 8.8. Prove [Proposition 8.24](#).

Problem 8.9. Prove [Proposition 8.25](#).

Problem 8.10. Complete the proof of [Theorem 8.29](#).

Problem 8.11. Give derivations of the following sequents:

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

Chapter 9

Natural Deduction

This chapter presents a natural deduction system in the style of Gentzen/Prawitz.

To include or exclude material relevant to natural deduction as a proof system, use the “prfND” tag.

9.1 Introduction

Logical systems commonly have not just a semantics, but also proof systems. The purpose of proof 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. Moreover, good proof systems have the property that any given sequence or arrangement of formulas can be verified mechanically to be a “correct” proof. The simplest (and historically first) proof 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 proof 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 truth trees, also known as tableaux proofs, and the sequent calculus. Some proof 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

9.1. INTRODUCTION

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.

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

One feature that distinguishes natural deduction from other proof systems is its use of assumptions. In almost every proof system a single formula is at the root of the tree of formulas—usually the conclusion—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 hypotheticals 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. An assumption that remains undischarged at the end of the derivation is (usually) essential to the truth of the conclusion, and so a derivation establishes that its undischarged assumptions entail its conclusion.

For any proof system it is crucial to verify that it in fact does what it's supposed to: provide a way to verify that a sentence is entailed by some others. This is called soundness; and we will prove it for the natural deduction system we use. It is also crucial to verify the converse: that the proof system is strong enough to verify that $\Gamma \models \varphi$ whenever this holds, that there is a derivation of φ from Γ whenever $\Gamma \models \varphi$. This is called completeness—but it is much harder to prove.

9.2 Rules and Derivations

This section collects all the rules propositional connectives and quantifiers, but not for identity. It is planned to divide this into separate sections on connectives and quantifiers so that proofs for propositional logic can be treated separately (issue #77).

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

Definition 9.1 (Inference). An *inference* is an expression of the form

$$\frac{\varphi}{\chi} \quad \text{or} \quad \frac{\varphi \quad \psi}{\chi}$$

where φ, ψ , and χ are formulas. φ and ψ are called the *upper formulas* or *premises* and χ the *lower formulas* or *conclusion* of the inference.

The rules for natural deduction are divided into two main types: *propositional* rules (quantifier-free) and *quantifier* rules. The rules come in pairs, an introduction and an elimination rule for each logical operator. They introduced 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 formula as “ $[\varphi]^n$ ”.

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

Propositional Rules

Rules for \perp

$$\frac{\varphi \quad \neg\varphi}{\perp} \perp\text{Intro} \quad \frac{\perp}{\varphi} \perp\text{Elim}$$

Rules for \wedge

$$\frac{\varphi \quad \psi}{\varphi \wedge \psi} \wedge\text{Intro} \quad \frac{\varphi \wedge \psi}{\varphi} \wedge\text{Elim} \quad \frac{\varphi \wedge \psi}{\psi} \wedge\text{Elim}$$

Rules for \vee

$$\frac{\varphi}{\varphi \vee \psi} \vee\text{Intro} \quad \frac{\psi}{\varphi \vee \psi} \vee\text{Intro} \quad \frac{\begin{array}{c} [\varphi]^n \\ \vdots \\ \varphi \vee \psi \end{array} \quad \begin{array}{c} [\psi]^n \\ \vdots \\ \varphi \vee \psi \end{array}}{\chi} \vee\text{Elim}$$

9.2. RULES AND DERIVATIONS

Rules for \neg

$$\begin{array}{c} [\varphi]^n \\ \vdots \\ \perp \\ n \frac{}{\neg\varphi} \neg\text{Intro} \end{array} \qquad \frac{\neg\neg\varphi}{\varphi} \neg\text{Elim}$$

Rules for \rightarrow

$$\begin{array}{c} [\varphi]^n \\ \vdots \\ \psi \\ n \frac{}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \end{array} \qquad \frac{\varphi \quad \varphi \rightarrow \psi}{\psi} \rightarrow \text{Elim}$$

Quantifier Rules

Rules for \forall

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

where t is a ground term (a term that does not contain any variables), and a is a constant symbol which does not occur in φ , or in any assumption which is undischarged in the derivation ending with the premise φ . We call a the *eigenvariable* of the $\forall\text{Intro}$ inference.

Rules for \exists

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

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

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

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

In $\exists\text{Intro}$ and $\forall\text{Elim}$ there are no restrictions, and the term t can be anything, so we do not have to worry about any conditions. On the other hand, in the $\exists\text{Elim}$ and $\forall\text{Intro}$ rules, the eigenvariable condition requires that a does

not occur anywhere else in the formula. Thus, if the upper formula is valid, the truth values of the formulas other than $\varphi(a)$ are independent of a .

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 Intro, \rightarrow Intro, \vee Elim and \exists Elim inference rules, and the label of the discharged assumption is placed beside the inference for clarity.

Definition 9.2 (Initial Formula). An *initial formula* or *assumption* is any formula in the topmost position of any branch.

Definition 9.3 (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. Every formula in the tree is an upper formula of an inference whose lower formula stands directly below that formula in the tree.

We then say that φ is the *end-formula* of the derivation and that φ is *derivable* from Γ .

9.3 Examples of Derivations

Example 9.4. Let’s give a derivation of the formula $(\varphi \wedge \psi) \rightarrow \varphi$.

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

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

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

$$\begin{array}{c} [\varphi \wedge \psi]^1 \\ \vdots \\ \vdots \\ \vdots \\ \varphi \\ 1 \frac{}{(\varphi \wedge \psi) \rightarrow \varphi} \rightarrow \text{Intro} \end{array}$$

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

9.3. EXAMPLES OF DERIVATIONS

gives us the following proof:

$$\frac{\frac{[\varphi \wedge \psi]^1}{\varphi} \wedge \text{Elim}}{^1 \frac{(\varphi \wedge \psi) \rightarrow \varphi}{\varphi} \rightarrow \text{Intro}}$$

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

Example 9.5. Now let's give a derivation of the formula $(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)$.

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

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

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

$$\frac{\begin{array}{c} [\neg\varphi \vee \psi]^1 \\ \vdots \\ \varphi \rightarrow \psi \end{array}}{^1 \frac{(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)}{\varphi \rightarrow \psi} \rightarrow \text{Intro}}$$

This leaves us with two possibilities to continue. Either we can keep working from the bottom up and look for another application of the \rightarrow Intro rule, or we can work from the top down and apply a \vee Elim rule. Let us apply the latter. We will use the assumption $\neg\varphi \vee \psi$ as the leftmost premise of \vee Elim. For a valid application of \vee Elim, the other two premises must be identical to the conclusion $\varphi \rightarrow \psi$, 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:

$$\frac{\begin{array}{c} [\neg\varphi]^2 \quad [\psi]^2 \\ \vdots \quad \vdots \\ \varphi \rightarrow \psi \quad \varphi \rightarrow \psi \end{array}}{\frac{[\neg\varphi \vee \psi]^1 \quad \varphi \rightarrow \psi \quad \varphi \rightarrow \psi}{\varphi \rightarrow \psi} \vee \text{Elim}} \rightarrow \text{Intro}$$

In each of the two branches on the right, we want to derive $\varphi \rightarrow \psi$, which

is best done using \rightarrow Intro.

$$\begin{array}{c}
 \begin{array}{c} [\neg\varphi]^2, [\varphi]^3 \\ \vdots \\ \psi \end{array} \quad \begin{array}{c} [\psi]^2, [\varphi]^4 \\ \vdots \\ \psi \end{array} \\
 \begin{array}{c} 2 \frac{[\neg\varphi \vee \psi]^1}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \quad 3 \frac{\psi}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \quad 4 \frac{\psi}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \\ \hline \varphi \rightarrow \psi \quad \vee\text{Elim} \\ 1 \frac{(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)}{\rightarrow \text{Intro}} \end{array}
 \end{array}$$

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 \perp Intro:

$$\begin{array}{c}
 \frac{[\neg\varphi]^2 \quad [\varphi]^3}{\perp} \perp \text{Intro} \\
 \vdots \\
 \psi
 \end{array}$$

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

$$\begin{array}{c}
 \begin{array}{c} [\neg\varphi]^2 \quad [\varphi]^3 \\ \vdots \\ \perp \end{array} \quad \begin{array}{c} [\psi]^2, [\varphi]^4 \\ \vdots \\ \psi \end{array} \\
 \begin{array}{c} \frac{\perp}{\psi} \perp \text{Elim} \\ 3 \frac{\psi}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \quad 4 \frac{\psi}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \\ \hline \varphi \rightarrow \psi \quad \vee\text{Elim} \\ 2 \frac{[\neg\varphi \vee \psi]^1}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \\ \hline 1 \frac{(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)}{\rightarrow \text{Intro}} \end{array}
 \end{array}$$

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 .

$$\begin{array}{c}
 \begin{array}{c} [\neg\varphi]^2 \quad [\varphi]^3 \\ \vdots \\ \perp \end{array} \quad \begin{array}{c} [\psi]^2 \\ \vdots \\ \psi \end{array} \\
 \begin{array}{c} \frac{\perp}{\psi} \perp \text{Elim} \\ 3 \frac{\psi}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \quad \frac{[\psi]^2}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \\ \hline \varphi \rightarrow \psi \quad \vee\text{Elim} \\ 2 \frac{[\neg\varphi \vee \psi]^1}{\varphi \rightarrow \psi} \rightarrow \text{Intro} \\ \hline 1 \frac{(\neg\varphi \vee \psi) \rightarrow (\varphi \rightarrow \psi)}{\rightarrow \text{Intro}} \end{array}
 \end{array}$$

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

9.3. EXAMPLES OF DERIVATIONS

Example 9.6. 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 $\exists x \neg \varphi(x) \rightarrow \neg \forall x \varphi(x)$. Starting as usual, we write

$$\frac{}{\exists x \neg \varphi(x) \rightarrow \neg \forall x \varphi(x)}$$

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

$$\frac{\begin{array}{c} [\exists x \neg \varphi(x)]^1 \\ \vdots \\ \neg \forall x \varphi(x) \end{array}}{\exists x \neg \varphi(x) \rightarrow \neg \forall x \varphi(x)} \rightarrow \text{Intro}$$

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 Elim rule. Here we must pay attention to the eigenvariable condition, and choose a constant that does not appear in $\exists x \varphi(x)$ or any assumptions that it depends on. (Since no constant symbols appear, however, any choice will do fine.)

$$\frac{\begin{array}{c} [\neg \varphi(a)]^2 \\ \vdots \\ \neg \forall x \varphi(x) \end{array} \quad \frac{[\exists x \neg \varphi(x)]^1}{\neg \forall x \varphi(x)} \exists \text{Elim}}{\exists x \neg \varphi(x) \rightarrow \neg \forall x \varphi(x)} \rightarrow \text{Intro}$$

In order to derive $\neg \forall x \varphi(x)$, we will attempt to use the \neg Intro 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 \rightarrow Intro inference. We can set it up as follows:

$$\frac{\begin{array}{c} [\neg \varphi(a)]^2, [\forall x \varphi(x)]^3 \\ \vdots \\ \perp \end{array} \quad \frac{[\exists x \neg \varphi(x)]^1}{\neg \forall x \varphi(x)} \exists \text{Elim}}{\exists x \neg \varphi(x) \rightarrow \neg \forall x \varphi(x)} \rightarrow \text{Intro}$$

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

$$\begin{array}{c}
 \frac{\frac{\frac{[\exists x \neg \varphi(x)]^1}{\neg \forall x \varphi(x)} \quad \frac{\frac{[\neg \varphi(a)]^2 \quad \frac{[\forall x \varphi(x)]^3}{\varphi(a)} \forall \text{Elim}}{\perp} \perp \text{Intro}}{\neg \forall x \varphi(x)} \neg \text{Intro}}{\exists x \neg \varphi(x) \rightarrow \neg \forall x \varphi(x)} \exists \text{Elim} \rightarrow \text{Intro}
 \end{array}$$

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 Elim, and the eigenvariable a does not occur in any assumptions it depends on, this is a correct derivation.

Example 9.7. 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 $\forall x (\psi(x) \rightarrow \chi(x, b))$. Starting as usual, we write the end-formula at the bottom.

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

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 Elim rule. Since it has an eigenvariable condition, we will apply that rule first. We get the following:

$$\begin{array}{c}
 \frac{\frac{\exists x (\varphi(x) \wedge \psi(x))}{\exists x \chi(x, b)} \quad \frac{[\varphi(a) \wedge \psi(a)]^1}{\exists x \chi(x, b)} \exists \text{Elim}}{\exists x \chi(x, b)} \exists \text{Elim}
 \end{array}$$

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

9.3. EXAMPLES OF DERIVATIONS

point to apply $\wedge\text{Elim}$ to separate out $\psi(a)$.

$$\frac{\frac{[\varphi(a) \wedge \psi(a)]^1}{\psi(a)} \wedge\text{Elim} \quad \vdots}{\frac{1 \quad \frac{\exists x (\varphi(x) \wedge \psi(x)) \quad \exists x \chi(x, b)}{\exists x \chi(x, b)} \exists\text{Elim}}{\exists x \chi(x, b)} \exists\text{Elim}$$

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

$$\frac{\frac{\frac{[\varphi(a) \wedge \psi(a)]^1}{\psi(a)} \wedge\text{Elim} \quad \frac{\frac{\forall x (\psi(x) \rightarrow \chi(x, b))}{\psi(a) \rightarrow \chi(a, b)} \forall\text{Elim}}{\chi(a, b)} \rightarrow\text{Elim} \quad \vdots}{\frac{1 \quad \frac{\exists x (\varphi(x) \wedge \psi(x)) \quad \exists x \chi(x, b)}{\exists x \chi(x, b)} \exists\text{Elim}}{\exists x \chi(x, b)} \exists\text{Elim}$$

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

$$\frac{\frac{\frac{[\varphi(a) \wedge \psi(a)]^1}{\psi(a)} \wedge\text{Elim} \quad \frac{\frac{\forall x (\psi(x) \rightarrow \chi(x, b))}{\psi(a) \rightarrow \chi(a, b)} \forall\text{Elim}}{\chi(a, b)} \rightarrow\text{Elim} \quad \frac{\chi(a, b)}{\exists x \chi(x, b)} \exists\text{Intro}}{\frac{1 \quad \frac{\exists x (\varphi(x) \wedge \psi(x)) \quad \exists x \chi(x, b)}{\exists x \chi(x, b)} \exists\text{Elim}}{\exists x \chi(x, b)} \exists\text{Elim}$$

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

Example 9.8. Give a derivation of the formula $\neg\forall x \varphi(x)$ from the assumptions $\forall x \varphi(x) \rightarrow \exists y \psi(y)$ and $\neg\exists y \psi(y)$. Starting as usual, we write the target formula at the bottom.

$$\overline{\neg\forall x \varphi(x)}$$

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

require that we figure out how to derive a contradiction.

$$\begin{array}{c} [\forall x \varphi(x)]^1 \\ \vdots \\ \perp \\ 1 \frac{}{\neg \forall x \varphi(x)} \neg\text{Intro} \end{array}$$

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

$$\begin{array}{c} \frac{[\forall x \varphi(x)]^1 \quad \forall x \varphi(x) \rightarrow \exists y \psi(y)}{\exists y \psi(y)} \rightarrow\text{Elim} \\ \vdots \\ \perp \\ 1 \frac{}{\neg \forall x \varphi(x)} \neg\text{Intro} \end{array}$$

We now have one final assumption to work with, and it looks like this will help us reach a contradiction by using $\perp\text{Intro}$.

$$\frac{\frac{[\forall x \varphi(x)]^1 \quad \forall x \varphi(x) \rightarrow \exists y \psi(y)}{\exists y \psi(y)} \rightarrow\text{Elim} \quad \neg \exists y \psi(y)}{1 \frac{}{\neg \forall x \varphi(x)} \neg\text{Intro}} \perp\text{Intro}$$

Example 9.9. Give a derivation of the formula $\varphi(x) \vee \neg\varphi(x)$.

$$\overline{\varphi(x) \vee \neg\varphi(x)}$$

The main connective of the formula is a disjunction. Since we have no assumptions to work from, we can't use $\vee\text{Intro}$. Since we don't want any undischarged assumptions in our proof, our best bet is to use $\neg\text{Intro}$ with the assumption $\neg(\varphi(x) \vee \neg\varphi(x))$. This will allow us to discharge the assumption at the end.

$$\begin{array}{c} [\neg(\varphi(x) \vee \neg\varphi(x))]^1 \\ \vdots \\ \perp \\ 1 \frac{}{\neg \neg(\varphi(x) \vee \neg\varphi(x))} \neg\text{Intro} \\ \frac{}{\varphi(x) \vee \neg\varphi(x)} \neg\text{Elim} \end{array}$$

Note that a straightforward application of the $\neg\text{Intro}$ rule leaves us with two negations. We can remove them with the $\neg\text{Elim}$ rule.

9.4. PROOF-THEORETIC NOTIONS

We appear to be stuck again, since the assumption we introduced has a negation as the main operator. So let's try to derive another contradiction! Let's assume $\varphi(x)$ for another \neg Intro. From there we can derive $\varphi(x) \vee \neg\varphi(x)$ and get our first contradiction.

$$\begin{array}{c}
 \frac{[\neg(\varphi(x) \vee \neg\varphi(x))]^1 \quad \frac{[\varphi(x)]^2}{\varphi(x) \vee \neg\varphi(x)} \vee\text{Intro}}{\perp} \perp\text{Intro} \\
 \frac{\perp}{\neg\varphi(x)} \neg\text{Intro} \\
 \vdots \\
 \frac{\perp}{\neg\neg(\varphi(x) \vee \neg\varphi(x))} \neg\text{Intro} \\
 \frac{\neg\neg(\varphi(x) \vee \neg\varphi(x))}{\varphi(x) \vee \neg\varphi(x)} \neg\text{Elim}
 \end{array}$$

Our second assumption is now discharged. We only need to derive one more contradiction in order to discharge our first assumption. Now we have something to work with— $\neg\varphi(x)$. We can use the same strategy as last time (\vee Intro) to derive a contradiction with our first assumption.

$$\begin{array}{c}
 \frac{[\neg(\varphi(x) \vee \neg\varphi(x))]^1 \quad \frac{[\varphi(x)]^2}{\varphi(x) \vee \neg\varphi(x)} \vee\text{Intro}}{\perp} \perp\text{Intro} \\
 \frac{\perp}{\neg\varphi(x)} \neg\text{Intro} \\
 \frac{\neg\varphi(x)}{\varphi(x) \vee \neg\varphi(x)} \vee\text{Intro} \\
 \frac{\varphi(x) \vee \neg\varphi(x) \quad [\neg(\varphi(x) \vee \neg\varphi(x))]^1}{\neg\neg(\varphi(x) \vee \neg\varphi(x))} \neg\text{Intro} \\
 \frac{\neg\neg(\varphi(x) \vee \neg\varphi(x))}{\varphi(x) \vee \neg\varphi(x)} \neg\text{Elim}
 \end{array}$$

And the proof is done!

9.4 Proof-Theoretic Notions

This section collects the properties of the provability relation required for the completeness theorem. If you find the location unmotivated, include it instead in the chapter on completeness.

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

Definition 9.10 (Derivability). A formula φ is *derivable* from a set of formulas Γ , $\Gamma \vdash \varphi$, if there is a derivation with end-formula 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 9.11 (Theorems). A formula φ is a *theorem* if there is a derivation of φ from the empty set, i.e., a derivation with end-formula φ in which all assumptions are discharged. We write $\vdash \varphi$ if φ is a theorem and $\nvdash \varphi$ if it is not.

Definition 9.12 (Consistency). A set of sentences Γ is *consistent* iff $\Gamma \nvdash \perp$. If Γ is not consistent, i.e., if $\Gamma \vdash \perp$, we say it is *inconsistent*.

Proposition 9.13. $\Gamma \vdash \varphi$ iff $\Gamma \cup \{\neg\varphi\}$ is inconsistent.

Proof. Exercise. □

Proposition 9.14. Γ is inconsistent iff $\Gamma \vdash \varphi$ for every sentence φ .

Proof. Exercise. □

Proposition 9.15. $\Gamma \vdash \varphi$ iff for some finite $\Gamma_0 \subseteq \Gamma$, $\Gamma_0 \vdash \varphi$.

Proof. Any derivation of φ from Γ can only contain finitely many undischarged assumptions. If all these undischarged assumptions are in Γ , then the set of them is a finite subset of Γ . The other direction is trivial, since a derivation from a subset of Γ is also a derivation from Γ . □

9.5 Properties of Derivability

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

Proposition 9.16 (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 9.17. If $\Gamma \vdash \varphi$ and $\Gamma \cup \{\varphi\} \vdash \perp$, then Γ is inconsistent.

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

$$\begin{array}{c}
 \begin{array}{c} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{array} \delta_1 \quad \begin{array}{c} [\varphi]^1 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{array} \delta_2 \\
 \hline
 \begin{array}{c} \varphi \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{array} \quad \begin{array}{c} 1 \quad \frac{\perp}{\neg\varphi} \text{Intro} \\ \neg\varphi \end{array} \\
 \hline
 \perp \quad \neg\text{Elim}
 \end{array}$$

9.5. PROPERTIES OF DERIVABILITY

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

Proposition 9.18. *If $\Gamma \cup \{\varphi\} \vdash \perp$, then $\Gamma \vdash \neg\varphi$.*

Proof. Suppose that $\Gamma \cup \{\varphi\} \vdash \perp$. Then there is a derivation of \perp from $\Gamma \cup \{\varphi\}$. Let δ be the derivation of \perp , and consider

$$\begin{array}{c} [\varphi]^1 \\ \vdots \\ \delta \\ \vdots \\ \perp \\ 1 \frac{}{\neg\varphi} \neg\text{Intro} \end{array}$$

\square

Proposition 9.19. *If $\Gamma \cup \{\varphi\} \vdash \perp$ and $\Gamma \cup \{\neg\varphi\} \vdash \perp$, then $\Gamma \vdash \perp$.*

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

$$\begin{array}{c} \begin{array}{c} [\varphi]^1 \\ \vdots \\ \delta_1 \\ \vdots \\ \perp \\ 1 \frac{}{\neg\varphi} \neg\text{Intro} \end{array} \quad \begin{array}{c} [\neg\varphi]^2 \\ \vdots \\ \delta_2 \\ \vdots \\ \perp \\ 2 \frac{}{\neg\neg\varphi} \neg\text{Intro} \end{array} \\ \hline \perp \quad \neg\text{Elim} \end{array}$$

Since the assumptions φ and $\neg\varphi$ are discharged, this is a derivation from Γ alone. Hence $\Gamma \vdash \perp$. \square

Proposition 9.20. *If $\Gamma \cup \{\varphi\} \vdash \perp$ and $\Gamma \cup \{\psi\} \vdash \perp$, then $\Gamma \cup \{\varphi \vee \psi\} \vdash \perp$.*

Proof. Exercise. \square

Proposition 9.21. *If $\Gamma \vdash \varphi$ or $\Gamma \vdash \psi$, then $\Gamma \vdash \varphi \vee \psi$.*

Proof. Suppose $\Gamma \vdash \varphi$. There is a derivation δ of φ from Γ . We can derive

$$\begin{array}{c} \vdots \\ \delta \\ \vdots \\ \varphi \\ \hline \varphi \vee \psi \quad \vee\text{Intro} \end{array}$$

Therefore $\Gamma \vdash \varphi \vee \psi$. The proof for when $\Gamma \vdash \psi$ is similar. \square

Proposition 9.22. *If $\Gamma \vdash \varphi \wedge \psi$ then $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$.*

Proof. If $\Gamma \vdash \varphi \wedge \psi$, there is a derivation δ of $\varphi \wedge \psi$ from Γ . Consider

$$\frac{\begin{array}{c} \vdots \\ \vdots \delta \\ \vdots \\ \varphi \wedge \psi \end{array}}{\varphi} \wedge \text{Elim}$$

Hence, $\Gamma \vdash \varphi$. A similar derivation shows that $\Gamma \vdash \psi$. \square

Proposition 9.23. *If $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$, then $\Gamma \vdash \varphi \wedge \psi$.*

Proof. If $\Gamma \vdash \varphi$ as well as $\Gamma \vdash \psi$, there are derivations δ_1 of φ and δ_2 of ψ from Γ . Consider

$$\frac{\begin{array}{c} \vdots \delta_1 \\ \vdots \\ \varphi \end{array} \quad \begin{array}{c} \vdots \delta_2 \\ \vdots \\ \psi \end{array}}{\varphi \wedge \psi} \wedge \text{Intro}$$

The undischarged assumptions of the new derivation are all in Γ , so we have $\Gamma \vdash \varphi \wedge \psi$. \square

Proposition 9.24. *If $\Gamma \vdash \varphi$ and $\Gamma \vdash \varphi \rightarrow \psi$, then $\Gamma \vdash \psi$.*

Proof. Suppose that $\Gamma \vdash \varphi$ and $\Gamma \vdash \varphi \rightarrow \psi$. There are derivations δ_1 of φ from Γ and δ_2 of $\varphi \rightarrow \psi$ from Γ . Consider:

$$\frac{\begin{array}{c} \vdots \delta_1 \\ \vdots \\ \varphi \end{array} \quad \begin{array}{c} \vdots \delta_2 \\ \vdots \\ \varphi \rightarrow \psi \end{array}}{\psi} \rightarrow \text{Elim}$$

This means that $\Gamma \vdash \psi$. \square

Proposition 9.25. *If $\Gamma \vdash \neg\varphi$ or $\Gamma \vdash \psi$, then $\Gamma \vdash \varphi \rightarrow \psi$.*

Proof. First suppose $\Gamma \vdash \neg\varphi$. Then there is a derivation δ of $\neg\varphi$ from Γ . The following derivation shows that $\Gamma \vdash \varphi \rightarrow \psi$:

$$\frac{\begin{array}{c} \vdots \delta \\ \vdots \\ \neg\varphi \end{array} \quad \frac{\begin{array}{c} [\varphi]^1 \\ \vdots \\ \perp \end{array}}{\psi} \perp \text{Intro}}{\psi} \perp \text{Elim}$$

$$^1 \frac{\varphi \rightarrow \psi}{\varphi \rightarrow \psi} \rightarrow \text{Intro}$$

9.5. PROPERTIES OF DERIVABILITY

Now suppose $\Gamma \vdash \psi$. Then there is a derivation δ of ψ from Γ . The following derivation shows that $\Gamma \vdash \varphi \rightarrow \psi$:

$$\frac{\frac{[\varphi]^1 \quad \frac{\vdots \delta}{\psi} \wedge \text{Intro}}{\varphi \wedge \psi} \wedge \text{Elim}}{1 \quad \frac{\psi}{\varphi \rightarrow \psi} \rightarrow \text{Intro}}$$

□

Theorem 9.26. *If c is a constant not occurring in Γ or $\varphi(x)$ and $\Gamma \vdash \varphi(c)$, then $\Gamma \vdash \forall x \varphi(x)$.*

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

Theorem 9.27. 1. *If $\Gamma \vdash \varphi(t)$ then $\Gamma \vdash \exists x \varphi(x)$.*

2. *If $\Gamma \vdash \forall x \varphi(x)$ then $\Gamma \vdash \varphi(t)$.*

Proof. 1. Suppose $\Gamma \vdash \varphi(t)$. Then there is a derivation δ of $\varphi(t)$ from Γ . The derivation

$$\frac{\frac{\vdots \delta}{\varphi(t)} \exists \text{Intro}}{\exists x \varphi(x)}$$

shows that $\Gamma \vdash \exists x \varphi(x)$.

2. Suppose $\Gamma \vdash \forall x \varphi(x)$. Then there is a derivation δ of $\forall x \varphi(x)$ from Γ . The derivation

$$\frac{\frac{\vdots \delta}{\forall x \varphi(x)} \forall \text{Elim}}{\varphi(t)}$$

shows that $\Gamma \vdash \varphi(t)$.

□

9.6 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 9.28 (Soundness). *If φ is derivable from the undischarged assumptions Γ , then $\Gamma \models \varphi$.*

Proof. Inductive Hypothesis: The premises of an inference rule follow from the undischarged assumptions of the subproofs ending in those premises.

Inductive Step: Show that φ follows from the undischarged assumptions of the entire proof.

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

If the number of inferences is 0, then δ consists only of an initial formula. Every initial formula φ is an undischarged assumption, and as such, any structure \mathfrak{M} that satisfies all of the undischarged assumptions of the proof also satisfies φ .

If the number of inferences is greater than 0, we distinguish cases according to the type of the lowermost inference. By induction hypothesis, we can assume that the premises of that inference follow from the undischarged assumptions of the sub-derivations ending in those premises, respectively.

First, we consider the possible inferences with only one premise.

1. Suppose that the last inference is \neg -Intro: The derivation has the form

$$\begin{array}{c}
 \Gamma, [\varphi]^n \\
 \vdots \\
 \delta_1 \\
 \vdots \\
 \frac{}{\neg\varphi} \neg\text{-Intro}
 \end{array}$$

9.6. SOUNDNESS

By inductive hypothesis, \perp follows from the undischarged assumptions $\Gamma \cup \{\varphi\}$ of δ_1 . Consider a structure \mathfrak{M} . We need to show that, if $\mathfrak{M} \models \Gamma$, then $\mathfrak{M} \models \neg\varphi$. Suppose for reductio that $\mathfrak{M} \models \Gamma$, but $\mathfrak{M} \not\models \neg\varphi$, i.e., $\mathfrak{M} \models \varphi$. This would mean that $\mathfrak{M} \models \Gamma \cup \{\varphi\}$. This is contrary to our inductive hypothesis. So, $\mathfrak{M} \models \neg\varphi$.

2. The last inference is \neg Elim: Exercise.
3. The last inference is \wedge Elim: 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:

$$\frac{\begin{array}{c} \Gamma \\ \vdots \\ \delta_1 \\ \vdots \\ \varphi \wedge \psi \end{array}}{\varphi} \wedge\text{Elim}$$

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 $\mathfrak{M} \models \Gamma$, then $\mathfrak{M} \models \varphi$. Suppose $\mathfrak{M} \models \Gamma$. By our inductive hypothesis ($\Gamma \models \varphi \vee \psi$), we know that $\mathfrak{M} \models \varphi \wedge \psi$. By definition, $\mathfrak{M} \models \varphi \wedge \psi$ iff $\mathfrak{M} \models \varphi$ and $\mathfrak{M} \models \psi$. (The case where ψ is inferred from $\varphi \wedge \psi$ is handled similarly.)

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

$$\frac{\begin{array}{c} \Gamma \\ \vdots \\ \delta_1 \\ \vdots \\ \varphi \end{array}}{\varphi \vee \psi} \vee\text{Intro}$$

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

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

$$\frac{\begin{array}{c} \Gamma, [\varphi]^n \\ \vdots \\ \delta_1 \\ \vdots \\ \psi \end{array}}{\varphi \rightarrow \psi} \rightarrow \text{Intro}$$

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 $\Gamma \models \varphi \rightarrow \psi$. For reductio, suppose that for some structure \mathfrak{M} , $\mathfrak{M} \models \Gamma$ but $\mathfrak{M} \not\models \varphi \rightarrow \psi$. 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, $\Gamma \models \varphi \rightarrow \psi$.

6. The last inference is \forall Intro: Then δ has the form

$$\frac{\begin{array}{c} \Gamma \\ \vdots \\ \delta_1 \\ \vdots \\ \varphi(a) \end{array}}{\forall x A(x)} \forall \text{Intro}$$

The premise $\varphi(a)$ is a consequence of the undischarged assumptions Γ by induction hypothesis. Consider some structure, \mathfrak{M} , such that $\mathfrak{M} \models \Gamma$. We need to show that $\mathfrak{M} \models \forall x \varphi(x)$. Since $\forall x \varphi(x)$ is a sentence, this means we have to show that for every variable assignment s , $\mathfrak{M}, s \models \varphi(x)$ (Proposition 6.40). Since Γ consists entirely of sentences, $\mathfrak{M}, s \models \psi$ for all $\psi \in \Gamma$ by Definition 6.34. 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 6.42. Since $\Gamma \models A(a)$, $\mathfrak{M}' \models A(a)$. $\mathfrak{M}', s \models \varphi(x)$ iff $\mathfrak{M}' \models \varphi(a)$ by Proposition 6.44 (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 6.41, $\mathfrak{M}, s \models \varphi(x)$. But s was an arbitrary variable assignment, so $\mathfrak{M} \models \forall x \varphi(x)$.

7. The last inference is \exists Intro: Exercise.
 8. The last inference is \forall Elim: Exercise.

Now let's consider the possible inferences with several premises: \forall Elim, \wedge Intro, \rightarrow Elim, and \exists Elim.

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

9.7. DERIVATIONS WITH IDENTITY PREDICATE

$$\frac{\begin{array}{c} \Gamma_1 \\ \vdots \\ \delta_1 \\ \vdots \\ \varphi \end{array} \quad \begin{array}{c} \Gamma_2 \\ \vdots \\ \delta_2 \\ \vdots \\ \psi \end{array}}{\varphi \wedge \psi} \wedge \text{Intro}$$

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

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

$$\frac{\begin{array}{c} \Gamma_1 \\ \vdots \\ \delta_1 \\ \vdots \\ \varphi \rightarrow \psi \end{array} \quad \begin{array}{c} \Gamma_2 \\ \vdots \\ \delta_2 \\ \vdots \\ \varphi \end{array}}{\psi} \rightarrow \text{Elim}$$

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 structure \mathfrak{M} . We need to show that, if $\mathfrak{M} \models \Gamma_1 \cup \Gamma_2$, then $\mathfrak{M} \models \psi$. Suppose $\mathfrak{M} \models \Gamma_1 \cup \Gamma_2$. Since $\Gamma_1 \models \varphi \rightarrow \psi$, $\mathfrak{M} \models \varphi \rightarrow \psi$. Since $\Gamma_2 \models \varphi$, we have $\mathfrak{M} \models \varphi$. This means that $\mathfrak{M} \models \psi$ (For if $\mathfrak{M} \not\models \psi$, since $\mathfrak{M} \models \varphi$, we'd have $\mathfrak{M} \not\models \varphi \rightarrow \psi$, contradicting $\mathfrak{M} \models \varphi \rightarrow \psi$).

4. The last inference is \exists Elim: Exercise.

□

Corollary 9.29. *If $\vdash \varphi$, then φ is valid.*

Corollary 9.30. *If Γ is satisfiable, then it is consistent.*

Proof. We prove the contrapositive. Suppose that Γ is not consistent. Then $\Gamma \vdash \perp$, i.e., there is a derivation of \perp from undischarged assumptions in Γ . By [Theorem 9.28](#), any structure \mathfrak{M} that satisfies Γ must satisfy \perp . Since $\mathfrak{M} \not\models \perp$ for every structure \mathfrak{M} , no \mathfrak{M} can satisfy Γ , i.e., Γ is not satisfiable. □

9.7 Derivations with Identity predicate

Derivations with the identity predicate require additional inference rules.

Rules for =:

$$\frac{}{t = t} = \text{Intro}$$

$$\frac{t_1 = t_2 \quad \varphi(t_1)}{\varphi(t_2)} = \text{Elim} \quad \text{and} \quad \frac{t_1 = t_2 \quad \varphi(t_2)}{\varphi(t_1)} = \text{Elim}$$

where t_1 and t_2 are closed terms. The $=$ Intro rule allows us to derive any identity statement of the form $t = t$ outright, from no assumptions.

Example 9.31. If s and t are closed terms, then $\varphi(s), s = t \vdash \varphi(t)$:

$$\frac{\varphi(s) \quad s = t}{\varphi(t)} = \text{Elim}$$

This may be familiar as the “principle of substitutability of identicals,” or Leibniz’ Law.

Example 9.32. 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:

$$\begin{array}{c} \exists x \forall y (\varphi(y) \rightarrow y = x) \quad [\varphi(a) \wedge \varphi(b)]^1 \\ \vdots \\ a = b \\ 1 \frac{a = b}{((\varphi(a) \wedge \varphi(b)) \rightarrow a = b)} \rightarrow \text{Intro} \\ \frac{((\varphi(a) \wedge \varphi(b)) \rightarrow a = b)}{\forall y ((\varphi(a) \wedge \varphi(y)) \rightarrow a = y)} \forall \text{Intro} \\ \frac{\forall y ((\varphi(a) \wedge \varphi(y)) \rightarrow a = y)}{\forall x \forall y ((\varphi(x) \wedge \varphi(y)) \rightarrow x = y)} \forall \text{Intro} \end{array}$$

We’ll now have to use the main assumption: since it is an existential formula, we use \exists Elim to derive the intermediary conclusion $a = b$.

$$\begin{array}{c} [\forall y (\varphi(y) \rightarrow y = c)]^2 \\ [\varphi(a) \wedge \varphi(b)]^1 \\ \vdots \\ a = b \\ 2 \frac{\exists x \forall y (\varphi(y) \rightarrow y = x) \quad a = b}{a = b} \exists \text{Elim} \\ 1 \frac{a = b}{((\varphi(a) \wedge \varphi(b)) \rightarrow a = b)} \rightarrow \text{Intro} \\ \frac{((\varphi(a) \wedge \varphi(b)) \rightarrow a = b)}{\forall y ((\varphi(a) \wedge \varphi(y)) \rightarrow a = y)} \forall \text{Intro} \\ \frac{\forall y ((\varphi(a) \wedge \varphi(y)) \rightarrow a = y)}{\forall x \forall y ((\varphi(x) \wedge \varphi(y)) \rightarrow x = y)} \forall \text{Intro} \end{array}$$

9.8. SOUNDNESS OF IDENTITY PREDICATE RULES

The sub-derivation on the top right is completed by using its assumptions to show that $a = c$ and $b = c$. This requires two separate derivations. The derivation for $a = c$ is as follows:

$$\frac{\frac{[\forall y (\varphi(y) \rightarrow y = c)]^2}{\varphi(a) \rightarrow a = c} \forall\text{Elim} \quad \frac{[\varphi(a) \wedge \varphi(b)]^1}{\varphi(a)} \wedge\text{Elim}}{a = c} \rightarrow \text{Elim}$$

From $a = c$ and $b = c$ we derive $a = b$ by $= \text{Elim}$.

9.8 Soundness of Identity predicate Rules

Proposition 9.33. *Natural deduction with rules for identity is sound.*

Proof. Any formula of the form $t = t$ is valid, since for every structure \mathfrak{M} , $\mathfrak{M} \models t = t$. (Note that we assume the term t to be ground, i.e., it contains no variables, so variable assignments are irrelevant).

Suppose the last inference in a derivation is $= \text{Elim}$, i.e., the derivation has the following form:

$$\frac{\begin{array}{c} \Gamma_1 \\ \vdots \\ \delta_1 \\ \vdots \\ t_1 = t_2 \end{array} \quad \begin{array}{c} \Gamma_2 \\ \vdots \\ \delta_2 \\ \vdots \\ \varphi(t_1) \end{array}}{\varphi(t_2)} = \text{Elim}$$

The premises $t_1 = t_2$ and $\varphi(t_1)$ are derived from undischarged assumptions Γ_1 and Γ_2 , respectively. We want to show that $\varphi(t_2)$ follows from $\Gamma_1 \cup \Gamma_2$. Consider a structure \mathfrak{M} with $\mathfrak{M} \models \Gamma_1 \cup \Gamma_2$. By induction hypothesis, $\mathfrak{M} \models \varphi(t_1)$ and $\mathfrak{M} \models t_1 = t_2$. Therefore, $\text{Val}^{\mathfrak{M}}(t_1) = \text{Val}^{\mathfrak{M}}(t_2)$. Let s be any variable assignment, and s' be the x -variant given by $s'(x) = \text{Val}^{\mathfrak{M}}(t_1) = \text{Val}^{\mathfrak{M}}(t_2)$. By [Proposition 6.44](#), $\mathfrak{M}, s \models \varphi(t_1)$ iff $\mathfrak{M}, s' \models \varphi(x)$ iff $\mathfrak{M}, s \models \varphi(t_2)$. Since $\mathfrak{M} \models \varphi(t_1)$, we have $\mathfrak{M} \models \varphi(t_2)$. \square

Problems

Problem 9.1. Give derivations of the following formulas:

1. $\neg(\varphi \rightarrow \psi) \rightarrow (\varphi \wedge \neg\psi)$
2. $\forall x (\varphi(x) \rightarrow \psi) \rightarrow (\exists y \varphi(y) \rightarrow \psi)$

Problem 9.2. Prove [Proposition 9.13](#)

Problem 9.3. Prove [Proposition 9.14](#)

Problem 9.4. Prove [Proposition 9.20](#)

Problem 9.5. Prove Proposition 9.21.

Problem 9.6. Prove Proposition 9.22.

Problem 9.7. Prove Proposition 9.23.

Problem 9.8. Prove Proposition 9.24.

Problem 9.9. Prove Proposition 9.25.

Problem 9.10. Complete the proof of Theorem 9.28.

Problem 9.11. Prove that $=$ is both symmetric and transitive, i.e., give derivations of $\forall x \forall y (x = y \rightarrow y = x)$ and $\forall x \forall y \forall z ((x = y \wedge y = z) \rightarrow x = z)$

Problem 9.12. Give derivations of the following formulas:

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

Chapter 10

The Completeness Theorem

10.1 Introduction

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

These aren't the only reasons the completeness theorem—or rather, its proof—is important. It has a number of important consequences, some of which we'll discuss separately. For instance, since any derivation that shows $\Gamma \vdash \varphi$ is finite and so can only use finitely many of the sentences in Γ , it follows by the completeness theorem that if φ is a consequence of Γ , it is already a consequence of a finite subset of Γ . This is called *compactness*. Equivalently, if every finite subset of Γ is consistent, then Γ itself must be consistent. It also follows from *the proof of* the completeness theorem that any satisfiable set of sentences has a finite or denumerable model. This result is called the Löwenheim-Skolem theorem.

10.2 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 tack. The shift in perspective required is this: completeness can also be formulated as: “if Γ is consistent, it has a model.” Perhaps we can use the information in Γ together with the hypothesis that it is consistent to construct a model. After all, we know what kind of model we are looking for: one that is as Γ describes it!

If Γ contains only atomic sentences, it is easy to construct a model for it: for atomic sentences are all of the form $P(a_1, \dots, a_n)$ where the a_i are constant symbols. So all we have to do is come up with a domain $|\mathfrak{M}|$ and an interpretation for P so that $\mathfrak{M} \models P(a_1, \dots, a_n)$. But nothing’s easier than that: put $|\mathfrak{M}| = \mathbb{N}$, $c_i^{\mathfrak{M}} = i$, and for every $P(a_1, \dots, a_n) \in \Gamma$, put the tuple $\langle k_1, \dots, k_n \rangle$ into $P^{\mathfrak{M}}$, where k_i is the index of the constant symbol a_i (i.e., $a_i \equiv c_{k_i}$).

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

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

This suggests the following idea: we add additional sentences to Γ so as to (a) keep the resulting set consistent and (b) make sure that for every possible atomic sentence φ , either φ is in the resulting set, or $\neg\varphi$, and (c) such that, whenever $\varphi \wedge \psi$ is in the set, so are both φ and ψ , if $\varphi \vee \psi$ is in the set, at least one of φ or ψ is also, etc. We keep doing this (potentially forever). Call the set of all sentences so added Γ^* . Then our construction above would provide us with a structure for which we could prove, by induction, that all sentences in Γ^* are true in \mathfrak{M} , and hence also all sentence in Γ since $\Gamma \subseteq \Gamma^*$.

There is one wrinkle in this plan: if $\exists x \varphi(x) \in \Gamma$ we would hope to be able to pick some constant symbol c and add $\varphi(c)$ in this process. But how do we know we can always do that? Perhaps we only have a few constant symbols in our language, and for each one of them we have $\neg\psi(c) \in \Gamma$. We can’t also add $\psi(c)$, since this would make the set inconsistent, and we wouldn’t know whether \mathfrak{M} has to make $\psi(c)$ or $\neg\psi(c)$ true. Moreover, it might happen that Γ contains only sentences in a language that has no constant symbols at all (e.g., the language of set theory).

The solution to this problem is to simply add infinitely many constants at the beginning, plus sentences that connect them with the quantifiers in the right way. (Of course, we have to verify that this cannot introduce an inconsistency.)

Our original construction works well if we only have constant symbols in the atomic sentences. But the language might also contain function symbols. In that case, it might be tricky to find the right functions on \mathbb{N} to assign to

10.3. MAXIMALLY CONSISTENT SETS OF SENTENCES

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, \dots, t_n as input, produces the term $f_i^n(t_1, \dots, t_n)$ as value.

The last piece of the puzzle is what to do with $=$. The predicate symbol $=$ has a fixed interpretation: $\mathfrak{M} \models t = t'$ iff $\text{Val}^{\mathfrak{M}}(t) = \text{Val}^{\mathfrak{M}}(t')$. 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: $(o + o) = o$). To solve this problem, we change the domain of \mathfrak{M} : instead of using terms as the objects in $|\mathfrak{M}|$, we use sets of terms, and each set is so that it contains all those terms which the sentences in Γ require to be equal. So, e.g., if Γ is a theory of arithmetic, one of these sets will contain: $o, (o + o), (o \times o)$, etc. This will be the set we assign to o , and it will turn out that this set is also the value of all the terms in it, e.g., also of $(o + o)$. Therefore, the sentence $(o + o) = o$ will be true in this revised structure.

10.3 Maximally Consistent Sets of Sentences

Definition 10.1 (Maximally consistent set). A set Γ of sentences is *maximally consistent* iff

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

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

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

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

Maximally consistent sets are important in the completeness proof since we can guarantee that every consistent set of sentences Γ is contained in a maximally consistent set Γ^* , and a maximally consistent set contains, for each sentence φ , either φ or its negation $\neg\varphi$. This is true in particular for atomic sentences, so from a maximally consistent set in a language suitably expanded

by constant symbols, we can construct a structure where the interpretation of predicate symbols is defined according to which atomic sentences are in Γ^* . This structure can then be shown to make all sentences in Γ^* (and hence also in Γ) true. The proof of this latter fact requires that $\neg\varphi \in \Gamma^*$ iff $\varphi \notin \Gamma^*$, $(\varphi \vee \psi) \in \Gamma^*$ iff $\varphi \in \Gamma^*$ or $\psi \in \Gamma^*$, etc.

Proposition 10.2. *Suppose Γ is maximally consistent. Then:*

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

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

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

Suppose that $\Gamma \vdash \varphi$. Suppose to the contrary that $\varphi \notin \Gamma$: then since Γ is maximally consistent, $\Gamma \cup \{\varphi\}$ is inconsistent, hence $\Gamma \cup \{\varphi\} \vdash \perp$. By [Propositions 8.17](#) and [9.17](#), Γ is inconsistent. This contradicts the assumption that Γ is consistent. Hence, it cannot be the case that $\varphi \notin \Gamma$, so $\varphi \in \Gamma$.

2. For any φ , either $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$.

Suppose to the contrary that for some φ both $\varphi \notin \Gamma$ and $\neg\varphi \notin \Gamma$. Since Γ is maximally consistent, $\Gamma \cup \{\varphi\}$ and $\Gamma \cup \{\neg\varphi\}$ are both inconsistent, so $\Gamma \cup \{\varphi\} \vdash \perp$ and $\Gamma \cup \{\neg\varphi\} \vdash \perp$. By [Propositions 8.19](#) and [9.19](#), Γ is inconsistent, a contradiction. Hence there cannot be such a sentence φ and, for every φ , $\varphi \in \Gamma$ or $\neg\varphi \in \Gamma$.

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

For the forward direction, suppose $(\varphi \wedge \psi) \in \Gamma$. Then $\Gamma \vdash \varphi \wedge \psi$. By [Propositions 8.22](#) and [9.22](#), $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$. By (1), $\varphi \in \Gamma$ and $\psi \in \Gamma$, as required.

For the reverse direction, let $\varphi \in \Gamma$ and $\psi \in \Gamma$. Then $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$. By [Propositions 8.23](#) and [9.23](#), $\Gamma \vdash \varphi \wedge \psi$. By (1), $(\varphi \wedge \psi) \in \Gamma$.

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

For the contrapositive of the forward direction, suppose that $\varphi \notin \Gamma$ and $\psi \notin \Gamma$. We want to show that $(\varphi \vee \psi) \notin \Gamma$. Since Γ is maximally consistent, $\Gamma \cup \{\varphi\} \vdash \perp$ and $\Gamma \cup \{\psi\} \vdash \perp$. By [Propositions 8.20](#) and [9.20](#), $\Gamma \cup \{(\varphi \vee \psi)\}$ is inconsistent. Hence, $(\varphi \vee \psi) \notin \Gamma$, as required.

10.4. HENKIN EXPANSION

For the reverse direction, suppose that $\varphi \in \Gamma$ or $\psi \in \Gamma$. Then $\Gamma \vdash \varphi$ or $\Gamma \vdash \psi$. By [Propositions 8.21](#) and [9.21](#), $\Gamma \vdash \varphi \vee \psi$. By (1), $(\varphi \vee \psi) \in \Gamma$, as required.

5. $(\varphi \rightarrow \psi) \in \Gamma$ iff either $\varphi \notin \Gamma$ or $\psi \in \Gamma$:

For the forward direction, let $(\varphi \rightarrow \psi) \in \Gamma$, and suppose to the contrary that $\varphi \in \Gamma$ and $\psi \notin \Gamma$. On these assumptions, $\Gamma \vdash \varphi \rightarrow \psi$ and $\Gamma \vdash \varphi$. By [Propositions 8.24](#) and [9.24](#), $\Gamma \vdash \psi$. But then by (1), $\psi \in \Gamma$, contradicting the assumption that $\psi \notin \Gamma$.

For the reverse direction, first consider the case where $\varphi \notin \Gamma$. By (2), $\neg\varphi \in \Gamma$ and hence $\Gamma \vdash \neg\varphi$. By [Propositions 8.25](#) and [9.25](#), $\Gamma \vdash \varphi \rightarrow \psi$. Again by (1), we get that $(\varphi \rightarrow \psi) \in \Gamma$, as required.

Now consider the case where $\psi \in \Gamma$. Then $\Gamma \vdash \psi$ and by [Propositions 8.25](#) and [9.25](#), $\Gamma \vdash \varphi \rightarrow \psi$. By (1), $(\varphi \rightarrow \psi) \in \Gamma$.

□

10.4 Henkin Expansion

Part of the challenge in proving the completeness theorem is that the model we construct from a maximally consistent set Γ must make all the quantified formulas in Γ true. In order to guarantee this, we use a trick due to Leon Henkin. In essence, the trick consists in expanding the language by infinitely many constants and adding, for each formula with one free variable $\varphi(x)$ a formula of the form $\exists x \varphi \rightarrow \varphi(c)$, where c is one of the new constant symbols. When we construct the structure satisfying Γ , this will guarantee that each true existential sentence has a witness among the new constants.

Lemma 10.3. *If Γ is consistent in \mathcal{L} and \mathcal{L}' is obtained from \mathcal{L} by adding a denumerable set of new constant symbols c_1, c_2, \dots , then Γ is consistent in \mathcal{L}' .*

Definition 10.4 (Saturated set). A set Γ of formulas of a language \mathcal{L} is *saturated* if and only if for each formula $\varphi \in \text{Frm}(\mathcal{L})$ and variable x there is a constant symbol c such that $\exists x \varphi \rightarrow \varphi(c) \in \Gamma$.

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

Definition 10.5. Let \mathcal{L}' be as in [Lemma 10.3](#). Fix an enumeration $\langle \varphi_1, x_1 \rangle, \langle \varphi_2, x_2 \rangle, \dots$ of all formula-variable pairs of \mathcal{L}' . We define the sentences θ_n by recursion on n . Assuming that $\theta_1, \dots, \theta_n$ have already been defined, let c_{n+1} be the first new constant symbol among the d_i that does not occur in $\theta_1, \dots, \theta_n$, and let θ_{n+1} be the formula $\exists x_{n+1} \varphi_{n+1}(x_{n+1}) \rightarrow \varphi_{n+1}(c_{n+1})$. This includes the case where $n = 0$ and the list of previous θ_i 's is empty, i.e., θ_1 is $\exists x_1 \varphi_1 \rightarrow \varphi_1(c_1)$.

Theorem 10.6. *Every consistent set Γ can be extended to a saturated consistent set Γ' .*

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

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

i.e., $\Gamma_n = \Gamma \cup \{\theta_1, \dots, \theta_n\}$, and let $\Gamma' = \bigcup_n \Gamma_n$. To show that Γ' is consistent it suffices to show, by induction on n , that each set Γ_n is consistent.

The induction basis is simply the claim that $\Gamma_0 = \Gamma$ is consistent, which is the hypothesis of the theorem. For the induction step, suppose that Γ_{n-1} is consistent but $\Gamma_n = \Gamma_{n-1} \cup \{\theta_n\}$ is inconsistent. Recall that θ_n is $\exists x_n \varphi_n(x_n) \rightarrow \varphi_n(c_n)$, where $\varphi(x)$ is a formula of \mathcal{L}' with only the variable x_n free and not containing any constant symbols c_i where $i \geq n$.

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

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

Here c_n does not occur in Γ_{n-1} or $\varphi_n(x_n)$ (remember, it was added only with θ_n). By Theorems 8.26 and 9.26, from $\Gamma \vdash \neg \varphi_n(c_n)$, we obtain $\Gamma \vdash \forall x_n \neg \varphi_n(x_n)$. Thus we have that both $\Gamma_{n-1} \vdash \exists x_n \varphi_n$ and $\Gamma_{n-1} \vdash \forall x_n \neg \varphi_n(x_n)$, so Γ itself is inconsistent. (Note that $\forall x_n \neg \varphi_n(x_n) \vdash \neg \exists x_n \varphi_n(x_n)$.) Contradiction: Γ_{n-1} was supposed to be consistent. Hence $\Gamma_n \cup \{\theta_n\}$ is consistent. \square

10.5 Lindenbaum's Lemma

We now prove a lemma that shows that any consistent set of sentences is contained in some set of sentences which is not just consistent, but maximally so, and moreover, is saturated. The proof works by first extending the set to a saturated set, and then adding one sentence at a time, guaranteeing at each step that the set remains consistent. The union of all stages in that construction then contains, for each sentence φ , either it or its negation $\neg \varphi$, is saturated, and is also consistent.

Lemma 10.7 (Lindenbaum's Lemma). *Every consistent set Γ can be extended to a maximally consistent saturated set Γ^* .*

Proof. Let Γ be consistent, and let Γ' be as in the proof of Theorem 10.6: we proved there that Γ' is a consistent saturated set in the richer language \mathcal{L}'

10.6. CONSTRUCTION OF A MODEL

(with the denumerable set of new constants). Let $\varphi_0, \varphi_1, \dots$ 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 $\Gamma^* = \bigcup_{n \geq 0} \Gamma_n$. Since $\Gamma' \subseteq \Gamma^*$, for each formula φ , Γ^* contains a formula of the form $\exists x \varphi \rightarrow \varphi(c)$ and thus is saturated.

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

Every formula of $\text{Frm}(\mathcal{L}')$ appears on the list used to define Γ^* . If $\varphi_n \notin \Gamma^*$, then that is because $\Gamma_n \cup \{\varphi_n\}$ was inconsistent. But that means that Γ^* is maximally consistent. \square

10.6 Construction of a Model

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

Definition 10.8 (Term model). Let Γ^* be a maximally consistent, saturated set of sentences in a language \mathcal{L} . The *term model* $\mathfrak{M}(\Gamma^*)$ of Γ^* is the structure defined as follows:

1. The domain $|\mathfrak{M}(\Gamma^*)|$ is the set of all closed terms of \mathcal{L} .
2. The interpretation of a constant symbol c is c itself: $c^{\mathfrak{M}(\Gamma^*)} = c$.
3. The function symbol f is assigned the function which, given as arguments the closed terms t_1, \dots, t_n , has as value the closed term $f(t_1, \dots, t_n)$:

$$f^{\mathfrak{M}(\Gamma^*)}(t_1, \dots, t_n) = f(t_1, \dots, t_n)$$

4. If R is an n -place predicate symbol, then $\langle t_1, \dots, t_n \rangle \in R^{\mathfrak{M}(\Gamma^*)}$ iff $R(t_1, \dots, t_n) \in \Gamma^*$.

Lemma 10.9 (Truth Lemma). Suppose φ does not contain $=$. Then $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\varphi \in \Gamma^*$.

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

1. $\varphi \equiv \perp$: $\mathfrak{M}(\Gamma^*) \not\models \perp$ by definition of satisfaction. On the other hand, $\perp \notin \Gamma^*$ since Γ^* is consistent.

2. $\varphi \equiv \top$: $\mathfrak{M}(\Gamma^*) \models \top$ by definition of satisfaction. On the other hand, $\top \in \Gamma^*$ since Γ^* is maximally consistent and $\Gamma^* \vdash \top$.
3. $\varphi \equiv R(t_1, \dots, t_n)$: $\mathfrak{M}(\Gamma^*) \models R(t_1, \dots, t_n)$ iff $\langle t_1, \dots, t_n \rangle \in R^{\mathfrak{M}(\Gamma^*)}$ (by the definition of satisfaction) iff $R(t_1, \dots, t_n) \in \Gamma^*$ (the construction of $\mathfrak{M}(\Gamma^*)$).
4. $\varphi \equiv \neg\psi$: $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\mathfrak{M}(\Gamma^*) \not\models \psi$ (by definition of satisfaction). By induction hypothesis, $\mathfrak{M}(\Gamma^*) \not\models \psi$ iff $\psi \notin \Gamma^*$. By [Proposition 10.2\(2\)](#), $\neg\psi \in \Gamma^*$ if $\psi \notin \Gamma^*$; and $\neg\psi \notin \Gamma^*$ if $\psi \in \Gamma^*$ since Γ^* is consistent.
5. $\varphi \equiv \psi \wedge \chi$: $\mathfrak{M}(\Gamma^*) \models \varphi$ iff we have both $\mathfrak{M}(\Gamma^*) \models \psi$ and $\mathfrak{M}(\Gamma^*) \models \chi$ (by definition of satisfaction) iff both $\psi \in \Gamma^*$ and $\chi \in \Gamma^*$ (by the induction hypothesis). By [Proposition 10.2\(3\)](#), this is the case iff $(\psi \wedge \chi) \in \Gamma^*$.
6. $\varphi \equiv \psi \vee \chi$: $\mathfrak{M}(\Gamma^*) \models \varphi$ iff at $\mathfrak{M}(\Gamma^*) \models \psi$ or $\mathfrak{M}(\Gamma^*) \models \chi$ (by definition of satisfaction) iff $\psi \in \Gamma^*$ or $\chi \in \Gamma^*$ (by induction hypothesis). This is the case iff $(\psi \vee \chi) \in \Gamma^*$ (by [Proposition 10.2\(4\)](#)).
7. $\varphi \equiv \psi \rightarrow \chi$: $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\mathfrak{M}(\Gamma^*) \not\models \psi$ or $\mathfrak{M}(\Gamma^*) \models \chi$ (by definition of satisfaction) iff $\psi \notin \Gamma^*$ or $\chi \in \Gamma^*$ (by induction hypothesis). This is the case iff $(\psi \rightarrow \chi) \in \Gamma^*$ (by [Proposition 10.2\(5\)](#)).
8. $\varphi \equiv \forall x \psi(x)$: Suppose that $\mathfrak{M}(\Gamma^*) \models \varphi$, then for every variable assignment s , $\mathfrak{M}(\Gamma^*), s \models \psi(x)$. Suppose to the contrary that $\forall x \psi(x) \notin \Gamma^*$. Then by [Proposition 10.2\(2\)](#), $\neg\forall x \psi(x) \in \Gamma^*$. By saturation, $(\exists x \neg\psi(x) \rightarrow \neg\psi(c)) \in \Gamma^*$ for some c , so by [Proposition 10.2\(1\)](#), $\neg\psi(c) \in \Gamma^*$. Since Γ^* is consistent, $\psi(c) \notin \Gamma^*$. By induction hypothesis, $\mathfrak{M}(\Gamma^*) \not\models \psi(c)$. Therefore, if s' is the variable assignment such that $s'(x) = c$, then, by [Proposition 6.44](#), $\mathfrak{M}(\Gamma^*), s' \not\models \psi(x)$, contradicting the earlier result that $\mathfrak{M}(\Gamma^*), s \models \psi(x)$ for all s . Thus, we have $\varphi \in \Gamma^*$.

Conversely, suppose that $\forall x \psi(x) \in \Gamma^*$. By [Theorems 8.27](#) and [9.27](#) together with [Proposition 10.2\(1\)](#), $\psi(t) \in \Gamma^*$ for every term $t \in |\mathfrak{M}(\Gamma^*)|$. By inductive hypothesis, $\mathfrak{M}(\Gamma^*) \models \psi(t)$ for every term $t \in |\mathfrak{M}(\Gamma^*)|$. Let s be the variable assignment with $s(x) = t$. Then $\mathfrak{M}(\Gamma^*), s \models \psi(x)$ for any such s , hence $\mathfrak{M}(\Gamma^*) \models \varphi$.

9. $\varphi \equiv \exists x \psi(x)$: First suppose that $\mathfrak{M}(\Gamma^*) \models \varphi$. By the definition of satisfaction, for some variable assignment s , $\mathfrak{M}(\Gamma^*), s \models \psi(x)$. The value $s(x)$ is some term $t \in |\mathfrak{M}(\Gamma^*)|$. Thus, $\mathfrak{M}(\Gamma^*) \models \psi(t)$, and by our induction hypothesis, $\psi(t) \in \Gamma^*$. By [Theorems 8.27](#) and [9.27](#) we have $\Gamma^* \vdash \exists x \psi(x)$. Then, by [Proposition 10.2\(1\)](#), we can conclude that $\varphi \in \Gamma^*$.

Conversely, suppose that $\exists x \psi(x) \in \Gamma^*$. Because Γ^* is saturated, $(\exists x \psi(x) \rightarrow \psi(c)) \in \Gamma^*$. By [Propositions 8.24](#) and [9.24](#) together with [Proposition 10.2\(1\)](#),

10.7. IDENTITY

$\psi(c) \in \Gamma^*$. By inductive hypothesis, $\mathfrak{M}(\Gamma^*) \models \psi(c)$. Now consider the variable assignment with $s(x) = c^{\mathfrak{M}(\Gamma^*)}$. Then $\mathfrak{M}(\Gamma^*), s \models \psi(x)$. By definition of satisfaction, $\mathfrak{M}(\Gamma^*) \models \exists x \psi(x)$.

□

10.7 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 10.10. Let Γ^* be a maximally consistent set of sentences in \mathcal{L} . We define the relation \approx on the set of closed terms of \mathcal{L} by

$$t \approx t' \quad \text{iff} \quad t = t' \in \Gamma^*$$

Proposition 10.11. *The relation \approx has the following properties:*

1. \approx is reflexive.
2. \approx is symmetric.
3. \approx is transitive.
4. If $t \approx t'$, f is a function symbol, and $t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n$ are terms, then
$$f(t_1, \dots, t_{i-1}, t, t_{i+1}, \dots, t_n) \approx f(t_1, \dots, t_{i-1}, t', t_{i+1}, \dots, t_n).$$
5. If $t \approx t'$, R is a predicate symbol, and $t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n$ are terms, then

$$R(t_1, \dots, t_{i-1}, t, t_{i+1}, \dots, t_n) \in \Gamma^* \quad \text{iff} \quad R(t_1, \dots, t_{i-1}, t', t_{i+1}, \dots, t_n) \in \Gamma^*.$$

Proof. Since Γ^* is maximally consistent, $t = t' \in \Gamma^*$ iff $\Gamma^* \vdash t = t'$. Thus it is enough to show the following:

1. $\Gamma^* \vdash t = t$ for all terms t .
2. If $\Gamma^* \vdash t = t'$ then $\Gamma^* \vdash t' = t$.
3. If $\Gamma^* \vdash t = t'$ and $\Gamma^* \vdash t' = t''$, then $\Gamma^* \vdash t = t''$.

4. If $\Gamma^* \vdash t = t'$, then

$$\Gamma^* \vdash f(t_1, \dots, t_{i-1}, t, t_{i+1}, \dots, t_n) = f(t_1, \dots, t_{i-1}, t', t_{i+1}, \dots, t_n)$$

for every n -place function symbol f and terms $t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n$.

5. If $\Gamma^* \vdash t = t'$ and $\Gamma^* \vdash R(t_1, \dots, t_{i-1}, t, t_{i+1}, \dots, t_n)$, then $\Gamma^* \vdash R(t_1, \dots, t_{i-1}, t', t_{i+1}, \dots, t_n)$
for every n -place predicate symbol R and terms $t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_n$.

□

Definition 10.12. Suppose Γ^* is a maximally consistent set in a language \mathcal{L} , t is a term, and \approx as in the previous definition. Then:

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

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

Definition 10.13. Let $\mathfrak{M} = \mathfrak{M}(\Gamma^*)$ be the term model for Γ^* . Then \mathfrak{M}/\approx is the following structure:

1. $|\mathfrak{M}/\approx| = \text{Trm}(\mathcal{L})/\approx$.
2. $c^{\mathfrak{M}/\approx} = [c]_{\approx}$
3. $f^{\mathfrak{M}/\approx}([t_1]_{\approx}, \dots, [t_n]_{\approx}) = [f(t_1, \dots, t_n)]_{\approx}$
4. $\langle [t_1]_{\approx}, \dots, [t_n]_{\approx} \rangle \in R^{\mathfrak{M}/\approx}$ iff $\mathfrak{M} \models R(t_1, \dots, t_n)$.

Note that we have defined $f^{\mathfrak{M}/\approx}$ and $R^{\mathfrak{M}/\approx}$ for elements of $\text{Trm}(\mathcal{L})/\approx$ by referring to them as $[t]_{\approx}$, i.e., via *representatives* $t \in [t]_{\approx}$. We have to make sure that these definitions do not depend on the choice of these representatives, i.e., that for some other choices t' which determine the same equivalence classes ($[t]_{\approx} = [t']_{\approx}$), the definitions yield the same result. For instance, if R is a one-place predicate symbol, the last clause of the definition says that $[t]_{\approx} \in R^{\mathfrak{M}/\approx}$ iff $\mathfrak{M} \models R(t)$. If for some other term t' with $t \approx t'$, $\mathfrak{M} \not\models R(t')$, then the definition would require $[t']_{\approx} \notin R^{\mathfrak{M}/\approx}$. If $t \approx t'$, then $[t]_{\approx} = [t']_{\approx}$, but we can't have both $[t]_{\approx} \in R^{\mathfrak{M}/\approx}$ and $[t]_{\approx} \notin R^{\mathfrak{M}/\approx}$. However, [Proposition 10.11](#) guarantees that this cannot happen.

Proposition 10.14. \mathfrak{M}/\approx is well defined, i.e., if $t_1, \dots, t_n, t'_1, \dots, t'_n$ are terms, and $t_i \approx t'_i$ then

1. $[f(t_1, \dots, t_n)]_{\approx} = [f(t'_1, \dots, t'_n)]_{\approx}$, i.e.,

$$f(t_1, \dots, t_n) \approx f(t'_1, \dots, t'_n)$$

and

10.8. THE COMPLETENESS THEOREM

2. $\mathfrak{M} \models R(t_1, \dots, t_n)$ iff $\mathfrak{M} \models R(t'_1, \dots, t'_n)$, i.e.,

$$R(t_1, \dots, t_n) \in \Gamma^* \text{ iff } R(t'_1, \dots, t'_n) \in \Gamma^*.$$

Proof. Follows from [Proposition 10.11](#) by induction on n . □

Lemma 10.15. $\mathfrak{M}/\approx \models \varphi$ iff $\varphi \in \Gamma^*$ for all sentences φ .

Proof. By induction on φ , just as in the proof of [Lemma 10.9](#). The only case that needs additional attention is when $\varphi \equiv t = t'$.

$$\begin{aligned} \mathfrak{M}/\approx \models t = t' &\text{ iff } [t]_{\approx} = [t']_{\approx} \text{ (by definition of } \mathfrak{M}/\approx) \\ &\text{ iff } t \approx t' \text{ (by definition of } [t]_{\approx}) \\ &\text{ iff } t = t' \in \Gamma^* \text{ (by definition of } \approx). \end{aligned}$$

□

Note that while $\mathfrak{M}(\Gamma^*)$ is always enumerable and infinite, \mathfrak{M}/\approx may be finite, since it may turn out that there are only finitely many classes $[t]_{\approx}$. This is to be expected, since Γ may contain sentences which require any structure in which they are true to be finite. For instance, $\forall x \forall y x = y$ is a consistent sentence, but is satisfied only in structures with a domain that contains exactly one element.

10.8 The Completeness Theorem

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

Theorem 10.16 (Completeness Theorem). *Let Γ be a set of sentences. If Γ is consistent, it is satisfiable.*

Proof. Suppose Γ is consistent. By [Lemma 10.7](#), there is a $\Gamma^* \supseteq \Gamma$ which is maximally consistent and saturated. If Γ does not contain $=$, then by [Lemma 10.9](#), $\mathfrak{M}(\Gamma^*) \models \varphi$ iff $\varphi \in \Gamma^*$. From this it follows in particular that for all $\varphi \in \Gamma$, $\mathfrak{M}(\Gamma^*) \models \varphi$, so Γ is satisfiable. If Γ does contain $=$, then by [Lemma 10.15](#), $\mathfrak{M}/\approx \models \varphi$ iff $\varphi \in \Gamma^*$ for all sentences φ . In particular, $\mathfrak{M}/\approx \models \varphi$ for all $\varphi \in \Gamma$, so Γ is satisfiable. □

Corollary 10.17 (Completeness Theorem, Second Version). *For all Γ and φ sentences: if $\Gamma \models \varphi$ then $\Gamma \vdash \varphi$.*

Proof. Note that the Γ 's in [Corollary 10.17](#) and [Theorem 10.16](#) are universally quantified. To make sure we do not confuse ourselves, let us restate [Theorem 10.16](#) using a different variable: for any set of sentences Δ , if Δ is consistent, it is satisfiable. By contraposition, if Δ is not satisfiable, then Δ is inconsistent. We will use this to prove the corollary.

Suppose that $\Gamma \models \varphi$. Then $\Gamma \cup \{\neg\varphi\}$ is unsatisfiable by [Proposition 6.49](#). Taking $\Gamma \cup \{\neg\varphi\}$ as our Δ , the previous version of [Theorem 10.16](#) gives us that $\Gamma \cup \{\neg\varphi\}$ is inconsistent. By [Propositions 8.13](#) and [9.13](#), $\Gamma \vdash \varphi$. \square

10.9 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 10.18. A set Γ of formulas is *finitely satisfiable* if and only if every finite $\Gamma_0 \subseteq \Gamma$ is satisfiable.

Theorem 10.19 (Compactness Theorem). *The following hold for any sentences Γ and φ :*

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

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

Now suppose that Γ is finitely satisfiable. Then every finite subset $\Gamma_0 \subseteq \Gamma$ is satisfiable. By soundness ([Theorems 8.29](#) and [9.28](#)), every finite subset is consistent. Then Γ itself must be consistent. For assume it is not, i.e., $\Gamma \vdash \perp$. But derivations are finite, and so already some finite subset $\Gamma_0 \subseteq \Gamma$ must be inconsistent (cf. [Propositions 8.15](#) and [9.15](#)). But we just showed they are all consistent, a contradiction. Now by completeness, since Γ is consistent, it is satisfiable. \square

Example 10.20. In every model \mathfrak{M} of a theory Γ , each term t of course picks out an element of $|\mathfrak{M}|$. Can we guarantee that it is also true that every element of $|\mathfrak{M}|$ is picked out by some term or other? In other words, are there theories Γ all models of which are covered? The compactness theorem shows that this is not the case if Γ has infinite models. Here's how to see this: Let \mathfrak{M} be

10.9. THE COMPACTNESS THEOREM

an infinite model of Γ , and let c be a constant symbol not in the language of Γ . Let Δ be the set of all sentences $c \neq t$ for t a term in the language \mathcal{L} of Γ , i.e.,

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

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 $a \neq \text{Val}^{\mathfrak{M}}(t)$ for all t occurring 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 $\text{Val}^{\mathfrak{M}}(c) \neq \text{Val}^{\mathfrak{M}}(t)$ for all terms t of \mathcal{L} .

