# Introduction to Mathematical Logic

Version: Draft B.2.e.

Sam Buss

August 14, 2023

# Contents

# Preface

This book provides an introduction to propositional and first logic with an emphasis on mathematical development and rigorous proofs. The first chapters (Chapters I-IV) cover the completeness and soundness theorems for propositional and first-order logic. The next three chapters cover algorithms and Turing machines, and finally the Gödel Second Incompleteness Theorem. A final still-to-be-written Chapter **??** covers Herbrand's Theorem as an optional topic. An appendix—also still-to-be written—covers primitive recursive and partial recursive functions.

The book is intended as an undergraduate textbook in mathematical logic, intended chiefly for students in mathematics, computer science and philosophy. The reader is expected to have a certain level of mathematical sophistication, especially the willingness to deal with abstraction and mathematical proofs. However, the book does not presuppose much in the way of prerequisites or mathematical knowledge. Apart from some short sections that discuss algebraic structures (such as groups and fields), most of the mathematical prerequisites will be covered in a course in discrete mathematics. In addition, it is expected that readers will be comfortable with reading and writing (informal) mathematical proofs.

My goal in writing this book was to write an elementary and straightforward introduction to classical logic, which is still mathematically rigorous. The proof systems used in the book are conventional proof systems (called "PL" and "FO") with modus ponens and generalization as the main inference rules. Other proof systems are certainly possible, but PL and FO are arguably the most straightforward and most traditional proof systems. Considerable efforts have been made to streamline the presentation without skimping unduly on mathematical rigor. Only countable languages are covered, but uncountable languages are treated in a few of the exercises.

**Acknowledgments.** This book is heavily influenced by earlier textbooks. In particular, I have taught similar courses in the past from the books of Enderton [5], Boolos, Burgess and Jeffries [1], and Mendelson [13], and I used Hodel's textbook [10] as a supplementary text while writing this book. I learned mathematical logic from Shoenfield's book [18] and this also heavily influenced my choice of topics.

This text was initially written while teaching a course on mathematical logic

at the University of California, San Diego in Fall 2021 and Winter 2022. Most of the two courses was taught in-person, but part was taught online due to Covid-19 closures. I wish to thank the course participants not only for inspiring me to write this text but also for helpful feedback. Thanks are due to the students Mark Barbone, Michael Bradley, Elijah Camarena, Trevor Castle, Isaiah Dailey, Adriana Herrera, Jaeyeong Hwang, Arbi Leka, Andrew Paul, Joseph Phillips, Ayoob Shahmoradi, Kin James Wong, Zijian Zhang, and Zhicheng Zheng and to the teaching assistants Ryan Mike and Nathan Wenger for specific suggestions and corrections that helped improve the text. Thanks are also due to Jeff Edmonds and Frederick Manners for discussion on the overall choice of topics. Further thanks are due to Jonathan Aberle, David Auerbach, Alex Blum. Dennis Hamilton, and Roussanka Loukanova for comments, suggestions, and corrections to drafts of this book. Thanks are also due to Jeff Edmonds and Frederick Manners for discussion on the overall choice of topics.

At present, this is still a preliminary draft. Corrections and, more generally, suggestions for improvements will be greatly appreciated!

*Sam Buss*
La Jolla, California
August 2023

# Introduction

Logic is, loosely speaking, the science of how to express concepts and situations and reason correctly about them. Mathematical logic is chiefly concerned with expressions in formal languages, how to ascribe meanings to formal expressions, and how to reason with formal expressions using inference rules. Mathematical logic is also concerned with algorithmic issues including definability, computability, and complexity. Mathematical logic is furthermore a principal tool in the study of the foundations of mathematics.

Logic has diverse applications, especially in mathematics and computer science. The modern development of mathematical logic was motivated by the desire to establish a logical foundation for mathematics.[1] From that point of view, mathematical logic is a branch of mathematics that attempts to understand and justify all mathematical reasoning. With the Soundness and Completeness Theorems for first-order logic and the development of set theory, mathematical logic has been immensely successful in this endeavor and, via first-order set theory, has succeeded in establishing a foundation for all of mathematics. On the other hand, the theory of computability and the Gödel Incompleteness Theorems have revealed important limitations on the use of mathematical logic, and on the use of *any* formal system, for proving mathematical truths.

The modern formal theory of computability arose largely from the theory of Turing machines.[2] Turing machines are an idealized model of computation; the famous Church-Turing thesis states that Turing machines define a very general, robust notion of computation that captures the intuitive notion of computation. The original motivation for the definition of Turing machines was to prove the undecidability (the noncomputability) of mathematical truths about the integers. Turing achieved this by proving the undecidability of the halting problem. Subsequent developments in the theory of computer science have explored more restrictive notions of computation, ranging from time-bounded Turing machines, to Boolean circuits and to the more speculative prospects of quantum computation, to mention only a few. In this way, the theory of computer science is an important part of mathematical logic. It would be equally valid to say this the other way around: mathematical logic (especially, but not only, its discrete or

[1] This was initiated primarily by Frege in 1879 but underwent extensive refinement to reach modern form.

[2] Turing machines were defined by Turing in 1936. Other equivalent definitions for computability were independently developed by Church, by Post, and by Gödel and Herbrand.

finitary aspects) is an important part of theoretical computer science.

Indeed, mathematical logic has many applications in theoretical—and not-so-theoretical—computer science, including the theory of databases, hardware and software verification, and computer-based theorem proving. Applications of computer-based theorem proving range from proving theorems about mundane properties (such as the correctness of a software program with an SMT solver), to proving advanced mathematical theorems (e.g., with the aid of computerized theorem proving systems such as Mizar, Isabelle, MetaMath, Coq, Lean, etc.)

With a diverse range of applications, there is also a diverse range of formal systems of logic. The present book focusses on the core, classical theory of first-order logic. The first two chapters take up propositional logic, which is interesting in its own right and serves as a warm-up for the development of first-order logic in Chapters III and IV. Propositional logic and first-order logic both have a formal notion of syntax for formulas, an intuitive and straightforward notion of semantics, and a proof systems based on inference rules. Furthermore, they both have Soundness and Completeness Theorems. Strikingly, first-order logic is in some sense the strongest logic with all of these properties. In this way, propositional logic and first-order logic serve as touchstone systems, against which all other formal systems can be compared.

The main topics covered in the book include the following:

**Syntax and informal semantics.** Propositional and first-order logic use formulas that are formed with precise syntactic rules. For propositional logic, formulas are formed with the connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$ and $\leftrightarrow$, with the informal meanings of "and", "or", "not", "if-then" and "if-and-only-if". First-order logic adds the quantifiers "$\forall x$" and "$\exists x$" with the intuitive meanings of "forall $x$" and "exists $x$" where the variable $x$ ranges over some domain of objects. First-order logic also adds function symbols and relation symbols.

**Formal semantics.** The informal meanings of propositional and first-order formulas are formalized by giving mathematical definitions of the truth or falsity of formulas. In general, the truth or falsity of a formula depends on how its non-logical components are interpreted in an "interpretation". Interpretations in propositional logic are rather simple: they consist of assigning values T or F to propositional variables. Interpretations in first-order logic are much more complicated: they require choosing a domain of objects (a "universe") for the range of variables and choosing meanings for the function symbols and relation symbols.

Some formulas are true under all interpretations; such formulas are called "valid".

**Proofs and provability.** Propositional logic and first-order logic both have effective proof systems, in which formulas are deduced from axioms using inference rules. For propositional logic, we use a proof system PL that has *modus ponens* as its sole inference rule. Our proof system FO for first-order logic has two inference rules, *modus ponens* and *generalization*.

**Soundness and completeness.** The Soundness Theorem states that only valid formulas have formal proofs. The Completeness Theorem states that all valid formulas have formal proofs. Both propositional and first-order logic satisfy the Soundness and Completeness Theorems. This is an amazing fact!

It is common to work with sets of axioms; for example, the axioms for groups, the axioms for real closed fields, the Peano Arithmetic (PA) axioms for the (non-negative) integers, or the axioms for Zermelo-Frankel (ZF) set theory. The Soundness and Completeness Theorems still apply, and state that a formula can be proved from the axioms if and only if is a logical consequence of the axioms. Given the fact that first-order logic is strong enough to encompass all of mathematics via a first-order theory ZF for sets, this means that there is a first-order formal system that can encompass essentially all mathematical validities.[3]

**Computability and decidability.** The notion of computability is central to mathematical logic. The intuitive conception of computability corresponds to what can be carried out by an idealized computer that is not constrained by time and space resources. This is made mathematically precise by Turing machines, which provide a simple but flexible computational model that can simulate the action of any modern-day computer and thus, by the "Church-Turing Thesis", gives a mathematical definition of what it means for a function to be computable.

A set or a relation is called computable, or "decidable", if there is a (Turing machine) algorithm for deciding when a given input is in the set. It is called "computably enumerable" or "Turing enumerable" if there is an algorithm that exhaustively lists the members of the set or relation.

**Undecidability.** Surprisingly, it turns out that there are simple-to-describe sets that are undecidable; correspondingly, there are simple-to-describe functions that are not computable. A prime example of this is the "halting problem". Namely, the set of (descriptions of) Turing machines that do not halt is undecidable. In other words, there is no algorithm which, given an arbitrary Turing machine $M$, determines correctly whether $M$ ever halts.

The undecidability of the halting problem can be leveraged to prove that the set of true first-order statements about the integers is undecidable (when working with the the language containing the functions symbols + and $\cdot$ for addition and multiplication). Furthermore, the set of true first-order statements about the integers is not even computably enumerable.

**Incompleteness.** A set of axioms is "complete" if every (closed) formula is either provable or disprovable from the axioms.[4] In effect, being complete means that the axioms are sufficient to fully describe what is true or false.

[3]However, as we discuss below, encompassing all mathematical *validities* is not the same as encompassing all mathematical *truths*.

[4]We are being a bit loose here in informally defining the notion of "complete". See Definition III.98 for the actual definition for first-order logic, which applies to theories.

The First and Second Incompleteness Theorems, both due to Gödel, state that there is no complete axiomatization of the first-order theory of the integers. The first-order theory of Peano Arithmetic (PA) is the most common axiomatization of the integers. The axioms of PA are decidable; it follows that the theorems of PA are computably enumerable. This combined, with the undecidability of the halting problem and the representability of the halting problem in arithmetic, means that Peano Arithmetic is not complete.

At first glance, the Completeness and Incompleteness Theorems might seem contradictory. But there is no contradiction here. The Completeness Theorem states, for instance, that all formulas $C$ that are logical consequences of the axioms of PA (Peano Arithmetic) have proofs from the axioms of PA. That is to say, if $C$ is true in any setting where the axioms of PA hold (i.e., in any interpretation of PA), then $C$ has a proof from the axioms of PA. The Incompleteness Theorem implies that there are formulas $C$ that are true for the actual integers but do not have proofs from the axioms of PA. The point is that there are other interpretations of PA, called "nonstandard models of PA", which are different from the actual or true integers! There are formulas $C$ that are true in the actual ("standard") integers but false in some nonstandard model. These are statements that are true about the (standard) integers but cannot be proved from the axioms of Peano arithmetic.

## How to read this book.

Figure 1 shows the dependencies among chapters. Although the suggestion is to read the chapters in order, it is also possible to skip the chapters on propositional and first-order proofs and still read part of the chapters on algorithms and Turing machines and the possibly-to-be-written chapter on Herbrand's theorem.

The book is intended to be used for a course in mathematical logic. Even if the chapters are covered in order, it is important to introduce algorithmic concepts and the notion of computability early in the course. This will help prepare students for the more rigorous coverage of these important topics in Chapters V and VI.

Alternatively, a course could start with the first part of Chapter V, then cover undecidabilty using Section V.6 and Chapter VI, and afterwards cover the first four chapters while adding in topics from the rest of Chapter V.

For quarter-based courses, the author attempts to cover up through the Completeness Theorem in the first quarter, and then cover the Incompleteness Theorems in the second quarter. This makes for a rushed course, and typically some details need to be omitted. This schedule would be more leisurely for two semesters instead of two quarters.

The chapters all have a range of exercises. Many of them test basic understanding of the material, but some of them introduce new concepts. Readers are strongly encouraged to attempt the exercises.

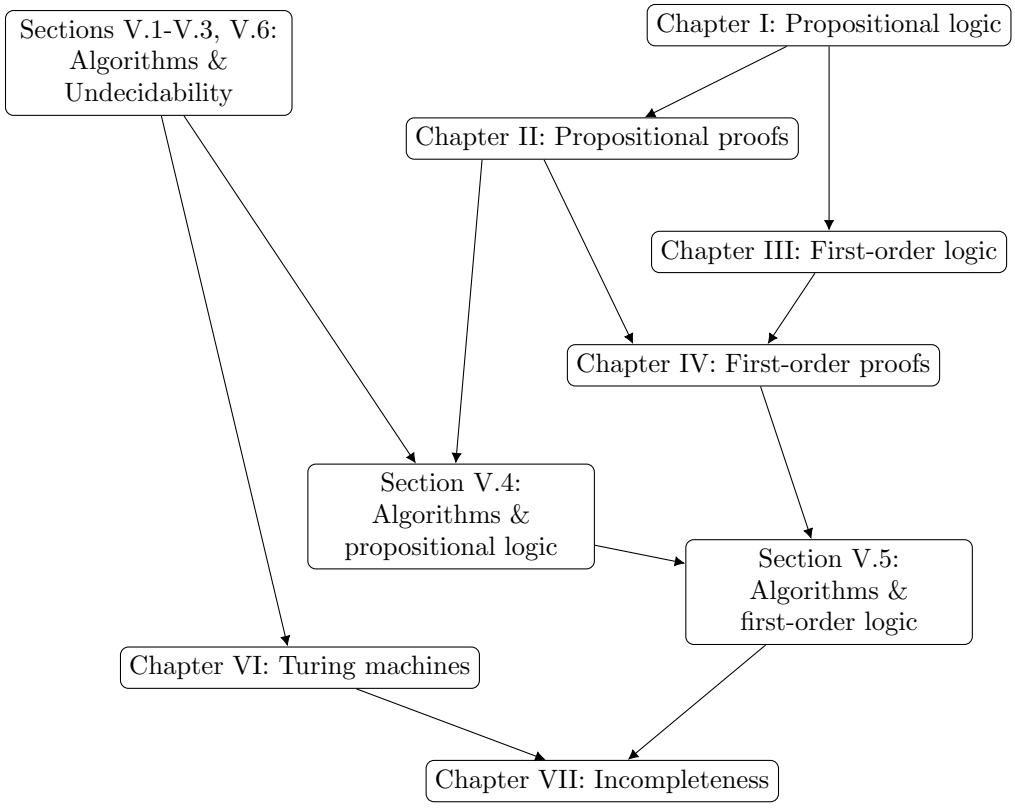



Figure 1: Chapter dependencies.

# Chapter I

# Propositional Logic: Syntax and Semantics

## I.1 Introduction to Propositional Logic

Propositional logic is used to express statements using "propositions" or "sentences" that take on true/false values. Propositional logic is sometimes called "sentential logic" or "Boolean logic"; the latter name is because it deals with the Boolean values of "True" and "False". As a simple example, consider three propositions denoted by variables $r, s, w$:

$r$ - "It is raining"
$s$ - "The sun is shining"
$w$ - "The grass is wet"

As propositions, they are assumed to take on definite true/false values. For example, it is assumed that it $r$ will assume a definite truth value of "True" or "False" indicating either that it is raining or that it is not raining; say at some particular time or place and without any ambiguity that might arise from issues such as whether a light mist counts as rain. Likewise, it assumed that $s$ and $w$ will assume definite truth values as to whether the sun is shining and the grass is wet.

More complex propositions are formed by combining propositional variables with operators such as "not" ($\neg$), "and" ($\wedge$), "or" ($\vee$). For example,

$\neg r$ - "It is not raining"
$r \wedge w$ - "It is raining and the grass is wet"
$r \vee s$ - "It is raining or the sun is shining"
$\neg(r \wedge s)$ - "It is not the case that both it is raining and the sun is shining"

The $\neg$ ("not") operator is also called *logical negation* or just *negation*. The $\wedge$ ("and") operator is also called *conjunction*, The $\vee$ ("or") operator is also called

*disjunction*. The connective $\vee$ denotes the "inclusive or" in that it allows the possibility that both arguments are true. Thus the formula $r \vee s$ is also true when both $r$ and $s$ are true.

Other common connectives include "if ... then ..." ($\rightarrow$) and "if and only if" ($\leftrightarrow$). The $\rightarrow$ operator is often called *implication*; and the $\leftrightarrow$ operator is the *equivalence* operator. For example,

$r \rightarrow w$ - "If it is raining then the grass is wet"
$w \rightarrow r$ - "The grass is wet only if it is raining"
$r \rightarrow w$ - "The grass is wet if it is raining"
$w \leftrightarrow r$ - "The grass is wet if and only if it is raining"

Some care needs to be taken in translating English sentences to propositional formulas since a natural language such as English allows many nuances that are completely lost in propositional logic. However, as shown in some of the above examples, "$B$ if $A$" generally means the same thing as "if $A$ then $B$" . It also means the same thing as "$A$ only if $B$".

Some less explicit uses of propositional connectives occur with "but" and "unless"; for example,

$r \wedge \neg w$ - "It is raining but the grass is not wet"
$(\neg w) \vee r$ - "The grass is not wet unless it is raining."

These illustrate that "but" may translate to the propositional connective $\wedge$ ("and"), and "$A$ unless $B$" may translate to $A \vee B$. The latter can also translate to $(\neg A) \rightarrow B$ because this means the same as "if $\neg A$, then $B$" or "if $A$ is false, then $B$ is true". On a related note, "If $A$, then $B$" means the same thing in propositional logic as $(\neg A) \vee B$.

For somewhat more problematic correspondences between English assertions and propositional logic, consider

$b$ - "You may buy a ticket"
$e$ - "You are over eighteen years old"

that are used to form the compound propositions

$e \rightarrow b$ - "You may buy a ticket if you are over eighteen"
$b \rightarrow e$ - "You may buy a ticket only if you are over eighteen"
$b \leftrightarrow e$ - "You may buy a ticket if and only if you are over eighteen"

If someone were to make these English statements in a real-world situation, they probably use any of the three to mean $b \leftrightarrow e$, i.e. that you may buy a ticket if and only if you are eighteen.[1] However, we use "if" and "only if" in a more specialized sense, as is common in logic and more generally in mathematics. This illustrates the need for caution in translating from a natural language to propositional logic.

[1] This ignores other such preconditions such as whether the ticket sales office is open and you have enough money.

Mathematical logic avoids the ambiguity of English. Instead, the propositional connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, etc., are purely truth functional, with values shown in Figure I.1. For instance, whether the formula $p \rightarrow q$ is true or false is determined solely by the truth or falsity of $p$ and $q$, and it does not matter whether there is some reason why the truth of $p$ should imply the truth of $q$. Indeed, $p \rightarrow q$ will always have the same truth value as $(\neg p) \vee q$.

This can be confusing sometimes, as it is somewhat counterintuitive that "if $p$, then $q$" should be true when $p$ is false, independently of whether $q$ is true or false. This is counterintuitive for the simple reason that in English, there is generally a distinction between "if $A$ then $B$" and "$B$ or not $A$". For instance, consider the rather dramatic difference in meanings of the two sentences

"If you will go to your house today, you will find your passport today",

and

"You will find your passport today, or you will not go to your house today".

The "if $A$ then $B$" form of the statement sounds like helpful advice on how to find your missing passport. On the other hand, the "$B$ or not $A$" form of the statement sounds like a threat from a border guard. These nuances are completely lost when translating the statements from English into propositional logic, since $A \rightarrow B$ means exactly the same thing as $B \vee \neg A$ in propositional logic.

In natural languages, a statement "if $A$ then $B$" statement usually means there is a reason why the truth of $A$ implies the truth of $B$. In propositional logic, however, statements like "If pigs have wings then horses can fly" can be true (because pigs do not have wings) in spite of the fact that wings on pigs would presumably have nothing to do with flying horses!

For an example of why this treatment of implications makes sense when dealing with mathematical statements, consider the propositions

$P(x)$ - "$x$ is a prime greater than 2"
$O(x)$ - "$x$ is odd"

Of course, we want the statement $\forall x(P(x) \rightarrow O(x))$,

"For all $x$, if $x$ is a prime greater than 2, then $x$ is odd",

to be true for $x$ ranging over all integers. Figure I.2 gives the truth values of $P(x)$ and $O(x)$ and $P(x) \rightarrow O(x)$ when $x = 1$, $x = 2$ and $x = 3$. We want $P(x) \rightarrow O(x)$ to be true in all three cases, including the two cases where $P(x)$ is false. Thus we want "F $\rightarrow$ F" and "F $\rightarrow$ T" both to evaluate as true to give the correct results in the cases where $x$ is equal to 1 or 2.

## I.2 Propositional Formulas

We now give the formal definitions of the syntax of propositional formulas. After that, the next section will define the truth value of a formula in terms of the truth values of its variables.

| $p$ | $\neg p$ |
|---|---|
| T | F |
| F | T |

| $p$ | $q$ | $p \vee q$ | $p \wedge q$ | $p \to q$ | $p \leftrightarrow q$ |
|---|---|---|---|---|---|
| T | T | T | T | T | T |
| T | F | T | F | F | F |
| F | T | T | F | T | F |
| F | F | F | F | T | T |

Figure I.1: The values of propositional connectives. "T" and "F" denote the Boolean values "True" and "False".

Propositional formulas will be expressions, that is strings of symbols. The permitted symbols are:

(a) Variables $p_1, p_2, p_3, \ldots$,

(b) Propositional connectives $\neg$, $\wedge$, $\vee$, $\to$ and $\leftrightarrow$, and

(c) Parentheses ( and ).

**Definition I.1.** The set of propositional formulas is inductively defined by:

(a) $p_i$ is a propositional formula, for any $i \geq 1$.

(b) If $A$ is a propositional formula, then so is $\neg A$.

(c) If $A$ and $B$ are propositional formulas, then so are $(A \vee B)$ and $(A \wedge B)$ and $(A \to B)$ and $(A \leftrightarrow B)$.

It is implicit in the definition that rules (a)-(c) are the only ways of generating propositional formulas. It is important that parentheses are included exactly as indicated.

**Example I.2.** Examples of propositional formulas include:

$$p_1 \qquad p_2 \qquad p_{1000} \qquad \neg p_2 \qquad \neg\neg p_2$$

$$(p_1 \to (\neg p_2 \to p_1)) \qquad ((p_1 \to \neg p_2) \to p_1)$$

$$\neg((p_1 \leftrightarrow \neg\neg\neg p_2) \to (\neg p_2 \vee (p_1 \wedge p_1))).$$

On the other hand, $p_1 \wedge p_2$ is not a propositional formula due to missing parentheses, and $(\neg p_3)$ is not a propositional formula due to having extra parentheses. Likewise, $p \vee r$ is not a propositional formula due to using variables other than the $p_i$'s.

**Informal abbreviations for formulas.** The formal definition of formulas requires the inclusion of many open and close parentheses. These are useful for uniquely parsing formulas but often detract from readability. Thus it is common to informally abbreviate formulas by omitting parentheses when writing out formulas. For instance, we can always omit writing the outermost parentheses without any ambiguity about what the formula is. However, it is important that we know how to reinsert parentheses to recover the actual formula. We do this according to the following rules of precedence:

| $x$ | $P(x)$ | $O(x)$ | $P(x) \to O(x)$ |
|---|---|---|---|
| 1 | F | T | T |
| 2 | F | F | T |
| 3 | T | T | T |

Figure I.2: Some values of $P(x) \to O(x)$.

- The negation sign, $\neg$, has the highest precedence.
- The connectives $\wedge$ and $\vee$ have the second highest precedence.
- The connectives $\to$ and $\leftrightarrow$ have the lowest precedence.
- If parentheses are omitted amongst connectives of the same precedence, they are associated from right to left.

Some examples are:

| Informal abbreviation | Actual formula |
|---|---|
| $\neg p_1 \vee \neg p_2 \vee p_3$ | $(\neg p_1 \vee (\neg p_2 \vee p_3))$ |
| $\neg p_1 \to \neg p_2 \to p_3$ | $(\neg p_1 \to (\neg p_2 \to p_3))$ |
| $\neg(p_1 \to \neg p_2) \to p_3$ | $(\neg(p_1 \to \neg p_2) \to p_3)$ |
| $p_1 \vee \neg p_2 \to \neg p_3 \wedge p_4$ | $((p_1 \vee \neg p_2) \to (\neg p_3 \wedge p_4))$ |

In addition to omitting parentheses, variable names such as $p, q, r, \ldots$ are frequently used instead of $p_1, p_2, p_3, \ldots$. This is just to improve readability; formally only variables $p_i$ can appear in formulas.

The truth functions for $\wedge$, $\vee$, and $\leftrightarrow$ are associative, so the convention about associating from right to left is not so important for them. However, $\to$ is not associative, as $p \to (q \to r)$ is not equivalent to $(p \to q) \to r$. The formula $p \to (q \to r)$ is however equivalent truth-functionally to the formula $p \wedge q \to r$. This is often a convenient and useful way to write a formula, and it is for this reason that we have chosen to associate from right to left.

It is best to avoid writing things like $A \vee B \wedge C$ or $A \to B \leftrightarrow C$, to avoid confusing readers. According to our conventions they mean $A \vee (B \wedge C)$ or $A \to (B \leftrightarrow C)$, but other authors use different conventions.

**Proofs by induction and recursive definitions.** The reason for being so careful about giving a formal mathematical definition of propositional formulas is that we will use to it prove theorems about propositional formulas. That is to say, we do not merely use formulas, we also prove theorems about formulas.

Definition I.1 gave an inductive definition for formulas. This allows properties of formulas to be proved by induction. Specifically, suppose we wish to prove that every propositional formula satisfies some property $\mathcal{Q}$. To prove this by induction, it suffices to show that:

(a) Every formula $p_i$ satisfies property $\mathcal{Q}$,

(b) If $A$ is a propositional formula that satisfies property $\mathcal{Q}$, then the formula $\neg A$ also satisfies property $\mathcal{Q}$, and

(c) If $A$ and $B$ are propositional formulas that each satisfy property $\mathcal{Q}$, then the formulas $(A \wedge B)$, $(A \vee B)$, $(A \to B)$ and $(A \leftrightarrow B)$ all satisfy property $\mathcal{Q}$.

The first case, (a), serves as the base case. Cases (b) and (c) serve as the induction steps. This means there are two induction steps to be proved, or even five induction steps if case (c) has to be treated as four different induction steps.

The third case, (c), of Definition I.1 for propositional formulas treats the case where the principal connective is $\wedge$, $\vee$, $\to$ or $\leftrightarrow$. It requires that parentheses be included so that the formula has the form $(A \circ B)$, for $\circ$ one of the four binary connectives. The presence of all these parentheses means that there is a unique way to parse any propositional formula. That is to say, if $C$ is a propositional formula, then there is a unique way to express it in one of the forms (a)-(c) of Definition I.1. This is called the *unique readability* property and means that propositional formulas can be uniquely parsed.

The inductive definition of propositional formulas in Definition I.1 and the unique readability property together allow using inductive definitions (also called "recursive definitions") to define functions of propositional formulas. The general framework for defining a function $h(A)$ by recursion on the complexity of the formula $A$ is as follows.

(a) For the base case, we define the value of $h$ for a propositional formula $p_i$.

(b) For the first case of recursion, we suppose that the value of $h$ for a formula $A$ is already defined. We use this to define the value of $h$ for the formula $\neg A$.

(c) For other cases of recursion, we suppose that the values of $h$ on the formulas $A$ and $B$ have already been defined. We use these to define the value of $h$ for the formula $(A \circ B)$ where $\circ$ is one of the connectives $\wedge$, $\vee$, $\to$ or $\leftrightarrow$.

The unique readability property means that there is exactly one way to express a propositional formula in one of the forms $p_i$ or $\neg A$ or $(A \circ B)$ to apply recursively one of the cases (a), (b), or (c). Thus the value $h$ is uniquely defined for all formulas.

A prominent use of definition by recursion is the Definition I.4 given in the next section for the truth value of a propositional formula. Section I.9 gives some examples of proofs by induction on the complexity of formulas.

## I.3 Definition of Truth in Propositional Logic

The previous section defined the *syntax* of propositional formulas; now we define their *semantics*, namely what it means for a propositional formula to be true or false. The truth or falsity of a formula depends on the truth values of the variables appearing in it. For instance, the formula $p_1 \to p_2$ could have value

either True (T) or False (F) depending on the values of $p_1$ and $p_2$. Accordingly, we start with an assignment $\varphi$ of truth values to the propositional variables.

**Definition I.3.** A *truth assignment* $\varphi$ is a mapping

$$\varphi : \{p_1, p_2, p_3, \ldots\} \to \{\mathrm{T}, \mathrm{F}\}$$

from the set of propositional variables to the set of truth values T and F.

A truth assignment $\varphi$ to the variables gives a truth value $\varphi(p_i)$ to each variable $p_i$. We can extend $\varphi$ to give truth values to all propositional formulas:

**Definition I.4** (Definition of Truth for Propositional Formulas)**.** Suppose $\varphi$ is a truth assignment. The function $\varphi$ is extended to have domain the set of all propositional formulas by defining $\varphi$ by recursion as:

(a) If $A$ is $\neg B$, then $\varphi(A) = \begin{cases} \mathrm{F} & \text{if } \varphi(B) = \mathrm{T} \\ \mathrm{T} & \text{otherwise} \end{cases}$

(b) If $A$ is $(B \vee C)$, then $\varphi(A) = \begin{cases} \mathrm{T} & \text{if } \varphi(B) = \mathrm{T} \text{ or } \varphi(C) = \mathrm{T} \\ \mathrm{F} & \text{otherwise} \end{cases}$

(c) If $A$ is $(B \wedge C)$, then $\varphi(A) = \begin{cases} \mathrm{T} & \text{if } \varphi(B) = \mathrm{T} \text{ and } \varphi(C) = \mathrm{T} \\ \mathrm{F} & \text{otherwise} \end{cases}$

(d) If $A$ is $(B \to C)$, then $\varphi(A) = \begin{cases} \mathrm{T} & \text{if } \varphi(B) = \mathrm{F} \text{ or } \varphi(C) = \mathrm{T} \\ \mathrm{F} & \text{otherwise} \end{cases}$

(e) If $A$ is $(B \leftrightarrow C)$, then $\varphi(A) = \begin{cases} \mathrm{T} & \text{if } \varphi(B) = \varphi(C) \\ \mathrm{F} & \text{otherwise} \end{cases}$

Note how the definition of truth follows the informal meanings as discussed in Section I.1.

**Example I.5.** Let $A$ be the formula $(p \vee q) \to (q \wedge p)$. There are only two variables $p, q$ in this formulas, so the truth value $\varphi(A)$ of $A$ depends on $\varphi(p)$ and $\varphi(q)$; however $\varphi(A)$ does not depend on $\varphi(r)$ for any other variable $r$. Since $\varphi(p)$ and $\varphi(q)$ can only be T or F, there are only four ways to set the truth values of $p$ and $q$. These are pictured in the truth table shown in Figure I.3 in the first two columns. The second-to-last column of the truth table gives the corresponding values of $\varphi((p \vee q) \to (q \wedge p))$.

In the second line of Figure I.3, $\varphi(p) = \mathrm{T}$ and $\varphi(q) = \mathrm{F}$. Hence according to two applications of the definition of truth, $\varphi(p \vee q) = \mathrm{T}$ and $\varphi(p \wedge q) = \mathrm{F}$. With one more application of the definition of truth, $\varphi((p \vee q) \to (q \wedge p)) = \mathrm{F}$. That is $A$ is false when $p$ is true and $q$ is false.

Let $B$ be the formula $(p \wedge q) \to (q \vee p)$. The final column of Figure I.3 shows the values of $\varphi(B)$ for all possible assignments of values to $p$ and $q$. Since $\varphi(B) = \mathrm{T}$ for four assignments, we call $B$ a "tautology" or say that $B$ is "tautologically valid". On the other hand, $\varphi(A)$ is T for some truth assignments and false for others. Therefore, $A$ is not a tautology; it is "satisfiable", however, since there is a truth assignment that makes $A$ true.

The next section discusses tautologies and satisfiability in more depth.

| $p$ | $q$ | $p \vee q$ | $p \wedge q$ | $p \vee q \to p \wedge q$ | $p \wedge q \to p \vee q$ |
|---|---|---|---|---|---|
| T | T | T | T | T | T |
| T | F | T | F | F | T |
| F | T | T | F | F | T |
| F | F | F | F | T | T |

Figure I.3: The truth tables for $(p \vee q) \to (p \wedge q)$ and $(p \wedge q) \to (p \vee q)$.

## I.4 Satisfiability, Tautologies, and Implication

**Satisfiability and tautologies.** A formula is a "tautology" if it is always true; it is "satisfiable" if it can sometimes be true. Formally:

**Definition I.6.** A formula $A$ is a *tautology* if $\varphi(A) = \mathrm{T}$ for all truth assignments $\varphi$. In this case, we also say that $A$ is *tautologically valid*.

A formula $A$ is *satisfiable* if there is some truth assignment $\varphi$ such that $\varphi(A) = \mathrm{T}$. In this case, we say that $\varphi$ is a *satisfying assignment* for $A$ or that $A$ is *satisfied by* $\varphi$. If there is no satisfying assignment for $A$, then $A$ is *unsatisfiable*.

The notation "$\vDash A$" will also be used to denote that $A$ is a tautology. For this, see Definition I.10 below.

**Example I.7.** The formula $p_1$ is satisfiable but not a tautology, since $\varphi(p_1)$ may equal either T or F. The formulas $p_1 \vee \neg p_1$ and $p_1 \to (p_2 \to p_1)$ are tautologies, and hence are satisfiable. The formula $p_1 \wedge \neg p_1$ is unsatisfiable.

As already discussed, Figure I.3 shows that $p \wedge q \to p \vee q$ is a tautology and that $p \vee q \to p \wedge q$ is satisfiable but not a tautology.

It may not be immediately obvious that the formula $p_1 \to p_2 \to p_1$ is a tautology; one way to check this is by the method of truth tables. Figure I.4 shows the truth table, which shows that $p_1 \to p_2 \to p_1$ is true under all four possible combinations of values of $\varphi(p_1)$ and $\varphi(p_2)$.

More generally, suppose a formula $A$ involves $k$ many distinct variables, $p_{i_1}, \ldots, p_{i_k}$. There are $2^k$ many ways to set the values of $\varphi(p_{i_j})$. A truth table for $A$ will thus have $2^k$ lines. This gives an algorithm for determining whether a given formula $A$ is satisfiable and/or a tautology; namely, consider all possible $2^k$ truth assignments $\varphi$ on the variables in $A$ and evaluate $\varphi(A)$. Then $A$ is a tautology if and only if $\varphi(A) = \mathrm{T}$ for all $2^k$ assignments. And $A$ is satisfiable if and only if $\varphi(A) = \mathrm{T}$ for at least one of the truth assignments.

Now let $\Gamma$ denote a set of propositional formulas. It is often interesting to know whether there is a truth assignment that makes every formula in $\Gamma$ true.

**Definition I.8.** A set $\Gamma$ of propositional formulas is *satisfiable* if there is a truth assignment $\varphi$ such that $\varphi(A) = \mathrm{T}$ for every $A \in \Gamma$. For such a $\varphi$, we say that $\varphi$ *satisfies* $\Gamma$ or that $\varphi$ is a *satisfying assignment* for $\Gamma$ or that $\Gamma$ is *satisfied by* $\varphi$.

| $p_1$ | $p_2$ | $p_2 \to p_1$ | $p_1 \to (p_2 \to p_1)$ |
|---|---|---|---|
| T | T | T | T |
| T | F | T | T |
| F | T | F | T |
| F | F | T | T |

| $p_1$ | $p_2$ | $p_1 \to (p_2 \to p_1)$ | |
|---|---|---|---|
| T | T | T | T |
| T | F | T | T |
| F | T | T | F |
| F | F | T | T |

Figure I.4: Truth tables showing that $p_1 \to p_2 \to p_1$ is a tautology. The tables are identical, but the one on the right is written in compact form. The circled values are the truth values for the entire formula.

The set $\Gamma$ may be either finite or infinite. If $\Gamma$ is finite, $\Gamma = \{A_1, A_2, \ldots, A_k\}$, then $\Gamma$ is satisfied by $\varphi$ if and only if $\varphi$ satisfies $A_1 \wedge A_2 \wedge \cdots \wedge A_k$.

If $\Gamma$ is finite, then the method of truth tables may be used to determine whether $\Gamma$ is satisfiable. However, if $\Gamma$ is infinite then the method of truth tables cannot be used. This is because in general there may be infinitely many variables in $\Gamma$, and one would have to consider infinitely many truth assignments.

**Example I.9.** Let $\Gamma$ be the set of formulas

$$\Gamma \;=\; \{p_i \leftrightarrow \neg p_{i+1} : i \geq 1\} \;=\; \{p_1 \leftrightarrow \neg p_2,\, p_2 \leftrightarrow \neg p_3,\, p_3 \leftrightarrow \neg p_4,\, \ldots\}.$$

There are exactly two satisfying assignments for $\Gamma$. The first one, $\varphi_1$, has $\varphi_1(p_{2j-1}) = \mathrm{T}$ and $\varphi_1(p_{2j}) = \mathrm{F}$ for all $j \geq 1$. The second one, $\varphi_2$, has $\varphi_2(p_{2j-1}) = \mathrm{F}$ and $\varphi_2(p_{2j}) = \mathrm{T}$ for all $j \geq 1$.

**Tautological implication.** A *tautological implication* is an implication that is true under any truth assignment. This is defined formally as follows:

**Definition I.10.** Let $A$ and $B$ be formulas, and $\Gamma$ be a set of formulas.

(a) We say that $A$ *tautologically implies* $B$, or just $A$ *implies* $B$ for short, provided that every truth assignment that satisfies $A$ also satisfies $B$. We write $A \vDash B$ to denote that $A$ tautologically implies $B$.

(b) We say that $\Gamma$ *tautologically implies* $B$, or just $\Gamma$ *implies* $B$ for short, provided that every truth assignment that satisfies $\Gamma$ also satisfies $B$. We write $\Gamma \vDash B$ to denote that $\Gamma$ tautologically implies $B$.

Unwinding Definitions I.8 and I.10(b), $\Gamma \vDash A$ means the same as

$$\text{For all } \varphi, \text{ if } \varphi(B) = \mathrm{T} \text{ for all } B \in \Gamma, \text{ then } \varphi(A) = \mathrm{T}.$$

Or equivalently,

$$\text{For all } \varphi, \text{ if } \varphi \text{ satisfies } \Gamma, \text{ then } \varphi \text{ satisfies } A.$$

The symbol "$\vDash$" is called the "double turnstile" sign. Note that $A \vDash B$ means the same thing as $\{A\} \vDash B$. It is common to take liberties in notation

and write things like $A, B \vDash C$ for $\{A, B\} \vDash C$, and $\Gamma, A \vDash B$ for $\Gamma \cup \{A\} \vDash C$. In all cases, it means that any truth assignment that satisfies all the formulas and sets of formulas to the left of the $\vDash$ sign also satisfies the formula to the right of the $\vDash$ sign. Finally, we write just $\vDash A$ for $\varnothing \vDash A$, where $\varnothing$ is the empty set of (no) formulas. The condition $\vDash A$ holds if and only $A$ is a tautology.

We write $\Gamma \nvDash A$ to mean that $\Gamma \vDash A$ is false.

**Example I.11.** Some examples of tautological (non)-implication with a finite set $\Gamma$ include:

(a) $p_1, p_1 \to p_2 \vDash p_2$.
(b) $p_1, p_2 \vDash p_2 \wedge p_1$.
(c) $p_1 \nvDash p_2$.
(d) $\vDash p_1 \to p_2 \to p_1$.
(e) $\nvDash (p_1 \to p_2) \to p_1$.

These can be verified by using truth tables. Note that (a) means the same as $\{p_1, p_1 \to p_2\} \vDash p_2$, and similarly for (b).

**Example I.12.** Let $\Gamma$ be the set of formulas from Example I.9, and $\varphi_1$ and $\varphi_2$ be its two satisfying assignments. Then

(a) $\Gamma \nvDash p_1$ and $\Gamma \nvDash \neg p_1$.
(b) $\Gamma \vDash p_1 \leftrightarrow p_3$.

Condition (a) holds since $\varphi_2(p_1) = \mathrm{F}$ and $\varphi_1(\neg p_1) = \mathrm{F}$. Condition (b) holds since $\varphi_1$ and $\varphi_2$ are the only satisfying assignments, and since $\varphi_i(p_1) = \varphi_i(p_3)$ and thus $\varphi_i(p_1 \leftrightarrow p_3) = \mathrm{T}$ for both $i = 1$ and $i = 2$.

**Theorem I.13.** *Suppose that $\Gamma$ and $\Delta$ are sets of formulas, and $\Gamma \subseteq \Delta$.*
(a) *If $\varphi$ satisfies $\Delta$, then $\varphi$ satisfies $\Gamma$.*
(b) *If $\Delta$ is satisfiable, then $\Gamma$ is also satisfiable.*
(c) *If $\Gamma \vDash A$, then $\Delta \vDash A$.*

Theorem I.13 is an immediate consequence of the definitions. □

**Implication from an unsatisfiable set.** An unsatisfiable set $\Gamma$ tautologically implies all formulas $B$:

**Theorem I.14.**
(a) *Let $A$ and $B$ be formulas. Then $A, \neg A \vDash B$.*
(b) *Let $B$ be a formula and $\Gamma$ be an unsatisfiable set of formulas. Then $\Gamma \vDash B$.*

Part (a) is a special case of part (b) since $\{A, \neg A\}$ is unsatisfiable. Part (b) is proved by observing that since there is no truth assignment satisfying $\Gamma$, the definition of tautological implication means that $\Gamma \vDash B$ holds trivially. □

**Tautologically equivalent formulas.** Two formulas are "tautologically equivalent" if they tautologically imply each other:

**Definition I.15.** We write $A \models\!\dashv B$ to denote that both $A \models B$ and $B \models A$. In this case, we say that $A$ and $B$ are *tautologically equivalent*. Immediately from the definitions, $A \models\!\dashv B$ holds if and only if $\varphi(A) = \varphi(B)$ for all truth assignments.

**The semantic deduction theorem.** The next theorem is the semantic version of what will later be called the Deduction Theorem (see Section II.3).

**Theorem I.16** (Semantic Deduction Theorem)**.**
*Let $A$ and $B$ be formulas and $\Gamma$ a set of formulas. Then,*

(a) $A \models B$ *if and only if* $\models A \to B$.

(b) $\Gamma, A \models B$ *if and only if* $\Gamma \models A \to B$.

*Proof.* Part (a) is the special case of (b) with $\Gamma$ equal to the empty set. So let's prove (b). The proof is basically just unwinding definitions. To start, the condition $\Gamma, A \models B$ means that for any truth assignment $\varphi$, if $\varphi$ satisfies $\Gamma \cup \{A\}$, then $\varphi(B) = \mathrm{T}$. Equivalently, this means that if $\varphi$ satisfies $\Gamma$ and $\varphi(A) = \mathrm{T}$, then $\varphi(B = \mathrm{T})$. By the definition of truth for $\to$, the condition

$$\text{"if } \varphi(A) = \mathrm{T}\text{, then } \varphi(B) = \mathrm{T}\text{"}$$

is equivalent to $\varphi(A \to B) = \mathrm{T}$. Therefore, $\Gamma, A \models B$ is equivalent to the condition that for all $\varphi$, if $\varphi$ satisfies $\Gamma$, then $\varphi(A \to B) = \mathrm{T}$. In other words, it is equivalent to $\Gamma \models A \to B$. □

**Example I.17.** Returning to the set $\Gamma$ of Examples I.9 and I.12, we have $\Gamma, p_1 \models p_3$ and $\Gamma, p_3 \models p_1$.

**Theorem I.18.** *Suppose $\Gamma$ is a finite set of formulas, $\Gamma = \{A_1, \ldots, A_k\}$. Then $\Gamma \models B$ if and only if $A_1 \wedge A_2 \wedge \cdots \wedge A_k \models B$.*

*Proof.* A truth assignment $\varphi$ satisfies $\Gamma$ if and only if it satisfies $A_1 \wedge A_2 \wedge \cdots \wedge A_k$. Therefore, $\Gamma \models B$ if and only if $A_1 \wedge A_2 \wedge \cdots \wedge A_k \models B$. □

**Theorem I.19.** $A \models\!\dashv B$ *if and only if* $\models A \leftrightarrow B$.

Theorem I.19 follows readily from the Semantic Deduction Theorem and the fact that $\models A \leftrightarrow B$ holds if and only both $\models A \to B$ and $\models B \to A$.

**Duality between satisfiability and validity.** The next two theorems show how the property of being satisfiable is dual to the property of being a tautology.

**Theorem I.20.** *Let $A$ be a formula. Then $A$ is a tautology if and only if $\neg A$ is not satisfiable.*

Theorem I.20 is a special case of the next theorem, with $\Gamma = \varnothing$. This is a precursor to Theorems II.19 and II.21 about proof by contradiction.

**Theorem I.21.** *Let $\Gamma$ be a set of formulas and $A$ be a formula. Then $\Gamma \vDash A$ if and only if $\Gamma \cup \{\neg A\}$ is unsatisfiable.*

*Proof.* The condition that $\Gamma \vDash A$ means that every truth assignment $\varphi$ that satisfies $\Gamma$ has $\varphi(A) = \mathrm{T}$. The condition that $\Gamma \cup \{\neg A\}$ is unsatisfiable is equivalent to the condition that every truth assignment that satisfies $\Gamma$ has $\varphi(\neg A) = \mathrm{F}$. These two conditions are clearly equivalent. □

## I.5 Truth Tables

We have already seen a couple of examples of truth tables. This section will discuss how to write truth tables more compactly, how to form "reduced" truth tables that can require fewer lines, and how reduced truth tables correspond to decision trees.

The principal purpose of a truth table is to determine whether or not a formula is satisfiable or tautologically valid or neither. For a tautology $A$, a truth table can serve as a proof that $A$ is valid.

There are straightforward algorithms to build truth tables, and to verify the correctness of truth tables. The algorithm to build truth tables works roughly as follows. The input is a formula $A$. The algorithm first parses $A$ into subformulas and determines the distinct variables that appear in $A$. If there are $k$ distinct variables, the algorithm then writes out a truth table with $2^k$ lines, one for possible assignment of truth values to the $k$ variables. For each of the $2^k$ assignments, the algorithm uses the definition of truth to compute the truth values of all of the subformulas of $A$, starting with the propositional variables and working up to the value of the entire formula $A$. If the value of $A$ is discovered to be True (T) for all $2^k$ truth assignments, the algorithm outputs that $A$ is a tautology. Otherwise, it outputs the $A$ is not a tautology. Similarly, if at least one assignment makes the value of $A$ true, then $A$ is satisfiable.

This is an example of an "effective" algorithm since the algorithm gives an answer, and the correct answer, for any input formula $A$. Unfortunately, it is not a very fast algorithm, since its run time turns out to be at least $2^k$ just because it has to create $2^k$ lines of the truth table. Thus, this is a so-called "exponential time" algorithm, and as $k$ grows large, it quickly becomes infeasible to run the algorithm because it is too slow.

For instance, the universe is believed to be approximately 13 billion years old, which is approximately $4 \times 10^{26}$ many nanoseconds. Therefore $2^{88}$ is approximately equal to the age of the universe in nanoseconds. Thus the truth table for a formula $A$ with more than about 100 variables would require more lines in its truth table than the age of the universe measured in nanoseconds. This is clearly not feasible to form!

| $p$ | $q$ | $r$ | $[p \to (q \to r)]$ | | | $\leftrightarrow$ | $[(p \wedge q) \to r]$ | |
|---|---|---|---|---|---|---|---|---|
| T | T | T | T | T | T | T | T | T |
| T | T | F | F | F | T | T | T | F |
| T | F | T | T | T | T | T | F | T |
| T | F | F | T | T | T | T | F | T |
| F | T | T | T | T | T | T | F | T |
| F | T | F | T | F | T | T | F | T |
| F | F | T | T | T | T | T | F | T |
| F | F | F | T | T | T | T | F | T |

Figure I.5: A compact truth table. The solid box shows the values of the whole formula. The dotted boxes show the values of the two subformulas $p \to q \to r$ and $(p \wedge q) \to r$. The unboxed columns give the truth values of $q \to r$ and $p \wedge q$.

**Compact form of truth tables.** Figure I.3 and the lefthand side of Figure I.4 gives two examples of truth tables in traditional notation. In these truth tables, there is a separate column for each subformula of the formula $A$ being evaluated.

The righthand side of Figure I.4 gives the truth table in more compact form. In that table, the formula $A$ (in this case, $p_1 \to p_2 \to p_1$) is written at the top of the right column but its subformulas are not rewritten into separate columns. The truth values of the subformulas of $A$ are written below their principal connectives. In this case, the truth values for the subformula $p_2 \to p_1$ are written below the principal $\to$ connective for the subformula. Likewise, the truth values for the whole formula are written below the principal connective of the whole formula, namely below the first $\to$ sign in $A$.

Figure I.5 shows another example of a compact truth table, for the formula

$$(p \to q \to r) \leftrightarrow (p \wedge q \to r). \tag{I.1}$$

This shows that (I.1) is a tautology. It follows that $p \to (q \to r)$ and $(p \wedge q) \to r$ are tautologically equivalent (see Theorem I.19).

**Reduced truth tables.** A big disadvantage of truth tables is that they can be awkwardly big, with $2^k$ many lines when there are $k$ variables. In many cases, it is possible to form "reduced" truth tables with fewer lines. Reduced truth tables are based on the fact that sometimes setting just a few of the variables to true or false is enough to force the value of the whole formula. For example, in Figure I.5 the formula will be true whenever $r$ is is true. Thus when $r$ is true, the values of $p$ and $q$ are unimportant.

This insight lets us form the reduced truth table shown in Figure I.6. The first line of the reduced truth table considers the case where $\varphi(r) = \mathrm{T}$, and $\varphi(p)$ and $\varphi(q)$ are not determined. In that first line, the definition of truth states that $\varphi(p \wedge q \to r)$ must equal T no matter what the truth value of $p \wedge q$ is. Likewise, $\varphi(q \to r)$ and $\varphi(p \to q \to r)$ must equal T. This lets us fill enough of the entries on the first line to determine that the entire formula has truth

| $p$ | $q$ | $r$ | $[p \to (q \to r)] \leftrightarrow [(p \wedge q) \to r]$ | | | | |
|---|---|---|---|---|---|---|---|
| - | - | T | T | T | T | - | T |
| - | F | F | T | T | T | F | T |
| T | T | F | F | F | T | T | F |
| F | T | F | T | F | T | F | T |

Figure I.6: A reduced truth table.

value T. The second line considers the case where $\varphi(r)$ = F and $\varphi(q)$ = F and $\varphi(p)$ has not been determined. The third and fourth lines consider the two cases where $\varphi(r)$ = F and $\varphi(q)$ = T and where $\varphi(p)$ is equal to either T or F.

Figure I.6 has half as many rows as Figure I.5, and still manages to cover all the relevant cases of how the truth assignment is set.

Unfortunately, even though reduced truth tables sometimes provide substantial savings in size over full truth tables, they can still require exponentially many lines. Nonetheless, they can be very useful when creating small tables by hand.

**Reduced truth tables and decision trees.** Each line in a reduced truth table for a formula $A$ corresponds to a "partial truth assignment". A *partial truth assignment* means an assignment of truth values to some subset of the propositional variables appearing in $A$. A crucial property of reduced truth tables is the partial truth assignments cover all possible truth assignments, in the sense that every possible truth assignment extends at least one of the partial truth assignments. This begs the question, however, of how do we know that the partial truth assignments from a reduced truth table cover all possible assignments.[2]

It is thus desirable that reduced truth tables use some pattern of partial truth assignments that ensures that they cover every possible truth assignment. One natural way to do this is to require that the partial truth assignments in the reduced truth table correspond to a *binary decision tree*. A binary decision tree means a procedure that queries the truth values of variables one at a time: when a variable $p_i$ is queried, the procedure receives the truth value $\varphi(p_i)$ of $p_i$. Rather than define this formally, we illustrate with an example based on the reduced truth table of Figure I.6.

A binary decision tree is shown in Figure I.7. This represents a procedure that traverses a branch in the tree, starting at the root node, querying the truth value of a variable at each node, and following the edge labeled with the variable's value to a child node. When a leaf node is reached, the value of the formula $A$ is known.

The decision tree in Figure I.7 has its root labeled with $r$; this is the first

[2]For readers familiar with the theory of P and NP: It is not hard to show that it is NP-hard to determine if a set of partial truth assignments covers all possible truth assignments.

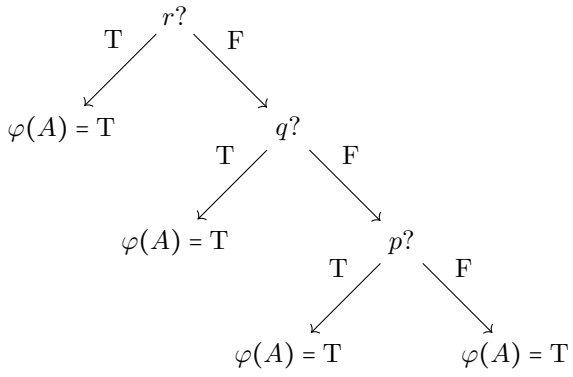



Figure I.7: The decision tree corresponding to the reduced truth table of Figure I.6. The formula $A$ is $(p \to q \to r) \to (p \wedge q \to r)$. The leaves all being labelled $\varphi(A) = \mathrm{T}$ indicates that $A$ is a tautology.

variable queried. There are two edges leaving $r$, so $r$ has two children. The first edge is labelled T meaning that this edge is followed when $\varphi(r) = \mathrm{T}$. The second edge is labelled F and that edge is followed when $\varphi(r) = \mathrm{F}$. One child below $r$ is a leaf labelled with "$\varphi(A) = \mathrm{T}$": that indicates that once that leaf is reached, straightforward calculations show that the entire formula $A$ has truth value T (namely, the calculations in the first line of the truth table in Figure I.6).

The other child of $r$ is labelled with $q$, so $q$ is queried next when $\varphi(r) = \mathrm{F}$. When $q$ is queried, its value $\varphi(q)$ will be either T or F. The process continues similarly until a leaf is reached. In this example, the leaves are all labelled "$\varphi(A) = \mathrm{T}$" since the formula is a tautology.

We still need to describe what "straightforward calculations" can be used at a leaf of the decision tree to determine the value of $A$. Some of the variables of $A$ will have already been determined to have truth value T or F. We let "T" and "F" now stand for a subformula of $A$ whose value has been determined to be either true or false. We permit the following two ways for determining further truth values for subformulas of $A$ (letting $B$ stand for any subformula of $A$):

(a) Any subformula of the form $\neg\mathrm{F}$ or $(\mathrm{T} \vee B)$ or $(B \vee \mathrm{T})$ or $(\mathrm{T} \wedge \mathrm{T})$ or $(\mathrm{F} \to B)$ or $(B \to \mathrm{T})$ or $(\mathrm{F} \leftrightarrow \mathrm{F})$ or $(\mathrm{T} \leftrightarrow \mathrm{T})$ can be determined to have truth value T.

(b) Any subformula of the form $\neg\mathrm{T}$ or $(\mathrm{F} \vee \mathrm{F})$ or $(\mathrm{F} \wedge B)$ or $(B \wedge \mathrm{F})$ or $(\mathrm{T} \to \mathrm{F})$ or $(\mathrm{F} \leftrightarrow \mathrm{T})$ or $(\mathrm{T} \leftrightarrow \mathrm{F})$ can be determined to have truth value F.

If in the end, the truth value of $A$ has been determined, then the current node is a valid leaf for the decision tree. If, however, the truth value of $A$ does not get determined in this way, then further variables need to be queried.

## I.6 Examples of Tautologies and Tautological Equivalences

Here we list some of the more common tautologies and tautological equivalences. These can all be proved with the method of truth tables (or decision trees). They can also be proved using the proof system PL that will be introduced in Chapter II.

**Simple tautologies on a single variable.**

| | |
|---|---|
| $p \vee \neg p$ | - Law of the Excluded Middle |
| $\neg(p \wedge \neg p)$ | - Noncontradiction |
| $p \to p$ | - Self-Implication |
| $p \leftrightarrow p$ | - Self-Equivalence |
| $\neg\neg p \leftrightarrow p$ | - Double Negation |
| $(\neg p \to p) \leftrightarrow p$ | - Equivalence with Implication from Negation |

**Simple tautological equivalences.**

| | |
|---|---|
| $\neg(p \vee q) \models\models (\neg p \wedge \neg q)$ | - De Morgan's Law |
| $\neg(p \wedge q) \models\models (\neg p \vee \neg q)$ | - De Morgan's Law |
| $p \vee p \models\models p$ | - Idempotency of $\vee$ |
| $p \wedge p \models\models p$ | - Idempotency of $\wedge$ |
| $p \wedge q \models\models q \wedge p$ | - Commutativity of $\wedge$ |
| $p \vee q \models\models q \vee p$ | - Commutativity of $\vee$ |
| $p \leftrightarrow q \models\models q \leftrightarrow p$ | - Commutativity of $\leftrightarrow$ |
| $p \wedge (q \wedge r) \models\models (p \wedge q) \wedge r$ | - Associativity of $\wedge$ |
| $p \vee (q \vee r) \models\models (p \vee q) \vee r$ | - Associativity of $\vee$ |
| $p \leftrightarrow (q \leftrightarrow r) \models\models (p \leftrightarrow q) \leftrightarrow r$ | - Associativity of $\leftrightarrow$ |
| $p \wedge (q \vee r) \models\models (p \wedge q) \vee (p \wedge r)$ | - Distributivity of $\wedge$ over $\vee$ |
| $p \vee (q \wedge r) \models\models (p \vee q) \wedge (p \vee r)$ | - Distributivity of $\vee$ over $\wedge$ |
| $p \to q \models\models \neg q \to \neg p$ | - Contrapositive (or Transposition) |
| $p \to (q \to r) \models\models (p \wedge q) \to r$ | - Exportation |

**Some tautologies often used as axioms.** Here we list some tautologies that are also used as axioms for Hilbert-style propositional proof systems. They are stated with explicit variables $p, q, r$; however, generally a Hilbert-style system allows any substitution instance of an axiom to be an axiom.

There are many other possible axioms that have been used in for propositional proof systems; only a few of them are listed below.

Chapter II defines a Hilbert-style proof system PL, that uses (all substitution instances of) the first four tautologies below as its axioms. For the second axiom (among others), be careful to understand how parentheses should be added to make it a fully parenthesized formula!

| | |
|---|---|
| $p \to (q \to p)$ | - An axiom for $\to$. An axiom of PL. |
| $(p \to q \to r) \to (p \to q) \to (p \to r)$ | - An axiom for $\to$. An axiom of PL |
| $\neg p \to (p \to q)$ | - An axiom for $\neg$. An axiom of PL |
| $(\neg p \to p) \to p$ | - An axiom for $\neg$. An axiom of PL |
| $(p \to q) \to (p \to \neg q) \to \neg p$ | - An axiom for $\neg$ |
| $p \wedge q \to p$ | - An axiom for $\wedge$ |
| $p \wedge q \to q$ | - An axiom for $\wedge$ |
| $p \to q \to p \wedge q$ | - An axiom for $\wedge$ |
| $p \to p \vee q$ | - An axiom for $\vee$ |
| $q \to p \vee q$ | - An axiom for $\vee$ |
| $(p \to r) \to (q \to r) \to (p \vee q \to r)$ | - An axiom for $\vee$ |
| $(p \to q) \to (r \vee p) \to (r \vee q)$ | - Principle of Summation (from *Principia Mathematica*) |
| $((p \to q) \to p) \to p$ | - Pierce's Law |

**Tautological equivalences defining $\vee$, $\wedge$ and $\leftrightarrow$.** The following three tautological implications can be used to define $\vee$, $\wedge$ and $\leftrightarrow$ in terms of $\neg$ and $\to$. This corresponds to the fact that $\{\neg, \to\}$ is an adequate set of connectives, as will be discussed later in Section I.7.

| | |
|---|---|
| $p \vee q \models\dashv \neg p \to q$ | - Definition of $\vee$ in PL |
| $p \wedge q \models\dashv \neg(p \to \neg q)$ | - Definition of $\wedge$ in PL |
| $p \leftrightarrow q \models\dashv (p \to q) \wedge (q \to p)$ | - Definition of $\leftrightarrow$ in PL |

**Some tautological implications often used for inferences.** Any valid tautological implication can be used for inferences; thus there are many possible valid inference rules. Some of the common ones include the following:

| | |
|---|---|
| $p,\ p \to q \models q$ | - Modus Ponens |
| $\neg q,\ p \to q \models \neg p$ | - Modus Tollens |
| $p \to q,\ q \to r \models p \to r$ | - Hypothetical Syllogism |

The proof system PL will allow only Modus Ponens as an inference rule. A Modus Ponens inference will be written in the form

$$\frac{A \qquad A \to B}{B}$$

where $A$ and $B$ are permitted to be any well-formed formulas. This inference rule means that if the two formulas $A$ and $A \to B$ have already been inferred, then $B$ may be inferred as well. Similarly, Modus Tollens and Hypothetical Syllogism correspond to the inference rules

$$\frac{A \to B \qquad \neg B}{\neg A} \qquad \text{and} \qquad \frac{A \to B \qquad B \to C}{A \to C}$$

## I.7 Boolean Functions and DNF and CNF

This section will show how Boolean functions can be represented by propositional formulas, and introduce Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF) formulas.

The motivation for looking at Boolean functions is to justify the use of the five connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$ for propositional formulas. As a first observation, it is not really necessary to use all five of these. In fact, we could have used with just the two connectives $\neg$ and $\vee$, and would have still be able to express everything that can be expressed using $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$. Similarly with could have used just $\neg$ and $\wedge$, or just $\neg$ and $\rightarrow$, without any loss of expressiveness. (These facts will be proved later in Theorems I.36 and I.37.) On the other hand, we could not have omitted $\neg$, as there is no way to express negation using only $\wedge, \vee, \rightarrow, \leftrightarrow$. (For this, see Theorem I.38 and Exercise I.20.)

This raises the question of whether it is enough to use only $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$. Could it be that there is some other connective beyond these that cannot be expressed using $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$? The answer this is no: the five connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ can express any Boolean function. To formalize this, we define the notion of a "Boolean function. A $k$-ary Boolean function can be viewed as a type of $k$-ary propositional connective. Our first goal will be to prove that every Boolean function can be represented by a propositional formula.

**Definition I.22.** Let $k \geq 1$. A $k$-ary *Boolean function* $f$ takes as input $k$ true/false values and outputs a true/false value; namely, it is a mapping

$$f : \{\mathrm{T}, \mathrm{F}\}^k \rightarrow \{\mathrm{T}, \mathrm{F}\}.$$

**Example I.23.** The following define three Boolean functions $f_{\neg p_1}$, $f_{p_1 \wedge p_2}$ and $f_{p_1 \oplus p_2 \oplus p_3}$:

$$f_{\neg p_1}(x) = \begin{cases} \mathrm{T} & \text{if } x \text{ equals F} \\ \mathrm{F} & \text{if } x \text{ equal T.} \end{cases}$$

$$f_{p_1 \wedge p_2}(x_1, x_2) = \begin{cases} \mathrm{T} & \text{if } x_1 \text{ and } x_2 \text{ both equal T} \\ \mathrm{F} & \text{otherwise.} \end{cases}$$

$$f_{p_1 \oplus p_2 \oplus p_3}(x_1, x_2, x_3) = \begin{cases} \mathrm{T} & \text{if an odd number of } x_1, x_2, x_3 \text{ equal T} \\ \mathrm{F} & \text{if an even number of } x_1, x_2, x_3 \text{ equal T.} \end{cases}$$

Note that $f_{\neg p_1}$ is unary (1-ary), $f_{p_1 \wedge p_2}$ is binary (2-ary) and $f_{p_1 \oplus p_2 \oplus p_3}$ is 3-ary.

The functions in the example are named $f_A$ where the subscript $A$ is a formula. The third one, $p_1 \oplus p_2 \oplus p_3$, uses the "parity connective" $\oplus$, also called "exclusive or" or "xor" for short. The parity connective will be discussed more in the next section; the notation "$\oplus$" comes from the fact that if T and F are identified with the constants 1 and 0, then $\oplus$ becomes addition mod 2.

**Adequacy of $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$.** The idea is that the formulas $A$ in the subscripts in the previous example "represent" the Boolean function. This is formalized in the next definition.

**Definition I.24.** Let $k \geq 1$ and let $A$ be a propositional formula that uses (at most) the variables $p_1, \ldots, p_k$. The $k$-ary Boolean function $f_A(x_1, x_2, \ldots, x_k)$ is defined by

$$f_A(x_1, \ldots, x_k) \;=\; \varphi(A), \qquad \text{where } \varphi(p_i) = x_i \text{ for } i = 1, 2, \ldots, k.$$

The propositional formula $A$ is said to *represent* the Boolean function $f_A$.

It is important to note that this gives a well-definition of $f_A$, since the values of the truth assignment $\varphi$ on $p_1, \ldots, p_k$ uniquely determine the truth value $\varphi(A)$ of $A$.[3]

From the definition, every propositional formula represents a Boolean function. Conversely, the next theorem shows that every Boolean function is represented by some propositional formula.

**Theorem I.25** (Adequacy of $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$)**.** *Let $f$ be a $k$-ary Boolean function. Then there is a propositional formula $A$ that represents $f$, i.e., with $f_A = f$.*

We first give an example to illustrate the proof idea and then will give the general proof. Consider the Boolean function $f$ defined by

$$f(x_1, x_2) \;=\; \begin{cases} \mathrm{T} & \text{if } x_1 = \mathrm{T} \text{ or } x_2 = \mathrm{F} \\ \mathrm{F} & \text{otherwise.} \end{cases}$$

You might notice that $f$ is the same as $f_{p_2 \rightarrow p_1}$, but let's pretend we do not notice that. The complete list of $f$'s values is:

| $x_1$ | $x_2$ | $f(x_1, x_2)$ |
|---|---|---|
| T | T | **T** |
| T | F | **T** |
| F | T | F |
| F | F | **T** |

There are three lines in the table for $f$ where $f(x_1, x_2) = \mathrm{T}$. This tells us that $f(x_1, x_2)$ is true precisely when $x_1 = x_2 = \mathrm{T}$, when $x_1 = \mathrm{T}$ and $x_2 = \mathrm{F}$, and when $x_1 = x_2 = \mathrm{F}$. In other words, $f(x_1, x_2)$ is represented by the formula

$$(p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2) \vee (\neg p_1 \wedge \neg p_2). \tag{I.2}$$

The first disjunct of I.2 is $p_1 \wedge p_2$; this is true exactly when the first line of the table of $f$'s value is applicable. The second disjunct and third disjuncts similarly correspond to the two other ways that $f$ can be true.

By the way, the formula (I.2) is probably not the "best" formula that represents $f$. For instance, $p_1 \vee \neg p_2$ is probably a better choice. But (I.2) is the formula representing $f$ which comes out of the general proof that we give next.

[3] Strictly speaking, we ought to use the notation $f_{A,k}$ instead of $f_A$ since it is not required that $p_k$ actually appears in $A$. However, we use the simpler notation $f_A$ as this should never cause any confusion.

*Proof of Theorem I.25.* Let $f$ be a $k$-ary Boolean function. If $f(x_1, \ldots, x_k) = \mathrm{F}$ for all inputs $x_1, \ldots, x_k \in \{T, F\}$, then $f$ is represented by the formula $p_1 \wedge \neg p_1$. So, suppose there is at least one input where $f$'s value is T.

Since $f$ has $k$ inputs, each of which has two possible values, there are $2^k$ many distinct inputs to $f$. Let $\varphi_1, \ldots, \varphi_{2^k}$ be the $2^k$ many possible truth assignments to $p_1, \ldots, p_k$. Although it is not important for the proof, it might be helpful to think of $\varphi_1$ as being the truth assignment that sets each $p_i$ to T, and $\varphi_{2^k}$ as being the truth assignment that sets each $p_i$ to F, and other $\varphi_i$'s as being the truth assignments listed in the usual order that lines appear in a truth table.

For $1 \le i \le 2^k$ and $1 \le j \le k$, define the formula $L_{i,j}$ by

$$L_{i,j} \;=\; \begin{cases} p_j & \text{if } \varphi_i(p_j) = \mathrm{T} \\ \neg p_j & \text{if } \varphi_i(p_j) = \mathrm{F}. \end{cases}$$

(The letter "$L$" stands for "literal".) Clearly $L_{i,j}$ is either $p_j$ or $\neg p_j$, and $\varphi_i(L_{i,j}) = \mathrm{T}$. Now define[4]

$$C_i \;=\; \bigwedge_{j=1}^{k} L_{i,j} \;:=\; L_{i,1} \wedge L_{i,2} \wedge \cdots \wedge L_{i,k}.$$

We have $\varphi_i(C_i) = \mathrm{T}$ since $\varphi_i(L_{i,j}) = \mathrm{T}$ for all $j$. Furthermore, if $i \neq i'$, then $\varphi_{i'}(C_i) = \mathrm{F}$. This is because $\varphi_{i'}(p_j) \neq \varphi_i(p_j)$ for at least one value of $j$, and hence $\varphi_{i'}(L_{i',j}) = \mathrm{F}$ for at least one $j$.

Now let $\mathcal{I}$ be the set of values $i$ such that

$$f(\varphi_i(p_1), \varphi_i(p_2), \ldots, \varphi_i(p_k)) \;=\; \mathrm{T}.$$

We can write $\mathcal{I} = \{i_1, i_2, \ldots, i_\ell\}$. Note $\ell \ge 1$ since $f$ is not identically equal to F for all inputs. Then define the formula $A$ to be

$$A \;=\; \bigvee_{m=1}^{\ell} C_m \;:=\; C_{i_1} \vee C_{i_2} \vee \cdots \vee C_{i_\ell}. \tag{I.3}$$

By inspection, $A$ represents the function $f$: this is because the $C_{i_m}$'s and their associated truth assignments $\varphi_{i_m}$ enumerate the cases where $f$ is true and because each $C_i$ is satisfied by a unique $\varphi_i$. This completes the proof Theorem I.25. □

**Disjunction and conjunctive normal forms.** Theorem I.21 showed that every function can be represented by a formula that uses only the connectives $\neg$, $\vee$ and $\wedge$. In fact, it shows that "disjunctive normal form" formulas suffice.

Any formula $A$ expressed in the form

$$A_1 \vee A_2 \vee \cdots \vee A_k,$$

[4]The "big and" notation, $\bigwedge$, is similar to a summation sign ($\sum$): it means the indicated conjunction (and) of the formulas $L_{i,1}$ though $L_{i,k}$.

with arbitrary parenthesization (*not* necessarily associated from right to left) is called a *disjunction*. It is permitted that $k = 1$. Each $A_i$ is called a *disjunct* of $A$. We will use the big or notation $\bigvee_{i=1}^{k} A_i$ to denote $A$; however, this notation is often reserved for the case where the parenthesization is the usual right-to-left order.

Similarly, any formula $B$ expressed in the form

$$B_1 \wedge B_2 \wedge \cdots \wedge B_k,$$

again with arbitrary parenthesization, not necessarily associated from right to left, is called a *conjunction*. Each $B_i$ is called a *conjunct* of $B$. We will use the big and notation $\bigwedge_{i=1}^{k} B_i$ to denote $B$; however, again often with parenthesization in the usual right-to-left order.

**Definition I.26.** A *literal* is a formula of the form $p_i$ or $\neg p_i$. In other words, a literal is a variable or the negation of a variable.

**Definition I.27.** A *conjunction of literals* is a formula of the form[5]

$$L_1 \wedge L_2 \wedge \cdots \wedge L_k,$$

where $k \geq 1$ and each $L_i$ is a literal.

A *disjunctive normal form* formula, or *DNF* formula for short, is any formula of the form

$$C_1 \vee C_2 \vee \cdots \vee C_\ell,$$

where $\ell \geq 1$ and each $C_i$ is a conjunction of literals.

**Definition I.28.** A *clause* is a disjunction of literals, namely a formula of the form

$$L_1 \vee L_2 \vee \cdots \vee L_k,$$

where $k \geq 1$ and each $L_i$ is a literal.

A *conjunctive normal form* formula, or *CNF* formula for short, is any formula of the form

$$C_1 \wedge C_2 \wedge \cdots \wedge C_\ell,$$

where $\ell \geq 1$ and each $C_i$ is a clause.

Theorem I.25 shows that every Boolean function $f$ can be represented by a propositional formula. In fact, what the proof showed is that the propositional formula representing $f$ can be taken to be a DNF formula. That is, we have already proved the next theorem.

**Theorem I.29** (Adequacy of DNF Formulas)**.** *Let $f$ be a $k$-ary Boolean function. Then there is a disjunctive normal form (DNF) formula $A$ that represents $f$, i.e., with $f_A = f$.*

[5]In computer science, a conjunction of literals is sometimes called a "term"; however, we avoid this terminology, since "term" has a different, well-established meaning in first-order logic.

*Proof.* The proof of Theorem I.25 already established this. If $f$ is the constant F function, and then $p_1 \wedge \neg p_1$ represents $f$; this formula is a DNF formula (and also a CNF formula). Otherwise, the formula (I.3) is a DNF formula that represents $f$. □

The same thing holds also for CNF formulas:

**Theorem I.30** (Adequacy of CNF Formulas)**.** *Let $f$ be a $k$-ary Boolean function. Then there is a conjunctive normal form (CNF) formula $A$ that represents $f$, i.e., with $f_A = f$.*

*Proof.* (Sketch) The proof of Theorem I.25 could be reworked to prove this directly, using the lines in the truth table for $f$ where $f$ takes on the value F. Instead of doing this, however, we sketch how to obtain Theorem I.30 as a corollary of Theorem I.29.

Let $f^\neg$ be the $k$-ary Boolean function which is the negation of $f$. That is,

$$f^\neg(x_1, \ldots, x_k) = \begin{cases} \text{T} & \text{if } f(x_1, \ldots, x_k) = \text{F} \\ \text{F} & \text{if } f(x_1, \ldots, x_k) = \text{T} \end{cases}$$

We just proved that $f^\neg$ is represented by a DNF formula $A$. Therefore $f$ is represented by $\neg A$. The formula $\neg A$ is the negation of a disjunction of conjunctions of literals. By De Morgan's law (see Section I.6), the negation of a conjunction of formulas is tautologically equivalent to the disjunction of the negations of the same formulas. Likewise, the negation of a disjunction of formulas is tautologically equivalent to the conjunction of the negations of the same formulas. Furthermore, the negation of a literal is clearly equivalent to a literal (possibly by canceling out two negation signs).

This allows $\neg A$ to be rewritten as a tautologically equivalent CNF formula by using De Morgan's laws to push the negation sign inside the formula. We leave the details to the reader. □

**Corollary I.31.** *Any propositional formula $A$ is tautologically equivalent to some disjunctive normal form (DNF) formula, and also tautologically equivalent to some conjunctive normal form (CNF) formula.*

*Proof.* Let $p_1, \ldots, p_k$ include all the variables in $A$. Let $f_A$ be the $k$-ary Boolean function represented by $A$. By Theorem I.29, $f_A$ is represented by some DNF formula $B$. Then, since $A$ and $B$ represent the same Boolean function, it must be that $\varphi(A) = \varphi(B)$ for all truth assignments. Therefore $A$ and $B$ are tautologically equivalent.

The proof that $A$ is tautologically equivalent to a CNF formula $C$ is identical, except it uses Theorem I.30 to obtain a CNF formula $C$ which defines $f_A$. □

**Example I.32.** Let $f_{p_1 \leftrightarrow p_2}$ be the Boolean functon presented by $p_1 \leftrightarrow p_2$. We have $f(x_1, x_2) = \text{T}$ precisely when either $x_1$ and $x_2$ are both equal to T or $x_1$ and $x_2$ are both equal to F. Therefore, using the construction from the proof of Theorem I.25, $f_{p_1 \leftrightarrow p_2}$ is represented by

$$(p_1 \wedge p_2) \vee (\neg p_1 \wedge \neg p_2). \tag{I.4}$$

Consequently, $p_1 \leftrightarrow p_2$ is tautologically equivalent to this formula (I.4).

Similar reasoning shows that $f^{\neg}_{p_1 \leftrightarrow p_2} = f_{\neg(p_1 \leftrightarrow p_2)}$ is represented by the DNF formula

$$(\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2). \tag{I.5}$$

The negation of this represents $f_{p_1 \leftrightarrow p_2}$. We can use De Morgan's law to obtain a CNF formula representing $f_{p_1 \leftrightarrow p_2}$ as follows:

| | | |
|---|---|---|
| $\neg[(\neg p_1 \wedge p_2) \vee (p_1 \wedge \neg p_2)]$ | | Negation of (I.5) |
| $\models\models$ | $[\neg(\neg p_1 \wedge p_2) \wedge \neg(p_1 \wedge \neg p_2)]$ | De Morgan's Law |
| $\models\models$ | $[(\neg\neg p_1 \vee \neg p_2) \wedge (\neg p_1 \vee \neg\neg p_2)]$ | De Morgan's Law |
| $\models\models$ | $[(p_1 \vee \neg p_2) \wedge (\neg p_1 \vee p_2)]$ | Double Negation |

Thus, $(p_1 \vee \neg p_2) \wedge (\neg p_1 \vee p_2)$ is a CNF formula representing $f_{p_1 \leftrightarrow p_2}$, and therefore is tautologically equivalent to $p_1 \leftrightarrow p_2$.

Theorem I.30 was proved by starting with a DNF formula representing the negated function $f^{\neg}$, adding a negation sign, and using De Morgan's laws to convert it to an equivalent CNF formula. An alternate way to prove this theorem would be to instead start with a DNF formula for $f$ (not for the negated function $f^{\neg}$) and then applying the distributive laws for $\vee$ and $\wedge$ and the law of the excluded middle to convert it to a CNF. As an example, we just saw that $(p_1 \wedge p_2) \vee (\neg p_1 \wedge \neg p_2)$ is a DNF formula representing $f_{p_1 \leftrightarrow p_2}$. This can be converted using distributive laws as follows, using $\top$ as a constant with truth value T:

| | | |
|---|---|---|
| $[p_1 \wedge p_2] \vee (\neg p_1 \wedge \neg p_2)$ | | DNF formula (I.5) |
| $\models\models$ | $([p_1 \wedge p_2] \vee \neg p_1) \wedge ([p_1 \wedge p_2] \vee \neg p_2)$ | Distributive Law |
| $\models\models$ | $(p_1 \vee \neg p_1) \wedge (p_2 \vee \neg p_1) \wedge (p_1 \vee \neg p_2) \wedge (p_2 \vee \neg p_2)$ | Distributive Law |
| $\models\models$ | $\top \wedge (p_2 \vee \neg p_1) \wedge (p_1 \vee \neg p_2) \wedge \top$ | Law of Excluded Middle |
| $\models\models$ | $(p_2 \vee \neg p_1) \wedge (p_1 \vee \neg p_2)$ | $\top \wedge A \models\models A$ |

The final formula is another CNF formula for $f_{p_1 \leftrightarrow p_2}$; it is the same as obtained in the previous example after using commutativity of $\vee$.

**Binary decision trees for non-tautologies.** The example of Figure I.7 gave a reduced decision tree for a tautology. It is also possible to give a decision tree for formulas that are not tautologies; for example, see Figure I.8. Since $A$ is satisfiable but not a tautology, some of the leaves are labeled with $\varphi(A) = \mathrm{T}$ and some with $\varphi(A) = \mathrm{F}$.

It is easy to convert a function with a decision tree of the type shown in Figure I.8 into a DNF formula. We leave the details to Exercise I.40, but the idea is that each branch in the decision tree leading to a leaf labeled with $\varphi(A) = \mathrm{T}$ corresponds to a disjunct in a disjunctive normal form formula tautologically equivalent to $A$.

| $p$ | $q$ | $r$ | $r \vee ((p \wedge \neg q) \wedge \neg r)$ | | |
|---|---|---|---|---|---|
| - | - | T | T | | |
| - | T | F | F | F | F |
| T | F | F | T | T | T |
| F | F | F | F | F | F |

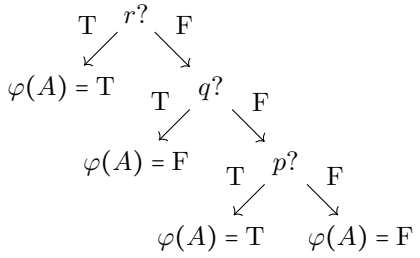



Figure I.8: A reduced truth table and a binary decision tree for a formula that is not a tautology. The formula is tautologically equivalent to the DNF formula $r \vee (\neg r \wedge \neg q \vee p)$.

## I.8 Propositional Languages

**Languages and adequacy.** A set $L$ of propositional connectives is called a *language*. So far, we have worked with the language $L = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$; however, we have already mentioned a few other connectives such as $\oplus$ (parity), $\top$ (the constant T) and $\bot$ (the constant F).

**Definition I.33.** Let $L$ be a (propositional) language. An $L$-formula is a propositional formula that uses only propositional connectives from $L$.

So far in this chapter, "formula" has always meant "$\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow\}$-formula". Chapter II will use a different convention; there "formula" will mean "$\{\neg, \rightarrow\}$-formula".

**Definition I.34.** A language $L$ is *adequate* if every Boolean function is represented by some $L$-formula.

We have already proved a couple of languages are adequate:

**Theorem I.35.** *The languages* $\{\neg, \vee, \wedge, \rightarrow, \leftrightarrow\}$ *and* $\{\neg, \vee, \wedge\}$ *are both adequate.*

*Proof.* The first part of the theorem was already stated as Theorem I.25. The second part is a consequence of Theorem I.29 or I.30. □

Note that if $L_1 \subset L_2$ and $L_1$ is adequate then so is $L_2$. This is immediate from the fact that every $L_1$-formula is an $L_2$-formula.

The last theorem can be improved to:

**Theorem I.36.** *The languages* $\{\neg, \vee\}$ *and* $\{\neg, \wedge\}$ *are both adequate.*

*Proof.* Let $f$ be a Boolean function. We show that $f$ is represented by some $\{\neg, \vee\}$-formula. By Theorem I.30, $f$ is represented by a CNF formula $A$, which is a formula of the form $C_1 \wedge C_2 \wedge \ldots \wedge C_k$ with the $C_i$'s all clauses and hence all $\{\neg, \vee\}$-formulas. Using De Morgan's law, $A$ is tautologically equivalent to $\neg(\neg C_1 \vee \cdots \vee \neg C_k\}$. This is a $\{\neg, \vee\}$-formula which represents $f$.

The fact that $f$ is also represented by some $\{\neg, \wedge\}$-formula is proved similarly. □

**Theorem I.37.** *The language $\{\neg, \to\}$ is adequate.*

*Proof.* Since $\{\neg, \vee\}$ is adequate, it suffices to show that every $\{\neg, \vee\}$-formula $A$ is tautologically equivalent to a $\{\neg, \to\}$-formula. For this, we use the fact that $B \vee C$ is tautologically equivalent to $\neg B \to C$ for any $B$ and $C$.

Specifically, transform the formula $A$ by repeatedly choosing a subformula of the form $(B \vee C)$ and replacing it with $(\neg B \to C)$. Each such replacement removes one $\vee$ connective and adds a $\neg$ connective and a $\to$ connective. Thus, the procedure ends with a $\{\neg, \to\}$-formula that is tautologically equivalent to $A$.[6] □

Now we give examples of a couple of languages that are not adequate.

**Theorem I.38.**
(a) $\{\neg\}$ *is not adequate.*
(b) $\{\wedge, \vee\}$ *is not adequate.*

*Proof.* For (a), note that a $\{\neg\}$-formula $A$ must be of the form $\neg\cdots\neg p_i$, with zero or more $\neg$'s applied to a variable. Since only one variable appears in $A$, it cannot be tautologically equivalent to a formula such as $p_1 \wedge p_2$ that depends on more than one variable. Hence, no $\{\neg\}$-formula can represent $f_{p_1 \wedge p_2}$ for instance.

For (b), let $A$ be any $\{\wedge, \vee\}$-formula. We claim that if $\varphi(p_i) = \mathrm{T}$ for all $i$, then $\varphi(A) = \mathrm{T}$. The claim implies that $A$ cannot represent $f_{\neg p_1}$, since $f_{\neg p_1} = \mathrm{F}$ when $\varphi(p_i) = \mathrm{T}$ for all $i$.

Fix $\varphi$ to be the assignment such that $\varphi(p_1) = \mathrm{T}$ for all $i$. We need to prove the claim that for any $\{\wedge, \vee\}$-formula $A$, $\varphi(A) = \mathrm{T}$. The proof is by induction on the complexity of $A$.

**Base case:** $A$ is a variable $p_i$. Then $\varphi(A) = \mathrm{T}$ by the choice of $\varphi$.

**Induction step:** $A$ is $(B \vee C)$ or $(B \wedge C)$. There are two induction hypotheses, one for $B$ and one for $C$: they state that $\varphi(B) = \mathrm{T}$ and $\varphi(C) = \mathrm{T}$. Therefore, by the definitions of truth for $\vee$ and for $\wedge$, we must have $\varphi(A) = \mathrm{T}$.

That completes the proof of the claim, and of part (b) of the theorem. □

**Other propositional connectives.** There are many other propositional connectives that can be used in addition to the standard ones of $\neg$, $\vee$, $\wedge$, $\to$ and $\leftrightarrow$.

The parity connective, $\oplus$, is defined so that

$$\varphi(A \oplus B) \;=\; \begin{cases} \mathrm{T} & \text{if } \varphi(A) \neq \varphi(B) \\ \mathrm{F} & \text{if } \varphi(A) = \varphi(B) \end{cases}$$

The notation "$\oplus$" reflects the fact that when T and F are identified with 1 and 0 (respectively), then $\oplus$ is the same as addition modulo two. Another name for

[6] See Corollary I.49.

the parity connective $\oplus$ is "exclusive or", since $\varphi(A \oplus B)$ is true if and only if exactly one of $\varphi(A)$ or $\varphi(B)$ is true. Exercise I.26 asks you to show that $A \oplus B \models\!\dashv \neg(A \leftrightarrow B)$.

Another very interesting connective is the *nand* connective, denoted "|". This is also called the *Sheffer stroke* after its discoverer H. Sheffer. "Nand" is short for "not-and". As the name indicates, $p|q$ has truth value equal to the negation of the conjunction of $p$ and $q$, so that for all $\varphi$,

$$\varphi(A|B) \;=\; \varphi(\neg(A \wedge B)).$$

A remarkable aspect of the | connective is that it is adequate all by itself:

**Theorem I.39.** $\{|\}$ *is adequate.*

*Proof.* It is easy to check that $\neg A$ is tautologically equivalent to $A|A$. Furthermore, $A \wedge B$ is tautologically equivalent to $\neg(A|B)$, and hence to $(A|B)|(A|B)$. Thus, any $\{\neg, \wedge\}$-formula can be converted to a tautologically equivalent $\{|\}$-formula by repeatedly replacing subformulas of the form $\neg A$ with $(A|A)$ and subformulas of the form $(A \wedge B)$ with $((A|B)|(A|B))$.

Theorem I.36 showed that $\{\neg, \wedge\}$ is adequate; therefore $\{|\}$ is also adequate. □

Exercise I.24 defines the dual *nor* connective, $\downarrow$, and asks for a proof that $\{\downarrow\}$ is adequate.

Two very simple connectives are the constants $\top$ and $\bot$. These are 0-ary or "nullary" connectives that do not take any arguments. Their truth values are defined by $\varphi(\top) = \mathrm{T}$ and $\varphi(\bot) = \mathrm{F}$.

**Theorem I.40.** $\{\bot, \rightarrow\}$ *is adequate.*

*Proof.* Any negated formula $\neg A$ is tautologically equivalent to $A \rightarrow \bot$. Since $\{\neg, \rightarrow\}$ is adequate (by Theorem I.37), it follows that $\{\bot, \rightarrow\}$ is adequate. □

We have so far introduced examples of 0-ary, 1-ary and 2-ary connectives. It is also possible to have $k$-ary connectives for $k > 2$. One example is the 3-ary connective $\mathit{Case}(\cdot,\cdot,\cdot)$; this can also be called the "if-then-else" connective. The truth value of $\mathit{Case}(A, B, C)$ is defined by

$$\varphi(\,\mathit{Case}(A, B, C)) \;=\; \begin{cases} \varphi(B) & \text{if } \varphi(A) = \mathrm{T} \\ \varphi(C) & \text{if } \varphi(A) = \mathrm{F}. \end{cases}$$

This mean one can think of $\mathit{Case}(A, B, C)$ as meaning the same as "If $A$ then $B$ else $C$". It is easy to check that $\mathit{Case}(p, q, r)$ is tautologically equivalent to the DNF formula $(p \wedge q) \vee (\neg p \wedge r)$. In view of the equivalence with "If $A$ then $B$ else $C$", it is also tautologically equivalent to $(p \rightarrow q) \wedge (\neg p \rightarrow r)$. Hence it is tautologically equivalent to the CNF formula $(\neg p \vee q) \wedge (p \vee r)$.

Exercise I.28 asks you to prove that $\{\neg, \mathit{Case}\}$ is adequate but $\{\mathit{Case}\}$ is not.

An example of a $k$-ary connective for arbitrary fixed $k \geq 1$ is the majority connective $Maj^k$. The semantics for majority connectives is defined by letting $Maj^k(A_1, \ldots, A_k)$ be true exactly when the number of true $A_i$'s is $\geq k/2$. Exercise I.29 deals with the adequacy of $Maj^k$ together with $\neg$.

## I.9 Examples of Proofs by Induction

This section gives some examples of proof by induction on the complexity of formulas. The proof of Theorem I.38 already used a proof by induction. We have in addition already used some "obvious" properties of formulas that, strictly speaking, should have been proved by induction. Most prominently this includes the unique readability property (which was needed in order to give definitions by recursion), as well as the fact that the truth of a formula $A$ only depends on the truth values of the variables that actually appear in $A$. The latter fact was used several times already; e.g. in Definition I.24 and Theorem I.38(a), and even more importantly, to justify the fact that truth tables need only finitely many lines.

We start by proving that the truth of $A$ only depends on the truth values of the variables that appear in $A$.

**Theorem I.41.** *Let $A$ be a propositional formula. Suppose $\varphi$ and $\varphi'$ are truth assignments such that for each $p_i$ that appears in $A$, $\varphi(p_i) = \varphi'(p_i)$. Then $\varphi(A) = \varphi'(A)$.*

Another way to state the hypothesis is that $\varphi(p_1) \neq \varphi'(p_i)$ only if $p_i$ does not appear in $A$.

*Proof.* We use induction on the complexity of $A$.

**Base case:** The base case is where $A$ is a variable $p_i$. Of course $p_i$ appears in $A$, so $\varphi(A) = \varphi'(A)$ by the assumption on $\varphi$ and $\varphi'$.

**Induction step #1:** Suppose $A$ is $\neg B$. Now any variable appearing in $B$ also appears in $A$. Thus $\varphi$ and $\varphi'$ have the same values for all variables in $B$. Therefore the induction hypothesis tells us that $\varphi(B) = \varphi'(B)$. The definition of truth for $\neg$ thus implies that $\varphi(A) = \varphi'(A)$.

**Induction step #2:** Suppose $A$ is $B \circ C$ where $\circ$ is one of the binary connectives $\vee$, $\wedge$, $\rightarrow$ or $\leftrightarrow$. Now any variable appearing in either $B$ or $C$ also appears in $A$. Thus $\varphi$ and $\varphi'$ have the same values for all variables in $B$ and all variables in $C$. There are two induction hypotheses: the first states that $\varphi(B) = \varphi'(B)$ and the second states that $\varphi(C) = \varphi'(C)$. The definition of truth for $\circ$ thus implies that $\varphi(A) = \varphi'(A)$. □

Notice how the base case and the induction steps of the proof by induction correspond exactly to the cases in the inductive definition of propositional

formulas in Definition I.1. The proof uses only two induction cases: the second induction case manages to roll four separate cases into one argument by considering $\circ$ to be any of the four connectives $\vee$, $\wedge$, $\rightarrow$ or $\leftrightarrow$.

The next theorem gives another simple example of a proof by induction.

**Theorem I.42.** *Let $A$ be a formula. The number of open parentheses appearing in $A$ is equal to the number of close parentheses appearing in $A$.*

This theorem is obvious of course, just because the inductive definition of propositional formulas paired up open and close parentheses. A formal proof uses induction.

*Proof.* We will use $m_A$ and $n_A$ to denote the numbers of open parentheses and close parentheses in $A$ (respectively). The proof proceeds by induction.

**Base case:** Suppose $A$ is $p_i$. There are no parentheses and $m_A = 0 = n_A$.

**Induction step #1:** Suppose $A$ is $\neg B$. The induction hypothesis is that $m_B = n_B$. Of course $A$ has the same numbers of open and close parentheses as $B$. Therefore $m_A = n_A$.

**Induction step #2:** Suppose $A$ is $(B \circ C)$ where $\circ$ is one of $\vee$, $\wedge$, $\rightarrow$ or $\leftrightarrow$. There are two induction hypotheses: that $m_B = n_B$ and that $m_C = n_C$. There is one more pair of open and close parentheses in $A$ than in $B$ and $C$ combined. Thus, $m_A = m_B + m_C + 1$ and $n_A = n_B + n_C + 1$; whence, $m_A = n_A$. □

The next theorem is less trivial, and it is also the basis for proving the unique readability property. Recall that a formula $A$ is an expression, namely a string of symbols. A *non-empty, proper initial subexpression of $A$* means any prefix of the string of symbols in $A$ that contains at least one symbol but is not all of $A$. For example, "(", "($p_1$", "($p_1\wedge$" and "($p_1 \wedge p_2$" are all of the non-empty proper initial subexpressions of "($p_1 \wedge p_2$)".

**Theorem I.43.** *Let $A$ be a formula in which the first symbol is an open parenthesis. Let $B$ be a non-empty, proper initial subexpression of $A$. Then $B$ contains more open parentheses than close parentheses. Consequently, $B$ is not a propositional formula.*

Exercises I.36-I.38 ask for proofs of Theorem I.43 and unique readability.

A final example of a theorem that can be proved by induction is:

**Theorem I.44.** *Let $A$ be a formula. Suppose $A$ has $m$ occurrences of binary connectives and $n$ occurrences of propositional variables. Prove that $n = m + 1$.*

Exercise I.39 asks for a proof of this theorem.

## I.10 Propositional Substitution

Propositional substitution means substituting a formula $B$ for a variable $p_i$ inside some formula $A$. In other words, it means replacing a variable $p$ in $A$ with a formula $B$. There are several reasons to look carefully at substitution. First of all, we need to prove that if $A$ is a tautology, then so is the formula obtained by substituting $B$ for every occurrence of $p$ in $A$. This will be important for the proof system PL defined in the next chapter. Second, we need to prove that if $B$ and $C$ are tautologically equivalent, and we replace an occurrence of $B$ as a subformula of $A$ with $C$, then the result is tautologically equivalent to $A$. This is a fact we have already used more than once, notably in Section I.8 when discussing adequate sets of connectives. For example, the proof of Theorem I.37 replaced subformulas $B \vee C$ with $\neg B \to C$, and argued that this preserved tautological equivalence. A third reason is that studying the substitution of propositional formulas for variables will give a glimpse of the issues we will encounter with substitution in first-order logic in later chapters.

We use the notation $A(B/p)$ to mean the formula that results from $A$ after replacing every occurrence of the variable $p$ in $A$ with the formula $B$. More generally, we write $A(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$ to denote the result of replacing, in parallel, each occurrence of each $p_i$ in $A$ with $B_i$. The formal definition is below, but first we consider a few examples.

**Example I.45.** Let $A$ be the formula $(p_1 \to (p_2 \to p_1))$. Letting $B$ be $((p_2 \wedge p_3) \to p_1)$, we have $A(((p_2 \wedge p_3) \to p_1)/p_1)$ is the formula

$$(((p_2 \wedge p_3) \to p_1) \to (p_2 \to ((p_2 \wedge p_3) \to p_1))).$$

As this example shows, it is permitted that $p_1$ appears in $B$.

For another example, $A(p_1/p_1)$ is the same as $A$.

Finally, let $C$ be $(p_2 \wedge p_3)$ and $D$ be $(p_1 \wedge p_4)$. Then $A(C, D/p_1, p_2)$ is the formula

$$((p_2 \wedge p_3) \to ((p_1 \wedge p_4) \to (p_2 \wedge p_3))).$$

Note that the two $p_1$'s in $A$ were replaced with $C$ and the $p_2$ in $A$ was replaced with $D$. But the $p_1$ and $p_2$ that appear in $C$ and $D$ do not change. This reflects the fact that the substitution of $C$ and $D$ for occurrences of $p_1$ and $p_2$ is done "in parallel".

If substitutions are to be done sequentially instead of parallel, we would use the notation $A(C/p_1)(D/p_2)$. This would mean first substituting $C$ for $p_1$ in $A$ and then substituting $D$ for $p_2$ in the resulting formula. You should convince yourself that this is the same $A(C'/p_1, D/p_2)$ where $C'$ is $C(D/p_2)$.

**Definition I.46.** Let $k \geq 1$ and $A, B_1, \ldots, B_k$ be propositional formulas and $p_{i_1}, \ldots, p_{i_k}$ be (distinct) propositional variables. Then the formula $A(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$ is recursively defined as follows:

(a) If $A$ is the variable $p_{i_j}$ for some $1 \leq j \leq k$, then $A(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$ is $B_j$.

(b) If $A$ is a variable $p_\ell$ with $\ell \notin \{i_1, \ldots, i_k\}$, then $A(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$ is $p_\ell$. In other words, $A$ is unchanged by the substitution.

(c) If $A$ is $\neg C$, then $A(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$ is $\neg C(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$. In other words, it is $\neg C'$ where $C'$ is $C(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$.

(d) If $A$ is $C \circ D$ for $\circ$ a binary connective, then $A(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$ is $C(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k}) \circ D(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$. In other words, it is $C' \circ D'$ where $C'$ is as before and $D'$ is $D(B_1, \ldots, B_k/p_{i_1}, \ldots, p_{i_k})$.

The reader is warned that sometimes different notations are used for substitution. For example, some authors will introduce a formula $A$ as $A = A(p)$, and then write $A(B)$ to denote what we mean by $A(B/p)$. This notation $A(B)$ has the advantage of being less cluttered and more readable, but it has the disadvantage of being a little ambiguous about whether $A(B)$ means the result of replacing all of the occurrences of $p$ with $B$ or just some of the occurrences. For the time being, we will stick with the notation $A(B/p)$, and it always means replacing all of the occurrences of $p$.

**Definition I.47.** Any formula of the form $A(B/p_i)$ is called an *instance* of $A$.

The next theorem states that if $B$ and $C$ have the same truth value, then the formulas $A(B/p)$ and $A(C/p)$ have the same truth value. It is a straightforward result, but is the crucial fact needed to prove the rest of our theorems on substitution of propositional formulas.

**Theorem I.48.** *Let $A$, $B$ and $C$ be formulas, and $p_i$ be a variable. Also, let $\varphi$ be a truth assignment. If $\varphi(B) = \varphi(C)$, then $\varphi(A(B/p_i)) = \varphi(A(C/p_i))$.*

*Proof.* Let $B$, $C$, $p_i$ and $\varphi$ be fixed. We prove that the theorem holds for all $A$, by using induction on $A$.

**Base case #1:** Suppose $A$ is the propositional variable $p_i$. Then $A(B/p_i)$ and $A(C/p_i)$ are just $B$ and $C$. So by the hypothesis that $\varphi(B) = \varphi(C)$, we have immediately $\varphi(A(B/p_i)) = \varphi(A(C/p_i))$.

**Base case #2:** Suppose $A$ is a variable $p_j$, with $j \neq i$. Then $A(B/p_i)$ and $A(C/p_i)$ are both just $A$, so of course $\varphi(A(B/p_i)) = \varphi(A(C/p_i))$.

**Induction step #1:** Suppose $A$ is $\neg D$, so $A(B/p_i)$ is $\neg D(B/p_i)$ and $A(C/p_i)$ is $\neg D(C/p_i)$. The induction hypothesis states that $\varphi(D(B/p_i)) = \varphi(D(C/p_i))$. By the definition of truth for $\neg$, we have immediately that $\varphi(A(B/p_i)) = \varphi(A(C/p_i))$.

**Induction step #2:** Suppose $A$ is $D \circ E$ where $\circ$ is one of $\wedge, \vee, \rightarrow, \leftrightarrow$. Thus $A(B/p_i)$ is $D(B/p_i) \circ E(B/p_i)$ and $A(C/p_i)$ is $D(C/p_i) \circ E(C/p_i)$. There are two induction hypotheses: that $\varphi(D(B/p_i)) = \varphi(D(C/p_i))$ and that $\varphi(E(B/p_i)) = \varphi(E(C/p_i))$. By the definition of truth for $\circ$, $\varphi(A(B/p_i)) = \varphi(A(C/p_i))$. $\square$

Theorem I.48 was stated relative to a single truth assignment. If we strengthen the hypothesis to say that $\varphi(B) = \varphi(C)$ for *all* truth assignments we get the following corollary.

**Corollary I.49.** *Suppose* $B \models\!\dashv C$. *Then* $A(B/p_i) \models\!\dashv A(C/p_i)$.

This corollary has already been used earlier, for instance in the proof of Theorem I.37. In that proof, we asserted that if a subformula $(B \vee C)$ of $A$ is replaced with $(\neg B \to C)$, then the resulting formula $A'$ is tautologically equivalent to $A$. Of course, we have so far only discussed substituting a formula for a *variable*, not for a subformula. Nonetheless, we can recast substitution so as to apply it to replacing a subformula with another formula.

Namely, let $p_\ell$ be a variable that does not appear in $A$, and let $A^*$ be the formula obtained from $A$ by replacing the subformula $(B \vee C)$ with $p_\ell$. Then $A$ is the same as $A^*((B \vee C)/p_\ell)$ and $A'$ is the same as $A^*((\neg B \to C)/p_\ell)$. The corollary states that $A^*((B \vee C)/p_\ell)$ and $A^*((\neg B \to C)/p_\ell)$ are tautologically equivalent. That is, that $A$ and $A'$ are tautologically equivalent. This is exactly what was needed for the proof of Theorem I.37.

The above corollary talked about substituting two tautologically equivalent formulas into the same formula $A$. The next theorem talks about substituting $A$ into two tautologically equivalent formulas.

**Theorem I.50.** *Suppose* $B \models\!\dashv C$. *Then* $B(A/p_i) \models\!\dashv C(A/p_i)$.

*Proof.* Let $\varphi$ be an arbitrary truth assignment. We need to show that $\varphi(B(A/p_i)) = \varphi(C(A/p_i))$. Let $p_\ell$ be a variable that does not appear in any of $A$, $B$ or $C$. Define $\varphi'$ to be the truth assignment such that $\varphi(p_\ell) = \varphi(A)$ and otherwise agrees with $\varphi$. In other words,

$$\varphi'(p_j) \;=\; \begin{cases} \varphi(A) & \text{if } j = \ell \\ \varphi(p_j) & \text{if } j \neq \ell. \end{cases} \tag{I.6}$$

We call $\varphi'$ a "$p_\ell$-variant" of $\varphi$ since $\varphi$ and $\varphi'$ agree on all variables except possibly $p_\ell$. Note that $\varphi$ and $\varphi'$ agree on all variables that appear in $B(A/p_i)$ or in $C(A/p_i)$. We have

$$\begin{array}{rcll} \varphi(B(A/p_i)) & = & \varphi'(B(A/p_i)) & \text{By Theorem I.41 since } p_\ell \text{ does not appear in } B(A/p_i) \\ & = & \varphi'(B(p_\ell/p_i)) & \text{By Theorem I.48 since } \varphi'(A) = \varphi'(p_\ell) \\ & = & \varphi'(C(p_\ell/p_i)) & \text{By } B \models\!\dashv C \text{ since } p_\ell \text{ does not appear in } B \text{ or } C \text{ [7]} \\ & = & \varphi'(C(A/p_i)) & \text{By Theorem I.48 again} \\ & = & \varphi(C(A/p_i)) & \text{By Theorem I.41 again.} \quad \square \end{array}$$

[7] Informally, $B(p_\ell/p_i)$ and $C(p_\ell/p_i)$ are obtained from $B$ and $C$ by renaming $p_i$ to $p_\ell$, and therefore $B \models\!\dashv C$ implies $B(p_\ell/p_i) \models\!\dashv C(p_\ell/p_i)$. Strictly speaking, we should introduce another truth assignment $\varphi''$ which is the $p_i$-variant of $\varphi'$ with $\varphi''(p_i) = \varphi'(p_\ell)$. Then we argue $\varphi'(B(p_\ell/p_i)) = \varphi''(B) = \varphi''(C) = \varphi'(C(p_\ell/p_i))$.

**Corollary I.51.**

(a) *Any instance of a tautology is a tautology.*
(b) *If* $A \vDash B$*, then* $A(C/p_i) \vDash B(C/p_i)$
(c) *If* $A_1, \ldots, A_k \vDash B$*, then* $A_1(C/p_i), \ldots A_k(C/p_i) \vDash B(C/p_i)$.

*Proof.* Let $A$ be a tautology, and $A(B/p_i)$ be an instance of $A$. To prove (a), we must show that $A(B/p_i)$ is a tautology. Let $p_\ell$ be a variable that does not occur in $A$. We have $A \vDash\dashv (p_\ell \vee \neg p_\ell)$ since both $A$ and $p_\ell \vee \neg p_\ell$ are tautologies. Therefore, by the previous theorem, substituting $B$ for $p_i$,

$$A(B/p_i) \vDash\dashv (p_\ell \vee \neg p_\ell),$$

since $p_i$ does not appear in $(p_\ell \vee \neg p_\ell)$. Therefore, $A(B/p_i)$ is a tautology.