Example 10.21. 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 \mathfrak{Q} and the obvious interpretations. Γ is the set of all sentences of \mathcal{L} true about the rational numbers. Of course, in \mathfrak{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 $\{0 < c\} \cup \{c < (1 \div \bar{k}) : k \in \mathbb{Z}^+\}$ (where $\bar{k} = (1 + (1 + \cdots + (1 + 1) \dots))$ with k 1's). For any finite subset Δ_0 of Δ there is a K such that all the sentences $c < \bar{k}$ 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 10.22. We know that first-order logic with identity predicate can express that the size of the domain must have some minimal size: The sentence $\varphi_{\geq n}$ (which says “there are at least n distinct objects”) is true only in structures where $|\mathfrak{M}|$ has at least n objects. So if we take

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

then any model of Δ must be infinite. Thus, we can guarantee that a theory only has infinite models by adding Δ to it: the models of $\Gamma \cup \Delta$ are all and only the infinite models of Γ .

So first-order logic can express infinitude. The compactness theorem shows that it cannot express finitude, however. For suppose some set of sentences Λ were satisfied in all and only finite structures. Then $\Delta \cup \Lambda$ is finitely satisfiable. Why? Suppose $\Delta' \cup \Lambda' \subseteq \Delta \cup \Lambda$ is finite with $\Delta' \subseteq \Delta$ and $\Lambda' \subseteq \Lambda$. Let n be the largest number such that $\Lambda_{\geq n} \in \Lambda'$. 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.

10.10 The Löwenheim-Skolem Theorem

The Löwenheim-Skolem Theorem says that if a theory has an infinite model, then it also has a model that is at most denumerable. An immediate consequence of this fact is that first-order logic cannot express that the size of a structure is non-enumerable: any sentence or set of sentences satisfied in all non-enumerable structures is also satisfied in some denumerable structure.

Theorem 10.23. *If Γ is consistent then it has a denumerable model, i.e., it is satisfiable in a structure whose domain is either finite or infinite but enumerable.*

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

Theorem 10.24. *If Γ is consistent set of sentences in the language of first-order logic without identity, then it has a denumerable model, i.e., it is satisfiable in a structure whose domain is infinite and enumerable.*

Proof. If Γ is consistent and contains no sentences in which identity appears, then the structure \mathfrak{M} delivered by the proof of the completeness theorem has a domain $|\mathfrak{M}|$ whose cardinality is identical to that of the set of the terms of the language \mathcal{L} . So \mathfrak{M} is denumerably infinite. \square

Example 10.25 (Skolem's Paradox). Zermelo-Fraenkel set theory **ZFC** is a very powerful framework in which practically all mathematical statements can be expressed, including facts about the sizes of sets. So for instance, **ZFC** can prove that the set \mathbb{R} of real numbers is non-enumerable, it can prove Cantor's Theorem that the power set of any set is larger than the set itself, etc. If **ZFC** is consistent, its models are all infinite, and moreover, they all contain elements about which the theory says that they are non-enumerable, such as the element that makes true the theorem of **ZFC** that the power set of the natural numbers exists. By the Löwenheim-Skolem Theorem, **ZFC** also has enumerable models—models that contain “non-enumerable” sets but which themselves are enumerable.

Problems

Problem 10.1. Complete the proof of [Proposition 10.2](#).

Problem 10.2. Complete the proof of [Lemma 10.9](#).

10.10. THE LÖWENHEIM-SKOLEM THEOREM

Problem 10.3. Complete the proof of [Proposition 10.11](#).

Problem 10.4. Use [Corollary 10.17](#) to prove [Theorem 10.16](#), thus showing that the two formulations of the completeness theorem are equivalent.

Problem 10.5. In order for a derivation system to be complete, its rules must be strong enough to prove every unsatisfiable set inconsistent. Which of the rules of **LK** were necessary to prove completeness? Are any of these rules not used anywhere in the proof? In order to answer these questions, make a list or diagram that shows which of the rules of **LK** were used in which results that lead up to the proof of [Theorem 10.16](#). Be sure to note any tacit uses of rules in these proofs.

Problem 10.6. Prove (1) of [Theorem 10.19](#).

Problem 10.7. In the standard model of arithmetic \mathfrak{N} , there is no element $k \in |\mathfrak{N}|$ which satisfies every formula $\bar{n} < x$ (where \bar{n} is $0'\dots'$ 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 $\bar{n} < x$.

Chapter 11

Beyond First-order Logic

This chapter, adapted from Jeremy Avigad’s logic notes, gives the briefest of glimpses into which other logical systems there are. It is intended as a chapter suggesting further topics for study in a course that does not cover them. Each one of the topics mentioned here will—hopefully—eventually receive its own part-level treatment in the Open Logic Project.

11.1 Overview

First-order logic is not the only system of logic of interest: there are many extensions and variations of first-order logic. A logic typically consists of the formal specification of a language, usually, but not always, a deductive system, and usually, but not always, an intended semantics. But the technical use of the term raises an obvious question: what do logics that are not first-order logic have to do with the word “logic,” used in the intuitive or philosophical sense? All of the systems described below are designed to model reasoning of some form or another; can we say what makes them logical?

No easy answers are forthcoming. The word “logic” is used in different ways and in different contexts, and the notion, like that of “truth,” has been analyzed from numerous philosophical stances. For example, one might take the goal of logical reasoning to be the determination of which statements are necessarily true, true a priori, true independent of the interpretation of the nonlogical terms, true by virtue of their form, or true by linguistic convention; and each of these conceptions requires a good deal of clarification. Even if one restricts one’s attention to the kind of logic used in mathematics, there is little agreement as to its scope. For example, in the *Principia Mathematica*, Russell and Whitehead tried to develop mathematics on the basis of logic, in the *logician* tradition begun by Frege. Their system of logic was a form of higher-type

11.2. MANY-SORTED LOGIC

logic similar to the one described below. In the end they were forced to introduce axioms which, by most standards, do not seem purely logical (notably, the axiom of infinity, and the axiom of reducibility), but one might nonetheless hold that some forms of higher-order reasoning should be accepted as logical. In contrast, Quine, whose ontology does not admit “propositions” as legitimate objects of discourse, argues that second-order and higher-order logic are really manifestations of set theory in sheep’s clothing; in other words, systems involving quantification over predicates are not purely logical.

For now, it is best to leave such philosophical issues for a rainy day, and simply think of the systems below as formal idealizations of various kinds of reasoning, logical or otherwise.

11.2 Many-Sorted Logic

In first-order logic, variables and quantifiers range over a single domain. But it is often useful to have multiple (disjoint) domains: for example, you might want to have a domain of numbers, a domain of geometric objects, a domain of functions from numbers to numbers, a domain of abelian groups, and so on.

Many-sorted logic provides this kind of framework. One starts with a list of “sorts”—the “sort” of an object indicates the “domain” it is supposed to inhabit. One then has variables and quantifiers for each sort, and (usually) an identity predicate for each sort. Functions and relations are also “typed” by the sorts of objects they can take as arguments. Otherwise, one keeps the usual rules of first-order logic, with versions of the quantifier-rules repeated for each sort.

For example, to study international relations we might choose a language with two sorts of objects, French citizens and German citizens. We might have a unary relation, “drinks wine,” for objects of the first sort; another unary relation, “eats wurst,” for objects of the second sort; and a binary relation, “forms a multinational married couple,” which takes two arguments, where the first argument is of the first sort and the second argument is of the second sort. If we use variables a, b, c to range over French citizens and x, y, z to range over German citizens, then

$$\forall a \forall x [(MarriedTo(a, x) \rightarrow (DrinksWine(a) \vee \neg EatsWurst(x)))]$$

asserts that if any French person is married to a German, either the French person drinks wine or the German doesn’t eat wurst.

Many-sorted logic can be embedded in first-order logic in a natural way, by lumping all the objects of the many-sorted domains together into one first-order domain, using unary predicate symbols to keep track of the sorts, and relativizing quantifiers. For example, the first-order language corresponding to the example above would have unary predicate symbols “*German*” and

“*French*,” in addition to the other relations described, with the sort requirements erased. A sorted quantifier $\forall x \varphi$, where x is a variable of the German sort, translates to

$$\forall x (German(x) \rightarrow \varphi).$$

We need to add axioms that insure that the sorts are separate—e.g., $\forall x \neg (German(x) \wedge French(x))$ —as well as axioms that guarantee that “drinks wine” only holds of objects satisfying the predicate *French*(x), etc. With these conventions and axioms, it is not difficult to show that many-sorted sentences translate to first-order sentences, and many-sorted derivations translate to first-order derivations. Also, many-sorted structures “translate” to corresponding first-order structures and vice-versa, so we also have a completeness theorem for many-sorted logic.

11.3 Second-Order logic

The language of second-order logic allows one to quantify not just over a domain of individuals, but over relations on that domain as well. Given a first-order language \mathcal{L} , for each k one adds variables R which range over k -ary relations, and allows quantification over those variables. If R is a variable for a k -ary relation, and t_1, \dots, t_k are ordinary (first-order) terms, $R(t_1, \dots, t_k)$ is an atomic formula. Otherwise, the set of formulas is defined just as in the case of first-order logic, with additional clauses for second-order quantification. Note that we only have the identity predicate for first-order terms: if R and S are relation variables of the same arity k , we can define $R = S$ to be an abbreviation for

$$\forall x_1 \dots \forall x_k (R(x_1, \dots, x_k) \leftrightarrow S(x_1, \dots, x_k)).$$

The rules for second-order logic simply extend the quantifier rules to the new second order variables. Here, however, one has to be a little bit careful to explain how these variables interact with the predicate symbols of \mathcal{L} , and with formulas of \mathcal{L} more generally. At the bare minimum, relation variables count as terms, so one has inferences of the form

$$\varphi(R) \vdash \exists R \varphi(R)$$

But if \mathcal{L} is the language of arithmetic with a constant relation symbol $<$, one would also expect the following inference to be valid:

$$x < y \vdash \exists R R(x, y)$$

or for a given formula φ ,

$$\varphi(x_1, \dots, x_k) \vdash \exists R R(x_1, \dots, x_k)$$

11.3. SECOND-ORDER LOGIC

More generally, we might want to allow inferences of the form

$$\varphi[\lambda\vec{x}. \psi(\vec{x})/R] \vdash \exists R \varphi$$

where $\varphi[\lambda\vec{x}. \psi(\vec{x})/R]$ denotes the result of replacing every atomic formula of the form Rt_1, \dots, t_k in φ by $\psi(t_1, \dots, t_k)$. This last rule is equivalent to having a *comprehension schema*, i.e., an axiom of the form

$$\exists R \forall x_1, \dots, x_k (\varphi(x_1, \dots, x_k) \leftrightarrow R(x_1, \dots, x_k)),$$

one for each formula φ in the second-order language, in which R is not a free variable. (Exercise: show that if R is allowed to occur in φ , this schema is inconsistent!)

When logicians refer to the “axioms of second-order logic” they usually mean the minimal extension of first-order logic by second-order quantifier rules together with the comprehension schema. But it is often interesting to study weaker subsystems of these axioms and rules. For example, note that in its full generality the axiom schema of comprehension is *impredicative*: it allows one to assert the existence of a relation $R(x_1, \dots, x_k)$ that is “defined” by a formula with second-order quantifiers; and these quantifiers range over the set of all such relations—a set which includes R itself! Around the turn of the twentieth century, a common reaction to Russell’s paradox was to lay the blame on such definitions, and to avoid them in developing the foundations of mathematics. If one prohibits the use of second-order quantifiers in the formula φ , one has a *predicative* form of comprehension, which is somewhat weaker.

From the semantic point of view, one can think of a second-order structure as consisting of a first-order structure for the language, coupled with a set of relations on the domain over which the second-order quantifiers range (more precisely, for each k there is a set of relations of arity k). Of course, if comprehension is included in the proof system, then we have the added requirement that there are enough relations in the “second-order part” to satisfy the comprehension axioms—otherwise the proof system is not sound! One easy way to insure that there are enough relations around is to take the second-order part to consist of *all* the relations on the first-order part. Such a structure is called *full*, and, in a sense, is really the “intended structure” for the language. If we restrict our attention to full structures we have what is known as the *full* second-order semantics. In that case, specifying a structure boils down to specifying the first-order part, since the contents of the second-order part follow from that implicitly.

To summarize, there is some ambiguity when talking about second-order logic. In terms of the proof system, one might have in mind either

1. A “minimal” second-order proof system, together with some comprehension axioms.

2. The “standard” second-order proof system, with full comprehension.

In terms of the semantics, one might be interested in either

1. The “weak” semantics, where a structure consists of a first-order part, together with a second-order part big enough to satisfy the comprehension axioms.
2. The “standard” second-order semantics, in which one considers full structures only.

When logicians do not specify the proof system or the semantics they have in mind, they are usually referring to the second item on each list. The advantage to using this semantics is that, as we will see, it gives us categorical descriptions of many natural mathematical structures; at the same time, the proof system is quite strong, and sound for this semantics. The drawback is that the proof system is *not* complete for the semantics; in fact, *no* effectively given proof system is complete for the full second-order semantics. On the other hand, we will see that the proof system *is* complete for the weakened semantics; this implies that if a sentence is not provable, then there is *some* structure, not necessarily the full one, in which it is false.

The language of second-order logic is quite rich. One can identify unary relations with subsets of the domain, and so in particular you can quantify over these sets; for example, one can express induction for the natural numbers with a single axiom

$$\forall R ((R(0) \wedge \forall x (R(x) \rightarrow R(x')))) \rightarrow \forall x R(x)).$$

If one takes the language of arithmetic to have symbols $0, /, +, \times$ and $<$, one can add the following axioms to describe their behavior:

1. $\forall x \neg x' = 0$
2. $\forall x \forall y (s(x) = s(y) \rightarrow x = y)$
3. $\forall x (x + 0) = x$
4. $\forall x \forall y (x + y') = (x + y)'$
5. $\forall x (x \times 0) = 0$
6. $\forall x \forall y (x \times y') = ((x \times y) + x)$
7. $\forall x \forall y (x < y \leftrightarrow \exists z y = (x + z'))$

It is not difficult to show that these axioms, together with the axiom of induction above, provide a categorical description of the structure \mathfrak{N} , the standard model of arithmetic, provided we are using the full second-order semantics. Given any structure \mathfrak{A} in which these axioms are true, define a function f from

11.3. SECOND-ORDER LOGIC

\mathbb{N} to the domain of \mathfrak{A} using ordinary recursion on \mathbb{N} , so that $f(0) = o^{\mathfrak{A}}$ and $f(x+1) = \iota^{\mathfrak{A}}(f(x))$. Using ordinary induction on \mathbb{N} and the fact that axioms (1) and (2) hold in \mathfrak{A} , we see that f is injective. To see that f is surjective, let P be the set of elements of $|\mathfrak{A}|$ that are in the range of f . Since \mathfrak{A} is full, P is in the second-order domain. By the construction of f , we know that $o^{\mathfrak{A}}$ is in P , and that P is closed under $\iota^{\mathfrak{A}}$. The fact that the induction axiom holds in \mathfrak{A} (in particular, for P) guarantees that P is equal to the entire first-order domain of \mathfrak{A} . This shows that f is a bijection. Showing that f is a homomorphism is no more difficult, using ordinary induction on \mathbb{N} repeatedly.

In set-theoretic terms, a function is just a special kind of relation; for example, a unary function f can be identified with a binary relation R satisfying $\forall x \exists y R(x, y)$. As a result, one can quantify over functions too. Using the full semantics, one can then define the class of infinite structures to be the class of structures \mathfrak{A} for which there is an injective function from the domain of \mathfrak{A} to a proper subset of itself:

$$\exists f (\forall x \forall y (f(x) = f(y) \rightarrow x = y) \wedge \exists y \forall x f(x) \neq y).$$

The negation of this sentence then defines the class of finite structures.

In addition, one can define the class of well-orderings, by adding the following to the definition of a linear ordering:

$$\forall P (\exists x P(x) \rightarrow \exists x (P(x) \wedge \forall y (y < x \rightarrow \neg P(y)))).$$

This asserts that every non-empty set has a least element, modulo the identification of “set” with “one-place relation”. For another example, one can express the notion of connectedness for graphs, by saying that there is no non-trivial separation of the vertices into disconnected parts:

$$\neg \exists A (\exists x A(x) \wedge \exists y \neg A(y) \wedge \forall w \forall z ((A(w) \wedge \neg A(z)) \rightarrow \neg R(w, z))).$$

For yet another example, you might try as an exercise to define the class of finite structures whose domain has even size. More strikingly, one can provide a categorical description of the real numbers as a complete ordered field containing the rationals.

In short, second-order logic is much more expressive than first-order logic. That’s the good news; now for the bad. We have already mentioned that there is no effective proof system that is complete for the full second-order semantics. For better or for worse, many of the properties of first-order logic are absent, including compactness and the Löwenheim-Skolem theorems.

On the other hand, if one is willing to give up the full second-order semantics in terms of the weaker one, then the minimal second-order proof system is complete for this semantics. In other words, if we read \vdash as “proves in the minimal system” and \models as “logically implies in the weaker semantics”, we can show that whenever $\Gamma \models \varphi$ then $\Gamma \vdash \varphi$. If one wants to include specific

comprehension axioms in the proof system, one has to restrict the semantics to second-order structures that satisfy these axioms: for example, if Δ consists of a set of comprehension axioms (possibly all of them), we have that if $\Gamma \cup \Delta \models \varphi$, then $\Gamma \cup \Delta \vdash \varphi$. In particular, if φ is not provable using the comprehension axioms we are considering, then there is a model of $\neg\varphi$ in which these comprehension axioms nonetheless hold.

The easiest way to see that the completeness theorem holds for the weaker semantics is to think of second-order logic as a many-sorted logic, as follows. One sort is interpreted as the ordinary “first-order” domain, and then for each k we have a domain of “relations of arity k .” We take the language to have built-in relation symbols “ $true_k(R, x_1, \dots, x_k)$ ” which is meant to assert that R holds of x_1, \dots, x_k , where R is a variable of the sort “ k -ary relation” and x_1, \dots, x_k are objects of the first-order sort.

With this identification, the weak second-order semantics is essentially the usual semantics for many-sorted logic; and we have already observed that many-sorted logic can be embedded in first-order logic. Modulo the translations back and forth, then, the weaker conception of second-order logic is really a form of first-order logic in disguise, where the domain contains both “objects” and “relations” governed by the appropriate axioms.

11.4 Higher-Order logic

Passing from first-order logic to second-order logic enabled us to talk about sets of objects in the first-order domain, within the formal language. Why stop there? For example, third-order logic should enable us to deal with sets of sets of objects, or perhaps even sets which contain both objects and sets of objects. And fourth-order logic will let us talk about sets of objects of that kind. As you may have guessed, one can iterate this idea arbitrarily.

In practice, higher-order logic is often formulated in terms of functions instead of relations. (Modulo the natural identifications, this difference is inessential.) Given some basic “sorts” A, B, C, \dots (which we will now call “types”), we can create new ones by stipulating

If σ and τ are finite types then so is $\sigma \rightarrow \tau$.

Think of types as syntactic “labels,” which classify the objects we want in our domain; $\sigma \rightarrow \tau$ describes those objects that are functions which take objects of type σ to objects of type τ . For example, we might want to have a type Ω of truth values, “true” and “false,” and a type \mathbb{N} of natural numbers. In that case, you can think of objects of type $\mathbb{N} \rightarrow \Omega$ as unary relations, or subsets of \mathbb{N} ; objects of type $\mathbb{N} \rightarrow \mathbb{N}$ are functions from natural numbers to natural numbers; and objects of type $(\mathbb{N} \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$ are “functionals,” that is, higher-type functions that take functions to numbers.

11.4. HIGHER-ORDER LOGIC

As in the case of second-order logic, one can think of higher-order logic as a kind of many-sorted logic, where there is a sort for each type of object we want to consider. But it is usually clearer just to define the syntax of higher-type logic from the ground up. For example, we can define a set of finite types inductively, as follows:

1. \mathbb{N} is a finite type.
2. If σ and τ are finite types, then so is $\sigma \rightarrow \tau$.
3. If σ and τ are finite types, so is $\sigma \times \tau$.

Intuitively, \mathbb{N} denotes the type of the natural numbers, $\sigma \rightarrow \tau$ denotes the type of functions from σ to τ , and $\sigma \times \tau$ denotes the type of pairs of objects, one from σ and one from τ . We can then define a set of terms inductively, as follows:

1. For each type σ , there is a stock of variables x, y, z, \dots of type σ
2. o is a term of type \mathbb{N}
3. S (successor) is a term of type $\mathbb{N} \rightarrow \mathbb{N}$
4. If s is a term of type σ , and t is a term of type $\mathbb{N} \rightarrow (\sigma \rightarrow \sigma)$, then R_{st} is a term of type $\mathbb{N} \rightarrow \sigma$
5. If s is a term of type $\tau \rightarrow \sigma$ and t is a term of type τ , then $s(t)$ is a term of type σ
6. If s is a term of type σ and x is a variable of type τ , then $\lambda x. s$ is a term of type $\tau \rightarrow \sigma$.
7. If s is a term of type σ and t is a term of type τ , then $\langle s, t \rangle$ is a term of type $\sigma \times \tau$.
8. If s is a term of type $\sigma \times \tau$ then $p_1(s)$ is a term of type σ and $p_2(s)$ is a term of type τ .

Intuitively, R_{st} denotes the function defined recursively by

$$\begin{aligned} R_{st}(0) &= s \\ R_{st}(x+1) &= t(x, R_{st}(x)), \end{aligned}$$

$\langle s, t \rangle$ denotes the pair whose first component is s and whose second component is t , and $p_1(s)$ and $p_2(s)$ denote the first and second elements ("projections") of s . Finally, $\lambda x. s$ denotes the function f defined by

$$f(x) = s$$

for any x of type σ ; so item (6) gives us a form of comprehension, enabling us to define functions using terms. Formulas are built up from identity predicate statements $s = t$ between terms of the same type, the usual propositional connectives, and higher-type quantification. One can then take the axioms of the system to be the basic equations governing the terms defined above, together with the usual rules of logic with quantifiers and identity predicate.

If one augments the finite type system with a type Ω of truth values, one has to include axioms which govern its use as well. In fact, if one is clever, one can get rid of complex formulas entirely, replacing them with terms of type Ω ! The proof system can then be modified accordingly. The result is essentially the *simple theory of types* set forth by Alonzo Church in the 1930s.

As in the case of second-order logic, there are different versions of higher-type semantics that one might want to use. In the full version, variables of type $\sigma \rightarrow \tau$ range over the set of *all* functions from the objects of type σ to objects of type τ . As you might expect, this semantics is too strong to admit a complete, effective proof system. But one can consider a weaker semantics, in which a structure consists of sets of elements T_τ for each type τ , together with appropriate operations for application, projection, etc. If the details are carried out correctly, one can obtain completeness theorems for the kinds of proof systems described above.

Higher-type logic is attractive because it provides a framework in which we can embed a good deal of mathematics in a natural way: starting with \mathbb{N} , one can define real numbers, continuous functions, and so on. It is also particularly attractive in the context of intuitionistic logic, since the types have clear “constructive” interpretations. In fact, one can develop constructive versions of higher-type semantics (based on intuitionistic, rather than classical logic) that clarify these constructive interpretations quite nicely, and are, in many ways, more interesting than the classical counterparts.

11.5 Intuitionistic Logic

In contrast to second-order and higher-order logic, intuitionistic first-order logic represents a restriction of the classical version, intended to model a more “constructive” kind of reasoning. The following examples may serve to illustrate some of the underlying motivations.

Suppose someone came up to you one day and announced that they had determined a natural number x , with the property that if x is prime, the Riemann hypothesis is true, and if x is composite, the Riemann hypothesis is false. Great news! Whether the Riemann hypothesis is true or not is one of the big open questions of mathematics, and here they seem to have reduced the problem to one of calculation, that is, to the determination of whether a specific number is prime or not.

11.5. INTUITIONISTIC LOGIC

What is the magic value of x ? They describe it as follows: x is the natural number that is equal to 7 if the Riemann hypothesis is true, and 9 otherwise.

Angrily, you demand your money back. From a classical point of view, the description above does in fact determine a unique value of x ; but what you really want is a value of x that is given *explicitly*.

To take another, perhaps less contrived example, consider the following question. We know that it is possible to raise an irrational number to a rational power, and get a rational result. For example, $\sqrt{2}^2 = 2$. What is less clear is whether or not it is possible to raise an irrational number to an *irrational* power, and get a rational result. The following theorem answers this in the affirmative:

Theorem 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 certainly rational. So, in this case, let a be $\sqrt{2}^{\sqrt{2}}$, and let b be $\sqrt{2}$. \square

Does this constitute a valid proof? Most mathematicians feel that it does. But again, there is something a little bit unsatisfying here: we have proved the existence of a pair of real numbers with a certain property, without being able to say *which* pair of numbers it is. It is possible to prove the same result, but in such a way that the pair a, b is given in the proof: take $a = \sqrt{3}$ and $b = \log_3 4$. Then

$$a^b = \sqrt{3}^{\log_3 4} = 3^{1/2 \cdot \log_3 4} = (3^{\log_3 4})^{1/2} = 4^{1/2} = 2,$$

since $3^{\log_3 x} = x$.

Intuitionistic logic is designed to model a kind of reasoning where moves like the one in the first proof are disallowed. Proving the existence of an x satisfying $\varphi(x)$ means that you have to give a specific x , and a proof that it satisfies φ , like in the second proof. Proving that φ or ψ holds requires that you can prove one or the other.

Formally speaking, intuitionistic first-order logic is what you get if you omit restrict a proof system for first-order logic in a certain way. Similarly, there are intuitionistic versions of second-order or higher-order logic. From the mathematical point of view, these are just formal deductive systems, but, as already noted, they are intended to model a kind of mathematical reasoning. One can take this to be the kind of reasoning that is justified on a certain philosophical view of mathematics (such as Brouwer's intuitionism); one can take it to be a kind of mathematical reasoning which is more "concrete" and satisfying (along the lines of Bishop's constructivism); and one can argue

about whether or not the formal description captures the informal motivation. But whatever philosophical positions we may hold, we can study intuitionistic logic as a formally presented logic; and for whatever reasons, many mathematical logicians find it interesting to do so.

There is an informal constructive interpretation of the intuitionist connectives, usually known as the Brouwer-Heyting-Kolmogorov interpretation. It runs as follows: a proof of $\varphi \wedge \psi$ consists of a proof of φ paired with a proof of ψ ; a proof of $\varphi \vee \psi$ consists of either a proof of φ , or a proof of ψ , where we have explicit information as to which is the case; a proof of $\varphi \rightarrow \psi$ consists of a procedure, which transforms a proof of φ to a proof of ψ ; a proof of $\forall x \varphi(x)$ consists of a procedure which returns a proof of $\varphi(x)$ for any value of x ; and a proof of $\exists x \varphi(x)$ consists of a value of x , together with a proof that this value satisfies φ . One can describe the interpretation in computational terms known as the “Curry-Howard isomorphism” or the “formulas-as-types paradigm”: think of a formula as specifying a certain kind of data type, and proofs as computational objects of these data types that enable us to see that the corresponding formula is true.

Intuitionistic logic is often thought of as being classical logic “minus” the law of the excluded middle. This following theorem makes this more precise.

Theorem 11.2. *Intuitionistically, the following axiom schemata are equivalent:*

1. $(\varphi \rightarrow \perp) \rightarrow \neg\varphi$.
2. $\varphi \vee \neg\varphi$
3. $\neg\neg\varphi \rightarrow \varphi$

Obtaining instances of one schema from either of the others is a good exercise in intuitionistic logic.

The first deductive systems for intuitionistic propositional logic, put forth as formalizations of Brouwer’s intuitionism, are due, independently, to Kolmogorov, Glivenko, and Heyting. The first formalization of intuitionistic first-order logic (and parts of intuitionist mathematics) is due to Heyting. Though a number of classically valid schemata are not intuitionistically valid, many are.

The *double-negation translation* describes an important relationship between classical and intuitionist logic. It is defined inductively follows (think of φ^N

11.5. INTUITIONISTIC LOGIC

as the “intuitionist” translation of the classical formula φ):

$$\begin{aligned}\varphi^N &\equiv \neg\neg\varphi \quad \text{for atomic formulas } \varphi \\ (\varphi \wedge \psi)^N &\equiv (\varphi^N \wedge \psi^N) \\ (\varphi \vee \psi)^N &\equiv \neg\neg(\varphi^N \vee \psi^N) \\ (\varphi \rightarrow \psi)^N &\equiv (\varphi^N \rightarrow \psi^N) \\ (\forall x \varphi)^N &\equiv \forall x \varphi^N \\ (\exists x \varphi)^N &\equiv \neg\neg\exists x \varphi^N\end{aligned}$$

Kolmogorov and Glivenko had versions of this translation for propositional logic; for predicate logic, it is due to Gödel and Gentzen, independently. We have

Theorem 11.3. 1. $\varphi \leftrightarrow \varphi^N$ is provable classically
 2. If φ is provable classically, then φ^N is provable intuitionistically.

We can now envision the following dialogue. Classical mathematician: “I’ve proved φ !” Intuitionist mathematician: “Your proof isn’t valid. What you’ve really proved is φ^N .” Classical mathematician: “Fine by me!” As far as the classical mathematician is concerned, the intuitionist is just splitting hairs, since the two are equivalent. But the intuitionist insists there is a difference.

Note that the above translation concerns pure logic only; it does not address the question as to what the appropriate *nonlogical* axioms are for classical and intuitionistic mathematics, or what the relationship is between them. But the following slight extension of the theorem above provides some useful information:

Theorem 11.4. If Γ proves φ classically, Γ^N proves φ^N intuitionistically.

In other words, if φ is provable from some hypotheses classically, then φ^N is provable from their double-negation translations.

To show that a sentence or propositional formula is intuitionistically valid, all you have to do is provide a proof. But how can you show that it is not valid? For that purpose, we need a semantics that is sound, and preferably complete. A semantics due to Kripke nicely fits the bill.

We can play the same game we did for classical logic: define the semantics, and prove soundness and completeness. It is worthwhile, however, to note the following distinction. In the case of classical logic, the semantics was the “obvious” one, in a sense implicit in the meaning of the connectives. Though one can provide some intuitive motivation for Kripke semantics, the latter does not offer the same feeling of inevitability. In addition, the notion of a classical structure is a natural mathematical one, so we can either take the notion of a structure to be a tool for studying classical first-order logic, or take

classical first-order logic to be a tool for studying mathematical structures. In contrast, Kripke structures can only be viewed as a logical construct; they don't seem to have independent mathematical interest.

A Kripke structure for a propositional language consists of a partial order $\text{Mod}(P)$ with a least element, and an “monotone” assignment of propositional variables to the elements of $\text{Mod}(P)$. The intuition is that the elements of $\text{Mod}(P)$ represent “worlds,” or “states of knowledge”; an element $p \geq q$ represents a “possible future state” of q ; and the propositional variables assigned to p are the propositions that are known to be true in state p . The forcing relation $\mathfrak{P}, p \Vdash \varphi$ then extends this relationship to arbitrary formulas in the language; read $\mathfrak{P}, p \Vdash \varphi$ as “ φ is true in state p .” The relationship is defined inductively, as follows:

1. $\mathfrak{P}, p \Vdash p_i$ iff p_i is one of the propositional variables assigned to p .
2. $\mathfrak{P}, p \nVdash \perp$.
3. $\mathfrak{P}, p \Vdash (\varphi \wedge \psi)$ iff $\mathfrak{P}, p \Vdash \varphi$ and $\mathfrak{P}, p \Vdash \psi$.
4. $\mathfrak{P}, p \Vdash (\varphi \vee \psi)$ iff $\mathfrak{P}, p \Vdash \varphi$ or $\mathfrak{P}, p \Vdash \psi$.
5. $\mathfrak{P}, p \Vdash (\varphi \rightarrow \psi)$ iff, whenever $q \geq p$ and $\mathfrak{P}, q \Vdash \varphi$, then $\mathfrak{P}, q \Vdash \psi$.

It is a good exercise to try to show that $\neg(p \wedge q) \rightarrow (\neg p \vee \neg q)$ is not intuitionistically valid, by cooking up a Kripke structure that provides a counterexample.

11.6 Modal Logics

Consider the following example of a conditional sentence:

If Jeremy is alone in that room, then he is drunk and naked and dancing on the chairs.

This is an example of a conditional assertion that may be materially true but nonetheless misleading, since it seems to suggest that there is a stronger link between the antecedent and conclusion other than simply that either the antecedent is false or the consequent true. That is, the wording suggests that the claim is not only true in this particular world (where it may be trivially true, because Jeremy is not alone in the room), but that, moreover, the conclusion *would have* been true *had* the antecedent been true. In other words, one can take the assertion to mean that the claim is true not just in this world, but in any “possible” world; or that it is *necessarily* true, as opposed to just true in this particular world.

Modal logic was designed to make sense of this kind of necessity. One obtains modal propositional logic from ordinary propositional logic by adding a

11.6. MODAL LOGICS

box operator; which is to say, if φ is a formula, so is $\Box\varphi$. Intuitively, $\Box\varphi$ asserts that φ is *necessarily* true, or true in any possible world. $\Diamond\varphi$ is usually taken to be an abbreviation for $\neg\Box\neg\varphi$, and can be read as asserting that φ is *possibly* true. Of course, modality can be added to predicate logic as well.

Kripke structures can be used to provide a semantics for modal logic; in fact, Kripke first designed this semantics with modal logic in mind. Rather than restricting to partial orders, more generally one has a set of “possible worlds,” P , and a binary “accessibility” relation $R(x, y)$ between worlds. Intuitively, $R(p, q)$ asserts that the world q is compatible with p ; i.e., if we are “in” world p , we have to entertain the possibility that the world could have been like q .

Modal logic is sometimes called an “intensional” logic, as opposed to an “extensional” one. The intended semantics for an extensional logic, like classical logic, will only refer to a single world, the “actual” one; while the semantics for an “intensional” logic relies on a more elaborate ontology. In addition to structuring necessity, one can use modality to structure other linguistic constructions, reinterpreting \Box and \Diamond according to the application. For example:

1. In provability logic, $\Box\varphi$ is read “ φ is provable” and $\Diamond\varphi$ is read “ φ is consistent.”
2. In epistemic logic, one might read $\Box\varphi$ as “I know φ ” or “I believe φ .”
3. In temporal logic, one can read $\Box\varphi$ as “ φ is always true” and $\Diamond\varphi$ as “ φ is sometimes true.”

One would like to augment logic with rules and axioms dealing with modality. For example, the system **S4** consists of the ordinary axioms and rules of propositional logic, together with the following axioms:

$$\begin{aligned}\Box(\varphi \rightarrow \psi) &\rightarrow (\Box\varphi \rightarrow \Box\psi) \\ \Box\varphi &\rightarrow \varphi \\ \Box\varphi &\rightarrow \Box\Box\varphi\end{aligned}$$

as well as a rule, “from φ conclude $\Box\varphi$.” **S5** adds the following axiom:

$$\Diamond\varphi \rightarrow \Box\Diamond\varphi$$

Variations of these axioms may be suitable for different applications; for example, **S5** is usually taken to characterize the notion of logical necessity. And the nice thing is that one can usually find a semantics for which the proof system is sound and complete by restricting the accessibility relation in the Kripke structures in natural ways. For example, **S4** corresponds to the class of Kripke structures in which the accessibility relation is reflexive and transitive. **S5** corresponds to the class of Kripke structures in which the accessibility

relation is *universal*, which is to say that every world is accessible from every other; so $\Box\varphi$ holds if and only if φ holds in every world.

11.7 Other Logics

As you may have gathered by now, it is not hard to design a new logic. You too can create your own a syntax, make up a deductive system, and fashion a semantics to go with it. You might have to be a bit clever if you want the proof system to be complete for the semantics, and it might take some effort to convince the world at large that your logic is truly interesting. But, in return, you can enjoy hours of good, clean fun, exploring your logic's mathematical and computational properties.

Recent decades have witnessed a veritable explosion of formal logics. Fuzzy logic is designed to model reasoning about vague properties. Probabilistic logic is designed to model reasoning about uncertainty. Default logics and nonmonotonic logics are designed to model defeasible forms of reasoning, which is to say, “reasonable” inferences that can later be overturned in the face of new information. There are epistemic logics, designed to model reasoning about knowledge; causal logics, designed to model reasoning about causal relationships; and even “deontic” logics, which are designed to model reasoning about moral and ethical obligations. Depending on whether the primary motivation for introducing these systems is philosophical, mathematical, or computational, you may find such creatures studies under the rubric of mathematical logic, philosophical logic, artificial intelligence, cognitive science, or elsewhere.

The list goes on and on, and the possibilities seem endless. We may never attain Leibniz' dream of reducing all of human reason to calculation—but that can't stop us from trying.

Part III

Model Theory

CHAPTER 11. BEYOND FIRST-ORDER LOGIC

Material on model theory is incomplete and experimental. It is currently simply an adaptation of Aldo Antonelli's notes on model theory, less those topics covered in the part on first-order logic (theories, completeness, compactness). It requires much more introduction, motivation, and explanation, as well as exercises, to be useful for a textbook. Andy Arana is at planning to work on this part specifically (issue #65).

Chapter 12

Basics of Model Theory

12.1 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 common 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 12.1. 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 12.2. If an \mathcal{L} -structure \mathfrak{M} is a reduct of an \mathcal{L}' -structure \mathfrak{M}' , then for all \mathcal{L} -sentences φ ,

$$\mathfrak{M} \models \varphi \text{ iff } \mathfrak{M}' \models \varphi.$$

Proof. Exercise. □

Definition 12.3. 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 $P^{\mathfrak{M}'} = R$.

12.2 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 12.4. 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 $f \in \mathcal{L}$ $f^{\mathfrak{M}}(a_1, \dots, a_n) = f^{\mathfrak{M}'}(a_1, \dots, a_n)$ for all $a_1, \dots, a_n \in |\mathfrak{M}|$.
4. For each n -place predicate symbol $R \in \mathcal{L}$, $\langle a_1, \dots, a_n \rangle \in R^{\mathfrak{M}}$ iff $\langle a_1, \dots, a_n \rangle \in R^{\mathfrak{M}'}$ for all $a_1, \dots, a_n \in |\mathfrak{M}|$.

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 $R^{\mathfrak{N}} = R^{\mathfrak{M}} \cap N^n$.

12.3 Overspill

Theorem 12.5. *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, \dots 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 $c_i \neq c_j$. \square

Proposition 12.6. *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 12.5](#). \square

12.4 Isomorphic Structures

First-order structures can be alike in one of two ways. One way in which they 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 enumerable 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 12.7. 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} , $\mathfrak{M} \models \varphi$ iff $\mathfrak{M}' \models \varphi$.

Definition 12.8. 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 $h: |\mathfrak{M}| \rightarrow |\mathfrak{M}'|$ 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 :

$$\langle a_1, \dots, a_n \rangle \in P^{\mathfrak{M}} \quad \text{iff} \quad \langle h(a_1), \dots, h(a_n) \rangle \in P^{\mathfrak{M}'};$$

5. for every n -place function symbol f :

$$h(f^{\mathfrak{M}}(a_1, \dots, a_n)) = f^{\mathfrak{M}'}(h(a_1), \dots, h(a_n)).$$

Theorem 12.9. 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. $h(\text{Val}_s^{\mathfrak{M}}(t)) = \text{Val}_{h \circ s}^{\mathfrak{M}'}(t)$.
- b. $\mathfrak{M}, s \models \varphi$ iff $\mathfrak{M}', h \circ s \models \varphi$.

The first is proved by induction on the complexity of t .

1. If $t \equiv c$, then $\text{Val}_s^{\mathfrak{M}}(c) = c^{\mathfrak{M}}$ and $\text{Val}_{h \circ s}^{\mathfrak{M}'}(c) = c^{\mathfrak{M}'}$. Thus, $h(\text{Val}_s^{\mathfrak{M}}(t)) = h(c^{\mathfrak{M}}) = c^{\mathfrak{M}'}$ (by (3) of Definition 12.8) $= \text{Val}_{h \circ s}^{\mathfrak{M}'}(t)$.
2. If $t \equiv x$, then $\text{Val}_s^{\mathfrak{M}}(x) = s(x)$ and $\text{Val}_{h \circ s}^{\mathfrak{M}'}(x) = h(s(x))$. Thus, $h(\text{Val}_s^{\mathfrak{M}}(x)) = h(s(x)) = \text{Val}_{h \circ s}^{\mathfrak{M}'}(x)$.
3. If $t \equiv f(t_1, \dots, t_n)$, then

$$\begin{aligned} \text{Val}_s^{\mathfrak{M}}(t) &= f^{\mathfrak{M}}(\text{Val}_s^{\mathfrak{M}}(t_1), \dots, \text{Val}_s^{\mathfrak{M}}(t_n)) \quad \text{and} \\ \text{Val}_{h \circ s}^{\mathfrak{M}'}(t) &= f^{\mathfrak{M}'}(\text{Val}_{h \circ s}^{\mathfrak{M}'}(t_1), \dots, \text{Val}_{h \circ s}^{\mathfrak{M}'}(t_n)). \end{aligned}$$

The induction hypothesis is that for each i , $h(\text{Val}_s^{\mathfrak{M}}(t_i)) = \text{Val}_{h \circ s}^{\mathfrak{M}'}(t_i)$. So,

$$\begin{aligned} h(\text{Val}_s^{\mathfrak{M}}(t)) &= h(f^{\mathfrak{M}}(\text{Val}_s^{\mathfrak{M}}(t_1), \dots, \text{Val}_s^{\mathfrak{M}}(t_n))) \\ &= h(f^{\mathfrak{M}}(\text{Val}_{h \circ s}^{\mathfrak{M}'}(t_1), \dots, \text{Val}_{h \circ s}^{\mathfrak{M}'}(t_n))) \end{aligned} \quad (12.1)$$

$$\begin{aligned} &= f^{\mathfrak{M}'}(\text{Val}_{h \circ s}^{\mathfrak{M}'}(t_1), \dots, \text{Val}_{h \circ s}^{\mathfrak{M}'}(t_n)) \quad (12.2) \\ &= \text{Val}_{h \circ s}^{\mathfrak{M}'}(t) \end{aligned}$$

Here, eq. (12.1) follows by induction hypothesis and eq. (12.2) by (5) of Definition 12.8.

Part (2) is left as an exercise.

If φ is a sentence, the assignments s and $h \circ s$ are irrelevant, and we have $\mathfrak{M} \models \varphi$ iff $\mathfrak{M}' \models \varphi$. \square

Definition 12.10. An *automorphism* of a structure \mathfrak{M} is an isomorphism of \mathfrak{M} onto itself.

12.5 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 12.11. Given a structure \mathfrak{M} , the *theory* of \mathfrak{M} is the set $\text{Th}(\mathfrak{M})$ of sentences that are true in \mathfrak{M} , i.e., $\text{Th}(\mathfrak{M}) = \{\varphi : \mathfrak{M} \models \varphi\}$.

We also use the term “theory” informally to refer to sets of sentences having an intended interpretation, whether deductively closed or not.

Proposition 12.12. For any \mathfrak{M} , $\text{Th}(\mathfrak{M})$ is complete.

Proof. For any sentence φ either $\mathfrak{M} \models \varphi$ or $\mathfrak{M} \models \neg\varphi$, so either $\varphi \in \text{Th}(\mathfrak{M})$ or $\neg\varphi \in \text{Th}(\mathfrak{M})$. \square

12.6. PARTIAL ISOMORPHISMS

Proposition 12.13. *If $\mathfrak{N} \models \varphi$ for every $\varphi \in \text{Th}(\mathfrak{M})$, then $\mathfrak{M} \equiv \mathfrak{N}$.*

Proof. Since $\mathfrak{N} \models \varphi$ for all $\varphi \in \text{Th}(\mathfrak{M})$, $\text{Th}(\mathfrak{M}) \subseteq \text{Th}(\mathfrak{N})$. If $\mathfrak{N} \models \varphi$, then $\mathfrak{N} \not\models \neg\varphi$, so $\neg\varphi \notin \text{Th}(\mathfrak{M})$. Since $\text{Th}(\mathfrak{M})$ is complete, $\varphi \in \text{Th}(\mathfrak{M})$. So, $\text{Th}(\mathfrak{N}) \subseteq \text{Th}(\mathfrak{M})$, and we have $\mathfrak{M} \equiv \mathfrak{N}$. \square

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 non-enumerable; however, since $\text{Th}(\mathfrak{R})$ is obviously consistent, by the Löwenheim-Skolem theorem it has an enumerable model, say \mathfrak{S} , and by [Proposition 12.13](#), $\mathfrak{R} \equiv \mathfrak{S}$. Moreover, since \mathfrak{R} and \mathfrak{S} are not isomorphic, this shows that the converse of [Theorem 12.9](#) fails in general.

12.6 Partial Isomorphisms

Definition 12.14. Given two structures \mathfrak{M} and \mathfrak{N} , a *partial isomorphism* from \mathfrak{M} to \mathfrak{N} is a finite partial function p taking arguments in $|\mathfrak{M}|$ and returning values in $|\mathfrak{N}|$, which satisfies the isomorphism conditions from [Definition 12.8](#) on its domain:

1. p is injective;
2. for every constant symbol c : if $p(c^{\mathfrak{M}})$ is defined, then $p(c^{\mathfrak{M}}) = c^{\mathfrak{N}}$;
3. for every n -place predicate symbol P : if a_1, \dots, a_n are in the domain of p , then $\langle a_1, \dots, a_n \rangle \in P^{\mathfrak{M}}$ if and only if $\langle p(a_1), \dots, p(a_n) \rangle \in P^{\mathfrak{N}}$;
4. for every n -place function symbol f : if a_1, \dots, a_n are in the domain of p , then $p(f^{\mathfrak{M}}(a_1, \dots, a_n)) = f^{\mathfrak{N}}(p(a_1), \dots, p(a_n))$.

That p is finite means that $\text{dom}(p)$ is finite.

Notice that the empty function \emptyset is always a partial isomorphism between any two structures.

Definition 12.15. Two structures \mathfrak{M} and \mathfrak{N} , are *partially isomorphic*, written $\mathfrak{M} \simeq_p \mathfrak{N}$, if and only if there is a non-empty set I of partial isomorphisms between \mathfrak{M} and \mathfrak{N} satisfying the *back-and-forth* property:

1. (*Forth*) For every $p \in I$ and $a \in |\mathfrak{M}|$ there is $q \in I$ such that $p \subseteq q$ and a is in the domain of q ;
2. (*Back*) For every $p \in I$ and $b \in |\mathfrak{N}|$ there is $q \in I$ such that $p \subseteq q$ and b is in the range of q .

Theorem 12.16. *If $\mathfrak{M} \simeq_p \mathfrak{N}$ and \mathfrak{M} and \mathfrak{N} are enumerable, then $\mathfrak{M} \simeq \mathfrak{N}$.*

Proof. Since \mathfrak{M} and \mathfrak{N} are enumerable, let $|\mathfrak{M}| = \{a_0, a_1, \dots\}$ and $|\mathfrak{N}| = \{b_0, b_1, \dots\}$. Starting with an arbitrary $p_0 \in I$, we define an increasing sequence of partial isomorphisms $p_0 \subseteq p_1 \subseteq p_2 \subseteq \dots$ as follows:

1. if $n + 1$ is odd, say $n = 2r$, then using the Forth property find a $p_{n+1} \in I$ such that $p_n \subseteq p_{n+1}$ and a_r is in the domain of p_{n+1} ;
2. if $n + 1$ is even, say $n + 1 = 2r$, then using the Back property find a $p_{n+1} \in I$ such that $p_n \subseteq p_{n+1}$ and b_r is in the range of p_{n+1} .

If we now put:

$$p = \bigcup_{n \geq 0} p_n,$$

we have that p is a an isomorphism between \mathfrak{M} and \mathfrak{N} . □

Theorem 12.17. *Suppose \mathfrak{M} and \mathfrak{N} are structures for a purely relational language (a language containing only predicate symbols, and no function symbols or constants). Then if $\mathfrak{M} \simeq_p \mathfrak{N}$, also $\mathfrak{M} \equiv \mathfrak{N}$.*

Proof. By induction on formulas, one shows that if a_1, \dots, a_n and b_1, \dots, b_n are such that there is a partial isomorphism p mapping each a_i to b_i and $s_1(x_i) = a_i$ and $s_2(x_i) = b_i$ (for $i = 1, \dots, n$), then $\mathfrak{M}, s_1 \models \varphi$ if and only if $\mathfrak{N}, s_2 \models \varphi$. The case for $n = 0$ gives $\mathfrak{M} \equiv \mathfrak{N}$. □

Remark 3. If function symbols are present, the previous result is still true, but one needs to consider the isomorphism induced by p between the substructure of \mathfrak{M} generated by a_1, \dots, a_n and the substructure of \mathfrak{N} generated by b_1, \dots, b_n .

The previous result can be “broken down” into stages by establishing a connection between the number of nested quantifiers in a formula and how many times the relevant partial isomorphisms can be extended.

Definition 12.18. For any formula φ , the *quantifier rank* of φ , denoted by $\text{qr}(\varphi) \in \mathbb{N}$, is recursively defined as the highest number of nested quantifiers in φ . Two structures \mathfrak{M} and \mathfrak{N} are *n-equivalent*, written $\mathfrak{M} \equiv_n \mathfrak{N}$, if they agree on all sentences of quantifier rank less than or equal to n .

Proposition 12.19. *Let \mathcal{L} be a finite purely relational language, i.e., a language containing finitely many predicate symbols and constant symbols, and no function symbols. Then for each $n \in \mathbb{N}$ there are only finitely many first-order sentences in the language \mathcal{L} that have quantifier rank no greater than n , up to logical equivalence.*

Proof. By induction on n . □

Definition 12.20. Given a structure \mathfrak{M} , let $|\mathfrak{M}|^{<\omega}$ be the set of all finite sequences over $|\mathfrak{M}|$. We use $\mathbf{a}, \mathbf{b}, \mathbf{c}, \dots$ to range over finite sequences of elements. If $\mathbf{a} \in |\mathfrak{M}|^{<\omega}$ and $a \in |\mathfrak{M}|$, then $\mathbf{a}a$ represents the *concatenation* of \mathbf{a} with a .

12.6. PARTIAL ISOMORPHISMS

Definition 12.21. Given structures \mathfrak{M} and \mathfrak{N} , we define relations $I_n \subseteq |\mathfrak{M}|^{<\omega} \times |\mathfrak{N}|^{<\omega}$ between sequences of equal length, by recursion on n as follows:

1. $I_0(\mathbf{a}, \mathbf{b})$ if and only if \mathbf{a} and \mathbf{b} satisfy the same atomic formulas in \mathfrak{M} and \mathfrak{N} ; i.e., if $s_1(x_i) = a_i$ and $s_2(x_i) = b_i$ and φ is atomic with all variables among x_1, \dots, x_n , then $\mathfrak{M}, s_1 \models \varphi$ if and only if $\mathfrak{N}, s_2 \models \varphi$.
2. $I_{n+1}(\mathbf{a}, \mathbf{b})$ if and only if for every $a \in A$ there is a $b \in B$ such that $I_n(\mathbf{a}a, \mathbf{b}b)$, and vice-versa.

Definition 12.22. Write $\mathfrak{M} \approx_n \mathfrak{N}$ if $I_n(\emptyset, \emptyset)$ holds of \mathfrak{M} and \mathfrak{N} (where \emptyset is the empty sequence).

Theorem 12.23. Let \mathcal{L} be a purely relational language. Then $I_n(\mathbf{a}, \mathbf{b})$ implies that for every φ such that $\text{qr}(\varphi) \leq n$, we have $\mathfrak{M}, \mathbf{a} \models \varphi$ if and only if $\mathfrak{N}, \mathbf{b} \models \varphi$ (where again \mathbf{a} satisfies φ if any s such that $s(x_i) = a_i$ satisfies φ). Moreover, if \mathcal{L} is finite, the converse also holds.

Proof. The proof that $I_n(\mathbf{a}, \mathbf{b})$ implies that \mathbf{a} and \mathbf{b} satisfy the same formulas of quantifier rank no greater than n is by an easy induction on φ . For the converse we proceed by induction on n , using [Proposition 12.19](#), which ensures that for each n there are at most finitely many non-equivalent formulas of that quantifier rank.

For $n = 0$ the hypothesis that \mathbf{a} and \mathbf{b} satisfy the same quantifier-free formulas gives that they satisfy the same atomic ones, so that $I_0(\mathbf{a}, \mathbf{b})$.

For the $n + 1$ case, suppose that \mathbf{a} and \mathbf{b} satisfy the same formulas of quantifier rank no greater than $n + 1$; in order to show that $I_{n+1}(\mathbf{a}, \mathbf{b})$ suffices to show that for each $a \in |\mathfrak{M}|$ there is a $b \in |\mathfrak{N}|$ such that $I_n(\mathbf{a}a, \mathbf{b}b)$, and by the inductive hypothesis again suffices to show that for each $a \in |\mathfrak{M}|$ there is a $b \in |\mathfrak{N}|$ such that $\mathbf{a}a$ and $\mathbf{b}b$ satisfy the same formulas of quantifier rank no greater than n .

Given $a \in |\mathfrak{M}|$, let τ_n^a be set of formulas $\psi(x, \mathbf{y})$ of rank no greater than n satisfied by $\mathbf{a}a$ in \mathfrak{M} ; τ_n^a is finite, so we can assume it is a single first-order formula. It follows that \mathbf{a} satisfies $\exists x \tau_n^a(x, \mathbf{y})$, which has quantifier rank no greater than $n + 1$. By hypothesis \mathbf{b} satisfies the same formula in \mathfrak{N} , so that there is a $b \in |\mathfrak{N}|$ such that $\mathbf{b}b$ satisfies τ_n^a ; in particular, $\mathbf{b}b$ satisfies the same formulas of quantifier rank no greater than n as $\mathbf{a}a$. Similarly one shows that for every $b \in |\mathfrak{N}|$ there is $a \in |\mathfrak{M}|$ such that $\mathbf{a}a$ and $\mathbf{b}b$ satisfy the same formulas of quantifier rank no greater than n , which completes the proof. \square

Corollary 12.24. If \mathfrak{M} and \mathfrak{N} are purely relational structures in a finite language, then $\mathfrak{M} \approx_n \mathfrak{N}$ if and only if $\mathfrak{M} \equiv_n \mathfrak{N}$. In particular $\mathfrak{M} \equiv \mathfrak{N}$ if and only if for each n , $\mathfrak{M} \approx_n \mathfrak{N}$.

12.7 Dense Linear Orders

Definition 12.25. A *dense linear ordering without endpoints* is a structure \mathfrak{M} for the language containing a single 2-place predicate symbol $<$ satisfying the following sentences:

1. $\forall x \, x < x$;
2. $\forall x \, \forall y \, \forall z \, (x < y \rightarrow (y < z \rightarrow x < z))$;
3. $\forall x \, \forall y \, (x < y \vee x = y \vee y < x)$;
4. $\forall x \, \exists y \, x < y$;
5. $\forall x \, \exists y \, y < x$;
6. $\forall x \, \forall y \, (x < y \rightarrow \exists z \, (x < z \wedge z < y))$.

Theorem 12.26. Any two enumerable dense linear orderings without endpoints are isomorphic.

Proof. Let \mathfrak{M}_1 and \mathfrak{M}_2 be enumerable dense linear orderings without endpoints, with $<_1 = <^{\mathfrak{M}_1}$ and $<_2 = <^{\mathfrak{M}_2}$, and let \mathcal{I} be the set of all partial isomorphisms between them. \mathcal{I} is not empty since at least $\emptyset \in \mathcal{I}$. We show that \mathcal{I} satisfies the Back-and-Forth property. Then $\mathfrak{M}_1 \simeq_p \mathfrak{M}_2$, and the theorem follows by Theorem 12.16.

To show \mathcal{I} satisfies the Forth property, let $p \in \mathcal{I}$ and let $p(a_i) = b_i$ for $i = 1, \dots, n$, and without loss of generality suppose $a_1 <_1 a_2 <_1 \dots <_1 a_n$. Given $a \in |\mathfrak{M}_1|$, find $b \in |\mathfrak{M}_2|$ as follows:

1. if $a <_2 a_1$ let $b \in |\mathfrak{M}_2|$ be such that $b <_2 b_1$;
2. if $a_n <_1 a$ let $b \in |\mathfrak{M}_2|$ be such that $b_n <_2 b$;
3. if $a_i <_1 a <_1 a_{i+1}$ for some i , then let $b \in |\mathfrak{M}_2|$ be such that $b_i <_2 b <_2 b_{i+1}$.

It is always possible to find a b with the desired property since \mathfrak{M}_2 is a dense linear ordering without endpoints. Define $q = p \cup \{ \langle a, b \rangle \}$ so that $q \in \mathcal{I}$ is the desired extension of p . This establishes the Forth property. The Back property is similar. So $\mathfrak{M}_1 \simeq_p \mathfrak{M}_2$; by Theorem 12.16, $\mathfrak{M}_1 \simeq \mathfrak{M}_2$. \square

Remark 4. Let \mathfrak{S} be any enumerable dense linear ordering without endpoints. Then (by Theorem 12.26) $\mathfrak{S} \simeq \mathfrak{Q}$, where $\mathfrak{Q} = (\mathbb{Q}, <)$ is the enumerable dense linear ordering having the set \mathbb{Q} of the rational numbers as its domain. Now consider again the structure $\mathfrak{R} = (\mathbb{R}, <)$ from Remark 2. We saw that there is an enumerable structure \mathfrak{S} such that $\mathfrak{R} \equiv \mathfrak{S}$. But \mathfrak{S} is an enumerable dense linear ordering without endpoints, and so it is isomorphic (and hence elementarily equivalent) to the structure \mathfrak{Q} . By transitivity of elementary equivalence,

12.7. DENSE LINEAR ORDERS

$\mathfrak{R} \equiv \mathfrak{Q}$. (We could have shown this directly by establishing $\mathfrak{R} \simeq_p \mathfrak{Q}$ by the same back-and-forth argument.)

Problems

Problem 12.1. Prove [Proposition 12.2](#).

Problem 12.2. Carry out the proof of (b) of [Theorem 12.9](#) in detail. Make sure to note where each of the five properties characterizing isomorphisms of [Definition 12.8](#) is used.

Problem 12.3. 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 $X = \{h(x) : x \in X\}$ (i.e., X is fixed under h).

Problem 12.4. Show in detail that p as defined in [Theorem 12.16](#) is in fact an isomorphism.

Problem 12.5. Complete the proof of [Theorem 12.26](#) by verifying that \mathcal{I} satisfies the Back property.

Chapter 13

Models of Arithmetic

13.1 Introduction

The *standard model* of arithmetic is the structure \mathfrak{N} with $|\mathfrak{N}| = \mathbb{N}$ in which o , ι , $+$, \times , and $<$ are interpreted as you would expect. That is, o is 0, ι is the successor function, $+$ is interpreted as addition and \times as multiplication of the numbers in \mathbb{N} . Specifically,

$$\begin{aligned} o^{\mathfrak{N}} &= 0 \\ \iota^{\mathfrak{N}}(n) &= n + 1 \\ +^{\mathfrak{N}}(n, m) &= n + m \\ \times^{\mathfrak{N}}(n, m) &= nm \end{aligned}$$

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., $\emptyset, a, aa, aaa, \dots$), and interpretations

$$\begin{aligned} o^{\mathfrak{M}} &= \emptyset \\ \iota^{\mathfrak{M}}(s) &= s \frown a \\ +^{\mathfrak{M}}(n, m) &= a^{n+m} \\ \times^{\mathfrak{M}}(n, m) &= a^{nm} \end{aligned}$$

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

13.2. STANDARD MODELS OF ARITHMETIC

by the values of the standard numerals \bar{n} , i.e.,

$$|\mathfrak{N}| = \{\text{Val}^{\mathfrak{N}}(\bar{n}) : n \in \mathbb{N}\}$$

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 $x \neq \text{Val}^{\mathfrak{M}}(\bar{n})$ for all n .

These non-standard elements are pretty neat: they are “infinite natural numbers.” But their existence also explains, in a sense, the incompleteness phenomena. Consider an example, e.g., the consistency statement for Peano arithmetic, $\text{Con}_{\mathbf{PA}}$, i.e., $\neg \exists x \text{Prf}_{\mathbf{PA}}(x, \ulcorner \perp \urcorner)$. Since \mathbf{PA} neither proves $\text{Con}_{\mathbf{PA}}$ nor $\neg \text{Con}_{\mathbf{PA}}$, either can be consistently added to \mathbf{PA} . Since \mathbf{PA} is consistent, $\mathfrak{N} \models \text{Con}_{\mathbf{PA}}$, and consequently $\mathfrak{N} \not\models \neg \text{Con}_{\mathbf{PA}}$. So \mathfrak{N} is *not* a model of $\mathbf{PA} \cup \{\neg \text{Con}_{\mathbf{PA}}\}$, and all its models must be nonstandard. Models of $\mathbf{PA} \cup \{\neg \text{Con}_{\mathbf{PA}}\}$ must contain some element that serves as the witness that makes $\exists x \text{Prf}_{\mathbf{PA}}(\ulcorner \perp \urcorner)$ true, i.e., a Gödel number of a derivation of a contradiction from \mathbf{PA} . Such an element can't be standard—since $\mathbf{PA} \vdash \neg \text{Prf}_{\mathbf{PA}}(\bar{n}, \ulcorner \perp \urcorner)$ for every n .

13.2 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 13.1. A structure for \mathcal{L}_A is *standard* if it is isomorphic to \mathfrak{N} .

Proposition 13.2. If a structure \mathfrak{M} is standard, its domain is the set of values of the standard numerals, i.e.,

$$|\mathfrak{M}| = \{\text{Val}^{\mathfrak{M}}(\bar{n}) : n \in \mathbb{N}\}$$

Proof. Clearly, every $\text{Val}^{\mathfrak{M}}(\bar{n}) \in |\mathfrak{M}|$. We just have to show that every $x \in |\mathfrak{M}|$ is equal to $\text{Val}^{\mathfrak{M}}(\bar{n})$ for some n . Since \mathfrak{M} is standard, it is isomorphic to \mathfrak{N} . Suppose $g: \mathbb{N} \rightarrow |\mathfrak{M}|$ is an isomorphism. Then $g(n) = g(\text{Val}^{\mathfrak{N}}(\bar{n})) = \text{Val}^{\mathfrak{M}}(\bar{n})$. But for every $x \in |\mathfrak{M}|$, there is an $n \in \mathbb{N}$ such that $g(n) = x$, since g is surjective. \square

If a structure \mathfrak{M} for \mathcal{L}_A is standard, the elements of its domain can all be named by the standard numerals $\bar{0}, \bar{1}, \bar{2}, \dots$, i.e., the terms o, o', o'' , 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 13.3. If $\mathfrak{M} \models \mathbf{Q}$, and $|\mathfrak{M}| = \{\text{Val}^{\mathfrak{M}}(\bar{n}) : n \in \mathbb{N}\}$, then \mathfrak{M} is standard.

Proof. We have to show that \mathfrak{M} is isomorphic to \mathfrak{N} . Consider the function $g: \mathbb{N} \rightarrow |\mathfrak{M}|$ defined by $g(n) = \text{Val}^{\mathfrak{M}}(\bar{n})$. By the hypothesis, g is surjective. It is also injective: $\mathbf{Q} \vdash \bar{n} \neq \bar{m}$ whenever $n \neq m$. Thus, since $\mathfrak{M} \models \mathbf{Q}$, $\mathfrak{M} \models \bar{n} \neq \bar{m}$, whenever $n \neq m$. Thus, if $n \neq m$, then $\text{Val}^{\mathfrak{M}}(\bar{n}) \neq \text{Val}^{\mathfrak{M}}(\bar{m})$, i.e., $g(n) \neq g(m)$.

We also have to verify that g is an isomorphism.

1. We have $g(o^{\mathfrak{N}}) = g(0)$ since, $o^{\mathfrak{N}} = 0$. By definition of g , $g(0) = \text{Val}^{\mathfrak{M}}(\bar{0})$. But $\bar{0}$ is just o , 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., $\text{Val}^{\mathfrak{M}}(o) = o^{\mathfrak{M}}$. So we have $g(o^{\mathfrak{N}}) = o^{\mathfrak{M}}$ as required.
2. $g(\iota^{\mathfrak{N}}(n)) = g(n+1)$, since ι in \mathfrak{N} is the successor function on \mathbb{N} . Then, $g(n+1) = \text{Val}^{\mathfrak{M}}(\overline{n+1})$ by definition of g . But $\overline{n+1}$ is the same term as \bar{n}' , so $\text{Val}^{\mathfrak{M}}(\overline{n+1}) = \text{Val}^{\mathfrak{M}}(\bar{n}')$. By the definition of the value function, this is $= \iota^{\mathfrak{M}}(\text{Val}^{\mathfrak{M}}(\bar{n}))$. Since $\text{Val}^{\mathfrak{M}}(\bar{n}) = g(n)$ we get $g(\iota^{\mathfrak{N}}(n)) = \iota^{\mathfrak{M}}(g(n))$.
3. $g(+^{\mathfrak{N}}(n, m)) = g(n+m)$, since $+$ in \mathfrak{N} is the addition function on \mathbb{N} . Then, $g(n+m) = \text{Val}^{\mathfrak{M}}(\overline{n+m})$ by definition of g . But $\mathbf{Q} \vdash \overline{n+m} = (\bar{n} + \bar{m})$, so $\text{Val}^{\mathfrak{M}}(\overline{n+m}) = \text{Val}^{\mathfrak{M}}(\bar{n} + \bar{m})$. By the definition of the value function, this is $= +^{\mathfrak{M}}(\text{Val}^{\mathfrak{M}}(\bar{n}), \text{Val}^{\mathfrak{M}}(\bar{m}))$. Since $\text{Val}^{\mathfrak{M}}(\bar{n}) = g(n)$ and $\text{Val}^{\mathfrak{M}}(\bar{m}) = g(m)$, 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 $\mathbf{Q} \vdash \bar{n} < \bar{m}$, and also $\mathfrak{M} \models \bar{n} < \bar{m}$. Thus $\langle \text{Val}^{\mathfrak{M}}(\bar{n}), \text{Val}^{\mathfrak{M}}(\bar{m}) \rangle \in <^{\mathfrak{M}}$, i.e., $\langle g(n), g(m) \rangle \in <^{\mathfrak{M}}$. If $n \not< m$, then $\mathbf{Q} \vdash \neg \bar{n} < \bar{m}$, and consequently $\mathfrak{M} \not\models \bar{n} < \bar{m}$. 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 $\bar{0}$, $\bar{1}$, $\bar{2}$, etc. So it isn't surprising that if \mathfrak{M} makes at least some basic statements about the \bar{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 \bar{n} 's name, it's the only one.

Proposition 13.4. *If \mathfrak{M} is standard, then g from the proof of Proposition 13.3 is the only isomorphism from \mathfrak{N} to \mathfrak{M} .*

Proof. Suppose $h: \mathbb{N} \rightarrow |\mathfrak{M}|$ is an isomorphism between \mathfrak{N} and \mathfrak{M} . We show that $g = h$ by induction on n . If $n = 0$, then $g(0) = o^{\mathfrak{M}}$ by definition of g . But since h is an isomorphism, $h(0) = h(o^{\mathfrak{N}}) = o^{\mathfrak{M}}$, so $g(0) = h(0)$.

13.3. NON-STANDARD MODELS

Now consider the case for $n + 1$. We have

$$\begin{aligned}
 g(n + 1) &= \text{Val}^{\mathfrak{M}}(\overline{n + 1}) \text{ by definition of } g \\
 &= \text{Val}^{\mathfrak{M}}(\overline{n'}) \\
 &= \iota^{\mathfrak{M}}(\text{Val}^{\mathfrak{M}}(\overline{n})) \\
 &= \iota^{\mathfrak{M}}(g(n)) \text{ by definition of } g \\
 &= \iota^{\mathfrak{M}}(h(n)) \text{ by induction hypothesis} \\
 &= h(\iota^{\mathfrak{N}}(n)) \text{ since } h \text{ is an isomorphism} \\
 &= h(n + 1)
 \end{aligned}$$

□

For any denumerable set X , there's a bijection between \mathbb{N} and X , so every such set X is potentially the domain of a standard model. In fact, once you pick an object $z \in X$ and a suitable function $s: X \rightarrow X$ as $\circ^{\mathfrak{X}}$ and $\iota^{\mathfrak{X}}$, the interpretation of $+$, \times , and $<$ is already fixed. Only functions $s = \iota^{\mathfrak{X}}$ that are both injective and surjective are suitable in a standard model. It has to be injective since the successor function in \mathfrak{N} is, and that ι is injective is expressed by a sentence true in \mathfrak{N} which \mathfrak{X} thus also has to make true. It has to be surjective because otherwise there would be some $x \in X$ not in the domain of s , i.e., the sentence $\forall x \exists y y' = x$ would be false—but it is true in \mathfrak{N} .

13.3 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 13.5. 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 $\text{Val}^{\mathfrak{M}}(\overline{n})$ for some $n \in \mathbb{N}$ are called *standard numbers* (of \mathfrak{M}), and those not, *non-standard numbers*.

By [Proposition 13.2](#), 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 13.6. *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 $g: \mathbb{N} \rightarrow |\mathfrak{M}|$ be an isomorphism. It is easy to see (by induction on n) that $g(\text{Val}^{\mathfrak{N}}(\overline{n})) = \text{Val}^{\mathfrak{M}}(\overline{n})$. 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 \bar{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 13.7. *Let $\mathbf{TA} = \{\varphi : \mathfrak{N} \models \varphi\}$ be the theory of \mathbf{N} . \mathbf{TA} has an enumerable non-standard model.*

Proof. Expand \mathcal{L}_A by a new constant symbol c and consider the set of sentences

$$\Gamma = \mathbf{TA} \cup \{c \neq \bar{0}, c \neq \bar{1}, c \neq \bar{2}, \dots\}$$

Any model \mathfrak{M}^c of Γ would contain an element $x = c^{\mathfrak{M}}$ which is non-standard, since $x \neq \text{Val}^{\mathfrak{M}}(\bar{n})$ for all $n \in \mathbb{N}$. Also, obviously, $\mathfrak{M}^c \models \mathbf{TA}$, 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 $\mathfrak{M} \models \mathbf{TA}$. 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 $c \neq \bar{n}$, but only finitely many. Suppose k is the largest number so that $c \neq \bar{k} \in \Gamma_0$. Define \mathfrak{N}_k by expanding \mathfrak{N} to include the interpretation $c^{\mathfrak{N}_k} = k + 1$. $\mathfrak{N}_k \models \Gamma_0$: if $\varphi \in \mathbf{TA}$, $\mathfrak{N}_k \models \varphi$ since \mathfrak{N}_k is just like \mathfrak{N} in all respects except c , and c does not occur in φ . And $\mathfrak{N}_k \models c \neq \bar{n}$, since $n \leq k$, and $\text{Val}^{\mathfrak{N}_k}(c) = k + 1$. Thus, every finite subset of Γ is satisfiable. \square

13.4 Models of \mathbf{Q}

We know that there are non-standard structures that make the same sentences true as \mathfrak{N} does, i.e., is a model of \mathbf{TA} . Since $\mathfrak{N} \models \mathbf{Q}$, any model of \mathbf{TA} is also a model of \mathbf{Q} . \mathbf{Q} is much weaker than \mathbf{TA} , e.g., $\mathbf{Q} \not\models \forall x \forall y (x + y) = (y + x)$. Weaker theories are easier to satisfy: they have more models. E.g., \mathbf{Q} has models which make $\forall x \forall y (x + y) = (y + x)$ false, but those cannot also be models of \mathbf{TA} , or \mathbf{PA} for that matter. Models of \mathbf{Q} are also relatively simple: we can specify them explicitly.

13.4. MODELS OF \mathbf{Q}

Example 13.8. Consider the structure \mathfrak{K} with domain $|\mathfrak{K}| = \mathbb{N} \cup \{a\}$ and interpretations

$$\begin{aligned} 0^{\mathfrak{K}} &= 0 \\ \iota^{\mathfrak{K}}(x) &= \begin{cases} x+1 & \text{if } x \in \mathbb{N} \\ a & \text{if } x = a \end{cases} \\ +^{\mathfrak{K}}(x, y) &= \begin{cases} x+y & \text{if } x, y \in \mathbb{N} \\ a & \text{otherwise} \end{cases} \\ \times^{\mathfrak{K}}(x, y) &= \begin{cases} xy & \text{if } x, y \in \mathbb{N} \\ a & \text{otherwise} \end{cases} \\ <^{\mathfrak{K}} &= \{\langle x, y \rangle : x, y \in \mathbb{N} \text{ and } x < y\} \cup \{\langle x, a \rangle : x \in |\mathfrak{K}|\} \end{aligned}$$

To show that $\mathfrak{K} \models \mathbf{Q}$ we have to verify that all axioms of \mathbf{Q} are true in \mathfrak{K} . For convenience, let's write x^* for $\iota^{\mathfrak{K}}(x)$ (the “successor” of x in \mathfrak{K}), $x \oplus y$ for $+^{\mathfrak{K}}(x, y)$ (the “sum” of x and y in \mathfrak{K}), $x \otimes y$ for $\times^{\mathfrak{K}}(x, y)$ (the “product” of x and y in \mathfrak{K}), and $x \odot y$ for $\langle x, y \rangle \in <^{\mathfrak{K}}$. With these abbreviations, we can give the operations in \mathfrak{K} more perspicuously as

x	x^*	$x \oplus y$	m	a	$x \otimes y$	m	a
n	$n+1$	n	$n+m$	a	n	nm	a
a	a	a	a	a	a	a	a

We have $n \odot m$ iff $n < m$ for $n, m \in \mathbb{N}$ and $x \odot a$ for all $x \in |\mathfrak{K}|$.

$\mathfrak{K} \models \forall x \forall y (x' = y' \rightarrow x = y)$ since $*$ is injective. $\mathfrak{K} \models \forall x 0 \neq x'$ since 0 is not a $*$ -successor in \mathfrak{K} . $\mathfrak{K} \models \forall x (x \neq 0 \rightarrow \exists y x = y')$ since for every $n > 0$, $n = (n-1)^*$, and $a = a^*$.

$\mathfrak{K} \models \forall x (x + 0) = x$ since $n \oplus 0 = n + 0 = n$, and $a \oplus 0 = a$ by definition of \oplus . $\mathfrak{K} \models \forall x \forall y (x + y') = (x + y)'$ is a bit trickier. If n, m are both standard, we have:

$$(n \oplus m^*) = (n + (m+1)) = (n+m) + 1 = (n \oplus m)^*$$

since \oplus and $*$ agree with $+$ and ι on standard numbers. Now suppose $x \in |\mathfrak{K}|$. Then

$$(x \oplus a^*) = (x \oplus a) = a = a^* = (x \oplus a)^*$$

The remaining case is if $y \in |\mathfrak{K}|$ but $x = a$. Here we also have to distinguish cases according to whether $y = n$ is standard or $y = b$:

$$\begin{aligned} (a \oplus n^*) &= (a \oplus (n+1)) = a = a^* = (x \oplus n)^* \\ (a \oplus a^*) &= (a \oplus a) = a = a^* = (x \oplus a)^* \end{aligned}$$

This is of course a bit more detailed than needed. For instance, since $a \oplus z = a$ whatever z is, we can immediately conclude $a \oplus a^* = a$. The remaining axioms can be verified the same way.

\mathfrak{K} is thus a model of \mathbf{Q} . Its “addition” \oplus is also commutative. But there are other sentences true in \mathfrak{N} but false in \mathfrak{K} , and vice versa. For instance, $a \otimes a$, so $\mathfrak{K} \models \exists x x < x$ and $\mathfrak{K} \not\models \forall x \neg x < x$. This shows that $\mathbf{Q} \not\models \forall x \neg x < x$.

Example 13.9. Consider the structure \mathcal{L} with domain $|\mathcal{L}| = \mathbb{N} \cup \{a, b\}$ and interpretations $\iota^{\mathcal{L}} = *$, $+^{\mathcal{L}} = \oplus$ given by

x	x^*	$x \oplus y$	m	a	b
n	$n + 1$	n	$n + m$	b	a
a	a	a	a	b	a
b	b	b	b	b	a

Since $*$ is injective, 0 is not in its range, and every $x \in |\mathcal{L}|$ other than 0 is, axioms Q_1 – Q_3 are true in \mathcal{L} . For any x , $x \oplus 0 = x$, so Q_4 is true as well. For Q_5 , consider $x \oplus y^*$ and $(x \oplus y)^*$. They are equal if x and y are both standard, since then $*$ and \oplus agree with ι and $+$. If x is non-standard, and y is standard, we have $x \oplus y^* = x = x^* = (x \oplus y)^*$. If x and y are both non-standard, we have four cases:

$$\begin{aligned} a \oplus a^* &= b = b^* = (a \oplus a)^* \\ b \oplus b^* &= a = a^* = (b \oplus b)^* \\ b \oplus a^* &= b = b^* = (b \oplus y)^* \\ a \oplus b^* &= a = a^* = (a \oplus b)^* \end{aligned}$$

If x is standard, but y is non-standard, we have

$$\begin{aligned} n \oplus a^* &= n \oplus a = b = b^* = (n \oplus a)^* \\ n \oplus b^* &= n \oplus b = a = a^* = (n \oplus b)^* \end{aligned}$$

So, $\mathcal{L} \models Q_5$. However, $a \oplus 0 \neq 0 \oplus a$, so $\mathcal{L} \not\models \forall x \forall y (x + y) = (y + x)$.

We’ve explicitly constructed models of \mathbf{Q} in which the non-standard elements live “beyond” the standard elements. In fact, that much is required by the axioms. A non-standard element x cannot be $\otimes 0$. Otherwise, for some z , $x \oplus z^* = 0$ by Q_8 . But then $0 = x \oplus z^* = (x \oplus z)^*$ by Q_5 , contradicting Q_2 . Also, for every n , $\mathbf{Q} \vdash \forall x (x < \bar{n}' \rightarrow (x = \bar{0} \vee x = \bar{1} \vee \dots \vee x = \bar{n}))$, so we can’t have $a \otimes n$ for any $n > 0$.

13.5 Computable Models of Arithmetic

The standard model \mathfrak{N} has two nice features. Its domain is the natural numbers \mathbb{N} , i.e., its elements are just the kinds of things we want to talk about

13.5. COMPUTABLE MODELS OF ARITHMETIC

using the language of arithmetic, and the standard numeral \bar{n} actually picks out n . The other nice feature is that the interpretations of the non-logical symbols of \mathcal{L}_A are all *computable*. The successor, addition, and multiplication functions which serve as $\iota^{\mathfrak{N}}$, $+\mathfrak{N}$, and $\times^{\mathfrak{N}}$ are computable functions of numbers. (Computable by Turing machines, or definable by primitive recursion, say.) And the less-than relation on \mathfrak{N} , i.e., $<^{\mathfrak{N}}$, is decidable.

Non-standard models of arithmetical theories such as **Q** and **PA** must contain non-standard elements. Thus their domains typically include elements in addition to \mathbb{N} . However, any countable structure can be built on any denumerable set, including \mathbb{N} . So there are also non-standard models with domain \mathbb{N} . In such models \mathfrak{M} , of course, at least some numbers cannot play the roles they usually play, since some k must be different from $\text{Val}^{\mathfrak{M}}(\bar{n})$ for all $n \in \mathbb{N}$.

Definition 13.10. A structure \mathfrak{M} for \mathcal{L}_A is *computable* iff $|\mathfrak{M}| = \mathbb{N}$ and $\iota^{\mathfrak{M}}$, $+\mathfrak{M}$, $\times^{\mathfrak{M}}$ are computable functions and $<^{\mathfrak{M}}$ is a decidable relation.

Example 13.11. Recall the structure \mathfrak{K} from [Example 13.8](#). Its domain was $|\mathfrak{K}| = \mathbb{N} \cup \{a\}$ and interpretations

$$\begin{aligned} o^{\mathfrak{K}} &= 0 \\ \iota^{\mathfrak{K}}(x) &= \begin{cases} x+1 & \text{if } x \in \mathbb{N} \\ a & \text{if } x = a \end{cases} \\ +^{\mathfrak{K}}(x, y) &= \begin{cases} x+y & \text{if } x, y \in \mathbb{N} \\ a & \text{otherwise} \end{cases} \\ \times^{\mathfrak{K}}(x, y) &= \begin{cases} xy & \text{if } x, y \in \mathbb{N} \\ a & \text{otherwise} \end{cases} \\ <^{\mathfrak{K}} &= \{\langle x, y \rangle : x, y \in \mathbb{N} \text{ and } x < y\} \cup \{\langle x, a \rangle : n \in |\mathfrak{K}|\} \end{aligned}$$

But $|\mathfrak{K}|$ is denumerable and so is equinumerous with \mathbb{N} . For instance, $g: \mathbb{N} \rightarrow |\mathfrak{K}|$ with $g(0) = a$ and $g(n) = n+1$ for $n > 0$ is a bijection. We can turn it into an isomorphism between a new model \mathfrak{K}' of **Q** and \mathfrak{K} . In \mathfrak{K}' , we have to assign different functions and relations to the symbols of \mathcal{L}_A , since different elements of \mathbb{N} play the roles of standard and non-standard numbers.

Specifically, 0 now plays the role of a , not of the smallest standard number. The smallest standard number is now 1. So we assign $o^{\mathfrak{K}'} = 1$. The successor function is also different now: given a standard number, i.e., an $n > 0$, it still returns $n+1$. But 0 now plays the role of a , which is its own successor. So

$\iota^{\mathfrak{K}'}(0) = 0$. For addition and multiplication we likewise have

$$\begin{aligned} +^{\mathfrak{K}'}(x, y) &= \begin{cases} x + y & \text{if } x, y > 0 \\ 0 & \text{otherwise} \end{cases} \\ \times^{\mathfrak{K}'}(x, y) &= \begin{cases} xy & \text{if } x, y > 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

And we have $\langle x, y \rangle \in <^{\mathfrak{K}'}$ iff $x < y$ and $x > 0$ and $y > 0$, or if $y = 0$.

All of these functions are computable functions of natural numbers and $<^{\mathfrak{K}'}$ is a decidable relation on \mathbb{N} —but they are not the same functions as successor, addition, and multiplication on \mathbb{N} , and $<^{\mathfrak{K}'}$ is not the same relation as $<$ on \mathbb{N} .

This example shows that **Q** has computable non-standard models with domain \mathbb{N} . However, the following result shows that this is not true for models of **PA** (and thus also for models of **TA**).

Theorem 13.12 (Tennenbaum’s Theorem). *\mathfrak{N} is the only computable model of **PA**.*

Problems

Problem 13.1. Show that the converse of [Proposition 13.2](#) is false, i.e., give an example of a structure \mathfrak{M} with $|\mathfrak{M}| = \{\text{Val}^{\mathfrak{M}}(\bar{n}) : n \in \mathbb{N}\}$ that is not isomorphic to \mathfrak{N} .

Problem 13.2. Recall that **Q** contains the axioms

$$\forall x \forall y (x' = y' \rightarrow x = y) \quad (Q_1)$$

$$\forall x \, 0 \neq x' \quad (Q_2)$$

$$\forall x (x \neq 0 \rightarrow \exists y \, x = y') \quad (Q_3)$$

Give structures $\mathfrak{M}_1, \mathfrak{M}_2, \mathfrak{M}_3$ such that

1. $\mathfrak{M}_1 \models Q_1, \mathfrak{M}_1 \models Q_2, \mathfrak{M}_1 \not\models Q_3$;
2. $\mathfrak{M}_2 \models Q_1, \mathfrak{M}_2 \not\models Q_2, \mathfrak{M}_2 \models Q_3$; and
3. $\mathfrak{M}_3 \not\models Q_1, \mathfrak{M}_3 \models Q_2, \mathfrak{M}_3 \models Q_3$;

Obviously, you just have to specify $0^{\mathfrak{M}_i}$ and $\iota^{\mathfrak{M}_i}$ for each.

Problem 13.3. Prove that \mathfrak{K} from [Example 13.8](#) satisfies the remaining axioms of **Q**,

$$\forall x (x \times 0) = 0 \quad (Q_6)$$

$$\forall x \forall y (x \times y') = ((x \times y) + x) \quad (Q_7)$$

$$\forall x \forall y (x < y \leftrightarrow \exists z (x + z' = y)) \quad (Q_8)$$

13.5. COMPUTABLE MODELS OF ARITHMETIC

Find a sentence only involving $+$ true in \mathfrak{N} but false in \mathfrak{K} .

Problem 13.4. Expand \mathcal{L} of [Example 13.9](#) to include \otimes and \odot that interpret \times and $<$. Show that your structure satisfies the remaining axioms of \mathbf{Q} ,

$$\forall x (x \times 0) = 0 \quad (Q_6)$$

$$\forall x \forall y (x \times y') = ((x \times y) + x) \quad (Q_7)$$

$$\forall x \forall y (x < y \leftrightarrow \exists z (x + z' = y)) \quad (Q_8)$$

Problem 13.5. In \mathcal{L} of [Example 13.9](#), $a^* = a$ and $b^* = b$. Is there a model of \mathbf{Q} in which $a^* = b$ and $b^* = a$?

Problem 13.6. Give a structure \mathcal{L}' with $|\mathcal{L}'| = \mathbb{N}$ isomorphic to \mathcal{L} of [Example 13.9](#).

Chapter 14

The Interpolation Theorem

14.1 Introduction

The interpolation theorem is the following result: Suppose $\models \varphi \rightarrow \psi$. Then there is a sentence χ such that $\models \varphi \rightarrow \chi$ and $\models \chi \rightarrow \psi$. Moreover, every constant symbol, function symbol, and predicate symbol (other than $=$) in χ occurs both in φ and ψ . The sentence χ is called an *interpolant* of φ and ψ .

The interpolation theorem is interesting in its own right, but its main importance lies in the fact that it can be used to prove results about definability in a theory, and the conditions under which combining two consistent theories results in a consistent theory. The first result is known as the Beth definability theorem; the second, Robinson's joint consistency theorem.

14.2 Separation of Sentences

A bit of groundwork is needed before we can proceed with the proof of the interpolation theorem. An interpolant for φ and ψ is a sentence χ such that $\varphi \models \chi$ and $\chi \models \psi$. By contraposition, the latter is true iff $\neg\psi \models \neg\chi$. A sentence χ with this property is said to *separate* φ and $\neg\psi$. So finding an interpolant for φ and ψ amounts to finding a sentence that separates φ and $\neg\psi$. As so often, it will be useful to consider a generalization: a sentence that separates two *sets* of sentences.

Definition 14.1. A sentence χ *separates* sets of sentences Γ and Δ if and only if $\Gamma \models \chi$ and $\Delta \models \neg\chi$. If no such sentence exists, then Γ and Δ are *inseparable*.

The inclusion relations between the classes of models of Γ , Δ and χ are represented below:

Lemma 14.2. Suppose \mathcal{L}_0 is the language containing every constant symbol, function symbol and predicate symbol (other than $=$) that occurs in both Γ and Δ , and let

14.2. SEPARATION OF SENTENCES

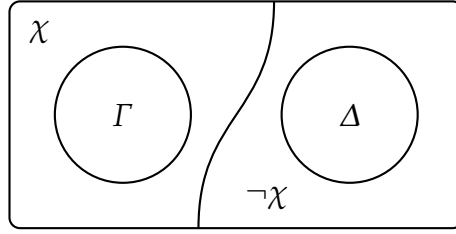


Figure 14.1: χ separates Γ and Δ

\mathcal{L}'_0 be obtained by the addition of infinitely many new constant symbols c_n for $n \geq 0$. Then if Γ and Δ are inseparable in \mathcal{L}_0 , they are also inseparable in \mathcal{L}'_0 .

Proof. We proceed indirectly: suppose by way of contradiction that Γ and Δ are separated in \mathcal{L}'_0 . Then $\Gamma \models \chi[c/x]$ and $\Delta \models \neg\chi[c/x]$ for some $\chi \in \mathcal{L}_0$ (where c is a new constant symbol—the case where χ contains more than one such new constant symbol is similar). By compactness, there are *finite* subsets Γ_0 of Γ and Δ_0 of Δ such that $\Gamma_0 \models \chi[c/x]$ and $\Delta_0 \models \neg\chi[c/x]$. Let γ be the conjunction of all formulas in Γ_0 and δ the conjunction of all formulas in Δ_0 . Then

$$\gamma \models \chi[c/x], \quad \delta \models \neg\chi[c/x].$$

From the former, by Generalization, we have $\gamma \models \forall x \chi$, and from the latter by contraposition, $\chi[c/x] \models \neg\delta$, whence also $\forall x \chi \models \neg\delta$. Contraposition again gives $\delta \models \neg\forall x \chi$. By monotony,

$$\Gamma \models \forall x \chi, \quad \Delta \models \neg\forall x \chi,$$

so that $\forall x \chi$ separates Γ and Δ in \mathcal{L}_0 . □

Lemma 14.3. Suppose that $\Gamma \cup \{\exists x \sigma\}$ and Δ are inseparable, and c is a new constant symbol not in Γ , Δ , or σ . Then $\Gamma \cup \{\exists x \sigma, \sigma[c/x]\}$ and Δ are also inseparable.

Proof. Suppose for contradiction that χ separates $\Gamma \cup \{\exists x \sigma, \sigma[c/x]\}$ and Δ , while at the same time $\Gamma \cup \{\exists x \sigma\}$ and Δ are inseparable. We distinguish two cases:

1. c does not occur in χ : in this case $\Gamma \cup \{\exists x \sigma, \neg\chi\}$ is satisfiable (otherwise χ separates $\Gamma \cup \{\exists x \sigma\}$ and Δ). It remains so if $\sigma[c/x]$ is added, so χ does not separate $\Gamma \cup \{\exists x \sigma, \sigma[c/x]\}$ and Δ after all.
2. c does occur in χ so that χ has the form $\chi[c/x]$. Then we have that

$$\Gamma \cup \{\exists x \sigma, \sigma[c/x]\} \models \chi[c/x],$$

whence $\Gamma, \exists x \sigma \models \forall x (\sigma \rightarrow \chi)$ by the Deduction Theorem and Generalization, and finally $\Gamma \cup \{\exists x \sigma\} \models \exists x \chi$. On the other hand, $\Delta \models \neg \chi[c/x]$ and hence by Generalization $\Delta \models \neg \exists x \chi$. So $\Gamma \cup \{\exists x \sigma\}$ and Δ are separable, a contradiction. \square

14.3 Craig's Interpolation Theorem

Theorem 14.4 (Craig's Interpolation Theorem). *If $\models \varphi \rightarrow \psi$, then there is a sentence χ such that $\models \varphi \rightarrow \chi$ and $\models \chi \rightarrow \psi$, and every constant symbol, function symbol, and predicate symbol (other than $=$) in χ occurs both in φ and ψ . The sentence χ is called an interpolant of φ and ψ .*

Proof. Suppose \mathcal{L}_1 is the language of φ and \mathcal{L}_2 is the language of ψ . Let $\mathcal{L}_0 = \mathcal{L}_1 \cap \mathcal{L}_2$. For each $i \in \{0, 1, 2\}$, let \mathcal{L}'_i be obtained from \mathcal{L}_i by adding the infinitely many new constant symbols c_0, c_1, c_2, \dots .

If φ is unsatisfiable, $\exists x x \neq x$ is an interpolant. If $\neg \psi$ is unsatisfiable (and hence ψ is valid), $\exists x x = x$ is an interpolant. So we may assume also that both φ and $\neg \psi$ are satisfiable.

In order to prove the contrapositive of the Interpolation Theorem, assume that there is no interpolant for φ and ψ . In other words, assume that $\{\varphi\}$ and $\{\neg \psi\}$ are inseparable in \mathcal{L}_0 .

Our goal is to extend the pair $(\{\varphi\}, \{\neg \psi\})$ to a maximally inseparable pair (Γ^*, Δ^*) . Let $\varphi_0, \varphi_1, \varphi_2, \dots$ enumerate the sentences of \mathcal{L}_1 , and $\psi_0, \psi_1, \psi_2, \dots$ enumerate the sentences of \mathcal{L}_2 . We define two increasing sequences of sets of sentences (Γ_n, Δ_n) , for $n \geq 0$, as follows. Put $\Gamma_0 = \{\varphi\}$ and $\Delta_0 = \{\neg \psi\}$. Assuming (Γ_n, Δ_n) are already defined, define Γ_{n+1} and Δ_{n+1} by:

1. If $\Gamma_n \cup \{\varphi_n\}$ and Δ_n are inseparable in \mathcal{L}'_0 , put φ_n in Γ_{n+1} . Moreover, if φ_n is an existential formula $\exists x \sigma$ then pick a new constant symbol c not occurring in $\Gamma_n, \Delta_n, \varphi_n$ or ψ_n , and put $\sigma[c/x]$ in Γ_{n+1} .
2. If Γ_{n+1} and $\Delta_n \cup \{\psi_n\}$ are inseparable in \mathcal{L}'_0 , put ψ_n in Δ_{n+1} . Moreover, if ψ_n is an existential formula $\exists x \sigma$, then pick a new constant symbol c not occurring in $\Gamma_{n+1}, \Delta_n, \varphi_n$ or ψ_n , and put $\sigma[c/x]$ in Δ_{n+1} .

Finally, define:

$$\Gamma^* = \bigcup_{n \geq 0} \Gamma_n, \quad \Delta^* = \bigcup_{n \geq 0} \Delta_n.$$

By simultaneous induction on n we can now prove:

1. Γ_n and Δ_n are inseparable in \mathcal{L}'_0 ;
2. Γ_{n+1} and Δ_n are inseparable in \mathcal{L}'_0 .

14.3. CRAIG'S INTERPOLATION THEOREM

The basis for (1) is given by Lemma 14.2. For part (2), we need to distinguish three cases:

1. If $\Gamma_0 \cup \{\varphi_0\}$ and Δ_0 are separable, then $\Gamma_1 = \Gamma_0$ and (2) is just (1);
2. If $\Gamma_1 = \Gamma_0 \cup \{\varphi_0\}$, then Γ_1 and Δ_0 are inseparable by construction.
3. It remains to consider the case where φ_0 is existential, so that $\Gamma_1 = \Gamma_0 \cup \{\exists x \sigma, \sigma[c/x]\}$. By construction, $\Gamma_0 \cup \{\exists x \sigma\}$ and Δ_0 are inseparable, so that by Lemma 14.3 also $\Gamma_0 \cup \{\exists x \sigma, \sigma[c/x]\}$ and Δ_0 are inseparable.

This completes the basis of the induction for (1) and (2) above. Now for the inductive step. For (1), if $\Delta_{n+1} = \Delta_n \cup \{\psi_n\}$ then Γ_{n+1} and Δ_{n+1} are inseparable by construction (even when ψ_n is existential, by Lemma 14.3); if $\Delta_{n+1} = \Delta_n$ (because Γ_{n+1} and $\Delta_n \cup \{\psi_n\}$ are separable), then we use the induction hypothesis on (2). For the inductive step for (2), if $\Gamma_{n+2} = \Gamma_{n+1} \cup \{\varphi_{n+1}\}$ then Γ_{n+2} and Δ_{n+1} are inseparable by construction (even when φ_{n+1} is existential, by Lemma 14.3); and if $\Gamma_{n+2} = \Gamma_{n+1}$ then we use the inductive case for (1) just proved. This concludes the induction on (1) and (2).

It follows that Γ^* and Δ^* are inseparable; if not, by compactness, there is $n \geq 0$ that separates Γ_n and Δ_n , against (1). In particular, Γ^* and Δ^* are consistent: for if the former or the latter is inconsistent, then they are separated by $\exists x x \neq x$ or $\forall x x = x$, respectively.

We now show that Γ^* is maximally consistent in \mathcal{L}'_1 and likewise Δ^* in \mathcal{L}'_2 . For the former, suppose that $\varphi_n \notin \Gamma^*$ and $\neg\varphi_n \notin \Gamma^*$, for some $n \geq 0$. If $\varphi_n \notin \Gamma^*$ then $\Gamma_n \cup \{\varphi_n\}$ is separable from Δ_n , and so there is $\chi \in \mathcal{L}'_0$ such that both:

$$\Gamma^* \models \varphi_n \rightarrow \chi, \quad \Delta^* \models \neg\chi.$$

Likewise, if $\neg\varphi_n \notin \Gamma^*$, there is $\chi' \in \mathcal{L}'_0$ such that both:

$$\Gamma^* \models \neg\varphi_n \rightarrow \chi', \quad \Delta^* \models \neg\chi'.$$

By propositional logic, $\Gamma^* \models \chi \vee \chi'$ and $\Delta^* \models \neg(\chi \vee \chi')$, so $\chi \vee \chi'$ separates Γ^* and Δ^* . A similar argument establishes that Δ^* is maximal.

Finally, we show that $\Gamma^* \cap \Delta^*$ is maximally consistent in \mathcal{L}'_0 . It is obviously consistent, since it is the intersection of consistent sets. To show maximality, let $\sigma \in \mathcal{L}'_0$. Now, Γ^* is maximal in $\mathcal{L}'_1 \supseteq \mathcal{L}'_0$, and similarly Δ^* is maximal in $\mathcal{L}'_2 \supseteq \mathcal{L}'_0$. It follows that either $\sigma \in \Gamma^*$ or $\neg\sigma \in \Gamma^*$, and either $\sigma \in \Delta^*$ or $\neg\sigma \in \Delta^*$. If $\sigma \in \Gamma^*$ and $\neg\sigma \in \Delta^*$ then σ would separate Γ^* and Δ^* ; and if $\neg\sigma \in \Gamma^*$ and $\sigma \in \Delta^*$ then Γ^* and Δ^* would be separated by $\neg\sigma$. Hence, either $\sigma \in \Gamma^* \cap \Delta^*$ or $\neg\sigma \in \Gamma^* \cap \Delta^*$, and $\Gamma^* \cap \Delta^*$ is maximal.

Since Γ^* is maximally consistent, it has a model \mathfrak{M}'_1 whose domain $|\mathfrak{M}'_1|$ comprises all and only the elements $c^{\mathfrak{M}'_1}$ interpreting the constant symbols—just like in the proof of the completeness theorem (Theorem 10.16). Similarly,

Δ^* has a model \mathfrak{M}'_2 whose domain $|\mathfrak{M}'_2|$ is given by the interpretations $c^{\mathfrak{M}'_2}$ of the constant symbols.

Let \mathfrak{M}_1 be obtained from \mathfrak{M}'_1 by dropping interpretations for constant symbols, function symbols, and predicate symbols in $\mathcal{L}'_1 \setminus \mathcal{L}'_0$, and similarly for \mathfrak{M}_2 . Then the map $h: M_1 \rightarrow M_2$ defined by $h(c^{\mathfrak{M}'_1}) = c^{\mathfrak{M}'_2}$ is an isomorphism in \mathcal{L}'_0 , because $\Gamma^* \cap \Delta^*$ is maximally consistent in \mathcal{L}'_0 , as shown. This follows because any \mathcal{L}'_0 -sentence either belongs to both Γ^* and Δ^* , or to neither: so $c^{\mathfrak{M}'_1} \in P^{\mathfrak{M}'_1}$ if and only if $P(c) \in \Gamma^*$ if and only if $P(c) \in \Delta^*$ if and only if $c^{\mathfrak{M}'_2} \in P^{\mathfrak{M}'_2}$. The other conditions satisfied by isomorphisms can be established similarly.

Let us now define a model \mathfrak{M} for the language $\mathcal{L}_1 \cup \mathcal{L}_2$ as follows:

1. The domain $|\mathfrak{M}|$ is just $|\mathfrak{M}_2|$, i.e., the set of all elements $c^{\mathfrak{M}'_2}$;
2. If a predicate symbol P is in $\mathcal{L}_2 \setminus \mathcal{L}_1$ then $P^{\mathfrak{M}} = P^{\mathfrak{M}'_2}$;
3. If a predicate P is in $\mathcal{L}_1 \setminus \mathcal{L}_2$ then $P^{\mathfrak{M}} = h(P^{\mathfrak{M}'_2})$, i.e., $\langle c_1^{\mathfrak{M}'_2}, \dots, c_n^{\mathfrak{M}'_2} \rangle \in P^{\mathfrak{M}}$ if and only if $\langle c_1^{\mathfrak{M}'_1}, \dots, c_n^{\mathfrak{M}'_1} \rangle \in P^{\mathfrak{M}'_1}$.
4. If a predicate symbol P is in \mathcal{L}_0 then $P^{\mathfrak{M}} = P^{\mathfrak{M}'_2} = h(P^{\mathfrak{M}'_1})$.
5. Function symbols of $\mathcal{L}_1 \cup \mathcal{L}_2$, including constant symbols, are handled similarly.

Finally, one shows by induction on formulas that \mathfrak{M} agrees with \mathfrak{M}'_1 on all formulas of \mathcal{L}'_1 and with \mathfrak{M}'_2 on all formulas of \mathcal{L}'_2 . In particular, $\mathfrak{M} \models \Gamma^* \cup \Delta^*$, whence $\mathfrak{M} \models \varphi$ and $\mathfrak{M} \models \neg\psi$, and $\not\models \varphi \rightarrow \psi$. This concludes the proof of Craig's Interpolation Theorem. \square

14.4 The Definability Theorem

One important application of the interpolation theorem is Beth's definability theorem. To define an n -place relation R we can give a formula χ with n free variables which does not involve R . This would be an *explicit* definition of R in terms of χ . We can then say also that a theory $\Sigma(P)$ in a language containing the n -place predicate symbol P explicitly defines P if it contains (or at least entails) a formalized explicit definition, i.e.,

$$\Sigma(P) \models \forall x_1 \dots \forall x_n (P(x_1, \dots, x_n) \leftrightarrow \chi(x_1, \dots, x_n)).$$

But an explicit definition is only one way of defining—in the sense of determining completely—a relation. A theory may also be such that the interpretation of P is fixed by the interpretation of the rest of the language in any model. The definability theorem states that whenever a theory fixes the interpretation of P in this way—whenever it *implicitly defines* P —then it also explicitly defines it.

14.4. THE DEFINABILITY THEOREM

Definition 14.5. Suppose \mathcal{L} is a language not containing the predicate symbol P . A set $\Sigma(P)$ of sentences of $\mathcal{L} \cup \{P\}$ *explicitly defines* P if and only if there is a formula $\chi(x_1, \dots, x_n)$ of \mathcal{L} such that

$$\Sigma(P) \models \forall x_1 \dots \forall x_n (P(x_1, \dots, x_n) \leftrightarrow \chi(x_1, \dots, x_n)).$$

Definition 14.6. Suppose \mathcal{L} is a language not containing the predicate symbols P and P' . A set $\Sigma(P)$ of sentences of $\mathcal{L} \cup \{P\}$ *implicitly defines* P if and only if

$$\Sigma(P) \cup \Sigma(P') \models \forall x_1 \dots \forall x_n (P(x_1, \dots, x_n) \leftrightarrow P'(x_1, \dots, x_n)),$$

where $\Sigma(P')$ is the result of uniformly replacing P with P' in $\Sigma(P)$.

In other words, for any model \mathfrak{M} and $R, R' \subseteq |\mathfrak{M}|^n$, if both $(\mathfrak{M}, R) \models \Sigma(P)$ and $(\mathfrak{M}, R') \models \Sigma(P')$, then $R = R'$; where (\mathfrak{M}, R) is the structure \mathfrak{M}' for the expansion of \mathcal{L} to $\mathcal{L} \cup \{P\}$ such that $P^{\mathfrak{M}'} = R$, and similarly for (\mathfrak{M}, R') .

Theorem 14.7 (Beth Definability Theorem). *A set $\Sigma(P)$ of $\mathcal{L} \cup \{P\}$ -formulas implicitly defines P if and only if $\Sigma(P)$ explicitly defines P .*

Proof. If $\Sigma(P)$ explicitly defines P then both

$$\begin{aligned} \Sigma(P) \models & \quad \forall x_1 \dots \forall x_n [(P(x_1, \dots, x_n) \leftrightarrow \chi(x_1, \dots, x_n))] \\ \Sigma(P') \models & \quad \forall x_1 \dots \forall x_n [(P'(x_1, \dots, x_n) \leftrightarrow \chi(x_1, \dots, x_n))] \end{aligned}$$

and the conclusion follows. For the converse: assume that $\Sigma(P)$ implicitly defines P . First, we add constant symbols c_1, \dots, c_n to \mathcal{L} . Then

$$\Sigma(P) \cup \Sigma(P') \models P(c_1, \dots, c_n) \rightarrow P'(c_1, \dots, c_n).$$

By compactness, there are finite sets $\Delta_0 \subseteq \Sigma(P)$ and $\Delta_1 \subseteq \Sigma(P')$ such that

$$\Delta_0 \cup \Delta_1 \models P(c_1, \dots, c_n) \rightarrow P'(c_1, \dots, c_n).$$

Let $\theta(P)$ be the conjunction of all sentences $\varphi(P)$ such that either $\varphi(P) \in \Delta_0$ or $\varphi(P') \in \Delta_1$ and let $\theta(P')$ be the conjunction of all sentences $\varphi(P')$ such that either $\varphi(P) \in \Delta_0$ or $\varphi(P') \in \Delta_1$. Then $\theta(P) \wedge \theta(P') \models P(c_1, \dots, c_n) \rightarrow P'(c_1, \dots, c_n)$. We can re-arrange this so that each predicate symbol occurs on one side of \models :

$$\theta(P) \wedge P(c_1, \dots, c_n) \models \theta(P') \rightarrow P'(c_1, \dots, c_n).$$

By Craig's Interpolation Theorem there is a sentence $\chi(c_1, \dots, c_n)$ not containing P or P' such that:

$$\theta(P) \wedge P(c_1, \dots, c_n) \models \chi(c_1, \dots, c_n); \quad \chi(c_1, \dots, c_n) \models \theta(P') \rightarrow P'(c_1, \dots, c_n).$$

From the former of these two entailments we have: $\theta(P) \models P(c_1, \dots, c_n) \rightarrow \chi(c_1, \dots, c_n)$. And from the latter, since an $\mathcal{L} \cup \{P\}$ -model $(\mathfrak{M}, R) \models \varphi(P)$

CHAPTER 14. THE INTERPOLATION THEOREM

if and only if the corresponding $\mathcal{L} \cup \{P'\}$ -model $(\mathfrak{M}, R) \models \varphi(P')$, we have $\chi(c_1, \dots, c_n) \models \theta(P) \rightarrow P(c_1, \dots, c_n)$, from which:

$$\theta(P) \models \chi(c_1, \dots, c_n) \rightarrow P(c_1, \dots, c_n).$$

Putting the two together, $\theta(P) \models P(c_1, \dots, c_n) \leftrightarrow \chi(c_1, \dots, c_n)$, and by monotony and generalization also

$$\Sigma(P) \models \forall x_1 \dots \forall x_n (P(x_1, \dots, x_n) \leftrightarrow \chi(x_1, \dots, x_n)). \quad \square$$

Chapter 15

Lindström's Theorem

15.1 Introduction

In this chapter we aim to prove Lindström's characterization of first-order logic as the maximal logic for which (given certain further constraints) the Compactness and the Downward Löwenheim-Skolem theorems hold ([Theorem 10.19](#) and [Theorem 10.23](#)). First, we need a more general characterization of the general class of logics to which the theorem applies. We will restrict ourselves to *relational* languages, i.e., languages which only contain predicate symbols and individual constants, but no function symbols.

15.2 Abstract Logics

Definition 15.1. An *abstract logic* is a pair $\langle L, \models_L \rangle$, where L is a function that assigns to each language \mathcal{L} a set $L(\mathcal{L})$ of sentences, and \models_L is a relation between structures for the language \mathcal{L} and elements of $L(\mathcal{L})$. In particular, $\langle F, \models \rangle$ is ordinary first-order logic, i.e., F is the function assigning to the language \mathcal{L} the set of first-order sentences built from the constants in \mathcal{L} , and \models is the satisfaction relation of first-order logic.

Notice that we are still employing the same notion of structure for a given language as for first-order logic, but we do not presuppose that sentences are built up from the basic symbols in \mathcal{L} in the usual way, nor that the relation \models_L is recursively defined in the same way as for first-order logic. So for instance the definition, being completely general, is intended to capture the case where sentences in $\langle L, \models_L \rangle$ contain infinitely long conjunctions or disjunction, or quantifiers other than \exists and \forall (e.g., “there are infinitely many x such that ...”), or perhaps infinitely long quantifier prefixes. To emphasize that “sentences” in $L(\mathcal{L})$ need not be ordinary sentences of first-order logic, in this chapter we use variables α, β, \dots to range over them, and reserve φ, ψ, \dots for ordinary first-order formulas.

Definition 15.2. Let $\text{Mod}_L(\alpha)$ denote the class $\{\mathfrak{M} : \mathfrak{M} \models_L \alpha\}$. If the language needs to be made explicit, we write $\text{Mod}_L^{\mathcal{L}}(\alpha)$. Two structures \mathfrak{M} and \mathfrak{N} for \mathcal{L} are *elementarily equivalent in* $\langle L, \models_L \rangle$, written $\mathfrak{M} \equiv_L \mathfrak{N}$, if the same sentences from $L(\mathcal{L})$ are true in each.

Definition 15.3. An abstract logic $\langle L, \models_L \rangle$ for the language \mathcal{L} is *normal* if it satisfies the following properties:

1. (*L-Monotony*) For languages \mathcal{L} and \mathcal{L}' , if $\mathcal{L} \subseteq \mathcal{L}'$, then $L(\mathcal{L}) \subseteq L(\mathcal{L}')$.
2. (*Expansion Property*) For each $\alpha \in L(\mathcal{L})$ there is a finite subset \mathcal{L}' of \mathcal{L} such that the relation $\mathfrak{M} \models_L \alpha$ depends only on the reduct of \mathfrak{M} to \mathcal{L}' ; i.e., if \mathfrak{M} and \mathfrak{N} have the same reduct to \mathcal{L}' then $\mathfrak{M} \models_L \alpha$ if and only if $\mathfrak{N} \models_L \alpha$.
3. (*Isomorphism Property*) If $\mathfrak{M} \models_L \alpha$ and $\mathfrak{M} \simeq \mathfrak{N}$ then also $\mathfrak{N} \models_L \alpha$.
4. (*Renaming Property*) The relation \models_L is preserved under renaming: if the language \mathcal{L}' is obtained from \mathcal{L} by replacing each symbol P by a symbol P' of the same arity and each constant c by a distinct constant c' , then for each structure \mathfrak{M} and sentence α , $\mathfrak{M} \models_L \alpha$ if and only if $\mathfrak{M}' \models_L \alpha'$, where \mathfrak{M}' is the \mathcal{L}' -structure corresponding to \mathfrak{M} and $\alpha' \in L(\mathcal{L}')$.
5. (*Boolean Property*) The abstract logic $\langle L, \models_L \rangle$ is closed under the Boolean connectives in the sense that for each $\alpha \in L(\mathcal{L})$ there is a $\beta \in L(\mathcal{L})$ such that $\mathfrak{M} \models_L \beta$ if and only if $\mathfrak{M} \not\models_L \alpha$, and for each α and β there is a γ such that $\text{Mod}_L(\gamma) = \text{Mod}_L(\alpha) \cap \text{Mod}_L(\beta)$. Similarly for atomic formulas and the other connectives.
6. (*Quantifier Property*) For each constant c in \mathcal{L} and $\alpha \in L(\mathcal{L})$ there is a $\beta \in L(\mathcal{L})$ such that

$$\text{Mod}_L^{\mathcal{L}'}(\beta) = \{\mathfrak{M} : (\mathfrak{M}, a) \in \text{Mod}_L^{\mathcal{L}}(\alpha) \text{ for some } a \in |\mathfrak{M}|\},$$

where $\mathcal{L}' = \mathcal{L} \setminus \{c\}$ and (\mathfrak{M}, a) is the expansion of \mathfrak{M} to \mathcal{L} assigning a to c .

7. (*Relativization Property*) Given a sentence $\alpha \in L(\mathcal{L})$ and symbols R, c_1, \dots, c_n not in \mathcal{L} , there is a sentence $\beta \in L(\mathcal{L} \cup \{R, c_1, \dots, c_n\})$ called the *relativization* of α to $R(x, c_1, \dots, c_n)$, such that for each structure \mathfrak{M} :

$$(\mathfrak{M}, X, b_1, \dots, b_n) \models_L \beta \text{ if and only if } \mathfrak{N} \models_L \alpha,$$

where \mathfrak{N} is the substructure of \mathfrak{M} with domain $|\mathfrak{N}| = \{a \in |\mathfrak{M}| : R^{\mathfrak{M}}(a, b_1, \dots, b_n)\}$ (see [Remark 1](#)), and $(\mathfrak{M}, X, b_1, \dots, b_n)$ is the expansion of \mathfrak{M} interpreting R, c_1, \dots, c_n by X, b_1, \dots, b_n , respectively (with $X \subseteq M^{n+1}$).

15.3. COMPACTNESS AND LÖWENHEIM-SKOLEM PROPERTIES

Definition 15.4. Given two abstract logics $\langle L_1, \models_{L_1} \rangle$ and $\langle L_2, \models_{L_2} \rangle$ we say that the latter is *at least as expressive* as the former, written $\langle L_1, \models_{L_1} \rangle \leq \langle L_2, \models_{L_2} \rangle$, if for each language \mathcal{L} and sentence $\alpha \in L_1(\mathcal{L})$ there is a sentence $\beta \in L_2(\mathcal{L})$ such that $\text{Mod}_{L_1}^{\mathcal{L}}(\alpha) = \text{Mod}_{L_2}^{\mathcal{L}}(\beta)$. The logics $\langle L_1, \models_{L_1} \rangle$ and $\langle L_2, \models_{L_2} \rangle$ are *equivalent* if $\langle L_1, \models_{L_1} \rangle \leq \langle L_2, \models_{L_2} \rangle$ and $\langle L_2, \models_{L_2} \rangle \leq \langle L_1, \models_{L_1} \rangle$.

Remark 5. First-order logic, i.e., the abstract logic $\langle F, \models \rangle$, is normal. In fact, the above properties are mostly straightforward for first-order logic. We just remark that the expansion property comes down to extensionality, and that the relativization of a sentence α to $R(x, c_1, \dots, c_n)$ is obtained by replacing each subformula $\forall x \beta$ by $\forall x (R(x, c_1, \dots, c_n) \rightarrow \beta)$. Moreover, if $\langle L, \models_L \rangle$ is normal, then $\langle F, \models \rangle \leq \langle L, \models_L \rangle$, as can be shown by induction on first-order formulas. Accordingly, with no loss in generality, we can assume that every first-order sentence belongs to every normal logic.

15.3 Compactness and Löwenheim-Skolem Properties

We now give the obvious extensions of compactness and Löwenheim-Skolem to the case of abstract logics.

Definition 15.5. An abstract logic $\langle L, \models_L \rangle$ has the *Compactness Property* if each set Γ of $L(\mathcal{L})$ -sentences is satisfiable whenever each finite $\Gamma_0 \subseteq \Gamma$ is satisfiable.

Definition 15.6. $\langle L, \models_L \rangle$ has the *Downward Löwenheim-Skolem property* if any satisfiable Γ has an enumerable model.

The notion of partial isomorphism from [Definition 12.15](#) is purely “algebraic” (i.e., given without reference to the sentences of the language but only to the constants provided by the language \mathcal{L} of the structures), and hence it applies to the case of abstract logics. In case of first-order logic, we know from [Theorem 12.17](#) that if two structures are partially isomorphic then they are elementarily equivalent. That proof does not carry over to abstract logics, for induction on formulas need not be available for arbitrary $\alpha \in L(\mathcal{L})$, but the theorem is true nonetheless, provided the Löwenheim-Skolem property holds.

Theorem 15.7. Suppose $\langle L, \models_L \rangle$ is a normal logic with the Löwenheim-Skolem property. Then any two structures that are partially isomorphic are elementarily equivalent in $\langle L, \models_L \rangle$.

Proof. Suppose $\mathfrak{M} \simeq_p \mathfrak{N}$, but for some α also $\mathfrak{M} \models_L \alpha$ while $\mathfrak{N} \not\models_L \alpha$. By the Isomorphism Property we can assume that $|\mathfrak{M}|$ and $|\mathfrak{N}|$ are disjoint, and by the Expansion Property we can assume that $\alpha \in L(\mathcal{L})$ for a finite language \mathcal{L} . Let \mathcal{I} be a set of partial isomorphisms between \mathfrak{M} and \mathfrak{N} , and with no loss of generality also assume that if $p \in \mathcal{I}$ and $q \subseteq p$ then also $q \in \mathcal{I}$.

$|\mathfrak{M}|^{<\omega}$ is the set of finite sequences of elements of $|\mathfrak{M}|$. Let S be the ternary relation over $|\mathfrak{M}|^{<\omega}$ representing concatenation, i.e., if $\mathbf{a}, \mathbf{b}, \mathbf{c} \in |\mathfrak{M}|^{<\omega}$ then $S(\mathbf{a}, \mathbf{b}, \mathbf{c})$ holds if and only if \mathbf{c} is the concatenation of \mathbf{a} and \mathbf{b} ; and let T be the ternary relation such that $T(\mathbf{a}, b, \mathbf{c})$ holds for $b \in M$ and $\mathbf{a}, \mathbf{c} \in |\mathfrak{M}|^{<\omega}$ if and only if $\mathbf{a} = a_1, \dots, a_n$ and $\mathbf{c} = a_1, \dots, a_n, b$. Pick new 3-place predicate symbols P and Q and form the structure \mathfrak{M}^* having the universe $|\mathfrak{M}| \cup |\mathfrak{M}|^{<\omega}$, having \mathfrak{M} as a substructure, and interpreting P and Q by the concatenation relations S and T (so \mathfrak{M}^* is in the language $\mathcal{L} \cup \{P, Q\}$).

Define $|\mathfrak{N}|^{<\omega}, S', T', P', Q'$ and \mathfrak{N}^* analogously. Since by hypothesis $\mathfrak{M} \simeq_p \mathfrak{N}$, there is a relation I between $|\mathfrak{M}|^{<\omega}$ and $|\mathfrak{N}|^{<\omega}$ such that $I(\mathbf{a}, \mathbf{b})$ holds if and only if \mathbf{a} and \mathbf{b} are isomorphic and satisfy the back-and-forth condition of Definition 12.15. Now, let \mathfrak{M} be the structure whose domain is the union of the domains of \mathfrak{M}^* and \mathfrak{N}^* , having \mathfrak{M}^* and \mathfrak{N}^* as substructures, in the language with one extra binary predicate symbol R interpreted by the relation I and predicate symbols denoting the domains $|\mathfrak{M}|^*$ and $|\mathfrak{N}|^*$.

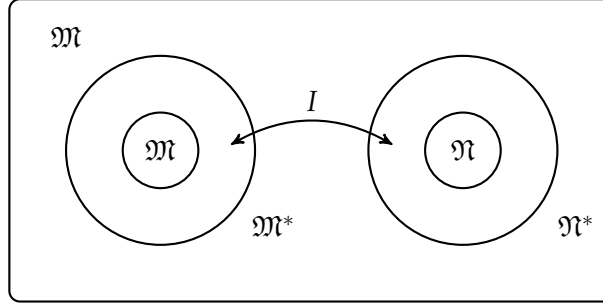


Figure 15.1: The structure \mathfrak{M} with the internal partial isomorphism.

The crucial observation is that in the language of the structure \mathfrak{M} there is a *first-order* sentence θ_1 true in \mathfrak{M} saying that $\mathfrak{M} \models_L \alpha$ and $\mathfrak{N} \not\models_L \alpha$ (this requires the Relativization Property), as well as a *first-order* sentence θ_2 true in \mathfrak{M} saying that $\mathfrak{M} \simeq_p \mathfrak{N}$ via the partial isomorphism I . By the Löwenheim-Skolem Property, θ_1 and θ_2 are jointly true in an enumerable model \mathfrak{M}_0 containing partially isomorphic substructures \mathfrak{M}_0 and \mathfrak{N}_0 such that $\mathfrak{M}_0 \models_L \alpha$ and $\mathfrak{N}_0 \not\models_L \alpha$. But enumerable partially isomorphic structures are in fact isomorphic by Theorem 12.16, contradicting the Isomorphism Property of normal abstract logics. \square

15.4 Lindström's Theorem

Lemma 15.8. Suppose $\alpha \in L(\mathcal{L})$, with \mathcal{L} finite, and assume also that there is an $n \in \mathbb{N}$ such that for any two structures \mathfrak{M} and \mathfrak{N} , if $\mathfrak{M} \equiv_n \mathfrak{N}$ and $\mathfrak{M} \models_L \alpha$ then also $\mathfrak{N} \models_L \alpha$. Then α is equivalent to a first-order sentence, i.e., there is a first-order θ such that $\text{Mod}_L(\alpha) = \text{Mod}_L(\theta)$.

15.4. LINDSTRÖM'S THEOREM

Proof. Let n be such that any two n -equivalent structures \mathfrak{M} and \mathfrak{N} agree on the value assigned to α . Recall [Proposition 12.19](#): there are only finitely many first-order sentences in a finite language that have quantifier rank no greater than n , up to logical equivalence. Now, for each fixed structure \mathfrak{M} let $\theta_{\mathfrak{M}}$ be the conjunction of all first-order sentences α true in \mathfrak{M} with $\text{qr}(\alpha) \leq n$ (this conjunction is finite), so that $\mathfrak{N} \models \theta_{\mathfrak{M}}$ if and only if $\mathfrak{N} \equiv_n \mathfrak{M}$. Then put $\theta = \bigvee \{\theta_{\mathfrak{M}} : \mathfrak{M} \models_L \alpha\}$; this disjunction is also finite (up to logical equivalence).

The conclusion $\text{Mod}_L(\alpha) = \text{Mod}_L(\theta)$ follows. In fact, if $\mathfrak{N} \models_L \theta$ then for some $\mathfrak{M} \models_L \alpha$ we have $\mathfrak{N} \models \theta_{\mathfrak{M}}$, whence also $\mathfrak{N} \models_L \alpha$ (by the hypothesis of the lemma). Conversely, if $\mathfrak{N} \models_L \alpha$ then $\theta_{\mathfrak{N}}$ is a disjunct in θ , and since $\mathfrak{N} \models \theta_{\mathfrak{N}}$, also $\mathfrak{N} \models_L \theta$. \square

Theorem 15.9 (Lindström's Theorem). *Suppose $\langle L, \models_L \rangle$ has the Compactness and the Löwenheim-Skolem Properties. Then $\langle L, \models_L \rangle \leq \langle F, \models \rangle$ (so $\langle L, \models_L \rangle$ is equivalent to first-order logic).*

Proof. By [Lemma 15.8](#), it suffices to show that for any $\alpha \in L(\mathcal{L})$, with \mathcal{L} finite, there is $n \in \mathbb{N}$ such that for any two structures \mathfrak{M} and \mathfrak{N} : if $\mathfrak{M} \equiv_n \mathfrak{N}$ then \mathfrak{M} and \mathfrak{N} agree on α . For then α is equivalent to a first-order sentence, from which $\langle L, \models_L \rangle \leq \langle F, \models \rangle$ follows. Since we are working in a finite, purely relational language, by [Theorem 12.23](#) we can replace the statement that $\mathfrak{M} \equiv_n \mathfrak{N}$ by the corresponding algebraic statement that $I_n(\emptyset, \emptyset)$.

Given α , suppose towards a contradiction that for each n there are structures \mathfrak{M}_n and \mathfrak{N}_n such that $I_n(\emptyset, \emptyset)$, but (say) $\mathfrak{M}_n \models_L \alpha$ whereas $\mathfrak{N}_n \not\models_L \alpha$. By the Isomorphism Property we can assume that all the \mathfrak{M}_n 's interpret the constants of the language by the same objects; furthermore, since there are only finitely many atomic sentences in the language, we may also assume that they satisfy the same atomic sentences (we can take a subsequence of the \mathfrak{M} 's otherwise). Let \mathfrak{M} be the union of all the \mathfrak{M}_n 's, i.e., the unique minimal structure having each \mathfrak{M}_n as a substructure. As in the proof of [Theorem 15.7](#), let \mathfrak{M}^* be the extension of \mathfrak{M} with domain $|\mathfrak{M}| \cup |\mathfrak{M}|^{<\omega}$, in the expanded language comprising the concatenation predicates P and Q .

Similarly, define \mathfrak{N}_n , \mathfrak{N} and \mathfrak{N}^* . Now let \mathfrak{M} be the structure whose domain comprises the domains of \mathfrak{M}^* and \mathfrak{N}^* as well as the natural numbers \mathbb{N} along with their natural ordering \leq , in the language with extra predicates representing the domains $|\mathfrak{M}|$, $|\mathfrak{N}|$, $|\mathfrak{M}|^{<\omega}$ and $|\mathfrak{N}|^{<\omega}$ as well as predicates coding the domains of \mathfrak{M}_n and \mathfrak{N}_n in the sense that:

$$\begin{aligned} |\mathfrak{M}_n| &= \{a \in |\mathfrak{M}| : R(a, n)\}; & |\mathfrak{N}_n| &= \{a \in |\mathfrak{N}| : S(a, n)\}; \\ |\mathfrak{M}_n|^{<\omega} &= \{a \in |\mathfrak{M}|^{<\omega} : R(a, n)\}; & |\mathfrak{N}_n|^{<\omega} &= \{a \in |\mathfrak{N}|^{<\omega} : S(a, n)\}. \end{aligned}$$

The structure \mathfrak{M} also has a ternary relation J such that $J(n, \mathbf{a}, \mathbf{b})$ holds if and only if $I_n(\mathbf{a}, \mathbf{b})$.

Now there is a sentence θ in the language \mathcal{L} augmented by R , S , J , etc., saying that \leq is a discrete linear ordering with first but no last element and

such that $\mathfrak{M}_n \models \alpha$, $\mathfrak{N}_n \not\models \alpha$, and for each n in the ordering, $J(n, \mathbf{a}, \mathbf{b})$ holds if and only if $I_n(\mathbf{a}, \mathbf{b})$.

Using the Compactness Property, we can find a model \mathfrak{M}^* of θ in which the ordering contains a non-standard element n^* . In particular then \mathfrak{M}^* will contain substructures \mathfrak{M}_{n^*} and \mathfrak{N}_{n^*} such that $\mathfrak{M}_{n^*} \models_L \alpha$ and $\mathfrak{N}_{n^*} \not\models_L \alpha$. But now we can define a set \mathcal{I} of pairs of k -tuples from $|\mathfrak{M}_{n^*}|$ and $|\mathfrak{N}_{n^*}|$ by putting $\langle \mathbf{a}, \mathbf{b} \rangle \in \mathcal{I}$ if and only if $J(n^* - k, \mathbf{a}, \mathbf{b})$, where k is the length of \mathbf{a} and \mathbf{b} . Since n^* is non-standard, for each standard k we have that $n^* - k > 0$, and the set \mathcal{I} witnesses the fact that $\mathfrak{M}_{n^*} \simeq_p \mathfrak{N}_{n^*}$. But by [Theorem 15.7](#), \mathfrak{M}_{n^*} is L -equivalent to \mathfrak{N}_{n^*} , a contradiction. \square

Part IV

Computability

CHAPTER 15. LINDSTRÖM'S THEOREM

This part is based on Jeremy Avigad's notes on computability theory. Only the chapter on recursive functions contains exercises yet, and everything could stand to be expanded with motivation, examples, details, and exercises.

Chapter 16

Recursive Functions

These are Jeremy Avigad's notes on recursive functions, revised and expanded by Richard Zach. This chapter does contain some exercises, and can be included independently to provide the basis for a discussion of arithmetization of syntax.

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 $f: \mathbb{N}^n \rightarrow \mathbb{N}$ that can be computed. So it is not surprising that at the beginning of the theory of computability, it was such functions that were studied. The most familiar examples of computable numerical functions, such as addition, multiplication, exponentiation (of natural numbers) share an interesting feature: they can be defined *recursively*. It is thus quite natural to attempt a general definition of *computable function* on the basis of recursive definitions. Among the many possible ways to define number-theoretic functions recursively, one particularly simple pattern of definition here becomes central: so-called *primitive recursion*.

In addition to computable functions, we might be interested in computable sets and relations. A set is computable if we can compute the answer to whether or not a given number is an element of the set, and a relation is computable iff we can compute whether or not a tuple $\langle n_1, \dots, n_k \rangle$ 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

Suppose we specify that a certain function l from \mathbb{N} to \mathbb{N} satisfies the following two clauses:

$$\begin{aligned} l(0) &= 1 \\ l(x+1) &= 2 \cdot l(x). \end{aligned}$$

It is pretty clear that there is only one function, l , that meets these two criteria. This is an instance of a *definition by primitive recursion*. We can define even more fundamental functions like addition and multiplication by

$$\begin{aligned} f(x, 0) &= x \\ f(x, y+1) &= f(x, y) + 1 \end{aligned}$$

and

$$\begin{aligned} g(x, 0) &= 0 \\ g(x, y+1) &= f(g(x, y), x). \end{aligned}$$

Exponentiation can also be defined recursively, by

$$\begin{aligned} h(x, 0) &= 1 \\ h(x, y+1) &= g(h(x, y), x). \end{aligned}$$

We can also compose functions to build more complex ones; for example,

$$\begin{aligned} k(x) &= x^x + (x+3) \cdot x \\ &= f(h(x, x), g(f(x, 3), x)). \end{aligned}$$

Let $\text{zero}(x)$ be the function that always returns 0, regardless of what x is, and let $\text{succ}(x) = x + 1$ be the successor function. The set of *primitive recursive*

16.2. PRIMITIVE RECURSION

functions is the set of functions from \mathbb{N}^n to \mathbb{N} that you get if you start with zero and succ by iterating the two operations above, primitive recursion and composition. The idea is that primitive recursive functions are defined in a straightforward and explicit way, so that it is intuitively clear that each one can be computed using finite means.

Definition 16.1. If f is a k -place function and g_0, \dots, g_{k-1} are l -place functions on the natural numbers, the *composition* of f with g_0, \dots, g_{k-1} is the l -place function h defined by

$$h(x_0, \dots, x_{l-1}) = f(g_0(x_0, \dots, x_{l-1}), \dots, g_{k-1}(x_0, \dots, x_{l-1})).$$

Definition 16.2. If f is a k -place function and g is a $(k+2)$ -place function, then the function defined by *primitive recursion from f and g* is the $(k+1)$ -place function h defined by the equations

$$\begin{aligned} h(0, z_0, \dots, z_{k-1}) &= f(z_0, \dots, z_{k-1}) \\ h(x+1, z_0, \dots, z_{k-1}) &= g(x, h(x, z_0, \dots, z_{k-1}), z_0, \dots, z_{k-1}) \end{aligned}$$

In addition to zero and succ, we will include among primitive recursive functions the projection functions,

$$P_i^n(x_0, \dots, x_{n-1}) = x_i,$$

for each natural number n and $i < n$. These are not terribly exciting in themselves: P_i^n is simply the k -place function that always returns its i th argument. But they allow us to define new functions by disregarding arguments or switching arguments, as we'll see later.

In the end, we have the following:

Definition 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_i^n is primitive recursive.
4. If f is a k -place primitive recursive function and g_0, \dots, g_{k-1} are l -place primitive recursive functions, then the composition of f with g_0, \dots, g_{k-1} is primitive recursive.
5. If f is a k -place primitive recursive function and g is a $k+2$ -place primitive recursive function, then the function defined by primitive recursion from f and g is primitive recursive.

Put more concisely, the set of primitive recursive functions is the smallest set containing zero, succ, and the projection functions P_j^n , and which is closed under composition and primitive recursion.

Another way of describing the set of primitive recursive functions keeps track of the “stage” at which a function enters the set. Let S_0 denote the set of starting functions: zero, succ, and the projections. Once S_i has been defined, let S_{i+1} be the set of all functions you get by applying a single instance of composition or primitive recursion to functions in S_i . Then

$$S = \bigcup_{i \in \mathbb{N}} S_i$$

is the set of all primitive recursive functions

Our definition of composition may seem too rigid, since g_0, \dots, g_{k-1} are all required to have the same arity l . (Remember that the *arity* of a function is the number of arguments; an l -place function has arity l .) But adding the projection functions provides the desired flexibility. For example, suppose f and g are 3-place functions and h is the 2-place function defined by

$$h(x, y) = f(x, g(x, x, y), y).$$

The definition of h can be rewritten with the projection functions, as

$$h(x, y) = f(P_0^2(x, y), g(P_0^2(x, y), P_0^2(x, y), P_1^2(x, y)), P_1^2(x, y)).$$

Then h is the composition of f with P_0^2 , l , and P_1^2 , where

$$l(x, y) = g(P_0^2(x, y), P_0^2(x, y), P_1^2(x, y)),$$

i.e., l is the composition of g with P_0^2 , P_0^2 , and P_1^2 .

For another example, let us again consider addition. This is described recursively by the following two equations:

$$\begin{aligned} x + 0 &= x \\ x + (y + 1) &= \text{succ}(x + y). \end{aligned}$$

In other words, addition is the function add defined recursively by the equations

$$\begin{aligned} \text{add}(0, x) &= x \\ \text{add}(y + 1, x) &= \text{succ}(\text{add}(y, x)). \end{aligned}$$

But even this is not a strict primitive recursive definition; we need to put it in the form

$$\begin{aligned} \text{add}(0, x) &= f(x) \\ \text{add}(y + 1, x) &= g(y, \text{add}(y, x), x) \end{aligned}$$

16.3. PRIMITIVE RECURSIVE FUNCTIONS ARE COMPUTABLE

for some 1-place primitive recursive function f and some 3-place primitive recursive function g . We can take f to be P_0^1 , and we can define g using composition,

$$g(y, w, x) = \text{succ}(P_1^3(y, w, x)).$$

The function g , being the composition of basic primitive recursive functions, is primitive recursive; and hence so is h . (Note that, strictly speaking, we have defined the function $g(y, x)$ meeting the recursive specification of $x + y$; in other words, the variables are in a different order. Luckily, addition is commutative, so here the difference is not important; otherwise, we could define the function g' by

$$g'(x, y) = g(P_1^2(y, x), P_0^2(y, x)) = g(y, x),$$

using composition.

One advantage to having the precise description of the primitive recursive functions is that we can be systematic in describing them. For example, we can assign a “notation” to each such function, as follows. Use symbols zero , succ , and P_i^n for zero, successor, and the projections. Now suppose f is defined by composition from a k -place function h and l -place functions g_0, \dots, g_{k-1} , and we have assigned notations H, G_0, \dots, G_{k-1} to the latter functions. Then, using a new symbol $\text{Comp}_{k,l}$, we can denote the function f by $\text{Comp}_{k,l}[H, G_0, \dots, G_{k-1}]$. For the functions defined by primitive recursion, we can use analogous notations of the form $\text{Rec}_k[G, H]$, where k denotes that arity of the function being defined. With this setup, we can denote the addition function by

$$\text{Rec}_2[P_0^1, \text{Comp}_{1,3}[\text{succ}, P_1^3]].$$

Having these notations sometimes proves useful.

16.3 Primitive Recursive Functions are Computable

Suppose a function h is defined by primitive recursion

$$\begin{aligned} h(0, \vec{z}) &= f(\vec{z}) \\ h(x + 1, \vec{z}) &= g(x, h(x, \vec{z}), \vec{z}) \end{aligned}$$

and suppose the functions f and g are computable. Then $h(0, \vec{z})$ can obviously be computed, since it is just $f(\vec{z})$ which we assume is computable. $h(1, \vec{z})$ can then also be computed, since $1 = 0 + 1$ and so $h(1, \vec{z})$ is just

$$g(0, h(0, \vec{z}), \vec{z}) = g(0, f(\vec{z}), \vec{z}).$$

We can go on in this way and compute

$$\begin{aligned} h(2, \vec{z}) &= g(1, g(0, f(\vec{z}), \vec{z}), \vec{z}) \\ h(3, \vec{z}) &= g(2, g(1, g(0, f(\vec{z}), \vec{z}), \vec{z}), \vec{z}) \\ h(4, \vec{z}) &= g(3, g(2, g(1, g(0, f(\vec{z}), \vec{z}), \vec{z}), \vec{z}), \vec{z}) \\ &\vdots \end{aligned}$$

Thus, to compute $h(x, \vec{z})$ in general, successively compute $h(0, \vec{z}), h(1, \vec{z}), \dots$, until we reach $h(x, \vec{z})$.

Thus, primitive recursion yields a new computable function if the functions f and g are computable. Composition of functions also results in a computable function if the functions f and g_i are computable.

Since the basic functions zero, succ, and P_i^n are computable, and composition and primitive recursion yield computable functions from computable functions, this means that every primitive recursive function is computable.

16.4 Examples of Primitive Recursive Functions

Here are some examples of primitive recursive functions:

1. Constants: for each natural number n , the function that always returns n primitive recursive function, since it is equal to $\text{succ}(\text{succ}(\dots \text{succ}(\text{zero}(x))))$.
2. The identity function: $\text{id}(x) = x$, i.e. P_0^1
3. Addition, $x + y$
4. Multiplication, $x \cdot y$
5. Exponentiation, x^y (with 0^0 defined to be 1)
6. Factorial, $x! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot x$
7. The predecessor function, $\text{pred}(x)$, defined by

$$\text{pred}(0) = 0, \quad \text{pred}(x + 1) = x$$

8. Truncated subtraction, $x \dot{-} y$, defined by

$$x \dot{-} 0 = x, \quad x \dot{-} (y + 1) = \text{pred}(x \dot{-} y)$$

9. Maximum, $\max(x, y)$, defined by

$$\max(x, y) = x + (y \dot{-} x)$$

10. Minimum, $\min(x, y)$

16.4. EXAMPLES OF PRIMITIVE RECURSIVE FUNCTIONS

11. Distance between x and y , $|x - y|$

In our definitions, we'll often use constants n . This is ok because the constant function $\text{const}_n(x)$ is primitive recursive (defined from zero and succ). So if, e.g., we want to define the function $f(x) = 2 \cdot x$ can obtain it by composition from $\text{const}_n(x)$ and multiplication as $f(x) = \text{const}_2(x) \cdot P_0^1(x)$. We'll make use of this trick from now on.

You'll also have noticed that the definition of pred does not, strictly speaking, fit into the pattern of definition by primitive recursion, since that pattern requires an extra argument. It is also odd in that it does not actually $\text{pred}(x)$ in the definition of $\text{pred}(x + 1)$. But we can define $\text{pred}'(x, y)$ by

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

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

The set of primitive recursive functions is further closed under the following two operations:

1. Finite sums: if $f(x, \vec{z})$ is primitive recursive, then so is the function

$$g(y, \vec{z}) = \sum_{x=0}^y f(x, \vec{z}).$$

2. Finite products: if $f(x, \vec{z})$ is primitive recursive, then so is the function

$$h(y, \vec{z}) = \prod_{x=0}^y f(x, \vec{z}).$$

For example, finite sums are defined recursively by the equations

$$g(0, \vec{z}) = f(0, \vec{z}), \quad g(y + 1, \vec{z}) = g(y, \vec{z}) + f(y + 1, \vec{z}).$$

We can also define boolean operations, where 1 stands for true, and 0 for false:

1. Negation, $\text{not}(x) = 1 \dot{-} x$
2. Conjunction, $\text{and}(x, y) = x \cdot y$

Other classical boolean operations like $\text{or}(x, y)$ and $\text{ifthen}(x, y)$ can be defined from these in the usual way.

16.5 Primitive Recursive Relations

Definition 16.4. A relation $R(\vec{x})$ is said to be primitive recursive if its characteristic function,

$$\chi_R(\vec{x}) = \begin{cases} 1 & \text{if } R(\vec{x}) \\ 0 & \text{otherwise} \end{cases}$$

is primitive recursive.

In other words, when one speaks of a primitive recursive relation $R(\vec{x})$, one is referring to a relation of the form $\chi_R(\vec{x}) = 1$, where χ_R is a primitive recursive function which, on any input, returns either 1 or 0. For example, the relation $\text{IsZero}(x)$, 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 $\text{IsZero}(x \dot{-} y)$

Furthermore, the set of primitive recursive relations is closed under boolean operations:

1. Negation, $\neg P$
2. Conjunction, $P \wedge Q$
3. Disjunction, $P \vee Q$
4. If ... then, $P \rightarrow Q$

are all primitive recursive, if P and Q are. For suppose $\chi_P(\vec{z})$ and $\chi_Q(\vec{z})$ are primitive recursive. Then the relation $R(\vec{z})$ that holds iff both $P(\vec{z})$ and $Q(\vec{z})$ hold has the characteristic function $\chi_R(\vec{z}) = \text{and}(\chi_P(\vec{z}), \chi_Q(\vec{z}))$.

One can also define relations using bounded quantification:

1. Bounded universal quantification: if $R(x, \vec{z})$ is a primitive recursive relation, then so is the relation

$$(\forall x < y) R(x, \vec{z})$$

which holds if and only if $R(x, \vec{z})$ holds for every x less than y .

2. Bounded existential quantification: if $R(x, \vec{z})$ is a primitive recursive relation, then so is

$$(\exists x < y) R(x, \vec{z}).$$

16.6. BOUNDED MINIMIZATION

By convention, we take $(\forall x < 0) R(x, \vec{z})$ to be true (for the trivial reason that there *are* no x less than 0) and $(\exists x < 0) R(x, \vec{z})$ to be false. A universal quantifier functions just like a finite product; it can also be defined directly by

$$g(0, \vec{z}) = 1, \quad g(y+1, \vec{z}) = \text{and}(g(y, \vec{z}), \chi_R(y, \vec{z})).$$

Bounded existential quantification can similarly be defined using *or*. Alternatively, it can be defined from bounded universal quantification, using the equivalence, $(\exists x < y) \varphi(x) \leftrightarrow \neg(\forall x < y) \neg\varphi(x)$. Note that, for example, a bounded quantifier of the form $(\exists x \leq y) \dots x \dots$ is equivalent to $(\exists x < y+1) \dots x \dots$.

Another useful primitive recursive function is:

1. The conditional function, $\text{cond}(x, y, z)$, defined by

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

This is defined recursively by

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

One can use this to justify:

1. Definition by cases: if $g_0(\vec{x}), \dots, g_m(\vec{x})$ are functions, and $R_1(\vec{x}), \dots, R_{m-1}(\vec{x})$ are relations, then the function f defined by

$$f(\vec{x}) = \begin{cases} g_0(\vec{x}) & \text{if } R_0(\vec{x}) \\ g_1(\vec{x}) & \text{if } R_1(\vec{x}) \text{ and not } R_0(\vec{x}) \\ \vdots & \\ g_{m-1}(\vec{x}) & \text{if } R_{m-1}(\vec{x}) \text{ and none of the previous hold} \\ g_m(\vec{x}) & \text{otherwise} \end{cases}$$

is also primitive recursive.

When $m = 1$, this is just the function defined by

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

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

16.6 Bounded Minimization

It is often useful to define a function as the least number satisfying some property or relation P . If P is decidable, we can compute this function simply by trying out all the possible numbers, $0, 1, 2, \dots$, 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 y and z to the least $x < y$ such that $R(x, z)$. It, too, can be computed, by testing whether $R(0, z)$, $R(1, z)$, \dots , $R(y - 1, z)$. But why is it primitive recursive?

Proposition 16.5. *If $R(x, \vec{z})$ is primitive recursive, so is the function $m_R(y, \vec{z})$ which returns the least x less than y such that $R(x, \vec{z})$ holds, if there is one, and 0 otherwise. We will write the function m_R as*

$$(\min x < y) R(x, \vec{z}),$$

Proof. Note that there can be no $x < 0$ such that $R(x, \vec{z})$ since there is no $x < 0$ at all. So $m_R(x, 0) = 0$.