Part (b) follows easily from part (a). First, $A \vDash B$ holds iff and only if $\vDash A \to B$. Likewise, $A(C/p_i) \vDash B(C/p_i)$ holds if and only if $\vDash A(C/p_i) \to B(C/p_i)$. Finally, $A(C/p_i) \to B(C/p_i)$ is equal to $(A \to B)(C/p_i)$. Thus (a) implies (b).

Part (c) is proved similarly to part (b) using the fact that $A_1, \ldots, A_k \vDash B$ holds if and only if $A_1 \wedge \cdots \wedge A_k \to B$ is a tautology. □

## Exercises

**Exercise I.1.** Which of the following are true statements? Assume that the statements such as "cats have wings" have their usual true/false value. Use the "logical"/"mathematical" interpretation of the connectives such as "if", "if-then", and "only if".

(a) Cats have wings only if dogs have wings.
(b) Parrots have wings only if dogs have wings.
(c) Parrots have wings if dogs have wings.
(d) If dogs have wings then parrots have wings.
(e) If cats have wings and dogs have wings, then cats have wings or dogs have wings.
(f) If cats have wings or dogs have wings, then cats have wings and dogs have wings.

Then, rewrite (a)-(c) as statements in the form "If ···, then ···."

**Exercise I.2.** Recall the formulas $e \to c$ and $c \to e$ discussed on page 8. Explain what the difference is between the meanings of $e \to c$ and $c \to e$. Is it possible for $e \to c$ to be true when $c \to e$ is false? If so, how?

**Exercise I.3.** Characterize each of the following sentences as being either true or false. Use the "logical"/"mathematical" interpretation of the connectives

such as "if-then", "only if", "unless", etc.

(a) If porpoises have wings then seagulls can fly.
(b) Porpoises have wings if seagulls can fly.
(c) Porpoises have wings only if seagulls can fly.
(d) Porpoises have wings if and only if seagulls can fly.
(e) Porpoises have wings unless seagulls can fly.
(f) Porpoises have wings but seagulls can fly.
(g) If porpoises have wings then seagulls cannot fly.
(h) Porpoises have wings if seagulls cannot fly.
(i) Porpoises have wings only if seagulls cannot fly.
(j) Porpoises have wings if and only if seagulls cannot fly.
(k) Porpoises have wings unless seagulls cannot fly.
(l) Porpoises have wings but seagulls cannot fly.

**Exercise I.4.** Use the variables

$p$ - "Propoises have wings"
$s$ - "Seagulls can fly"

to reexpress the English sentences (a)-(l) of Exercise I.3 as propositional formulas.

**Exercise I.5.** Use the variables

$b$ - "The mirror will break"
$g$ - "The mirror is made of glass"
$s$ - "The mirror is made of silver"
$w$ - "The mirror is high on the wall"

to reexpress the sentences (a)-(g) as propositional formulas. You do not need to include all the parentheses required in the formal definition of formulas. However, as always, you should use parentheses when necessary to clarify the meaning of the propositional formulas.

(a) The mirror will break if it is made of glass.
(b) The mirror will break only if is made of glass.
(c) If the mirror is high on the wall, it will break.
(d) Unless it is made of glass, the mirror will not break.
(e) The mirror is made of either glass or silver, but not both.
(f) The mirror will break unless it is made of silver or glass.
(g) The mirror is high on the wall, but it will not break unless it is made of glass.

Note the placement of the comma in sentence (g).

**Exercise I.6.** Use the variables

$f$ - "The coffee pot falls"
$b$ - "The coffee pot breaks"
$g$ - "The coffee pot is made of glass"
$t$ - "The coffee pot is made of titanium"

and propositional connectives $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ to express the sentences (a)-(g) as propositional formulas. When necessary, use parentheses to clarify the meaning of the propositional formulas. (You do not need to include all the parentheses required in the formal definition of formulas.)

(a) The coffee pot breaks only if it is made of glass.
(b) The coffee pot breaks if it is made of glass and falls.
(c) The coffee pot is made of titanium or glass, but not both.
(d) The coffee pot falls if and only if it is made of glass.
(e) The coffee pot falls but does not break.
(f) The coffee does not break unless it is made of glass.
(g) The coffee pot falls, but it does not break unless it is made of glass.

Again, watch the placement of the comma in (g).

**Exercise I.7.** Give the actual formulas (with correct parenthesization) corresponding to the following informal abbreviations. If the informal abbreviation is already an actual formula, please state this.

(a) $\neg p_1 \rightarrow p_2 \rightarrow \neg p_3$.
(b) $p_1 \vee p_3 \rightarrow \neg p_1 \rightarrow p_4 \rightarrow \neg p_1$.
(c) $p_1 \vee p_3 \leftrightarrow \neg p_1 \wedge p_4 \leftrightarrow \neg p_1$.
(d) $\neg p_1 \vee p_2 \vee \neg p_3$.
(e) $(p_1 \rightarrow p_2) \rightarrow p_3$.
(f) $(\neg p_1) \rightarrow p_2$.
(g) $\neg p_1 \rightarrow p_2$.
(h) $\neg(p_1 \rightarrow p_2)$.

**Exercise I.8.** Show that $\not\models (p \rightarrow q) \rightarrow (q \rightarrow p)$.

**Exercise I.9.** For each formula (a)-(e) on the left, which formula or formulas on the right is it tautologically equivalent to?

(a) $p \oplus q$
(b) $\neg(p \wedge q)$
(c) $p \wedge q \rightarrow p \vee q$
(d) $p \vee q \rightarrow p \wedge q$
(e) $(q \rightarrow p) \wedge (\neg p \rightarrow q)$

(i) $p \leftrightarrow q$
(ii) $p \leftrightarrow \neg q$
(iii) $p \leftrightarrow p$
(iv) $q \leftrightarrow p \leftrightarrow q$
(v) None of the above.

**Exercise I.10.** Use truth tables or reduced truth tables to prove the two De Morgan's laws:

$$\begin{array}{lll} \neg(p \vee q) & \models\dashv & (\neg p \wedge \neg q) \\ \neg(p \wedge q) & \models\dashv & (\neg p \vee \neg q). \end{array}$$

**Exercise I.11.** Use reduced truth tables to prove the two distributivity laws:

$$\begin{array}{lll} p \wedge (q \vee r) & \models\dashv & (p \wedge q) \vee (p \wedge r) \\ p \vee (q \wedge r) & \models\dashv & (p \vee q) \wedge (p \vee r). \end{array}$$

**Exercise I.12.** Show $\models (A \rightarrow B) \rightarrow (\neg A \rightarrow B) \rightarrow B$, where $A$ and $B$ are arbitrary propositional formulas.

**Exercise I.13.** Use the method of truth tables to show that $\leftrightarrow$ is associative.

**Exercise I.14.** (Non-distributivity for $\wedge$ over $\leftrightarrow$.) For each of the following either prove it is true or give a truth assignment $\varphi$ which shows it is false. (To prove it is true, you may either argue informally, or use truth tables or reduced truth tables.)

(a) $p \wedge (q \leftrightarrow r) \vDash (p \wedge q) \leftrightarrow (p \wedge r)$.
(b) $(p \wedge q) \leftrightarrow (p \wedge r) \vDash p \wedge (q \leftrightarrow r)$.

**Exercise I.15.** (Non-distributivity for $\leftrightarrow$ over $\wedge$.) For each of the following either prove it is true or give a truth assignment $\varphi$ which shows it is false. (To prove it is true, you may either argue informally, or use truth tables or reduced truth tables.)

(a) $p \leftrightarrow (q \wedge r) \vDash (p \leftrightarrow q) \wedge (p \leftrightarrow r)$.
(b) $(p \leftrightarrow q) \wedge (p \leftrightarrow r) \vDash p \leftrightarrow (q \wedge r)$.

**Exercise I.16.** Let $\Gamma$ be a set of formulas.

(a) Suppose $\Gamma$ is finite. Prove that there exists a formula $A$ such that $\Gamma \vDash A$ and $A \vDash \Gamma$. (The latter means that for all $B \in \Gamma$, $A \vDash B$.)
(b) Give an example of an infinite, satisfiable $\Gamma$ for which there does not exist a satisfiable formula $A$ such that $A \vDash \Gamma$.

**Exercise I.17.** Is there a formula that uses at least three variables and is in *both* a CNF formula and a DNF formula? If so, give an example. If not, explain why not.

**Exercise I.18.** Let $A$ be the formula $(p \to q) \wedge (q \to r)$.

(a) Give a CNF formula that is tautologically equivalent to $A$.
(b) Give a DNF formula that is tautologically equivalent to $A$.

**Exercise I.19.** Now let $A$ be the formula $p \leftrightarrow (q \to r \wedge p)$.

(a) Give a CNF formula that is tautologically equivalent to $A$.
(b) Give a DNF formula that is tautologically equivalent to $A$.

**Exercise I.20.** Recall the function $f_{p_1 \oplus p_2 \oplus p_3}$ from Example I.23.

(a) Give a DNF formula that represents $f_{p_1 \oplus p_2 \oplus p_3}$.
(b) Give a CNF formula that represents $f_{p_1 \oplus p_2 \oplus p_3}$.

**Exercise I.21.** Let $f$ be the Boolean function defined by $f(x_1, x_2, x_3, x_4) = \mathrm{T}$ if and only if at least three of $x_1, x_2, x_3, x_4$ are equal to T. Give a propositional formula that represents $f$.

**Exercise I.22.** Prove that $\{\neg, \to\}$ is adequate.

**Exercise I.23.** Prove that $\{\vee, \wedge, \to, \leftrightarrow\}$ is not adequate.

**Exercise I.24.** The binary "nor" connective is denoted "$\downarrow$". Its truth is defined so that $\varphi(A \downarrow B)$ is equal to $\varphi(\neg(A \vee B))$. ("Nor" means "not-or".) Show that $\{\downarrow\}$ is adequate. Give $\{\downarrow\}$-formulas which are tautologically equivalent to $\neg p$, to $p \vee q$, and to $p \wedge q$.

**Exercise I.25.** Consider a formula

$$p_1 \leftrightarrow p_2 \leftrightarrow p_3 \leftrightarrow \cdots \leftrightarrow p_k.$$

Give a simple characterization of when $\varphi(p_1 \leftrightarrow p_2 \leftrightarrow p_3 \leftrightarrow \cdots \leftrightarrow p_k) = \mathrm{T}$. Your answer should be in terms of the number of $i$'s such that $\varphi(p_i)$ is equal to T or F. [Hint: Recall that $\leftrightarrow$ is associative.]

**Exercise I.26.** Use the method of truth tables to show that $\neg(p_1 \oplus p_2)$ is tautologically equivalent to $p_1 \leftrightarrow p_2$.

**Exercise I.27.**

(a) Show that $\{\oplus, \leftrightarrow\}$ is not adequate.
(a) Show that $\{\neg, \oplus, \leftrightarrow\}$ is not adequate.
(b) Show that $\{\rightarrow, \oplus\}$ is adequate.

**Exercise I.28.** Recall that *Case* is the 3-ary "if-then-else" connective.

(a) Show that $\{Case\}$ is not adequate.
(b) Show that $\{\neg, Case\}$ is adequate.

**Exercise I.29.** Fix $k \geq 2$. Recall $Maj^k$ is the $k$-ary majority connective.

(a) Prove that $\{Maj^k\}$ is not adequate.
(b) Suppose that $k$ is even. Prove that $\{\neg, Maj^k\}$ is adequate. [Hint: Try the $k = 2$ case first. What Boolean function is represented by $Maj^2(p_1, p_2)$?]
(c) Suppose that $k$ is odd. Prove that $\{\neg, Maj^k\}$ is not adequate.

**Exercise I.30.** Theorem I.39 and Exercise I.24 showed that the singleton sets $\{|\}$ and $\{\downarrow\}$ (that is, "nand" and "nor") are both adequate. Prove that there is no other binary connective $\circ$ such that $\{\circ\}$ is adequate. (See also Exercise I.35.)

**Exercise I.31.** Prove that $\{Maj^3, \oplus\}$ is not an adequate set of propositional connectives.

**Exercise I.32.** (a) Let $\circledast$ be a new binary connective defined by letting the truth value of $p \circledast q$ be the same as the truth value of $Case(p, p|q, p \rightarrow q)$. Is $\{\circledast\}$ adequate? If so, give $\{\circledast\}$-formulas that are tautologically equivalent to $\neg p$ and to $p \vee q$. If not, explain why it is not adequate.

(b) Let $\odot$ be a new binary connective defined by letting the truth value of $p \odot q$ be the same as the truth value of $Case(p, p|q, p \downarrow q)$. Is $\{\odot\}$ adequate? If so, give $\{\odot\}$-formulas that are tautologically equivalent to $\neg p$ and to $p \vee q$. If not, explain why it is not adequate.

(c) Let $f$ be a new 3-ary binary connective defined by letting the truth value of $f(p, q, r)$ be the same as the truth value of $Case(p, p \rightarrow q, q \rightarrow r)$. Is $\{f\}$ adequate? If so, give $\{f\}$-formulas that are tautologically equivalent to $\neg p$ and to $p \vee q$. If not, explain why it is not adequate.

**Exercise I.33.** Let $A$ be $p_1 \to (p_1 \vee p_2)$ and let $B$ be $p_1 \vee p_3$. Write out the following formulas explicitly:

(a) $A(B/p_1)$.
(b) $A(B/p_2)$.
(c) $A(B/p_3)$.
(d) $B(A/p_1)$.
(e) $B(A/p_2)$.
(f) $B(A/p_3)$.
(g) $B(A, B/p_1, p_3)$.

**Exercise I.34.** Prove the two following assertions. Your answers should use proof by induction on the complexity of formulas.

(a) If $A$ is an $\{\wedge, \vee, \to, \leftrightarrow\}$-formula then $A$ is satisfiable.
(b) If $A$ is an $\{\wedge, \vee\}$-formula, then $A$ is not a tautology.

**Exercise I.35.** There are two nullary (0-ary) Boolean functions, represented by $\top$ and $\bot$ (the constants T and F). There are four unary (1-ary) Boolean functions; namely, the two constant functions T and F (also represented by $\top$ and $\bot$), the identify function $f_{p_1}$, and the negation function $f_{\neg p_1}$. How many binary (2-ary) Boolean functions are there? In general, how many $k$-ary Boolean functions are there? Justify your answers. [Hint: Consider how many lines there are in a truth table over $k$ variables.]

**Exercise I.36.** Prove Theorem I.43. Suppose that $A$ is a formula with first symbol a parenthesis. Prove that any non-empty, proper initial subexpression of $A$ has more open parentheses than close parentheses.

[A suggested way to work this problem is to prove the following using induction on the complexity of $A$. Suppose that $A$ is a formula — not necessarily starting with an open parenthesis. Also suppose $B$ is a non-empty, proper initial subexpression of $A$. Prove that either $B$ is all $\neg$ signs or $B$ has more open parentheses than close parentheses.]

**Exercise I.37.** Suppose $A$ is a formula, and $B$ is a non-empty, proper initial subexpression of $A$. Prove that $B$ is not a formula. [Suggestion: use induction on the length of formulas to handle the case where $A$ and $B$ start with a negation symbol ($\neg$); use the previous exercise to handle the case where they start with a parenthesis.]

**Exercise I.38.** (Unique Readability.) Work with fully parenthesized formulas as specified in the inductive definition of propositional formulas.

(a) Suppose that a formula $A$ can be written in both the form $(B \circ C)$ and the form $(D \circ' E)$ where $B, C, D, E$ are formulas, and $\circ$ and $\circ'$ are binary connectives. Prove that these two forms must be identical, with $B = D$ and $C = E$ and $\circ = \circ'$. [Hint: One of $B$ or $D$ must be a non-empty initial subexpression of the other. Then use the previous exercise.]
(b) Prove the unique readability property for propositional formulas.

**Exercise I.39.** Prove Theorem I.44 about the numbers of binary connectives and propositional variables appearing in a formula.

**Exercise I.40.** (Converting a decision tree to a DNF or CNF formula.) Suppose you are given a decision tree $\mathcal{T}$. It represents a Boolean function $f$. Describe an easy method that uses $\mathcal{T}$ to construct a DNF formula that represents the same $f$. As a hint, consider the paths that lead from the root of $\mathcal{T}$ to leaves labeled with "T".

Also, describe an easy method to construct a CNF formula that represents $f$. For this, you should consider paths from the root to leaves labelled "F".

# Chapter II

# Propositional Logic: Proofs

## II.1 Introduction to Propositional Proofs

This chapter introduces a Hilbert-style proof system for propositional logic called PL. The method of truth tables is already a proof system, but PL has the advantage of providing a formal proof system based on step-by-step reasoning starting with axioms or other assumptions and using Modus Ponens as an inference rule. The Modus Ponens rule is written as

$$\frac{A \qquad A \to B}{B}$$

and allows the formula $B$ to be inferred provided that the two formulas $A$ and $A \to B$ have already been inferred. This allows PL to model (to a certain extent) how humans construct proofs.

The introduction of the proof system PL is a first step towards the development of *metamathematics*. "Metamathematics" means the use of mathematical tools, especially the use of mathematical logic, to study the formalization of mathematics itself. The definition of PL-proofs provides a mathematical definition of proofs; this allows us to treat proofs as mathematical objects in their own right, and even to prove theorems about proofs. First-order logic will give a much more meaningful treatment of metamathematics, but propositional logic and PL-proofs already illustrate many of the key concepts.

The purpose of a propositional proof is to establish that some formula is a tautology, or that some tautological implication holds. We will write $\vdash A$ to denote that $A$ has a propositional proof, and will write $B_1, \ldots, B_k \vdash A$ to denote that $A$ has a PL-proof from the hypotheses $B_1, \ldots, B_k$. Thus the single turnstile sign "$\vdash$" is used for provability, in contrast to the double turnstile notation "$\models$" for tautological validity/implication.

There are many, many proof systems, even for propositional logic. When choosing a proof system, there are several desirable properties to keep in mind. Note, however, that not all of these objectives can be fully achieved.

(1) ***Algorithmic***. Proofs will be strings of symbols (also called "expressions") with specified syntactic properties. There should be an algorithm, which given a string $w$ of symbols, determines whether $w$ is a valid proof, and if so, what formula it proves, or what tautological implication it proves. We have not yet formally defined the concept of "algorithm", but informally, this means any procedure that can be carried out by an appropriate (idealized) computer program.

(2) ***Soundness***. A formula $A$ should have a proof only if it is a tautology. Similarly, if $A$ can be proved from the hypotheses $B_1, \ldots, B_k$, then the tautological implication $B_1, \ldots, B_k \vDash A$ should hold.

(3) ***Completeness***. Conversely, any tautology $A$ should have a proof. Similarly, if $B_1, \ldots, B_k \vDash A$ holds, then there should be a proof of $A$ from the hypotheses $B_1, \ldots, B_k$.[1]
Taken together, the soundness and completeness properties mean that $\vdash A$ holds if and only if $\vDash A$ holds. The same holds for tautological implications as well.

(4) ***User-friendly***. This could also be called ***Human-centric***. There are several aspects to this. (i) A proof system should be able to simulate human reasoning efficiently. (ii) In particular, it should permit reasoning using step-by-step inferences. (iii) And, proofs should be understandable to humans without too much effort.

(5) ***Elegance***. A proof system should be mathematically elegant, and without an excessively large number of axioms or rules.

(6) ***Efficient proof search***. There should be efficient, practical algorithms for searching for and constructing proofs.
Unfortunately, it is open problem (related to the P versus NP question) whether truly efficient proof search is always possible. Nonetheless, some proof systems are better than others at allowing efficient proof search. This is a large and active area of ongoing research, and the state of the art allows finding proofs for at least some very large propositional formulas, even many formulas with 100,000's or millions of variables!

The method of truth tables can serve as a propositional proof system. For this, we just need to establish a canonical way to code an entire truth table as a string of symbols. Truth tables certainly meet criteria (1)-(3) above. They fail criteria (6) since truth tables are generally exponentially large. They also fail criteria (4) and (5).

The Hilbert-style proof system PL meets well the criteria (1)-(5), except for that condition (4.iii) of being readily comprehensible. It is an open question

[1] Completeness is sometimes called "Adequacy', e.g. by Hodel [10], but we follow here the common convention and call it "completeness".

whether PL satisfies criteria (6) of allowing efficient proof search; in fact, it is generally conjectured that proof search can be very hard for PL, requiring exponential time in the worst case.

## II.2 The Proof System PL

We now define the propositional proof system PL. This is a so-called "Hilbert-style" proof system.[2]

In the interests of making PL as simple as possible, PL uses only $\{\neg, \rightarrow\}$-formulas. Other connectives such $\vee$, $\wedge$ and $\leftrightarrow$ are abbreviations for equivalent formulas that use only $\{\neg, \rightarrow\}$. Specifically,

| | | |
|---|---|---|
| $A \vee B$ | is an abbreviation for | $\neg A \rightarrow B$ |
| $A \wedge B$ | is an abbreviation for | $\neg(A \rightarrow \neg B)$ |
| $A \leftrightarrow B$ | is an abbreviation for | $(A \rightarrow B) \wedge (B \rightarrow A)$ |

PL-proofs start with axioms or other hypotheses and infer new formulas with Modus Ponens. Four types of axioms are permitted in PL-proofs.

PL **axioms.** The axioms of PL are all formulas of the forms:[3]

**PL1:** $A \rightarrow (B \rightarrow A)$

**PL2:** $[A \rightarrow (B \rightarrow C)] \rightarrow [(A \rightarrow B) \rightarrow (A \rightarrow C)]$

**PL3:** $\neg A \rightarrow (A \rightarrow B)$

**PL4:** $(\neg A \rightarrow A) \rightarrow A$

where $A$, $B$ and $C$ may be any $\{\neg, \rightarrow\}$-formulas.

**Modus Ponens.** The only inference rule for PL is Modus Ponens; namely, from $A$ and $A \rightarrow B$, it is permitted to infer $B$. This is denoted as

$$\frac{A \qquad A \rightarrow B}{\text{B}}$$

Here $A$ and $B$ may be any $\{\neg, \rightarrow\}$-formulas.

[2]Hilbert-style systems are named after David Hilbert, one of the founding fathers of mathematical logic, as this style of proof was used in D. Hilbert and W. Ackermann's 1928 book *Grundzüge der theoretischen Logic* (*Foundations of Mathematical Logic*). A system similar to a Hilbert-style system was already used by A.N. Whitehead and B. Russell in 1910 in their classic *Principia Mathematica*. Even earlier, G. Frege used a precursor to Hilbert-style proof systems in his 1879 *Begriffsschrift*. All three of these were landmark developments in establishing the modern study of mathematical logic.

The Hilbert-style system PL that we use is taken from Hodel [10], who named it L.

[3]The square brackets in PL2 are the same as usual parentheses and are used just to improve readability.

**Definition II.1.** Let $A$ be a $\{\neg, \to\}$-formula. A PL-*proof* of $A$ is a sequence of $\{\neg, \to\}$-formulas

$$A_1, A_2, A_3, \ldots, A_\ell$$

such that $A_\ell$ is $A$ and such that each $A_i$ satisfies one of the following conditions:

(a) $A_i$ is a PL-axiom; or

(b) $A_i$ is *inferred by Modus Ponens* from two earlier formulas $A_j$ and $A_k$ with $j, k < i$. Specifically we have $A_k$ equal to $A_j \to A_i$, so the Modus Ponens inference has the form

$$\frac{A_j \qquad A_j \to A_i}{A_i}$$

The final formula $A_\ell$ is the *conclusion* of the PL-proof and is the formula that is proved.

More generally, we can use a set $\Gamma$ of formulas as extra hypotheses in a PL-proof that can be used freely in the proof.

**Definition II.2.** A PL-*proof of $A$ from the hypotheses* $\Gamma$ is a sequence of $\{\neg, \to\}$-formulas $A_1, A_2, \ldots, A_\ell$ such that $A_\ell$ is $A$ and such that each $A_i$ satisfies one of the following conditions:

(a) $A_i$ is a PL-axiom;

(b) $A_i$ is in $\Gamma$; or

(c) $A_i$ is *inferred by Modus Ponens* from two earlier formulas $A_j$ and $A_k$ with $j, k < i$.

PL-proofs are also sometimes called PL-*derivations*. We write $\vdash A$ to denote that $A$ has a PL-proof; in this case we say that $A$ is a *theorem*. Similarly, we write $\Gamma \vdash A$ to denote that $A$ has a PL-proof from the hypotheses $\Gamma$; we then say that $A$ is a *theorem of* $\Gamma$. Note that it is permitted that $\Gamma$ is infinite; however, a single proof can contain only finitely many formulas, and thus can use only finitely many hypotheses from $\Gamma$.

When $\Gamma$ is a finite set $\{B_1, \ldots, B_k\}$, we usually omit the set braces and write just $B_1, \ldots, B_k \vdash A$. We also abuse notation by writing things like $\Gamma, A \vdash B$ instead of $\Gamma \cup \{A\} \vdash B$.

**Theorem II.3.**

(a) *If* $\Gamma \vdash A$ *and* $\Gamma' \supset \Gamma$*, then* $\Gamma' \vdash A$.

(b) *If* $\Gamma \vdash A$*, then there is a finite subset* $\Gamma_0 \subseteq \Gamma$ *such that* $\Gamma_0 \vdash A$.

*Proof.* Part (a) is immediate from the definition of $\Gamma \vdash A$. Part (b) follows from the observation that a proof can contain only finitely many formulas. □

To simplify notation, we assume that all formulas are $\{\neg, \to\}$-formulas for the rest of the chapter.

**Example II.4.** There is a PL-proof of $(\neg A \to A) \to (\neg A \to B)$, where $A$ and $B$ are any formulas. The PL-proof consists of the following three formulas:

| | |
|---|---|
| 1. $\neg A \to A \to B$ | Axiom PL3 |
| 2. $(\neg A \to A \to B) \to (\neg A \to A) \to (\neg A \to B)$ | Axiom PL2 |
| 3. $(\neg A \to A) \to (\neg A \to B)$ | Modus Ponens, 1, 2 |

The PL2 axiom above is formed by letting $A$, $B$ and $C$ in the definition of the PL2 axiom be the formulas $\neg A$, $A$ and $B$, respectively.

**Example II.5.** There is a PL-proof of $A \to A$, where $A$ is any formula. That is, $\vdash A \to A$. The PL-proof consists of the following five formulas:

| | |
|---|---|
| 1. $(A \to ((A \to A) \to A)) \to (A \to (A \to A)) \to (A \to A)$ | Axiom PL2 |
| 2. $A \to ((A \to A) \to A)$ | Axiom PL1 |
| 3. $(A \to (A \to A)) \to (A \to A)$ | Modus Ponens, 1, 2 |
| 4. $A \to (A \to A)$ | Axiom PL1 |
| 5. $A \to A$ | Modus Ponens, 3, 4 |

The PL2 axiom above is formed by letting $B$ be the formula $A \to A$, and letting $C$ be the formula $A$. The first PL1 axiom is obtained by letting $B$ be $A \to A$; the second is formed by letting $B$ be $A$.

**Example II.6.** For any $A$ and $B$, the formulas $B \to (A \vee B)$ and $\neg\neg A \to (A \vee B)$ have PL-proofs. To see this, recall that a formula $C \vee D$ is an abbreviation for $\neg C \to D$. Thus the two formulas are actually equal to $B \to (\neg A \to B)$ and $\neg\neg A \to (\neg A \to B)$. The first is an instance of the axiom PL1; the second is an instance of the axiom PL3. Therefore they each have a PL-proof consisting a single formula, namely an axiom.

Exercise II.11 asks you to show that $A \to (A \vee B)$ has a PL-proof.

**Example II.7.** The formula $A \to B$ has a PL-proof from the hypothesis $B$. That is, $B \vdash A \to B$. The PL-proof has three formulas:

| | |
|---|---|
| $B \to (A \to B)$ | Axiom PL1 |
| $B$ | Hypothesis |
| $A \to B$ | Modus Ponens |

**Example II.8.** Any formula $A$ has a proof from itself as a hypothesis; that is, $A \vdash A$. The PL proof consists of just the single formula $A$.

**Theorem II.9.** *The set of theorems (of* PL*) is closed under substitution. In other words, if $\vdash A$, then $\vdash A(B/p_i)$ for any formulas $A$ and $B$ and any variable $p_i$.*

*Proof.* The set of axioms of PL is clearly closed under substitution. Also, the valid Modus Ponens inferences are closed under substitution, since

$$\frac{D(B/p_i) \qquad (D \to E)(B/p_i)}{E(B/p_i)} \quad \text{is the same as} \quad \frac{D(B/p_i) \qquad D(B/p_i) \to E(B/p_i)}{E(B/p_i)}$$

It follows that if $A_1$, $A_2$, $A_3, \ldots$, $A_\ell$ is a PL-proof, then so is

$$A_1(B/p_i),\ A_2(B/p_i),\ A_3(B/p_i), \ldots,\ A_\ell(B/p_i).$$

This is proved by induction on the steps in the PL-proof. □

## II.3 Deduction Theorem

The Deduction Theorem is our first main tool for proving the existence of PL-proofs. It is the analogue for provability of Theorem I.16, which stated that $\Gamma \vDash A \to B$ is equivalent to $\Gamma, A \vDash B$.

The intuition behind the Deduction Theorem is that proving the implication $A \to B$ is equivalent to assuming that $A$ holds and proving $B$ with the aid of the hypothesis $A$.

**Theorem II.10** (Deduction Theorem)**.** $\Gamma, A \vdash B$ *if and only if* $\Gamma \vdash A \to B$.

*Proof.* The "if" direction is very easy. Suppose $\Gamma \vdash A \to B$. Then also $\Gamma, A \vdash A \to B$. Since $\Gamma, A \vdash A$, Modus Ponens, gives $\Gamma, A \vdash B$.

The "only if" direction is the important direction. Suppose that $\Gamma, A \vdash B$ and that

$$C_1, C_2, C_3, \ldots, C_\ell$$

is a PL-proof of $B$ from $\Gamma, A$ so that $C_\ell$ is the formula $B$. Each $C_i$ is an axiom, a member of $\Gamma$ or the formula $A$, or is inferred by Modus Ponens.

We prove by induction on $i$ that $\Gamma \vdash A \to C_i$. The proof by induction splits into three cases.

**Case 1:** Suppose that $C_i$ is either an axiom or a member of $\Gamma$. Then, certainly $\Gamma \vdash C_i$. Also, $C_i \to (A \to C_i)$ is an instance of Axiom PL1. Thus, by Modus Ponens, $\Gamma \vdash A \to C_i$, as desired.

**Case 2:** Suppose $C_i$ is $A$. By Example II.5, $\Gamma \vdash A \to A$, which is the same as $\Gamma \vdash A \to C_i$, as desired.

**Case 3:** Suppose $C_i$ is inferred from Modus Ponens from $C_j$ and $C_k$ with $j, k < i$. Without loss of generality, $C_k$ is equal to $C_j \to C_i$. The induction hypotheses for $C_j$ and $C_k$ are that $A \to C_j$ and $C \to C_k$ are theorems of $\Gamma$. We prove that there is a proof of $A \to C_i$ from $\Gamma$ as follows:

| | |
|---|---|
| $\Gamma \vdash A \to C_j$ | Induction hypothesis |
| $\Gamma \vdash A \to C_j \to C_i$ | Induction hypothesis |
| $\Gamma \vdash (A \to C_j \to C_i) \to (A \to C_j) \to (A \to C_i)$ | Axiom PL2 |
| $\Gamma \vdash (A \to C_j) \to (A \to C_i)$ | Modus Ponens |
| $\Gamma \vdash A \to C_i$ | Modus Ponens |

That completes the proof by induction and, since $A \to C_\ell$ is the same as $A \to B$, it also completes the proof of the Deduction Theorem. □

Note how Axiom PL2 is exactly what is needed to make the argument work easily in Case 3 above.

Theorem II.9 and the Deduction Theorem are some first examples of "meta-mathematics", that is the use of mathematical tools to study mathematical objects such as theorems and proofs. We have given formal definitions of "theorems" and "proofs" as mathematical objects, and now Theorem II.9 and the

Deduction Theorem are theorems that state properties about (PL-)theorems; their proofs use induction on a given PL-proof to prove the existence of another PL-proof.

**Hypothetical Syllogism.** As an example of how to use the Deduction Theorem, we show that the inference rule of Hypothetical Syllogism,

$$\frac{A \to B \qquad B \to C}{A \to C}\,, \tag{II.1}$$

is a derived rule of inference for PL.

**Theorem II.11.** *For $A$, $B$ and $C$ any formulas,*

$$\vdash (A \to B) \to ((B \to C) \to (A \to C)).$$

*Proof.* We use "$\Leftrightarrow$" to mean "if and only if". Applying the Deduction Theorem three times gives the equivalences

$$\begin{array}{lll} \vdash (A \to B) \to ((B \to C) \to (A \to C)) & & \\ \quad \Leftrightarrow \quad A \to B \vdash (B \to C) \to (A \to C) & \text{Deduction Theorem} \\ \quad \Leftrightarrow \quad A \to B,\ B \to C \vdash A \to C & \text{Deduction Theorem} \\ \quad \Leftrightarrow \quad A \to B,\ B \to C, A \vdash C & \text{Deduction Theorem} \end{array}$$

And, $A \to B,\ B \to C, A \vdash C$ holds by combining the three hypotheses with two uses of Modus Ponens to prove $C$. □

**Corollary II.12.** *If $\Gamma \vDash A \to B$ and $\Gamma \vDash B \to C$, then $\Gamma \vDash A \to C$.*

The corollary implies that the Hypothetical Syllogism inference rule is admissible in PL-proofs as a "derived rule" of inference.[4] That is, although Modus Ponens in the only allowed rule of inference for PL-proofs, we may nonetheless allow Hypothetical Syllogism inferences when proving the existence of a PL-proof. This is because Hypothetic Syllogism can be simulated by multiple steps in a PL-proof. Consequently, if Hypothetical Syllogism were to be added to PL as an additional rule of inference, it would not make the proof system PL any stronger.

## II.4 Consistency, Inconsistency, and Proof by Contradiction

**Consistency and inconsistency.** A set $\Gamma$ is said to be "inconsistent" if it is possible to derive a contradiction from $\Gamma$. Otherwise, is it "consistent". Formally, these are defined as follows:

**Definition II.13.** A set $\Gamma$ of formulas is *inconsistent* if, for some formula $A$, both $A$ and $\neg A$ are theorems of $\Gamma$. In other words, if both $\Gamma \vdash A$ and $\Gamma \vdash \neg A$. Otherwise, $\Gamma$ is *consistent*.

[4] Derived rules of inference are sometimes called "admissible rules" of inference.

**Example II.14.** As a simple example, the set $\Gamma = \{\neg p_1 \to p_1, \neg\neg p_1 \to \neg p_1\}$ is inconsistent. To see this, first note that the PL3 axiom $(\neg p_1 \to p_1) \to p_1$ and Modus Ponens can be used to show that $\Gamma \vdash p_1$. Similarly, the PL3 axiom $(\neg\neg p_1 \to \neg p_1) \to \neg p_1$ means that $\Gamma \vdash \neg p_1$. Thus $\Gamma$ is inconsistent.

**Theorem II.15.** *If $\Gamma$ is consistent and $\Pi \subseteq \Gamma$, then $\Pi$ is consistent.*

*Proof.* If $\Pi$ is inconsistent, there are proofs of $A$ and $\neg A$ from the hypotheses $\Pi$. These are also proofs from the hypotheses $\Gamma$. □

It is a remarkable, but simple, fact that if $\Gamma$ is inconsistent, then it can be used to prove *any* formula:

**Theorem II.16.** *$\Gamma$ is inconsistent if and only if $\Gamma \vdash B$ for every formula $B$.*

*Proof.* Certainly, if $\Gamma \vdash B$ for all formulas $B$, then $\Gamma \vdash p_1$ and $\Gamma \vdash \neg p_1$ and hence $\Gamma$ is inconsistent.

Conversely, suppose $\Gamma$ is inconsistent and thus $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ for some formula $A$. Let $B$ be any formula. Now, $\neg A \to (A \to B)$ is an instance of Axiom PL3. From this, with two uses of Modus Ponens, we conclude that $\Gamma \vdash B$. □

**Corollary II.17.** *Let $A$ and $B$ be formulas. Then $A, \neg A \vdash B$.*

Note that the corollary is the analogue of Theorem I.14(a) for provability in place of tautological implication. It is an immediate consequence of Theorem II.16. In fact, the last two sentences of the proof of Theorem II.16 were just a proof of the corollary. □

**Theorem II.18.** *If $\Gamma$ is inconsistent, then there is a finite subset $\Gamma_0$ of $\Gamma$ which is inconsistent.*

*Proof.* Suppose $\Gamma \vdash A$ and $\Gamma \vdash \neg A$. By Theorem II.3(b) there are finite subsets $\Gamma_1$ and $\Gamma_2$ of $\Gamma$ such that $\Gamma_1 \vdash A$ and $\Gamma_2 \vdash \neg A$. Let $\Gamma_0 = \Gamma_1 \cup \Gamma_2$. Then $\Gamma_0 \subseteq \Gamma$ and both $A$ and $\neg A$ are theorems of $\Gamma_0$. That is, $\Gamma_0$ is a finite, inconsistent subset of $\Gamma$. □

The converse to Theorem II.18 also holds, and its proof is immediate from Theorem II.15. That is, $\Gamma$ is inconsistent if and only if some finite subset of $\Gamma$ is inconsistent. For more on this, see Section II.8 on the Compactness Theorem.

**Proof by contradiction.** Proof by contradiction is a powerful technique for finding and constructing proofs. The idea is that to prove a formula $A$, it is sufficient to assume that $A$ is false and obtain a contradiction.

**Theorem II.19** (Proof by Contradiction, First Version)**.**

$$\Gamma \vdash A \qquad \textit{if and only if} \qquad \Gamma \cup \{\neg A\} \textit{ is inconsistent.}$$

*Proof.* First, suppose $\Gamma \vdash A$. Then clearly, from the hypotheses $\Gamma \cup \{\neg A\}$ there are proofs of both $A$ and $\neg A$. Therefore $\Gamma \cup \{\neg A\}$ is inconsistent.

Now suppose $\Gamma \cup \{\neg A\}$ is inconsistent. By Theorem II.16, $\Gamma \cup \{\neg A\}$ has a proof of every formula; in particular, it has a proof of $A$. Now,

| | | | |
|---|---|---|---|
| $\Gamma, \neg A \vdash A$ | | | |
| | $\Leftrightarrow$ | $\Gamma \vdash \neg A \to A$ | Deduction Theorem |
| | $\Rightarrow$ | $\Gamma \vdash A$ | By PL4 axiom $(\neg A \to A) \to A$ and Modus Ponens |

Hence $\Gamma \vdash A$. □

**Corollary II.20.** $\vdash \neg\neg A \to A$.

*Proof.* By the Deduction Theorem and Theorem II.19, $\vdash \neg\neg A \to A$ holds if and only if $\{\neg\neg A, \neg A\}$ is inconsistent. But this is obvious since both $\neg A$ and $\neg\neg A$ have proofs from the hypotheses $\{\neg\neg A, \neg A\}$. □

The second version of the Proof by Contradiction theorem interchanges the roles of $A$ and $\neg A$.

**Theorem II.21** (Proof by Contradiction, Second Version)**.**

$$\Gamma \vdash \neg A \qquad \textit{if and only if} \qquad \Gamma \cup \{A\} \textit{ is inconsistent.}$$

*Proof.* First, suppose $\Gamma \vdash \neg A$. Then $\Gamma \cup \{A\}$ has both $A$ and $\neg A$ as theorems and thus is inconsistent.

Second, suppose $\Gamma \cup \{A\}$ is inconsistent. We have

| | | | |
|---|---|---|---|
| $\Gamma \cup \{A\}$ is inconsistent | | | |
| | $\Rightarrow$ | $\Gamma, A \vdash \neg A$ | Theorem II.16 |
| | $\Leftrightarrow$ | $\Gamma \vdash A \to \neg A$ | Deduction Theorem |
| | $\Rightarrow$ | $\Gamma \vdash \neg\neg A \to \neg A$ | By $\vdash \neg\neg A \to A$ (Corollary II.20) and Hypothetical Syllogism |
| | $\Rightarrow$ | $\Gamma \vdash \neg A$ | By PL4 axiom $(\neg\neg A \to \neg A) \to \neg A$ and Modus Ponens |

Thus we have $\Gamma \vdash \neg A$ and the theorem is proved. □

**Corollary II.22.** $\vdash A \to \neg\neg A$.

*Proof.* By the Deduction Theorem, it suffices to prove that $A \vdash \neg\neg A$. By the previous theorem, $A \vdash \neg\neg A$ holds if and only if $\{A, \neg A\}$ is inconsistent. But this is obvious since both $A$ and $\neg A$ have proofs from the hypotheses $\{A, \neg A\}$. □

**Example II.23.** We show that $\{A \wedge \neg A\}$ is inconsistent. Recalling the convention that $\wedge$ is used to abbreviate a $\{\neg, \to\}$-formula, this is identical to showing that $\{\neg(A \to \neg\neg A)\}$ is inconsistent. By the first version of Proof by Contradiction (Theorem II.19), this is equivalent to $\vdash (A \to \neg\neg A)$. And this is true by Corollary II.22.

**Modus Tollens.** The Modus Tollens inference rule has the form

$$\frac{A \to B \qquad \neg B}{\neg A}$$

This turns out to be a valid derived rule for PL. Its validity is justified by the following.

**Theorem II.24.** *For any formulas $A$ and $B$, $\vdash (A \to B) \to (\neg B \to \neg A)$.*

*Proof.* We have

| | | |
|---|---|---|
| $\vdash (A \to B) \to (\neg B \to \neg A)$ | | |
| $\Leftrightarrow$ | $A \to B, \neg B \vdash \neg A$ | Deduction Theorem (used twice) |
| $\Leftrightarrow$ | $\{A \to B, \neg B, A\}$ is inconsistent | Proof by Contradiction (2nd version) |
| $\Leftrightarrow$ | $A \to B, A \vdash B$ | Proof by Contradiction (1st version) |

The final assertion holds by Modus Ponens. □

**Corollary II.25.** *If $\Gamma \vdash A \to B$ and $\Gamma \vdash \neg B$, then $\Gamma \vdash \neg A$.*

**Proof by Cases.** The technique of "proof-by-cases" allows proving a formula $B$ by splitting into two cases depending on whether another formula $A$ is true or false. Specifically, it means proving both $A \to B$ and $\neg A \to B$. Since at least one of $A$ and $\neg A$ must be true (for any particular truth assignment), this suffices to prove $B$.

**Theorem II.26.** *Suppose $\Gamma \vdash A \to B$ and $\Gamma \vdash \neg A \to B$. Then $\Gamma \vdash B$.*

*Proof.* To prove $\Gamma \vdash B$, it suffices to prove that $\Gamma \cup \{\neg B\}$ is inconsistent. Since $\Gamma \vdash A \to B$, we have $\Gamma, \neg B \vdash \neg A$ by Modus Tollens. Similarly, since $\Gamma \vdash \neg A \to B$, we have $\Gamma, \neg B \vdash \neg\neg A$. Since both $\neg A$ and $\neg\neg A$ are theorems of $\Gamma \cup \{\neg B\}$, the set $\Gamma \cup \{\neg B\}$ is inconsistent. □

**Corollary II.27.** *Suppose $\Gamma, A \vdash B$ and $\Gamma, \neg A \vdash B$. Then $\Gamma \vdash B$.*

An important special case of the proof-by-cases principle is Corollary II.29 which states that a consistent $\Gamma$ can be extended at least one of $\Gamma \cup \{A\}$ or $\Gamma \cup \{\neg A\}$ and remain consistent. This will be a corollary of the next theorem.

**Theorem II.28.** *$\Gamma$ is inconsistent if and only both $\Gamma \cup \{A\}$ and $\Gamma \cup \{\neg A\}$ are inconsistent.*

*Proof.* By Theorem II.15, if $\Gamma$ is inconsistent, then so are both $\Gamma \cup \{A\}$ and $\Gamma \cup \{\neg A\}$.

So suppose that both of $\Gamma \cup \{A\}$ and $\Gamma \cup \{\neg A\}$ are inconsistent. Let $B$ be an arbitrary formula. We want to show $\Gamma \vdash B$, as that will establish that *all* formulas are theorems of $\Gamma$. We have that both $\Gamma \cup \{A\} \vdash B$ and $\Gamma \cup \{\neg A\} \vdash B$ since those two theories are presumed to be inconsistent. Thus, by proof-by-cases (Corollary II.27), $\Gamma \vdash B$. □

**Corollary II.29.** *Suppose* $\Gamma$ *is consistent. Then at least one of* $\Gamma \cup \{A\}$ *and* $\Gamma \cup \{\neg A\}$ *is consistent.*

For another example of how to use proof-by-cases, we prove the following.

**Theorem II.30.**
(a) $\Gamma, (A \to B) \vdash C$ *if and only if both* $\Gamma, \neg A \vdash C$ *and* $\Gamma, B, \vdash C$.
(b) $\Gamma, (A \to B)$ *is inconsistent if and only if both* $\Gamma \cup \{\neg A\}$ *and* $\Gamma \cup \{B\}$ *are inconsistent.*

*Proof.* Since a set is inconsistent if and only if it has all formulas $C$ as theorems, part (a) implies part (b). So we prove (a). First suppose $\Gamma, A \to B \vdash C$. Recall that $\neg A \to (A \to B)$ is an axiom PL3. Thus $\Gamma, \neg A \vdash A \to B$. It follows readily that $\Gamma, \neg A \vdash C$. Also recall that $B \to (A \to B)$ is an axiom PL1. By similar reasoning, it follows readily again that $\Gamma, B \vdash C$.

Now suppose that both $\Gamma, \neg A \vdash C$ and $\Gamma, B, \vdash C$. To prove (a), we use proof-by-cases, using the formula $A$ for the cases. Namely, it suffices to prove that

(i) $\Gamma, (A \to B), A \vdash C$, and
(ii) $\Gamma, (A \to B), \neg A \vdash C$

Item (ii) is obvious, since by hypothesis even $\Gamma, \neg A \vdash C$. To establish (i), use Modus Ponens to derive $B$ from $A \to B$ and $A$. Then use the assumption that $\Gamma, B \vdash C$. □

Proof-by-cases can be a powerful technique for proving the existence of PL-proofs, sometimes greatly simplifying the proofs. However, it can be tricky to choose the formula to use for defining the two cases. The proof of Theorem II.30 used $A$ and $\neg A$ for the two cases of the proof-by-cases argument. It would have also worked well to use $B$ and $\neg B$ for the two cases of the proof-by-cases argument. However, sometimes it can be very difficult to find a good formula for defining the two cases.

## II.5 Constructing PL-Proofs

The previous two sections have given a large variety of tools for proving statements such as "$\Gamma \vdash A$" or "$\Gamma$ is inconsistent". The ones that tend to be the most useful in practice include the following (1)-(5) and (6)-(9).

(1) To prove $\Gamma \vdash A \to B$, it suffices to prove $\Gamma, A \vdash B$.
(2) To prove $\Gamma \vdash \neg A$, it suffices to prove $\Gamma \cup \{A\}$ is inconsistent.
(3) To prove $\Gamma \vdash p_i$, it suffices to prove that $\Gamma \cup \{\neg p_i\}$ is inconsistent.
(4) To prove $\Gamma, \neg A$ is inconsistent, it suffices to prove that $\Gamma \vdash A$.
(5) To prove $\Gamma \cup \{A \to B\}$ is inconsistent, it suffices to prove first that $\Gamma \vdash A$ and second that $\Gamma \cup \{B\}$ is inconsistent.

Item (5) can be justified by Theorem II.30(b).

It can be shown that repeatedly applying the reductions of items (1)-(5) is sufficient to generate a proof of any $\{\neg, \to\}$-tautology. For this, see Exercise II.29. Some additional principles that are also very useful in practice include:

(6) To prove $\Gamma \vdash A$, it suffices to prove that $\Gamma \cup \{\neg A\}$ is inconsistent. (This is a more general form of item (3), the first version of proof by contradiction.)

(7) To prove $\Gamma \vdash B$, it suffices to prove both that $\Gamma, A \vdash B$ and that $\Gamma, \neg A \vdash B$. (This is proof-by-cases.)

(8) Suppose $\Gamma$ contains the two formulas $A$ and $A \to B$. Let $\Gamma^*$ be $\Gamma \cup \{B\}$. To prove that $\Gamma$ is inconsistent, it suffices to prove that $\Gamma^*$ is inconsistent. To prove that $\Gamma \vdash C$, it suffices to prove that $\Gamma^* \vdash C$. (This is justified by Modus Ponens.)

(9) Suppose $\Gamma$ contains the two formulas $\neg B$ and $A \to B$. Let $\Gamma^*$ be $\Gamma \cup \{\neg A\}$. To prove that $\Gamma$ is inconsistent, it suffices to prove that $\Gamma^*$ is inconsistent. To prove that $\Gamma \vdash C$, it suffices to prove that $\Gamma^* \vdash C$. (This is justified by Modus Tollens.)

(10) Suppose $\Gamma$ contains the two formulas $A \to B$ and $B \to C$. Let $\Gamma^*$ be $\Gamma \cup \{A \to C\}$. To prove that $\Gamma$ is inconsistent, it suffices to prove that $\Gamma^*$ is inconsistent. To prove that $\Gamma \vdash D$, it suffices to prove that $\Gamma^* \vdash D$. (This is justified by Hypothetical Syllogism.)

Of course, the previous sections gave several other possible principles that can help with proving the existence of PL-proofs. For example, a formula $\neg\neg A$ may be replaced with the formula $A$. Nonetheless, items (1)-(10) are often (but not always) good strategies to follow. There will generally be multiple possibilities for which of (1)-(10) to apply; it can require some cleverness (or luck) to decide whether and how to best apply items (1)-(10), or other strategies.

Item (7) can be much harder to use than the other eight items since it requires choosing the formula $A$. For this, it helps to have some intuition on what formula $A$ could help with showing the existence of a PL-proof.

Items (8) and (9) are still valid if $\Gamma^*$ is instead defined to omit the formula $A \to B$.

A chart of the main definitions and theorems that can be used for proving the existence of PL-proof is shown on page 57.

**Example II.31.** We show that

$$A \to B \vee C,\ B \to D,\ C \to D \vdash A \to D.$$

Since $B \vee C$ is an abbreviation for $\neg B \to C$, this is identical to

$$A \to \neg B \to C,\ B \to D,\ C \to D \vdash A \to D.$$

By item (1), or the Deduction Theorem, it suffices to show

$$A,\ A \to \neg B \to C,\ B \to D,\ C \to D \vdash D.$$

**Constructing PL-proofs**

**Logical axioms:** PL1: $A \to (B \to A)$
PL2: $[A \to (B \to C)] \to [(A \to B) \to (A \to C)]$
PL3: $\neg A \to (A \to B)$
PL4: $(\neg A \to A) \to A$

**Modus Ponens:** $\dfrac{A \to B \qquad A}{B}$

**Abbreviations:** $A \vee B$, $A \wedge B$ and $A \leftrightarrow B$ abbreviate, respectively, $\neg A \to B$, $\neg(A \to \neg B)$ and $(A \to B) \wedge (B \to A)$.

**Deduction Theorem:** $\Gamma \vdash A \to B$ iff $\Gamma, A \vdash B$.

**Proof by Contradiction:**

$\Gamma \vdash A$ iff $\Gamma \cup \{\neg A\}$ is inconsistent.
$\Gamma \vdash \neg A$ iff $\Gamma \cup \{A\}$ is inconsistent.

**Proof by Cases:** If $\Gamma, A \vdash B$ and $\Gamma, \neg A \vdash B$, then $\Gamma \vdash B$.

**Derived (admissible) rules of inference:**

**Modus Tollens:** $\dfrac{A \to B \qquad \neg B}{\neg A}$

**Hypothetical Syllogism:** $\dfrac{A \to B \qquad B \to C}{A \to C}$

Using item (6), or proof by contradiction, it suffices to show that

$$\{A, A \to \neg B \to C, B \to D, C \to D, \neg D\} \text{ is inconsistent.}$$

Using item (9) twice, namely using Modus Tollens on the last three formulas, it suffices to show that

$$\{A, A \to \neg B \to C, \neg B, \neg C, \neg D\} \text{ is inconsistent.}$$

Using item (8) twice, namely using Modus Ponens, it suffices to show that

$$\{A, C, \neg B, \neg C, \neg D\} \text{ is inconsistent.}$$

This is inconsistent as it contains both $C$ and $\neg C$ as members.

## II.6 Soundness and Completeness Theorems for PL

The Soundness and Completeness Theorems express the fact that PL is a good proof system for propositional logic. The Soundness Theorem states that PL

is "sound" in the sense that any PL-theorem $A$ is a tautology. Furthermore, it states that if $A$ is provable from a set $\Gamma$ of hypotheses, then $\Gamma$ tautologically implies $A$. This is really a basic property for any system of propositional logic since we of course only want to have proofs of formulas that are tautologically implied.

The Completeness Theorem states the converse, namely that PL is "complete". This means firstly that if $A$ is a tautology, then $A$ has a PL-proof. And, secondly, that if $A$ is a tautological consequence of $\Gamma$, then $\Gamma \vdash A$. In other words, PL with its four axiom schemes, PL1-PL4, and with the sole rule of inference of Modus Ponens is strong enough to give proofs of all tautologies. This is fairly remarkable!

We now state the Soundness and Completeness Theorems more carefully and prove the Soundness Theorem. After that, Section II.7 will present the more difficult proof of the Completeness Theorem.

**Theorem II.32** (Soundness Theorem for PL)**.**
(a) *If* $\Gamma$ *is satisfiable, then* $\Gamma$ *is consistent.*
(b) *If* $\Gamma \vdash A$*, then* $\Gamma \vDash A$*.*

**Theorem II.33** (Completeness Theorem for PL)**.**
(a) *If* $\Gamma$ *is consistent, then* $\Gamma$ *is satisfiable.*
(b) *If* $\Gamma \vDash A$*, then* $\Gamma \vdash A$*.*

Taken together, the Soundness and Completeness Theorems state that satisfiability is equivalent to consistency. They also state that $\vDash$ (tautological implication) is equivalent to $\vdash$ (provability). In other words, $\Gamma \vdash A$ is equivalent to $\Gamma \vDash A$. In the special case with $\Gamma$ the empty set, $\vdash A$ is equivalent to $\vDash A$. In other words, $A$ has a proof in PL if and only if $A$ is a tautology.

The Completeness Theorem allows us to introduce a new derived rule of inference for PL called Tautological Implication. For this, suppose $A_1, \ldots, A_k \vDash B$. Then the following Tautological Implication (TAUT) is admissible as a derived rule for PL-proofs:

$$\text{TAUT: } \frac{A_1 \qquad A_2 \qquad \cdots \qquad A_k}{B}$$

To prove that TAUT is an admissible inference rule when $A_1, \ldots, A_k \vDash B$, note that the Completeness Theorem implies that $A_1, \ldots, A_k \vdash B$. So $k$ uses of the Deduction Theorem imply that $\vdash A_1 \to A_2 \to \cdots \to A_k \to B$. Then, if the hypotheses $A_1, \ldots, A_k$ have been derived, $k$ uses of Modus Ponens allows $B$ to be derived.

Hypothetical Syllogism and Modus Tollens (and also Modus Ponens) are special cases of the Tautological Implication rule. In fact, by the Soundness and Completeness Theorems, the Tautological Implication rule is the strongest possible admissible rule of inference for PL.

In both the Soundness and Completeness Theorems, the two parts (a) and (b) easily imply each other. Let's start by proving that part (a) of the Soundness Theorem implies part (b). For this, we note that

| | | | |
|---|---|---|---|
| $\Gamma \vdash A$ | | | |
| | $\Leftrightarrow$ | $\Gamma \cup \{\neg A\}$ is inconsistent | Proof by Contradiction, Theorem II.19 |
| | $\Rightarrow$ | $\Gamma \cup \{\neg A\}$ is unsatisfiable | Soundness Theorem II.32, part (a) |
| | $\Leftrightarrow$ | $\Gamma \vDash A$ | Theorem I.21 |

That finishes the proof of (b) from (a) for the Soundness Theorem. For the Completeness Theorem, we argue similarly:

| | | | |
|---|---|---|---|
| $\Gamma \vDash A$ | | | |
| | $\Leftrightarrow$ | $\Gamma \cup \{\neg A\}$ is unsatisfiable | Theorem I.21 |
| | $\Rightarrow$ | $\Gamma \cup \{\neg A\}$ is inconsistent | Completeness Theorem II.33, part (a) |
| | $\Leftrightarrow$ | $\Gamma \vdash A$ | Proof by Contradiction, Theorem II.19 |

That finishes the proof that part (a) of the Completeness Theorem implies part (b).

We now prove the Soundness Theorem.

*Proof of the Soundness Theorem for* PL. Suppose $\Gamma$ is a set of formulas, and $\Gamma \vdash A$. We shall prove that $\Gamma \vDash A$. (Essentially we are proving part (b) of the Soundness Theorem, and will then use that to prove part (a).) Let

$$A_1, A_2, \ldots, A_\ell$$

be a PL-proof of $A$. Suppose $\varphi$ is a truth assignment satisfying $\Gamma$. We prove that $\varphi(A_i) = \mathrm{T}$ for $i = 1, \ldots, \ell$, by induction on $i$. Since $A_\ell$ is $A$, this implies that $\varphi(A) = \mathrm{T}$.

There are two base cases for the proof by induction. If $A_i$ is a PL-axiom, then $A_i$ is a tautology, so of course $\varphi(A_i) = \mathrm{T}$. If $A \in \Gamma$, then $\varphi(A_i) = \mathrm{T}$ since $\varphi$ satisfies $\Gamma$. For the induction step, suppose $A_i$ is inferred by Modus Ponens from $A_j$ and $A_k$ where $j, k < i$, and $A_k$ is equal to $A_j \to A_i$. The two induction hypotheses are that $\varphi(A_j) = \mathrm{T}$ and $\varphi(A_j \to A_i) = \mathrm{T}$. By the definition of truth for $A_j \to A_i$, we must thus have $\varphi(A_i) = \mathrm{T}$. That completes the proof by induction.

Now we can prove part (a) of the Soundness Theorem. Suppose $\Gamma$ is inconsistent. Then $\Gamma \vdash A$ and $\Gamma \vdash \neg A$. By the previous two paragraphs, $\Gamma \vDash A$ and $\Gamma \vDash \neg A$ for some formula $A$. Suppose $\varphi$ is a truth assignment satisfying $\Gamma$. By $\Gamma \vDash A$, $\varphi(A) = \mathrm{T}$. Similarly, by $\Gamma \vDash \neg A$, $\varphi(\neg A) = \mathrm{T}$. This is impossible. Therefore, there is no $\varphi$ that satisfies $\Gamma$; i.e., $\Gamma$ is unsatisfiable. That proves part (a), and finishes the proof of the Soundness Theorem for PL. □

## II.7 Proof of the Completeness Theorem

The proof of the Completeness Theorem II.33 is harder than the proof just given for the Soundness Theorem. The proof will be given in two parts. First, we state and prove Lindenbaum's Theorem about the existence of a complete, consistent set $\Pi$ of formulas extending $\Gamma$. Then Lemma II.37 uses the set $\Pi$ to define a truth assignment $\varphi$ that satisfies $\Pi$, and hence satisfies $\Gamma$.

First, we give the definition of what it means for $\Gamma$ to be "complete":

**Definition II.34.** A set $\Gamma$ of formulas is *complete* provided that, for every formula $A$, either $A \in \Gamma$ or $\neg A \in \Gamma$.

It is trivial that, if $\Gamma$ is complete, then for every formula $A$, either $\Gamma \vdash A$ or $\Gamma \vdash \neg A$. The idea behind completeness, is that $\Gamma$ is "maximal" in that there is no way to add another formula to $\Gamma$ and still have a consistent set of formulas.

**Theorem II.35** (Lindenbaum's Theorem)**.** *Suppose $\Gamma$ is a consistent set of formulas. Then there is a consistent, complete set $\Pi$ of formulas such that $\Gamma \subseteq \Pi$.*

*Proof.* There are countably many propositional formulas, so the set of all propositional formulas can be enumerated in an infinite sequence

$$A_1, A_2, A_3, A_4, \ldots.$$

The crucial property is that *every* propositional formula appears in the sequence as one of the $A_i$'s.

We define a sequence of sets $\Gamma_i$ of formulas for $i = 0, 1, 2, \ldots$. Each $\Gamma_i$ will be consistent, and they satisfy $\Gamma_{i+1} \supseteq \Gamma_i$. To start the process, set $\Gamma_0 = \Gamma$. Inductively define

$$\Gamma_{i+1} = \begin{cases} \Gamma_i \cup \{A_i\} & \text{if } \Gamma \cup \{A_i\} \text{ is consistent} \\ \Gamma_i \cup \{\neg A_i\} & \text{otherwise.} \end{cases}$$

**Claim.** *For each $i$,*
(a) *$\Gamma_i \supseteq \Gamma_{i-1}$ and $\Gamma_i \supseteq \Gamma$.*
(b) *$\Gamma_i$ is consistent.*

Part (a) of the claim is immediate from the definitions. Part (b) is proved by induction on $i$. For $i = 0$, $\Gamma_0 = \Gamma$ is consistent by assumption. For the induction step, assume $\Gamma_i$ is consistent. Then $\Gamma_{i+1}$ is consistent since by Corollary II.29, if $\Gamma_i \cup \{A_i\}$ is not consistent, then $\Gamma_i \cup \{\neg A_i\}$ is consistent. □

Now define $\Pi$ to equal the union $\bigcup_i \Gamma_i$. Every formula $A$ is equal to some $A_i$. This means that either $A$ or $\neg A$ is a member of $\Gamma_{i+1}$. Therefore, either $A \in \Pi$ or $\neg A \in \Pi$. Hence, $\Pi$ is complete.

To prove $\Pi$ is consistent, suppose to the contrary that $\Pi$ is inconsistent. By Theorem II.18, there is a finite $\Pi_0 \subseteq \Pi$ which is inconsistent. Since $\Pi_0$ is finite and a subset of $\bigcup_i \Gamma_i$, and since the $\Gamma_i$'s are increasing, there is a single $\Gamma_i$ such that $\Pi_0 \subseteq \Gamma_i$. But this gives a contradiction since $\Gamma_i$ is consistent and its subset $\Pi_0$ is inconsistent (contradicting Theorem II.15). Therefore, $\Pi$ cannot be inconsistent.

That completes the proof of Lindenbaum's Theorem. □

**Lemma II.36.** *Suppose $\Pi$ is complete and consistent. Let $A$ and $B$ be any formulas. Then,*

(a) *$A \in \Pi$ if and only $\neg A \notin \Pi$.*

(b) $(A \to B) \in \Pi$ *if and only if* $A \notin \Pi$ *or* $B \in \Pi$.

*Proof.* Since $\Pi$ is complete, at least one of $A$ and $\neg A$ must be in $\Pi$. But since $\Pi$ is consistent, they cannot both be in $\Pi$. Thus (a) must hold.

Now we prove (b). First suppose $A \to B$ is in $\Pi$. Also suppose, for the sake of contradiction, that both $A \in \Pi$ and $B \notin \Pi$. By (a), $\neg B \in \Pi$. This is a contradiction, since $\{A \to B, A, \neg B\}$ is inconsistent and is a subset of $\Pi$. This shows that if $(A \to B) \in \Pi$, then $A \notin \Pi$ or $B \in \Pi$.

Second, suppose $A \to B$ is not in $\Pi$. By (a), $\neg(A \to B)$ is in $\Pi$. Then, since $\{\neg(A \to B), \neg A\}$ is inconsistent, $\neg A$ cannot be in $\Pi$. Therefore, by (a), $A \in \Pi$. And, since $\{\neg(A \to B), B\}$ is inconsistent, $B$ is not in $\Pi$. Thus we have shown that if $A \to B$ is not in $\Pi$, then $A \in \Pi$ and $B \notin \Pi$. That completes the proof of part (b). □

**Lemma II.37.** *Suppose* $\Pi$ *is complete and consistent. Then* $\Pi$ *is satisfiable.*

*Proof.* We must define a truth assignment $\varphi$ satisfying $\Pi$. For this, define $\varphi$'s truth values for variables $p_i$ by:

$$\varphi(p_i) \;=\; \begin{cases} \mathrm{T} & \text{if } p_i \in \Pi \\ \mathrm{F} & \text{otherwise.} \end{cases}$$

To prove the lemma, it suffices to prove the claim that:

**Claim.** *For every* $A$, $\varphi(A) = \mathrm{T}$ *if and only if* $A \in \Pi$.

The claim is proved by induction on the complexity of the formula $A$. If $A$ is a variable $p_i$, the claim holds immediately by the definition of $\varphi$. For the first induction step, suppose $A$ is $\neg B$. The induction hypothesis tells that the claim holds for $B$. Thus,

| | | | |
|---|---|---|---|
| $\neg B \in \Pi$ | $\Leftrightarrow$ | $B \notin \Pi$ | Lemma II.36(a) |
| | $\Leftrightarrow$ | $\varphi(B) = \mathrm{F}$ | Induction hypothesis |
| | $\Leftrightarrow$ | $\varphi(\neg B) = \mathrm{T}$ | Definition of truth |

For the second induction step, suppose $A$ is $B \to C$. Then

| | | | |
|---|---|---|---|
| $(B \to C) \in \Gamma$ | $\Leftrightarrow$ | $B \notin \Gamma$ or $C \in \Gamma$ | Lemma II.36(b) |
| | $\Leftrightarrow$ | $\varphi(B) = \mathrm{F}$ or $\varphi(C) = \mathrm{T}$ | Induction hypotheses for $B$ and $C$ |
| | $\Leftrightarrow$ | $\varphi(B \to C) = \mathrm{T}$ | Definition of truth |

That completes the proof of the claim.

Since $\varphi(A) = \mathrm{T}$ for every $A \in \Gamma$, the set $\Gamma$ is satisfiable with satisfying assignment $\varphi$. That finishes the proof of the lemma. □

Lindenbaum's Theorem and Lemma II.37 together imply part (a) of the Completeness Theorem II.33. Lindenbaum's Theorem gives a complete, consistent $\Pi$ extending the consistent set $\Gamma$, and Lemma II.37 gives a truth assignment that satisfies $\Pi$ and hence satisfies its subset $\Gamma$.

As argued in Section II.6, part (a) of the Completeness Theorem implies part (b). Thus we have finished the proof of the Completeness Theorem for PL.

## II.8 The Compactness Theorem for PL

The *Compactness Theorem* is one of the fundamental properties of both propositional and first-order logic. It has a very simple proof from the Soundness and Completeness Theorems.

**Definition II.38.** A set $\Gamma$ of formula is *finitely satisfiable* if every finite subset $\Pi$ of $\Gamma$ is satisfiable.

**Theorem II.39** (Compactness Theorem)**.**
(a) $\Gamma$ *is satisfiable if and only if* $\Gamma$ *is finitely satisfiable.*
(b) $\Gamma \vDash A$ *if and only there is finite subset* $\Gamma_0$ *of* $\Gamma$ *such that* $\Gamma_0 \vDash A$.

*Proof.* By the Soundness and Completeness Theorems, $\Gamma$ is satisfiable if and only if $\Gamma$ is consistent. Likewise, $\Gamma$ is finitely satisfiable if and only if every finite subset of $\Gamma$ is consistent. Part (a) is now immediate from the fact that $\Gamma$ is consistent if and only if every finite subset of $\Gamma$ is consistent. (See Theorem II.18.)

The proof of (b) is similar. It follows from the fact that $\Gamma \vdash A$ if and only if $\Gamma_0 \vdash A$ for some finite $\Gamma_0 \subseteq \Gamma$. (See Theorem II.3(b).) □

## Exercises

Your answers for Exercises II.1-II.22 should not use the Completeness Theorem. Exercises II.1-II.3 can be done using only the material of Section II.2. Exercises II.4-II.7 are intended to be done using only the techniques developed up through Section II.3 (on the Deduction Theorem).

**Exercise II.1.** Give explicit PL-proofs for the following formulas. The explicit proofs should show every line in the PL-proofs.
(a) $A \to B \vdash A \to B$.
(b) $A \to B, B \vdash B$.
(c) $A \to B, A \vdash B$.
(d) $\neg A, A \vdash B$

**Exercise II.2.** Show that $\vdash (A \to B) \to (A \to A)$ by giving an explicit three line PL-proof of that formula.

**Exercise II.3.** Give explicit PL-proofs for:
(a) $A \wedge B, \neg B \vdash A$
(b) $A \wedge B, \neg A \vdash B$

**Exercise II.4.** Prove the following. You only need to prove that a PL-proof exists; you do not need to give it explicitly.
(a) $A \to B, C \vdash A \to (C \to B)$. Are both hypotheses needed?
(b) $A \to B \to C \vdash B \to A \to C$.

**Exercise II.5.**
(a) Prove $\vdash (A \to B) \to (B \to C) \to (C \to D) \to (A \to D)$.
(b) Conclude the following generalization of Hypothetical Syllogism is a valid derived rule of inference for PL:

$$\frac{A \to B \qquad B \to C \qquad C \to D}{A \to D}$$

**Exercise II.6.** Prove $\vdash (A \vee B) \to \neg A \to B$.

**Exercise II.7.** Prove $A \to B \vdash (\neg A \to A) \to B$.

**Exercise II.8.** Show $\vdash (A \to B) \to (\neg A \to B) \to B$. (Compare with Exercise I.12.)

**Exercise II.9.** Prove that the following are inconsistent.
(a) $\{\neg(p_1 \to p_1)\}$.
(b) $\{p_1 \wedge \neg p_1\}$.

**Exercise II.10.** Prove that $\Gamma$ is inconsistent if and only if $\Gamma \vdash \neg(p_1 \to p_1)$.

**Exercise II.11.** Show that the following formulas have PL-proofs. (See Example II.6.)
(a) $A \to A \vee B$.
(b) $B \to A \vee B$.
(c) $\neg(A \vee B) \to (\neg A \wedge \neg B)$.

**Exercise II.12.** Show that the following formulas have PL-proofs.
(a) $(A \wedge B) \to A$.
(b) $(A \wedge B) \to B$.
(c) $\neg(A \wedge B) \to (\neg A \vee \neg B)$.
(d) $A \to B \to A \wedge B$.

**Exercise II.13.** Do the following for each of the formulas (a)-(c): (i) State whether it is unsatisfiable, or satisfiable but not a tautology, or a tautology; (ii) If it is not a tautology, give a truth assignment that falsifies it; (ii) Otherwise if it is a tautology, prove that it has a PL-proof.
(a) $p \to (q \to p) \to p$.
(b) $(p \to (q \to p)) \to p$.
(c) $((p \to q) \to p) \to p$.

**Exercise II.14.** Prove that $\Gamma \cup \{A \wedge B\}$ is inconsistent if and only if $\Gamma \cup \{A, B\}$ is inconsistent. [Hint: You may use the results of Exercise II.12 in your answer.]

**Exercise II.15.** Show that $\Gamma, A \vee B$ is inconsistent if and only if both $\Gamma \cup \{A\}$ and $\Gamma \cup \{B\}$ are inconsistent.

**Exercise II.16.** Show that $\Gamma, A \vee B \vdash C$ holds if and only if $\Gamma, A \vdash C$ and $\Gamma, B \vdash C$ both hold. [HInt: Use proof-by-cases.]

**Exercise II.17.** Show that $\Gamma, A \leftrightarrow B$ is inconsistent if and only if both $\Gamma \cup \{A, B\}$ and $\Gamma \cup \{\neg A, \neg B\}$ are inconsistent.

**Exercise II.18.** Prove that $\vdash ((p \to q) \to p) \to p$. This tautology is known as Pierce's Law and is sometimes used as an axiom related to the Law of the Excluded Middle. (This is a duplicate of Exercise II.13.)

**Exercise II.19.** Show that $\{A \to B, A, \neg B\}$ is inconsistent. (This was used in the proof of Lemma II.36.)

**Exercise II.20.** Prove
(a) $\{\neg(A \to B), \neg A\}$ is inconsistent.
(b) $\{\neg(A \to B), B\}$ is inconsistent.
These facts were used in the proof of Lemma II.36.

**Exercise II.21.** Prove that $\Gamma \cup \{A \to B\}$ is inconsistent if and only if $\Gamma \vdash A$ and $\Gamma \cup \{B\}$ is inconsistent. (Compare to Theorem II.30 and to item (5) in the list in Section II.5.)

**Exercise II.22.** (Generalization of proof-by-cases.) Suppose that $\Gamma \vdash A \vee B$ and $\Gamma \vdash A \to C$ and $\Gamma \vdash B \to C$. Prove that $\Gamma \vdash C$. [Hint: One way to do this is to prove

$$\vdash (A \to C) \to (B \to C) \to (A \vee B \to C).$$

This is the last axiom for $\vee$ on page 23 and is related to Exercise II.16. In any event, it is suggested to use proof-by-cases.]

**Exercise II.23.** Give a direct proof that part (a) of the Soundness Theorem II.33 implies part (b). (See the last part of the proof of the Soundness Theorem.)

**Exercise II.24.** Suppose that $\Gamma \vDash p_i$ or $\Gamma \vDash \neg p_i$, for every $i$. Prove that, for every formula $A$, $\Gamma \vDash A$ or $\Gamma \vDash \neg A$. (This is similar to being complete; however, instead of having one of $A$ or $\neg A$ a member of $\Gamma$, we have one of $A$ or $\neg A$ tautologically implied by $\Gamma$.)

**Exercise II.25.** Suppose that $\Gamma$ and $\Delta$ are sets of formulas and that $\Gamma \cup \Delta$ is unsatisfiable.
(a) Prove that there is a finite $\Gamma' \subseteq \Gamma$ and a finite $\Delta' \subseteq \Delta$ such that $\Gamma' \cup \Delta'$ is unsatisfiable.
(b) Prove that there is a formula $A$ such that $\Gamma \vDash A$ and $\Delta \vDash \neg A$.

**Exercise II.26.** Two sets $\Gamma$ and $\Delta$ of formulas are called *complementary* if every truth assignment $\varphi$, satisfies exactly one of $\Gamma$ and $\Delta$. Suppose $\Gamma$ and $\Delta$ are complementary. Prove there are subsets $\Gamma' \subseteq \Gamma$ and $\Delta' \subseteq \Delta$ such that $\Gamma'$ and $\Delta'$ are finite and such that $\Gamma' \vDash \Gamma$ and $\Delta' \vDash \Delta$. (Recall that the notation $\Gamma' \vDash \Gamma$ means $\Gamma' \vDash A$ for all $A \in \Gamma$.)

**Exercise II.27.** (This builds on Exercise II.26.)
(a) Prove that a set $\Gamma$ has a complementary set $\Delta$ if and only if there is a formula $A$ such that $\Gamma \vDash A$ and $A \vDash \Gamma$.
(b) Give an example of a set $\Gamma$ which does not have a complementary set. Prove that it does not have a complementary set.

**Exercise II.28.** Use the Compactness Theorem for propositional logic to prove that a graph $G$ is 3-colorable if and only if every finite subgraph is 3-colorable. ("3-colorable" means there is an assignment of three colors to the vertices of the graph so that no edge connects vertices assigned the same color.) For this, fix a graph $G$. Use propositional variables $r_i, g_i, b_i$ whose intended meanings are that "Vertex $i$ is red", "Vertex $i$ is green", and "Vertex $i$ is blue", respectively. Let $\Gamma$ be a set of formulas using these variables that expresses the conditions that (a) each vertex has a color assigned to it, and (b) if two vertices $i$ and $j$ are joined by an edge in $G$, then they are not assigned the same color. The set $\Gamma$ should be satisfiable if and only if $G$ is 3-colorable. Then apply the Compactness Theorem.

**Exercise II.29**★ Show that the reductions of items (1)-(5) of Section II.5 can be used to give an alternate proof of the Completeness Theorem for PL by working out the details of the following proof sketch.

First, define $N(\Gamma)$ to be the number of occurrences of $\rightarrow$'s and $\neg$'s in formulas in $\Gamma$ minus the number of negated literals $\neg p_i$ appearing in $\Gamma$. In other words, $N(\Gamma)$ counts only the negation signs $\neg$ that are not part of literals. Then prove by induction on $N$ that the following two statements both hold:

> If $N(\Gamma) = 0$ and $\Gamma$ is unsatisfiable, then $\Gamma$ is inconsistent.
> If $N(\Gamma) + N(\{A\}) = N$ and $\Gamma \vDash A$, then $\Gamma \vdash A$.

Each induction step will use one or two of the reductions given in items (1)-(5) of Section II.5.

In the base case, $N(\Gamma) = 0$ means that $\Gamma$ is a set of literals. In this case, show that either $\Gamma$ is satisfiable or $\Gamma$ is inconsistent. Also show that if $N(\Gamma) = N(\{A\}) = 0$, then either $\Gamma \nvDash A$ or $\Gamma \vdash A$.

**Exercise II.30**★ (For readers who know some set theory.) The proof of Lindenbaum's Theorem II.35 depended on the fact that, since there are only countably many variables $p_i$, there are only countably many formulas. It is also possible to allow uncountably many propositional variables $p_\alpha$ where $\alpha \in \mathcal{I}$ for an arbitrary set $\mathcal{I}$. Use Zorn's Lemma to prove Lindenbaum's Theorem holds also when there are uncountably many variables and hence uncountably many formulas. Use this to prove the Completeness and Compactness Theorems hold also for propositional formulas over uncountably many variables.

**Exercise II.31**★ Give a proof of the Compactness Theorem that does not depend on the Soundness or Completeness Theorems. Use the following proof sketch.

(a) Prove that if $\Gamma$ is finitely satisfiable and $A$ is a formula, then at least one of $\Gamma \cup \{A\}$ or $\Gamma \cup \{\neg A\}$ is finitely satisfiable.
(b) Prove the analogue of Lindenbaum's Theorem for finitely satisfiability instead of consistency. Namely, prove that if $\Gamma$ is finitely satisfiable, then there is a complete, finitely satisfiable $\Pi \supseteq \Gamma$.
(c) Prove that the properties of Lemma II.36 hold under the assumption that $\Pi$ is complete and finitely satisfiable.
(d) Prove the analogue of Lemma II.37 that states that if $\Pi$ is complete and

finitely satisfiable, then $\Pi$ is satisfiable. (Exactly the same proof works for this part with no changes needed.)

# Chapter III

# First-Order Logic: Syntax and Semantics

## III.1 Introduction to First-order Logic

First-order logic is a very expressive and powerful formal system, capable of formalizing statements in a wide range of topics. It is particularly well suited to the formalization of mathematical statements, but it also works well in many applications governed by well-defined properties and rules. First-order logic augments propositional logic by having variables that range over some domain of "objects" or "individuals". These variables can be quantified with universal ($\forall$) quantifiers and existential ($\exists$) quantifiers, so that a formula can express properties of the entire domain of objects. First-order logic includes "predicates" or "relations" that describe properties of individuals or relationships between individuals. It also includes "constants" that name individuals, and "functions" that map individuals to individuals. First-order logic incorporates propositional logic so that all the constructions from the previous two chapters are still relevant for first-order logic.

The name "first-order" refers to the fact that the quantifiers act on individuals. Second-order logic and higher-order logic allow quantifying over sets of individuals or over functions acting on individuals. Higher-order logics are ostensibly stronger than first-order logic. On the other hand, practically any formal logic can be recast as a first-order logic (albeit perhaps in an unnatural manner). This makes first-order logic a kind of "ultimate" logic; furthermore, first-order logic has very nice properties, including having a proof system with the Soundness and Completeness Theorems. In addition, set theory is usually formalized as a first-order logic, and type theories can be recast as first-order logics (at least in principle). Arguably, set theory and type theories can formalize all of mathematics as presently practiced, so this means that all of mathematics can be formalized using first-order logic.

We start by presenting three small examples to illustrate the essential in-

gredients of first-order formulas. After that, Section III.2 will give the formal definition of first-order formulas.

**Natural language example.** Let's start with the sentence

*Everyone has read a book they don't like.*

This can be expressed as the first-order formula

$$\forall x\,(\mathit{Person}(x) \to \exists y\,[\mathit{Book}(y) \wedge \mathit{Read}(x,y) \wedge \neg \mathit{Likes}(x,y)]). \tag{III.1}$$

using "predicate symbols" (also called "relation symbols") with the meanings:

$$\begin{array}{rl} \mathit{Person}(x) & \text{means "}x\text{ is a person"} \\ \mathit{Book}(y) & \text{means "}y\text{ is a book"} \\ \mathit{Read}(x,y) & \text{means "}x\text{ has read }y\text{"} \\ \mathit{Likes}(x,y) & \text{means "}x\text{ likes }y\text{".} \end{array}$$

These predicates take individuals as inputs and output true/false values. The quantifiers "$\forall x$" and "$\exists y$" mean "for all $x$" and "for some $y$", of course. We are being deliberately vague about the universe of objects that the quantified variable range over, but it should include, say, the universe of all people and all objects.

For a second example, consider

*Everyone's mother likes* Moby Dick.

This can be expressed in first-order logic as:

$$\forall x[\mathit{Person}(x) \to \mathit{Likes}(\mathit{Mother}(x), \mathit{Moby\ Dick})]. \tag{III.2}$$

Here "*Mother* is a "function symbol", giving the mother of $x$ as a function of $x$. This makes the implicit assumption that everyone has exactly one mother. The "*Moby Dick*" is a "constant symbol"; namely it refers to a specific object.

For a third example, consider

*No two people have read exactly the same books.*

We can express this as

$$\begin{aligned} \forall x\,\forall w\,[&\mathit{Person}(x) \wedge \mathit{Person}(w) \wedge \neg x = w \\ &\to \exists y(\mathit{Book}(y) \wedge \neg(\mathit{Read}(x,y) \leftrightarrow \mathit{Read}(w,y)))]. \end{aligned} \tag{III.3}$$

This introduces the equality predicate, $=$. It is common to write "$x \neq y$" as an abbreviation for "$\neg x = y$". As a final example, consider

*No one likes a book unless they have read it.*

This can be expressed as:

$$\forall x\, \forall z\, [Person(x) \wedge Book(z) \wedge Likes(x, z) \rightarrow Read(x, z)] \tag{III.4}$$

or as

$$\neg\exists x\, \exists z\, [Person(x) \wedge Book(z) \wedge Likes(x, z) \wedge \neg Read(x, z)]. \tag{III.5}$$

The formulas (III.1) and (III.2) both use the construction

$$\text{“}\forall x(Person(x) \rightarrow \cdots)\text{”}.$$

This is a common construction used to state that every person $x$ satisfies the property in the ellipses “$\cdots$”. In other words, if we let *Persons* denote the set of all people, then “$\forall x(Person(x) \rightarrow \cdots)$” means the same thing as “$\forall x \in Persons\,(\cdots)$”. The dual construction for existential quantification is illustrated in formulas (III.1) and (III.3): these two formulas use the construction “$\exists y(Book(y) \wedge \cdots)$” to mean the same thing as “$\exists y \in Books\,(\cdots)$” where *Books* is the set of all books.[1]

These two constructions are also illustrated in the formulas (III.4) and (III.5). The former uses the pattern “$\forall\cdots \rightarrow \cdots$” and the latter uses the pattern “$\exists\cdots\wedge\cdots$”. The correspondence between (III.4) and (III.5) can be understood by noting that “$\forall x$” means the same as “$\neg\exists x\neg$”. To state this in English, a property holds for all $x$ if and only if it is not the case that the property fails for some $x$. If the “$\forall x$” and “$\forall y$” of (III.4) are replaced by “$\neg\exists x\neg$” and “$\neg\exists y\neg$”, then tautological equivalences show that the result is equivalent to (III.5). For more on this, see Section III.4 below.

The above examples illustrate some of the features of first-order logic, but they also show that first-order logic is not good at capturing many of the nuances of natural language. For example, it is presumed that for every $x$ and $y$, the predicate $Read(x, y)$ is either true or false. There is no consideration of whether $x$ might have read only part of $y$, or just skimmed it, etc. Likewise, the constant “*Moby Dick*” is presumed to denote one particular book; there is no allowance for nuance about different editions of the book, etc. In the same vein, it is presumed that everyone has a unique mother; something which is true in most cases, but of course is not universally true. Finally, there is presumed to be a definite, fixed universe of objects that quantified variables range over; this can also be a problematic assumption in the setting of natural language.

First-order logic is better at capturing mathematical statements, as the next examples show.

**Group theory examples.** A group is a mathematical structure with a binary operation that is associative and has an identity element and inverses. To formalize groups, we first need to choose a first-order language, namely the

[1]The notations “$\forall x \in$” and “$\exists x \in$” will not be included in the syntax for first-order formulas. As just discussed, this does not cause any loss of expressiveness for first-order formulas.

predicate symbols, function symbols, and constant symbols that may be used in formulas. For (multiplicative) groups, we can use the language with three symbols: a binary function $\cdot$ for multiplication, a unary inverse function $^{-1}$, and a constant symbol 1 for the identity element. As is usual, the equality predicate, $=$, is also allowed. The three axioms for groups can be written as:[2]

$$\begin{array}{rl} \forall x\, \forall y\, \forall z\, [x \cdot (y \cdot z) = (x \cdot y) \cdot z] & \text{- Associativity} \\ \forall x\, (1 \cdot x = x \wedge x \cdot 1 = x) & \text{- Identity} \\ \forall x\, (x \cdot x^{-1} = 1 \wedge x^{-1} \cdot x = 1) & \text{- Inverses} \end{array} \tag{III.6}$$

There are many possible groups of course. However, these three axioms exactly characterize groups in the sense that a mathematical structure is a group if and only it satisfies these three axioms.

Other languages for groups can be used instead of $\cdot$, $^{-1}$ and 1. For instance, it is also possible to use the language containing only the binary group operation $\cdot$. This would not reduce the expressibility of formulas. Indeed, the identity and inverse axioms can be rewritten so as to use just the function symbol $\cdot$ as:

$$\begin{array}{rl} \exists z\, \forall x\, (x \cdot z = x) & \text{- Right identity} \\ \forall x\, \exists y\, \forall u\, [u \cdot (x \cdot y) = u] & \text{- Right inverses} \end{array}$$

To keep things simpler, these axioms state only the existence of a right identity element and of right inverses; however, it is easy to modify them to give the full identity and inverses axioms.[3]

Many group properties can be expressed in first-order logic. For instance, $\forall x\, (x \neq 1 \rightarrow x{\cdot}x \neq 1)$ states that no element has order two. Similarly, $\forall x\, (x \neq 1 \rightarrow x{\cdot}(x{\cdot}x) \neq 1)$ states that no element has order three. In the same way, for any $k \geq 2$, there is a formula $T_k$ expressing that no element has order $k$. Specifically, let $k \geq 2$ be a fixed integer. Let $x^k$ denote the $k$-fold product of $x$ with itself, and let $T_k$ be the formula $\forall x(x \neq 1 \rightarrow x^k \neq 1)$. The formula $T_k$ states that the order of $x$ does not divide $k$. Let $Div(k)$ denote the (finite) set of proper divisors of $k$. Then $T_k(x) \wedge x^k = 1 \wedge \bigwedge_{i \in Div(k)} \neg T_i(x)$ expresses that $x$ has order $k$.

As another example, the center of a group can be defined by the formula $\forall x\, (z \cdot x = x \cdot z)$, as this characterizes the members $z$ of the center. Note how $z$ is not quantified in this. This is because the formula expresses a property of the object $z$.

However, there are many common concepts in group theory that are not first-order expressible. For instance, first-order logic is unable to talk directly about subgroups since variables range only over group elements and there is no way to existentially or universally quantify over subgroups. Certainly, special subgroups (such as the center) can be defined, but properties of arbitrary subgroups cannot in general be expressed in first-order logic.

[2] When we formally define first-order logic in the next section, we will require all function symbols to be written in prefix notation. Thus, strictly speaking, we should write $\cdot(x, y)$ and $^{-1}(x)$ instead of $x \cdot y$ and $x^{-1}$. However, for readability, we use the more common notations of $x \cdot y$ and $x^{-1}$ as abbreviations for terms written in prefix notation.

[3] And, in any event, it is well known that right identity and right inverses axioms (together with associativity) imply the full identity and inverses axioms. In addition, it is easy to see that the right inverses axiom is stated in a way that implies the right identity axiom.

A group is called *torsion-free* if there are no elements of finite order (other than the identity). The infinite set of formulas

$$\Gamma \;=\; \{T_2, T_3, T_4, \ldots\}$$

expresses the property of being torsion-free. However, as we see later in Section IV.5 no single formula, nor finite set of formulas, characterizes the property of a group being torsion-free. It will follow that there is no set $\Pi$ of first-order formulas, finite or infinite, that characterizes the property of *not* being torsion-free. For this, see Theorem IV.48.

**The theory of the integers.** The first-order theory of the integers is of central importance in mathematical logic, as well as in mathematics in general. For logicians, it is typical to define the integers $\mathbb{N}$ as $\{0, 1, 2, \ldots\}$, namely starting with 0. So, henceforth, we generally use "integer" to mean "nonnegative integer".

It is traditional for the language of arithmetic to include the following symbols: the constant 0, the unary successor function $S$, and the binary addition and multiplication function $+$ and $\cdot$. These have their usual meanings; the successor function $S(x)$ is intended to denote $S(x) = x + 1$. As usual, the equality symbol $=$ is also allowed. Many properties of integers can be expressed with these symbols. For instance, the constants 1 and 2 can be represented by the terms $S(0)$ and $S(S(0))$. And, the relation "$x \le y$" is expressible with

$$\exists z\,(x + z = y), \tag{III.7}$$

Note this formula has two "free" variables $x, y$ (that is, $x$ and $y$ are not quantified), and one "bound" (quantified) variable $z$. This means that the formula expresses a property of $x$ and $y$.

The property "$x$ is a divisor of $y$" can be expressed as $\exists z(z \cdot x = y)$. The property that "$x$ is even" can be expressed as either

$$\exists v\,(v + v = x) \qquad \text{or} \qquad \exists z\,(z \cdot S(S(0)) = x.$$

Then, "$x$ is a power of 2" can expressed by saying that any divisor of $x$ either is equal to 1 or is even; namely,

$$\forall u\,(\exists z\,(z \cdot u = x) \to u = S(0) \vee \exists v\,(v + v = u)).$$

The property that $x$ is prime can be expressed by

$$x \neq S(0) \wedge \forall y\,(\exists z\,(y \cdot z = x) \to y = S(0) \vee y = x). \tag{III.8}$$

This expresses that any divisor $y$ of $x$ is equal to either 1 or $x$.

Let $\mathit{Prime}(x)$ denote the formula (III.8). The property that there exist infinitely many primes can be expressed by

$$\forall x\,\exists y\,(x \le y \wedge \mathit{Prime}(y)). \tag{III.9}$$

| | Theory of the integers | Group theory |
|---|---|---|
| Language $L$: | $0,\ S,\ +,\ \cdot,\ =$ | $1,\ \cdot,\ ()^{-1},\ =$ |
| $L$-Terms: | $x_1 + 0$ and $S(x_1 \cdot x_2)$ | $x_1 \cdot x_2$ and $(x_1 \cdot 1)^{-1}$ |
| Atomic $L$-formulas: | $x_1 + 0 = S(x_1 \cdot x_2)$ | $x_1 \cdot x_2 = (x_1 \cdot 1)^{-1}$ |
| $L$-formulas: | $\forall x_1\, \exists x_2\, (x_1 + 0 = S(x_1 \cdot x_2))$ | $\exists x_1\, \forall x_2\, (x_1 \cdot x_2 = (x_1 \cdot 1)^{-1})$ |

Figure III.1: Examples of the components of the definition of first-order formulas

Here $Prime(y)$ means the result of substituting $y$ for each occurrence of $x$ in $Prime(x)$.[4] The formula (III.9) does not say directly that there are infinitely many primes, as the property of infinitude cannot be directly expressed in first-order logic. Instead, (III.9) states that there are arbitrarily large primes.

In contrast to group theory, where mathematicians study many, very different, groups, there is only one set of integers. As is shown later, first-order logic is very expressive for the integers. Nonetheless, there are inherent difficulties in expressing, axiomatizing, and proving some important concepts about the integers. In fact, it is impossible to rule out "nonstandard integers" which satisfy exactly the same first-order properties as the integers. It is also impossible to give a decidable set of axioms that implies exactly the true first-order statements about the integers. These limitations will be established later, in the context of the Completeness and Compactness Theorems and in the context of undecidability and the Gödel Incompleteness Theorems.

## III.2 First-order Syntax

This section defines the syntax of first-order formulas. The definition goes through a four-step process. First, we define the notion of a language $L$. A language specifies which non-logical symbols may be used in formulas. The symbols $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\forall$, $\exists$ and $=$ can be considered *logical symbols* since they have a prescribed meaning in all settings. The non-logical symbols however can have different meanings in different settings. For example, $\cdot$ can mean different things in different groups and yet another thing in the integers. Once a language $L$ is fixed, we define the notions of $L$-terms, atomic $L$-formulas, and finally $L$-formulas. Figure III.1 shows examples of these four concepts.

First-order formulas are *expressions*, i.e. strings of symbols, that are built from logical symbols, the equality sign, non-logical symbols, variables, and parentheses. The logical symbols include propositional connectives, the existential quantifier $\exists$, and the universal quantifier $\forall$. Variables $x_i$ are intended to

[4] For a detailed discussion of substitution, see Section III.6. There the formula $Prime(y/x)$ will mean the result of substituting $y$ for $x$ in $Prime$, and the notation $Prime(y)$ will be a convenient informal notation.

range over a domain of "objects" or "individuals". The non-logical symbols are specified by a first-order language. For example, for the theory of groups, we might take the language to consist of $\cdot$, $^{-1}$, 1, and $=$. Or for the theory of the integers, we might use the language 0, $S$, $+$, $\cdot$ and $=$. The specification of the non-logical symbols and their attributes is called a language:[5]

**Definition III.1.** A *language* $L$ is a set of symbols along with information about what kind of symbols they are and their arities. (The "arity" of a function or predicate is the number of arguments it takes.) The following kinds of symbols can be constituents of a language.

(a) ***Constant symbols***. These are intended to denote particular objects in the domain of discourse. (A constant symbol has arity zero.)

(b) ***Function symbols***, along their arities.

(c) ***Predicate symbols***, along with their arities. These are often called "relation symbols".

(d) ***The equality sign*** $=$. This is included in a language as a special binary predicate symbol unless it is explicitly excluded as a symbol.

**Example III.2.** The language of the theory of the integers described in the previous section had the following constituents:

(a) The constant symbol 0.

(b) The unary function symbol $S$ and the two binary function symbols $+$ and $\cdot$.

(c) Apart from equality, no predicate symbols.

(d) The binary predicate symbol for equality, $=$.

This language is denoted $L_{\mathrm{PA}}$. The subscript "PA" stands for "Peano Arithmetic", which will be defined later in Section VII.2.

The predicates $\leq$ and $\mathit{Prime}(-)$ were discussed above. These are not part of $L_{\mathrm{PA}}$. They can instead be expressed by $L_{\mathrm{PA}}$-formulas.

Once a language $L$ has been fixed, we can define the notion of a first-order formula over the language $L$. For this, we shall define first "terms", then "atomic formulas", and finally "formulas". Terms and formulas are expressions, namely strings of symbols. The symbols that may appear in an $L$-term or an $L$-formula include:

(a) The symbols of $L$,

(b) The propositional connectives $\neg$, $\vee$, $\wedge$, $\rightarrow$ and $\leftrightarrow$.

(c) The quantifier symbols $\forall$ and $\exists$.

(d) Variables $x_1, x_2, x_3, \ldots$.

(e) Parentheses and commas. Parentheses show precedence and delimit the scope of functions and predicates; commas separate arguments to functions and predicates.

[5] A language is sometimes called a "signature".

Predicate symbols and quantifier symbols do not appear in terms.

**Definition III.3.** Let $L$ be a language. The *$L$-terms* are inductively defined by

(a) For $i \geq 1$, the variable $x_i$ is a term.

(b) If $c$ is a constant symbol of $L$, then $c$ is a term.

(c) If $f$ is a $k$-ary function symbol of $L$ where $k \geq 1$, and if $t_1, t_2, \ldots, t_k$ are $L$-terms, then

$$f(t_1, t_2, \ldots, t_k)$$

is an $L$-term.[6]

As examples of infix notation, $+(0, S(x_1))$ and $\cdot(S(S(x_2)), +(S(x_3), S(x_4)))$ are $L_{\text{PA}}$-terms. Their intended meanings, in more readable notation, are $0+(x_1+1)$ and $(x_2+2)\cdot((x_3+1)+(x_4+1))$. These terms will denote objects in the domain of individuals, depending on what objects the variables $x_i$ are equal to.

**Definition III.4.** A term is *closed* if does not contain any variables. In other words, a closed term is a term that is built from only constant symbols and function symbols.

The simplest type of formulas is the atomic formulas.

**Definition III.5.** Let $L$ be a language. There are two ways of forming an *atomic $L$-formula*.

(a) If $P$ is a $k$-ary predicate symbol of $L$ and $t_1, \ldots, t_k$ are $L$-terms, then $P(t_1, \ldots, t_k)$ is an atomic $L$-formula.

(b) If $L$ contains the symbol $=$ and $t_1$ and $t_2$ are $L$-terms, then $t_1 = t_2$ is an atomic $L$-formula.

For example, $+(0, S(x_1)) = \cdot(S(S(x_2)), +(S(x_3), S(x_4)))$ is an atomic $L_{\text{PA}}$-formula. If we add $\leq$ as a binary predicate symbol, then

$$\leq (+(0, S(x_1)), \cdot(S(S(x_2)), +(S(x_3), S(x_4))))$$

is an atomic $L_{\text{PA}} \cup \{\leq\}$-formula. These two formulas could be either true or false, depending on what objects the variables $x_i$ are equal to.

Finally, we can define $L$-formulas.

**Definition III.6.** Let $L$ be a language. The $L$-formulas are inductively defined as follows.

(a) Any atomic $L$-formula is an $L$-formula.

[6]The use of parentheses and commas in the term $f(t_1, t_2, \ldots, t_k)$ is redundant. The redundancy comes from the fact that functions have a fixed, constant arity, and since we use prefix (Polish) notation for terms, and there is a simple parsing algorithm that could uniquely recover the underlying formula even if parentheses and commas were omitted. Nonetheless, in practice, parentheses and commas make it much easier to read and comprehend terms, so we choose to include them.

(b) If $A$ is an $L$-formula, then $\neg A$ is an $L$-formula.

(c) If $A$ and $B$ are $L$-formulas, then $(A \vee B)$, $(A \wedge B)$, $(A \to B)$ and $(A \leftrightarrow B)$ are $L$-formulas.

(d) If $A$ is an $L$-formula and $x_i$ is a variable, then $\forall x_i A$ and $\exists x_i A$ are $L$-formulas.

When the language $L$ is understood, we generally refer to $L$-terms and $L$-formulas as just "terms" and "formulas".

**Abbreviations for formulas.** It is often convenient to use informal notations for formulas that that are easier to read but do not fully obey the above definition of formulas. As an example, $\mathit{Prime}(x)$ as defined in (III.8) does not fit the formal definition for an $L_{\mathrm{PA}}$-formula; instead it is meant to be the readable transcription of an $L_{\mathrm{PA}}$-formula. To write the actual formula, we must add parentheses and we need to replace $x, y, z$ with some choice of variables $x_i$. If we, say, use $x_1, x_2, x_3$ for $x, y, z$, then it becomes

$$\neg x_1 = S(0) \wedge \forall x_2 \left(\exists x_3 \left(x_2 \cdot x_3 = x_1\right) \to \left(x_2 = S(0) \vee x_2 = x_1\right)\right). \tag{III.10}$$

The actual formula, with all parentheses and in prefix notation, is

$$\left(\neg x_1 = S(0) \wedge \forall x_2 \left(\exists x_3 \cdot (x_2, x_3) = x_1 \to \left(x_2 = S(0) \vee x_2 = x_1\right)\right)\right). \tag{III.11}$$

The formula (III.8) is an informal way of writing this.

This is an example of how we often write first-order formulas in more readable form, similarly to what was done for propositional logic earlier in Section I.2.

- First of all, we often omit parentheses. The order of precedence of operations is similar to propositional logic:
  (a) Negation $\neg$ and quantifiers $\forall$ and $\exists$ have the highest precedence.
  (b) $\wedge$ and $\vee$ have the second highest precedence.
  (c) $\to$ and $\leftrightarrow$ have the lowest precedence.
  (d) Binary connectives of the same precedence associate from right to left.
- Second, we often use variable names such as $x, y, z, \ldots$ instead of $x_1, x_2, x_3, \ldots$.
- Third, when it helps with readability, we often use infix notation to write binary functions (such as $+$ and $\cdot$) and binary predicates (such as $\leq$ and $<$). E.g., we write $s + t$ and $s \leq t$ as abbreviations of $+(s, t)$ and $\leq(s, t)$. We also use $s \neq t$ as a shorthand notation for $\neg s = t$.
- Fourth, we often add parentheses to make formulas more readable. A common example is when writing quantifiers with formulas that use infix notation. For instance, we might write $\forall x(0 \leq x)$ instead of $\forall x\, 0 \leq x$, just to make the formula more readable. In the same vein, we will generally write $\forall x(S(x) \neq x)$ instead of the more correct $\forall x S(x) \neq x$.

- Other conventions are used in the literature. For instance, parentheses are often used around quantifiers, to write $(\forall x)$ and $(\exists x)$ instead of just $\forall z$ and $\exists x$. Sometimes dots are used instead of parentheses; for example by writing $\exists x.x = 0 \vee S(x) = 0$ instead of $\exists x(x = 0 \vee S(x) = 0)$. We will avoid these conventions in this book however.

The use of variables such as $x, y, z$ to informally denote $x_1, x_2, x_3$ brings up a technical complication. Namely, how did we decide to use $x_1, x_2, x_3$ instead of other $x_i$'s? To illustrate this, consider the following three formulas, all of which are versions of (III.8):

$$x_1 \neq S(0) \wedge \forall x_2 \left(\exists x_3 \left(x_2 \cdot x_3 = x_1\right) \rightarrow x_2 = S(0) \vee x_2 = x_1\right) \tag{III.12}$$

$$x_1 \neq S(0) \wedge \forall x_5 \left(\exists x_6 \left(x_5 \cdot x_6 = x_1\right) \rightarrow x_5 = S(0) \vee x_5 = x_1\right) \tag{III.13}$$

$$x_4 \neq S(0) \wedge \forall x_5 \left(\exists x_6 \left(x_5 \cdot x_6 = x_4\right) \rightarrow x_5 = S(0) \vee x_5 = x_4\right) \tag{III.14}$$

The first two formulas have used $x_1$ for the variable $x$, and thus assert that $x_1$ is prime. The difference between these two formulas is that one uses $x_2, x_3$ for $y, z$ and the other uses $x_5, x_6$ for $y, z$. But these are quantified variables, usually called "bound" variables. Thus the formulas (III.12) and (III.13) have the same meaning, they both say that $x_1$ is prime, and they both equally well can serve as the formula $\mathit{Prime}(x_1)$.

On the other hand, (III.14) uses $x_4$ for $x$, and thus can serve as the formula $\mathit{Prime}(x_4)$ asserting that $x_4$ is prime, not that $x_1$ is prime. Therefore, (III.14) is not equivalent to the first two formulas! The variable $x$ in (III.8) is called a "free" variable. Likewise, $x_1$ in (III.12) and (III.13), and $x_4$ in (III.14) are "free" variables. These formulas assert that their free variable is a prime.

This topic of free variables will be revisited in Sections III.3 and III.6. For now, we just formally define what it means for an occurrence of a variable to be free or bound. But first, consider the formula

$$\forall x_1 \left(x_1 = 0 \vee P(x_1)\right) \wedge x_1 = x_1.$$

as an example. The two occurrences of $x_1$ in the subformula $(x_1 = 0 \vee P(x_1))$ are bound occurrences; they are "bound by" the quantifier $\forall x_1$. The two occurrences of $x_1$ in the subformula $x_1 = x_1$ are free occurrences. The symbol $x_1$ in the quantifier $\forall x_i$ is neither a free nor bound occurrence.

**Definition III.7.** Let $A$ be a formula, and let $x_i$ be an occurrence of a variable in $A$ that does not immediately follow a $\forall$ or $\exists$ symbol. The notions of $x_i$ being a *free* or *bound* are defined recursively as follows. At the same time, for $x_i$ a bound occurrence, we define which quantifier $x_i$ is *bound by*. In all cases, the occurrence of $x_i$ is either free or bound, but not both.

(a) If $A$ is an atomic formula, then the occurrence $x_i$ is free in $A$.

(b) If $A$ is $\neg B$ or $B \circ C$ for $\circ$ a propositional connective, then the occurrence of $x_i$ is free or bound in $A$ according to whether the corresponding occurrence in $B$ or in $C$ is free or bound. If the occurrence of $x_i$ is bound, then it is *bound by* the same quantifier in $A$ that it is bound by in $B$ or $C$.

(c) If $A$ is $\exists x_i B$ or $\forall x_i B$, then every occurrence of $x_i$ is bound in $A$ (and is not free in $A$). If an occurrence of $x_i$is free in $B$, then we say that that

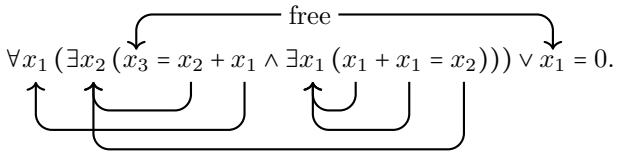



Figure III.2: The first occurrence of $x_1$ is bound by the first quantifier $\forall x_i$. The second and third occurrences of $x_1$ are bound by the final quantifier $\exists x_i$. The fourth occurrence of $x_1$ is free.

occurrence in $A$ is *bound by* the quantifier $\exists x_i$ or $\forall x_i$ at the start of $A$. Otherwise, the occurrence of $x_i$ is bound by the same quantifier in $A$ that it is bound by in $B$.

(d) If $j \neq i$ and $A$ is $\exists x_j B$ or $\forall x_j B$, then the occurrence of $x_i$ is free or bound in $A$ according to whether the corresponding occurrence in $B$ is free or bound. If the occurrence of $x_i$ is bound, then it is *bound by* the same quantifier in $A$ that it is bound by in $B$.

It is also possible to give a characterization of what it means for a variable to be free or bound in terms of the "scope" of a quantifier.