In case the bound is $y + 1$ we have three cases: (a) There is an $x < y$ such that $R(x, \vec{z})$, in which case $m_R(y + 1, \vec{z}) = m_R(y, \vec{z})$. (b) There is no such x but $R(y, \vec{z})$ holds, then $m_R(y + 1, \vec{z}) = y$. (c) There is no $x < y + 1$ such that $R(x, \vec{z})$, then $m_R(y + 1, \vec{z}) = 0$. So,

$$m_R(0, \vec{z}) = 0$$

$$m_R(y + 1, \vec{z}) = \begin{cases} m_R(y, \vec{z}) & \text{if } (\exists x < y) R(x, \vec{z}) \\ y & \text{otherwise, provided } R(y, \vec{z}) \\ 0 & \text{otherwise.} \end{cases}$$

□

The choice of “0 otherwise” is somewhat arbitrary. It is in fact even easier to recursively define the function m'_R which returns the least x less than y such that $R(x, \vec{z})$ holds, and $y + 1$ otherwise. When we use \min , however, we will always know that the least x such that $R(x, \vec{z})$ exists and is less than y . Thus, in practice, we will not have to worry about the possibility that if $(\min x < y) R(x, \vec{z}) = 0$ we do not know if that value indicates that $R(0, \vec{z})$ or that for no $x < y$, $R(x, \vec{z})$. As with bounded quantification, $(\min x \leq y) \dots$ can be understood as $(\min x < y + 1) \dots$.

16.7 Primes

Bounded quantification and bounded minimization provide us with a good deal of machinery to show that natural functions and relations are primitive recursive. For example, consider the relation “ x divides y ”, written $x \mid y$. $x \mid y$ holds if division of x by y is possible without remainder, i.e., if y is an integer multiple of x . (If it doesn't hold, i.e., the remainder when dividing x by y is > 0 , we write $x \nmid y$.) In other words, $x \mid y$ iff for some z , $x \cdot z = y$.

16.8. SEQUENCES

Obviously, any such z , if it exists, must be $\leq y$. So, we have that $x \mid y$ iff for some $z \leq y$, $x \cdot z = y$. We can define the relation $x \mid y$ by bounded existential quantification from $=$ and multiplication by

$$x \mid y \Leftrightarrow (\exists z \leq y) (x \cdot z) = y.$$

We've thus shown that $x \mid y$ is primitive recursive.

A natural number x is *prime* if it is neither 0 nor 1 and is only divisible by 1 and itself. In other words, prime numbers are such that, whenever $y \mid x$, either $y = 1$ or $y = x$. To test if x is prime, we only have to check if $y \mid x$ for all $y \leq x$, since if $y > x$, then automatically $y \nmid x$. So, the relation $\text{Prime}(x)$, which holds iff x is prime, can be defined by

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

and is thus primitive recursive.

The primes are 2, 3, 5, 7, 11, etc. Consider the function $p(x)$ which returns the x th 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 $\text{nextPrime}(x)$, which returns the first prime number larger than x , p can be easily defined using primitive recursion:

$$\begin{aligned} p(0) &= 2 \\ p(x+1) &= \text{nextPrime}(p(x)) \end{aligned}$$

Since $\text{nextPrime}(x)$ is the least y such that $y > x$ and y is prime, it can be easily computed by unbounded search. But it can also be defined by bounded minimization, thanks to a result due to Euclid: there is always a prime number between x and $x! + 1$.

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

This shows, that $\text{nextPrime}(x)$ and hence $p(x)$ are (not just computable but) primitive recursive.

(If you're curious, here's a quick proof of Euclid's theorem. Suppose p_n is the largest prime $\leq x$ and consider the product $p = p_0 \cdot p_1 \cdot \dots \cdot p_n$ 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 $\leq x! + 1$.)

16.8 Sequences

The set of primitive recursive functions is remarkably robust. But we will be able to do even more once we have developed an adequate means of handling

sequences. We will identify finite sequences of natural numbers with natural numbers in the following way: the sequence $\langle a_0, a_1, a_2, \dots, a_k \rangle$ corresponds to the number

$$p_0^{a_0+1} \cdot p_1^{a_1+1} \cdot p_2^{a_2+1} \cdot \dots \cdot p_k^{a_k+1}.$$

We add one to the exponents to guarantee that, for example, the sequences $\langle 2, 7, 3 \rangle$ and $\langle 2, 7, 3, 0, 0 \rangle$ have distinct numeric codes. We can take both 0 and 1 to code the empty sequence; for concreteness, let \emptyset denote 0.

Let us define the following functions:

1. $\text{len}(s)$, which returns the length of the sequence s : Let $R(i, s)$ be the relation defined by

$$R(i, s) \text{ iff } p_i \mid s \wedge (\forall j < s) (j > i \rightarrow p_j \nmid s)$$

R is primitive recursive. Now let

$$\text{len}(s) = \begin{cases} 0 & \text{if } s = 0 \text{ or } s = 1 \\ 1 + (\min i < s) R(i, s) & \text{otherwise} \end{cases}$$

Note that we need to bound the search on i ; clearly s provides an acceptable bound.

2. $\text{append}(s, a)$, which returns the result of appending a to the sequence s :

$$\text{append}(s, a) = \begin{cases} 2^{a+1} & \text{if } s = 0 \text{ or } s = 1 \\ s \cdot p_{\text{len}(s)}^{a+1} & \text{otherwise} \end{cases}$$

3. $\text{element}(s, i)$, which returns the i th element of s (where the initial element is called the 0th), or 0 if i is greater than or equal to the length of s :

$$\text{element}(s, i) = \begin{cases} 0 & \text{if } i \geq \text{len}(s) \\ \min j < s (p_i^{j+2} \nmid s) - 1 & \text{otherwise} \end{cases}$$

Instead of using the official names for the functions defined above, we introduce a more compact notation. We will use $(s)_i$ instead of $\text{element}(s, i)$, and $\langle s_0, \dots, s_k \rangle$ to abbreviate

$$\text{append}(\text{append}(\dots \text{append}(\emptyset, s_0) \dots), s_k).$$

Note that if s has length k , the elements of s are $(s)_0, \dots, (s)_{k-1}$.

It will be useful for us to be able to bound the numeric code of a sequence in terms of its length and its largest element. Suppose s is a sequence of length k , each element of which is less than equal to some number x . Then s has at

16.9. OTHER RECURSIONS

most k prime factors, each at most p_{k-1} , and each raised to at most $x + 1$ in the prime factorization of s . In other words, if we define

$$\text{sequenceBound}(x, k) = p_{k-1}^{k \cdot (x+1)},$$

then the numeric code of the sequence s described above is at most $\text{sequenceBound}(x, k)$.

Having such a bound on sequences gives us a way of defining new functions using bounded search. For example, suppose we want to define the function $\text{concat}(s, t)$, which concatenates two sequences. One first option is to define 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

$$\text{concat}(s, t) = \text{hconcat}(s, t, \text{len}(t)).$$

But using bounded search, we can be lazy. All we need to do is write down a primitive recursive *specification* of the object (number) we are looking for, and a bound on how far to look. The following works:

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

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

16.9 Other Recursions

Using pairing and sequencing, we can justify more exotic (and useful) forms of primitive recursion. For example, it is often useful to define two functions simultaneously, such as in the following definition:

$$\begin{aligned} f_0(0, \vec{z}) &= k_0(\vec{z}) \\ f_1(0, \vec{z}) &= k_1(\vec{z}) \\ f_0(x + 1, \vec{z}) &= h_0(x, f_0(x, \vec{z}), f_1(x, \vec{z}), \vec{z}) \\ f_1(x + 1, \vec{z}) &= h_1(x, f_0(x, \vec{z}), f_1(x, \vec{z}), \vec{z}) \end{aligned}$$

This is an instance of *simultaneous recursion*. Another useful way of defining functions is to give the value of $f(x + 1, \vec{z})$ in terms of *all* the values $f(0, \vec{z})$,

$\dots, f(x, \vec{z})$, as in the following definition:

$$\begin{aligned} f(0, \vec{z}) &= g(\vec{z}) \\ f(x+1, \vec{z}) &= h(x, \langle f(0, \vec{z}), \dots, f(x, \vec{z}) \rangle, \vec{z}). \end{aligned}$$

The following schema captures this idea more succinctly:

$$f(x, \vec{z}) = h(x, \langle f(0, \vec{z}), \dots, f(x-1, \vec{z}) \rangle)$$

with the understanding that the second argument to h is just the empty sequence when x is 0. In either formulation, the idea is that in computing the “successor step,” the function f can make use of the entire sequence of values computed so far. This is known as a *course-of-values* recursion. For a particular example, it can be used to justify the following type of definition:

$$f(x, \vec{z}) = \begin{cases} h(x, f(k(x, \vec{z}), \vec{z}), \vec{z}) & \text{if } k(x, \vec{z}) < x \\ g(x, \vec{z}) & \text{otherwise} \end{cases}$$

In other words, the value of f at x can be computed in terms of the value of f at *any* previous value, given by k .

You should think about how to obtain these functions using ordinary primitive recursion. One final version of primitive recursion is more flexible in that one is allowed to change the *parameters* (side values) along the way:

$$\begin{aligned} f(0, \vec{z}) &= g(\vec{z}) \\ f(x+1, \vec{z}) &= h(x, f(x, k(\vec{z})), \vec{z}) \end{aligned}$$

This, too, can be simulated with ordinary primitive recursion. (Doing so is tricky. For a hint, try unwinding the computation by hand.)

Finally, notice that we can always extend our “universe” by defining additional objects in terms of the natural numbers, and defining primitive recursive functions that operate on them. For example, we can take an integer to be given by a pair $\langle m, n \rangle$ of natural numbers, which, intuitively, represents the integer $m - n$. In other words, we say

$$\text{Integer}(x) \Leftrightarrow \text{length}(x) = 2$$

and then we define the following:

1. $\text{iequal}(x, y)$
2. $\text{iplus}(x, y)$
3. $\text{iminus}(x, y)$
4. $\text{itimes}(x, y)$

Similarly, we can define a rational number to be a pair $\langle x, y \rangle$ of integers with $y \neq 0$, representing the value x/y . And we can define qequal , qplus , qminus , qtimes , qdivides , and so on.

16.10 Non-Primitive Recursive Functions

The primitive recursive functions do not exhaust the intuitively computable functions. It should be intuitively clear that we can make a list of all the unary primitive recursive functions, f_0, f_1, f_2, \dots 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 $g(g(\dots g(x)))$, with n g ’s in all; and define a sequence g_0, g_1, \dots 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:

$$\begin{aligned} \#(0) &= \langle 0 \rangle \\ \#(S) &= \langle 1 \rangle \\ \#(P_i^n) &= \langle 2, n, i \rangle \\ \#(\text{Comp}_{k,l}[H, G_0, \dots, G_{k-1}]) &= \langle 3, k, l, \#(H), \#(G_0), \dots, \#(G_{k-1}) \rangle \\ \#(\text{Rec}_l[G, H]) &= \langle 4, l, \#(G), \#(H) \rangle \end{aligned}$$

Here I am using the fact that every sequence of numbers can be viewed as a natural number, using the codes from the last section. The upshot is that every code is assigned a natural number. Of course, some sequences (and hence some numbers) do not correspond to notations; but we can let f_i be the unary primitive recursive function with notation coded as i , if i codes such a

notation; and the constant 0 function otherwise. The net result is that we have an explicit way of enumerating the unary primitive recursive functions.

(In fact, some functions, like the constant zero function, will appear more than once on the list. This is not just an artifact of our coding, but also a result of the fact that the constant zero function has more than one notation. We will later see that one can not computably avoid these repetitions; for example, there is no computable function that decides whether or not a given notation represents the constant zero function.)

We can now take the function $g(x, y)$ to be given by $f_x(y)$, where f_x refers to the enumeration we have just described. How do we know that $g(x, y)$ is computable? Intuitively, this is clear: to compute $g(x, y)$, first “unpack” x , and see if it 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. But, as noted above, working with Turing machines directly is unpleasant. Soon we will have built up enough machinery to show that $g(x, y)$ is computable, appealing to a model of computation that can be *simulated* on a Turing machine: namely, the recursive functions.

16.11 Partial Recursive Functions

To motivate the definition of the recursive functions, note that our proof that there are computable functions that are not primitive recursive actually establishes much more. The argument was simple: all we used was the fact that it is possible to enumerate functions f_0, f_1, \dots 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

16.11. PARTIAL RECURSIVE FUNCTIONS

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, \dots, g_k all are partial functions, then

$$h(g_0(\vec{x}), \dots, g_k(\vec{x}))$$

is defined if and only if each g_i is defined at \vec{x} , and h is defined at $g_0(\vec{x}), \dots, g_k(\vec{x})$. 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 $f(0, \vec{z}), f(1, \vec{z}), \dots, f(x, \vec{z})$ 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 $f(x, \bar{z})$ is not everywhere defined (whereas, in contrast, the characteristic function of a relation is always total).

Definition 16.6. The set of *partial recursive functions* is the smallest set of partial functions from the natural numbers to the natural numbers (of various arities) containing zero, successor, and projections, and closed under composition, primitive recursion, and unbounded search.

Of course, some of the partial recursive functions will happen to be total, i.e., defined for every argument.

Definition 16.7. The set of *recursive functions* is the set of partial recursive functions that are total.

A recursive function is sometimes called “total recursive” to emphasize that it is defined everywhere.

16.12 The Normal Form Theorem

Theorem 16.8 (Kleene’s Normal Form Theorem). *There is a primitive recursive relation $T(e, x, s)$ and a primitive recursive function $U(s)$, with the following property: if f is any partial recursive function, then for some e ,*

$$f(x) \simeq U(\mu s \, T(e, x, s))$$

for every x .

The proof of the normal form theorem is involved, but the basic idea is simple. Every partial recursive function has an *index* e , intuitively, a number coding its program or definition. If $f(x) \downarrow$, the computation can be recorded systematically and coded by some number s , and that s codes the computation of f on input x can be checked primitive recursively using only x and the definition e . This means that T is primitive recursive. Given the full record of the computation s , the “upshot” of s is the value of $f(x)$, and it can be obtained from s primitive recursively as well.

The normal form theorem shows that only a single unbounded search is required for the definition of any partial recursive function. We can use the numbers e as “names” of partial recursive functions, and write φ_e for the function f defined by the equation in the theorem. Note that any partial recursive function can have more than one index—in fact, every partial recursive function has infinitely many indices.

16.13. THE HALTING PROBLEM

16.13 The Halting Problem

The *halting problem* in general is the problem of deciding, given the specification e (e.g., program) of a computable function and a number n , whether the computation of the function on input n halts, i.e., produces a result. Famously, Alan Turing proved that this problem itself cannot be solved by a computable function, i.e., the function

$$h(e, n) = \begin{cases} 1 & \text{if computation } e \text{ halts on input } n \\ 0 & \text{otherwise,} \end{cases}$$

is not computable.

In the context of partial recursive functions, the role of the specification of a program may be played by the index e given in Kleene's normal form theorem. If f is a partial recursive function, any e for which the equation in the normal form theorem holds, is an index of f . Given a number e , the normal form theorem states that

$$\varphi_e(x) \simeq U(\mu s T(e, x, s))$$

is partial recursive, and for every partial recursive $f: \mathbb{N} \rightarrow \mathbb{N}$, there is an $e \in \mathbb{N}$ such that $\varphi_e(x) \simeq f(x)$ for all $x \in \mathbb{N}$. In fact, for each such f there is not just one, but infinitely many such e . The *halting function* h is defined by

$$h(e, x) = \begin{cases} 1 & \text{if } \varphi_e(x) \downarrow \\ 0 & \text{otherwise.} \end{cases}$$

Note that $h(e, x) = 0$ if $\varphi_e(x) \uparrow$, but also when e is not the index of a partial recursive function at all.

Theorem 16.9. *The halting function h is not partial recursive.*

Proof. If h were partial recursive, we could define

$$d(y) = \begin{cases} 1 & \text{if } h(y, y) = 0 \\ \mu x \ x \neq y & \text{otherwise.} \end{cases}$$

From this definition it follows that

1. $d(y) \downarrow$ iff $\varphi_y(y) \uparrow$ or y is not the index of a partial recursive function.
2. $d(y) \uparrow$ iff $\varphi_y(y) \downarrow$.

If h were partial recursive, then d would be partial recursive as well. Thus, by the Kleene normal form theorem, it has an index e_d . Consider the value of $h(e_d, e_d)$. There are two possible cases, 0 and 1.

1. If $h(e_d, e_d) = 1$ then $\varphi_{e_d}(e_d) \downarrow$. But $\varphi_{e_d} \simeq d$, and $d(e_d)$ is defined iff $h(e_d, e_d) = 0$. So $h(e_d, e_d) \neq 1$.
2. If $h(e_d, e_d) = 0$ then either e_d is not the index of a partial recursive function, or it is and $\varphi_{e_d}(e_d) \uparrow$. But again, $\varphi_{e_d} \simeq d$, and $d(e_d)$ is undefined iff $\varphi_{e_d}(e_d) \downarrow$.

The upshot is that e_d cannot, after all, be the index of a partial recursive function. But if h were partial recursive, d would be too, and so our definition of e_d as an index of it would be admissible. We must conclude that h cannot be partial recursive. \square

16.14 General Recursive Functions

There is another way to obtain a set of total functions. Say a total function $f(x, \vec{z})$ is *regular* if for every sequence of natural numbers \vec{z} , there is an x such that $f(x, \vec{z}) = 0$. In other words, the regular functions are exactly those functions to which one can apply unbounded search, and end up with a total function. One can, conservatively, restrict unbounded search to regular functions:

Definition 16.10. The set of *general recursive functions* is the smallest set of functions from the natural numbers to the natural numbers (of various arities) containing zero, successor, and projections, and closed under composition, primitive recursion, and unbounded search applied to *regular* functions.

Clearly every general recursive function is total. The difference between [Definition 16.10](#) and [Definition 16.7](#) is that in the latter one is allowed to use partial recursive functions along the way; the only requirement is that the function you end up with at the end is total. So the word “general,” a historic relic, is a misnomer; on the surface, [Definition 16.10](#) is *less* general than [Definition 16.7](#). But, fortunately, the difference is illusory; though the definitions are different, the set of general recursive functions and the set of recursive functions are one and the same.

Problems

Problem 16.1. Multiplication satisfies the recursive equations

$$\begin{aligned} 0 \cdot y &= y \\ (x + 1) \cdot y &= (x \cdot y) + x \end{aligned}$$

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

16.14. GENERAL RECURSIVE FUNCTIONS

Problem 16.2. Show that

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

is primitive recursive.

Problem 16.3. Show that $d(x, y) = \lfloor x/y \rfloor$ (i.e., division, where you disregard everything after the decimal point) is primitive recursive. When $y = 0$, we stipulate $d(x, y) = 0$. Give an explicit definition of d using primitive recursion and composition. You will have to detour through an auxiliary function—you cannot use recursion on the arguments x or y themselves.

Problem 16.4. Suppose $R(x, \vec{z})$ is primitive recursive. Define the function $m'_R(y, \vec{z})$ which returns the least x less than y such that $R(x, \vec{z})$ holds, if there is one, and $y + 1$ otherwise, by primitive recursion from χ_R .

Problem 16.5. Define integer division $d(x, y)$ using bounded minimization.

Problem 16.6. Show that there is a primitive recursive function $\text{sconcat}(s)$ with the property that

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

Chapter 17

The Lambda Calculus

This chapter needs to be expanded (issue #66).

17.1 Introduction

The lambda calculus was originally designed by Alonzo Church in the early 1930s as a basis for constructive logic, and *not* as a model of the computable functions. But soon after the Turing computable functions, the recursive functions, and the general recursive functions were shown to be equivalent, lambda computability was added to the list. The fact that this initially came as a small surprise makes the characterization all the more interesting.

Lambda notation is a convenient way of referring to a function directly by a symbolic expression which defines it, instead of defining a name for it. Instead of saying “let f be the function defined by $f(x) = x + 3$,” one can say, “let f be the function $\lambda x. (x + 3)$.” In other words, $\lambda x. (x + 3)$ is just a *name* for the function that adds three to its argument. In this expression, x is a dummy variable, or a placeholder: the same function can just as well be denoted by $\lambda y. (y + 3)$. The notation works even with other parameters around. For example, suppose $g(x, y)$ is a function of two variables, and k is a natural number. Then $\lambda x. g(x, k)$ is the function which maps any x to $g(x, k)$.

This way of defining a function from a symbolic expression is known as *lambda abstraction*. The flip side of lambda abstraction is *application*: assuming one has a function f (say, defined on the natural numbers), one can *apply* it to any value, like 2. In conventional notation, of course, we write $f(2)$ for the result.

What happens when you combine lambda abstraction with application? Then the resulting expression can be simplified, by “plugging” the applicand in for the abstracted variable. For example,

$$(\lambda x. (x + 3))(2)$$

17.2. THE SYNTAX OF THE LAMBDA CALCULUS

can be simplified to $2 + 3$.

Up to this point, we have done nothing but introduce new notations for conventional notions. The lambda calculus, however, represents a more radical departure from the set-theoretic viewpoint. In this framework:

1. Everything denotes a function.
2. Functions can be defined using lambda abstraction.
3. Anything can be applied to anything else.

For example, if F is a term in the lambda calculus, $F(F)$ is always assumed to be meaningful. This liberal framework is known as the *untyped* lambda calculus, where “untyped” means “no restriction on what can be applied to what.”

There is also a *typed* lambda calculus, which is an important variation on the untyped version. Although in many ways the typed lambda calculus is similar to the untyped one, it is much easier to reconcile with a classical set-theoretic framework, and has some very different properties.

Research on the lambda calculus has proved to be central in theoretical computer science, and in the design of programming languages. LISP, designed by John McCarthy in the 1950s, is an early example of a language that was influenced by these ideas.

17.2 The Syntax of the Lambda Calculus

One starts with a sequence of variables x, y, z, \dots and some constant symbols a, b, c, \dots . The set of terms is defined inductively, as follows:

1. Each variable is a term.
2. Each constant is a term.
3. If M and N are terms, so is (MN) .
4. If M is a term and x is a variable, then $(\lambda x. M)$ is a term.

The system without any constants at all is called the *pure* lambda calculus.

We will follow a few notational conventions:

1. When parentheses are left out, application takes place from left to right. For example, if M, N, P , and Q are terms, then $MNPQ$ abbreviates $((MN)P)Q$.
2. Again, when parentheses are left out, lambda abstraction is to be given the widest scope possible. For example, $\lambda x. MNP$ is read $\lambda x. (MNP)$.

3. A lambda can be used to abstract multiple variables. For example, $\lambda xyz. M$ is short for $\lambda x. \lambda y. \lambda z. M$.

For example,

$$\lambda xy. xxyx\lambda z. xz$$

abbreviates

$$\lambda x. \lambda y. (((xx)y)x)\lambda z. (xz)).$$

You should memorize these conventions. They will drive you crazy at first, but you will get used to them, and after a while they will drive you less crazy than having to deal with a morass of parentheses.

Two terms that differ only in the names of the bound variables are called α -equivalent; for example, $\lambda x. x$ and $\lambda y. y$. It will be convenient to think of these as being the “same” term; in other words, when we say that M and N are the same, we also mean “up to renamings of the bound variables.” Variables that are in the scope of a λ are called “bound”, while others are called “free.” There are no free variables in the previous example; but in

$$(\lambda z. yz)x$$

y and x are free, and z is bound.

17.3 Reduction of Lambda Terms

What can one do with lambda terms? Simplify them. If M and N are any lambda terms and x is any variable, we can use $M[N/x]$ to denote the result of substituting N for x in M , after renaming any bound variables of M that would interfere with the free variables of N after the substitution. For example,

$$(\lambda w. xxw)[yzy/x] = \lambda w. (yzy)(yzy)w.$$

Alternative notations for substitution are $[N/x]M$, $M[N/x]$, and also $M[x/N]$. Beware!

Intuitively, $(\lambda x. M)N$ and $M[N/x]$ have the same meaning; the act of replacing the first term by the second is called β -conversion. More generally, if it is possible to convert a term P to P' by β -conversion of some subterm, one says P β -reduces to P' in one step. If P can be converted to P' with any number of one-step reductions (possibly none), then P β -reduces to P' . A term that cannot be β -reduced any further is called β -irreducible, or β -normal. I will say “reduces” instead of “ β -reduces,” etc., when the context is clear.

Let us consider some examples.

1. We have

$$\begin{aligned} (\lambda x. xxy)\lambda z. z &\triangleright_1 (\lambda z. z)(\lambda z. z)y \\ &\triangleright_1 (\lambda z. z)y \\ &\triangleright_1 y \end{aligned}$$

17.4. THE CHURCH-ROSSER PROPERTY

2. “Simplifying” a term can make it more complex:

$$\begin{aligned} (\lambda x. xxy)(\lambda x. xxy) &\triangleright_1 (\lambda x. xxy)(\lambda x. xxy)y \\ &\triangleright_1 (\lambda x. xxy)(\lambda x. xxy)yy \\ &\triangleright_1 \dots \end{aligned}$$

3. It can also leave a term unchanged:

$$(\lambda x. xx)(\lambda x. xx) \triangleright_1 (\lambda x. xx)(\lambda x. xx)$$

4. Also, some terms can be reduced in more than one way; for example,

$$(\lambda x. (\lambda y. yx)z)v \triangleright_1 (\lambda y. yv)z$$

by contracting the outermost application; and

$$(\lambda x. (\lambda y. yx)z)v \triangleright_1 (\lambda x. zx)v$$

by contracting the innermost one. Note, in this case, however, that both terms further reduce to the same term, zv .

The final outcome in the last example is not a coincidence, but rather illustrates a deep and important property of the lambda calculus, known as the “Church-Rosser property.”

17.4 The Church-Rosser Property

Theorem 17.1. *Let M , N_1 , and N_2 be terms, such that $M \triangleright N_1$ and $M \triangleright N_2$. Then there is a term P such that $N_1 \triangleright P$ and $N_2 \triangleright P$.*

Corollary 17.2. *Suppose M can be reduced to normal form. Then this normal form is unique.*

Proof. If $M \triangleright N_1$ and $M \triangleright N_2$, by the previous theorem there is a term P such that N_1 and N_2 both reduce to P . If N_1 and N_2 are both in normal form, this can only happen if $N_1 = P = N_2$. \square

Finally, we will say that two terms M and N are β -equivalent, or just *equivalent*, if they reduce to a common term; in other words, if there is some P such that $M \triangleright P$ and $N \triangleright P$. This is written $M \equiv N$. Using [Theorem 17.1](#), you can check that \equiv is an equivalence relation, with the additional property that for every M and N , if $M \triangleright N$ or $N \triangleright M$, then $M \equiv N$. (In fact, one can show that \equiv is the *smallest* equivalence relation having this property.)

17.5 Representability by Lambda Terms

How can the lambda calculus serve as a model of computation? At first, it is not even clear how to make sense of this statement. To talk about computability on the natural numbers, we need to find a suitable representation for such numbers. Here is one that works surprisingly well.

Definition 17.3. For each natural number n , define the *numeral* \bar{n} to be the lambda term $\lambda x. \lambda y. (x(x(x(\dots x(y))))))$, where there are n x 's in all.

The terms \bar{n} are “iterators”: on input f , \bar{n} returns the function mapping y to $f^n(y)$. Note that each numeral is normal. We can now say what it means for a lambda term to “compute” a function on the natural numbers.

Definition 17.4. Let $f(x_0, \dots, x_{n-1})$ be an n -ary partial function from \mathbb{N} to \mathbb{N} . We say a lambda term X *represents* f if for every sequence of natural numbers m_0, \dots, m_{n-1} ,

$$X\bar{m}_0\bar{m}_1\dots\bar{m}_{n-1} \triangleright \overline{f(m_0, m_1, \dots, m_{n-1})}$$

if $f(m_0, \dots, m_{n-1})$ is defined, and $X\bar{m}_0\bar{m}_1\dots\bar{m}_{n-1}$ has no normal form otherwise.

Theorem 17.5. A function f is a partial computable function if and only if it is represented by a lambda term.

This theorem is somewhat striking. As a model of computation, the lambda calculus is a rather simple calculus; the only operations are lambda abstraction and application! From these meager resources, however, it is possible to implement any computational procedure.

17.6 Lambda Representable Functions are Computable

Theorem 17.6. If a partial function f is represented by a lambda term, it is computable.

Proof. Suppose a function f is represented by a lambda term X . Let us describe an informal procedure to compute f . On input m_0, \dots, m_{n-1} , write down the term $X\bar{m}_0\dots\bar{m}_{n-1}$. Build a tree, first writing down all the one-step reductions of the original term; below that, write all the one-step reductions of those (i.e., the two-step reductions of the original term); and keep going. If you ever reach a numeral, return that as the answer; otherwise, the function is undefined.

An appeal to Church's thesis tells us that this function is computable. A better way to prove the theorem would be to give a recursive description of this search procedure. For example, one could define a sequence primitive recursive functions and relations, “IsASubterm,” “Substitute,” “ReducesToInOneStep,”

17.7. COMPUTABLE FUNCTIONS ARE LAMBDA REPRESENTABLE

“ReductionSequence,” “Numeral,” etc. The partial recursive procedure for computing $f(m_0, \dots, m_{n-1})$ is then to search for a sequence of one-step reductions starting with $X\overline{m_0} \dots \overline{m_{n-1}}$ and ending with a numeral, and return the number corresponding to that numeral. The details are long and tedious but otherwise routine. \square

17.7 Computable Functions are Lambda Representable

Theorem 17.7. *Every computable partial function is representable by a lambda term.*

Proof. We need to show that every partial computable function f is represented by a lambda term \bar{f} . By Kleene’s normal form theorem, it suffices to show that every primitive recursive function is represented by a lambda term, and then that the functions so represented are closed under suitable compositions and unbounded search. To show that every primitive recursive function is represented by a lambda term, it suffices to show that the initial functions are represented, and that the partial functions that are represented by lambda terms are closed under composition, primitive recursion, and unbounded search. \square

We will use a more conventional notation to make the rest of the proof more readable. For example, we will write $M(x, y, z)$ instead of $Mxyz$. While this is suggestive, you should remember that terms in the untyped lambda calculus do not have associated arities; so, for the same term M , it makes just as much sense to write $M(x, y)$ and $M(x, y, z, w)$. But using this notation indicates that we are treating M as a function of three variables, and helps make the intentions behind the definitions clearer. In a similar way, we will say “define M by $M(x, y, z) = \dots$ ” instead of “define M by $M = \lambda x. \lambda y. \lambda z. \dots$ ”

17.8 The Basic Primitive Recursive Functions are Lambda Representable

Lemma 17.8. *The functions 0, S, and P_i^n are lambda representable.*

Proof. Zero, $\bar{0}$, is just $\lambda x. \lambda y. y$.

The successor function \bar{S} , is defined by $\bar{S}(u) = \lambda x. \lambda y. x(uxy)$. You should think about why this works; for each numeral \bar{n} , thought of as an iterator, and each function f , $S(\bar{n}, f)$ is a function that, on input y , applies f n times starting with y , and then applies it once more.

There is nothing to say about projections: $\bar{P}_i^n(x_0, \dots, x_{n-1}) = x_i$. In other words, by our conventions, \bar{P}_i^n is the lambda term $\lambda x_0. \dots \lambda x_{n-1}. x_i$. \square

17.9 Lambda Representable Functions Closed under Composition

Lemma 17.9. *The lambda representable functions are closed under composition.*

Proof. Suppose f is defined by composition from h, g_0, \dots, g_{k-1} . Assuming h, g_0, \dots, g_{k-1} are represented by $\bar{h}, \bar{g}_0, \dots, \bar{g}_{k-1}$, respectively, we need to find a term \bar{f} representing f . But we can simply define \bar{f} by

$$\bar{f}(x_0, \dots, x_{l-1}) = \bar{h}(\bar{g}_0(x_0, \dots, x_{l-1}), \dots, \bar{g}_{k-1}(x_0, \dots, x_{l-1})).$$

In other words, the language of the lambda calculus is well suited to represent composition. \square

17.10 Lambda Representable Functions Closed under Primitive Recursion

When it comes to primitive recursion, we finally need to do some work. We will have to proceed in stages. As before, on the assumption that we already have terms \bar{g} and \bar{h} representing functions g and h , respectively, we want a term \bar{f} representing the function f defined by

$$\begin{aligned} f(0, \vec{z}) &= g(\vec{z}) \\ f(x+1, \vec{z}) &= h(z, f(x, \vec{z}), \vec{z}). \end{aligned}$$

So, in general, given lambda terms G' and H' , it suffices to find a term F such that

$$\begin{aligned} F(\bar{0}, \vec{z}) &\equiv G'(\vec{z}) \\ F(\overline{n+1}, \vec{z}) &\equiv H'(\bar{n}, F(\bar{n}, \vec{z}), \vec{z}) \end{aligned}$$

for every natural number n ; the fact that G' and H' represent g and h means that whenever we plug in numerals \bar{m} for \vec{z} , $F(\overline{n+1}, \bar{m})$ will normalize to the right answer.

But for this, it suffices to find a term F satisfying

$$\begin{aligned} F(\bar{0}) &\equiv G \\ F(\overline{n+1}) &\equiv H(\bar{n}, F(\bar{n})) \end{aligned}$$

for every natural number n , where

$$\begin{aligned} G &= \lambda \vec{z}. G'(\vec{z}) \text{ and} \\ H(u, v) &= \lambda \vec{z}. H'(u, v(u, \vec{z}), \vec{z}). \end{aligned}$$

17.10. LAMBDA REPRESENTABLE FUNCTIONS CLOSED UNDER PRIMITIVE RECURSION

In other words, with lambda trickery, we can avoid having to worry about the extra parameters \bar{z} —they just get absorbed in the lambda notation.

Before we define the term F , we need a mechanism for handling ordered pairs. This is provided by the next lemma.

Lemma 17.10. *There is a lambda term D such that for each pair of lambda terms M and N , $D(M, N)(\bar{0}) \triangleright M$ and $D(M, N)(\bar{1}) \triangleright N$.*

Proof. First, define the lambda term K by

$$K(y) = \lambda x. y.$$

In other words, K is the term $\lambda y. \lambda x. y$. Looking at it differently, for every M , $K(M)$ is a constant function that returns M on any input.

Now define $D(x, y, z)$ by $D(x, y, z) = z(K(y))x$. Then we have

$$\begin{aligned} D(M, N, \bar{0}) &\triangleright \bar{0}(K(N))M \triangleright M \text{ and} \\ D(M, N, \bar{1}) &\triangleright \bar{1}(K(N))M \triangleright K(N)M \triangleright N, \end{aligned}$$

as required. □

The idea is that $D(M, N)$ represents the pair $\langle M, N \rangle$, and if P is assumed to represent such a pair, $P(\bar{0})$ and $P(\bar{1})$ represent the left and right projections, $(P)_0$ and $(P)_1$. We will use the latter notations.

Lemma 17.11. *The lambda representable functions are closed under primitive recursion.*

Proof. We need to show that given any terms, G and H , we can find a term F such that

$$\begin{aligned} F(\bar{0}) &\equiv G \\ F(\overline{n+1}) &\equiv H(\bar{n}, F(\bar{n})) \end{aligned}$$

for every natural number n . The idea is roughly to compute sequences of *pairs*

$$\langle \bar{0}, F(\bar{0}) \rangle, \langle \bar{1}, F(\bar{1}) \rangle, \dots,$$

using numerals as iterators. Notice that the first pair is just $\langle \bar{0}, G \rangle$. Given a pair $\langle \bar{n}, F(\bar{n}) \rangle$, the next pair, $\langle \overline{n+1}, F(\overline{n+1}) \rangle$ is supposed to be equivalent to $\langle \overline{n+1}, H(\bar{n}, F(\bar{n})) \rangle$. We will design a lambda term T that makes this one-step transition.

The details are as follows. Define $T(u)$ by

$$T(u) = \langle S((u)_0), H((u)_0, (u)_1) \rangle.$$

Now it is easy to verify that for any number n ,

$$T(\langle \bar{n}, M \rangle) \triangleright \langle \overline{n+1}, H(\bar{n}, M) \rangle.$$

As suggested above, given G and H , define $F(u)$ by

$$F(u) = (u(T, \langle \bar{0}, G \rangle))_1.$$

In other words, on input \bar{n} , F iterates T n times on $\langle \bar{0}, G \rangle$, and then returns the second component. To start with, we have

$$1. \bar{0}(T, \langle \bar{0}, G \rangle) \equiv \langle \bar{0}, G \rangle$$

$$2. F(\bar{0}) \equiv G$$

By induction on n , we can show that for each natural number one has the following:

$$1. \overline{n+1}(T, \langle \bar{0}, G \rangle) \equiv \langle \overline{n+1}, F(\overline{n+1}) \rangle$$

$$2. F(\overline{n+1}) \equiv H(\bar{n}, F(\bar{n}))$$

For the second clause, we have

$$\begin{aligned} F(\overline{n+1}) &\triangleright (\overline{n+1}(T, \langle \bar{0}, G \rangle))_1 \\ &\equiv (T(\bar{n}(T, \langle \bar{0}, G \rangle)))_1 \\ &\equiv (T(\langle \bar{n}, F(\bar{n}) \rangle))_1 \\ &\equiv (\langle \overline{n+1}, H(\bar{n}, F(\bar{n})) \rangle)_1 \\ &\equiv H(\bar{n}, F(\bar{n})). \end{aligned}$$

Here we have used the induction hypothesis on the second-to-last line. For the first clause, we have

$$\begin{aligned} \overline{n+1}(T, \langle \bar{0}, G \rangle) &\equiv T(\bar{n}(T, \langle \bar{0}, G \rangle)) \\ &\equiv T(\langle \bar{n}, F(\bar{n}) \rangle) \\ &\equiv \langle \overline{n+1}, H(\bar{n}, F(\bar{n})) \rangle \\ &\equiv \langle \overline{n+1}, F(\overline{n+1}) \rangle. \end{aligned}$$

Here we have used the second clause in the last line. So we have shown $F(\bar{0}) \equiv G$ and, for every n , $F(\overline{n+1}) \equiv H(\bar{n}, F(\bar{n}))$, which is exactly what we needed. \square

17.11 Fixed-Point Combinators

Suppose you have a lambda term g , and you want another term k with the property that k is β -equivalent to gk . Define terms

$$\text{diag}(x) = xx$$

and

$$l(x) = g(\text{diag}(x))$$

17.12. LAMBDA REPRESENTABLE FUNCTIONS CLOSED UNDER MINIMIZATION

using our notational conventions; in other words, l is the term $\lambda x. g(xx)$. Let k be the term ll . Then we have

$$\begin{aligned} k &= (\lambda x. g(xx))(\lambda x. g(xx)) \\ &\triangleright g((\lambda x. g(xx))(\lambda x. g(xx))) \\ &= gk. \end{aligned}$$

If one takes

$$Y = \lambda g. ((\lambda x. g(xx))(\lambda x. g(xx)))$$

then Yg and $g(Yg)$ reduce to a common term; so $Yg \equiv_{\beta} g(Yg)$. This is known as “Curry’s combinator.” If instead one takes

$$Y = (\lambda xg. g(xg))(\lambda xg. g(xg))$$

then in fact Yg reduces to $g(Yg)$, which is a stronger statement. This latter version of Y is known as “Turing’s combinator.”

17.12 Lambda Representable Functions Closed under Minimization

Lemma 17.12. *Suppose $f(x, y)$ is primitive recursive. Let g be defined by*

$$g(x) \simeq \mu y f(x, y).$$

Then g is represented by a lambda term.

Proof. The idea is roughly as follows. Given x , we will use the fixed-point lambda term Y to define a function $h_x(n)$ which searches for a y starting at n ; then $g(x)$ is just $h_x(0)$. The function h_x can be expressed as the solution of a fixed-point equation:

$$h_x(n) \simeq \begin{cases} n & \text{if } f(x, n) = 0 \\ h_x(n+1) & \text{otherwise.} \end{cases}$$

Here are the details. Since f is primitive recursive, it is represented by some term F . Remember that we also have a lambda term D , such that $D(M, N, \bar{0}) \triangleright M$ and $D(M, N, \bar{1}) \triangleright N$. Fixing x for the moment, to represent h_x we want to find a term H (depending on x) satisfying

$$H(\bar{n}) \equiv D(\bar{n}, H(S(\bar{n})), F(x, \bar{n})).$$

We can do this using the fixed-point term Y . First, let U be the term

$$\lambda h. \lambda z. D(z, (h(Sz)), F(x, z)),$$

and then let H be the term YU . Notice that the only free variable in H is x . Let us show that H satisfies the equation above.

By the definition of Y , we have

$$H = YU \equiv U(YU) = U(H).$$

In particular, for each natural number n , we have

$$\begin{aligned} H(\bar{n}) &\equiv U(H, \bar{n}) \\ &\triangleright D(\bar{n}, H(S(\bar{n})), F(x, \bar{n})), \end{aligned}$$

as required. Notice that if you substitute a numeral \bar{m} for x in the last line, the expression reduces to \bar{n} if $F(\bar{m}, \bar{n})$ reduces to $\bar{0}$, and it reduces to $H(S(\bar{n}))$ if $F(\bar{m}, \bar{n})$ reduces to any other numeral.

To finish off the proof, let G be $\lambda x. H(\bar{0})$. Then G represents g ; in other words, for every m , $G(\bar{m})$ reduces to $\overline{g(m)}$, if $g(m)$ is defined, and has no normal form otherwise. \square

Chapter 18

Computability Theory

Material in this chapter should be reviewed and expanded. In particular, there are no exercises yet.

18.1 Introduction

The branch of logic known as *Computability Theory* deals with issues having to do with the computability, or relative computability, of functions and sets. It is a evidence of Kleene's influence that the subject used to be known as *Recursion Theory*, and today, both names are commonly used.

Let us call a function $f: \mathbb{N} \rightarrow \mathbb{N}$ *partial computable* if it can be computed in some model of computation. If f is total we will simply say that f is *computable*. A relation R with computable characteristic function χ_R is also called computable. If f and g are partial functions, we will write $f(x) \downarrow$ to mean that f is defined at x , i.e., x is in the domain of f ; and $f(x) \uparrow$ to mean the opposite, i.e., that f is not defined at x . We will use $f(x) \simeq g(x)$ to mean that either $f(x)$ and $g(x)$ are both undefined, or they are both defined and equal.

One can explore the subject without having to refer to a specific model of computation. To do this, one shows that there is a universal partial computable function, $\text{Un}(k, x)$. This allows us to enumerate the partial computable functions. We will adopt the notation φ_k to denote the k -th unary partial computable function, defined by $\varphi_k(x) \simeq \text{Un}(k, x)$. (Kleene used $\{k\}$ for this purpose, but this notation has not been used as much recently.) Slightly more generally, we can uniformly enumerate the partial computable functions of arbitrary arities, and we will use φ_k^n to denote the k -th n -ary partial recursive function.

Recall that if $f(\vec{x}, y)$ is a total or partial function, then $\mu y f(\vec{x}, y)$ is the function of \vec{x} that returns the least y such that $f(\vec{x}, y) = 0$, assuming that all of $f(\vec{x}, 0), \dots, f(\vec{x}, y - 1)$ are defined; if there is no such y , $\mu y f(\vec{x}, y)$ is undefined.

If $R(\vec{x}, y)$ is a relation, $\mu y R(\vec{x}, y)$ is defined to be the least y such that $R(\vec{x}, y)$ is true; in other words, the least y such that *one minus* the characteristic function of R is equal to zero at \vec{x}, y .

To show that a function is computable, there are two ways one can proceed:

1. Rigorously: describe a Turing machine or partial recursive function explicitly, and show that it computes the function you have in mind;
2. Informally: describe an algorithm that computes it, and appeal to Church's thesis.

There is no fine line between the two; a detailed description of an algorithm should provide enough information so that it is relatively clear how one could, in principle, design the right Turing machine or sequence of partial recursive definitions. Fully rigorous definitions are unlikely to be informative, and we will try to find a happy medium between these two approaches; in short, we will try to find intuitive yet rigorous proofs that the precise definitions could be obtained.

18.2 Coding Computations

In every model of computation, it is possible to do the following:

1. Describe the *definitions* of computable functions in a systematic way. For instance, you can think of Turing machine specifications, recursive definitions, or programs in a programming language as providing these definitions.
2. Describe the complete record of the computation of a function given by some definition for a given input. For instance, a Turing machine computation can be described by the sequence of configurations (state of the machine, contents of the tape) for each step of computation.
3. Test whether a putative record of a computation is in fact the record of how a computable function with a given definition would be computed for a given input.
4. Extract from such a description of the complete record of a computation the value of the function for a given input. For instance, the contents of the tape in the very last step of a halting Turing machine computation is the value.

Using coding, it is possible to assign to each description of a computable function a numerical *index* in such a way that the instructions can be recovered from the index in a computable way. Similarly, the complete record of a computation can be coded by a single number as well. The resulting arithmetical

18.3. THE NORMAL FORM THEOREM

relation “ s codes the record of computation of the function with index e for input x ” and the function “output of computation sequence with code s ” are then computable; in fact, they are primitive recursive.

This fundamental fact is very powerful, and allows us to prove a number of striking and important results about computability, independently of the model of computation chosen.

18.3 The Normal Form Theorem

Theorem 18.1 (Kleene’s Normal Form Theorem). *There are a primitive recursive relation $T(k, x, s)$ and a primitive recursive function $U(s)$, with the following property: if f is any partial computable function, then for some k ,*

$$f(x) \simeq U(\mu s \, T(k, x, s))$$

for every x .

Proof Sketch. For any model of computation one can rigorously define a description of the computable function f and code such description using a natural number k . One can also rigorously define a notion of “computation sequence” which records the process of computing the function with index k for input x . These computation sequences can likewise be coded as numbers s . This can be done in such a way that (a) it is decidable whether a number s codes the computation sequence of the function with index k on input x and (b) what the end result of the computation sequence coded by s is. In fact, the relation in (a) and the function in (b) are primitive recursive. \square

In order to give a rigorous proof of the Normal Form Theorem, we would have to fix a model of computation and carry out the coding of descriptions of computable functions and of computation sequences in detail, and verify that the relation T and function U are primitive recursive. For most applications, it suffices that T and U are computable and that U is total.

It is probably best to remember the proof of the normal form theorem in slogan form: $\mu s \, T(k, x, s)$ searches for a computation sequence of the function with index k on input x , and U returns the output of the computation sequence if one can be found.

T and U can be used to define the enumeration $\varphi_0, \varphi_1, \varphi_2, \dots$. From now on, we will assume that we have fixed a suitable choice of T and U , and take the equation

$$\varphi_e(x) \simeq U(\mu s \, T(e, x, s))$$

to be the *definition* of φ_e .

Here is another useful fact:

Theorem 18.2. *Every partial computable function has infinitely many indices.*

Again, this is intuitively clear. Given any (description of) a computable function, one can come up with a different description which computes the same function (input-output pair) but does so, e.g., by first doing something that has no effect on the computation (say, test if $0 = 0$, or count to 5, etc.). The index of the altered description will always be different from the original index. Both are indices of the same function, just computed slightly differently.

18.4 The s - m - n Theorem

The next theorem is known as the “ s - m - n theorem,” for a reason that will be clear in a moment. The hard part is understanding just what the theorem says; once you understand the statement, it will seem fairly obvious.

Theorem 18.3. *For each pair of natural numbers n and m , there is a primitive recursive function s_n^m such that for every sequence $x, a_0, \dots, a_{m-1}, y_0, \dots, y_{n-1}$, we have*

$$\varphi_{s_n^m(x, a_0, \dots, a_{m-1})}^n(y_0, \dots, y_{n-1}) \simeq \varphi_x^{m+n}(a_0, \dots, a_{m-1}, y_0, \dots, y_{n-1}).$$

It is helpful to think of s_n^m as acting on *programs*. That is, s_n^m takes a program, x , for an $(m+n)$ -ary function, as well as fixed inputs a_0, \dots, a_{m-1} ; and it returns a program, $s_n^m(x, a_0, \dots, a_{m-1})$, for the n -ary function of the remaining arguments. If you think of x as the description of a Turing machine, then $s_n^m(x, a_0, \dots, a_{m-1})$ is the Turing machine that, on input y_0, \dots, y_{n-1} , prepends a_0, \dots, a_{m-1} to the input string, and runs x . Each s_n^m is then just a primitive recursive function that finds a code for the appropriate Turing machine.

18.5 The Universal Partial Computable Function

Theorem 18.4. *There is a universal partial computable function $\text{Un}(k, x)$. In other words, there is a function $\text{Un}(k, x)$ such that:*

1. $\text{Un}(k, x)$ is partial computable.
2. If $f(x)$ is any partial computable function, then there is a natural number k such that $f(x) \simeq \text{Un}(k, x)$ for every x .

Proof. Let $\text{Un}(k, x) \simeq U(\mu s T(k, x, s))$ in Kleene’s normal form theorem. □

This is just a precise way of saying that we have an effective enumeration of the partial computable functions; the idea is that if we write f_k for the function defined by $f_k(x) = \text{Un}(k, x)$, then the sequence f_0, f_1, f_2, \dots includes all the partial computable functions, with the property that $f_k(x)$ can be computed “uniformly” in k and x . For simplicity, we are using a binary function that is universal for unary functions, but by coding sequences of numbers we can easily generalize this to more arguments. For example, note that

18.6. NO UNIVERSAL COMPUTABLE FUNCTION

if $f(x, y, z)$ is a 3-place partial recursive function, then the function $g(x) \simeq f((x)_0, (x)_1, (x)_2)$ is a unary recursive function.

18.6 No Universal Computable Function

Theorem 18.5. *There is no universal computable function. In other words, the universal function $\text{Un}'(k, x) = \varphi_k(x)$ is not computable.*

Proof. This theorem says that there is no *total* computable function that is universal for the total computable functions. The proof is a simple diagonalization: if $\text{Un}'(k, x)$ were total and computable, then

$$d(x) = \text{Un}'(x, x) + 1$$

would also be total and computable. However, for every k , $d(k)$ is not equal to $\text{Un}'(k, k)$. \square

Theorem [Theorem 18.4](#) above shows that we can get around this diagonalization argument, but only at the expense of allowing partial functions. It is worth trying to understand what goes wrong with the diagonalization argument, when we try to apply it in the partial case. In particular, the function $h(x) = \text{Un}(x, x) + 1$ is partial recursive. Suppose h is the k -th function in the enumeration; what can we say about $h(k)$?

18.7 The Halting Problem

Since, in our construction, $\text{Un}(k, x)$ is defined if and only if the computation of the function coded by k produces a value for input x , it is natural to ask if we can decide whether this is the case. And in fact, it is not. For the Turing machine model of computation, this means that whether a given Turing machine halts on a given input is computationally undecidable. The following theorem is therefore known as the “undecidability of the halting problem.” I will provide two proofs below. The first continues the thread of our previous discussion, while the second is more direct.

Theorem 18.6. *Let*

$$h(k, x) = \begin{cases} 1 & \text{if } \text{Un}(k, x) \text{ is defined} \\ 0 & \text{otherwise.} \end{cases}$$

Then h is not computable.

Proof. If h were computable, we would have a universal computable function, as follows. Suppose h is computable, and define

$$\text{Un}'(k, x) = \begin{cases} f_n \text{Un}(k, x) & \text{if } h(k, x) = 1 \\ 0 & \text{otherwise.} \end{cases}$$

But now $\text{Un}'(k, x)$ is a total function, and is computable if h is. For instance, we could define g using primitive recursion, by

$$\begin{aligned} g(0, k, x) &\simeq 0 \\ g(y + 1, k, x) &\simeq \text{Un}(k, x); \end{aligned}$$

then

$$\text{Un}'(k, x) \simeq g(h(k, x), k, x).$$

And since $\text{Un}'(k, x)$ agrees with $\text{Un}(k, x)$ wherever the latter is defined, Un' is universal for those partial computable functions that happen to be total. But this contradicts [Theorem 18.5](#). \square

Proof. Suppose $h(k, x)$ were computable. Define the function g by

$$g(x) = \begin{cases} 0 & \text{if } h(x, x) = 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

The function g is partial computable; for example, one can define it as $\mu y \, h(x, x) = 0$. So, for some k , $g(x) \simeq \text{Un}(k, x)$ for every x . Is g defined at k ? If it is, then, by the definition of g , $h(k, k) = 0$. By the definition of f , this means that $\text{Un}(k, k)$ is undefined; but by our assumption that $g(k) \simeq \text{Un}(k, x)$ for every x , this means that $g(k)$ is undefined, a contradiction. On the other hand, if $g(k)$ is undefined, then $h(k, k) \neq 0$, and so $h(k, k) = 1$. But this means that $\text{Un}(k, k)$ is defined, i.e., that $g(k)$ is defined. \square

We can describe this argument in terms of Turing machines. Suppose there were a Turing machine H that took as input a description of a Turing machine K and an input x , and decided whether or not K halts on input x . Then we could build another Turing machine G which takes a single input x , calls H to decide if machine x halts on input x , and does the opposite. In other words, if H reports that x halts on input x , G goes into an infinite loop, and if H reports that x doesn't halt on input x , then G just halts. Does G halt on input G ? The argument above shows that it does if and only if it doesn't—a contradiction. So our supposition that there is a such Turing machine H , is false.

18.8 Comparison with Russell's Paradox

It is instructive to compare and contrast the arguments in this section with Russell's paradox:

1. Russell's paradox: let $S = \{x : x \notin x\}$. Then $x \in S$ if and only if $x \notin S$, a contradiction.

Conclusion: There is no such set S . Assuming the existence of a "set of all sets" is inconsistent with the other axioms of set theory.

18.8. COMPARISON WITH RUSSELL'S PARADOX

2. A modification of Russell's paradox: let F be the "function" from the set of all functions to $\{0, 1\}$, defined by

$$F(f) = \begin{cases} 1 & \text{if } f \text{ is in the domain of } f, \text{ and } f(f) = 0 \\ 0 & \text{otherwise} \end{cases}$$

A similar argument shows that $F(F) = 0$ if and only if $F(F) = 1$, a contradiction.

Conclusion: F is not a function. The "set of all functions" is too big to be the domain of a function.

3. The diagonalization argument: let f_0, f_1, \dots be the enumeration of the partial computable functions, and let $G: \mathbb{N} \rightarrow \{0, 1\}$ be defined by

$$G(x) = \begin{cases} 1 & \text{if } f_x(x) \downarrow = 0 \\ 0 & \text{otherwise} \end{cases}$$

If G is computable, then it is the function f_k for some k . But then $G(k) = 1$ if and only if $G(k) = 0$, a contradiction.

Conclusion: G is not computable. Note that according to the axioms of set theory, G is still a function; there is no paradox here, just a clarification.

That talk of partial functions, computable functions, partial computable functions, and so on can be confusing. The set of all partial functions from \mathbb{N} to \mathbb{N} is a big collection of objects. Some of them are total, some of them are computable, some are both total and computable, and some are neither. Keep in mind that when we say "function," by default, we mean a total function. Thus we have:

1. computable functions
2. partial computable functions that are not total
3. functions that are not computable
4. partial functions that are neither total nor computable

To sort this out, it might help to draw a big square representing all the partial functions from \mathbb{N} to \mathbb{N} , and then mark off two overlapping regions, corresponding to the total functions and the computable partial functions, respectively. It is a good exercise to see if you can describe an object in each of the resulting regions in the diagram.

18.9 Computable Sets

We can extend the notion of computability from computable functions to computable sets:

Definition 18.7. Let S be a set of natural numbers. Then S is *computable* iff its characteristic function is. In other words, S is computable iff the function

$$\chi_S(x) = \begin{cases} 1 & \text{if } x \in S \\ 0 & \text{otherwise} \end{cases}$$

is computable. Similarly, a relation $R(x_0, \dots, x_{k-1})$ is computable if and only if its characteristic function is.

Computable sets are also called *decidable*.

Notice that we now have a number of notions of computability: for partial functions, for functions, and for sets. Do not get them confused! The Turing machine computing a partial function returns the output of the function, for input values at which the function is defined; the Turing machine computing a set returns either 1 or 0, after deciding whether or not the input value is in the set or not.

18.10 Computably Enumerable Sets

Definition 18.8. A set is *computably enumerable* if it is empty or the range of a computable function.

Historical Remarks Computably enumerable sets are also called *recursively enumerable* instead. This is the original terminology, and today both are commonly used, as well as the abbreviations “c.e.” and “r.e.”

You should think about what the definition means, and why the terminology is appropriate. The idea is that if S is the range of the computable function f , then

$$S = \{f(0), f(1), f(2), \dots\},$$

and so f can be seen as “enumerating” the elements of S . Note that according to the definition, f need not be an increasing function, i.e., the enumeration need not be in increasing order. In fact, f need not even be injective, so that the constant function $f(x) = 0$ enumerates the set $\{0\}$.

Any computable set is computably enumerable. To see this, suppose S is computable. If S is empty, then by definition it is computably enumerable. Otherwise, let a be any element of S . Define f by

$$f(x) = \begin{cases} x & \text{if } \chi_S(x) = 1 \\ a & \text{otherwise.} \end{cases}$$

Then f is a computable function, and S is the range of f .

18.11 Equivalent Definitions of Computably Enumerable Sets

The following gives a number of important equivalent statements of what it means to be computably enumerable.

Theorem 18.9. *Let S be a set of natural numbers. Then the following are equivalent:*

1. S is computably enumerable.
2. S is the range of a partial computable function.
3. S is empty or the range of a primitive recursive function.
4. S is the domain of a partial computable function.

The first three clauses say that we can equivalently take any non-empty computably enumerable set to be enumerated by either a computable function, a partial computable function, or a primitive recursive function. The fourth clause tells us that if S is computably enumerable, then for some index e ,

$$S = \{x : \varphi_e(x) \downarrow\}.$$

In other words, S is the set of inputs on for which the computation of φ_e halts. For that reason, computably enumerable sets are sometimes called *semi-decidable*: if a number is in the set, you eventually get a “yes,” but if it isn’t, you never get a “no”!

Proof. Since every primitive recursive function is computable and every computable function is partial computable, (3) implies (1) and (1) implies (2). (Note that if S is empty, S is the range of the partial computable function that is nowhere defined.) If we show that (2) implies (3), we will have shown the first three clauses equivalent.

So, suppose S is the range of the partial computable function φ_e . If S is empty, we are done. Otherwise, let a be any element of S . By Kleene’s normal form theorem, we can write

$$\varphi_e(x) = U(\mu s T(e, x, s)).$$

In particular, $\varphi_e(x) \downarrow = y$ if and only if there is an s such that $T(e, x, s)$ and $U(s) = y$. Define $f(z)$ by

$$f(z) = \begin{cases} U((z)_1) & \text{if } T(e, (z)_0, (z)_1) \\ a & \text{otherwise.} \end{cases}$$

Then f is primitive recursive, because T and U are. Expressed in terms of Turing machines, if z codes a pair $\langle (z)_0, (z)_1 \rangle$ such that $(z)_1$ is a halting computation of machine e on input $(z)_0$, then f returns the output of the computation;

otherwise, it returns a . We need to show that S is the range of f , i.e., for any natural number y , $y \in S$ if and only if it is in the range of f . In the forwards direction, suppose $y \in S$. Then y is in the range of φ_e , so for some x and s , $T(e, x, s)$ and $U(s) = y$; but then $y = f(\langle x, s \rangle)$. Conversely, suppose y is in the range of f . Then either $y = a$, or for some z , $T(e, (z)_0, (z)_1)$ and $U((z)_1) = y$. Since, in the latter case, $\varphi_e(x) \downarrow = y$, either way, y is in S .

(The notation $\varphi_e(x) \downarrow = y$ means “ $\varphi_e(x)$ is defined and equal to y .” We could just as well use $\varphi_e(x) = y$, but the extra arrow is sometimes helpful in reminding us that we are dealing with a partial function.)

To finish up the proof of [Theorem 18.9](#), it suffices to show that (1) and (4) are equivalent. First, let us show that (1) implies (4). Suppose S is the range of a computable function f , i.e.,

$$S = \{y : \text{for some } x, f(x) = y\}.$$

Let

$$g(y) = \mu x f(x) = y.$$

Then g is a partial computable function, and $g(y)$ is defined if and only if for some x , $f(x) = y$. In other words, the domain of g is the range of f . Expressed in terms of Turing machines: given a Turing machine F that enumerates the elements of S , let G be the Turing machine that semi-decides S by searching through the outputs of F to see if a given element is in the set.

Finally, to show (4) implies (1), suppose that S is the domain of the partial computable function φ_e , i.e.,

$$S = \{x : \varphi_e(x) \downarrow\}.$$

If S is empty, we are done; otherwise, let a be any element of S . Define f by

$$f(z) = \begin{cases} (z)_0 & \text{if } T(e, (z)_0, (z)_1) \\ a & \text{otherwise.} \end{cases}$$

Then, as above, a number x is in the range of f if and only if $\varphi_e(x) \downarrow$, i.e., if and only if $x \in S$. Expressed in terms of Turing machines: given a machine M_e that semi-decides S , enumerate the elements of S by running through all possible Turing machine computations, and returning the inputs that correspond to halting computations. \square

The fourth clause of [Theorem 18.9](#) provides us with a convenient way of enumerating the computably enumerable sets: for each e , let W_e denote the domain of φ_e . Then if A is any computably enumerable set, $A = W_e$, for some e .

The following provides yet another characterization of the computably enumerable sets.

18.12. UNION AND INTERSECTION OF C.E. SETS

Theorem 18.10. *A set S is computably enumerable if and only if there is a computable relation $R(x, y)$ such that*

$$S = \{x : \exists y R(x, y)\}.$$

Proof. In the forward direction, suppose S is computably enumerable. Then for some e , $S = W_e$. For this value of e we can write S as

$$S = \{x : \exists y T(e, x, y)\}.$$

In the reverse direction, suppose $S = \{x : \exists y R(x, y)\}$. Define f by

$$f(x) \simeq \mu y \text{ Atom } Rx, y.$$

Then f is partial computable, and S is the domain of f . □

18.12 Computably Enumerable Sets are Closed under Union and Intersection

The following theorem gives some closure properties on the set of computably enumerable sets.

Theorem 18.11. *Suppose A and B are computably enumerable. Then so are $A \cap B$ and $A \cup B$.*

Proof. [Theorem 18.9](#) allows us to use various characterizations of the computably enumerable sets. By way of illustration, we will provide a few different proofs.

For the first proof, suppose A is enumerated by a computable function f , and B is enumerated by a computable function g . Let

$$\begin{aligned} h(x) &= \mu y (f(y) = x \vee g(y) = x) \text{ and} \\ j(x) &= \mu y (f((y)_0) = x \wedge g((y)_1) = x). \end{aligned}$$

Then $A \cup B$ is the domain of h , and $A \cap B$ is the domain of j .

Here is what is going on, in computational terms: given procedures that enumerate A and B , we can semi-decide if an element x is in $A \cup B$ by looking for x in either enumeration; and we can semi-decide if an element x is in $A \cap B$ for looking for x in both enumerations at the same time.

For the second proof, suppose again that A is enumerated by f and B is enumerated by g . Let

$$k(x) = \begin{cases} f(x/2) & \text{if } x \text{ is even} \\ g((x-1)/2) & \text{if } x \text{ is odd.} \end{cases}$$

Then k enumerates $A \cup B$; the idea is that k just alternates between the enumerations offered by f and g . Enumerating $A \cap B$ is trickier. If $A \cap B$ is empty, it

is trivially computably enumerable. Otherwise, let c be any element of $A \cap B$, and define l by

$$l(x) = \begin{cases} f((x)_0) & \text{if } f((x)_0) = g((x)_1) \\ c & \text{otherwise.} \end{cases}$$

In computational terms, l runs through pairs of elements in the enumerations of f and g , and outputs every match it finds; otherwise, it just stalls by outputting c .

For the last proof, suppose A is the domain of the partial function $m(x)$ and B is the domain of the partial function $n(x)$. Then $A \cap B$ is the domain of the partial function $m(x) + n(x)$.

In computational terms, if A is the set of values for which m halts and B is the set of values for which n halts, $A \cap B$ is the set of values for which both procedures halt.

Expressing $A \cup B$ as a set of halting values is more difficult, because one has to simulate m and n in parallel. Let d be an index for m and let e be an index for n ; in other words, $m = \varphi_d$ and $n = \varphi_e$. Then $A \cup B$ is the domain of the function

$$p(x) = \mu y (T(d, x, y) \vee T(e, x, y)).$$

In computational terms, on input x , p searches for either a halting computation for m or a halting computation for n , and halts if it finds either one. \square

18.13 Computably Enumerable Sets not Closed under Complement

Suppose A is computably enumerable. Is the complement of A , $\bar{A} = \mathbb{N} \setminus A$, necessarily computably enumerable as well? The following theorem and corollary show that the answer is “no.”

Theorem 18.12. *Let A be any set of natural numbers. Then A is computable if and only if both A and \bar{A} are computably enumerable.*

Proof. The forwards direction is easy: if A is computable, then \bar{A} is computable as well ($\chi_A = 1 - \chi_{\bar{A}}$), and so both are computably enumerable.

In the other direction, suppose A and \bar{A} are both computably enumerable. Let A be the domain of φ_d , and let \bar{A} be the domain of φ_e . Define h by

$$h(x) = \mu s (T(d, x, s) \vee T(e, x, s)).$$

In other words, on input x , h searches for either a halting computation of φ_d or a halting computation of φ_e . Now, if $x \in A$, it will succeed in the first case, and if $x \in \bar{A}$, it will succeed in the second case. So, h is a total computable

18.14. REDUCIBILITY

function. But now we have that for every x , $x \in A$ if and only if $T(e, x, h(x))$, i.e., if φ_e is the one that is defined. Since $T(e, x, h(x))$ is a computable relation, A is computable. \square

It is easier to understand what is going on in informal computational terms: to decide A , on input x search for halting computations of φ_e and φ_f . One of them is bound to halt; if it is φ_e , then x is in A , and otherwise, x is in \bar{A} .

Corollary 18.13. \bar{K}_0 is not computably enumerable.

Proof. We know that K_0 is computably enumerable, but not computable. If \bar{K}_0 were computably enumerable, then K_0 would be computable by [Theorem 18.12](#). \square

18.14 Reducibility

We now know that there is at least one set, K_0 , that is computably enumerable but not computable. It should be clear that there are others. The method of reducibility provides a powerful method of showing that other sets have these properties, without constantly having to return to first principles.

Generally speaking, a “reduction” of a set A to a set B is a method of transforming answers to whether or not elements are in B into answers as to whether or not elements are in A . We will focus on a notion called “many-one reducibility,” but there are many other notions of reducibility available, with varying properties. Notions of reducibility are also central to the study of computational complexity, where efficiency issues have to be considered as well. For example, a set is said to be “NP-complete” if it is in NP and every NP problem can be reduced to it, using a notion of reduction that is similar to the one described below, only with the added requirement that the reduction can be computed in polynomial time.

We have already used this notion implicitly. Define the set K by

$$K = \{x : \varphi_x(x) \downarrow\},$$

i.e., $K = \{x : x \in W_x\}$. Our proof that the halting problem is unsolvable, [Theorem 18.6](#), shows most directly that K is not computable. Recall that K_0 is the set

$$K_0 = \{\langle e, x \rangle : \varphi_e(x) \downarrow\}.$$

i.e. $K_0 = \{\langle x, e \rangle : x \in W_e\}$. It is easy to extend any proof of the uncomputability of K to the uncomputability of K_0 : if K_0 were computable, we could decide whether or not an element x is in K simply by asking whether or not the pair $\langle x, x \rangle$ is in K_0 . The function f which maps x to $\langle x, x \rangle$ is an example of a *reduction* of K to K_0 .

Definition 18.14. Let A and B be sets. Then A is said to be *many-one reducible* to B , written $A \leq_m B$, if there is a computable function f such that for every natural number x ,

$$x \in A \quad \text{if and only if} \quad f(x) \in B.$$

If A is many-one reducible to B and vice-versa, then A and B are said to be *many-one equivalent*, written $A \equiv_m B$.

If the function f in the definition above happens to be injective, A is said to be *one-one reducible* to B . Most of the reductions described below meet this stronger requirement, but we will not use this fact.

It is true, but by no means obvious, that one-one reducibility really is a stronger requirement than many-one reducibility. In other words, there are infinite sets A and B such that A is many-one reducible to B but not one-one reducible to B .

18.15 Properties of Reducibility

The intuition behind writing $A \leq_m B$ is that A is “no harder than” B . The following two propositions support this intuition.

Proposition 18.15. *If $A \leq_m B$ and $B \leq_m C$, then $A \leq_m C$.*

Proof. Composing a reduction of A to B with a reduction of B to C yields a reduction of A to C . (You should check the details!) \square

Proposition 18.16. *Let A and B be any sets, and suppose A is many-one reducible to B .*

1. *If B is computably enumerable, so is A .*
2. *If B is computable, so is A .*

Proof. Let f be a many-one reduction from A to B . For the first claim, just check that if B is the domain of a partial function g , then A is the domain of $g \circ f$:

$$\begin{aligned} x \in A & \text{ iff } f(x) \in B \\ & \text{ iff } g(f(x)) \downarrow. \end{aligned}$$

For the second claim, remember that if B is computable then B and \bar{B} are computably enumerable. It is not hard to check that f is also a many-one reduction of \bar{A} to \bar{B} , so, by the first part of this proof, A and \bar{A} are computably enumerable. So A is computable as well. (Alternatively, you can check that $\chi_A = \chi_B \circ f$; so if χ_B is computable, then so is χ_A .) \square

18.16. COMPLETE COMPUTABLY ENUMERABLE SETS

A more general notion of reducibility called *Turing reducibility* is useful in other contexts, especially for proving undecidability results. Note that by [Corollary 18.13](#), the complement of K_0 is not reducible to K_0 , since it is not computably enumerable. But, intuitively, if you knew the answers to questions about K_0 , you would know the answer to questions about its complement as well. A set A is said to be Turing reducible to B if one can determine answers to questions in A using a computable procedure that can ask questions about B . This is more liberal than many-one reducibility, in which (1) you are only allowed to ask one question about B , and (2) a “yes” answer has to translate to a “yes” answer to the question about A , and similarly for “no.” It is still the case that if A is Turing reducible to B and B is computable then A is computable as well (though, as we have seen, the analogous statement does not hold for computable enumerability).

You should think about the various notions of reducibility we have discussed, and understand the distinctions between them. We will, however, only deal with many-one reducibility in this chapter. Incidentally, both types of reducibility discussed in the last paragraph have analogues in computational complexity, with the added requirement that the Turing machines run in polynomial time: the complexity version of many-one reducibility is known as *Karp reducibility*, while the complexity version of Turing reducibility is known as *Cook reducibility*.

18.16 Complete Computably Enumerable Sets

Definition 18.17. A set A is a *complete computably enumerable set* (under many-one reducibility) if

1. A is computably enumerable, and
2. for any other computably enumerable set B , $B \leq_m A$.

In other words, complete computably enumerable sets are the “hardest” computably enumerable sets possible; they allow one to answer questions about *any* computably enumerable set.

Theorem 18.18. K , K_0 , and K_1 are all complete computably enumerable sets.

Proof. To see that K_0 is complete, let B be any computably enumerable set. Then for some index e ,

$$B = W_e = \{x : \varphi_e(x) \downarrow\}.$$

Let f be the function $f(x) = \langle e, x \rangle$. Then for every natural number x , $x \in B$ if and only if $f(x) \in K_0$. In other words, f reduces B to K_0 .

To see that K_1 is complete, note that in the proof of [Proposition 18.19](#) we reduced K_0 to it. So, by [Proposition 18.15](#), any computably enumerable set can be reduced to K_1 as well.

K can be reduced to K_0 in much the same way. □

So, it turns out that all the examples of computably enumerable sets that we have considered so far are either computable, or complete. This should seem strange! Are there any examples of computably enumerable sets that are neither computable nor complete? The answer is yes, but it wasn't until the middle of the 1950s that this was established by Friedberg and Muchnik, independently.

18.17 An Example of Reducibility

Let us consider an application of [Proposition 18.16](#).

Proposition 18.19. *Let*

$$K_1 = \{e : \varphi_e(0) \downarrow\}.$$

Then K_1 is computably enumerable but not computable.

Proof. Since $K_1 = \{e : \exists s T(e, 0, s)\}$, K_1 is computably enumerable by [Theorem 18.10](#).

To show that K_1 is not computable, let us show that K_0 is reducible to it.

This is a little bit tricky, since using K_1 we can only ask questions about computations that start with a particular input, 0. Suppose you have a smart friend who can answer questions of this type (friends like this are known as “oracles”). Then suppose someone comes up to you and asks you whether or not $\langle e, x \rangle$ is in K_0 , that is, whether or not machine e halts on input x . One thing you can do is build another machine, e_x , that, for *any* input, ignores that input and instead runs e on input x . Then clearly the question as to whether machine e halts on input x is equivalent to the question as to whether machine e_x halts on input 0 (or any other input). So, then you ask your friend whether this new machine, e_x , halts on input 0; your friend's answer to the modified question provides the answer to the original one. This provides the desired reduction of K_0 to K_1 .

Using the universal partial computable function, let f be the 3-ary function defined by

$$f(x, y, z) \simeq \varphi_x(y).$$

Note that f ignores its third input entirely. Pick an index e such that $f = \varphi_e^3$; so we have

$$\varphi_e^3(x, y, z) \simeq \varphi_x(y).$$

18.18. TOTALITY IS UNDECIDABLE

By the s - m - n theorem, there is a function $s(e, x, y)$ such that, for every z ,

$$\begin{aligned}\varphi_{s(e,x,y)}(z) &\simeq \varphi_e^3(x, y, z) \\ &\simeq \varphi_x(y).\end{aligned}$$

In terms of the informal argument above, $s(e, x, y)$ is an index for the machine that, for any input z , ignores that input and computes $\varphi_x(y)$.

In particular, we have

$$\varphi_{s(e,x,y)}(0) \downarrow \quad \text{if and only if} \quad \varphi_x(y) \downarrow.$$

In other words, $\langle x, y \rangle \in K_0$ if and only if $s(e, x, y) \in K_1$. So the function g defined by

$$g(w) = s(e, (w)_0, (w)_1)$$

is a reduction of K_0 to K_1 . □

18.18 Totality is Undecidable

Let us consider one more example of using the s - m - n theorem to show that something is noncomputable. Let Tot be the set of indices of total computable functions, i.e.

$$\text{Tot} = \{x : \text{for every } y, \varphi_x(y) \downarrow\}.$$

Proposition 18.20. *Tot is not computable.*

Proof. To see that Tot is not computable, it suffices to show that K is reducible to it. Let $h(x, y)$ be defined by

$$h(x, y) \simeq \begin{cases} 0 & \text{if } x \in K \\ \text{undefined} & \text{otherwise} \end{cases}$$

Note that $h(x, y)$ does not depend on y at all. It should not be hard to see that h is partial computable: on input x, y , we compute h by first simulating the function φ_x on input x ; if this computation halts, $h(x, y)$ outputs 0 and halts. So $h(x, y)$ is just $Z(\mu s T(x, x, s))$, where Z is the constant zero function.

Using the s - m - n theorem, there is a primitive recursive function $k(x)$ such that for every x and y ,

$$\varphi_{k(x)}(y) = \begin{cases} 0 & \text{if } x \in K \\ \text{undefined} & \text{otherwise} \end{cases}$$

So $\varphi_{k(x)}$ is total if $x \in K$, and undefined otherwise. Thus, k is a reduction of K to Tot. □

It turns out that Tot is not even computably enumerable—its complexity lies further up on the “arithmetic hierarchy.” But we will not worry about this strengthening here.

18.19 Rice's Theorem

If you think about it, you will see that the specifics of Tot do not play into the proof of [Proposition 18.20](#). We designed $h(x, y)$ to act like the constant function $j(y) = 0$ exactly when x is in K ; but we could just as well have made it act like any other partial computable function under those circumstances. This observation lets us state a more general theorem, which says, roughly, that no nontrivial property of computable functions is decidable.

Keep in mind that $\varphi_0, \varphi_1, \varphi_2, \dots$ is our standard enumeration of the partial computable functions.

Theorem 18.21 (Rice's Theorem). *Let C be any set of partial computable functions, and let $A = \{n : \varphi_n \in C\}$. If A is computable, then either C is \emptyset or C is the set of all the partial computable functions.*

An *index set* is a set A with the property that if n and m are indices which “compute” the same function, then either both n and m are in A , or neither is. It is not hard to see that the set A in the theorem has this property. Conversely, if A is an index set and C is the set of functions computed by these indices, then $A = \{n : \varphi_n \in C\}$.

With this terminology, Rice's theorem is equivalent to saying that no nontrivial index set is decidable. To understand what the theorem says, it is helpful to emphasize the distinction between *programs* (say, in your favorite programming language) and the functions they compute. There are certainly questions about programs (indices), which are syntactic objects, that are computable: does this program have more than 150 symbols? Does it have more than 22 lines? Does it have a “while” statement? Does the string “hello world” every appear in the argument to a “print” statement? Rice's theorem says that no nontrivial question about the program's *behavior* is computable. This includes questions like these: does the program halt on input 0? Does it ever halt? Does it ever output an even number?

Proof of Rice's theorem. Suppose C is neither \emptyset nor the set of all the partial computable functions, and let A be the set of indices of functions in C . We will show that if A were computable, we could solve the halting problem; so A is not computable.

Without loss of generality, we can assume that the function f which is nowhere defined is not in C (otherwise, switch C and its complement in the argument below). Let g be any function in C . The idea is that if we could decide A , we could tell the difference between indices computing f , and indices computing g ; and then we could use that capability to solve the halting problem.

18.19. RICE'S THEOREM

Here's how. Using the universal computation predicate, we can define a function

$$h(x, y) \simeq \begin{cases} \text{undefined} & \text{if } \varphi_x(x) \uparrow \\ g(y) & \text{otherwise.} \end{cases}$$

To compute h , first we try to compute $\varphi_x(x)$; if that computation halts, we go on to compute $g(y)$; and if *that* computation halts, we return the output. More formally, we can write

$$h(x, y) \simeq P_0^2(g(y), \text{Un}(x, x)).$$

where $P_0^2(z_0, z_1) = z_0$ is the 2-place projection function returning the 0-th argument, which is computable.

Then h is a composition of partial computable functions, and the right side is defined and equal to $g(y)$ just when $\text{Un}(x, x)$ and $g(y)$ are both defined.

Notice that for a fixed x , if $\varphi_x(x)$ is undefined, then $h(x, y)$ is undefined for every y ; and if $\varphi_x(x)$ is defined, then $h(x, y) \simeq g(y)$. So, for any fixed value of x , either $h(x, y)$ acts just like f or it acts just like g , and deciding whether or not $\varphi_x(x)$ is defined amounts to deciding which of these two cases holds. But this amounts to deciding whether or not $h_x(y) \simeq h(x, y)$ is in C or not, and if A were computable, we could do just that.

More formally, since h is partial computable, it is equal to the function φ_k for some index k . By the s - m - n theorem there is a primitive recursive function s such that for each x , $\varphi_{s(k,x)}(y) = h_x(y)$. Now we have that for each x , if $\varphi_x(x) \downarrow$, then $\varphi_{s(k,x)}$ is the same function as g , and so $s(k, x)$ is in A . On the other hand, if $\varphi_x(x) \uparrow$, then $\varphi_{s(k,x)}$ is the same function as f , and so $s(k, x)$ is not in A . In other words we have that for every x , $x \in K$ if and only if $s(k, x) \in A$. If A were computable, K would be also, which is a contradiction. So A is not computable. \square

Rice's theorem is very powerful. The following immediate corollary shows some sample applications.

Corollary 18.22. *The following sets are undecidable.*

1. $\{x : 17 \text{ is in the range of } \varphi_x\}$
2. $\{x : \varphi_x \text{ is constant}\}$
3. $\{x : \varphi_x \text{ is total}\}$
4. $\{x : \text{whenever } y < y', \varphi_x(y) \downarrow, \text{ and if } \varphi_x(y') \downarrow, \text{ then } \varphi_x(y) < \varphi_x(y')\}$

Proof. These are all nontrivial index sets. \square

18.20 The Fixed-Point Theorem

Let's consider the halting problem again. As temporary notation, let us write $\ulcorner \varphi_x(y) \urcorner$ for $\langle x, y \rangle$; think of this as representing a "name" for the value $\varphi_x(y)$. With this notation, we can reword one of our proofs that the halting problem is undecidable.

Question: is there a computable function h , with the following property? For every x and y ,

$$h(\ulcorner \varphi_x(y) \urcorner) = \begin{cases} 1 & \text{if } \varphi_x(y) \downarrow \\ 0 & \text{otherwise.} \end{cases}$$

Answer: No; otherwise, the partial function

$$g(x) \simeq \begin{cases} 0 & \text{if } h(\ulcorner \varphi_x(x) \urcorner) = 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

would be computable, and so have some index e . But then we have

$$\varphi_e(e) \simeq \begin{cases} 0 & \text{if } h(\ulcorner \varphi_e(e) \urcorner) = 0 \\ \text{undefined} & \text{otherwise,} \end{cases}$$

in which case $\varphi_e(e)$ is defined if and only if it isn't, a contradiction.

Now, take a look at the equation with φ_e . There is an instance of self-reference there, in a sense: we have arranged for the value of $\varphi_e(e)$ to depend on $\ulcorner \varphi_e(e) \urcorner$, in a certain way. The fixed-point theorem says that we *can* do this, in general—not just for the sake of proving contradictions.

Lemma 18.23 gives two equivalent ways of stating the fixed-point theorem. Logically speaking, the fact that the statements are equivalent follows from the fact that they are both true; but what we really mean is that each one follows straightforwardly from the other, so that they can be taken as alternative statements of the same theorem.

Lemma 18.23. *The following statements are equivalent:*

1. *For every partial computable function $g(x, y)$, there is an index e such that for every y ,*

$$\varphi_e(y) \simeq g(e, y).$$

2. *For every computable function $f(x)$, there is an index e such that for every y ,*

$$\varphi_e(y) \simeq \varphi_{f(e)}(y).$$

Proof. (1) \Rightarrow (2): Given f , define g by $g(x, y) \simeq \text{Un}(f(x), y)$. Use (1) to get an index e such that for every y ,

$$\begin{aligned} \varphi_e(y) &= \text{Un}(f(e), y) \\ &= \varphi_{f(e)}(y). \end{aligned}$$

18.20. THE FIXED-POINT THEOREM

(2) \Rightarrow (1): Given g , use the s - m - n theorem to get f such that for every x and y , $\varphi_{f(x)}(y) \simeq g(x, y)$. Use (2) to get an index e such that

$$\begin{aligned}\varphi_e(y) &= \varphi_{f(e)}(y) \\ &= g(e, y).\end{aligned}$$

This concludes the proof. \square

Before showing that statement (1) is true (and hence (2) as well), consider how bizarre it is. Think of e as being a computer program; statement (1) says that given any partial computable $g(x, y)$, you can find a computer program e that computes $g_e(y) \simeq g(e, y)$. In other words, you can find a computer program that computes a function that references the program itself.

Theorem 18.24. *The two statements in [Lemma 18.23](#) are true. Specifically, for every partial computable function $g(x, y)$, there is an index e such that for every y ,*

$$\varphi_e(y) \simeq g(e, y).$$

Proof. The ingredients are already implicit in the discussion of the halting problem above. Let $\text{diag}(x)$ be a computable function which for each x returns an index for the function $f_x(y) \simeq \varphi_x(x, y)$, i.e.

$$\varphi_{\text{diag}(x)}(y) \simeq \varphi_x(x, y).$$

Think of diag as a function that transforms a program for a 2-ary function into a program for a 1-ary function, obtained by fixing the original program as its first argument. The function diag can be defined formally as follows: first define s by

$$s(x, y) \simeq \text{Un}^2(x, x, y),$$

where Un^2 is a 3-ary function that is universal for partial computable 2-ary functions. Then, by the s - m - n theorem, we can find a primitive recursive function diag satisfying

$$\varphi_{\text{diag}(x)}(y) \simeq s(x, y).$$

Now, define the function l by

$$l(x, y) \simeq g(\text{diag}(x), y).$$

and let $\ulcorner l \urcorner$ be an index for l . Finally, let $e = \text{diag}(\ulcorner l \urcorner)$. Then for every y , we have

$$\begin{aligned}\varphi_e(y) &\simeq \varphi_{\text{diag}(\ulcorner l \urcorner)}(y) \\ &\simeq \varphi_{\ulcorner l \urcorner}(\ulcorner l \urcorner, y) \\ &\simeq l(\ulcorner l \urcorner, y) \\ &\simeq g(\text{diag}(\ulcorner l \urcorner), y) \\ &\simeq g(e, y),\end{aligned}$$

as required. \square

What's going on? Suppose you are given the task of writing a computer program that prints itself out. Suppose further, however, that you are working with a programming language with a rich and bizarre library of string functions. In particular, suppose your programming language has a function `diag` which works as follows: given an input string s , `diag` locates each instance of the symbol 'x' occurring in s , and replaces it by a quoted version of the original string. For example, given the string

```
hello x world
```

as input, the function returns

```
hello 'hello x world' world
```

as output. In that case, it is easy to write the desired program; you can check that

```
print(diag('print(diag(x))'))
```

does the trick. For more common programming languages like C++ and Java, the same idea (with a more involved implementation) still works.

We are only a couple of steps away from the proof of the fixed-point theorem. Suppose a variant of the `print` function `print(x , y)` accepts a string x and another numeric argument y , and prints the string x repeatedly, y times. Then the “program”

```
getinput(y); print(diag('getinput(y); print(diag(x), y)'), y)
```

prints itself out y times, on input y . Replacing the `getinput`—`print`—`diag` skeleton by an arbitrary function $g(x, y)$ yields

```
g(diag('g(diag(x), y)'), y)
```

which is a program that, on input y , runs g on the program itself and y . Thinking of “quoting” with “using an index for,” we have the proof above.

For now, it is o.k. if you want to think of the proof as formal trickery, or black magic. But you should be able to reconstruct the details of the argument given above. When we prove the incompleteness theorems (and the related “fixed-point theorem”) we will discuss other ways of understanding why it works.

The same idea can be used to get a “fixed point” combinator. Suppose you have a lambda term g , and you want another term k with the property that k is β -equivalent to gk . Define terms

$$\text{diag}(x) = xx$$

and

$$l(x) = g(\text{diag}(x))$$

18.21. APPLYING THE FIXED-POINT THEOREM

using our notational conventions; in other words, l is the term $\lambda x. g(xx)$. Let k be the term ll . Then we have

$$\begin{aligned} k &= (\lambda x. g(xx))(\lambda x. g(xx)) \\ &\triangleright g((\lambda x. g(xx))(\lambda x. g(xx))) \\ &= gk. \end{aligned}$$

If one takes

$$Y = \lambda g. ((\lambda x. g(xx))(\lambda x. g(xx)))$$

then Yg and $g(Yg)$ reduce to a common term; so $Yg \equiv_{\beta} g(Yg)$. This is known as “Curry’s combinator.” If instead one takes

$$Y = (\lambda xg. g(xg))(\lambda xg. g(xg))$$

then in fact Yg reduces to $g(Yg)$, which is a stronger statement. This latter version of Y is known as “Turing’s combinator.”

18.21 Applying the Fixed-Point Theorem

The fixed-point theorem essentially lets us define partial computable functions in terms of their indices. For example, we can find an index e such that for every y ,

$$\varphi_e(y) = e + y.$$

As another example, one can use the proof of the fixed-point theorem to design a program in Java or C++ that prints itself out.

Remember that if for each e , we let W_e be the domain of φ_e , then the sequence W_0, W_1, W_2, \dots enumerates the computably enumerable sets. Some of these sets are computable. One can ask if there is an algorithm which takes as input a value x , and, if W_x happens to be computable, returns an index for its characteristic function. The answer is “no,” there is no such algorithm:

Theorem 18.25. *There is no partial computable function f with the following property: whenever W_e is computable, then $f(e)$ is defined and $\varphi_{f(e)}$ is its characteristic function.*

Proof. Let f be any computable function; we will construct an e such that W_e is computable, but $\varphi_{f(e)}$ is not its characteristic function. Using the fixed point theorem, we can find an index e such that

$$\varphi_e(y) \simeq \begin{cases} 0 & \text{if } y = 0 \text{ and } \varphi_{f(e)}(0) \downarrow = 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

That is, e is obtained by applying the fixed-point theorem to the function defined by

$$g(x, y) \simeq \begin{cases} 0 & \text{if } y = 0 \text{ and } \varphi_{f(x)}(0) \downarrow = 0 \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Informally, we can see that g is partial computable, as follows: on input x and y , the algorithm first checks to see if y is equal to 0. If it is, the algorithm computes $f(x)$, and then uses the universal machine to compute $\varphi_{f(x)}(0)$. If this last computation halts and returns 0, the algorithm returns 0; otherwise, the algorithm doesn't halt.

But now notice that if $\varphi_{f(e)}(0)$ is defined and equal to 0, then $\varphi_e(y)$ is defined exactly when y is equal to 0, so $W_e = \{0\}$. If $\varphi_{f(e)}(0)$ is not defined, or is defined but not equal to 0, then $W_e = \emptyset$. Either way, $\varphi_{f(e)}$ is not the characteristic function of W_e , since it gives the wrong answer on input 0. \square

18.22 Defining Functions using Self-Reference

It is generally useful to be able to define functions in terms of themselves. For example, given computable functions k , l , and m , the fixed-point lemma tells us that there is a partial computable function f satisfying the following equation for every y :

$$f(y) \simeq \begin{cases} k(y) & \text{if } l(y) = 0 \\ f(m(y)) & \text{otherwise.} \end{cases}$$

Again, more specifically, f is obtained by letting

$$g(x, y) \simeq \begin{cases} k(y) & \text{if } l(y) = 0 \\ \varphi_x(m(y)) & \text{otherwise} \end{cases}$$

and then using the fixed-point lemma to find an index e such that $\varphi_e(y) = g(e, y)$.

For a concrete example, the “greatest common divisor” function $\text{gcd}(u, v)$ can be defined by

$$\text{gcd}(u, v) \simeq \begin{cases} v & \text{if } 0 = 0 \\ \text{gcd}(\text{mod}(v, u), u) & \text{otherwise} \end{cases}$$

where $\text{mod}(v, u)$ denotes the remainder of dividing v by u . An appeal to the fixed-point lemma shows that gcd is partial computable. (In fact, this can be put in the format above, letting y code the pair $\langle u, v \rangle$.) A subsequent induction on u then shows that, in fact, gcd is total.

Of course, one can cook up self-referential definitions that are much fancier than the examples just discussed. Most programming languages support definitions of functions in terms of themselves, one way or another. Note that this is a little bit less dramatic than being able to define a function in terms of an *index* for an algorithm computing the functions, which is what, in full generality, the fixed-point theorem lets you do.

18.23 Minimization with Lambda Terms

When it comes to the lambda calculus, we've shown the following:

1. Every primitive recursive function is represented by a lambda term.
2. There is a lambda term Y such that for any lambda term G , $YG \triangleright G(YG)$.

To show that every partial computable function is represented by some lambda term, we only need to show the following.

Lemma 18.26. *Suppose $f(x, y)$ is primitive recursive. Let g be defined by*

$$g(x) \simeq \mu y \, f(x, y) = 0.$$

Then g is represented by a lambda term.

Proof. The idea is roughly as follows. Given x , we will use the fixed-point lambda term Y to define a function $h_x(n)$ which searches for a y starting at n ; then $g(x)$ is just $h_x(0)$. The function h_x can be expressed as the solution of a fixed-point equation:

$$h_x(n) \simeq \begin{cases} n & \text{if } f(x, n) = 0 \\ h_x(n+1) & \text{otherwise.} \end{cases}$$

Here are the details. Since f is primitive recursive, it is represented by some term F . Remember that we also have a lambda term D such that $D(M, N, \bar{0}) \triangleright M$ and $D(M, N, \bar{1}) \triangleright N$. Fixing x for the moment, to represent h_x we want to find a term H (depending on x) satisfying

$$H(\bar{n}) \equiv D(\bar{n}, H(S(\bar{n})), F(x, \bar{n})).$$

We can do this using the fixed-point term Y . First, let U be the term

$$\lambda h. \lambda z. D(z, (h(Sz)), F(x, z)),$$

and then let H be the term YU . Notice that the only free variable in H is x . Let us show that H satisfies the equation above.

By the definition of Y , we have

$$H = YU \equiv U(YU) = U(H).$$

In particular, for each natural number n , we have

$$\begin{aligned} H(\bar{n}) &\equiv U(H, \bar{n}) \\ &\triangleright D(\bar{n}, H(S(\bar{n})), F(x, \bar{n})), \end{aligned}$$

as required. Notice that if you substitute a numeral \bar{m} for x in the last line, the expression reduces to \bar{n} if $F(\bar{m}, \bar{n})$ reduces to $\bar{0}$, and it reduces to $H(S(\bar{n}))$ if $F(\bar{m}, \bar{n})$ reduces to any other numeral.

To finish off the proof, let G be $\lambda x. H(\bar{0})$. Then G represents g ; in other words, for every m , $G(\bar{m})$ reduces to $\bar{g(m)}$, if $g(m)$ is defined, and has no normal form otherwise. \square

Problems

Problem 18.1. Give a reduction of K to K_0 .

Part V

Turing Machines

The material in this part is a basic and informal introduction to Turing machines. It needs more examples and exercises, and perhaps information on available Turing machine simulators. The proof of the unsolvability of the decision problem uses a successor function, hence all models are infinite. One could strengthen the result by using a successor relation instead. There probably are subtle oversights; use these as checks on students' attention (but also file issues!).

Chapter 19

Turing Machine Computations

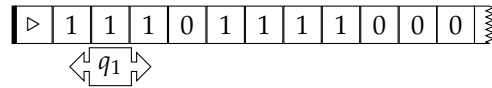
19.1 Introduction

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, but we will mainly make do with three: \triangleright , 0, and 1. When the mechanism is started, the tape is empty (i.e., each square contains the symbol 0) except for the leftmost square, which contains \triangleright , and a finite number of squares which contain the *input*. At any time, the mechanism is in one of a finite number of *states*. At the outset, the head scans the leftmost square and in a specified *initial state*. At each step of the mechanism’s run, the content of the square currently scanned together with the state the mechanism is in and the Turing machine program determine what happens next. The Turing machine program is given by a partial function which takes as input a state q

and a symbol σ and outputs a triple $\langle q', \sigma', D \rangle$. Whenever the mechanism is in state q and reads symbol σ , it replaces the symbol on the current square with σ' , the head moves left, right, or stays put according to whether D is L , R , or N , and the mechanism goes into state q' .

For instance, consider the situation below:



The tape of the Turing machine contains the end-of-tape symbol \triangleright on the leftmost square, followed by three 1's, a 0, four more 1's, and the rest of the tape is filled with 0'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 $\langle q_5, 0, R \rangle$, we would now replace the 1 on the third square with a 0, move right to the fourth square, and change the state of the machine to q_5 .

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.

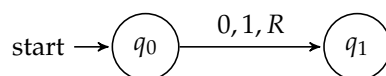
Our goal is to try to define the notion of computability “in principle,” i.e., without taking into account practical limitations of time and space. Of course, with the broadest definition of computability in place, one can then go on to consider computation with bounded resources; this forms the heart of the subject known as “computational complexity.”

19.2. REPRESENTING TURING MACHINES

Historical Remarks Alan Turing invented Turing machines in 1936. While his interest at the time was the decidability of first-order logic, the paper has been described as a definitive paper on the foundations of computer design. In the paper, Turing focuses on computable real numbers, i.e., real numbers whose decimal expansions are computable; but he notes that it is not hard to adapt his notions to computable functions on the natural numbers, and so on. Notice that this was a full five years before the first working general purpose computer was built in 1941 (by the German Konrad Zuse in his parent's living room), seven years before Turing and his colleagues at Bletchley Park built the code-breaking Colossus (1943), nine years before the American ENIAC (1945), twelve years before the first British general purpose computer—the Manchester Small-Scale Experimental Machine—was built in Manchester (1948), and thirteen years before the Americans first tested the BINAC (1949). The Manchester SSEM has the distinction of being the first stored-program computer—previous machines had to be rewired by hand for each new task.

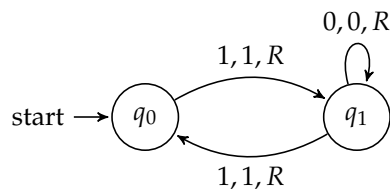
19.2 Representing Turing Machines

Turing machines can be represented visually by *state diagrams*. The diagrams are composed of state cells connected by arrows. Unsurprisingly, each state cell represents a state of the machine. Each arrow represents an instruction that can be carried out from that state, with the specifics of the instruction written above or below the appropriate arrow. Consider the following machine, which has only two internal states, q_0 and q_1 , and one instruction:



Recall that the Turing machine has a read/write head and a tape with the input written on it. The instruction can be read as *if reading a blank in state q_0 , write a stroke, move right, and move to state q_1* . This is equivalent to the transition function mapping $\langle q_0, 0 \rangle$ to $\langle q_1, 1, R \rangle$.

Example 19.1. Even Machine: The following Turing machine halts if, and only if, there are an even number of strokes on the tape.



CHAPTER 19. TURING MACHINE COMPUTATIONS

The state diagram corresponds to the following transition function:

$$\delta(q_0, 1) = \langle q_1, 1, R \rangle,$$

$$\delta(q_1, 1) = \langle q_0, 1, R \rangle,$$

$$\delta(q_1, 0) = \langle q_1, 0, R \rangle$$

The above machine halts only when the input is an even number of strokes. Otherwise, the machine (theoretically) continues to operate indefinitely. For any machine and input, it is possible to trace through the *configurations* of the machine in order to determine the output. We will give a formal definition of configurations later. For now, we can intuitively think of configurations as a series of diagrams showing the state of the machine at any point in time during operation. Configurations show the content of the tape, the state of the machine and the location of the read/write head.

Let us trace through the configurations of the even machine if it is started with an input of 4 1s. In this case, we expect that the machine will halt. We will then run the machine on an input of 3 1s, where the machine will run forever.

The machine starts in state q_0 , scanning the leftmost 1. We can represent the initial state of the machine as follows:

$$\triangleright_0 1110 \dots$$

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 .

$$\triangleright 11_1 110 \dots$$

Since the machine is now in state q_1 scanning a stroke, we have to “follow” the instruction $\delta(q_1, 1) = \langle q_0, 1, R \rangle$. This results in the configuration

$$\triangleright 111_0 10 \dots$$

As the machine continues, the rules are applied again in the same order, resulting in the following two configurations:

$$\triangleright 1111_1 0 \dots$$

$$\triangleright 11110_0 \dots$$

The machine is now in state q_0 scanning a blank. Based on the transition diagram, we can easily see that there is no instruction to be carried out, and thus the machine has halted. This means that the input has been accepted.

19.2. REPRESENTING TURING MACHINES

Suppose next we start the machine with an input of three strokes. The first few configurations are similar, as the same instructions are carried out, with only a small difference of the tape input:

▷1₀110...

▷11₁10...

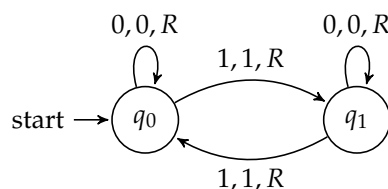
▷111₀0...

▷1110₁...

The machine has now traversed past all the strokes, and is reading a blank in state q_1 . As shown in the diagram, there is an instruction of the form $\delta(q_1, 0) = \langle q_1, 0, R \rangle$. Since the tape is infinitely blank to the right, the machine will continue to execute this instruction *forever*, staying in state q_1 and moving ever further to the right. The machine will never halt, and does not accept the input.

It is important to note that not all machines will halt. If halting means that the machine runs out of instructions to execute, then we can create a machine that never halts simply by ensuring that there is an outgoing arrow for each symbol at each state. The even machine can be modified to run infinitely by adding an instruction for scanning a blank at q_0 .

Example 19.2.



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.

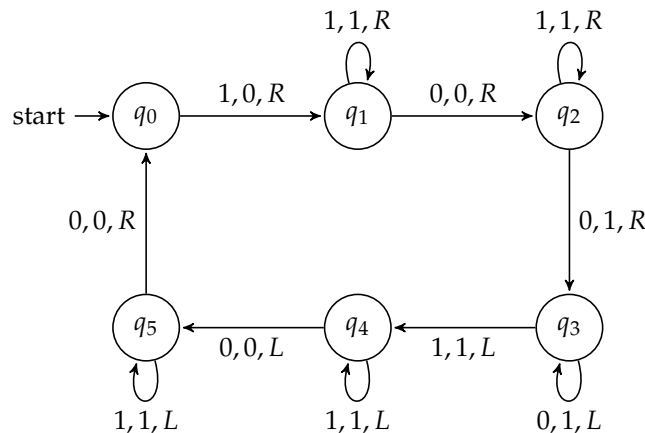
Example 19.3. The machine table for the even machine is:

	0	1
q_0		$1, q_1, R$
q_1	$0, q_1, 0$	$1, q_0, R$

As we can see, the machine halts when scanning a blank in state q_0 .

So far we have only considered machines that read and accept input. However, Turing machines have the capacity to both read and write. An example of such a machine (although there are many, many examples) is a *doubler*. A doubler, when started with a block of n strokes on the tape, outputs a block of $2n$ strokes.

Example 19.4. Before building a doubler machine, it is important to come up with a *strategy* for solving the problem. Since the machine (as we have formulated it) cannot remember how many strokes it has read, we need to come up with a way to keep track of all the strokes on the tape. One such way is to separate the output from the input with a blank. The machine can then erase the first stroke from the input, traverse over the rest of the input, leave a blank, and write two new strokes. The machine will then go back and find the second stroke in the input, and double that one as well. For each one stroke of input, it will write two strokes of output. By erasing the input as the machine goes, we can guarantee that no stroke is missed or doubled twice. When the entire input is erased, there will be $2n$ strokes left on the tape.



19.3 Turing Machines

The formal definition of what constitutes a Turing machine looks abstract, but is actually simple: it merely packs into one mathematical structure all the information needed to specify the workings of a Turing machine. This includes (1) which states the machine can be in, (2) which symbols are allowed to be on the tape, (3) which state the machine should start in, and (4) what the instruction set of the machine is.

Definition 19.5 (Turing machine). A Turing machine $T = \langle Q, \Sigma, q_0, \delta \rangle$ consists of

19.4. CONFIGURATIONS AND COMPUTATIONS

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* $\delta: Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, N\}$.

The partial function δ is also called the *transition function* of T .

We assume that the tape is infinite in one direction only. For this reason it is useful to designate a special symbol \triangleright as a marker for the left end of the tape. This makes it easier for Turing machine programs to tell when they're "in danger" of running off the tape.

Example 19.6. *Even Machine:* The even machine is formally the quadruple $\langle Q, \Sigma, q_0, \delta \rangle$ where

$$\begin{aligned} Q &= \{q_0, q_1\} \\ \Sigma &= \{\triangleright, 0, 1\}, \\ \delta(q_0, 1) &= \langle q_1, 1, R \rangle, \\ \delta(q_1, 1) &= \langle q_0, 1, R \rangle, \\ \delta(q_1, 0) &= \langle q_1, 0, R \rangle. \end{aligned}$$

19.4 Configurations and Computations

Recall tracing through the configurations of the even machine earlier. The imaginary mechanism consisting of tape, read/write head, and Turing machine program is really just in intuitive way of visualizing what a Turing machine computation is. Formally, we can define the computation of a Turing machine on a given input as a sequence of *configurations*—and a configuration in turn is a sequence of symbols (corresponding to the contents of the tape at a given point in the computation), a number indicating the position of the read/write head, and a state. Using these, we can define what the Turing machine M computes on a given input.

Definition 19.7 (Configuration). A *configuration* of Turing machine $M = \langle Q, \Sigma, q_0, \delta \rangle$ is a triple $\langle C, n, q \rangle$ where

1. $C \in \Sigma^*$ is a finite sequence of symbols from Σ ,
2. $n \in \mathbb{N}$ is a number $< \text{len}(C)$, and
3. $q \in Q$

Intuitively, the sequence C is the content of the tape (symbols of all squares from the leftmost square to the last non-blank or previously visited square), n is the number of the square the read/write head is scanning (beginning with 0 being the number of the leftmost square), and q is the current state of the machine.

The potential input for a Turing machine is a sequence of symbols, usually a sequence that encodes a number in some form. The initial configuration of the Turing machine is that configuration in which we start the Turing machine to work on that input: the tape contains the tape end marker immediately followed by the input written on the squares to the right, the read/write head is scanning the leftmost square of the input (i.e., the square to the right of the left end marker), and the mechanism is in the designated start state q_0 .

Definition 19.8 (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 19.9. We say that a configuration $\langle C, n, q \rangle$ *yields* $\langle C', n', q' \rangle$ in one step (according to M), iff

1. the n -th symbol of C is σ ,
2. the instruction set of M specifies $\delta(q, \sigma) = \langle q', \sigma', D \rangle$,
3. the n -th symbol of C' is σ' , and
4.
 - a) $D = L$ and $n' = n - 1$ if $n > 0$, otherwise $n' = 0$, or
 - b) $D = R$ and $n' = n + 1$, or
 - c) $D = N$ and $n' = n$,
5. if $n' > \text{len}(C)$, then $\text{len}(C') = \text{len}(C) + 1$ and the n' -th symbol of C' is 0.
6. for all i such that $i < \text{len}(C')$ and $i \neq n$, $C'(i) = C(i)$,

Definition 19.10. A *run* of M on input I is a sequence C_i of configurations of M , where C_0 is the initial configuration of M for input I , and each C_i yields C_{i+1} in one step.

We say that M *halts on input I after k steps* if $C_k = \langle C, n, q \rangle$, the n th 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.

19.5 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.

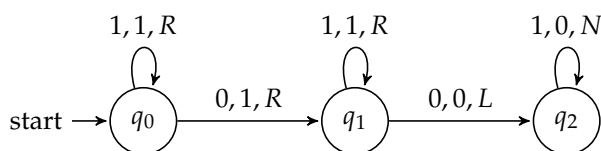
Definition 19.11 (Computation). A Turing machine M *computes* the function $f: \mathbb{N}^n \rightarrow \mathbb{N}$ iff M halts on input

$$1^{k_1}01^{k_2}0 \dots 01^{k_n}$$

with output $1^{f(k_1, \dots, k_n)}$.

Example 19.12. *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.

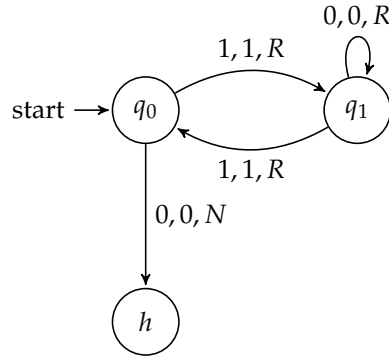


19.6 Halting States

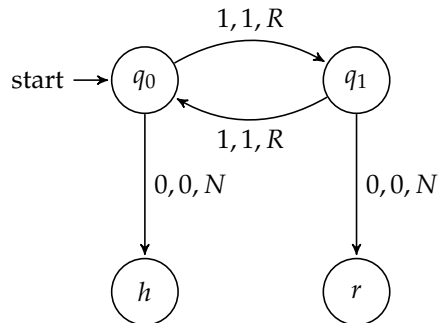
Although we have defined our machines to halt only when there is no instruction to carry out, common representations of Turing machines have a dedicated *halting state*, h , such that $h \in Q$.

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 19.13. *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.

19.7 Combining Turing Machines

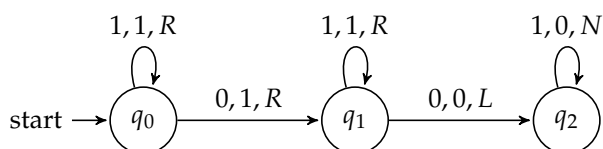
The examples of Turing machines we have seen so far have been fairly simple in nature. But in fact, any problem that can be solved with any modern programming language can also be solved with Turing machines. To build more complex Turing machines, it is important to convince ourselves that we

19.7. COMBINING TURING MACHINES

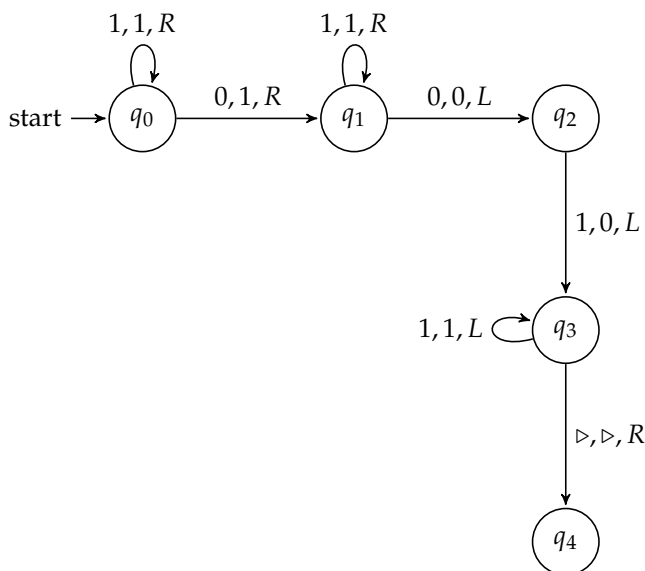
can combine them, so we can build machines to solve more complex problems by breaking the procedure into simpler parts. If we can find a natural way to break a complex problem down into constituent parts, we can tackle the problem in several stages, creating several simple Turing machines and combining them into one machine that can solve the problem. This point is especially important when tackling the Halting Problem in the next section.

Example 19.14. *Combining Machines:* Design a machine that computes the function $f(m, n) = 2(m + n)$.

In order to build this machine, we can combine two machines we are already familiar with: the addition machine, and the doubler. We begin by drawing a state diagram for the addition machine.

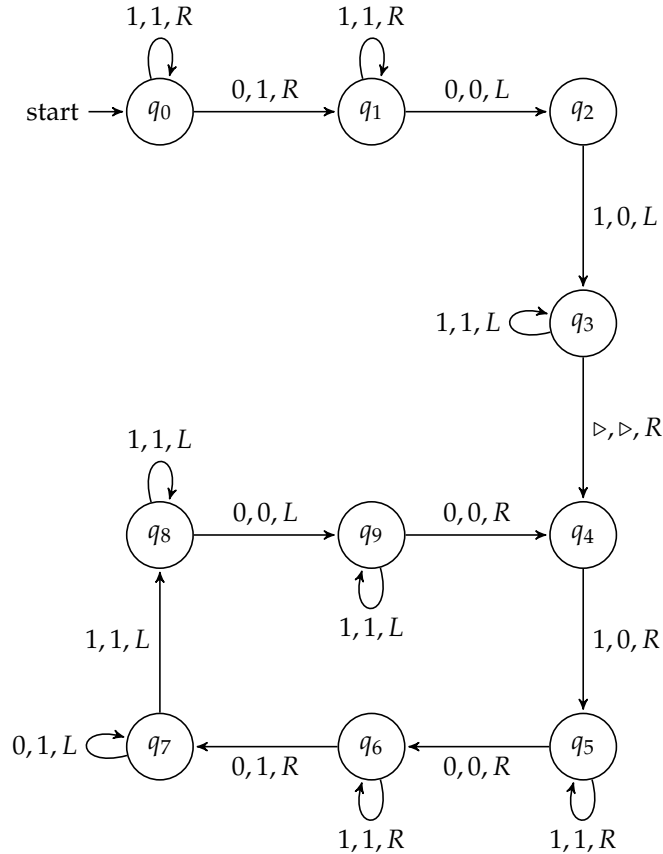


Instead of halting at state q_2 , we want to continue operation in order to double the output. Recall that the doubler machine erases the first stroke in the input and writes two strokes in a separate output. Let's add an instruction to make sure the tape head is reading the first stroke of the output of the addition machine.



It is now easy to double the input—all we have to do is connect the doubler machine onto state q_4 . This requires renaming the states of the doubler machine so that they start at q_4 instead of q_0 —this way we don't end up with two

starting states. The final diagram should look like:



19.8 Variants of Turing Machines

There are in fact many possible ways to define Turing machines, of which ours is only one. In some ways, our definition is more liberal than others. We allow arbitrary finite alphabets, a more restricted definition might allow only two tape symbols, 1 and 0. We allow the machine to write a symbol to the tape and move at the same time, other definitions allow either writing or moving. We allow the possibility of writing without moving the tape head, other definitions leave out the N "instruction." In other ways, our definition is more restrictive. We assumed that the tape is infinite in one direction only, other definitions allow the tape to be infinite both to the left and the right. In fact, one can even even allow any number of separate tapes, or even an infinite grid of squares. We represent the instruction set of the Turing machine by a transition function; other definitions use a transition relation where the machine has more than one possible instruction in any given situation.

19.9. THE CHURCH-TURING THESIS

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).

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.*

19.9 The Church-Turing Thesis

Turing machines are supposed to be a precise replacement for the concept of an effective procedure. Turing took it that anyone who grasped the concept of an effective procedure and the concept of a Turing machine would have the intuition that anything that could be done via an effective procedure could be done by Turing machine. This claim is given support by the fact that all the other proposed precise replacements for the concept of an effective procedure turn out to be extensionally equivalent to the concept of a Turing machine—that is, they can compute exactly the same set of functions. This claim is called the *Church-Turing thesis*.

Definition 19.15 (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 which cannot be computed by a Turing machine. 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 19.1. Choose an arbitrary input and trace through the configurations of the doubler machine in [Example 19.4](#).

Problem 19.2. The double machine in [Example 19.4](#) 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 19.3. Design a Turing-machine with alphabet $\{0, A, B\}$ that accepts any string of *As* and *Bs* where the number of *As* is the same as the number of *Bs* and all the *As* precede all the *Bs*, and rejects any string where the number of *As* is not equal to the number of *Bs* or the *As* do not precede all the *Bs*. (E.g., the machine should accept *AABB*, and *AAABBB*, but reject both *AAB* and *AABBAABB*.)

Problem 19.4. Design a Turing-machine with alphabet $\{0, A, B\}$ that takes as input any string α of *As* and *Bs* and duplicates them to produce an output of the form $\alpha\alpha$. (E.g. input *ABBA* should result in output *ABBAABBA*).

Problem 19.5. *Alphabetical?*: Design a Turing-machine with alphabet $\{0, A, B\}$ that when given as input a finite sequence of *As* and *Bs* checks to see if all the *As* appear left of all the *Bs* or not. The machine should leave the input string on the tape, and output either halt if the string is “alphabetical”, or loop forever if the string is not.

Problem 19.6. *Alphabetizer*: Design a Turing-machine with alphabet $\{0, A, B\}$ that takes as input a finite sequence of *As* and *Bs* rearranges them so that all the *As* are to the left of all the *Bs*. (e.g., the sequence *BABAA* should become the sequence *AAABB*, and the sequence *ABBABB* should become the sequence *AABBBB*).

19.9. THE CHURCH-TURING THESIS

Problem 19.7. Trace through the configurations of the machine for input $\langle 3, 5 \rangle$.

Problem 19.8. *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 19.9. *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 19.10. 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 20

Undecidability

20.1 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

20.1. INTRODUCTION

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 non-enumerable, but since we can enumerate all the Turing machines, the set of Turing-computable functions is only denumerable.

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.

20.2 Enumerating Turing Machines

We can show that the set of all Turing-machines is enumerable. 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 denumerable vocabulary for specifying Turing machines: $\sigma_0 = \triangleright, \sigma_1 = 0, \sigma_2 = 1, \sigma_3, \dots, R, L, N, q_0, q_1, \dots$. 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

$$Q = \{q'_0, \dots, q'_n\} \subseteq \{q_0, q_1, \dots\} \text{ and} \\ \Sigma = \{\triangleright, \sigma'_1, \sigma'_2, \dots, \sigma'_m\} \subseteq \{\sigma_0, \sigma_1, \dots\}.$$

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 20.1. *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 denumerable alphabet is enumerable. This gives us that the set of descriptions of Turing machines, as a subset of the finite strings from the enumerable 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 enumerable. 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 enumerable. 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. \square

20.3 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

20.3. THE HALTING PROBLEM

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 20.2 (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 20.3 (Halting problem). The *Halting Problem* is the problem of determining (for any m, w) whether the Turing machine M_e halts for an input of n strokes.

We show that h is not Turing-computable by showing that a related function, s , is not Turing-computable. This proof relies on the fact that anything that can be computed by a Turing machine can be computed using just two symbols: 0 and 1, and the fact that two Turing machines can be hooked together to create a single machine.

Definition 20.4. 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 20.5. *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. \square

Theorem 20.6 (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. \square

20.4 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

20.5. REPRESENTING TURING MACHINES

instruction set of M and the input w and a sentence $\alpha(M, w)$ expressing “ M eventually halts” such that:

$$\models \tau(M, w) \rightarrow \alpha(M, w) \text{ iff } M \text{ halts for input } w.$$

The bulk of our proof will consist in describing these sentences $\tau(M, w)$ and $\alpha(M, w)$ and verifying that $\tau(M, w) \rightarrow \alpha(M, w)$ is valid iff M halts on input w .

20.5 Representing Turing Machines

In order to represent Turing machines and their behavior by a sentence of first-order logic, we have to define a suitable language. The language consists of two parts: predicate symbols for describing configurations of the machine, and expressions for numbering execution steps (“moments”) and positions on the tape.

We introduce two kinds of predicate symbols, both of them 2-place: For each state q , a predicate symbol Q_q , and for each tape symbol σ , a predicate symbol S_σ . The former allow us to describe the state of M and the position of its tape head, the latter allow us to describe the contents of the tape.

In order to express the positions of the tape head and the number of steps executed, we need a way to express numbers. This is done using a constant symbol o , and a 1-place function ι , the successor function. By convention it is written *after* its argument (and we leave out the parentheses). So o names the leftmost position on the tape as well as the time before the first execution step (the initial configuration), o' names the square to the right of the leftmost square, and the time after the first execution step, and so on. We also introduce a predicate symbol $<$ to express both the ordering of tape positions (when it means “to the left of”) and execution steps (then it means “before”).

Once we have the language in place, we list the “axioms” of $\tau(M, w)$, i.e., the sentences which, taken together, describe the behavior of M when run on input w . There will be sentences which lay down conditions on o , ι , and $<$, sentences that describes the input configuration, and sentences that describe what the configuration of M is after it executes a particular instruction.

Definition 20.7. 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, $Q_q(\bar{m}, \bar{n})$ expresses “after n steps, M is in state q scanning the n th square.”
2. A two-place predicate symbol $S_\sigma(x, y)$ for every symbol $\sigma \in \Sigma$. Intuitively, $S_\sigma(\bar{m}, \bar{n})$ expresses “after n steps, the m th square contains symbol σ .”
3. A constant symbol o

4. A one-place function symbol $'$
5. A two-place predicate symbol $<$

For each number n there is a canonical term \bar{n} , the *numeral* for n , which represents it in \mathcal{L}_M . $\bar{0}$ is o , $\bar{1}$ is o' , $\bar{2}$ is o'' , and so on. More formally:

$$\begin{aligned}\bar{0} &= o \\ \overline{n+1} &= \bar{n}'\end{aligned}$$

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:

$$\forall x \forall y (x' = y' \rightarrow x = y)$$

- 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:

$$\forall x \forall y \forall z ((x < y \wedge y < z) \rightarrow x < z)$$

- d) A sentence that connects $<$ and $=$:

$$\forall x \forall y (x < y \rightarrow x \neq y)$$

2. Axioms describing the input configuration:

- a) After after 0 steps—before the machine starts— M is in the initial state q_0 , scanning square 1:

$$Q_{q_0}(\bar{1}, \bar{0})$$

- b) The first $k+1$ squares contain the symbols $\triangleright, \sigma_{i_1}, \dots, \sigma_{i_k}$:

$$S_{\triangleright}(\bar{0}, \bar{0}) \wedge S_{\sigma_{i_1}}(\bar{1}, \bar{0}) \wedge \dots \wedge S_{\sigma_{i_k}}(\bar{n}, \bar{0})$$

- c) Otherwise, the tape is empty:

$$\forall x (\bar{k} < x \rightarrow S_0(x, \bar{0}))$$

20.5. REPRESENTING TURING MACHINES

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

$$\forall z ((z < x \vee x < z) \wedge S_\sigma(z, y)) \rightarrow S_\sigma(z, y')$$

where $\sigma \in \Sigma$. We use $\varphi(\bar{m}, \bar{n})$ 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:

$$\begin{aligned} \forall x \forall y ((Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \rightarrow \\ (Q_{q_j}(x', y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y))) \end{aligned}$$

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:

$$\begin{aligned} \forall x \forall y ((Q_{q_i}(x', y) \wedge S_\sigma(x', y)) \rightarrow \\ (Q_{q_j}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\ \forall y ((Q_{q_i}(0, y) \wedge S_\sigma(0, y)) \rightarrow \\ (Q_{q_j}(0, y') \wedge S_{\sigma'}(0, y') \wedge \varphi(0, y))) \end{aligned}$$

Take a moment to think about how this works: now we don't start with “if scanning square $x \dots$ ” but: “if scanning square $x + 1 \dots$ ” 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 $1, 2, \dots$, 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 after y steps.

c) For every instruction $\delta(q_i, \sigma) = \langle q_j, \sigma', N \rangle$, the sentence:

$$\begin{aligned} \forall x \forall y ((Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \rightarrow \\ (Q_{q_j}(x, y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y))) \end{aligned}$$

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

$$\exists x \exists y \left(\bigvee_{\langle q, \sigma \rangle \in X} (Q_q(x, y) \wedge S_\sigma(x, y)) \right)$$

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 20.8. *If $m < k$, then $\tau(M, w) \models \bar{m} < \bar{k}$*

Proof. Exercise. □

20.6 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, $\tau(M, w) \rightarrow \alpha(M, w)$) 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 $\tau(M, w) \rightarrow \alpha(M, w)$.

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 , $\tau(M, w) \models \chi(M, w, n)$, 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 $\chi(M, w, n) \models \alpha(M, w)$, 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, $\tau(M, w) \models \chi(M, w, n)$ where $\chi(M, w, n)$ describes a halting configuration, and since in that case $\chi(M, w, n) \models \alpha(M, w)$, we get that $T(M, w) \models \alpha(M, w)$, i.e., that $\models \tau(M, w) \rightarrow \alpha(M, w)$.

20.6. VERIFYING THE REPRESENTATION

The strategy for the converse is very different. Here we assume that $\models \tau(M, w) \rightarrow \alpha(M, w)$ and have to prove that M halts on input w . From the hypothesis we get that $\tau(M, w) \models \alpha(M, w)$, i.e., $\alpha(M, w)$ is true in every structure in which $\tau(M, w)$ is true. So we'll describe a structure \mathfrak{M} in which $\tau(M, w)$ is true: its domain will be \mathbb{N} , and the interpretation of all the Q_q and S_σ will be given by the configurations of M during a run on input w . So, e.g., $\mathfrak{M} \models Q_q(\bar{m}, \bar{n})$ iff T , when run on input w for n steps, is in state q and scanning square m . Now since $\tau(M, w) \models \alpha(M, w)$ by hypothesis, and since $\mathfrak{M} \models \tau(M, w)$ by construction, $\mathfrak{M} \models \alpha(M, w)$. But $\mathfrak{M} \models \alpha(M, w)$ iff there is some $n \in |\mathfrak{M}| = \mathbb{N}$ so that M , run on input w , is in a halting configuration after n steps.

Definition 20.9. Let $\chi(M, w, n)$ be the sentence

$$Q_q(\bar{m}, \bar{n}) \wedge S_{\sigma_0}(\bar{0}, \bar{n}) \wedge \cdots \wedge S_{\sigma_k}(\bar{k}, \bar{n}) \wedge \forall x (\bar{k} < x \rightarrow S_0(x, \bar{n}))$$

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 20.10. If M run on input w is in a halting configuration after n steps, then $\chi(M, w, n) \models \alpha(M, w)$.

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 $Q_q(\bar{m}, \bar{n})$ and $S_\sigma(\bar{m}, \bar{n})$. These clauses together imply $\alpha(M, w)$:

$$\exists x \exists y \left(\bigvee_{\langle q, \sigma \rangle \in X} (Q_q(x, y) \wedge S_\sigma(x, y)) \right)$$

since $Q_{q'}(\bar{m}, \bar{n}) \wedge S_{\sigma'}(\bar{m}, \bar{n}) \models \bigvee_{\langle q, \sigma \rangle \in X} (Q_q(\bar{m}, \bar{n}) \wedge S_\sigma(\bar{m}, \bar{n}))$, as $\langle q', \sigma' \rangle \in X$. \square

So if M halts for input w , then there is some n such that $\chi(M, w, n) \models \alpha(M, w)$. We will now show that for any time n , $\tau(M, w) \models \chi(M, w, n)$.

Lemma 20.11. For each n , if M has not halted after n steps, $\tau(M, w) \models \chi(M, w, n)$.

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 n th step, then $\tau(M, w) \models \chi(M, w, n)$. We have to show that (unless $\chi(M, w, n)$ describes a halting configuration), $\tau(M, w) \models \chi(M, w, n + 1)$.

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 20.7, (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, \bar{m} for x and \bar{n} for y):

$$(Q_q(\bar{m}, \bar{n}) \wedge S_\sigma(\bar{m}, \bar{n})) \rightarrow (Q_{q'}(\bar{m}', \bar{n}') \wedge S_{\sigma'}(\bar{m}, \bar{n}') \wedge \varphi(\bar{m}, \bar{n})).$$

By induction hypothesis, $\tau(M, w) \models \chi(M, w, n)$, i.e.,

$$Q_q(\bar{m}, \bar{n}) \wedge S_{\sigma_0}(\bar{0}, \bar{n}) \wedge \cdots \wedge S_{\sigma_k}(\bar{k}, \bar{n}) \wedge \forall x (\bar{k} < x \rightarrow S_0(x, \bar{n}))$$

Since after n steps, tape square m contains σ , the corresponding conjunct is $S_\sigma(\bar{m}, \bar{n})$, so this entails:

$$Q_q(\bar{m}, \bar{n}) \wedge S_\sigma(\bar{m}, \bar{n})$$

We now get

$$\begin{aligned} & Q_{q'}(\bar{m}', \bar{n}') \wedge S_{\sigma'}(\bar{m}, \bar{n}') \wedge \\ & S_{\sigma_0}(\bar{0}, \bar{n}') \wedge \cdots \wedge S_{\sigma_k}(\bar{k}, \bar{n}') \wedge \\ & \forall x (\bar{k} < x \rightarrow S_0(x, \bar{n}')) \end{aligned}$$

as follows: The first line comes directly from the consequent of the preceding conditional, by modus ponens. Each conjunct in the middle

20.6. VERIFYING THE REPRESENTATION

line—which excludes $S_{\sigma_m}(\bar{m}, \bar{n}')$ —follows from the corresponding conjunct in $\chi(M, w, n)$ together with $\varphi(\bar{m}, \bar{n})$.

If $m < k$, $\tau(M, w) \vdash \bar{m} < \bar{k}$ (Proposition 20.8) and by transitivity of $<$, we have $\forall x (\bar{k} < x \rightarrow \bar{m} < x)$. If $m = k$, then $\forall x (\bar{k} < x \rightarrow \bar{m} < x)$ by logic alone. The last line then follows from the corresponding conjunct in $\chi(M, w, n)$, $\forall x (\bar{k} < x \rightarrow \bar{m} < x)$, and $\varphi(\bar{m}, \bar{n})$. 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, $S_0(\bar{k}', \bar{n}')$, and the last conjunct is $\forall x (\bar{k}' < x \rightarrow S_0(x, \bar{n}'))$. We have to verify that these two sentences are also implied.

We already have $\forall x (\bar{k} < x \rightarrow S_0(x, \bar{n}'))$. In particular, this gives us $\bar{k} < \bar{k}' \rightarrow S_0(\bar{k}', \bar{n}')$. From the axiom $\forall x x < x'$ we get $\bar{k} < \bar{k}'$. By modus ponens, $S_0(\bar{k}', \bar{n}')$ follows.

Also, since $\tau(M, w) \vdash \bar{k} < \bar{k}'$, the axiom for transitivity of $<$ gives us $\forall x (\bar{k}' < x \rightarrow S_0(x, \bar{n}'))$. (We leave the verification of this as an exercise.)

2. Suppose there is an instruction of the form (2). Then, by Definition 20.7, (3b),

$$\begin{aligned} & \forall x \forall y ((Q_q(x', y) \wedge S_{\sigma}(x', y)) \rightarrow \\ & \quad (Q_{q'}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\ & \forall y ((Q_{q_i}(0, y) \wedge S_{\sigma}(0, y)) \rightarrow \\ & \quad (Q_{q_j}(0, y') \wedge S_{\sigma'}(0, y') \wedge \varphi(0, y))) \end{aligned}$$

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:

$$\begin{aligned} & (Q_q(\bar{l}', \bar{n}) \wedge S_{\sigma}(\bar{l}', \bar{n})) \rightarrow \\ & \quad (Q_{q'}(\bar{l}, \bar{n}') \wedge S_{\sigma'}(\bar{l}', \bar{n}') \wedge \varphi(\bar{l}, \bar{n})) \end{aligned}$$

Otherwise, let $l = m = 0$ and consider the following sentence entailed by the second conjunct:

$$\begin{aligned} & ((Q_{q_i}(0, \bar{n}) \wedge S_{\sigma}(0, \bar{n})) \rightarrow \\ & \quad (Q_{q_j}(0, \bar{n}') \wedge S_{\sigma'}(0, \bar{n}') \wedge \varphi(0, \bar{n}))) \end{aligned}$$

Either sentence implies

$$\begin{aligned} & Q_{q'}(\bar{l}, \bar{n}') \wedge S_{\sigma'}(\bar{m}, \bar{n}') \wedge \\ & S_{\sigma_0}(\bar{0}, \bar{n}') \wedge \cdots \wedge S_{\sigma_k}(\bar{k}, \bar{n}') \wedge \\ & \forall x (\bar{k} < x \rightarrow S_0(x, \bar{n}')) \end{aligned}$$

as before. (Note that in the first case, $\bar{l}' = \bar{m}$ and in the second case $\bar{l} = 0$.) But this just is $\chi(M, w, n+1)$.

3. Case (3) is left as an exercise.

We have shown that for any n , $\tau(M, w) \models \chi(M, w, n)$. □

Lemma 20.12. *If M halts on input w , then $\tau(M, w) \rightarrow \alpha(M, w)$ is valid.*

Proof. By Lemma 20.11, 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 $Q_q(\bar{m}, \bar{k})$ and $S_\sigma(\bar{m}, \bar{k})$ with $\delta(q, \sigma)$ undefined. By Lemma 20.10 Thus, $\chi(M, w, k) \models \alpha(M, w)$. But since $(M, w) \models \chi(M, w, k)$, we have $\tau(M, w) \models \alpha(M, w)$ and therefore $\tau(M, w) \rightarrow \alpha(M, w)$ is valid. □

To complete the verification of our claim, we also have to establish the reverse direction: if $\tau(M, w) \rightarrow \alpha(M, w)$ is valid, then M does in fact halt when started on input m .

Lemma 20.13. *If $\tau(M, w) \rightarrow \alpha(M, w)$, 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:

$$\begin{aligned} Q_q^{\mathfrak{M}} &= \{ \langle m, n \rangle : \begin{array}{l} \text{started on } w, \text{ after } n \text{ steps,} \\ M \text{ is in state } q \text{ scanning square } m \end{array} \} \\ S_\sigma^{\mathfrak{M}} &= \{ \langle m, n \rangle : \begin{array}{l} \text{started on } w, \text{ after } n \text{ steps,} \\ \text{square } m \text{ of } M \text{ contains symbol } \sigma \end{array} \} \end{aligned}$$

In other words, we construct the structure \mathfrak{M} so that it describes what M started on input w actually does, step by step. Clearly, $\mathfrak{M} \models \tau(M, w)$. If $\tau(M, w) \rightarrow \alpha(M, w)$, then also $\mathfrak{M} \models \alpha(M, w)$, i.e.,

$$\mathfrak{M} \models \exists x \exists y \left(\bigvee_{\langle q, \sigma \rangle \in X} (Q_q(x, y) \wedge S_\sigma(x, y)) \right).$$

As $|\mathfrak{M}| = \mathbb{N}$, there must be $m, n \in \mathbb{N}$ so that $\mathfrak{M} \models Q_q(\bar{m}, \bar{n}) \wedge S_\sigma(\bar{m}, \bar{n})$ 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. □

20.7 The Decision Problem is Unsolvable

Theorem 20.14. *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 $\tau(M_e, w) \rightarrow \alpha(M_e, w)$ and halts, scanning the leftmost square on the tape. The machine $E \frown D$ would then, given input e and w , first compute $\tau(M_e, w) \rightarrow \alpha(M_e, w)$ and then run the decision problem machine D on that input. D halts with output 1 iff $\tau(M_e, w) \rightarrow \alpha(M_e, w)$ is valid and outputs 0 otherwise. By [Lemma 20.13](#) and [Lemma 20.12](#), $\tau(M_e, w) \rightarrow \alpha(M_e, w)$ 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 20.6](#), that no such Turing machine can exist. \square

Problems

Problem 20.1. 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 20.2. 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 20.3. We proved that the halting problem is unsolvable if the input is a number e , which identifies a Turing machine M_e via an enumeration of all Turing machines. What if we allow the description of Turing machines from [section 20.2](#) 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 20.4. Prove [Proposition 20.8](#). (Hint: use induction on $k - m$).

Problem 20.5. Complete case (3) of the proof of [Lemma 20.11](#).

Problem 20.6. Give a derivation of $S_{\sigma_i}(\bar{i}, \bar{n}')$ from $S_{\sigma_i}(\bar{i}, \bar{n})$ and $\varphi(m, n)$ (assuming $i \neq m$, i.e., either $i < m$ or $m < i$).

Problem 20.7. Give a derivation of $\forall x (\bar{k}' < x \rightarrow S_0(x, \bar{n}'))$ from $\forall x (\bar{k} < x \rightarrow S_0(x, \bar{n}))$, $\forall x x < x'$, and $\forall x \forall y \forall z ((x < y \wedge y < z) \rightarrow x < z)$.

Part VI

Incompleteness

Material in this part covers the incompleteness theorems. It depends on material in the parts on first-order logic (esp., the proof system), the material on recursive functions (in the computability part). It is based on Jeremy Avigad's notes with revisions by Richard Zach.

Chapter 21

Introduction to Incompleteness

21.1 Historical Background

In this section, we will briefly discuss historical developments that will help put the incompleteness theorems in context. In particular, we will give a very sketchy overview of the history of mathematical logic; and then say a few words about the history of the foundations of mathematics.

The phrase “mathematical logic” is ambiguous. One can interpret the word “mathematical” as describing the subject matter, as in, “the logic of mathematics,” denoting the principles of mathematical reasoning; or as describing the methods, as in “the mathematics of logic,” denoting a mathematical study of the principles of reasoning. The account that follows involves mathematical logic in both senses, often at the same time.

The study of logic began, essentially, with Aristotle, who lived approximately 384–322 BCE. His *Categories*, *Prior analytics*, and *Posterior analytics* include systematic studies of the principles of scientific reasoning, including a thorough and systematic study of the syllogism.

Aristotle’s logic dominated scholastic philosophy through the middle ages; indeed, as late as eighteenth century Kant maintained that Aristotle’s logic was perfect and in no need of revision. But the theory of the syllogism is far too limited to model anything but the most superficial aspects of mathematical reasoning. A century earlier, Leibniz, a contemporary of Newton’s, imagined a complete “calculus” for logical reasoning, and made some rudimentary steps towards designing such a calculus, essentially describing a version of propositional logic.

The nineteenth century was a watershed for logic. In 1854 George Boole wrote *The Laws of Thought*, with a thorough algebraic study of propositional logic that is not far from modern presentations. In 1879 Gottlob Frege published his *Begriffsschrift* (Concept writing) which extends propositional logic with quantifiers and relations, and thus includes first-order logic. In fact, Frege’s logical systems included higher-order logic as well, and more. In his

CHAPTER 21. INTRODUCTION TO INCOMPLETENESS

Basic Laws of Arithmetic, Frege set out to show that all of arithmetic could be derived in his Begriffsschrift from purely logical assumption. Unfortunately, these assumptions turned out to be inconsistent, as Russell showed in 1902. But setting aside the inconsistent axiom, Frege more or less invented modern logic singlehandedly, a startling achievement. Quantificational logic was also developed independently by algebraically-minded thinkers after Boole, including Peirce and Schröder.

Let us now turn to developments in the foundations of mathematics. Of course, since logic plays an important role in mathematics, there is a good deal of interaction with the developments I just described. For example, Frege developed his logic with the explicit purpose of showing that all of mathematics could be based solely on his logical framework; in particular, he wished to show that mathematics consists of a priori *analytic* truths instead of, as Kant had maintained, a priori *synthetic* ones.

Many take the birth of mathematics proper to have occurred with the Greeks. Euclid's *Elements*, written around 300 B.C., is already a mature representative of Greek mathematics, with its emphasis on rigor and precision. The definitions and proofs in Euclid's *Elements* survive more or less in tact in high school geometry textbooks today (to the extent that geometry is still taught in high schools). This model of mathematical reasoning has been held to be a paradigm for rigorous argumentation not only in mathematics but in branches of philosophy as well. (Spinoza even presented moral and religious arguments in the Euclidean style, which is strange to see!)

Calculus was invented by Newton and Leibniz in the seventeenth century. (A fierce priority dispute raged for centuries, but most scholars today hold that the two developments were for the most part independent.) Calculus involves reasoning about, for example, infinite sums of infinitely small quantities; these features fueled criticism by Bishop Berkeley, who argued that belief in God was no less rational than the mathematics of his time. The methods of calculus were widely used in the eighteenth century, for example by Leonhard Euler, who used calculations involving infinite sums with dramatic results.

In the nineteenth century, mathematicians tried to address Berkeley's criticisms by putting calculus on a firmer foundation. Efforts by Cauchy, Weierstrass, Bolzano, and others led to our contemporary definitions of limits, continuity, differentiation, and integration in terms of "epsilon and deltas," in other words, devoid of any reference to infinitesimals. Later in the century, mathematicians tried to push further, and explain all aspects of calculus, including the real numbers themselves, in terms of the natural numbers. (Kronecker: "God created the whole numbers, all else is the work of man.") In 1872, Dedekind wrote "Continuity and the irrational numbers," where he showed how to "construct" the real numbers as sets of rational numbers (which, as you know, can be viewed as pairs of natural numbers); in 1888 he wrote "Was sind und was sollen die Zahlen" (roughly, "What are the natural num-

21.1. HISTORICAL BACKGROUND

bers, and what should they be?") which aimed to explain the natural numbers in purely "logical" terms. In 1887 Kronecker wrote "Über den Zahlbegriff" ("On the concept of number") where he spoke of representing all mathematical object in terms of the integers; in 1889 Giuseppe Peano gave formal, symbolic axioms for the natural numbers.

The end of the nineteenth century also brought a new boldness in dealing with the infinite. Before then, infinitary objects and structures (like the set of natural numbers) were treated gingerly; "infinitely many" was understood as "as many as you want," and "approaches in the limit" was understood as "gets as close as you want." But Georg Cantor showed that it was possible to take the infinite at face value. Work by Cantor, Dedekind, and others help to introduce the general set-theoretic understanding of mathematics that is now widely accepted.

This brings us to twentieth century developments in logic and foundations. In 1902 Russell discovered the paradox in Frege's logical system. In 1904 Zermelo proved Cantor's well-ordering principle, using the so-called "axiom of choice"; the legitimacy of this axiom prompted a good deal of debate. Between 1910 and 1913 the three volumes of Russell and Whitehead's *Principia Mathematica* appeared, extending the Fregean program of establishing mathematics on logical grounds. Unfortunately, Russell and Whitehead were forced to adopt two principles that seemed hard to justify as purely logical: an axiom of infinity and an axiom of "reducibility." In the 1900's Poincaré criticized the use of "impredicative definitions" in mathematics, and in the 1910's Brouwer began proposing to refound all of mathematics in an "intuitionistic" basis, which avoided the use of the law of the excluded middle ($\varphi \vee \neg\varphi$).

Strange days indeed! The program of reducing all of mathematics to logic is now referred to as "logicism," and is commonly viewed as having failed, due to the difficulties mentioned above. The program of developing mathematics in terms of intuitionistic mental constructions is called "intuitionism," and is viewed as posing overly severe restrictions on everyday mathematics. Around the turn of the century, David Hilbert, one of the most influential mathematicians of all time, was a strong supporter of the new, abstract methods introduced by Cantor and Dedekind: "no one will drive us from the paradise that Cantor has created for us." At the same time, he was sensitive to foundational criticisms of these new methods (oddly enough, now called "classical"). He proposed a way of having one's cake and eating it too:

1. Represent classical methods with formal axioms and rules; represent mathematical questions as formulas in an axiomatic system.
2. Use safe, "finitary" methods to prove that these formal deductive systems are consistent.

Hilbert's work went a long way toward accomplishing the first goal. In 1899, he had done this for geometry in his celebrated book *Foundations of ge-*

ometry. In subsequent years, he and a number of his students and collaborators worked on other areas of mathematics to do what Hilbert had done for geometry. Hilbert himself gave axiom systems for arithmetic and analysis. Zermelo gave an axiomatization of set theory, which was expanded on by Fraenkel, Skolem, von Neumann, and others. By the mid-1920s, there were two approaches that laid claim to the title of an axiomatization of “all” of mathematics, the *Principia mathematica* of Russell and Whitehead, and what came to be known as Zermelo-Fraenkel set theory.

In 1921, Hilbert set out on a research project to establish the goal of proving these systems to be consistent. He was aided in this project by several of his students, in particular Bernays, Ackermann, and later Gentzen. The basic idea for accomplishing this goal was to cast the question of the possibility of a derivation of an inconsistency in mathematics as a combinatorial problem about possible sequences of symbols, namely possible sequences of sentences which meet the criterion of being a correct derivation of, say, $\varphi \wedge \neg\varphi$ from the axioms of an axiom system for arithmetic, analysis, or set theory. A proof of the impossibility of such a sequence of symbols would—since it is itself a mathematical proof—be formalizable in these axiomatic systems. In other words, there would be some sentence Con which states that, say, arithmetic is consistent. Moreover, this sentence should be provable in the systems in question, especially if its proof requires only very restricted, “finitary” means.

The second aim, that the axiom systems developed would settle every mathematical question, can be made precise in two ways. In one way, we can formulate it as follows: For any sentence φ in the language of an axiom system for mathematics, either φ or $\neg\varphi$ is provable from the axioms. If this were true, then there would be no sentences which can neither be proved nor refuted on the basis of the axioms, no questions which the axioms do not settle. An axiom system with this property is called *complete*. Of course, for any given sentence it might still be a difficult task to determine which of the two alternatives holds. But in principle there should be a method to do so. In fact, for the axiom and derivation systems considered by Hilbert, completeness would imply that such a method exists—although Hilbert did not realize this. The second way to interpret the question would be this stronger requirement: that there be a mechanical, computational method which would determine, for a given sentence φ , whether it is derivable from the axioms or not.

In 1931, Gödel proved the two “incompleteness theorems,” which showed that this program could not succeed. There is no axiom system for mathematics which is complete, specifically, the sentence that expresses the consistency of the axioms is a sentence which can neither be proved nor refuted.

This struck a lethal blow to Hilbert’s original program. However, as is so often the case in mathematics, it also opened up exciting new avenues for research. If there is no one, all-encompassing formal system of mathematics, it makes sense to develop more circumscribed systems and investigate what

21.2. DEFINITIONS

can be proved in them. It also makes sense to develop less restricted methods of proof for establishing the consistency of these systems, and to find ways to measure how hard it is to prove their consistency. Since Gödel showed that (almost) every formal system has questions it cannot settle, it makes sense to look for “interesting” questions a given formal system cannot settle, and to figure out how strong a formal system has to be to settle them. To the present day, logicians have been pursuing these questions in a new mathematical discipline, the theory of proofs.