**Definition III.8.** Let $\forall x B$ or $\exists x B$ be a subformula occurring in $A$. The *scope* of the quantifier $\forall x$ or $\exists x$ is the subformula $B$. (The scope does not include the quantifier itself.)

Then, an occurrence of $x_i$ that does not immediately follow a quantifier symbol $\forall$ or $\exists$ is bound in $A$ if and only if it is in the scope of a quantifier $\exists x_i$ or $\forall x_i$ in $A$. It is bound by the leading quantifier of $\forall x_i B$ or $\exists x_i B$ if and only if it is not within the scope of any other occurrence of $\forall x_i$ or $\exists x_i$ in the subformula $B$.

**Example III.9.** Let $A$ be the formula $\exists x_2(x_2 + x_2 = x_1) \vee x_3 = 0$, which asserts that $x_1$ is even or $x_3$ is equal to 0. The occurrences of $x_1$ and $x_3$ are free in $A$. The two occurrences of $x_2$ in the term $x_2 + x_2$ are bound occurrences. The formula $A$ might also be (informally) written as $A(x_1, x_3)$ to indicate it is asserting properties of $x_1$ and $x_3$.

On the other hand, let $B$ be $\exists x_3(x_3 + x_3 = x_1) \vee x_3 = 0$. In this formula, there are two bound occurrences of $x_3$ and one free occurrence of $x_3$. It still asserts, like $A$, that $x_1$ is even or $x_3$ equals 0. Here $x_3$ is being used in two completely different, independent ways.

**Example III.10.** Figure III.2 shows an example of free and bound occurrences of variables in a formula the variable $x_1$ is used in two different ways as a quantified variable and once as a free variable.

Admittedly, it is bad form to use variables in multiple ways in the same formula like in Figure III.2, if for no other reason than because it is confusing

to a reader. But it is *permitted* by the syntax of first-order formulas and will need to be dealt with in the next section when giving the definition of truth.

Another complication is that if we want to adapt $A$ to say that "$x_2$ is even or $x_3 = 0$", we cannot just replace the $x_1$ in $A$ with $x_2$ because this would create a new bound occurrence of $x_2$. This problem will be addressed in Section III.6 on substitution by renaming the bound variable $x_2$.

**Definition III.11.** A first-order formula with no free occurrences of variables is called a *sentence*.

Sentences are also called "closed formulas".

For example, $\forall x(0 \leq x)$ is a sentence, whereas $0 \leq x$ is not. The second formula asserts something about its free variable $x$, namely that it is greater than or equal to zero. The first formula does not have any free variables, so it asserts something about truth in the domain of discourse.

**Unique readability and induction on formulas.** Similarly to the situation for propositional formulas, first-order terms and first-order formulas satisfy the *unique readability* property. Namely, every term can be *uniquely* written in one of the forms

(a) A constant symbol $c$,

(b) A variable symbol $x_i$, or

(c) A compound term $f(t_1, \ldots, t_k)$ for $f$ a function symbol and $t_1, \ldots, t_k$ terms.

In addition, every atomic subformula can be uniquely written as $t_1 = t_2$ or $P(t_1, \ldots, t_k)$ where $P$ is a $k$-ary predicate and the $t_i$'s are terms. Finally, every formula can be written uniquely in one of the form

(a) As an atomic formula,

(b) As $\neg B$, or $B \wedge C$, $B \vee C$, $B \to C$ or $B \leftrightarrow C$,

(c) As $\forall x_i B$ or $\exists x_i B$.

The inductive definitions of terms and formulas mean that we can use induction on the complexity of terms or formulas. The unique readability properties mean that we can also give recursive definitions of new functions based on the complexity of terms and formulas. In fact, Definition III.7 already used recursion on the complexity of formulas to define the notions of free and bound variables, and Definition III.8 implicitly used the unique readability of formulas.

The unique readability properties can be proved by induction on the complexity of formulas. As with propositional formulas (see Exercise I.38), the basic idea is that there are sufficient parentheses included to make the order of precedence of operations unambiguous.

Almost as important as unique readability is that there are efficient algorithms to parse terms and formulas. These algorithms can decide which of the forms (a)-(c) above apply to a term or a formula, and can even express the term or formula as a parse tree.

# III.3 Structures and the Definition of Truth

## III.3.1 Structures

The definition of truth is considerably more complicated for first-order logic than for propositional logic. In propositional logic, formulas contain the logical symbols $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ that have fixed meanings and variables $p_i$ that range over only two possible values. To define a truth value for a propositional formula, all that was necessary was to start with a truth assignment $\varphi$ that gives true/false values to the variables. In contrast, first-order formulas use the logical symbols $\neg, \vee, \wedge, \rightarrow, \leftrightarrow, \forall, \exists$ and the equality symbol $=$ that have fixed meanings. They also use variables $x_i$ that range over some domain of objects as well as constant, function, and predicate symbols. Before a first-order formula can be given a value, it is necessary to specify a domain of objects, to specify which objects are denoted by the constant symbols, and to specify the values of all the functions and predicates for all inputs from the domain. All this information is called an "interpretation" or a "structure". If there are free variables in the formula, it is also necessary to specify which objects are denoted by the free variables. This information is called an "object assignment".

The original definition of truth for first-order formulas in a general setting was given by A. Tarski in 1933. Perhaps surprisingly, this was well after first-order languages and proofs were well-developed. The definition of truth given below is quite close to Tarski's original conception. It proceeds in three stages. First, it is necessary to define the values of terms as objects in the domain of individuals. This uses the interpretations of functions and the values assigned to variables and constant symbols. Second, it is necessary to define the truth of atomic formulas. This uses the interpretations of predicate symbols. Finally, a recursive definition defines the truth of an arbitrary formula by using the usual meanings of the logical connectives $\neg, \vee, \wedge, \rightarrow, \leftrightarrow, \forall, \exists$.

But first, we must define the notion of "interpretation" or "structure". (The two terms are synonymous.) Interpretations or structures are also sometimes called "models". It is common to use a variety of different symbols to denote structures, including $\mathfrak{A}$, $\mathfrak{B}$, $\mathfrak{C}$ (the Fraktur letters "A", "B", and "C") or $\mathcal{M}, \mathcal{N}$ ("M" stands for "model") or $\mathcal{I}, \mathcal{J}$ ("I" stands for "interpretation"). We work with a fixed language $L$.

**Definition III.12.** Fix a first-order language $L$. An *$L$-structure* $\mathfrak{A}$ consists of the following items:

(a) A non-empty set called the *domain* or *universe* of $\mathfrak{A}$. This set is denoted $|\mathfrak{A}|$. This is the set of *objects* that variables can range over.

(b) For each constant symbol $c$ in $L$, a member of the universe $|\mathfrak{A}|$. This member is denoted $c^{\mathfrak{A}}$ and is called the *interpretation* of $c$ in $\mathfrak{A}$.

(c) For each $k$-ary predicate symbol $P$ in $L$, a subset of $|\mathfrak{A}|^k$; that is, a set of $k$-tuples of members of $|\mathfrak{A}|$. This subset is denoted $P^{\mathfrak{A}}$ and is called the *interpretation* of $P$ in $\mathfrak{A}$.

(d) For each $k$-ary function symbol $f$ in $L$, a subset of $|\mathfrak{A}|^{k+1}$ that is the graph of a function $f : |\mathfrak{A}|^k \to |\mathfrak{A}|$. This subset is denoted $f^{\mathfrak{A}}$ and is called the *interpretation* of $f$ in $\mathfrak{A}$.
What it means for $f^{\mathfrak{A}}$ to be "the graph of a function" is that $f^{\mathfrak{A}}$ is a set of $(k+1)$-tuples, and that for every sequence $a_1, \ldots, a_k$ of $k$ members of $|\mathfrak{A}|$, there is a unique $a_{k+1}$ such that $\langle a_1, \ldots, a_k, a_{k+1}\rangle \in f^{\mathfrak{A}}$.
The notation $f^{\mathfrak{A}}$ is used to denote both the graph of a function and the function itself. In particular, we write

$$f^{\mathfrak{A}}(a_1, \ldots, a_k)$$

to denote the unique value $a_{k+1}$ such that $\langle a_1, \ldots, a_k, a_{k+1}\rangle \in f^{\mathfrak{A}}$.

The $L$-structure thus consists of the specifications of $|\mathfrak{A}|$ and of all the values of $c^{\mathfrak{A}}$, $P^{|}$, and $f^{\mathfrak{A}}$. The intuition is that the constant symbol $c$ denotes the object $c^{\mathfrak{A}}$ in $|\mathfrak{A}|$. Likewise, the intuition is that $P^{\mathfrak{A}}$ is the set of tuples for which $P(a_1, \ldots, a_k)$ is true in $\mathfrak{A}$. Or, informally,[7]

$$P(a_1, \ldots, a_k) \text{ is true for } \mathfrak{A} \quad \text{if and only if} \quad \langle a_1, \ldots, a_k\rangle \in P^{\mathfrak{A}}.$$

Finally, the intuition is that

$$a_{k+1} = f(a_1, \ldots, a_k) \text{ is true for } \mathfrak{A} \quad \text{if and only if} \quad \langle a_1, \ldots, a_k, a_{k+1}\rangle \in f^{\mathfrak{A}},$$

namely, that $f(a_1, \ldots, a_k)$ is equal to $f^{\mathfrak{A}}(a_1, \ldots, a_k)$. These informal intuitions will be made formal below in the definition of truth.

**Example III.13.** Consider the language of arithmetic, $L_{\text{PA}}$, with non-logical symbols $0, S, +, \cdot$, as defined in Example III.2. We give two possible $L_{\text{PA}}$-structures. The first one, denoted $\mathcal{N}$ corresponds to the nonnegative integers. The universe $|\mathcal{N}|$ of $\mathcal{N}$ is the set of nonnegative integers, $S$ is interpreted as the "increment by 1" function, and the symbols 0, + and $\cdot$ have their usual interpretations. Formally, we let

$$\begin{aligned}
|\mathcal{N}| &= \mathbb{N}, \text{ i.e., the set of nonnegative integers.}\\
0^{\mathcal{N}} &= 0, \text{ i.e., the integer 0.}\\
S^{\mathcal{N}} &= \{\langle i, j\rangle : i, j \in \mathbb{N},\ j = i + 1\}\\
+^{\mathcal{N}} &= \{\langle i, j, k\rangle : i, j, k \in \mathbb{N},\ i + j = k\}\\
\cdot^{\mathcal{N}} &= \{\langle i, j, k\rangle : i, j, k \in \mathbb{N},\ i \cdot j = k\}
\end{aligned}$$

**Example III.14.** The second $L_{\text{PA}}$-structure is the set $\mathbb{Z}$ of all integers with the usual meanings for 0, + and $\cdot$. We denote this structure $\mathcal{Z}$. Formally, we let

$$\begin{aligned}
|\mathcal{Z}| &= \mathbb{Z}, \text{ i.e., the set of all integers.}\\
0^{\mathcal{Z}} &= 0, \text{ i.e., the integer 0.}\\
S^{\mathcal{Z}} &= \{\langle i, j\rangle : i, j \in \mathbb{Z},\ j = i + 1\}\\
+^{\mathcal{Z}} &= \{\langle i, j, k\rangle : i, j, k \in \mathbb{Z},\ i + j = k\}\\
\cdot^{\mathcal{Z}} &= \{\langle i, j, k\rangle : i, j, k \in \mathbb{Z},\ i \cdot j = k\}
\end{aligned}$$

[7] The notation $\langle a_1, \ldots, a_k\rangle$ denotes a $k$-tuple of objects. When $k = 1$, the convention is that $\langle a_1\rangle$ is the same object as $a_1$ itself, so we can use "$\langle a_1\rangle$" and "$a_1$" interchangeably.

The structures $\mathcal{N}$ and $\mathcal{Z}$ are not isomorphic, and the truth of sentences and formulas can be different in $\mathcal{N}$ and $\mathcal{Z}$. For instance, the two sentences

$$\forall x\, \exists y\, (S(y) = x) \qquad \text{and} \qquad \forall x\, \exists y\, (x + y = 0 \wedge y + x = 0)$$

are true in $\mathcal{Z}$ but false in $\mathcal{N}$. The first sentence states that every object $x$ is the successor of some $y$; the second states that every object has an additive inverse. Conversely, the negations of these two sentences are true in $\mathcal{N}$ and false in $\mathcal{Z}$.

**Example III.15.** Let $L$ be the language of groups with nonlogical symbols 1, $\cdot$ and $^{-1}$. The integers with addition mod 3 can be viewed as an $L$-structure $\mathcal{Z}_3$. For this, the universe is $|\mathcal{Z}_3| = \{0, 1, 2\}$ and the group operations are given by:

| $\cdot^{\mathcal{Z}_3}$ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 1 | 1 | 2 | 0 |
| 2 | 2 | 0 | 1 |

and

| | $(^{-1})^{\mathcal{Z}_3}$ |
|---|---|
| 0 | 0 |
| 1 | 2 |
| 2 | 1 |

The corresponding structure $\mathcal{Z}_3$ is formally specified as

$$\begin{aligned} 1^{\mathcal{Z}_3} &= 0 \quad \text{(the identity element for the group operation)} \\ \cdot^{\mathcal{Z}_3} &= \{\langle 0,0,0\rangle, \langle 0,1,1\rangle, \langle 0,2,2\rangle, \langle 1,0,1\rangle, \langle 1,1,2\rangle, \langle 1,2,0\rangle, \\ &\qquad \langle 2,0,2\rangle, \langle 2,1,0\rangle, \langle 2,2,1\rangle\} \\ (^{-1})^{\mathcal{Z}_3} &= \{\langle 0,0\rangle, \langle 1,2\rangle, \langle 2,1\rangle\} \end{aligned}$$

The last example is potentially confusing since the language $L$ uses multiplicative notation, but the structure is based on the group of order three which is usually described as an *additive* group, namely addition mod 3. However, the point is that the symbols of a language can stand for anything. That is, a $k$-ary function symbol can be interpreted in a structure as *any* $k$-ary function on the universe; a $k$-ary predicate symbol can be interpreted as *any* $k$-ary predicate on the universe (a set of $k$-tuples), and a constant symbol can be interpreted as *any* member of the universe. In the last example, the symbol $\cdot$ could be interpreted as any binary operation on the universe, and then first-order statements can express simple properties such as whether $\cdot$ is associative or commutative, etc. This is similar in spirit to the abstract approach taken in group theory or more generally in algebra.

The example chose the interpretation $\cdot^{\mathcal{Z}_3}$ of the binary function symbol $\cdot$ to be the usual addition operation on $\mathcal{Z}_3$, and the interpretation $1^{\mathcal{Z}_3}$ of the constant symbol 1 to be the identity element 0 of $\mathcal{Z}_3$. This is potentially confusing since "1" is being used in two ways: First, "1" is a *symbol* in the language $L$. Second, "1" is an *element* (in the universe) of $\mathcal{Z}_3$. These are completely different meanings for "1", and the reader should be careful to distinguish between them.

## III.3.2 Definition of truth

**Object assignments.** Our next goal is to define the truth of an $L$-formula or an $L$-sentence in a structure. Suppose, for example, that we wish to determine

whether the $L$-sentence

$$\forall x\, \exists y\, (x \cdot y = x) \tag{III.15}$$

is true in a given $L$-structure. Our definition of truth will use induction on the complexity of the formula (III.15). For this reason, we must define the truth of its subformula $\exists y(x \cdot y = x)$. But this is a *formula*, not a sentence, since it uses the free variable $x$. In particular, the truth of $\exists y(x \cdot y = x)$ depends on which object in the universe $|\mathfrak{A}|$ is denoted by $x$. This will be specified by a function called an "object assignment" that maps variables to objects in the universe:

**Definition III.16.** Let $\mathfrak{A}$ be a structure. An *object assignment* for $\mathfrak{A}$ is a function $\sigma$ with domain the set of variables $x_1, x_2, x_3, \ldots$ and with range the universe of $\mathfrak{A}$; namely,

$$\sigma : \{x_1, x_2, x_3, \ldots\} \to |\mathfrak{A}|.$$

The point of an object assignment $\sigma$ is that $\sigma(x_i)$ specifies which member of the universe is denoted by $x_i$.

**Denotation of terms.** The first step in the definition of truth is to define what objects are denoted by terms. We start with an object assignment $\sigma$ and extend it to have domain the set of $L$-terms. The idea is to use the interpretations of the constant symbols and function symbols as given by the structure $\mathfrak{A}$, and use these to compute the value of a term in the most straightforward way possible.

**Definition III.17.** Suppose $\sigma$ is an object assignment for the $L$-structure $\mathfrak{A}$. The function $\sigma$ is extended to have domain the set of all $L$-terms $t$ by defining $\sigma(t)$ by recursion as:

(a) If $t$ is a constant symbol $c \in L$, then $\sigma(t) = c^{\mathfrak{A}}$.

(b) If $t$ is of the form $f(t_1, t_2, \ldots, t_k)$ for a $k$-ary function symbol $f \in L$, then

$$\sigma(t) \;=\; f^{\mathfrak{A}}(\sigma(t_1), \sigma(t_2), \ldots, \sigma(t_k)).$$

**Example III.18.** Let $\mathcal{Z}_3$ be the structure described in Example III.15. Consider the term $(x \cdot y)^{-1} \cdot x$, and suppose $\sigma(x) = 1$ and $\sigma(y) = 2$. Then $\sigma(x \cdot y) = 0$, and $\sigma((x \cdot y)^{-1}) = 0$, and $\sigma((x \cdot y)^{-1} \cdot x) = 1$.

The value of $\sigma(t)$ depends on the structure $\mathfrak{A}$ in addition to $\sigma$ and $t$. However, we will generally be working with a particular $\mathfrak{A}$, so there will usually be no confusion in suppressing $\mathfrak{A}$ in the notation and writing just "$\sigma(t)$".

Note that the value of $\sigma(t)$ depends only on the values of $\sigma(x_i)$ for the variables $x_i$ that actually appear in $t$. This should be obvious of course and can be proved by induction on the complexity of terms. If $t$ is a closed term, then $\sigma(t)$ is independent of the choice of $\sigma$, it only depends on $\mathfrak{A}$. In that case, we write $t^{\mathfrak{A}}$ instead of $\sigma(t)$:

**Definition III.19.** If $t$ is a closed term, then $t^{\mathfrak{A}}$ denotes $\sigma(A)$ where $\sigma$ is an arbitrary object assignment.

**Definition of truth for atomic formulas.** The definition of truth for general $L$-formulas starts with the definition of truth for atomic formulas. We use the notation $\mathfrak{A} \vDash A[\sigma]$ to indicate that $A$ is true in the structure with the object assignment $\sigma$.

**Definition III.20.** Let $\mathfrak{A}$ be an $L$-structure, $\sigma$ be an object assignment, and $A$ be an atomic $L$-formula. We define $A$ *is true in* $\mathfrak{A}$ *with object assignment* $\sigma$, written $\mathfrak{A} \vDash A[\sigma]$ , as follows:

(a) If $A$ is $s = t$, where $s$ and $t$ are $L$-terms then,

$$\mathfrak{A} \vDash A[\sigma] \quad \text{if and only if} \quad \sigma(s) = \sigma(t).$$

(b) If $A$ is $P(t_1, \ldots t_k)$ where $P$ is a $k$-ary predicate and the $t_i$'s are $L$-terms, then

$$\mathfrak{A} \vDash A[\sigma] \quad \text{if and only if} \quad \langle \sigma(t_1), \ldots, \sigma(t_k) \rangle \in P^{\mathfrak{A}}.$$

**Definition of truth for general first-order formulas.** We are almost ready to give the definition of truth for general formulas $A$. We will continue to write $\mathfrak{A} \vDash A[\sigma]$ to denote that $A$ is true in $\mathfrak{A}$ with object assignment $\sigma$. First, however, we need to define the notion of an "$x_i$-variant" of an object assignment.

**Definition III.21.** Let $\sigma$ be an object assignment. An $x_i$*-variant* of $\sigma$ is an object assignment $\tau$ such that $\tau(x_j) = \sigma(x_j)$ for all $j \neq i$.

An $x_i$-variant is allowed to have $\tau(x_i)$ be any member of the universe $|\mathfrak{A}|$. Note that it is permitted that $\tau(x_i)$ is equal to $\sigma(x_i)$; in this case, $\tau$ and $\sigma$ are identical.

We use $x_i$-variants to handle the definition of truth for quantifiers. To motivate this, let $A$ be the formula $\forall x_i (x_j \leq x_i)$ and suppose we wish to define $\mathfrak{A} \vDash A[\sigma]$, namely the truth of $A$ in a structure $\mathfrak{A}$ with object assignment $\sigma$. The variable $x_j$ is free in $A$, so the value of $\sigma(x_j)$ is needed when defining $\mathfrak{A} \vDash A[\sigma]$. However, $x_i$ is not free in $\sigma$, and thus $\sigma(x_i)$ will have no bearing on whether $\mathfrak{A} \vDash A[\sigma]$ holds. The recursive definition of truth will define $\mathfrak{A} \vDash A[\sigma]$ in terms of $\mathfrak{A} \vDash A[\tau]$ where $\tau$ ranges over all $x_i$-variants of $\sigma$. In other words, the value $\tau(x_j)$ is kept fixed and equals $\sigma(x_j)$, but all possible values for $\tau(x_i)$ must be considered. The point is that since $A$ starts with quantifier $\forall x_i$, it is necessary to consider all possible values for $\tau(x_i)$ to evaluate the truth of $A$.

The condition $\mathfrak{A} \vDash A[\sigma]$ is defined inductively, using recursion on the complexity of $A$. The case base is when $A$ is atomic, and the inductive steps handle the propositional connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$ and the quantifiers $\forall, \exists$.

We write $\mathfrak{A} \nvDash A[\sigma]$ to denote that $\mathfrak{A} \vDash A[\sigma]$ is false.

**Definition III.22.** Let $\mathfrak{A}$ be an $L$-structure, $\sigma$ be an object assignment, and $A$ be an atomic $L$-formula. If $A$ is atomic, $\mathfrak{A} \vDash A[\sigma]$ has already been defined. For non-atomic $A$, we recursively define $A$ *is true in* $\mathfrak{A}$ *with object assignment* $\sigma$, written $\mathfrak{A} \vDash A[\sigma]$, as follows:

(a) If $A$ is $\neg B$, then $\mathfrak{A} \vDash A[\sigma]$ holds if and only if

$$\mathfrak{A} \nvDash B[\sigma].$$

(b) If $A$ is $(B \vee C)$, then $\mathfrak{A} \vDash A[\sigma]$ holds if and only if

$$\mathfrak{A} \vDash B[\sigma] \text{ or } \mathfrak{A} \vDash C[\sigma].$$

(c) If $A$ is $(B \wedge C)$, then $\mathfrak{A} \vDash A[\sigma]$ holds if and only if

$$\mathfrak{A} \vDash B[\sigma] \text{ and } \mathfrak{A} \vDash C[\sigma].$$

(d) If $A$ is $(B \to C)$, then $\mathfrak{A} \vDash A[\sigma]$ holds if and only if

$$\mathfrak{A} \nvDash B[\sigma] \text{ or } \mathfrak{A} \vDash C[\sigma]$$

(e) If $A$ is $(B \leftrightarrow C)$, then $\mathfrak{A} \vDash A[\sigma]$ holds if and only if

$$\mathfrak{A} \vDash B[\sigma] \text{ and } \mathfrak{A} \vDash C[\sigma] \text{ are both true or both false.}$$

(f) If $A$ is $\exists x_i\, B$, then $\mathfrak{A} \vDash A[\sigma]$ holds if and only if

$$\text{for some } x_i\text{-variant } \tau \text{ of } \sigma,\ \mathfrak{A} \vDash B[\tau].$$

(g) If $A$ is $\forall x_i\, B$, then $\mathfrak{A} \vDash A[\sigma]$ holds if and only if

$$\text{for every } x_i\text{-variant } \tau \text{ of } \sigma,\ \mathfrak{A} \vDash B[\tau].$$

The definition of truth for the propositional connectives in (a)-(e) is identical to the way Definition I.4 defined the truth of propositional variables. Cases (f) and (g) handle the truth of quantified formulas by using $x_i$-variants $\tau$ of $\sigma$. As already discussed, these $x_i$-variants change the object assignment $\sigma$ to map $x_i$ to an arbitrary member of $|\mathfrak{A}|$.

The truth of $\mathfrak{A} \vDash A[\sigma]$ depends only the values of $\sigma(x_i)$ for those variables $x_i$ that appear free in $A$; it does not depend on the values $\sigma(x_i)$ for variables $x_i$ that do not appear free in $A$. Informally, this is because the $x_i$-variants used to handle quantifiers $\forall x_i$ and $\exists x_i$ "overwrite" the value of $\sigma(x_i)$. It also reflects our informal understanding that bound variables could be renamed arbitrarily and not affect the truth of $\mathfrak{A} \vDash A[\sigma]$; that is, bound variables can be renamed arbitrarily as long as there is no clash between variables. For a precise statement of this, see Theorem III.64 later in this chapter.

The next theorem formalizes the idea that the truth of $\mathfrak{A} \vDash A[\sigma]$ depends only the values of $\sigma(x_i)$ for $x_i$'s that appear free in $A$.

**Theorem III.23.** *Let $\sigma$ and $\tau$ be object assignments such that $\sigma(x_i) = \tau(x_i)$ for every variable $x_i$ that appears free in $A$. Then $\mathfrak{A} \vDash A[\sigma]$ if and only if $\mathfrak{A} \vDash A[\tau]$.*

The proof of Theorem III.23 is left to the reader. The first stage of the proof uses induction on terms to prove that, for all $\sigma$, the value of $\sigma(t)$ depends only on the values $\sigma(x_i)$ for variables $x_i$ that actually appear in $t$. The theorem is then proved by induction on the complexity of $A$.

Recall that a sentence is a formula with no free occurrences of variables. Theorem III.23 immediately gives:

**Corollary III.24.** *If $A$ is a sentence, then the truth of $\mathfrak{A} \vDash A[\sigma]$ does not depend on the choice of $\sigma$.*

The corollary justifies the next definition; namely, we can omit using an object assignment $\sigma$ when discussing the truth of a *sentence*.

**Definition III.25.** Let $A$ be an $L$-sentence and $\mathfrak{A}$ an $L$-structure. The $A$ *is true in* $\mathfrak{A}$, written $\mathfrak{A} \vDash A$, if and only if $A$ is true in $\mathfrak{A}$ under any object assignment $\sigma$.

**Example III.26.** This example and the next example work with the group $\mathcal{Z}_3$ of Example III.15. First, we consider the following sentence $A$:

$$\exists x_1 \, \forall x_2 \, (x_1 \cdot x_2 = x_2). \tag{III.16}$$

Note that this asserts the existence of a left identity. We claim that $\mathcal{Z}_3 \vDash A[\sigma]$ for every object assignment $\sigma$. Since $A$ is a sentence, we can also write this as $\mathcal{Z}_3 \vDash A$, without needing to mention the object assignment $\sigma$. To prove this, we must show that

$$\mathcal{Z}_3 \vDash \forall x_2 \, (x_1 \cdot x_2 = x_2)[\tau] \tag{III.17}$$

is true for some $x_1$-variant $\tau$ of $\sigma$. There are three $x_1$-variants that might be considered, as we can have $\tau(x_1) = 0$ or $\tau(x_1) = 1$ or $\tau(x_1) = 2$. Since are trying to show that (III.16) is true, and since $x_1$ is *existentially* quantified, it is enough to work with only one $x_1$-variant $\tau$, namely the one that makes (III.17) true. We of course take $\tau(x_1)$ equal to 0 since this is the identity element.

We now need to show that (III.17) holds for this $\tau$. Since $x_2$ is *universally* quantified, we must consider all three $x_2$-variants $\pi$ of $\tau$, and show that

$$\mathcal{Z}_3 \vDash (x_1 \cdot x_2 = x_2)[\pi] \tag{III.18}$$

holds for all three. For instance, we can set $\pi(x_2) = 1$. To evaluate the truth of (III.18), we find that $\pi(x_1 \cdot x_2) = 1$ and $\pi(x_2) = 1$. Therefore, by the definition of truth for the atomic formula $x_1 \cdot x_2 = x_2$, we have that (III.18) is true when $\pi(x_2) = 1$. Similar arguments work when $\pi(x_2) = 0$ and $\pi(x_2) = 2$.

**Example III.27.** Now consider the sentence $B$ equal to $\exists x_1 \, \forall x_2 \, (x_1 \cdot x_2 = x_1)$. We claim that

$$\mathcal{Z}_3 \nvDash \exists x_1 \, \forall x_2 \, (x_1 \cdot x_2 = x_1)[\sigma]$$

for any object assignment $\sigma$. Since $B$ is a sentence, we can also write this as $\mathcal{Z}_3 \nvDash B$, without needing to mention $\sigma$. To prove this, we must show that

$$\mathcal{Z}_3 \nvDash \forall x_2 \, (x_1 \cdot x_2 = x_1)[\tau] \tag{III.19}$$

is false for every $x_1$-variant $\tau$ of $\sigma$. There are three $x_1$-variants that must be considered, namely with $\tau(x_1) = 0$, $\tau(x_1) = 1$ and $\tau(x_1) = 2$. We do just $\tau(x_1) = 0$; the other two cases are similar.

To prove that (III.19) is false for this $\tau$, we must find a $x_2$-variant $\pi$ of $\tau$ such that

$$\mathcal{Z}_3 \vDash (x_1 \cdot x_2 = x_1)[\pi] \tag{III.20}$$

is false. It is enough to find one such $\pi$ since $x_2$ is *universally* quantified. We can take $\pi(x_2) = 1$ for instance. Now, $\pi(x_1 \cdot x_2) = 1$ and $\pi(x_1) = 0$. Therefore, by the definition of truth for the atomic formula $x_1 \cdot x_2 = x_1$, we have that (III.18) is false, as desired.

## III.4 Satisfaction and Logical Implication

The definition of truth for first-order formulas was just given in the previous section. This lets us define the notions of "satisfiability", "logical validity" and "logical implication". We start with validity.

**Validity of first-order formulas.** We assume that a language $L$ has been fixed, and all formulas are $L$-formulas and structures are $L$-structures.

**Definition III.28.** Let $A$ be a first-order formula. We say $A$ is *logically valid*, or just *valid* for short, and write $\vDash A$, provided $\mathfrak{A} \vDash A[\sigma]$ holds for all structures $\mathfrak{A}$ and object assignments $\sigma$.

If $A$ is a first-order sentence, we equivalently have that $A$ is *valid*, written $\vDash A$ provided that for all structures $\mathfrak{A}$, we have $\mathfrak{A} \vDash A$.

**Example III.29.** Let $L$ contain the binary predicate $P$ and the unary function $f$. Then

$$\begin{aligned}
&\vDash \forall x(x = x)\\
&\vDash \forall x\, \exists y(x = y)\\
&\vDash \forall x\, \forall y(x = y \wedge P(x, z) \to P(y, z))\\
&\vDash \forall x\, \forall y(f(x) = f(y)) \to \forall x(f(x) = f(f(x))).
\end{aligned}$$

All these examples are about properties of equality. Note that the first, second, and fourth formulas are sentences, whereas the third one contains the free variable $z$ and thus is not a sentence. One could equivalently write

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

As we define next, this is called a "generalization" of $\forall x\, \forall y\, (x = y \wedge P(x, z) \to P(y, z))$.

**Definition III.30.** Suppose $A$ is a formula and $x_{i_1}, \ldots, x_{i_k}$ is a sequence of variables, with $k \geq 0$. The formula

$$\forall x_{i_1}\, \forall x_{i_2} \cdots \forall x_{i_k} A \tag{III.21}$$

is called a *generalization* of $A$. If the variables $x_{i_1}, \ldots, x_{i_k}$ are distinct and enumerate the variables that appear free in $A$, then (III.21) is called the *universal closure* of $A$.[8]

[8] Strictly speaking, it should be called *a* universal closure of $A$, since there can be multiple ways to order the variables $x_{i_1}, \ldots, x_{i_k}$. However, it is more common to call it *the* universal closure. At any rate, no matter how the variables are ordered, the universal closures are all logically equivalent to each other. If necessary, the definition of universal closure could be disambiguated by requiring $i_1 < i_2 < \cdots < i_k$.

**Theorem III.31.** *Let $A$ be a formula and $x_i$ be a variable. Then $\vDash A$ holds if and only if $\vDash \forall x_i A$. Therefore, if $B$ is a generalization of $A$, then $\vDash A$ holds if and only if $\vDash B$ holds.*

*Proof.* Let $B$ be $\forall x_i\, A$. Then $\vDash B$ holds if and only if for all structures $\mathfrak{A}$, and all $x_i$-variants $\tau$ of all object assignments $\sigma$, we have $\mathfrak{A} \vDash A[\tau]$. And $\vDash A$ holds if and only if, for all structures $\mathfrak{A}$ and all objects assignments $\tau$, we have $\mathfrak{A} \vDash A[\tau]$. But these two ways of selecting $\tau$ give the same possibilities for $\tau$. Therefore $\vDash B$ is equivalent to $\vDash A$.

The second part of the theorem is proved by induction on the number $k$ of quantifiers added to $A$ in the generalization. □

Theorem III.31 means that the question of whether a formula $A$ is valid can be reduced to the question of whether the universal closure of $A$ is valid. The universal closure of $A$ is of course a sentence. We will next discuss the notion of implication between sentences, and only afterward define implication between formulas.

**Logical implication from first-order sentences.** Let $\Gamma$ be a set of sentences, and $A$ be either a sentence or a general formula. We shall define $\Gamma$ "logically implies" $A$ whenever the truth of $\Gamma$ implies the truth of $A$. This is denoted $\Gamma \vDash A$ using the same notation as was used for tautological implication. However, logical implication is a much more powerful notion than tautological implication. We continue to work with a fixed language $L$. First, we define the crucial notion of a "model" of $\Gamma$.

**Definition III.32.** Let $\Gamma$ be a set of sentences and $\mathfrak{A}$ a structure. Then $\mathfrak{A}$ is called a *model* of $\Gamma$, written $\mathfrak{A} \vDash \Gamma$ if, for every sentence $B \in \Gamma$, we have $\mathfrak{A} \vDash B$.

When this holds, we say that $\Gamma$ is *satisfied by* $\mathfrak{A}$. We also say that $\Gamma$ is *satisfiable*.

**Definition III.33.** Let $\Gamma$ be a set of sentences and $A$ be a sentence. Then $\Gamma$ *logically implies* $A$ provided that for every structure $\mathfrak{A}$, if $\mathfrak{A} \vDash \Gamma$, then $\mathfrak{A} \vDash A$.

More generally, suppose $\Gamma$ is a set of sentences and $A$ is a formula. Then Then $\Gamma$ *logically implies* $A$ provided that for every structure $\mathfrak{A}$, if $\mathfrak{A} \vDash \Gamma$, then $\mathfrak{A} \vDash A[\sigma]$ for every object assignment $\sigma$. In this case, we also say that $A$ is a *logical consequence* of $\Gamma$.

We use the double turnstile notation $\Gamma \vDash A$ to denote that $\Gamma$ logically implies $A$.

Note that the double turnstile sign, $\vDash$ is being used in two different (closely related) ways. The first way, $\mathfrak{A} \vDash \Gamma$ or $\mathfrak{A} \vDash A$, indicates what is true in the single structure $\mathfrak{A}$. The second way, $\Gamma \vDash A$, indicates that $A$ follows from $\Gamma$ in all structures that satisfy $\Gamma$. That is, the first way is a statement about a single structure $\mathfrak{A}$; the second way is a statement about all possible models of $\Gamma$. These two different usages are well-established but potentially confusing. It is important to watch out for the distinction between the two meanings.

Just as was done in propositional logic, it is common to take liberties in notation by omitting set brackets, and writing things like $A, B \vDash C$ and $\Gamma, A \vDash B$ instead of $\{A, B\} \vDash C$ and $\Gamma \cup \{A\} \vDash B$.

**Definition III.34.** Let $A$ and $B$ be sentences. We say $A$ and $B$ are *logically equivalent* provided that both $A \vDash B$ and $B \vDash A$ hold. We write $A \vDash\dashv B$ to denote that $A$ and $B$ are logically equivalent.

We use the notation $\Gamma \nvDash A$ to indicate that $\Gamma$ does not logically imply $A$.

**Example III.35.** Let $L$ contain the binary predicate $P$ and the unary function $f$. Then

(a) $\forall x\, \forall y\, P(x,y) \vDash P(x, f(x))$,
(b) $\forall x\, \forall y\, P(x,y) \vDash \forall x\, P(x,x)$,
(c) $\forall x\, \forall y\, P(x,y) \vDash \forall x\, P(x, f(x))$,
(d) $\forall x\, \forall y\, P(x,y) \vDash\dashv \forall y\, \forall x\, P(x,y)$,
(e) $\exists y\, \forall x\, P(x,y) \vDash \forall x\, \exists y\, P(x,y)$, and
(f) $\forall x\, \exists y\, P(x,y) \nvDash \exists y\, \forall x\, P(x,y)$.

It is left to the reader to check that these logical implications are correct. The non-implication of item (f) will be discussed again in Example III.54.

As mentioned earlier, "$\forall x$" has the same meaning as "$\neg\exists x\neg$". This is formalized by the next theorem; a slight variation is proved later as Theorem III.52.

**Theorem III.36.** *Let $A$ be a formula.*

(a) $\forall x\, A \vDash\dashv \neg\exists x\neg A$.
(b) $\exists x\, A \vDash\dashv \neg\forall x\neg A$.

*Proof.* We prove (b); the proof of (a) is similar. We must show that $\mathfrak{A} \vDash \exists x\, A$ holds if and only if $\mathfrak{A} \vDash \neg\forall x\neg A$ holds. This is proved as follows.

$$
\begin{aligned}
\mathfrak{A} \vDash \neg\forall x\neg A[\sigma] \quad &\Leftrightarrow \quad \mathfrak{A} \nvDash \forall x\neg A[\sigma] \\
&\Leftrightarrow \quad \text{It is not true that } \mathfrak{A} \vDash \forall x\neg A[\sigma] \\
&\Leftrightarrow \quad \text{It is not true that for all } x\text{-variants } \tau \text{ of } \sigma,\ \mathfrak{A} \vDash \neg A[\tau] \\
&\Leftrightarrow \quad \mathfrak{A} \nvDash \neg A[\tau], \text{ for some } x\text{-variant } \tau \text{ of } \sigma \\
&\Leftrightarrow \quad \mathfrak{A} \vDash A[\tau], \text{ for some } x\text{-variant } \tau \text{ of } \sigma \\
&\Leftrightarrow \quad \mathfrak{A} \vDash \exists x\, A[\sigma].
\end{aligned}
$$

□

Theorem III.31 also applies to logical implication from a set of sentences:

**Theorem III.37.** *Suppose $\Gamma$ is a set of sentences, $A$ is a formula, and $\Gamma \vDash A$. Let $B$ be a generalization of $A$. Then $\Gamma \vDash B$.*

The proof of this is essentially identical to the proof of Theorem III.31, and is left to the reader.

**Definition III.38.** Let $\Gamma$ and $\Delta$ be sets of sentences. Then $\Gamma \vDash \Delta$ means that $\Gamma \vDash A$ for every $A \in \Delta$. And, $\Gamma \vDash\dashv \Delta$ means that $\Gamma \vDash \Delta$ and $\Delta \vDash \Gamma$

The same notations can be used when $\Gamma$ and $\Delta$ are sets of formulas; this depends on the next definitions.

**Logical implication from first-order formulas.** Now we define what it means for $\Gamma$ to logically imply $A$ when $\Gamma$ is a set of formulas (instead of a set of sentences). This is definition is somewhat fraught because there are two different conventions that might be adopted. The definition we adopt is the following.

**Definition III.39.** Let $\Gamma$ be a set of formulas. Further let $\mathfrak{A}$ a structure and $\sigma$ be an object assignment. Then the pair $(\mathfrak{A}, \sigma)$ *satisfies* $\Gamma$, written $\mathfrak{A} \vDash \Gamma[\sigma]$ if, for every formula $B \in \Gamma$, we have $\mathfrak{A} \vDash B[\sigma]$.

When this holds, we say that $\Gamma$ is *satisfied by* $(\mathfrak{A}, \sigma)$. We also say that $\Gamma$ is *satisfiable*.

**Definition III.40.** Let $\Gamma$ be a set of formulas and $A$ be a formula. Then $\Gamma$ *logically implies* $A$, written $\Gamma \vDash A$, if and only if for every structure $\mathfrak{A}$ and every object assignment $\sigma$, if $\mathfrak{A} \vDash \Gamma[\sigma]$ then $\mathfrak{A} \vDash A[\sigma]$. In other words, $\Gamma \vDash A$ holds provided that, for every pair $(\mathfrak{A}, \sigma)$ satisfying $\Gamma$, we have $\mathfrak{A} \vDash A[\sigma]$. When $\Gamma \vDash A$ holds, we also say that $A$ is a *logical consequence* of $\Gamma$.

When $\Gamma$ is a set of sentences, there are two definitions of logical implication ($\Gamma \vDash A$). The first was given in Definition III.32; the second just given in in Definition III.40 since $\Gamma$ is also a set of formulas. (A sentence is always a formula, but not necessarily vice-versa.) It is easy to see that the two definitions coincide for $\Gamma$ a set of sentences since the truth of $\mathfrak{A} \vDash \Gamma[\sigma]$ does not depend on $\sigma$.

**Example III.41.** Let $P$ be a unary predicate symbol. Some simple examples of logical implication and non-implication include:

(a) $x = y \vDash P(x) \leftrightarrow P(y)$.
This holds, since if $\mathfrak{A} \vDash (x = y)[\sigma]$, then $\sigma(x) = \sigma(y)$, so of course $\sigma(x) \in P^{\mathfrak{A}}$ if and only if $\sigma(y) \in P^{\mathfrak{A}}$.

(b) $P(x) \nvDash P(y)$.
This is because there are $(\mathfrak{A}, \sigma)$ such that $\sigma(x) \neq \sigma(y)$, and $\mathfrak{A} \vDash P(x)[\sigma]$ and $\mathfrak{A} \vDash \neg P(y)[\sigma]$. (See also Example III.57 below.)

(c) $P(x) \nvDash \forall x\, P(x)$.
This holds for similar reasons as (b).

(d) $P(x) \vDash \exists x\, P(x)$.
If $\mathfrak{A} \vDash P(x)[\sigma]$, then the definition of truth implies that $\mathfrak{A} \vDash \exists x\, P(x)[\sigma]$. (And, in fact, since $\exists x\, P(x)$ is a sentence, we have $\mathfrak{A} \vDash \exists x\, P(x)$.)

(e) $x = y \vDash y = x$.
This follows from the definition of truth for the formulas $x = y$ and $y = x$.

**Example III.42.** Here is a less simple example that will be useful later. Suppose $A$ and $C$ are formulas and that the variable $x$ does not appear free in $C$. Then

$$\forall x\,(C \to A) \vDash\dashv C \to \forall x\, A. \qquad \text{(provided } x \text{ is not free in } C\text{)}$$

To prove this, let $\mathfrak{A}$ be a structure and $\sigma$ be an object assignment. We must prove that $\mathfrak{A} \vDash \forall x\,(C \to A)[\sigma]$ holds if and only if $\mathfrak{A} \vDash (C \to \forall x\, A)[\sigma]$ holds.

The proof splits into two cases. First, assume that $\mathfrak{A} \not\models C[\sigma]$. By the definition of truth, $\mathfrak{A} \models (C \to \forall x\, A)[\sigma]$ holds. Furthermore, since $x$ is not free in $C$, $\mathfrak{A} \not\models C[\tau]$ for every $x$-variant $\tau$ of $\sigma$. Therefore, $\mathfrak{A} \models (C \to A)[\tau]$ for every $x$-variant of $\sigma$. Therefore, once again by the definition of truth, $\mathfrak{A} \models \forall x\, (C \to A)[\sigma]$ holds. That proves the first case.

Second, assume $\mathfrak{A} \models C[\sigma]$. Since $x$ is not free $C$, $\mathfrak{A} \models C[\tau]$, for all $x$-variants $\tau$ of $\sigma$. By the definition of truth, $\mathfrak{A} \models (C \to \forall x\, A)[\sigma]$ holds if and only if $\mathfrak{A} \models (\forall x\, A)[\sigma]$. Similarly, for every $x$-variant $\tau$ of $\sigma$, $\mathfrak{A} \models (C \to A)[\tau]$ if and only if $\mathfrak{A} \models A[\tau]$. Using the definition of truth twice, this means that $\mathfrak{A} \models \forall x\, (C \to A)[\sigma]$ holds if and only if $\mathfrak{A} \models \forall x\, A[\sigma]$. That completes the second case. □

This example is one of the results that will support the conversion of formulas to prenex form. For related results, see Lemma III.80 later in this chapter.

The next theorem gives some simple observations about logical implication.

**Theorem III.43.** *Let $\Gamma$ and $\Pi$ be sets of formulas such that $\Gamma \subseteq \Pi$.*
(a) *If $\Pi$ contains only sentences and $\mathfrak{A} \models \Pi$, then $\mathfrak{A} \models \Gamma$.*
(b) *If $(\mathfrak{A}, \sigma)$ satisfies $\Pi$, then it also satisfies $\Gamma$.*
(c) *If $\Pi$ is satisfiable, then $\Gamma$ is satisfiable.*
(d) *If $\Gamma \models A$, then $\Pi \models A$.*

Theorem III.43 is immediate from the definitions.

**The semantic deduction theorem.** We now state the semantic version of the Deduction Theorem for first-order logic. This is analogous to the Semantic Deduction Theorem for propositional logic that was proved earlier as Theorem I.16. It is the *semantic* version of the actual Deduction Theorem, which will be discussed in Section IV.2.3. (That Deduction Theorem, however, will require that $A$ is a sentence.)

**Theorem III.44** (Semantic Deduction Theorem)**.** *Let $\Gamma$ be a set of formulas and $A$ and $B$ be formulas. Then $\Gamma \models A \to B$ holds if and only if $\Gamma, A \models B$ holds.*

Note that the theorem also holds for $\Gamma$ a set of sentences by the remark after Definition III.40.

*Proof.* By the definitions of truth and logical implication, both $\Gamma \models A \to B$ and $\Gamma, A \models B$ are equivalent to the condition that for any structure $\mathfrak{A}$ and object assignment $\sigma$, if $\mathfrak{A} \models \Gamma[\sigma]$ and $\mathfrak{A} \models A[\sigma]$ then $\mathfrak{A} \models B[\sigma]$. □

**Duality between satisfiability and validity.** Theorems I.20 and I.21 established a duality between tautological validity and satisfiability for propositional formulas. The same kind of duality holds for logical validity and satisfiability of first-order formulas.

**Theorem III.45.** *Let $\Gamma$ be a set of formulas and let $A$ be a formula.*
(a) *$\models A$ holds if and only if $\{\neg A\}$ is unsatisfiable.*
(b) *$\Gamma \models A$ holds if and only if $\Gamma \cup \{\neg A\}$ is unsatisfiable.*

*Proof.* Part (a) is the special case of (b) with $\Gamma = \varnothing$, so we need only to prove part (b). Note that $\Gamma \vDash A$ means that for any pair $(\mathfrak{A}, \sigma)$, if $\mathfrak{A} \vDash B[\sigma]$ for every $B \in \Gamma$ then $\mathfrak{A} \vDash A[\sigma]$. Also, $\{\Gamma, \neg A\}$ is unsatisfiable means that there is no pair $(\mathfrak{A}, \sigma)$ such that $\mathfrak{A} \vDash \neg A[\sigma]$ and such that $\mathfrak{A} \vDash B[\sigma]$ for all $B \in \Gamma$. By the definition of truth for $\neg A$, these two conditions are clearly equivalent. □

Theorem III.45 is the semantic version of the proof by contradiction principles that will be established later in Section IV.2.3 for first-order provability and inconsistency. The difference, however, is that the proof by contradiction principle of Section IV.2.3 will require that $A$ is a sentence.

**Tautologies in first-order logic.** We have just defined the notions of logical validity and logical implication. The adjective "logical" means that the validity of the implication holds because of the combined meanings of the propositional connectives $(\neg, \wedge, \vee, \rightarrow, \leftrightarrow)$ and the quantifiers $(\forall, \exists)$. We can also talk about "tautological" validity and "tautological" implication for first-order formulas; loosely speaking, the adjective "tautological" means that the validity or the implication holds just because of the meanings of the propositional connectives. For example, $\forall x P(x) \wedge \forall x Q(x)$ tautologically implies $\forall x Q(x) \wedge \forall x P(x)$, just because of the meaning of the connective $\wedge$.

For the next definition, recall the definition of substitution of formulas for propositional formulas in Section I.10. The notation $A(B_1, \ldots, B_k/p_1, \ldots, p_k)$ means the formula obtained by replacing each occurrence of $p_i$ in $A$ with $B_i$.

**Definition III.46.** Let $C$ be a propositional tautology, and suppose $C$ involves only the propositional variables $p_1, \ldots, p_k$. Let $B_1, \ldots, B_k$ be arbitrary first-order formulas. Then $C(B_1, \ldots, B_k/p_1, \ldots, p_k)$ is called a *tautology.*

**Definition III.47.** Let $\Gamma$ be a set $\{A_1, \ldots, A_k\}$ of first-order formulas and $B$ be a first-order formula. Then $\Gamma$ *tautologically implies* $B$ if

$$A_1 \rightarrow A_2 \rightarrow \cdots \rightarrow A_k \rightarrow B$$

is a tautology.

If $\Gamma$ is an infinite set, then $\Gamma$ tautologically implies $A$ if some finite subset of $\Gamma$ *tautologically implies* $A$.[9]

**Example III.48.** A few examples of tautological implication include

(a) $x = y \wedge y = z$ tautologically implies $y = z \wedge x = y$.

(b) $x = y \wedge y = z$ does not tautologically imply $y = x$.

(c) $P(x) \vee Q(x)$ tautologically implies $Q(x) \vee P(x)$.

(d) $\forall x\,(P(x) \vee Q(x))$ does not tautologically imply $\forall x\,(Q(x) \vee P(x))$.

[9]The Compactness Theorem for propositional logic allows us to define tautological implication for infinite sets $\Gamma$ in this way.

In (b) the order of the equality is reversed from $x = y$ to $y = x$; for this reason, it is not tautologically implied. Of course, $x = y$ *logically* implies $y = x$, but it does not *tautologically* imply it.

In (d), the subformula $P(x) \vee Q(x)$ is replaced by the tautologically equivalent $Q(x) \vee P(x)$. However, the whole formula $\forall x\,(P(x) \vee Q(x))$ is not tautologically equivalent to $\forall x\,(Q(x) \vee P(x))$. This is because the tautological equivalence is inside the scope of the quantifier $\forall x$.

Of course, $\forall x\,(P(x) \vee Q(x))$ is *logically* equivalent to $\forall x\,(Q(x) \vee P(x))$. This is a topic we will take up again in Example III.51.

**Theorem III.49.** *Let $\Gamma$ be a (finite) set of first-order formulas, and $A$ be a first-order formula.*

(a) *If $A$ is a tautology, then $\vDash A$.*
(b) *If $\Gamma$ tautologically implies $A$, then $\Gamma \vDash A$.*

*Proof.* To prove (a), suppose that $\mathfrak{A}$ is a structure and $\sigma$ is an object assignment. We must show that $\mathfrak{A} \vDash A[\sigma]$. Since $A$ is a tautology, $A$ is equal to $C(B_1, \ldots, B_k/p_1, \ldots, p_k)$ for some propositional tautology $C$ and first-order formulas $B_1, \ldots, B_k$. Let $\varphi$ be the (propositional) truth assignment defined by letting

$$\varphi(p_i) \;=\; \begin{cases} \mathrm{T} & \text{if } \mathfrak{A} \vDash B_i[\sigma] \\ \mathrm{F} & \text{if } \mathfrak{A} \nvDash B_i[\sigma] \end{cases}$$

We claim that for every subformula $D$ of $C$, we have

$$\varphi(D) = \mathrm{T} \qquad \text{if and only if} \qquad \mathfrak{A} \vDash D(B_1, \ldots, B_k/p_1, \ldots, p_k)[\sigma].$$

This claim is proved by induction on the complexity of subformulas $D$ of $C$. The base case is where $D$ is a single variable $p_i$: the claim holds for these $D$'s by the definition of $\varphi$. The argument for the induction step breaks into separate cases depending on whether the outermost connective of $D$ is a $\neg$, $\vee$, $\wedge$, $\rightarrow$ or $\leftrightarrow$. But in all five cases, the argument is trivial since the induction definition of truth for propositional formulas exactly parallels the recursive definition of truth for first-order formulas when the outermost connective is a propositional connective. We leave the details of the induction step to the reader.

Taking $D$ to be the formula $C$, the claim implies that $\varphi(C) = \mathrm{T}$ if and only if $\mathfrak{A} \vDash C[\sigma]$. Since $C$ is a tautology, we certainly have $\varphi(C) = \mathrm{T}$. Therefore, the claim gives $\mathfrak{A} \vDash C[\sigma]$. That completes the proof of part (a).

For part (b), suppose $\Gamma$ contains the formulas $B_1, B_2, \ldots, B_k$ and

$$B_1 \rightarrow \cdots \rightarrow B_k \rightarrow A$$

is a tautology. By part (a), $\vDash B_1 \rightarrow \cdots \rightarrow B_k \rightarrow A$. So by the Semantic Deduction Theorem III.44, $B_1 \ldots, B_k \vDash A$. Since $B_1, \ldots, B_k$ are in $\Gamma$, we thus have $\Gamma \vDash A$ as desired. $\square$

**Substitution of logically equivalent subformulas.** The next theorem states the meaning of a formula does not change if a subformula is replaced with a logically equivalent formula.

**Theorem III.50.** *Let $A$ be a formula and $B$ be a subformula of $A$. Let $C$ be a formula that is logically equivalent to $B$, i.e., $B \models\!\models C$. Let $A^*$ be the formula obtained by replacing its subformula $B$ with $C$. Then $A$ and $A^*$ are logically equivalent, i.e., $A \models\!\models A^*$.*

**Example III.51.** Example III.48 noted that $P(x) \vee Q(x)$ and $Q(x) \vee P(x)$ are tautologically equivalent; therefore they are also logically equivalent (by Theorem III.49). Thus, $\forall x(P(x) \vee Q(x))$ and $\forall x(Q(x) \vee P(x))$ are logically equivalent.

*Proof of Theorem III.50.* The proof will use induction on the complexity of $A$, ignoring the complexity of $B$. Suppose $D$ is a subformula of $A$ that has $B$ as a subformula; this includes the cases where $D$ is one of $A$ or $B$. Define $D^*$ to be the subformula that results after replacing the subformula $B$ in $D$ with $C$. On the other hand, suppose $D$ is a subformula of $A$ that neither is a subformula of $B$ nor contains $B$ as a subformula; then let $D^*$ be $D$.

We claim that $D \models\!\models D^*$ for any subformula $D$ of $A$ that is not a proper subformula of the subformula $B$ in $A$. This is proved by induction on the complexity of the formula $D$. There are two base cases to consider. The first base case is where $D$ is the same as $B$, so $D^*$ is $C$. Thus $D \models\!\models D^*$ since $B \models\!\models C$. The second base case is where $D^*$ is the same as $D$; then of course $D \models\!\models D^*$. In the induction steps, $D$ is a formula $\neg D_1$ or $D_1 \circ D_2$ or $\forall x_i\, D_1$ or $\exists x_i D_1$, and $D^*$ is the corresponding formula $\neg D_1^*$ or $D_1^* \circ D_2^*$ or $\forall x_i\, D_1^*$ or $\exists x_i D_1^*$. The induction hypotheses states that $D_1 \models\!\models D_1^*$ and (when appropriate) $D_2 \models\!\models D_2^*$. From the definition of truth, the truth value of $\mathfrak{A} \models D[\sigma]$ depends only on the truth values of $\mathfrak{A} \models D_i[\tau]$, where $\tau$ ranges over all $x_i$-variants of $\sigma$. By the induction hypothesis, the truth values of $\mathfrak{A} \models D_i[\tau]$ are the same as the truth values of $\mathfrak{A} \models D_i^*[\tau]$. Therefore, $\mathfrak{A} \models D[\sigma]$ if and only if $\mathfrak{A} \models D^*[\sigma]$. Since this holds for all structures $\mathfrak{A}$ and object assignments $\sigma$, we have proved $D \models\!\models D^*$. □

As a simple corollary, Theorem III.36 on quantifiers and negations can be restated as:

**Theorem III.52.** *Let $A$ be a formula.*
(a) $\neg\forall x\, A \models\!\models \exists x \neg A$.
(b) $\neg\exists x\, A \models\!\models \forall x \neg A$.

*Proof.* Since $A$ and $\neg\neg A$, are tautologically equivalent, and hence logically equivalent, Theorem III.50 tells us that $\neg\forall x\, A \models\!\models \neg\forall x\, \neg\neg A$. And by Theorem III.36, $\neg\forall x\, \neg\neg A$ is logically equivalent to $\exists x \neg A$. That proves (a); the proof of (b) is similar. □

## III.5 Counterexamples to Validity and Implication

This section gives examples of some very simple finite structures; we use them to show that formulas are not valid, or that logical implications do not hold. Another way to view these constructions is that they give structures that show the satisfiability of formulas.

**Example III.53.** We show that

$$\not\models \forall x_1(P(x_1) \to \forall x_2 P(x_2)), \tag{III.22}$$

or, equivalently, that $\neg\forall x_1(P(x_1) \to \forall x_2 P(x_2))$ is satisfiable. The first-order language is $L_1 = \{P\}$; it contains only the unary predicate symbol $P$.[10] Define the structure $\mathfrak{A}_1$ to have universe $|\mathfrak{A}_1| = \{0, 1\}$, and $P^{\mathfrak{A}_1} = \{0\}$.

We claim that $\mathfrak{A}_1 \not\models \forall x_1 (P(x_1) \to \forall x_2 P(x_2))$. To see this, suppose $\sigma$ is an object assignment with $\sigma(x) = 0$. Then $\mathfrak{A}_1 \models P(x)[\sigma]$ since $\sigma(x) = 0 \in P^{\mathfrak{A}_1}$. Suppose $\tau$ is an object assignment such that $\tau(x_2) = 1$. Then $\mathfrak{A}_1 \not\models P(x_2)[\tau]$ since $1 \notin P_1^{\mathfrak{A}}$. Therefore, $\mathfrak{A}_1 \not\models \forall x_2 P(x_2)[\pi]$ for an arbitrary object assignment $\pi$ (since $\tau$ can be taken as an $x_2$-variant of $\pi$). We thus have $\mathfrak{A}_1 \models P(x)[\sigma]$ and $\mathfrak{A}_1 \not\models \forall x_2 P(x_2)[\sigma]$. Applying the definition of truth, we get first that $\mathfrak{A}_1 \not\models (P(x_1) \to \forall x_2 P(x_2))[\sigma]$, and second that $\mathfrak{A}_1 \not\models \forall x_1 (P(x_1) \to \forall x_2 P(x_2))[\sigma]$.

Therefore $\forall x_1(P(x_1) \to \forall x_2 P(x_2))$ is not logically valid, and (III.22) holds.

**Example III.54.** We show that

$$\forall x\, \exists y\, P(x, y) \not\models \exists y\, \forall x\, P(x, y), \tag{III.23}$$

or, equivalently, that $\{\forall x\, \exists y\, P(x, y), \neg\exists y\, \forall x\, P(x, y)\}$ is satisfiable. Now the language is $L = \{P\}$ with $P$ a binary predicate symbol.

Consider the structure $\mathfrak{A}_2$ defined with $|\mathfrak{A}_2| = \{0, 1\}$ and $P^{\mathfrak{A}_2} = \{\langle 0, 0\rangle, \langle 1, 1\rangle\}$. Then (III.23) follows from the facts that $\mathfrak{A}_2 \models \forall x\, \exists y\, P(x, y)$, but $\mathfrak{A}_2 \not\models \exists y\, \forall x\, P(x, y)$. These two facts can be easily checked using the definition of truth.

**Example III.55.** We show that

$$\forall x\, (P(x) \vee Q(x)) \not\models \forall x\, P(x) \vee \forall x\, Q(x), \tag{III.24}$$

or, equivalently, that $\{\forall x\, (P(x) \vee Q(x)), \neg(\forall x\, P(x) \vee \forall x\, Q(x))\}$ is satisfiable. The language contains two unary predicate symbols $P$ and $Q$.

Let $\mathfrak{A}_3$ be the structure with $|\mathfrak{A}_3| = \{0, 1\}$, with $P^{\mathfrak{A}_3} = \{0\}$ and $Q^{\mathfrak{A}_3} = \{1\}$. It is easy to check that $\mathfrak{A}_3 \models \forall x\, (P(x) \vee Q(x))$ and $\mathfrak{A}_3 \not\models \forall x\, P(x) \vee \forall x\, Q(x)$. Thus, (III.24) holds.

In the above example, we wrote $P^{\mathfrak{A}_3} = \{0\}$ and $Q^{\mathfrak{A}_3} = \{1\}$ instead of $P^{\mathfrak{A}_3} = \{\langle 0\rangle\}$ and $Q^{\mathfrak{A}_3} = \{\langle 1\rangle\}$. This is because we identify a 1-tuple with the object in the 1-tuple. For instance, "$\langle 0\rangle$" is the same as "0".

[10] Unless otherwise specified, all languages are presumed to include the equality sign $=$. However, this makes no difference in the present case since the formula (III.22) does not use the equality sign.

**Example III.56.** Working with the same language as the previous example,

$$\exists x\, P(x) \wedge \exists x\, Q(x) \nvDash \exists x\, (P(x) \wedge Q(x)). \tag{III.25}$$

or, equivalently, that $\{\exists x\, P(x) \wedge \exists x\, Q(x), \neg\exists x\, (P(x) \wedge Q(x))\}$ is satisfiable. Let $\mathfrak{A}_3$ be the same structure as in the previous example. It is easy to check that $\mathfrak{A}_3 \vDash \exists x\, P(x) \wedge \exists x\, Q(x)$ and $\mathfrak{A}_3 \nvDash \exists x\, (P(x) \wedge Q(x))$. Thus (III.25) holds.

**Example III.57.** This is a redo of III.41(b). Working with the same language again,

$$P(x) \nvDash P(y),$$

or, equivalently, $\{P(x), \neg P(y)\}$ is satisfiable. Since $P(x)$ and $P(y)$ have free occurrences of variables, a counterexample must involve an object assignment. Let $\mathfrak{A}_3$ be the same structure as in the previous examples. Let $\sigma$ be an object assignment such that $\sigma(x) = 0$ and $\sigma(y) = 1$. By the definition of truth for atomic formulas, $\mathfrak{A}_3 \vDash P(x)[\sigma]$ and $\mathfrak{A}_3 \nvDash P(y)[\sigma]$. Therefore $P(x) \nvDash P(y)$.

# III.6 Substitution of Terms for Variables

## III.6.1 Substitution and substitutability

Substitution is used to replace free variables in formulas with terms; it is one of the important ways to modify and reuse formulas. For example, let $A$ be the formula

$$\exists x_2\, (x_2 + x_2 = x_1)$$

that can be used to express that "$x_1$ is even". To express that $x_1 \cdot x_1 + x_3 + 1$ is even, the free occurrence of $x_1$ in $A$ is replaced with $x_1 \cdot x_1 + x_3 + 1$. This gives the formula

$$\exists x_2\, (x_2 + x_2 = x_1 \cdot x_1 + x_3 + 1) \tag{III.26}$$

which indeed does state that $x_1 \cdot x_1 + x_3 + 1$ is even.

On the other hand, if we wish to express $x_1 + x_2$ is even and try substituting $x_1 + x_2$ for $x_1$ in $A$, the resulting formula is

$$\exists x_2\, (x_2 + x_2 = x_1 + x_2). \tag{III.27}$$

This does *not* express that $x_1 + x_2$ is even; instead, it is a logically valid formula. The problem is that the "$x_2$" in the term "$x_1 + x_2$" was "captured" or "bound" by the quantifier $\exists x_2$. As will be defined momentarily, this problem arises because $x_1 + x_2$ is not "substitutable" for $x_1$ in $A$. The fix for this problem is to form an "alphabetic variant" of $A$ by renaming the bound variable $x_2$ to, say, $x_3$ before substituting $x_1 + x_2$ for $x_1$.

We write $A(t/x)$ to denote the formula obtained by substituting the term $t$ for each free occurrence of $x$ in $A$. In other words, each free occurrence of $x$ is replaced with the expression $t$. For example, with $A$ as above, the formula (III.26) will be denoted $A(x_1 \cdot x_1 + x_3 + 1/x_1)$ and (III.27) will be denoted $A(x_1 + x_2/x_1)$.

More generally, for $t_1, \ldots, t_\ell$ terms and $x_{i_1}, \ldots, x_{i_\ell}$ distinct variables, the formula $A(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is obtained by replacing, for each $i$, every free occurrence of $x_i$ in $A$ with $t_i$. The notation $A(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ indicates that the substitutions are done "in parallel", namely replacing all free occurrences of the $x_{i_j}$'s at once. The formal definition by recursion is as follows.

**Definition III.58** (Substitution into terms)**.** Let $s$ be a term, $x_{i_1} \ldots, x_{i_\ell}$ be distinct variables and $t_1, \ldots, t_\ell$ be terms. Then the *substitution* $s(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is the term recursively defined by:

**Base case #1:** If $s$ is $x_{i_j}$, $1 \le j \le \ell$, then $s(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is $t_j$.

**Base case #2:** If $s$ is either a constant symbol or a variable $x_i$ with $i \notin \{i_1, \ldots, i_\ell\}$, then $s(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is equal to $s$.

**Recursive step:** If $s$ is $f(r_1, \ldots, r_k)$ where $f$ is a $k$-ary function symbol then $s(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is equal to $s(r'_1, \ldots, r'_k)$, where each $r'_i$ is $r_i(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$.

**Definition III.59** (Substitution into formulas)**.** Let $A$ be a formula, $x_{i_1} \ldots, x_{i_\ell}$ be distinct variables and $t_1, \ldots, t_\ell$ be terms. Then the *substitution* $A(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is recursively defined by:

**Base case:** Suppose $A$ is an atomic formula $P(r_1, \ldots, r_k)$ or $r_1 = r_2$, respectively. Let each $r'_i$ be $r_i(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$. Then $A(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is equal to $P(r'_1, \ldots, r'_k)$ or $r'_1 = r'_2$, respectively.

**Recursive step, case #1:** Suppose $A$ is $\neg B$. Then $A(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is equal to $\neg B'$ where $B'$ is $B(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$.

**Recursive step, case #2:** Suppose $A$ is $B \circ C$, where $\circ$ is one of $\wedge, \vee, \to, \leftrightarrow$. Then $A(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is equal to $B' \circ C'$ where $B'$ is $B(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ and $C'$ is $C(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$

**Recursive step, case #3:** Suppose $A$ is $Q x_i \, B$ where $Q x_i$ denotes either $\forall x_i$ or $\exists x_i$. Also suppose $i \notin \{i_1, \ldots, i_\ell\}$. Then $A(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is equal to $Q x_i \, B(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$.

**Recursive step, case #4:** Suppose $A$ is $Q x_{i_j} \, B$. Let $B'$ equal

$$B(t_1, \ldots, t_{j-1}, t_{j+1}, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_{j-1}}, x_{i_{j+1}}, \ldots, x_{i_\ell}). \tag{III.28}$$

(Note the entries with subscript $j$ are skipped.) Then $A(t_1, \ldots, t_\ell / x_{i_1}, \ldots, x_{i_\ell})$ is equal to $Q x_{i_j} \, B'$.

The just-given definition of substitution did not pay attention to whether any free variables were "captured" by the substitution. To rule this out, we require that each $t_j$ is "substitutable" for $x_{i_j}$. Informally, this means that there is no variable $x_k$ appearing in $t_j$ such that there is a free occurrence of $x_{i_j}$ in $A$ which is in the scope of a quantifier $Q x_k$.

**Definition III.60** (Substitutable)**.** Let $A$ be a formula, $x_i$ be a variable, and $t$ be a term. Then $t$ *is substitutable for* $x_i$ *in* $A$ is defined recursively as follows:

- If $A$ is atomic, then $t$ is substitutable for $x_i$ in $A$.
- If $A$ is $\neg B$, then $t$ is substitutable for $x_i$ in $A$ if and only if $t$ is substitutable for $x_i$ in $B$.
- If $A$ is $B \circ C$ where $\circ$ is a binary propositional connective, then $t$ is substitutable for $x_i$ in $A$ if and only if $t$ is substitutable for $x_i$ in both $B$ and $C$.
- If $A$ is $Qx_i\, B$, then $t$ is substitutable for $x_i$ in $A$.
- If $A$ is $Qx_j\, B$ with $j \neq i$, then $t$ is substitutable for $x_i$ in $A$ if and only if either (i) $x_i$ is not free in $B$ or (ii) the variable $x_j$ does not appear in $t$.

The next theorem states the main property needed for substitution.

**Theorem III.61.** *Let $t_1, \ldots, t_\ell$ be terms and $x_{i_1}, \ldots, x_{i_\ell}$ be distinct variables. We write $\vec{t}$ and $\vec{x}$ as shorthand notations for $t_1, \ldots, t_\ell$ and $x_{i_1}, \ldots, x_{i_\ell}$.*

(a) *Let $s$ be a term. Then*

$$\vDash \bigwedge_{j=1}^{\ell} x_{i_j} = t_j \;\rightarrow\; s = s(\vec{t}/\vec{x}).$$

(b) *Let $A$ be a formula and suppose that each $t_j$ is substitutable for $x_{i_j}$ in $A$. Then*

$$\vDash \bigwedge_{j=1}^{\ell} x_{i_j} = t_j \;\rightarrow\; [A \leftrightarrow A(\vec{t}/\vec{x})]. \tag{III.29}$$

Theorem III.61 will be proved by induction on the complexity of terms and formulas. As preparation for this proof, the next lemma gives some special cases of the theorem.

**Lemma III.62.**

(a) *Let $P$ be a $k$-ary predicate symbol, and $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$ be terms. Then*

$$\vDash \bigwedge_{i=1}^{k} s_i = t_i \;\rightarrow\; [P(s_1, \ldots, s_k) \leftrightarrow P(t_1, \ldots t_k)]. \tag{III.30}$$

(b) *Let $f$ be a $k$-ary function symbol, and $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$ be terms. Then*

$$\vDash \bigwedge_{i=1}^{k} s_i = t_i \;\rightarrow\; f(s_1, \ldots, s_k) = f(t_1, \ldots t_k). \tag{III.31}$$

*Proof of Lemma III.62.* Let $\mathfrak{A}$ be a structure and $\sigma$ an object assignment. The only way that the formula (III.30) can fail to be satisfied by the pair $(\mathfrak{A}, \sigma)$ is if $\sigma(s_i) = \sigma(t_i)$ for all $i$. But in that case, clearly $\langle \sigma(s_1), \ldots, \sigma(s_k) \rangle \in P^{\mathfrak{A}}$ if and only if $\langle \sigma(t_1), \ldots, \sigma(t_k) \rangle \in P^{\mathfrak{A}}$, and hence $\mathfrak{A} \vDash P(\vec{s}) \leftrightarrow P(\vec{t})[\sigma]$. That proves part (a). Part (b) is proved similarly. □

Another simple but useful lemma states that if $x_i$ is not free in $A$ then a substitution for $x_i$ has no effect.

**Lemma III.63.**
(a) *If $x_i$ does not appear free in $A$, Then $A(t/x_i)$ is equal to $A$.*
(b) *If $x_{i_j}$ does not appear free in $A$, then $A(t_1,\ldots,t_k/x_{i_1},\ldots,x_{i_k})$ is equal to*

$$A(t_1,\ldots,t_{j-1},t_{j+1},\ldots,t_\ell/x_{i_1},\ldots,x_{i_{j-1}},x_{i_{j+1}},\ldots,x_{i_\ell}).$$

*Proof of Lemma III.63.* It is enough to prove (b), since (a) is a special case. First, it necessary to prove that if $x_{i_j}$ does not appear in a term $s$, then $s(t_1,\ldots,t_k/x_{i_1},\ldots,x_{i_k})$ is equal to

$$s(t_1,\ldots,t_{j-1},t_{j+1},\ldots,t_\ell/x_{i_1},\ldots,x_{i_{j-1}},x_{i_{j+1}},\ldots,x_{i_\ell}).$$

This is readily proved by induction on the complexity of $s$. From this, the lemma holds for atomic formulas $A$. For general formulas $A$ the proof proceeds by induction on the complexity of $A$. The induction steps corresponding to the first three recursion cases of the definition are trivial, since in the recursive steps #1-#3, $x_i$ has a free occurrence in $A$ if and only if it has a free occurrence in $B$ (or $C$). For recursion step #4, the lemma clearly holds since, in light of (III.28), $A(t_1,\ldots,t_k/x_{i_1},\ldots,x_{i_k})$ is equal to

$$A(t_1,\ldots,t_{j-1},t_{j+1},\ldots,t_\ell/x_{i_1},\ldots,x_{i_{j-1}},x_{i_{j+1}},\ldots,x_{i_\ell}). \qquad \square$$

*Proof of Theorem III.61.* Part (a) is proved readily by induction on the complexity of the term $s$. The base cases where $s$ is a variable $x_j$ and $s$ is a constant symbol are trivial. The induction step is immediate with the aid of Lemma III.62(b). The details are left to the reader.

Part (b) is proved induction on the complexity of the formula $A$. The base case where $A$ is an atomic formula $P(\cdots)$, is the same as part (a) of Lemma III.62 and has already been proved. The base case where $A$ has the form $x_\ell = x_k$ is trivial, and left to the reader.

The induction step splits into cases depending on how the formula $A$ is formed. By Lemma III.63, we can assume without loss of generality that each $x_{i_j}$ occurs free in $A$.

If $A$ is $\neg B$ or $B \circ C$, the induction hypotheses state that

$$\bigwedge_{i=1}^{\ell} x_{i_j} = t_j \to [B \leftrightarrow B(\vec{t}/\vec{x})] \qquad \text{and} \qquad \bigwedge_{i=1}^{\ell} x_{i_j} = t_j \to [C \leftrightarrow C(\vec{t}/\vec{x})]$$

are logically valid. (Only the first one is needed for the case where $A$ is $\neg B$). These formulas tautologically imply the desired formula (III.29).

The second case of the induction step is when $A$ has the form $\forall x_k\, B$. The induction hypothesis is that

$$\vDash \bigwedge_{i=1}^{\ell} x_{i_j} = t_j \;\to\; [B \leftrightarrow B(\vec{t}/\vec{x})]. \tag{III.32}$$

Theorem III.31 states that the following generalization is valid:

$$\models\ \forall x_k \Big( \bigwedge_{i=1}^{\ell} x_{i_j} = t_j\ \rightarrow\ [B \leftrightarrow B(\vec{t}/\vec{x})] \Big).$$

By the assumption that each $x_{i_j}$ appears free in $A$, the variable $x_k$ is distinct from all the $x_{i_j}$'s. Since each $x_{i_j}$ appears free in $A$ and since $t_j$ is substitutable for $x_{i_j}$ in $A$, the variable $x_k$ does not appear in $t_j$. Therefore the variable $x_k$ does not appear in the conjunction on the lefthand side of (III.32). Thus the equivalence established in Example III.42 on page 89 gives

$$\models\ \bigwedge_{i=1}^{\ell} x_{i_j} = t_j\ \rightarrow\ \forall x_k\, [B \leftrightarrow B(\vec{t}/\vec{x})].$$

It is a general principle that $\forall x\,(C \leftrightarrow D) \models (\forall x\, C \leftrightarrow \forall x\, D)$ for arbitrary formulas $C$ and $D$. Exercise III.18 asks for a proof of this. Thus we get

$$\models\ \bigwedge_{i=1}^{\ell} x_{i_j} = t_j\ \rightarrow\ [\forall x_k\, B \leftrightarrow \forall x_k\, B(\vec{t}/\vec{x})].$$

That finishes the inductive step for $A$ of the form $\forall x_k\, B$.

The final induction step is for $A$ of the form $\exists x_k\, B$. This case can be reduced to the earlier cases since $A \models\models \neg\forall x_k\, \neg B$. The arguments for the above induction cases (used twice for the $\neg$'s and once for the $\forall x_k$) give

$$\models\ \bigwedge_{i=1}^{\ell} x_{i_j} = t_j\ \rightarrow\ \neg\forall x_k\, \neg B \leftrightarrow \neg\forall x_k\, \neg B(\vec{t}/\vec{x})].$$

This is logically equivalent to (III.29), and finishes the proof of the induction case for $A$ of the form $\exists x_k\, B$. □

## III.6.2 Alphabetic variants (renaming of bound variables).

A alphabetic variant of a formula $A$ means any formula obtained by renaming the bound variables of $A$ in a consistent way. In general, any alphabetic variants of $A$ are logically equivalent to $A$. For example, if $A$ is the formula $\exists x_2\,(x_2 + x_2 = x_1)$, then $\exists x_3\,(x_3 + x_3 = x_1)$ is an alphabetic variant $B$. Both formulas $A$ and $B$ assert that $x_1$ is even when they are interpreted over the integers. The advantage of the alphabetic variant arises when a term $t$ is not substitutable for $x_1$ in $A$, but we nonetheless want to use $A$ to assert that $t$ is even. Renaming the bound variables can allow $t$ to be substituted into an alphabetic variant. For instance, we can use the formula $B(x_2/x_1)$, namely $\exists x_3\,(x_3 + x_3 = x_2)$ to assert that $x_2$ is even — this works even though $x_2$ is not substitutable in $A$.

The next theorem gives the technical conditions that allow the use of alphabetic variants.

**Theorem III.64.** *Let $x_i$ and $x_j$ be distinct variables and let $A$ be a formula of the form $\exists x_i\, B$ or $\forall x_i\, B$. Suppose that there is no free occurrence of $x_j$ in $B$*

*and that $x_j$ is substitutable for $x_i$ in $B$. Let $C$ be $B(x_j/x_i)$, and let $D$ be $\exists x_j\, C$ or $\forall x_j\, C$, respectively. Then $A$ and $D$ are logically equivalent, $A \models\!\dashv D$. That is,*

$$\exists x_i\, B \models\!\dashv \exists x_j\, B(x_j/x_i) \qquad \textit{and} \qquad \forall x_i\, B \models\!\dashv \forall x_j\, B(x_j/x_i).$$

*Proof.* We first prove the theorem for the case where $A$ is $\exists x_i\, B$ and $D$ is $\exists x_j\, C$. From Theorem III.61, we have

$$\models x_i = x_j \to [B \leftrightarrow B(x_j/x_i)]. \tag{III.33}$$

Suppose $\mathfrak{A}$ is a structure and $\sigma$ an object assignment. We need to show that $\mathfrak{A} \models \exists x_i\, B[\sigma]$ holds if and only if $\mathfrak{A} \models \exists x_j\, B(x_j/x_i)$ holds. By the definition of truth,

$$\mathfrak{A} \models \exists x_i\, B[\sigma] \quad \Leftrightarrow \quad \mathfrak{A} \models B[\tau] \text{ for some } x_i\text{-variant } \tau \text{ of } \sigma. \tag{III.34}$$

Let's define an $\{x, y\}$-variant of $\sigma$ to be a $y$-variant of an $x$-variant of $\sigma$. In other words, a $\{x, y\}$-variant of $\sigma$ is an object assignment $\pi$ which differs from $\sigma$ on at most the values it assigns to $x$ and $y$. Since $x_j$ does not have a free occurrence in $B$ (or in $A$), the truth of $\mathfrak{A} \models B[\tau]$ does not depend on the value of $\tau(x_j)$. Therefore, from (III.34),

$$\mathfrak{A} \models \exists x_i\, B[\sigma] \quad \Leftrightarrow \quad \mathfrak{A} \models B[\pi], \text{ for some } \{x_i, x_j\}\text{-variant } \pi \text{ of } \sigma \text{ such that } \pi(x_i) = \pi(x_j) \quad .$$

Similar reasoning, using the fact that $x_i$ does not appear free in $B(x_j/x_i)$, shows that

$$\mathfrak{A} \models \exists x_j\, B(x_j/x_i)[\sigma] \quad \Leftrightarrow \quad \mathfrak{A} \models B(x_j/x_i)[\pi,] \text{ for some } \{x_i, x_j\}\text{-variant } \pi \text{ of } \sigma \text{ such that } \pi(x_i) = \pi(x_j) \quad .$$

Finally, the validity of (III.33) implies that if $\pi(x_i) = \pi(x_j)$, then $\mathfrak{A} \models B[\pi]$ if and only $\mathfrak{A} \models B(x_j/x_i)[\pi]$. It follows that $\mathfrak{A} \models \exists x_i\, B[\sigma]$ is true if and only if $\mathfrak{A} \models \exists x_j\, B(x_j/x_i)[\sigma]$ is true.

The proof for the case of a universal quantifier can be reduced to the case of an existential quantifier. By Theorem III.36 or III.52, and the definition of truth for $\neg$, we have that $\forall x_i\, B \models\!\dashv \forall x_j\, B(x_j/x_i)$ holds if and only if $\exists x_i\, \neg B \models\!\dashv \exists x_j\, \neg B(x_j/x_i)$. The latter holds by the already-proved first part of the theorem. □

**Example III.65.** We earlier used the following formula (III.8), called *Prime*,

$$x \neq S(0) \wedge \forall y\, (\exists z\, (y \cdot z = x) \to y = S(0) \vee y = x).$$

to express the condition that $x$ is prime. Let $A$ denote its alphabetic variant

$$x \neq S(0) \wedge \forall u\, (\exists v\, (y \cdot v = x) \to u = S(0) \vee u = x).$$

For instance, we can use $A(y + z/x)$ to express that $y + z$ is prime even though $y + z$ is not substitutable for $x$ in *Prime*.

We can show that *Prime* $\models\dashv A$ by using Theorem III.64 twice. First, the theorem shows that $\exists z\,(y \cdot z = x)$ and $\exists v\,(y \cdot v = x)$ are logically equivalent. Then by Theorem III.50 on replacing a subformula with a logically equivalent subformula, $\forall y\,(\exists v\,(y \cdot v = x) \to y = S(0) \vee y = x)$ is logically equivalent to $\forall y\,(\exists v\,(y \cdot v = x) \to y = S(0) \vee y = x)$. Applying Theorem III.64 again to rename $y$ to $u$ in the latter formula, and using Theorem III.50 again, gives that *Prime* and $A$ are logically equivalent.

### III.6.3 Universal instantiation and existential introduction.

The principle of universal instantiation states that if $\forall x\,A$ is true, then $A(t/x)$ is true for any $t$ which is substitutable for $x$. Contrapositively, the principle of existential introduction states that if $A(t/x)$ is true, then $\exists x\,A$ is true, again assuming $t$ is substitutable for $x$ in $A$.

The universal instantiation formulas will be used as axioms for the Hilbert-style proof system FO for first-order logic in the next chapter.

**Theorem III.66** (Universal Instantiation)**.** *Suppose $t$ is substitutable for $x$ in $A$. Then*

$$\models \forall x\,A \to A(t/x).$$

*Proof.* Without loss of generality, the variable $x$ does not appear in $t$ (since otherwise, by Theorem III.64, the bound variable $x$ could be replaced by a different variable). Suppose $\mathfrak{A} \models \forall x\,A[\sigma]$; we must show that $\mathfrak{A} \models A(t/x)[\sigma]$. Let $\tau$ be the $x$-variant of $\sigma$ such that $\tau(x) = \sigma(t)$. By the definition of truth, $\mathfrak{A} \models A[\tau]$.

By Theorem III.61(b), $x = t \wedge A \to A(t/x)$ is valid. Therefore, $\mathfrak{A} \models A(t/x)[\tau]$. Since $x$ does not appear in $A(t)$, and since $\tau$ is an $x$-variant of $\sigma$. Theorem III.23 implies that $\mathfrak{A} \vdash A(t/x)[\sigma]$ holds, as desired. □

**Corollary III.67** (Existential Introduction)**.** *Suppose $t$ is substitutable for $x$ in $A$. Then*

$$\models A(t/x) \to \exists x\,A.$$

The corollary follows immediately from Theorem III.66 using the logical equivalence between $\exists x\,A$ and $\neg\forall x\,\neg A$. □

**Example III.68.** The formula $\forall x\,\exists y\,(x + y = 0) \to \exists y\,(S(S(0)) + y = 0)$ and the formula $\exists y\,(S(S(0)) + y = 0) \to \exists x\,\exists y\,(x + y = 0)$ are both logically valid.

**Corollary III.69.** *For any formula $A$, $\forall x\,A \to \exists x\,A$ is logically valid.*

This corollary follows by tautological implication from the previous corollary and theorem, taking $t$ to be a variable that is not quantified in $A$. □.

**Corollary III.70.** *If $\models A$ and $t$ is substitutable for $x$ in $A$, then $\models A(t/x)$. More generally, if $\models A$ and each term $t_i$ is substitutable for $x_{i,j}$ in $A$, then $\models A(\vec{t}/\vec{x})$.*

This corollary is immediate frem Theorems III.31 and III.66.

### III.6.4 Relaxed notations for substitution.

The notation "$A(t/x)$" for substitution is somewhat cumbersome to work with, and it is common to use the more relaxed, user-friendly notations "$A(x)$" and "$A(t)$" instead. In addition, when working with a structure $\mathfrak{A}$, if $b$ is an object in $|\mathfrak{A}|$, it is convenient to write $\mathfrak{A} \vDash A(b)$ to mean that $b$ satisfies the property of $A(x)$ in $\mathfrak{A}$. This latter notation avoids needing to mention an object assignment $\sigma$ that maps $x$ to $b$.

Let's make these more relaxed notations precise. When using relaxed notation for substitution, a formula $A$ will be introduced with the name "$A(x)$" where $x$ is a variable. Common terminology would be "Let $A(x)$ be a formula", or "Let $A = A(x)$ be a formula". This means that $A(x)$ is a synonym for $A$ and that terms will be substituted for the variable $x$. Thereafter, if $t$ is a term, the notation "$A(t)$" is used to denote the formula $A(t/x)$ provided that $t$ is substitutable for $x$ in $A$. If $t$ is not substitutable for $x$ in $A(x)$, then there are two possible conventions. The first possible convention is that $A(t)$ is undefined when $t$ is not substitutable for $x$ in $A$. This is the convention that is used in this textbook. The second possible convention is that a suitable alphabetic variant $A'$ of $A$ is picked so that $t$ is substitutable for $x$ in $A$. By Theorem III.64 on alphabetic variants, $A(x) \vDash\dashv A'(x)$. Then $A(t)$ denotes $A'(t/x)$. The latter convention is somewhat ambiguous since there are many choices for the alphabetic variant $A'$, but in practice, the ambiguity causes no problems.

**Example III.71.** Let $A(x)$ be the formula $\forall y\, Q(x, y)$. Then $A(f(0) + z)$ is the formula $\forall y\, Q(f(0) + z, q)$. The formula $A(y)$ is either undefined (in the first convention), or is a formula such as $\forall w\, Q(y, w)$ (under the second convention) where the new variable $w$ is picked so as to not appear in $Q$.

The next example is a little more subtle. It shows the importance of how a formula is first introduced.

**Example III.72.** Let $A(y)$ be the formula $\exists z\,(x + z = y)$. Then $A(x)$ is $\exists z\,(x + z = x)$ and $A(0)$ is $\exists z\,(x + z = 0)$.

Now let $B(x)$ be the formula $\exists z\,(x + z = x)$. Then $B(0)$ is $\exists z\,(0 + z = 0)$.

In the second example, $A(x)$ and $B(x)$ are the same formula, but $A(0)$ and $B(0)$ are different formulas. The point is that $A$ was first introduced in the form $A(y)$, and therefore the substitutions into $A$ are based on replacing the variable $y$. This illustrates the importance of being careful with relaxed notations for substitution.

Relaxed notation for substitution can also be used for substituting multiple terms in parallel. In this case, we introduce a formula as $A = A(x_{i_1}, \ldots, x_{i_k})$ then write $A(t_1, \ldots, t_k)$ to denote $A(\vec{t}/\vec{x})$. We will use this notation only if each $t_j$ is substitutable for $x_{i_j}$ in $A$.

It is important to note that the relaxed notation introducing a formula as $A = A(x)$ or $A = A(x_{i_1}, \ldots, x_{i_k})$ allows there to be other free variables in $A$ beyond the variable $x$ or the variables, $x_{i_1}, \ldots, x_{i_k}$. Example III.72 already showed an example of this, with $x$ free in $A(y)$.

Some of the above theorems can be restated with the relaxed substitution notation as follows. Notice how the relaxed notation makes the results substantially easier to understand.

**Theorem III.73.**

(a) (Restatement of Theorem III.64 on alphabetic variants.) *Suppose $y$ does not appear free in $B(x)$ and that $y$ is substitutable for $x$ in $B(x)$. Then*

$$\exists x\, B(x) \models\dashv \exists y\, B(y) \qquad \textit{and} \qquad \forall x\, B(x) \models\dashv \forall y\, B(y).$$

(b) (Restatements of Theorem III.66 and Corollary III.67 on universal instantiation and existential introduction.) *Suppose that $t$ is substitutable for $x$ in $A(x)$.*

$$\models \forall x\, A(x) \to A(t) \qquad \textit{and} \qquad \models A(t) \to \exists x\, A(x).$$

**Relaxed substitution notation for objects.** A related relaxed notation allows us to avoid using object assignments. Suppose that $\mathfrak{A}$ is a structure, $A(x)$ is a formula with no free variables other than $x$, and $b \in |\mathfrak{A}|$. Then, the notation $\mathfrak{A} \models A(b)$ is shorthand notation indicating that $\mathfrak{A} \models A[\sigma]$ holds for any $\sigma$ such that $\sigma(x) = b$.

More generally, suppose $A = A(x_{i_1}, \ldots, x_{i_k})$ is a formula with no variables appearing free in $A$ other than the $x_{i_j}$'s. Let $b_1, \ldots, b_k \in |\mathfrak{A}|$. Then $\mathfrak{A} \models A(b_1, \ldots, b_k)$ means that $\mathfrak{A} \models A[\sigma]$ for any $\sigma$ such that $\sigma(x_{i_j}) = b_j$ for all $j$.

The next simple theorem illustrates how the relaxed notation is used:

**Theorem III.74.** *Suppose $A(x_1, \ldots, x_1)$ contains only $x_1, \ldots x_k$ as free variables. Let $\mathfrak{A}$ be a structure and $\sigma$ be an object assignment. Set $b_i = \sigma(t_i)$ so each $b_i \in |\mathfrak{A}|$. Then*

$$\mathfrak{A} \models A(b_1, \ldots, b_k) \quad \Leftrightarrow \quad \mathfrak{A}(t_1, \ldots, t_k)[\sigma].$$

*Proof.* (Sketch) This follows from the logical validity of

$$\bigwedge_{i=1}^{k} b_i = t_i \to (A(b_1, \ldots, b_k) \leftrightarrow A(t_1, \ldots t_k)),$$

which follows from Theorem III.61(b). □

## III.7 Principles of Equality

We now discuss several principles of equality. The basic underlying philosophy is that if objects are equal, then they should satisfy the same properties. The first principles are that equality (=) defines an equivalence relation.

**Theorem III.75.** *Let $r$, $s$ and $t$ be terms.*
(a) $\vDash t = t$. (Reflexivity)
(b) $\vDash s = t \to t = s$. (Symmetry)
(c) $\vDash r = s \to s = t \to r = t$. (Transitivity)

*Proof.* These logical validities follow immediately from the definition of truth. For instance, transitivity follows from the fact that if an object assignment $\sigma$ satisfies $\sigma(r) = \sigma(s)$ and $\sigma(s) = \sigma(t)$, then $\sigma(r) = \sigma(t)$. □

The second principles are that predicate and function symbols respect equality. These were already stated and proved in Lemma III.62, and are restated here for convenience:

$$\vDash \bigwedge_{i=1}^{k} s_i = t_i \;\to\; [P(s_1, \ldots, s_k) \leftrightarrow P(t_1, \ldots t_k)].$$

and

$$\vDash \bigwedge_{i=1}^{k} s_i = t_i \;\to\; f(s_1, \ldots, s_k) = f(t_1, \ldots t_k).$$

A more general version of the equality principle holds for arbitrary formulas $A$ in place of a predicate $P$:

**Theorem III.76.** *Let $A(x_{i_1}, \ldots, x_{i_k})$ be a formula and let $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$ be terms. Suppose that each $s_j$ and $t_j$ is substitutable for $x_{i_j}$ in $A$. Then*

$$\vDash \bigwedge_{i=1}^{k} s_i = t_i \;\to\; [A(s_1, \ldots, s_k) \leftrightarrow A(t_1, \ldots t_k)]. \tag{III.35}$$

*Proof.* Without loss of generality, the variables $x_{i_1} \ldots, x_{i_k}$ do not appear in the terms $s_1, \ldots, s_k$ and $t_1, \ldots, t_k$, since otherwise the variables $x_{i_j}$ can be renamed in $A$. Equation (III.29) of Theorem III.61 gives

$$\vDash \bigwedge_{i=1}^{\ell} x_{i_j} = t_j \;\to\; [A(\vec{x}) \leftrightarrow A(\vec{t})].$$

From this, Corollary III.70 gives (III.35) as desired. □

## III.8 Prenex Formulas

A "prenex formula" is a formula in which all quantifiers are at the front of the formula. For example, $\forall x\, \exists y\, \forall z\, (A \wedge B)$ is a prenex formula, but $\forall x\, \exists y\, (A \wedge \forall z\, B)$ is not. It turns out that every first-order formula can be reexpressed in prenex form.

**Definition III.77.** The *prenex formulas* are inductively defined by:
(a) If $A$ is quantifier-free, i.e., if $A$ contains no quantifiers, then $A$ is a prenex formula.
(b) If $A$ is a prenex formula and $x_i$ is a variable, then $\forall x_i\, A$ and $\exists x_i\, A$ are prenex formulas.

An alternative definition of prenex formulas can be given in terms of not having a quantifier in the scope of a propositional connective. Definition III.8 already defined the scope of a quantifier. The scope of a propositional connective is defined similarly:

**Definition III.78.** Let $A$ be a formula. Let $B$ be a subformula of the form $\neg C$. The *scope* of this negation sign is the subformula $C$. Now let $B$ be a subformula of the form $C \circ D$ with $\circ$ a binary propositional connective ($\wedge$, $\vee$, $\rightarrow$, or $\leftrightarrow$). The scope of this connective $\circ$ is the two subformulas $C$ and $D$.

Clearly, a formula $A$ is prenex if and only if no quantifier occurs in $A$ in the scope of a propositional connective.

**Theorem III.79.** *Every formula $A$ is logically equivalent to a prenex formula.*

The next lemma is a crucial ingredient in the proof Theorem III.79.

**Lemma III.80.** *Let $A$ and $C$ be formulas, and assume $x$ is not free in $C$. Then*

(a) $C \wedge \exists x\, A \models\dashv \exists x\, (C \wedge A)$
(b) $C \wedge \forall x\, A \models\dashv \forall x\, (C \wedge A)$
(c) $C \vee \forall x\, A \models\dashv \forall x\, (C \vee A)$
(d) $C \vee \exists x\, A \models\dashv \exists x\, (C \vee A)$
(e) $C \rightarrow \forall x\, A \models\dashv \forall x\, (C \rightarrow A)$
(f) $C \rightarrow \exists x\, A \models\dashv \exists x\, (C \rightarrow A)$
(g) $(\forall x\, A) \rightarrow C \models\dashv \exists x\, (A \rightarrow C)$
(h) $(\exists x\, A) \rightarrow C \models\dashv \forall x\, (A \rightarrow C)$

*Proof of Lemma III.80.* We will sketch the proof, and leave some of the details to the exercises. First, part (e) was already proved in Example III.42. We claim that parts (a), (c), and (h) are essentially equivalent to part (e). For instance, to prove part (h), we note that $(\exists x\, A) \rightarrow C$ is tautologically equivalent to $\neg C \rightarrow \neg \exists x\, A$. The latter is logically equivalent to $\neg C \rightarrow \forall x\, \neg A$ by Theorem III.52. That in turn is logically equivalent to $\forall x\, (\neg C \rightarrow \neg A)$ since part (e) holds in general. This last formula is logically equivalent to $\forall x\, (A \rightarrow C)$, and thus part (h) is proved. Similar arguments prove parts (a) and (c) from (e) by expressing $\wedge$ and $\vee$ in terms of $\rightarrow$ and $\neg$.

Now let's prove part (d). As is proved in Exercise III.15, $\exists x\, (C \vee A)$ is logically equivalent to $\exists x\, C \vee \exists x\, A$.[11] And since $x$ is not free in $C$, Exercise III.16 states that $\exists x\, C$ is logically equivalent to $C$. It follows that $\exists x\, (C \vee A)$ is logically equivalent to $C \vee \exists x\, A$, so (d) is proved. Parts (b), (f), and (g) are essentially equivalent to part (b), analogously to the way that parts (a), (c), (e), and (h) were essentially equivalent to each other. □

Theorem III.79 is now proved by using the lemma; the main new thing is that we have to handle the case where $x$ is free in $C$ by using alphabetic variants.

[11] The application of Exercise III.15 does not need the assumption that $x$ is not free in $C$.

*Proof of Theorem III.79 on prenex formulas.* Without loss of generality, $A$ does not contain the symbol $\leftrightarrow$ since any subformula $B \leftrightarrow C$ can replaced by the tautologically equivalent $(B \to C) \wedge (C \to B)$.

We modify $A$ step-by-step by moving quantifiers in front of propositional connectives. Each step will reduce the number of pairs of occurrences of propositional connectives ($\neg$ or $\circ$) and quantifiers $Qx$ such that $Qx$ lies the scope of the propositional connective. If $A$ is not in prenex form, then there must be at least one subformula of the form $\neg Qx\,B$ or $Qx\,B \circ C$ or $C \circ Qx\,C$.

Suppose $A$ has a subformula $\neg Qx\,B$ where $Qx$ denotes either $\forall x$ or $\exists x$. Let $\overline{Q}x$ denote the dual quantifier $\exists x$ or $\forall x$, respectively. Then $\neg Qx\,B$ is logically equivalent to $\overline{Q}x\,\neg$. Replacing $\neg Qx\,B$ in $A$ with $\overline{Q}x\,\neg B$ yields a formula that is logically equivalent to $A$ and has one less pair of occurrences of propositional connectives and quantifiers with the quantifier in the scope of the propositional connective.

Now suppose $A$ has a subformula $(Qx\,B) \vee C$. If $x$ does not appear free in $C$, then replace this subformula with the logically equivalent $Qx\,(B \vee C)$; the logical equivalence follows from part (c) or (d) of Lemma III.80. On the other hand, if $x$ does appear free in $C$, choose a new variable $y$ that does not appear anywhere in $A$. Then $Qy\,B(y/x)$ is an alphabetic variant of $Qx\,B$ and thus is logically equivalent to $Qx\,B$. Since $y$ was chosen to be a variable that is not used at all in $A$, it clearly has no free occurrence in $C$. Therefore, $Qy\,(B(y/x) \vee C)$ is logically equivalent to $(Qx\,B) \vee C$. Replacing $(Qx\,B) \vee C$ with $Qy\,(B(y/x) \vee C)$ in $A$ yields a logically equivalent formula. It also reduces the number of pairs of propositional connectives and quantifiers with the quantifier inside the scope of the logical connective.

The remaining cases where $A$ has a subformula of the form $((Qx\,B) \circ C$ or $(C \circ (Qx\,B)$ are similar. These are proved using the other logical equivalences from Lemma III.80.

We leave the rest of the details of the proof to the reader. □

**Example III.81.** Let $A$ be the formula $\forall x\,P(x) \vee \exists y\,Q(z,y)$. Using prenex operations we have the tautological equivalences

$$\forall x\,P(x) \vee \exists y\,Q(z,y) \;\models\!\dashv\; \forall x\,(P(x) \vee \exists y\,Q(z,y)) \;\models\!\dashv\; \forall x\,\exists y\,(P(x) \vee Q(z,y)).$$

Thus $A$ is logically equivalent to the prenex formula $\forall x\,\exists y\,(P(x) \vee Q(x,y))$.

It is possible to pull out the quantifiers in a different order:

$$\forall x\,P(x) \vee \exists y\,Q(z,y) \;\models\!\dashv\; \exists y\,(\forall x\,P(x) \vee Q(z,y)) \;\models\!\dashv\; \exists y\,\forall x\,(P(x) \vee Q(z,y)).$$

So $A$ is also logically equivalent to the prenex formula $\exists y\,\forall x\,(P(x) \vee Q(x,y))$. Of course, these two prenex forms of $A$ are logically equivalent to each other as well.

**Example III.82.** Now let $A$ be the formula $\forall x\,P(x,y) \to \forall x\,\exists y\,Q(x,y)$. The variable $x$ is quantified twice, and the variable $y$ has both free and bound occurrences. Accordingly, we must work with alphabetic variants to convert $A$ into prenex form. This can be done step-by-step as:

$$
\begin{array}{lll}
\forall x\, P(x,y) \to \forall x\, \exists y\, Q(x,y) & & \\
\quad \models\!\dashv & \exists x\, (P(x,y) \to \forall x\, \exists y\, Q(x,y) & \text{Lemma III.80(g)} \\
\quad \models\!\dashv & \exists x\, (P(x,y) \to \forall x'\, \exists y\, Q(x',y) & \text{Alphabetic variant} \\
\quad \models\!\dashv & \exists x\, \forall x'\, (P(x,y) \to \exists y\, Q(x',y) & \text{Lemma III.80(e)} \\
\quad \models\!\dashv & \exists x\, \forall x'\, (P(x,y) \to \exists y'\, Q(x',y') & \text{Alphabetic variant} \\
\quad \models\!\dashv & \exists x\, \forall x'\, \exists y'\, (P(x,y) \to Q(x',y') & \text{Lemma III.80(f)}
\end{array}
$$

Just as in the previous example, it would be possible to pull the quantifiers out in a different order. Indeed, the initial block of quantifiers could also end up as "$\forall x'\, \exists x\, \exists y'$" or "$\forall x'\, \exists y'\, \exists x$".

## III.9 Examples of Logical Principles

This section collects together some logical equivalences and logical implications that pertain to quantifiers. Some of the results below have been proven already. For many of the others, see Exercises III.15-III.17.

**Some logical equivalences for quantifiers.**

$$
\begin{array}{rl}
\forall x\, \forall y\, A \models\!\dashv \forall y\, \forall x\, A & \text{- Quantifier exchange} \\
\exists x\, \exists y\, A \models\!\dashv \exists y\, \exists x\, A & \text{- Quantifier exchange} \\
\forall x\, (A \wedge B) \models\!\dashv \forall x\, A \wedge \forall x\, B & \text{- Distributing } \forall \text{ over } \wedge \\
\exists x\, (A \vee B) \models\!\dashv \exists x\, A \vee \exists x\, B & \text{- Distributing } \exists \text{ over } \vee \\
\exists x\, (A \to B) \models\!\dashv \forall x\, A \to \exists x\, B & \text{- Distributing } \exists \text{ over } \to
\end{array}
$$

**Some logical implications for quantifiers.**

$$
\begin{array}{ll}
\models \forall x\, A \to \exists x\, A & \text{- Corollary III.69} \\
\models \exists x\, \forall y\, A \to \forall y\, \exists x\, A & \\
\models \forall x\, A \to A & \\
\models \forall x\, A \to A(t/x) & \text{- If } t \text{ is substitutable for } x \text{ in } A \\
\models A \to \exists x\, A & \\
\models A(t/x) \to \exists x\, A & \text{- If } t \text{ is substitutable for } x \text{ in } A \\
\models \forall x\, (A \to B) \to \exists x\, A \to \exists x\, B & \\
\models \forall x\, (A \to B) \to \forall x\, A \to \forall x\, B & \\
\models \forall x\, \forall y\, A \to \forall x\, A(x/y) & \text{- If } x \text{ is substitutable for } y \text{ in } A.
\end{array}
$$

The final logical validity may be a little confusing; it can be expressed more clearly with the relaxed notation for substitution as

$$\models \forall x\, \forall y\, A(x,y) \to \forall x\, A(x,x).$$

The implications above in general do not reverse; for instance, $\models A \to \forall x\, A$ is not true in general. They do reverse in some special cases, notably when the quantifiers are "vacuous" and do not actually bind any variables. For instance, the next two equivalences follow from Exercise III.16.

$$
\begin{array}{ll}
\models A \leftrightarrow \forall x\, A & \text{- If } x \text{ does not occur free in } A \\
\models \exists x\, A \leftrightarrow A & \text{- If } x \text{ does not occur free in } A
\end{array}
$$

**Logical equivalences supporting prenex form.** In all of the below formulas, it is assumed that $x$ does not have any free occurrences in $C$. Other than the first four equivalences, these are stated in Lemma III.80.

$$\begin{array}{rl}
\forall x\, A \models\dashv \neg\exists x\, \neg A & \text{- Theorem III.36} \\
\exists x\, A \models\dashv \neg\forall x\, \neg A & \text{- } \prime\prime \\
\neg\forall x\, A \models\dashv \exists x\, \neg A & \text{- Theorem III.52} \\
\neg\exists x\, A \models\dashv \forall x\, \neg A & \text{- } \prime\prime \\
C \wedge \exists x\, A \models\dashv \exists x\, (C \wedge A) & \\
C \wedge \forall x\, A \models\dashv \forall x\, (C \wedge A) & \\
C \vee \forall x\, A \models\dashv \forall x\, (C \vee A) & \\
C \vee \exists x\, A \models\dashv \exists x\, (C \vee A) & \\
C \to \forall x\, A \models\dashv \forall x\, (C \to A) & \text{- Example III.42} \\
C \to \exists x\, A \models\dashv \exists x\, (C \to A) & \\
(\forall x\, A) \to C \models\dashv \exists x\, (A \to C) & \\
(\exists x\, A) \to C \models\dashv \forall x\, (A \to C) &
\end{array}$$

To reiterate: it is assumed that $x$ is not free in $C$ in the above logical equivalences.

## III.10 Semantic Theorems on Constants

A common method of constructing informal proofs is to introduce a new name for some unnamed object. For instance, if we are trying to prove a universal statement $\forall x\, A(x)$, we might introduce a new name "$c$" to represent an arbitrary fixed value of $x$. If we then succeed in proving that $A(c)$ holds then, since $c$ was arbitrary, we can conclude that $\forall x\, A(x)$ holds. Dually, if we are given an existential statement $\exists x\, A(x)$ as a hypothesis, we might introduce a new name "$c$" for an arbitrary object representing an $x$ such that $A(x)$. We then are justified in adding $A(c)$ as a new hypothesis.

These two informal reasoning methods are made formal by the next theorem and corollary.

**Theorem III.83.** *Let $A(x)$ be a formula and $\Gamma$ be a set of formulas. Suppose that the constant symbol $c$ does not appear in $A(x)$ or in any formula in $\Gamma$. Then*

$$\Gamma \cup \{\exists x\, A(x)\} \text{ is satisfiable} \quad \text{if and only if} \quad \Gamma \cup \{A(c)\} \text{ is satisfiable.}$$

*If in addition $B$ is a formula that does not contain $c$, then*

$$\Gamma,\, \exists x\, A(x) \models B \quad \text{if and only if} \quad \Gamma,\, A(c) \models B.$$

**Corollary III.84.** *Let $A(x)$ be a formula. Let $\Gamma$ be a set of formulas such that the constant symbol $c$ does not appear in $A(x)$ or any formula in $\Gamma$. Suppose $\Gamma \models A(c)$. Then $\Gamma \models \forall x A(x)$.*

The corollary is proved by applying the theorem to the formula $\neg A$ and using the proof by contradiction principle of Theorem III.45. The basic idea for the proof of the first part of Theorem III.83 is rather simple: if we have a model for $\exists x\, A(x)$, we can enlarge the language to include $c$ as a new constant symbol, and let the interpretation of $c$ be some object that makes $A(c)$ true. Such an object is guaranteed to exist if and only if $\exists x\, A(x)$ is true. Making this intuition formal takes a little work and we start with a rather general definition that will be useful later too.

**Definition III.85.** Let $L$ and $L'$ be languages with $L \subseteq L'$. Suppose $\mathfrak{A}$ is an $L$-structure and $\mathfrak{B}$ is an $L'$-structure. We say that $\mathfrak{B}$ is an *expansion* of $\mathfrak{A}$, and that $\mathfrak{A}$ is a *restriction* of $\mathfrak{B}$ provided that

- They have the same universe: $|\mathfrak{A}| = |\mathfrak{B}|$, and
- The interpretations of the symbols in $L$ are the same in $\mathfrak{A}$ and $\mathfrak{B}$. In other words, $c^{\mathfrak{A}} = c^{\mathfrak{B}}$ and $P^{\mathfrak{A}} = P^{\mathfrak{B}}$ and $f^{\mathfrak{A}} = f^{\mathfrak{B}}$ for all symbols $c, P, f \in L$.

**Example III.86.** $\mathcal{N}' = (\mathbb{N}, 0, S, +, \cdot, <)$ is an expansion of $\mathcal{N} = (\mathbb{N}, 0, S, +, \cdot)$.

As hinted at in the example, any definable predicate, object, or function of a structure $\mathfrak{A}$ can be used to form an extension of $A$. Much more general constructions can be used for expansions of course. A basic property of expansions is that the truth of an $L$-sentence is the same in $A$ and $B$.

**Theorem III.87.** *Let $L$, $L'$, $\mathfrak{A}$ and $\mathfrak{B}$ be as above with $\mathfrak{B}$ an expansion of $\mathfrak{A}$. Let $\sigma$ be any object assignment. (Note that although $\mathfrak{A}$ and $\mathfrak{B}$ have different languages, they have the same object assignments by virtue of having the same universe.)*

(a) *For any formula $A$, $\mathfrak{A} \vDash A[\sigma]$ if and only if $\mathfrak{B} \vDash A[\sigma]$.*
(b) *For any sentence $A$, $\mathfrak{A} \vDash A$ if and only if $\mathfrak{B} \vDash A$.*

A formal proof of Theorem III.87 would use induction on the complexity of the formula $A$. However, we omit the proof, since it should be completely clear that the theorem must hold: Examining the definition of truth, the truth of a formula in a structure depends only on the universe and on the interpretations of the non-logical symbols that appear in the formula. And, the formula $A$ uses only non-logical symbols from $L$, which have the same interpretations in $\mathfrak{A}$ as in $\mathfrak{B}$. □

*Proof of Theorem III.83.* Let $L$ be the language containing the non-logical symbols that appear in $A$ and $\Gamma$. Let $L'$ be $L \cup \{c\}$ where $c \notin L$ is a constant symbol. The first part of the theorem states that $\Gamma \cup \{\exists x\, A(x)\}$ is satisfied by some $L$-structure $\mathfrak{A}$ if and only if $\Gamma \cup \{A(c)\}$ is satisfied by some $L'$-structure $\mathfrak{B}$.

First suppose that an $L'$-structure $\mathfrak{B}$ with object assignment $\sigma$ satisfies $\Gamma \cup \{A(c)\}$, namely $\mathfrak{B} \vDash A(c)[\sigma]$. Then certainly $\mathfrak{B} \vDash \exists x\, A(x)[\sigma]$. Letting $\mathfrak{A}$ be the restriction of $\mathfrak{B}$ to the language $L$, Theorem III.87 implies that $\mathfrak{A} \vDash (\Gamma \cup \{\exists x\, A(x)\})[\sigma]$.

Second, suppose that $(\mathfrak{A}, \sigma)$ satisfies $\Gamma \cup \{\exists x\, A(x)\}$. By the definition of truth, there is a $a \in |\mathfrak{A}|$ such that $\mathfrak{A} \vDash A(x)[\tau]$ where $\tau$ is the $x$-variant of $\sigma$ with

$\tau(x) = a$. Let $\mathfrak{B}$ be the expansion of $\mathfrak{A}$ to the language $L'$ obtained by letting $c^{\mathfrak{B}} = a$. By Theorem III.61(b),

$$\mathfrak{B} \vDash (x = c \to (A(x) \leftrightarrow A(c))[\tau].$$

Since $\tau(x) = c^{\mathfrak{B}}$ and $\mathfrak{B} \vDash A(x)[\tau]$, we have $\mathfrak{B} \vDash A(c)[\tau]$. Since $\tau$ is an $x$-variant of $\sigma$, $\mathfrak{B} \vDash A(c)[\sigma]$. Finally, $\mathfrak{B} \vDash \Gamma[\sigma]$ by Theorem III.87 since $\mathfrak{A} \vDash \Gamma[\sigma]$. That proves the first part of the theorem.

The second part of the theorem follows immediately since $\Gamma, \exists x\, A(x) \vDash B$ holds if and only $\Gamma, \neg B, \exists x\, A(x)$ is unsatisfiable and since $\Gamma, A(c) \vDash B$ holds if and only $\Gamma, \neg B, A(c)$ is unsatisfiable. □

The next theorem lets us reexpress a logical implication $\Gamma \vDash A$ involving formulas as a logical implication involving only sentences. The general idea is to replace free variables in $\Gamma$ and $A$ with new constant symbols. To illustrate this, consider the fact that $P(x, y) \vDash \neg\neg P(x, y)$. We wish to reexpress this using sentences instead of the formulas involving the free variables $x$ and $y$. For this, we introduce two new constant symbols, say $c$ and $d$. Then $P(x, y) \vDash \neg\neg P(x, y)$ is equivalent to $P(c, d) \vDash \neg\neg P(c, d)$.

To formalize this generally, we start with an $L$-formula $A$ and a set $\Gamma$ of $L$-formulas. We let $L'$ be $L \cup \{d_1, d_2, d_3, \ldots\}$ with infinitely many new constant symbols. We write $A(\vec{d}/\vec{x})$ and $\Gamma(\vec{d}/\vec{x})$ to denote the result of replacing every free occurrence of every $x_i$ with $d_i$. Thus, $A(\vec{d}/\vec{x})$ is an $L'$-sentence and $\Gamma(\vec{d}/\vec{x})$ is a set of $L'$-sentences.[12]

**Theorem III.88.** *Let $L$, $L'$ and $d_1, d_2, d_3, \ldots$ be as above. Let $\Gamma$ be a set of $L$-formulas, and $A$ be an $L$-formula. (So the constant symbols $d_i$ do not appear in $\Gamma$ or $A$.)*

(a) *$\Gamma$ is satisfiable if and only if $\Gamma(\vec{d}/\vec{x})$ is satisfiable.*

(b) *$\Gamma \vDash A$ if and only if $\Gamma(\vec{d}/\vec{x}) \vDash A(\vec{d}/\vec{x})$.*

*Proof.* To state (a) more precisely, it states that there is an $L$-structure $\mathfrak{A}$ and an object assignment $\sigma$ so that $(\mathfrak{A}, \sigma)$ satisfies $\Gamma$ if and only if there is a $L'$-structure $\mathfrak{B}$ which satisfies $\Gamma(\vec{d}/\vec{x})$. The relationship between $\mathfrak{A}$ and $\mathfrak{B}$ is that $\mathfrak{B}$ is the expansion of $\mathfrak{A}$ to the language $L'$ with $d_i^{\mathfrak{B}} = \sigma(x_i)$. Reasoning as in the proof of Theorem III.83 proves the desired equivalence.

Part (b) is an immediate consequence of part (a). □

# III.11 Definability

## III.11.1 Definability of structures

Many mathematical notions such as groups, rings, fields, integral domains, graphs, partial orders, total orderings, lattices, Boolean algebras, etc. can be

[12] The notation here assumes that the language $L$ is countable, and hence the number of formulas in $\Gamma$ and, in particular, the number of free variables in $\Gamma$ is countable. The construction works, however, also for uncountable languages where $\Gamma$ might use uncountably many variables.

defined with first-order axioms. The example of groups was discussed in Section III.1. Letting $L = \{1, \cdot, ()^{-1}\}$, where 1 is a constant symbol, $\cdot$ is a binary function symbol, and $()^{-1}$ is a unary function symbol, then the class of all groups can be characterized as being the class of $L$-structures that satisfy the three (first-order) axioms for groups given in Equations (III.6).[13] Because there are only finitely many axioms for groups, we call the class of groups an "elementary class".[14]

**Definition III.89.** Let $L$ be a language. If $\Gamma$ is a set of $L$-sentences, then $\operatorname{Mod}\Gamma$, the *models of* $\Gamma$ is the class of $L$-structures which are models of $\Gamma$. If $A$ is a sentence, then $\operatorname{Mod} A$ denotes the class of $L$-structures which satisfy $A$, namely $\operatorname{Mod}\{A\}$.

**Definition III.90.** Let $L$ be a language and $\mathcal{S}$ be a class of $L$-structures. The class $\mathcal{S}$ is an *elementary class* (EC) if there is an $L$-sentence $A$ such that $\mathcal{S} = \operatorname{Mod} A$. The class $\mathcal{S}$ is an *elementary class in the wide sense* ($\mathrm{EC}_\Delta$) if there is a set of $L$-sentences $\Gamma$ such that $\mathcal{S} = \operatorname{Mod}\Gamma$.

If $\Gamma$ is finite, then we can let $A$ be the sentence $\bigwedge\Gamma$, the conjunction of the members of $\Gamma$. Therefore, for a finite set $\Gamma$, $\operatorname{Mod}\Gamma = \operatorname{Mod} A$ is EC. For instance, the class of groups is EC.

The torsion-free groups is an example of an elementary class in the wide sense, namely an $\mathrm{EC}_\Delta$ class. Recall that $T_k$ was defined to be the sentence $\forall x(x \neq 1 \to x^k \neq 1)$. Letting $\Gamma$ be the set of sentences consisting of the three group axioms and the sentences $T_k$ for $k \geq 2$, the class of torsion-free groups is equal to $\operatorname{Mod}\Gamma$ and thus is $\mathrm{EC}_\Delta$. On the other, it will be a consequence of the Compactness Theorem in Section IV.5 that the class of torsion-free groups is not EC and that the class of groups that are not torsion-free is not even $\mathrm{EC}_\Delta$.

**Example III.91.** We show that directed graphs and also undirected graphs are elementary classes. Let the language be $L = \{E\}$ for $E$ a binary connective. For directed graphs, we view $E(x, y)$ as meaning there is an edge from vertex $x$ to vertex $y$. Let $A$ be the sentence $\forall x\,(\neg E(x, x))$; this expresses the property that there are no loops in the directed graph. Then the class of directed graphs (without loops) is equal to $\operatorname{Mod} A$, and hence it is EC.

The class of undirected graphs uses the same language $L$. In an undirected graph, the edge relation must be symmetric. This can be expressed by the sentence $B$ equal to $\forall x\,\forall y\,(E(x, y) \leftrightarrow E(y, x))$. Since the class of undirected graphs has two axioms $A$ and $B$, it is EC.

Exercise III.27 asks you to prove that the class of infinite graphs and the class of cycle-free graphs are elementary in the wide sense ($\mathrm{EC}_\Delta$). These two classes are not EC.

**Example III.92.** A *linear order* is a structure over the language $L = \{<\}$ which satisfies the following three sentences (axioms):

[13] For readers familiar with set theory: We call it the *class* of groups instead of the *set* of all groups, because the class of all groups is not a set.

[14] In this context, the term "elementary" means "first-order".

(a) $\forall x\,(\neg\, x < x)$. Irreflexivity.

(b) $\forall x\,\forall y\,\forall z\,(x < y \wedge y < z \rightarrow x < z)$. Transitivity.

(c) $\forall x\,\forall y\,(x = y \vee x < y \vee y < x)$. Strong connectivity.

A linear order is *dense* if it additionally satisfies the axiom

(d) $\forall x\,\forall y\,(x < y \rightarrow \exists z\,(x < z \wedge z < y))$.

That is, a linear order is dense if, for every two distinct objects $x$ and $y$, there is an object $z$ strictly between $x$ and $y$. Clearly, the class of strict linear orders and the class of dense linear orders (DLO's) are both EC.

**Example III.93.** A field is a structure over the language $0, 1, +, \cdot$ where 0 and 1 are constant symbols and $+$ and $\cdot$ are binary function symbols. The usual axioms for fields are the sentences:

(a) $\forall x\,\forall y\,(x + y = y + x)$. Addition is commutative.[15]

(b) $\forall x\,\forall y\,(x + (y + z) = (x + y) + z)$. Addition is associative.

(c) $\forall x\,(x + 0 = x \wedge 0 + x = x)$. Additive identity.

(d) $\forall x\,\exists y\,(x + y = 0 \wedge y + x = 0)$. Additive inverses.

(e) $\forall x\,\forall y\,(x \cdot y = y \cdot x)$. Multiplication is commutative.

(f) $\forall x\,\forall y\,(x \cdot (y \cdot z) = (x \cdot y) \cdot z)$. Multiplication is associative.

(g) $\forall x\,(x \cdot 1 = x \wedge 1 \cdot x = x)$. Multiplicative identity.

(h) $\forall x\,(x \neq 0 \rightarrow \exists y\,(x \cdot y = 1 \wedge y \cdot x = 1))$. Multiplicative inverses.

(i) $\forall x\,\forall y\,\forall z\,((x + y) \cdot z = (x \cdot z) + (y \cdot z))$. Distributivity.

(j) $0 \neq 1$.

Thus the class of fields is EC.

**Example III.94.** An ordered field is a field that has a strict linear order that is compatible with the field operations. The language of ordered fields is $0, 1, +, \cdot, <$. The axioms for ordered fields consist of the axioms (a)-(c) of strict linear orders from Example III.92, axioms (a)-(j) for fields from Example III.93, plus the following two axioms that connect the order $<$ to the field operations.

(a) $\forall x\,\forall y\,\forall z\,(x < y \rightarrow x + z < y + z)$.

(b) $\forall x\,\forall y\,(0 < x \wedge 0 < y \rightarrow 0 < x \cdot y)$.

Thus the class of ordered fields is EC.

**Example III.95.** A real closed field is an ordered field that satisfies exactly the same first-order properties as the field $\mathcal{R} = (\mathbb{R}, 0, 1, +, \cdot, <)$ of real numbers. The axioms for real closed fields include all the axioms for ordered fields, plus the following infinite set of axioms:

[15] The commutativity of addition follows from (i.e., is logically implied by) the other nine axioms. Thus, this axiom could be omitted.

(a) Every positive field element has a square root:

$$\forall x\,(0 < x \to \exists y\,(y \cdot y = x)).$$

(b) Every odd degree polynomial has a root: namely, if $n$ is odd:

$$\forall a_0\,\forall a_1 \cdots \forall a_n\,(a_n \neq 0 \to \exists x\,(a_n \cdot x^n + a_{n-1} \cdot x^{n-1} + \cdots + a_1 x + a_0 = 0)).$$

Here $x^k$, for $k \in \mathbb{N}$, means the $k$-fold product of $x$ with itself. (The exponential function is *not* part of the language for real closed fields.)

There are infinitely many axioms for real closed fields of type (b), namely one for each odd $n \geq 3$. Hence the class of real closed fields is $\mathrm{EC}_\Delta$. (It can be shown that the class of real closed fields is not EC.)

The most prominent example of a model of the axioms of real closed fields is the field $\mathcal{R} = (\mathbb{R}, 0, 1, +, \cdot, <)$ of the real numbers. It is straightforward to see that all the axioms of real closed fields are satisfied by $\mathcal{R}$. More than this is true: the above axioms for real closed fields are sufficient to logically imply all sentences that are true in $\mathbb{R}$. The next definitions make this precise.

**Definition III.96.** A *theory* over a language $L$ is set $T$ of $L$-sentences which is closed under logical consequence. In other words, if $A$ is an $L$-sentence and $T \vDash A$, then $A \in T$.

**Example III.97.** The theory of groups is the set of sentences that are true in all groups. Equivalently, the theory of groups is the set of logical consequences of the three axioms for groups that were given earlier in (III.6).

The theory DLO of dense linear orders is the set of logical consequences of the axioms (a)-(d) given in Example (III.92).

The theory RCF of real closed fields is the set of logical consequences of the axioms for real closed fields presented across Examples (III.92)-(III.95).

The example of RCF is particularly interesting because it turns out that the axioms for RCF completely characterize what is true in the structure $\mathbb{R}$ of real numbers. Indeed, for any sentence $A$ in the language of RCF either $\mathrm{RCF} \vDash A$ or $\mathrm{RCF} \vDash \neg A$.[16] Since RCF is closed under logical consequence this means that, for every sentence $A$, either $A \in \mathrm{RCF}$ or $\neg A \in \mathrm{RCF}$. That is to say, RCF is "complete":

**Definition III.98.** A theory $T$ (over the language $L$) is *complete* if, for every $L$-sentence $A$, either $A \in T$ or $\neg A \in T$.

**Definition III.99.** Let $\mathfrak{A}$ be an $L$-structure. The *theory* of $\mathfrak{A}$ is denoted $\mathrm{Th}\,\mathfrak{A}$ and is the set of $L$-sentences

$$\mathrm{Th}\,\mathfrak{A} \;=\; \{A : A \text{ is an } L\text{-sentence and } \mathfrak{A} \vDash A\}.$$

[16] This important result was proved by Tarski [19], but its proof is too complex for us to present it.

Note that $\operatorname{Th}\mathfrak{A}$ must be complete. This is because any sentence $A$ will be either true or false in $\mathfrak{A}$, and thus one of $A$ or $\neg A$ will be in $\operatorname{Th}\mathfrak{A}$.

**Definition III.100.** Two $L$-structures $\mathfrak{A}$ and $\mathfrak{B}$ are *elementarily equivalent* if $\operatorname{Th}\mathfrak{A}$ and $\operatorname{Th}\mathfrak{B}$ are equal. We write $\mathfrak{A} \equiv \mathfrak{B}$ to denote that $\mathfrak{A}$ and $\mathfrak{B}$ are elementarily equivalent.

Unwinding the definitions, $\mathfrak{A}$ and $\mathfrak{B}$ are elementarily equivalent if and only if they satisfy exactly the same $L$-sentences. The adverb "elementarily" is used since "elementary" is sometimes used to mean "first-order".

It is also useful to talk about the theory of a class of structures:

**Definition III.101.** Let $\mathcal{S}$ be a class of $L$-structures. The *theory* of $\mathcal{S}$ is denoted $\operatorname{Th}\mathcal{S}$ and is the set of $L$-sentences true in all members of $\mathcal{S}$.

$$\operatorname{Th}\mathcal{S} \;=\; \{A : A \text{ is an } L\text{-sentence and } \mathfrak{A} \vDash A \text{ for all } \mathfrak{A} \in \mathcal{S}\}.$$

For example, if $\mathcal{S}$ is the class of all groups, then $\operatorname{Th}\mathcal{S}$ is the theory of groups, namely the set of sentences true in all groups. In general, $\operatorname{Th}\mathcal{S}$ may not be complete. For single structure $\mathfrak{A}$, $\operatorname{Th}\mathfrak{A}$ is the same as $\operatorname{Th}\{\mathfrak{A}\}$ and is complete.

**Example III.102.** $\operatorname{Th}\mathcal{R}$ is equal to RCF. Any two models of RCF are elementarily equivalent. This follows immediate from the fact that RCF is complete and that $\mathcal{R} \vDash \text{RCF}$ so $\text{RCF} \subseteq \operatorname{Th}\mathcal{R}$. (The proof of these assertions is beyond the scope of this book.)

**Example III.103.** The theory of groups is not complete. For example, some groups are order two and satisfy $\forall x\,(x \cdot x = 1)$, whereas other groups do not satisfy this.

**Example III.104.** The theory DLO of dense linear orders is not complete. For example, consider the following sentence which asserts that there is no least member of the order (i.e., there is no "left endpoint")

$$\forall x\,\exists y\,(y < x). \tag{III.36}$$

This sentence is true in some models of DLO; for instance in the structure $(\mathbb{Q}, <)$ of the usual ordering on the rational numbers. On the other hand, it is false in other models of DLO. An example is $(\mathbb{Q}^{\geq 0}, <)$ of the usual ordering on the nonnegative rational numbers. It is easy to check that both $(\mathbb{Q}, <)$ and $(\mathbb{Q}^{\geq 0}, <)$ satisfy all the axioms for dense linear order.

**Example III.105.** The theory of *dense linear order (DLO) without endpoints* is axiomatized as DLO plus two axioms asserting there is no minimum or maximum element:

$$\forall x\,\exists y\,(y < x) \qquad \text{and} \qquad \forall x\,\exists y\,(x < y).$$

The first of these was already given in Equation (III.36). The set of rationals $(\mathbb{Q}, <)$ is an example of a model of this theory. It can be shown that the theory of DLO without endpoints is complete. In fact, any countable model of the theory DLO without endpoints is isomorphic to $(\mathbb{Q}, <)$.

**The theory of arrays.** The theory of arrays is rather different from the above examples; instead of defining a conventional mathematical structure, it defines a dynamic construction used in computer programming. As such, the theory of arrays might be a small part of a theory for proving the correctness of programs. Indeed, the theory of arrays can be incorporated into widely used software verification theorem provers.

The theory of arrays deals with three sorts of objects: arrays, indices, and values. When writing an informal expression `a[i] = x`, the "`a`" is an array, "`i`" is an index, and "`x`" is a value. The most natural way to deal with these three sorts of objects is to use a "sorted" logic where variables are labeled explicitly as being of different "sorts" or "types" and as ranging over an appropriate (sub)universe of objects. In other words, a sorted logic uses a separate universe for each sort of object. To formalize sorts within our (non-sorted) first-order logic, we introduce three unary predicates *Array*, *Index* and *Value* and let the formulas

$$Array(z), \qquad Index(z) \quad \text{and} \qquad Value(z)$$

denote that $z$ is an array, an index, or a value, respectively. We could optionally include an axiom that states that every object $z$ satisfies exactly one of the three predicates, but it is not strictly speaking necessary.

The theory of arrays uses two function symbols *Read* and *Write*. The *Read* function takes two inputs: an array $a$ and an index $i$. It returns a value $x$, which is intended to be the value stored in the array $a$ at the location specified by the index $i$. The function $Write(a, i, x)$ takes three inputs: an array $a$, an index $i$ and a value $x$. It returns an array $b$ which is the array obtained from $a$ by changing the value at the location indexed by $i$ to the value $y$. Note that the intuitive action of the function *Write* is that it makes a copy of the array $a$, updates its value at location $i$ to equal $x$, and returns the new updated array. This differs somewhat from the way software programs update an array value without making a copy of the array, but it still allows reasoning about the action of read and write operations.

In the literature, the axioms for arrays would typically be expressed by using (something similar to) the following implications:

$$a : Array; i : Index \Rightarrow Read(a, i) : Value$$
$$a : Array; i : Index; x : Value \Rightarrow Write(a, i, x) : Array$$
$$a : Array; i : Index; x : Value \Rightarrow Read(\, Write(a, i, x), i) = x$$
$$a : Array; i, j : Index; x : Value \Rightarrow i \neq j \rightarrow Read(\, Write(a, i, x), j) = Read(a, j)$$
$$a, b : Array \Rightarrow \forall i : Index(Read(a, i) = Read(b, i)) \rightarrow a = b$$

A notation such as "$a : Array$" means that $a$ is a variable that ranges over objects of type "array". Similarly, "$Read(a, i) : Value$" means that that $Read(a, i)$ is an object of type "value". The variables in the implications are implicitly universally quantified. The semicolons to the expression before the "$\Rightarrow$" act as if they are conjunctions. We do *not* adopt these notations; instead, the above

axioms can be reexpressed in the syntax of first-order logic as:

$$\forall a\, \forall i\, [\mathit{Array}(a) \wedge \mathit{Index}(i) \rightarrow \mathit{Value}(\mathit{Read}(a,i))]$$
$$\forall a\, \forall i\, \forall x\, [\mathit{Array}(a) \wedge \mathit{Index}(i) \wedge \mathit{Value}(x) \rightarrow \mathit{Array}(\mathit{Write}(a,i,x))]$$
$$\forall a\, \forall i\, \forall x\, [\mathit{Array}(a) \wedge \mathit{Index}(i) \wedge \mathit{Value}(x) \rightarrow \mathit{Read}(\mathit{Write}(a,i,x),i) = x]$$
$$\forall a\, \forall i\, \forall j\, \forall x\, [\mathit{Array}(a) \wedge \mathit{Index}(i) \wedge \mathit{Index}(j) \wedge \mathit{Value}(x) \wedge i \neq j \rightarrow \mathit{Read}(\mathit{Write}(a,i,x),j) = \mathit{Read}(a,j)]$$
$$\forall a\, \forall b\, (\mathit{Array}(a) \wedge \mathit{Array}(b) \rightarrow [\forall i(\mathit{Index}(i) \rightarrow \mathit{Read}(a,i) = \mathit{Read}(b,i)) \rightarrow a = b]$$

*Array*, *Index*, and *Value* are unary predicates; *Read* is a binary function symbol, and *Write* is a 3-ary function symbol. The first two axioms specify the sorts of the functions *Read* and *Write*. The third axiom states that if one writes $x$ to location $i$ of an array, and reads from the same location, one gets the value $x$ back. The fourth axiom states that if one writes $x$ to location $i$, it does not change the value stored at any different location $j$.

The fifth axiom is called the axiom of extensionality. It states that if two arrays have exactly the same contents, then they are actually the same array. By equality principle of Theorem III.76 the converse implication will automatically hold; namely, if $a = b$ then $\forall i\, (\mathit{Read}(a,i) = \mathit{Read}(b,i))$ holds. Thus, the axiom of extensionality actually states that $\forall i(\mathit{Index}(i) \rightarrow \mathit{Read}(a,i) = \mathit{Read}(b,i))$ can be taken as the *definition* of equality.

### III.11.2 Definability in structures

We discussed above how a set of axioms can define a theory $T$, and thus a class of structures $\operatorname{Mod} T$. For a different notion of first-order definability, we work with a fixed structure $\mathfrak{A}$ and discuss how objects, relations, or functions over $\mathfrak{A}$ can be defined using first-order formulas. First, a formula with $k$ free variables defines a $k$-ary relation in the natural way:

**Definition III.106** (Definability of a relation in $\mathfrak{A}$)**.** Let $\mathfrak{A}$ be a structure, and let $B = B(x_1, \ldots, x_k)$ be a first-order formula containing only $x_1, \ldots, x_k$ free. Then $B$ *defines* the $k$-ary relation on $|\mathfrak{A}|$ equal to

$$\{\langle a_1, \ldots, a_k\rangle : a_1, \ldots, a_k \in |\mathfrak{A}| \text{ and } \mathfrak{A} \vDash B(a_1, \ldots, a_k)\}.$$

This relation is said to be *definable* in $\mathfrak{A}$.

For example, we earlier discussed that the center of a group can be defined by the formula $\forall x_2\, (x_1 \cdot x_2 = x_2 \cdot x_1)$. As another example, the earlier-discussed formula $\mathit{Prime}(x_1)$ defines the set of primes in the structure $\mathcal{N}$ of nonnegative integers.

A related notion is the definability of a particular object in $\mathcal{A}$. This is the same as defining a unary relation that has a single member:

**Definition III.107** (Definability of an individual in $\mathfrak{A}$)**.** Let $\mathfrak{A}$ be a structure and let $a \in |\mathfrak{A}|$. The object $a$ is *definable* in $\mathfrak{A}$ if there is a formula $B(x_1)$ with

$x_1$ the only variable free in $B$ so that $a$ is the unique member of $|\mathfrak{A}|$ such that $\mathfrak{A} \vDash B(a)$. In this case, we say that $B$ *defines* the object $a$ in $\mathfrak{A}$.

For example, in $\mathcal{N} = (\mathbb{N}, 0, S, +, \cdot)$, the object 1 is defined by $x_1 = S(0)$ and also by $\forall x_2\,(x_1 \cdot x_2 = x_2)$.

**Definition III.108.** Let $B(x)$ be a formula. We write $\exists! x\, B$, namely "there exists a unique $x$ such that $B$ holds", as an abbreviation for the formula

$$\exists x\,[B(x) \wedge \forall y\,(B(y) \to y = x)].$$

Thus, $B(x)$ defines an object in $\mathfrak{A}$ if and only if $\mathfrak{A} \vDash \exists! x\, B(x)$. Finally, a function is said to be definable in $\mathfrak{A}$ if and only if its graph is definable in $\mathfrak{A}$:

**Definition III.109** (Definability of a function in $\mathfrak{A}$)**.** Let $\mathfrak{A}$ be a structure, and let $B = B(x_1, \ldots, x_k, x_{k+1})$ be a first-order formula containing only $x_1, \ldots, x_{k+1}$ free. Suppose that

$$\mathfrak{A} \vDash \forall x_1 \ldots \forall x_k\, \exists! x_{k+1} B(x_1, \ldots, x_k, x_{k+1}).$$

Then $B$ *defines* the $k$-ary function $f$ on $|\mathfrak{A}|$ such that, for $a_1, \ldots, a_k \in |\mathfrak{A}|$, we have $f(a_1, \ldots, a_k)$ is equal to the (unique) $a_{k+1}$ in $|\mathfrak{A}|$ such that $\mathfrak{A} \vDash B(a_1, \ldots, a_k, a_{k+1})$ holds. When this holds, we say that $f$ is *definable* in $\mathfrak{A}$.

**Example III.110.** Let $\mathfrak{A}$ be a linear order. Thus $\mathfrak{A}$ is an $L$-structure for the language $L = \{<\}$. A non-strict ordering $\leq$ can be defined for $\mathfrak{A}$ with the formula

$$x_1 < x_2 \vee x_1 = x_2.$$

**Example III.111.** As usual, let $\mathcal{N}$ be the nonnegative integers with the language $0, S, +, \cdot$. The binary relation $\leq$ can be defined by

$$\exists x_3\,(x_1 + x_3 = x_2).$$

This is verified by noting that the condition is indeed equivalent to $x_1 \leq x_2$. The relation $<$ can be defined by $\exists x_3\,(x_1 + S(x_3) = x_2)$.

The integer square root function $x \mapsto \lfloor \sqrt{x} \rfloor$ can be defined in $\mathcal{N}$ by

$$x_2 \cdot x_2 \leq x_1 \wedge x_1 < S(x_2) \cdot S(x_2).$$

This last example illustrates an important principle: if a function, predicate, or object is definable in an $L$-structure $\mathfrak{A}$, then we can effectively augment the language $L$ to include that definable function or predicate or object. Thus, it was permissible to claim that the square root function $\lfloor \sqrt{x} \rfloor$ is definable by defining with a formula using $<$ and $\leq$, since the definable symbols $<$ and $\leq$ could be replaced with the use of formulas over the original language $(0, S, +, \cdot)$. In particular, $\lfloor \sqrt{x} \rfloor$ is defined by

$$\exists x_3\,(x_2 \cdot x_2 + x_3 = x_1) \wedge \exists x_3\,(x_1 + S(x_3) = S(x_2) \cdot S(x_2)).$$

It is easy to replace defined predicate symbols with their definition. It is a little more difficult to replace defined objects or defined functions with their definitions. The next section will describe formally how to remove defined objects and functions; in the meantime, the next example illustrates the general principle.

**Example III.112.** Let $\mathfrak{A}$ be an $L$-structure. Suppose that, in $\mathfrak{A}$, an object $a$ is defined by the formula $B_a(x_1)$, a unary function $f$ is defined by the formula $B_f(x_1, x_2)$, and a binary function $g$ is defined by $B_g(x_1, x_2, x_3)$. Let $P$ be a unary predicate symbol in $L$. Then $P(g(f(x), a)$ is can be expressed in $\mathfrak{A}$ by the formula

$$\exists u\, \exists v\, \exists w\, (B_a(u) \wedge B_f(x, v) \wedge B_g(v, u, w) \wedge P(w)).$$

Note that $B_a(u)$ enforces that $u = a$, and $B_f(x, v)$ enforces that $f(x) = v$, and finally $B_g(v, u, w)$ enforces that $w = g(v, u) = g(f(x), a)$.

The above definitions were about definability in a structure; The next section will define the notion of definability in a *theory* using essentially the same constructions.

## III.12 Extensions by Definitions

Section III.11 defined what it means for predicates, constants, and functions to be definable in a structure. We now discuss how the same constructions can be used to define and introduce new predicate symbols, constant symbols, or function symbols in a theory. We shall see that adding new defined symbols is "conservative" in that does not increase the power of the theory.

For the rest of this section, let $T$ be a fixed $L$-theory. Symbols such as $Q$, $f$, $c$ are presumed to be new non-logical symbols that are not in $L$.

**Definition III.113.** Any formula $B(x_1, \ldots, x_n)$ with no free variables other than $x_1, \ldots, x_n$ *defines* an $n$-ary predicate $Q$ in $T$; the *defining axiom* for $Q$ is the sentence $Def_Q$:

$$\forall x_1 \cdots \forall x_n\, [Q(x_1, \ldots, x_n) \leftrightarrow B(x_1, \ldots, x_n)].$$

**Definition III.114.** Let $L' \supseteq L$. Suppose $\Gamma$ is an set of $L$-sentences and $\Delta$ is a set of $L'$-sentences. We say that $\Delta$ is a *conservative extension* of $\Gamma$ provided that $\Delta \models \Gamma$ and that, for every $L$-sentence $A$, if $\Delta \models A$ then $\Gamma \models A$. We write $\Gamma \preccurlyeq \Delta$ to denote that $\Delta$ is a conservative extension of $\Gamma$.

Thus, if $\Delta$ is a conservative extension $\Gamma$ then $\Gamma$ and $\Delta$ have the same $L$-consequences. This is useful when we want to augment $\Gamma$ with $L'$-sentences without changing the $L$-consquences.

**Theorem III.115.** *Let $T$ be a theory. Let the $n$-ary predicate $Q$ be defined by $B(x_1, \ldots, x_n)$. Let $L' = L \cup \{Q\}$ and $T'$ be the theory with axioms $T + Def_Q$. Then*

(a) *$T'$ is a conservative extension of $T$.*

(b) *For every $L'$-formula $A$ there is an $L$-formula $A^*$ such that $T' \vDash A \leftrightarrow A^*$.*

The theorem states that when the defined predicate $Q$ is added to $T$, the resulting theory $T'$ cannot prove any new $L$-sentence that $T$ could not already prove. Furthermore, anything that can be expressed with the new symbol $Q$ can already be expressed without it.

*Proof.* We first prove (a). Clearly $T' \vDash T$. Suppose $A$ is an $L$-sentence such that $T \nvDash A$. We'll show that $T' \nvDash A$ to prove the conservativity. Since $T \nvDash A$, there is a model $\mathfrak{A}$ of $T \cup \{\neg A\}$. Expand $\mathfrak{A}$ to a $L'$-model $\mathfrak{B}$ by choosing the interpretation of $Q$ to be

$$Q^{\mathfrak{B}} \;=\; \{\langle a_1, \ldots, a_n\rangle : \mathfrak{A} \vDash B(a_1, \ldots, a_n),\ a_1, \ldots, a_n \in |\mathfrak{A}|\}.$$

Since $\mathfrak{B}$ is an expansion of $\mathfrak{A}$, exactly the same $L$-sentences are true in $\mathfrak{A}$ as are true in $\mathfrak{B}$ (by Theorem III.87). By choice of $Q^{\mathfrak{B}}$, we have $\mathfrak{B} \vDash Def_Q$. Therefore $T' \cup \{\neg A\}$ is satisfied by $\mathfrak{B}$. Hence $T' \nvDash A$.

The proof of (b) is almost trivial. To form $A^*$, replace every subformula in $A$ of the form $Q(t_1, \ldots, t_n)$ with the subformula $B(t_1, \ldots, t_n)$, using an alphabetic variant of $B$ if necessary to avoid clashes with bound variables. Certainly, $Def_Q \vDash Q(\vec{t}) \leftrightarrow B(\vec{t})$. Therefore, by Theorem III.50, $Def_Q \vDash A \leftrightarrow A^*$, and thus $T' \vDash A \leftrightarrow A^*$. □

Similar constructions work for introducing defined function symbols and predicate symbols. The added complication is that the theory $T$ needs to prove the uniqueness and totality conditions for a function or a constant before that function or constant can be introduced as a defined function symbol. The totality condition is crucial; the uniqueness condition can be omitted but then a less faithful extension is obtained.

We'll show how this works for functions. After that, the definability of constants can be viewed as the same as introducing a 0-ary function.

**Definition III.116.** Let $T$ and $L$ be as above. Let $B(x_1, \ldots, x_n, x_{n+1})$ be an $L$-formula with no free variables other than $x_1, \ldots, x_{n+1}$.

(a) The formula $B$ *defines* a function $f$ in $T$ if

$$T \vDash \forall x_1 \cdots \forall x_n \, \exists! x_{n+1} \, B(x_1, \ldots, x_n, x_{n+1}). \tag{III.37}$$

(b) The formula $B$ *ambiguously defines* a function $f$ in $T$ if

$$T \vDash \forall x_1 \cdots \forall x_n \, \exists x_{n+1} \, B(x_1, \ldots, x_n, x_{n+1}). \tag{III.38}$$

In either case, the *defining axiom* for $f$ is the sentence $Def_f$:

$$\forall x_1 \cdots \forall x_n \, \forall x_{n+1} \, [f(x_1, \ldots, x_n) = x_{n+1} \rightarrow B(x_1, \ldots, x_n, x_{n+1})].$$

When $B$ (unambiguously) defines $f$, the uniqueness condition in Equation (III.37) means that the defining axiom $Def_f$ is equivalent (provably in $T$) to

$$\forall x_1 \cdots \forall x_n \, \forall x_{n+1} \, [f(x_1, \ldots, x_n) = x_{n+1} \leftrightarrow B(x_1, \ldots, x_n, x_{n+1})].$$

The next theorem states that adding an (ambiguously) defined function symbol does not add any new $L$-consequences.

**Theorem III.117.** *Let the $n$-ary function $f$ be definable or ambiguously definable by $B(x_1, \ldots, x_{n+1})$ in $T$. Let $L' = L \cup \{f\}$ and $T'$ be the theory with axioms $T + Def_f$. Then $T'$ is a conservative extension of $T$.*

*Proof.* The proof is similar to the proof of Theorem III.115(a). Suppose $T \nvDash A$ where $A$ is an $L$-sentence. Let $\mathfrak{A}$ be an $L$-model of $T \cup \{\neg A\}$. The sentence (III.38) is true in $\mathfrak{A}$. Therefore for every $a_1, \ldots, a_n \in |\mathfrak{A}|$, there is at least one $b \in |\mathfrak{A}|$ such that $\mathfrak{A} \vDash B(a_1, \ldots, a_n, b)$; pick one and denote it $b_{a_1,\ldots,a_n}$. (This uses the Axiom of Choice if $f$ is only ambiguously defined.) Then expand $\mathfrak{A}$ to a $L'$-structure $\mathfrak{B}$ by letting the interpretation of $f$ satisfy

$$f^{\mathfrak{B}}(a_1, \ldots, a_n) \;=\; b_{a_1,\ldots,a_n}.$$

Since $\mathfrak{B}$ is an expansion of $\mathfrak{A}$, the structures $\mathfrak{A}$ and $\mathfrak{B}$ satisfy the same $L$-sentences. Thus $\mathfrak{B} \vDash T \cup \{\neg A\}$. Finally, by choice of $f^{\mathfrak{B}}$, we have $\mathfrak{B} \vDash Def_f$. Therefore $T, Def_f \nvDash A$. □

The next theorem is for defined function symbols, not ambiguously defined function symbols. It states that formulas using the defined function symbol can be rewritten as provably equivalent formulas that do not use the defined function symbol.

**Theorem III.118.** *Let the $n$-ary function $f$ be definable by $B(x_1, \ldots, x_n, x_{n+1})$ in $T$. Let $L' = L \cup \{f\}$ and $T'$ be the theory with axioms $T + Def_f$. Then, for every $L'$-formula $A$, there is an $L$-formula $A^*$ such that $T' \vDash A \leftrightarrow A^*$.*

This is very similar to Theorem III.115(b), but its proof is more complicated. The problem is that we have to indirectly replace uses of $f$ and that terms may include multiple, nested occurrences of $f$.

**Example III.119.** To understand the idea behind the proof of the theorem, consider the example of a binary function $f$ which is defined in $T$ by a formula $B(x_1, x_2, x_3)$. Suppose $A$ is the formula $\forall x \, Q(a + f(x, f(x, a))$ where $Q$ is a predicate symbol, $a$ is a constant symbol, and $+$ of course is a binary function symbol. To form $A^*$ such that $T \vDash A \leftrightarrow A^*$, we let $A^*$ be

$$\forall x \, \exists y \, \exists z \, [B(x, a, y) \wedge B(x, y, z) \wedge Q(a + z)].$$

Let $Uniq_f$ be the sentence from (III.37) expressing unique existence. Then $\{Uniq_f, Def_f\}$ logically implies the sentences

$$\forall x \, [Q(a + f(x, f(x, a))) \leftrightarrow \exists y \, (B(x, a, y) \wedge Q(a + f(x, y)))].$$

and

$$\forall x\, \forall y\, [Q(a + f(x, y)) \leftrightarrow \exists z\, (B(x, y, z) \wedge Q(a + z))].$$

Putting these together, a little work shows that $\mathit{Uniq}_f, \mathit{Def}_f \vDash A \leftrightarrow A^*$. Hence $T \vDash A \leftrightarrow A^*$.

*Proof of Theorem III.118.* Suppose $A$ has $k$ occurrences of the symbol $f$. We will construct a sentence $A'$ with only $k{-}1$ occurrences of $f$ such that $\mathit{Uniq}_f, \mathit{Def}_f \vDash A \leftrightarrow A'$. Iterating this $k$ times yields the desired $A^*$. Choose an occurrence of a term $f(t_1, \ldots, t_k)$ in $A$ such that $f$ does not appear in any of its arguments $t_i$. For shorter notation, we write $\vec{t}$ instead of $t_1, \ldots, t_k$. Let $C(f(\vec{t}))$ denote the atomic subformula of $A$ where $f(\vec{t})$ occurs. In this notation, $C = C(y)$ is an atomic formula in which there is a single occurrence of $y$. Let $C'$ be the formula

$$\exists z\, (B(\vec{t}, z) \wedge C(z))$$

where $z$ is a variable that does not appear in $C(f(\vec{t}))$. Then $\mathit{Uniq}_f, \mathit{Def}_f \vDash C \leftrightarrow C'$. Form $A'$ by replacing the subformula $C(f(\vec{t}))$ of $A$ with $C'$. Then $\mathit{Uniq}_f, \mathit{Def}_f \vDash A \leftrightarrow A'$. □

## III.13 A Game-Theoretic Definition of Truth

The Tarskian definition of truth for first-order formulas, as given in Definition III.22, defined the meanings of "∀" and "∃" in terms of the English "for all" and "for some". This section presents another way of thinking about the meanings of first-order formulas in terms of game semantics. This gives a more dynamic and more intuitive way of visualizing truth, namely in terms of the existence of winning strategies for games played by two players.

We first describe how the game semantics works to characterize the truth of prenex formulas. The game is played by two players, somewhat whimsically named "Alice" and "Eve". Alice and Eve are presented with a $L$-structure $\mathfrak{A}$ and a $L$-formula $B(y_1, \ldots, y_\ell)$ and members $a_1, \ldots, a_\ell \in \mathfrak{A}$. There are no other variables appearing free in $B$ other than $y_1, \ldots, y_\ell$. Eve is attempting to prove that $B(a_1, \ldots, a_\ell)$ is true in $\mathfrak{A}$; conversely, Alice is attempting to prove that it is false in $\mathfrak{A}$. Eve will act by giving values for existentially quantified variables; Alice will act by giving values for universally quantified variables. (The names "Alice" and "Eve" are chosen to have initials "A" and "E" for "∀" and "∃".) The game halts when presented with a quantifier-free formula, at which point Eve wins if $\mathfrak{A} \vDash B(\vec{a})$ and Alice wins if $\mathfrak{A} \nvDash B(\vec{a})$.

The structure $\mathfrak{A}$ stays the same throughout the game. The formula $B(y_1, \ldots, y_\ell)$ and the objects $a_1, \ldots, a_\ell$ change after each step. For each move in the game, there are three possible situations;

∃-*move:* Suppose $B(a_1, \ldots, a_\ell)$ begins with an existential quantifier, and thus has the form $\exists y_{\ell+1}\, C(a_1, \ldots, a_\ell, y_{\ell+1})$. Then it is Eve's turn to move and, as her move, she selects an object $a_{\ell+1}$. The next round of the game considers $C(a_1, \ldots, a_\ell, a_{\ell+1})$ in place of $B(a_1, \ldots, a_\ell)$.

$\forall$*-move:* Suppose $B(a_1, \ldots, a_\ell)$ begins with a universal quantifier and thus has the form $\forall y_{\ell+1}\, C(a_1, \ldots, a_\ell, y_{\ell+1})$. Then it is Alice's turn to move and, as her move, she selects an object $a_{\ell+1}$. The next round of the game considers $C(a_1, \ldots, a_\ell, a_{\ell+1})$.

*Quantifier-free:* If $B(a_1, \ldots, a_\ell)$ is quantifier-free, the game ends. Eve wins the game if $B(a_1, \ldots, a_\ell)$ is true in $\mathfrak{A}$. Otherwise, Alice wins.

**Example III.120.** This and the next example are based on the structure shown in Figure III.3. The language consists of a binary relation $E$ and equality ($=$). The universe is equal to $\{0, 1, 2, 3, 4, 5, 6, 7\}$. We consider two ways to write formulas $\mathrm{Dist}_4(x, y)$ that state there is a path of length exactly 4 between objects $a$ and $b$.

(a) First let $\mathrm{Dist}(x, y)$ be the formula

$$\exists z_1\, \exists z_2\, \exists z_3\, [E(x, z_1) \wedge E(z_1, z_2) \wedge E(z_2, z_3) \wedge E(z_3, y)].$$

With $x$ and $y$ equal to 0 and 4, Alice and Eve play the game on $\mathrm{Dist}(0, 4)$. This formula is true in the structure $\mathfrak{A}$. Since the quantifiers are all existential, Eve makes all the moves. Her winning strategy is to set $a_1 = 1$, $a_2 = 2$, and $a_3 = 3$. Then, since

$$E(0, 1) \wedge E(1, 2) \wedge E(2, 3) \wedge E(3, 4)$$

is true, Eve wins the game. On the other hand, Eve does not have a winning strategies for the games on $\mathrm{Dist}(0, 3)$ and $\mathrm{Dist}0, 5)$, reflecting the fact that these are false.
The formula $\mathrm{Dist}_4$ is a straightforward way to express that there is a path of length 4. It has the disadvantage, however, that when generalizing this to form a formula $\mathrm{Dist}_\ell$ about having a path of length $\ell > 4$, a total of $\ell - 1$ quantifiers.

(b) Now consider another formula that expresses the same property as $\mathrm{Dist}_4(x, y)$:

$$\exists z_2\, \forall u\, \exists z'\, [(u = x \to E(x, z') \wedge E(z', z_2)) \wedge (u \neq x \to E(z_2, z') \wedge E(z', y))].$$

When there is a length 4 path from $x$ to $y$, Eve's winning strategy is to start by picking $z_2$ to be the midpoint of such a path. Alice can then either choose $u$ to equal $x$ to challenge the fact there is path from $x$ to $z_2$ of length 2 or choose $u$ to equal $z_2$ to challenge the fact there is a path from $z_2$ to $y$ of length 2. Eve replies by setting $z'$ to be either the midpoint of the first length 2 subpath (if Alice selected $x$) or the midpoint of the second length 2 subpage (if Alice selected $z_2$.[17]

[17] Our analysis omitted considering the case where $z_2$ is equal to $x$, but the formula and the corresponding game still work in that case too.

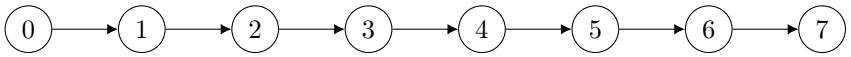



Figure III.3: The structure $\mathfrak{A}$ for Examples III.120 and III.121 is a directed graph on eight vertices. The directed edges from $i$ to $i{+}1$ indicate that $E(i, i{+}1)$ is true.

**Example III.121.** (a) Continuing the previous example, here is a formula $\mathrm{Dist}_8(x, y)$ that expresses there is path of length exactly 8 from $x$ to $y$:

$$\begin{aligned}\exists z_4\, \forall u\, \exists z'\, \forall u'\, \exists z''\, [&(u = x \wedge u' = x \to E(x, z'') \wedge E(z'', z'))\wedge\\ &(u = x \wedge u' \neq x \to E(z', z'') \wedge E(z'', z_4)) \wedge\\ &(u \neq x \wedge u' = x \to E(z_4, z'') \wedge E(z'', z')) \wedge\\ &(u \neq x \wedge u' \neq x \to E(z', z'') \wedge E(z'', y))\ ].\end{aligned}$$

The idea is that Eve chooses $z_4$ to be the midpoint of a length 8 path from $x$ to $y$. Then Alice either chooses $u$ to be $x$ to indicate there is no length 4 path from $x$ to $z_4$ or chooses $u \neq x$ to indicate there is no length 4 path from $z_4$ to $y$. Then Eve picks the midpoint $z'$ of the length 4 path that was challenged by Alice, either from $x$ to $z_4$ or from $z_4$ to $y$. Next, Alice picks $u'$ to be equal to $x$ to challenge the existence of the first subpath of length 2, or to be unequal to $z$ to challenge the existence of the second claimed subpage of length 2. Finally, Eves chooses the middle vertex of that subpath of length 2.

This example generalizes to formulas for $\mathrm{Dist}_\ell$ that require only $O(\log \ell)$ quantifiers. However, these still require $O(\ell \log \ell)$ size formulas overall.

(b) Here is yet another construction that gives formulas equivalent to $\mathrm{Dist}_\ell$ of size only $O(\ell)$. For the $\ell = 8$, the example is

$$\begin{aligned}&\exists z_4\, \forall u\, \exists x'\, \exists z'\, \exists y' \forall u'\, \exists x''\, \exists z''\, \exists y''\\ &\quad(\ [(u = x \to x' = x \wedge y' = z_4) \wedge (u \neq x \to x' = z_4 \wedge y' = y)]\\ &\quad\ \wedge [(u = x' \to x'' = x' \wedge y' = z') \to (u \neq x' \to x'' = z' \wedge y'' = y')]\\ &\quad\ \wedge E(x'', z'') \wedge E(z'', y'')\ ).\end{aligned}$$

The idea for this construction is that Eve's winning strategy (if any) is based on choosing values so that there is a length 4 path from $x'$ to $y'$ with midpoint $z'$ and there is a length 2 path from $x''$ to $y''$ with midpoint $z''$.

Game semantics are frequently as a tool to show that certain concepts are not expressible by first-order formulas, especially in the study of Finite Model Theory. Many of these constructions are based on "Ehrenfeucht-Fraïssé" games and its variants.

## Exercises

**Exercise III.1.** Using the predicates and function symbols from Section III.1, express the following statements in first-order logic.

(a) "*John* has not read any books."
(b) "*John*'s mother has not read any books."
(c) "Every book has been read by at least one person."
(d) "There is a book that no one has read."
(e) "No one likes every book they have read."
(f) "*John* likes every book that his mother likes."

**Exercise III.2.** Continuing to use the predicates and function symbols from Section III.1, express the following statements in first-order logic.

(a) "Everyone has a read a book that no one else has."
(b) "Someone has read a book that no one else has."
(c) "*John* has read exactly one book."
(d) "*John* has read exactly two books."
(e) "*John* has read every book that has ever been read (by anybody)." Equivalently, "If *John* hasn't read a book, then no one has read it."

**Exercise III.3.** Express the following as first-order formulas. Use the first-order language with unary predicates $Jazz(x)$ and $Kpop(x)$, the binary predicates $Likes(x, y)$ and $Knows(x, y)$, the constant symbols *Joan* and *John*, and the equality sign $=$. $Jazz(x)$ means "$x$ is a jazz musician", $Kpop(x)$ means "$x$ is a K-pop musician", $Likes(x, y)$ means "$x$ likes $y$", and $Knows(x, y)$ means "$x$ knows $y$". Variables range over the universe of people.

(a) Jazz musicians do not like K-pop musicians.
(b) *Joan* knows a jazz musician who likes every K-pop musician.
(c) *John* likes everyone he knows.
(d) There is no one whom *John* and *Joan* both like.
(e) One of *John* and *Joan* is a jazz musician, and the other is a K-pop musician.
(f) Everyone that *John* likes knows a jazz musician.
(g) Every K-pop musician knows a jazz musician who does not like any K-pop musicians.

**Exercise III.4.** A quote that is sometimes (mis?)attributed to Abraham Lincoln is: "You can fool some of the people all of the time. You can fool all of the people some of the time. But you cannot fool all of the people all of the time." Let these three assertions be:

(a) You can fool some of the people all of the time.
(b) You can fool all of the people some of the time.
(c) You cannot fool all of the people all of the time.

Express these three sentences in first-order logic. Let variables range over the universe that includes all people and all times. Let $Person(x)$ mean $x$ is a person, and $Time(x)$ mean $x$ is a time. Let $F(x, t)$ mean you can fool person $x$ at time $t$.

Sentences (a) and (b) each have two possible translations to first-order logic that are not logically equivalent. Give both translations for these two. Explain

how the two translations differ in meaning. Also, give a translation of (c) to first-order logic. (This exercise is taken from the textbooks of Enderton [5] and Mendelsohn [13].)

**Exercise III.5.** Let $P$ be a unary predicate, $Q$ a binary predicate, $f$ a unary function, $g$ a binary function, and $c$ a constant symbol. Answer questions (i)-(iii) about the following expressions:

(a) $c$
(b) $x_3$
(c) $g(c, x_3)$
(d) $g(c, x_3) =$
(e) $c = g(f(x_2), x_3)$
(f) $g(f(x_1), c) = g(f(x_1))$
(g) $P(c = g(f(x_2), x_3))$
(h) $f(c) = \neg g(x_1, x_1)$
(i) $Q(c, x_1) = P(c)$
(j) $\forall x_1 \, (P(x_1) \to Q(x_1, x_1))$
(k) $\forall x_1 \in P(x_1) \, Q(x_1, x_1)$
(l) $\forall x_1 \, (Q(x_1, x_1) \wedge x_1 = c)$

(i) Which of these are syntactically correct terms?
(ii) Which of these are syntactically correct atomic formulas?
(iii) Which of these are syntactically correct formulas? (For the purposes of this exercise, a formula still counts as syntactically correct if some parentheses are omitted or extra parentheses are added.)

**Exercise III.6.** Express the following about least common multiples and greatest common divisors in $L_{\mathrm{PA}}$, over the nonnegative integers.

(a) If $x$ and $y$ are non-zero, there is a least non-zero value $z$ which is a multiple of both $x$ and $y$.
(b) If $x$ and $y$ are not both zero, there is a greatest common divisor $z$ of $x$ and $y$ and this common divisor is $\geq 1$.

**Exercise III.7.** Let $c$ be a constant symbol and $f$ be a unary function symbol. Give an example of a structure $\mathfrak{A}$ which satisfies

$$\Gamma \;=\; \{\forall x \, \forall y \, (x \neq y \to f(x) \neq f(y)), \; \forall x \, f(x) \neq c\}.$$

In this and subsequent exercises, describe the universe of structure and the interpretations of the non-logical symbols explicitly using set notation.

**Exercise III.8.** Let $E$ be a binary predicate symbol. Give a structure $\mathfrak{B}$ such that $\mathfrak{B} \vDash \forall x \, \forall y \, (E(x, y) \to \neg E(y, x))$.

**Exercise III.9.** Let $f$ be a unary function symbol. Give a structure $\mathfrak{C}$ such that $\mathfrak{C} \vDash \forall x \, (x \neq f(x) \wedge x \neq f(f(x)))$.

**Exercise III.10.** Let $\Gamma$ be the set of formulas $\{x_1 \neq x_2, P(x_1) \to \forall x_3 \, P(x_3)\}$. Give an example of a structure $\mathfrak{A}$ and an object assignment $\sigma$ that illustrates that $\Gamma \nvDash \neg P(x_1)$.

**Exercise III.11.** Work in the language with a unary predicate $P(x)$ and (as usual) the equality sign $=$. Express the following as first-order sentences.

(a) There is a unique $x$ satisfying $P(x)$.
(b) There are at least two objects $x$ that satisfy $P(x)$.
(c) There are at least three objects $x$ that satisfy $P(x)$.
(d) There are exactly two objects $x$ that satisfy $P(x)$.

**Exercise III.12.** Work in the same language as the previous exercise. For $k \geq 0$, let $AtMost_k$, $AtLeast_k$ and $Exactly_k$ be the assertions that there are at most $k$, at least $k$, and exactly $k$ (respectively) many objects $x$ satisfying property $P(x)$.

(a) Give general constructions, for arbitrary fixed $k$, of first-order sentences that express the assertions $AtMost_k$, $AtLeast_k$ and $Exactly_k$.

(b) Analyze the sizes of the sentences you constructed for $AtMost_k$, $AtLeast_k$ and $Exactly_k$. Do they use $O(k^2)$ many symbols? Do they use $O(k)$ many symbols?[18]

(The most straightforward constructions use $O(k^2)$ many symbols, but it is not too hard to express them using only $O(k)$ many symbols. To construct formulas that require only $O(k)$ many symbols, the suggestion is to start with $AtMost_k$.)

**Exercise III.13.** Work in the language $L$ with the symbol = and no non-logical symbols. $\Gamma$ said to have a model of cardinality $n$ if there is a structure $\mathfrak{A}$ so that $\mathfrak{A} \models \Gamma$ and its universe $|\mathfrak{A}|$ has size $n$. Give an example of a set $\Gamma$ of $L$-sentences which has models of cardinality $2n$ for all integers $n > 0$, but does not have any model of cardinality $2n+1$ for any integer $n \geq 0$. Call this the "even cardinality finite models" property.

**Exercise III.14.** Give a language $L'$ and a finite set $\Pi$ of $L'$-sentences so that the "even cardinality finite models" property of the previous exercise holds for $\Pi$. Explain why your example works. (In contrast to the previous exercise, you can choose a language $L'$ but you must use a *finite* set $\Pi$.)

**Exercise III.15.** Let $A$ and $B$ be formulas.

(a) Prove that $\exists x\,(A \vee B) \models\!\dashv \exists x\,A \vee \exists x\,B$.

(b) Prove that $\forall x\,(A \wedge B) \models\!\dashv \forall x\,A \wedge \forall x\,B$.

[Hint: These are fairly easy to prove using the definition of truth. Part (b) can be proved as a consequence of part (a), or vice-versa; by expressing $\wedge$ in terms of $\vee$ and $\forall$ in terms in of $\exists$. Part (a) was used in the proof of Lemma III.80.]

**Exercise III.16.** Suppose $x$ does not have a free occurrence in the formula $A$. Prove that $A \models\!\dashv \exists x\,A$ and $A \models\!\dashv \forall x\,A$. (The first part, $A \models\!\dashv \exists x\,A$, was used in the proof of Lemma III.80.)

**Exercise III.17.** Prove the following by using the definition of truth.

(a) $\models \forall x\,(A \to B) \to \exists x\,A \to \exists x\,B$.

(b) $\models \forall x\,(A \to B) \to \forall x\,A \to \forall x\,B$.

**Exercise III.18.** Prove (a)-(d). $A$ and $B$ are formulas; $P$ and $Q$ are unary predicate symbols. Prove (a) using the definition of truth. Prove (b)-(d) by giving a structure that illustrates the non-implication. (Part (a) was used in

[18] $O(k)$ and $O(k^2)$ are "big-Oh notation". $O(k)$ means having size bounded by $c \cdot k$ for some constant $c$. $O(k^2)$ means having size bounded by $c \cdot k^2$ for some constant $c$.

the proof of Theorem III.61.)

(a) $\forall x\,(A \leftrightarrow B) \vDash \forall x\,A \leftrightarrow \forall x\,B$.
(b) $\forall x\,P(x) \leftrightarrow \forall x\,Q(x) \nvDash \forall x\,(P(x) \leftrightarrow Q(x))$.
(c) $\exists x\,(P(x) \leftrightarrow Q(x)) \nvDash \exists x\,P(x) \leftrightarrow \exists x\,Q(x)$.
(d) $\exists x\,P(x) \leftrightarrow \exists x\,Q(x) \nvDash \exists x\,(P(x) \leftrightarrow Q(x))$.

**Exercise III.19.** Suppose $x, y, z, u, v$ are distinct variables. Let $A$ be the formula

$$\exists x\,[x \le y \wedge \forall y\,(P(y,z) \to \exists z(y \le z))].$$

(a) Write out the formula $A$ and label which occurrences of variables are free occurrences and which are bound occurrences.
(b) What is $A(g(0,u)/z)$?
(c) What is $A(f(v), g(0,u)/y, z)$?
(d) What is $A(g(w,0), f(v), g(0,u)/x, y, z)$?
(e) Give an alphabetic variant $B$ of $A$ so that $h(x,y,z)$ is substitutable for $y$ in $B$. Rename as few bound variables as possible.
(f) Give an alphabetic variant $C$ of $A$ so that $h(x,y,z)$ is substitutable for $z$ in $C$. Rename as few bound variables as possible.

**Exercise III.20.** Carry out the following substitutions by giving the term or formula that results from the substitution.

(a) Let $t$ be the term $f(g(g(0,z), y+z))$. What is $t(0/z)$?
(b) For the same term $t$, what is $t(f(0), 0/x, z)$?
(c) Let $A$ be $\forall x\,[y = x \to \exists y\,(P(x,y) \to x = z \vee y = z)]$. What is $A(g(0,u)/z)$?
(d) For the same $A$, what is $A(f(v), g(0,u)/y, z)$?
(e) For the same $A$, what is $A(g(w,0), f(v), g(0,u)/x, y, z)$?

**Exercise III.21.** Let $A$ be the same formula as in the previous exercise. Give an example of an alphabetic variant $B$ of $A$ such that $x + y + z$ is substitutable for $z$ in $B$.

**Exercise III.22.** For each of the following statements, state whether it is true for all formulas $A$ or not. If not true, give a counterexample.

(a) The term 0 is substitutable for $x_1$ in $A$. (0 is a constant symbol.)
(b) The term $x_1$ is substitutable for $x_1$ in $A$.
(c) The term $x_2$ is substitutable for $x_1$ in $A$.
(d) If the term $x_2$ is substitutable for $x_1$ in $A$, then the term $x_1$ is substitutable for $x_2$ in $A(x_2/x_1)$.

**Exercise III.23.** For each of the following statements, state whether it is true for all formulas $A$, terms $t$, and variables $x$. If not true, give a counterexample.

(a) If $t$ is a closed term, then $t$ is substitutable for $x$ in $A$.
(b) The term $f(x,x)$ is substitutable for $x$ in $A$.
(c) The formula $x = x$ is substitutable for $x$ in $A$.
(d) If $t$ is substitutable for $x$ in $A$, then $A(t/x)$ is a formula.
(e) If $t$ is not substitutable for $x$ in $A$, then $A(t/x)$ is a formula.
(f) If $A(t/x)$ is a formula, then $t$ is substitutable for $x$ in $A$.

**Exercise III.24.** Show that the principle of universal instantiation needs the assumption that $t$ is substitutable for $x$ in $A$. To do this, give an example of a formula $A$ and a term $t$ such that $\forall x\, A \to A(t/x)$ is *not* logically valid.

**Exercise III.25.** For each of the following, give a logically equivalent prenex formula.

(a) $\exists x\, P(x) \to \exists x\, Q(x,x)$.
(b) $\exists x\, P(x) \leftrightarrow \exists x\, Q(x,x)$.
(c) $\forall x\, \exists y\, P(x,y) \to \exists x\, \forall y\, P(x,y)$.
(d) $\neg\forall y\, [\exists u\, (g(u) \le y) \to \forall x\, (y < x \to \exists y\, (f(x) \le y))]$.

**Exercise III.26.** Give an example of formulas $A$ and $B$ such that

$$\exists x A \wedge B \not\models \exists x\, (A \wedge B). \tag{III.39}$$

Justify your answer by giving a structure $\mathfrak{A}$ and an object assignment $\sigma$ showing that (III.39) holds.

**Exercise III.27.** Show the following.

(a) The class of infinite undirected graphs is an elementary class in the wide sense ($\mathrm{EC}_\Delta$).
(b) The class of undirected graphs that do not contain a cycle is an elementary class in the wide sense ($\mathrm{EC}_\Delta$).

**Exercise III.28.** Prove the following statements.

(a) The intersection of two elementary classes is an elementary class.
(b) The union of two elementary classes is an elementary class.
(c) The intersection of two $\mathrm{EC}_\Delta$ classes is $\mathrm{EC}_\Delta$.
(d) The intersection of an arbitrary collection of $\mathrm{EC}_\Delta$ classes is $\mathrm{EC}_\Delta$.
(e) The union of two $\mathrm{EC}_\Delta$ classes is $\mathrm{EC}_\Delta$.

**Exercise III.29.** (For readers with some knowledge of field theory. Compare with Exercise IV.21.) Show that, for a fixed prime $p$, the class of fields of characteristic $p$ is an elementary class. Show that the class of fields of characteristic zero is $\mathrm{EC}_\Delta$.

**Exercise III.30.** The *truncated subtraction* function $\dot{-}$ is defined on the nonnegative integers by $x \dot{-} y = \max\{x - y, 0\}$. Prove that $\dot{-}$ is definable in $\mathcal{N}$. Also, prove that the *excess over a square* function, $x \dot{-} \lfloor\sqrt{x}\rfloor$ is definable in $\mathcal{N}$ by giving a formula over the language $0, S, +, \cdot$ that defines it.

**Exercise III.31.** Work in the language $L = \{S, \cdot\}$ where $S$ is the successor function and $\cdot$ is integer multiplication. Let the structure $\mathcal{N}_{S,\cdot} = (\mathbb{N}, S, \cdot)$ be the nonnegative integers with the usual successor and multiplication functions. Show that the function $+$ is definable in $\mathcal{N}^{S,\cdot}$. [Hint: First show that, for $c > 0$, $a + b = c$ is true if and only if $(xz+1)(yz+1) = z^2(xy+1)+1$.] This result is due to Julia Robinson [1949].

**Exercise III.32.** Let $\mathcal{Q} = (\mathbb{Q}, 0, +, \cdot)$ be the structure of the rationals.
(a) Show that the object 1 is definable in $\mathcal{Q}$.
(b) Show that the function $x \mapsto -x$ is definable in $\mathcal{Q}$.
(c) Let $f(x, y) = x/y$ if $y \neq 0$ and let $f(x, 0) = 0$. Show that $f$ is definable in $\mathcal{Q}$.
(d) Show that the set of positive rational numbers is definable in $\mathcal{Q}$. [Hint: You may use the Lagrange four square theorem, which states that any positive integer is the sum of four integer squares.]

**Exercise III.33.** Show that the following two formulas are logical consequences of the theory of arrays.
(a) $\forall a\, \forall i\, [Array(a) \wedge Index(i) \to a = Write(a, i, Read(a, i))]$.
(b) $\forall a\, \forall i\, \forall j\, \forall x\, \forall y\, [Array(a) \wedge Index(i) \wedge Index(j) \wedge i \neq j \wedge Value(x) \wedge Value(y) \to Write(\, Write(a, i, x), j, y) = Write(\, Write(a, j, y), i, x)]$.
Both of these need to use the axiom of extensionality.

**Exercise III.34.** Example III.119 constructed a formula $A^*$ such that $\{Uniq_f, Def_f\} \vDash A \leftrightarrow A^*$. Here is an alternate construction. Let $A^{**}$ be the sentence

$$\forall x\, \forall y\, \forall z\, [B(x, a, y) \wedge B(x, y, z) \to Q(a + z)].$$

Prove that $\{Uniq_f, Def_f\} \vDash A \leftrightarrow A^{**}$.

**Exercise III.35.** Work with a fixed language $L$. Let $\mathcal{S}, \mathcal{S}_1, \mathcal{S}_2$ be classes of structures. Let $T, T_1, T_2$ be theories. (Recall that a theory is a set of sentences closed under logical consequence.)
(a) Prove that if $T_1 \subseteq T_2$ then $\operatorname{Mod} T_2 \subseteq \operatorname{Mod} T_1$.
(b) Prove that if $\mathcal{S}_1 \subseteq \mathcal{S}_2$ then $\operatorname{Th} \mathcal{S}_2 \subseteq \operatorname{Th} \mathcal{S}_1$.
(c) Prove that $T = \operatorname{Th} \operatorname{Mod} T$.
(d) Prove that $\operatorname{Th} \mathcal{S} = \operatorname{Th} \operatorname{Mod} \operatorname{Th} \mathcal{S}$.
(e) Give an example of $\mathcal{S}$ such that $S \neq \operatorname{Mod} \operatorname{Th} \mathcal{S}$. Explain why your example works.

**Exercise III.36★** A purely existential formula is a prenex formula in which all the quantifiers are existential. Let $P$ be a unary predicate symbol. Prove that $\forall x\, P(x)$ is not logically equivalent to any purely existential formula.

**Exercise III.37★** Let $A$ be the formula $Q(c) \leftrightarrow \forall x P(x)$, where $P$ and $Q$ are binary predicate symbols and $c$ is a constant symbol. Prove
(a) There is no purely universal prenex formula $B$ such that $B \vDash\!\dashv A$.
(b) There is no purely existential prenex formula $B$ such that $B \vDash\!\dashv A$.
When forming prenex formulas, the first step was to eliminate any use of the $\leftrightarrow$ symbol by replacing subformulas $A \leftrightarrow B$ with $(A \to B) \wedge (B \to A)$. The point of this exercise is that it shows that something like this must be done. In other words, it proves that there are no prenex equivalences for $\leftrightarrow$ that are as simple in form as the logical equivalences of Lemma III.80.

**Exercise III.38.** Examples III.120 and III.121 showed how to give first-order formulas expressing that there is a path of length exactly $\ell$ for $\ell = 4$ and $\ell = 8$.

Rework these examples to instead give formulas expressing that there is a path from $x$ to $y$ of length *at most* $\ell$, for $\ell = 4$ and $\ell = 8$.

**Exercise III.39.**

(a) Generalize the two constructions of Example III.121 arbitrary powers of two $\ell$.

(b) Generalize the two constructions of Example III.121 arbitrary values of $\ell$ (not necessarily a power of two).

**Exercise III.40**★ The two constructions of Example III.121 gave formulas for $\text{Dist}_8$ that used the equality symbol (=). Rework these examples to not use the equality symbol. [Hint: This is not hard, but it is a little tricky.]

# Chapter IV

# First-Order Logic: Proofs

## IV.1 Introduction to First-Order Proofs

This chapter introduces a Hilbert-style proof system for first-order logic called FO. It will be built on the proof system PL for propositional logic but augmented with axioms and rules of inference for equality and for quantifiers. FO-proofs proceed in a step-by-step fashion starting with axioms or other hypotheses and using Modus Ponens and Generalization as inference rules. The Modus Ponens rule is identical to the Modus Ponens rule of propositional logic. The Generalization rule will allow $\forall x\, A(x)$ to be inferred from $A(x)$. This reflects the fact that if $A(x)$ is valid, so is $\forall x\, A(x)$. (The actual generalization rule (Gen), defined on page 135, is a bit more general than this.) The upshot is that FO-proofs can model (to a certain extent) how humans construct proofs.

The proof system FO enjoys both a Soundness Theorem and a Completeness Theorem. The Soundness Theorem states that FO proves only formulas that follow logically from a given set of hypotheses. The Completeness Theorem states that FO can prove *every* formula that follows from a given set of hypotheses. Having both Soundness and Completeness is a wonderful (and amazing!) state of affairs. It means that the rather simple formal system, FO, can fully capture all valid first-order reasoning.

There are a couple of serious catches or caveats though. First, Soundness and Completeness concern logical validity and logical implication in *arbitrary* structures not necessarily in particular structures. We are often interested instead about truth in particular structures. A notable example is the structure $\mathcal{N} = (\mathbb{N}, 0, S, +, \cdot)$, the so-called "standard model" of the integers (see Example III.13). Mathematicians are very interested in what properties, including first-order properties, are true in $\mathcal{N}$. Some non-trivial and deep assertions about the structure $\mathcal{N}$ can be made in first-order logic. This includes famous open problems such as the Riemann hypothesis and the P versus NP question. The proof system FO is not capable of proving all true statements about $\mathcal{N}$; instead, it can only prove all statements that follow from (say) the usual axioms

for $\mathcal{N}$. This is not just a limitation of the proof system FO with the usual axioms of arithmetic; indeed, there is no effectively axiomatizable proof system that exactly captures the true formulas of $\mathcal{N}$. Nor is there any algorithm for determining the truth of arbitrary first-order formulas in the structure $\mathcal{N}$.

These limitations, the non-existence of a complete proof system for the true formulas of $\mathcal{N}$ and the non-existence of an algorithm for determining the truth or falsity of formulas in $\mathcal{N}$, will be discussed—and formalized and proved—in later chapters. For now, let us point out of some the (rather amazing!) good properties of the proof system FO. (Compare this discussion to the properties of PL mentioned at the beginning of Chapter II.)

(1) ***Algorithmic***. FO-proofs are strings of symbols (also called "expressions") with specified syntactic properties. There is an algorithm, which given a string $w$ of symbols, determines whether $w$ is a valid FO-proof, and if so, what formula it proves, or what tautological implication(s) it proves.

(2) ***Soundness***. A first-order formula $A$ has an FO-proof only if it is a logically valid. Similarly, if $A$ has an FO-proof from the hypotheses $B_1, \ldots, B_k$ where $B_1, \ldots, B_k$ are sentences, then $B_1, \ldots, B_k \vDash A$.

(3) ***Completeness***. Conversely, any logically valid first-order formula $A$ has an FO-proof. More generally, if $B_1, \ldots, B_k \vDash A$ holds where the $B_i$'s are sentences, then there is an FO-proof of $A$ from the hypotheses $B_1, \ldots, B_k$.[1] Taken together, the soundness and completeness properties mean that $A$ has an FO-proof if and only if $\vDash A$ holds. The same holds as well for logical consequences of sentences.

(4) ***User-friendly***. This could also be called ***Human-centric***. There are several aspects to this. (i) FO can simulate human reasoning fairly efficiently. (ii) In particular, FO permits reasoning using step-by-step inferences. (iii) However, it is arguable, and the subject of present-day research investigations, whether FO-proofs (or proofs in any suitably strong proof system) can be reliably presented in a way the user-friendly for humans to read and understand.

(5) ***Elegance***. The system FO is mathematically elegant, without an excessively large number of axioms or rules.

As for effectiveness, there is no effective, universal algorithm for deciding whether a given formula is logically valid or, equivalently, has an FO proof. There are of course algorithms for *searching* for proofs; these can be designed to always succeed when an FO-proof exists. But these algorithms, in general, cannot determine reliably whether a proof actually exists. These topics will be taken up in detail starting in Chapter V and culminating in Chapter VII.

[1] Completeness is sometimes called "Adequacy", but we follow here the common convention and call it "completeness".

# IV.2 The Proof System FO

This section defines the proof system FO for first-order logic. This is a so-called "Hilbert-style" proof system.[2] An FO-proof is used to show that a formula $A$ is logically valid, or that $A$ is logically implied by a set $\Gamma$ of sentences. We write $\vdash A$ to denote $A$ has an FO-proof, in which case $A$ is logically valid. We write $\Gamma \vdash A$ to denote that $A$ has a proof from $\Gamma$. The Soundness and Completeness Theorems will imply that $A$ has an FO-proof exactly when $A$ is logically valid. In other words, $\vdash A$ holds exactly when $\vDash A$ holds.

Likewise, when $\Gamma$ is a set of *sentences*, then $\Gamma \vdash A$ is equivalent to $\Gamma \vDash A$ by the Soundness and Completeness Theorems. Thus $A$ is FO-provable from a set $\Gamma$ of sentences if and only if it is a logical consequence of $\Gamma$. However, if $\Gamma$ is a set of *formulas*, $\Gamma \vdash A$ and $\Gamma \vDash A$ are not equivalent. The reason for this is that FO-proofs allow the Generalization rule and can derive $\forall x\, A(x)$ from $A(x)$. In effect, this means we should think of $A$ and the formulas in $\Gamma$ as being implicitly universally quantified. This means that $\Gamma \vdash A$ is equivalent to the statement that the universal closure of $A$ is provable from the set of universal closures of formulas in $\Gamma$. For more on this, see the first part of Section IV.2.2 below, up through the "Side remark".

The proof system FO works exclusively with formulas that use only the logical connectives $\neg$ (negation), $\to$ (implication) and $\forall$ (universal quantification). This is analogous to the way that the propositional proof system PL worked with $\{\neg, \to\}$-formulas. Other logical connectives are *abbreviations* for formulas that use only the permitted connectives $\neg, \to, \forall$. Namely,

$$\begin{array}{rl} A \vee B & \text{is an abbreviation for } \neg A \to B \\ A \wedge B & \text{is an abbreviation for } \neg(A \to \neg B) \\ A \leftrightarrow B & \text{is an abbreviation for } (A \to B) \wedge (B \to A) \\ \exists x\, A & \text{is an abbreviation for } \neg \forall x \neg A \end{array}$$

Throughout this chapter, the terminology "formula" means a first-order formula that uses only the connectives $\neg$, $\to$ and $\forall$. As usual, we fix a first-order language $L$ of non-logical symbols; so "formula" means "$L$-formula".

## IV.2.1 The definition of the proof system FO

FO-proofs start with axioms or other hypotheses and infer new formulas with the inference rules Modus Ponens and Generalization. The following types of axioms are permitted in FO-proofs.

**Propositional axioms.** The propositional axioms are all formulas of the form

**PL1:** $A \to (B \to A)$

**PL2:** $[A \to (B \to C)] \to [(A \to B) \to (A \to C)]$

**PL3:** $\neg A \to (A \to B)$

[2] See the footnote on page 47.

**PL4:** $(\neg A \to A) \to A$

Here $A$, $B$ and $C$ may be any formulas.

**Equality axioms.** Nearly always, the language $L$ includes the symbol $=$ for equality; if so, special equality axioms are needed. The first equality axioms state that $=$ defines an equivalence relation. Letting $x, y, z$ be arbitrary variables, the first three equality axioms are

**EQ1:** $x = x$. (Reflexivity of $=$)

**EQ2:** $x = y \to y = x$. (Symmetry of $=$)

**EQ3:** $x = y \to y = z \to x = z$. (Transitivity of $=$)

For each $k$-ary function symbol $f$ in the language $L$, and letting $y_1, \ldots, y_k$ and $z_1, \ldots, z_k$ be arbitrary variables, there are axioms stating that $f$ respects the equality relation:

**EQ$_f$:** $y_1 = z_1 \to y_2 = z_2 \to \cdots \to y_k = z_k \to f(y_1, \ldots, y_k) = f(z_1, \ldots, z_k)$.

For each $k$-ary predicate symbol $P$ in the language $L$, and letting $y_1, \ldots, y_k$ and $z_1, \ldots, z_k$ be arbitrary variables, there are axioms stating that $P$ respects the equality relation:

**EQ$_P$:** $y_1 = z_1 \to y_2 = z_2 \to \cdots \to y_k = z_k \to P(y_1, \ldots, y_k) \to P(z_1, \ldots, z_k)$.

The axioms for equality are stated with variables that are implicitly universally quantified. For instance, as we shall see, the axioms $x = x$ and $x = y \to y = x$ can be used to derive $\forall x\,(x = x)$ and $\forall x\,\forall y\,(x = y \to y = x)$. So EQ1 and EQ2 really do state that equality is reflexive and symmetric. Similarly, EQ3 really does state that equality is transitive.

The axiom EQ$_P$ has the conclusion that $P(\vec{y}) \to P(\vec{z})$ instead of $P(\vec{y}) \leftrightarrow P(\vec{z})$. However, the symmetry of equality means that this implication holds in both directions under the hypotheses that $y_i = z_i$ for all $i$.

**Axioms of universal instantiation.** Suppose the term $t$ is substitutable for the variable $x$ in the formula $A$. The corresponding universal instantiation axiom is

**UI:** $\forall x\, A \to A(t/x)$.

In keeping with our convention on relaxed notations for substitution, the UI axiom can also be written as $\forall x\, A(x) \to A(t)$.

FO admits two rules of inference, Modus Ponens and Generalization.

**Modus Ponens inference rule.** For arbitrary formulas $A$ and $B$, the Modus Ponens rule is

**MP:** $\dfrac{A \qquad A \to B}{B}$

**Generalization inference rule.** For $A$ and $C$ formulas, and $x$ a variable such that $x$ does not appear free in $C$, the Generalization rule is

**Gen:** $\dfrac{C \to A}{C \to \forall x\, A}$

The variable $x$ is called the *generalization variable* or the *eigenvariable* for the inference.

**Example IV.1.** Examples of the equality axioms for functions and predicates include

$$x_1 = x_2 \to x_3 = x_4 \to f(x_1, x_3) = f(x_2, x_4)$$
$$x_1 = x_2 \to x_2 = x_3 \to f(x_1, x_2) = f(x_2, x_3)$$
$$x_1 = x_1 \to x_2 = x_3 \to P(x_1, x_2) \to P(x_1, x_3)$$

The last two illustrate that the variables $y_1, \ldots, y_k$ and $z_1, \ldots, z_k$ do not need to be distinct. The formal definition of formulas allows only the use of variables $x_1, x_2, x_3, \ldots$. However, we use variable names such as $x, y, z$ or $y_j$ or $z_j$ as informal designations of arbitrary variables $x_i$.

**Definition IV.2.** Let $\Gamma$ be a set of formulas and $A$ formula. An FO-*proof* of $A$ from the hypotheses $\Gamma$ is a sequence of formulas

$$B_1, B_2, B_3, \ldots, B_k$$

such that $B_k$ is $A$ and such that, for each $i = 1, \ldots, k$, (at least) one of the following conditions holds:

(a) $B_i$ is one of the above-listed FO-axioms: PL1, PL2, PL3, PL4, EQ1, EQ2, EQ3, $\mathrm{EQ}_f$, $\mathrm{EQ}_P$, or UI;

(b) $B_i$ is a member of $\Gamma$, namely $B_i$ is a hypothesis (HYP);

(c) $B_i$ is inferred by Modus Ponens (MP) from some $B_j$ and $B_k$ where $j, k < i$, so w.l.o.g., $B_k$ is $B_j \to B_i$; or

(d) $B_i$ is inferred by Generalization (Gen) from some $B_j$ with $j < i$, so $B_i$ has the form $C \to \forall x D$ and $B_j$ has the form $C \to D$ and $x$ does not appear free in $C$.

The final formula in the FO-proof is $A$ and is called the *conclusion* of the proof. We write $\Gamma \vdash A$ to denote that $A$ has a proof from $\Gamma$. In this case, $A$ is a *theorem* of $\Gamma$.

**Definition IV.3.** If $\Gamma = \varnothing$ (so there are no hypotheses, we write $\vdash A$ to denote that $\varnothing \vdash A$. In this case, we write $\vdash A$ and call $A$ a (first-order) *theorem*.

FO-proofs are sometimes called FO-*derivations*. Similarly to what was done with PL-derivations, we often abuse notation for sets of hypotheses and write things like $\Gamma, A \vdash B$ and $B_1, \ldots, B_k \vdash A$ instead of $\Gamma \cup \{A\} \vdash B$ and $\{B_1, \ldots, B_k\} \vdash A$.

**Example IV.4.** Let's show that $\forall x\, A \vdash A$ for $A$ an arbitrary formula. The complete FO-proof can be written out explicitly as the sequence of formulas

| | |
|---|---|
| $\forall x\, A \to A$ | Axiom UI, with $x$ as the term $t$ |
| $\forall x\, A$ | Hypothesis (since $\Gamma = \{\forall x\, A\}$) |
| $A$ | Modus Ponens |

**Example IV.5.** Now we show that $\forall x\, A$, $\forall x\, (A \to B) \vdash \forall x\, B$. Let $y$ be any variable that does not appear free in $B$. The FO-proof can be written out explicitly as:

| | |
|---|---|
| $\forall x\, (A \to B) \to (A \to B)$ | Axiom UI |
| $\forall x\, (A \to B)$ | Hypothesis |
| $A \to B$ | Modus Ponens |
| $\forall x\, A \to A$ | Axiom UI |
| $\forall x\, A$ | Hypothesis |
| $A$ | Modus Ponens |
| $B$ | Modus Ponens |
| $B \to y = y \to B$ | Axiom PL1 |
| $y = y \to B$ | Modus Ponens |
| $y = y \to \forall x\, B$ | Generalization (since $x$ is not free in $y = y$) |
| $y = y$ | Axiom EQ1 |
| $\forall x\, B$ | Modus Ponens |

The last five lines derive $\forall x\, B$ from $B$ with the aid of Generalization. The only purpose of the formula $y = y$ is to serve as the hypothesis $C$ for the Generalization rule.

**Example IV.6.** We show that $\vdash y = z \to f(x, y) = f(x, z)$. The FO proof is:

| | |
|---|---|
| $x = x \to y = z \to f(x, y) = f(x, z)$ | Axiom $\mathrm{EQ}_f$ |
| $x = x$ | Axiom EQ1 |
| $y = z \to f(x, y) = f(x, z)$ | Modus Ponens |

**Theorem IV.7.** *Let $A$ be a formula. Then*
(a) $\forall x\, A \vdash A$
(b) $A \vdash \forall x\, A$

*Proof.* Part (a) was already shown in Example IV.4. For part (b), let $B$ be any FO axiom such that $x$ is not free in $B$. For instance, we can always take $B$ to be $\forall x\, A \to \forall x\, A \to \forall x\, A$. Alternately, assuming that equality is in the language $L$, we can let $B$ be $y = y$ as was done in Example IV.5. Then an FO-proof of $\forall x\, A$ from $A$ is:

| | |
|---|---|
| $A$ | Hypothesis |
| $A \to B \to A$ | Axiom PL1 |
| $B \to A$ | Modus Ponens |
| $B \to \forall x\, A$ | Generalization, since $x$ is not free in $B$ |
| $B$ | By choice of $B$ as a axiom |
| $\forall x\, A$ | Modus Ponens $\square$ |

**Axioms and Inference Rules for FO-proofs**

**PL1:** $A \to B \to A$

**PL2:** $(A \to B \to C) \to (A \to B) \to (A \to C)$

**PL3:** $\neg A \to A \to B$

**PL4:** $(\neg A \to A) \to A$

**EQ1:** $x = x$

**EQ2:** $x = y \to y = x$

**EQ3:** $x = y \to y = z \to x = z$

**EQ$_f$:** $y_1 = z_1 \to \cdots \to y_k = z_k \to f(y_1, \ldots, y_k) = f(z_1, \ldots, z_k)$

**EQ$_P$:** $y_1 = z_1 \to \cdots \to y_k = z_k \to P(y_1, \ldots, y_k) \to P(z_1, \ldots, z_k)$

**Universal Instantiation (UI):** $\forall x\, A(x) \to A(t)$.

**Modus Ponens:** $\dfrac{A \qquad A \to B}{B}$.

**Generalization (Gen):** $\dfrac{C \to A}{C \to \forall x\, A}$ provided $x$ not free in $C$

$A \vee B$, $A \wedge B$, and $\exists x\, A$ abbreviate $\neg A \to B$, $\neg(A \to \neg B)$ and $\neg \forall x \neg A$.

Part (b) of the theorem means that a simplified form of the Generalization rule can be used as a derived (admissible) rule of inference:

$$\frac{A}{\forall x\, A} \text{ Generalization (simplified version)} \tag{IV.1}$$

We thus have two possible forms of Generalization. The first is the original rule of FO with the hypothesis $C$ present; the second is the simplified version (IV.1). The latter is a derived rule; it is not an actual FO inference rule, but it can be simulated by multiple steps in an FO-proof.

Another useful derived rule of inference for FO is

$$\frac{\forall x A(x)}{A(t)} \quad \text{Universal Instantiation (UI) Rule}$$

This inference can be simulated in FO using the UI axiom $\forall x A(x) \to A(t)$ and Modus Ponens.

Combining the Generalization rule (IV.1) and the UI rule gives the Substitution rule:

$$\frac{A(x)}{A(t)} \quad \text{Substitution Rule}$$

This is also a derived rule of inference for FO.

## IV.2.2 Generalization and tautological implication

**Generalization and the meaning of free variables.** There is an important difference between the logical implication $\Gamma \vDash A$ and FO-provabilty $\Gamma \vdash A$ in the way that free variables in $\Gamma$ are interpreted. Namely, for $\Gamma \vdash A$, any free variables that appear in $\Gamma$ are interpreted as being universally quantified. On the other hand, for $\Gamma \vDash A$, the values of the free variables are interpreted as being fixed by some object assignment.

An illustrative example is the meanings of $P(x_1) \vDash P(x_2)$ versus $P(x_1) \vdash P(x_2)$. The logical implication $P(x_1) \vDash P(x_2)$ is false. For instance, we can choose $\mathfrak{A}$ and $\sigma$ with $|\mathfrak{A}| = \{0, 1\}$, $P^{\mathfrak{A}} = \{0\}$, $\sigma(x_1) = 0$ and $\sigma(x_2) = 1$. On the other hand, $P(x_1) \vdash P(x_2)$ is true, as shown by the following:

| | |
|---|---|
| $P(x_1) \vdash P(x_1)$ | Hypothesis |
| $P(x_1) \vdash \forall x_1\, P(x_1)$ | Generalization (IV.1) |
| $P(x_1) \vdash \forall x_1\, P(x_1) \to P(x_2)$ | UI |
| $P(x_1) \vdash P(x_2)$ | Modus Ponens |

or even more succinctly by

| | |
|---|---|
| $P(x_1) \vdash P(x_1)$ | Hypothesis |
| $P(x_1) \vdash P(x_2)$ | Substitution Rule |

Recall from Definition III.30 that a *generalization* of $A$ is formed by adding universal quantifiers to the front of $A$. The universal closure of $A$ is obtained by adding universal quantifiers for those variables which appear free in $A$. We let $\forall(A)$ be the universal closure $\forall x_1, \ldots, x_{i_k}\, A$ as shown in Equation (III.21), where $x_{i_1}, \ldots, x_{i_k}$ are the free variables of $A$ taken (say) in order of increasing subscripts. For $\Gamma$ a set of formulas, we let $\forall(\Gamma)$ denote the set of sentences $\{\forall(B) : B \in \Gamma\}$.

**Theorem IV.8.** *Let $\Gamma$ be a set of formulas and $A$ be a formula.*
(a) *$\Gamma \vdash A$ if and only if $\forall(\Gamma) \vdash A$.*
(b) *$\Gamma \vdash A$ if and only if $\Gamma \vdash \forall(A)$.*
(c) *$\Gamma \vdash A$ if and only if $\forall(\Gamma) \vdash \forall(A)$.*

*Proof.* Part (b) is immediate from Theorem IV.7. Part (a) follows almost as easily from the same theorem. Namely, any formula in $\Gamma$ can be derived with an FO-proof from its universal closure in $\forall(\Gamma)$; conversely, any sentence in $\forall(\Gamma)$ can be derived from the corresponding formula in $\Gamma$. It follows that $\Gamma \vdash A$ holds if and only if $\forall(\Gamma) \vdash A$ holds.

Part (c) follows immediately from parts (a) and (b). □

As a consequence of the last theorem, when we consider questions about whether $\Gamma \vdash A$, we may assume in general that $\Gamma$ is a set of sentences. This is because if $\Gamma$ is not a set of sentences, then we may use $\forall(\Gamma)$ in place of $\Gamma$, and consider whether $\forall(\Gamma) \vdash A$. Furthermore, the Soundness and Completeness theorems below will state that if $\Gamma$ is a set of sentences, then $\Gamma \vDash A$ holds exactly when $\Gamma \vdash A$ holds.

**Side remark:** The reader is warned that many introductory textbooks and other expository texts define $\Gamma \models A$ and $\Gamma \vdash A$ to handle free variables differently. It is fairly common, for instance in the textbooks of Shoenfield [18], Mendelson [13] Manin [11], Monk [14], and Hodel [10] to define the logical implication $\Gamma \models A$ differently than us; in effect treating free variables in $\Gamma$ and $A$ as being universally quantified. They define a formula to be valid in a model $\mathfrak{A}$ provided it is satisfied by $\mathfrak{A}$ with all possible truth assignments; they then define $\Gamma \models A$ to mean that, for all $A$, if every member of $\Gamma$ is valid in $\mathfrak{A}$, then $A$ is valid in $\mathfrak{A}$. Under this definition, $\Gamma \models A$ turns out to be equivalent to $\Gamma \vdash A$ (via the Soundness and Completeness theorems). Other authors, such as Enderton [5], instead define $\Gamma \vdash A$ differently from us; they treat free variables similarly to constant symbols, and do not allow the Generalization rule.

Yet other authors, e.g., the model theory textbooks of Chang and Keisler [3], Marker [12], and the Wikipedia entry for the Completeness Theorem[3] sidestep this issue by defining the notion of $\Gamma \models A$ only for the case where $A$ is a sentence and $\Gamma$ is a set of sentences. Their versions of the Soundness and Completeness Theorems for $\Gamma \models A$ and $\Gamma \vdash A$ are stated only for the case where $A$ is a sentence and $\Gamma$ is a set of sentences.

On the other hand, the textbook of Hinman [9] uses the same conventions about the meanings of $\Gamma \models A$ and $\Gamma \vdash A$ as we do, and states the Soundness and Completeness Theorems in the same way. We adopt this convention since it gives both "$\Gamma \models A$" and "$\Gamma \vdash A$" their most natural meanings; our versions of the Soundness and Completeness Theorems state that $\Gamma \models A$ and $\Gamma \vdash A$ are equivalent provided that $\Gamma$ is a set of sentences.

**Tautologies.** First-order tautologies and first-order tautological implication were defined in Definitions III.46 and III.47. Since FO contains the four axioms PL1-PL4 for propositional logic, and since the propositional proof system PL is complete and can prove all propositional tautologies, it follows that FO-proofs can establish all first-order tautologies and first-order tautological implications:

**Theorem IV.9.** *Let $A$ be a formula, and $\Gamma$ a set of formulas.*
(a) *If $A$ is a tautology, then $\vdash A$.*
(b) *If $\Gamma$ tautologically implies $A$, then $\Gamma \vdash A$.*

*Proof.* Assume $A$ is a tautology. This means that there is a tautological propositional formula $B(p_1, \ldots, p_\ell)$ and first-order formulas $C_1, \ldots, C_\ell$ so that $A$ is $B(C_1, \ldots, C_\ell / p_1 \ldots, p_\ell)$. By the Completeness Theorem for propositional logic, the tautology $B$ has a PL-proof

$$D_1, D_2, \ldots, D_m,$$

where $D_m$ is $B$ of course. Substituting the formulas $C_i$ for the variables $p_i$ yields an FO-proof

$$D_1(\vec{C}/\vec{p}), D_2(\vec{C}/\vec{p}), \ldots, D_m(\vec{C}/\vec{p})$$

[3] *Gödel's Completeness Theorem*, Wikipedia, The Free Encyclopedia, Retrieved November 28, 2021, 15:23 UTC.

of $A$, since $A$ is the same as $D_m(\vec{C}/\vec{p})$, i.e., $B(\vec{C}/\vec{p})$. That proves (a).

For part (b), suppose $\Gamma$ tautologically implies $A$. This means that there is a finite subset $\{B_1, \ldots, B_k\}$ of $\Gamma$ such that

$$B_1 \to B_2 \to \cdots \to B_k \to A$$

is a tautology. By part (a), this has an FO-proof. Introducing the $B_i$'s as hypotheses from $\Gamma$ and using Modus Ponens $k$ times, gives $\Gamma \vdash A$. □

**Corollary IV.10.** *Modus Tollens, Hypothetical Syllogism, and the tautological implication rule* TAUT *are admissible rules of inference for* FO.

**Example IV.11.** We show that $x = y \to g(f(x), z)) = g(f(y), z)$ has an FO-proof:

| | |
|---|---|
| $\vdash x = y \to f(x) = f(y)$ | Axiom $\text{EQ}_f$ |
| $\vdash u = v \to z = z \to g(u, z) = g(v, z)$ | Axiom $\text{EQ}_g$ |
| $\vdash (f(x) = f(y) \to z = z \to g(f(x), z) = g(f(y), z))$ | Substitution Rule (twice) |
| $\vdash z = z$ | Axiom EQ1 |
| $\vdash x = y \to g(f(x), z)) = g(f(y), z)$ | TAUT |

The final step uses the TAUT rule based on the first, third, and fourth lines.

**Example IV.12.** Let $A(x)$ be a formula.[4] We claim that $A(t) \to \exists x\, A(x)$ has an FO-proof.

To prove this, recall that $\exists x\, A(x)$ is an abbreviation for the formula $\neg\forall x\, \neg A(x)$. Thus, $A(t) \to \exists x\, A(x)$ is a shorthand notation for $A(t) \to \neg\forall x\, \neg A(x)$ This formula is tautologically equivalent to the UI axiom $\forall x\, \neg A(x) \to \neg A(t)$. Thus, by the TAUT rule (see Corollary IV.10), $A(t) \to \exists x\, A(x)$ has an FO-proof.

This example gives another useful derived rule of inference for FO. Let $A(x)$ be a formula. The Existential Introduction rule (EI) is:

$$\frac{A(t)}{\exists x\, A(x)} \quad \text{Existential Introduction (EI) Rule}$$

## IV.2.3 The Deduction theorem and (in)consistency

The Deduction Theorem formalizes the intuition that, when trying to prove an implication $A \to B$, it is sufficient to prove $B$ under the assumption of $A$ as an additional hypothesis. However, it is not the the case that $\Gamma, A \vDash B$ always implies $\Gamma \vDash A \to B$ when $A$ is a formula. It does work, however, when $A$ is a sentence.

[4] As a reminder, when we use the relaxed notation for substitution to discuss the formulas $A(x)$ and $A(t)$, this means that $A = A(x)$ is a formula, $t$ is substitutable for $x$ in $A$, and $A(t)$ is the formula $A(t/x)$.

**Theorem IV.13** (Deduction Theorem for FO)**.** *Let $A$ be a sentence, $B$ be a formula, and $\Gamma$ be a set of sentences. Then*

$$\Gamma, A \vdash B \qquad \textit{if and only if} \qquad \Gamma \vdash A \to B.$$

The problem with $A$ being a formula instead of a sentence is that a proof of $B$ from $\Gamma \cup \{A\}$ might use the Generalization rule (GEN) with a variable $x$ that is free in $A$. If we assume that there is no such use of Generalization, then the Deduction Theorem can still hold:

**Theorem IV.14** (Refined Deduction Theorem for FO)**.** *Let $A$ and $B$ be formulas, and $\Gamma$ be a set of formulas.*

(a) *If $\Gamma \vdash A \to B$, then $\Gamma, A \vdash B$.*

(b) *Suppose $\Gamma, A \vdash B$ so there is an* FO*-proof $P$ of $B$ from $\Gamma \cup \{A\}$. Also, suppose that no variable $x$ that appears free in $A$ is used as an eigenvariable for a Generalization inference in the proof $P$. Then $\Gamma \vdash A \to B$.*

*Proof.* Since there are no free variables in a sentence $A$, Theorem IV.13 is a special case of Theorem IV.14. The proof of part (a) of Theorem IV.14 is immediate by Modus Ponens. To prove part (b), suppose that

$$C_1, C_2, C_3, \ldots, C_\ell$$

is an FO-proof of $B$ from $\Gamma, A$ so that $C_\ell$ is the formula $B$. Each $C_i$ is an axiom, a member of $\Gamma$ or the formula $A$, or is inferred by Modus Ponens or by Generalization. We prove by induction on $i$ that $\Gamma \vdash A \to C_i$ for each $i$. The proof by induction splits into four cases. (The first three cases are essentially the same as in the proof of the propositional Deduction Theorem II.10; Case 4 is the only new case.)

**Case 1:** Suppose that $C_i$ is either an axiom or a member of $\Gamma$. Then, certainly $\Gamma \vdash C_i$, and by TAUT, $\Gamma \vdash A \to C_i$ as desired.

**Case 2:** Suppose $C_i$ is $A$ so that $A \to C_i$ is the same as $A \to A$. This is a tautology and thus has an FO-proof.

**Case 3:** Suppose $C_i$ is inferred from Modus Ponens from $C_j$ and $C_k$ with $j, k < i$ and with $C_k$ equal to $C_j \to C_i$. The two induction hypotheses give that $\Gamma \vdash A \to C_j$ and $\Gamma \vdash A \to C_j \to C_i$. From these, $\Gamma \vdash A \to C_i$ by the TAUT rule.

**Case 4:** Suppose $C_i$ is the formula $D \to \forall x\, B$ and was inferred by Generalization from a formula $C_j$, $j < i$, which is equal to $D \to B$ with $x$ not free in $D$. The induction hypothesis is that $\Gamma \vdash A \to C_j$; i.e., that $\Gamma \vdash A \to (D \to B)$. By TAUT, this implies that $\Gamma \vdash A \wedge D \to B$. Since neither $A$ nor $D$ contains a free occurrence of $x$, an application of the Generalization Rule gives that $\Gamma \vdash A \wedge D \to \forall x\, B$. A final use of TAUT gives that $\Gamma \vdash A \to D \to \forall x\, B$. In other words, $\Gamma \vdash A \to C_i$.

That completes the proof by induction and the proof of the Deduction Theorem. □

We next discuss consistency and inconsistency. Their definition and their properties in first-order logic are almost identical to their properties in propositional logic.

**Definition IV.15.** A set $\Gamma$ of formulas is *inconsistent* if any only if there is some formula $A$ such that $\Gamma \vdash A$ and $\Gamma \vdash \neg A$.

First-order inconsistency (in FO) enjoys many of the properties that were established early for propositional inconsistency (for PL). For example, Theorems II.16 and II.18—and their proofs—hold also for FO:

**Theorem IV.16.** *Let $\Gamma$ be a set of formulas.*
(a) *$\Gamma$ is inconsistent if and only if $\Gamma \vdash B$ for all formulas $B$.*
(b) *$\Gamma$ is inconsistent if and only if there is a finite subset $\Gamma_0$ of $\Gamma$ which is inconsistent.*

The next two theorems are the analogues of Theorems II.19 and II.21 and Corollary II.27 about proofs by contradiction and proofs by cases.

**Theorem IV.17** (Proof by Contradiction for FO)**.** *Let $A$ be a sentence and $\Gamma$ be a set of sentences.*
(a) *$\Gamma \cup \{\neg A\}$ is inconsistent if and only if $\Gamma \vdash A$.*
(b) *$\Gamma \cup \{A\}$ is inconsistent if and only if $\Gamma \vdash \neg A$.*

**Theorem IV.18** (Proof-by-cases for FO)**.** *Let $A$ be a sentence, $B$ be a formula, and $\Gamma$ be a set of sentences. Suppose that $\Gamma, A \vdash B$ and $\Gamma, \neg A \vdash B$. Then $\Gamma \vdash B$.*

The proofs of Theorems IV.17 and IV.18 are identical, word-for-word, to the proofs of Theorems II.19 and II.21 and Corollary II.27. The proofs do use the Deduction Theorem for $A$ and $\neg A$ however, and thus we need the assumption that $A$ is a sentence.

Similarly, FO also satisfies the following theorem about finding a consistent extension of $\Gamma$.

**Theorem IV.19.** *Let $\Gamma$ be a consistent set of formulas and let $A$ be a sentence. Then at least one of $\Gamma \cup \{A\}$ and $\Gamma \cup \{\neg A\}$ is consistent.*

The proof of Theorem IV.19 is identical to the proofs used for Theorem II.28 and Corollary II.29. Theorem IV.19 will be useful in the proof of Lindenbaum's theorem below when proving the Completeness Theorem.

One more simple, but useful, observation is that $\Gamma$ is consistent if and only $\forall(\Gamma)$ is consistent. This is an immediate consequence of Theorem IV.8.

## IV.2.4 Syntactic theorems on constants

The Generalization rule allows inferring $\forall x\, A$ from $A$. There is also a form of generalization that allows inferring $\forall x\, A$ from $A(c/x)$ where $c$ is a constant. To get some intuition for this, suppose we are trying to prove $\forall x\, A(x)$ holds. A common proof technique is to introduce a new name $d$ for an object, letting $d$ denote an arbitrary object, and then prove that $A(d)$ holds. Since $d$ was an arbitrary object, we conclude that $\forall x\, A(x)$ holds.