21.2 Definitions

In order to carry out Hilbert’s project of formalizing mathematics and showing that such a formalization is consistent and complete, the first order of business would be that of picking a language, logical framework, and a system of axioms. For our purposes, let us suppose that mathematics can be formalized in a first-order language, i.e., that there is some set of constant symbols, function symbols, and predicate symbols which, together with the connectives and quantifiers of first-order logic, allow us to express the claims of mathematics. Most people agree that such a language exists: the language of set theory, in which \in is the only non-logical symbol. That such a simple language is so expressive is of course a very implausible claim at first sight, and it took a lot of work to establish that practically of all mathematics can be expressed in this very austere vocabulary. To keep things simple, for now, let’s restrict our discussion to arithmetic, so the part of mathematics that just deals with the natural numbers \mathbb{N} . The natural language in which to express facts of arithmetic is \mathcal{L}_A . \mathcal{L}_A contains a single two-place predicate symbol $<$, a single constant symbol 0 , one one-place function symbol $'$, and two two-place function symbols $+$ and \times .

Definition 21.1. A set of sentences Γ is a *theory* if it is closed under entailment, i.e., if $\Gamma = \{\varphi : \Gamma \models \varphi\}$.

There are two easy ways to specify theories. One is as the set of sentences true in some structure. For instance, consider the structure for \mathcal{L}_A in which the domain is \mathbb{N} and all non-logical symbols are interpreted as you would expect.

Definition 21.2. The *standard model of arithmetic* is the structure \mathfrak{N} defined as follows:

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

Definition 21.3. The theory of *true arithmetic* is the set of sentences satisfied in the standard model of arithmetic, i.e.,

$$\mathbf{TA} = \{\varphi : \mathfrak{N} \models \varphi\}.$$

\mathbf{TA} is a theory, for whenever $\mathbf{TA} \models \varphi$, φ is satisfied in every structure which satisfies \mathbf{TA} . Since $\mathfrak{M} \models \mathbf{TA}$, $\mathfrak{M} \models \varphi$, and so $\varphi \in \mathbf{TA}$.

The other way to specify a theory Γ is as the set of sentences entailed by some set of sentences Γ_0 . In that case, Γ is the “closure” of Γ_0 under entailment. Specifying a theory this way is only interesting if Γ_0 is explicitly specified, e.g., if the elements of Γ_0 are listed. At the very least, Γ_0 has to be decidable, i.e., there has to be a computable test for when a sentence counts as an element of Γ_0 or not. We call the sentences in Γ_0 *axioms* for Γ , and Γ *axiomatized* by Γ_0 .

Definition 21.4. A theory Γ is *axiomatized* by Γ_0 iff

$$\Gamma = \{\varphi : \Gamma_0 \models \varphi\}$$

Definition 21.5. The theory \mathbf{Q} axiomatized by the following sentences is known as “Robinson’s \mathbf{Q} ” and is a very simple theory of arithmetic.

$$\forall x \forall y (x' = y' \rightarrow x = y) \quad (Q_1)$$

$$\forall x 0 \neq x' \quad (Q_2)$$

$$\forall x (x \neq 0 \rightarrow \exists y x = y') \quad (Q_3)$$

$$\forall x (x + 0) = x \quad (Q_4)$$

$$\forall x \forall y (x + y') = (x + y)' \quad (Q_5)$$

$$\forall x (x \times 0) = 0 \quad (Q_6)$$

$$\forall x \forall y (x \times y') = ((x \times y) + x) \quad (Q_7)$$

$$\forall x \forall y (x < y \leftrightarrow \exists z (x + z' = y)) \quad (Q_8)$$

The set of sentences $\{Q_1, \dots, Q_8\}$ are the axioms of \mathbf{Q} , so \mathbf{Q} consists of all sentences entailed by them:

$$\mathbf{Q} = \{\varphi : \{Q_1, \dots, Q_8\} \models \varphi\}.$$

Definition 21.6. Suppose $\varphi(x)$ is a formula in \mathcal{L}_A with free variables x and y_1, \dots, y_n . Then any sentence of the form

$$\forall y_1 \dots \forall y_n ((\varphi(0) \wedge \forall x (\varphi(x) \rightarrow \varphi(x'))) \rightarrow \forall x \varphi(x))$$

is an instance of the *induction schema*.

Peano arithmetic \mathbf{PA} is the theory axiomatized by the axioms of \mathbf{Q} together with all instances of the induction schema.

21.2. DEFINITIONS

Every instance of the induction schema is true in \mathfrak{N} . This is easiest to see if the formula φ only has one free variable x . Then $\varphi(x)$ defines a subset X_A of \mathbb{N} in \mathfrak{N} . X_A is the set of all $n \in \mathbb{N}$ such that $\mathfrak{N}, s \models \varphi(x)$ when $s(x) = n$. The corresponding instance of the induction schema is

$$((\varphi(0) \wedge \forall x (\varphi(x) \rightarrow \varphi(x')))) \rightarrow \forall x \varphi(x))$$

If its antecedent is true in \mathfrak{N} , then $0 \in X_A$ and, whenever $n \in X_A$, so is $n + 1$. Since $0 \in X_A$, we get $1 \in X_A$. With $1 \in X_A$ we get $2 \in X_A$. And so on. So for every $n \in \mathbb{N}$, $n \in X_A$. But this means that $\forall x \varphi(x)$ is satisfied in \mathfrak{N} .

Both **Q** and **PA** are axiomatized theories. The big question is, how strong are they? For instance, can **PA** prove all the truths about \mathbb{N} that can be expressed in \mathcal{L}_A ? Specifically, do the axioms of **PA** settle all the questions that can be formulated in \mathcal{L}_A ?

Another way to put this is to ask: Is **PA** = **TA**? For **TA** obviously does prove (i.e., it includes) all the truths about \mathbb{N} , and it settles all the questions that can be formulated in \mathcal{L}_A , since if φ is a sentence in \mathcal{L}_A , then either $\mathfrak{N} \models \varphi$ or $\mathfrak{N} \models \neg\varphi$, and so either **TA** $\models \varphi$ or **TA** $\models \neg\varphi$. Call such a theory complete.

Definition 21.7. A theory Γ is *complete* iff for every sentence φ in its language, either $\Gamma \models \varphi$ or $\Gamma \models \neg\varphi$.

By the Completeness Theorem, $\Gamma \models \varphi$ iff $\Gamma \vdash \varphi$, so Γ is complete iff for every sentence φ in its language, either $\Gamma \vdash \varphi$ or $\Gamma \vdash \neg\varphi$.

Another question we are led to ask is this: Is there a computational procedure we can use to test if a sentence is in **TA**, in **PA**, or even just in **Q**? We can make this more precise by defining when a set (e.g., a set of sentences) is decidable.

Definition 21.8. A set X is *decidable* iff its characteristic function $\chi_X: X \rightarrow \{0, 1\}$, with

$$\chi_X(x) = \begin{cases} 1 & \text{if } x \in X \\ 0 & \text{if } x \notin X \end{cases}$$

is computable.

So our question becomes: Is **TA** (**PA**, **Q**) decidable?

The answer to all these questions will be: no. None of these theories is decidable. However, this phenomenon is not specific to these particular theories. In fact, *any* theory that satisfies certain conditions is subject to the same results. One of these conditions, which **Q** and **PA** satisfy, is that they are axiomatized by a decidable set of axioms.

Definition 21.9. A theory is *axiomatizable* if it is axiomatized by a decidable set of axioms.

Example 21.10. Any theory axiomatized by a finite set of sentences is axiomatizable, since any finite set is decidable. Thus, \mathbf{Q} , for instance, is axiomatizable.

Schematically axiomatized theories like \mathbf{PA} are also axiomatizable. For to test if ψ is among the axioms of \mathbf{PA} , i.e., to compute the function χ_X where $\chi_X(\psi) = 1$ if ψ is an axiom of \mathbf{PA} and $= 0$ otherwise, we can do the following: First, check if ψ is one of the axioms of \mathbf{Q} . If it is, the answer is “yes” and the value of $\chi_X(\psi) = 1$. If not, test if it is an instance of the induction schema. This can be done systematically; in this case, perhaps it’s easiest to see that it can be done as follows: Any instance of the induction schema begins with a number of universal quantifiers, and then a sub-formula that is a conditional. The consequent of that conditional is $\forall x \varphi(x, y_1, \dots, y_n)$ where x and y_1, \dots, y_n are all the free variables of φ and the initial quantifiers of ψ bind the variables y_1, \dots, y_n . Once we have extracted this φ and checked that its free variables match the variables bound by the universal quantifiers at the front and $\forall x$, we go on to check that the antecedent of the conditional matches

$$\varphi(0, y_1, \dots, y_n) \wedge \forall x (\varphi(x, y_1, \dots, y_n) \rightarrow \varphi(x', y_1, \dots, y_n))$$

Again, if it does, ψ is an instance of the induction schema, and if it doesn’t, ψ isn’t.

In answering this question—and the more general question of which theories are complete or decidable—it will be useful to consider also the following definition. Recall that a set X is enumerable iff it is empty or if there is a surjective function $f: \mathbb{N} \rightarrow X$. Such a function is called an enumeration of X .

Definition 21.11. A set X is called *computably enumerable* (c.e. for short) iff it is empty or it has a computable enumeration.

In addition to axiomatizability, another condition on theories to which the incompleteness theorems apply will be that they are strong enough to prove basic facts about computable functions and decidable relations. By “basic facts,” we mean sentences which express what the values of computable functions are for each of their arguments. And by “strong enough” we mean that the theories in question count these sentences among its theorems. For instance, consider a prototypical computable function: addition. The value of $+$ for arguments 2 and 3 is 5, i.e., $2 + 3 = 5$. A sentence in the language of arithmetic that expresses that the value of $+$ for arguments 2 and 3 is 5 is: $(\bar{2} + \bar{3}) = \bar{5}$. And, e.g., \mathbf{Q} proves this sentence. More generally, we would like there to be, for each computable function $f(x_1, x_2)$ a formula $\varphi_f(x_1, x_2, y)$ in \mathcal{L}_A such that $\mathbf{Q} \vdash \varphi_f(\bar{n}_1, \bar{n}_2, \bar{m})$ whenever $f(n_1, n_2) = m$. In this way, \mathbf{Q} proves that the value of f for arguments n_1, n_2 is m . In fact, we require that it proves a bit more, namely that no other number is the value of f for arguments n_1, n_2 . And the same goes for decidable relations. This is made precise in the following two definitions.

21.3. OVERVIEW OF INCOMPLETENESS RESULTS

Definition 21.12. A formula $\varphi(x_1, \dots, x_k, y)$ represents the function $f: \mathbb{N}^k \rightarrow \mathbb{N}$ in Γ iff whenever $f(n_1, \dots, n_k) = m$, then

1. $\Gamma \vdash \varphi(\overline{n_1}, \dots, \overline{n_k}, \overline{m})$, and
2. $\Gamma \vdash \forall y (\varphi(\overline{n_1}, \dots, \overline{n_k}, \overline{y}) \rightarrow y = \overline{m})$.

Definition 21.13. A formula $\varphi(x_1, \dots, x_k)$ represents the relation $R \subseteq \mathbb{N}^k$ iff,

1. whenever $R(n_1, \dots, n_k)$, $\Gamma \vdash \varphi(\overline{n_1}, \dots, \overline{n_k})$, and
2. whenever not $R(n_1, \dots, n_k)$, $\Gamma \vdash \neg \varphi(\overline{n_1}, \dots, \overline{n_k})$.

A theory is “strong enough” for the incompleteness theorems to apply if it represents all computable functions and all decidable relations. \mathbf{Q} and its extensions satisfy this condition, but it will take us a while to establish this—it’s a non-trivial fact about the kinds of things \mathbf{Q} can prove, and it’s hard to show because \mathbf{Q} has only a few axioms from which we’ll have to prove all these facts. However, \mathbf{Q} is a very weak theory. So although it’s hard to prove that \mathbf{Q} represents all computable functions, most interesting theories are stronger than \mathbf{Q} , i.e., prove more than \mathbf{Q} does. And if \mathbf{Q} proves something, any stronger theory does; since \mathbf{Q} represents all computable functions, every stronger theory does. This means that many interesting theories meet this condition of the incompleteness theorems. So our hard work will pay off, since it shows that the incompleteness theorems apply to a wide range of theories. Certainly, any theory aiming to formalize “all of mathematics” must prove everything that \mathbf{Q} proves, since it should at the very least be able to capture the results of elementary computations. So any theory that is a candidate for a theory of “all of mathematics” will be one to which the incompleteness theorems apply.

21.3 Overview of Incompleteness Results

Hilbert expected that mathematics could be formalized in an axiomatizable theory which it would be possible to prove complete and decidable. Moreover, he aimed to prove the consistency of this theory with very weak, “finitary,” means, which would defend classical mathematics against the challenges of intuitionism. Gödel’s incompleteness theorems showed that these goals cannot be achieved.

Gödel’s first incompleteness theorem showed that a version of Russell and Whitehead’s *Principia Mathematica* is not complete. But the proof was actually very general and applies to a wide variety of theories. This means that it wasn’t just that *Principia Mathematica* did not manage to completely capture mathematics, but that *no* acceptable theory does. It took a while to isolate the features of theories that suffice for the incompleteness theorems to apply,

and to generalize Gödel's proof to apply make it depend only on these features. But we are now in a position to state a very general version of the first incompleteness theorem for theories in the language \mathcal{L}_A of arithmetic.

Theorem 21.14. *If Γ is a consistent and axiomatizable theory in \mathcal{L}_A which represents all computable functions and decidable relations, then Γ is not complete.*

To say that Γ is not complete is to say that for at least one sentence φ , $\Gamma \not\vdash \varphi$ and $\Gamma \not\vdash \neg\varphi$. Such a sentence is called *independent* (of Γ). We can in fact relatively quickly prove that there must be independent sentences. But the power of Gödel's proof of the theorem lies in the fact that it exhibits a *specific example* of such an independent sentence. The intriguing construction produces a sentence G_Γ , called a *Gödel sentence* for Γ , which is unprovable because in Γ , G_Γ is equivalent to the claim that G_Γ is unprovable in Γ . It does so *constructively*, i.e., given an axiomatization of Γ and a description of the proof system, the proof gives a method for actually writing down G_Γ .

The construction in Gödel's proof requires that we find a way to express in \mathcal{L}_A the properties of and operations on terms and formulas of \mathcal{L}_A itself. These include properties such as " φ is a sentence," " δ is a derivation of φ ," and operations such as $\varphi[t/x]$. This way must (a) express these properties and relations via a "coding" of symbols and sequences thereof (which is what terms, formulas, derivations, etc. are) as natural numbers (which is what \mathcal{L}_A can talk about). It must (b) do this in such a way that Γ will prove the relevant facts, so we must show that these properties are coded by decidable properties of natural numbers and the operations correspond to computable functions on natural numbers. This is called "arithmetization of syntax."

Before we investigate how syntax can be arithmetized, however, we will consider the condition that Γ is "strong enough," i.e., represents all computable functions and decidable relations. This requires that we give a precise definition of "computable." This can be done in a number of ways, e.g., via the model of Turing machines, or as those functions computable by programs in some general-purpose programming language. Since our aim is to represent these functions and relations in a theory in the language \mathcal{L}_A , however, it is best to pick a simple definition of computability of just numerical functions. This is the notion of *recursive function*. So we will first discuss the recursive functions. We will then show that **Q** already represents all recursive functions and relations. This will allow us to apply the incompleteness theorem to specific theories such as **Q** and **PA**, since we will have established that these are examples of theories that are "strong enough."

The end result of the arithmetization of syntax is a formula $\text{Prov}_\Gamma(x)$ which, via the coding of formulas as numbers, expresses provability from the axioms of Γ . Specifically, if φ is coded by the number n , and $\Gamma \vdash \varphi$, then $\Gamma \vdash \text{Prov}_\Gamma(\bar{n})$. This "provability predicate" for Γ allows us also to express, in a certain sense, the consistency of Γ as a sentence of \mathcal{L}_A : let the "consistency statement" for Γ

21.4. UNDECIDABILITY AND INCOMPLETENESS

be the sentence $\neg \text{Prov}_\Gamma(\bar{n})$, where we take n to be the code of a contradiction, e.g., of \perp . The second incompleteness theorem states that consistent axiomatizable theories also do not prove their own consistency statements. The conditions required for this theorem to apply are a bit more stringent than just that the theory represents all computable functions and decidable relations, but we will show that **PA** satisfies them.

21.4 Undecidability and Incompleteness

Gödel's proof of the incompleteness theorems require arithmetization of syntax. But even without that we can obtain some nice results just on the assumption that a theory represents all decidable relations. The proof is a diagonal argument similar to the proof of the undecidability of the halting problem.

Theorem 21.15. *If Γ is a consistent theory that represents every decidable relation, then Γ is not decidable.*

Proof. Suppose Γ were decidable. We show that if Γ represents every decidable relation, it must be inconsistent.

Decidable properties (one-place relations) are represented by formulas with one free variable. Let $\varphi_0(x), \varphi_1(x), \dots$, be a computable enumeration of all such formulas. Now consider the following set $D \subseteq \mathbb{N}$:

$$D = \{n : \Gamma \vdash \neg \varphi_n(\bar{n})\}$$

The set D is decidable, since we can test if $n \in D$ by first computing $\varphi_n(x)$, and from this $\neg \varphi_n(\bar{n})$. Obviously, substituting the term \bar{n} for every free occurrence of x in $\varphi_n(x)$ and prefixing $\varphi(\bar{n})$ by \neg is a mechanical matter. By assumption, Γ is decidable, so we can test if $\neg \varphi(\bar{n}) \in \Gamma$. If it is, $n \in D$, and if it isn't, $n \notin D$. So D is likewise decidable.

Since Γ represents all decidable properties, it represents D . And the formulas which represent D in Γ are all among $\varphi_0(x), \varphi_1(x), \dots$. So let d be a number such that $\varphi_d(x)$ represents D in Γ . If $d \notin D$, then, since $\varphi_d(x)$ represents D , $\Gamma \vdash \neg \varphi_d(\bar{d})$. But that means that d meets the defining condition of D , and so $d \in D$. This contradicts $d \notin D$. So by indirect proof, $d \in D$.

Since $d \in D$, by the definition of D , $\Gamma \vdash \neg \varphi_d(\bar{d})$. On the other hand, since $\varphi_d(x)$ represents D in Γ , $\Gamma \vdash \varphi_d(\bar{d})$. Hence, Γ is inconsistent. \square

The preceding theorem shows that no theory that represents all decidable relations can be decidable. We will show that **Q** does represent all decidable relations; this means that all theories that include **Q**, such as **PA** and **TA**, also do, and hence also are not decidable.

We can also use this result to obtain a weak version of the first incompleteness theorem. Any theory that is axiomatizable and complete is decidable. Consistent theories that are axiomatizable and represent all decidable properties then cannot be complete.

Theorem 21.16. *If Γ is axiomatizable and complete it is decidable.*

Proof. Any inconsistent theory is decidable, since inconsistent theories contain all sentences, so the answer to the question “is $\varphi \in \Gamma$ ” is always “yes,” i.e., can be decided.

So suppose Γ is consistent, and furthermore is axiomatizable, and complete. Since Γ is axiomatizable, it is computably enumerable. For we can enumerate all the correct derivations from the axioms of Γ by a computable function. From a correct derivation we can compute the sentence it derives, and so together there is a computable function that enumerates all theorems of Γ . A sentence is a theorem of Γ iff $\neg\varphi$ is not a theorem, since Γ is consistent and complete. We can therefore decide if $\varphi \in \Gamma$ as follows. Enumerate all theorems of Γ . When φ appears on this list, we know that $\Gamma \vdash \varphi$. When $\neg\varphi$ appears on this list, we know that $\Gamma \not\vdash \varphi$. Since Γ is complete, one of these cases eventually obtains, so the procedure eventually produces an answer. \square

Corollary 21.17. *If Γ is consistent, axiomatizable, and represents every decidable property, it is not complete.*

Proof. If Γ were complete, it would be decidable by the previous theorem (since it is axiomatizable and consistent). But since Γ represents every decidable property, it is not decidable, by the first theorem. \square

Once we have established that, e.g., \mathbf{Q} , represents all decidable properties, the corollary tells us that \mathbf{Q} must be incomplete. However, its proof does not provide an example of an independent sentence; it merely shows that such a sentence must exist. For this, we have to arithmetize syntax and follow Gödel’s original proof idea. And of course, we still have to show the first claim, namely that \mathbf{Q} does, in fact, represent all decidable properties.

Problems

Problem 21.1. Show that $\mathbf{TA} = \{\varphi : \mathfrak{N} \models \varphi\}$ is not axiomatizable. You may assume that \mathbf{TA} represents all decidable properties.

Chapter 22

Arithmetization of Syntax

22.1 Introduction

In order to connect computability and logic, we need a way to talk about the objects of logic (symbols, terms, formulas, derivations), operations on them, and their properties and relations, in a way amenable to computational treatment. We can do this directly, by considering computable functions and relations on symbols, sequences of symbols, and other objects built from them. Since the objects of logical syntax are all finite and built from an enumerable sets of symbols, this is possible for some models of computation. But other models of computation—such as the recursive functions—are restricted to numbers, their relations and functions. Moreover, ultimately we also want to be able to deal with syntax within certain theories, specifically, in theories formulated in the language of arithmetic. In these cases it is necessary to *arithmetize* syntax, i.e., to represent syntactic objects, operations on them, and their relations, as numbers, arithmetical functions, and arithmetical relations, respectively. The idea, which goes back to Leibniz, is to assign numbers to syntactic objects.

It is relatively straightforward to assign numbers to symbols as their “codes.” Some symbols pose a bit of a challenge, since, e.g., there are infinitely many variables, and even infinitely many function symbols of each arity n . But of course it’s possible to assign numbers to symbols systematically in such a way that, say, v_2 and v_3 are assigned different codes. Sequences of symbols (such as terms and formulas) are a bigger challenge. But if can deal with sequences of numbers purely arithmetically (e.g., by the powers-of-primes coding of sequences), we can extend the coding of individual symbols to coding of sequences of symbols, and then further to sequences or other arrangements of formulas, such as derivations. This extended coding is called “Gödel numbering.” Every term, formula, and derivation is assigned a Gödel number.

By coding sequences of symbols as sequences of their codes, and by choosing 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, of being the Gödel number 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 , 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 function maps the Gödel numbers k , l , m to the Gödel number n . We will see that this function is primitive recursive.

Arithmetization of syntax is not just of abstract interest, although it was originally a non-trivial insight that languages like the language of arithmetic, which do not come with mechanisms for “talking about” languages can, after all, formalize complex properties of expressions. It is then just a small step to ask what a theory in this language, such as Peano arithmetic, can *prove* about its own language (including, e.g., whether sentences are provable or true). This leads us to the famous limitative theorems of Gödel (about unprovability) and Tarski (the undefinability of truth). But the trick of arithmetizing syntax is also important in order to prove some important results in computability theory, e.g., about the computational power of theories or the relationship between different models of computability. The arithmetization of syntax serves as a model for arithmetizing other objects and properties. For instance, it is similarly possible to arithmetize configurations and computations (say, of Turing machines). This makes it possible to simulate computations in one model (e.g., Turing machines) in another (e.g., recursive functions).

22.2 Coding Symbols

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

$$\perp \quad \neg \quad \vee \quad \wedge \quad \rightarrow \quad \forall \quad \exists \quad = \quad (\quad) \quad ,$$

together with enumerable sets of variables and constant symbols, and enumerable sets of function symbols and predicate symbols of arbitrary arity. We can assign *codes* to each of these symbols in such a way that every symbol is

22.2. CODING SYMBOLS

assigned a unique number as its code, and no two different symbols are assigned the same number. We know that this is possible since the set of all symbols is enumerable and so there is a bijection between it and the set of natural numbers. But we want to make sure that we can recover the symbol (as well as some information about it, e.g., the arity of a function symbol) from its code in a computable way. There are many possible ways of doing this, of course. Here is one such way, which uses primitive recursive functions. (Recall that $\langle n_0, \dots, n_k \rangle$ is the number coding the sequence of numbers n_0, \dots, n_k .)

Definition 22.1. 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:

\perp	\neg	\vee	\wedge	\rightarrow	\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 $c_s = \langle 1, i \rangle$.
3. If s is the i -th constant symbol c_i^n , then $c_s = \langle 2, i \rangle$.
4. If s is the i -th n -ary function symbol f_i^n , then $c_s = \langle 3, n, i \rangle$.
5. If s is the i -th n -ary predicate symbol P_i^n , then $c_s = \langle 4, n, i \rangle$.

Proposition 22.2. The following relations are primitive recursive:

1. $\text{Fn}(x, n)$ iff x is the code of f_i^n for some i , i.e., x is the code of an n -ary function symbol.
2. $\text{Pred}(x, n)$ iff x is the code of P_i^n for some i or x is the code of $=$ and $n = 2$, i.e., x is the code of an n -ary predicate symbol.

Definition 22.3. If s_0, \dots, s_{n-1} is a sequence of symbols, its *Gödel number* is $\langle c_{s_0}, \dots, c_{s_{n-1}} \rangle$.

Note that *codes* and *Gödel numbers* are different things. For instance, the variable v_5 has a code $c_{v_5} = \langle 1, 5 \rangle = 2^2 \cdot 3^6$. But the variable v_5 considered as a term is also a sequence of symbols (of length 1). The *Gödel number* $\#v_5\#$ of the term v_5 is $\langle c_{v_5} \rangle = 2^{c_{v_5}+1} = 2^{2^2 \cdot 3^6 + 1}$.

Example 22.4. Recall that if k_0, \dots, k_{n-1} is a sequence of numbers, then the code of the sequence $\langle k_0, \dots, k_{n-1} \rangle$ in the power-of-primes coding is

$$2^{k_0+1} \cdot 3^{k_1+1} \cdot \dots \cdot p_{n-1}^{k_{n-1}},$$

where p_i is the i -th prime (starting with $p_0 = 2$). So for instance, the formula $v_0 = 0$, or, more explicitly, $\langle v_0, c_0 \rangle$, has the Gödel number

$$\langle c_0, c_{\langle v_0, c_0 \rangle} \rangle.$$

Here, c_0 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 $\# \langle v_0, c_0 \rangle$ is

$$\begin{aligned} 2^{c_0+1} \cdot 3^{c_{\langle v_0, c_0 \rangle}+1} \cdot 5^{c_{v_0}+1} \cdot 7^{c_0+1} \cdot 11^{c_{c_0}+1} \cdot 13^{c_{\langle v_0, c_0 \rangle}+1} = \\ 2^{2^1 \cdot 3^8 + 1} \cdot 3^{2^1 \cdot 3^9 + 1} \cdot 5^{2^2 \cdot 3^1 + 1} \cdot 7^{2^1 \cdot 3^{11} + 1} \cdot 11^{2^3 \cdot 3^1 + 1} \cdot 13^{2^1 \cdot 3^{10} + 1} = \\ 2^{13 \cdot 123} \cdot 3^{39 \cdot 367} \cdot 5^{13} \cdot 7^{354 \cdot 295} \cdot 11^{25} \cdot 13^{118 \cdot 099}. \end{aligned}$$

22.3 Coding Terms

A term is simply a certain kind of sequence of symbols: it is built up inductively from constants and variables according to the formation rules for terms. Since sequences of symbols can be coded as numbers—using a coding scheme for the symbols plus a way to code sequences of numbers—assigning Gödel numbers to terms is not difficult. The challenge is rather to show that the property a number has if it is the Gödel number of a correctly formed term is computable, or in fact primitive recursive.

Proposition 22.5. *The relations $\text{Term}(x)$ and $\text{CTerm}(x)$ 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 $s_0, \dots, s_{k-1} = s$ 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, \dots, t_n occurring prior to place i using an n -place function symbol f_j^n .

To show that the corresponding relation on Gödel numbers is primitive recursive, we have to express this condition primitive recursively, i.e., using primitive recursive functions, relations, and bounded quantification.

Suppose y is the number that codes the sequence s_0, \dots, s_{k-1} , i.e., $y = \langle \#s_0, \dots, \#s_{k-1} \rangle$. It codes a formation sequence for the term with Gödel number x iff for all $i < k$:

1. there is a j such that $(y)_i = \#v_j$, or

22.3. CODING TERMS

2. there is a j such that $(y)_i = {}^{\#}c_j^{\#}$, or
3. there is an n and a number $z = \langle z_1, \dots, z_n \rangle$ such that each z_l is equal to some $(y)_{i'}$ for $i' < i$ and

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

and moreover $(y)_{k-1} = x$. The function $\text{flatten}(z)$ turns the sequence $\langle {}^{\#}t_1^{\#}, \dots, {}^{\#}t_n^{\#} \rangle$ into ${}^{\#}t_1, \dots, {}^{\#}t_n^{\#}$ and is primitive recursive.

The indices j, n , the Gödel numbers z_l of the terms t_l , and the code z of the sequence $\langle z_1, \dots, z_n \rangle$, in (3) are all less than y . We can replace k above with $\text{len}(y)$. Hence we can express “ y is the code of a formation sequence of the term with Gödel number x ” in a way that shows that this relation is primitive recursive.

We now just have to convince ourselves that there is a primitive recursive bound on y . But if x is the Gödel number of a term, it must have a formation sequence with at most $\text{len}(x)$ terms (since every term in the formation sequence of s must start at some place in s , and no two subterms can start at the same place). The Gödel number of each subterm of s is of course $\leq x$. Hence, there always is a formation sequence with code $\leq x^{\text{len}(x)}$.

For CTerm, simply leave out the clause for variables. □

Alternative proof of Proposition 22.5. The inductive definition says that constant symbols and variables are terms, and if t_1, \dots, t_n are terms, then so is $f_j^n(t_1, \dots, t_n)$, for any n and j . So terms are formed in stages: constant symbols and variables at stage 0, terms involving one function symbol at stage 1, those involving at least two nested function symbols at stage 2, etc. Let's say that a sequence of symbols s is a term of level l iff s can be formed by applying the inductive definition of terms l (or fewer) times, i.e., it “becomes” a term by stage l or before. So s is a term of level $l + 1$ iff

1. s is a variable v_j , or
2. s is a constant symbol c_j , or
3. s is built from n terms t_1, \dots, t_n of level l and an n -place function symbol f_j^n .

To show that the corresponding relation on Gödel numbers is primitive recursive, we have to express this condition primitive recursively, i.e., using primitive recursive functions, relations, and bounded quantification.

The number x is the Gödel number of a term s of level $l + 1$ iff

1. there is a j such that $x = {}^{\#}v_j^{\#}$, or
2. there is a j such that $x = {}^{\#}c_j^{\#}$, or

3. there is an n , a j , and a number $z = \langle z_1, \dots, z_n \rangle$ such that each z_i is the Gödel number of a term of level l and

$$x = {}^*f_j^n({}^\# \frown \text{flatten}(z) \frown {}^\#)^\#,$$

and moreover $(y)_{k-1} = x$.

The indices j, n , the Gödel numbers z_i of the terms t_i , and the code z of the sequence $\langle z_1, \dots, z_n \rangle$, in (3) are all less than x . So we get a primitive recursive definition by:

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

We can now define $\text{Term}(x)$ by $\text{ITerm}(x, x)$, since the level of a term is always less than the Gödel number of the term. \square

Proposition 22.6. *The function $\text{num}(n) = {}^*\bar{n}^\#$ is primitive recursive.*

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

$$\begin{aligned} \text{num}(0) &= {}^*\text{o}^\# \\ \text{num}(n+1) &= {}^*\iota({}^\# \frown \text{num}(n) \frown {}^\#)^\#. \end{aligned}$$

\square

22.4 Coding Formulas

Proposition 22.7. *The relation $\text{Atom}(x)$ which holds iff x is the Gödel number of an atomic formula, is primitive recursive.*

Proof. The number x is the Gödel number of an atomic formula iff one of the following holds:

1. There are $n, j < x$, and $z < x$ such that for each $i < n$, $\text{Term}((z)_i)$ and $x =$

$${}^*P_j^n({}^\# \frown \text{flatten}(z) \frown {}^\#)^\#.$$

2. There are $z_1, z_2 < x$ such that $\text{Term}(z_1)$, $\text{Term}(z_2)$, and $x =$

$${}^*({}^\# \frown z_1 \frown {}^\#, {}^\# \frown z_2 \frown {}^\#)^\#.$$

3. $x = {}^*\perp^\#$.

22.5. SUBSTITUTION

4. $x = \ulcorner \top \urcorner$.

□

Proposition 22.8. *The relation $\text{Frm}(x)$ 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 $s_0, \dots, s_{k-1} = s$ 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 $\langle s_0, \dots, s_{k-1} \rangle$) is less than the code x of s . □

Proposition 22.9. *The relation $\text{FreeOcc}(x, z, i)$, 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 22.10. *The property $\text{Sent}(x)$ 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 $\text{Sent}(x)$ holds iff

$$(\forall i < \text{len}(x)) (\forall z < x) ((\exists j < z) z = \ulcorner v_j \urcorner \rightarrow \neg \text{FreeOcc}(x, z, i)).$$

□

22.5 Substitution

Proposition 22.11. *There is a primitive recursive function $\text{Subst}(x, y, z)$ with the property that*

$$\text{Subst}(\ulcorner \varphi \urcorner, \ulcorner t \urcorner, \ulcorner u \urcorner) = \ulcorner \varphi[t/u] \urcorner$$

Proof. We can then define a function hSubst by primitive recursion as follows:

$$\begin{aligned} \text{hSubst}(x, y, z, 0) &= \emptyset \\ \text{hSubst}(x, y, z, i + 1) &= \begin{cases} \text{hSubst}(x, y, z, i) \smallfrown y & \text{if } \text{FreeOcc}(x, z, i + 1) \\ \text{append}(\text{hSubst}(x, y, z, i), (x)_{i+1}) & \text{otherwise.} \end{cases} \end{aligned}$$

$\text{Subst}(x, y, z)$ can now be defined as $\text{hSubst}(x, y, z, \text{len}(x))$. □

Proposition 22.12. *The relation $\text{FreeFor}(x, y, z)$, 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. □

22.6 Derivations in LK

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

Definition 22.13. If Γ is a finite set of sentences, $\Gamma = \{\varphi_1, \dots, \varphi_n\}$, then ${}^*\Gamma^\# = \langle {}^*\varphi_1^\#, \dots, {}^*\varphi_n^\# \rangle$.

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

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

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

1. $\langle 0, {}^*\Gamma \Rightarrow \Delta^\# \rangle$ if π consists only of the initial sequent $\Gamma \Rightarrow \Delta$.
2. $\langle 1, {}^*\Gamma \Rightarrow \Delta^\#, k, {}^*\pi'^\# \rangle$ if π ends in an inference with one premise, k is given by the following table according to which rule was used in the last inference, and π' is the immediate subproof ending in the premise of the last inference.

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

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

3. $\langle 2, {}^*\Gamma \Rightarrow \Delta^\#, k, {}^*\pi'^\#, {}^*\pi''^\# \rangle$ if π ends in an inference with two premises, k is given by the following table according to which rule was used in the last inference, and π' , π'' are the immediate subproof ending in the left and right premise of the last inference, respectively.

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

Having settled on a representation of derivations, we must also show that we can manipulate such derivations primitive recursively, and express their essential properties and relations so. Some operations are simple: e.g., given a Gödel number d of a derivation, $(s)_1$ gives us the Gödel number of its end-sequent. Some are much harder. We'll at least sketch how to do this. The goal is to show that the relation " π is a derivation of φ from Γ " is primitive recursive on the Gödel numbers of π and φ .

Proposition 22.14. *The following relations are primitive recursive:*

1. $\varphi \in \Gamma$.

22.6. DERIVATIONS IN LK

2. $\Gamma \subseteq \Delta$.
3. $\Gamma \Rightarrow \Delta$ is an initial sequent.
4. $\Gamma \Rightarrow \Delta$ follows from $\Gamma' \Rightarrow \Delta'$ (and $\Gamma'' \Rightarrow \Delta''$) by a rule of LK.
5. π is a correct LK-derivation.

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

1. $\varphi \in \Gamma$ iff $\# \varphi^\#$ occurs in the sequence $\# \Gamma^\#$, i.e., $\text{IsIn}(x, g) \Leftrightarrow (\exists i < \text{len}(g)) (g)_i = x$. We'll abbreviate this as $x \in g$.
2. $\Gamma \subseteq \Delta$ iff every element of $\# \Gamma^\#$ is also an element of $\# \Delta^\#$, i.e., $\text{SubSet}(g, d) \Leftrightarrow (\forall i < \text{len}(g)) (g)_i \in d$. We'll abbreviate this as $g \subseteq d$.
3. $\Gamma \Rightarrow \Delta$ is an initial sequent if either there is a sentence φ such that $\Gamma \Rightarrow \Delta$ is $\varphi \Rightarrow \varphi$, or there is a term t such that $\Gamma \Rightarrow \Delta$ is $\emptyset \Rightarrow t = t$. In terms of Gödel numbers, $\text{InitSeq}(s)$ holds iff

$$(\exists x < s) (\text{Sent}(x) \wedge s = \langle \langle x \rangle, \langle x \rangle \rangle) \vee \\ (\exists t < s) (\text{Term}(t) \wedge s = \langle 0, \langle \# = (\# \frown t \frown \#, \# \frown t \frown \#)^\# \rangle \rangle).$$

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

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

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

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

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

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

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

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

□

Proposition 22.15. *Suppose Γ is a primitive recursive set of sentences. Then the relation $\text{Prf}_{\Gamma}(x, y)$ expressing "x is the code of a derivation π of $\Gamma_0 \Rightarrow \varphi$ for some finite $\Gamma_0 \subseteq \Gamma$ and x is the Gödel number of φ " is primitive recursive.*

Proof. Suppose " $y \in \Gamma$ " is given by the primitive recursive predicate $R_{\Gamma}(y)$. We have to show that $\text{Prf}_{\Gamma}(x, y)$ which holds iff y is the Gödel number of a sentence φ and x is the code of an **LK**-derivation with end sequent $\Gamma_0 \Rightarrow \varphi$ is primitive recursive.

22.7. DERIVATIONS IN NATURAL DEDUCTION

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

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

□

22.7 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 end-formula, the labels, and the representations of the sub-derivations leading to the premises of the last inference.

Definition 22.16. If δ is a derivation in natural deduction, then $^*\delta^\#$ is

1. $\langle 0, ^*\varphi^\#, n \rangle$ if δ consists only of the assumption φ . The number n is 0 if it is an undischarged assumption, and the numerical label otherwise.
2. $\langle 1, ^*\varphi^\#, n, k, ^*\delta_1^\# \rangle$ if δ ends in an inference with one premise, k is given by the following table according to which rule was used in the last inference, and δ_1 is the immediate subproof ending in the premise of the last inference. n is the label of the inference, or 0 if the inference does not discharge any assumptions.

Rule:	\perp Elim	\neg Intro	\neg Elim	\wedge Elim	\vee Intro
k :	1	2	3	4	5

Rule:	\rightarrow Intro	\forall Intro	\forall Elim	\exists Intro	$=$ Intro
k :	6	7	8	9	10

3. $\langle 2, ^*\varphi^\#, n, k, ^*\delta_1^\#, ^*\delta_2^\# \rangle$ if δ ends in an inference with two premises, k is given by the following table according to which rule was used in the last inference, and δ_1, δ_2 are the immediate subderivations ending in the left and right premise of the last inference, respectively. n is the label of the inference, or 0 if the inference does not discharge any assumptions.

Rule: \perp Intro \wedge Intro \rightarrow Elim
 k: 1 2 3

4. $\langle 3, {}^{\#}\varphi^{\#}, n, {}^{\#}\delta_1^{\#}, {}^{\#}\delta_2^{\#}, {}^{\#}\delta_3^{\#} \rangle$ if δ ends in an \vee Elim inference. $\delta_1, \delta_2, \delta_3$ are the immediate subderivations ending in the left, middle, and right premise of the last inference, respectively, and n is the label of the inference.

Example 22.17. Consider the very simple derivation

$$\frac{\frac{[(\varphi \wedge \psi)]^1}{\varphi} \wedge \text{Elim}}{(\varphi \rightarrow \psi)} \rightarrow \text{Intro}$$

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 premise of \rightarrow Intro would be $d_1 = \langle 1, {}^{\#}\varphi^{\#}, 0, 4, d_0 \rangle$. The Gödel number of the entire derivation then is $\langle 1, {}^{\#}(\varphi \rightarrow \psi)^{\#}, 1, 6, d_1 \rangle$, i.e.,

$$2^2 \cdot 3^{({}^{\#}(\varphi \rightarrow \psi)^{\#} + 1)} \cdot 5^2 \cdot 7^7 \cdot 11^{(2^2 \cdot 3^{({}^{\#}\varphi^{\#} + 1)} \cdot 5^1 \cdot 7^5 \cdot 11^{(2^1 \cdot 3^{({}^{\#}(\varphi \wedge \psi)^{\#} + 1)} \cdot 5^2)})}.$$

Having settled on a representation of derivations, we must also show that we can manipulate such derivations primitive recursively, and express their essential properties and relations so. Some operations are simple: e.g., given a Gödel number d of a derivation, $(d)_1$ gives us the Gödel number of its end-formula. Some are much harder. We'll at least sketch how to do this. The goal is to show that the relation " δ is a derivation of φ from Γ " is primitive recursive on the Gödel numbers of δ and φ .

Proposition 22.18. *The following relations are primitive recursive:*

1. φ occurs as an assumption in δ with label n .
2. All assumption in δ with label n are of the form φ (i.e., we can discharge the assumption φ using label n in δ).
3. φ is an undischarged assumption of δ .
4. An inference with conclusion φ , upper derivations δ_1 (and δ_2, δ_3), and discharge label n is correct.
5. δ is a correct natural deduction derivation.

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

22.7. DERIVATIONS IN NATURAL DEDUCTION

1. We want to show that $\text{Assum}(x, d, n)$, which holds if x is the Gödel number of an assumption of the derivation with Gödel number d labelled n , is primitive recursive. For this we need a helper relation $\text{hAssum}(x, d, n, i)$ which holds if the formula φ with Gödel number x occurs as an initial formula with label n in the derivation with Gödel number d within i inferences up from the end-formula.

$$\begin{aligned}
 \text{hAssum}(x, d, n, 0) &\Leftrightarrow 1 \\
 \text{hAssum}(x, d, n, i + 1) &\Leftrightarrow \\
 &\quad \text{Sent}(x) \wedge (d = \langle 0, x, n \rangle \vee \\
 &\quad ((d)_0 = 1 \wedge \text{hAssum}(x, (d)_4, n, i)) \vee \\
 &\quad ((d)_0 = 2 \wedge (\text{hAssum}(x, (d)_4, n, i) \vee \\
 &\quad \quad \text{hAssum}(x, (d)_5, n, i))) \vee \\
 &\quad ((d)_0 = 3 \wedge (\text{hAssum}(x, (d)_3, n, i) \vee \\
 &\quad \quad \text{hAssum}(x, (d)_2, n, i)) \vee \text{hAssum}(x, (d)_3, n, i))
 \end{aligned}$$

If the number i is large enough, e.g., larger than the maximum number of inferences between an initial formula and the end-formula of δ , it holds of x, d, n , and i iff φ is an initial formula in δ labelled n . The number d itself is larger than that maximum number of inferences. So we can define

$$\text{Assum}(x, d, n) = \text{hAssum}(x, d, n, d).$$

2. We want to show that $\text{Discharge}(x, d, n)$, 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 $(\forall y < d) (\text{Assum}(y, d, n) \rightarrow y = x)$.
3. An occurrence of an assumption is not open if it occurs with label n in a subderivation that ends in a rule with discharge label n . Define the helper relation $\text{hNotOpen}(x, d, n, i)$ as

$$\begin{aligned}
 \text{hNotOpen}(x, d, n, 0) &\Leftrightarrow 1 \\
 \text{hNotOpen}(x, d, n, i + 1) &\Leftrightarrow \\
 &\quad (d)_2 = n \vee \\
 &\quad ((d)_0 = 1 \wedge \text{hNotOpen}(x, (d)_4, n, i)) \vee \\
 &\quad ((d)_0 = 2 \wedge \text{hNotOpen}(x, (d)_4, n, i) \wedge \\
 &\quad \quad \text{hNotOpen}(x, (d)_5, n, i))) \vee \\
 &\quad ((d)_0 = 3 \wedge \text{hNotOpen}(x, (d)_3, n, i) \wedge \\
 &\quad \quad \text{hNotOpen}(x, (d)_4, n, i) \wedge \text{hNotOpen}(x, (d)_5, n, i))
 \end{aligned}$$

Note that all assumptions of the form φ labelled n are discharged in δ iff either the last inference of δ discharges them (i.e., the last inference has label n), or if it is discharged in all of the immediate subderivations.

A formula φ is an open assumption of δ iff it is an initial formula of δ (with label n) and is not discharged in δ (by a rule with label n). We can then define $\text{OpenAssum}(x, d)$ as

$$(\exists n < d) (\text{Assum}(x, d, n, d) \wedge \neg \text{hNotOpen}(x, d, n, d)).$$

4. Here we have to show that for each rule of inference R the relation $\text{FollowsBy}_R(x, d_1, n)$ which holds if x is the Gödel number of the conclusion and d_1 is the Gödel number of a derivation ending in the premise of a correct application of R with label n is primitive recursive, and similarly for rules with two or three premises.

The simplest case is that of the $=$ Intro rule. Here there is no premise, i.e., $d_1 = 0$. However, φ must be of the form $t = t$, for a closed term t . Here, a primitive recursive definition is

$$(\exists t < x) (\text{CITerm}(t) \wedge x = (\#(\# \frown t \frown \#, \# \frown t \frown \#)\#)) \wedge d_1 = 0).$$

For a more complicated example, $\text{FollowsBy}_{\rightarrow \text{Intro}}(x, d_1, n)$ holds iff φ is of the form $(\psi \rightarrow \chi)$, the end-formula of δ is χ , and any initial formula in δ labelled n is of the form ψ . We can express this primitive recursively by

$$\begin{aligned} (\exists y < x) (\text{Sent}(y) \wedge \text{Discharge}(y, d_1) \wedge \\ (\exists z < x) (\text{Sent}(y) \wedge (d_1)_1 = z) \wedge \\ x = (\#(\# \frown y \frown \# \rightarrow \# \frown z \frown \#)\#)) \end{aligned}$$

(Think of y as the Gödel number of ψ and z as that of χ .)

For another example, consider $\exists \text{Intro}$. Here, φ is the conclusion of a correct inference with one upper derivation iff there is a formula ψ , a closed term t and a variable x such that $\psi[t/x]$ is the end-formula of the upper derivation and $\exists x \psi$ is the conclusion φ , i.e., the formula with Gödel number x . So $\text{FollowsBy}_{\exists \text{Intro}}(x, d_1, n)$ holds iff

$$\begin{aligned} \text{Sent}(x) \wedge (\exists y < x) (\exists v < x) (\exists t < d) (\text{Frm}(y) \wedge \text{Term}(t) \wedge \text{Var}(v) \wedge \\ \text{FreeFor}(y, t, v) \wedge \text{Subst}(y, t, v) = (d_1)_1 \wedge x = (\#\exists \# \frown v \frown z)) \end{aligned}$$

5. We first define a helper relation $\text{hDeriv}(d, i)$ which holds if d codes a correct derivation at least to i inferences up from the end sequent. $\text{hDeriv}(d, 0)$

22.7. DERIVATIONS IN NATURAL DEDUCTION

holds always. Otherwise, $\text{hDeriv}(d, i + 1)$ iff either d just codes an assumption or d ends in a correct inference and the codes of the immediate sub-derivations satisfy $\text{hDeriv}(d', i)$.

$$\begin{aligned}
 & \text{hDeriv}(d, 0) \Leftrightarrow 1 \\
 & \text{hDeriv}(d, i + 1) \Leftrightarrow \\
 & \quad (\exists x < d) (\exists n < d) (\text{Sent}(x) \wedge d = \langle 0, x, n \rangle) \vee \\
 & \quad ((d)_0 = 1 \wedge \\
 & \quad \quad ((d)_3 = 1 \wedge \text{FollowsBy}_{\perp\text{Elim}}((d)_1, (d)_4, (d)_2) \vee \\
 & \quad \quad \vdots \\
 & \quad \quad ((d)_3 = 10 \wedge \text{FollowsBy}_{=\text{Intro}}((d)_1, (d)_4, (d)_2)) \wedge \\
 & \quad \quad \text{nDeriv}((d)_4, i)) \vee \\
 & \quad ((d)_0 = 2 \wedge \\
 & \quad \quad ((d)_3 = 1 \wedge \text{FollowsBy}_{\perp\text{Intro}}((d)_1, (d)_4, (d)_5, (d)_2)) \vee \\
 & \quad \quad \vdots \\
 & \quad \quad ((d)_3 = 3 \wedge \text{FollowsBy}_{\rightarrow\text{Elim}}((d)_1, (d)_4, (d)_5, (d)_2)) \wedge \\
 & \quad \quad \text{hDeriv}((d)_4, i) \wedge \text{hDeriv}((d)_5, i)) \vee \\
 & \quad ((d)_0 = 3 \wedge \\
 & \quad \quad \text{FollowsBy}_{\vee\text{Elim}}((d)_1, (d)_3, (d)_4, (d)_5, (d)_2) \wedge \\
 & \quad \quad \text{hDeriv}((d)_3, i) \wedge \text{hDeriv}((d)_4, i) \wedge \text{hDeriv}((d)_5, i))
 \end{aligned}$$

This is a primitive recursive definition. Again we can define $\text{Deriv}(d)$ as $\text{hDeriv}(d, d)$.

□

Proposition 22.19. *Suppose Γ is a primitive recursive set of sentences. Then the relation $\text{Prf}_\Gamma(x, y)$ expressing “ x is the code of a derivation δ of φ from undischarged assumptions in Γ and y is the Gödel number of φ ” is primitive recursive.*

Proof. Suppose “ $y \in \Gamma$ ” is given by the primitive recursive predicate $R_\Gamma(y)$. We have to show that $\text{Prf}_\Gamma(x, y)$ which holds iff y is the Gödel number of a sentence φ and x is the code of a natural deduction derivation with end formula φ and all undischarged assumptions in Γ is primitive recursive.

By the previous proposition, the property $\text{Deriv}(x)$ which holds iff x is the code of a correct derivation δ in natural deduction is primitive recursive. If x is such a code, then $(x)_1$ is the code of the end-formula of δ . Thus we can define $\text{Prf}_\Gamma(x, y)$ by

$$\begin{aligned}
 \text{Prf}_\Gamma(x, y) & \Leftrightarrow \text{Deriv}(x) \wedge (x)_1 = y \wedge \\
 & (\forall z < x) (\text{OpenAssum}(z, x) \rightarrow R_\Gamma(z))
 \end{aligned}$$

□

Problems

Problem 22.1. Show that the function $\text{flatten}(z)$, which turns the sequence $\langle \#t_1\#, \dots, \#t_n\# \rangle$ into $\#t_1, \dots, t_n\#$, is primitive recursive.

Problem 22.2. Give a detailed proof of [Proposition 22.8](#) along the lines of the first proof of [Proposition 22.5](#)

Problem 22.3. Give a detailed proof of [Proposition 22.8](#) along the lines of the alternate proof of [Proposition 22.5](#)

Problem 22.4. Prove [Proposition 22.9](#). You may make use of the fact that any substring of a formula which is a formula is a sub-formula of it.

Problem 22.5. Prove [Proposition 22.12](#)

Problem 22.6. Define the following relations as in [Proposition 22.14](#):

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

Problem 22.7. Define the following relations as in [Proposition 22.18](#):

1. $\text{FollowsBy}_{\rightarrow\text{Elim}}(x, d_1, d_2, n)$,
2. $\text{FollowsBy}_{=\text{Elim}}(x, d_1, d_2, n)$,
3. $\text{FollowsBy}_{\vee\text{Elim}}(x, d_1, d_2, d_3, n)$,
4. $\text{FollowsBy}_{\vee\text{Intro}}(x, d_1, n)$.

For the last one, you will have to also show that you can test primitive recursively if the formula with Gödel number x and the derivation with Gödel number d satisfy the eigenvariable condition, i.e., the eigenvariable a of the $\vee\text{Intro}$ inference occurs neither in x nor in an open assumption of d .

Chapter 23

Representability in \mathbf{Q}

23.1 Introduction

We will describe a very minimal such theory called “ \mathbf{Q} ” (or, sometimes, “Robinson’s \mathbf{Q} ,” after Raphael Robinson). We will say what it means for a function to be *representable* in \mathbf{Q} , and then we will prove the following:

A function is representable in \mathbf{Q} if and only if it is computable.

For one thing, this provides us with another model of computability. But we will also use it to show that the set $\{\varphi : \mathbf{Q} \vdash \varphi\}$ is not decidable, by reducing the halting problem to it. By the time we are done, we will have proved much stronger things than this.

The language of \mathbf{Q} is the language of arithmetic; \mathbf{Q} consists of the following axioms (to be used in conjunction with the other axioms and rules of first-order logic with identity predicate):

$$\forall x \forall y (x' = y' \rightarrow x = y) \quad (\mathbf{Q}_1)$$

$$\forall x \ 0 \neq x' \quad (\mathbf{Q}_2)$$

$$\forall x (x \neq 0 \rightarrow \exists y \ x = y') \quad (\mathbf{Q}_3)$$

$$\forall x (x + 0) = x \quad (\mathbf{Q}_4)$$

$$\forall x \forall y (x + y') = (x + y)' \quad (\mathbf{Q}_5)$$

$$\forall x (x \times 0) = 0 \quad (\mathbf{Q}_6)$$

$$\forall x \forall y (x \times y') = ((x \times y) + x) \quad (\mathbf{Q}_7)$$

$$\forall x \forall y (x < y \leftrightarrow \exists z (z' + x) = y) \quad (\mathbf{Q}_8)$$

For each natural number n , define the numeral \bar{n} to be the term $0''\dots'$ where there are n tick marks in all. So, $\bar{0}$ is the constant symbol 0 by itself, $\bar{1}$ is $0'$, $\bar{2}$ is $0''$, etc.

As a theory of arithmetic, \mathbf{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 \mathbf{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 \mathbf{Q} is interesting is because it has a *finite* set of axioms.

A stronger theory than \mathbf{Q} (called *Peano arithmetic* \mathbf{PA}) is obtained by adding a schema of induction to \mathbf{Q} :

$$(\varphi(0) \wedge \forall x (\varphi(x) \rightarrow \varphi(x'))) \rightarrow \forall x \varphi(x)$$

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

$$\forall y ((\varphi(0) \wedge \forall x (\varphi(x) \rightarrow \varphi(x'))) \rightarrow \forall x \varphi(x))$$

Using instances of the induction schema, one can prove much more from the axioms of \mathbf{PA} than from those of \mathbf{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 23.1. A function $f(x_0, \dots, x_k)$ from the natural numbers to the natural numbers is said to be *representable in \mathbf{Q}* if there is a formula $\varphi_f(x_0, \dots, x_k, y)$ such that whenever $f(n_0, \dots, n_k) = m$, \mathbf{Q} proves

1. $\varphi_f(\overline{n_0}, \dots, \overline{n_k}, \overline{m})$
2. $\forall y (\varphi_f(\overline{n_0}, \dots, \overline{n_k}, y) \rightarrow \overline{m} = y).$

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

Theorem 23.2. *A function is representable in \mathbf{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 \mathbf{Q} . Recall that a function is general recursive if it can be defined from zero, the successor function succ , and the projection functions P_i^n , using composition, primitive recursion, and regular minimization. So one way of showing that every general recursive function is representable in \mathbf{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

23.2. FUNCTIONS REPRESENTABLE IN \mathbf{Q} ARE COMPUTABLE

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 \mathbf{Q} , and the representable functions are closed under composition and regular minimization.

23.2 Functions Representable in \mathbf{Q} are Computable

Lemma 23.3. *Every function that is representable in \mathbf{Q} is computable.*

Proof. Let's first give the intuitive idea for why this is true. If $f(x_0, \dots, x_k)$ is representable in \mathbf{Q} , there is a formula $\varphi(x_0, \dots, x_k, y)$ such that

$$\mathbf{Q} \vdash \varphi_f(\overline{n_0}, \dots, \overline{n_k}, \overline{m}) \quad \text{iff} \quad m = f(n_0, \dots, n_k).$$

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 $\varphi_f(\overline{n_0}, \dots, \overline{n_k}, \overline{m})$. If it is, m must be $= f(n_0, \dots, n_k)$ and we've found the value of f . The search terminates because $\mathbf{Q} \vdash \varphi_f(\overline{n_0}, \dots, \overline{n_k}, \overline{f(n_0, \dots, n_k)})$, 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 $\text{Prf}_{\mathbf{Q}}(d, y)$, which holds iff d is the Gödel number of a derivation of the formula with Gödel number x from the axioms of \mathbf{Q} , is (primitive) recursive. Other primitive recursive functions we'll need are num (Proposition 22.6) and Subst (Proposition 22.11). From these, it is possible to define f by minimization; thus, f is recursive.

First, define

$$\begin{aligned} A(n_0, \dots, n_k, m) = \\ \text{Subst}(\text{Subst}(\dots \text{Subst}(\ulcorner \varphi_f \urcorner, \text{num}(n_0), \ulcorner x_0 \urcorner), \\ \dots), \text{num}(n_k), \ulcorner x_k \urcorner), \text{num}(m), \ulcorner y \urcorner) \end{aligned}$$

This looks complicated, but it's just the function $A(n_0, \dots, n_k, m) = \# \varphi_f(\bar{n}_0, \dots, \bar{n}_k, \bar{m})^\#$.

Now, consider the relation $R(n_0, \dots, n_k, s)$ which holds if $(s)_0$ is the Gödel number of a derivation from \mathbf{Q} of $\varphi_f(\bar{n}_0, \dots, \bar{n}_k, (s)_1)$:

$$R(n_0, \dots, n_k, s) \text{ iff } \text{Prf}_{\mathbf{Q}}((s)_0, A(n_0, \dots, n_k, (s)_1))$$

If we can find an s such that $R(n_0, \dots, n_k, s)$ 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 $A_f(\bar{n}_0, \dots, \bar{n}_k, (s)_1)$. 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, \dots, n_k , there is a derivation δ of $\mathbf{Q} \vdash \varphi_f(\bar{n}_0, \dots, \bar{n}_k, \bar{f}(n_0, \dots, n_k))$, so $R(n_0, \dots, n_k, s)$ holds for $s = \langle \# \delta^\#, f(n_0, \dots, n_k) \rangle$. So, we can write f as

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

□

23.3 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 23.4. *There is a function $\beta(d, i)$ such that for every sequence a_0, \dots, 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 $\langle a_0, \dots, a_n \rangle$, and $\beta(d, i)$ returning the i -th element. (Note that this “coding” does *not* use the prover-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, \dots, 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.

23.3. THE BETA FUNCTION LEMMA

Definition 23.5. 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 23.6. $a \equiv b \pmod{c}$ 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 23.7. Suppose x_0, \dots, x_n are (pairwise) relatively prime. Let y_0, \dots, y_n be any numbers. Then there is a number z such that

$$\begin{aligned} z &\equiv y_0 \pmod{x_0} \\ z &\equiv y_1 \pmod{x_1} \\ &\vdots \\ z &\equiv y_n \pmod{x_n}. \end{aligned}$$

Here is how we will use the Chinese Remainder theorem: if x_0, \dots, x_n are bigger than y_0, \dots, y_n respectively, then we can take z to code the sequence $\langle y_0, \dots, y_n \rangle$. 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, \dots, x_n .

A couple of observations will help us in this regard. Given y_0, \dots, y_n , let

$$j = \max(n, y_0, \dots, y_n) + 1,$$

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, \dots, 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, \dots, a_n , as above, let

$$j = \max(n, a_0, \dots, a_n) + 1,$$

and let $d_1 = j!$. By the observations above, we know that $1 + d_1, 1 + 2d_1, \dots, 1 + (n + 1)d_1$ are relatively prime and all are bigger than a_0, \dots, a_n . By the Chinese Remainder theorem there is a value d_0 such that for each i ,

$$d_0 \equiv a_i \pmod{1 + (i + 1)d_1}$$

23.4. SIMULATING PRIMITIVE RECURSION

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.

23.4 Simulating Primitive Recursion

Now we can show that definition by primitive recursion can be “simulated” by regular minimization using the beta function. Suppose we have $f(\vec{z})$ and $g(u, v, \vec{z})$. Then the function $h(x, \vec{z})$ defined from f and g by primitive recursion is

$$\begin{aligned} h(0, \vec{z}) &= f(\vec{z}) \\ h(x + 1, \vec{z}) &= g(x, h(x, \vec{z}), \vec{z}). \end{aligned}$$

We need to show that h can be defined from f and g using just composition and regular minimization, using the basic functions and functions defined from them using composition and regular minimization (such as β).

Lemma 23.8. *If h can be defined from f and g using primitive recursion, it can be defined from f , g , the functions zero, succ, P_i^n , add, mult, $\chi=$, using composition and regular minimization.*

Proof. First, define an auxiliary function $\hat{h}(x, \vec{z})$ which returns the least number d such that d codes a sequence which satisfies

1. $(d)_0 = f(\vec{z})$, and
2. for each $i < x$, $(d)_{i+1} = g(i, (d)_i, \vec{z})$,

where now $(d)_i$ is short for $\beta(d, i)$. In other words, \hat{h} returns the sequence $\langle h(0, \vec{z}), h(1, \vec{z}), \dots, h(x, \vec{z}) \rangle$. We can write \hat{h} as

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

Note: no primitive recursion is needed here, just minimization. The function we minimize is regular because of the beta function lemma [Lemma 23.4](#).

But now we have

$$h(x, \vec{z}) = \beta(\hat{h}(x, \vec{z}), x),$$

so h can be defined from the basic functions using just composition and regular minimization. \square

23.5 Basic Functions are Representable in \mathbf{Q}

First we have to show that all the basic functions are representable in \mathbf{Q} . In the end, we need to show how to assign to each k -ary basic function $f(x_0, \dots, x_{k-1})$ a formula $\varphi_f(x_0, \dots, x_{k-1}, y)$ 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 \mathbf{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 , \mathbf{Q} proves

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

Proposition 23.9. *The zero function $\text{zero}(x) = 0$ is represented in \mathbf{Q} by $y = 0$.*

Proposition 23.10. *The successor function $\text{succ}(x) = x + 1$ is represented in \mathbf{Q} by $y = x'$.*

Proposition 23.11. *The projection function $P_i^n(x_0, \dots, x_{n-1}) = x_i$ is represented in \mathbf{Q} by $y = x_i$.*

Proposition 23.12. *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 \mathbf{Q} by

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

The proof requires the following lemma.

Lemma 23.13. *Given natural numbers n and m , if $n \neq m$, then $\mathbf{Q} \vdash \bar{n} \neq \bar{m}$.*

Proof. Use induction on n to show that for every m , if $n \neq m$, then $\mathbf{Q} \vdash \bar{n} \neq \bar{m}$.

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 \bar{k} , we can conclude $0 \neq \bar{k}'$. But \bar{k}' is just \bar{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

23.5. BASIC FUNCTIONS ARE REPRESENTABLE IN \mathbf{Q}

for n we have $\mathbf{Q} \vdash \bar{n} \neq \bar{k}$. We have an axiom that says $\forall x \forall y x' = y' \rightarrow x = y$. Using a quantifier axiom, we have $\bar{n}' = \bar{k}' \rightarrow \bar{n} = \bar{k}$. Using propositional logic, we can conclude, in \mathbf{Q} , $\bar{n} \neq \bar{k} \rightarrow \bar{n}' \neq \bar{k}'$. Using modus ponens, we can conclude $\bar{n}' \neq \bar{k}'$, which is what we want, since \bar{k}' is \bar{m} . \square

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 23.12. If $n = m$, then \bar{n} and \bar{m} are the same term, and $\chi_{=}(n, m) = 1$. But $\mathbf{Q} \vdash (\bar{n} = \bar{m} \wedge \bar{1} = \bar{1})$, so it proves $\varphi_{=}(n, m, \bar{1})$. If $n \neq m$, then $\chi_{=}(n, m) = 0$. By Lemma 23.13, $\mathbf{Q} \vdash \bar{n} \neq \bar{m}$ and so also $(\bar{n} \neq \bar{m} \wedge 0 = 0)$. Thus $\mathbf{Q} \vdash \varphi_{=}(n, m, \bar{0})$.

For the second part, we also have two cases. If $n = m$, we have to show that that $\mathbf{Q} \vdash \forall (\varphi_{=}(n, m, y) \rightarrow y = \bar{1})$. Arguing informally, suppose $\varphi_{=}(n, m, y)$, i.e.,

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

The left disjunct implies $y = \bar{1}$ by logic; the right contradicts $\bar{n} = \bar{n}$ which is provable by logic.

Suppose, on the other hand, that $n \neq m$. Then $\varphi_{=}(n, m, y)$ is

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

Here, the left disjunct contradicts $\bar{n} \neq \bar{m}$, which is provable in \mathbf{Q} by Lemma 23.13; the right disjunct entails $y = \bar{0}$. \square

Proposition 23.14. *The addition function $\text{add}(x_0, x_1) = x_0 + x_1$ is represented in \mathbf{Q} by*

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

Lemma 23.15. $\mathbf{Q} \vdash (\bar{n} + \bar{m}) = \overline{n + m}$

Proof. We prove this by induction on m . If $m = 0$, the claim is that $\mathbf{Q} \vdash (\bar{n} + 0) = \bar{n}$. This follows by axiom Q_4 . Now suppose the claim for m ; let's prove the claim for $m + 1$, i.e., prove that $\mathbf{Q} \vdash (\bar{n} + \bar{m} + \bar{1}) = \overline{n + m + 1}$. Note that $\overline{m + 1}$ is just \bar{m}' , and $\overline{n + m + 1}$ is just $\overline{n + m}'$. By axiom Q_5 , $\mathbf{Q} \vdash (\bar{n} + \bar{m}') = (\bar{n} + \bar{m})'$. By induction hypothesis, $\mathbf{Q} \vdash (\bar{n} + \bar{m}) = \overline{n + m}$. So $\mathbf{Q} \vdash (\bar{n} + \bar{m}') = \overline{n + m}'$. \square

Proof of Proposition 23.14. The formula $\varphi_{\text{add}}(x_0, x_1, y)$ representing add is $y = (x_0 + x_1)$. First we show that if $\text{add}(n, m) = k$, then $\mathbf{Q} \vdash \varphi_{\text{add}}(\bar{n}, \bar{m}, \bar{k})$, i.e., $\mathbf{Q} \vdash \bar{k} = (\bar{n} + \bar{m})$. But since $k = n + m$, \bar{k} just is $\overline{n + m}$, and we've shown in Lemma 23.15 that $\mathbf{Q} \vdash (\bar{n} + \bar{m}) = \overline{n + m}$.

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

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

Suppose we have $\bar{n} + \bar{m} = y$. Since

$$\mathbf{Q} \vdash (\bar{n} + \bar{m}) = \overline{n + m},$$

we can replace the left side with $\overline{n + m}$ and get $\overline{n + m} = y$, for arbitrary y . \square

Proposition 23.16. *The multiplication function $\text{mult}(x_0, x_1) = x_0 \cdot x_1$ is represented in \mathbf{Q} by*

$$y = (x_0 \times x_1).$$

Proof. Exercise. \square

Lemma 23.17. $\mathbf{Q} \vdash (\bar{n} \times \bar{m}) = \overline{n \cdot m}$

Proof. Exercise. \square

23.6 Composition is Representable in \mathbf{Q}

Suppose h is defined by

$$h(x_0, \dots, x_{l-1}) = f(g_0(x_0, \dots, x_{l-1}), \dots, g_{k-1}(x_0, \dots, x_{l-1})).$$

where we have already found formulas $\varphi_f, \varphi_{g_0}, \dots, \varphi_{g_{k-1}}$ representing the functions f , and g_0, \dots, 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 \mathbf{Q} proves the relevant formulas.

Proposition 23.18. *If $h(n) = m$, then $\mathbf{Q} \vdash \varphi_h(\bar{n}, \bar{m})$.*

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

$$\mathbf{Q} \vdash \varphi_g(\bar{n}, \bar{k})$$

since φ_g represents g , and

$$\mathbf{Q} \vdash \varphi_f(\bar{k}, \bar{m})$$

23.6. COMPOSITION IS REPRESENTABLE IN \mathbf{Q}

since φ_f represents f . Thus,

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

and consequently also

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

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

Proposition 23.19. *If $h(n) = m$, then $\mathbf{Q} \vdash \forall z (\varphi_h(\bar{n}, z) \rightarrow z = \bar{m})$.*

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

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

since φ_g represents g , and

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

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

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

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

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

Proposition 23.20. *If $\varphi_f(y_0, \dots, y_{k-1}, z)$ represents $f(y_0, \dots, y_{k-1})$ in \mathbf{Q} , and $\varphi_{g_i}(x_0, \dots, x_{l-1}, y)$ represents $g_i(x_0, \dots, x_{l-1})$ in \mathbf{Q} , then*

$$\begin{aligned} \exists y_0, \dots, \exists y_{k-1} (\varphi_{g_0}(x_0, \dots, x_{l-1}, y_0) \wedge \dots \wedge \\ \varphi_{g_{k-1}}(x_0, \dots, x_{l-1}, y_{k-1}) \wedge \varphi_f(y_0, \dots, y_{k-1}, z)) \end{aligned}$$

represents

$$h(x_0, \dots, x_{k-1}) = f(g_0(x_0, \dots, x_{l-1}), \dots, g_{k-1}(x_0, \dots, x_{l-1})).$$

Proof. Exercise. □

23.7 Regular Minimization is Representable in \mathbf{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:

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

In the general case, of course, we would have to replace z with z_0, \dots, z_k .

The proof, again, will involve some lemmas about things \mathbf{Q} is strong enough to prove.

Lemma 23.21. *For every variable x and every natural number n ,*

$$\mathbf{Q} \vdash (x' + \bar{n}) = (x + \bar{n})'.$$

Proof. The proof is, as usual, by induction on n . In the base case, $n = 0$, we need to show that \mathbf{Q} proves $(x' + 0) = (x + 0)'$. But we have:

$$\mathbf{Q} \vdash (x' + 0) = x' \quad \text{by axiom } Q_4 \tag{23.1}$$

$$\mathbf{Q} \vdash (x + 0) = x \quad \text{by axiom } Q_4 \tag{23.2}$$

$$\mathbf{Q} \vdash (x + 0)' = x' \quad \text{by eq. (23.2)} \tag{23.3}$$

$$\mathbf{Q} \vdash (x' + 0) = (x + 0)' \quad \text{by eq. (23.1) and eq. (23.3)}$$

In the induction step, we can assume that we have shown that $\mathbf{Q} \vdash (x' + \bar{n}) = (x + \bar{n})'$. Since $\overline{n+1}$ is \bar{n}' , we need to show that \mathbf{Q} proves $(x' + \bar{n}') = (x + \bar{n}')'$. We have:

$$\mathbf{Q} \vdash (x' + \bar{n}') = (x' + \bar{n})' \quad \text{by axiom } Q_5 \tag{23.4}$$

$$\mathbf{Q} \vdash (x' + \bar{n}') = (x + \bar{n}')' \quad \text{inductive hypothesis} \tag{23.5}$$

$$\mathbf{Q} \vdash (x' + \bar{n})' = (x + \bar{n}')' \quad \text{by eq. (23.4) and eq. (23.5).}$$

□

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 23.22. 1. $\mathbf{Q} \vdash \forall x \neg x < 0$.

2. *For every natural number n ,*

$$\mathbf{Q} \vdash \forall x (x < \overline{n+1} \rightarrow (x = 0 \vee \dots \vee x = \bar{n})).$$

23.7. REGULAR MINIMIZATION IS REPRESENTABLE IN \mathbf{Q}

Proof. Let us do 1 and part of 2, informally (i.e., only giving hints as to how to construct the formal derivation).

For part 1, by the definition of $<$, we need to prove $\neg \exists y (y' + x) = 0$ in \mathbf{Q} , which is equivalent (using the axioms and rules of first-order logic) to $\forall y (y' + x) \neq 0$. Here is the idea: suppose $(y' + x) = 0$. If $x = 0$, we have $(y' + 0) = 0$. But by axiom Q_4 of \mathbf{Q} , we have $(y' + 0) = y'$, and by axiom Q_2 we have $y' \neq 0$, a contradiction. So $\forall y (y' + x) \neq 0$. If $x \neq 0$, by axiom Q_3 , there is a z such that $x = z'$. But then we have $(y' + z') = 0$. By axiom Q_5 , we have $(y' + z)' = 0$, again contradicting axiom Q_2 .