A related intuition arises when trying to prove a formula $B$ from a hypothesis $\exists x\, A(x)$. A common proof technique for this is to introduce a new name $d$ for an object that makes $A(d)$ true. Such an object $d$ must exist if $\exists x\, A(x)$ holds. Then we prove $B$ from the hypothesis that $A(d)$ holds. From this, we conclude that $\exists x\, A(x)$ implies $B$.

These intuitions are formalized by parts (a) and (b.i) of the next theorem.

**Theorem IV.20.** (Theorem on Constants) *Let $\Gamma$ be a set of formulas and that $A(x)$ and $B$ are formulas. Suppose that the constant symbol $c$ does not appear in $A(x)$ or $B$ or in any formula in $\Gamma$. Then*

(a) $\Gamma \vdash A(c)$ *if and only if* $\Gamma \vdash \forall x\, A(x)$.

(b) *Further suppose that $x$ is the only free variable in $A(x)$, so that $\exists x\, A(x)$ and $A(c)$ are sentences. Then*
   (b.i) $\Gamma \cup \{\exists x\, A(x)\} \vdash B$ *if and only if* $\Gamma \cup \{A(c)\} \vdash B$.
   (b.ii) $\Gamma \cup \{\exists x\, A(x)\}$ *is consistent if and only if* $\Gamma \cup \{A(c)\}$ *is consistent.*

*Proof.* We prove (a) first. If $\Gamma \vdash \forall x(x)\, A$, then $\Gamma \vdash A(c)$ follows by Modus Ponens from the UI axiom $\forall x\, A(x) \to A(c)$. So suppose $\Gamma \vdash A(c)$; we need to prove $\Gamma \vdash \forall x\, A(x)$. The proof of $A(c)$ is a sequence of formulas $B_1, B_2, \ldots, B_\ell$. Let $x_n$ be a variable that does not appear in $A(x)$ or in any formula $B_i$ in the proof. For all $i$, let $B'_i$ be the result of replacing every occurrence of $c$ in $B_i$ with $x_n$.

We claim that $B'_1, \ldots, B'_\ell$ is an FO-proof of $A(x_n)$. Note that $x_n$ is substitutable for $x$ in $A(x)$, and $B'_\ell$ is equal to $A(x_n)$. (This is because $x_n$ does not appear in $B_\ell$ and of course is not quantified in $B_\ell$.) To show that each $B'_i$ is correctly introduced as an FO-axioms or as a hypothesis or with an FO inference rule, we must show the following:

- If $B_i \in \Gamma$ is a hypothesis, then $B'_i$ is equal to $B_i$ by the assumption that $c$ does not appear in any formula in $\Gamma$.
- Axioms PL1-PL4 remain correct PL1-PL4 axioms after replacing $c$ with $x_n$.
- The constant $c$ is not even allowed to appear in an equality axiom. Thus if $B_i$ is an equality axiom, $B'_i$ is equal to $B_i$.
- If $B_i$ is a UI axiom, it has the form $\forall x_k\, D(x_k) \to D(t)$, with $k \neq n$ since $x_n$ does not appear in $B_i$. Thus $B'_i$ has the form $\forall x_k\, D'(x_k) \to D'(t')$ where $D'$ and $t'$ are obtained from $D$ and $t$ by replacing each occurrence of $c$ with $x_n$. In other words, $B'_i$ is a UI axiom.
- If $B_i$ is inferred by Modus Ponens from $B_j$ and $B_k$, then clearly $B'_i$ is inferred by Modus Ponens from $B'_j$ and $B'_k$.

- Suppose $B_i$ is inferred by the Generalization rule from $B_j$. Then $B_j$ has the form $C \to D$, and $B_i$ has the form $C \to \forall x_k\, D$, and $k \neq n$. The formula $B'_j$ and $B'_i$ have the forms $C' \to D'$ and $C' \to \forall x_k\, D'$ where $C'$ and $D'$ are obtained from $C$ and $D$ by replacing each occurrence of $c$ with $x_n$. Since $k \neq n$, this means $B'_i$ is inferred from $B'_j$ by a Generalization rule.

That shows that $B'_1, \ldots, B'_\ell$ is a valid FO-proof of $A(x_n)$ from $\Gamma$. Then by the simplified Generalization rule, $\Gamma \vdash \forall x_n\, A(x_n)$. If $x_n$ is the same as the variable $x$, the proof of part (a) is complete. However, it may not be possible to have picked $x_n$ to be the same as $x$, since $x$ might appear in the $B_i$'s. In this case, since $\Gamma \vdash \forall x_n\, A(x_n)$, the UI rule gives $\Gamma \vdash A(x)$. From this, by the simplified Generalization rule again, $\Gamma \vdash \forall x\, A(x)$. That completes the proof of part (a).

With the aid of Theorem IV.17, part (b.i) follows readily from part (a). By Theorem IV.8 we can assume w.l.o.g. that $B$ is a sentence, by replacing $B$ with its universal closure $\forall(B)$. We have:

| | | |
|---|---|---|
| $\Gamma \cup \{\exists x\, A\} \vdash B$ | | |
| $\Leftrightarrow$ | $\Gamma \cup \{\exists x\, A, \neg B\}$ is inconsistent | Theorem IV.17(a) |
| $\Leftrightarrow$ | $\Gamma \cup \{\neg\forall x\, \neg A, \neg B\}$ is inconsistent | $\exists x$ is an abbreviation for $\neg\forall x\, \neg$ |
| $\Leftrightarrow$ | $\Gamma, \neg B \vdash \forall x\, \neg A$ | Theorem IV.17(a) |
| $\Leftrightarrow$ | $\Gamma, \neg B \vdash \neg A(c/x)$ | Part (a) |
| $\Leftrightarrow$ | $\Gamma, A(c/x) \vdash B$ | Two uses of Theorem IV.17 |

Part (b.ii) follows immediately from part (b.i) by taking $B$ to be the negation of a FO axiom. □

**Example IV.21.** We show that $\forall x\, (A(x) \to B(x)) \vdash \exists x\, A(x) \to \exists x\, B(x)$. By the Deduction Theorem IV.14, it will suffice to give an FO-proof of

$$\exists x\, A(x), \forall x\, (A(x) \to B(x)) \vdash \exists x\, B(x),$$

provided the FO-proof does not use any variable other than $x$ as an eigenvariable in any Generalization rule. (Note $A$ and $B$ are permitted to have free variables other than $x$.) By Theorem IV.20, it suffices to prove that

$$A(c), \forall x\, (A(x) \to B(x)) \vdash \exists x\, B(x),$$

where $c$ is a new constant symbol (i.e., $c$ does not appear in $A$ or $B$). We have

| | |
|---|---|
| $A(c), \forall x\, (A(x) \to B(x)) \vdash A(c) \to B(c)$ | UI rule |
| $A(c), \forall x\, (A(x) \to B(x)) \vdash B(c)$ | Modus Ponens |
| $A(c), \forall x\, (A(x) \to B(x)) \vdash \exists x\, B(x)$ | Existential Introduction (EI) |

The FO-proofs underlying the use of UI and EI rules did not use the Generalization rule, so the condition that no free variable of $\exists x\, A(x)$ is used as an eigenvariable is trivially satisfied.

**Constructing FO-proofs**

**Derived rules:**

**Propositional rules:** Modus Tollens, Hypothetical Syllogism and more generally Tautological Implication (TAUT).

**Generalization:** $\dfrac{A(x)}{\forall x\, A(x)}$ **EI Rule:** $\dfrac{A(t)}{\forall x\, A(x)}$

**UI Rule:** $\dfrac{\forall x\, A(x)}{A(t)}$ **Substitution:** $\dfrac{A(x)}{A(t)}$

For Deduction/Contradiction/Cases, $A$ is a sentence.

**Deduction Theorem:** $\Gamma \vdash A \to B$ iff $\Gamma, A \vdash B$.

**Proof by Contradiction:** $\Gamma \vdash A$ iff $\Gamma \cup \{\neg A\}$ is inconsistent.
$\Gamma \vdash \neg A$ iff $\Gamma \cup \{A\}$ is inconsistent.

**Proof by Cases:** If $\Gamma, A \vdash B$ and $\Gamma, \neg A \vdash B$, then $\Gamma \vdash B$.

**Theorem on Constants.** $c$ is a new constant symbol not in $\Gamma$, $A$ or $B$.
For $A(x)$ a formula, $\Gamma \vdash \forall x\, A(x)$ iff $\Gamma \vdash A(c)$.
If $\exists x\, A(x)$ is a sentence, $\Gamma \cup \{\exists x\, A(x)\} \vdash B$ iff $\Gamma \cup \{A(c)\} \vdash B$

# IV.3 The Soundness and Completeness Theorems

The Soundness and Completeness Theorems for first-order logic express the fact that FO is a good proof system. The Soundness Theorem states that FO is "sound" in the sense that any FO-theorem $A$ is a logically valid formula. Likewise, if $A$ is provable using hypotheses from a set $\Gamma$ of sentences, then $\Gamma$ logically implies $A$. This is really a basic property for any proof system for first-order logic since we of course only want to have proofs of formulas that are logically implied.

The Completeness Theorem states the converse, namely that FO is "complete". This means firstly that if $A$ is logically valid, then $A$ has an FO-proof. And, secondly, that if $A$ is a logical consequence of a set $\Gamma$ of sentences, then $\Gamma \vdash A$. In other words, FO is strong enough to give proofs of all logical validities and all logical implications. This is really remarkable!

We now state the Soundness and Completeness Theorems more carefully and prove the Soundness Theorem. After that, Section IV.4 will present the more difficult proof of the Completeness Theorem.

**Theorem IV.22** (Soundness Theorem for FO)**.** *Let $A$ be a formula and $\Gamma$ be*

*a set of sentences.*
(a) *If* $\Gamma$ *is satisfiable, then* $\Gamma$ *is consistent.*
(b) *If* $\Gamma \vdash A$*, then* $\Gamma \vDash A$*.*

**Theorem IV.23** (Completeness Theorem for FO)**.** *Let* $A$ *be a formula and* $\Gamma$ *be a set of formulas.*
(a) *If* $\Gamma$ *is consistent, then* $\Gamma$ *is satisfiable.*
(b) *If* $\Gamma \vDash A$*, then* $\Gamma \vdash A$*.*

Putting the Soundness and Completeness Theorems together gives

**Corollary IV.24.** *Let* $A$ *be a formula and* $\Gamma$ *be a set of sentences.*
(a) $\Gamma$ *is consistent if and only if* $\Gamma$ *is satisfiable.*
(b) $\Gamma \vDash A$ *if and only if* $\Gamma \vdash A$*.*

When there are no hypotheses, so $\Gamma = \varnothing$, we have:

**Corollary IV.25.** *Let* $A$ *be a formula. Then* $\vDash A$ *if and only if* $\vdash A$*.*

Parts (a) and (b) of the Soundness Theorem are easily seen to be equivalent to each other. Likewise, parts (a) and (b) of the Completeness Theorem are easily seen to be equivalent to each other. To illustrate this, we first show that part (b) of the Soundness Theorem implies part (a).

Choose $B$ to be a formula which is both FO-provable and logically valid. For instance, take $B$ to be a a PL1 axiom. For $\Gamma$ a set of sentences, we have

| | | | |
|---|---|---|---|
| $\Gamma$ is inconsistent | | | |
| | $\Leftrightarrow$ | $\Gamma \vdash \neg B$ | By choice of $B$ and Theorem IV.16 |
| | $\Rightarrow$ | $\Gamma \vDash \neg B$ | By part (b) of the Soundness Theorem |
| | $\Leftrightarrow$ | $\Gamma$ is unsatisfiable | By choice of $B$ as logically valid. |

That proves that part (b) of the Soundness Theorem implies part (a). To prove part (b) of the Completeness Theorem from part (a), recall that $\forall(A)$ is the universal closure of $A$ and argue as follows:

| | | | |
|---|---|---|---|
| $\Gamma \vDash A$ | | | |
| | $\Leftrightarrow$ | $\Gamma \vDash \forall(A)$ | By Theorem III.37(b) |
| | $\Leftrightarrow$ | $\Gamma \cup \{\neg\forall(A)\}$ is unsatisfiable | Theorem III.45(b) |
| | $\Rightarrow$ | $\Gamma \cup \{\neg\forall(A)\}$ is inconsistent | By part (a) of the Completeness Theorem |
| | $\Leftrightarrow$ | $\Gamma \vdash \forall(A)$ | Theorem IV.17 (Proof by Contradiction) |
| | $\Leftrightarrow$ | $\Gamma \vdash A$ | Theorem IV.8(b) |

We still need to prove part (b) of the Soundness Theorem and part (a) of the Completeness Theorem. These proofs will be presented in the next sections. After that, Section IV.5 will state and prove the Compactness Theorem for FO for the general case where $\Gamma$ is a set of formulas.

**The proof of the Soundness Theorem.** We already proved that part (b) of the Soundness Theorem implies part (a). Now we prove part (b).

*Proof of part (b) of the Soundness Theorem.* Assume that $\Gamma$ is a set of sentences and $\Gamma \vdash A$. We must show $\Gamma \models A$. Let

$$B_1, B_2, \ldots, B_k$$

be an FO-proof of $A$ from $\Gamma$. We will prove by induction on $i$ that $\Gamma \models \forall(B_i)$. More specifically, we fix a structure $\mathfrak{A}$ such that $\mathfrak{A} \models \Gamma$. Then we show by induction on $i$ that $\mathfrak{A} \models B_i[\sigma]$ holds for every object assignment $\sigma$. It is important to note that $\sigma$ is not fixed; instead, the induction hypothesis holds for all object assignments $\sigma$.

The proof breaks into cases depending on how $B_i$ is inferred:

**Case 1:** Suppose $B_i$ is a hypothesis, $B_i \in \Gamma$. By assumption, $\mathfrak{A} \models B_i$, so this case is trivial.

**Case 2:** Suppose $B_i$ is axiom from PL1-PL4, and thus a tautology. Then, for all $\sigma$, $\mathfrak{A} \models B_i[\sigma]$ by Theorem III.49(a).

**Case 3:** Suppose $B_i$ is one of the equality axioms EQ1-EQ3. Since the definition of truth interprets = as true equality (see Theorem III.75), $\mathfrak{A} \models B_i[\sigma]$ holds for all $\sigma$.

**Case 4:** Suppose $B_i$ is one of the equality axioms $\mathrm{EQ}_f$ or $\mathrm{EQ}_P$. By Lemma III.62, $\mathfrak{A} \models B_i[\sigma]$ holds for all $\sigma$.

**Case 5:** Suppose $B_i$ is a UI axiom $\forall x\, C(x) \to C(t)$. Then $\mathfrak{A} \models B_i[\sigma]$ holds by Theorem III.66.

**Case 6:** Suppose $B_i$ is introduced by Modus Ponens from $B_j$ and $B_k$ where $B_k$ is $B_j \to B_i$. Fix an arbitrary object assignment $\sigma$. By the induction hypotheses, $\mathfrak{A} \models B_j[\sigma]$ and $\mathfrak{A} \models (B_j \to B_k)[\sigma]$. The definition of truth for $\to$, yields that $\mathfrak{A} \models B_j[\sigma]$ holds.

**Case 7:** Suppose $B_i$ is introduced by a Generalization (GEN) inference. Then $B_i$ has the form $C \to \forall x\, D$ and for some $j < i$, $B_j$ has the form $C \to D$ where $x$ is not free in $C$. The induction hypothesis states that, for all $\sigma$, $\mathfrak{A} \models (C \to D)[\sigma]$. Thus by the definition of truth, $\mathfrak{A} \vdash \forall x\, (C \to D)[\sigma]$ for all $\sigma$. Therefore, by Example III.42, $\mathfrak{A} \vdash (C \to \forall x\, D)[\sigma]$ holds for all $\sigma$.

That completes the proof of the Soundness Theorem. □

## IV.4 Proof of the Completeness Theorem

We now prove the Completeness Theorem IV.23. For this, it suffices to prove part (a), so we assume that $\Gamma$ is a consistent set of formulas and need to prove that $\Gamma$ is satisfiable by constructing a structure $\mathfrak{A}$ which satisfies $\Gamma$. The proof

will use the same ideas as the proof of the Completeness Theorem for the propositional proof system PL; however, there are substantial additional ingredients needed to handle quantifiers and to construct a model $\mathfrak{A}$ of $\Gamma$. Very briefly the proof requires the following steps: First, additional constant symbols are introduced and $\Gamma$ is extended to a set $\Delta$ of sentences which is "strongly Henkin". Second, Lindenbaum's Theorem is used to further extend $\Gamma$ to to a set $\Pi$ of sentences which is complete and still consistent and strongly Henkin. Third, a structure $\mathfrak{A}$ is constructed that satisfies $\Pi$.

We will need to be careful at times about what language is being used. We assume that $\Gamma$ is a set of $L$-sentences. We also use a larger language $L^+$ which extends $L$ with the addition of constant symbols $d_i$ so that

$$L^+ \;=\; L \cup \{d_1, d_2, d_3, \ldots\}. \tag{IV.2}$$

The constant symbols $d_i$ are new and are not in $L$. The sets $\Delta$ and $\Pi$ will be sets of $L^+$-sentences. The structure $\mathfrak{A}$ will be an $L^+$-structure, but in the end, $\mathfrak{A}$ will be restricted to the language $L$ so as to obtain an $L$-structure satisfying $\Gamma$.

**Henkin and strongly Henkin.** Recall that a closed term is a term that does not contain any variables $x_i$. For example, $0 + S(0)$ is a closed $L_{\mathrm{PA}}$-term, but $0 + x_1$ is not.

**Definition IV.26.** Let $\Gamma$ be a set of $L$-sentences. The set $\Gamma$ is *Henkin* if the following holds: For any sentence $\exists x\, A(x)$ such that $\Gamma \vdash \exists x\, A(x)$, there is a closed $L$-term $t$ such that $\Gamma \vdash A(t)$.

The point of being Henkin is that, for any provable sentence $\exists x\, A(x)$ that asserts the existence of an object, there is a closed term $t$ that gives an example of such an object. The closed term $t$, being variable-free, serves as a name for a particular object; it can be considered to be "witness" for the truth of $\exists x\, A(x)$.

The Henkin property was stated using $\exists x$. By convention, $\exists x$ is an abbreviation for $\neg\forall x\neg$. Therefore, an equivalent way to state the Henkin property is as follows: For any formula $A$, if $\Gamma \vdash \neg\forall x\, A(x)$, then for some closed term $t$, $\Gamma \vdash \neg A(t)$. The term $t$ can be viewed as a "counterexample" that witnesses the falsity of $\forall x\, A(x)$.

**Definition IV.27.** Let $\Gamma$ be a set of $L$-sentences. The set $\Gamma$ is *strongly Henkin* if the following holds: For any sentence $\forall x\, A(x)$, there is a constant symbol $c \in L$ such that the sentence

$$A(c) \to \forall x\, A(x)$$

is a member of $\Gamma$.

Note that if $\Gamma$ is strongly Henkin, then $\Gamma$ is Henkin. This is because $c$ is a closed term and by the previous remark and Modus Tollens.

Given that $\Gamma$ is consistent, we wish to form a larger set $\Delta$ of sentences which augments $\Gamma$ to be both consistent and strongly Henkin. For this, we must introduce new constant symbols. (Indeed, the language $L$ may not have any constant symbols at all, and thus there may not even be any closed $L$-terms.)

**Lemma IV.28.** *Suppose $\exists x\, A(x)$ is an $L$-sentence and $\Gamma$ is a consistent set of $L$-sentences. Let $c$ be a new constant symbol, so $c \notin L$. Then*

$$\Gamma \cup \{A(c) \to \forall x\, A(x)\} \text{ is consistent.}$$

*Proof.* Suppose the conclusion fails. Then, by the Proof by Contradiction Theorem IV.17(b),

$$\Gamma \vdash \neg[A(c) \to \forall x\, A(x)].$$

Therefore, by the TAUT rule, $\Gamma \vdash \neg\forall x\, A(x)$ and $\Gamma \vdash A(c)$. From the latter, since $c$ is new, the Theorem on Constants IV.20 implies that $\Gamma \vdash \forall x\, A(x)$. This contradicts the consistency of $\Gamma$. □

Henceforth, we let $d_1, d_2, d_3, \ldots$ be new constant symbols that do not appear in $\Gamma$, and let $L^+$ be the language (IV.2) consisting of $L$ plus the constant symbols $d_i$.

**Theorem IV.29.** *Suppose $\Gamma$ is a consistent set of $L$-sentences. Then there is a consistent and strongly Henkin set $\Delta$ of $L^+$-sentences such that $\Delta \supseteq \Gamma$.*

*Proof.* We will form $\Delta$ by adding sentences to $\Gamma$ one at a time while preserving consistency. Enumerate all $L^+$-sentences that start with a universal quantifier in an infinite sequence as

$$\forall x_{k_1} A_1(x_{k_1}),\ \forall x_{k_2} A_2(x_{k_2}),\ \forall x_{k_3} A_3(x_{k_3}),\ \ldots$$

so that no constant symbol $d_j$ appears in any $\forall x_{k_i} A_i(x_{k_i})$ with $i \le j$.[5] The important point is that every sentence of the form $\forall x\, A(x)$ appears in the sequence at least once. It is OK if the same sentence appears multiple times in the sequence.

Define $\Gamma_0 = \Gamma$ and for each $i > 0$, define

$$\Gamma_i \;=\; \Gamma_{i-1} \cup \{A_i(d_i) \to \forall x_{k_i}\, A(x_{k_i})\}. \tag{IV.3}$$

We claim that each $\Gamma_i$ is consistent. This is readily proved by induction on $i$; the induction step can use Lemma IV.28 since $d_i$ does not appear in $\Gamma_{i-1}$ or in $A(x_{k_i})$.

The set $\Delta$ is defined as $\Delta = \bigcup_i \Gamma_i$. Note $\Gamma_i \supset \Gamma_{i-1}$ for all $i$. If $\Delta$ was inconsistent, then since proofs are finite, some finite subset of $\Delta$ would be inconsistent. That would imply that some $\Gamma_i$ is inconsistent. Therefore, $\Delta$ must be consistent. By construction, $\Delta$ contains all sentences of the form $A_i(d_i) \to \forall x_{k_i}\, A(x_{k_i})$ and thus is strongly Henkin. □

[5] One way to do this is to enumerate the $L^+$-sentences in arbitrary order, discarding the ones that do not start with the symbol $\forall$. To ensure that the constant symbol $d_j$ does not appear in the first $j$ formulas, one can insert extra sentences into the sequence, say of the form $\exists x_1\, x_1 = x_1$ (if the equality sign, $=$, is in $L$) or $\forall x_1\, (P(x_1, \ldots, x_1) \leftrightarrow P(x_1, \ldots, x_1))$. These extra inserted sentences serve no purpose except to delay the appearance of a constant symbol $d_j$ until after the $j$-th sentence.

**Lindenbaum's theorem.** Lindenbaum's Theorem was earlier proved for the propositional proof system PL. The same construction works also for sets of first-order sentences.

**Definition IV.30.** A set $\Gamma$ of sentences is *complete* if, for every sentence $A$, either $A \in \Gamma$ or $\neg A \in \Gamma$.

**Theorem IV.31** (Lindenbaum's Theorem)**.** *Suppose $\Gamma$ is a consistent set of sentences. Then there is a consistent, complete set $\Pi$ of sentences such that $\Gamma \subseteq \Pi$.*

*Proof.* The proof is identical to the proof of Lindenbaum's Theorem II.35 except that now we are working with first-order sentences instead of propositional formulas and that Theorem IV.19 is used instead of Corollary II.29. □

So far, we have established that $\Gamma$ can be extended to a complete and strongly Henkin set of sentences.

**Corollary IV.32.** *Suppose $\Gamma$ is a consistent set of $L$-sentences. There there is a consistent, complete, strongly Henkin set $\Pi$ of $L^+$-sentences such that $\Gamma \subset \Pi$.*

*Proof.* First use Theorem IV.29 to form $\Delta$ as a strongly Henkin, consistent set of $L^+$-sentences such that $\Delta \supset \Gamma$. Then apply Lindenbaum's Theorem (IV.31) to obtains a consistent, strongly Henkin, complete set $\Pi$ of $L^+$-sentences such that $\Pi \supset \Delta$. □

Note that Lindenbaum's Theorem is applied to $\Delta$, not $\Gamma$. Namely, the strongly Henkin construction is done before Lindenbaum's Theorem is invoked.

**Lemma IV.33.** *Let $\Pi$ be a strongly Henkin, consistent, complete set of $L^+$-sentences as above. Suppose $A$, $B$ and $\forall x\, C(x)$ are $L^+$-sentences.*

(a) *$A \in \Pi$ if and only if $\neg A \notin \Pi$.*
(b) *$A \to B \in \Pi$ if and only if $A \notin \Pi$ or $B \in \Pi$.*
(c) *If $\Pi \vdash A$, then $A \in \Pi$.*
(d) *$\neg\forall x\, C(x) \in \Pi$ if and only if, for some closed term $t$, $\neg C(t) \in \Pi$.*
(e) *$\forall x\, C(x) \in \Pi$ if and only if, for all closed terms $t$, $C(t) \in \Pi$.*

*Proof.* The proofs of parts (a) and (b) are identical to the proof of Lemma II.36 except that now $A$ and $B$ are first-order sentences instead of propositional formulas. For part (c), suppose $\Pi \vdash A$. Then $\neg A$ cannot be in $\Pi$, since otherwise $\Pi$ would be inconsistent. Then, by (a), $A \in \Pi$.

For (d), first suppose $\neg\forall x\, C(x) \in \Pi$. Since $\Pi$ is strongly Henkin, it is also Henkin; therefore $\Pi \vdash \neg C(t)$ for some closed term $t$. By (c), $\neg C(t)$ is in $\Pi$. Second suppose $\neg C(t) \in \Pi$. Since $\neg C(t) \vdash \neg\forall x\, C(x)$, part (c) implies that $\neg\forall x\, C \in \Gamma$. That proves (d).

Part (e) is immediate from parts (a) and (d). □

**Construction of a Henkin model.** The next step in the proof of the Completeness Theorem is to define an $L^+$-structure $\mathfrak{A}$ that will be shown to satisfy $\Pi$. The intuition for constructing $\mathfrak{A}$ is that the complete set $\Pi$ will guide the choice of what is true in $A$. This is analogous to what was done in the proof of the Completeness Theorem II.33 for propositional logic (PL), where a truth assignment was defined by letting $\varphi(p_i) = \mathrm{T}$ if and only if $p_i \in \Pi$. And, indeed, the structure $\mathfrak{A}$ will be defined so that $P^{\mathfrak{A}}(t_1, \ldots, t_k)$ is true if and only if $P(t_1, \ldots, t_k) \in \Pi$.

But, since we are working in first-order logic, we also must define the universe $|\mathfrak{A}|$ and give interpretations to the constant symbols and function symbols. The solution to this is simultaneously ingenious and straightforward: the universe of $\mathfrak{A}$ consists of (essentially) the closed $L^+$-terms. If the equality sign is not included in the language $L$, this idea works without modification. However, if the equality sign, $=$, is part of the language $L$, then the universe of $\mathfrak{A}$ consists of the set of closed $L^+$-terms, but identifying terms $s$ and $t$ such that $s = t$ is in $\Pi$. This identification is done by showing that "$s = t \in \Pi$" is an equivalence relation $\sim$ on terms. We will write $[s]$ to denote the equivalence class containing $s$. The universe $|\mathfrak{A}|$ will be the set of equivalence classes $[s]$, for $s$ ranging over the closed $L^+$-terms.

**Definition IV.34** (Definition of $\sim$)**.** Let $s$ and $t$ be closed $L^+$-terms.

(a) If $=$ is not a symbol of $L$, then $s \sim t$ holds if $s$ and $t$ are the same term. (In other words, $\sim$ is the identity relation.)

(b) If $=$ is a symbol of $L^+$, then $s \sim t$ holds if the sentence $s = t$ is in $\Pi$.

**Lemma IV.35.** *The relation $\sim$ is an equivalence relation.*

*Proof.* By definition, $\sim$ is an equivalence relation if is reflexive, symmetric and transistive. If $=$ is not in the language, then $\sim$ is the identity relation and of course is an equivalence relation. So suppose that the equality sign, $=$, is in $L$.

To show reflexivity, we must prove that $s \sim s$ by proving that $s = s$ is in $\Pi$. It is easy to check that $\Pi \vdash s = s$ (using an EQ1 axiom and the Substitution rule). Hence, by Lemma IV.33(c), $s = s \in \Pi$.

To show symmetry, we must show that if $s \sim t$ then $t \sim s$, or equivalently, that if $s = t \in \Pi$ then $t = s \in \Pi$. By EQ2 and the Substitution rule, $\vdash s = t \to t = s$. Suppose $s = t \in \Pi$. Then $\Pi \vdash t = s$; thus $t = s \in \Pi$ by Lemma IV.33(c).

Transitivity works similarly: we must show that if $r \sim s$ and $s \sim t$, then $r \sim t$. Suppose $r = s \in \Pi$ and $s = t \in \Pi$. It will suffice to show $r = t \in \Pi$. Now $\vdash r = s \to s = t \to r = t$, as it is a substitution instance of an EQ3 axiom. Then, using Modus Ponens twice, $\Gamma \vdash r = t$, so $r = t \in \Gamma$. □

**Definition IV.36.** The $L^+$-structure $|\mathfrak{A}|$ is defined by letting

$$|\mathfrak{A}| \;=\; \{[s] : s \text{ is a closed } L^+\text{-term}\}, \tag{IV.4}$$

and defining the interpretations of the non-logical symbols by

$$\begin{aligned} c^{\mathfrak{A}} &= [c] \\ f^{\mathfrak{A}} &= \{\langle [s_1], \ldots, [s_k], [f(s_1, \ldots, s_k)]\rangle : s_1, \ldots s_k \text{ are closed } L^+\text{-terms}\} \\ P^{\mathfrak{A}} &= \{\langle [s_1], \ldots, [s_k]\rangle : P(s_1, \ldots, s_k) \in \Pi\}. \end{aligned}$$

A more intuitive way to express the definition of the $k$-ary function $f^{\mathfrak{A}}$ is that

$$f^{\mathfrak{A}}([s_1], \ldots, [s_k]) = [f(s_1, \ldots, s_k)].$$

Note that each $[s_i]$ is an object in $|\mathfrak{A}|$. The function $f^{\mathfrak{A}}$ takes as input the $k$ objects $[s_1], \ldots, [s_k]$; it outputs the object $[f(s_1, \ldots, s_k)]$. Similarly, a more intuitive way to write the definition of the $k$-ary predicate $P^{\mathfrak{A}}$ is that

$$P^{\mathfrak{A}}([s_1], \ldots, [s_k]) \text{ is true} \qquad \text{if and only if} \qquad P(s_1, \ldots, s_k) \in \Pi.$$

We still need to show that $f^{\mathfrak{A}}$ and $P^{\mathfrak{A}}$ are *well-defined* by the above definition. The issue is that $f^{\mathfrak{A}}([s_1], \ldots, [s_k])$ was defined in terms of particular representatives $s_1, \ldots, s_k$ of equivalence classes. To show that $f^{\mathfrak{A}}$ has a well-definition, we must show that, for all closed $L^+$-terms $s_1, \ldots, s_k$ and $r_1, \ldots, r_k$,

$$\text{If } r_i \sim s_i \text{ for all } i, \text{ then } f(r_1, \ldots, r_k) \sim f(s_1, \ldots, s_k).$$

This is equivalent to showing that the definition of $f^{\mathfrak{A}}$ as a set of $(k+1)$-tuples defines the graph of a (single-valued) function.

Similarly to show that $P^{\mathfrak{A}}$ is well-defined, we must show that,

$$\text{If } r_i \sim s_i \text{ for all } i, \text{ then } P(r_1, \ldots, r_k) \in \Pi \text{ iff } P(s_1, \ldots, s_k) \in \Pi.$$

**Lemma IV.37.** *The functions $f^{\mathfrak{A}}$ and predicates $P^{\mathfrak{A}}$ are well-defined.*

*Proof.* Suppose that $r_i \sim s_i$ for $i = 1, \ldots, k$. Thus $r_i = s_i \in \Pi$ for all $i$. In addition, if $f$ is a $k$-ary function symbol, then

$$\vdash r_1 = s_1 \to r_2 = s_2 \to \cdots \to r_k = s_k \to f(r_1, \ldots, r_k) = f(s_1, \ldots, s_k),$$

since this is a substitution instance of an $\mathrm{EQ}_f$ axiom. Therefore, $\Pi \vdash f(r_1, \ldots, r_k) = f(s_1, \ldots, s_k)$, so $f(r_1, \ldots, r_k) \sim f(s_1, \ldots, s_k)$. In addition, if $P$ is a $k$-ary predicate symbol,

$$\vdash r_1 = s_1 \to r_2 = s_2 \to \cdots \to r_k = s_k \to (P(r_1, \ldots, r_k) \leftrightarrow P(s_1, \ldots, s_k)).$$

Thus, $\Pi \vdash P(r_1, \ldots, r_k)$ if and only if $\Pi \vdash P(s_1, \ldots, s_k)$. Thus $P(r_1, \ldots, r_k) \in \Pi$ if and only if $P(s_1, \ldots, s_k)$. □

**Lemma IV.38.** *Let $\mathfrak{A}$ be as defined above and let $\sigma$ be an object assignment. For all closed $L^+$-terms $t$, we have $\sigma(t) = [t]$. Hence $t^{\mathfrak{A}} = [t]$.*

*Proof.* This is proved by induction on $t$. The base case is where $t$ is a constant symbol. By the definition of truth, $\sigma(c) = c^{\mathfrak{A}} = [c]$.

The induction step is where $t$ is $f(s_1, \ldots, s_k)$. The $k$ many induction hypotheses state that $\sigma(s_i) = [s_i]$. Thus

$$\begin{array}{lcll} (f(s_1,\ldots,s_k))^{\mathfrak{A}} & = & \sigma(f(s_1,\ldots,s_k)) & \text{Definition III.19} \\ & = & f^{\mathfrak{A}}(\sigma(s_1),\ldots\sigma(s_k)) & \text{Definition of truth} \\ & = & f^{\mathfrak{A}}([s_1],\ldots,[s_k]) & \text{Induction hypotheses} \\ & = & [f(s_1,\ldots,s_k)] & \text{Definition of } f^{\mathfrak{A}} \quad \square \end{array}$$

**Final stage of the proof of the Completeness Theorem.** We are now ready to finish the proof of the Completeness Theorem. We have constructed a consistent, complete, strongly Henkin set $\Pi$ of sentences with $\Gamma \subset \Pi$. We have also defined an $L^+$-structure $\mathfrak{A}$. The remaining step is to show that $\mathfrak{A} \vDash \Pi$. From that, it follows that $\mathfrak{A} \vDash \Gamma$. Of course, $\mathfrak{A}$ is an $L^+$-structure not an $L$-structure, but by taking the restriction of $\mathfrak{A}$ to the language $L$, i.e. by discarding the interpretations of the constant symbols $d_i$ in $L^+ \setminus L$, we obtain an $L$-structure that satisfies $\Gamma$. (See Theorem III.87.)

**Lemma IV.39.** $\mathfrak{A} \vDash \Pi$. *Hence* $\mathfrak{A} \vDash \Gamma$.

*Proof.* We shall prove:

**Claim.** *For every $L^+$-sentence $A$, we have $\mathfrak{A} \vDash A$ if and only if $A \in \Pi$.*

The proof of the claim is by induction on the logical complexity of $A$. Two of the induction steps are very similar to the proof of the claim on page 61 used for the proof of the Completeness Theorem for PL, but we include them here for completeness.[6]

**Base case #1:** Suppose $A$ is an atomic formula $P(s_1,\ldots,s_k)$. By Lemma IV.38, $\sigma(s_i) = [s_i]$ for any truth assignment $\sigma$. Therefore, by the definition of truth, and the definition of $P^{\mathfrak{A}}$,

$$\mathfrak{A} \vDash P(s_1,\ldots,s_k) \;\Leftrightarrow\; \langle [s_1],\ldots,[s_k]\rangle \in P^{\mathfrak{A}} \;\Leftrightarrow P(s_1,\ldots,s_k) \in \Pi.$$

**Base case #2:** Suppose $A$ is an atomic formula $s_1 = s_2$. By Lemma IV.38, $\sigma(s_1) = [s_1]$ and $\sigma(s_2) = [s_2]$ for any truth assignment $\sigma$. Therefore $\mathfrak{A} \vDash s_1 = s_2$ if and only $[s_1] = [s_2]$. That holds if and only if $s_1 \sim s_2$, and that is further equivalent to $s_1 = s_2 \in \Pi$ by the definition of $\sim$. Thus, $\mathfrak{A} \vDash s_1 = s_1$ is true if and only if $s_1 = s_2$ is in $\Pi$.

**Induction step #1:** Suppose $A$ is $\neg B$. The induction hypothesis tells that the claim holds for $B$. Thus,

$$\begin{array}{lcll} \neg B \in \Pi & \Leftrightarrow & B \notin \Pi & \text{Lemma IV.33(a)} \\ & \Leftrightarrow & \mathfrak{A} \nvDash B & \text{Induction hypothesis} \\ & \Leftrightarrow & \mathfrak{A} \vDash \neg B & \text{Definition of truth} \end{array}$$

**Induction step #2:** Suppose $A$ is $B \to C$. The induction hypothesis holds for both $B$ and $C$. Then

[6] Pun not intended!

| | | | |
|---|---|---|---|
| $(B \to C) \in \Pi$ | $\Leftrightarrow$ | $B \notin \Pi$ or $C \in \Pi$ | Lemma IV.33(b) |
| | $\Leftrightarrow$ | $\mathfrak{A} \nvDash B$ or $\mathfrak{A} \vDash C$ | Induction hypotheses for $B$ and $C$ |
| | $\Leftrightarrow$ | $\mathfrak{A} \vDash (B \to C)$ | Definition of truth |

**Induction step #3:** Suppose $A$ is $\forall x\, B(x)$. The induction hypothesis holds for $B(t)$ for every closed $L^+$-term $t$. Then

| | | | |
|---|---|---|---|
| $\forall x\, B(x) \in \Pi$ | $\Leftrightarrow$ | for all closed $L^+$-terms, $B(t) \in \Pi$ | Lemma IV.33(e) |
| | $\Leftrightarrow$ | for all closed $L^+$-terms $t$, $\mathfrak{A} \vDash B(t)$ | Induction hypotheses for $B(t)$ |
| | $\Leftrightarrow$ | for all closed $L^+$-terms $t$, $\mathfrak{A} \vDash B(t^{\mathfrak{A}})$ | Theorem III.74 |
| | $\Leftrightarrow$ | for all $\sigma$, $\mathfrak{A} \vDash B(x)[\sigma]$ | Lemma IV.38 |
| | $\Leftrightarrow$ | $\mathfrak{A} \vDash \forall x\, B(x)$ | Definition of truth |

Therefore $\mathfrak{A} \vDash \forall x\, B(x)$ if and only if $\forall x\, B(x) \in \Pi$. □

That concludes the proof of the Completeness Theorem.

## IV.5 Compactness Theorem

The Compactness Theorem states that $\Gamma \vDash A$ if and only if there is a finite subset $\Gamma_0$ of $\Gamma$ such that $\Gamma_0 \vDash A$:

**Theorem IV.40** (Compactness Theorem for Sentences)**.** *Let $\Gamma$ be a set of sentences and $A$ be a formula.*

(a) *$\Gamma$ is satisfiable if and only if $\Gamma$ is finitely satisfiable.*
(b) *$\Gamma \vDash A$ if and only if there is a finite subset $\Gamma_0$ of $\Gamma$ such that $\Gamma_0 \vDash A$.*

*Proof.* This follows immediately from parts (a) of the Soundness and Completeness Theorems. Namely, suppose $\Gamma \vDash A$. Then $\Gamma \vdash A$ by the Completeness Theorem, and thus by the finiteness of proofs, there is a finite subset $\Gamma_0$ of $\Gamma$ such that $\Gamma_0 \vdash A$. Finally, the Soundness Theorem implies that $\Gamma_0 \vDash A$. □

The Compactness Theorem also holds when $\Gamma$ is a set of formulas.

**Corollary IV.41** (Compactness Theorem for Formulas)**.** *Let $\Gamma$ be a set of formulas and $A$ be a formula.*

(a) *$\Gamma$ is satisfiable if and only if $\Gamma$ is finitely satisfiable.*
(b) *$\Gamma \vDash A$ if and only if there is a finite subset $\Gamma_0$ of $\Gamma$ such that $\Gamma_0 \vDash A$.*

*Proof.* This follows immediately from the Compactness Theorem for sentences with the aid of Theorem III.88 about replacing variables with new constant symbols for semantic implication. Let $A$ and $\Gamma$ use the language $L$. Let $L'$ be obtained by adding new constant symbols $d_1, d_2, d_3, \ldots$. Then by Theorem III.88, $\Gamma$ is satisfiable if and only if $\Gamma(\vec{d}/\vec{x})$ is satisfiable. Likewise, $\Gamma$ is finitely satisfiable if and only if $\Gamma(\vec{d}/\vec{x})$ is finitely satisfiable. And, by Theorem IV.40, $\Gamma(\vec{d}/\vec{x})$ is satisfiable if and only if it is finitely satisfiable. That proves part (a). Part (b) is proved similarly. □

## Applications of the Compactness Theorem.

**Finiteness and overspill.** Our first application of the Compactness Theorem is about the undefinability of finiteness. We now again work with an arbitrary fixed language $L$ and assume that it contains the equality sign.[7]

**Definition IV.42.** A structure $\mathfrak{A}$ is *finite* if its universe $|\mathfrak{A}|$ is finite. The *cardinality* of $\mathfrak{A}$ is the cardinality of $|\mathfrak{A}|$.

**Theorem IV.43.** *There is no sentence $A$ that defines the set of finite structures. In other words, the class of finite structures is not an elementary class (*EC*).*

*Proof.* Suppose, for the sake of a contradiction, that there is a sentence $A$ such that, for all structures $\mathfrak{A}$, $\mathfrak{A} \vDash A$ holds if and only if $\mathfrak{A}$ is finite. From Exercise III.12, for any fixed $k > 1$, there is a formula $AtLeast_k$ which states that $|\mathfrak{A}|$ has cardinality at least $k$. For example, one way to construct $AtLeast_k$ is to form the sentence $\exists x_1 \cdots \exists x_k \bigwedge_{i<j} x_i \neq x_j$.

Let $\Gamma$ be the set $\{AtLeast_k : k \geq 2\}$. Clearly, $\mathfrak{A} \vDash \Gamma$ holds if and only if $\mathfrak{A}$ is infinite. Therefore, $\Gamma \cup \{A\}$ is an unsatisfiable set of sentences, since any model of $\Gamma \cup \{A\}$ would have to be both finite and infinite. By the Compactness Theorem, there is a finite subset $\Gamma'$ of $\Gamma$ such that $\Gamma' \cup \{A\}$ is unsatisfiable.

Let $k$ be the maximum value such that $AtLeast_k$ is in $\Gamma'$. Let $\mathfrak{A}$ be any structure of size at least $k$. There must exist such a structure $\mathfrak{A}$ since it can be formed by letting the universe have $k$ objects and defining the interpretations of the non-logical symbols arbitrarily. But then, $\mathfrak{A} \vDash A$ since it is finite, and $\mathfrak{A}$ has size $\geq k$. Therefore $\mathfrak{A}$ satisfies every sentence in $\Gamma' \cup \{A\}$, contradicting the unsatisfiability of $\Gamma' \cup \{A\}$. □

The set $\Gamma$ from the proof defines the class of infinite structures, since $\mathfrak{A} \vDash \Gamma$ holds if and only if $A$ is infinite. In the terminology of Section III.11, this means that the class of infinite structures is an elementary class in the wide sense ($\mathrm{EC}_\Delta$), and is equal to $\operatorname{Mod}\Gamma$.

We can now state an even stronger form of Theorem IV.43 that states that there is not even an infinite set of sentences that defines the property of being finite.

**Theorem IV.44.** *The class of finite structures is not* $\mathrm{EC}_\Delta$*. That is, there is no set* $\Pi$ *of sentences such that, for all structures* $\mathfrak{A}$*,* $\mathfrak{A} \vDash \Pi$ *if and only if* $\mathfrak{A}$ *is finite.*

Theorem IV.44 is an immediate corollary of Theorem IV.43 and the next theorem.

**Theorem IV.45.** *Let $\mathcal{S}$ be a class of $L$-structures and $\mathcal{T}$ be the complement of $\mathcal{S}$, namely $\mathcal{T}$ is the class of $L$-structures which are not in $\mathcal{S}$. Suppose both $\mathcal{S}$ and $\mathcal{T}$ are* $\mathrm{EC}_\Delta$*. Then both $\mathcal{S}$ and $\mathcal{T}$ are* EC*.*

[7] It is natural to assume the presence of the equality sign, =, when talking about the finiteness of models. However, with some difficulty, the results on the undefinability of finiteness can be reformulated to apply to languages without equality.

*Proof.* Suppose $\mathcal{S} = \operatorname{Mod}\Gamma$ and $\mathcal{T} = \operatorname{Mod}\Pi$; in other words, $\mathfrak{A} \in \mathcal{S}$ if and only if $\mathfrak{A} \vDash \Gamma$, and $\mathfrak{A} \in \mathcal{T}$ if and only if $\mathfrak{A} \vDash \Pi$. Since $\mathcal{S} \cap \mathcal{T}$ is empty, $\Gamma \cup \Pi$ is inconsistent. By the Compactness Theorem, there is a finite subset of $\Gamma \cup \Pi$ which is inconsistent. Express this inconsistent subset as $\Gamma' \cup \Pi'$ where $\Gamma' \subseteq \Gamma$ and $\Pi' \subseteq \Pi$. Let $A$ be the sentence $\bigwedge \Gamma'$ and $B$ be the sentence $\bigwedge \Pi'$.

Suppose $\mathfrak{A} \vDash A$, or equivalently that $\mathfrak{A} \vDash \Gamma'$. By the unsatisfiability of $\Gamma' \cup \Pi'$, we have $\mathfrak{A} \nvDash \Pi$. Therefore $\mathfrak{A} \notin \mathcal{T}$; hence $\mathfrak{A} \in \mathcal{S}$. On the hand, suppose $\mathfrak{A} \nvDash A$. Then $\mathfrak{A} \nvDash \Gamma$, so $\mathfrak{A} \notin \mathcal{S}$. This shows that $\mathfrak{A} \vDash A$ if and only $\mathfrak{A} \in \mathcal{S}$. In other words, $\mathcal{S} = \operatorname{Mod} A$. A similar argument shows $\mathcal{T} = \operatorname{Mod} B$. Therefore, $\mathcal{S}$ and $\mathcal{T}$ are elementary classes (EC). □

The next theorem states that a theory with arbitrarily large models has an infinite model. This is often called an "overspill" theorem because it states that a property about arbitrarily large finite cardinalities spills over to infinite cardinalities. It will come up again later as a form of the Löwenheim-Skolem in Section IV.6.

**Theorem IV.46.** *Let $\Gamma$ be a set of sentences and suppose that, for every $k \in \mathbb{N}$, $\Gamma$ has a model of cardinality $\geq k$. Then $\Gamma$ has an infinite model.*

*Proof.* Let $\Pi$ be the set of sentences $\Gamma \cup \{AtLeast_k : k \geq 1\}$. Since $\Gamma$ has models of cardinality $\geq k$ for all $k$, $\Pi$ is finitely satisfiable. Hence, by the Compactness Theorem, $\Pi$ is satisfiable. Any model of $\Pi$ is a model of $\Gamma$ of course. In addition, it must have cardinality $\geq k$ for all integers $k$. Hence it must be an infinite model of $\Gamma$. □

**A nonstandard model of the theory of the integers.** The theory of the integers is equal to $\operatorname{Th}\mathcal{N}$, namely the set of sentences true in $\mathcal{N} = (\mathbb{N}, 0, S, +, \cdot)$. By the remark after Definition III.99, $\operatorname{Th}\mathcal{N}$ is complete. By definition, it fully describes all first-order truths of the integers. Nonetheless, it does not uniquely characterize the integers.

**Theorem IV.47.** *Let $L_{\mathrm{PA}} = \{0, S, +, \cdot\}$ be the language of $\mathcal{N}$. There is an $L_{\mathrm{PA}}$-structure $\mathfrak{A}$ which is elementarily equivalent to $\mathcal{N}$ but not isomorphic to $\mathcal{N}$.*

Such a structure $\mathfrak{A}$ is called a *nonstandard model* of the integers. Recall that $\mathfrak{A}$ and $\mathcal{N}$ being elementarily equivalent means that $\operatorname{Th}\mathfrak{A} = \operatorname{Th}\mathcal{N}$. We haven't yet written out the precise definition of isomorphic (for this see Definition IV.53), but it means what one would expect; namely, that there is a bijection of the two universes that preserves the meanings of the constants, predicates, and functions.

*Proof.* Let $c$ be a new constant symbol. Let $L' = L_{\mathrm{PA}} \cup \{c\}$. For each $i \in \mathbb{N}$, let $\underline{i}$ denote the term $S(S(\cdots S(S(0))\cdots))$ with $i$ occurrences of $S$. For instance, $\underline{0}$ is the term 0, and $\underline{1}$ is the term $S(0)$, and $\underline{2}$ is the term $S(S(0))$, etc. Thus $\underline{i}$ is a closed term, and in $\mathcal{N}$ it denotes the integer $i$. In other words, $\underline{i}^{\mathcal{N}}$ is equal to $i$. The terms $\underline{i}$ are called *numerals*.

Let $\Gamma$ be the set of sentences $\mathrm{Th}\,\mathcal{N} \cup \{c \neq \underline{i} : i \geq 0\}$. It is not hard to see that every finite subset of $\Gamma$ is consistent. To see this, consider the set of sentences $\Gamma_n$ defined by $\Gamma_n = \mathrm{Th}\,\mathcal{N} \cup \{c \neq \underline{i} : 0 \leq i < n\}$. Then $\Gamma_n$ is satisfiable, since we can expand $\mathcal{N}$ to the language $L'$ by setting the interpretation $c^{\mathcal{N}}$ equal to the object $n \in \mathbb{N}$. Every finite subset of $\Gamma$ is a subset of some $\Gamma_n$; hence $\Gamma$ is finitely satisfiable. By compactness, $\Gamma$ is also satisfied by some structure $\mathfrak{A}$.

By construction, $\mathfrak{A} \models \mathrm{Th}\,\mathcal{N}$. Let $\mathfrak{B}$ be the restriction of $\mathfrak{A}$ to the langauge $L_{\mathrm{PA}}$. By Theorem III.87, $\mathfrak{B} \models \mathrm{Th}\,\mathcal{N}$. In $|\mathfrak{B}| = |\mathfrak{A}|$, there is an object, namely $c^{\mathfrak{A}}$ that is distinct from $\underline{i}^{\mathfrak{A}}$ for all $i \in \mathbb{N}$. However, in $\mathcal{N}$, every object is equal to some $\underline{i}^{\mathcal{N}}$. (Note that $\underline{i}^{\mathcal{N}}$ is the same as $i$.) Therefore $\mathcal{N}$ and $\mathfrak{B}$ are not isomorphic. This is because any isomorphism $\pi : \mathcal{N} \to \mathfrak{B}$ has to map each $i = i^{\mathcal{N}}$ to $\underline{i}^{\mathfrak{A}} = \underline{i}^{\mathfrak{B}}$, but then $c^{\mathfrak{A}}$ is not in the range of $\pi$. □

The existence of a nonstandard model of the integers means that it is not possible to give a set of axioms that are true for the integers and false for any other (nonisomorphic) structure. This foreshadows the Gödel Incompleteness Theorems, which state that there is no effective way to give a set of axioms that imply all statements true in the integers. That, however, is a much deeper and more important result than the existence of nonstandard models; it will be stated properly and proved in subsequent chapters.

**Torsion-free groups.** The class of torsion-free groups was earlier shown to $\mathrm{EC}_\Delta$ as the theory of groups augmented with the axioms

$$T_k \;:=\; \forall x(x \neq 1 \to x^k \neq 1)$$

axiomatizes the torsion-free groups.

**Theorem IV.48.** *The class of torsion-free groups is not an elementary class.*

*Proof.* Suppose the class of torsion-free groups is EC, so it is equal to $\mathrm{Mod}\,A$ for some sentence $A$. Let $\Gamma$ be set containing the three group axioms and the sentences $T_k$. Then, $A \models \Gamma$ and $\Gamma \models A$. By compactness, there is a finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models A$. Let the $T_k$'s that appear in $\Gamma_0$ be $T_{k_1}, \ldots T_{k_n}$. Choose a prime $p$ that does not divide any of $k_1, \ldots k_n$, and let $\mathfrak{A}$ be a cyclic group of order $p$. Then $\mathfrak{A} \models \Gamma_0$, but since it is not torsion-free, $\mathfrak{A} \not\models A$. This is a contradiction. □

## IV.6 Cardinalities and Löwenheim-Skolem Theorems

Theorem IV.46 already established the overspill property that a theory with arbitrarily large finite models has an infinite model. This section will prove further properties about the cardinalities of models of a theory. We discuss countable languages and models. We do not give full proofs for the theorems

on uncountable languages and models; these results are not used in the rest of the text and readers can skip them if they wish.

We assume the reader has some basic knowledge of cardinalities in set theory. To simplify working with cardinalities, we assume that the Axiom of Choice holds. We write $|X|$ for the cardinality of a set $X$. Any set $X$ can be categorized in exactly one of the following ways:

(a) $X$ is *finite*. That is $|X| \in \mathbb{N}$.

(b) $X$ is *countably infinite*. That is, $X$ is equinumerous with $\mathbb{N}$.

(c) $X$ is *uncountable*. That is, $|X| > |\mathbb{N}|$.

A set is called *countable* if it is either finite or countably infinite.

**Theorem IV.49.** *Let $\Gamma$ be a consistent set of $L$-sentences where $L$ is countable. Then $\Gamma$ has a countable model.*

*Proof.* This is a direct consequence of the proof of the Completeness Theorem. That proof added countable many constant symbols $d_i$ to form a countably infinite language $L^+$ (when constructing a strongly Henkin extension of $\Gamma$). It then formed an $L^+$-structure $\mathfrak{A}$ with domain consisting of the equivalence classes $[s]$ where $s$ is a closed $L^+$-term. Since $L^+$ is countable, there are countably many closed $L^+$-terms. Thus there are countably many equivalence classes $[s]$. Therefore, $|\mathfrak{A}|$ is countable, so $\mathfrak{A}$ is countable. □

This last theorem has some simple, but surprising corollaries. The first states that there is a countable structure that is elementarily equivalent to the real numbers $\mathcal{R} = (R, 0, 1, +, \cdot)$.

**Corollary IV.50.** *There is a countable model of* $\operatorname{Th}\mathcal{R}$.

This means that first-order logic is unable to fully characterize the real numbers since it cannot exclude the existence of a countable model. The same holds for ZF, the first-order Zermelo-Fraenkel set theory. This is often called the "Skolem paradox" (because it was first proved by Skolem in 1922):

**Corollary IV.51.** *If* ZF *is consistent, there is a countable model of* ZF.

The Skolem paradox is not an actual paradox, but it superficially seems paradoxical since the theory ZF can prove the existence of uncountable sets, but still has a countable model. The resolution of the "paradox" is that in any countable model $\mathfrak{A}$ of ZF, there will be an object $a$ that satisfies (relative to truth in $\mathfrak{A}$) the first-order property of being an infinite set. Specifically, an object $a$ in $|\mathfrak{A}|$ is uncountable in the sense of $\mathfrak{A}$ if there is no bijection in $\mathfrak{A}$ between $a$ and the interpretation of the natural numbers in $\mathfrak{A}$.

By the proofs of Theorems IV.47 and IV.49, we also get:

**Corollary IV.52.** *There is a countable nonstandard model of the integers.*

We define next the notion of countable categoricity. For this, two models are said to be isomorphic if they are identical up to renaming the objects in the universe. Formally this is defined by:

**Definition IV.53.** Two $L$-structures $\mathfrak{A}$ and $\mathfrak{B}$ are *isomorphic*, written $\mathfrak{A} \cong \mathfrak{B}$, if there is a bijection $\pi : |\mathfrak{A}| \to |\mathfrak{B}|$ such that

(a) For every constant symbol $c \in L$, $\pi(c^{\mathfrak{A}}) = c^{\mathfrak{B}}$.

(b) For every $k$-ary function symbol $f \in L$ and every $a_1, \ldots, a_k \in |\mathfrak{A}|$,

$$\pi(f^{\mathfrak{A}}(a_1, \ldots, a_k)) \;=\; f^{\mathfrak{B}}(\pi(a_1), \ldots, \pi(a_k)).$$

(c) For every $k$-ary predicate symbol $P \in L$, and every $a_1, \ldots, a_k \in |\mathfrak{A}|$,

$$\langle a_1, \ldots, a_k \rangle \in P^{\mathfrak{A}} \quad \text{if and only if} \quad \langle \pi(a_1), \ldots, \pi(a_k) \rangle \in P^{\mathfrak{B}}.$$

Recall that the notation $\mathfrak{A} \equiv \mathfrak{B}$ indicates that $\mathfrak{A}$ and $\mathfrak{B}$ are elementarily equivalent.

**Theorem IV.54.** *If $\mathfrak{A} \cong \mathfrak{B}$, then $\mathfrak{A} \equiv \mathfrak{B}$.*

The idea behind Theorem IV.54 is clear: if $\mathfrak{A}$ and $\mathfrak{B}$ are identical except for the identities of objects in their universes, then they satisfy the same first-order properties. We leave the proof to Exercise IV.23.

For the next definition, $\aleph_0$ is used to denote the cardinality of $\mathbb{N}$. ("$\aleph$" is the Hebrew letter "aleph".)

**Definition IV.55.** A set of sentences $\Gamma$ is $\aleph_0$-*categorical* if $\Gamma$ has exactly one countably infinite model up to isomorphism.

The terminology "$\omega$-categorical" is often used instead of "$\aleph_0$-categorical".

**Theorem IV.56** (Countable Łoś-Vaught Test)**.** *Suppose $L$ is countable. Also suppose that $T$ is an $\aleph_0$-categorial $L$-theory and that $T$ has no finite models. Then $T$ is complete.*

*Proof.* Suppose $T$ is not complete, so there is a sentence $A$ such that neither $A$ nor $\neg A$ is a consequence of $T$. Therefore $T \cup \{A\}$ and $T \cup \{\neg A\}$ are both consistent. By Theorem IV.49, there are countable models $\mathfrak{A}$ and $\mathfrak{B}$ of $T \cup \{A\}$ and $T \cup \{\neg A\}$. Since $T$ has no finite models, $\mathfrak{A}$ and $\mathfrak{B}$ are countably infinite. Since $T$ is $\aleph_0$-categorical, they are isomorphic. But in light of Theorem IV.54, this is a contradiction since they disagree on the truth of $A$. □

**Example IV.57.** The theory of dense linear order (DLO) without endpoints is an example of an $\aleph_0$-categorial theory. This was originally proved by Cantor in the 1890's, but we shall omit the proof. It follows that the theory of DLO without endpoints is complete.

On the other hand, the theory DLO is not complete since the axiom about the existence of a left (say) endpoint is neither provable nor refutable from the DLO axioms. It is also not $\aleph_0$-categorical. For instance, let $\mathbb{Q}^{\geq 0}$ be the set of nonnegative rationals and $\mathcal{Q}^{\geq 0}$ be the structure $(\mathbb{Q}^{\geq 0}, <)$. Then $\mathcal{Q}$ and $\mathcal{Q}^{\geq 0}$ are countable models of DLO but are not isomorphic.

**Example IV.58.** The theory $\mathrm{Th}\,\mathcal{N}$ of the integers is complete but not $\aleph_0$-categorical by Corollary IV.52.

The Countable Łos-Vaught Test can also be stated for a set $\Gamma$ of sentences. Let $\mathrm{Cn}\,\Gamma$ denote the set of sentences which are logical consequences of $\Gamma$, namely $\mathrm{Cn}\,\Gamma = \{A : A \text{ is a sentence and } \Gamma \vDash A\}$. Note $\mathrm{Cn}\,\Gamma$ must be a theory. Theorem IV.56 implies that if $\Gamma$ is $\aleph_0$-categorical and has no finite models, then $\mathrm{Cn}\,\Gamma$ is complete.

We now discuss uncountable languages.

**Definition IV.59.** Let $L$ be a language. The *cardinality* of $L$ is denoted $card(L)$ and is equal to the maximum of $|L|$ and $\aleph_0$.

Equivalently, $card(L) = \aleph_0$ if $L$ is countable, and $card(L) = |L|$ if $L$ is infinite. The point of the definition of $card(L)$ is that there are a total of $card(L)$ many formulas.

The Soundness, Completeness and Compactness Theorems all hold for uncountable languages. The proof of the Completeness Theorem needs modification to apply to uncountable languages: the primary modification is in the proof of Theorem IV.29 constructing a strongly Henkin extension of $\Gamma$. See Exercise IV.29 for an outline of the proof.

The proof as sketched in Exercise IV.29 proves a strengthed form of the Completeness Theorem for languages of arbitrary cardinality:

**Theorem IV.60** (Completeness Theorem)**.** *Let $\Gamma$ be a consistent set of $L$-sentences. Then $\Gamma$ has a model of cardinality at most $card(L)$.*

As a consequence, we get the following result, which is a variant of the Löwenheim-Skolem theorem.

**Theorem IV.61.** *Let $\Gamma$ be a set of $L$-sentences. Suppose that $\Gamma$ has an infinite model or that $\Gamma$ has arbitrarily large finite models. Then for all cardinals $\lambda \geq card(L)$, $\Gamma$ has a model of cardinality $\lambda$.*

By Theoren IV.46, if $\Gamma$ has arbitrarily large models, then $\Gamma$ has an infinite model. So the last part of the hypothesis is redundant.

*Proof.* Let $\kappa = card(L)$ and $\lambda \geq \kappa$. Let $\{d_\alpha : \alpha < \lambda\}$ be a set of $\lambda$ many new constant symbols.[8] Let $L'$ be $L \cup \{d_\alpha : \alpha < \lambda\}$. Clearly $card(L') = \lambda$.

Let $\Pi$ be $\Gamma \cup \{d_\alpha \neq d_\beta : \alpha < \beta < \lambda\}$. We claim that $\Pi$ is finitely satisfiable. Any finite subset $\Delta$ of $\Pi$ can mention only $k$ many $d_\alpha$'s for some finite $k$. Since $\Gamma$ has a model $\mathfrak{A}$ of cardinality $\geq k$, this means $\Delta$ can be satisfied by expanding $\mathfrak{A}$ to have distinct interpretations for the $d_\alpha$'s mentioned in $\Delta$. Since every finite subset $\Delta$ of $\Pi$ is satisfiable, $\Pi$ is consistent. Therefore, by the Completeness Theorem IV.61, $\Pi$ is satisfied by some structure $\mathfrak{A}$ of cardinality at most $\lambda$. Since $\Pi$ contains the sentences $d_\alpha \neq d_\beta$, any model of $\Pi$ must have cardinality at least $\lambda$. Since $\Pi \supset \Gamma$, it follows that $\mathfrak{A}$ is the desired model of $\Gamma$ of cardinality $\lambda$. □

**Corollary IV.62.** *There is an uncountable nonstandard model of the integers.*

[8] The definition of cardinals in set theory ensures that for any cardinal $\lambda$, there are there are $\lambda$ many cardinals $\alpha < \lambda$.

**Definition IV.63.** Let $\kappa$ be a cardinality. A set of sentences $\Gamma$ is *$\kappa$-categorical* if $\Gamma$ has exactly one model of cardinality $\kappa$ up to isomorphism.

**Theorem IV.64** (Łoś-Vaught Test)**.** *Suppose that $\lambda \geq card(L)$. Also suppose that $T$ is an $\lambda$-categorial $L$-theory and that $T$ has no finite models. Then $T$ is complete.*

*Proof.* The proof of this is essentially identical to the proof of the Countable Łoś-Vaught Test in Theorem IV.56. Suppose $T$ is not complete, so there is a sentence $A$ such that neither $A$ nor $\neg A$ is a consequence of $T$. Therefore $T \cup \{A\}$ and $T \cup \{\neg A\}$ are both consistent. By Theorem IV.60, there are models $\mathfrak{A}$ and $\mathfrak{B}$ of $T \cup \{A\}$ and $T \cup \{\neg A\}$. Since $T$ has no finite models, $\mathfrak{A}$ and $\mathfrak{B}$ are both infinite. Thus by Theorem IV.64, there are models $\mathfrak{A}'$ and $\mathfrak{B}'$ of $T \cup \{A\}$ and $T \cup \{\neg A\}$ of cardinality $\lambda$. Since $T$ is $\lambda$-categorical, $\mathfrak{A}'$ and $\mathfrak{B}'$ are isomorphic. But in light of Theorem IV.54, this is a contradiction since they disagree on the truth of $A$. □

## Exercises

Your answers to Exercises IV.1-IV.9 should not use the Completeness Theorem.

**Exercise IV.1.** Suppose that $y$ and $t$ are each substitutable for $x$ in $A$.

(a) Show that $\forall x\, A \vdash A(t/x)$ by giving an explicit FO-proof of $A(t/x)$ from $\forall x\, A$. (An "explicit" proof means writing out all the formulas in the proof, indicating if they are a hypothesis, if they are an axiom or if they are inferred by Modus Ponens or Generalization.)

(b) Show that $\forall x\, A \vdash \forall y\, A(y/x)$ by giving an explicit proof of $A(y/x)$ from $\forall x\, A$.

**Exercise IV.2.** Let $P$ and $Q$ be unary predicate symbols, and 0 be a constant symbol. Give an explicit FO proof (by listing all the formulas in the proof) of

$$\forall x\,(P(x) \to Q(x)),\ \forall y\, P(y) \ \vdash\ P(0) \to \forall z\, Q(z).$$

**Exercise IV.3.** Show that $\{\forall x\, \exists y\, P(x,y),\ \neg P(c,z)\}$ is inconsistent by giving an explicit FO proof (or explicit FO proofs).

**Exercise IV.4.** Prove the following:

(a) $\vdash f(x) = f(x)$.

(b) $\vdash f(x) = f(y) \to f(y) = f(x)$.

(c) $\vdash x = y \to u = v \to g(f(x), f(u)) = g(f(y), f(v))$.

**Exercise IV.5.** Prove that $\vdash \forall x\, A \to \exists x\, A$.

**Exercise IV.6.** Suppose that $\forall x\, A \to C$ is a sentence. In other words, $x$ does not occur free in $C$ and $x$ is the only variable which appears free in $A$. Prove:

(a) $\forall x\,(A \to C) \vdash \exists x\, A \to C$.

(b) $\exists x\, A \to C \vdash \forall x\,(A \to C)$.

**Exercise IV.7.** Redo the previous exercise (Exercise IV.6) under only the assumption that $x$ does not occur free in $C$ (but without any assumptions on what other variables might appear free in $A$ or $C$).

**Exercise IV.8.** Prove that $\exists y\, \forall x\, Q(x,y) \vdash \forall x\, \exists y\, Q(x,y)$. [Hint: There are several ways to work this; you might find the Theorem on Constants to be helpful.]

**Exercise IV.9.** Let $P$ and $Q$ be unary predicate symbols.
(a) Prove that $\forall x\, (P(x) \to Q(x)) \to \exists x\, P(x) \to \exists x\, Q(x)$.
(b) Prove that $\vdash \forall x\, \forall y\, (P(x) \to Q(y)) \to \exists x\, P(x) \to \forall y\, Q(y)$.
(c) Prove that $\vdash \exists x\, \forall y\, (P(y) \to P(x))$.

**Exercise IV.10.** Theorem IV.13 stating the Deduction Theorem for FO required $A$ to be a sentence. Show that this hypothesis cannot be eliminated by giving an example of formulas $A$ and $B$ such that $A \vdash B$ but $\nvdash A \to B$. (You can use the Soundness Theorem to prove $\nvdash A \to B$.)

**Exercise IV.11.** Theorem IV.18 on proof-by-cases for FO required $A$ to be a sentence. Shown that this hypothesis cannot be eliminated by giving an example of formulas $A$ and $B$ such that both $A \vdash B$ and $\neg A \vdash B$ hold and such that $\nvdash B$.

**Exercise IV.12.** Theorem IV.17 on Proof by Contradiction for FO required $A$ to be a sentence. Shown that this hypothesis cannot be eliminated by giving an example of a formula $A$ such that $\{A\}$ is inconsistent, but $\nvdash \neg A$.

**Exercise IV.13.** Part (b.ii) of the Theorem on Constants IV.20 required $x$ to be the only free variable in $A(x)$. Prove that this hypothesis cannot be eliminated by giving an example of a formula $A(x,y)$ such that $\{\exists x\, A(x,y)\}$ is consistent but $\{A(c,y)\}$ is inconsistent,

**Exercise IV.14.** Suppose that $\Gamma$ and $\Delta$ are sets of formulas and $\Gamma \cup \Delta$ is unsatisfiable.
(a) Prove that there is a finite $\Gamma' \subseteq \Gamma$ and a finite $\Delta' \subseteq \Delta$ such that $\Gamma' \cup \Delta'$ is unsatisfiable.
(b) Prove that there is a formula $A$ such that $\Gamma \vDash A$ and $\Delta \vDash \neg A$.
This has the identical statement (and the identical proof) as Exercise II.25 but of course applies to first-order logic instead of propositional logic.

**Exercise IV.15.** Suppose $\Gamma$ is a set of sentences and $\Delta$ is a finite set of sentences such that $\Gamma \vDash\!\dashv \Delta$. Prove that there is a finite subset $\Gamma'$ of $\Gamma$ such that $\Gamma' \vDash\!\dashv \Gamma$.

**Exercise IV.16.** Let $k_0 \in \mathbb{N}$. Given an example of a theory that has models of cardinality $k_0$ and models of infinite cardinality, but does not have models of any finite cardinality other than $k_0$.

**Exercise IV.17.** Let $A$ be a sentence and $\Gamma$ be a set of sentences. Suppose that $A$ is true in every infinite model of $\Gamma$. Prove that there is a $k \in \mathbb{N}$ such that $A$ is true in every model of $\Gamma$ of cardinality $\geq k$.

**Exercise IV.18.** Work in the language $L$ with the symbol $=$ and no non-logical symbols. Prove that there is no finite set $\Gamma$ of $L$-sentences which satisfies the "even cardinality finite models" property of Exercise III.13.

**Exercise IV.19.** Prove that there is no formula $A(x, y)$ that defines, in every graph, the property of $x$ and $y$ being in the same connected component. For the purposes of this exercise, such a formula $A$ would need to work in all graphs whether finite or infinite.

**Exercise IV.20.** Let $T$ be the theory of linear orders from Example III.92 using the language $L = \{<\}$. A model $\mathfrak{A}$ of $T$ is called *well-founded* if there is no sequence $a_0, a_1, a_2, \ldots$ in $|\mathfrak{A}|$ such that $a_{i+1} <^{\mathfrak{A}} a_i$ holds for all $i \in \mathbb{N}$. Show that there is no set $\Gamma$ of sentences over $T$ that expresses the property of being well-founded. That is, there is no set $\Gamma$ of sentences such that for all models $\mathfrak{A}$ of $T$, we have $\mathfrak{A} \models \Gamma$ if and only if $\mathfrak{A}$ is well-founded.

**Exercise IV.21.** (For readers who know some field theory. Compare with Exercise III.29.) Prove that the class of the fields of finite (non-zero) characteristic is not an $\mathrm{EC}_\Delta$. That is, there is no theory $T$ in the language $0, 1, +, \cdot$ which is true for exactly the fields of finite (non-zero) characteristic.

**Exercise IV.22.** Let $c$ be the cardinality of $\mathbb{R}$ (the "continuum"), Use the following sketch to prove that there is a nonstandard model of $\operatorname{Th}\mathcal{R}$ of cardinality $c$. Thus $\operatorname{Th}\mathcal{R}$ is not $c$-categorical. (If fact, $\operatorname{Th}\mathcal{R}$ is not $\kappa$-categorical for any infinite $\kappa$.)

(a) Recall that $L = (0, 1, +, \cdot, <)$ be the language of $\mathcal{R}$. Let the numeral $n$ be the term $1 + 1 + \cdots + 1$ with value $n \in \mathbb{N}$. An ordered field is *archimedean* if every field element $x > 0$ satisfies $x < n$ for some $n \in \aleph$.
(b) Prove that $\operatorname{Th}\mathcal{R}$ has a model $\mathfrak{A}$ which is a non-archimedean field of cardinality $c$.
(c) Prove that $\mathfrak{A}$ is not isomorphic to $\mathcal{R}$.
(d) Conclude that $\operatorname{Th} R$ is not $c$-categorical.

**Exercise IV.23.** Prove Theorem IV.54 that isomorphic structures are elementarily equivalent. The proof will assume that $\mathfrak{A} \cong \mathfrak{B}$ and use induction on the complexity of $A$ to prove that, for all $\sigma$, $\mathfrak{A} \models A[\sigma]$ holds if and only if $\mathfrak{B} \models A[\sigma]$ holds.

**Exercise IV.24.** Let $L$ be the language containing only the equality symbol $=$. Let $\Gamma$ be $\{AtLeast_k : k \geq 2\}$. Prove that $\Gamma$ is $\aleph_0$-categorical and complete.

**Exercise IV.25.** Let $L$ be the language with a unary predicate symbol $P$ (and, as always, equality). Let $\Gamma$ be a set of sentences expressing that there are infinitely many objects $x$ satisfying $P(x)$ and infinitely many objects $x$ satisfying $\neg P(x)$.

(a) Describe explicitly the formulas in $\Gamma$.
(b) Show that $\Gamma$ is $\aleph_0$-categorial. Conclude that the theory axiomatized by $\Gamma$ is complete.
(c) Show that $\Gamma$ is not $\kappa$-categorical for $\kappa > \aleph_0$.

**Exercise IV.26.** Give an example of a theory that is $\kappa$-categorical for all infinite $\kappa$.

**Exercise IV.27★** A theory is called *categorical* if all models of $T$ are isomorphic.

(a) Describe what must happen if $T$ is categorical.

(b) Suppose $T$ is a categorical theory over a finite language $L$. Prove that $T$ is finitely axiomatizable, that is, that there is a finite set of sentences $\Gamma$ such that $\Gamma \models\!\dashv T$.

**Exercise IV.28.** Give an example of how Theorem IV.56 (the Łoś-Vaught Test) can fail if the hypothesis that $T$ has no finite models is omitted.

**Exercise IV.29★** (For readers who know some set theory.) Use the following proof outline to prove the Completeness Theorem IV.60 for uncountable languages. (Compare this to Exercise II.30.) Let $\Gamma$ be a consistent set of $L$-sentences. Let $\kappa = \mathit{card}(L)$ be the cardinality of $L$ and suppose $\kappa > \aleph_0$.

(a) There are $\kappa$ many distinct $L$-formulas.

(b) Extend $\Gamma$ to a strongly Henkin set $\Delta$ of sentences. Let $D$ be a set of $\kappa$ many new constant symbols, with $D$ the disjoint union $D = \bigcup_{i\in\mathbb{N}} D_i$ and each $D_i$ of cardinality $\kappa$. Let $L_i = L \cup \bigcup_{i<j} D_i$ and $L^+ = \bigcup_i L_i$. There is the same number of $L_i$-sentences as there are constants in $D_{i+1}$. Therefore each $L_i$-sentence $A$ can be associated with a distinct constant $d_A \in D_{i+1}$, so $A \mapsto d_A$ is injective. Let $\Delta$ be the set $\Gamma$ plus the sentences $A(d_{\forall x A(x)}) \to \forall x A(x)$ for all $L^+$-sentences $\forall x A(x)$. Clearly $\Delta$ is strongly Henkin. Prove that $\Delta$ is consistent. For this, use induction on $i$ and the finiteness of proofs.

(c) Use Zorn's Lemma to prove there is a complete, consistent set $\Pi$ of sentences with $\Pi \supset \Delta$.

(d) Use the same constructions as in proof for the countable version of the Completeness Theorem to prove that there is a Henkin model $\mathfrak{A}$ of $\Pi$. Thus $\mathfrak{A}$ (and its restriction to the language $L$) satisfies $\Gamma$.

(e) The universe of $\mathfrak{A}$ is equal to the interpretations of the constants in $D$. That is, $|A| = \{d^{\mathfrak{A}} : d \in D\}$. Therefore the cardinality of $\mathfrak{A}$ is at most $\kappa$.

# Chapter V

# Algorithms, Informally

The notions of an "algorithm" or 'effective 'procedure" or "computability" are central topics in mathematical logic. This chapter will deal with *informal* descriptions of algorithms and computability. It also discusses noncomputability, based on an informal formulation of the halting problem. As applications, this chapter discusses algorithms for recognizing properties of formulas and proofs in both propositional logic and first-order logic. Later chapters will study *formal* notions of computability, including Turing machines and recursive functions. The formal notions of computability will correspond extremely well with our informal notions of algorithms.

An informal notion of algorithm is perfectly adequate for proving results about things that *are* computable. However, it is also of great interest to understand what is *not* computable. And, a big advantage of working with the formal definitions of computability will be that it will let us prove theorems about what cannot be computed, e.g., the Halting Problem for Turing machines. This will further form the basis for results about the limits of provability in first-order logic, in the form of Gödel's Incompleteness Theorems.

First, however, the present chapter discusses algorithms, and computability and non-computability, based on intuitive, informal definitions of these concepts.

## V.1 Informal Definition of Algorithms

An algorithm is meant to be a precisely described procedure or method for solving a problem. One way to think of algorithms is that they are procedures that could be carried by an idealized computer. In particular, an algorithm has a finite set of instructions, and at any given instant in time, the algorithm is working with only a finite amount of data. Working on an "idealized" computer, however, means that the algorithm is not constrained by practical considerations such as the amount of memory available, the power that is consumed, the time that is used, etc.

Generally, an algorithm accepts as input a string of symbols from some

finite alphabet $\Sigma$ of symbols. The output of the algorithm can be either a "Yes"/"No" answer or another string of symbols over $\Sigma$. Of course, we are also interested in algorithms that deal with objects such as integers and with general combinatorial objects, but these can be encoded by strings of symbols. For instance, integers can be represented by strings of 0's and 1's in binary notation, or strings of digits 0-9 in decimal notation, etc. Likewise, a combinatorial object such as a graph can be encoded by a string of symbols. Thus there is no real loss of generality in requiring algorithms to work only with symbols from a finite alphabet $\Sigma$.

The informal notion of an algorithm can be captured by the following principles.

- The algorithm is specified by a finite set of instructions that fully and unambiguously specify the actions of the algorithm.
- The algorithm operates in a step-by-step fashion. At each step, it uses or manipulates only a finite amount of data. The data is discrete or combinatorial in nature, of the type that can be encoded by symbols drawn from a finite alphabet of symbols.
- The input to the algorithm is encoded by a string of symbols from the finite alphabet. The algorithm returns an answer after running for a finite number of steps and then entering a special "output state" that indicates the answer is ready. The answer is encoded in a straightforward way as a string of symbols in the data as maintained by the algorithm. After outputting an answer, the algorithm can continue where it left off, and potentially output further answers. Alternatively, the algorithm could run forever and never return a single answer.

All of these requirements are met by a modern-day computer program. In practice, a computer's memory is organized into words and pages, and stored in a hierarchy of different memory types (cache memory, main memory, internal disk, external disk, etc.); however, in the end, the entire contents of a computer's memory can be viewed as a binary string of 0's and 1's. A computer program is a finite list of instructions, and at any given instance in time the computer's next action is based on only a finite number of bits. And, indeed we will argue that idealized computer programs, namely algorithms that are not constrained by time or space requirements, exactly capture our informal notion of algorithm.

It is important for the informal notion of algorithm that the instructions "fully and unambiguously" specify the actions of the algorithm while only taking a finite amount of data into account. This rules out things like the use of randomization or quantum computing. It also rules out appeals to human intuition or human judgment. An example of a disallowed appeal to human judgment would be an algorithm that is programmed to form a digital image, and then take one action if the resulting image is "attractive" and another action if the image is "unattractive". The problem, of course, is that this is not an unambiguous decision.

Randomization however can, in many cases, be simulated by algorithms that do not use randomization. Whether quantum computing or other (possibly yet-to-be-discovered) physical processes could extend the power of algorithms is unknown.

A more subtle limitation of algorithms is that they cannot work directly with real numbers. Consider, for instance, the situation where it is wished for an algorithm to decide whether $\pi$ is a root of the polynomial $9x^4 - 240x^2 + 1492$.[1] Or more generally, whether a given real number is a root of a given polynomial. This cannot be done directly in a single step, because a real number can be non-rational, and may not be expressible as a finite amount of data. Indeed, although special real numbers such $\pi$ can be described succinctly, an arbitrary real number cannot be "given" by a finite description. The best that can be done is to give the algorithm a finite description of the real number, for instance in the form of an algorithm that generates its binary or decimal representation. This of course corresponds well to computation in ordinary computers, which likewise cannot work directly with (infinite precision) real numbers.

The limitation about using only a finite amount of data is even more restrictive than just prohibiting using an infinite amount of data all at once. It instead means that there is a fixed upper bound on the amount of data that can be used in a single algorithmic step: this upper bound is independent of the size of the input. For example, consider an algorithm that takes the binary representation of an integer $n$ as input, and is required to decide whether the input $n$ is a prime number. It is not permitted to just look at the entire number and see directly whether it is prime. Instead, the algorithm must break its work into smaller steps, say based on trial division, to determine whether $n$ is prime. Even things like adding or multiplying integers (presented, say, in binary notation) cannot be done in a single step. Instead, they need to be done step-by-style, perhaps by using grade school-style algorithms adapted to base 2 instead of base 10. This again corresponds well to how ordinary computers need to carry out computations for arbitrary precision arithmetic.

To illustrate this, consider the following problem. The input is a string of 0's and 1's. We want an algorithm to determine the parity of the string, namely if there is an even or odd number of 1's. Suppose the string is very long, consisting of millions or billions of symbols or even more. For the sake of concreteness, we can picture the string as being presented to us in a multi-volume collection of books filled with text containing just 0's or 1's, and then imagine how we would determine its parity by hand. It is even more useful to picture the string as being written on an extremely long ribbon of paper. Then a natural algorithm is to start at the beginning of the string and scan the entire string from left to right, keeping track at each step of whether an even number or an odd number of 1's has been encountered so far. This algorithm needs only to consider a finite amount of the input data at any given step, and remember a finite amount of information (namely the parity of the bits encountered so far).[2]

---

[1] This is the Kochański approximation to $\pi$: one of its roots is $\sqrt{40/3 - 2\sqrt{3}} \approx 3.1415333387$.

[2] The attentive reader will have noticed that the algorithm also has to remember its position

This example raises the question of the efficiency of algorithms, namely of how quickly they can be implemented in practical settings. The notions of algorithms and computability that are discussed in this book do not take efficiency into consideration at all. If an algorithm takes an enormous amount of time, even more time than could possibly be available within the lifespan of the universe, this does not disqualify it from being an "effective" algorithm. On the other hand, the efficiency of an algorithm is obviously of great importance in real-world applications. In theoretical computer science, the efficiency of algorithms is categorized in many ways, for instance, whether the algorithm runs in "linear" time, "polynomial" time, "exponential" time, etc. These notions all have precise mathematical definitions (which we will not present), but they are not important for our notion of an "effective" algorithm. Nonetheless, we take a short detour to discuss these concepts.

For our brief tour of the efficiency of algorithms, first, consider addition and multiplication of integers presented in binary notation. Suppose two integers are presented in base 2 as binary strings written, say, in a multi-volume sets of books, or on very long ribbons. The goal is to algorithmically determine their sum or their product, say with the end result written in its own multi-volume set of books or its own very long scroll. For integer addition, the grade school algorithm (adapted to base 2) is quite efficient. Indeed if things are arranged so that you can scan the inputs from right to left, namely starting at the low-order bits, and write the output in right-to-left order, then the sum can be written almost as fast as the two input summands can be scanned. This is a quite efficient algorithm and is an example of a "linear time" algorithm. The grade school algorithm for integer multiplication is less efficient; it is a "quadratic time" algorithm. There are algorithms for integer multiplication that are more efficient than quadratic time; however, the most efficient ones are quite complex. (It is an open question whether there is a linear time algorithm for integer multiplication.)

The primality testing problem is the problem of deciding whether an integer is a prime. For primality testing, the input is an integer, coded as binary string, and the output is a "Yes" or "No" answer depending on whether the input is a prime or not. The straightforward algorithm based on trial division is an "exponential time" algorithm, but there is a more sophisticated and complex algorithm for primality testing which uses only "polynomial time". The prime factorization problem is the problem of, given an integer $n$ as input, computing the prime factorization of $n$. The best-known algorithms for prime factorization require exponential time, and it is open whether it has a linear time algorithm or even a polynomial time algorithm.

An algorithm is informally called *feasible* if it is efficient enough to be car-

---

in the input string, and this not cannot be encoded by a fixed number of bits. This is unavoidable. A better way to state the requirement about using only a finite amount of information is that the algorithm has a finite set of pointers to finite sub-portions of the data, that it can only act on and alter data that is pointed to, and that the pointer values can be updated only in certain restricted (finitely describable) ways. In the Turing machine models, these pointers will be called "tape head positions".

ried out in practice. The feasibility of an algorithm is a rather vague concept. Roughly speaking, it means something along the lines of the following: For reasonably sized inputs, the algorithm can be successfully carried by a computer program that could be run in the real world. Being "run in the real world" could mean anything from running on a single computer, to running on a supercomputer, or to running on a network that somehow has co-opted a large fraction of the world's computers, and with the calculations being completed in, say, months. Being a "reasonably sized" input, could be anything from consisting of hundreds of symbols, to millions of symbols (e.g., fitting in a multivolume set of books), to trillions of symbols, or more.

The notion of "feasible" is thus very vague and depends on the application; it certainly lacks a mathematical definition. For convenience, many people identify being feasible with being a polynomial time algorithm, even though it is an imperfect identification. In most cases, everyone agrees that linear time algorithms should count as feasible and that exponential time algorithms are infeasible.[3] Quadratic-time algorithms may generally be considered as feasible, but this gets much less clear as the problem size increases and there are many situations where quadratic-time algorithms become infeasible.

In this book, we present the theory of "effective" algorithms, without taking into consideration whether they are feasible.

## V.2 The Church-Turing Thesis

The present chapter discusses algorithms in an informal way, without presenting a formal definition of "computable" or "algorithm". Later chapters will give three mathematical definitions of "computable": the first one is based on Turing machines, the second one is based on definability in theories of the integers, and the third definition is based on recursive function theory. The Church-Turing thesis states that our informal notion of "computable" corresponds precisely to the mathematically defined notions of "computable" as based on Turing machines or recursive function theory.

The Church-Turing thesis is often called "Church's Thesis" as it was first proposed by Church in 1936 for "lambda-definable" functions. Also in 1936 and working independently, Turing proposed a concrete model for computation, now known as Turing machines. A definition of computable functions was given a few years earlier in work by Gödel and Herbrand; however, they did not formulate a version of the Church-Turing thesis.

One reason for believing the Church-Turing thesis is that these three definitions of computability (lambda definability, Turing computable, and recursiveness) are mathematically equivalent. Indeed there are many other mathematically equivalent definitions of "computable". This indicates that these definitions define a robust notion of computability.

[3]But there are many exceptions to this, as this depends on the constant factors in the asymptotics.

| Generic terminology | Turing machines | Recursive function theory |
|---|---|---|
| Computable | Turing computable | Recursive |
| Decidable *(for relations only)* | Turing decidable | Recursive |
| Computably enumerable (c.e.) *(for relations only)* | Turing enumerable | Recursively enumerable (r.e.) |
| Partial computable *(for functions only)* | Partial Turing computable | Partial recursive |

Figure V.1: Terminology for computability

A second, even better reason for believing the Church-Turing thesis is that Turing machines, for all their simplicity, can carry out a broad range of algorithms. As convincingly argued by Turing in his original paper on the topic, any effective procedure, of the type that people can agree is really effective, can be carried out by a Turing machine. This will be discussed more in the next chapter.

The Church-Turing thesis has become to be seen as being "obviously true", based on the fact that a Turing machine can simulate the kinds of operations carried out by modern-day computers. This still leaves open the possibility some new physical principle might be discovered (say, based on quantum mechanics or alternate universes or who knows what) that permits computations that cannot be carried out by Turing machines. However, unless or until such things happen, we can unquestioningly accept the Church-Turing thesis as being true.

An important (nonobvious) consequence of the Church-Turing thesis is that it implies there is a "universal algorithm" that can simulate any other algorithm.[4] The existence of universal algorithms makes it possible to write general-purpose compilers and interpreters, which can compile or interpret any program. In theoretical computer science, it makes it possible to study "meta-algorithms", which are algorithms that take algorithms as inputs.

Figure V.1 summaries some of the terminology that will be used when describing computable relations and functions. In the present chapter, the terms "computable" and "decidable" are used in their informal sense. This is sufficient for proving results about what ***is*** computable. Later chapters will use terminology such as "computable" and "decidable" in their mathematically formal sense, as defined in terms of Turing machines or recursive function theory. Having mathematical definitions for computability is not only interesting in its own right; it also is needed in order to prove theorems about what **is not** computable.

[4]Turing [1936] was the first to describe a universal algorithm, in the same paper that introduced Turing machines.. Similar constructions were given by Church slightly earlier in 1935-1936.

# V.3 Basic Definitions for Computability

We now define various ways in which functions and relations can be computable. For now, we work with $k$-ary relations that take strings of symbols as inputs. Likewise, we work with functions that take strings of symbols as inputs and output a string of symbols.

However, Section V.3.2 will discuss how the various notions of computability also apply to functions and relations that act on integers; namely, by using a binary representation or base 10 representation of integers. Chapter VII covers the Incompleteness Theorems and representable relations and functions, and will work extensively with functions and relations that act on integers. Appendix **??** discusses the traditional definitions of recursive sets and functions.

## V.3.1 Computability and decidability

**Definition V.1.** An *alphabet* is a non-empty, finite set $\Sigma$ of *symbols*. The set $\Sigma^*$ is the set of strings of symbols from $\Sigma$. Equivalently, $\Sigma^*$ is the set of finite sequences of symbols from $\Sigma$. The *empty string* is denoted $\epsilon$; it is the string with no symbols. The *length* of a string $w$ is denoted $|w|$ and is the number of occurrences of symbols in $w$.

**Example V.2.** Let $\Sigma = \{0, 1\}$. There is a single string in $\Sigma^*$ of length 0, namely the empty string $\epsilon$. There are two strings in $\Sigma^*$ of length one, namely 0 and 1. There are four strings in $\Sigma^*$ of length two, namely 00, 01, 10, and 11.

We often use letters such as $a, b, c, \ldots$ or digits $0, 1, 2, \ldots$ to denote symbols in $\Sigma$. We often use letters such as $u, v, w, x, y, z, \ldots$ to denote strings. Strings are sometimes called "expressions" or "words". We fix an alphabet $\Sigma$ for the rest of this section.

This chapter, and Chapter VI, will primarily use the convention that functions and relations act on strings of symbols.

**Definition V.3.** Let $k \geq 1$. A $k$-ary function (on $\Sigma^*$) is a function

$$f : (\Sigma^*)^k \to \Sigma^*;$$

that is, $f$ takes $k$ strings as inputs and outputs a string.

**Definition V.4.** A $k$-ary relation $R$ (on $\Sigma^*$) is a subset of $(\Sigma^*)^k$. Members of $R$ are $k$-tuples of strings from $\Sigma^*$. For $w_1, \ldots, w_k$, we say that

$$R(w_1, \ldots, w_k) \text{ holds} \qquad \text{if and only if} \qquad \langle w_1, \ldots, w_k \rangle \in R.$$

A unary relation $R$ is also called a set or a language.[5] (Calling a unary relation a set is consistent with our convention that a 1-tuple $\langle w \rangle$ is identical to $w$. See the comment after Example III.55.)

[5]This terminology "language" is common in theoretical computer science. It has nothing to do with our earlier definition of a "language" as a set of non-logical symbols for first-order formulas.

**Example V.5.** The *reversal* of a string $w$ is denoted $w^{\mathrm{R}}$ and equal to the string $w$ written backward. For example, if $w = 1101$ then $w^{\mathrm{R}} = 1011$. Also, $\epsilon^{\mathrm{R}} = \epsilon$. In general, if $w = a_1a_2\cdots a_{k-1}a_k$, then $w^{\mathrm{R}} = a_ka_{k-1}\cdots a_2a_1$. Note that $|w| = |w^{\mathrm{R}}|$. The reversal function $w \mapsto w^{\mathrm{R}}$ is a unary function.

The *concatenation* of strings $u$ and $v$ is the string $uv$. That is, if $u = a_1\cdots a_k$ and $v = b_1\cdots b_\ell$, then $uv = a_1\cdots a_kb_1\cdots b_\ell$. As a special case, note that $u\epsilon = \epsilon u = u$. Of course, $|uv| = |u| + |v|$. The concatenation function that maps the pair $(u, v)$ to $uv$ is a binary function.

A string $w$ is a *palindrome* if $w = w^{\mathrm{R}}$. The set of palindromes is a unary relation.

**Example V.6.** Let $w \in \Sigma^*$ and $k \in \mathbb{N}$. The *iterated concatenation* of $w$ with itself $k$ times is denoted $w^k$. This can be recursively defined by $w_0 = \epsilon$ and $w^{i+1} = w^iw$. For example, if $w = 110$, then $w^3 = 110110110$. A useful form of iterated concatenation is the binary function that maps $(u, v)$ to $u^{|v|}$; i.e., it concatenates $|v|$ many copies of $u$.

**Definition V.7.** A $k$-ary function $f$ (on $\Sigma^*$) is *computable* provided there is an algorithm $M_f$ which when given as input any $k$ strings $w_1, \ldots, w_k$ from $\Sigma^*$ and produces as output the string that is equal to $f(w_1, \ldots, w_k)$.

We write $M_f(w_1, \ldots, w_k)$ for the output string produced by $M_f$ on input $w_1, \ldots, w_k$. So $M_f$ is an algorithm for $f$ provided

$$M_f(w_1, \ldots, w_k) \;=\; f(w_1, \ldots, w_k), \qquad \text{for all } w_1, \ldots, w_k \in \Sigma^*.$$

It is important to note that for $f$ to be computable, $M_f$ must succeed in producing the correct output no matter what strings $w_1, \ldots, w_k$ are input.

**Definition V.8.** A $k$-ary relation $R$ (on $\Sigma^*$) is *decidable* provided there is an algorithm $M_R$ which when given as input any $k$ strings $w_1, \ldots, w_k$ from $\Sigma^*$ produces the answer "Accept" or "Reject" depending on whether $R(w_1, \ldots, w_k)$ holds or not. When this holds, we say that the algorithm $M_R$ *decides* the relation $R$.

If $R$ is not decidable, we say that $R$ is *undecidable*.

We have left undefined what it means for $M_R$ to produce an answer of "Accept" or "Reject"; the precise details are unimportant; all that is needed is that algorithm has some definitive, unambiguous way of signaling acceptance or rejection. We generally use the terminology "$M_R(w_1, \ldots, w_k)$ accepts" or "$M_R(w_1, \ldots, w_k)$ rejects" instead of saying that $M_R(w_1, \ldots, w_k)$ produces the answer "Accept" or "Reject". We also use the terminology "$M_R$ accepts $w_1, \ldots, w_k$" or "$M_R$ rejects $w_1, \ldots, w_k$".

In other words, an algorithm $M$ decides $R$ provided that, for all $w_1, \ldots, w_k$,

$$\begin{aligned} M(w_1, \ldots, w_k) \text{ accepts} &\quad\Leftrightarrow\quad R(w_1, \ldots, w_k) \text{ holds} \\ M(w_1, \ldots, w_k) \text{ rejects} &\quad\Leftrightarrow\quad R(w_1, \ldots, w_k) \text{ does not hold.} \end{aligned}$$

Once again, it is important that $M_R$ is required to either accept or reject, no matter what strings $w_1, \ldots, w_k$ are input.

At this point, the reader might object to the previous definitions defining the notions of "computable" and "decidable" in terms of the undefined notion of "algorithm". The intention, however, is that we are currently working with an informal notion of algorithm as outlined in Section V.1. In later chapters, we will give formal mathematical definitions of "algorithm" in the form of Turing machines or recursive function definitions. At that point, the definitions of "computable" and "decidable" also become formal mathematical definitions. All the theorems stated below will still hold under the formal definition of algorithm.

**Example V.9.** Let $\Sigma$ be an arbitrary alphabet. The reversal function $w \mapsto w^{\mathrm{R}}$, the concatenation function, and the iterated concatenation function are all computable. The set of palindromes is a decidable set.

The empty set $\varnothing$ is a decidable set; it is decided by the algorithm which rejects all inputs. The set $\Sigma^*$ is also decidable; it is decided by the algorithm that accepts all inputs.

The empty set should not be confused with the set $\{\epsilon\}$ containing only the empty string. The set $\{\epsilon\}$ also decidable: it is decided by an algorithm that checks if its input $w$ contains any symbols at all: if so, it rejects, and if not, it accepts.

The "complement" of a relation is the set of $k$-tuples that are not in the relation:

**Definition V.10.** Let $R$ be a $k$-ary relation on $\Sigma^*$, so $R \subseteq (\Sigma^*)^k$. The *complement* of $R$ is denoted $\overline{R}$ and is equal to $(\Sigma^*)^k \setminus R$.

**Theorem V.11.** *A relation $R$ is decidable if and only if its complement $\overline{R}$ is decidable.*

*Proof.* Suppose that a $k$-ary relation $R$ is decided by an algorithm $M$. Let $M'$ be the algorithm that acts just like $M$ except that it rejects if $M$ accepts and it accepts if $M$ rejects. We can write this in algorithm form as:

> Input: Strings $w_1, \ldots, w_k$
> Algorithm: $M'$
> Run $M$ on input $w_1, \ldots, w_k$.
> If $M$ accepts, then reject.
> If $M$ rejects, then accept.

Clearly $M'$ decides $\overline{R}$. This shows that if $R$ is decidable, then so is $\overline{R}$. Since the complement of $\overline{R}$ is the same as $R$, it follows that if $\overline{R}$ is decidable then $R$ is decidable. □

## V.3.2 Functions and relations on the integers.

As defined above, functions and relations are defined to act on strings of symbols, i.e., members of $\Sigma^*$; this will be the most appropriate when using Turing

machines for computations. However, when studying recursive functions and definability in theories of arithmetic, we will instead want to talk about functions and relations that act on integers, not on strings. For this, we can view integers as being coded by strings of symbols. One way to do this is via base 10 representation:

**Example V.12.** Let $\Sigma$ be the set of digits $\{0,1,2,3,4,5,6,7,8,9\}$. Any member $w$ of $\Sigma^*$ can be viewed as the base 10 representation of a nonnegative integer $num_{10}(w)$, possibly with extra leading zeros. For instance, $num_{10}(1) = num_{10}(01)$ is equal to the integer 1. Similarly, $num_{10}(27) = num_{10}(027)$ is equal the integer 27. The empty string is a binary representation for 0; that is, $num_{10}(\epsilon)$ is the integer 0. Conversely, for $i \in \mathbb{N}$, $str_{10}(i)$ is the unique base 10 representation for $i$ that does not start with a leading 0.

The integer addition defines a binary function on $\Sigma^*$, namely, $(u,v) \mapsto str_{10}(num_{10}(u) + num_{10}(v))$. The inequality predicate $LE$ defined so that

$$LE(u,v) \text{ holds} \quad \text{if and only if} \quad num_{10}(u) \leq num_{10}(v)$$

is an example of a binary relation on $\Sigma^*$. The integer addition function is computable and the relation $LE$ is decidable.

We could have as equally well used base 2 (or any other base) in the example. It would also work to use unary notation where $\Sigma = \{1\}$ has only a single symbol, and the string $1^n$ represents the integer $n$. There are of course algorithms to convert between base 10 and base 2 and unary representations of integers. Thus, from the point of view of computability, it does not make any difference which base is used to represent integers as strings.

**Definition V.13.** Let $\Sigma = \{0,1\}$. Let *str* and *num* be similar to the functions in the previous example, but using base 2 instead of base 10. Suppose $f : \mathbb{N}^k \to \mathbb{N}$ is a $k$-ary function on the integers. Then $f^{str}$ is the $k$-ary function on $\Sigma^*$ defined by

$$f^{str}(w_1,\ldots,w_k) \;=\; str(f(num(w_1),\ldots,num(w_k))).$$

Then $f$ is defined to be *computable* if and only $f^{str}$ is computable. Similarly, suppose $R \subseteq \mathbb{N}^k$ is a $k$-ary relation on the integers. Then $R^{str}$ is the relation defined by

$$R^{str}(w_1,\ldots,w_k) \text{ holds} \quad \text{if and only if} \quad R(num(w_1),\ldots,num(w_k)) \text{ holds.}$$

Then $R$ is defined to be *decidable* if and only if $R^{str}$ is decidable.

Similar conventions apply to the to-be-defined notions of "semidecidable", "computably enumerable" and "partial computable".

**Example V.14.** Many functions on the integers are computable. This includes, for instance, addition $(n,m) \mapsto n+m$, multiplication $(n,m) \mapsto n \cdot m$, and exponentiation $(n,m) \mapsto n^m$. Another computable function is the *truncated subtraction* function $\dot{-}$ defined by

$$n \dot{-} m \;=\; \begin{cases} n-m & \text{if } n > m \\ 0 & \text{otherwise.} \end{cases}$$

That is, $n \dot{-} m = \max\{n - m, 0\}$. Note that truncated subtraction, unlike subtraction, is total on $\mathbb{N}$. Likewise many relations on the integers are decidable. This includes, for instance, the set of primes (a unary relation) and the binary relation $\leq$.

**Definition V.15.** Let $R$ be a $k$-ary relation on $\mathbb{N}$. The *characteristic function* of $R$ is denoted $\chi_R$ and is the $k$-ary function on the integers defined by

$$\chi_R(n_1, \ldots, n_k) = \begin{cases} 1 & \text{if } R(n_1, \ldots, n_k) \\ 0 & \text{otherwise.} \end{cases}$$

**Example V.16.** The characteristic function of the binary relation $\leq$ is the function

$$\chi_{\leq}(n, m) = \begin{cases} 1 & \text{if } n \leq m \\ 0 & \text{otherwise.} \end{cases}$$

Namely, $\chi_{\leq}(n, m) = \min\{1, m \dot{-} n\}$.

**Theorem V.17.** *A $k$-relation $R$ on $\mathbb{N}$ is decidable if and only if its characteristic function $\chi_R$ is computable.*

*Proof.* Suppose that an algorithm $M$ decides the relation $R$. Modify the algorithm $M$ so that instead of accepting it outputs 1, and instead of rejecting, it outputs 0. The resulting algorithm $N$ computes the characteristic function $\chi_R$.

Conversely, suppose $N$ computes $\chi_R$. Modify $N$ so that when it is about to output 1, it accepts; and when it is about to output 0, it rejects. This yields an algorithm $M$ that decides $R$. □

## V.3.3 Computable enumerability

The above definitions of computable functions and decidable relations required algorithms that return an answer on all inputs. It is possible, however, for an algorithm to run forever and never produce an output. This leads naturally to the concept of computably enumerable relations. First, however, we define "semidecidable".

**Definition V.18.** A $k$-ary relation $R$ (on $\Sigma^*$) is *semidecidable* provided there is an algorithm $M_R$ when give as input any $k$ strings $w_1, \ldots, w_k$ from $\Sigma^*$, produces the answer "Accept" if and only if $R(w_1, \ldots, w_k)$ holds. If $R(w_1, \ldots, w_k)$ does not hold, then $M_R(w_1, \ldots, w_k)$ may either produce the answer "Reject" or may run forever and fail to produce an answer.

When this holds, we say that "$M_R$ *semidecides* the relation $R$".

It is common in the literature to say that "$M_R$ *recognises* $R$" or "$M_R$ *accepts* $R$" to mean that "$M_R$ *semidecides* $R$". The three terms are equivalent. "Semidecides" is the least common terminology; nonetheless, we will often use "semidecides" just because it is less ambiguous. In any case, what it means is that for all $w_1, \ldots, w_k$,

$$M(w_1, \ldots, w_k) \text{ accepts } \Leftrightarrow R(w_1, \ldots, w_k) \text{ holds.}$$

It is no longer required that $M_R$ rejects inputs $w_1, \ldots, w_k$ not in $R$; it is just disallowed for $M_R$ to accept them. The intuition is that if $\langle w_1, \ldots, w_k \rangle$ is in $R$, then running $M_R(w_1, \ldots, w_k)$ will eventually tells us that $\langle w_1, \ldots, w_k \rangle$ is in $R$; however, if $\langle w_1, \ldots, w_k \rangle$ is not in $R$, then running $M_R(w_1, \ldots, w_k)$ may never yield an answer. That is, for $\langle w_1, \ldots, w_k \rangle$ is not in $R$, the algorithm $M_R$ might never give any indication of whether $\langle w_1, \ldots, w_k \rangle$ is in $R$ or not.

**Example V.19.** Consider the binary relation $R$ on the integers defined by letting $R(n, m)$ be true if and only if there is an integer $i \geq n$ such that both $i$ and $i + m$ are prime. That is,

$$R = \{\langle n, m \rangle \in \mathbb{N}^2 : \text{ for some } i \geq n, \text{ both of } i \text{ and } i{+}m \text{ are prime})\}.$$

We claim that $R$ is semidecidable. To prove this, consider the following algorithm.

Input: Integers $n$ and $m$
Algorithm:
For $i = n$, $n{+}1$, $n{+}2$, $n{+}3$, ...
    If $i$ and $i + m$ are both prime
        Accept (and halt).
    End-if
End-for

Note that the for-loop is potentially an infinite loop: if $\langle n, m \rangle \notin R$, the algorithm will run forever.

We have so far worked with algorithms that, on any given input, either compute a single answer and then halt, or run forever without producing any answer. We can also consider algorithms that generate or "enumerate" a series of outputs, perhaps an infinite list of outputs. For this, we assume that the algorithm does not take any inputs at all: it just starts running. As it runs, it may upon occasion output a string in $\Sigma^*$. After outputting the string, it continues running and successively outputting strings. It may output a finite number of strings (possibly no strings at all) or it may run forever and output an infinite number of strings. We call such an algorithm an "enumerator". Such a machine $M$ enumerates, i.e., successively outputs a sequence of zero or more strings

$$v_0,\ v_1,\ v_2, \ldots,$$

where $v_i$ is the $i$-th string output by the algorithm. It is permitted that the algorithm enumerates the same output string more than once.