For part 2, use induction on n . Let us consider the base case, when $n = 0$. In that case, we need to show $x < \bar{1} \rightarrow x = 0$. Suppose $x < \bar{1}$. Then by the defining axiom for $<$, we have $\exists y (y' + x) = 0'$. Suppose y has that property; so we have $y' + x = 0'$.

We need to show $x = 0$. By axiom Q_3 , if $x \neq 0$, we get $x = z'$ for some z . Then we have $(y' + z') = 0'$. By axiom Q_5 of \mathbf{Q} , we have $(y' + z)' = 0'$. By axiom Q_1 , we have $(y' + z) = 0$. But this means, by definition, $z < 0$, contradicting part 1. \square

Lemma 23.23. *For every $m \in \mathbb{N}$,*

$$\mathbf{Q} \vdash \forall y ((y < \bar{m} \vee \bar{m} < y) \vee y = \bar{m}).$$

Proof. By induction on m . First, consider the case $m = 0$. $\mathbf{Q} \vdash \forall y (y \neq 0 \rightarrow \exists z y = z')$ by Q_3 . But if $y = z'$, then $(z' + 0) = (y + 0)$ by the logic of $=$. By Q_4 , $(y + 0) = y$, so we have $(z' + 0) = y$, and hence $\exists z (z' + 0) = y$. By the definition of $<$ in Q_8 , $0 < y$. If $0 < y$, then also $0 < y \vee y < 0$. We obtain: $y \neq 0 \rightarrow (0 < y \vee y < 0)$, which is equivalent to $(0 < y \vee y < 0) \vee y = 0$.

Now suppose we have

$$\mathbf{Q} \vdash \forall y ((y < \bar{m} \vee \bar{m} < y) \vee y = \bar{m})$$

and we want to show

$$\mathbf{Q} \vdash \forall y ((y < \overline{m+1} \vee \overline{m+1} < y) \vee y = \overline{m+1})$$

The first disjunct $y < \bar{m}$ is equivalent (by Q_8) to $\exists z (z' + y) = \bar{m}$. If $(z' + y) = \bar{m}$, then also $(z' + y)' = \bar{m}'$. By Q_4 , $(z' + y)' = (z'' + y)$. Hence, $(z'' + y) = \bar{m}'$. We get $\exists u (u' + y) = \overline{m+1}$ by existentially generalizing on z' and keeping in mind that \bar{m}' is $\overline{m+1}$. Hence, if $y < \bar{m}$ then $y < \overline{m+1}$.

Now suppose $\bar{m} < y$, i.e., $\exists z (z' + \bar{m}) = y$. By Q_3 and some logic, we have $z = 0 \vee \exists u z = u'$. If $z = 0$, we have $(0' + \bar{m}) = y$. Since $\mathbf{Q} \vdash (0' + \bar{m}) = \overline{m+1}$,

we have $y = \overline{m+1}$. Now suppose $\exists u z = u'$. Then:

$$\begin{aligned} y &= (z' + \overline{m}) \quad \text{by assumption} \\ (z' + \overline{m}) &= (u'' + \overline{m}) \quad \text{from } z = u' \\ (u'' + \overline{m}) &= (u' + \overline{m})' \quad \text{by Lemma 23.21} \\ (u' + \overline{m})' &= (u' + \overline{m}') \quad \text{by } Q_5, \text{ so} \\ y &= (u' + \overline{m+1}) \end{aligned}$$

By existential generalization, $\exists u (u' + \overline{m+1}) = y$, i.e., $\overline{m+1} < y$. So, if $\overline{m} < y$, then $\overline{m+1} < y \vee y = \overline{m+1}$.

Finally, assume $y = \overline{m}$. Then, since $\mathbf{Q} \vdash (o' + \overline{m}) = \overline{m+1}$, $(o' + y) = \overline{m+1}$. From this we get $\exists z (z' + y) = \overline{m+1}$, or $y < \overline{m+1}$.

Hence, from each disjunct of the case for m , we can obtain the case for $m+1$. \square

Proposition 23.24. *If $\varphi_g(x, z, y)$ represents $g(x, y)$ in \mathbf{Q} , then*

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

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.,

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

Since $\varphi_g(x, z, y)$ represents $g(x, z)$ and $g(m, n) = 0$ if $f(n) = m$, we have

$$\mathbf{Q} \vdash \varphi_g(\overline{m}, \overline{n}, o).$$

If $f(n) = m$, then for every $k < m$, $g(k, n) \neq 0$. So

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

We get that

$$\mathbf{Q} \vdash \forall w (w < \overline{m} \rightarrow \neg \varphi_g(w, \overline{n}, o)). \quad (23.6)$$

by Lemma 23.22 (by (1) in case $m = 0$ and by (2) otherwise).

Now let's show that if $f(n) = m$, then $\mathbf{Q} \vdash \forall y (\varphi_f(\overline{n}, y) \rightarrow y = \overline{m})$. We again sketch the argument informally, leaving the formalization to the reader.

Suppose $\varphi_f(\overline{n}, y)$. From this we get (a) $\varphi_g(y, \overline{n}, o)$ and (b) $\forall w (w < y \rightarrow \neg \varphi_g(w, \overline{n}, o))$. By Lemma 23.23, $(y < \overline{m} \vee \overline{m} < y) \vee y = \overline{m}$. We'll show that both $y < \overline{m}$ and $\overline{m} < y$ leads to a contradiction.

If $\overline{m} < y$, then $\neg \varphi_g(\overline{m}, \overline{n}, o)$ from (b). But $m = f(n)$, so $g(m, n) = 0$, and so $\mathbf{Q} \vdash \varphi_g(\overline{m}, \overline{n}, o)$ since φ_g represents g . So we have a contradiction.

Now suppose $y < \overline{m}$. Then since $\mathbf{Q} \vdash \forall w (w < \overline{m} \rightarrow \neg \varphi_g(w, \overline{n}, o))$ by eq. (23.6), we get $\neg \varphi_g(y, \overline{n}, o)$. This again contradicts (a). \square

23.8 Computable Functions are Representable in \mathbf{Q}

Theorem 23.25. *Every computable function is representable in \mathbf{Q} .*

Proof. For definiteness, and using the Church-Turing Thesis, let's say that a function is computable iff it is general recursive. The general recursive functions are those which can be defined from the zero function zero , the successor function succ , and the projection function P_i^n using composition, primitive recursion, and regular minimization. By Lemma 23.8, any function h that can be defined from f and g can also be defined using composition and regular minimization from f , g , and zero , succ , P_i^n , add , mult , $\chi_{=}$. Consequently, a function is general recursive iff it can be defined from zero , succ , P_i^n , add , mult , $\chi_{=}$ using composition and regular minimization.

We've furthermore shown that the basic functions in question are representable in \mathbf{Q} (Propositions 23.9 to 23.12, 23.14 and 23.16), and that any function defined from representable functions by composition or regular minimization (Proposition 23.20, Proposition 23.24) is also representable. Thus every general recursive function is representable in \mathbf{Q} . \square

We have shown that the set of computable functions can be characterized as the set of functions representable in \mathbf{Q} . In fact, the proof is more general. From the definition of representability, it is not hard to see that any theory extending \mathbf{Q} (or in which one can interpret \mathbf{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.

23.9 Representing Relations

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

Definition 23.26. A relation $R(x_0, \dots, x_k)$ on the natural numbers is *representable in \mathbf{Q}* if there is a formula $\varphi_R(x_0, \dots, x_k)$ such that whenever $R(n_0, \dots, n_k)$ is true, \mathbf{Q} proves $\varphi_R(\overline{n_0}, \dots, \overline{n_k})$, and whenever $R(n_0, \dots, n_k)$ is false, \mathbf{Q} proves $\neg \varphi_R(\overline{n_0}, \dots, \overline{n_k})$.

Theorem 23.27. *A relation is representable in \mathbf{Q} if and only if it is computable.*

Proof. For the forwards direction, suppose $R(x_0, \dots, x_k)$ is represented by the formula $\varphi_R(x_0, \dots, x_k)$. Here is an algorithm for computing R : on input n_0, \dots, n_k , simultaneously search for a proof of $\varphi_R(\bar{n}_0, \dots, \bar{n}_k)$ and a proof of $\neg\varphi_R(\bar{n}_0, \dots, \bar{n}_k)$. 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 $R(x_0, \dots, x_k)$ is computable. By definition, this means that the function $\chi_R(x_0, \dots, x_k)$ is computable. By Theorem 23.2, χ_R is represented by a formula, say $\varphi_{\chi_R}(x_0, \dots, x_k, y)$. Let $\varphi_R(x_0, \dots, x_k)$ be the formula $\varphi_{\chi_R}(x_0, \dots, x_k, \bar{1})$. Then for any n_0, \dots, n_k , if $R(n_0, \dots, n_k)$ is true, then $\chi_R(n_0, \dots, n_k) = 1$, in which case \mathbf{Q} proves $\varphi_{\chi_R}(\bar{n}_0, \dots, \bar{n}_k, \bar{1})$, and so \mathbf{Q} proves $\varphi_R(\bar{n}_0, \dots, \bar{n}_k)$. On the other hand, if $R(n_0, \dots, n_k)$ is false, then $\chi_R(n_0, \dots, n_k) = 0$. This means that \mathbf{Q} proves

$$\forall y (\varphi_{\chi_R}(\bar{n}_0, \dots, \bar{n}_k, y) \rightarrow y = \bar{0}).$$

Since \mathbf{Q} proves $\bar{0} \neq \bar{1}$, \mathbf{Q} proves $\neg\varphi_{\chi_R}(\bar{n}_0, \dots, \bar{n}_k, \bar{1})$, and so it proves $\neg\varphi_R(\bar{n}_0, \dots, \bar{n}_k)$. \square

23.10 Undecidability

We call a theory \mathbf{T} *undecidable* if there is no computational procedure which, after finitely many steps and unfailingly, provides a correct answer to the question “does \mathbf{T} prove φ ?” for any sentence φ in the language of \mathbf{T} . So \mathbf{Q} would be decidable iff there were a computational procedure which decides, given a sentence φ in the language of arithmetic, whether $\mathbf{Q} \vdash \varphi$ or not. We can make this more precise by asking: Is the relation $\text{Prov}_{\mathbf{Q}}(y)$, which holds of y iff y is the Gödel number of a sentence provable in \mathbf{Q} , recursive? The answer is: no.

Theorem 23.28. *\mathbf{Q} is undecidable, i.e., the relation*

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

is not recursive.

Proof. Suppose it were. Then we could solve the halting problem as follows: Given e and n , we know that $\varphi_e(n) \downarrow$ iff there is an s such that $T(e, n, s)$, where T is Kleene’s predicate from Theorem 16.8. Since T is primitive recursive it is representable in \mathbf{Q} by a formula ψ_T , that is, $\mathbf{Q} \vdash \psi_T(\bar{e}, \bar{n}, \bar{s})$ iff $T(e, n, s)$. If $\mathbf{Q} \vdash \psi_T(\bar{e}, \bar{n}, \bar{s})$ then also $\mathbf{Q} \vdash \exists y \psi_T(\bar{e}, \bar{n}, y)$. If no such s exists, then $\mathbf{Q} \vdash \neg\psi_T(\bar{e}, \bar{n}, \bar{s})$ for every s . But \mathbf{Q} is ω -consistent, i.e., if $\mathbf{Q} \vdash \neg\varphi(\bar{n})$ for every $n \in \mathbb{N}$, then $\mathbf{Q} \not\vdash \exists y \varphi(y)$. We know this because the axioms of \mathbf{Q} are true in the standard model \mathfrak{N} . So, $\mathbf{Q} \not\vdash \exists y \psi_T(\bar{e}, \bar{n}, y)$. In other words, $\mathbf{Q} \vdash \exists y \psi_T(\bar{e}, \bar{n}, y)$ 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 $\# \exists y \psi_T(\bar{e}, \bar{n}, y) \#$, let $g(e, n)$ be the primitive recursive function which

23.10. UNDECIDABILITY

does that. So

$$h(e, n) = \begin{cases} 1 & \text{if } \text{Prov}_{\mathbf{Q}}(g(e, n)) \\ 0 & \text{otherwise.} \end{cases}$$

This would show that h is recursive if $\text{Prov}_{\mathbf{Q}}$ is. But h is not recursive, by [Theorem 16.9](#), so $\text{Prov}_{\mathbf{Q}}$ cannot be either. \square

Corollary 23.29. *First-order logic is undecidable.*

Proof. If first-order logic were decidable, provability in \mathbf{Q} would be as well, since $\mathbf{Q} \vdash \varphi$ iff $\vdash \omega \rightarrow \varphi$, where ω is the conjunction of the axioms of \mathbf{Q} . \square

Problems

Problem 23.1. Prove that $y = 0$, $y = x'$, and $y = x_i$ represent zero, succ, and P_i^n , respectively.

Problem 23.2. Prove [Lemma 23.17](#).

Problem 23.3. Use [Lemma 23.17](#) to prove [Proposition 23.16](#).

Problem 23.4. Using the proofs of [Proposition 23.19](#) and [Proposition 23.19](#) as a guide, carry out the proof of [Proposition 23.20](#) in detail.

Problem 23.5. Show that if R is representable in \mathbf{Q} , so is χ_R .

This chapter depends on material in the chapter on computability theory, but can be left out if that hasn't been covered. It's currently a basic conversion of Jeremy Avigad's notes, has not been revised, and is missing exercises.

Chapter 24

Theories and Computability

24.1 Introduction

This section should be rewritten.

We have the following:

1. A definition of what it means for a function to be representable in \mathbf{Q} ([Definition 23.1](#))
2. a definition of what it means for a relation to be representable in \mathbf{Q} ([Definition 23.26](#))
3. a theorem asserting that the representable functions of \mathbf{Q} are exactly the computable ones ([Theorem 23.2](#))
4. a theorem asserting that the representable relations of \mathbf{Q} are exactly the computable ones ([Theorem 23.27](#))

A *theory* is a set of sentences that is deductively closed, that is, with the property that whenever T proves φ then φ is in T . It is probably best to think of a theory as being a collection of sentences, together with all the things that these sentences imply. From now on, I will use \mathbf{Q} to refer to the *theory* consisting of the set of sentences derivable from the eight axioms in [section 23.1](#). Remember that we can code formula of \mathbf{Q} as numbers; if φ is such a formula, let $\ulcorner\varphi\urcorner$ denote the number coding φ . Modulo this coding, we can now ask whether various sets of formulas are computable or not.

24.2 \mathbf{Q} is C.e.-Complete

Theorem 24.1. *\mathbf{Q} is c.e. but not decidable. In fact, it is a complete c.e. set.*

24.3. ω -CONSISTENT EXTENSIONS OF \mathbf{Q} ARE UNDECIDABLE

Proof. It is not hard to see that \mathbf{Q} is c.e., since it is the set of (codes for) sentences y such that there is a proof x of y in \mathbf{Q} :

$$\mathbf{Q} = \{y : \exists x \text{Prf}_{\mathbf{Q}}(x, y)\}.$$

But we know that $\text{Prf}_{\mathbf{Q}}(x, y)$ is computable (in fact, primitive recursive), and any set that can be written in the above form is c.e.

Saying that it is a complete c.e. set is equivalent to saying that $K \leq_m \mathbf{Q}$, where $K = \{x : \varphi_x(x) \downarrow\}$. So let us show that K is reducible to \mathbf{Q} . Since Kleene's predicate $T(e, x, s)$ is primitive recursive, it is representable in \mathbf{Q} , say, by φ_T . Then for every x , we have

$$\begin{aligned} x \in K &\rightarrow \exists s T(x, x, s) \\ &\rightarrow \exists s (\mathbf{Q} \vdash \varphi_T(\bar{x}, \bar{x}, \bar{s})) \\ &\rightarrow \mathbf{Q} \vdash \exists s \varphi_T(\bar{x}, \bar{x}, s). \end{aligned}$$

Conversely, if $\mathbf{Q} \vdash \exists s \varphi_T(\bar{x}, \bar{x}, s)$, then, in fact, for some natural number n the formula $\varphi_T(\bar{x}, \bar{x}, \bar{n})$ must be true. Now, if $T(x, x, n)$ were false, \mathbf{Q} would prove $\neg \varphi_T(\bar{x}, \bar{x}, \bar{n})$, since φ_T represents T . But then \mathbf{Q} proves a false formula, which is a contradiction. So $T(x, x, n)$ must be true, which implies $\varphi_x(x) \downarrow$.

In short, we have that for every x , x is in K if and only if \mathbf{Q} proves $\exists s T(\bar{x}, \bar{x}, s)$. So the function f which takes x to (a code for) the sentence $\exists s T(\bar{x}, \bar{x}, s)$ is a reduction of K to \mathbf{Q} . \square

24.3 ω -Consistent Extensions of \mathbf{Q} are Undecidable

The proof that \mathbf{Q} is c.e.-complete relied on the fact that any sentence provable in \mathbf{Q} is "true" of the natural numbers. The next definition and theorem strengthen this theorem, by pinpointing just those aspects of "truth" that were needed in the proof above. Don't dwell on this theorem too long, though, because we will soon strengthen it even further. We include it mainly for historical purposes: Gödel's original paper used the notion of ω -consistency, but his result was strengthened by replacing ω -consistency with ordinary consistency soon after.

Definition 24.2. A theory \mathbf{T} is ω -consistent if the following holds: if $\exists x \varphi(x)$ is any sentence and \mathbf{T} proves $\neg \varphi(\bar{0}), \neg \varphi(\bar{1}), \neg \varphi(\bar{2}), \dots$ then \mathbf{T} does not prove $\exists x \varphi(x)$.

Theorem 24.3. Let \mathbf{T} be any ω -consistent theory that includes \mathbf{Q} . Then \mathbf{T} is not decidable.

Proof. If \mathbf{T} includes \mathbf{Q} , then \mathbf{T} represents the computable functions and relations. We need only modify the previous proof. As above, if $x \in K$, then \mathbf{T} proves $\exists s \varphi_T(\bar{x}, \bar{x}, s)$. Conversely, suppose \mathbf{T} proves $\exists s \varphi_T(\bar{x}, \bar{x}, s)$. Then x

must be in K : otherwise, there is no halting computation of machine x on input x ; since φ_T represents Kleene's T relation, \mathbf{T} proves $\neg\varphi_T(\bar{x}, \bar{x}, \bar{0})$, $\neg\varphi_T(\bar{x}, \bar{x}, \bar{1})$, \dots , making \mathbf{T} ω -inconsistent. \square

24.4 Consistent Extensions of \mathbf{Q} are Undecidable

Remember that a theory is *consistent* if it does not prove both φ and $\neg\varphi$ for any formula φ . Since anything follows from a contradiction, an inconsistent theory is trivial: every sentence is provable. Clearly, if a theory is ω -consistent, then it is consistent. But being consistent is a weaker requirement (i.e., there are theories that are consistent but not ω -consistent.). We can weaken the assumption in Definition 24.2 to simple consistency to obtain a stronger theorem.

Lemma 24.4. *There is no "universal computable relation." That is, there is no binary computable relation $R(x, y)$, with the following property: whenever $S(y)$ is a unary computable relation, there is some k such that for every y , $S(y)$ is true if and only if $R(k, y)$ is true.*

Proof. Suppose $R(x, y)$ is a universal computable relation. Let $S(y)$ be the relation $\neg R(y, y)$. Since $S(y)$ is computable, for some k , $S(y)$ is equivalent to $R(k, y)$. But then we have that $S(k)$ is equivalent to both $R(k, k)$ and $\neg R(k, k)$, which is a contradiction. \square

Theorem 24.5. *Let \mathbf{T} be any consistent theory that includes \mathbf{Q} . Then \mathbf{T} is not decidable.*

Proof. Suppose \mathbf{T} is a consistent, decidable extension of \mathbf{Q} . We will obtain a contradiction by using \mathbf{T} to define a universal computable relation.

Let $R(x, y)$ hold if and only if

x codes a formula $\theta(u)$, and \mathbf{T} proves $\theta(\bar{y})$.

Since we are assuming that \mathbf{T} is decidable, R is computable. Let us show that R is universal. If $S(y)$ is any computable relation, then it is representable in \mathbf{Q} (and hence \mathbf{T}) by a formula $\theta_S(u)$. Then for every n , we have

$$\begin{aligned} S(\bar{n}) &\rightarrow T \vdash \theta_S(\bar{n}) \\ &\rightarrow R(\ulcorner \theta_S(u) \urcorner, n) \end{aligned}$$

and

$$\begin{aligned} \neg S(\bar{n}) &\rightarrow T \vdash \neg\theta_S(\bar{n}) \\ &\rightarrow T \nvdash \theta_S(\bar{n}) \quad (\text{since } \mathbf{T} \text{ is consistent}) \\ &\rightarrow \neg R(\ulcorner \theta_S(u) \urcorner, n). \end{aligned}$$

That is, for every y , $S(y)$ is true if and only if $R(\ulcorner \theta_S(u) \urcorner, y)$ is. So R is universal, and we have the contradiction we were looking for. \square

24.5. AXIOMATIZABLE THEORIES

Let “true arithmetic” be the theory $\{\varphi : \mathbb{N} \models \varphi\}$, that is, the set of sentences in the language of arithmetic that are true in the standard interpretation.

Corollary 24.6. *True arithmetic is not decidable.*

24.5 Axiomatizable Theories

A theory \mathbf{T} is said to be *axiomatizable* if it has a computable set of axioms A . (Saying that A is a set of axioms for \mathbf{T} means $T = \{\varphi : A \vdash \varphi\}$.) Any “reasonable” axiomatization of the natural numbers will have this property. In particular, any theory with a finite set of axioms is axiomatizable.

Lemma 24.7. *Suppose \mathbf{T} is axiomatizable. Then \mathbf{T} is computably enumerable.*

Proof. Suppose A is a computable set of axioms for \mathbf{T} . To determine if $\varphi \in T$, just search for a proof of φ from the axioms.

Put slightly differently, φ is in \mathbf{T} if and only if there is a finite list of axioms ψ_1, \dots, ψ_k in A and a proof of $(\psi_1 \wedge \dots \wedge \psi_k) \rightarrow \varphi$ in first-order logic. But we already know that any set with a definition of the form “there exists ... such that ...” is c.e., provided the second “...” is computable. \square

24.6 Axiomatizable Complete Theories are Decidable

A theory is said to be *complete* if for every sentence φ , either φ or $\neg\varphi$ is provable.

Lemma 24.8. *Suppose a theory \mathbf{T} is complete and axiomatizable. Then \mathbf{T} is decidable.*

Proof. Suppose \mathbf{T} is complete and A is a computable set of axioms. If \mathbf{T} is inconsistent, it is clearly computable. (Algorithm: “just say yes.”) So we can assume that \mathbf{T} is also consistent.

To decide whether or not a sentence φ is in \mathbf{T} , simultaneously search for a proof of φ from A and a proof of $\neg\varphi$. Since \mathbf{T} is complete, you are bound to find one or another; and since \mathbf{T} is consistent, if you find a proof of $\neg\varphi$, there is no proof of φ .

Put in different terms, we already know that \mathbf{T} is c.e.; so by a theorem we proved before, it suffices to show that the complement of \mathbf{T} is c.e. also. But a formula φ is in $\bar{\mathbf{T}}$ if and only if $\neg\varphi$ is in \mathbf{T} ; so $\bar{\mathbf{T}} \leq_m \mathbf{T}$. \square

24.7 \mathbf{Q} has no Complete, Consistent, Axiomatizable Extensions

Theorem 24.9. *There is no complete, consistent, axiomatizable extension of \mathbf{Q} .*

Proof. We already know that there is no consistent, decidable extension of \mathbf{Q} . But if \mathbf{T} is complete and axiomatized, then it is decidable. \square

This theorem is not that far from Gödel's original 1931 formulation of the First Incompleteness Theorem. Aside from the more modern terminology, the key differences are this: Gödel has " ω -consistent" instead of "consistent"; and he could not say "axiomatizable" in full generality, since the formal notion of computability was not in place yet. (The formal models of computability were developed over the following decade, including by Gödel, and in large part to be able to characterize the kinds of theories that are susceptible to the Gödel phenomenon.)

The theorem says you can't have it all, namely, completeness, consistency, and axiomatizability. If you give up any one of these, though, you can have the other two: \mathbf{Q} is consistent and computably axiomatized, but not complete; the inconsistent theory is complete, and computably axiomatized (say, by $\{0 \neq 0\}$), but not consistent; and the set of true sentences of arithmetic is complete and consistent, but it is not computably axiomatized.

24.8 Sentences Provable and Refutable in \mathbf{Q} are Computably Inseparable

Let $\bar{\mathbf{Q}}$ be the set of sentences whose *negations* are provable in \mathbf{Q} , i.e., $\bar{\mathbf{Q}} = \{\varphi : \mathbf{Q} \vdash \neg\varphi\}$. Remember that disjoint sets A and B are said to be computably inseparable if there is no computable set C such that $A \subseteq C$ and $B \subseteq \bar{C}$.

Lemma 24.10. \mathbf{Q} and $\bar{\mathbf{Q}}$ are computably inseparable.

Proof. Suppose C is a computable set such that $\mathbf{Q} \subseteq C$ and $\bar{\mathbf{Q}} \subseteq \bar{C}$. Let $R(x, y)$ be the relation

x codes a formula $\theta(u)$ and $\theta(\bar{y})$ is in C .

We will show that $R(x, y)$ is a universal computable relation, yielding a contradiction.

Suppose $S(y)$ is computable, represented by $\theta_S(u)$ in \mathbf{Q} . Then

$$\begin{aligned} S(\bar{n}) &\rightarrow \mathbf{Q} \vdash \theta_S(\bar{n}) \\ &\rightarrow \theta_S(\bar{n}) \in C \end{aligned}$$

and

$$\begin{aligned} \neg S(\bar{n}) &\rightarrow \mathbf{Q} \vdash \neg\theta_S(\bar{n}) \\ &\rightarrow \theta_S(\bar{n}) \in \bar{\mathbf{Q}} \\ &\rightarrow \theta_S(\bar{n}) \notin C \end{aligned}$$

So $S(y)$ is equivalent to $R(\#(\theta_S(\bar{u})), y)$. \square

24.9 Theories Consistent with \mathbf{Q} are Undecidable

The following theorem says that not only is \mathbf{Q} undecidable, but, in fact, any theory that does not disagree with \mathbf{Q} is undecidable.

Theorem 24.11. *Let \mathbf{T} be any theory in the language of arithmetic that is consistent with \mathbf{Q} (i.e., $\mathbf{T} \cup \mathbf{Q}$ is consistent). Then \mathbf{T} is undecidable.*

Proof. Remember that \mathbf{Q} has a finite set of axioms, Q_1, \dots, Q_8 . We can even replace these by a single axiom, $\alpha = Q_1 \wedge \dots \wedge Q_8$.

Suppose \mathbf{T} is a decidable theory consistent with \mathbf{Q} . Let

$$C = \{\varphi : \mathbf{T} \vdash \alpha \rightarrow \varphi\}.$$

We show that C would be a computable separation of \mathbf{Q} and $\bar{\mathbf{Q}}$, a contradiction. First, if φ is in \mathbf{Q} , then φ is provable from the axioms of \mathbf{Q} ; by the deduction theorem, there is a proof of $\alpha \rightarrow \varphi$ in first-order logic. So φ is in C .

On the other hand, if φ is in $\bar{\mathbf{Q}}$, then there is a proof of $\alpha \rightarrow \neg\varphi$ in first-order logic. If \mathbf{T} also proves $\alpha \rightarrow \varphi$, then \mathbf{T} proves $\neg\alpha$, in which case $\mathbf{T} \cup \mathbf{Q}$ is inconsistent. But we are assuming $\mathbf{T} \cup \mathbf{Q}$ is consistent, so \mathbf{T} does not prove $\alpha \rightarrow \varphi$, and so φ is not in C .

We've shown that if φ is in \mathbf{Q} , then it is in C , and if φ is in $\bar{\mathbf{Q}}$, then it is in \bar{C} . So C is a computable separation, which is the contradiction we were looking for. \square

This theorem is very powerful. For example, it implies:

Corollary 24.12. *First-order logic for the language of arithmetic (that is, the set $\{\varphi : \varphi \text{ is provable in first-order logic}\}$) is undecidable.*

Proof. First-order logic is the set of consequences of \emptyset , which is consistent with \mathbf{Q} . \square

24.10 Theories in which \mathbf{Q} is Interpretable are Undecidable

We can strengthen these results even more. Informally, an interpretation of a language \mathcal{L}_1 in another language \mathcal{L}_2 involves defining the universe, relation symbols, and function symbols of \mathcal{L}_1 with formulas in \mathcal{L}_2 . Though we won't take the time to do this, one can make this definition precise.

Theorem 24.13. *Suppose \mathbf{T} is a theory in a language in which one can interpret the language of arithmetic, in such a way that \mathbf{T} is consistent with the interpretation of \mathbf{Q} . Then \mathbf{T} is undecidable. If \mathbf{T} proves the interpretation of the axioms of \mathbf{Q} , then no consistent extension of \mathbf{T} is decidable.*

The proof is just a small modification of the proof of the last theorem; one could use a counterexample to get a separation of \mathbf{Q} and $\bar{\mathbf{Q}}$. One can take **ZFC**, Zermelo-Fraenkel set theory with the axiom of choice, to be an axiomatic foundation that is powerful enough to carry out a good deal of ordinary mathematics. In **ZFC** one can define the natural numbers, and via this interpretation, the axioms of \mathbf{Q} are true. So we have

Corollary 24.14. *There is no decidable extension of **ZFC**.*

Corollary 24.15. *There is no complete, consistent, computably axiomatizable extension of **ZFC**.*

The language of **ZFC** has only a single binary relation, \in . (In fact, you don't even need equality.) So we have

Corollary 24.16. *First-order logic for any language with a binary relation symbol is undecidable.*

This result extends to any language with two unary function symbols, since one can use these to simulate a binary relation symbol. The results just cited are tight: it turns out that first-order logic for a language with only *unary* relation symbols and at most one *unary* function symbol is decidable.

One more bit of trivia. We know that the set of sentences in the language $0, ', +, \times, <$ true in the standard model is undecidable. In fact, one can define $<$ in terms of the other symbols, and then one can define $+$ in terms of \times and $'$. So the set of true sentences in the language $0, ', \times$ is undecidable. On the other hand, Presburger has shown that the set of sentences in the language $0, ', +$ true in the language of arithmetic is decidable. The procedure is computationally infeasible, however.

Chapter 25

Incompleteness and Provability

25.1 Introduction

Hilbert thought that a system of axioms for a mathematical structure, such as the natural numbers, is inadequate unless it allows one to derive all true statements about the structure. Combined with his later interest in formal systems of deduction, this suggests that he thought that we should guarantee that, say, the formal systems we are using to reason about the natural numbers is not only consistent, but also *complete*, i.e., every statement in its language is either provable or its negation is. Gödel's first incompleteness theorem shows that no such system of axioms exists: there is no complete, consistent, axiomatizable formal system for arithmetic. In fact, no "sufficiently strong," consistent, axiomatizable mathematical theory is complete.

A more important goal of Hilbert's, the centerpiece of his program for the justification of modern ("classical") mathematics, was to find finitary consistency proofs for formal systems representing classical reasoning. With regard to Hilbert's program, then, Gödel's second incompleteness theorem was a much bigger blow. The second incompleteness theorem can be stated in vague terms, like the first incompleteness theorem. Roughly speaking, it says that no sufficiently strong theory of arithmetic can prove its own consistency. We will have to take "sufficiently strong" to include a little bit more than \mathbf{Q} .

The idea behind Gödel's original proof of the incompleteness theorem can be found in the Epimenides paradox. Epimenides, a Cretan, asserted that all Cretans are liars; a more direct form of the paradox is the assertion "this sentence is false." Essentially, by replacing truth with provability, Gödel was able to formalize a sentence which, in a roundabout way, asserts that it itself is not provable. If that sentence were provable, the theory would then be inconsistent. Assuming ω -consistency—a property stronger than consistency—Gödel was able to show that this sentence is also not refutable from the system of axioms he was considering.

The first challenge is to understand how one can construct a sentence that

refers to itself. For every formula φ in the language of \mathbf{Q} , let $\ulcorner \varphi \urcorner$ denote the numeral corresponding to $\# \varphi \#$. Think about what this means: φ is a formula in the language of \mathbf{Q} , $\# \varphi \#$ is a natural number, and $\ulcorner \varphi \urcorner$ is a *term* in the language of \mathbf{Q} . So every formula φ in the language of \mathbf{Q} has a *name*, $\ulcorner \varphi \urcorner$, which is a term in the language of \mathbf{Q} ; this provides us with a conceptual framework in which formulas in the language of \mathbf{Q} can “say” things about other formulas. The following lemma is known as the fixed-point lemma.

Lemma 25.1. *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} proves $\varphi \leftrightarrow \psi(\ulcorner \varphi \urcorner)$.*

The lemma asserts that given any property $\psi(x)$, there is a sentence φ that asserts “ $\psi(x)$ is true of me.”

How can we construct such a sentence? Consider the following version of the Epimenides paradox, due to Quine:

“Yields falsehood when preceded by its quotation” yields falsehood when preceded by its quotation.

This sentence is not directly self-referential. It simply makes an assertion about the syntactic objects between quotes, and, in doing so, it is on par with sentences like

1. “Robert” is a nice name.
2. “I ran.” is a short sentence.
3. “Has three words” has three words.

But what happens when one takes the phrase “yields falsehood when preceded by its quotation,” and precedes it with a quoted version of itself? Then one has the original sentence! In short, the sentence asserts that it is false.

25.2 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

25.2. THE FIXED-POINT LEMMA

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 $\alpha(\bar{n})$, where $n = \# \alpha(x)^\#$. (From now on, let's abbreviate $\# \alpha(x)^\#$ 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 $diag(n)$ which computes the Gödel number of the diagonalization of the formula with Gödel number n , we could write $\alpha(x)$ as $\psi(diag(x))$. And Quine's version of the liar sentence would then be the diagonalization of it, i.e., $\alpha(\ulcorner \alpha \urcorner)$ or $\psi(diag(\ulcorner \psi(diag(x)) \urcorner))$. Of course, $\psi(x)$ could now be any other property, and the same construction would work. For the incompleteness theorem, we'll take $\psi(x)$ to be " x is unprovable in \mathbf{T} ." Then $\alpha(x)$ would be "yields a sentence unprovable in \mathbf{T} when applied to the Gödel number of its diagonalization."

To formalize this in \mathbf{T} , we have to find a way to formalize $diag$. The function $diag(n)$ is computable, in fact, it is primitive recursive: if n is the Gödel number of a formula $\alpha(x)$, $diag(n)$ 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., $\# \alpha(x)^\#$.) If $diag$ were a function symbol in \mathbf{T} representing the function $diag$, we could take φ to be the formula $\psi(diag(\ulcorner \psi(diag(x)) \urcorner))$. Notice that

$$\begin{aligned} diag(\# \psi(diag(x))^\#) &= \# \psi(diag(\ulcorner \psi(diag(x)) \urcorner))^\# \\ &= \# \varphi^\#. \end{aligned}$$

Assuming \mathbf{T} can prove

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

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

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

Lemma 25.2. *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 $\exists y (\theta_{diag}(x, y) \wedge \psi(y))$ and let φ be its diagonalization, i.e., the formula $\alpha(\ulcorner \alpha(x) \urcorner)$.

Since θ_{diag} represents diag , and $\text{diag}(\ulcorner \alpha(x) \urcorner) = \ulcorner \varphi \urcorner$, \mathbf{Q} can prove

$$\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, \ulcorner \varphi \urcorner) \quad (25.1)$$

$$\forall y (\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y) \rightarrow y = \ulcorner \varphi \urcorner). \quad (25.2)$$

Now we show that $\mathbf{Q} \vdash \varphi \leftrightarrow \psi(\ulcorner \varphi \urcorner)$. We argue informally, using just logic and facts provable in \mathbf{Q} .

First, suppose φ , i.e., $\alpha(\ulcorner \alpha(x) \urcorner)$. Going back to the definition of $\alpha(x)$, we see that $\alpha(\ulcorner \alpha(x) \urcorner)$ just is

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

Consider such a y . Since $\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y)$, by eq. (25.2), $y = \ulcorner \varphi \urcorner$. So, from $\psi(y)$ we have $\psi(\ulcorner \varphi \urcorner)$.

Now suppose $\psi(\ulcorner \varphi \urcorner)$. By eq. (25.1), we have $\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, \ulcorner \varphi \urcorner) \wedge \psi(\ulcorner \varphi \urcorner)$. It follows that $\exists y (\theta_{\text{diag}}(\ulcorner \alpha(x) \urcorner, y) \wedge \psi(y))$. But that's just $\alpha(\ulcorner \alpha(x) \urcorner)$, i.e., φ . \square

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.

25.3 The First Incompleteness Theorem

We can now describe Gödel's original proof of the first incompleteness theorem. Let \mathbf{T} be any computably axiomatized theory in a language extending the language of arithmetic, such that \mathbf{T} includes the axioms of \mathbf{Q} . This means that, in particular, \mathbf{T} represents computable functions and relations.

We have argued that, given a reasonable coding of formulas and proofs as numbers, the relation $\text{Prf}_T(x, y)$ is computable, where $\text{Prf}_T(x, y)$ holds if and only if x is the Gödel number of a derivation of the formula with Gödel number y in \mathbf{T} . In fact, for the particular theory that Gödel had in mind, Gödel was able to show that this relation is primitive recursive, using the list of 45 functions and relations in his paper. The 45th relation, xBy , is just $\text{Prf}_T(x, y)$ for his particular choice of \mathbf{T} . Remember that where Gödel uses the word "recursive" in his paper, we would now use the phrase "primitive recursive."

Since $\text{Prf}_T(x, y)$ is computable, it is representable in \mathbf{T} . We will use $\text{Prf}_T(x, y)$ to refer to the formula that represents it. Let $\text{Prov}_T(y)$ be the formula $\exists x \text{Prf}_T(x, y)$. This describes the 46th relation, $\text{Bew}(y)$, on Gödel's list. As Gödel notes, this is the only relation that "cannot be asserted to be recursive." What he probably meant is this: from the definition, it is not clear that it is computable; and later developments, in fact, show that it isn't.

Definition 25.3. A theory \mathbf{T} is ω -consistent if the following holds: if $\exists x \varphi(x)$ is any sentence and \mathbf{T} proves $\neg\varphi(\bar{0}), \neg\varphi(\bar{1}), \neg\varphi(\bar{2}), \dots$ then \mathbf{T} does not prove $\exists x \varphi(x)$.

25.4. ROSSER'S THEOREM

We can now prove the following.

Theorem 25.4. *Let \mathbf{T} be any ω -consistent, axiomatizable theory extending \mathbf{Q} . Then \mathbf{T} is not complete.*

Proof. Let \mathbf{T} be an axiomatizable theory containing \mathbf{Q} . Then $\text{Prf}_T(x, y)$ is decidable, hence representable in \mathbf{Q} by a formula $\text{Prf}_T(x, y)$. Let $\text{Prov}_T(y)$ be the formula we described above. By the fixed-point lemma, there is a formula γ_T such that \mathbf{Q} (and hence \mathbf{T}) proves

$$\gamma_T \leftrightarrow \neg \text{Prov}_T(\ulcorner \gamma_T \urcorner). \quad (25.3)$$

Note that φ says, in essence, “ φ is not provable.”

We claim that

1. If \mathbf{T} is consistent, \mathbf{T} doesn't prove γ_T
2. If \mathbf{T} is ω -consistent, \mathbf{T} doesn't prove $\neg \gamma_T$.

This means that if \mathbf{T} is ω -consistent, it is incomplete, since it proves neither γ_T nor $\neg \gamma_T$. Let us take each claim in turn.

Suppose \mathbf{T} proves γ_T . Then there is a derivation, and so, for some number m , the relation $\text{Prf}_T(m, \ulcorner \gamma_T \urcorner)$ holds. But then \mathbf{Q} proves the sentence $\text{Prf}_T(\bar{m}, \ulcorner \gamma_T \urcorner)$. So \mathbf{Q} proves $\exists x \text{Prf}_T(x, \ulcorner \gamma_T \urcorner)$, which is, by definition, $\text{Prov}_T(\ulcorner \gamma_T \urcorner)$. By eq. (25.3), \mathbf{Q} proves $\neg \gamma_T$, and since \mathbf{T} extends \mathbf{Q} , so does \mathbf{T} . We have shown that if \mathbf{T} proves γ_T , then it also proves $\neg \gamma_T$, and hence it would be inconsistent.

For the second claim, let us show that if \mathbf{T} proves $\neg \gamma_T$, then it is ω -inconsistent. Suppose \mathbf{T} proves $\neg \gamma_T$. If \mathbf{T} is inconsistent, it is ω -inconsistent, and we are done. Otherwise, \mathbf{T} is consistent, so it does not prove γ_T . Since there is no proof of γ_T in \mathbf{T} , \mathbf{Q} proves

$$\neg \text{Prf}_T(\bar{0}, \ulcorner \gamma_T \urcorner), \neg \text{Prf}_T(\bar{1}, \ulcorner \gamma_T \urcorner), \neg \text{Prf}_T(\bar{2}, \ulcorner \gamma_T \urcorner), \dots$$

and so does \mathbf{T} . On the other hand, by eq. (25.3), $\neg \gamma_T$ is equivalent to $\exists x \text{Prf}_T(x, \ulcorner \gamma_T \urcorner)$. So \mathbf{T} is ω -inconsistent. \square

25.4 Rosser's Theorem

Can we modify Gödel's proof to get a stronger result, replacing “ ω -consistent” with simply “consistent”? The answer is “yes,” using a trick discovered by Rosser. Rosser's trick is to use a “modified” provability predicate $\text{RProv}_T(y)$ instead of $\text{Prov}_T(y)$.

Theorem 25.5. *Let \mathbf{T} be any consistent, axiomatizable theory extending \mathbf{Q} . Then \mathbf{T} is not complete.*

Proof. Recall that $\text{Prov}_T(y)$ is defined as $\exists x \text{Prf}_T(x, y)$, where $\text{Prf}_T(x, y)$ 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 $\text{not}(x)$ be the primitive recursive function which does the following: if x is the code of a formula φ , $\text{not}(x)$ is a code of $\neg\varphi$. Then $\text{Ref}_T(x, y)$ holds iff $\text{Prf}_T(x, \text{not}(y))$. Let $\text{Ref}_T(x, y)$ represent it. Then, if $\mathbf{T} \vdash \neg\varphi$ and δ is a corresponding derivation, $\mathbf{Q} \vdash \text{Ref}_T(\ulcorner\delta\urcorner, \ulcorner\varphi\urcorner)$. We define $\text{RProv}_T(y)$ as

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

Roughly, $\text{RProv}_T(y)$ says “there is a proof of y in \mathbf{T} , and there is no shorter refutation of y .” (You might find it convenient to read $\text{RProv}_T(y)$ as “ y is shmovable.”) Assuming \mathbf{T} is consistent, $\text{RProv}_T(y)$ is true of the same numbers as $\text{Prov}_T(y)$; 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 *inconsistent*, then the two do *not* hold of the same numbers!)

By the fixed-point lemma, there is a formula ρ_T such that

$$\mathbf{Q} \vdash \rho_T \leftrightarrow \neg \text{RProv}_T(\ulcorner\rho_T\urcorner). \quad (25.4)$$

In contrast to the proof of [Theorem 25.4](#), here we claim that if \mathbf{T} is consistent, \mathbf{T} doesn’t prove ρ_T , and \mathbf{T} also doesn’t prove $\neg\rho_T$. (In other words, we don’t need the assumption of ω -consistency.)

First, let’s show that $\mathbf{T} \not\vdash \rho_T$. Suppose it did, so there is a derivation of ρ_T from T ; let n be its Gödel number. Then $\mathbf{Q} \vdash \text{Prf}_T(\bar{n}, \ulcorner\rho_T\urcorner)$, 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 $k < n$, $\mathbf{Q} \vdash \neg \text{Ref}_T(\bar{k}, \ulcorner\rho_T\urcorner)$. By [Lemma 23.22\(2\)](#), $\mathbf{Q} \vdash \forall z (z < \bar{n} \rightarrow \neg \text{Ref}_T(z, \ulcorner\rho_T\urcorner))$. Thus,

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

but that’s just $\text{RProv}_T(\ulcorner\rho_T\urcorner)$. By [eq. \(25.4\)](#), $\mathbf{Q} \vdash \neg\rho_T$. 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 $\mathbf{T} \not\vdash \neg\rho_T$. Again, suppose it did, and suppose n is the Gödel number of a derivation of $\neg\rho_T$. Then $\text{Ref}_T(n, \ulcorner\neg\rho_T\urcorner)$ holds, and since Ref_T represents Ref_T in \mathbf{Q} , $\mathbf{Q} \vdash \text{Ref}_T(\bar{n}, \ulcorner\neg\rho_T\urcorner)$. We’ll again show that \mathbf{T} would then be inconsistent because it would also prove ρ_T . Since $\mathbf{Q} \vdash \rho_T \leftrightarrow \neg \text{RProv}_T(\ulcorner\rho_T\urcorner)$, and since \mathbf{T} extends \mathbf{Q} , it suffices to show that $\mathbf{Q} \vdash \neg \text{RProv}_T(\ulcorner\rho_T\urcorner)$. The sentence $\neg \text{RProv}_T(\ulcorner\rho_T\urcorner)$, i.e.,

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

25.5. COMPARISON WITH GÖDEL'S ORIGINAL PAPER

is logically equivalent to

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

We argue informally using logic, making use of facts about what **Q** proves. Suppose x is arbitrary and $\text{Prf}_T(x, \ulcorner \rho_T \urcorner)$. We already know that $\mathbf{T} \not\vdash \rho_T$, and so for every k , $\mathbf{Q} \vdash \neg \text{Prf}_T(\bar{k}, \ulcorner \rho_T \urcorner)$. Thus, for every k it follows that $x \neq \bar{k}$. In particular, we have (a) that $x \neq \bar{n}$. We also have $\neg(x = \bar{0} \vee x = \bar{1} \vee \dots \vee x = \bar{n} - 1)$ and so by Lemma 23.22(2), (b) $\neg(x < \bar{n})$. By Lemma 23.23, $\bar{n} < x$. Since $\mathbf{Q} \vdash \text{Ref}_T(\bar{n}, \ulcorner \rho_T \urcorner)$, we have $\bar{n} < x \wedge \text{Ref}_T(\bar{n}, \ulcorner \rho_T \urcorner)$, and from that $\exists z (z < x \wedge \text{Ref}_T(z, \ulcorner \rho_T \urcorner))$. Since x was arbitrary we get

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

as required. \square

25.5 Comparison with Gödel's Original Paper

It is worthwhile to spend some time with Gödel's 1931 paper. The introduction sketches the ideas we have just discussed. Even if you just skim through the paper, it is easy to see what is going on at each stage: first Gödel describes the formal system P (syntax, axioms, proof rules); then he defines the primitive recursive functions and relations; then he shows that xBy is primitive recursive, and argues that the primitive recursive functions and relations are represented in **P**. He then goes on to prove the incompleteness theorem, as above. In section 3, he shows that one can take the unprovable assertion to be a sentence in the language of arithmetic. This is the origin of the β -lemma, which is what we also used to handle sequences in showing that the recursive functions are representable in **Q**. Gödel doesn't go so far to isolate a minimal set of axioms that suffice, but we now know that **Q** will do the trick. Finally, in Section 4, he sketches a proof of the second incompleteness theorem.

25.6 The Provability Conditions for PA

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

$$(\varphi(0) \wedge \forall x (\varphi(x) \rightarrow \varphi(x'))) \rightarrow \forall x \varphi(x)$$

for every formula φ . Notice that this is really a *schema*, which is to say, infinitely many axioms (and it turns out that **PA** is *not* finitely axiomatizable). But since one can effectively determine whether or not a string of symbols is an instance of an induction axiom, the set of axioms for **PA** is computable. **PA** is a much more robust theory than **Q**. For example, one can easily prove that addition and multiplication are commutative, using induction in the usual

way. In fact, most finitary number-theoretic and combinatorial arguments can be carried out in **PA**.

Since **PA** is computably axiomatized, the provability predicate $\text{Prf}_{\mathbf{PA}}(x, y)$ is computable and hence represented in **Q** (and so, in **PA**). As before, I will take $\text{Prf}_{\mathbf{PA}}(x, y)$ to denote the formula representing the relation. Let $\text{Prov}_{\mathbf{PA}}(y)$ be the formula $\exists x \text{Prf}_{\mathbf{PA}}(x, y)$, which, intuitively says, “ y is provable from the axioms of **PA**.” The reason we need a little bit more than the axioms of **Q** is we need to know that the theory we are using is strong enough to prove a few basic facts about this provability predicate. In fact, what we need are the following facts:

P1. If $\mathbf{PA} \vdash \varphi$, then $\mathbf{PA} \vdash \text{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner)$

P2. For all formulas φ and ψ ,

$$\mathbf{PA} \vdash \text{Prov}_{\mathbf{PA}}(\ulcorner \varphi \rightarrow \psi \urcorner) \rightarrow (\text{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner) \rightarrow \text{Prov}_{\mathbf{PA}}(\ulcorner \psi \urcorner))$$

P3. For every formula φ ,

$$\mathbf{PA} \vdash \text{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner) \rightarrow \text{Prov}_{\mathbf{PA}}(\ulcorner \text{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner) \urcorner).$$

The only way to verify that these three properties hold is to describe the formula $\text{Prov}_{\mathbf{PA}}(y)$ carefully and use the axioms of **PA** to describe the relevant formal proofs. Conditions (1) and (2) are easy; it is really condition (3) that requires work. (Think about what kind of work it entails...) Carrying out the details would be tedious and uninteresting, so here we will ask you to take it on faith that **PA** has the three properties listed above. A reasonable choice of $\text{Prov}_{\mathbf{PA}}(y)$ will also satisfy

P4. If $\mathbf{PA} \vdash \text{Prov}_{\mathbf{PA}}(\ulcorner \varphi \urcorner)$, then $\mathbf{PA} \vdash \varphi$.

But we will not need this fact.

Incidentally, Gödel was lazy in the same way we are being now. At the end of the 1931 paper, he sketches the proof of the second incompleteness theorem, and promises the details in a later paper. He never got around to it; since everyone who understood the argument believed that it could be carried out (he did not need to fill in the details.)

25.7 The Second Incompleteness Theorem

How can we express the assertion that **PA** doesn’t prove its own consistency? Saying **PA** is inconsistent amounts to saying that **PA** proves $0 = 1$. So we can take $\text{Con}_{\mathbf{PA}}$ to be the formula $\neg \text{Prov}_{\mathbf{PA}}(\ulcorner 0 = 1 \urcorner)$, and then the following theorem does the job:

Theorem 25.6. *Assuming **PA** is consistent, then **PA** does not prove $\text{Con}_{\mathbf{PA}}$.*

25.7. THE SECOND INCOMPLETENESS THEOREM

It is important to note that the theorem depends on the particular representation of $\text{Con}_{\mathbf{PA}}$ (i.e., the particular representation of $\text{Prov}_{\mathbf{PA}}(y)$). All we will use is that the representation of $\text{Prov}_{\mathbf{PA}}(y)$ has the three properties above, so the theorem generalizes to any theory with a provability predicate having these properties.

It is informative to read Gödel's sketch of an argument, since the theorem follows like a good punch line. It goes like this. Let $\gamma_{\mathbf{PA}}$ be the Gödel sentence that we constructed in the proof of [Theorem 25.4](#). We have shown "If \mathbf{PA} is consistent, then \mathbf{PA} does not prove $\gamma_{\mathbf{PA}}$." If we formalize this *in* \mathbf{PA} , we have a proof of

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

Now suppose \mathbf{PA} proves $\text{Con}_{\mathbf{PA}}$. Then it proves $\neg \text{Prov}_{\mathbf{PA}}(\ulcorner \gamma_{\mathbf{PA}} \urcorner)$. But since $\gamma_{\mathbf{PA}}$ is a Gödel sentence, this is equivalent to $\gamma_{\mathbf{PA}}$. So \mathbf{PA} proves $\gamma_{\mathbf{PA}}$.

But: we know that if \mathbf{PA} is consistent, it doesn't prove $\gamma_{\mathbf{PA}}$! So if \mathbf{PA} is consistent, it can't prove $\text{Con}_{\mathbf{PA}}$.

To make the argument more precise, we will let $\gamma_{\mathbf{PA}}$ be the Gödel sentence for \mathbf{PA} and use the provability conditions (1)–(3) above to show that \mathbf{PA} proves $\text{Con}_{\mathbf{PA}} \rightarrow \gamma_{\mathbf{PA}}$. This will show that \mathbf{PA} doesn't prove $\text{Con}_{\mathbf{PA}}$. Here is a sketch

of the proof, in **PA**. (For simplicity, we drop the **PA** subscripts.)

$$\gamma \leftrightarrow \neg \text{Prov}(\ulcorner \gamma \urcorner) \quad (25.5)$$

γ is a Gödel sentence

$$\gamma \rightarrow \neg \text{Prov}(\ulcorner \gamma \urcorner) \quad (25.6)$$

from eq. (25.5)

$$\gamma \rightarrow (\text{Prov}(\ulcorner \gamma \urcorner) \rightarrow \perp) \quad (25.7)$$

from eq. (25.6) by logic

$$\text{Prov}(\ulcorner \gamma \rightarrow (\text{Prov}(\ulcorner \gamma \urcorner) \rightarrow \perp) \urcorner) \quad (25.8)$$

by from eq. (25.7) by condition P1

$$\text{Prov}(\ulcorner \gamma \urcorner) \rightarrow \text{Prov}(\ulcorner (\text{Prov}(\ulcorner \gamma \urcorner) \rightarrow \perp) \urcorner) \quad (25.9)$$

from eq. (25.8) by condition P2

$$\text{Prov}(\ulcorner \gamma \urcorner) \rightarrow (\text{Prov}(\ulcorner \text{Prov}(\ulcorner \gamma \urcorner) \urcorner) \rightarrow \text{Prov}(\ulcorner \perp \urcorner)) \quad (25.10)$$

from eq. (25.9) by condition P2 and logic

$$\text{Prov}(\ulcorner \gamma \urcorner) \rightarrow \text{Prov}(\ulcorner \text{Prov}(\ulcorner \gamma \urcorner) \urcorner) \quad (25.11)$$

by P3

$$\text{Prov}(\ulcorner \gamma \urcorner) \rightarrow \text{Prov}(\ulcorner \perp \urcorner) \quad (25.12)$$

from eq. (25.10) and eq. (25.11) by logic

$$\text{Con} \rightarrow \neg \text{Prov}(\ulcorner \gamma \urcorner) \quad (25.13)$$

contraposition of eq. (25.12) and $\text{Con} \equiv \neg \text{Prov}(\ulcorner \perp \urcorner)$

$$\text{Con} \rightarrow \gamma$$

from eq. (25.5) and eq. (25.13) by logic

The use of logic in the above just elementary facts from propositional logic, e.g., eq. (25.7) uses $\vdash \neg \varphi \leftrightarrow (\varphi \rightarrow \perp)$ and eq. (25.12) uses $\varphi \rightarrow (\psi \rightarrow \chi), \varphi \rightarrow \psi \vdash \varphi \rightarrow \chi$. The use of condition P2 in eq. (25.9) and eq. (25.10) relies on instances of P2, $\text{Prov}(\ulcorner \varphi \rightarrow \psi \urcorner) \rightarrow (\text{Prov}(\ulcorner \varphi \urcorner) \rightarrow \text{Prov}(\ulcorner \psi \urcorner))$. In the first one, $\varphi \equiv \gamma$ and $\psi \equiv \text{Prov}(\ulcorner \gamma \urcorner) \rightarrow \perp$; in the second, $\varphi \equiv \text{Prov}(\ulcorner \gamma \urcorner)$ and $\psi \equiv \perp$.

The more abstract version of the incompleteness theorem is as follows:

Theorem 25.7. *Let \mathbf{T} be any axiomatized theory extending \mathbf{Q} and let $\text{Prov}_{\mathbf{T}}(y)$ be any formula satisfying provability conditions P1–P3 for \mathbf{T} . Then if \mathbf{T} is consistent, then \mathbf{T} does not prove $\text{Con}_{\mathbf{T}}$.*

The moral of the story is that no “reasonable” consistent theory for mathematics can prove its own consistency. Suppose \mathbf{T} is a theory of mathematics that includes \mathbf{Q} and Hilbert’s “finitary” reasoning (whatever that may be). Then, the whole of \mathbf{T} cannot prove the consistency of \mathbf{T} , and so, a fortiori, the finitary fragment can’t prove the consistency of \mathbf{T} either. In that sense, there cannot be a finitary consistency proof for “all of mathematics.”

25.8. LÖB'S THEOREM

There is some leeway in interpreting the term “finitary,” and Gödel, in the 1931 paper, grants the possibility that something we may consider “finitary” may lie outside the kinds of mathematics Hilbert wanted to formalize. But Gödel was being charitable; today, it is hard to see how we might find something that can reasonably be called finitary but is not formalizable in, say, ZFC.

25.8 Löb's Theorem

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

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

It is not provable, because if $\mathbf{T} \vdash \gamma$, (a) by provability condition (1), $\mathbf{T} \vdash \text{Prov}_T(\ulcorner \gamma \urcorner)$, and (b) $\mathbf{T} \vdash \gamma$ together with $\mathbf{T} \vdash \neg\text{Prov}_T(\ulcorner \gamma \urcorner) \leftrightarrow \gamma$ gives $\mathbf{T} \vdash \neg\text{Prov}_T(\ulcorner \gamma \urcorner)$, and so \mathbf{T} would be inconsistent. Now it is natural to ask about the status of a fixed point of $\text{Prov}_T(x)$, i.e., a sentence δ such that

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

If it were provable, $\mathbf{T} \vdash \text{Prov}_T(\ulcorner \delta \urcorner)$ by condition (1), but the same conclusion follows if we apply modus ponens to the equivalence above. Hence, we don't get that \mathbf{T} is inconsistent, at least not by the same argument as in the case of the Gödel sentence. This of course does not show that \mathbf{T} *does* prove δ .

We can make headway on this question if we generalize it a bit. The left-to-right direction of the fixed point equivalence, $\text{Prov}_T(\ulcorner \delta \urcorner) \rightarrow \delta$, is an instance of a general schema called a *reflection principle*: $\text{Prov}_T(\ulcorner \varphi \urcorner) \rightarrow \varphi$. It is called that because it expresses, in a sense, that \mathbf{T} can “reflect” about what it can prove; basically it says, “If \mathbf{T} can prove φ , then φ is true,” for any φ . This is true for sound theories only, of course, and this suggests that theories will in general not prove every instance of it. So which instances can a theory (strong enough, and satisfying the provability conditions) prove? Certainly all those where φ itself is provable. And that's it, as the next result shows.

Theorem 25.8. *Let \mathbf{T} be an axiomatizable theory extending \mathbf{Q} , and suppose $\text{Prov}_T(y)$ is a formula satisfying conditions P1–P3 from [section 25.7](#). If \mathbf{T} proves $\text{Prov}_T(\ulcorner \varphi \urcorner) \rightarrow \varphi$, then in fact \mathbf{T} proves φ .*

Put differently, if $\mathbf{T} \not\vdash \varphi$, then $\mathbf{T} \not\vdash \text{Prov}_T(\ulcorner \varphi \urcorner) \rightarrow \varphi$. This result is known as Löb's theorem.

The heuristic for the proof of Löb's theorem is a clever proof that Santa Claus exists. (If you don't like that conclusion, you are free to substitute any other conclusion you would like.) Here it is:

1. Let X be the sentence, “If X is true, then Santa Claus exists.”

2. Suppose X is true.
3. Then what it says holds; i.e., we have: if X is true, then Santa Claus exists.
4. Since we are assuming X is true, we can conclude that Santa Claus exists, by modus ponens from (2) and (3).
5. We have succeeded in deriving (4), “Santa Claus exists,” from the assumption (2), “ X is true.” By conditional proof, we have shown: “If X is true, then Santa Claus exists.”
6. But this is just the sentence X . So we have shown that X is true.
7. But then, by the argument (2)–(4) above, Santa Claus exists.

A formalization of this idea, replacing “is true” with “is provable,” and “Santa Claus exists” with φ , yields the proof of Löb’s theorem. The trick is to apply the fixed-point lemma to the formula $\text{Prov}_T(y) \rightarrow \varphi$. The fixed point of that corresponds to the sentence X in the preceding sketch.

Proof. Suppose φ is a sentence such that \mathbf{T} proves $\text{Prov}_T(\ulcorner \varphi \urcorner) \rightarrow \varphi$. Let $\psi(y)$ be the formula $\text{Prov}_T(y) \rightarrow \varphi$, and use the fixed-point lemma to find a sentence θ such that \mathbf{T} proves $\theta \leftrightarrow \psi(\ulcorner \theta \urcorner)$. Then each of the following is provable

25.8. LÖB'S THEOREM

in \mathbf{T} :

$$\theta \leftrightarrow (\text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow \varphi) \quad (25.14)$$

θ is a fixed point of $\psi(y)$

$$\theta \rightarrow (\text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow \varphi) \quad (25.15)$$

from eq. (25.14)

$$\text{Prov}_T(\ulcorner \theta \rightarrow (\text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow \varphi) \urcorner) \quad (25.16)$$

from eq. (25.15) by condition P1

$$\text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow \text{Prov}_T(\ulcorner \text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow \varphi \urcorner) \quad (25.17)$$

from eq. (25.16) using condition P2

$$\text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow (\text{Prov}_T(\ulcorner \text{Prov}_T(\ulcorner \theta \urcorner) \urcorner) \rightarrow \text{Prov}_T(\ulcorner \varphi \urcorner)) \quad (25.18)$$

from eq. (25.17) using P2 again

$$\text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow \text{Prov}_T(\ulcorner \text{Prov}_T(\ulcorner \theta \urcorner) \urcorner) \quad (25.19)$$

by provability condition P3

$$\text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow \text{Prov}_T(\ulcorner \varphi \urcorner) \quad (25.20)$$

from eq. (25.18) and eq. (25.19)

$$\text{Prov}_T(\ulcorner \varphi \urcorner) \rightarrow \varphi \quad (25.21)$$

by assumption of the theorem

$$\text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow \varphi \quad (25.22)$$

from eq. (25.20) and eq. (25.21)

$$(\text{Prov}_T(\ulcorner \theta \urcorner) \rightarrow \varphi) \rightarrow \theta \quad (25.23)$$

from eq. (25.14)

$$\theta \quad (25.24)$$

from eq. (25.22) and eq. (25.23)

$$\text{Prov}_T(\ulcorner \theta \urcorner) \quad (25.25)$$

from eq. (25.24) by condition P1

$$\varphi \quad \text{from eq. (25.21) and eq. (25.25)}$$

□

With Löb's theorem in hand, there is a short proof of the first incompleteness theorem (for theories having a provability predicate satisfying conditions P1–P3: if $\mathbf{T} \vdash \text{Prov}_T(\ulcorner \perp \urcorner) \rightarrow \perp$, then $\mathbf{T} \vdash \perp$. If \mathbf{T} is consistent, $\mathbf{T} \not\vdash \perp$. So, $\mathbf{T} \not\vdash \text{Prov}_T(\ulcorner \perp \urcorner) \rightarrow \perp$, i.e., $\mathbf{T} \not\vdash \text{Con}_{\mathbf{T}}$. We can also apply it to show that δ , the fixed point of $\text{Prov}_T(x)$, is provable. For since

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

in particular

$$\mathbf{T} \vdash \text{Prov}_T(\ulcorner \delta \urcorner) \rightarrow \delta$$

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

25.9 The Undefinability of Truth

The notion of *definability* depends on having a formal semantics for the language of arithmetic. We have described a set of formulas and sentences in the language of arithmetic. The “intended interpretation” is to read such sentences as making assertions about the natural numbers, and such an assertion can be true or false. Let \mathfrak{N} be the structure with domain \mathbb{N} and the standard interpretation for the symbols in the language of arithmetic. Then $\mathfrak{N} \models \varphi$ means “ φ is true in the standard interpretation.”

Definition 25.9. A relation $R(x_1, \dots, x_k)$ of natural numbers is *definable* in \mathfrak{N} if and only if there is a formula $\varphi(x_1, \dots, x_k)$ in the language of arithmetic such that for every n_1, \dots, n_k , $R(n_1, \dots, n_k)$ if and only if $\mathfrak{N} \models \varphi(\bar{n}_1, \dots, \bar{n}_k)$.

Put differently, a relation is definable in \mathfrak{N} if and only if it is representable in the theory \mathbf{TA} , where $\mathbf{TA} = \{\varphi : \mathfrak{N} \models \varphi\}$ is the set of true sentences of arithmetic. (If this is not immediately clear to you, you should go back and check the definitions and convince yourself that this is the case.)

Lemma 25.10. *Every computable relation is definable in \mathfrak{N} .*

Proof. It is easy to check that the formula representing a relation in \mathbf{Q} defines the same relation in \mathfrak{N} . \square

Now one can ask, is the converse also true? That is, is every relation definable in \mathfrak{N} computable? The answer is no. For example:

Lemma 25.11. *The halting relation is definable in \mathfrak{N} .*

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

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

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

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

so $\exists s \theta_T(z, x, s)$ defines H in \mathfrak{N} . \square

What about \mathbf{TA} itself? Is it definable in arithmetic? That is: is the set $\{\varphi^\# : \mathfrak{N} \models \varphi\}$ definable in arithmetic? Tarski's theorem answers this in the negative.

25.9. THE UNDEFINABILITY OF TRUTH

Theorem 25.12. *The set of true statements of arithmetic is not definable in arithmetic.*

Proof. Suppose $\theta(x)$ defined it. By the fixed-point lemma, there is a formula φ such that \mathbf{Q} proves $\varphi \leftrightarrow \neg\theta(\ulcorner\varphi\urcorner)$, and hence $\mathfrak{N} \models \varphi \leftrightarrow \neg\theta(\ulcorner\varphi\urcorner)$. But then $\mathfrak{N} \models \varphi$ if and only if $\mathfrak{N} \models \neg\theta(\ulcorner\varphi\urcorner)$, which contradicts the fact that $\theta(y)$ is supposed to define the set of true statements of arithmetic. \square

Tarski applied this analysis to a more general philosophical notion of truth. Given any language L , Tarski argued that an adequate notion of truth for L would have to satisfy, for each sentence X ,

‘ X ’ is true if and only if X .

Tarski’s oft-quoted example, for English, is the sentence

‘Snow is white’ is true if and only if snow is white.

However, for any language strong enough to represent the diagonal function, and any linguistic predicate $T(x)$, we can construct a sentence X satisfying “ X if and only if not $T(\ulcorner X \urcorner)$.” Given that we do not want a truth predicate to declare some sentences to be both true and false, Tarski concluded that one cannot specify a truth predicate for all sentences in a language without, somehow, stepping outside the bounds of the language. In other words, a truth predicate for a language cannot be defined in the language itself.

Problems

Problem 25.1. Show that \mathbf{PA} proves $\gamma_{\mathbf{PA}} \rightarrow \text{Con}_{\mathbf{PA}}$.

Problem 25.2. Let \mathbf{T} be a computably axiomatized theory, and let $\text{Prov}_{\mathbf{T}}$ be a provability predicate for \mathbf{T} . Consider the following four statements:

1. If $T \vdash \varphi$, then $T \vdash \text{Prov}_{\mathbf{T}}(\ulcorner\varphi\urcorner)$.
2. $T \vdash \varphi \rightarrow \text{Prov}_{\mathbf{T}}(\ulcorner\varphi\urcorner)$.
3. If $T \vdash \text{Prov}_{\mathbf{T}}(\ulcorner\varphi\urcorner)$, then $T \vdash \varphi$.
4. $T \vdash \text{Prov}_{\mathbf{T}}(\ulcorner\varphi\urcorner) \rightarrow \varphi$

Under what conditions are each of these statements true?

Problem 25.3. Show that $Q(n) \Leftrightarrow n \in \{\ulcorner\varphi\urcorner : \mathbf{Q} \vdash \varphi\}$ is definable in arithmetic.

Part VII

Second-order Logic

25.9. *THE UNDEFINABILITY OF TRUTH*

This is the beginnings of a part on second-order logic.

Chapter 26

Syntax and Semantics

Basic syntax and semantics for SOL covered so far. As a chapter it's too short. Substitution for second-order variables has to be covered to be able to talk about derivation systems for SOL, and there's some subtle issues there.

26.1 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 : $\mathfrak{M}, s \models \varphi$ 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 $\mathfrak{M}, s \models \varphi$, 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 func-

26.2. TERMS AND FORMULAS

tion variables, n -place relations for predicate variables). For instance, while $\forall v_0 (P_0^1(v_0) \vee \neg P_0^1(v_0))$ is a formula in both first- and second-order logic, in the latter we can also consider $\forall V_0^1 \forall v_0 (V_0^1(v_0) \vee \neg V_0^1(v_0))$ and $\exists V_0^1 \forall v_0 (V_0^1(v_0) \vee \neg V_0^1(v_0))$. Since these contain no free variables, they are sentences of second-order logic. Here, V_0^1 is a second-order 1-place predicate variable. The allowable interpretations of V_0^1 are the same that we can assign to a 1-place predicate symbol like P_0^1 , i.e., subsets of $|\mathfrak{M}|$. Quantification over them then amounts to saying that $\forall v_0 (V_0^1(v_0) \vee \neg V_0^1(v_0))$ 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.

26.2 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 denumerable set of second-order relation variables of every arity n :
 $V_0^n, V_1^n, V_2^n, \dots$
2. A denumerable 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 26.1 (Second-order Terms). The set of *second-order terms* of \mathcal{L} , $\text{Trm}^2(\mathcal{L})$, is defined by adding to [Definition 6.4](#) 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 26.2 (Second-order formula). The set of *second-order formulas* $\text{Frm}^2(\mathcal{L})$ of the language \mathcal{L} is defined by adding to [Definition 6.4](#) 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.

26.3 Satisfaction

To define the satisfaction relation $\mathfrak{M}, s \models \varphi$ for second-order formulas, we have to extend the definitions to cover second-order variables.

Definition 26.3 (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., $s(u_i^n): |\mathfrak{M}|^n \rightarrow |\mathfrak{M}|$;

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 26.4 (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* $\text{Val}_s^{\mathfrak{M}}(t)$ is defined as for first-order terms, plus the following clause:

$$t \equiv u(t_1, \dots, t_n):$$

$$\text{Val}_s^{\mathfrak{M}}(t) = s(u)(\text{Val}_s^{\mathfrak{M}}(t_1), \dots, \text{Val}_s^{\mathfrak{M}}(t_n)).$$

Definition 26.5 (Satisfaction). For second-order formulas φ , the definition of satisfaction is like [Definition 6.34](#) with the addition of:

26.4. SEMANTIC NOTIONS

1. $\varphi \equiv X^n t_1, \dots, t_n$: $\mathfrak{M}, s \models \varphi$ iff $\langle \text{Val}_s^{\mathfrak{M}}(t_1), \dots, \text{Val}_s^{\mathfrak{M}}(t_n) \rangle \in s(X^n)$.
2. $\varphi \equiv \forall X \psi$: $\mathfrak{M}, s \models \varphi$ iff for every X -variant s' of s , $\mathfrak{M}, s' \models \psi$.
3. $\varphi \equiv \exists X \psi$: $\mathfrak{M}, s \models \varphi$ iff there is an X -variant s' of s so that $\mathfrak{M}, s' \models \psi$.
4. $\varphi \equiv \forall u \psi$: $\mathfrak{M}, s \models \varphi$ iff for every u -variant s' of s , $\mathfrak{M}, s' \models \psi$.
5. $\varphi \equiv \exists u \psi$: $\mathfrak{M}, s \models \varphi$ iff there is an u -variant s' of s so that $\mathfrak{M}, s' \models \psi$.