**Definition V.20.** Let $M$ be an enumerator as above, outputting strings in $\Sigma^*$. Then $M$ is said to *enumerate* the set $\{v_1, v_2, v_3, \ldots\}$. When this holds, the set is called *computably enumerable*, or *c.e.* for short.

**Definition V.21.** The definition of enumerator can be generalized in the obvious way to allow outputting $k$-tuples of strings, that is, outputting members of $(\Sigma^*)^k$. In this case, the $i$-th output $v_i$ is a $k$-tuple of strings, i.e., $v_i \in (\Sigma^*)^k$.

A $k$-ary relation $R$ is said to be *computably enumerable* if it is equal to the set of $k$-tuples output by some enumerator.

**Example V.22.** The empty set is c.e. since it is enumerated by the algorithm which just loops and never outputs anything.

Any finite set $\{w_1, \ldots, w_k\}$ is c.e., since the algorithm can output successively the $k$-strings $w_1, \ldots, w_k$ and enter an infinite loop and never produce any further outputs. Alternatively, it is enumerated by the algorithm that first outputs successively $w_1, \ldots, w_{k-1}$ and then enters an infinite loop that repeatedly outputs $w_k$.

**Example V.23.** The set $\Sigma^*$ of all strings over $\Sigma$ is c.e. Let $k = |\Sigma|$ be the number of symbols in $\Sigma$. A possible algorithm that enumerates $\Sigma^*$ acts as follows: It first outputs the empty string $\epsilon$. It then outputs, one at a time, the $k$ strings in $\Sigma^*$ of length 1, next the $k^2$ many strings of length 2, next the $k^3$ many strings of length 3, etc.:

Algorithm:
For $i = 0, 1\, 2 \ldots$
    Output, one at a time, the $|\Sigma|^i$ many strings in $\Sigma^*$ of length $i$.
End-for

Note that this algorithm runs forever; this is of course necessary since $\Sigma^*$ is infinite.

Likewise, the $k$-ary relation $(\Sigma^*)^k$ containing all $k$-tuples of strings over $\Sigma$ is c.e. Define the *total length* of a $k$-tuple $\langle w_1, \ldots, w_k \rangle$ to equal $\sum_j |w_j|$. For each $i$, there are finitely many $k$-tuples of total weight $i$. An algorithm to enumerate $(\Sigma^*)^k$ is thus

Algorithm:
For $i = 0, 1\, 2 \ldots$
    Output, one at a time, the $k$-tuples in $(\Sigma^*)^k$ of total length $i$.
End-for

**Theorem V.24.** *If $R$ is decidable then $R$ is computably enumerable.*

*Proof.* Suppose a $k$-ary relation $R$ is decided by an algorithm $M$. Then $R$ is enumerated by the following algorithm:

Assumption: $M$ decides $R$.
Algorithm:
For $i = 1, 2, 3, 4, \ldots$
    Let $v_i$ be the $i$-th $k$-tuple in a computable enumeration of $(\Sigma^*)^k$.
    Run $M$ on input $v_i$ until it either accepts or rejects.
    If $M$ accepts the input $v_i$, then output $v_i$ (but do not halt).
End-for

Since $M$ decides $R$, running $M$ any input $v_i$ will eventually lead to either acceptance or rejection. Therefore this algorithm never gets stuck and every $v_i$ in $(\Sigma^*)^k$ eventually gets considered. □

We shall see later that not every computably enumerable set is decidable. On the other hand, it is a pleasant surprise that semidecidability and computable enumerability are equivalent:

**Theorem V.25.** *A $k$-ary relation $R$ is semidecidable if and only if it is computably enumerable.*

*Proof.* First, we prove the "if" direction. Suppose $R$ is computably enumerated by an enumerator algorithm $M$. Here is an algorithm that semidecides $R$:

> Assumption: $M$ enumerates $R$.
> Input: A $k$-tuple $v$ in $(\Sigma^*)^k$
> Algorithm:
> Run the algorithm $M$, watching what it outputs.
> If $M$ ever outputs the $k$-tuple $v$, then accept and halt.

Now we prove the "only if" direction. Suppose that $N$ is an algorithm that semidecides $R$. Unlike in the algorithm used for the proof of Theorem V.24, we cannot enumerate the members of $R$ by running $N$ to completion on input $v_1$, then on input $v_2$, then on input $v_3$, etc. The problem is that $N$ does not halt for all inputs: it accepts inputs that are in $R$, but it may run forever and give no answer on inputs that are not in $R$. Instead, we must take turns with trial computations of $N$ on many different $v_i$'s. This process is called "dovetailing".

Here is a dovetailing algorithm that enumerates $R$:

> Assumption: $N$ semidecides $R$.
> Algorithm:
> For $i = 1, 2, 3, 4, \ldots$
>  For $j = 1, 2, 3, \ldots, i$
>  Let $v_j$ be the $j$-th $k$-tuple in a computable enumeration of $(\Sigma^*)^k$.
>  Run $N$ on input $v_j$ for a maximum of $i$ steps.
>  If $N$ on input $v_j$ accepts with $\leq i$ steps, then output $v_j$ (and do not halt).
>  End-for
> End-for

Note that the inner loop is finite, and the output loop is infinite. This enumerator algorithm works since it never gets stuck on any $v_j$. Instead, it runs $N$ for a fixed number $i$ of steps on input $v_j$ and then (temporarily) gives up on $v_j$. In the next iteration of the outer loop, it again tries $N$ on input $v_j$, now for $i+1$ steps. In this way, any $v_j$ in $R$ will eventually be discovered to be accepted by $N$. □

The last algorithm used an important property of algorithms. It is assumed that an algorithm $N$ uses a finite set of unambiguous instructions and runs step-by-step. Each step in a computation of $N$ involves applying the instructions once. This means we can track the number of steps taken by $N$ — this is needed in order to control the dovetailing.

**Definition V.26.** A relation $R$ is co-computably enumerable (co-c.e.) if its complement $\overline{R}$ is computably enumerable (c.e.).

Theorem V.24 showed that if $R$ is decidable, then $R$ is c.e. Since $\overline{R}$ is decidable if $R$ is decidable, it follows that if $R$ is decidable then $R$ is both c.e. and co-c.e. The next theorem states that the converse holds too.

**Theorem V.27.** *$R$ is both c.e. and co-c.e. if and only if $R$ is decidable.*

*Proof.* Let $R$ be a $k$-ary relation. As just remarked, it suffices to show that if $R$ is both c.e. and co-c.e. then $R$ is decidable. Suppose that $M$ enumerates $R$ and $N$ enumerates $\overline{R}$. Then the following is an algorithm that decides $R$:

> Assumption: $M$ enumerates $R$, and $N$ enumerates $\overline{R}$.
> Input: $v \in (\Sigma^*)^k$
> Algorithm:
> Run $M$ and $N$ in parallel, watching what strings the output.
> If $M$ outputs $v$, accept. ($v \in R$).
> If $N$ outputs $v$, reject. ($v \notin R$).

Since $M$ and $N$ enumerate $R$ and $\overline{R}$, eventually one of them will output $v$. Thus the algorithm eventually halts on all inputs $v$ and correctly decides whether $v \in R$. □

There are several possibilities for how to run $M$ and $N$ "in parallel" in the above algorithm. One way would be to interleave the computations of $M$ and $N$ by alternating between running $M$ for a few steps (saving the intermediate results) and running $N$ for a few steps (again, saving the intermediate results). Another possibility is to take the finite sets of unambiguous instructions for both $M$ and $N$ and conjoin them to give a set of instructions for running $M$ and $N$ simultaneously. The conjoined instructions would still be a finite set of unambiguous instructions, and thus give an effective procedure for running $M$ and $N$ simultaneously.

### Partial computable functions

**Definition V.28.** A $k$-ary *partial function* $f : (\Sigma^*)^k \to \Sigma^*$ is a function with domain a subset of $(\Sigma^*)^k$ and range a subset of $\Sigma^*$.

We say that $f(w_1, \ldots, w_k)$ *diverges*, written $f(w_1, \ldots, w_k)\uparrow$, if $\langle w_1, \ldots, w_k \rangle$ is not in the domain of $f$.

We say that $f(w_1, \ldots, w_k)$ *converges*, written $f(w_1, \ldots, w_k)\downarrow$, if $\langle w_1, \ldots, w_k \rangle$ is in the domain of $f$. We write $f(w_1, \ldots, w_k) = w$ to mean that $f(w_1, \ldots, w_k)$ converges and $w$ is the function value. We also equivalently write $f(w_1, \ldots, w_k)\downarrow = w$ when we want to stress the fact that $f(w_1, \ldots, w_k)$ converges.

The above definition has overloaded the notation "$f : (\Sigma^*)^k \to \Sigma^*$", as this this notation can be used for both partial functions and ordinary (that is, total) functions. Our conventions will be that the terminology $f$ is a "function" means

that $f$ is total and thus converges for all inputs. If $f$ is partial or might be partial, then we will say explicitly that $f$ is a "partial function". It is permitted that a partial function has domain equal to all of $(\Sigma^*)^k$; that is, a partial function may actually be total.

**Definition V.29.** A $k$-ary *partial computable* function is a partial function $f : (\Sigma^*)^k \to \Sigma^*$ for which there is an algorithm $M$ so that, for all $w_1, \ldots, w_k \in \Sigma^*$,

$$M(w_1, \ldots, w_k) \text{ halts and outputs } w \quad \Leftrightarrow \quad f(w_1, \ldots, w_k)\!\downarrow = w$$
$$M(w_1, \ldots, w_k) \text{ never halts} \quad \Leftrightarrow \quad f(w_1, \ldots, w_k)\!\uparrow$$

**Example V.30.** Every computable function is a partial computable function.

**Example V.31.** Referring back to Example V.19, let $f : \mathbb{N} \to \mathbb{N}$ be the partial function defined by letting $f(n, m)$ equal the least $i \geq n$ such that $i$ and $i+m$ are both prime if there is such an $i$, and letting $f(n, m)$ be undefined if no such $i$ exists.

The function $f$ can be defined using minimization operator notation:

$$f(n, m) \;=\; \mu i\,(i \geq n, \text{ and } i \text{ and } i+m \text{ are prime}).$$

The minimization notation "$\mu i\, Q(i)$" means "the least $i$ such that property $Q(i)$ holds". It is understood that the value of $\mu i\, Q(i)$ is undefined if no such $i$ exists.

An algorithm that computes $f$ is:

Input: Integers $n$ and $m$
Algorithm:
For $i = n$, $n$+1, $n$+2, $n$+3, ...
    If $i$ and $i + m$ are both prime
        Output $i$ (and halt).
    End-if
End-for

Note that this differs from the algorithm in Example V.19 only in that it outputs $i$ instead of just accepting.

Exercise V.14 asks you to prove that a set is c.e. if and only if it is equal to the range of a partial computable function. Examples V.19 and V.31 provide a strong hint of how to prove this. Exercises V.13, V.15 and V.16 give other characterizations of c.e. sets in terms of being the domain or range of a function.

Recall the definition of the *graph* of a function $f$:

**Definition V.32.** Let $f : (\Sigma^*)^k \to \Sigma^*$ be a $k$-ary partial function. The *graph* of $f$ is denoted $G_f$ and is the $(k+1)$-ary relation on $\Sigma^*$ defined by

$$G_f \;=\; \{\langle w_1, \ldots, w_k, v\rangle : f(w_1, \ldots, w_k)\!\downarrow = v\}.$$

**Theorem V.33.** *Let $f : (\Sigma^*)^k \to \Sigma^*$ be a $k$-ary partial function. The following are equivalent:*

(a) *$f$ is partial computable.*

(b) *The graph $G_f$ of $f$ is computably enumerable.*

*Proof.* First, suppose there is an algorithm $M$ that enumerates $G_f$. An algorithm $N(w_1, \ldots, w_k)$ can partial compute $f(w_1 \ldots, w_n)$ by running $M$ until (and if) it outputs a $(k+1)$-tuple of the form $\langle w_1, \ldots, w_k, v\rangle$ and at that point $N(w_1, \ldots, w_k)$ outputs $v$ as the value of $f(w_1 \ldots, w_n)$.

For the converse, suppose $f$ is partial computed by an algorithm $N$. The following algorithm $M$ semidecides $G_f$.

> Assumption: $N$ partial computes $f$.
> Input: A $k$-tuple $\langle w_1, \ldots, w_k, v\rangle$
> Algorithm $M$:
> Run $N(w_1, \ldots, w_k)$ until (and if) it produces an output $v'$.
> If $v' = v$, accept (and halt).
> Otherwise, reject (and halt).

Note that the algorithm does not halt if $f(w_1, \ldots, w_k)\uparrow$, since in that case, $N(w_1, \ldots, w_k)$ does not halt. It is clear that $M(w_1, \ldots, w_k, v)$ accepts if and only if $f(w_1, \ldots, w_k)\downarrow = v$. □

## V.4 Algorithms for Propositional Logic

We now discuss an algorithm for validity and implication in propositional logic. The first main result will be that the set of tautologies is decidable. The second main result will be that if a set $\Gamma$ of propositional formulas is decidable, or even c.e., then the set of its tautological consequences is c.e. Our proofs will all be based on the method of truth tables.

First, however, let's consider algorithms for recognizing syntactically correct formulas. To begin, we have to express formulas as strings (expressions) over a finite alphabet $\Sigma$. The definition of propositional formulas in Definition I.1 used an infinite set of symbols as each $p_i$ is a distinct symbol. The informal notion of algorithm only allows strings over *finite* alphabets. Accordingly, we will encode formulas as strings over the alphabet $\Sigma^{\text{prop}}$ with 17 symbols:

$$\{\wedge, \vee, \rightarrow, \leftrightarrow, (, ), p, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

The digits 0-9 are used for writing out the subscripts of propositional variables as base 10 integers. For example, the formula $p_7 \vee p_{42} \rightarrow p_{747}$ will be represented by the $\Sigma^{\text{prop}}$ string "$((p7 \vee p42) \rightarrow p747)$". Note that we require that a formula be fully parenthesized when encoded as a $\Sigma^{\text{prop}}$-expression.[6]

To simplify terminology in the sequel, we frequently conflate propositional formulas and their representations as $\Sigma^{\text{prop}}$-strings.

[6]The convention that the $\Sigma^{\text{prop}}$ representations of formulas include full parenthesization is not crucial. There are certainly algorithms that can produce the fully parenthesized expression from a formula that is written without all of its parentheses according to the conventions of Section I.2.

**Theorem V.34.** *The set of syntactically correct propositional formulas is decidable.*

What this theorem means is that the set of $\Sigma^{\text{prop}}$-strings that represent syntactically correct propositional formulas is decidable. Indeed, it is straightforward to write a parsing algorithm that decides this. The main difficulty for the parsing algorithm is track the counts of open parentheses and close parentheses; for this, see Theorems I.42 and I.43 and Exercises I.36-I.38 on unique readability. For proving Theorem V.34, the parsing algorithm only needs to accept or reject according to whether its input encodes a syntactically correct propositional formula. But in fact parsing algorithms can also effectively build a parse tree.

**Theorem V.35.** *The set of tautologies is decidable.*

*Proof.* The algorithm first checks whether its input codes a syntactically correct proposition formula. If so, it uses the method of truth tables to decide whether a given propositional formula is a tautology. □

The next theorem states that there is an algorithm that can decide whether a finite set $\Gamma$ of sentences is satisfiable, and an algorithm that can decide whether $\Gamma \vDash A$. The idea is just that the method of truth tables can be used to algorithmically determine if $\Gamma$ is satisfiable. However, there is the complication that the definition of concepts such as "decidable" are based on $k$-ary relations that take a fixed number of strings as input. Now we want an algorithm to take as input a set $\Gamma$ whose size is not specified ahead of time. For this, the finite set $\Gamma$ is encoded into a single string.

This is straightforward. One way to do it is to enlarge the alphabet $\Sigma^{\text{prop}}$ by adding comma as an additional symbol; namely, define $\Sigma^{\text{prop+}} = \Sigma^{\text{prop}} \cup \{\,,\}$. Then an arbitrary finite set $\Gamma$ of formulas can be encoded as a string over the alphabet $\Sigma^{\text{prop+}}$ by just concatenating formulas separated with commas. That is, if $\Gamma = \{A, \ldots, A_k\}$, then $\Gamma$ is encoded by the string that consists of the encodings of the $A_i$'s as $\Sigma^{\text{prop}}$-strings separated by commas. For instance, if $\Gamma = \{p_1, \neg p_3 \vee p_{17}\}$, then $\Gamma$ can be encoded by the string "$p1$,$\neg(p3 \vee p17)$".[7]

This encoding of a finite, variable-length set into a single string is an example of the use of "sequence coding" to encode multiple strings into a single string. We will see more examples of this later on when sequence coding is used to let a single integer encode a finite length sequence of integers.

**Theorem V.36.** *Work in propositional logic.*

(a) *The unary relation*

$$\{\Gamma : \Gamma \textit{ is a finite satisfiable set of formulas}\}$$

[7] In fact, the commas could be omitted, and the breaks between formulas can still be detected. The use of commas makes the breaks between formulas explicit and does not depend on the details of how formulas are defined.

*is decidable.*

(b) *The binary relation*

$$\{\langle \Gamma, A\rangle : \Gamma \text{ is a finite set of formulas, } A \text{ is a formula, and } \Gamma \vDash A\}$$

*is decidable.*

*Proof.* The algorithm for (a) does the following: It first checks whether $\Gamma$ correctly encodes a (finite) set of syntactically correct formulas. If not, it rejects. Otherwise, it uses the method of truth tables to determine whether $\Gamma$ is satisfiable. The algorithm for (b) is similar. □

Now we consider *infinite* sets $\Gamma$ of sentences. We will consider the case where the set $\Gamma$ is decidable or computably enumerable. However, unlike in the previous theorem, $\Gamma$ is now a fixed set (i.e., $\Gamma$ is not given as part of the input).

**Theorem V.37.** *Let $\Gamma$ be a computably enumerable set of propositional formulas. Then the set of tautological consequences of $\Gamma$, namely $\operatorname{Cn}\Gamma = \{A : \Gamma \vDash A\}$, is computably enumerable.*

*Proof.* Suppose $M$ is an algorithm that enumerates $\Gamma$. By Theorem V.25, it suffices to prove that $\operatorname{Cn}\Gamma$ is semidecidable. We claim the following algorithm $N$ works:

Assumption: $M$ enumerates $\Gamma$.
Input: $A \in (\Sigma^{\text{prop}})^*$
Algorithm $N$:
If $A$ is not a syntactically correct formula, reject.
For $i = 0, 1, 2, 3, \ldots$
    Run $M$ until (and if) it generates $i$ members of $\Gamma$.
    Let $\Gamma_i$ be the first $i$ members enumerated by $M$.
    If $\Gamma_i \vDash A$, then accept (and halt).
End-for

It is clear that the only way the algorithm $N$ can accept $A$ is if some finite subset $\Gamma_i$ of $\Gamma$ tautologically implies $A$. Conversely, if $\Gamma$ tautologically implies $A$, then some finite subset $\Pi$ of $\Gamma$ tautologically implies $A$. Eventually, $M$ will enumerate a subset $\Gamma_i$ of $\Gamma$ such that $\Pi \subseteq \Gamma_i$. At that point, $N$ will accept $A$.

Therefore, the algorithm $N$ correctly semidecides the tautological consequences of $\Gamma$, and thus $\operatorname{Cn}\Gamma$ is computably enumerable. □

The algorithm $N$ was written so as to work even in the case where $\Gamma$ is finite. It is even permitted that $\Gamma$ is empty: in this case, for-loop executes only for $i = 0$ and uses $\Gamma_i = \varnothing$. More generally, if $\Gamma$ is finite, the for-loop might execute only finitely many rounds and then, once all members of $\Gamma$ have been enumerated, continue running $M$ forever while waiting to see if $M$ will produce another member of $\Gamma$.

Note that, however, the fact Theorem V.37 holds for finite $\Gamma$ is also an immediate corollary of part (b) of Theorem V.36.

The obvious next question is what happens when $\Gamma$ is decidable instead of computably enumerable? A natural first thought is that the set $\{A : \Gamma \vDash A\}$ might be decidable. But this is not the case. To give a non-rigorous argument of how it can fail to be decidable consider, for example, a set $\Gamma$ that *might* contain the propositional formulas of the form $\bigvee_{j=1}^{k} p_i$ for some $j, k$ (but definitely does not contain any other formulas). Since $\bigvee_{j=1}^{k} p_i$ is the $k$-fold disjunction of $p_i$ with itself, it is tautologically equivalent to $p_i$. Then $\Gamma \vDash p_i$ holds if and only if one of the formulas $\bigvee_{j=1}^{k} p_i$ is in $\Gamma$. If $\Gamma$ is decidable, then an algorithm given $i$ as input can determine, for any finite number of values $k$, whether $\bigvee_{j=1}^{k} p_i$ is a member of $\Gamma$. But there is no way for the algorithm to rule out the possibility that $\Gamma$ contains $\bigvee_{j=1}^{k} p_i$ for some other values of $k$. Equivalently, there is no way for the algorithm to decide whether $\Gamma \vDash p_i$ when given $i$ as input.

In addition, it turns out that if $\Gamma$ is a computably enumerable set of formulas, then there is a decidable set $\Pi$ of logically equivalent formulas. That is, if $\Gamma$ is c.e., there is a decidable $\Pi$ such that $\Gamma \vDash\dashv \Pi$. This is known as Craig's Theorem, and you are asked to prove it in Exercise V.22.

Exercise V.26 asks you to prove that there is a decidable $\Gamma$ (equivalently, a computably enumerable $\Gamma$) whose set of logical consequences is undecidable. That proof will depend on the methods that will be developed in Section V.6.

## V.5 Algorithms for First-Order Logic

We now discuss validity and implication for first-order logic. The first main result in this seection is that the set of logically valid formulas is computably enumerable. The second main result is that the set of logical consequences of a computably enumerable set of sentences is also computably enumerable.

It is assumed that we are working with a fixed language $L$ of nonlogical symbols. First-order $L$-formulas can be expressed as strings of symbols over the alphabet $\Sigma^{\text{fo}-L}$ defined as

$$\Sigma^{\text{fo}-L} \;=\; L \cup \{=, \neg, \rightarrow, \forall, (, ), x, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

This is an alphabet of cardinality $|L|$+17. For example, the formula $\forall x_{17}\,(f(x_{17}) = x_{17} \rightarrow R(x_{17}))$ will be encoded with the $\Sigma^{\text{fo}-L}$-string "$\forall x17(f(x17)=x17\rightarrow R(x17))$". Sets of first-order formulas, and first-order proofs (FO-proofs), will be expressed as strings of symbols over the alphabet

$$\Sigma^{\text{fo}-L+} = \Sigma^{\text{fo}-L} \cup \{,\}.$$

The additional symbol, comma, serves to separate formulas that appear in a set of formulas or in an FO-proof. For example, the sequence of formulas $x_1 = x_1$ followed by $\neg x_{12} = x_{12}$ would be encoded by the string "$x1=x1,\neg x12=x12$" of length 14.

We often conflate formulas and proofs with the strings that encode them. For instance, part (c) of the next theorem states that the set of syntactically correct $L$-formulas is decidable; strictly speaking, this means that the set of $\Sigma^{\text{fo}-L}$-strings that encode syntactically correct $L$-formulas is decidable.

**Theorem V.38.** *let $L$ be a finite language. Consider the following function and relations which take inputs in $(\Sigma^{\text{fo}-L})^*$.*

(a) *The set of syntactically correct $L$-terms is decidable.*
(b) *The set of syntactically correct atomic $L$-formulas is decidable.*
(c) *The set of syntactically correct $L$-formulas is decidable.*
(d) *Let $Subst(A, x, t)$ be the 3-ary relation on $(\Sigma^{\text{fo}-L})^*$ which is true for exactly the $L$-formulas $A$ such that the $L$-term $t$ is substitutable for the variable $x$ in $A$. The relation Subst is decidable.*
(e) *Let $Sub(A, x, t)$ be the 3-ary function which, for inputs $A$ an $L$-formula, $t$ an $L$-term, and $x$ a variable produces the $L$-formula $A(t/x)$. For all other inputs, $Sub(A, x, t)$ is the empty string (denoting an error condition). The function Sub is computable.*
(f) *The set $L$-formulas which are valid* FO *axioms is decidable.*

*Proof.* There are straightforward algorithms for all of these. □

**Theorem V.39.** *Let $L$ be a finite language. The set of pairs $\langle \Pi, P \rangle$ such that $P$ is valid* FO*-proof from the hypotheses $\Pi$ is decidable.*

*Proof.* The input is a pair $v, w$ of strings in $(\Sigma^{\text{fo}-L})^*$. The algorithm works as follows. First, if $v$ is not an encoding of a set of formulas $\Pi$ or if $w$ is not an encoding of a sequence $P$ of formulas, then reject. Otherwise, check that each formula in $P$ is a valid FO-axiom, is a member of $\Pi$, is inferred by Modus Ponens from two earlier formulas in $P$, or is inferred by the Generalization Rule from an earlier formula in $P$. If this holds for every formula in $P$ accept. Otherwise, reject.

It is clear that an algorithm can be designed to carry out these checks. □

Part (a) of the next theorem states that the set of valid first-order formulas is computably enumerable. This is a fairly remarkable fact because the validity of a formula means that the formula is true in all possible structures and object assignments. Its proof is based on the fact that a first-order formula is valid if and only if has an FO-proof, namely, on the fact that Completeness and Soundness Theorems hold.

**Theorem V.40.** *Let $L$ be a finite language.*

(a) *The set of valid $L$-formulas is computably enumerable.*
(b) *The binary relation*

$$\{\langle \Gamma, A \rangle : \Gamma \text{ is a finite set of formulas and } \Gamma \vDash A\}$$

*is computably enumerable.*

*Proof.* The idea of the proof is that the algorithm can enumerate all possible proofs, seeking a proof of $A$ (from the hypotheses $\Gamma$). If $\vDash A$ holds (or $\Gamma \vDash A$ holds), then there does exist such a proof and it will eventually be found. In more detail, to prove (b) it will suffice to give an algorithm that semidecides the binary relation $\{\langle \Gamma, A \rangle : \Gamma \vDash A\}$. The following algorithm can be used. It uses the computable enumeration of strings in $(\Sigma^{\text{fo}-L+})^*$ from Definition V.23 to enumerate all members of $(\Sigma^{\text{fo}-L+})^*$ as $v_1, v_2, v_3, \ldots$.

Input: $A \in (\Sigma^{\text{fo}})^*$ and $\Gamma \in (\Sigma^{\text{fo}-L+})^*$.
Algorithm $N$:
If $A$ is not a syntactically correct formula, reject.
If $\Gamma$ is not a finite set of syntactically correct formulas, reject.
For $i = 1, 2, 3, \ldots$
    Let $v_i$ be the $i$-th member of a computable enumeration of $(\Sigma^{\text{fo}-L+})^*$.
    If $v_i$ encodes a syntactically correct FO-proof of $A$ from $\Gamma$,
        accept (and halt).
End-for

That proves part (b). The proof of (a) is similar but uses $\Gamma = \varnothing$. □

Note how the above algorithm is very simple-minded and very inefficient. It makes no attempt to analyze the logical structure of $A$ or formulas in $\Gamma$. It definitely makes no attempt to "understand" why $\Gamma$ might logically imply $A$. It will of course find a shortest proof of $A$ from $\Gamma$ (where length is measured by the length of the encoded proof). But this is done by a blind, brute-force search for a proof.

Theorem V.40(b) was stated for a finite set of hypotheses $\Gamma$. A similar result holds $\Gamma$ is a computably enumerable set:

**Theorem V.41.** *Let $L$ be a finite language. Suppose $\Gamma$ is a computably enumerable set of $L$-formulas. Then the set of $L$-formulas $A$ such that $\Gamma \vDash A$ is computably enumerable.*

*Proof.* The idea for this proof is to simultaneously enumerate members of $\Gamma$ and use a brute-force search through all possible proofs of $A$. If $\Gamma \vDash A$ holds, then eventually a sufficiently large finite subset $\Gamma_i$ of $\Gamma$ will have been found such that $\Gamma_i \vDash A$. Eventually, the brute-force search for proofs will succeed in finding a proof of $A$ from $\Gamma_i$.

To make this argument precise, it suffices to show that the set of $L$-formulas $A$ such that $\Gamma \vDash A$ is semidecidable. The following algorithm $N$ semidecides this:

Assumption: $M$ enumerates $\Gamma$.
Input: $A \in (\Sigma^{\text{fo}-L})^*$
Algorithm $N$:
If $A$ is not a syntactically correct formula, reject.
For $i = 0, 1, 2, 3, \ldots$
    Run $M$ for a total of $i$ steps.
    Let $\Gamma_i$ be the subset of $\Gamma$ enumerated by $M$ within $i$ steps.
    For $j = 1, 2, 3, \ldots, i$,
        Let $v_j$ be the $j$-th member of a computable enumeration of $(\Sigma^{\text{fo}-L+})^*$.
        If $v_j$ encodes a syntactically correct FO-proof of $A$ from $\Gamma_i$,
            accept (and halt).
    End-for
End-for

It is clear that the only way the algorithm $N$ can accept $A$ is if some finite subset $\Gamma_i$ of $\Gamma$ logically implies $A$. Conversely, if $\Gamma$ logically implies $A$, then some finite subset $\Pi$ of $\Gamma$ logically implies $A$. Eventually, $M$ will enumerate a subset $\Gamma_{i_0}$ of $\Gamma$ such that $\Pi \subseteq \Gamma_{i_0}$. By construction, $\Pi \subseteq \Gamma_i$ for all $i \geq i_0$. There is a proof of $\Pi \vDash A$; this is also a proof of $\Gamma_i \vDash A$. Such a proof is encoded by some $v_{j_0}$. Hence, once $i \geq \min\{i_0, j_0\}$, the algorithm $N$ will accept and halt.

It follows that the algorithm $N$ correctly semidecides the logical consequences of $\Gamma$. Thus $A$ is computably enumerable. □

It is interesting to compare the above algorithm to the algorithm of Theorem V.37 for propositional logic. The main difference is that the earlier algorithm could use the method of truth tables, whereas the new algorithm dovetails the enumeration of $\Gamma$ with brute-force proof search. The second difference is that the set $\Gamma_i$ in the proof of Theorem V.37 was the first $i$ elements output by $M$ in the enumeration of $\Gamma$. For Theorem V.41, it was important that $\Gamma_i$ is instead the elements enumerated in the first $i$ steps of $M$.[8]

**Theorem V.42.** *Let $L$ be a finite language and $\Gamma$ be a set of $L$-sentences. Let $T = \operatorname{Cn}\Gamma$ be the set of logical consequences of $\Gamma$. Suppose that $T$ is complete and that $\Gamma$ is c.e. Then $T$ is decidable.*

*Proof.* If $T$ is inconsistent, then $T$ is the set of all $L$-sentences and is certainly decidable. So, suppose $T$ is consistent. In this case, $A \in T$ if and only if $\neg A \notin T$.

Theorem V.41 states that $T$ is c.e. Let $\Pi$ be the set of sentences $A$ such that $T \vDash \neg A$. Since $T$ is c.e., there is an algorithm $M$ that enumerates $T$.

By modifying $M$ so as to add a negation sign $\neg$ to the front of each output, we obtain an algorithm $N$ that enumerates $\Pi$; hence $\Pi$ is c.e. The complement $\overline{T}$ of $T$ is the union of $\Pi$ with the set of strings that do not encode valid $L$-sentences. Thus $\overline{T}$ is the union of a c.e. set and a decidable set, and hence (by Exercise V.1), $\overline{T}$ is also c.e. Since $T$ and $\overline{T}$ are both c.e., $T$ is decidable. □

For a more direct proof of Theorem V.42 consider the following algorithm which decides $T$, assuming $T$ is consistent:

> Assumption: $M$ enumerates $T$.
> Input: $A \in (\Sigma^{\text{fo}-L})^*$
> Algorithm $N$:
> If $A$ is not a syntactically correct sentence, reject.
> Run $M$ and watch the sentences enumerated as $M$ runs.
> If $A$ is output by $M$, accept (and halt).
> If $\neg A$ is output by $M$, reject (and halt).
> End-for

This algorithm works since $T$ is consistent and complete; consequently either $A \in T$ or $\neg A \in T$, and in either case $M$ eventually outputs one of $A$ and $\neg A$ and halts.

[8] More precisely, it is important if that $\Gamma_i$ is defined in this way if the algorithm is to work for finite sets $\Gamma$ as well as infinite sets $\Gamma$.

**Example V.43.** The theory of dense linear order (DLO) without endpoints is complete and has a finite set of axioms. Therefore the theory DLO without endpoints is decidable.

**Example V.44.** Let $L$ be the language with equality (=) but no non-logical symbol. Let $\Gamma$ be $\{AtLeast_k : k \geq 2\}$. By Exercise IV.24, the set $\mathrm{Cn}\,\Gamma$ is complete. Clearly, $\Gamma$ is decidable. It follows that $\mathrm{Cn}\,\Gamma$ is decidable.

**Definition V.45.** A theory $T$ is *axiomatizable* if there is a decidable set of sentences $\Gamma$ such that $T = \mathrm{Cn}\,\Gamma$.

It follows from Craig's Theorem (which is stated in Exercise V.22) that if $T$ has a c.e. set of axioms $\Gamma$, then $T$ is axiomatizable. In light of this, we can restate Theorem V.42 as:

**Theorem V.46.** *If $T$ is axiomatizable and complete, then $T$ is decidable.*

## V.6 Undecidability

This section discusses methods for showing that functions are ***not*** computable or relations are ***not*** decidable. It is hard to do this with only an informal definition of the notion of algorithm since the informal definition does not put any apparent limits on the power of algorithms. However, once we give formal, mathematically rigorous, definitions of algorithms, then we can prove theorems about what is not computable.

Nonetheless, by exploiting the full power of the Church-Turing thesis, we can prove results about non-computability and non-decidability in a very general way. The key technical result will establish the non-computability of the "halting problem".

The Church-Turing thesis provides the link between the informal and formal definitions of "algorithm". In fact, even before giving a mathematical definition of algorithms, notably in terms of Turing machines, we can exploit the Church-Turing thesis to prove results about non-computability. Our proofs will be completely rigorous, but will assume some properties about the formal definitions of algorithms that will be provided later, notably when Turing machines are defined.

For simplicity, and concreteness, we let $\Gamma = \{0, 1\}$, and restrict our attention to algorithms that accept strings from $\Gamma^*$ as input and output strings from $\Gamma^*$ (if they generate outputs).[9] The Church-Turing thesis implicitly includes the following assumptions:

**A unified representation scheme for all algorithms:** There is a single set of "instruction types" that suffices to describe an arbitrary algorithm. The informal notion of algorithm required that any algorithm be described by a

[9] This can be done without loss of generality, since a larger alphabet $\Sigma^*$ can be accommodated by representing distinct symbols from $\Sigma$ by distinct finite length codewords from $\Gamma^* = \{0, 1\}^*$.

finite set of unambiguous instructions. The further assumption that there is a unified representation scheme means that there are only finitely many types of instructions. Any particular algorithm can be defined by combining instructions appropriately.

Concomitantly, any algorithm can be described by a finite string, over the alphabet $\Gamma$, that encodes the instructions for the algorithm. This is similar to the way that arbitrary first-order $L$-formulas can be described by strings over $\Sigma^{\text{fo}-L+}$. If $M$ is an algorithm, we write $\ulcorner M \urcorner$ to denote a string that encodes the unambiguous instructions for $M$. The string $\ulcorner M \urcorner$ is called the "Gödel number" of $M$.[10]

**A universal algorithm.** A universal algorithm is an algorithm that can simulate any other algorithm. Specifically, there is a universal algorithm $U(v, w)$ that takes two inputs: The string $v$ is supposed to equal $\ulcorner M \urcorner$ encoding the finite set of instructions for some algorithm $M$. The string $w$ is intended to be the input to $M$.

This universal algorithm $U$, when given inputs $v = \ulcorner M \urcorner$ and $w$, is able to simulate the action of the algorithm $M$ when given the input $w$ in a step-by-step fashion. As it simulates $M(w)$, the algorithm $U$ can detect things such as whether $M$ has accepted or rejected, whether $M$ has halted or is still running, and whether the algorithm has just finished outputting a string.

In other words, the universal algorithm is able to parse the instructions encoded by the string $v = \ulcorner M \urcorner$, and then act upon those instructions to simulate the actions of the algorithm $M$. A universal algorithm is possible only because of the assumption of a unified representation for algorithms.

**Malleability of algorithms.** It is implicit in the notion of a universal algorithm that a string $\ulcorner M \urcorner$ encoding a set of instructions can be parsed and even modified to form a different algorithm. For example, given the Gödel number $\ulcorner M \urcorner$ of an algorithm $M$, one can construct the Gödel number $\ulcorner N \urcorner$ of an algorithm $N$ that acts exactly like $M$ except that $N$ will accept if $M$ rejects, and $N$ will reject if $M$ accepts. Furthermore, this mapping $\ulcorner M \urcorner \mapsto \ulcorner N \urcorner$ is a computable function.

For a second example, suppose $\ulcorner M \urcorner$ is the Gödel number of an algorithm $M$ that accepts a pair of inputs $\langle v, w \rangle$. Then for any particular value of $v$, we can form the Gödel number $\ulcorner M_v \urcorner$ of an algorithm $M_v$ such that $M_v$ takes a single input $w$ and acts exactly like $M$ on input $\langle v, w \rangle$. In other words, $M_v$ is the same as $M$, but with $v$ set to a fixed value. The malleability condition asserts that the map $\langle \ulcorner M \urcorner, v \rangle \mapsto \ulcorner M_v \urcorner$ is computable.

As a third example, suppose $\ulcorner M_1 \urcorner$ and $\ulcorner M_2 \urcorner$ are Gödel numbers for algorithms that compute unary functions $f_1$ and $f_2$. Then an algorithm $N$ can be

[10]The terminology "Gödel number" is a bit inaccurate at the moment, since $\ulcorner M \urcorner$ is a string in $\Gamma^*$, not a number. The terminology reflects the historical fact that the first use of Gödel numbers was based on integers, not strings. Chapter VII will use integers instead of members of $\Gamma^*$ to encode the description of a Turing machine; however, for the time being, our Gödel numbers are strings from $\{0, 1\}^*$.

constructed that computes the composition $f_2 \circ f_1$ by letting $N$ first run $M_1$ and use the output of $M_1$ as the input to $M_2$. The mapping $\langle \ulcorner M_1, M_2 \rangle \mapsto \ulcorner N \urcorner$ can be a computable function.

These three assumptions of a unified representation for algorithmic instructions, a universal algorithm, and the malleability of algorithms may sound very strong, but they should not be particularly surprising. First, given our intuition that modern-day computers can carry out arbitrary algorithms[11], we see immediately that instructions can be presented from some finite instruction set of the type used by computers. The Gödel number of an algorithm is just the source code for the algorithm. That is all that is meant by the existence of a uniform representation. Second, the fact that compilers and interpreters can handle arbitrary programs means that they are in effect universal algorithms. Third, the malleability of algorithms just means that the instructions for an algorithm (as encoded by its Gödel number or its source code) may be locally modified to change the behavior of the program. In particular, the malleability of algorithms allows us to modify the inputs to a program or the output behavior of a program and to compose two programs. It is important to note that these kinds of operations only modify the input/output conventions for programs; they do not require understanding anything at all about the inner workings of programs.

The Church-Turing thesis states that arbitrary algorithms can be implemented via Turing machines. This gives another justification for the assumptions of a unified representation for algorithmic instructions, a universal algorithm, and malleability. Turing machines act like modern-day computers, except with a highly restricted set of possible instructions. Furthermore, Turing's original paper describes a universal Turing machine, which is just a universal algorithm specialized for Turing machines.

As a philosophical side remark, it is worth mentioning that the existence of a universal algorithm is a rather striking fact, even an *a priori* unexpected fact. If one did not know the Church-Turing thesis, it could certainly seem plausible that there is a hierarchy of possible "instruction sets" for algorithms, with the instruction sets increasing in computational strength as one goes up the hierarchy. That is to say, one might envision a hierarchy of increasingly strong programming languages $C_1$, $C_2$, $C_3$, etc., such that each programming language $C_{i+1}$ can express more algorithms than the programming language $C_i$. In this case, there could be no universal algorithm, since it could not be written in any particular programming language $C_i$. This scenario is precluded by the Church-Turing thesis!

The rest of this section will prove undecidability results. The assumption of a unified representation for algorithms is crucial for the results below. However, the full strength of the malleability assumption will be used only for the final result about the halting problem ($\mathrm{Halt}_0$). The assumption that there is a universal algorithm will not be used at all.

[11] Putting aside considerations of efficiency and the limitations of time and space.

## V.6.1 Undecidability via cardinality

The "unified representation" assumption that every algorithm $M$ can be described by some string $\ulcorner M \urcorner$ (over a fixed language $\Gamma$) immediately implies that there are only countably many algorithms, just because there are only countable many possible strings in $\Gamma^*$. However, there are uncountably many subsets of $\mathbb{N}$, and likewise, uncountably many subsets of $\Sigma^*$ for any alphabet $\Sigma^*$. In other words, there are more subsets of $\Sigma^*$ than there are algorithms $M$. This gives immediately the following theorem:

**Theorem V.47.** *For any alphabet $\Sigma$, there is a subset of $\Sigma^*$ which is not decidable. Likewise, there is a relation on $\mathbb{N}$ which is not decidable.*

Proving this theorem using the countable/uncountable distinction is unfortunately not particularly illuminating, since it gives no idea what particular set is not decidable. We'll improve on this below by defining the "halting problem" and proving it is not decidable. That will give a constructive definition of a set that is not decidable. It will also serve as the basis for proving many other sets to not be decidable.

First, however, it is interesting to review the proof that there are uncountably many subsets of $\mathbb{N}$. (Essentially the same proof shows that there are uncountable many subsets of $\Sigma^*$.) This theorem was originally proved by Cantor in 1891 to prove the uncountability of the set of real numbers; its proof hinges on what is called the Cantor diagonal argument. We will use the diagonal argument again for the halting problem.

**Theorem V.48** (Cantor)**.** *There are uncountably many subsets of $\mathbb{N}$.*

*Proof.* The proof is by contradiction. Suppose there are countably many subsets of $\mathbb{N}$. Thus all the subsets of $\mathbb{N}$ can be enumerated in an infinite sequence $X_0, X_1, X_2, \ldots$. That is, each $X_i$ is a subset of $\mathbb{N}$, and every subset of $\mathbb{N}$ is equal to (at least) one of the sets $X_i$.

Define a subset $Y$ of $\mathbb{N}$ by specifying, for all $i \in \mathbb{N}$, that

$$i \in Y \;\Leftrightarrow\; i \notin X_i. \tag{V.1}$$

We claim that it is impossible that $Y = X_i$ for any fixed $i$. The reason is that, if $Y = X_i$, then $i \in Y$ if and only if $i \notin X_i$, and the latter is the same as $i \notin Y$. And, having $i \in Y$ if and only if $i \notin Y$ is impossible. On the other hand, $Y$ is definitely a subset of $\mathbb{N}$, so by the choice of the $X_i$'s, the set $Y$ is equal to some $X_i$.

This is a contradiction. Thus, we have proved that it is impossible that the set of subsets of $\mathbb{N}$ is countable. □

The proof of Theorem V.48 is illustrated in Figure V.2. The figure makes it evident why this proof is called a diagonal argument.

| | 0 | 1 | 2 | 3 | 4 | ⋯ |
|---|---|---|---|---|---|---|
| $X_0$ | **0** | 1 | 1 | 1 | 0 | ⋯ |
| $X_1$ | 1 | **1** | 0 | 1 | 1 | ⋯ |
| $X_2$ | 0 | 0 | **1** | 1 | 1 | ⋯ |
| $X_3$ | 1 | 0 | 1 | **0** | 0 | ⋯ |
| $X_4$ | 1 | 1 | 0 | 0 | **1** | ⋯ |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋱ |
| $Y$ | **1** | **0** | **0** | **1** | **0** | ⋯ |

Figure V.2: An example of how the Cantor diagonal argument works. Each $X_i$ is a subset of $\mathbb{N}$. The entry in the row $X_i$ and column $j$ is equal to 1 if $j \in X_i$ and equal to 0 otherwise. In other words, it is the value of $\chi_{X_i}(j)$. The set $Y$ is defined by Equation V.1 by letting the value for $\chi_Y(i) = 1 - \chi(X_i)$. In other words, $Y$ is defined by flipping the values (shown in boldface) on the diagonal. Therefore, the set $Y$ cannot be equal to any $X_i$.

### V.6.2 Undecidability via the halting problem

We now give an example of a particular, concrete problem set that is not decidable, namely the halting problem. The proof that the halting problem is undecidable will exploit the diagonal argument used in the proof of Cantor's Theorem V.47. Now, however, we must diagonalize against all decidable sets.

By the assumption of uniform representations for algorithms, every algorithm $M$ has a representation $\ulcorner M \urcorner$ which is a string over a (fixed) alphabet $\Gamma$. The string $\ulcorner M \urcorner$ is often called the "Gödel number" of $M$. We assume without loss of generality that *any* string in $\Gamma$ can be viewed a Gödel number for an algorithm. This can be done without loss of generality since any $w \in \Gamma^*$ that does not encode a syntactically correct set of instructions for an algorithm can be treated as being a Gödel number for a fixed trivial algorithm, say the algorithm that ignores its input and immediately rejects and halts.

We define three versions of the halting problem. Recall that $\Gamma = \{0, 1\}$ is being used both as the alphabet for inputs to algorithms and as the alphabet used to encode Gödel numbers of Turing machines. On any given input $w \in \Gamma^*$, an algorithm will either eventually halt (with an output or with an accept/reject decision) or will run forever. We write $M(w)$ for the action of the algorithm $M$ when given the input $w$.

**Definition V.49.** The *halting problem with input* is the binary relation

$$\text{Halt}_1 \;=\; \{\langle \ulcorner M \urcorner, w\rangle : M(w) \text{ halts}\}.^{12}$$

[12] A more correct, but overly pedantic way to define $\text{Halt}_1$ would be that it equals

$$\{\langle v, w\rangle : v \text{ is a Gödel number for an algorithm } M \text{ and } M(w) \text{ halts}\}.$$

The *self-halting problem* is the unary relation defined by

$$\text{Halt}_{\text{Self}} = \{\ulcorner M \urcorner : M(\ulcorner M \urcorner) \text{ halts}\}.$$

The *halting problem* is the unary relation defined by

$$\text{Halt}_0 = \{\ulcorner M \urcorner : M(\epsilon) \text{ halts}\}.$$

Note that something somewhat self-referential is happening in the definition $\text{Halt}_{\text{Self}}$. We have defined that $\ulcorner M \urcorner$ is "self-halting", namely is in $\text{Halt}_{\text{Self}}$, if and only if the algorithm $M$ eventually halts when it is run with its input equal to its own Gödel number $\ulcorner M \urcorner$. There is nothing circular or otherwise problematic about this definition. And, certainly, the other two halting problems are defined straightforwardly enough without any self-referential aspects.

We will use the diagonal method to prove that the self-halting problem is undecidable. The undecidability of the self-halting problem can then be used to prove the undecidability of $\text{Halt}_1$ and $\text{Halt}_0$.

**Theorem V.50.** *The self-halting problem* $\text{Halt}_{\text{Self}}$ *is undecidable.*

*Proof.* Suppose, for the sake of a contradiction that there is some algorithm $M$ that decides the self-halting problem $\text{Halt}_{\text{Self}}$. Modify $M$ to form a new algorithm $N$ by letting $N(w)$ run identically like $M(w)$ until $M(w)$ halts. If $M(w)$ is about to reject, $N(w)$ halts (and, say, accepts). If $M(w)$ is about to accept, $N(w)$ enters an infinite loop and never halts. In other words, $N$ is the following algorithm:

> Assumption: $M$ decides $\text{Halt}_{\text{Self}}$.
> Input: $w \in \Gamma^*$
> Algorithm:
> Run $M$ on input $w$ until it halts.
> If $M(w)$ rejects,
>  Accept and halt.
> If $M(w)$ accepts,
>  Enter an infinite loop and never halt.

Since $N$ is an algorithm, it of course must have a Gödel number $\ulcorner N \urcorner$. Taking the input $w$ to $N$ to be equal to $\ulcorner N \urcorner$, we have

| | | | |
|---|---|---|---|
| $N(\ulcorner N \urcorner)$ halts | $\Leftrightarrow$ | $M(\ulcorner N \urcorner)$ rejects | |
| | $\Leftrightarrow$ | $\ulcorner N \urcorner \notin \text{Halt}_{\text{Self}}$ | Since $M$ decides $\text{Halt}_{\text{Self}}$ |
| | $\Leftrightarrow$ | $N(\ulcorner N \urcorner)$ does not halt | By the definition of $\text{Halt}_{\text{Self}}$ |

This gives that $N(\ulcorner N \urcorner)$ halts if and only if it does not halt, which is of course a contradiction. Therefore, there cannot be an algorithm $M$ that decides the self-halting problem $\text{Halt}_{\text{Self}}$. □

Figure V.3 gives a more pictorial version of the proof, and illustrates how it is a diagonal argument.

| | $v_0$ | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $\cdots$ |
|---|---|---|---|---|---|---|
| $M_0$ | **NH** | H | H | H | NH | $\cdots$ |
| $M_1$ | H | **H** | NH | H | H | $\cdots$ |
| $M_2$ | NH | NH | **H** | H | H | $\cdots$ |
| $M_3$ | H | NH | H | **NH** | NH | $\cdots$ |
| $M_4$ | H | H | NH | NH | **H** | $\cdots$ |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ |
| $N$ | **H** | **NH** | **NH** | **H** | **NH** | $\cdots$ |

Figure V.3: Enumerate all strings in $\Gamma^*$ as $v_0, v_1, v_2, \ldots$. The enumeration order is unimportant, but for the sake of consistency with earlier constructions, they can be taken in order of increasing length, lexicographically ordering strings of a given length. Each $v_i$ is the Gödel number $v_i = \ulcorner M_i \urcorner$ of an algorithm $M_i$. The entry in the row $M_i$ and column $v_j$ is "H" or "NH" depending on whether $M_i(v_j)$ halts ("H") or not ("NH"). The diagonal entries, shown in boldface, indicate whether $M_i(\ulcorner M_i \urcorner)$ halts or not. In the final row $Y$, the column $v_i$ entry indicates whether $N(v_i)$ halts. The final row cannot be equal to any other row in the infinite table, contradicting the assumption that $N$ is an algorithm and hence equal to some $M_i$.

The undecidability of the self-halting problem can be used to prove the undecidability of the other two halting problems. The halting problem with input, $\text{Halt}_1$, is a generalization of the self-halting problem, so its undecidability follows immediately.

**Theorem V.51.** $\text{Halt}_1$ *is undecidable.*

*Proof.* Suppose that $\text{Halt}_1$ is decided by an algorithm $M$. Then the following algorithm $N$ decides $\text{Halt}_{\text{Self}}$:

> Assumption: $M$ decides $\text{Halt}_1$.
> Input: $w \in \Gamma^*$
> Algorithm:
> Run $M(w, w)$ until it halts.
> If $M(w, w)$ accepts, accept and halt.
> If $M(w, w)$ rejects, reject and halt.

The point is that the input $w$ is equal to a Gödel number $\ulcorner M_w \urcorner$ of an algorithm $M_w$, and that $\ulcorner M_w \urcorner \in \text{Halt}_{\text{Self}}$ if and only if $\langle \ulcorner M_w \urcorner, \ulcorner M_w \urcorner \rangle \in \text{Halt}_1$, since these both mean "$M_w(\ulcorner M_w \urcorner)$ halts".

Thus an algorithm $M$ that decides $\text{Halt}_1$ gives immediately an algorithm that decides $\text{Self}_{\text{Halt}}$. But here is no algorithm that decides $\text{Self}_{\text{Halt}}$, so $\text{Halt}_1$ must be undecidable. □

The undecidability of the $\text{Halt}_{\text{Self}}$ and $\text{Halt}_1$ were proved without using the assumption of a universal algorithm, and without using the full strength of the

malleability assumption. In fact, none of the algorithms discussed yet in this chapter have an input which is used as a Gödel number of an algorithm. For example, the two last proofs created an algorithm $N$ assuming the existence of some particular, concrete algorithm $M$. Those algorithms were formulated directly in terms of a fixed $M$; what they did *not* was take $\ulcorner M \urcorner$ as an input and use this to create $\ulcorner N \urcorner$.

The next theorem will need to use the full strength of the malleability assumption. Specifically, the proof will use an algorithm that takes the Gödel number $\ulcorner M \urcorner$ of an algorithm, and modifies $\ulcorner M \urcorner$ to the obtain a Gödel number $\ulcorner N \urcorner$ of a related algorithm.

**Theorem V.52.** *The halting problem* $\text{Halt}_0$ *is undecidable.*

Our proof will use the malleability assumption, but does not need to use the universal algorithm. We use two assumptions about the malleability of the algorithms.

- The first assumption is that there is an algorithm, which given a string $w$ as input, produces the Gödel number of an algorithm $M_w$ that computes the constant function which equals $w$ for all inputs. To be precise, let $g_w$ denote the unary function such that $g_w(v) = w$ for all $v$. We claim that there is a computable function $f$ such that, for all $w$, the value of $f(w)$ is equal to the Gödel number $\ulcorner M_w \urcorner$ of an algorithm $M_w$ which computes the constant function $g_w$.
  To state this assumption in terms of computer programs, this is saying there is an effective procedure which, given any $w$, generates the source code for a program that ignores its input and just outputs $w$. This program might consist primarily of a print statement that outputs the string $w$.

- The second assumption is that there is a computable binary function $f'$ such that the following holds: Whenever $M_1$ is an algorithm that (partial) computes a unary function $h$, and $M_2$ is an algorithm that takes a single input $w$, then $f'(\ulcorner M_1 \urcorner, \ulcorner M_2 \urcorner)$ is equal to the Gödel number $\ulcorner M' \urcorner$ of an algorithm $M'$ that, on input $w \in \Sigma^*$, acts by first computing $v = h(w)$ and then runs $M_2(v)$. In particular, whenever $M_1$ is an algorithm that computes a unary function $h$, and $M_2$ is an algorithm that decides a unary relation $R$, then $f'(\ulcorner M_1 \urcorner, \ulcorner M_2 \urcorner)$ is equal to the Gödel number $\ulcorner M' \urcorner$ of an algorithm $M'$ that decides the unary relation $R \circ h = \{w : h(w) \in R\}$.
  The intuition is that $M'(w)$ acts by first running the algorithm $M_1$, and then running $M_2$ using the output from $M_1$ as the input to $M_2$. To state this in terms of computer programs, it means just means that the source code for the programs $M_1$ and $M_2$ can be combined to form the source code for $M'$ by letting the output of $M_1$ be the input to $M_2$ and then letting the resulting output of $M_2$ be the final output of $M'$.

*Proof.* We prove Theorem V.52 by contradiction. Suppose that $\text{Halt}_0$ is decided by an algorithm $N$. This means that $N(\ulcorner M \urcorner)$ correctly decides whether $M(\epsilon)$ halts, for all inputs $\ulcorner M \urcorner$. Consider the following algorithm $N'$:

Assumption: $N$ decides $\mathrm{Halt}_0$, and $f, f'$ are as above.
Input: $\ulcorner M \urcorner \in \Sigma^*$
Algorithm $N'$:
Run $N$ on input $f'(f(\ulcorner M \urcorner), \ulcorner M \urcorner)$.
If $N$ accepts, then accept.
If $N$ rejects, then reject.

To understand this, note that $f(\ulcorner M \urcorner)$ is the Gödel number of an algorithm that outputs $\ulcorner M \urcorner$. Thus $f'(f(\ulcorner M \urcorner), \ulcorner M \urcorner)$ equals the Gödel number of an algorithm $M'$ that, for any input $w$, running $M'$ on input $w$ yields the same result as running $M$ on input $\ulcorner M \urcorner$. We have:

$$\begin{aligned} N'(\ulcorner M \urcorner) \text{ accepts} &\Leftrightarrow N(f'(f(\ulcorner M \urcorner), \ulcorner M \urcorner)) \text{ accepts} \\ &\Leftrightarrow f'(f(\ulcorner M \urcorner), \ulcorner M \urcorner) \in \mathrm{Halt}_0 \\ &\Leftrightarrow \ulcorner M' \urcorner \in \mathrm{Halt}_0 \\ &\Leftrightarrow M'(\epsilon) \text{ halts} \\ &\Leftrightarrow M(\ulcorner M \urcorner) \text{ halts} \\ &\Leftrightarrow \ulcorner M \urcorner \in \mathrm{Halt}_{\mathrm{Self}} \end{aligned}$$

Since $N'$ accepts its input $\ulcorner M \urcorner$ if and only if $\ulcorner M \urcorner \in \mathrm{Halt}_{\mathrm{Self}}$, the set $\mathrm{Halt}_{\mathrm{Self}}$ is decided by $N'$. But this contradicts Theorem V.50 that $\mathrm{Halt}_{\mathrm{Self}}$ is undecidable. Therefore, $\mathrm{Halt}_0$ is not decidable. □

The above proof implicitly used the notion of a "many-one reduction" from $\mathrm{Halt}_{\mathrm{Self}}$ to $\mathrm{Halt}_0$:

**Definition V.53.** Let $R$ and $S$ be unary relations on $\Sigma^*$ (or on $\mathbb{N}$). A *many-one reduction* from $R$ to $S$ is a computable function $f : \Sigma^* \to \Sigma^*$ (or $f : \mathbb{N} \to \mathbb{N}$) such that for all $w \in \Sigma^*$ (or, all $w \in \mathbb{N}$), we have $w \in R$ if and only if $f(w) \in S$.

If there is a many-one reduction from $R$ to $S$, then $R$ is said to be *many-one reducible* to $S$.

The point of a many-reduction from $R$ to $S$ is that deciding (or semideciding) $R$ can be reduced to the problem of of deciding (or semideciding) $S$.

In the above proof, letting $g(w) = f'(f(w), w)$, the proof argument showed that $g$ is a many-one reduction from $\mathrm{Halt}_{\mathrm{Self}}$ to $\mathrm{Halt}_0$. This was used to show that $\mathrm{Halt}_0$ is undecidable since $\mathrm{Halt}_{\mathrm{Self}}$ is undecidable. The general theorem this is based on is:

**Theorem V.54.** *Suppose $R$ and $S$ are unary relations and that $f$ is a many-one reduction from $R$ to $S$. If $S$ is decidable, then $R$ is decidable. If $S$ is semidecidable, then $R$ is semidecidable. Thus if $S$ is c.e., $R$ is c.e.*

*Proof.* The proof of this is simple: To decide whether $w \in R$, compute $f(w)$ and decide whether $f(w) \in S$. □

**Example V.55.** The proof of Theorem V.51 showed that $\mathrm{Halt}_1$ is undecidable by observing that the function $w \mapsto \langle w, w \rangle$ is a many-one reduction from $\mathrm{Halt}_{\mathrm{Self}}$

to $\text{Halt}_1$. Likewise, the proof of Theorem V.52 showing $\text{Halt}_0$ is undecidable used a many-one reduction from $\text{Halt}_{\text{Self}}$ to $\text{Halt}_0$.

### V.6.3 Self-referential algorithms

A key step in the above proof that $\text{Halt}_0$ is undecidable was to construct an algorithm $M'$ such that for any input $w$, running $M'(w)$ has the same effect as running $M(\ulcorner M \urcorner)$. This means that $M'$ has the effect of "running $M$ on itself", where the "itself" part is referring to $\ulcorner M \urcorner$. This construction can be sharpened to create an algorithm $D_M$ that computes $M(\ulcorner D_M \urcorner)$. This can be phrased informally to say that "$D_M$ is running M on itself", where the "itself" part refers $\ulcorner D_M \urcorner$. Loosely speaking, $D_M$ is doing something with its own Gödel number, namely running $M$ on it. For this reason, $D_M$ is an example of a "self-referential" algorithm, because it acts on its own Gödel number. (The letter "$D$" stands for "diagonal" since $D_M$ can be used in diagonal arguments to prove something is not computable.)

**Example V.56.** Let $M$ be the algorithm that computes the identity function, namely $M(w)$ outputs $w$. Then $D_M$ is an algorithm with Gödel number $\ulcorner D_M \urcorner$ such that, for any input $w$, $D_M(w)$ outputs $\ulcorner D_M \urcorner$. (The input $w$ to $D_M$ is ignored.)

Note that $M(\ulcorner D_M \urcorner)$ and $D_M(w)$ produce the identical output, namely $\ulcorner D_M \urcorner$.

We still need to prove that diagonal algorithms $D_M$ always exist. The existence of an algorithm $D_M$ as in the example that outputs its own Gödel number may seem a little counterintuitive (or maybe even *very* counterintuitive). If algorithms are written as C programs for instance, this means there is a C program that outputs its own source code. Such a program is called a "quine" or a "self-printing program". Quines and self-printing programs are indeed possible. In fact, they are possible for essentially any computer language, not just for C programs. The reader is encouraged to search the internet for examples.

We'll show that the existence of self-referential algorithms follows from the malleability assumptions. For this, we formally state and prove the existence of a self-referential algorithm $D_M$:

**Theorem V.57** (Diagonal Theorem for Algorithms)**.** *Let $M$ be an algorithm that takes a single input $w \in \Sigma^*$. There is an algorithm $D_M$ with Gödel number $\ulcorner D_M \urcorner$ such that, for any input $w$, running $D_M(w)$ produces the same result as running $M(\ulcorner D_M \urcorner)$.*

Note that the input $w$ to $D_M$ is ignored. The only reason for having the input $w$ is that by convention any function must take at least one input.

*Proof.* Let $f$ and $f'$ be the computable functions described in items (a) and (b) immediately following the statement of Theorem V.52. Thus $f(w)$ produces the Gödel number $\ulcorner M_w \urcorner$ of an algorithm $M_w$ that ignores its input and outputs $w$. And $f'(\ulcorner M_1 \urcorner, \ulcorner M_2 \urcorner)$ is equal to the Gödel number of an algorithm that has the same effect as running $M_2$ on the output (if any) of $M_1$. For the

present proofs, the important points are that $f'(f(\ulcorner M_1 \urcorner), \ulcorner M_2 \urcorner)$ is the Gödel number of a Turing machine which does the same thing (halts, rejects, accepts, and/or outputs something) as $M_2(\ulcorner M_1 \urcorner)$, and that the mapping $\langle \ulcorner M_1 \urcorner, \ulcorner M_2 \urcorner \rangle \mapsto f'(f(\ulcorner M_1 \urcorner), \ulcorner M_2 \urcorner)$ is computable.

Form an algorithm $E_M$ so that $E_M(w)$ runs $M(f'(f(w), w))$. Since any $w$ is a Gödel number $\ulcorner N \urcorner$ of an algorithm $N$, this is the same as $E_M(\ulcorner N \urcorner) = M(f'(f(\ulcorner N \urcorner), \ulcorner N \urcorner))$. By construction, $E_M(\ulcorner N \urcorner)$ always yields the same result as running $M$ on the Gödel number of an algorithm that ignores its input and computes $N(\ulcorner N \urcorner)$.[13]

Define $D_M$ to be the algorithm with Gödel number $\ulcorner D_M \urcorner$ equal to $f'(f(\ulcorner E_M \urcorner), \ulcorner E_M \urcorner)$. The algorithm $D_M$ acts by first forming $\ulcorner E_M \urcorner$ and then running $E_M(\ulcorner E_M \urcorner)$. The action of $E_M(\ulcorner E_M \urcorner)$ is to form $f'(f(\ulcorner E_M \urcorner), \ulcorner E_M \urcorner)$ and run $M$ with input $f'(f(\ulcorner E_M \urcorner), \ulcorner E_M \urcorner)$. To summarize, the algorithm $D_M$ acts as follow:

> Assumption: $M$ is an algorithm that takes one input.
> Input: None (or, ignored)
> Algorithm $D_M$:
> Run $M$ on input $f'(f(\ulcorner E_M \urcorner), \ulcorner E_M \urcorner)$.
> (The result of $M$ is used as the result of $D_M$.)

We chose $D_M$ so that $\ulcorner D_M \urcorner$ is equal to $f'(f(\ulcorner E_M \urcorner), \ulcorner E_M \urcorner)$. Thus, the effect of $D_M$ is the same as $M(\ulcorner D_M \urcorner)$. This proves the Diagonal Theorem. □

The Diagonal Theorem can be used to give another proof of the undecidability of $\text{Halt}_0$ that does not require using the fact that $\text{Halt}_{\text{Self}}$ is undecidable.

*Second proof of Theorem V.52.* The proof is again by contradiction. Suppose $\text{Halt}_0$ is decided by an algorithm $N$. Note that $N(w)$ always halts and either accepts or rejects. Modify $N$ to form an algorithm $N'$ so that if $N(w)$ rejects then $N'(w)$ halts and if $N(w)$ accepts, $N'$ enters an infinite loop and never halts. Namely:

> Assumption: $N$ decides $\text{Halt}_0$.
> Input: $w \in \Sigma^*$
> Algorithm $N'$:
> Run $N$ on input $w$.
> If $N(w)$ rejects, then halt.
> If $N(w)$ accepts, then enter an infinite loop and never halt.

Form the algorithm $D_{N'}$ as in the Diagonal Theorem. Then,

$$\begin{aligned} D_{N'}(\epsilon) \text{ halts} &\Leftrightarrow N'(\ulcorner D_{N'} \urcorner) \text{ halts} \\ &\Leftrightarrow N(\ulcorner D_{N'} \urcorner) \text{ rejects} \\ &\Leftrightarrow D_{N'}(\epsilon) \text{ does not halt (since } N \text{ decides } \text{Halt}_0) \end{aligned}$$

This is a contradiction and proves the theorem. □

[13] The names of the variables $M$ and $N$ of variables are swapped here, but $E_N$ is the same as the algorithm $N'$ of the proof of Theorem V.52 except that now running $M$ might have possibly outcomes other than accept and reject.

See Exercise V.38 for another version of the Diagonal Theorem that allows an additional input $v$ as a side parameter.

## V.6.4 Computably inseparable c.e. sets

For another application of the Diagonal Theorem, we prove the existence of "computably inseparable" pairs of computably enumerable sets.

**Definition V.58.** Let $X$ and $Y$ be disjoint subsets of $\Sigma^*$. The sets $X$ and $Y$ said to be *computably separable* if there is a decidable set $Z$ such that $X \subseteq Z$ and $Y \cap Z = \varnothing$. Otherwise, they are *computably inseparable*

The separation conditions can also be written as $X \subseteq Z$ and $Y \subseteq \overline{Z}$. Note that if $X$ and $Y$ are disjoint and computably inseparable then neither one is decidable. Otherwise, we could take one of $Z = X$ or $Z = \overline{Y}$ as the decidable separating set.

**Theorem V.59.** *There are computably enumerable sets $X$ and $Y$ which are computably inseparable.*

*Proof.* Define $\text{Accept}_0$ to be the set of Gödel numbers $\ulcorner M \urcorner$ such that $M(\epsilon)$ eventually accepts. Similarly, define $\text{Reject}_0$ to be the set of Gödel numbers $\ulcorner M \urcorner$ such that $M(\epsilon)$ eventually rejects.

We claim that $\text{Accept}_0$ and $\text{Reject}_0$ are c.e. For this, it is enough to show they are semidecidable. We omit the proof that they are semidecidable here: Exercise V.30(b) asks you to supply it.

Now we show that $\text{Accept}_0$ and $\text{Reject}_0$ are computably inseparable. Suppose for the sake of a contradiction that there is a decidable set $Z$ such that $\text{Accept}_0 \subseteq Z$ and $\text{Reject}_0 \subseteq \overline{Z}$. The complement $\overline{Z}$ is also decidable, by some algorithm $N$. Let $D_N$ be the self-referential algorithm from the Diagonal Theorem V.57. Then

$$\begin{array}{lcll} D_N(\epsilon) \text{ accepts} & \Leftrightarrow & N(\ulcorner D_N \urcorner) \text{ accepts} & \text{by choice of } D_N \\ & \Leftrightarrow & \ulcorner D_N \urcorner \notin Z & \text{by choice of } N \\ & \Rightarrow & \ulcorner D_N(\epsilon) \urcorner \text{ does not accept} & \text{by choice of } Z \end{array}$$

Therefore $D_N(\epsilon)$ does not accept. But also,

$$\begin{array}{lcl} D_N \text{ rejects} & \Leftrightarrow & N(\ulcorner D_N \urcorner) \text{ rejects} \\ & \Leftrightarrow & \ulcorner D_N \urcorner \in Z \\ & \Rightarrow & \ulcorner D_N(\epsilon) \urcorner \text{ does not reject} \end{array}$$

Thus $D_N(\epsilon)$ does not reject. Now we have a contradiction: The algorithm $N$ deciding $\overline{Z}$ has to halt on all inputs, so $D_N(\epsilon)$ must halt and either accept or reject. Therefore no decidable separating set $Z$ can exist. □

### V.6.5 Rice's Theorem

Rice's Theorem states that for any non-trivial property $P$ of computably enumerable sets, it is not decidable whether a given algorithm enumerates a member of $P$. In other words, there is no decision procedure, which given a Gödel number $\ulcorner M \urcorner$, decides whether the set enumerated by $M$ satisfies the property $P$. Informally, this means there is no general way to examine the source code of a program (the Gödel number of an algorithm) and determine reliably what the program does. This can be viewed as a theoretical barrier for things such as checking the correctness of programs, or checking that a program is virus-free, etc.

Let's make this precise with a couple of definitions and a theorem.

**Definition V.60.** Let $P$ be a collection of c.e. subsets of $\mathbb{N}$. We call $P$ a *non-trivial* property of c.e. sets if $P$ is nonempty and does not contain all c.e. subsets of $\mathbb{N}$. In other words, at least one c.e. set is in $P$ but not all c.e. sets are in $P$.

**Example V.61.** The set of nonempty c.e. subsets of $\mathbb{N}$ is a non-trivial property of c.e. sets. The set of finite subsets of $N$ is a non-trivial property of c.e. sets. The set of c.e. subsets of the even integers is a non-trivial property of c.e. sets.

**Definition V.62.** Let $M$ be an algorithm that enumerates integers. We let $L^e(M)$ denote the set enumerated by $M$.

**Theorem V.63.** *Suppose $P$ is a non-trivial collection of subsets of $\mathbb{N}$. Let $X$ be the set of Gödel numbers of algorithms that enumerate a member of $P$,*

$$X := \{\ulcorner M \urcorner : L^e(M) \in P\}.$$

*Then $X$ is not decidable.*

Rice's Theorem is stated (as is usual) about computably enumerable subsets of $\mathbb{N}$. Similar results also hold for partial computable functions. For this, see Exercise V.39

*Proof.* Suppose, for the sake of a contradiction, that $M$ is an algorithm that decides the set $X$. Let $M_1$ and $M_2$ be algorithms such that $L^e(M_1) \in P$ and $L^e(M_2) \notin P$; these exist since $P$ is a non-trivial property of c.e. sets. Let $N$ be the following algorithm:

> Assumption: $M$ decides $X$.
> Input: $w \in \Sigma^*$
> Algorithm $N$:
> If $M(w)$ accepts,
> Run $M_2(\epsilon)$, outputting whatever $M_2(\epsilon)$ outputs.
> If $M(w)$ rejects,
> Run $M_1(\epsilon)$, outputting whatever $M_1(\epsilon)$ outputs.

Use the Diagonal Theorem to form $D_N$. This means that $D_N(\epsilon)$ runs by first checking whether $M(\ulcorner D_N \urcorner)$ accepts. If so, $D_N(\epsilon)$ enumerates (outputs) the strings in $L^e(M_2)$. Otherwise, $M(\ulcorner D_N \urcorner)$ rejects, and $D_N(\epsilon)$ enumerates the strings in $L^e(M_1)$. Thus,

$$\begin{aligned} M(\ulcorner D_N \urcorner) \text{ accepts} &\Leftrightarrow L^e(D_N) = L^e(M_2) \\ &\Leftrightarrow L^e(D_N) \notin P \\ &\Leftrightarrow M(\ulcorner D_N \urcorner) \text{ rejects} \end{aligned}$$

The second equivalence holds since $L^e(D_N)$ is either $L^e(M_1)$ or $L^e(M_2)$ depending on whether $M(\ulcorner D_M \urcorner)$ accepts or rejects.

This gives the desired contradiction. □

We have finished our proofs of undecidability for now. So far, our proofs have worked with an informal notion of algorithms, under the assumptions of uniform representations and malleability of algorithms. The next chapter will define Turing machines. Once Turing machines have been formalized, we will have a justification for the Church-Turing Thesis and it will evident that the corresponding problems for Turing machines are likewise undecidable. For example, the three formulations of the halting problem, $\text{Halt}_{\text{Self}}$, $\text{Halt}_1$ and $\text{Halt}_0$, when recast to work with Gödel numbers of Turing machines are undecidable. Likewise, the Diagonal Theorem, Theorem V.59 on computable inseparability, and Rice's Theorem all still hold when working with Turing machines.

## Exercises

**Exercise V.1.** Prove the following. Let $R$ and $S$ be subsets of $\Sigma^*$ where $\Sigma^* = \{0,1\}$. Let $f, g : \Sigma^* \to \Sigma^*$. (Alternatively, let $R$ and $S$ be subsets of $\mathbb{N}$, and let $f, g : \mathbb{N} \to \mathbb{N}$.)

(a) If $R$ and $S$ are decidable, then $R \cup S$ is decidable.
(b) If $R$ and $S$ are c.e. (computably enumerable), then $R \cup S$ is c.e.
(c) If $R$ and $S$ are decidable, then $R \cap S$ is decidable.
(d) If $R$ and $S$ are c.e., then $R \cap S$ is c.e.
(e) If $f$ and $g$ are computable, then $f \circ g$ is computable.
(f) If $f$ and $g$ are partial computable, then $f \circ g$ is partial computable.
(g) If $f$ is computable and $R$ is decidable, then $\{w : R(f(w))\}$ is decidable.
(h) If $f$ is computable and $R$ is c.e., then $\{w : R(f(w))\}$ is c.e.
(i) If $f$ is partial computable and $R$ is c.e., then $\{w : f(w)\downarrow \text{ and } R(f(w))\}$ is c.e.

**Exercise V.2.** Suppose $R$ is a finite relation (i.e., $R$ is true for only finitely many inputs). Prove that $R$ is decidable. The relation $R$ is called *cofinite* if $\overline{R}$ is finite. Also prove that if $R$ is cofinite, then $R$ is decidable.

**Exercise V.3.** Suppose $R$ and $S$ are $k$-relations on $\Sigma^*$. Define the *symmetric difference* $R \vartriangle S$ of $R$ and $S$ to equal $(R \smallsetminus S) \cup (S \smallsetminus R)$. Suppose that $R \vartriangle S$ is

finite. Prove that $R$ is decidable if and only if $S$ is decidable. Prove that $R$ is c.e. if and only if $S$ is c.e.

**Exercise V.4.** Let $\Sigma = \{a\}$. Let $f : \Sigma^* \to \Sigma^*$ be the unary function so that for all $w \in \Sigma^*$,

$$f(w) = \begin{cases} \epsilon & \text{if the Riemann hypothesis is true} \\ a & \text{if the Riemann hypothesis is false.} \end{cases}$$

Prove that $f$ is computable.

**Exercise V.5.** Let $\Sigma = \{1\}$. Let $f$ be the unary function defined by

$$f(1^n) = \begin{cases} \epsilon & \text{if there is a run of } n \text{ consecutive 7's in the decimal expanson of } \pi \\ 1 & \text{otherwise.} \end{cases}$$

Prove that $f$ is computable.

**Exercise V.6.** Let $R$ and $S$ be unary relations on $\Sigma^*$ (that is, subsets of $\Sigma^*$). Define the *concatenation* $R \circ S$ of $R$ and $S$ to be $\{vw : v \in R \text{ and } w \in S\}$. For example, $\{0, 11\} \circ \{10, 110\} = \{010, 0110, 1110, 11110\}$.

(a) Prove that if $R$ and $S$ are decidable, then $R \circ S$ is decidable.
(b) Prove that if $R$ and $S$ are c.e., then $R \circ S$ is c.e.

**Exercise V.7.** Let $R$ be a unary relation on $\Sigma^*$. Define the *Kleene star* $R^*$ of $R$ to be the set

$$R^* = \{w_1 \circ w_2 \circ \cdots \circ w_k : k \geq 0 \text{ and } w_1, \ldots w_k \in R\}.$$

Thus $R^*$ is the set of strings formed by concatenating zero or more members of $R$. For example, $\{0, 11\}^* = \{\epsilon, 0, 00, 11, 000, 011, 110, \ldots\}$.

(a) Prove that if $R$ is decidable, then $R^*$ is decidable.
(b) Prove that if $R$ is c.e., then $R^*$ is c.e.

**Exercise V.8.** Let $R$ be a binary relation on $\Sigma^*$. Define $S$ to be the set $\{w \in \Sigma^* : \exists v \in \Sigma^* \, (\langle v, w \rangle \in R)\}$. We call $S$ the *projection* of $R$ onto its second coordinate. Suppose that $R$ is c.e. Prove that $S$ is c.e.

**Exercise V.9.** Suppose that $S$ is c.e. Prove that there is a decidable binary relation $R$ such that $S$ is the projection of $R$ onto its second coordinate, namely that $S$ is equal to $\{w \in \Sigma^* : \exists v \in \Sigma^* \, (\langle v, w \rangle \in R)\}$. (This is the converse to the previous exercise. Therefore, $S$ is c.e. if and only if it is the projection of a decidable set.)

**Exercise V.10.** Suppose that $R \subseteq \mathbb{N}$ is enumerated by an algorithm $M$ in increasing order. In other words, $M$ successively outputs $n_1, n_2, \ldots$ so that $R = \{n_1, n_2, n_3, \ldots\}$ and $n_i < n_{i+1}$ for each $i$. Prove that $R$ is decidable. What if the assumption is weakened to have the enumeration be in non-decreasing order, namely to the condition that $n_i \leq n_{i+1}$? Must $R$ still be decidable?

**Exercise V.11.** Assume $R \subseteq \mathbb{N}$ is an infinite decidable set. Prove that there is an algorithm $M$ that enumerates $R$ in increasing order. That is, there is an algorithm that enumerates $R$ in as $n_1, n_2, \ldots$, with each $n_i < n_{i+1}$.

**Exercise V.12.** The following assertions are all ***false*** in general! Proving that they are false requires using a formal definition of algorithm. Instead of giving formal proofs, explain why the types of constructions used in Section V.3 do not work to prove these assertions. Here $R$, $S$ and the $R_i$'s are unary relations.

False Assertion (a): If $R$ is c.e. then $\overline{R}$ is c.e.
False Assertion (b): If $R$ and $S$ are c.e., then $R \setminus S$ is c.e.
False Assertion (c): If each $R_i$ is decidable, then $\bigcup_i \{i\} \times R_i$ is decidable.
False Assertion (d): If each $R_i$ is decidable, then $\bigcup_i \{i\} \times R_i$ is c.e.

**Exercise V.13.** Prove that a $k$-ary relation is computably enumerable if and only if it is the domain of a partial computable function.

**Exercise V.14.** Prove that a set is computably enumerable if and only if is the range of a partial computable function.

**Exercise V.15.** Prove that a set is computably enumerable if and only if it is empty or is the range of a computable function.

**Exercise V.16.** Prove that a set is computably enumerable if and only if it is finite or is the range of an injective computable function. (A function is *injective* if and only if it is one-to-one.)

**Exercise V.17.** Let $f$ be a (total) $k$-ary function, $f : (\Sigma^*)^k \to \Sigma^*$. Prove the following are equivalent.

(a) $f$ is computable.
(b) The graph $G_f$ of $f$ is decidable.
(c) The graph $G_f$ of $f$ is computably enumerable.

(See also Exercises V.31 and V.32.)

**Exercise V.18.** Let $R$ be a set (a unary relation) and $f$ be a unary function. The *preimage* $f^{-1}[R]$ of $R$ under $f$ is the set $\{x : f(x) \in R\}$.

(a) Suppose $R$ is decidable and $f$ is computable. Prove that $f^{-1}[R]$ is decidable.
(b) Suppose $R$ is c.e. and $f$ is partial computable. Prove that $f^{-1}[R]$ is c.e.

**Exercise V.19.** Suppose that $f$ and $g$ are (total) unary integer functions, that $g$ is computable, and that $f(n) \leq g(n)$ for all $n$. Let $G_f$ be the graph of $f$. Prove that the following are equivalent:

(a) $f$ is computable.
(b) $G_f$ is c.e.
(c) $G_f$ is co-c.e.

(See also Exercise V.17.)

**Exercise V.20.** Suppose that $X$ is c.e., that $Y$ is co-c.e., and $X$ is many-one reducible to $Y$. Prove that $X$ is decidable.

**Exercise V.21.** Prove that every infinite c.e. set $R$ has an infinite decidable subset $S$.

**Exercise V.22.** Prove Craig's Theorem:

(a) Suppose $\Gamma$ is c.e. set of propositional formulas. Prove there is a decidable set $\Pi$ of propositional formulas such that $\Gamma \models\!\models \Pi$.

(b) Suppose $\Gamma$ is c.e. set of first-order sentences. Prove there is a decidable set $\Pi$ of first-order sentences such that $\Gamma \models\!\models \Pi$.

[Hint: The proofs for (a) and (b) are identical!]

**Exercise V.23.** Characterize the following as being decidable, c.e., co-c.e., or partial computable, or that it apparently is not any of these. Justify your answers. For the last option, it is not necessary to give a proof that it does not fall into any of the four categories, just to observe that the techniques of this chapter do not put it into any of the four categories. By a "*least* formula" or "*least* proof", we mean that strings are ordered (a) first by length, and (b) second by lexicographic (dictionary) order according to some ordering of the underlying alphabet.

(a) The set of propositional formulas that have a PL-proof.

(b) The function that maps every propositional formula $A$ to its least PL-proof, or to $\epsilon$ if $A$ does not have a proof.

(c) The function that maps a propositional formula $A$ to the least formula $B$ which is tautologically equivalent to $A$.

(d) The set of first-order sentences that have an FO-proof.

(e) For a fixed first-order sentence $A$, the set of sentences that are logically equivalent to $A$.

(f) The set of finite sets $\Pi$ of first-order sentences which are consistent.

(g) The set of finite sets $\Pi$ of first-order sentences which are inconsistent.

(h) The function that maps every first-order sentence $A$ to its least FO-proof.

(i) The function that maps every first-order sentence $A$ to the least sentence $B$ which is logically equivalent to $A$.

**Exercise V.24.** Suppose that $T$ is a consistent, decidable theory. Prove that there is a complete, consistent, decidable theory $S$ extending $T$. [Hint: Show that the proof of Lindenbaum's Theorem can be made effective.]

*Exercises V.25-V.36 require the techniques of Section V.6 (or subsequent sections). In some cases, your answer will need to use a general form of the malleability assumption. Be sure to indicate explicitly where your answers use the malleability assumption. (Basically, this is whenever a Gödel number of an algorithm is being modified to produce a Gödel number for another algorithm.)*

**Exercise V.25.** Prove that binary relation $\text{Accept}_1$ defined by

$$\text{Accept}_1(\ulcorner M \urcorner, w) \quad \Leftrightarrow \quad M(w) \text{ accepts}$$

is undecidable.

**Exercise V.26.** Prove there is a decidable set of propositional formulas $\Gamma$ such that the set $\{i : \Gamma \vDash p_i\}$ is undecidable.

**Exercise V.27.** Prove that there is a consistent, complete first-order theory $T$ which is undecidable. [Hint: You might find it easier to work with a countably infinite non-logical language.]

**Exercise V.28.** Prove that the following assertion is ***false*** in general.

False Assertion: If each $R_i$ is a decidable set, then $\bigcup_{i\in\mathbb{N}}(\{i\} \times R_i)$ is c.e.

[Hint: Use the fact that there is an undecidable set.]

**Exercise V.29.** Let $R$ be a set and $f$ be a unary function. The *image* $f[R]$ of $R$ under $f$ is the set $\{f(x) : x \in R\}$. Prove that there are a decidable set $R$ and a computable function $f$ such that the set $f[R]$ is undecidable.

**Exercise V.30.** For this exercise, you will need to use the assumption that there is a universal algorithm $U$.

(a) Prove that the sets $\text{Halt}_0$, $\text{Halt}_1$ and $\text{Halt}_{\text{Self}}$ are c.e. Conclude that these three sets are not co-c.e.

(b) Prove that the sets $\text{Accept}_0$ and $\text{Reject}_0$ are c.e.. Use this prove that these two sets are not co-c.e.

**Exercise V.31.** Prove that there is a partial computable function $f$ such that graph $G_f$ of $f$ is not decidable. Use this to prove that $G_f$ is not co-c.e.

**Exercise V.32.** Prove that there is a (total) function $f$ such that the graph $G_f$ of $f$ is co-c.e., but $f$ is not computable. [Hint: Let $f(\ulcorner M \urcorner)$ equal $i + 1$ if the algorithm $M$ halts after making exactly $i$ steps, and equal 0 otherwise.]

**Exercise V.33.** Define the integer function $f : \mathbb{N} \to \mathbb{N}$ by

$$f(i) \;=\; |\{\ulcorner M \urcorner \in \Sigma^* : |\ulcorner M \urcorner| = i \text{ and } \ulcorner M \urcorner \in \text{Halt}_0\}|.$$

In other words, $f(i)$ is the number of strings $w$ of length $i$ such that $w$ is the Gödel number $\ulcorner M \urcorner$ of an algorithm $M$ that halts when run on the empty string. Prove that $f$ is not computable.

**Exercise V.34.** Prove that $\text{Halt}_1$ and $\text{Halt}_{\text{Self}}$ are many-reducible to $\text{Halt}_0$.

**Exercise V.35.** Suppose that $R$ is a c.e. relation. Prove that $R$ is many-one reducible to $\text{Halt}_0$. (Because of this property, and the fact that $\text{Halt}_0$ is c.e., $\text{Halt}_0$ is said to be "many-one complete" for the computably enumerable functions.) [Hint: You might first show that $R$ is many-one reducible to $\text{Halt}_1$.]

**Exercise V.36.** Define Total to be the set

$$\text{Total} \;=\; \{\ulcorner M \urcorner : \text{Algorithm } M \text{ halts on all inputs } w\}.$$

In other words, Total is the set of Gödel numbers $\ulcorner M \urcorner$ such that $M$ computes a (total) function.

(a) Give a many-one reduction from $\text{Halt}_0$ to Total.

(b) Give a many-one reduction from $\overline{\text{Halt}_0}$ to Total. $\overline{\text{Halt}_0}$ denotes the complement of $\text{Halt}_0$.

(c) Prove that Total is neither c.e. nor co-c.e.

**Exercise V.37.** Let $X$ be the set $\{\ulcorner M \urcorner : L^e(M) \text{ is finite}\}$.

(a) Prove that $X$ is not decidable.

(b) Prove that $X$ is not c.e.

(c) Prove that $X$ is not co-c.e.

[Hint: Use Rice's Theorem and many-one reductions from $\overline{\text{Halt}_0}$ and $\text{Halt}_0$.]

**Exercise V.38.** Prove the following version of the Diagonal Theorem V.57 which allows a self-referential program with an input $v$:

> Let $M$ be an algorithm that takes two inputs $u, v \in \Sigma^*$. There is an algorithm $D_M$ with Gödel number $\ulcorner D_M \urcorner$ such that, for any input $v \in \Sigma^*$, running $D_M(v)$ produces the same result as running $M(\ulcorner D_M \urcorner, v)$.

**Exercise V.39.** Prove a version of Rice's Theorem for partial computable functions.

(a) Formulate the definition of *nontrivial property* of partial computable functions.

(b) Give two examples of nontrivial properties of partial computable functions.

(c) Suppose $\mathcal{P}$ is a nontrivial property of partial computable functions. Let $X$ be the set of Gödel numbers $\ulcorner M \urcorner$ of algorithms $M$ which partial compute a function in $\mathcal{P}$. Prove that $X$ is undecidable.

[Hint: It can be helpful to use the version of the Diagonal Theorem given in the previous exercise.]

**Exercise V.40.** Give an example of a non-trivial property $P$ of computably enumerable subsets of $\mathbb{N}$ such that the set of Gödel numbers $\ulcorner M \urcorner$ of algorithms $M$ such that $L^e(M) \in P$ is computably enumerable. (This blocks one possible way of generalizing Rice's Theorem in that in some cases there exists a c.e. separating set.)

**Exercise V.41.** For each set $X$ and $Y$ state whether it is (i) decidable, (ii) computably enumerable, and (iii) co-c.e. Justify (prove) your answers. It is OK to use the results of Exercises V.20, V.36 or V.39 to justify your answers. If you find some of the questions to be ambiguous, explain how they are ambiguous and what assumptions you have made to answer the question.

(a) The set $X$ is the set of Gödel numbers of algorithms which partial compute a function that has finite domain.

(b) The set $Y$ is the set of Gödel numbers of algorithms which semidecide a set.

**Exercise V.42.** (Roger's Fixed Point Theorem.) Suppose $f$ is a (total) computable function on $\{0,1\}^*$. Prove that there is an algorithm $M$ with Gödel

number $\ulcorner M \urcorner$, such that, letting $N$ be the algorithm with Gödel number $\ulcorner N \urcorner = f(\ulcorner M \urcorner)$, the algorithms $M$ and $N$ give the same results on all inputs. [Hint: You should use the Diagonal Theorem as extended in Exercise V.38 to take a parameter $v$. You should also use a universal algorithm $U$.]

**Exercise V.43.** Let $f(x, y)$ be a computable, binary function on $\mathbb{N}$. Define $h(x)$ to be the unary partial function on $\mathbb{N}$ such that, for all $x \in \mathbb{N}$,

$$h(x) \;=\; \mu m\,(f(x, m) = 0),$$

i.e.,

$$h(x) \;=\; \begin{cases} \text{the least value } m \text{ such that } f(x,m) = 0 \text{ if such an } m \text{ exists} \\ \text{undefined,} \qquad \text{if no such } m \text{ exists.} \end{cases}$$

(Saying "$h(x) =$ undefined" means $h(x)\!\uparrow$, i.e., $h(x)$ diverges.) Prove that $h$ is partial computable.

**Exercise V.44.** Let $f(x, y)$ be a partial computable, binary function on $\mathbb{N}$. Define $h(x)$ to be the unary partial function on $\mathbb{N}$ such that, for all $n \in \mathbb{N}$,

$$h(x) \;=\; \mu m\,(f(x, m) = 0),$$

i.e.,

$$h(x) \;=\; \begin{cases} \text{the least value } m \text{ such that } f(x,m) = 0 \text{ and such that for all } p < m, \\ \qquad\qquad f(x,p) \text{ converges and is greater than zero} \\ \qquad \text{if such an } m \text{ exists} \\ \text{undefined} \qquad \text{if no such } m \text{ exists.} \end{cases}$$

Prove that $h$ is partial computable.

**Exercise V.45.** Let $f(x, y)$ be a partial computable, binary function on $\mathbb{N}$. Define $g(x)$ to be the unary partial function on $\mathbb{N}$ such that, for all $x \in \mathbb{N}$,

$$g(n) \;=\; \begin{cases} \text{the least value } m \text{ such that } f(x,m) = 0 \text{ and such that for all } p < m, \\ \qquad\qquad f(n,p) \text{ diverges or is greater than zero} \\ \qquad \text{if such an } m \text{ exists} \\ \text{undefined} \qquad \text{if no such } m \text{ exists.} \end{cases}$$

Give an example of partial computable $f$ such that $g$ is not partial computable. (For this reason, the notation "$\mu m\,(\cdots)$" is used for functions $h$ as defined in Exercises V.43 and V.44, but is not used for the function $g$.)

# Chapter VI

# Turing Machines

The previous chapter developed the notion of algorithm in an informal way, including the Church-Turing thesis, the malleability of algorithms, universal algorithms, diagonal theorems, and undecidability. We next define a formal notion of computability based on Turing machines. Turing machines provide a specific model for computation that meets all the criteria for algorithms. Namely, a Turing machine acts in a step-by-step fashion following a finite set of unambiguous definite instructions. A Turing machine accepts a string of symbols as input and performs all of its calculations with operations on symbols.

Specifically, a Turing machine stores its data on a tape that holds symbols from a finite alphabet. It accesses the tape by writing and reading symbols using a tape head that is constrained to move one tape cell left or right at a time. A Turing machine can read and write only a single symbol on its tape at a time. In addition to the tape, there is a finite amount of additional "state" memory.

All this means that Turing machines are simplistic models of computation. Nonetheless, Turing machines provide a powerful model of computation. Indeed, according to the Church-Turing thesis, Turing machines suffice to capture any model of effective computation.

## VI.1 Definitions for Turing Machines

A Turing machine stores data on an infinite tape, each position (also called a "cell") on the tape can hold a single symbol from an alphabet $\Gamma$. The tape contents is accessed with a tape head that can read from or write into a single cell at a time. The tape head can move a single step leftward or rightward, one cell at a time. In addition to the tape contents, a Turing machine maintains a current "state" value; there are only finitely many possible states. The action of the machine is controlled by a "finite control" or "transition function" that describes the action of the Turing machine based on the current state and the current symbol being read by the tape head.

Turing machines have severe restrictions on how they can access data on their tape, since the contents of only one tape cell can be read at a time and since the tape head can only move one cell at a time. Modern-day computers can of course access data in much more flexible ways, especially with the use of random access memory (RAM) to directly access data by its address in memory. Nonetheless, setting aside issues of efficiency, a Turing machine can perform any operation that can be done by a computer. The point is that Turing machines were designed to be a minimalistic model of computation that still captures all possible effective computations.

The key features of a Turing machine are as follows:

(a) The Turing machine stores its data on an infinitely long, linear tape. The tape consists of cells, each cell contains a single symbol from the *tape alphabet* $\Gamma$.[1]

(b) The Turing machine reads one cell at a time from the tape. The current cell being read is called the *tape head position*. The tape head can move one cell left or right at a time.

(c) There is an alphabet $\Sigma$ called the *input alphabet*. We have $\Sigma \subsetneq \Gamma$. There is a special *blank symbol* $\#$ which is in $\Gamma$ but not in $\Sigma$. A Turing machine is initialized with only finitely many non-blank symbols on its tape. If the Turing machine takes a single input $w \in \Sigma^*$, it is initialized with the input $w$ written in consecutive tape cells and the rest of the tape cells hold blank symbols $\#$. A Turing machine that takes $k$ inputs $w_1, \ldots, w_k$ is initialized with the string $w_1\#w_2\#\cdots\#w_{k-1}\#w_k$ and the rest of the tape blank. The tape head position starts at the first (leftmost) symbol of its first input.[2]

(d) The Turing machine has a *finite control* which consists of a finite set $Q$ of *states*, and a finite list of instructions for how the Turing machine should act. These instructions are encoded with a *transition function* $\delta$.

(e) At any given step, the Turing machine is in a particular state $q \in Q$, called the *current state*, and is reading the symbol $a \in \Gamma$ in the tape cell under the tape head. The value $\delta(q, a)$ gives the instructions on what symbol should be written in the current tape cell, whether to move the tape head left or right one square, and what the next state should be.

(f) There is a designated *start state* (or "initial state") often denoted $q_0$; the Turing machine starts running in its start state.

(g) There is a set $Q_{\text{halt}}$ of designated halting states. A Turing machine that is used to decide or semidecide a relation has two halting states, often denoted $q_{\text{acc}}$ and $q_{\text{rej}}$, that are designated accepting and rejecting states. A Turing machine that computes or partial computes a function or enumerates a set, has a single designated *output state*, often denoted $q_{\text{out}}$. When the output state is entered, the output string is equal to the maximum

[1] Multitape Turing machines can use finitely many tapes.

[2] Definition VI.4 gives the formal definition of how inputs are given to a Turing machine.

length string $w$ that is contained in the tape cells starting at the current tape cell and going rightward to the first symbol that is in $\Gamma \setminus \Sigma$. Generally, we impose the further condition that a string $w$ is output when $\#w\#$ appears on the tape with the tape head positioned over the first symbol of $w$. Similar conventions are adopted for a Turing machine that outputs $k$-tuples of strings in $\Sigma^*$ in order to enumerate a $k$-ary relation.[3]

The next definition formalizes the description of a Turing machine as a 8-tuple $(\Gamma, \Sigma, Q, q_0, \delta, Q_{\text{halt}}, q_{\text{acc}}, q_{\text{rej}})$ or 7-tuple $(\Gamma, \Sigma, Q, q_0, \delta, Q_{\text{halt}}, q_{\text{out}})$

**Definition VI.1.** A Turing machine with accept/reject states is specified by an 8-tuple $(\Gamma, \Sigma, Q, q_0, \delta, Q_{\text{halt}}, q_{\text{acc}}, q_{\text{rej}})$ such that

(a) $\Gamma$ and $\Sigma$ are the *tape alphabet* and the *input alphabet*. We have $\Sigma \subset \Gamma$; the *blank symbol* is a member of $\Gamma \setminus \Sigma$.

(b) $Q$ is a finite set of *states*.

(c) $q_0 \in Q$ is the *start state*.

(d) There are two distinct designated accept and reject states $q_{\text{acc}}, q_{\text{rej}} \in Q$.

(e) $Q_{\text{halt}} = \{q_{\text{acc}}, q_{\text{rej}}\}$ is the set of *halting states*.

(f) The *transition function* $\delta$ is a function $\delta : (Q \setminus Q_{\text{halt}}) \times \Gamma \to \Gamma \times \{\text{L}, \text{R}\} \times Q$. Thus, when $q$ is a non-halting state and $a$ is a tape alphabet symbol, $\delta(q, a)$ is equal to a triple $(a', \text{L}, q')$ or $(a', \text{R}, q')$ where $a'$ is a tape alphabet symbol and $q'$ is a state. The symbols L and R mean "move Left" and "move Right".

**Definition VI.2.** A Turing machine that outputs strings is specified by a 7-tuple $(\Gamma, \Sigma, Q, q_0, \delta, Q_{\text{halt}}, q_{\text{out}})$ such that $\Gamma$, $\Sigma$, $Q$, $q_0$ and $\delta$ are as above and such that

(d′) $q_{\text{out}} \in Q$ is the *output state*.

(e′) The set $Q_{\text{halt}}$ of *halting states* is equal to either $\{q_{\text{out}}\}$ or the empty set $\varnothing$.

The meaning of the transition function is that $\delta(q, a)$ gives the instructions on what the Turing machine should do when in state $q$ reading the symbol $a$. The values

$$\delta(q, a) = (a', \text{L}, q') \qquad \text{and} \qquad \delta(q, a) = (a', \text{R}, q')$$

give the instructions that when in state $q$ reading the symbol $a$, the Turing machine should overwrite $a$ with $a'$, move the tape head one cell to the left or right (respectively), and then enter state $q'$.

A *configuration* (also called an 'instantaneous description") of a Turing machine is a complete description of the Turing machine at a given step: this includes specifying the contents of the tape, the current state, and the current position of the tape head. A computation of Turing machine thus consists of a

[3]Definition VI.5 formally defines how a Turing accepts and rejects strings (or $k$-tuples of strings). Definitions VI.9 and VI.12 formally define what it means for a Turing machine to output a string or a $k$-tuple of strings.

sequence of configurations; the initial configuration is in the initial state (usually called $q_0$) and has the input string or strings written on the tape starting at the tape head position. (See Definition VI.4 below.) The final configuration (if any) is in a halting state and is called a halting configuration. (Definitions VI.5, VI.9 and VI.12 describe how a Turing machine returns results.) It is possible that the Turing machine runs forever without entering a halting configuration.

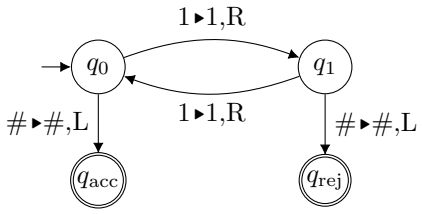
A Turing machine transitions from one configuration to the next as specified by the transition function; see Definition VI.14 below.

**Example VI.3.** For a first, simple example, we construct a Turing machine that accepts exactly inputs of even length. We'll use the input alphabet $\Sigma = \{1\}$ and the tape alphabet $\Gamma = \{1, \#\}$. The action of the machine is informally described as follows. It alternates between two states $q_0$ and $q_1$ while scanning the input string from left to right. It maintains the invariant that it is in state $q_0$ or $q_1$ depending on whether an even or odd number of 1's have been seen so far. When the first blank symbol (#) is reached, the machine enters either $q_{\text{acc}}$ or $q_{\text{rej}}$.

Figure VI.1 shows three configurations of the machine acting on the input 1111. The transition function is given by

$$\begin{aligned} \delta(q_0, 1) &= (1, \text{R}, q_1) \\ \delta(q_1, 1) &= (1, \text{R}, q_0) \\ \delta(q_0, \#) &= (\#, \text{L}, q_{\text{acc}}) \\ \delta(q_1, \#) &= (\#, \text{L}, q_{\text{rej}}) \end{aligned}$$

It is also possible to draw the Turing machine with a *state diagram*:




The state diagram shows a directed edge for each value of $\delta$. An edge from state $q$ to state $q'$ labeled with $a$▸$b$,$D$ with $D$ either R or L indicates that $\delta(q, a) = (b, D, q')$. The arrow coming into the state $q_0$ indicates it is the start state. The double circles on $q_{\text{acc}}$ and $q_{\text{rej}}$ indicate they are halting states.

The Turing machine above accepts (that is, decides) the set $\{w \in \Sigma^* : |w| \text{ has even length}\}$. The next definitions make this formal.

**Definition VI.4.** Let $M$ be a Turing machine with input alphabet $\Sigma$. The machine $M$ is *given* $w \in \Sigma^*$ *as input* if it is started in state $q_0$, the tape is entirely blank other than the sequence of consecutive tape cells containing the string $w$, and the tape head position is over the first (leftmost) symbol of $w$, or over a blank symbol if $w = \epsilon$.

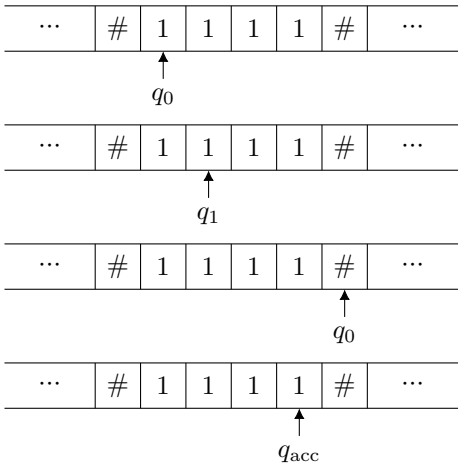



Figure VI.1: The first two configurations and the last two configurations of the Turing machine $M$ from Example VI.3 running on input "1111". (Two configurations are omitted.) $M$ starts in state $q_0$ with the tape head positioned over the leftmost symbol 1. In the next step, $M$ is in state $q_1$ with the tape head positioned over the second 1. Three steps later, it is in state $q_0$ with the tape head over the blank symbol (#) following the final 1. In the final configuration, it has moved left one square and is in state $q_{\text{acc}}$. This is an accepting configuration and thereby a halting configuration. The position of the tape head in an accepting or rejecting configuration is unimportant.

The machine $M$ is *given a $k$-tuple $w_1, \ldots, w_k \in \Sigma^*$ as input* if it is started in state $q_0$, the tape is entirely blank apart from a sequence of consecutive tape cells containing the string $w_1\#w_2\#\cdots\#w_k$, and the tape head position is over the first (leftmost) symbol of $w_1$, or if $w_1 = \epsilon$, it is over the blank symbol where $w_1$ would have started.

**Definition VI.5.** Let $M$ be a Turing machine with accept and reject states. Fix $k \geq 1$. We say $M$ *accepts* a $k$-tuple $w_1, \ldots, w_k$, if, when it is given $w_1, \ldots, w_k$ as input, it eventually enters its accepting state. If it eventually enters its rejecting state, then we say $M$ *rejects* $w_1, \ldots, w_k$.

Note that it is possible that $M$ neither accepts nor rejects $W$ and instead runs forever without halting.

**Definition VI.6.** The notation $L(M)$ denotes the set of strings $w$ such that $M$ accepts $w$. This is called the *language* of $M$.

**Definition VI.7.** A subset $R$ of $\Sigma^*$ is *Turing semidecidable* if there is a Turing machine $M$ such that $L(M) = R$. Note that when given an $w \notin R$, $M$ may either reject $w$ or may run forever without halting. We also say that $M$ *semidecides* $R$.

If there is a Turing machine $M$ such that $L(M) = R$ and such that $M$ halts on all inputs, then we say that $R$ is *Turing decidable*. In this case, $M$ *decides* $R$.

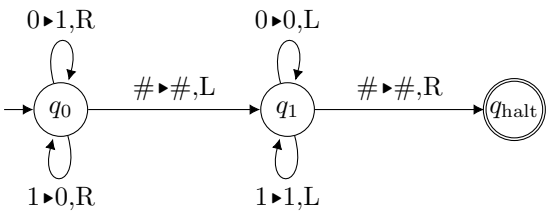
The above Example VI.3 shows that the set of even strings over $\Sigma = \{1\}$ is Turing decidable. By the Church-Turing Thesis, being Turing semidecidable is the same as being semidecidable (and the same as being computably enumerable). Similarly, being Turing decidable is the same as being decidable.

**Example VI.8.** Let $\Sigma = \{0, 1\}$ and define the binary complement function $f$ be the unary function such that $f(w)$ is obtained from $w$ by interchanging 0's and 1's in $w$. For example, $f(1011) = 0100$. A Turing machine $M$ that computes $f$ is as follows: The tape alphabet is $\Gamma = \{0, 1, \#\}$ and the states are $Q = \{q_0, q_1, q_{\text{out}}\}$. The machine $M$ starts at the first symbol of $w$, and first scans from left to right overwriting 0's with 1's, and 1's with 0's. When it reaches the first blank symbol, it scans back to the beginning of $w$. It does this by scanning leftward until reaching a #, and then moving one tape cell rightward before ending up in the output state $q_{\text{out}}$. When $M$ halts, the tape head is positioned over the first symbol of the output string.

The machine $M$ uses state $q_0$ to scan rightward, and then state $q_1$ to scan leftward. The transition function is given by

$$\begin{aligned} \delta(q_0, 0) &= (1, \mathrm{R}, q_0) & \qquad \delta(q_1, 0) &= (0, \mathrm{L}, q_1) \\ \delta(q_0, 1) &= (0, \mathrm{R}, q_0) & \qquad \delta(q_1, 1) &= (1, \mathrm{L}, q_1) \\ \delta(q_0, \#) &= (\#, \mathrm{L}, q_1) & \qquad \delta(q_1, \#) &= (\#, \mathrm{R}, q_{\text{out}}) \end{aligned}$$

and $q_{\text{out}}$ is a halting state, so $Q_{\text{Halt}} = \{q_{\text{out}}\}$. The state diagram for $M$ is:




Three configurations of $M$ when given 1011 as input are shown in Figure VI.2.

The above example gives a Turing machine that computes the binary complement function. This is formalized in the next definitions.

**Definition VI.9** (Outputting a string)**.** Let $M$ be a Turing machine with a halting output state $q_{\text{out}}$ and with input alphabet $\Sigma$. When (and if) $M$ enters $q_{\text{out}}$, the output string is the string $v \in \Sigma^*$ written on the tape, with the tape head positioned at the first (leftmost symbol) of $v$, and with $v$ terminated on the tape with a member of $\Gamma \setminus \Sigma$. Generally, we require w.l.o.g. that $v$ be terminated with #.[4]

[4]The reason for allowing $w$ to be terminated by a symbol from $\Gamma \setminus \Sigma$ other than the blank symbol # is that we want the output string to be a well-defined member of $\Sigma^*$ whenever $M$ enters its output state.

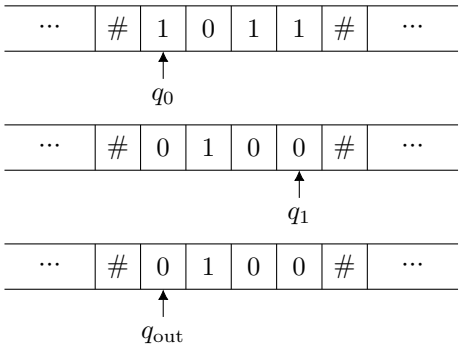



Figure VI.2: The initial configuration, one of the middle configurations, and the halting configuration of the Turing machine $M$ from Example VI.8 running on input "1011". The middle configuration shows when state $q_1$ is entered; it is reached after five steps. The final configuration is a halting output configuration. It is required that the output configuration has the tape head positioned over the first symbol of the output string.

**Definition VI.10.** Let $M$ and $k$ be as above. Then $M$ *partial computes* the function $f$ such that, for all $w_1, \ldots, w_k \in \Sigma^*$,

- If $f(w_1, \ldots, w_k)\downarrow = v$, then the the machine $M$ given $w_1, \ldots, w_k \in \Sigma^*$ as input eventually halts with output $v$;
- If $f(w_1, \ldots, w_k)\uparrow$ then $M$ given $w_1, \ldots, w_k \in \Sigma^*$ as input never halts.

We say that $f$ is *Turing partial-computable*. Note that $f$ is uniquely determined by $M$ and $k$.

If $f$ is total, then we say $M$ *computes* $f$ and that $f$ is *Turing computable*.

By the Church-Turing Thesis, a function $f$ is computable if and only if it is Turing computable; and it is partial computable if and only if it is Turing partial-computable.

We have so far defined the notions of "Turing semidecidable", "Turing decidable", and "Turing (partial) computable". We next give an example of an "Turing enumerable" set.

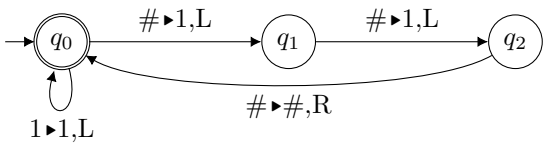
**Example VI.11.** Let $\Sigma = \{1\}$ and consider the set $R = \{1^n : n \text{ is even}\}$. We construct a Turing machine $M$ that enumerates $R$. Let the tape alphabet be $\Gamma = \{\#, 1\}$, let the set of states be $Q = \{q_0, q_1, q_2\}$, let the start state be $q_0$, let the output state (usually called $q_{\text{out}}$) also be $q_0$, and let $Q_{\text{halt}} = \varnothing$ so there is no halting state. Define the transition function $\delta$ so that

$$\begin{aligned} \delta(q_0, \#) &= (1, \mathrm{L}, q_1) & \qquad \delta(q_1, \#) &= (1, \mathrm{L}, q_2) \\ \delta(q_0, 1) &= (1, \mathrm{L}, q_0) & \qquad \delta(q_2, \#) &= (1, \mathrm{R}, q_0) \end{aligned}$$

The values of $\delta(q_1, 1)$ and $\delta(q_2, 1)$ are immaterial for the action of $M$ when started on a completely blank tape, since $M$ will never read a 1 while in state $q_1$ or $q_2$. We therefore omit specifying those values of $\delta$.

The machine $M$ produces infinitely many outputs and never halts. It acts by repeatedly adding two 1's the front of the string on the tape and outputting the result. The states $q_0$ and $q_1$ are used to add the 1's, moving the tape head leftward with each symbol. The state $q_2$ is used to step back to output a string in state $q_0$. As shown in Figure VI.3, the strings that are output are $\epsilon$, then 11, then $\epsilon$, then 1111, etc. The fact that the empty string $\epsilon$ is repeatedly output is OK since a Turing machine is permitted to output duplicate strings when enumerating a set.[5]

The state diagram for $M$ is:




The double circle on $q_0$ indicates it is an output state; however, it is not a halting state. The states $q_1$ and $q_2$ have only a single outgoing edge. This is in keeping with the fact that the values of $\delta(q_1, 1)$ and $\delta(q_2, 1)$ are immaterial for the operation of $M$ when started on a blank tape.

The machine $M$ is said to "enumerate" the set $R$. This is formalized in the next definitions. The first definition gives the conventions for how a Turing machine outputs a $k$-tuple of strings. It coincides with Definition VI.9 when $k = 1$.

**Definition VI.12** (Outputting a $k$-tuple of strings)**.** Let $M$ be a Turing machine with a non-halting output state $q_{\text{out}}$ and with input alphabet $\Sigma$, and let $k \geq 1$. Whenever $M$ enters $q_{\text{out}}$, it outputs a $k$-tuple $\langle w_1, \ldots, w_k \rangle$ of strings from $\Sigma^*$. By convention, the output strings are written on the tape in form $v = w_1 \#_1 w_2 \#_2 \ldots \#_{k-1} w_k \#_k$ where each $w_i \in \Sigma^*$ and each $\#_i \in \Gamma \setminus \Sigma$, and the tape head is positioned on the first symbol of $w_1$ or on the symbol $\#_1$ if $w_1 = \epsilon$. The symbols $\#_i$ do not need to be the blank symbol #; however, it is the usual convention that the $\#_i$ are all equal to #.

When working with a Turing machine with a non-halting output state, we fix a value for $k$ so that the machine is viewed as outputting $k$-tuples of strings.

**Definition VI.13.** Let $M$ and $k$ be as above. Then $M$ *enumerates* the $k$-ary relation $R$ containing exactly the $k$-tuples that are output by $M$. The relation may contain finitely many or infinitely many tuples. We say that $R$ is *Turing enumerable*.

By the Church-Turing Thesis, $R$ is Turing enumerable if and only if it is computably enumerable.

[5]The duplicate outputs could be eliminated by using a four state Turing machine.

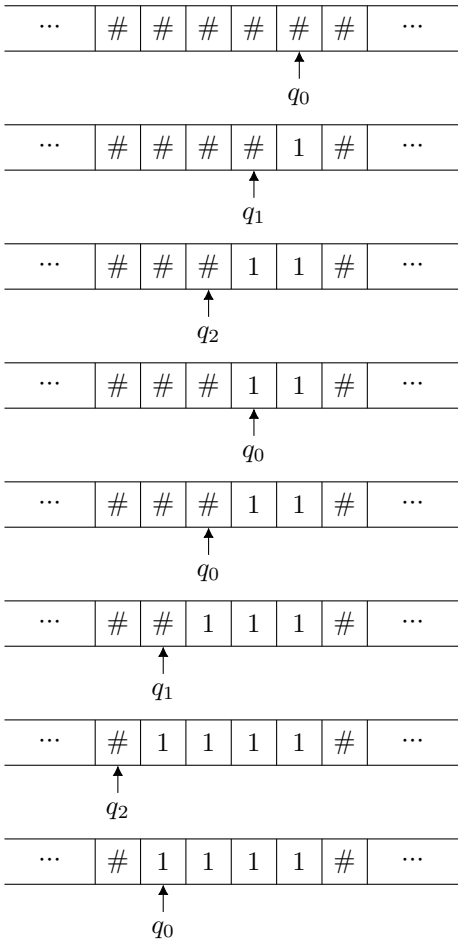



Figure VI.3: The first eight configuration of the Turing machine $M$ from Example VI.11 when started on a completely blank tape. The first, fourth, fifth and eighth configurations are in state $q_0$ and are output configurations; they output $\epsilon$, 11, $\epsilon$, and 1111.

| Output convention | Action | Definition |
|---|---|---|
| $Q_{\text{halt}} = \{q_{\text{acc}}, q_{\text{rej}}\}$; Always halts | Decides a relation | VI.7 |
| $Q_{\text{halt}} = \{q_{\text{acc}}, q_{\text{rej}}\}$; May or may not halt | Semidecides a relation | VI.7 |
| $Q_{\text{halt}} = \{q_{\text{out}}\}$; Always halts | Computes a function | VI.10 |
| $Q_{\text{halt}} = \{q_{\text{out}}\}$; May or may not halt | Partial computes a function | VI.10 |
| $Q_{\text{halt}} = \varnothing$; May enter $q_{\text{out}}$ repeatedly | Enumerates a relation | VI.13 |

Figure VI.4: The different types of Turing machines.

At this point, it should be clear how a Turing machine operates. But, for the sake of completeness, we give the formal definition of how a Turing machine transitions from one configuration to the next:

**Definition VI.14.** Suppose that a Turing machine is in a configuration $C$ such that (a) the machine is in a non-halting state $q$; (b) the tape contains the string $vaw$ where $v, w \in \Gamma^*$, $a \in \Gamma$, $|v| \geq 1$, $|w| \geq 1$, and the rest of the tape is blank (contains only the symbols #); and (c) the tape head is positioned over the symbol $a$. The next configuration $C'$ that follows $C$ in the Turing machine's computation is defined by:

- If $\delta(q, a) = (a', \text{L}, q')$, then $C'$ has $va'w$ written on the tape in place of $vaw$, is in state $q'$, and has the tape head positioned over the last symbol of $v$.
- If $\delta(q, a) = (a', \text{R}, q')$, then $C'$ has $va'w$ written on the tape in place of $vaw$, is in state $q'$, and has the tape head positioned over the first symbol of $w$.

## VI.2 More Constructions of Turing Machines

The Church-Turing Thesis tells us that Turing machines are powerful enough to simulate any algorithm. So far, however, we have seen only very simple Turing machines. In this section, we'll present some more complex examples of Turing machines. Our goal is to show that Turing machines can carry out any computation that can be implemented on an idealized modern-day computer. This will give strong evidence for the Church-Turing Thesis.

**The breadcrumb technique.** The single biggest limitations on Turing machines are that they have only finitely many states, must store their data on a linear tape, and can only access a single tape cell at a time. This makes even

simple tasks difficult, for instance copying a string from one area of the tape to another area of the tape. The next example shows how to do this with the aid of adding two new symbols $0'$ and $1'$ to the tape alphabet. We'll refer informally to the act replacing a 0 or a 1 with $0'$ or $1'$ as "adding a breadcrumb" to the symbol 0 or 1. As the next example shows, a breadcrumb can be used to mark the location of the symbol currently copied.[6]

**Example VI.15.** We design a Turing machine $M$ that when given as input a string $w \in \{0,1\}^*$, makes a copy of $w$ so that the tape now contains $w\#w$. We assume the machine states with its tape head over the first symbol of $w$. The machine will halt with its tape head over the first symbol of the first $w$ in $w\#w$.

The difficulty in making a copy of $w$ is that the machine $M$ has to copy $w$ one symbol at a time. To do this, it has to keep track of which symbol is currently being copied. This will be done by placing a marker or a "breadcrumb" at the symbol being copied. We will enlarge the tape alphabet to be $\Gamma = \{\epsilon, 0, 1, 0', 1'\}$; the two additional symbols $0'$ and $1'$ are used as markers or "breadcrumbs" showing which symbol is currently being copied.

The Turing machine will act as follows. It first places a "breadcrumb" on the first symbol 0 or 1 of $w$, by replacing it with $0'$ or $1'$. It then scans past the end of $w$ and writes that symbol 0 or 1. It then scans back leftward to the breadcrumb, moves the breadcrumb one symbol rightward, and scans rightward to place a copy of the second symbol of $w$. This process continues until all of $w$ is copied. A complete example of this for $w = 010$ is shown in Figure VI.6.

The states used by $M$ are:

- $q_0$ is used to place a breadcrumb on the first symbol of $w$. $q_0$ also detects if $w = \epsilon$.
- $q_{0,1}$ and $q_{0,2}$ are used to scan rightward to the second # symbol and overwrite it with a 0.
  $q_{1,1}$ and $q_{1,2}$ do the same, but overwrite the # with a 1.
- $q_3$ scans leftward to the breadcrumb and removes it.
- $q_4$ adds a breadcrumb to the next symbol of $w$ to the right of the previous breadcrumb if that symbol is 0 or 1. If that symbol is #, then the copying is complete.
- $q_5$ is entered once the copying is complete to place the tape head at the start of the first $w$.
- $q_6$ is the halting output state.

The state diagram for $M$ is shown in Figure VI.5.

The "breadcrumb" technique allows a Turing machine to mark a symbol 0 or 1 by replacing it with $0'$ or $1'$. It is also possible to use multiple types of breadcrumb. For instance, we could also use $0''$ and $1''$ as new symbols, so we could put "double breadcrumbs" on a symbol.

Breadcrumbing is a simple but important tool that allows a Turing machine to correlate the contents of the tape at different locations, in spite of the limitation of having a single tape head. Example VI.15 showed copying from one

[6]The terminology "breadcrumb" alludes to the fairy tale *Hansel and Gretel* in which a trail of dropped breadcrumbs is used to mark a path in the woods.

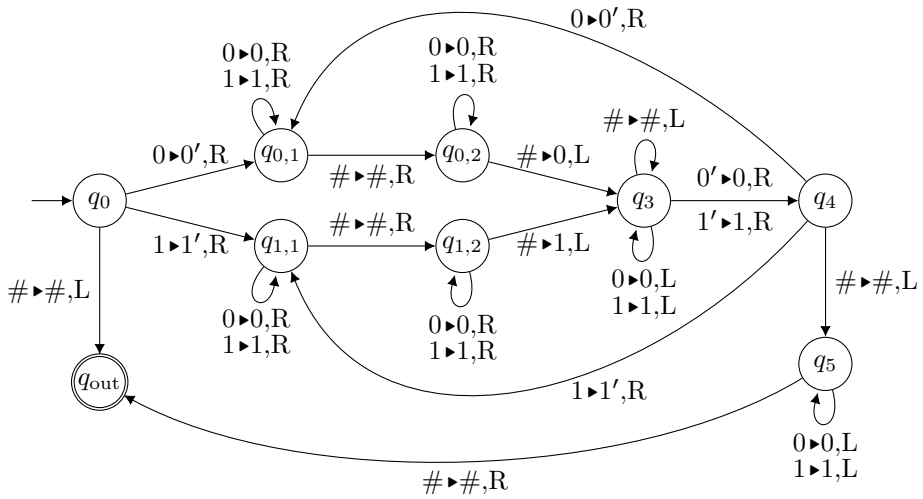



Figure VI.5: The state diagram for the Turing machine from Example VI.15.

location to another using a breadcrumb to make the location being copied from and using a blank symbol at the end of the string to mark the location being copied to. The same technique allows a Turing machine to compare strings at distant locations on the tape:

**Theorem VI.16.** *The following are Turing decidable.*
(a) *The binary relation* $\{\langle w, w\rangle : w \in \{0,1\}^*\}$.
(b) *The unary relation* $\{ww : w \in \{0,1\}^*\}$.

*Proof.* (Sketch) We give just a general overview of how the Turing machines operate. Part (a) asserts that there is a Turing machine $M_1$ such that when $M_1$ is started on the first symbol of $u\#v$ on an otherwise blank tape with $u, v \in \{0,1\}^*$, $M_1$ eventually accepts if $u = v$ and eventually rejects otherwise. $M_1$ starts by placing a breadcrumb on the first symbol of $u$ and scanning leftward to place a breadcrumb on the first symbol of $v$. If those two symbols are different, $M_1$ rejects. Otherwise, $M_1$ shuttles back and forth, moving the breadcrumbs rightward one tape cell at a time and comparing the breadcrumbed symbols. If there is ever a discrepancy in the symbols in $u$ and $v$, or if $u$ and $v$ turn out to not have the same length, then $M_1$ rejects. Otherwise, $M_1$ accepts.

The algorithm for part (b) is similar. The input to the Turing machine $M_2$ is now a single string $u$, and $M_2$ must decide whether $u$ has the form $ww$. The difficulty is that the middle of $u$ is not marked. So, $M_2$ must first locate the midpoint of $u$. To do this, it places breadcrumbs on the first and last symbols of $u$, and then shuttles back and forth, moving the first breadcrumb rightward

$q_0$: # **0** 1 0 # # # # #
$q_{0,1}$: # 0′ **1** 0 # # # # #
$q_{0,1}$: # 0′ 1 **0** # # # # #
$q_{0,1}$: # 0′ 1 0 **#** # # # #
$q_{0,2}$: # 0′ 1 0 # **#** # # #
$q_3$: # 0′ 1 0 **#** 0 # # #
$q_3$: # 0′ 1 **0** # 0 # # #
$q_3$: # 0′ **1** 0 # 0 # # #
$q_3$: # **0′** 1 0 # 0 # # #
$q_4$: # 0 **1** 0 # 0 # # #
$q_{1,1}$: # 0 1′ **0** # 0 # # #
$q_{1,1}$: # 0 1′ 0 **#** 0 # # #
$q_{1,2}$: # 0 1′ 0 # **0** # # #
$q_{1,2}$: # 0 1′ 0 # 0 **#** # #
$q_3$: # 0 1′ 0 # **0** 1 # #
$q_3$: # 0 1′ 0 **#** 0 1 # #
$q_3$: # 0 1′ **0** # 0 1 # #
$q_3$: # 0 **1′** 0 # 0 1 # #
$q_4$: # 0 1 **0** # 0 1 # #
$q_{0,1}$: # 0 1 0′ **#** 0 1 # #
$q_{0,2}$: # 0 1 0′ # **0** 1 # #
$q_{0,2}$: # 0 1 0′ # 0 **1** # #
$q_{0,2}$: # 0 1 0′ # 0 1 **#** #
$q_3$: # 0 1 0′ # 0 **1** 0 #
$q_3$: # 0 1 0′ # **0** 1 0 #
$q_3$: # 0 1 0′ **#** 0 1 0 #
$q_3$: # 0 1 **0′** # 0 1 0 #
$q_4$: # 0 1 0 **#** 0 1 0 #
$q_5$: # 0 1 **0** # 0 1 0 #
$q_5$: # 0 **1** 0 # 0 1 0 #
$q_5$: # **0** 1 0 # 0 1 0 #
$q_5$: **#** 0 1 0 # 0 1 0 #
$q_{\text{out}}$: # **0** 1 0 # 0 1 0 #

Figure VI.6: The execution of the Turing machine from Example VI.15 when run on the input $w = 010$. We use a compact representation for the configurations, showing the state and the tape contents. Underlined bold-font symbols show the position of the tape head.

one tape cell at a time and the second breadcrumb leftward one tape cell at a time. When the breadcrumbs meet in the middle of $u$, the machine $M_2$ can determine if $|u|$ is even or odd. If $|u|$ is odd, $M_2$ rejects. Otherwise $M_2$ can place a breadcrumb on the first symbol of the second half of $u$, and scan back to the beginning of $u$ to place a breadcrumb on the first symbol of $u$. Now $M_2$ shuttles back-and-forth, comparing symbols from the first half of $u$ and the second half of $u$, and moving the breadcrumbs rightward one tape cell at a time. If a discrepancy is found, then $M_2$ rejects. The process stops once the second breadcrumb passes the last symbol of $u$; at this point $M_2$ accepts. □

**Functions and relations on integers.** We now introduce conventions for how Turing machines can define functions and relations on (nonnegative) integers. When working with integers, the input alphabet is $\Sigma = \{0, 1\}$ and integers are encoded with their binary representations. The definition is a repeat of Definition V.13.

**Definition VI.17.** If $n \in \mathbb{N}$, then $str(n)$ is the usual binary representation of $n$; thus $str(n)$ is in $\{0, 1\}^*$. By convention, $str(0)$ is the string 0. For $n > 0$, $str(n)$ does not include any leading 0's.

For $w \in \{0, 1\}^*$, $num(w)$ is the integer $n$ for which $w$ is a binary representation; it is permitted that $w$ may contain leading zeros. By convention, the empty string $\epsilon$ is a binary representation of 0. For example, $num(101) = num(0101) = num(00101) = 5$ and $num(\epsilon) = num(0) = num(00) = 0$.

Turing machines define functions and relations on the integers by working with their binary representations.

**Definition VI.18.** Let $R \subseteq \mathbb{N}^k$ be a $k$-ary relation on the integers. A Turing machine $M$ *semidecides* $R$ if, for all $n_1, \ldots, n_k \in \mathbb{N}$, $M(str(n_1), \ldots, str(n_k))$ accepts if and only if $R(n_1, \ldots, n_k)$ holds. The machine $M$ *decides* $R$ if in addition, for all $n_1, \ldots, n_k \in \mathbb{N}$, $M(str(n_1), \ldots, str(n_k))$ rejects if $R(n_1, \ldots, n_k)$ does not hold.

**Definition VI.19.** Let $f : \mathbb{N}^k \to \mathbb{N}$ be a $k$-ary function. A Turing machine $M$ *partial computes* $f$ if, for every $n \geq 0$, (a) $M(str(n))$ halts if only if $f(n)\downarrow$ and (b) if $M(str(n))$ halts, it outputs a string $w$ such that $num(w) = f(n)$. If $f$ is total, then $M$ *computes* $f$.

**Definition VI.20.** Let $R \subseteq \mathbb{N}^k$ be a $k$-ary relation on the integers. A Turing machine $M$ *enumerates* $R$ if $R$ is the relation containing exactly the tuples $\langle num(w_1), \ldots, num(w_k)\rangle$ such that $M(\epsilon)$ outputs $\langle w_1, \ldots, w_k\rangle$.

These three definitions depend only on what the Turing machines do for inputs of the form $str(n)$. Since the *num* function allows leading zeros, any output of the Turing machines can be interpreted as an integer or a $k$-tuple of integers.[7]

[7] Smullyan's *dyadic representation* provides an alternative representation of integers with

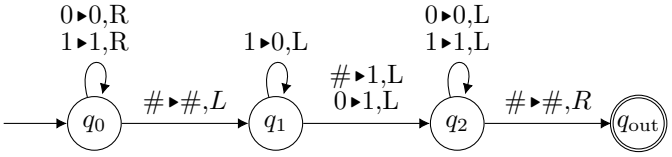
**Theorem VI.21.** *The following functions on* $\mathbb{N}$ *are Turing computable:*

(a) *The successor function* $S(n) = n + 1$.

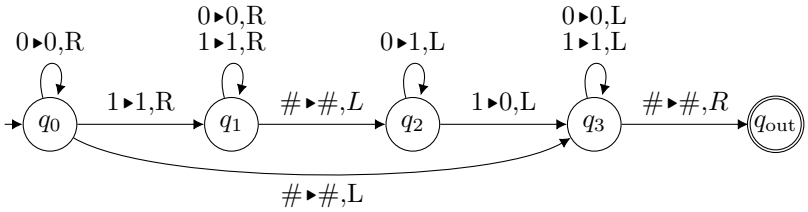
(b) *The predecessor function*

$$P(n) \;=\; n \dot{-} 1 \;=\; \max\{0, n-1\} \;=\; \begin{cases} n-1 & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

*Proof.* We first exhibit a Turing machine $M_S$ for the successor function; in fact, we'll design it so it can take any binary representation $w$ as input, not just the representations $str(n)$. The algorithm is used on the usual "gradeschool" algorithm for adding 1 in binary representation. The machine $M_S$ starts at the left symbol of $w$; the rest of the tape is blank. $M_S$ first scans to the right end of $w$, namely the low-order bit, using state $q_0$. State $q_1$ scans back leftward changing low order 1's to 0's until a 0 is encountered. That 0 is changed to 1; then state $q_2$ scans back to the start of (the now-updated) $w$. The state diagram for $M_S$ is:




Now we exhibit a Turing machine $M_P$ for the predecessor function. It works similarly to $M_S$, but now, scanning from the left end, low-order 0's are changed to 1's, and the first encountered 1 is changed to a 0. In addition, $M_P$ has to check (with state $q_0$) whether its input represents the integer 0. In this case, the output is the same as the input. The state diagram for $M_P$ is:




The purpose of state $q_0$ is to detect when the input is a binary representation for zero; in that case, the Turing machine transitions on the edge from $q_0$ to $q_3$, and the integer is not decremented. □

strings over $\{0,1\}$. In dyadic representation, strings are ordered first by length, and then lexicographically. For instance, the integers 0 through 8 are represented by the strings $\epsilon, 0, 1, 00, 01, 10, 11, 000, 001$. The general formula is that $a_\ell a_{\ell-1} \cdots a_1 a_0$ is the dyadic representation of $\sum_{i=0}^{\ell} a_i \cdot 2^i$. This has the advantage that, unlike the situation for binary representations, integers have unique dyadic representations. We use binary representation, however, since it is much more familiar.

**Implementation of loops.** We next describe how to perform addition and subtraction of integers. This will also illustrate how to implement loops with Turing machines.

We describe algorithms for addition and subtraction based on repeatedly incrementing and decrementing integers by 1. Of course, this is much less efficient than, say, using the grade-school algorithms adapted to base 2. But the efficiency or inefficiency of the algorithms is not the point: we are only interested in the question of computability.

**Theorem VI.22.** *The following functions on* $\mathbb{N}$ *are Turing computable:*

(a) *The addition function* $\langle m, n\rangle \mapsto m + n$.

(b) *The truncated subtraction function*

$$\langle m, n\rangle \;\mapsto\; m \dot{-} n \;=\; \max\{0, m-n\} \;=\; \begin{cases} m-n & \text{if } m \geq n \\ 0 & \text{if } m < m \end{cases}$$

*Proof.* The idea for the addition function is to repeatedly increment the integer $m$ by 1, a total of $n$ times. To do this, the machine repeatedly decrements $n$ by 1 and increments $m$ by 1. The process stops once $n$'s value reaches zero.

The Turing machine starts with the tape containing "$\#u\#v\#$"; the tape head is on the first symbol of $u$ and the tape is otherwise blank. The strings $u$ and $v$ are binary representations of $m$ and $n$. To slightly abuse notation, we can also say that the tape contents are equal to "$\#m\#n\#$". To further abuse notation, we can talk about the Turing updating the values of $m$ and $n$ in place on the tape. Thus to add $m$ and $n$, the Turing machine implements the following steps:

1. Scan rightward to the first symbol of $n$.
2. Decrement $n$ by 1. If $n$ is discovered to already be zero (before decrementing), go to step 5.
3. Scan leftward to the first symbol of $m$.
4. Increment $m$ by 1. Go to step 1.
5. Scan leftward to the first symbol of $m$ and halt. This value of $m$ is the desired sum.

The actions of incrementing $m$ and decrementing $n$ can be carried out by the Turing machines given by the previous theorem. Note that the Turing machine for decrementing $n$ already includes the test of whether $n$ is zero (as the edge from $q_0$ to $q_3$ will be traversed if it is attempted to subtract from zero). The other actions, scanning leftward to $m$ or rightward to $n$ are easily implemented with a couple of additional states. As $m$ is incremented, it may occupy more tape cells, but this does not cause any problem as the tape is presumed to be blank to the left of the input $\#m\#n\#$ so there is room for $m$ to grow. The entire Turing machine for computing $m + n$ is shown in Figure VI.7.

The truncated subtraction function is implemented similarly, except $m$ is decremented instead of incremented. It can halt once either $m$ or $n$ is decremented to 0. □

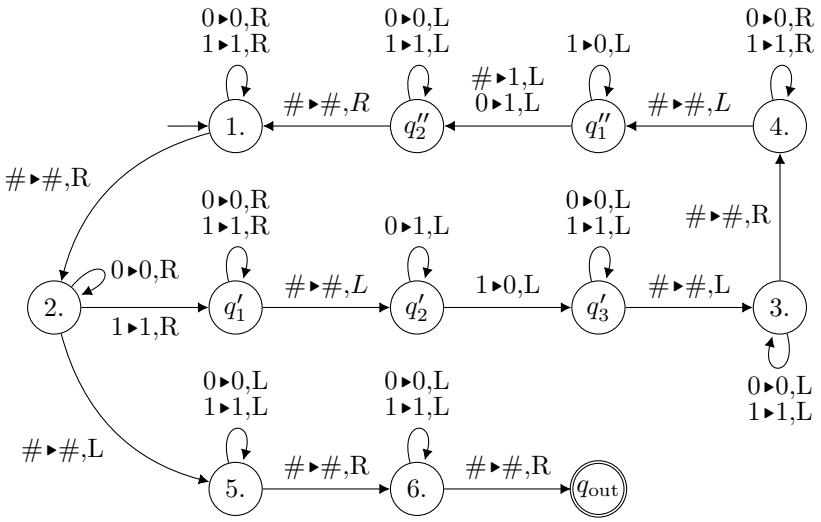



Figure VI.7: A Turing machine computing $\langle m, n\rangle \mapsto m+n$. This implementation assumes that, if $m = 0$, it is not encoded by the empty word $\epsilon$. The states $q_1', q_2', q_3'$, respectively the states $q_1'', q_2''$, are the internal states from the Turing machines for the predecessor and successor functions of Theorem VI.21.

This Turing machine was formed in a modular fashion from the Turing machines for successor or predecessor. It therefore contains some inefficiencies. In particular, states 3. and 4. could be eliminated, since state 3. scans from the right end of $n$ to the left end, and state 4. scans back to the right end. Similarly, states $q_2''$ and 1. could be combined since $q_2''$ scans to the left end of $n$ and state 1. scans back to the right end.

**Theorem VI.23.** *The integer multiplication function $\langle m, b\rangle \mapsto m \cdot n$ is Turing computable.*

*Proof.* We give an informal proof of this, similar to the proof of Theorem VI.22. The multiplication will be carried out by iterated addition. Since addition is carried out by looping and incrementing, this means that a doubly-nested loop is used to implement multiplication by starting with 0 repeatedly adding 1.

Continuing to abuse notation, the Turing machine starts with tape contents $\#m\#n\#$. It further initializes the tape by writing, to the left of the input, a copy of $m$ called $m'$ and an integer $N$ set to 0. At this point, the tape contents is equal to $\#N\#m'\#m\#n\#$. The value $m'$ will vary in value between 0 and $m$. The value $N$ will be repeatedly incremented, taking up space to the left as it grows. The Turing machine overall runs as follows:

1. Initialize the tape to hold $\#N\#m'\#m\#n\#$ with $N = 0$ and $m' = m$.
2. If $n = 0$, halt and output $N$. Otherwise decrement $n$.
3. Add $m'$ to $N$. This results in $m' = 0$.
4. Set $m'$ equal to $m$, and go to step 2.

We leave it to the reader to confirm that a Turing machine can be designed to carry out these steps. □

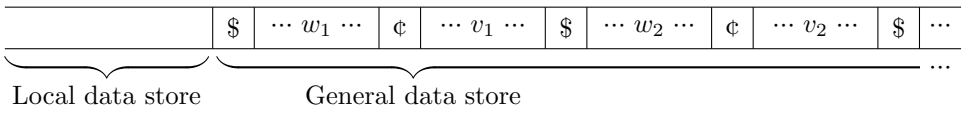
## VI.3 The Church-Turing Thesis

So far we have seen that Turing machines can handle several programming techniques. This includes conditional testing and "if ⋯ then ⋯ else ⋯" type controls. It also includes basic integer operations and controlled loops. We next argue that Turing machines satisfy the conditions of the Church-Turing Thesis. For this, we argue that Turing machines can emulate general-purpose memory access, where data values are stored and retrieved via memory locations or variable names.

Conditional testing, controlled loops, and general memory access are enough to emulate the action of full-featured programming languages. Thus, we argue that this shows that Turing machines fulfill the Church-Turing Thesis.

To be concrete, we propose a particular method of implementing general-purpose memory access on a Turing machine. The data values will be strings of symbols, say over the alphabet $\{0, 1\}$. There will three other special symbols \$, \$′ and ¢. The machine will also use strings as addresses of memory locations, or equivalently as names of variables. The Turing machine will read and write values into the general-purpose memory into locations using the addresses or variable names.

To be specific, the tape can be set up in the form



The general data store extends to the right on the tape. It holds a sequence of pairs of values $w_i, v_i$ where $w_i$ is intended to be a "address" or "variable name" and $v_i$ is the data value stored as the location $w_i$. (In this scenario, the $w_i$'s are not required to be in sorted order.) The local data store is intended to hold a small number of addresses and data values in a working memory area. Data items are fetched from the general data store to the local data store, where their values can be used and updated. Values updated in the local data store can then be copied back into the general data store. This style of computation, where data is fetched or downloaded from a general data store to a local working area and copied or uploaded back to the general store, reflects the "Von Neumann architecture" used by modern-day computers.

Data in the local data store can be updated using copying, comparisons, conditional branching, and loops. To fetch data from the general data store, the Turing machine starts with an address value $w$. It sequentially compares $w$ to $w_1, w_2, w_3, \ldots$ until it finds a $w_i$ which is equal to $w$. The sequential comparison can keep track of the current $w_i$ by placing a breadcrumb on its leading \$ (say by replacing the preceding \$ with \$′). It further uses breadcrumbs to compare $w$ and $w_i$ symbol-by-symbol. Once a $w_i$ equal to $w$ is found, the accompanying $v_i$ can be copied the local data store, again using breadcrumbs to track the current symbol that the is being copied. The right end of the general data store needs to be marked in some way, say by the presence of a \$′. This allows detecting when an address $w$ is not yet present in the general data store; in this case, a new entry may be created in the data store for $w$, say with the value $v = \epsilon$.

Similar techniques can be used to copy values from the local data store into the general data store. There is an added complication that a new value $v$ overwriting a former value $v_i$ may have a different length. This can be accommodated either by shifting the right end of the general data store to create the right amount of space to hold $v$ or by invalidating the old entry $w_i ¢ v_i$ (say, by overwriting it with \$'s) and creating a new entry $w_i ¢ v$ at the right end of the general data store.

At this point, we hope the reader is convinced that a Turing machine can simulate a general-purpose computer, and that thereby the Church-Turing Thesis holds with Turing machines providing an adequate mathematical model for arbitrary effective computation.

Of course, this simulation of a general-purpose computer by a Turing machine is highly inefficient. Indeed, the Von Neumann bottleneck, namely the shuttling of data back and forth between the general data store and the local data store, is really inefficient due to the linear nature of the Turing machine's tape. However, the point is not to give an *efficient* simulation of general-purpose computers or arbitrary algorithms by Turing machines. We are only interested *effective* simulations, and are not concerned with practical considerations such as the running time of the Turing machine.

## VI.4 Universal Turing Machines

A universal Turing machine is a Turing machine $U$ that can simulate any Turing machine $M$ when given the Gödel number of $M$. In light of the Church-Turing thesis, a universal Turing machine must exist. This is just because it is possible to write an algorithm that uses $\ulcorner M \urcorner$ to simulate $M$ and because that algorithm can be implemented on a Turing machine.

At this point, it is hoped that the reader is convinced that the Church-Turing Thesis is true and thus convinced that there is a universal Turing machine. We therefore will not attempt to describe a universal Turing machine. What we will do instead is propose one possible way that the inputs to a universal Turing machine can be formatted.

For this, here is one possible way of defining the Gödel number $\ulcorner M \urcorner$ of a Turing machine $M$. In this proposal, the Gödel number $\ulcorner M \urcorner$ can be viewed as a string of bits that specify the following information:

- The cardinality $|\Gamma|$ of the tape alphabet $\Gamma$.
- The cardinality $|\Sigma|$ of the input alphabet $\Sigma$.
- The number $|Q|$ of states. W.l.o.g. the states are named $q_0, q_1, \ldots, q_{|Q|-1}$ and $q_0$ is the start state.
- The number $|Q_{\text{halt}}|$ of halting states. If this is 0 or 1, there is a single output state $q_{\text{out}}$ which, w.l.o.g., is equal to $q_{|Q|-1}$. If $|Q_{\text{halt}}| = 2$, there are two output states $q_{\text{acc}}$ and $q_{\text{rej}}$ which w.l.o.g. are equal to $q_{|Q|-2}$ and $q_{|Q|-1}$.
- The transition function $\delta : (Q \setminus Q_{\text{Halt}}) \times \Gamma \to |\Gamma| \times \{\text{R}, \text{L}\} \times |Q|$ has $(|Q| - |Q_{\text{halt}}|) \cdot |\Gamma|$ many values. Letting $\ell_\Gamma = \lceil \log_2 |\Gamma| \rceil$ and $\ell_Q = \lceil \log_2 |Q| \rceil$, each value of $\delta$ can be encoded with $\ell_\Gamma + 1 + \ell_Q$ many bits. By concatenating the binary strings encoding the values of $\delta$, the entire function $\delta$ is encoded by a single binary string $\text{Encode}_\delta$ of length

$$\ell_\delta \;:=\; (|Q| - |Q_{\text{halt}}|) \cdot |\Gamma| \cdot (\ell_\Gamma + 1 + \ell_Q).$$

The Gödel number $\ulcorner M \urcorner$ of the Turing machine $M$ must encode the above information about $M$. For the sake of concreteness, $\ulcorner M \urcorner$ can then be defined to be the binary string

$$1^{|\Gamma|}01^{|\Sigma|}01^{|Q|}01^{|Q_{\text{Halt}}|}0\text{Encode}_\delta$$

of length $|\Gamma| + |\Sigma| + |Q| + |Q_{\text{Halt}}| + 4 + \ell_\delta$.

The universal Turing machine $U$ will take two inputs $\ulcorner M \urcorner$ and a string $w$: the result of running $U(\ulcorner M \urcorner, w)$ is the same as the result of running $M(w)$. This exposes another complication. Namely, the universal machine $U$ uses a particular input alphabet $\Sigma_U$; but in the definition above, the size of the input alphabet $\Sigma$ was allowed to be specified in the Gödel number $\ulcorner M \urcorner$. This means there is a mismatch in the inputs $w$ that $M$ can accept and the inputs $w$ that $U$ can accept.

There are a couple of ways around this. The first, and perhaps the best, is to just mandate that all machines have the same alphabet. In this approach, $U$ is a universal machine for those Turing machines $M$ with the same input alphabet

$\Sigma = \{0,1\}$ as $U$. The second approach would be to allow machines $M$ to have arbitrary input alphabets even though $U$ must use a fixed input alphabet, say $\Sigma = \{0,1\}$. In this approach, an input $w$ to $M$ is encoded as a binary string $w^*$ and then $U(\ulcorner M \urcorner, w^*)$ simulates the action of $M(w)$. If $U(\ulcorner M \urcorner, w^*)$ produces the result that $M$ outputs $v$, then $U$ outputs the encoded value $v^*$ of $v$.

A natural way to define the encoding $w^*$ of $w$ is to use a fixed length binary encoding wherein each symbol $a$ of $\Sigma$ is encoded by a distinct string $v_a \in \{0,1\}^*$ of length exactly $\ell_{|\Sigma|}$. If $w$ is $a_1 a_2 \cdots a_k$ where each $a_i \in \{0,1\}$, then $w^*$ is just the concatenation $v_{a_1} v_{a_2} \cdots v_{a_k}$ of the code words for the $a_i$.

**Using the tape alphabet $\{0,1,\#\}$** We now restrict attention to Turing machines which use input alphabet $\Sigma = \{0,1\}$. The universal Turing machine $U$ uses some fixed tape alphabet $\Gamma_U$, where $\Gamma_U \supseteq \{0,1,\#\}$. It is interesting to ask whether a universal Turing machine $U$ can be designed with $\Gamma_U = \{0,1,\#\}$. The answer is yes, but this may not be immediately evident. Indeed, the earlier arguments justifying the Church-Turing Thesis depended on having Turing machines that can copy strings long distances, or compare two strings on the tape that are not located near each other on the tape. Those Turing machines were built with the aid of "breadcrumbs", namely the tape alphabet contained symbols such s $0'$ and $1'$, thus the tape alphabet $\Gamma$ for these machines is larger than just $\{0,1,\#\}$.

To address this, we describe a way to transform an arbitrary Turing machine $M_1$ with input alphabet $\Sigma = \{0,1\}$ and with arbitrary tape alphabet $\Gamma_1$ to an equivalent Turing machine $M_2$ with tape alphabet equal to $\Gamma_2 = \{0,1,\#\}$. The idea is to encode symbols from $\Gamma_1$ by fixed length binary codewords over the alphabet $\Gamma_2$. Each code word will have length $\ell = \lceil \log_2 |\Gamma_1| \rceil$. Without loss of generality, the codewords for the symbols 0 and 1 are $0^\ell$ and $1^\ell$; in addition, $M_2$ should use $\ell$ many #'s in place of a 0/1-codeword for #. A configuration of $M_1$ with the tape contents equal to $\cdots \# a_1 a_2 \ldots a_k \# \cdots$ and the tape head positioned on the symbol $a_i$ will correspond to the configuration of $M_2$ with tape contents $\cdots \# v_1 v_2 \cdots v_k \# \cdots$ where each $a_i$ has been replaced with its codeword $v_i$, and with $M_2$'s tape head positioned over the first symbol of $v_i$.

$M_2$ simulates $M_1$ by keeping track of $M_1$'s current state and simulating a single step of $M_1$ by the following steps: (a) First taking $\ell - 1$ steps to scan rightward and read the current codeword, using $2^\ell - 1$ many states. This lets $M_2$ know what symbol $M_1$ is currently reading. (b) Then moving leftward $\ell - 1$ steps to overwrite the codeword with the new codeword for the symbol $M_1$ would write. And, (c) finally moving leftward or rightward $\ell$ tape cells, according to whether $M_1$'s tape head moves left or right, to position the tape head to be ready to simulate the next step of $M_1$.

The Turing machine $M_2$ can thus emulate the operation of $M_1$ using the tape alphabet $\Gamma_2$. But before $M_2$ can begin emulating the operation of $M_1$, it needs to convert its input string $w \in \Sigma^*$ into its encoded version $w^*$. This is done by replacing each symbol in $w$ with its $\ell$-symbol codeword. And, once $M_2$'s emulation of $M_1$ discovers that $M_1$ has output a string $v^*$, then $M_2$ has

to convert it the encoded output string $v^*$ back to the string $v \in \Sigma^*$ for $M_2$ to output it. For example, suppose codewords have length $\ell = 2$. Then if $M_1$ takes input "01" and outputs "010", then $M_2$ first replaces the input with "0011", then emulates $M_1$ using codewords of length 2, and finally converts the encoded output "001100" to just "010".

These conversions from $w$ to $w^*$ and from $v^*$ to $v$ have to be done using the tape alphabet $\Gamma_2 = \{0, 1, \#\}$. Exercise VI.7 asks you to give algorithms for this under the assumption that the codewords have length $\ell = 2$ and that the codewords for "0" and"1" are "00" and "11". Similar methods can be used for longer codewords, still using the tape alphabet $\Gamma_2 = \{0, 1, \#\}$.

This construction allows any Turing machine with input alphabet $\{0, 1\}$ be converted into an equivalent Turing machine that uses the tape alphabet $\{0, 1, \#\}$. In particular, there is a universal Turing machine $U$ which uses the tape alphabet $\{0, 1, \#\}$. The only caveat is that if the universal Turing machine is to handle Turing machines that have input alphabet $\Sigma$ other than $\{0, 1\}$ then the universal machine has to accept encoded inputs and output encoded outputs.

**Using the tape alphabet $\{1, \#\}$** It is even possible to restrict Turing machines to use the tape alphabet $\{1, \#\}$. Turing machines with an alphabet of size 2, say $\Sigma = \{1, \#\}$, are typically envisioned as working on (non-negative) integers. An input or output string of the form $1^n$ is viewed as representing in the integer $n$.

We claim that any Turing machine with input alphabet $\Sigma = \{1\}$ can be simulated by a Turing machine with tape alphabet $\{1, \#\}$. This is proved almost exactly like the just-given construction reducing Turing machines to machines that use $\Gamma = \{0, 1, \#\}$. The only difference is now codewords use the two symbols # and 1 instead of 0 and 1. We formalize this as a theorem as it will be useful in Chapter VII when we discuss how to represent decidable predicates and computable functions in first-order theories of arithmetic.

Recall that Definitions VI.18-VI.20 defined the computability of functions and the decidability of relations on $\mathbb{N}$ using Turing machines that encoded integers as $\{0, 1\}$-strings in binary notations.

**Theorem VI.24.** *Let $\Sigma = \{1\}$ and $\Gamma = \{\#, 1\}$.*

(a) *If $R$ is a decidable $k$-ary predicate on $\mathbb{N}$, then there is a Turing machine with input alphabet $\Sigma$ and tape alphabet $\Gamma$ that decides $R$, using unary notation to encode its inputs.*

(b) *If $f$ is a computable $k$-ary function on $\mathbb{N}$, then there is a Turing machine with input alphabet $\Sigma$ and tape alphabet $\Gamma$ that computes $f$, using unary notation to encode its inputs and outputs.*

*Proof.* (Sketch) Turing machines with input alphabet $\{1\}$ represent integers $n$ in unary notation with strings $1^n$. Exercise VI.6 asks you to show Turing machines can convert between binary and unary notations. Thus for purposes of decid-

ability or computability, it does not matter whether integers are represented using binary notation or unary notation.

Using unary notation for integers, the "codeword" construction above shows that decidable predicates and computable functions can be decided or computed by a Turing machine with tape alphabet $\{1, \#\}$. □

This theorem will be helpful in Chapter VII, when we show that Turing machine computations are representable.

**Multitape Turing machines.** So far, we have discussed only Turing machines that have a single two-way infinite tape, with a single tape head. It is common to also consider multitape Turing machines that have $k$ many tapes for some $k > 1$. Each of the $k$ tapes has its own tape head and the tape heads can move independently. The transition function $\delta$ for a $k$ tape Turing machine takes as input the current state, and the $k$ symbols being read by the $k$ tape heads. The value of the transition function is the $k$ many symbols that overwrite the symbols on the $k$ tape, and $k$ many values from $\{\mathrm{R}, \mathrm{L}, \mathrm{N}\}$ for "move right", "move left" or "no movement", and the next state. Thus the transition function $\delta$ is a mapping

$$\delta \ : \ Q \times \Gamma^k \ \rightarrow \ \Gamma^k \times \{\mathrm{R}, \mathrm{L}, \mathrm{N}\}^k \times Q.$$

A tape head of a multitape Turing machine is allowed to remain stationary instead of moving left or right (using the code "N" for "no movement") so as to have complete flexibility in the independent movement of the tape heads. The first tape of a multitape Turing machine is the designated input/output tape. The input tape holds the inputs when the machine is started, and the rest of the tapes are initially blank. The output tape is usually the same as the input tape and holds the output when the Turing machine outputs a value.

A multitape Turing machine $M$ can be simulated by a single tape Turing machine $N$, albeit with a quadratic slowdown in running time. The usual approach for constructing $N$ from $M$ is something similar to the following. We let $\Gamma'$ be the set containing the symbols $a'$ for $a \in \Gamma$. In other words, $\Gamma'$ contains the "breadcrumbed" copies of symbols of $\Gamma$. Then the tape alphabet of $N$ is $(\Gamma \cup \Gamma')^k$, so each symbol is a $k$-tuple of symbols from $\Gamma$ possibly with breadcrumbs added. In this way the contents of the $k$-tapes $M$ can be written "in $k$ parallel tracks" on the tape of $N$; a tape cell of $N$ holds the symbols of the corresponding cells on the $k$ tapes of $M$. A symbol is marked with a breadcrumb to indicate that the corresponding tape head is currently reading that symbol. The intuition is that the tape of $N$ has $k$ many tracks, with each track holding the contents of one tape of $M$.

It is straightforward to show that a Turing machine $N$ of this type can simulate the action of $M$. The finite state control of $N$ can remember the relative positions of the $k$ tape heads on the $k$ tapes, relative to the tape head position of $N$. Thus, $N$ can scan back and forth between the different tape head positions and make the appropriate updates to the contents of the $k$ tracks on its tape. We leave the details to the reader to work out.

## VI.5 Malleability of Turing Machines

The malleability of Turing machines allows algorithms, implemented on Turing machines possibly, to parse and modify Turing machines in terms of their Gödel numbers. For instance, the construction from the previous section that converts a Turing machine $M_1$ with input alphabet $\Sigma = \{0, 1\}$ and arbitrary tape alphabet into an equivalent Turing machine $M_2$ with tape alphabet $\Gamma = \{0, 1, \#\}$ was entirely constructive, in the sense that there is a (straightforward) algorithm that maps $\ulcorner M_1 \urcorner$ to $\ulcorner M_2 \urcorner$. By the Church-Turing thesis, the mapping $\ulcorner M_1 \urcorner \mapsto \ulcorner M_2 \urcorner$ and be computed by a Turing machine. Similarly, a $k$-tape Turing machine can be algorithmically converted into an equivalent 1-tape Turing machine.

It is particularly important that the malleability assumptions used in the proof of Theorem V.52 about the undecidability of $\text{Halt}_0$ hold for Turing machines. Specifically, when working with Gödel numbers of Turing machines, the mapping sending $\ulcorner M \urcorner$ and $\ulcorner N \urcorner$ to $f'(f(\ulcorner N \urcorner), \ulcorner M \urcorner)$ is computable by a Turing machine. This construction was crucial to the construction of a self-referential algorithm $D_N$ in Section V.6.3.

Let's break this down and consider $f(\ulcorner M \urcorner)$ and $f'(\ulcorner M_1 \urcorner, \ulcorner M_2 \urcorner)$ separately. First consider $f(\ulcorner M \urcorner)$. When working with Turing machines, the function $f$ has the property that, for any $w \in \{0, 1\}$, $f(w)$ is equal to the Gödel number of a machine $M_w$ that ignores its input and outputs the string $w$. The machine $M_w$ uses input alphabet $\Sigma = \{0, 1\}$ and the alphabet $\Gamma = \{0, 1, \#\}$. It starts by using a fixed number of states (two states in fact) to overwrite its input with #'s. It then uses $|w|$ states to write the string $w$ on the tape. It then uses another constant set of states (one state can suffice if $w$ is written right-to-left) to position the tape head on the first symbol of $w$ and halt. This is a total of $|w| + c_f$ many states for some contant $c_f$. (In fact, $c_f$ can equal three.)

Second consider $f'(\ulcorner M_1 \urcorner, \ulcorner M_2 \urcorner)$. The idea is that $f'(\ulcorner M_1 \urcorner, \ulcorner M_2 \urcorner)$ is the Gödel number of a Turing machine that first runs the Turing machine $M_1$ and then uses the output of $M_1$ as the input to $M_2$ and runs $M_2$. There is a bit of a complication since, by convention, $M_2$ is expecting a tape to be completely blank except for its input. But this can be readily fixed. Either we can modify $M_1$ so that it erases everything on the tape except the output string before halting. Or, we can modify $M_2$ to write blanks into any "new" part of the tape before using it. For the former option, $M_1$ is modified use a new breadcrumbed symbol $\#'$ to mark the leftmost and rightmost tape cells visited. As more tape cells are visited, the modified $M_1$ detects this and moves the delimiting $\#'$ symbols as needed. $M_1$ is further modified so that, before halting, it scans left and right to find the $\#'$ symbols, overwriting all of the visited tape cells with #'s except for the tape cells containing the output. For the latter option, $M_2$ can be modified similarly to use a new breadcrumbed symbol $\#'$ to mark the leftmost and rightmost visited tape cells. When new tape cells need to be visited, the delimiting symbol $\#'$ is moved over and the new cells are initialized to hold #'s.

Detecting and maintaining the delimiting $\#'$ symbols requires extra states: it basically replaces each state of $M_1$ or $M_2$ with a small (constant) number of

new states. Then eliminating the use of the new symbol $\#'$, so as to use the tape alphabet $\Gamma = \{0, 1, \#\}$ requires another constant factor blowup in the number of states: this is based on the construction in the previous section. The end result is that the number of states in the Turing machine with Gödel number $f'(\ulcorner M_1 \urcorner, \ulcorner M_2 \urcorner)$ is bounded by $O(n_1 + n_2)$ where $M_1$ and $M_2$ have $n_1$ and $n_2$ states each.

In conclusion, Turing computable functions, Turing decidable relations, Turing enumerable relations, etc. satisfy the malleability conditions of Chapter V. Thus, all the results of Chapter V apply to Turing machines. This includes the undecidability of the halting problems $\text{Halt}_0$, $\text{Halt}_1$ and $\text{Halt}_{\text{Self}}$ for Turing machines, the Diagonal Lemma, and Rice's Theorem.

## Exercises

**Exercise VI.1.** Describe Turing machines by drawing a state diagram and specifying which states are halting, accepting, rejecting, or output states.

(a) Prove that the set $\{1^n 0 : n \geq 0\} = \{0, 10, 110, 1110, 11110, \ldots\}$ is Turing decidable by giving a Turing machine that decides it.

(b) Prove that the set $\{1^n 0 : n \geq 0\} = \{0, 10, 110, 1110, 11110, \ldots\}$ is Turing enumerable by giving a Turing machine that enumerates it.

**Exercise VI.2.** Let the input alphabet have $k$ symbols $\Sigma = \{s_1, \ldots s_k\}$. Describe a Turing machine $M$ that decides the set

$R = \{w \in \Sigma^* : w \text{ is non-empty and has its first symbol the same as the last symbol}\}$.

How many states does your machine $M$ have as a function of $k$?

**Exercise VI.3.** Let the input alphabet be $\Sigma = \{0, 1\}$. Give the state diagram for a Turing machine that computes the string concatenation function $\langle u, v \rangle \mapsto uv$. Describe the input and output conditions. Where is the tape head placed in the halting configuration? What are the initial configurations in the cases either $u$ or $v$ are the empty string?

**Exercise VI.4.** Give the explicit state diagram of a Turing machine $M_=$ that computes the binary equality relation. Namely, $M_=$ accepts the pair $\langle v, w \rangle$ if $v = w$, and rejects if $v \neq w$. [Hint: You might wish to use the breadcrumb technique, similar to what was done for copying a string in Example VI.15; this is especially useful if you want to carry out the comparison without destroying the values of $v$ and $w$. It is also possible to non-destructively compare $v$ and $w$ without using breadcrumbs.]

**Exercise VI.5.** Recall that $w^{\text{R}}$ is the reversal of $w$. Prove the following by giving an explicit description of a Turing machine, preferably in the form of a state diagram. (One way to do these is with the breadcrumb technique. There

are Turing machines with ten states that work, but we do not know if ten states is optimal.)

(a) Prove that the reversal function $w \mapsto w^{\mathrm{R}}$ is Turing computable.
(b) Prove that $\{w : w \in \Sigma^* \text{ and } w = w^{\mathrm{R}}\}$ is Turing decidable. This is the set of palindromes.

**Exercise VI.6.** Give high-level descriptions of how Turing machines with input and tape alphabets $\Sigma = \Gamma = \{0, 1, \#\}$ can convert between unary and binary representations for integers, so that the following functions are computable:

(a) The function $1^n \mapsto \mathit{str}(n)$,
(b) The function $w \mapsto 1^n$ where $w$ is a binary representation of $n$.

**Exercise VI.7.** Give the state diagrams for Turing machines that use only the tape symbols $\Gamma = \{0, 1, \#\}$ and compute the following functions:

(a) The symbol-doubling function $a_1a_2\cdots a_{n-1}a_n \mapsto a_1a_1a_2a_2\cdots a_{n-1}a_{n-1}a_na_n$; so that 0's are replaced with "00", and 1's with "11".
(b) The symbol-halving function $a_1a_2a_3a_4\cdots a_{n-1}a_n \mapsto a_2a_4a_6\cdots a_{2\lfloor n/2 \rfloor}$ that computes an inverse to the symbol-doubling function.

**Exercise VI.8.**

(a) Prove that there is a Turing machine $M$ such that: (i) If $M$ is started with the input tape is completely blank, then $M$ does not halt, and (ii) If $M$ is started with a non-blank symbol anywhere on the tape, then $M$ halts.
(b) Prove that there is no Turing machine $N$ such that: (i) If $N$ is started with the input tape completely blank, then $N$ eventually accepts, and (ii) If $N$ is started with a non-blank symbol anywhere on the tape, then $N$ rejects.

**Exercise VI.9.** For this exercise, restrict attention to Turing machines that use the alphabets $\Sigma = \Pi = \{\#, 1\}$ and partial compute unary functions. Prove that it is undecidable whether a given Turing machine has the least possible number of states. That is, let $X$ be the binary relation

$$X = \{\langle \ulcorner M \urcorner, n\rangle : M \text{ is a Turing machine and there is no Turing machine } M' \text{ that partial computes the same partial function as } M \text{ and has at most } n \text{ states.}\}$$

Prove $X$ is undecidable. As a consequence, there is no algorithm to minimize the number of states in a Turing machine.

**Exercise VI.10.** (The Busy Beaver function for running time.) For this exercise and the next one, restrict attention to Turing Machines over the alphabets $\Sigma = \Gamma = \{1, \#\}$. With this restriction, there are only finitely many Turing machines with $n$ states (up to renaming of states). Define the running-time version of the Busy Beaver function $\mathrm{BB}_{\mathrm{steps}} : \mathbb{N} \to \mathbb{N}$ by

$$\mathrm{BB}_{\mathrm{steps}}(n) = \max\{m \geq 0 : \text{there is a Turing Machine } M \text{ with } n \text{ states such that } M(\epsilon) \text{ halts after exactly } m \text{ steps.}\}$$

(a) Prove that $\mathrm{BB}_{\mathrm{steps}}(n)$ is not computable by proving that otherwise $\mathrm{Halt}_0^{\mathrm{TM}}$ would be decidable. (Here $\mathrm{Halt}_0^{\mathrm{TM}}$ means the version of $\mathrm{Halt}_0$ where Gödel

numbers of algorithms are given in the form of Turing machines. All the theorems proved in Chapter V for $\text{Halt}_0$ apply also to $\text{Halt}_0^{\text{TM}}$.)

(b) Suppose $f : \mathbb{N} \to \mathbb{N}$ is computable. Prove that $\text{BB}_{\text{steps}}(n)$ eventually dominates $f$, namely there is an $N$ such that $f(n) < \text{BB}_{\text{steps}}(n)$ for all $n > N$.

**Exercise VI.11.** (Busy Beaver function for output value.) Use the same conventions as the previous exercise. The Busy Beaver function $\text{BB} : \mathbb{N} \to \mathbb{N}$ is defined by

$$\text{BB}(n) = \max\{m \geq 0 : \text{There is a Turing Machine } M \text{ with } n \text{ states such that } M(\epsilon) \text{ halts with } m \text{ many 1's written on the tape. }\}$$

Suppose $f : \mathbb{N} \to \mathbb{N}$ is computable. Prove that $\text{BB}(n)$ eventually dominates $f$. Conclude that BB is not computable.

# Chapter VII

# Arithmetic and Incompleteness

This chapter discusses first-order theories of arithmetic and the First and Second Gödel Incompleteness Theorems. The key result is that the theory $\mathrm{Th}\,\mathcal{N}$ of the integers is undecidable. It follows that there is no (decidable) axiomatization for the set of true statements about the integers, even when restricting to the language $L_{\mathrm{PA}} = \{0, S, +, \cdot\}$. This fact is the essence of Gödel's First Incompleteness Theorem.

## VII.1 Intensional and Extensional Approaches

There are two traditional ways to prove the First Incompleteness Theorem. The first way, sometimes called the "intensional" approach, works with the first-order Peano Arithmetic theory (PA). Peano Arithmetic is formalized over the language $L_{\mathrm{PA}}$ with induction axioms as its primary axioms. Although Peano Arithmetic can talk only about integers, it can in fact formalize the metamathematical syntax of first-order logic. It does this by encoding metamathematical concepts such as "first-order formula" and "FO-proof" with Gödel numbers, namely with integers. Via Gödel numbers, Peano Arithmetic can formulate and reason about the syntactic properties of first-order formulas, sentences, proofs, and theorems; in particular, it can formulate the syntactic properties of proofs and theorems of PA itself. This includes all the syntactic concepts developed in Chapters III and IV, including the concepts such as provability and consistency. This allows Peano Arithmetic to "reason" about its own consistency, and to formulate a sentence $\mathrm{Con}_{\mathrm{PA}}$ expressing that Peano Arithmetic itself is consistent.

The First Incompleteness Theorem for Peano Arithmetic can be proved by using a diagonal construction similar to the Diagonal Theorem V.57 but now using a sentence $A$ that expresses "The sentence $A$ does not have a PA-proof", or informally, the sentence $A$ asserts "I am not PA-provable". The First In-

completeness Theorem is proved by showing that (under the assumption that PA is consistent) this sentence $A$ is true in $\mathcal{N}$, but is not a theorem of Peano Arithmetic. The Gödel Second Incompleteness Theorem takes this further by showing that Peano Arithmetic (unless inconsistent) cannot prove the sentence $\mathrm{Con}_{\mathrm{PA}}$ expressing the consistency of Peano Arithmetic. Thus $\mathrm{Con}_{\mathrm{PA}}$ is an explicit and comprehensible example of a true sentence that cannot be proved by Peano Arithmetic.

Peano Arithmetic is limited to dealing with finite objects and cannot reason directly with infinite sets. It can however reason indirectly about definable infinite sets. For instance, Peano Arithmetic can formalize the notions of Turing machines and computations of Turing machines. It can thus indirectly reason about Turing decidable sets and Turing enumerable sets in the sense that it can reason about the members of the sets (rather than about the sets themselves).[1] This permits Peano Arithmetic to formalize a version of the Completeness Theorem. However, Peano Arithmetic cannot formalize the definition of truth in a general way. Hence, it cannot even state the general form of the Soundness Theorem.

The second traditional way to prove the First Incompleteness Theorem is the "extensional" approach. For this, we use a very weak subtheory Q of PA (called "Robinson's theory Q") of , for even stronger results, we use an even weaker subtheory called R. The language of both Q and R is $L_{\mathrm{PA}}$. The axioms of Q state simple defining properties of zero, successor, addition, and multiplication, but Q has no induction axioms. The theory Q is quite weak; unlike PA, it cannot even prove elementary things such as that addition is commutative, $\forall x\,\forall y\,(x + y = y + x)$. In the *intensional* approach, PA does prove this. In the *extensional* approach, we have only that Q can prove particular instances of $x + y = y + x$; e.g., Q can prove $\underline{3} + \underline{4} = \underline{4} + \underline{3}$ where "$\underline{3}$" and "$\underline{4}$" denote the terms $S(S(S(0)))$ and $S(S(S(S(0))))$.

We shall see later that Q can "represent" every Turing decidable set $R$ in an extensional fashion. What this means is that there is a formula $G_R(x)$ defining membership in $R$ such that for any particular integer $n$, Q can prove either $G_R(\underline{n})$ or $\neg G_R(\underline{n})$ depending on whether $n$ is in $R$ or not. Similarly, every Turing computable function $f$ can be represented by a formula $G_f$ in Q. This means that, for any $n \in \mathbb{N}$, if $f(n) = m$, then Q can prove the sentence $\forall x\,(G_f(\underline{n}, x) \leftrightarrow x = \underline{m})$. (The precise definitions for "represent" are given below.)

Perhaps surprisingly, the converse holds too and any set which is representable in Q is Turing decidable. This has a very general proof that has little to do with Q itself. Namely, since Q is finitely axiomatized, its logical consequences $\mathrm{Cn}\,Q$ are Turing enumerable. Thus, to decide whether $n \in R$, one can enumerate the consequences of Q until obtaining either $G_R(\underline{n})$ or $\neg G_R(\underline{n})$. One of these must appear, and it tells us whether $n$ is in $R$ or not. A similar proof shows that every representable function of Q is Turing computable.

[1] In fact, Peano Arithmetic can define many sets beyond decidable, c.e. and co-c.e. sets.

From these results, it will follow that a set is representable in Q if and only it is Turing decidable. In light of the Church-Turing Thesis, we have that any $R$ is decidable if and only if it is representable in Q.

The First Incompleteness Theorem will be proved from this fact. Specifically, a diagonal construction rather similar to the Diagonal Theorem V.57 for decidable sets can be used to form an $L_{\text{PA}}$-sentence $D$ that asserts it is not a consequence of Q. From the consistency of Q, it will follow that neither $D$ nor $\neg D$ is a consequence of Q.

Nearly all the claims just made about Q are also true for the weaker theory R. In particular, R also represents precisely the relations which are decidable and the functions which are computable. The First Incompleteness Theorem also applies to R.

In fact, the same arguments apply to any consistent, axiomatizable theory $T$ containing R. In particular, for any such theory $T$, the sets representable in $T$ are precisely the Turing decidable sets. The diagonal method thus implies that there is no complete, consistent, axiomatizable theory $T$ containing R. From this, it follows that $\text{Th}\mathcal{N}$ is not axiomatizable and hence not (Turing) decidable.

This chapter will carry out the extensional approach in full detail. There are several reasons for using the extensional approach. The first, and most important reason, is that it is technically easier than the intensional approach. Second, it provides more justification for the Church-Turing Thesis, since it shows that the decidable sets can be robustly defined in terms of any consistent axiomatizable theory $T \supseteq R$. In other words, stronger theories do not give more decidable sets. Third, by working with R and Q, we delineate more-or-less exactly where undecidability kicks in. After carrying out the extensional approach, Section VII.9 will sketch a high-level proof of how the intensional approach proceeds. This high-level proof will be based on Peano Arithmetic PA.[2]
.

## VII.2 Four Theories of Arithmetic

We now define three axiomatizations for arithmetic. There are two quite weak theories, called Q and R. We also define the quite strong theory PA of Peano Arithmetic. These are all subtheories of $\text{Th}\mathcal{N}$, so they give partial axiomatizations of truth over the integers.[3] This gives four theories R, Q, PA and $\text{Th}\mathcal{N}$, in order of increasing strength.

All four theories use the language $L_{\text{PA}} = \{0, S, +, \cdot\}$. As usual, we write $+$ and $\cdot$ in infix notation. We often omit parentheses when applying the successor

[2]The best way to carry out the intensional approach is to base it on Q, not PA. This is done by interpreting the bounded arithmetic theory PV or $\text{S}^1_2$ into Q and carrying out the intensional proof of the Incompleteness Theorem in PV or $\text{S}^1_2$. For this, see Cook [1975] and Buss [1986] for intensional formalizations of metamathematics in PV and $\text{S}^1_2$, and see Pudlák [1985] and Nelson [1986] for interpretations in Q. These developments are beyond the scope of the present textbook, however.

[3]By "integer", we continue to always mean "nonnegative integer".

function $S$. For example $S0$ and $SSx$ are shorthand notations for $S(0)$ and $S(S(x))$; the intuition is that they denote 1 and $x+2$.

**Definition VII.1.** The theory Q is the theory with the following seven axioms

$Q_1$: $\forall x\,\forall y\,(Sx = Sy \to x = y)$

$Q_2$: $\forall x\,(Sx \neq 0)$

$Q_3$: $\forall x\,(x \neq 0 \to \exists y\,(Sy = x))$

$Q_4$: $\forall x\,(x + 0 = x)$

$Q_5$: $\forall x\,\forall y\,(x + Sy = S(x + y))$

$Q_6$: $\forall x\,(x \cdot 0 = 0)$

$Q_7$: $\forall x\,\forall y\,(x \cdot Sy = xy + x)$

The first three axioms give properties of the successor function $S$. Since $S$ is a function symbol, every object has a unique successor. Axioms $Q_1$ and $Q_3$ state that every nonzero element as a unique predecessor. Axiom $Q_2$ states that 0 has no predecessor; this reflects the fact that we are axiomatizing the nonnegative integers.

Axioms $Q_4$ and $Q_5$ give a kind of characterization of addition $x+y$ based on induction on $y$. Axioms $Q_6$ and $Q_7$ give a similar characterization of multiplication. For instance, informally speaking, $Q_7$ states that $x \cdot (y+1) = xy + x$.

As we shall see, Q is strong enough to represent all Turing decidable sets and functions; nonetheless Q is very weak. The much stronger theory of Peano Arithmetic is obtained by adding induction axioms for every formula $A(x)$:

**Definition VII.2.** Let $A = A(x)$ be a formula. The induction axiom $Ind_A$ is the sentence equal to the universal closure of the statement

$$A(0) \wedge \forall x\,(A(x) \to A(Sx)) \to \forall x\,A(x).$$

**Example VII.3.** Let $A(x)$ be the formula $0 + x = x$. The induction axiom for $A(x)$ is

$Ind_A$**:** $\quad 0 + 0 = 0 \wedge \forall x\,(0 + x = x \to 0 + Sx = Sx) \to \forall x\,(0 + x = x)$.

For an example where there are free variables other than $x$, let $B(x)$ be $x{+}y = y{+}x$. Then the induction axiom for $B(x)$ is the sentence

$Ind_B$**:** $\quad \forall y\,[0{+}y = y{+}0 \wedge \forall x\,(x{+}y = y{+}x \to Sx{+}y = y{+}Sx) \to \forall x\,(x{+}y = y{+}x)]$.

**Definition VII.4.** The theory PA of Peano Arithmetic is axiomatized with the seven axioms of Q plus the induction axioms $Ind_A$ for every $L_{\mathrm{PA}}$-formula $A$.

**Example VII.5.** We prove that the induction axioms enable PA to prove $\forall x\,(0 + x = x)$. Let $A$ be $0 + x = x$, so that that $Ind_A$ is the formula displayed above in Example VII.3. It is obvious that $\mathrm{PA} \vdash 0 = 0$, since that is an instance of an equality axiom. Second, we claim that PA proves $0 + x = x \to 0 + Sx = Sx$. By the refined Deduction Theorem IV.14(b) it will suffice to prove that $\mathrm{PA}, 0{+}x{=}x \vdash 0 + Sx = Sx$ without using generalization on the variable $x$. This is easily proved by reasoning as follows:

$$\begin{array}{rcll} 0 + Sx & = & S(0 + x) & \text{By axiom } Q_5 \\ & = & Sx & \text{By the induction hypothesis that } 0 + x = x \ . \end{array}$$

We have shown that PA proves the hypotheses of $Ind_A$. Therefore, PA proves its conclusion, $\forall x\,(0 + x = x)$.

The theory Q is too weak even to prove that $0 + x = x$. Exercise VII.2 asks for a proof of this. Therefore Q is a proper subtheory of PA.

The next definition is needed before we can define the third theory R.

**Definition VII.6.** Let $n \geq 0$ be an integer. The notation $S^{(n)}t$ denotes the term $S(S(\cdots S(S(t))\cdots))$ where there are $n$ many applications of $S$. For $n = 0$, $S^{(n)}t$ is just the term $t$.

The *numeral* $\underline{n}$ is the term $S^{(n)}0$.

Thus, $S^{(3)}x$ is the term $S(S(S(x)))$, and $S^{(4)}0$ is the term $S(S(S(S(4))))$. These are often written as $SSSx$ and $SSSS0$ for short. The numerals $\underline{0}$, $\underline{1}$, $\underline{2}$ and $\underline{3}$ are equal to the terms 0, $S0$, $SS0$ and $SSS0$, etc. Of course, in the standard model, these terms represent the integers $0, 1, 2, 3, \ldots$. It is important to note that the notations $S^{(n)}t$ and $\underline{n}$ are used *only* if $n$ denotes a particular integer. That is to say, the terms $S^{(n)}t$ and $\underline{n}$ do not involve $n$ as a variable; instead, $n$ is hardcoded in these terms via $n$ many appearances of the unary function symbol $S$.

**Example VII.7.** The atomic formula $\underline{3} + \underline{2}$ is the same as $SSS0 + SS0$. The theory Q can prove $\underline{3} + \underline{2} = \underline{5}$ by reasoning as follows:

$$\begin{array}{rcll} SSS0 + SS0 & = & S(SSS0 + S0) & \text{by Axiom } Q_5 \\ & = & SS(SSS0 + 0) & \text{by Axiom } Q_5 \text{ again} \\ & = & SSSSS0 & \text{by Axiom } Q_4 \end{array}$$

Similar reasoning shows that Q can prove $\underline{m} + \underline{n} = \underline{m + n}$ for all $m, n \in \mathbb{N}$. For this, see Theorem VII.33.

**Notation VII.8.** For the rest of this chapter, the notation $s \leq t$ denotes the formula $\exists x\,(x + s = t)$ where $x$ is a variable that does not appear in $s$ or $t$. The notation $s < t$ denotes the formula $s \leq t \wedge s \neq t$, namely the formula $\exists x\,(x + s = t) \wedge s \neq t$, where again $x$ does not appear in $s$ or $t$.

We can view $\leq$ as being added to the language PA via an extension by definitions in the sense of Definition III.113. From Theorem III.115, it does not change the power of a theory to add $\leq$ as a defined symbol.

The point of including $\leq$ as a defined symbol is that it makes the axioms of R much more natural to state. In this regard, it is important $\exists x\,(x + s = t)$ is used instead of $\exists x\,(s + x = t)$ as the definition of $s \leq t$: this makes a difference because Q cannot prove $x + s = s + x$.

Now that we have defined the weak theory Q, we also define the even weaker theory R.

**Definition VII.9.** The theory R uses the language $\{0, S, +, \cdot\}$ and has the following infinite set of axioms.

$\mathrm{R}_{\neq}$: For each $n \neq m \in \mathbb{N}$, the formula $\underline{n} \neq \underline{m}$ is an axiom of R.

$\mathrm{R}_{+}$: For each $n, m \in \mathbb{N}$, the formula $\underline{n} + \underline{m} = \underline{n+m}$ is an axiom of R.

$\mathrm{R}_{\bullet}$: For each $n, m \in \mathbb{N}$, the formula $\underline{n} \cdot \underline{m} = \underline{n \cdot m}$ is an axiom of R.

$\mathrm{R}_{\leq}$: For each $n \in \mathbb{N}$, the formula $\forall x\,(x \leq \underline{n} \vee \underline{n} < x)$ is an axiom of R.

$\mathrm{R}'_{\leq}$: For each $n \in \mathbb{N}$, the formula

$$\forall x\,[\,x \leq \underline{n} \rightarrow x = \underline{0} \vee x = \underline{1} \vee \cdots \vee x = \underline{n-1} \vee x = \underline{n}\,]$$

is an axiom of R.

Recall that $r \leq s$ is an abbreviation for $\exists x(x + r = s)$ and that $r < s$ is an abbreviation for $r \leq s \wedge r \neq s$.

**Example VII.10.** With $n = 3$ and $m = 2$, the following are axioms of R:

$$SSS0 \neq SS0 \qquad SS0 + SSS0 = SSSSS0$$

$$SS0 \cdot SSS0 = SSSSSS0 \qquad \forall x\,(x \leq SSS0 \vee SSS0 \leq x)$$

$$\forall x\,(x \leq SSS0 \rightarrow x = 0 \vee x = S0 \vee x = SS0 \vee x = SSS0).$$

The theory R is really very weak. Indeed, it does not prove any of the axioms of Q. As a small example of something that can be proved in R, we have

**Theorem VII.11.**

(a) *Let* $m \leq n \in \mathbb{N}$. *Then* $\mathrm{R} \vdash \underline{m} \leq \underline{n}$.

(b) *Let* $n \in \mathbb{N}$. $\mathrm{R} \vdash \forall x\,(x \leq \underline{n} \leftrightarrow x = \underline{0} \vee x = \underline{1} \vee \cdots \vee x = \underline{n-1} \vee x = \underline{n})$.

(c) *Let* $n \in \mathbb{N}$. $\mathrm{R} \vdash \forall x\,(x < \underline{n} \leftrightarrow x = \underline{0} \vee x = \underline{1} \vee \cdots \vee x = \underline{n-1})$.

(d) *Let* $n \in \mathbb{N}$. $\mathrm{R} \vdash \forall x\,(x < \underline{n} \vee x = \underline{n} \vee \underline{n} < x)$.

*Proof.* To prove part (a), let $m \leq n$ and set $p = n - m \geq 0$. By the axiom $\mathrm{R}_{+}$, the theory R proves that $\underline{p} + \underline{m} = \underline{n}$. Therefore, by the definition of $\leq$ (see Notation VII.8), R proves $\underline{m} \leq \underline{n}$.

The forward implication of (b) is just Axiom $\mathrm{R}'_{\leq}$. The reverse implication is immediate from part (a).

The forward implication of (c) follows from part (b) and the definition of $<$ in Notation VII.8. The reverse implication of (c) follows from the definition of $<$, part (b), and from the the fact that Axiom $\mathrm{R}_{\neq}$ implies that $\mathrm{R} \vdash \underline{m} \neq \underline{n}$ for all $m < n$.

Part (d) follows directly from $\mathrm{R}_{\leq}$ and parts (b) and (c). □

Section VII.5 will show R is a subtheory of Q. Sections VII.6-VII.8 will show many more things that are provable in R, most notably that all Turing decidable relations and Turing computable functions are representable in R.

**Axioms of** Q

$Q_1$: $\forall x\, \forall y\, (Sx = Sy \to x = y)$
$Q_2$: $\forall x\, (Sx \neq 0)$
$Q_3$: $\forall x\, (x \neq 0 \to \exists y\, (Sy = x))$
$Q_4$: $\forall x\, (x + 0 = x)$
$Q_5$: $\forall x\, \forall y\, (x + Sy = S(x + y))$
$Q_6$: $\forall x\, (x \cdot 0 = 0)$
$Q_7$: $\forall x\, \forall y\, (x \cdot Sy = xy + x)$

**Axioms of** R

$R_{\neq}$: For each $n \neq m \in \mathbb{N}$, the formula $\underline{n} \neq \underline{m}$ is an axiom of R.
$R_{+}$: For each $n, m \in \mathbb{N}$, the formula $\underline{n} + \underline{m} = \underline{n+m}$ is an axiom of R.
$R_{\bullet}$: For each $n, m \in \mathbb{N}$, the formula $\underline{n} \cdot \underline{m} = \underline{n \cdot m}$ is an axiom of R.
$R_{\leq}$: For each $n \in \mathbb{N}$, the formula $\forall x\, (x \leq \underline{n} \vee \underline{n} < x)$ is an axiom of R.
$R'_{\leq}$: For each $n \in \mathbb{N}$, the formula

$$\forall x\, [\, x \leq \underline{n} \to x = \underline{0} \vee x = \underline{1} \vee \cdots \vee x = \underline{n-1} \vee x = \underline{n}\,]$$

is an axiom of R.

$s \leq t$ is an abbreviation for $\exists x\, (x + s = t)$, where $x$ is a variable not appearing in $s$ or $t$.
$s < t$ is an abbreviation for $s \leq t \wedge s \neq t$.

## VII.3 Representability

To state results in full generality, we let $T$ be a consistent theory in the language $L_{\mathrm{PA}} = \{0, S, +, \cdot\}$. Generally, $T$ will contain Q, or at least R, as a subtheory. However, it is not required that $T$ is a subtheory of $\mathrm{Th}\mathcal{N}$. In other words, $T$ must be consistent but might contain sentences that are not true in $\mathcal{N}$.

Informally, a relation $R$ or function $f$ is "representable" in $T$ if every particular numeric instance of $R$ or $f$ is provable in $T$. (For this reason, the terminology "numeral-wise representable" is sometimes used instead.) This is formally defined as follows.

**Definition VII.12.** A $k$-ary relation $R$ on $\mathbb{N}$ is *representable* in $T$ if there is a formula $A_R(x_1, x_2, \ldots, x_k)$ such that for all $n_1, \ldots, n_k$ in $\mathbb{N}$,

(a) If $R(n_1, n_2, \ldots, n_k)$ holds, then $T \vdash A_R(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k})$, and
(b) If $R(n_1, n_2, \ldots, n_k)$ is false, then $T \vdash \neg A_R(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k})$.

In this case, we say that $R$ is *represented* by $A_R$ in $T$.

Since $T$ is consistent, it follows immediately from these conditions that $T$ proves $A_R(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k})$ iff $R(n_1, n_2, \ldots, n_k)$ is true. Similarly, $T$ proves $\neg A_R(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k})$ iff $R(n_1, n_2, \ldots, n_k)$ is false.

**Example VII.13.** As two simple examples, we show that the binary relations $=$ and $\leq$ are representable in the theory R. The formula $A_=(x, y)$ that represents $=$ is the formula $x = y$. To prove that $A_=(x, y)$ represents the binary equality relation $=$, we must show two things: (a) for all $n \in \mathbb{N}$, $R \vdash \underline{n} = \underline{n}$, and (b) for all $n \neq m$, $R \vdash \neg \underline{n} = \underline{m}$. The first item, (a), is just an instance of an equality axiom. The second item, (b), is just an $R_{\neq}$ axiom.

Now let $A_\leq(x, y)$ be the formula $x \leq y$; in other words, $\exists z(z + x = y)$. To show that the formula $A_\leq(x, y)$ represents $\leq$, we must show two things: (a) for all $m \leq n$, R proves $\underline{m} \leq \underline{n}$, and (b) for all $m > n$, R proves $\neg \underline{m} \leq \underline{n}$. Item (a) is the same as part (a) of Theorem VII.11. To prove item (b) let $m > n$. For each $m' \leq n$, axiom $R_{\neq}$ gives that R proves $\underline{m} \neq \underline{m'}$. Therefore part (b) of Theorem VII.11 implies that R proves $\neg \underline{m} \leq \underline{n}$.

**Definition VII.14.** A $k$-ary function $f$ on $\mathbb{N}$ is *representable* in $T$ if there is a formula $A_f(x_1, x_2, \ldots, x_k, y)$ such that for all $n_1, \ldots, n_k$ in $\mathbb{N}$, and for $m = f(n_1, n_2, \ldots, n_k)$, the theory $T$ proves

$$\forall y \, [\, A_f(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}, y) \leftrightarrow y = \underline{m} \,]. \tag{VII.1}$$

In this case, we say we say that $f$ is *represented* by $A_f$ in $T$.

The next theorem gives an equivalent formulation of representability for theories $T \supseteq$ R.

**Theorem VII.15.** *Suppose the theory $T$ extends* R. *A $k$-ary function $f$ is represented in $T$ by $A_f$ if and only if the following three conditions hold for all $n_1, \ldots, n_k$ in $\mathbb{N}$ and $m = f(n_1, n_2, \ldots, n_k)$.*

(a) *$T$ proves $A_f(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}, \underline{m})$.*

(b) *$T$ proves $\neg A_f(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}, \underline{m'})$ for all $m' \neq m$.*

(c) *$T$ proves $\forall x \, \forall y \, (A_f(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}, x) \wedge A_f(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}, y) \rightarrow x = y)$.*

*Proof.* The formulas of (a) and (c) taken together are logically equivalent to the formula (VII.1. Thus $A_f$ represents $f$ if and only if conditions (a) and (c) hold. To complete the proof of the theorem, we must show that, for $T \supseteq$ R, if conditions (a) and (c) hold, then condition (b) holds.

Suppose (a) and (c) hold and $m' \neq m$. By (c), $T$ proves

$$\forall x \, \forall y \, (A_f(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}, \underline{m}) \wedge A_f(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}, \underline{m'}) \rightarrow \underline{m} = \underline{m'}).$$

Therefore, by (a) and by the fact that $T \vdash \underline{m} \neq \underline{m'}$ from Axiom $R_{\neq}$, we get that $T$ proves $\neg A_f(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}, \underline{m'})$. That establishes (b). □

**Example VII.16.** As three simple examples, the successor, addition, and multiplication functions are representable in R. The formula $A_S(x_1, y)$ that represents the successor function $n \mapsto n+1$ is just the formula $S(x_1) = y$. To verify this, we must show that for every $n \in \mathbb{N}$, the theory R proves $S(\underline{n}) = \underline{n+1}$ and $\forall y\,(S(\underline{n}) = y \to y = \underline{n+1})$. These both follow immediately by equality axioms (without needing any axioms of R) since $S(\underline{n})$ and $\underline{n+1}$ are the same term, namely the term formed from $n+1$ applications of the function symbol $S$ to 0.

The formula $A_+(x_1, x_2, y)$ that represents the addition function $\langle n_1, n_2\rangle \mapsto n_1{+}n_2$ is just the formula $x_1 + x_2 = y$. To verify this, we must show that, for all $n_1, n_2 \in \mathbb{N}$, R proves $\underline{n_1} + \underline{n_2} = \underline{n_1 + n_2}$ and $\forall y\,(\underline{n_1} + \underline{n_2} = y \to y = \underline{n_1 + n_2})$. The first formula is an $\mathrm{R}_+$-axiom. The second formula follows from the first formula via an equality axiom.

The formula $A_\cdot(x_1, x_2, y)$ that represents the multiplication function $\langle n_1, n_2\rangle \mapsto n_1{\cdot}n_2$ is just the formula $x_1 \cdot x_2 = y$. This is verified by the same argument as was used for the addition function.

**Theorem VII.17.** *Suppose $T$ is a consistent theory in the language $L_{\mathrm{PA}}$.*

(a) *Suppose $R$ is a $k$-ary relation represented by $A_R$ in $T$. Then, for all $n_1, \ldots, n_k \in \mathbb{N}$, $T$ proves $A_R(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k})$ if and only if $R(n_1, n_2, \ldots, n_k)$ is true. Similarly, $T$ proves $\neg A_R(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k})$ if and only if $R(n_1, n_2, \ldots, n_k)$ is false.*

(b) *Suppose $f$ is a $k$-ary function represented by $A_f$ in $T$. Then for all $n_1, \ldots, n_k$ and $m = f(n_1, \ldots, n_k)$, $T$ proves $A_f(\underline{n_1}, \ldots, \underline{n_k}, \underline{m})$ and for all $m' \neq m$, $T$ proves $\neg A_f(\underline{n_1}, \ldots, \underline{n_k}, \underline{m'})$.*

*Proof.* This is immediate from the definitions of "representability". □

**Theorem VII.18.** *If the theory $T$ extends the theory* R*, then every relation representable in* R *is representable in $T$ and every function representable in* R *is representable in $T$.*

*Proof.* This is also immediate from the definitions. □

We shall prove in the sequel that the representable relations of the theory R are precisely the decidable relations and that the representable functions of R are precisely the computable functions. It will follow immediately that the same holds for any axiomatizable theory $T \supseteq \mathrm{R}$. For the moment, we can prove one direction of the inclusions:

**Theorem VII.19.** *Suppose $T$ is a consistent, axiomatizable theory and $T \supseteq R$.*

(a) *Suppose $S$ is a representable relation of $T$. Then $S$ is decidable.*

(b) *Suppose $f$ is a representable function of $T$. Then $f$ is computable.*

*In particular, since* R *is axiomatizable and consistent,* (a) *and* (b) *hold for* R *itself.*

*Proof.* By Theorem V.41, the theory $T$ is computably enumerable. Let $A_S$ represent $S$ in $T$. To decide whether $S(n_1, \ldots, n_k)$ holds, form the sentence

$A_S(\underline{n}_1, \ldots, \underline{n}_k)$ and enumerate the members of the theory $T$ until one of the sentences $A_S(\underline{n}_1, \ldots, \underline{n}_k)$ or $\neg A_S(\underline{n}_1, \ldots, \underline{n}_k)$ is enumerated. Since $A_S$ represents $S$, one of these must be enumerated, and since $T$ is consistent, it indicates whether or not $S(n_1, \ldots, n_k)$ holds.

For (b), let $A_f$ represent $f$ in $T$. To compute the value of $f(n_1, \ldots, n_l)$, enumerate the members of $T$ until producing a formula of the form $A_f(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}, \underline{m})$. Since $A_f$ represents $f$, such a formula must appear and then since $T$ is consistent, $f(n_1, \ldots, n_k) = m$. □

Note that the above theorems hold even if $T$ is not a subtheory of $\mathrm{Th}\mathcal{N}$. In other words, the theorems hold even for theories $T$ that might contain sentences that are false for the integers.

# VII.4 The First Incompleteness Theorem

We now give the statement and proof of Gödel's First Incompleteness Theorem. We will prove several forms of the First Incompleteness Theorem for theories $T$ that extend R. This includes theories $T$ such as Robinson's theory Q and Peano arithmetic PA, since we shall later show (in Theorem VII.33) that $\mathrm{Q} \vDash \mathrm{R}$.

The proof of the First Incompleteness theorem will depend on the following theorem, which will be proved later in Section VII.8:

**Theorem VII.20.** *Every Turing decidable relation on* $\mathbb{N}$ *is representable in the theory* R. *Every Turing computable function on* $\mathbb{N}$ *is representable in* R.

By Theorem VII.18, the same holds for every axiomatizable theory $T$ extending R. In particular, it holds for the theory Q. Accordingly, the rest of this section will work with an arbitrary axiomatizable, consistent theory $T \supseteq R$.

We will first discuss how to encode an $L_{\mathrm{PA}}$-term $t$ or an $L_{\mathrm{PA}}$-formula $A$ with an integer $\ulcorner t \urcorner$ or $\ulcorner A \urcorner$, which is called the "Gödel number" of $t$ or $A$. We will also discuss the representability of the substitution function $Sub(A, x, t)$. We then state and prove the Incompleteness Theorem. In fact, we will give three proofs. The first proof works directly off the undecidability of the halting problem; it proves the First Incompleteness Theorem for true theories and more generally for "$\omega$-consistent" theories. The second and third proofs establish a stronger form of the First Incompleteness Theory. These two proofs are variations of each other, they both use Cantor-style diagonalization. The third proof gives the full construction of "self-referential" formulas that can be obtained via a new version of the Diagonal Theorem that applies to first-order formulas.

## VII.4.1 Gödel numbering, numerals, and substitution.

Theories of arithmetic such as R, Q and PA deal only with integers; however, it is possible for them to deal indirectly with syntactic metamathematical objects such as terms, formulas, and proofs by using Gödel numbers to represent metamathematical objects. Section V.5 already discussed representing metamathematical objects with strings of symbols over a finite alphabet. We now

want to represent metamathematical object with integers: this will be done similarly to Section V.5, but differs by using strings over the alphabet $\{0,1\}$ which are interpreted as binary representations of integers.

The details of how Gödel numbers represent terms, formulas, and proofs are not terribly important, as long as it is sufficiently constructive so that algorithms (Turing machines) can recognize, parse, and manipulate them. For concreteness, we propose one possible way to define Gödel numbers for the first-order language $L_{\mathrm{PA}}$. Terms, formulas, and FO-proofs in the language $L_{\mathrm{PA}}$ can be viewed straightforwardly as strings of characters over the 14 symbol alphabet

$$\Delta \;=\; \{\,\neg,\, \to,\, \forall,\, (,\, ),\, x, \emptyset,\, 1,\, 0,\, S,\, +,\, \cdot,\, =,\, \text{comma}\}$$

There are two different zeroes in the list: the symbol $\emptyset$ is used when writing indices of variables in binary notation, whereas 0 is used as the non-logical symbol whose standard interpretation is the integer 0.[4] Variables $x_i$ are represented by a string $x$ followed by the binary representation of $i$; e.g., the terms $x_0$, $x_3$ and $x_5$ are represented by the strings $x\emptyset$, $x11$ and $x1\emptyset1$. This convention for coding variables and their subscripts allows terms, formulas, sentences, and proofs over the language of PA to be straightforwardly written as strings over $\Delta^*$.

To further convert strings from $\Delta^*$ to integers, we use a fixed length 4-bit binary encoding of symbols in $\Delta$ to encode a string in $\Delta^*$ as a base 2 integer. There are 14 symbols in $\Delta$. Each symbol of $\Delta$ can be represented by a distinct 4-bit fixed length code word; e.g. "$\neg$" is encoded by "0001", and "$\to$" by "0010", "$\forall$" by "0011", etc., up through "1110" encoding a comma. (The commas are used to separate formulas in Gödel numbers of proofs.) For example, the formula $x_2 = 0$ would be encoded as in string $\Delta^*$ as "$x1\emptyset$=0", and then further encoded by the binary string "0110 1000 0111 1101 1001". This is the base 2 representation of the base 10 number 427993. Thus, the Gödel number $\ulcorner x_2 = 0 \urcorner$ of "$x_2 = 0$" is the integer 428009.

Gödel numbers of terms, formulas, proofs, etc. can certainly be handled by algorithms (Turing machines). For instance, the set of integers that are valid Gödel numbers of PA terms is decidable. This is simply because there are algorithms for recognizing and parsing terms. Similarly, the sets of Gödel numbers of formulas and of sentences are decidable.

Theorem V.38 already discussed a variety of decidable relations and computable functions acting on Gödel numbers of formulas. We will need several additional computable functions that act on Gödel numbers of terms and formulas. The first one is the 3-ary substitution function *Sub* which accepts as input the Gödel numbers $\ulcorner A \urcorner$, $\ulcorner x_i \urcorner$ and $\ulcorner t \urcorner$ of a formula $A$, a variable $x_i$ and a term $t$ and produces the Gödel number $\ulcorner A(t/x_i) \urcorner$:

$$\mathit{Sub}(\ulcorner A \urcorner, \ulcorner x_i \urcorner, \ulcorner t \urcorner) \;:=\; \ulcorner A(t/x_i) \urcorner.$$

[4] So we are using four different concepts of zero: "$\emptyset$" is the alphabet symbol used for zeros in subscripts, and "0" is is used variously as a non-logic symbol, as an alphabet symbol, or as the actual integer zero.

| | |
|---|---|
| **Computable functions:** | $Sub(\ulcorner A\urcorner, \ulcorner x_i\urcorner, \ulcorner t\urcorner) \coloneqq \ulcorner A(t/x_i)\urcorner$.<br>$Num(n) \coloneqq \ulcorner \underline{n}\urcorner$. |
| **Decidable relations:** | $Proof_T \coloneqq \{\ulcorner P\urcorner : P \text{ is a } T\text{-proof}\}$.<br>$Prf_T \coloneqq \{\langle \ulcorner P\urcorner, \ulcorner A\urcorner\rangle : P \text{ is a } T\text{-proof of } A\}$. |
| **Computably enumerable set:** | $Thm_T \coloneqq \{\ulcorner A\urcorner : T \vdash A\}$. |

Figure VII.1: Some definable metamathematical functions and relations.

Another important function is the *Num* function that computes the Gödel numbers of numerals. Namely,

$$Num(n) \coloneqq \ulcorner \underline{n}\urcorner.$$

For instance, $Num(3) = \ulcorner S(S(S(0)))\urcorner$.

The set of Gödel numbers of valid proofs of an axiomatizable theory is also decidable. More precisely, let $T$ be an axiomatizable theory in the language of PA. We define $Proof_T$, the set of (Gödel numbers of) $T$-proofs, to be the unary relation defined by

$$Proof_T(n) \Leftrightarrow n \text{ is the Gödel number of a valid } T\text{-proof.}$$

This is decidable since there is an algorithm for recognizing syntactically correct $T$-proofs. We further define $Prf_T$ to be the binary relation which holds for exactly the pairs $\langle \ulcorner P\urcorner, \ulcorner A\urcorner\rangle$ such that $P$ is a valid $T$-proof of the formula $A$; namely,

$$Prf_T(n, m) \Leftrightarrow n \text{ is valid } T\text{-proof } P \text{ and the final formula in } P \text{ has Gödel number } m.$$

*Prf* is also decidable since there is an algorithm which can recognize valid $T$-proofs and extract their final formula.

Finally, we define the set $Thm_T$ of theorems of $T$. This set however is not in general decidable, only computably enumerable. Specifically,

$$Thm_T(m) \Leftrightarrow m = \ulcorner A\urcorner \text{ for a formula } A \text{ such that } T \vdash A.$$

$Thm_T$ is computably enumerable since it can be semidecided by an algorithm that does a brute force search for a $T$-proof of the formula $A$ with Gödel number $n$. An alternate way to define $Thm_T$ is:

$$Thm_T(m) \Leftrightarrow \exists y\, Prf_T(y, m).$$

This definition of $Thm_T$ as an existential quantification of $Prf_T$ shows that $Thm_T$ is computably enumerable (by Exercise V.8).

### VII.4.2 Undecidability of true theories of arithmetic

We are now ready to give the first proof of the First Incompleteness Theorem. This proof will apply to "true" theories of arithmetic, and more generally to "ω-consistent" theories. It is based on a direct many-one reduction to $\text{Halt}_0^{\text{TM}}$, the halting problem for Turing machines.

**Definition VII.21.** A theory $T$ is a *true theory of arithmetic* if it has language $L_{\mathrm{PA}}$ and contains only sentences that are true about $\mathcal{N}$. The latter condition is equivalent to $T \subseteq \mathrm{Th}\,\mathcal{N}$.

**Definition VII.22.** Let $T$ be a theory with language $L_{\mathrm{PA}}$. The theory $T$ is *$\omega$-inconsistent* provided the following holds for some formula $A(x_i)$ with one free variable $x_i$:

(a) $T$ proves $\exists x_i\, A(x_i)$, and

(b) For all $n \in \mathbb{N}$, $T$ proves $\neg A(\underline{n})$.

We say that $T$ is *$\omega$-consistent* provided that $T$ is not $\omega$-inconsistent.

It is clear that $\mathrm{Th}\,\mathcal{N}$ is $\omega$-consistent. Hence any true theory of arithmetic is $\omega$-consistent. This is because it cannot be the case that every $A(\underline{n})$ is false in $\mathcal{N}$ while $\exists x_i\, A(x_i)$ is true. Also, note that a $\omega$-consistent theory must be consistent.

The next theorem states our first version of the First Incompleteness Theorem. It is stated for $\omega$-consistent theories, but of course applies to any true theory that extends R.

**Theorem VII.23** (First Incompleteness Theorem — version 1)**.** *Suppose $T \supseteq \mathrm{R}$ is an axiomatizable, $\omega$-consistent theory. Then $T$ is not decidable.*

*Proof.* Since $T$ extends R, Theorems VII.18 and VII.20 tell us that every decidable relation is representable in $T$. Section VI.4 earlier defined the Gödel number $\ulcorner M \urcorner$ of a Turing machine $M$ to be a binary string. The initial symbol of $\ulcorner M \urcorner$ is 1, so we can view $\ulcorner M \urcorner$ also as being the integer whose base 2 representation is given by $\ulcorner M \urcorner$. We write $\underline{\ulcorner M \urcorner}$ to be the numeral for the integer $\ulcorner M \urcorner$; namely, if $i = \ulcorner M \urcorner \in \mathbb{N}$, then $\underline{\ulcorner M \urcorner}$ denotes the numeral $\underline{i}$.

Let HS be the decidable binary relation such that, for $\ulcorner M \urcorner$ the Gödel number of a Turing machine $M$,

$$\mathrm{HS}(\ulcorner M \urcorner, n) \;\Leftrightarrow\; M(\epsilon) \text{ halts in} \leq n \text{ steps.}$$

Here "HS" stands for "Halts-Steps". HS is a decidable relation, since it can be decided by using a universal Turing machine $U$ and running $U(\ulcorner M \urcorner, \epsilon)$ until it has simulated $M(\epsilon)$ for $n$ steps. This will discover whether $M$ has halted within at most $n$ steps. Since HS is decidable, it is representable in $T$, by a formula $A_{\mathrm{HS}}(x, y)$.

The predicate $\mathrm{Halt}_0^{\mathrm{TM}}$, expressing the halting problem for Turing machines, is undecidable. We claim that $\mathrm{Halt}_0^{\mathrm{TM}}(\ulcorner M \urcorner)$ is true if and only if $T \vdash \exists y\, A_{\mathrm{HS}}(\underline{\ulcorner M \urcorner}, y)$. To prove this, first suppose that $\mathrm{Halt}_0^{\mathrm{TM}}(\ulcorner M \urcorner)$ holds and $M(\epsilon)$ halts. Then $M(\epsilon)$ halts within some number $n$ of steps. Since $A_{\mathrm{HS}}$ represents HS, this means $T \vdash A_{\mathrm{HS}}(\underline{\ulcorner M \urcorner}, \underline{n})$ and thus of course $T \vdash \exists y\, A_{\mathrm{HS}}(\underline{\ulcorner M \urcorner}, y)$.

To prove the converse direction, suppose that $T \vdash \exists y\, A_{\mathrm{HS}}(\underline{\ulcorner M \urcorner}, y)$ but that $\mathrm{Halt}_0^{\mathrm{TM}}(\ulcorner M \urcorner)$ does not hold. The latter condition implies that $\mathrm{HS}(\ulcorner M \urcorner, n)$ is

false for all $n$. Since $A_{\mathrm{HS}}$ represents HS, that implies that $T \vdash \neg A_{\mathrm{HS}}(\underline{\ulcorner M \urcorner}, \underline{n})$. This contradicts the $\omega$-consistency of $T$ and finishes the proof of the claim.

We have shown that $\mathrm{Halt}_0^{\mathrm{TM}}(\ulcorner M \urcorner)$ is true if and only if $T \vdash \exists y\, A_{\mathrm{HS}}(\underline{\ulcorner M \urcorner}, y)$. In other words, $\mathrm{Halt}_0^{\mathrm{TM}}(\ulcorner M \urcorner)$ is true if and only if $\ulcorner A_{\mathrm{HS}}(\underline{\ulcorner M \urcorner}, y) \urcorner \in \mathit{Thm}_T$. Since the function mapping $\ulcorner M \urcorner$ to the Gödel number of the formula $A_{\mathrm{HS}}(\underline{\ulcorner M \urcorner}, y)$ is computable, and since $\mathrm{Halt}_0^{\mathrm{TM}}$ is undecidable, it must be that $\mathit{Thm}_T$ is undecidable. That proves Theorem VII.23. □

### VII.4.3 Undecidability via diagonalization

We now prove a stronger version of the First Incompleteness Theorem that applies to theories that may be not true or even $\omega$-consistent.

**Theorem VII.24** (First Incompleteness Theorem — version 2)**.** *Suppose $T \supseteq \mathrm{R}$ is an axiomatizable, consistent theory. Then $T$ is not decidable.*

**Corollary VII.25.** *There is no consistent and decidable theory $T$ extending* R*.*

*Proof of the corollary.* Any decidable theory is axiomatizable since the theory $T$ itself can be taken as the set of axioms. Thus the corollary follows immediately from Theorem VII.24. □

*Proof of Theorem VII.24.* ] Suppose for the sake of a contradiction that $T$ is decidable, that is, the set $\mathit{Thm}_T$ is decidable. Define the unary function *SelfSub* by

$$\mathit{SelfSub}(m) \;:=\; \mathit{Sub}(m, \ulcorner x_1 \urcorner, \mathit{Num}(m)).$$

To understand this, recall that $\mathit{Num}(m)$ is equal to $\ulcorner \underline{m} \urcorner$, the Gödel number of the numeral $\underline{m}$. We are interested in the case where $m$ is the Gödel number $\ulcorner C \urcorner$ of a formula $C = C(x_1)$ with $x_1$ the only variable appearing free in $C$. In this case, $\mathit{SelfSub}(\ulcorner C \urcorner)$ is equal the Gödel number of the formula $C(\underline{\ulcorner C \urcorner}/x_1)$; i.e.,

$$\mathit{SelfSub}(\ulcorner C \urcorner) \;=\; \mathit{Sub}(\ulcorner C \urcorner, \ulcorner x_1 \urcorner, \mathit{Num}(\ulcorner C \urcorner)) \;=\; \ulcorner C(\underline{\ulcorner C \urcorner}) \urcorner.$$

Define *ThmSS* to be the unary relation on $N$ defined by

$$\mathit{ThmSS}_T(m) \;\Leftrightarrow\; \mathit{Thm}_T(\mathit{SelfSub}(m)).$$

Since *SelfSub* is a computable function and since we have assumed that $\mathit{Thm}_T$ is decidable, the relation $\mathit{ThmSS}_T$ is also decidable. Hence $\mathit{ThmSS}_T$ is representable in R, and hence representable in $T$, by some formula $A_{\mathit{ThmSS}_T}(x_1)$. Let $B_{\mathit{ThmSS}_T}(x_1)$ be the formula $\neg A_{\mathit{ThmSS}_T}(x_1)$.

We obtain a contradiction as follows:

$$\begin{array}{lll}
T \vdash B_{\mathit{ThmSS}_T}(\underline{\ulcorner B_{\mathit{ThmSS}_T} \urcorner}) & & \\
\quad \Leftrightarrow \quad T \vdash \neg A_{\mathit{ThmSS}_T}(\underline{\ulcorner B_{\mathit{ThmSS}_T} \urcorner}) & & \text{Definition of } B_{\mathit{ThmSS}_T} \\
\quad \Leftrightarrow \quad \ulcorner B_{\mathit{ThmSS}_T} \urcorner \notin \mathit{ThmSS}_T & & A_{\mathit{ThmSS}_T} \text{ represents } \mathit{ThmSS}_T \\
\quad \Leftrightarrow \quad T \nvdash B_{\mathit{ThmSS}_T}(\underline{\ulcorner B_{\mathit{ThmSS}_T} \urcorner}) & & \text{Definition of } \mathit{ThmSS}_T
\end{array}$$

That completes the proof of Theorem VII.24. □

The intuition behind this proof is that the formula $B_{ThmSS_T}(\underline{\ulcorner B_{ThmSS_T} \urcorner})$ is indirectly referring to itself and saying that it is not a theorem of $T$. This is reminiscent of Cantor's diagonal argument for the uncountability of the reals. The next section gives another proof, which uses a self-referential formula $D_A$ based on a strengthened form of the diagonal argument.