Example 26.6. $\mathfrak{M}, s \models \forall z (Xz \leftrightarrow \neg Yz)$ whenever $s(Y) = |\mathfrak{M}| \setminus s(X)$. So for instance, let $|\mathfrak{M}| = \{1, 2, 3\}$, $s(X) = \{1, 2\}$ and $s(Y) = \{3\}$.

$\mathfrak{M}, s \models \exists Y (\exists y Yy \wedge \forall z (Xz \leftrightarrow \neg Yz))$ if there is an $s' \sim_Y s$ such that $\mathfrak{M}, s' \models (\exists y Yy \wedge \forall z (Xz \leftrightarrow \neg Yz))$. And that is the case iff $s'(Y) \neq \emptyset$ (so that $\mathfrak{M}, s' \models \exists y Yy$) and, as before, $s'(Y) = |\mathfrak{M}| \setminus s'(X)$. In other words, $\mathfrak{M}, s \models \exists Y (\exists y Yy \wedge \forall z (Xz \leftrightarrow \neg Yz))$ iff $|\mathfrak{M}| \setminus s(X)$ is non-empty, or, $s(X) \neq |\mathfrak{M}|$. So, the formula is satisfied, e.g., if $s(X) = \{1, 2\}$ but not if $s(X) = \{1, 2, 3\}$.

26.4 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 26.7 (Validity). A sentence φ is *valid*, $\models \varphi$, iff $\mathfrak{M} \models \varphi$ for every structure \mathfrak{M} .

Definition 26.8 (Entailment). A set of sentences Γ *entails* a sentence φ , $\Gamma \models \varphi$, iff for every structure \mathfrak{M} with $\mathfrak{M} \models \Gamma$, $\mathfrak{M} \models \varphi$.

Definition 26.9 (Satisfiability). A set of sentences Γ is *satisfiable* if $\mathfrak{M} \models \Gamma$ for some structure \mathfrak{M} . If Γ is not satisfiable it is called *unsatisfiable*.

26.5 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 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.

Example 26.10. 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 $\varphi_R(v_0, v_1)$ with only the variables v_0 and v_1 free, such that $R(x, y)$ holds (i.e., $\langle x, y \rangle \in R$) iff $\mathfrak{M}, s \models \varphi_R(v_0, v_1)$ for $s(v_0) = x$ and $s(v_1) = y$. For instance, in first-order logic we can define the identity relation $\text{Id}_{|\mathfrak{M}|}$ (i.e., $\{\langle x, x \rangle : x \in |\mathfrak{M}|\}$) by the formula $v_0 = v_1$. In second-order logic, we can define this relation *without* $=$. For if x and y 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 x and y are different, then they are not elements of the same subsets: e.g., $x \in \{x\}$ but $y \notin \{x\}$ if $x \neq y$. So “being elements of the same subsets of $|\mathfrak{M}|$ ” is a relation that holds of x and y iff $x = y$. 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 $\text{Id}_{|\mathfrak{M}|}$:

$$\forall X (X(v_0) \leftrightarrow X(v_1))$$

Example 26.11. If R is a two-place predicate symbol, $R^{\mathfrak{M}}$ is a two-place relation on $|\mathfrak{M}|$. Its *transitive closure* R^* is the relation that holds between x and y if for some z_1, \dots, z_k , $R(x, z_1), R(z_1, z_2), \dots, R(z_k, y)$ holds. This includes the case if $k = 0$, i.e., if $R(x, y)$ holds. 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 :

$$\begin{aligned} \psi_R(X) \equiv & \forall x \forall y (R(x, y) \rightarrow X(x, y)) \wedge \\ & \forall x \forall y \forall z ((X(x, y) \wedge X(y, z)) \rightarrow X(x, z)) \end{aligned}$$

Here, somewhat confusingly, we use R as the predicate symbol for R . 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

$$R^*(X) \equiv \psi_R(X) \wedge \forall Y (\psi_R(Y) \rightarrow \forall x \forall y (X(x, y) \rightarrow Y(x, y)))$$

$\mathfrak{M}, s \models R^*(X)$ iff $s(X) = R^*$. The transitive closure of R cannot be expressed in first-order logic.

26.6 Describing Infinite and Enumerable Domains

A set M is (Dedekind) infinite iff there is an injective function $f: M \rightarrow M$ which is not surjective, i.e., with $\text{dom}(f) \neq M$. In first-order logic, we can

26.6. DESCRIBING INFINITE AND ENUMERABLE DOMAINS

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 $\text{ran}(f) \neq |\mathfrak{M}|$:

$$\forall x \forall y (f(x) = f(y) \rightarrow x = y) \wedge \exists y \forall x y \neq f(x)$$

If \mathfrak{M} satisfies this sentence, $f^{\mathfrak{M}} : |\mathfrak{M}| \rightarrow |\mathfrak{M}|$ 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 have the sentence in pure second-order logic

$$\text{Inf} \equiv \exists u (\forall x \forall y (u(x) = u(y) \rightarrow x = y) \wedge \exists y \forall x y \neq u(x))$$

$\mathfrak{M} \models \text{Inf}$ iff $|\mathfrak{M}|$ is infinite. We can then define $\text{Fin} \equiv \neg \text{Inf}$; $\mathfrak{M} \models \text{Fin}$ 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 26.12. $\mathfrak{M} \models \text{Inf}$ iff $|\mathfrak{M}|$ is infinite.

Proof. $\mathfrak{M} \models \text{Inf}$ iff $\mathfrak{M}, s \models \forall x \forall y (u(x) = u(y) \rightarrow x = y) \wedge \exists y \forall x y \neq u(x)$ 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 $f : |\mathfrak{M}| \rightarrow |\mathfrak{M}|$ with $\text{dom}(f) \neq |\mathfrak{M}|$, then $s(u) = f$ is such a variable assignment. \square

A set M is enumerable if there is an enumeration

$$m_0, m_1, m_2, \dots$$

of its elements (without repetitions). Such an enumeration exists iff there is an element $z \in M$ and a function $f : M \rightarrow M$ such that $z, f(z), f(f(z))$ 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 $z, f(z), f(f(z)), \dots$, is an enumeration of M , and M is enumerable. 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 enumerable:

$$\text{Count} \equiv \exists z \exists u \forall X ((X(z) \wedge \forall x (X(x) \rightarrow X(u(x)))) \rightarrow \forall x X(x))$$

Proposition 26.13. $\mathfrak{M} \models \text{Count}$ iff $|\mathfrak{M}|$ is enumerable.

Proof. Suppose $|\mathfrak{M}|$ is enumerable, and let m_0, m_1, \dots , be an enumeration. By removing repetitions 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

$$\mathfrak{M}, s \models \forall X ((X(z) \wedge \forall x (X(x) \rightarrow X(u(x)))) \rightarrow \forall x X(x))$$

Suppose $s' \sim_X s$, and $M = s'(X)$. Suppose further that $\mathfrak{M}, s' \models (X(z) \wedge \forall x (X(x) \rightarrow X(u(x))))$. 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 $\mathfrak{M} \models \forall x X(x)s'$. Since s' was an arbitrary X -variant of s , we are done.

Now assume that

$$\mathfrak{M}, s \models \forall X ((X(z) \wedge \forall x (X(x) \rightarrow X(u(x)))) \rightarrow \forall x X(x))$$

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

$$\mathfrak{M}, s' \models ((X(z) \wedge \forall x (X(x) \rightarrow X(u(x)))) \rightarrow \forall x X(x))$$

by assumption. Also, $\mathfrak{M}, s' \models X(z)$ since $s'(X) = M \ni m = s'(z)$, and also $\mathfrak{M}, s' \models \forall x (X(x) \rightarrow X(u(x)))$ since whenever $x \in M$ also $f(x) \in M$. So, since both antecedent and conditional are satisfied, the consequent must also be: $\mathfrak{M}, s' \models \forall x X(x)$. But that means that $M = |\mathfrak{M}|$, and so $|\mathfrak{M}|$ is enumerable. \square

Problems

Problem 26.1. Show that $\forall X (X(v_0) \rightarrow X(v_1))$ (note: \rightarrow not \leftrightarrow !) defines $\text{Id}_{|\mathfrak{M}|}$.

Problem 26.2. The sentence $\text{Inf} \wedge \text{Count}$ is true in all and only denumerable domains. Adjust the definition of Count so that it becomes a different sentence that directly expresses that the domain is denumerable, and prove that it does.

Chapter 27

Metatheory of Second-order Logic

27.1 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 $f: \mathbb{N} \rightarrow \text{Sent}(\mathcal{L})$ 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 denumerable 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.

27.2 Second-order Arithmetic

Recall that the theory **PA** of Peano arithmetic includes the eight axioms of **Q**,

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

plus all sentences of the form

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

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 **PA**², induction can be stated as a single sentence. **PA**² consists of the first eight axioms above plus the (second-order) *induction axiom*:

$$\forall X \, (X(0) \wedge \forall x \, (X(x) \rightarrow X(x'))) \rightarrow \forall x \, X(x)$$

It says that if a subset X of the domain contains $0^{\mathfrak{M}}$ and with any $x \in |\mathfrak{M}|$ also contains $i^{\mathfrak{M}}(x)$ (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 \bar{n} for $n \in \mathbb{N}$. A model of **PA**² contains no non-standard numbers.

Theorem 27.1. *If $\mathfrak{M} \models \mathbf{PA}^2$ then $|\mathfrak{M}| = \{\text{Val}^n(M) : n \in \mathbb{N}\}$.*

Proof. Let $N = \{\text{Val}^{\mathfrak{M}}(\bar{n}) : n \in \mathbb{N}\}$, and suppose $\mathfrak{M} \models \mathbf{PA}^2$. Of course, for any $n \in \mathbb{N}$, $\text{Val}^{\mathfrak{M}}(\bar{n}) \in |\mathfrak{M}|$, so $N \subseteq |\mathfrak{M}|$.

Now for inclusion in the other direction. Consider a variable assignment s with $s(X) = N$. By assumption,

$$\begin{aligned} \mathfrak{M} &\models \forall X \, (X(0) \wedge \forall x \, (X(x) \rightarrow X(x'))) \rightarrow \forall x \, X(x), \text{ thus} \\ \mathfrak{M}, s &\models (X(0) \wedge \forall x \, (X(x) \rightarrow X(x'))) \rightarrow \forall x \, X(x). \end{aligned}$$

Consider the antecedent of this conditional. $\text{Val}^{\mathfrak{M}}(0) \in N$, and so $\mathfrak{M}, s \models X(0)$. The second conjunct, $\forall x \, (X(x) \rightarrow X(x'))$ is also satisfied. For suppose $x \in N$. By definition of N , $x = \text{Val}^{\mathfrak{M}}(\bar{n})$ for some n . That gives $i^{\mathfrak{M}}(x) = \text{Val}^{\mathfrak{M}}(\overline{n+1}) \in N$. So, $i^{\mathfrak{M}}(x) \in N$.

27.3. SECOND-ORDER LOGIC IS NOT AXIOMATIZABLE

We have that $\mathfrak{M}, s \models X(0) \wedge \forall x (X(x) \rightarrow X(x'))$. 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, $|\mathfrak{M}| \subseteq N$. \square

Corollary 27.2. *Any two models of \mathbf{PA}^2 are isomorphic.*

Proof. By [Theorem 27.1](#), the domain of any model of \mathbf{PA}^2 is exhausted by $\text{Val}^{\mathfrak{M}}(\bar{n})$. Any such model is also a model of \mathbf{Q} . By [Proposition 13.3](#), any such model is standard, i.e., isomorphic to \mathfrak{N} . \square

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 27.3. *Let \mathbf{PA}^{2+} be the second-order theory containing the first two arithmetical axioms (the successor axioms) and the second-order induction axiom. $>$, $+$, and \times are definable in \mathbf{PA}^{2+} .*

Proof. Exercise. \square

Corollary 27.4. $\mathfrak{M} \models \mathbf{PA}^2$ iff $\mathfrak{M} \models \mathbf{PA}^{2+}$.

Proof. Immediate from [Proposition 27.3](#). \square

27.3 Second-order Logic is not Axiomatizable

Theorem 27.5. *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. \square

Theorem 27.6. *There is no sound and complete proof system for second-order logic.*

Proof. Let φ be a sentence in the language of arithmetic. $\mathfrak{N} \models \varphi$ iff $\mathbf{PA}^2 \models \varphi$. Let P be the conjunction of the nine axioms of \mathbf{PA}^2 . $\mathbf{PA}^2 \models \varphi$ iff $\models P \rightarrow \varphi$, i.e., $\mathfrak{M} \models P \rightarrow \varphi$. 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 $\mathbf{PA}^2 \models \varphi$ iff $\mathfrak{N} \models \varphi$. Thus if there were a sound and complete proof system for second-order logic, we could use it to define a computable enumeration $f: \mathbb{N} \rightarrow \text{Sent}(\mathcal{L}_A)$ 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. \square

27.4 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 $\Gamma \models \varphi$, then already $\Gamma_0 \models \varphi$ for some finite subset $\Gamma_0 \subseteq \Gamma$. In this version it is an immediate corollary of the completeness theorem: for if $\Gamma \models \varphi$, 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 27.7. *Second-order logic is not compact.*

Proof. Recall that

$$\text{Inf} \equiv \exists u \forall x \forall y (u(x) = u(y) \rightarrow x = y)$$

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.,

$$\varphi^{\geq n} \equiv \exists x_1 \dots \exists x_n (x_1 \neq x_2 \wedge x_1 \neq x_3 \wedge \dots \wedge x_{n-1} \neq x_n)$$

Consider

$$\Gamma = \{\neg \text{Inf}, \varphi^{\geq 1}, \varphi^{\geq 2}, \varphi^{\geq 3}, \dots\}$$

It is finitely satisfiable, since for any finite subset Γ_0 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 $\mathfrak{M} \models \neg \text{Inf}$, $|\mathfrak{M}|$ must be finite, say, of size k . Then $\mathfrak{M} \not\models \varphi^{\geq k+1}$. \square

27.5 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 an enumerable 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 enumerable. There is also an Upward Löwenheim-Skolem Theorem, which guarantees that if a set of sentences has a denumerable model it also has a non-enumerable model. Both theorems fail in second-order logic.

Theorem 27.8. *The Löwenheim-Skolem Theorem fails for second-order logic: There are sentences with infinite models but no enumerable models.*

27.5. THE LÖWENHEIM-SKOLEM THEOREM FAILS FOR SECOND-ORDER LOGIC

Proof. Recall that

$$\text{Count} \equiv \exists z \exists u \forall X ((X(z) \wedge \forall x (X(x) \rightarrow X(u(x)))) \rightarrow \forall x X(x))$$

is true in a structure \mathfrak{M} iff $|\mathfrak{M}|$ is enumerable. So $\text{Inf} \wedge \neg\text{Count}$ is true in \mathfrak{M} iff $|\mathfrak{M}|$ is both infinite and not enumerable. There are such structures—take any non-enumerable set as the domain, e.g., $\wp(\mathbb{N})$ or \mathbb{R} . So $\text{Inf} \wedge \text{Count}$ has infinite models but no enumerable models. \square

Theorem 27.9. *There are sentences with denumerable but not with non-enumerable models.*

Proof. $\text{Count} \wedge \text{Inf}$ is true in \mathbb{N} but not in any structure \mathfrak{M} with $|\mathfrak{M}|$ non-enumerable. \square

Problems

Problem 27.1. Prove [Proposition 27.3](#).

Problem 27.2. Give an example of a set Γ and a sentence φ so that $\Gamma \models \varphi$ but for every finite subset $\Gamma_0 \subseteq \Gamma$, $\Gamma_0 \not\models \varphi$.

Chapter 28

Second-order Logic and Set Theory

This section deals with coding powersets and the continuum in second-order logic. The results are stated but proofs have yet to be filled in. There are no problems yet—and the definitions and results themselves may have problems. Use with caution and report anything that’s false or unclear.

28.1 Introduction

Since second-order logic can quantify over subsets of the domain as well as functions, it is to be expected that some amount, at least, of set theory can be carried out in second-order logic. By “carry out,” we mean that it is possible to express set theoretic properties and statements in second-order logic, and is possible without any special, non-logical vocabulary for sets (e.g., the membership predicate symbol of set theory). For instance, we can define unions and intersections of sets and the subset relationship, but also compare the sizes of sets, and state results such as Cantor’s Theorem.

28.2 Comparing Sets

Proposition 28.1. *The formula $\forall x (X(x) \rightarrow Y(x))$ defines the subset relation, i.e., $\mathfrak{M}, s \models \forall x (X(x) \rightarrow Y(x))$ iff $s(X) \subseteq S(y)$.*

Proposition 28.2. *The formula $\forall x (X(x) \leftrightarrow Y(x))$ defines the identity relation on sets, i.e., $\mathfrak{M}, s \models \forall x (X(x) \leftrightarrow Y(x))$ iff $s(X) = S(y)$.*

Proposition 28.3. *The formula $\exists x X(x)$ defines the property of being non-empty, i.e., $\mathfrak{M}, s \models \exists x X(x)$ iff $s(X) \neq \emptyset$.*

28.3. CARDINALITIES OF SETS

A set X is no larger than a set Y , $X \preceq Y$, iff there is an injective function $f: X \rightarrow Y$. Since we can express that a function is injective, and also that its values for arguments in X are in Y , we can also define the relation of being no larger than on subsets of the domain.

Proposition 28.4. *The formula*

$$\exists u (\forall x (X(x) \rightarrow Y(u(x))) \wedge \forall x \forall y (u(x) = u(y) \rightarrow x = y))$$

defines the relation of being no larger than.

Two sets are the same size, or “equinumerous,” $X \approx Y$, iff there is a bijective function $f: X \rightarrow Y$.

Proposition 28.5. *The formula*

$$\begin{aligned} \exists u (\forall x (X(x) \rightarrow Y(u(x))) \wedge \\ \forall x \forall y (u(x) = u(y) \rightarrow x = y) \wedge \\ \forall y (Y(y) \rightarrow \exists x (X(x) \wedge y = u(x)))) \end{aligned}$$

defines the relation of being equinumerous with.

We will abbreviate these formulas, respectively, as $X \subseteq Y$, $X = Y$, $X \neq \emptyset$, $X \preceq Y$, and $X \approx Y$. (This may be slightly confusing, since we use the same notation when we speak informally about sets X and Y —but here the notation is an abbreviation for formulas in second-order logic involving one-place relation variables X and Y .)

Proposition 28.6. *The sentence $\forall X \forall Y ((X \preceq Y \wedge Y \preceq X) \rightarrow X \approx Y)$ is valid.*

Proof. The is satisfied in a structure \mathfrak{M} if, for any subsets $X \subseteq |\mathfrak{X}|$ and $Y \subseteq |\mathfrak{M}|$, if $X \preceq Y$ and $Y \preceq X$ then $X \approx Y$. But this holds for *any* sets X and Y —it is the Schröder-Bernstein Theorem. \square

28.3 Cardinalities of Sets

Just as we can express that the domain is finite or infinite, enumerable or non-enumerable, we can define the property of a subset of $|\mathfrak{M}|$ being finite or infinite, enumerable or non-enumerable.

Proposition 28.7. *The formula $\text{Inf}(X) \equiv$*

$$\begin{aligned} \exists u (\forall x \forall y (u(x) = u(y) \rightarrow x = y) \wedge \\ \exists y (X(y) \wedge \forall x (X(x) \rightarrow y \neq u(x)))) \end{aligned}$$

is satisfied with respect to a variable assignment s iff $s(X)$ is infinite.

Proposition 28.8. *The formula $\text{Count}(X) \equiv$*

$$\begin{aligned} \exists z \exists u (X(z) \wedge \forall x (X(x) \rightarrow X(u(x))) \wedge \\ \forall Y ((Y(z) \wedge \forall x (Y(x) \rightarrow Y(u(x)))) \rightarrow X = Y)) \end{aligned}$$

is satisfied with respect to a variable assignment s iff $s(X)$ is enumerable

We know from Cantor's Theorem that there are non-enumerable sets, and in fact, that there are infinitely many different levels of infinite sizes. Set theory develops an entire arithmetic of sizes of sets, and assigns infinite cardinal numbers to sets. The natural numbers serve as the cardinal numbers measuring the sizes of finite sets. The cardinality of denumerable sets is the first infinite cardinality, called \aleph_0 ("aleph-nought" or "aleph-zero"). The next infinite size is \aleph_1 . It is the smallest size a set can be without being countable (i.e., of size \aleph_0). We can define "X has size \aleph_0 " as $\text{Aleph}_0(X) \leftrightarrow \text{Inf}(X) \wedge \text{Count}(X)$. X has size \aleph_1 iff all its subsets are finite or have size \aleph_0 , but is not itself of size \aleph_0 . Hence we can express this by the formula $\text{Aleph}_1(X) \equiv \forall Y (Y \subseteq X \rightarrow (\neg \text{Inf}(Y) \vee \text{Aleph}_0(Y))) \wedge \neg \text{Aleph}_0(X)$. Being of size \aleph_2 is defined similarly, etc.

There is one size of special interest, the so-called cardinality of the continuum. It is the size of $\wp(\mathbb{N})$, or, equivalently, the size of \mathbb{R} . That a set is the size of the continuum can also be expressed in second-order logic, but requires a bit more work.

28.4 The Power of the Continuum

In second-order logic we can quantify over subsets of the domain, but not over sets of subsets of the domain. To do this directly, we would need *third-order* logic. For instance, if we wanted to state Cantor's Theorem that there is no injective function from the power set of a set to the set itself, we might try to formulate it as "for every set X, and every set P, if P is the power set of X, then not $P \preceq X$. And to say that P is the power set of X would require formalizing that the elements of P are all and only the subsets of X, so something like $\forall Y (P(Y) \leftrightarrow Y \subseteq X)$. The problem lies in $P(Y)$: that is not a formula of second-order logic, since only terms can be arguments to one-place relation variables like P.

We can, however, *simulate* quantification over sets of sets, if the domain is large enough. The idea is to make use of the fact that two-place relations R relates elements of the domain to elements of the domain. Given such an R, we can collect all the elements to which some x is R-related: $\{y \in |\mathfrak{M}| : R(x, y)\}$ is the set "coded by" x. Converseley, if $Z \subseteq \wp(|\mathfrak{M}|)$ is some collection of subsets of $|\mathfrak{M}|$, and there are at least as many elements of $|\mathfrak{M}|$ as there are sets in Z, then there is also a relation $R \subseteq |\mathfrak{M}|^2$ such that every $Y \in Z$ is coded by some x using R.

28.4. THE POWER OF THE CONTINUUM

Definition 28.9. If $R \subseteq |\mathfrak{M}|^2$, then x R -codes $\{y \in |\mathfrak{M}| : R(x, y)\}$. Y R -codes $\wp(X)$ iff for every $Z \subseteq X$, some $x \in Y$ R -codes Z , and every $x \in Y$ R -codes some $Z \subseteq X$.

Proposition 28.10. *The formula*

$$\text{Codes}(x, R, Y) \equiv \forall y (Y(y) \leftrightarrow R(x, y))$$

expresses that $s(x)$ $s(R)$ -codes $s(Y)$. The formula

$$\begin{aligned} \text{Pow}(Y, R, X) \equiv \\ \forall Z (Z \subseteq X \rightarrow \exists x (Y(x) \wedge \text{Codes}(x, R, Z))) \wedge \\ \forall x (Y(x) \rightarrow \forall Z (\text{Codes}(x, R, Z) \rightarrow Z \subseteq X)) \end{aligned}$$

expresses that $s(Y)$ $s(R)$ -codes the power set of $s(X)$.

With this trick, we can express statements about the power set by quantifying over the codes of subsets rather than the subsets themselves. For instance, Cantor's Theorem can now be expressed by saying that there is no injective function from the domain of any relation that codes the power set of X to X itself.

Proposition 28.11. *The sentence*

$$\begin{aligned} \forall X \forall R (\text{Pow}(R, X) \rightarrow \\ \neg \exists u (\forall x \forall y (u(x) = u(y) \rightarrow x = y) \wedge \\ \forall Y (\text{Codes}(x, R, Y) \rightarrow X(u(x))))) \end{aligned}$$

is valid.

The power set of a denumerable set is non-enumerable, and so its cardinality is larger than that of any denumerable set (which is \aleph_0). The size of $\wp(\mathbb{R})$ is called the "power of the continuum," since it is the same size as the points on the real number line, \mathbb{R} . If the domain is large enough to code the power set of a denumerable set, we can express that a set is the size of the continuum by saying that it is equinumerous with any set Y that codes the power set of set X of size \aleph_0 . (If the domain is not large enough, i.e., it contains no subset equinumerous with \mathbb{R} , then there can also be no relation that codes $\wp(X)$.)

Proposition 28.12. *If $\mathbb{R} \preceq |\mathfrak{M}|$, then the formula*

$$\text{Cont}(X) \equiv \forall X \forall Y \forall R ((\text{Aleph}_0(X) \wedge \text{Pow}(Y, R, X)) \rightarrow \neg Y \preceq X)$$

expresses that $s(X) \approx \mathbb{R}$.

Proposition 28.13. $|\mathfrak{M}| \approx \mathbb{R}$ iff

$$\begin{aligned} \mathfrak{M} \models \exists X \exists Y \exists R \big(\text{Aleph}_0(X) \wedge \text{Pow}(Y, R, X) \wedge \\ \exists u \big(\forall x \forall y \big(u(x) = u(y) \rightarrow x = y \big) \wedge \\ \forall y \big(Y(y) \rightarrow \exists x y = u(x) \big) \big) \big) \end{aligned}$$

The Continuum Hypothesis is the statement that the size of the continuum is the first non-enumerable cardinality, i.e, that $\wp(\mathbb{N})$ has size \aleph_1 .

Proposition 28.14. *The Continuum Hypothesis is true iff*

$$\text{CH} \equiv \forall X \big(\text{Aleph}_1(X) \leftrightarrow \text{Cont}(x) \big)$$

is valid.

Note that it isn't true that $\neg\text{CH}$ is valid iff the Continuum Hypothesis is false. In an enumerable domain, there are no subsets of size \aleph_1 and also no subsets of the size of the continuum, so CH is always true in an enumerable domain. However, we can give a different sentence that is valid iff the Continuum Hypothesis is false:

Proposition 28.15. *The Continuum Hypothesis is false iff*

$$\text{NCH} \equiv \forall X \big(\text{Cont}(X) \rightarrow \exists Y \big(Y \subseteq X \wedge \neg\text{Count}(X) \wedge \neg X \approx Y \big) \big)$$

is valid.

Part VIII

Methods

CHAPTER 28. SECOND-ORDER LOGIC AND SET THEORY

This part covers general and methodological material, especially explanations of various proof methods a non-mathematics student may be unfamiliar with. It currently contains a chapter on how to write proofs, and a chapter on induction, but additional sections for thos, exercises, and a chapter on mathematical terminology is also planned.

Chapter 29

Proofs

29.1 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 signifi-

cant, 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 $\psi \rightarrow \chi$ ” 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 $\psi \rightarrow \chi$,” you’re proving something about all formulas of a certain sort only (namely, conditionals), and it’s understood that $\psi \rightarrow \chi$ is an arbitrary conditional that your proof will talk about.

29.2 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 x , write “assume that x ,” or “suppose that x ”). 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.

29.3 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 impor-

29.3. USING DEFINITIONS

tant 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.

Later on we will prove that $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$. 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. (Later on, we'll also have to use the definitions of \cup and \cap , of course). Sets are defined to be identical whenever they have the same elements. So the definition we have to unpack is:

Definition 29.1. Sets X and Y are *identical*, $X = Y$, if every element of X is an element of Y , and vice versa.

This definition uses X and Y as placeholders for arbitrary sets. What it defines—the definiendum—is the expression " $X = Y$ " by giving the condition under which $X = Y$ is true. This condition—"every element of X is an element of Y , and vice versa"—is the definiens.¹

When you apply the definition, you have to match the X and Y in the definition to the case you're dealing with. So, say, if you're asked to show that $U = W$, the definition tells you that in order to do so, you have to show that every element of U is an element of W , and vice versa. In our case, it means that order for $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$, each $z \in X \cup (Y \cap Z)$ must also be in $(X \cup Y) \cap (X \cup Z)$, and vice versa. The expression $X \cup (Y \cap Z)$ plays the role of X in the definition, and $(X \cup Y) \cap (X \cup Z)$ that of Y . Since X is used both in the definition and in the statement of the theorem to be proved, 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 claim by showing that every element of X is an element of Y , and vice versa—that would show that $X = Y$, not that $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$.

Within the proof we are dealing with set-theoretic notions like union and intersection, and so we must also know the meanings of the symbols \cup and \cap in order to understand how the proof should proceed. And sometimes, unpacking the definition gives rise to further definitions to unpack. For instance, $X \cup Y$ is defined as $\{z : z \in X \text{ or } z \in Y\}$. So if you want to prove that $x \in X \cup Y$, unpacking the definition of \cup tells you that you have to prove $x \in \{z : z \in X \text{ or } z \in Y\}$. Now you also have to remember that

¹In this particular case—and very confusingly!—when $X = Y$, the sets X and Y 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.

$x \in \{z : \dots z \dots\}$ iff $\dots x \dots$. So, further unpacking the definition of the $\{z : \dots z \dots\}$ notation, what you have to show is: $x \in X$ or $x \in Y$.

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.

29.4 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 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 $U = V$ (a). Then we may apply the definition of $=$ for sets and infer: “Thus, by definition from (a), every element of U is an element of V 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 $U = V$, but we want to. If $U = V$ is the conclusion we aim for, then we can restate this aim also by applying the definition: to prove $U = V$ we have to prove that every element of U is an element of V and vice versa. So our proof will have the form: (a) prove that every element of U is an element of V ; (b) every element of V is an element of U ; (c) therefore, from (a) and (b) by definition of $=$, $U = V$. But we would usually not write it this way. Instead we might write something like,

We want to show $U = V$. By definition of $=$, this amounts to showing that every element of U is an element of V and vice versa.

(a) ... (a proof that every element of U is an element of V) ...

(b) ... (a proof that every element of V is an element of U) ...

29.4. INFERENCE PATTERNS

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 $U = V$ we've established that every element of U is an element of V and vice versa. From this we can conclude that every element of V is an element of U (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 $U = V$ you will find that, after unpacking the definition of $=$, you have to prove: every element of U is an element of V and every element of V is an element of U).

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 .

Recall that a biconditional (p 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 instance and 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 " and 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 $X \subseteq Y$, that means (unpacking the definition of \subseteq), that, for every $x \in X$, $x \in Y$. Thus, if you already know that $z \in X$, you can conclude $z \in Y$.

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 P s are Q s."

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* P s 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, X 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 than 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 .

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.

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 $x \in Y \cup Z$. $Y \cup Z$ is defined as $\{x : x \in Y \text{ or } x \in Z\}$. In other words, by definition, $x \in Y$ or $x \in Z$. We would prove that $x \in X$ from this by first assuming that $x \in Y$, and proving $x \in X$ from this assumption, and then assume $x \in Z$, and again prove $x \in X$ from this. You would write "We distinguish cases" under the assumption, then "Case (1): $x \in Y$ " underneath, and "Case (2): $x \in Z$ halfway

29.4. INFERENCE PATTERNS

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 29.2. *Suppose $Y \subseteq U$ and $Z \subseteq V$. Then $Y \cup Z \subseteq U \cup V$.*

Proof. Assume (a) that $Y \subseteq U$ and (b) $Z \subseteq V$. By definition, any $x \in Y$ is also $\in U$ (c) and any $x \in Z$ is also $\in V$ (d). To show that $Y \cup Z \subseteq U \cup V$, we have to show that if $x \in Y \cup Z$ then $x \in U \cup V$ (by definition of \subseteq). $x \in Y \cup Z$ iff $x \in Y$ or $x \in Z$ (by definition of \cup). Similarly, $x \in U \cup V$ iff $x \in U$ or $x \in V$. So, we have to show: for any x , if $x \in Y$ or $x \in Z$, then $x \in U$ or $x \in V$.

(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 $x \in Y$ or $x \in Z$ and show that $x \in U$ or $x \in V$.)

Suppose that $x \in Y$ or $x \in Z$. We have to show that $x \in U$ or $x \in V$. We distinguish cases.

Case 1: $x \in Y$. By (c), $x \in U$. Thus, $x \in U$ or $x \in V$. (Here we've made the inference discussed in the preceding subsection!)

Case 2: $x \in Z$. By (d), $x \in V$. Thus, $x \in U$ or $x \in V$. □

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 $\dots x \dots$ ", i.e., that some object that has the property described by " $\dots x \dots$ ". In this case you'll have to identify a suitable object show that it 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 \dots " (where \dots specifies which object you have in mind), possibly proving that \dots 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 29.3. *Suppose that $x \in Y$. Then there is an X such that $X \subseteq Y$ and $X \neq \emptyset$.*

Proof. Assume $x \in Y$. Let $X = \{x\}$. (Here we've defined the set X by enumerating its elements. Since we assume that x is an object, and we can always

for the set containing any number of objects by enumeration, we don't have to show that we've succeeded in defining a set X here. However, we still have to show that X has the properties required by the proposition. The proof isn't complete without that!) Since $x \in X$, $X \neq \emptyset$. (This relies on the definition of X as $\{x\}$ and the obvious facts that $x \in \{x\}$ and $x \notin \emptyset$.) Since x is the only element of $\{x\}$, and $x \in Y$, every element of X is also an element of Y . By definition of \subseteq , $X \subseteq Y$. \square

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 , $x \in X$ " or "there is an $x \in X$." If you want to use it in your proof, you can just pretend that you have a name for one of the things in your hypothesis says exist. Since X contains at least one thing, there are things to which that name might refer. You might of course not be able to pick one or describe it further (other than that $x \in X$). 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.) You might go from "for some x , $x \in X$ " to "Let $a \in X$." Now you reason about a , use some other hypotheses, etc., and come to a conclusion, p . If p no longer mentions a , p is independent of the assumption that $a \in X$, and you've shown that it follows just from the assumption "for some x , $x \in X$."

Proposition 29.4. *If $X \neq \emptyset$, then $X \cup Y \neq \emptyset$.*

Proof. Here the hypothesis that $X \neq \emptyset$ hides an existential claim, which you get to only by unpacking a few definitions. The definition of $=$ tells us that $X = \emptyset$ iff every $x \in X$ is also in \emptyset and every $x \in \emptyset$ is also $\in X$. Negating both sides, we get: $X \neq \emptyset$ iff either some $x \in X$ is $\notin \emptyset$ or some $x \in \emptyset$ is $\notin X$. Since nothing is $\in \emptyset$, the second disjunct can never be true, and " $x \in X$ and $x \notin \emptyset$ " reduces to just $x \in X$. So $X \neq \emptyset$ iff for some x , $x \in X$. That's an existence claim.

Suppose $X \neq \emptyset$, i.e., for some x , $x \in X$. Let $a \in X$.

Now we've introduced a name for one of the things $\in X$. We'll use it, only assuming that $a \in X$:

Since $a \in X$, $a \in X \cup Y$, by definition of \cup . So for some x , $x \in X \cup Y$, i.e., $X \cup Y \neq \emptyset$.

In that last step, we went from " $a \in X \cup Y$ " to "for some x , $x \in X \cup Y$." That didn't mention a anymore, so we know that "for some x , $x \in X \cup Y$ " follows from "for some x , $x \in X$ alone." \square

29.5. AN EXAMPLE

It's maybe good practice to keep bound variables like " x " separate from hypothetical names like a , like we did. In practice, however, we often don't and just use x , like so:

Suppose $X \neq \emptyset$, i.e., there is an $x \in X$. By definition of \cup , $x \in X \cup Y$. So $X \cup Y \neq \emptyset$.

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 $X \neq \emptyset$ and $Y \neq \emptyset$ then $X \cap Y \neq \emptyset$ " (which is not true).

Suppose $X \neq \emptyset$ and $Y \neq \emptyset$. So for some x , $x \in X$ and also for some x , $x \in Y$. Since $x \in X$ and $x \in Y$, $x \in X \cap Y$, by definition of \cap . So $X \cap Y \neq \emptyset$.

Can you spot where the incorrect step occurs and explain why the result does not hold?

29.5 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 29.5. *For any sets X , Y , and Z , $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$*

Let's prove it!

Proof. First we unpack the definition of " $=$ " in the statement of the proposition. Recall that proving equality between sets means showing that the sets have the same elements. That is, all elements of $X \cup (Y \cap Z)$ are also elements of $(X \cup Y) \cap (X \cup Z)$, and vice versa. The "vice versa" means that also every element of $(X \cup Y) \cap (X \cup Z)$ must be an element of $X \cup (Y \cap Z)$. So in unpacking the definition, we see that we have to prove a conjunction. Let's record this:

By definition, $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$ iff every element of $X \cup (Y \cap Z)$ is also an element of $(X \cup Y) \cap (X \cup Z)$, and every element of $(X \cup Y) \cap (X \cup Z)$ is an element of $X \cup (Y \cap Z)$.

Since this is a conjunction, we must prove each conjunct separately. Let's start with the first: let's prove that every element of $X \cup (Y \cap Z)$ is also an element of $(X \cup Y) \cap (X \cup Z)$.

This is a universal claim, and so we consider an arbitrary element of $X \cup (Y \cap Z)$ and show that it must also be an element of $(X \cup Y) \cap (X \cup Z)$. We'll pick a variable to call this arbitrary element by, say, z . Our proof continues:

First, we prove that every element of $X \cup (Y \cap Z)$ is also an element of $(X \cup Y) \cap (X \cup Z)$. Let $z \in X \cup (Y \cap Z)$. We have to show that $z \in (X \cup Y) \cap (X \cup Z)$.

Now it is time to unpack the definition of \cup and \cap . For instance, the definition of \cup is: $X \cup Y = \{z : z \in X \text{ or } z \in Y\}$. When we apply the definition to " $X \cup (Y \cap Z)$," the role of the " Y " in the definition is now played by " $Y \cap Z$," so $X \cup (Y \cap Z) = \{z : z \in X \text{ or } z \in Y \cap Z\}$. So our assumption that $z \in X \cup (Y \cap Z)$ amounts to: $z \in \{z : z \in X \text{ or } z \in Y \cap Z\}$. And $z \in \{z : \dots z \dots\}$ iff $\dots z \dots$, i.e., in this case, $z \in X$ or $z \in Y \cap Z$.

By the definition of \cup , either $z \in X$ or $z \in Y \cap Z$.

Since this is a disjunction, it will be useful to apply proof by cases. So we take the two cases, and show that in each one, the conclusion we're aiming for (namely, " $z \in (X \cup Y) \cap (X \cup Z)$ ") obtains.

Case 1: Suppose that $z \in X$.

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 $z \in (X \cup Y) \cap (X \cup Z)$. Based on the definition of \cap , if we want to show that $z \in (X \cup Y) \cap (X \cup Z)$, we have to show that it's in both $(X \cup Y)$ and $(X \cup Z)$. The answer is immediate. But $z \in X \cup Y$ iff $z \in X$ or $z \in Y$, and we already have (as the assumption of case 1) that $z \in X$. By the same reasoning—switching Z for Y — $z \in X \cup Z$. This argument went in the reverse direction, so let's record our reasoning in the direction needed in our proof.

Since $z \in X$, $z \in X$ or $z \in Y$, and hence, by definition of \cup , $z \in X \cup Y$. Similarly, $z \in X \cup Z$. But this means that $z \in (X \cup Y) \cap (X \cup Z)$, 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 $z \in Y \cap Z$.

Case 2: Suppose that $z \in Y \cap Z$.

Again, we are working with the intersection of two sets. Since $z \in Y \cap Z$, z must be an element of both Y and Z .

Since $z \in Y \cap Z$, z must be an element of both Y and Z , by definition of \cap .

It's time to look at our conclusion again. We have to show that z is in both $(X \cup Y)$ and $(X \cup Z)$. And again, the solution is immediate.

29.5. AN EXAMPLE

Since $z \in Y$, $z \in (X \cup Y)$. Since $z \in Z$, also $z \in (X \cup Z)$. So, $z \in (X \cup Y) \cap (X \cup Z)$.

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 $z \in X \cup (Y \cap Z)$ then $z \in (X \cup Y) \cap (X \cup Z)$.

Now we just want to show the other direction, that every element of $(X \cup Y) \cap (X \cup Z)$ is an element of $X \cup (Y \cap Z)$. 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 $z \in (X \cup Y) \cap (X \cup Z)$. We want to show that $z \in X \cup (Y \cap Z)$.

We are now working from the hypothesis that $z \in (X \cup Y) \cap (X \cup Z)$. 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 $X \cup Y$ and $X \cup Z$, by definition of \cap . And by the definition of \cup , we can further unpack this to: either $z \in X$ or $z \in Y$, and also either $z \in X$ or $z \in Z$. 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 X , Y or Z . But that would be a mistake. We have to be careful, so let's consider each disjunction in turn.

By definition of \cap , $z \in X \cup Y$ and $z \in X \cup Z$. By definition of \cup , $z \in X$ or $z \in Y$. We distinguish cases.

Since we're focusing on the first disjunction, we haven't unpacked the second one yet. In fact, we don't need it. The first case is $z \in X$, and an element of a set is also an element of that union of that set with any other. So case 1 is easy:

Case 1: Suppose that $z \in X$. It follows that $z \in X \cup (Y \cap Z)$.

Now for the second case, $z \in Y$. Here we'll unpack the second \cup and do another proof-by-cases:

Case 2: Suppose that $z \in Y$. Since $z \in X \cup Z$, either $z \in X$ or $z \in Z$. We distinguish cases further:

Case 2a: $z \in X$. Then, again, $z \in X \cup (Y \cap Z)$.

Ok, this was a bit weird. We didn't actually need the assumption that $z \in Y$ for this case, but that's ok.

Case 2b: $z \in Z$. Then $z \in Y$ and $z \in Z$, so $z \in Y \cap Z$, and consequently, $z \in X \cup (Y \cap Z)$.

This concludes both proof-by-cases and so we're done with the second half. Since we've proved both directions, the proof is complete.

So, if $z \in (X \cup Y) \cap (X \cup Z)$ then $z \in X \cup (Y \cap Z)$.

Together, we've showed that $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$.

□

29.6 Another Example

Proposition 29.6. *If $X \subseteq Z$, then $X \cup (Z \setminus X) = Z$.*

Proof. We begin by observing that this is a conditional statement. It is tacitly universally quantified: the proposition holds for all sets X and Z . So X and Z are variables for arbitrary sets. To prove such a statement, we assume the antecedent and prove the consequent.

Suppose that $X \subseteq Z$. We want to show that $X \cup (Z \setminus X) = Z$.

What do we know? We know that $X \subseteq Z$. Let's unpack the definition of \subseteq : the assumption means that all elements of X are also elements of Z . Let's write this down—it's an important fact that we'll use throughout the proof.

By the definition of \subseteq , since $X \subseteq Z$, for all z , if $z \in X$, then $z \in Z$.

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 $X \cup (Z \setminus X) = Z$, and so we set up a proof similarly to the last example: we show that every element of $X \cup (Z \setminus X)$ is also an element of Z and, conversely, every element of Z is an element of $X \cup (Z \setminus X)$. We can shorten this to: $X \cup (Z \setminus X) \subseteq Z$ and $Z \subseteq X \cup (Z \setminus X)$. (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 $X \cup (Z \setminus X)$ is also an element of Z , we assume that for an arbitrary z that $z \in X \cup (Z \setminus X)$ and show that $z \in Z$. By the definition of \cup , we can conclude that $z \in X$ or $z \in Z \setminus X$ from $z \in X \cup (Z \setminus X)$. You should now be getting the hang of this.

29.7. INDIRECT PROOF

$X \cup (Z \setminus X) = Z$ iff $X \cup (Z \setminus X) \subseteq Z$ and $Z \subseteq (X \cup (Z \setminus X))$. First we prove that $X \cup (Z \setminus X) \subseteq Z$. Let $z \in X \cup (Z \setminus X)$. So, either $z \in X$ or $z \in (Z \setminus X)$.

We've arrived at a disjunction, and from it we want to prove that $z \in Z$. We do this using proof by cases.

Case 1: $z \in X$. Since for all z , if $z \in X$, $z \in Z$, we have that $z \in Z$.

Here we've used the fact recorded earlier which followed from the hypothesis of the proposition that $X \subseteq Z$. The first case is complete, and we turn to the second case, $z \in (Z \setminus X)$. Recall that $Z \setminus X$ denotes the *difference* of the two sets, i.e., the set of all elements of Z which are not elements of X . Let's use state what the definition gives us. But an element of Z not in X is in particular an element of Z .

Case 2: $z \in (Z \setminus X)$. This means that $z \in Z$ and $z \notin X$. So, in particular, $z \in Z$.

Great, we've solved the first direction. Now for the second direction. Here we prove that $Z \subseteq X \cup (Z \setminus X)$. So we assume that $z \in Z$ and prove that $z \in X \cup (Z \setminus X)$.

Now let $z \in Z$. We want to show that $z \in X$ or $z \in Z \setminus X$.

Since all elements of X are also elements of Z , and $Z \setminus X$ is the set of all things that are elements of Z but not X , it follows that z is either in X or $Z \setminus X$. But 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: $z \in X$ or $z \notin X$. 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 $z \in X$.) Since this is a disjunction, we can again use proof-by-cases.

Either $z \in X$ or $z \notin X$. In the former case, $z \in X \cup (Z \setminus X)$. In the latter case, $z \in Z$ and $z \notin X$, so $z \in Z \setminus X$. But then $z \in X \cup (Z \setminus X)$.

Our proof is complete: we have shown that $X \cup (Z \setminus X) = Z$. □

29.7 Indirect Proof

In the first instance, indirect proof 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 $\neg p$. A promising strategy—and in many cases the only 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 $\neg p$ indirectly, 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. Therefore, indirect proofs are also often called “proofs by contradiction.” 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 29.7. *If $X \subseteq Y$ and $Y = \emptyset$, then $X = \emptyset$.*

Proof. Since this is a conditional claim, we assume the antecedent and want to prove the consequent:

Suppose $X \subseteq Y$ and $Y = \emptyset$. We want to show that $X = \emptyset$.

Now let’s consider the definition of \emptyset and $=$ for sets. $X = \emptyset$ if every element of X is also an element of \emptyset and (vice versa). And \emptyset is defined as the set with no elements. So $X = \emptyset$ iff X has no elements, i.e., it’s not the case that there is an $x \in X$.

$X = \emptyset$ iff there is no $x \in X$.

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 $x \in X$. To use indirect proof, we have to assume the corresponding positive claim p , i.e., there is an $x \in X$. We indicate that we’re doing an indirect proof by writing “We proceed indirectly:” or, “By way of contradiction,” or even just “Suppose not.” We then state the assumption of p .

We proceed indirectly. Suppose there is an $x \in X$.

This is now the new assumption we’ll use to obtain a contradiction. We have two more assumptions: that $X \subseteq Y$ and that $Y = \emptyset$. The first gives us that $x \in Y$:

Since $X \subseteq Y$, $x \in Y$.

But now by unpacking the definition of $Y = \emptyset$ as before, we see that this conclusion conflicts with the second assumption. Since $x \in Y$ but $x \notin \emptyset$, we have $Y \neq \emptyset$.

Since $x \in Y$ but $x \notin \emptyset$, $Y \neq \emptyset$. This contradicts the assumption that $Y = \emptyset$.

29.7. INDIRECT PROOF

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 ($X \subseteq Y$ and $Y = \emptyset$) are not contested, it must be the last assumption introduced (there is an $x \in X$) that must be false. But if we want to be thorough, we can spell this out.

Thus, our assumption that there is an $x \in X$ must be false, hence,
 $X = \emptyset$ by indirect proof. \square

Every positive claim is trivially equivalent to a negative claim: p iff $\neg\neg p$. So indirect proofs can also be used to establish positive claims: 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 indirect proof, and hence p . Crucially, it is sometimes easier to work with $\neg p$ as an assumption than it is to prove p directly. And even when a direct proof is just as simple (as in the next example), some people prefer to proceed indirectly. If the double negation confuses you, think of an indirect proof of some claim as a proof of a contradiction from the *opposite* claim. So, an indirect proof of $\neg p$ is a proof of a contradiction from the assumption p ; and indirect proof of p is a proof of a contradiction from $\neg p$.

Proposition 29.8. $X \subseteq X \cup Y$.

Proof. On the face of it, this is a positive claim: every $x \in X$ is also in $x \cup Y$. The opposite of that is: some $x \in X$ is $\notin X \cup Y$. So we can prove it indirectly by assuming this opposite claim, and showing that it leads to a contradiction.

Suppose not, i.e., $X \not\subseteq X \cup Y$.

We have a definition of $X \subseteq X \cup Y$: every $x \in X$ is also $\in X \cup Y$. To understand what $X \not\subseteq X \cup Y$ means, we have to use some elementary logical manipulation on the unpacked definition: it's false that every $x \in X$ is also $\in X \cup Y$ iff there is *some* $x \in X$ that is $\notin X \cup Y$. (This is a place where you want to be very careful: many students' attempted indirect proofs fail because they analyze the negation of a claim like "all As are Bs" incorrectly.) In other words, $X \not\subseteq X \cup Y$ iff there is an x such that $x \in X$ and $x \notin X \cup Y$. From then on, it's easy.

So, there is an $x \in X$ such that $x \notin X \cup Y$. By definition of \cup ,
 $x \in X \cup Y$ iff $x \in X$ or $x \in Y$. Since $x \in X$, we have $x \in X \cup Y$.
This contradicts the assumption that $x \notin X \cup Y$. \square

Proposition 29.9. If $X \subseteq Y$ and $Y \subseteq Z$ then $X \subseteq Z$.

Proof. First, set up the required conditional proof:

Suppose $X \subseteq Y$ and $Y \subseteq Z$. We want to show $X \subseteq Z$.

Let's proceed indirectly.

Suppose not, i.e., $X \not\subseteq Z$.

As before, we reason that $X \not\subseteq Z$ iff not every $x \in X$ is also $\in Z$, i.e., some $x \in X$ is $\notin Z$. 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 $x \in X$ and $x \notin Z$.

Now we can use the assumption that (some) $x \in X$ and $x \notin Z$ to get to our contradiction. Of course, we'll have to use the other two assumptions to do it.

Since $X \subseteq Y$, $x \in Y$. Since $Y \subseteq Z$, $x \in Z$. But this contradicts $x \notin Z$. \square

Proposition 29.10. *If $X \cup Y = X \cap Y$ then $X = Y$.*

Proof. The beginning is now routine:

Suppose $X \cup Y = X \cap Y$. Assume, by way of contradiction, that $X \neq Y$.

Our assumption for the indirect proof is that $X \neq Y$. Since $X = Y$ iff $X \subseteq Y$ and $Y \subseteq X$, we get that $X \neq Y$ iff $X \not\subseteq Y$ or $Y \not\subseteq X$. (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.

$X \neq Y$ iff $X \not\subseteq Y$ or $Y \not\subseteq X$. We distinguish cases.

In the first case, we assume $X \not\subseteq Y$, i.e., for some x , $x \in X$ but $x \notin Y$. $X \cap Y$ is defined as those elements that X and Y have in common, so if something isn't in one of them it's not in the intersection. $X \cup Y$ is X together with Y , so anything in either is also in the union. This tells us that $x \in X \cup Y$ but $x \notin X \cap Y$, and hence that $X \cap Y \neq X \cup Y$.

Case 1: $X \not\subseteq Y$. Then for some x , $x \in X$ but $x \notin Y$. Since $x \notin Y$, then $x \notin X \cap Y$. Since $x \in X$, $x \in X \cup Y$. So, $X \cap Y \neq X \cup Y$, contradicting the assumption that $X \cap Y = X \cup Y$.

Case 2: $Y \not\subseteq X$. Then for some y , $y \in Y$ but $y \notin X$. As before, we have $y \in X \cup Y$ but $y \notin X \cap Y$, and so $X \cap Y \neq X \cup Y$, again contradicting $X \cap Y = X \cup Y$. \square

29.8 Reading Proofs

Proofs you find in textbooks and articles very seldom give all the details we have so far included in our examples. Authors often 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 29.11 (Absorption). *For all sets X, Y ,*

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

Proof. If $z \in X \cap (X \cup Y)$, then $z \in X$, so $X \cap (X \cup Y) \subseteq X$. Now suppose $z \in X$. Then also $z \in X \cup Y$, and therefore also $z \in X \cap (X \cup Y)$. \square

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 X and Y , and when the proof mentions X or Y , these are variables for arbitrary sets. The general claim 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 $z \in X \cap (X \cup Y)$, then $z \in X$, so $X \cap (X \cup Y) \subseteq X$."

This is the first half of the proof of the identity: it establishes that if an arbitrary z is an element of the left side, it is also an element of the right, i.e., $X \cap (X \cup Y) \subseteq X$. Assume that $z \in X \cap (X \cup Y)$. Since z is an element of the intersection of two sets iff it is an element of both sets, we can conclude that $z \in X$ and also $z \in X \cup Y$. In particular, $z \in X$, which is what we wanted to show. 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 $X \subseteq X \cap (X \cup Y)$.

"Now suppose $z \in X$. Then also $z \in X \cup Y$, and therefore also $z \in X \cap (X \cup Y)$."

We start by assuming that $z \in X$, since we are showing that, for any z , if $z \in X$ then $z \in X \cap (X \cup Y)$. To show that $z \in X \cap (X \cup Y)$, we have to show (by definition of " \cap ") that (i) $z \in X$ and also (ii) $z \in X \cup Y$. Here (i) is just our assumption, so there is nothing further to prove, 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 $z \in X$, and $X \cup Y$ is the union of X and Y , this is the case here. So $z \in X \cup Y$. We've shown both (i)

$z \in X$ and (ii) $z \in X \cup Y$, hence, by definition of “ \cap ,” $z \in X \cap (X \cup Y)$. 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 $z \in X$ that $z \in X \cup Y$, and from $z \in X$ and $z \in X \cup Y$ that $z \in X \cap (X \cup Y)$.

Here’s another version of the proof above, with everything made explicit:

Proof. [By definition of $=$ for sets, $X \cap (X \cup Y) = X$ we have to show (a) $X \cap (X \cup Y) \subseteq X$ and (b) $X \cap (X \cup Y) \supseteq X$. (a): By definition of \subseteq , we have to show that if $z \in X \cap (X \cup Y)$, then $z \in X$.] If $z \in X \cap (X \cup Y)$, then $z \in X$ [using a conjunction, since by definition of \cap , $z \in X \cap (X \cup Y)$ iff $z \in X$ and $z \in X \cup Y$], so $X \cap (X \cup Y) \subseteq X$. [(b): By definition of \subseteq , we have to show that if $z \in X$, then $z \in X \cap (X \cup Y)$.] Now suppose [(1)] $z \in X$. Then also [(2)] $z \in X \cup Y$ [since by (1) $z \in X$ or $z \in Y$, which by definition of \cup means $z \in X \cup Y$], and therefore also $z \in X \cap (X \cup Y)$ [since the definition of \cap requires that $z \in X$, i.e., (1), and $z \in X \cup Y$, i.e., (2)]. \square

29.9 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 they 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

29.10. OTHER RESOURCES

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.

29.10 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* by Daniel Solow and *How to Prove It: A Structured Approach* by Daniel Velleman in particular. The *Book of Proof* by Richard Hammack and *Mathematical Reasoning* by Ted Sundstrom are books on proof that are freely available. Philosophers might find *More Precisely: The Math you need to do Philosophy* by Eric Steinhart 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" by Michael Hutchings and "How to write proofs" by Eugenia Chang.

Motivational Videos

Feel like you have no motivation to do your homework? Feeling down? These videos might help!

- https://www.youtube.com/watch?v=ZXsQAXx_ao0
- <https://www.youtube.com/watch?v=BQ4yd2W50No>
- <https://www.youtube.com/watch?v=StTqXEQ2l-Y>

Problems

Problem 29.1. Suppose you are asked to prove that $X \cap Y \neq \emptyset$. Unpack all the definitions occurring here, i.e., restate this in a way that does not mention “ \cap ”, “ $=$ ”, or “ \emptyset ”.

Problem 29.2. Prove *indirectly* that $X \cap Y \subseteq X$.

Problem 29.3. Expand the following proof of $X \cup (X \cap Y) = X$, 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 $z \in X \cup (X \cap Y)$ then $z \in X$ or $z \in X \cap Y$. If $z \in X \cap Y$, $z \in X$. Any $z \in X$ is also $\in X \cup (X \cap Y)$. □

Chapter 30

Induction

30.1 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 particular 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 than 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) whenever a number n has the property, so does $n + 1$. Induction on natural numbers can then also often be used to prove general facts 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 such 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.

30.2 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 a number has it, the next number has it too. If we abbreviate “number n has property P ” by $P(n)$, 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 n , if $P(n)$ then $P(n + 1)$.

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 n , if $P(n)$ then $P(n + 1)$ ” by putting 0 for n . 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 k , after doing this k steps, we eventually arrive at $P(k)$. So (1) and (2) together established $P(k)$ for any $k \in \mathbb{N}$.

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 30.1. *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 $P(k + 1)$.

30.2. INDUCTION ON \mathbb{N}

(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. \square

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 30.2. $s_n = n(n+1)/2$.

Proof. We have to prove (1) that $s_0 = 0 \cdot (0+1)/2$ and (2) if $s_n = n(n+1)/2$ then $s_{n+1} = (n+1)(n+2)/2$. (1) is obvious. To prove (2), we assume the inductive hypothesis: $s_n = n(n+1)/2$. Using it, we have to show that $s_{n+1} = (n+1)(n+2)/2$.

What is s_{n+1} ? By the definition, $s_{n+1} = s_n + (n+1)$. By inductive hypothesis, $s_n = n(n+1)/2$. We can substitute this into the previous equation, and

then just need a bit of arithmetic of fractions:

$$\begin{aligned}
 s_{n+1} &= \frac{n(n+1)}{2} + (n+1) = \\
 &= \frac{n(n+1)}{2} + \frac{2(n+1)}{2} = \\
 &= \frac{n(n+1) + 2(n+1)}{2} = \\
 &= \frac{(n+2)(n+1)}{2}.
 \end{aligned}$$

□

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 $n+1$ to the case for n .

30.3 Strong Induction

In the principle of induction discussed above, we prove $P(0)$ and also if $P(n)$, then $P(n+1)$. In the second part, we assume that $P(n)$ is true and use this assumption to prove $P(n+1)$. Equivalently, of course, we could assume $P(n-1)$ and use it to prove $P(n)$ —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 $n-1$ of n , but for all numbers smaller than n , and use this assumption to establish the claim for n . This also gives us the claim $P(k)$ for all $k \in \mathbb{N}$. 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 $l < n$ then $P(n)$, we know this in particular for $n=1$. So we can conclude $P(2)$. With this we have proved $P(0), P(1), P(2)$, i.e., $P(l)$ for all $l < 3$, and since we have also the conditional, if $P(l)$ for all $l < 3$, then $P(3)$, we can conclude $P(3)$, and so on.

In fact, if we can establish the general conditional "for all n , if $P(l)$ for all $l < n$, then $P(n)$," we do not have to establish $P(0)$ anymore, since it follows from it. For remember that a general claim like "for all $l < n$, $P(l)$ " is true if there are no $l < n$. This is a case of vacuous quantification: "all As are Bs " is true if there are no As , $\forall x (\varphi(x) \rightarrow \psi(x))$ is true if no x satisfies $\varphi(x)$. In this case, the formalized version would be " $\forall l (l < n \rightarrow P(l))$ "—and that is true if there are no $l < n$. And if $n=0$ that's exactly the case: no $l < 0$, hence "for all $l < 0$, $P(l)$ " is true, whatever P is. A proof of "if $P(l)$ for all $l < n$, then $P(n)$ " thus automatically establishes $P(0)$.

30.4. INDUCTIVE DEFINITIONS

This variant is useful if establishing the claim for n can't be made to just rely on the claim for $n - 1$ but may require the assumption that it is true for one or more $l < n$.

30.4 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 simpler example, consider strings of parentheses, such as $"()("$ or $"()(())"$. In the second string, the parentheses "balance," in the first one, they don't. The shortest such expression is $"()"$. Actually, the very shortest string of parentheses in which every opening parenthesis has a matching closing parenthesis is $"",$ i.e., the empty sequence \emptyset . If we already have a parenthesis expression p , then putting matching parentheses around it makes another balanced parenthesis expression. And if p and p' are two balanced parentheses expressions, writing one after the other, $"pp'"$ is also a balanced parenthesis expression. In fact, any sequence of balanced parentheses can be generated in this way, and we might use these operations to *define* the set of such expressions. This is an *inductive definition*.

Definition 30.3 (Paraexpressions). The set of *parexpressions* is inductively defined as follows:

1. \emptyset is a parexpression.
2. If p is a parexpression, then so is (p) .
3. If p and p' are parexpressions $\neq \emptyset$, then so is pp' .
4. Nothing else is a parexpression.

(Note that we have not yet proved that every balanced parenthesis expression is a parexpression, although it is quite clear that every parexpression is a balanced parenthesis expression.)

The key feature of inductive definitions is that if you want to prove something about all parexpressions, the definition tells you which cases you must consider. For instance, if you are told that q is a parexpression, the inductive definition tells you what q can look like: q can be \emptyset , it can be (p) for some other parexpression p , or it can be pp' for two parexpressions p and $p' \neq \emptyset$. Because of clause (4), those are all the 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 parexpression 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 parexpression of length n is $n/2$.

Proposition 30.4. *For any n , the number of $($ in a parexpression 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 $k < n$, any parexpression of length k has $k/2$ $($'s, then any parexpression of length n has $n/2$ $($'s.

To show this conditional, assume that its antecedent is true, i.e., assume that for any $k < n$, parexpressions of length k contain $k/2$ $($'s. We call this assumption the inductive hypothesis. We want to show the same is true for parexpressions of length n .

So suppose q is a parexpression of length n . Because parexpressions are inductively defined, we have three cases: (1) q is \emptyset , (2) q is (p) for some parexpression p , or (3) q is pp' for some parexpressions p and $p' \neq \emptyset$.

1. q is \emptyset . Then $n = 0$, and the number of $($ in q is also 0. Since $0 = 0/2$, the claim holds.
2. q is (p) for some parexpression p . Since q contains two more symbols than p , $\text{len}(p) = n - 2$, in particular, $\text{len}(p) < n$, so the inductive hypothesis applies: the number of $($ in p is $\text{len}(p)/2$. The number of $($ in q is $1 +$ the number of $($ in p , so $= 1 + \text{len}(p)/2$, and since $\text{len}(p) = n - 2$, this gives $1 + (n - 2)/2 = n/2$.
3. q is pp' for some parexpression p and $p' \neq \emptyset$. Since neither p nor $p' = \emptyset$, both $\text{len}(p)$ and $\text{len}(p') < n$. Thus the inductive hypothesis applies in each case: The number of $($ in p is $\text{len}(p)/2$, and the number of $($ in p' is $\text{len}(p')/2$. On the other hand, the number of $($ in q is obviously the sum of the numbers of $($ in p and p' , since $q = pp'$. Hence, the number of $($ in q is $\text{len}(p)/2 + \text{len}(p')/2 = (\text{len}(p) + \text{len}(p'))/2 = \text{len}(pp')/2 = n/2$.

In each case, we've shown that the number of $($ in q is $n/2$ (on the basis of the inductive hypothesis). By strong induction, the proposition follows. \square

30.5 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 is 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

30.5. STRUCTURAL INDUCTION

from old ones. In the case of parexpressions, for instance, the initial object is \emptyset and the operations are

$$\begin{aligned} o_1(p) &= (p) \\ o_2(q, q') &= qq' \end{aligned}$$

You can even think of the natural numbers \mathbb{N} themselves as being given by 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 parexpressions, we would prove that it is true about \emptyset , that it is true of (p) provided it is true of p , and that it is true about qq' provided it is true of q and q' individually.

Proposition 30.5. *The number of (equals the number of) in any parexpression p .*

Proof. We use structural induction. Parexpressions are inductively defined, with initial object \emptyset and the operations o_1 and o_2 .

1. The claim is true for \emptyset , since the number of (in $\emptyset = 0$ and the number of) in \emptyset also = 0.
2. Suppose the number of (in p equals the number of) in p . We have to show that this is also true for (p) , i.e., $o_1(p)$. But the number of (in (p) is $1 +$ the number of (in p . And the number of) in (p) is $1 +$ the number of) in p , so the claim also holds for (p) .
3. Suppose the number of (in q equals the number of), and the same is true for q' . The number of (in $o_2(p, p')$, i.e., in pp' , is the sum of the number of (in p and p' . The number of) in $o_2(p, p')$, i.e., in pp' , is the sum of the number of) in p and p' . The number of (in $o_2(p, p')$ equals the number of) in $o_2(p, p')$.

The result follows by structural induction. □

Part IX

History

Chapter 31

Biographies

31.1 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 1884, 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.



Figure 31.1: Georg Cantor

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 \(2012\)](#).

31.2 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. 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.



Figure 31.2: Alonzo Church

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 \(forthcoming\)](#). Church's original writings on the lambda calculus and the Entscheidungsproblem (Church's Thesis) are [Church \(1936a,b\)](#). [Aspray \(1984\)](#) records

31.3. GERHARD GENTZEN

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](#)).

31.3 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 homeschooled 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 wrote poems for his mother and plays for the school theatre.

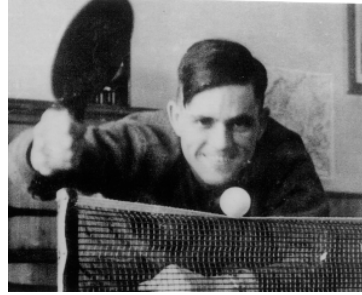


Figure 31.3: Gerhard Gentzen

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.

31.4 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 Kurt 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.



Figure 31.4: Kurt Gödel

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

31.5. EMMY NOETHER

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](#)).

31.5 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 lec-

ture 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 decreasing 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*.



Figure 31.5: Emmy Noether

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).

31.6. RÓZSA PÉTER

31.6 Rózsa Péter

Rózsa Péter was born Rózsa 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



Figure 31.6: Rózsa Péter

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).

31.7 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. At a young age Robinson recalls being intrigued by numbers (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.

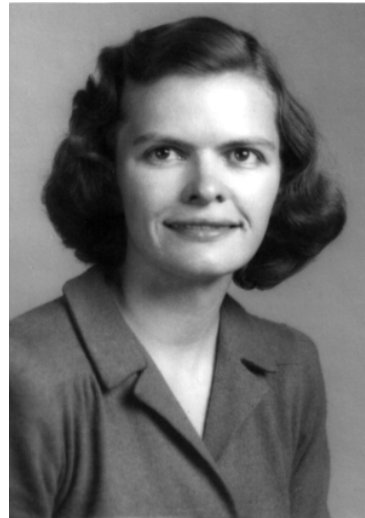


Figure 31.7: Julia Robinson

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 highly influenced by mathematician Raphael Robinson. They quickly 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

31.7. JULIA ROBINSON

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 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 for the next decade. 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).

31.8 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.

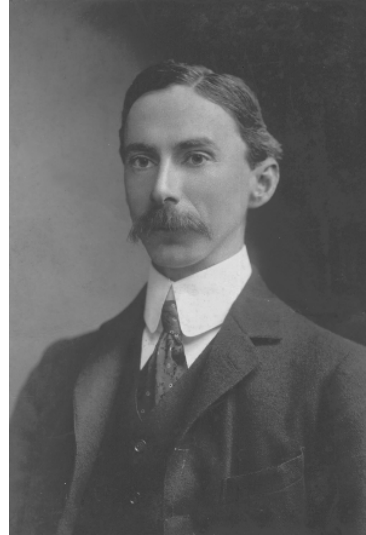


Figure 31.8: Bertrand Russell

31.9. ALFRED TARSKI

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.).

31.9 Alfred Tarski

Alfred Tarski was born on January 14, 1901 in Warsaw, Poland (then part of the Russian Empire). Often 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,



Figure 31.9: Alfred Tarski

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).

31.10 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.



Figure 31.10: Alan Turing

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 in-

31.11. ERNST ZERMELO

roduction 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 Benedict 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 \(1937\)](#).

31.11 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 Göttingen, 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 honorary 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.



Figure 31.11: Ernst Zermelo

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

Georg Cantor, p. 403: Portrait of Georg Cantor by Otto Zeth courtesy of the Universitätsarchiv, Martin-Luther Universität Halle–Wittenberg. UAHW Rep. 40-VI, Nr. 3 Bild 102.

Alonzo Church, p. 404: Portrait of Alonzo Church, undated, photographer unknown. Alonzo Church Papers; 1924–1995, (C0948) Box 60, Folder 3. Manuscripts Division, Department of Rare Books and Special Collections, Princeton University Library. © Princeton University. The Open Logic Project has obtained permission to use this image for inclusion in non-commercial OLP-derived materials. Permission from Princeton University is required for any other use.

Gerhard Gentzen, p. 405: Portrait of Gerhard Gentzen playing ping-pong courtesy of Ekhart Mentzler-Trott.

Kurt Gödel, p. 406: Portrait of Kurt Gödel, ca. 1925, photographer unknown. From the Shelby White and Leon Levy Archives Center, Institute for Advanced Study, Princeton, NJ, USA, on deposit at Princeton University Library, Manuscript Division, Department of Rare Books and Special Collections, Kurt Gödel Papers, (C0282), Box 14b, #110000. The Open Logic Project has obtained permission from the Institute's Archives Center to use this image for inclusion in non-commercial OLP-derived materials. Permission from the Archives Center is required for any other use.

Emmy Noether, p. 408: Portrait of Emmy Noether, ca. 1922, courtesy of the Abteilung für Handschriften und Seltene Drucke, Niedersächsische Staats- und Universitätsbibliothek Göttingen, Cod. Ms. D. Hilbert 754, Bl. 14 Nr. 73. Restored from an original scan by Joel Fuller.

Rózsa Péter, p. 409: Portrait of Rózsa Péter, undated, photographer unknown. Courtesy of Béla Andrásfai.

Julia Robinson, p. 410: Portrait of Julia Robinson, unknown photographer, courtesy of Neil D. Reid. The Open Logic Project has obtained permission to use this image for inclusion in non-commercial OLP-derived materials. Permission is required for any other use.

Bertrand Russell, p. 412: Portrait of Bertrand Russell, ca. 1907, courtesy of the William Ready Division of Archives and Research Collections, McMaster University Library. Bertrand Russell Archives, Box 2, f. 4.

Photo Credits

Alfred Tarski, p. 413: Passport photo of Alfred Tarski, 1939. Cropped and restored from a scan of Tarski's passport by Joel Fuller. Original courtesy of Bancroft Library, University of California, Berkeley. Alfred Tarski Papers, Banc MSS 84/49. The Open Logic Project has obtained permission to use this image for inclusion in non-commercial OLP-derived materials. Permission from Bancroft Library is required for any other use.

Alan Turing, p. 414: Portrait of Alan Mathison Turing by Elliott & Fry, 29 March 1951, NPG x82217, © National Portrait Gallery, London. Used under a Creative Commons BY-NC-ND 3.0 license.

Ernst Zermelo, p. 416: Portrait of Ernst Zermelo, ca. 1922, courtesy of the Abteilung für Handschriften und Seltene Drucke, Niedersächsische Staats- und Universitätsbibliothek Göttingen, Cod. Ms. D. Hilbert 754, Bl. 6 Nr. 25.

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/finding_aids/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.
- 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/b00sslj0>. Audio Recording.
- Duncan, Arlene. 2015. The Bertrand Russell Research Centre. URL <http://russell.mcmaster.ca/>.

BIBLIOGRAPHY

- 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. forthcoming. Alonzo Church: Life and Work. In *The Collected Works of Alonzo Church*. Cambridge: 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*

BIBLIOGRAPHY

- 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.
- Hodges, Andrew. 2014. *Alan Turing: The Enigma*. London: Vintage.
- 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?primary_key=1508&search_category=author&search_page=1&search_form=get_results. 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: Akadémiai 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=6V3wNs4uv_4C&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=80oLTiVW_lc. Video Interview.
- 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.

BIBLIOGRAPHY

- Smith, Peter. 2013. *An Introduction to Gödel's Theorems*. Cambridge: Cambridge University Press.
- 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.
- 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).