## VII.4.4 Undecidability via self-reference

We now give a second proof of the second version of the First Incompleteness Theorem, Theorem VII.24. This proof is based on a self-referential formula. This proof can be viewed as analogous to the second proof of the undecidability of the halting problem in Section V.6.3. Indeed, it uses a Diagonal Theorem that is very similar in spirit to the Diagonal Theorem V.57 for Turing machines.

**Theorem VII.26** (Diagonal Theorem for Theories of Arithmetic)**.**
*Let $A(x_1)$ be an $L_{\mathrm{PA}}$-formula. There is a formula $D_A$ such that* R *proves*

$$D_A \leftrightarrow A(\underline{\ulcorner D_A \urcorner}).$$

Thus, provably in R, the formula $D_A$ expresses the condition that the property defined by $A(x)$ holds for $x$ equal to the Gödel number of $D_A$ itself. It is somewhat remarkable that this equivalence of $D_A$ and $A(\underline{\ulcorner D_A \urcorner})$ is provable in the very weak theory R. This illustrates, however, how fundamental the Diagonal Lemma is.

*Proof.* Recall the function *SelfSub* from the previous proof. We would like to define $E_A(x_1)$ to be the formula $A(\mathit{SelfSub}(x_1))$; however, it is not possible to do this directly since *SelfSub* is an integer function, not a function symbol of first-order logic. Instead, we let the formula $A_{\mathit{SelfSub}}(x_1, y)$ represent the function *SelfSub* in R, and define $E_A(x_1)$ by

$$E_A(x_1) \;\coloneqq\; \exists y\,[A_{\mathit{SelfSub}}(x_1, y) \wedge A(y)]. \tag{VII.2}$$

Note how $E_A(x_1)$ states indirectly that $A(\mathit{SelfSub}(x_1))$ holds.

Let $D_A$ be the formula $E_A(\underline{\ulcorner E_A \urcorner})$, so that the Gödel number $\ulcorner D_A \urcorner$ of $D_A$ is equal to $\mathit{SelfSub}(\ulcorner E_A \urcorner)$, and $D_A$ is the formula

$$\exists y\,[A_{\mathit{SelfSub}}(\underline{\ulcorner E_A \urcorner}, y) \wedge A(y)].$$

Since $A_{\mathit{SelfSub}}(x_1, y)$ represents *SelfSub*,

$$\mathrm{R} \;\vdash\; A_{\mathit{SelfSub}}(\underline{\ulcorner E_A \urcorner}, y) \leftrightarrow y = \underline{\ulcorner D_A \urcorner}.$$

Therefore, by Theorem III.61(b) on substitution, R proves $D_A \leftrightarrow A(\underline{\ulcorner D_A \urcorner})$. That finishes the proof of the Diagonal Theorem. □

The second proof of Theorem VII.24 is in essence the same as the first proof, but it uses the Diagonal Theorem explicitly:

*Second proof of Theorem VII.24.* Suppose for the sake of a contradiction that $T$ is decidable, that is, the set $Thm_T$ is decidable. Hence the complement $\overline{Thm_T}$ is also decidable. Let the formula $A(x_1)$ represent $\overline{Thm_T}$ in the theory R and hence in the theory $T$. The intuition is that $A(x_1)$ says

$$\text{“}x_1 \text{ is not provable in } T\text{”},$$

or more accurately,

$$\text{“}x_1 \text{ is not the Gödel number of a theorem of } T\text{”}.$$

Let $D_A$ be the formula from the Diagonal Theorem VII.26 such that R proves $D_A \leftrightarrow A(\underline{\ulcorner D_A \urcorner})$. Then

$$\begin{array}{rcll} T \vdash D_A & \Leftrightarrow & \ulcorner D_A \urcorner \in Thm_T & \text{Definition of } Thm_T \\ & \Leftrightarrow & T \vdash \neg A(\underline{\ulcorner D_A \urcorner}) & A \text{ represents } \overline{Thm_T} \\ & \Leftrightarrow & T \vdash \neg D_A & \mathrm{R} \vdash D_A \leftrightarrow A(\underline{\ulcorner D_A \urcorner}) \text{ and } T \supseteq \mathrm{R}. \end{array}$$

This contradicts the consistency of $T$ and thereby completes the second proof of Theorem VII.24. □

Note how the formula $D_A$ was chosen to express the condition that $D_A$ is not a theorem of $T$. Loosely speaking, $D_A$ says of itself, “I am not $T$-provable”. The above proof uses this fact, plus the hypothesis that the property of being “$T$-provable” is decidable to obtain a contradiction.

**Corollary VII.27.**
(a) R, Q *and* PA *are not complete and not decidable.*
(b) $\mathrm{Th}\mathcal{N}$ *is not axiomatizable and not decidable.*

*Proof.* Part (a) follows from the fact that R, Q and PA are axiomatizable and consistent and by Theorems V.46 and VII.24.

Part (b) follows by the fact that $\mathrm{Th}\mathcal{N}$ is complete and consistent, and again by Theorems V.46 and VII.24.

Alternatively, since all four theories $\omega$-consistent, the corollary can be proved using Theorem VII.23 instead of Theorem VII.24 □

**Corollary VII.28.** *Let $T$ be a consistent, axiomatizable theory extending* R. *There is a true sentence $C$ (that is, $\mathcal{N} \models C$) such that $T \nvdash C$.*

Thus, for instance, PA does not prove all sentences that are true about the integers. We will see some explicit examples of such formulas $C$ when discussing the Second Incompleteness Theorem in Section VII.9.

*Proof.* By the First Incompleteness Theorem, $T$ is not complete. Hence there is some sentence $B$ such that neither $B$ nor $\neg B$ is a consequence of $T$. One of $B$ and $\neg B$ is true and is the desired formula $C$. □

**Definition VII.29.** Work with an arbitrary first-order language $L$. A sentence $B$ is *independent* of $T$ if neither $B$ nor $\neg B$ is a consequences of $T$.

Note that $T$ is not complete if and only if there is an independent sentence $B$.

The previous proof established:

**Corollary VII.30.** *Let $T$ be a consistent, axiomatizable theory extending* R*. Then there is a sentence $C$ which is independent of $T$.*

## VII.4.5 Undecidability of pure first-order logic

An immediate consequence of the undecidability of the finitely axiomatized theory Q is that the pure first-order logic in the language of PA is undecidable. This is established by the next theorem.

**Theorem VII.31.** *The set of valid $L_{\mathrm{PA}}$-sentences is undecidable.*

*Proof.* Let $\mathrm{Q}_{1-7}$ be the conjunction of the seven axioms of Q,

$$\mathrm{Q}_{1-7} \;:=\; \mathrm{Q}_1 \wedge \mathrm{Q}_2 \wedge \mathrm{Q}_3 \wedge \mathrm{Q}_4 \wedge \mathrm{Q}_5 \wedge \mathrm{Q}_6 \wedge \mathrm{Q}_7.$$

A sentence $A$ is provable in Q if and only if $\mathrm{Q}_{1-7} \to A$ is valid. Therefore the mapping $\ulcorner A \urcorner \mapsto \ulcorner \mathrm{Q}_{1-7} \to A \urcorner$ is a many-one reduction from theory Q to the set of logically valid $L_{\mathrm{PA}}$-sentences. Since Q is undecidable, the set of logically valid $L_{\mathrm{PA}}$-sentences is undecidable. □

The statement of Theorem VII.31 does not depend on any properties of the integers at all. All it uses is the fact that the language contains a constant symbol and two binary function symbols. In fact, the same theorem holds even for the case where there the language $L$ consists of a single binary predicate symbol. This can be proved by letting the only non-logical symbol be $\in$ and showing that there is a finitely axiomatized fragment of Zermelo-Fraenkel set theory that can interpret the theory Q. Describing Zermelo-Fraenkel set theory and what it means to interpret Q is beyond the scope of the present text however.[5]

[5]For the reader with knowledge of set theory: One way to argue this is that Zermelo-Fraenkel set theory ZF can interpret PA. Hence, a priori, some finite fragment of ZF can interpret Q.

### VII.4.6 Undefinability of truth

The above results imply that the theory $\mathrm{Th}\mathcal{N}$ is not decidable, so there is no algorithm that can decide whether an arbitrary given sentence is true in the integers. Of course, there are many properties that can be defined in $\mathcal{N}$ that are not decidable. A prominent example is the set of logical consequences of PA itself. Namely, let $\textsc{Prf}_{\mathrm{PA}}(x_1, x_2)$ be a formula that represents the binary predicate $Prf_{\mathrm{PA}}$ in R.[6] Then let $\textsc{Thm}_{\mathrm{PA}}$ be the formula

$$\exists y\, \textsc{Prf}_{\mathrm{PA}}(\ulcorner A \urcorner, y).$$

Clearly, $\textsc{Thm}_{\mathrm{PA}}$ is an $L_{\mathrm{PA}}$-formula that defines the set of formulas provable from the axioms of PA. So this set is definable in $\mathcal{N}$ even though, by the First Incompleteness Theorem, it is not decidable.

This raises the question of whether the theory $\mathrm{Th}\mathcal{N}$ is definable in $\mathcal{N}$. That is, the question of whether there is a formula $Tr(x_1)$ such that $\mathcal{N} \vDash Tr(\ulcorner A \urcorner)$ holds for exactly the sentences $A$ which are true in $\mathcal{N}$.

Such a formula would be called a "definition of truth". Tarski's theorem on the undefinability of truth shows that such a definition of truth does not exist.

**Theorem VII.32** (Undefinability of Truth)**.** *There is no formula $Tr(x_1)$ such that for all sentences $A$, $\mathcal{N} \vDash Tr(\ulcorner A \urcorner)$ holds if and only if $\mathcal{N} \vDash A$.*

In other words, there is no formula $Tr(x_1)$ such that for all sentences $A$, $\mathcal{N} \vDash A \leftrightarrow Tr(\ulcorner A \urcorner)$.

*Proof.* The proof is by contradiction. Assume that there is a definition of truth $Tr(x_1)$. By the Diagonal Theorem VII.26, there is a sentence $D_{\neg Tr}$ such that R proves

$$D_{\neg Tr} \leftrightarrow \neg Tr(\ulcorner \underline{D_{\neg Tr}} \urcorner).$$

Since R is a true theory, this equivalence holds in $\mathcal{N}$. We have

$$\begin{aligned} \mathcal{N} \vDash D_{\neg Tr} &\Leftrightarrow \mathcal{N} \vDash \neg Tr(\ulcorner \underline{D_{\neg Tr}} \urcorner) && \text{By choice of } D_{\neg Tr} \\ &\Leftrightarrow \mathcal{N} \nvDash D_{\neg Tr} && Tr \text{ is a definition of truth} \end{aligned}$$

This is a contradiction and finishes the proof of Theorem VII.32. □

## VII.5 Q implies R

This section is devoted to proving that $\mathrm{Q} \vDash \mathrm{R}$, i.e., $\mathrm{Q} \supset \mathrm{R}$. Consequently, the First Incompleteness Theorems apply to the theory Q.

**Theorem VII.33.** $\mathrm{Q} \vDash \mathrm{R}$.

The theorem is proved with a sequence of simple lemmas. The first states that Q proves the $\mathrm{R}_+$ axioms:

[6] Earlier we would have used the notation $A_{Prf_{\mathrm{PA}}}$ instead of $\textsc{Prf}_{\mathrm{PA}}$.

**Lemma VII.34.** *Let* $m, n \in \mathbb{N}$. *Then* $\mathrm{Q} \vDash \underline{m} + \underline{n} = \underline{m+n}$. *Equivalently,* $\mathrm{Q} \vDash S^m 0 + S^n 0 = S^{m+n} 0$.

*Proof.* Fix $m \in \mathbb{N}$. We prove that $\mathrm{Q} \vDash \underline{m} + \underline{n} = \underline{m+n}$ by induction on $n$. The base case, where $n = 0$, is that $\mathrm{Q} \vDash \underline{m} + 0 = \underline{m}$. This is an immediate consequence of axiom $\mathrm{Q}_4$. (See page 243 for the axioms of Q.)

For the induction step, let $n \geq 0$ and note that Q can prove the following equalities:

$$\begin{aligned}
\underline{m} + \underline{n{+}1} &= \underline{m} + S(\underline{n}) && \underline{n{+}1} \text{ and } S(\underline{n}) \text{ are the same term} \\
&= S(\underline{m} + \underline{n}) && \text{By axiom } \mathrm{Q}_5 \\
&= S(\underline{m+n}) && \text{By the induction hypothesis} \\
&= \underline{m+n+1} && S(\underline{m+n}) \text{ and } \underline{m+n+1} \text{ are the same term.}
\end{aligned}$$ □

The lemma was proved using induction. Of course, the induction axioms are not axioms of Q; rather, the induction is carried out outside the theory Q.

**Lemma VII.35.** *Let* $m \leq n$. *Then* $\mathrm{Q} \vDash \underline{m} \leq \underline{n}$.

*Proof.* This is immediate from the previous lemma, which states that $\mathrm{Q} \vDash \underline{n-m} + \underline{m} = \underline{n}$ and the fact that $m \leq n$ is an abbreviation for the formula $\exists x\,(x + \underline{m} = \underline{n})$. □

The next lemma states that Q proves the $\mathrm{R}_\bullet$ axioms:

**Lemma VII.36.** *Let* $m, n \in \mathbb{N}$. *Then* $\mathrm{Q} \vDash \underline{m} \cdot \underline{n} = \underline{m \cdot n}$. *Equivalently,* $\mathrm{Q} \vDash S^m 0 \cdot S^n 0 = S^{m \cdot n} 0$.

*Proof.* Fix $m \in \mathbb{N}$. We prove that $\mathrm{Q} \vDash \underline{m} \cdot \underline{n} = \underline{m \cdot n}$ by induction on $n$. The base case, where $n = 0$, is that $\mathrm{Q} \vDash \underline{m} \cdot 0 = 0$. This is an immediate consequence of axiom $\mathrm{Q}_6$.

For the induction step, let $n \geq 0$ and note that Q can prove the following equalities:

$$\begin{aligned}
\underline{m} \cdot \underline{n{+}1} &= \underline{m} \cdot S(\underline{n}) && \underline{n{+}1} \text{ and } S(\underline{n}) \text{ are the same term} \\
&= \underline{m} \cdot \underline{n} + \underline{m} && \text{By axiom } \mathrm{Q}_7 \\
&= \underline{m \cdot n} + \underline{m} && \text{By the induction hypothesis} \\
&= \underline{m \cdot n + m} && \text{By Lemma VII.34} \\
&= \underline{m \cdot (n+1)} && \underline{m \cdot n + m} \text{ and } \underline{m \cdot (n+1)} \text{ are the same term.}
\end{aligned}$$ □

The next lemma states that Q proves the $\mathrm{R}_{\neq}$ axioms:

**Lemma VII.37.** *Fix* $m \neq n$. *Then* $Q \vDash \underline{m} \neq \underline{n}$.

*Proof.* Without loss of generality, $m > n$. We prove the lemma by induction on $n$. In the base case where $n = 0$, $\underline{m}$ is the same as $S(\underline{m{-}1})$. And Axiom $\mathrm{Q}_2$ implies $\underline{m} \neq 0$. For the induction step, $n > 0$. The terms $\underline{m}$ and $\underline{n}$ are the same as $S(\underline{m{-}1})$ and $S(\underline{n{-}1})$. Therefore, Axiom $\mathrm{Q}_1$ implies that $\underline{m} = \underline{n} \to \underline{m{-}1} = \underline{n{-}1}$ The induction hypothesis states that $Q \vDash \underline{m{-}1} \neq \underline{n{-}1}$; hence $Q \vDash \underline{m} \neq \underline{n}$. □

The above lemmas have shown that Q proves the axioms $\mathrm{R}_{\neq}$, $\mathrm{R}_+$, and $\mathrm{R}_\bullet$.

**Lemma VII.38.** *Fix $m > n$. Then $Q \vDash m \not\leq n$. Hence also $Q \vDash m \not< n$.*

*Proof.* Recall that $\underline{m} \not\leq \underline{n}$ is an abbreviation for $\neg\exists x\,(x + \underline{m} = \underline{n})$. Thus, since we can let $p = m - n - 1$, we must prove that for all $n \geq 0$ and $p \geq 0$, $\mathrm{Q} \vDash \neg\exists x\,(x + \underline{n+p+1} = \underline{n})$. Since $S(\underline{n+p})$ is the same term as $\underline{n+p+1}$, we must prove $\mathrm{Q} \vDash \neg\exists x\,(x + S(\underline{n+p}) = \underline{n})$. By axiom $\mathrm{Q}_5$, $x + S(\underline{n+p}) = S(x + \underline{n+p})$. Thus it suffices to show that

$$\mathrm{Q} \vDash \neg\exists x\,(S(x + \underline{n+p}) = \underline{n}).$$

This is proved by induction on $n$.

The base case, where $n = 0$, is immediate by axiom $\mathrm{Q}_2$. For the induction step, we must show that $Q \vDash \neg\exists x\,(S(x + \underline{n+1+p}) = \underline{n+1})$. Axiom $\mathrm{Q}_5$ implies

$$S(x + \underline{n+1+p}) \;=\; S(S(x + \underline{n+p})).$$

Also, $\underline{n+1}$ is the same as $S(\underline{n})$. Axiom $\mathrm{Q}_1$ implies that

$$S(S(x + \underline{n+p})) = S(\underline{n}) \;\rightarrow\; S(x + \underline{n+p}) = \underline{n}.$$

From this, the induction step follows immediately from the induction hypothesis, which states that $Q \vDash \neg\exists x\,(S(x + \underline{n+p}) = \underline{n})$. □

The next two lemmas establish the axioms $\mathrm{R}_{\leq}$ and $\mathrm{R}'_{\leq}$ in Q.

**Lemma VII.39.** *Let $n \in \mathbb{N}$. Then $\mathrm{Q} \vDash \forall x\,(x \leq \underline{n} \vee \underline{n} < x)$.*

Recall again that $x \leq \underline{n}$ is an abbreviation for $\exists z\,(z + x = \underline{n})$ and further that $\underline{n} < x$ is an abbreviation for $\underline{n} \leq x \wedge \underline{n} \neq x$.

*Proof.* First note that Q proves $\underline{n} + 0 = \underline{n}$ and thus Q proves $x = 0 \rightarrow x \leq \underline{n}$. Therefore it suffices to prove that

$$\mathrm{Q} \;\vDash\; \forall x\,(x \neq 0 \rightarrow x \leq \underline{n} \vee \underline{n} < x).$$

This is proved by induction on $n$. The base case is $n = 0$. In this case, note that $x + 0 = x$ by axiom $\mathrm{Q}_3$, whence $\underline{0} \leq x$. From $x \neq 0$, it follows that $0 < \underline{x}$.

For the induction step, let $n \geq 0$. We need to show $x \neq 0 \rightarrow x \leq \underline{n+1} \vee \underline{n+1} < x$. We argue informally using reasoning that can be formalized in Q. The assumption that $x \neq 0$ implies that there is a $y$ such that $x = S(y)$ by axiom $\mathrm{Q}_3$. The induction hypothesis states that $y \leq \underline{n} \vee \underline{n} < y$. (The induction hypothesis is needed only to handle the case $y \neq 0$.) If $y \leq \underline{n}$, then $x = S(y) \leq S(\underline{n}) = \underline{n+1}$ by axiom $\mathrm{Q}_5$. On the other hand, if $\underline{n} < y$, then $\underline{n} \neq y$ (since $\underline{n} \not< \underline{n}$) and there is a $z$ such that $z + \underline{n} = y$. From $\underline{n} \neq y$, axiom $\mathrm{Q}_1$ gives $\underline{n+1} \neq S(y) = x$. From $z + \underline{n} = y$, axiom $\mathrm{Q}_5$ gives $z + \underline{n+1} = z + S(\underline{n}) = S(y) = x$, so $\underline{n+1} \leq x$. From these $\underline{n+1} < x$ is immediate. □

**Lemma VII.40.** *Let $n \in \mathbb{N}$. Then* Q *proves*

$$\forall x\,[\,x \leq \underline{n} \;\rightarrow\; x = \underline{0} \vee x = \underline{1} \vee \cdots \vee x = \underline{n-1} \vee x = \underline{n}\,]$$

*Proof.* This is proved by induction on $n$. Again, we argue informally in Q, namely using reasoning that can be formalized by Q. Note that we can always assume that $x \neq 0$, in which case, there is a $y$ such that $S(y) = x$ by axiom $Q_3$. For the base case, it will suffice to assume that $x = S(y) \neq 0$ and prove that $x \not\leq 0$. This follows as axioms $Q_4$ and $Q_2$ imply that $z + S(y) = S(z + y) \neq 0$ and hence $z + x = 0$ cannot hold.

For the induction step, we assume that $x \leq \underline{n+1}$ and prove that

$$x = \underline{0} \vee x = \underline{1} \vee \cdots \vee x = \underline{n} \vee x = \underline{n+1}. \tag{VII.3}$$

The assumption $x \leq \underline{n+1}$ means that there is a $z$ so that $z + x = \underline{n+1}$. Using $x = S(y)$ and axiom $Q_5$, we have $S(z+y) = S(\underline{n})$. Then from axiom $Q_1$, $z+y = \underline{n}$, so $y \leq \underline{n}$. The induction hypothesis implies that

$$y = \underline{0} \vee y = \underline{1} \vee \cdots \vee y = \underline{n-1} \vee y = \underline{n}.$$

If $y = \underline{i}$ for some $i \leq n$, we get $x = S(y) = S(\underline{i}) = \underline{i+1}$. In other words, $x = \underline{k}$ for some $k \leq n + 1$ as desired. □

The above lemmas complete the proof of Theorem VII.33. □

## VII.6 Techniques for Representability

The main goal of the next three sections is to prove Theorem VII.20, that every decidable predicate and every computable function is representable in R. We have so far established only a few things to be representable in R. This includes the equality (=) and less-than-or-equal-to (≤) relations, and the successor, addition, and multiplication functions. We shall ramp up rather quickly to show that many other relations and functions are representable.

Our theorems about representability will be expressed in terms of representability in R, but of course apply also to any consistent theory $T \supseteq \mathrm{R}$. Since R is a true theory, any formula $A_S(x_1, \ldots, x_k)$ that represents a $k$-ary relation $S$ also defines that relation in the non-negative integers $\mathcal{N}$ (in the sense of Definition III.106). Likewise, any formula $A_f(x_1, \ldots, x_k, y)$ that represents a $k$-ary function $f$ also defines the function $f$ in $\mathcal{N}$.

### Boolean combinations of relations.

Boolean combinations of relations are formed by taking set complementation, union, intersection, etc. These operations preserve the property of being representable:

**Theorem VII.41.** *Let $k \in \mathbb{N}$ and suppose $S_1$ and $S_2$ are $k$-ary relations and representable in* R*. Then the following relations are also representable in* R*:*

(a) $\overline{S_1} := \mathbb{N}^k \setminus S_1 = \{\langle n_1, \ldots, n_k\rangle \in \mathbb{N}^k : \langle n_1, \ldots, n_k\rangle \notin S_1\}$.

(b) $S_1 \cap S_2$.

(c) $S_1 \cup S_2$.

(d) $S_1 \setminus S_2$.

*Proof.* Let $A_1(x_1, \ldots, x_k)$ and $A_2(x_1, \ldots, x_k)$ represent $S_1$ and $S_2$ in R. For (a), we claim that $\neg A_1$ represents $\overline{S_1}$. To prove this, we must show that, for all $n_1, \ldots, n_k \in \mathbb{N}$,

(i) If $S_1(n_1, n_2, \ldots, n_k)$ is false, then $\mathrm{R} \vdash \neg A_1(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k})$, and

(ii) If $S_1(n_1, n_2, \ldots, n_k)$ is true, then $\mathrm{R} \vdash \neg\neg A_1(\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k})$.

These follow immediately from the definition of what it means for $A_1$ to represent $S_1$.

For (b), we claim that $A_1 \wedge A_2$ represents $S_1 \cap S_2$. To prove this, we must show that, for all $n_1, \ldots, n_k \in \mathbb{N}$,

(i) If $S_1(\vec{n})$ and $S_2(\vec{n})$ are both true, then $\mathrm{R} \vdash A_1(\underline{\vec{n}}) \wedge A_2(\underline{\vec{n}})$ and

(ii) If $S_1(\vec{n})$ and $S_2(\vec{n})$ are not both true, then $\mathrm{R} \vdash \neg(A_1(\underline{\vec{n}}) \wedge A_2(\underline{\vec{n}}))$,

where $\vec{n}$ and $\underline{\vec{n}}$ are shorthand notations for $n_1, \ldots, n_k$ and $\underline{n_1}, \underline{n_2}, \ldots, \underline{n_k}$. By the definition of the representability of $S_1$ and $S_2$, if $S_1(\vec{n})$ and $S_2(\vec{n})$ are both true then R proves both $A_1(\underline{\vec{n}})$ and $A_2(\underline{\vec{n}})$. Thus (i) holds. Otherwise, if $S_1(\vec{n})$ and $S_2(\vec{n})$ are not both true, then R proves at least one of $\neg A_1(\underline{\vec{n}})$ or $\neg A_2(\underline{\vec{n}})$. Thus (ii) holds.

Part (c) can be proved by showing similarly that $A_1 \vee A_2$ represents $S_1 \cup S_2$. Alternatively, part (c) follows from (a) and (b), since $S_1 \cup S_2 = \overline{\overline{S_1} \cap \overline{S_2}}$. Part (d) is proved by a similar argument. □

**Example VII.42.** We show that the greater-than ($>$) relation, the less-than ($<$) and the greater-than-or-equal-to ($\geq$) relation are representable in R. The greater-than relation is $\{\langle m, n\rangle : m > n\}$. This is the complement of the less-than-or-equal-to ($\leq$) which was shown to be representable in Example VII.13. Hence, by Theorem VII.41, the greater-than ($>$) relation is representable.

The less-than $<$ relation is the set difference of the less-than-or-equal-to $\leq$ relation and the equality $=$ relation. Those two relations are represented by $x_1 \leq x_2$ and $x_1 = x_2$. Hence, by Theorem VII.41, the less-than $<$ relation is represented by the formula $x_1 \leq x_2 \wedge x_1 \neq x_2$. As specified in Notation VII.8, this is the same as the formula $x_1 < x_2$.

## Composition with representable functions

We now take the topic of composition and representability. The goal is to have flexibility in combining representable relations and functions to form new representable relations and functions. As a simple example, we would like to be able to say that the 2-ary relation defined by $x_1 \leq x_2 \wedge x_2 \leq x_3$ is representable, just because $\leq$ is representable. Note this does not quite fit the framework of the previous theorem on Boolean combinations, since $x_1 \leq x_2$ and "$x_2 \leq x_3$ involve different sets of variables. For another example, we would like to know that $P(x) \cdot P(y) \leq x + y$ defines a representable relation, just because the relation $\leq$

and the functions $P$ and + are representable. All of these will follow from the following general theorem.

**Theorem VII.43.** (Composition with Representable Functions) *Work in the theory* R. *Suppose $g_1, g_2, \ldots, g_\ell$ are $k$-ary representable functions. Let each $g_i$ be represented by the formula $A_{g_i}(x_1, \ldots, x_k, y)$. We write $\vec{x}$ and $\vec{n}$ to denote $x_1, \ldots, x_k$ and $n_1, \ldots, n_k$.*

(a) *Suppose $S$ is an $\ell$-ary representable relation and is represented by the formula $A_S(y_1, \ldots, y_\ell)$. Then there is a formula $A_{S'}(x_1, \ldots, x_k)$ that represents the $k$-ary relation $S'$ defined by*

$$S'(\vec{n}) \;\Leftrightarrow\; S(g_1(\vec{n}), g_2(\vec{n}), \ldots, g_\ell(\vec{n})).$$

(b) *Suppose $f$ is an $\ell$-ary representable function and is represented by the formula $A_f(y_1, \ldots, y_\ell, z)$. Then there is a formula $A_{f'}(x_1, \ldots, x_k, z)$ that represents the $k$-ary function $f'$ defined by*

$$f'(\vec{n}) \;=\; f(g_1(\vec{n}), g_2(\vec{n}), \ldots, g_\ell(\vec{n})).$$

*Proof.* By the definition of representability, for all $i = 1, \ldots, \ell$ and all $n_1, \ldots, n_k \in \mathbb{N}$,

$$\mathrm{R} \;\vdash\; \forall y\,[A_{g_i}(\underline{n_1}, \ldots, \underline{n_k}, y) \leftrightarrow y = \underline{g_i(n_1, \ldots, n_k)}] \tag{VII.4}$$

To prove (a), we know that for all $m_1, \ldots, m_\ell \in \mathbb{N}$,

(i) If $S(m_1, \ldots, m_\ell)$ holds, then R proves $A_S(\underline{m_1}, \ldots, \underline{m_\ell})$.

(ii) If $S(m_1, \ldots, m_\ell)$ is false, then R proves $\neg A_S(\underline{m_1}, \ldots, \underline{m_\ell})$.

The intuition is that we would like to let $A_{S'}$ be the "formula"

$$A_S(g_1(\vec{x}), g_2(\vec{x}), \ldots, g_\ell(\vec{x})),$$

but this makes no sense the $g_i$'s are not function symbols, they are functions. Instead, we exploit the representability of the $g_i$'s and define $A_{S'}(x_1, \ldots, x_k)$ to be the formula[7]

$$A_{S'}(x_1, \ldots, x_k) \;\coloneqq\; \exists y_1\, \exists y_2 \cdots \exists y_\ell \Big[ \big( \textstyle\bigwedge_{i=1}^{\ell} A_{g_i}(\vec{x}, y_i) \big) \wedge A_S(y_1, \ldots, y_\ell) \Big].$$

We need to show that, for all $n_1, \ldots, n_k \in \mathbb{N}$,

(i′) If $S'(n_1, \ldots, n_k)$ holds, then R proves $A_{S'}(\underline{n_1}, \ldots, \underline{n_k})$.

(ii′) If $S'(n_1, \ldots, n_k)$ is false, then R proves $\neg A_{S'}(\underline{n_1}, \ldots, \underline{n_k})$.

It is straightforward to prove (i′), since if $S'(n_1, \ldots, n_k)$ holds, then letting $m_i = g(n_1, \ldots, n_k)$ and using (VII.4) and condition (i) shows that R proves $A_{S'}(\underline{\vec{n}})$. The proof that (ii′) holds is similar, using condition (ii) in place of (i).

[7] We could equally as well have used the formula

$$\forall y_1\, \forall y_2 \cdots \forall y_\ell \Big[ \big( \textstyle\bigwedge_{i=1}^{\ell} A_{g_i}(\vec{x}, y_i) \big) \to A_S(y_1, \ldots, y_\ell) \Big].$$

The proof of (b) is similar to the proof of (a). By the representability of $f$, we have that for all $m_1, \ldots, m_\ell \in \mathbb{N}$,

$$\mathrm{R} \vdash \forall z\,[A_f(\underline{m_1}, \ldots, \underline{m_\ell}) = z \leftrightarrow z = \underline{f(m_1, \ldots, m_\ell)}] \tag{VII.5}$$

We let $A_{f'}$ be the formula[8]

$$A_{f'}(x_1, \ldots, x_k, z) \;:=\; \exists y_1\, \exists y_2 \cdots \exists y_\ell \Big[\big(\textstyle\bigwedge_{i=1}^{\ell} A_{g_i}(\vec{x}, y_i)\big) \wedge A_f(y_1, \ldots, y_\ell, z)\Big].$$

We need to show that, for all $n_1, \ldots, n_k \in \mathbb{N}$,

$$\mathrm{R} \vdash \forall z\,[A_f(\underline{n_1}, \ldots, \underline{n_k}, z) \leftrightarrow z = \underline{f(n_1, \ldots, n_k)}].$$

Letting $m_i = g_i(n_1, \ldots, n_\ell)$, this follows from instances of (VII.4) and (VII.5). □

To make good use of Theorem VII.43, we need to know that the "projection" functions are representable.

**Definition VII.44.** Let $1 \le i \le k$ be integers. The *projection* function $\pi^k_i$ is the $k$-ary function defined by

$$\pi(x_1, \ldots, x_k) \;=\; x_i.$$

**Theorem VII.45.** *The projection functions $\pi^k_i$ are representable in* R.

*Proof.* This is very simple to prove. Let $A_{\pi^k_i}(x_1, \ldots, x_k, y)$ be the formula $y = x_i$. The fact that $A_{\pi^k_i}$ represents $\pi^k_i$ is easy to prove with the $\mathrm{R}_{\neq}$ axioms. □

**Example VII.46.** We show that the 3-ary relation $S = \{\langle n_1, n_2, n_3\rangle : n_1 \le n_2 \le n_3\}$, which was discussed before the statement of Theorem VII.43, is representable. Define the relations

$$\begin{aligned} S_0 &:= \{\langle n_1, n_2\rangle : n_1 \le n_2\} \\ S_1 &:= \{\langle n_1, n_2, n_3\rangle : n_1 \le n_2\} \\ S_2 &:= \{\langle n_1, n_2, n_3\rangle : n_2 \le n_3\}. \end{aligned}$$

Example (VII.13) showed that $S_0$ is representable. Since for all $n_1, n_2, n_3$

$$\begin{aligned} S_1(n_1, n_2, n_3) &\;\Leftrightarrow\; S_0(\pi^3_1(n_1, n_2, n_3), \pi^3_2(n_1, n_2, n_3)) \\ S_2(n_1, n_2, n_3) &\;\Leftrightarrow\; S_0(\pi^3_2(n_1, n_2, n_3), \pi^3_3(n_1, n_2, n_3)), \end{aligned}$$

Theorem VII.43(a) gives that $S_1$ and $S_2$ are representable. Since $S = S_1 \cap S_2$, Theorem VII.41 implies that $S$ is also representable.

[8] Similarly to before, we could equally as well have used the formula

$$\forall y_1\, \forall y_2 \cdots \forall y_\ell \Big[\big(\textstyle\bigwedge_{i=1}^{\ell} A_{g_i}(\vec{x}, y_i)\big) \to A_f(y_1, \ldots, y_\ell, z)\Big].$$

**Example VII.47.** Now consider $S = \{\langle n_1, n_2\rangle : P(n_1) \cdot P(n_2) \leq n_1 + n_2\}$. We show this is representable using Theorem VII.43. Since the predecessor function $P$ and the multiplication function $\cdot$ are representable, Theorems VII.43 and VII.45 tell us that

$$h(x_1, x_2) = P(x_1) \cdot P(x_2) = P(\pi^2_1(x_1, x_2)) \cdot P(\pi^2_2(x_1, x_2))$$

is a representable function. Then, with $S_0$ the relation $\leq$ as in the previous example,

$$S = \{\langle n_1, n_2\rangle : S_0(h(n_1, n_2), n_1 + n_2)\}$$

is representable by Theorem VII.43 since addition $(+)$ is representable.

As a general principle, if $L$ is a first-order language for the integers containing functions symbols interpreted by representable functions and containing predicate symbols interpreted by representable relations, then any $L$-term defines a representable function, and any atomic $L$-formula defines a representable relation. This should be clear from the last two examples, so we omit the proof. (A formal proof would first use induction on the complexity of terms along with Theorems VII.43 and VII.45, and then use induction on the complexity of quantifier-free formulas using Theorem VII.41.)

## Bounded quantifiers

We now show that bounded quantifiers can be used freely when defining representable functions relations.

**Definition VII.48** (Bounded quantifiers)**.** Let $A$ be a formula, $x$ be a variable and $t$ be a term not involving $x$. The notations $\forall x{\leq}t\, A$ and $\exists x{\leq}t\, A$ are abbreviations for the formulas $\forall x\,(x \leq t \to A)$ and $\exists x\,(x \leq t \wedge A)$. The constructions "$\forall x{\leq}t$" and "$\exists x{\leq}t$" are called *bounded quantifiers.*

The notations $\forall x{<}t\, A$ and $\exists x{<}t\, A$ are defined similarly.

**Example VII.49.** Here are some examples of properties that can be expressed with the aid of bounded quantifiers:

| | | |
|---|---|---|
| $y \mid x$ | $\exists z{\leq}x\,(y \cdot z = x)$ | $x$ is a multiple of $y$, or $y$ divides $x$ |
| $\lfloor x/y \rfloor = z$ | $\exists r{<}y\,(y \cdot z + r = x)$ | $x$ divided by $y$ rounded down |
| $Rem(x, y) = r$ | $r < y \wedge \exists z{\leq}x\,(z \cdot y + r = x)$ | Remainder of $x$ divided by $y$ |

The second and third examples only make sense for non-zero values of $y$. Our convention is that $\lfloor x/0 \rfloor = 0$ and $Rem(x, 0) = 0$.

**Theorem VII.50.** *Suppose that $S$ is a $k$-ary representable relation, represented in* R *by the formula $A_S(x_1, \ldots, x_k, y)$. Then the formulas*

$$\forall y{\leq}x_k\, A_S(x_1, \ldots, x_k, y) \qquad \textit{and} \qquad \exists y{\leq}x_k\, A_S(x_1, \ldots, x_k, y)$$

*represent, in* R*, the $k$-ary relations*

$$S_\forall \;:=\; \{\langle m_1,\dots,m_k\rangle : \textit{for all } n \le m_k,\ S(m_1,\dots,m_k,n)\}$$

*and*

$$S_\exists \;:=\; \{\langle m_1,\dots,m_k\rangle : \textit{for some } n \le m_k,\ S(m_1,\dots,m_k,n)\}$$

The theorem also holds for bounded quantifiers $\forall y{<}x_k\, A$ and $\exists y{<}x_k\, A$ with strict inequality.

*Proof.* We prove the $\forall$ case. First suppose $m_1,\dots,m_k \in \mathbb{N}$ and $S_\forall(m_1,\dots,m_k)$ is true. Since $A$ represents $S$, this implies that, for all $n \le m_k$, R proves $A_S(\underline{m_1},\dots,\underline{m_k},\underline{n})$. Let $A_{S_\forall}(x_1,\dots,x_k)$ be the formula $\forall y{\le}x_k\, A_S(x_1,\dots,x_{k-1},y)$. Recall from Theorem VII.11(b) that R proves that

$$y \le \underline{m_k} \leftrightarrow y = 0 \vee y = \underline{1} \vee \cdots \vee y = \underline{m_k}.$$

It thus follows that R proves $\forall y{\le}\underline{m_k}\, A_S(\underline{m_1},\dots,\underline{m_k})$. In other words, R proves $A_{S_\forall}(\underline{m_1},\dots,\underline{m_k})$.

Second, suppose that $S_\forall(m_1,\dots,m_k)$ is false. Thus there is some $n \le m_k$ such that $S(m_1,\dots,m_k,n)$ is false and by the representability of $S$, the theory R proves $\neg A_S(\underline{m_1},\dots,\underline{m_k},\underline{n})$. Since $\mathrm{R} \vdash \underline{n} \le \underline{m_k}$, it follows that R proves $\neg A_{S_\forall}(\underline{m_1},\dots,\underline{m_k})$ as desired.

The $\exists$ case of the theorem follows immediately from the $\forall$ by duality since $\exists y{\le}x_k\, A_S(x_1,\dots,x_k,y)$ is logically equivalent $\neg\forall y{\le}x_k \neg A_S(x_1,\dots,x_k,y)$ and since Theorem VII.41 states that the set of representable relations is closed under complementation. □

**Corollary VII.51.** *The following relations are representable in* R *(the first three are from Example VII.49):*

(a) *The binary relation $y \mid x$.*

(b) *The graph of the binary function $\lfloor x/y \rfloor$.*

(c) *The graph of the binary function $Rem(x,y)$.*

(d) *The unary relation Power2* $:= \{n : n \textit{ is a power of } 2\} = \{2^i : i \in \mathbb{N}\}$.

*Proof.* The representability of $y \mid x$ follows from the formula in Example VII.49, and from Theorem VII.50. The representability of the graph of $\lfloor x/y \rfloor$ also follows from Example VII.49, but adjusted to handle the case where $y = 0$. Namely, the 3-ary relation $\lfloor x/y \rfloor = z$ can be defined by

$$[y \ne 0 \wedge \exists r{<}y\,(y \cdot z + r = x)] \vee [y = 0 \wedge z = 0].$$

The representability of the 3-ary relation $Rem(x,y) = z$ is proved similarly.

The representability of the property $Power2(x)$ follows from the fact it can be expressed as:

$$(\forall z{\le}x)[z \mid x \;\rightarrow\; z = \underline{1} \vee \underline{2} \mid z].$$ □

The earlier mentioned "general principle" discussed after Example VII.47 can now be updated to allow bounded quantifiers to be used in formulas representing relations and functions. As before, suppose $L$ is a first-order language for the integers containing function symbols interpreted by representable functions and predicate symbols interpreted by representable relations. Then any $L$-formula formed from atomic $L$-formula using propositional connectives and bounded quantifiers represents a relation (in R). This can be proved formally by induction on the complexity of formulas using Theorems VII.41, VII.43, VII.45 and VII.50.

## Graph of a function.

We next show that a function $f$ is representable if and only if its graph $G_f$ is a representable relation.

**Example VII.52.** The predecessor function $P(n) = n \dot{-} 1$ was defined in Theorem VI.21. Its graph is the set of ordered pairs

$$G_P \;:=\; \{\langle m,n\rangle : P(m) = n\} \;=\; \{\langle m,n\rangle : S(n) = m \vee (m = 0 \wedge n = 0)\}.$$

The graph $G_P$ of $P$ is a binary relation. We claim it is represented by the formula $A_{G_P}$ defined by

$$A_{G_P}(x_1,x_2) \;:=\; S(x_2) = x_1 \vee (x_1 = 0 \wedge x_2 = 0).$$

To prove this, it is necessary and sufficient to show that R proves the following:

(i) R proves $S(0) = 0 \vee (0 = 0 \wedge 0 = 0)$.

(ii) For $n \neq 0$, R proves $\neg[S(\underline{n}) = 0 \vee (0 = 0 \wedge \underline{n} = 0)]$.

(iii) For $m > 0$, R proves $S(\underline{m-1}) = \underline{m} \vee (\underline{m} = 0 \wedge \underline{m-1} = 0)$

(iv) For $m > 0$ and $n \neq m - 1$, R proves $\neg[S(\underline{n}) = \underline{m} \vee (\underline{m} = 0 \wedge \underline{n} = 0)]$.

Items (i) and (iii) express the conditions that if $n = P(m)$, then R proves $A_{G_P}(m,n)$. Items (ii) and (iv) express the conditions that if $n \neq P(m)$, then R proves $\neg A_{G_P}(m,n)$. All of (i)-(iv) are logical consequences of the $R_{\neq}$ axioms of R.

**Example VII.53.** Continuing the previous example, we claim that the predecessor *function* is also representable. For this, we need a formula $A_P(x_1,y)$ that satisfies, for every $m \in \mathbb{N}$,

$$\mathrm{R} \;\vdash\; \forall y\,[A_P(\underline{m},y) \leftrightarrow y = \underline{m \dot{-} 1}].$$

Such a formula must define the graph $G_P$ of $P$; hence it is tempting to set $A_P$ equal to the formula $A_{G_P}$. This, however, does not work, as R does not prove $\forall y\,(Sy \neq 0)$ and hence does not prove $\forall y\,[A_{G_P}(0,y) \to y = 0]$. Nonetheless, the predecessor function $P$ is representable. The formula $A_P$ representing $P$ can be chosen to be

$$A_P(x_1,y) \;:=\; (x_1 \neq 0 \wedge S(y) = x_1) \vee (x_1 = 0 \wedge y = 0).$$

Exercise VII.7(a) asks you to prove that this choice of $A_P$ represents the predecessor function $P$.

We have seen that both the predecessor function and the graph of the predecessor relation are representable. This is an example of the next theorem.

**Theorem VII.54.** *Let $f$ be a $k$-ary function on $\mathbb{N}$. The graph $G_f$ of $f$ is representable in* R *if and only if $f$ is representable in* R.

*Proof.* It is easy to prove that if $A_f(x_1, \ldots, x_k, y)$ is representable, then $A_f$ also represents the graph $G_f$ of $f$. This follows by the fact that the $\mathrm{R}_{\neq}$ axioms and the condition (VII.1) defining what it means for $A_f$ to represent $f$ imply the conditions (a) and (b) in Definition VII.12 about the representability of a relation.

Conversely, suppose $A_{G_f}(x_1, \ldots, x, y)$ represents the $(k+1)$-ary relation $G_f$. As the previous examples show, $A_{G_f}$ may not represent $f$. We instead define $A_f(x_1, \ldots, x_k, y)$ to be the formula

$$A_f(x_1, \ldots, x_k, y) \;\coloneqq\; A_{G_f}(x_1, \ldots, x_k, y) \wedge \forall z{<}y\, (\neg A_{G_f}(x_1, \ldots, x_k, z)).$$

We claim that $A_f$ represents $f$. To prove this, we show that, for any $n_1 \ldots, n_k \in \mathbb{N}$ and for $m = f(n_1, \ldots, n_k)$,

(i) R proves $A_f(\underline{n_1}, \ldots, \underline{n_k}, \underline{m})$, and

(ii) R proves $A_f(\underline{n_1}, \ldots, \underline{n_k}, y) \to y = \underline{m}$.

To help prove (i) and (ii), we will also prove

(iii) R proves $y < \underline{m} \to \neg A_f(\underline{n_1}, \ldots, \underline{n_k}, y)$, and

(iv) R proves $\underline{m} < y \to \neg A_f(\underline{n_1}, \ldots, \underline{n_k}, y)$.

Theorem VII.11(c) states that R proves that $y < \underline{m} \to y = 0 \vee y = \underline{1} \vee \cdots \vee y = \underline{m{-}1}$. Since $A_{G_f}$ represents the graph of $f$, we have that R proves $A_{G_f}(\underline{n_1}, \ldots, \underline{n_k}, \underline{m})$ and that R proves $\neg A_{G_f}(\underline{n_1}, \ldots, \underline{n_k}, \underline{m'})$ for each $m' < m$. Items (i) and (iii) follows immediately from these facts and the definition of $A_f$.

Item (iv) holds since R proves $A_{G_f}(\underline{n_1}, \ldots, \underline{n_k}, \underline{m})$ and by the definition of $A_f$.

Theorem VII.11(d) established that R proves $y < \underline{m} \vee y = \underline{m} \vee \underline{m} < y$. Therefore, item (ii) follows from (iii) and (iv). □

**Example VII.55.** The functions $\lfloor x/y \rfloor$ and $Rem(x, y)$ are representable in R. This is because their graphs are representable by Corollary VII.51.

**Example VII.56.** The functions $z = \max(x, y)$ and $z = \min(x, y)$ are representable since their graphs are represented by the formulas

$$(x \le y \wedge z = y) \vee (\neg x \le y \wedge z = x) \quad \text{and} \quad (x \le y \wedge z = x) \vee (\neg x \le y \wedge z = y).$$

The $\dot{-}$ function $z = x \dot{-} y$ is representable since its graph $G_{\dot{-}}$ is represented by the formula $(y \le x \wedge z + y = x) \vee (\neg y \le x \wedge z = 0)$.

## Regular minimization

The minimization operator $\mu x\,(\cdots)$ is used to define a (partial) computable function, letting "$\mu x(\cdots)$" mean "the least $x$ such that $\cdots$". We have already encountered three different forms of the minimization operator in Example V.31 Exercises V.43 and V.44. To repeat the definitions, we have:

**Definition VII.57.** Let $S(x_1, \ldots, x_k, y)$ be a $(k+1)$-ary relation. Then

$$g(x_1, \ldots, x_k) \;=\; \mu y\, S(x_1, \ldots, x_k, y) \tag{VII.6}$$

defines the $k$-ary partial function such that $g(n_1, \ldots, n_k) = m$ provided that $m$ is the least value (if any) such that $S(n_1, \ldots, n_k, m)$ holds. If there is no such $y$, then $g(n_1, \ldots, n_k)$ is undefined (diverges).

Let $f(x_1, \ldots, x_k, y)$ be a $(k+1)$-ary function. Then

$$h(x_1, \ldots, n_k) \;=\; \mu y\,(f(x_1, \ldots, x_k, y) = 0) \tag{VII.7}$$

defines the $k$-ary partial function such that $h(n_1, \ldots, n_k) = m$ provided that $m$ is the least value (if any) such that $f(n_1, \ldots, n_k, m)\!\downarrow = 0$ and such that for all $m' < m$, $f(n_1, \ldots, n_m)$ converges and is non-zero. If there is no such $y$, then $h(n_1, \ldots, n_k)$ is undefined (diverges).

If $g$ and $h$ are total functions, the definitions (VII.6) and (VII.7) are called *regular minimization*. For instance, the function $g$ is defined by regular minimization provided that for all $\vec{n} \in \mathbb{N}$, there is an $m \in \mathbb{N}$ such that $S(n_1, \ldots, n_k, m)$.

The method of Example V.31 shows that if $S$ is decidable, then the function $g$ defined by (VII.6) is partial computable. If (VII.6) is a regular minimization, then $g$ is computable. In both cases, the algorithm for computing $g(n_1, \ldots, n_k)$ acts by successively checking whether $S(n_1, \ldots, n_k, m)$ holds for $m = 0, 1, 2, \ldots$. Similarly, the function $h$ defined by (VII.7) is partial computable if $f$ is partial computable.

**Theorem VII.58.**

(a) *Let $g$ be a $k$-ary function defined by regular minimization from a relation $S$ by (VII.6). Suppose that $S$ is representable in* R. *Then $g$ is representable in* R.

(a) *Let $h$ be a $k$-ary function defined by regular minimization from a function $f$ by (VII.7). Suppose that $S$ is representable in* R. *Then $h$ is representable in* R.

*Proof.* Let $A_S(x_1, \ldots, x, y)$ represent $S$. Then $g(x_1, \ldots, x_k, y)$ is represented by

$$A_S(x_1, \ldots, x_k, y) \wedge \forall z{<}y\,(\neg A_S(x_1, \ldots, x_k, y)).$$

That proves part (a). Part (b) follows from part (a), since if $f$ is representable, then $G_f$ is representable (by Theorem VII.54) and $\mu y\,(f(x_1, \ldots, x_k, y) = 0)$ is equal to $\mu y\, G_f(x_1, \ldots, x_k, y, 0)$. □

**Example VII.59.** The greatest common divisor and least common divisor are representable, since

$$\begin{aligned} lcm(x,y) &= \mu z\,(x = 0 \vee y = 0 \vee (0 < z \wedge x \mid z \wedge y \mid z)) \\ gcd(x,y) &= \max\{z \le x : z \mid x \wedge z \mid y\} \;=\; x - \mu w\,((x-w) \mid x \wedge (x-w) \mid y). \end{aligned}$$

The definition of *gcd* using the construction "$x - \mu w\,(\cdots)$" illustrates the use of bounded maximization.

**Example VII.60.** Corollary VII.51 showed that the set *Power2* of powers of 2 is representable. The unary function that gives the least power of 2 greater than $x$ is equal to

$$2^{\lfloor \log_2 x \rfloor + 1} \;=\; \mu w\,(Power2(w) \wedge x < w).$$

(The left-hand side does not make sense for $x = 0$.) The greatest power of 2 that is less than or equal to $x$ is equal to

$$2^{\lfloor \log_2 x \rfloor} \;=\; \mu w\,(Power2(w) \wedge x < 2 \cdot w).$$

We adopt this last equation as the definition of $2^{\lfloor \log_2 x \rfloor}$. This is tantamount to adopting the convention that $\lfloor \log_2 0 \rfloor = 1$.

The function $2^{\lfloor \log_2 x \rfloor}$ is defined with regular minimization and thus is representable. This will be an important function for representing functions associated with sequence coding.

## VII.7 Representability of Sequence Coding

We are still working towards proving Theorem VII.20 that all decidable relations and computable predicates are representable. This section takes a key step toward that proof by describing a method of assigning Gödel numbers to variable-length sequences of integers and how to represent functions that manipulate these sequences.

Consider a sequence of integers $\langle n_1 \ldots, n_k \rangle$. Mathematically, we can formulate this as a function from $f : \{0, \ldots, k-1\} \to \mathbb{N}$, with $f(i) = a_{i+1}$. We wish to encode the sequent with a single integer, called a Gödel number. Instead of using corner quote marks $\ulcorner \cdots \urcorner$, we will instead write the Gödel number of a sequence as $\langle\!\langle n_1, \ldots, n_k \rangle\!\rangle$.

Our goal is to define functions that manipulate Gödel numbers of sequences in natural ways, and show that these functions are representable. The functions we are interested include the binary Gödel $\beta$ function, the unary *Len* (sequence length) function, and the binary function $^\frown$ for sequence concatenation.

- $\beta(i, \langle\!\langle n_0, \ldots, n_{k-1} \rangle\!\rangle) \;:=\; n_i$.
- $Len(\langle\!\langle n_0, \ldots, n_{k-1} \rangle\!\rangle) \;:=\; k$.
- $\langle\!\langle n_1, \ldots, n_k \rangle\!\rangle^\frown \langle\!\langle m_1, \ldots, n_\ell \rangle\!\rangle \;:=\; \langle\!\langle n_1, \ldots, n_k, m_1, \ldots, m_\ell \rangle\!\rangle$ .

and the function $n \mapsto \langle\!\langle n \rangle\!\rangle$, where $\langle\!\langle n \rangle\!\rangle$ is the Gödel number of a sequence of length one.

Composition allows defining further functions such as the binary function

$$\langle\!\langle n_1, \ldots, n_k \rangle\!\rangle * n_{k+1} \coloneqq \langle\!\langle n_1, \ldots, n_k, n_{k+1} \rangle\!\rangle^\frown \langle\!\langle n_{k+1} \rangle\!\rangle = \langle\!\langle n_1, \ldots, n_k, n_{k+1} \rangle\!\rangle.$$

and the unary function

$$\mathit{Last}(\langle\!\langle n_1, \ldots, n_k \rangle\!\rangle) \coloneqq \beta(\mathit{Len}(\langle\!\langle n_1, \ldots, n_k \rangle\!\rangle) \dot{-} 1, \langle\!\langle n_1, \ldots, n_k \rangle\!\rangle) = n_k.$$

The unary relation $\mathit{IsSeq}(w)$ will be true if $w$ is the Gödel number of a sequence,

$$\mathit{IsSeq}(w) \Leftrightarrow w = \langle\!\langle n_1, \ldots, n_k \rangle\!\rangle \text{ for some } n_1, \ldots, n_k.$$

The $\beta$ function is the crucial one of the above, but it will convenient to also use *IsSeq*, *Len*, and *Last*.

There are several common ways to define the Gödel number of a sequence. Gödel's original definition used a generalization of the Chinese remainder theorem. Another common approach is to use the exponents in prime factorizations to code sequence elements; for instance, the sequence $\langle n_1, n_2, n_3, n_4 \rangle$ could have the integer with prime factorization $2^{n_1} 3^{n_2} 5^{n_3} 7^{n_4} 11$ as its Gödel number.

A third popular approach, and the one we prefer, is to use a base $w$ encoding with $w$ greater than the sequence elements. For convenience, we'll use a power of two as the base $w$. Specifically, a Gödel number for the sequence $\langle n_0, \ldots, n_{k-1} \rangle$ will be equal to

$$n_0 + n_1 \cdot 2^p + n_2 \cdot 2^{2p} + n_3 \cdot 2^{3p} + \cdots + n_{k-1} \cdot 2^{(k-1)\cdot p} + 2^{k \cdot p} + 2^{(k+1)\cdot p}. \qquad \text{(VII.8)}$$

For this, it is necessary that $n_i < 2^p$ for all $i$. Figure VII.2 illustrates this.

**Definition VII.61.** For any $p$ such that $2^p > n_i$ for all $i$, the integer (VII.8) is called a Gödel number of the sequence $\langle a_0, \ldots, a_{k-1} \rangle$. This value is not unique, as it is a Gödel number as long as $p$ is sufficiently large.

The notation $\langle\!\langle n_0, \ldots, n_{k-1} \rangle\!\rangle$ denotes the least such Gödel number of the sequence, namely the value (VII.8) where $p$ is chosen so that $\max_i\{n_i\} < 2^p \leq 2 \cdot \max_i\{n_i\}$.

**Example VII.62.** Let's compute the Gödel number for the sequence $\langle 7, 0, 11 \rangle$. The binary expansions of 7, 0, and 11 are 111, 0, and 1011. Thus, $2^p = 16$ and $p = 4$. Therefore $\langle\!\langle 7, 0, 11 \rangle\!\rangle$ has binary representation

$$1\,0001\,1011\,0000\,0111$$

In decimal form, this is $2^{16} + 2^{12} + 11 \cdot 2^8 + 0 \cdot 2^4 + 7 = 72455$.

We need to show that the $\beta$ function, the *Len* function, etc. are representable. The primary difficulty turns out to be showing that the base 2 exponential function $i \mapsto 2^i$ is representable. But, pending showing this, we can work directly with powers of 2.

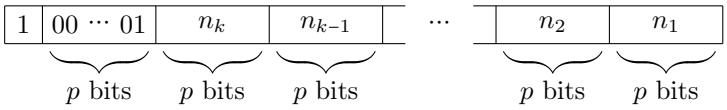



Figure VII.2: The binary representation of a Gödel number of $\langle n_1, \ldots, n_k \rangle$ has a leading 1 and then $k+1$ blocks, of $p$ bits, containing successively the binary representations of 1 and $n_k, \ldots, n_1$. The high order bits "100···01" serve to determine the value of $2^p$ with the aid of the *TwoExpP* function.

As a first step, we define a function $\mathit{TwoExpP}(u)$ which, given a Gödel number $u$ of the form (VII.8), computes the value of $2^p$ by

$$\begin{aligned} \mathit{RMSB}(u) &= u - 2^{\lfloor \log_2 u \rfloor} \\ \mathit{TwoExpP}(u) &= 2^{\lfloor \log_2 u \rfloor} / 2^{\lfloor \log_2 \mathit{RMSB}(u) \rfloor}. \end{aligned}$$

Here "*RMSB*" stands for "remove most significant bit", and $\mathit{RMSB}(u)$ is the value obtained from the binary representation of $u$ after removing its most significant bit. $\mathit{TwoExpP}(u)$ gives the power of two $2^p$ such that $u$ is equal to $(2^p + 1)2^q + a$ for some $a$ and $q$ with $a < 2^q$. "*TwoExpP*" stands for "two exponent $p$". The function *TwoExpP* is used to locate the first two 1's in the binary representation of $u$.

**Example VII.63.** Suppose $u = 81$ with binary representation 101001. Then $\mathit{RMSB}(u) = 17$ with binary representation 1001. Also, $2^{\lfloor \log_2 u \rfloor} = 64$ and $2^{\lfloor \log_2 \mathit{RMSB}(u) \rfloor} = 16$ with binary representations 100000 and 1000. From this, $\mathit{TwoExpP}(u)$ equals 4, with binary representation 100. This reflects the fact that the binary representation of $u$ starts with 101···.

The strategy for defining the $\beta$ function will be to set $\beta(i, u) = \mathit{Rem}(\lfloor u/2^{i \cdot p} \rfloor, 2^p)$, where $2^p = \mathit{TwoExpP}(u)$. This will work, since the function $u \mapsto \lfloor u/2^{i \cdot p} \rfloor$ strips away $i \cdot p$ many low order bits of $u$, and the remainder function $v \mapsto \mathit{Rem}(v, 2^p)$ extracts the $p$ many low order bits of $v$.

The difficulty is showing that $2^{i \cdot p}$ can be computed by a representable function. For this, the first step is to show that the set of powers of $2^p$ is representable. We define the relation $\mathit{PowPow2}(w, y)$ to be true if $w$ is a power of 2 and $y$ is a power of $w$. ("*PowPow2*" is short for "power of a power of two".) This is done with

$$\mathit{PowPow2}(w, y) \Leftrightarrow \mathit{Power2}(w) \wedge \mathit{Power2}(y) \wedge \exists z \leq 2y \, (\mathit{Rem}(z, w) = 1 \wedge z = y + \lfloor z/w \rfloor).$$

This definition works since $z$ can be taken to equal

$$2^{i \cdot p} + 2^{(i-1) \cdot p} + \cdots + 2^{2p} + 2^p + 1$$

when $w = 2^p$ and $y = (2^p)^i = 2^{i \cdot p}$. Conversely, it is easy to check that for $w$ and $y$ powers of 2, there is a value $z$ satisfying the conditions of $\mathit{PowPow2}(w, y)$ only

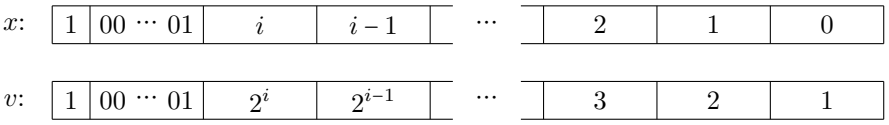



Figure VII.3: The values $x$ and $v$ needed for the formula (VII.9) representing the function $z = 2^i$. The Gödel numbers for these can use $p = i+1$ many bits per block, so the binary representations of $x$ and $v$ can have length $(i+1)^2 + i + 1$, which is $\leq 3i+3$ for all $i$. Thus $x, v$ are less than $2^{3i+3} = 8z^3$.

if $y$ is a power of $w$. This can be verified by considering the binary expansion of $z$: The condition "$\mathit{Rem}(z,w) = 1$" forces the low order $p$ bits of $z$ to be $0\cdots01$, and the operation $z \mapsto \lfloor z/w \rfloor$ performs a "shift-right" by $p$ bits.

For example, if $w = 16$ with binary representation 1000, and $y = 16^3 = 1024$ with binary representation 1 000 000 000, then $z = 16^3 + 16^2 + 16 + 1$ with binary representation 1 001 001 001. This value for $z$ shows that $y$ is a power of $w$.

As mentioned already, we shall define $\beta(i,u) = \mathit{Rem}(\lfloor u/2^{i\cdot p} \rfloor, 2^p)$. As a preliminary definition, note that the 3-ary function

$$\beta_{WY}(w,y,u) \;=\; \mathit{Rem}(\lfloor u/y \rfloor, w)$$

is representable. If $w = \mathit{TwoExpP}(y) = 2^p$ and $y = w^i = 2^{i\cdot p}$, then $\beta(i,u) = \beta_{WY}(w,y,u)$ and $\beta(i+1,u) = \beta_{WY}(w,y\cdot w,u)$.

The graph of the exponentiation function $2^i$ can be defined with the aid of the $\beta_{WY}$ function by:

$$\begin{aligned}
z = 2^i \;\Leftrightarrow\;& \\
&\exists x{\leq}8z^3\; \exists v{\leq}8z^3\; \exists w{\leq}x \\
&\quad [w = \mathit{TwoExpP}(v) \wedge \beta_{WY}(w,1,x) = 0 \wedge \beta_{WY}(w,1,v) = 1 \\
&\qquad \wedge\; \forall y{\leq}z^3\, [\mathit{PowPow2}(w,y) \wedge y\cdot w^3 \leq x \\
&\qquad\qquad \rightarrow\; \beta_{WY}(w,y\cdot w,x) = \beta_{WY}(w,y,x) + 1 \\
&\qquad\qquad\quad \wedge \beta_{WY}(w,y\cdot w,v) = 2\cdot\beta_{WY}(w,y,v)] \\
&\qquad \wedge\; \beta_{WY}(w, 2^{\lfloor x \rfloor}/w^2, x) = i \;\wedge\; \beta_{WY}(w, 2^{\lfloor x \rfloor}/w^2, v) = z\;\; ].
\end{aligned} \tag{VII.9}$$

As shown in Figure (VII.3), the intent is that $v$ encodes a sequence containing successive powers of 2, namely that $v$ is the Gödel number

$$\langle\!\langle 1, 2, 4, \ldots, 2^{i-1}, 2^i \rangle\!\rangle \;=\; 2^{(i+2)(i+1)} + 2^{(i+1)^2} + \sum_{j=0}^{i} 2^j \cdot 2^{j\cdot(i+1)}.$$

The equalities "$\beta_{WY}(w,1,v) = 1$" and "$\beta_{WY}(w,y{\cdot}w,v) = 2{\cdot}\beta_{WY}(w,y,v)$" in (VII.9) ensure that the sequence encoded by $v$ starts with 1 and its successive values increase by a factor of 2. Since largest value in the sequence is $2^i$ and has $i+1$ bits, the block size $p$ can be as small as $i+1$; this allows $w = \mathit{TwoExpP}(v)$ to equal $2^{i+1}$.

Furthermore, $x$ is intended to be a Gödel number for $\langle 0, 1, 2, \ldots, i{-}1, i\rangle$, using the same values for $w$ and $p$:

$$x \;=\; 2^{(i+2)(i+1)} + 2^{(i+1)^2} + \sum_{j=0}^{i} j \cdot 2^{j\cdot(i+1)}.$$

The equalities "$\beta_{WY}(w, 1, x) = 0$" and "$\beta_{WY}(w, y \cdot w, x) = \beta_{WY}(w, y, x) + 1$" in (VII.9) ensure that the sequence encoded by $x$ starts with 0 and its successive values increment by 1.

The conditions $\beta_{WY}(w, 2^{\lfloor x\rfloor}/w^2, x) = i$ and $\beta_{WY}(w, 2^{\lfloor x\rfloor}/w^2, v) = z$ ensure that the sequence encoded by $x$ ends with $i$ and the sequence encoded by $v$ ends with $z$. When all these conditions hold, $z$ must equal $2^i$.

Thus the graph of the exponentiation function is representable by (VII.9). Hence the exponentiation function is representable.

Since the graph of the exponentiation function $i \mapsto 2^i$ is representable, so is the graph of the base 2 logarithm function $x \mapsto \lfloor \log_2 x\rfloor$. Thus $\lfloor \log_2 x\rfloor$ is representable as a function. (Recall that $\lfloor \log_2 0\rfloor$ is equal to 1 by convention.)

We can now show that the $\beta$ function, the *Len* function and the *IsSeq* relation are representable. First, *IsSeq* is defined as

$$\mathit{IsSeq}(u) \;\Leftrightarrow\; \mathit{PowPow2}(\mathit{TwoExpP}(u), 2^{\lfloor u\rfloor})$$

which checks that $u$ has the form $2^{(k+1)} + 2^k + a$ for some $a < 2^k$. The *Len* function is defined by

$$\mathit{Len}(u) = \lfloor \log_2 u\rfloor / \lfloor \log_2 \mathit{TwoExpP}(u)\rfloor - 1.$$

and hence is representable. Finally, the Gödel $\beta$ function is defined by letting $\beta(i, u) = 0$ if $\mathit{IsSeq}(u)$ is false or $\mathit{Len}(u) \le i$, and otherwise setting

$$\beta(i, x) \;:=\; \beta_{WY}(\mathit{TwoExpP}(u), 2^{i\cdot\lfloor \log_2 \mathit{TwoExpP}(u)\rfloor}, u).$$

Hence the Gödel $\beta$ function is representable. We have shown:

**Theorem VII.64.** *The following are representable:*
(a) *The binary Gödel $\beta$ function.*
(b) *The unary sequence length function Len.*
(c) *The unary relation IsSeq.*

The other sequence functions mentioned earlier are also representable, e.g., see Exercise VII.10. Some useful ones include the unary function $\langle\!\langle n_0\rangle\!\rangle$, the binary function $\langle\!\langle n_0, n_1\rangle\!\rangle$, and the the 3-ary function $\langle\!\langle n_0, n_1, n_2\rangle\!\rangle$. For instance to see that the last one is representable, note that

$$\langle\!\langle n_0, n_1, n_2\rangle\!\rangle \;=\; 2^{4p} + 2^{3p} + n_2 2^{2p} + n_1 2^p + n_0$$

where $p = \lfloor \log_2(\max\{n_0, n_1, n_2\})\rfloor$.

| | |
|---|---|
| $\langle n_0, n_1, \ldots n_{\ell-1} \rangle$ | Informal notation for a sequence. |
| $\langle\!\langle n_0, n_1, \ldots n_{\ell-1} \rangle\!\rangle$ | Gödel number of a sequence. |
| $\beta(i, \langle\!\langle n_0, n_1, \ldots n_{\ell-1} \rangle\!\rangle) = n_i$ | Gödel $\beta$ function. |
| $Len(\langle\!\langle n_0, n_1, \ldots n_{\ell-1} \rangle\!\rangle) = \ell$ | Length of the sequence. |
| $Last(\langle\!\langle n_0, n_1, \ldots n_{\ell-1} \rangle\!\rangle) = n_{\ell-1}$ | Last entry of the sequence |
| $IsSeq(u)$ | $u$ is a Gödel number of a sequence |

Figure VII.4: The important notations for sequence coding functions.

Another useful representable function is the function $Bit(i, x)$ which gives the value of the $i$-th in the binary representation of $x$:

$$Bit(i, x) \;:=\; Rem(\lfloor x/2^i \rfloor, 2).$$

$Bit(i, x)$ is equal to the $i$-th bit of $x$, counting bits starting with the low order bit of $x$ as bit number 0. For example, $Bit(0, x) = 1$ if and only if $x$ is odd.

## VII.8 Representability of Turing Computations

This section shows that Turing machine computations can be defined by representable functions and relations. This is the final main step in proving Theorem VII.20, that every decidable relation and every computable function is representable in R.

Theorem VI.24 lets us make some simplifying assumptions about Turing machines that act on integers. Namely, w.l.og., any Turing machine uses the input alphabet $\Sigma = \{1\}$ and the tape alphabet be $\Gamma = \{\#, 1\}$, and its inputs and outputs (if any) are coded in unary notation with the string $1^n$ denoting the integer $n$.

Fix a Turing machine $M$ satisfying these conditions. Recall that a "configuration" for $M$ is a complete description of a stage in the execution of $M$.

**Definition VII.65.** A *configuration* (also called an *instantaneous description*) of $M$ consists of a specification of (a) the tape contents of $M$, (b) the current state of $M$, and (c) the head position of $M$.

Suppose $v, w \in \{\#, 1\}^*$ and that the tape contents consists of $vw$ on an otherwise blank tape. Further, suppose that the tape head is positioned on the first (leftmost) symbol of $w$ and that the current state is $q_i$. Then the triple $\langle v, i, w \rangle$ *encodes* the configuration of $M$.

We view a string in $\{\#, 1\}^*$ as being equivalent to a binary string by implicitly replacing #'s with 0's.

**Definition VII.66.** Let $w_{0/\#}$ denote the string obtained from $w$ by replacing each # with 0. Recall that $num(u)$ is the integer with binary representation $u$.

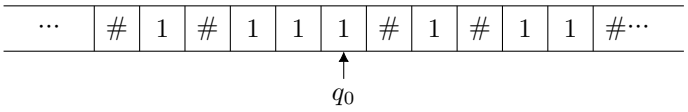



Figure VII.5: The Turing machine configuration for Example VII.67.

Suppose $C = \langle v, i, w\rangle$ encodes a configuration of $M$. Then the integer $\ulcorner C \urcorner := \langle\!\langle\, num(v_{0/\#}), i, num(w^R_{0/\#})\,\rangle\!\rangle$ is the *Gödel number* of the configuration.

Recall that $\langle\!\langle\cdots\rangle\!\rangle$ is the notation for the Gödel number of a sequence. Thus $\langle\!\langle v, i, w\rangle\!\rangle$ is the encoded triple that contains: first the integer with binary notation as specified by $v$ interpreting #'s as 0's; second the index $i$ of the current state; and third the integer with binary notation as specified by the reversal $w^{\mathrm{R}}$ or $w$, again interpreting #'s as 0's.

**Example VII.67.** Suppose the tape contains "···##1#111#1#11##···" as shown in Figure VII.5 and is otherwise all blank, with the tape head placed over the fourth 1 and the Turing machine in state $q_7$. Then a triple $\langle v, i, w\rangle$ encoding this configuration $C$ must satisfy the following.

(a) The string $v$ can be "1#11" or "#1#11" or "##1#11", etc, with any number of leading blanks.

(b) The string $w$ can be "1#1#11" or "1#1#11#" or "1#1#11##", etc, with any number of trailing blanks.

Then the binary strings $v_{0/\#}$ and $w^{\mathrm{R}}_{0/\#}$ satisfy the following.

(a′) The string $v_{0/\#}$ can be "1011" or "01011" or "001011" , etc, with any number of leading 0's. In any event, these are binary representations of the (decimal) integer 11.

(b′) The string $w^{\mathrm{R}}_{0/\#}$ can be "110101" or "0110101" or "00110101" , etc, with any number of leading 0's. These are all binary representations of 53.

Then $\ulcorner C \urcorner$ is equal to the integer $\langle\!\langle 11, 7, 53\rangle\!\rangle$.

Note that for both $v$ and $w$, the inclusion of extra #'s translates to including extra leading 0's in the binary representations $v_{0/\#}$ and $w^{\mathrm{R}}_{0/\#}$. The leading zeros of course make no difference to the integers they represent, and thus no difference in the value of $\ulcorner C \urcorner$. This corresponds to the fact that having extra leading blanks in $v$ or trailing blanks in $w$ does not change the Turing machine configuration.

Now that Gödel number of configurations have been defined, we can define the Gödel number of a complete computation of the Turing machine $M$ as the sequence of configurations for $M$ at every step in the computation. Specifically, a computation $\mathcal{C}$ that takes $\ell$ steps and uses the configurations $C_0, C_1, \ldots, C_\ell$ is encoded by the sequence

$$\langle\!\langle \ulcorner C_0 \urcorner, \ulcorner C_1 \urcorner, \ulcorner C_2 \urcorner, \ldots, \ulcorner C_\ell \urcorner \rangle\!\rangle. \tag{VII.10}$$

This is called the *Gödel number* $\ulcorner\mathcal{C}\urcorner$ of the computation $\mathcal{C}$. Recall that the notation $\langle\!\langle\cdots\rangle\!\rangle$ is used to denote an integer that encodes a variable length sequence, as handled with the Gödel $\beta$ function.

We now need to argue that there are representable functions and relations which can be used to parse the correctness of a Gödel number $\ulcorner C\urcorner$ of a computation and even to compute $\ulcorner C\urcorner$ as a function of $M$'s inputs; that is, assuming $M$ halts on all inputs.

Let's start with exhibiting a formula representing the initial configuration $C_0$. If $M$ computes a unary function or a unary relation, and its input is the integer $n$, then $C_0$ should be $\langle\epsilon, 0, 1^n\rangle$. This indicates that $M$ starts in (w.l.o.g.) state $q_0$ with the tape completely blank except for a string of $n$ 1's starting at the initial tape head position. The function $n \mapsto \ulcorner C_0\urcorner$ is clearly representable because $\ulcorner C_0\urcorner$ is equal to

$$\mathit{Init}_1(n) := \langle\!\langle 0, 0, 2^n - 1\rangle\!\rangle$$

and because the base 2 exponentiation function and the sequence triple function are representable.

If $M$ takes two inputs, $n_1, n_2$, then initial configuration $C_0$ should be $\langle\epsilon, 0, 1^{n_1}\#1^{n_2}\rangle$; hence $w^{\mathrm{R}}_{0/\#}$ equals $1^{n_2}01^{n_1}$ and $\ulcorner C_0\urcorner$ should equal $\mathit{Init}_2(n_1, n_2) = \langle\!\langle 0, 0, (2^{n_1+n_2+1}-1)-2^{n_2}\rangle\!\rangle$. This is again a representable function. Similar constructions work to define $\mathit{Init}_k$ when $M$ accepts $k$ inputs.

Now let's consider extracting the output value from the final configuration $C_\ell$. First suppose $M$ has two halting states $q_{\mathrm{acc}}$ and $q_{\mathrm{rej}}$, which w.l.o.g. have indices $n-1$ and $n-2$. So $C_\ell$ is a halting configuration if and only $\beta(1, \ulcorner C_\ell\urcorner)$ is equal to either $|Q|-1$ or $|Q|-2$ where $Q$ is the set of $M$'s states. Note that since $M$ is fixed, so is $|Q|$. In this vein, there are representable relations that check whether $x$ is equal to the Gödel number $\ulcorner C_\ell\urcorner$ of an accepting or rejecting configuration:

$$\mathit{Accept}_M(x) \;\Leftrightarrow\; \beta(1,x) = \underline{|Q|-1} \qquad \text{and} \qquad \mathit{Reject}_M(x) \;\Leftrightarrow\; \beta(1,x) = \underline{|Q|-2}.$$

Let $\mathit{Halting}_M(x)$ be defined by $\mathit{Accept}_M(x) \vee \mathit{Reject}_M(x)$. The subscript $M$ is used in the notation since these relations depend on the number $|Q|$ of states of $M$.

If $M$ computes a function and has only one halting state, then w.l.o.g. the halting state has index $|Q| - 1$, and there is a representable unary function $\mathit{Halting}_M$ that checks whether $x$ is the Gödel number $\ulcorner C_\ell\urcorner$ of a halting configuration, namely,

$$\mathit{Halting}_M(x) \;\Leftrightarrow\; \beta(1,x) = \underline{|Q|-1}.$$

The halting configuration $C_\ell$ will be a triple of the form $\langle v, \underline{|Q|-1}, 1^m 0 w\rangle$ where $m$ is the value output by the configuration and $v, w \in \{\#, 1\}^*$. When forming the Gödel number of $C_\ell$, the string $w^{\mathrm{R}}_{0/\#}$ has the form $w^{\mathrm{R}}01^m$. We can extract the value $m$ by using the representable function

$$\mathit{Output}(x) \;:=\; \mu i\,(\mathit{Bit}(i, \beta(2,x)) = 0).$$

To see that $\mathit{Output}(\ulcorner C_\ell \urcorner)$ correctly computes the integer output by the configuration $C_\ell$, note that $\beta(2, \ulcorner C_\ell \urcorner)$ is equal to the integer with binary representation $w^{\mathrm{R}}01^m$.

So far, we have handled computing the Gödel number $\ulcorner C_0 \urcorner$ of the initial configuration as a representable function; and we have also handled detecting whether a configuration $\ulcorner C_\ell$ is accepting, or what the output of $C_\ell$ is. We still need to explain how to handle defining how the $(i+1)$-st configuration $C_{i+1}$ follows from the $i$th-configuration. We shall define a representable binary relation $\mathit{Next}_M$ so that $\mathit{Next}_M(\ulcorner C \urcorner, \ulcorner C' \urcorner)$ holds if and only if the configuration $C'$ follows by a single step of $M$ from the configuration $C$.[9]

The transition function $\delta$ for $M$ controls how the computation runs. There are $|Q \setminus Q_{\mathrm{Halt}}|$ many non-halting states, and two symbols # and 1. Hence, there $2 \cdot (|Q \setminus Q_{\mathrm{Halt}}|$ many values of $\delta$ to consider, namely the values of $\delta(i, a)$ where $i$ is the index of a non-halting state and $a \in \{\#, 1\}$ is a tape symbol.

We first define, for a fixed tape symbol $a \in \{\#, 1\}$ and a state index $i$, the relation $\mathit{Next}^{i,a}_M$ so that $\mathit{Next}^{i,a}_M(\ulcorner C \urcorner, \ulcorner C' \urcorner)$ holds if and only if the configuration $C'$ follows by a single step of $M$ from $C$ using the transition rule $\delta(i, a)$. Let $\delta(i, a) = \langle a', D, i' \rangle\rangle$ where $D \in \{\mathrm{R}, \mathrm{L}\}$. If $D$ is equal to "R" (for "move right"), then $\mathit{Next}^{i,a}_M$ is defined so that

$$\begin{aligned}\mathit{Next}^{i,a}_M(\langle\!\langle m, j, n \rangle\!\rangle, \langle\!\langle m', j', n' \rangle\!\rangle) \quad \Leftrightarrow \quad & j = i \wedge \mathit{Bit}(0, n) = n_a \wedge j' = i' \\ & \wedge n' = \lfloor n/2 \rfloor \wedge m' = 2 \cdot m + n_{a'},\end{aligned}$$

where

$$n_a = \begin{cases} 0 & \text{if } a \text{ is } \# \\ 1 & \text{if } a \text{ is } 1. \end{cases} \qquad \text{and} \qquad n_{a'} = \begin{cases} 0 & \text{if } a' \text{ is } \# \\ 1 & \text{if } a' \text{ is } 1. \end{cases}$$

More formally, the relation $\mathit{Next}^{i,a}_M$ can be expressed as

$$\begin{aligned}\mathit{Next}^{i,a}_M(x, y) \quad \Leftrightarrow \quad & \beta(1, x) = i \wedge \mathit{Bit}(0, \beta(2, x)) = n_a \wedge \beta(1, y) = i' \\ & \wedge \beta(2, y) = \lfloor \beta(2, x)/2 \rfloor \wedge \beta(0, y) = 2 \cdot \beta(0, x) + n_{a'}.\end{aligned}$$

To understand the definition of $\mathit{Next}^{i,a}_M(x, y)$ where $x = \langle\!\langle m, j, n \rangle\!\rangle$ and $y = \langle\!\langle m', j', n' \rangle\!\rangle$, note that it will be true if and only if

(a) $j = i$ and $j' = i'$

(b) The lower order bit of $m = \beta(2, x)$ equal to $n_a$, as this indicates the symbol being read;

(c) $n' = \lfloor n/2 \rfloor$, so that $n' = \beta(2, y)$ is obtained from $n = \beta(2, x)$ by shifting one bit right, as this is what happens to the tape contents on the tape to the right of tape head as the tape head shifts rightward; and

(d) $m' = \beta(0, y)$ is obtained from $m = \beta(0, x)$ by shifting one bit left and adding in the newly written bit $n_{a'}$, as this is what happens when the tape head shifts leftward. So $m' = 2m + n_{a'}$.

[9] This formalizes Definition VI.14 of how a Turing machine transitions from one configuration to the next.

Note items (b) and (c) used the fact that $n$ and $n'$ were obtained by reversing the strings $w$ and $w'$ to the right of the tape head (since Gödel numbers of configurations were defined with $w^{\mathrm{R}}_{0/\#}$ instead of with $w_{0/\#}$). This means that a single step of the Turing machine updates the *low-order* bits of both $m$ and $n$ to form $m'$ and $n'$. Without this trick, we would have had to work with the high-order bits of $n$ and $n'$.

If, on the other hand, $D$ is L (for "move left"), so $\delta(i,a) = \langle a', L, i' \rangle\rangle$, we define

$$\begin{aligned} \mathit{Next}^{i,a}_M(x,y) \quad \Leftrightarrow \quad & \beta(1,x) = i \wedge \mathit{Bit}(0,\beta(2,x)) = n_a \\ & \wedge\ \beta(1,y) = i' \wedge \beta(0,y) = \lfloor \beta(0,x)/2 \rfloor \\ & \wedge\ \beta(2,y) = 4 \cdot \lfloor \beta(2,x)/2 \rfloor + 2 \cdot n_{a'} + \mathit{Bit}(0,\beta(0,x)). \end{aligned}$$

This is similar to the previous case, but modified to shift the tape head leftward.

Finally, we define $\mathit{Next}_M(x,y)$ by

$$\mathit{Next}_M(x,y) \ \Leftrightarrow\ \mathit{IsSeq}(y) \wedge \mathit{Len}(y) = 3 \wedge \bigvee_{i,a} \mathit{Next}^{i,a}_M(x,y).$$

This states that the triple $x$ is obtained from $x$ by one of the $2|Q \setminus Q_{\mathrm{Halt}}|$ many transition rules of $\delta$. The "$\mathit{IsSeq}(y) \wedge \mathit{Len}(y) = 3$" condition is included to make the choice of $y$ unique.

Now we are ready to show that all Turing decidable relations are representable and all Turing computable functions are decidable. Let $M$ be a Turing machine that always halts: $M$ either decides a relation and has two halting states $q_{\mathrm{acc}}$ and $q_{\mathrm{rej}}$ or computes a function and has a single halting state $q_{\mathrm{halt}}$. Assume that $M$ takes $k$-tuples as inputs; we again write $\vec{n}$ and $\vec{x}$ for $n_1, \ldots, n_k$ and $x_1, \ldots, x_k$. First define $\mathit{Comp}_M(\vec{n}, m)$ to mean that $m$ is the Gödel number $m = \ulcorner \mathcal{C} \urcorner$ of a complete computation of $M$ that starts with input $\vec{n}$ and reaches a halting configuration:

$$\begin{aligned} \mathit{Comp}_M(\vec{x},y) \quad \Leftrightarrow \quad & \beta(0,y) = \mathit{Init}_k(\vec{x}) \wedge \mathit{Halting}_M(\beta(\mathit{Len}(y) - 1, y) \wedge \\ & (\forall j < \mathit{Len}(y) \dot{-} 1)[\mathit{Next}_M(\beta(j,x), \beta(j+1,x))]. \end{aligned}$$

In other words, $\mathit{Comp}_M(x,y)$ holds if and only $y$ is equal to $\langle\!\langle \ulcorner C_0 \urcorner, \ldots, \ulcorner C_\ell \urcorner \rangle\!\rangle$, where (a) $C_0$ is the initial configuration for input $\vec{x}$, (b) $\ell = \mathit{Len}(w) + 1$ and the final configuration $C_\ell$ is a halting configuration, (c) Each $C_{i+1}$ is the configuration following from $C_i$ after a single step by $M$. The relation $\mathit{Comp}_M$ is a representable since $\mathit{Init}_k$, $\mathit{Halting}_M$ and $\mathit{Next}_M$ are representable.

By assumption, $M$ always halts. Therefore, for all inputs $\vec{n}$, there is a $m$ encoding a complete computation of $M$. The halting configuration can be computed as a function of the input $\vec{n}$ by the function

$$\mathit{FinalConfig}_M(\vec{x}) \ \coloneqq\ \mathit{Last}(\, \mu y\, \mathit{Comp}_M(\vec{x},y)\,) \tag{VII.11}$$

since $\mu y\, \mathit{Comp}_M(\vec{x},y)$ is a total function that gives a value $y$ that encodes a complete, halting computation of $M$. By Theorem VII.58, the functions $\mu y\, \mathit{Comp}_M(\vec{x},y)$ and hence $\mathit{FinalConfig}_M$ are representable in R.

**Theorem VII.68** (Restatement of Theorem VII.20)**.** *Every decidable relation on* $\mathbb{N}$ *is representable in the theory* R*. Every computable function on* $\mathbb{N}$ *is representable in* R*.*

*Proof.* Let $M$ be a Turing machine that always halts. If $M$ decides a relation and has two halting states $q_{\text{acc}}$ and $q_{\text{rej}}$, then the relation decided by $M$ is definable by

$$\mathit{Accept}_M(\mathit{FinalConfig}_M(\vec{x}));$$

therefore, it is a representable relation.

If $M$ computes a function and has the single halting state $q_{\text{halt}}$, then the function $f(\vec{x})$ computed by $M$ is definable by

$$y = \mathit{Output}(\mathit{FinalConfig}_M(\vec{x})). \tag{VII.12}$$

Therefore it is a representable relation. □

The proof of Theorem VII.20 also gives a normal form for partial computable functions:

**Theorem VII.69** (Kleene Normal Form)**.** *Let* $f(\vec{x})$ *be a* $k$*-ary partial computable function. Then there is a unary computable function* $U(y)$ *and a decidable* $(k+1)$*-ary relation* $T(\vec{x}, y)$ *such that, for all* $n \in \mathbb{N}$*,*

$$f(\vec{n}) \;=\; U(\,\mu m\, T(\vec{n}, m)\,).$$

*Proof.* Let $M$ be a Turing machine that partial computes $f$. Let $T(\vec{x}, y)$ be $\mathit{Comp}_M(\vec{x}, y)$. Let $U(y)$ be the composition $\mathit{Output} \circ \mathit{FinalConfig}$ of the unary functions *Output* and *FinalConfig*. Then by (VII.11) and (VII.12),

$$f(\vec{n}) \;=\; U(\,\mu y\, T(\vec{n}, y)\,).$$ □

## VII.9 The Second Incompleteness Theorem

The section proves the Second Incompleteness Theorem and some of its consequences, including Löb's Theorem. The proof uses the extensional approach as discussed in Section VII.1. This means that the theory $T$ needs to be substantially stronger than R so that it can prove simple metamathematical assertions about proofs and theorems.

For the first part of this section, let $T$ be a fixed axiomatizable, consistent theory $T$ extending R. (Later in the section, we will need more assumptions about $T$.) A sentence $B$ is said to be *independent* of $T$ if neither $B$ nor $\neg B$ is a consequence of $T$. The First Incompleteness Theorem states that $T$ is not complete. However, the proofs of the First Incompleteness Theorem in Section VII.4 were proofs by contradiction, and they did not give any example of a sentence $B$ which is independent of $T$. This section will address this by providing examples of independent sentences.

The first example will be essentially the diagonal formula $D_A$ used in the proof of the First Incompleteness Theorem based on self-reference. The second example will be a formula expressing the consistency of $T$. Thus the second example will show that, with appropriate conditions, a theory $T$ of arithmetic cannot prove the self-consistency statement "$T$ is consistent". This fact is known as the "Second Incompleteness Theorem".

Recall that $\mathit{Proof}_T$ and $\mathit{Prf}_T$ are decidable relations with $\mathit{Proof}_T(w)$ expressing that $w$ is the Gödel number of a $T$-proof, and $\mathit{Prf}_T(w, v)$ expressing that $w$ is the Gödel number of a $T$-proof of the formula with Gödel number $v$. Since these predicates are decidable, Theorem VII.20 tells us they are representable in R. We write $\textsc{Proof}_T$ and $\textsc{Prf}_T$ to denote the formulas which represent $\mathit{Proof}_T$ and $\mathit{Prf}_T$ in R. So,

$$\begin{array}{lll} \textsc{Proof}_T(x_1) & \text{represents} & \mathit{Proof}_T \\ \textsc{Prf}_T(x_1, x_2) & \text{represents} & \mathit{Prf}_T \end{array}$$

The difference in fonts, as in "$\mathit{Prf}_T$" versus "$\textsc{Prf}_T$", is just to stress that the former is a binary relation on $\mathbb{N}$ where the latter is an $L_{\mathrm{PA}}$-formula.

Since R is a true theory and $\textsc{Prf}_T$ represents $\mathit{Prf}_T$ in R, it also defines $\mathit{Prf}_T$ in $\mathcal{N}$. Namely, for all $n, m \in \mathbb{N}$,

$$\mathcal{N} \vDash \textsc{Prf}_T(n, m) \quad \Leftrightarrow \quad \langle n, m \rangle \in \mathit{Prf}_T.$$

The formula $\textsc{Thm}_T(x)$ is defined to be the formula $\exists y\, \textsc{Prf}_T(y, x)$. Since $\textsc{Prf}_T$ defines the relation $\mathit{Prf}_T$ in $\mathcal{N}$, it follows that $\textsc{Thm}_T(x)$ defines the set of theorems of $T$. Namely, for all $m \in \mathbb{N}$,

$$\mathcal{N} \vDash \textsc{Thm}_T(m) \quad \Leftrightarrow \quad m = \ulcorner A \urcorner \text{ for some } A \text{ such that } T \vDash A.$$

Let $D$ be a formula obtained from the Diagonal Theorem VII.26 such that[10]

$$\mathrm{R} \vdash D \leftrightarrow \neg \textsc{Thm}_T(\underline{\ulcorner D \urcorner}). \tag{VII.13}$$

**Theorem VII.70** (Incompleteness Theorem — explicit unprovable true formula)**.** *Let $T$ be a consistent, axiomatizable extension of* R*. Suppose $D$ satisfies* (VII.13)*. Then $D$ is true in $\mathcal{N}$ and $T \nvdash D$.*

The formula $D$ says "I am not provable in $T$". The intuition for the proof of Theorem VII.70 is as follows. First, for simplicity, let's assume that $T$ is a true theory. If $D$ is false, then $T$ does prove $D$. But this would mean $T$ proves a false formula, contradicting the assumption that $T$ is a true theory. So $D$ must be true and hence not provable in $T$.

To extend the intuition to the general case where we do not use the assumption that $T$ is true, we argue as follows. First, if $T$ proves $D$, then $T$ proves

[10] Note that this $D$ is very similar to the $D_A$ used in the last proof of Section VII.4 of the First Incompleteness Theorem, since in that proof, $A$ was presumed to represent $\overline{\mathit{Thm}_T}$. The difference is that the earlier proof was by contradiction, so the existence of $A$ was based on the (false) assumption that $\mathit{Thm}_T$ is decidable and hence representable in R. The definition of $D$ now uses just the fact that $\mathit{Thm}_T$ is *definable* in $\mathcal{N}$.

"$T$ does not prove $D$"; this is by the self-referential choice of $D$. On the other hand, if $T$ proves $D$, there is a finite, explicit $T$-proof of $D$ with some Gödel number $m$. Examining this proof shows that it is indeed a proof; this can be done in R and hence in $T$, showing that $T$ proves "$T$ proves $D$". But $T$ is consistent, so it cannot prove both "$T$ does not prove $D$" and "$T$ proves $D$". Thus $T$ cannot prove $D$.

The proof of Theorem VII.70 makes this informal argument formal:

*Proof.* From (VII.13), we have

$$T \vdash D \;\Leftrightarrow\; T \vdash \neg \textsc{Thm}_T(\ulcorner \underline{D} \urcorner).$$

On the other hand, we have

| | | |
|---|---|---|
| $T \vdash D$ | | |
| $\Leftrightarrow$ | $\mathcal{N} \vDash \textsc{Thm}_T(\ulcorner \underline{D} \urcorner)$ | By definition of $\textsc{Thm}_T$ |
| $\Leftrightarrow$ | $\mathcal{N} \vDash \textsc{Prf}_T(\underline{m}, \ulcorner \underline{D} \urcorner)$, for some $m \in \mathbb{N}$ | By definition of $\textsc{Thm}_T$ |
| $\Leftrightarrow$ | $\mathrm{R} \vdash \textsc{Prf}_T(\underline{m}, \ulcorner \underline{D} \urcorner)$, for some $m \in \mathbb{N}$ | $\textsc{Prf}_T$ represents $Prf_T$ |
| $\Leftrightarrow$ | $\mathrm{R} \vdash \textsc{Thm}_T(\ulcorner \underline{D} \urcorner)$ | By definition of $\textsc{Thm}_T$ and since R is a true theory |
| $\Rightarrow$ | $T \vdash \textsc{Thm}_T(\ulcorner \underline{D} \urcorner)$ | Since $T \supseteq \mathrm{R}$ |

We shown that if $T \vdash D$, then both $\textsc{Thm}_T(\ulcorner \underline{D} \urcorner)$ and $\neg\textsc{Thm}_T(\ulcorner \underline{D} \urcorner)$ are consequences of $T$, contradicting the consistency of $T$. Therefore $T \nvdash D$. □

Theorem VII.70 left open the possibility that $T$ proves $\neg D$. This can be ruled out if $T$ is $\omega$-consistent:

**Theorem VII.71** (Incompleteness Theorem — explicit independent formula)**.** *Let $T$ be an $\omega$-consistent, axiomatizable extension of* R*. Suppose $D$ satisfies* (VII.13)*. Then $D$ is independent of $T$.*

*Proof.* From the previous theorem, $T \nvdash D$, so we only need to show that $T \nvdash \neg D$. Suppose that $T \vdash \neg D$. By (VII.13), $T \vdash \exists x\, \textsc{Prf}_T(x, \ulcorner \underline{D} \urcorner)$.

On the other hand, by the previous theorem, there is no $T$-proof of $D$, hence $Prf_T(m, \ulcorner D \urcorner)$ is false for every $m \in \mathbb{N}$. Since $\textsc{Prf}_T$ represents $Prf_T$, we have $T \vdash \neg\textsc{Prf}_T(\underline{m}, \ulcorner \underline{D} \urcorner)$ for every $m$. This contradicts the $\omega$-consistency of $T$. □

Theorem VII.71 gives an independent sentence $D$. This works even for $T$ a very natural and strong theory such as Peano Arithmetic (PA). The problem is that $D$ is a somewhat strange, self-referential, formula. As such, one might object to the importance of $D$ as an independent sentence just because it is self-referential. The Second Incompleteness Theorem addresses this by showing giving an alternative formula $\textsc{Con}_T$ which is independent of $T$ (after making some additional, but natural, assumptions about $T$). $\textsc{Con}_T$ express the property that "$T$ is consistent" so, informally, the Second Incompleteness Theorem states that no arithmetic theory $T$ can prove its own consistency (under suitable assumptions about $T$).

**The Second Incompleteness Theorem.** The intuition for the Second Incompleteness Theorem comes from the observation that Theorem VII.70 can be restated as follows:

**Corollary VII.72.** *Let $T$ be an axiomatizable extension of* R*. Then,*

*If $T$ is consistent, then $D$ is true in $\mathcal{N}$.*

We already know that $T$ does not prove $D$. From the corollary, we can make an intuitive leap and conjecture that since $D$ is unprovable in $T$, also the consistency of $T$ is unprovable in $T$. Making this "intuitive leap" formal is exactly what is needed to prove the Second Incompleteness Theorem. As we shall see next, the notion of "$T$ is consistent" can be formalized as a sentence $\textsc{Con}_T$. Then, with some additional assumptions on $T$, we will be able to establish that $T$ proves $\textsc{Con}_T \to D$ and consequently that $T$ does not prove $\textsc{Con}_T$.

**Definition VII.73.** Let $T$ be an axiomatizable theory in the language $L_{\text{PA}}$. Then $\textsc{Con}_T$ is the sentence

$$\neg \textsc{Thm}_T(\underline{\ulcorner 0 = 1 \urcorner}),$$

where we write "0=1" as a shorthand notation for "0=S(0)" or "0=$\underline{1}$". In other words, $\underline{\ulcorner 0 = 1 \urcorner}$ means the term $S^m 0$ where $m \in \mathbb{N}$ is the Gödel number of the formula $0 = S(0)$.

Similarly, $\textsc{Con}_T$ is the sentence $\neg \exists x \, \textsc{Prf}_T(x, \underline{\ulcorner 0 = 1 \urcorner})$.

We need some extra conditions for the proof of the Second Incompleteness Theorem. The idea is that $T$ should not only contain R but should also be strong enough to prove simple properties about the formulas $\textsc{Prf}_T$ and $\textsc{Thm}_T$. These are expressed by the Hilbert-Bernays-Löb conditions:

**Definition VII.74.** Let $T \supseteq R$ be an axiomatizable theory in the language $L_{\text{PA}}$. The Hilbert-Bernays-Löb conditions for $T$ state the following hold for all $L_{\text{PA}}$-sentences $A$ and $B$:

$\text{HBL}_1$: If $T \vdash A$, then $T \vdash \textsc{Thm}_T(\underline{\ulcorner A \urcorner})$.

$\text{HBL}_2$: $T$ proves $\textsc{Thm}_T(\underline{\ulcorner A \urcorner}) \to \textsc{Thm}_T(\underline{\ulcorner \textsc{Thm}_T(\underline{\ulcorner A \urcorner}) \urcorner})$.

$\text{HBL}_3$: $T$ proves $\textsc{Thm}_T(\underline{\ulcorner A \urcorner}) \wedge \textsc{Thm}_T(\underline{\ulcorner A \to B \urcorner}) \to \textsc{Thm}_T(\underline{\ulcorner B \urcorner})$.

It turns out that the Hilbert-Bernays-Löb conditions are very natural and hold for many sufficiently powerful theories $T$ of arithmetic, including PA. For this, formulas such as $\textsc{Prf}_T$ and $\textsc{Thm}_T$ must be formulated in a sufficiently straightforward and natural way.

It is not hard to see that the condition $\text{HBL}_1$ holds for any axiomatizable theory $T$ that extends R. At least, this holds if $\textsc{Thm}_T$ is formalized as described earlier, using a formula $\textsc{Prf}_T$ that represents $Prf_T$ in R. The argument is just that if $T \vdash A$, then there is a particular proof, with some Gödel number $m$, and

then $Prf_T(m, \ulcorner A \urcorner)$ is true and hence $T$ can prove that $\textsc{Prf}_T(\underline{m}, \underline{\ulcorner A \urcorner})$ holds. From this, $T$ can prove $\textsc{Thm}_T(\underline{\ulcorner A \urcorner})$.

The condition $\text{HBL}_2$ can be viewed as a formalized version of $\text{HBL}_1$. That is, $\text{HBL}_2$ states that the implication of $\text{HBL}_1$ can be proved as a general principle in $T$. The argument is formalizes the fact that if $A$ has a $T$-proof $P$ there is a definable method to transform $P$ into a proof of $\textsc{Thm}_T(\underline{\ulcorner A \urcorner})$. The difficulty is that the proof $P$ is not known: it is just the (unknown) value of the leading existential quantifier in $\textsc{Thm}_T$. Carrying out the proof of $\text{HBL}_2$ in a theory such as PA can be detailed and lengthy; we do not present it here.

The condition $\text{HBL}_3$ is simpler. Informally, it states that $T$ can prove that if $A$ has a proof $P_1$ and $A \to B$ has a proof $P_2$, then there is a proof $P_3$ of $B$. The intuition is that the proof of $B$ is formed by concatenating the proofs $P_1$ and $P_2$, and then appending the formula $B$ as derived by a use of modus ponens. Viewed in this way, the condition $\text{HBL}_3$ is almost a triviality, as it is just a formalized version of Modus Ponens. However, it does depend on the theory $T$ being strong enough to define — and prove properties of — predicates and functions that parse and manipulate proofs.

We henceforth work with theories that satisfy the Hilbert-Bernays-Löb conditions. One such theory is the theory PA of Peano Arithmetic. PA is a true theory, but there are also non-true theories that satisfy the Hilbert-Bernays-Löb conditions, for instance the theory $\text{PA} + \neg\text{Con}_{\text{PA}}$.

**Lemma VII.75.** *Suppose $T$ is a consistent axiomatizable theory extending* R *that satisfies the Hilbert-Bernays-Löb conditions. Let $A$ be an $L_{\text{PA}}$-formula. Then*

(a) *$T$ proves* $\neg\textsc{Con}_T \to \textsc{Thm}_T(\underline{\ulcorner A \urcorner})$.

(b) *$T$ proves*

$$\textsc{Thm}_T(\underline{\ulcorner A \urcorner}) \to \textsc{Thm}_T(\underline{\ulcorner \neg A \urcorner}) \to \textsc{Thm}_T(\underline{\ulcorner \neg \textsc{Con}_T \urcorner}).$$

Lemma VII.75 shows the theory $T$ can formalize enough metamathematics to prove that $\textsc{Con}_T$ properly formalizes the statement that $T$ is consistent. Namely, part (a) states that if $T$ is inconsistent, then every formula $A$ is a consequence of $T$. And, part (b) states that if $T$ proves directly contradictory formulas, then $T$ proves that $T$ is inconsistent. Furthermore, these facts are provable by $T$.

*Proof.* We prove (a) first. Since $T \supseteq \text{R}$, $T \vdash 0 \neq 1$. Therefore, $T \vdash 0 = 1 \to A$. By $\text{HBL}_1$, $T$ proves $\textsc{Thm}_T(\underline{\ulcorner 0 = 1 \to A \urcorner})$. By $\text{HBL}_3$, $T$ proves

$$\textsc{Thm}_T(\underline{\ulcorner 0 = 1 \urcorner}) \to \textsc{Thm}_T(\underline{\ulcorner 0 = 1 \to A \urcorner}) \to \textsc{Thm}_T(\underline{\ulcorner A \urcorner}).$$

$\text{Con}_T$ is the same as $\neg\textsc{Thm}_T(\underline{\ulcorner 0 = 1 \urcorner})$. Part (a) follows tautologically.

We now prove (b). By $\text{HBL}_1$, $T$ proves $\textsc{Thm}_T(\underline{\ulcorner A \to \neg A \to 0 = 1 \urcorner})$. By two uses of $\text{HBL}_3$,

$$T \vdash \textsc{Thm}_T(\underline{\ulcorner A \urcorner}) \wedge \textsc{Thm}_T(\underline{\ulcorner A \to \neg A \to 0 = 1 \urcorner}) \to \textsc{Thm}_T(\underline{\ulcorner \neg A \to 0 = 1 \urcorner})$$

and

$$T \vdash \textsc{Thm}_T(\underline{\ulcorner \neg A \urcorner}) \wedge \textsc{Thm}_T(\underline{\ulcorner \neg A \to 0 = 1 \urcorner}) \to \textsc{Thm}_T(\underline{\ulcorner 0 = 1 \urcorner}).$$

Putting these together proves part (b). □

**Theorem VII.76** (Second Incompleteness Theorem)**.** *Suppose $T$ is a consistent axiomatizable theory extending* R *that satisfies the Hilbert-Bernays-Löb conditions. Then*

$$T \nvdash \textsc{Con}_T.$$

*Proof.* Let $D$ be the unprovable self-referential formula of (VII.13), chosen so that $R \vdash D \leftrightarrow \neg\textsc{Thm}_T(\underline{\ulcorner D \urcorner})$. Theorem VII.70 established that $T \nvdash D$. We shall prove that $T$ proves $\neg D \to \neg\textsc{Con}_T$. From this, it follows that $T$ proves $\textsc{Con}_T \to D$ and therefore $T \nvdash \textsc{Con}_T$.

Note that

$$T \vdash \neg D \to \textsc{Thm}_T(\underline{\ulcorner D \urcorner}) \tag{VII.14}$$

by choice of $D$. By $\text{HBL}_2$,

$$T \vdash \textsc{Thm}_T(\underline{\ulcorner D \urcorner}) \to \textsc{Thm}_T(\underline{\ulcorner \textsc{Thm}_T(\underline{\ulcorner D \urcorner}) \urcorner}). \tag{VII.15}$$

By choice of $D$ again, $T$ proves $\textsc{Thm}_T(\underline{\ulcorner D \urcorner}) \to \neg D$. Hence by $\text{HBL}_1$,

$$T \vdash \textsc{Thm}(\underline{\ulcorner \textsc{Thm}_T(\underline{\ulcorner D \urcorner}) \to \neg D \urcorner}). \tag{VII.16}$$

From $\text{HBL}_3$, (VII.16) gives that

$$T \vdash \textsc{Thm}(\underline{\ulcorner \textsc{Thm}_T(\underline{\ulcorner D \urcorner}) \urcorner}) \to \textsc{Thm}(\underline{\ulcorner \neg D \urcorner}). \tag{VII.17}$$

From (VII.14), (VII.15) and (VII.17), we obtain

$$T \vdash \neg D \to \textsc{Thm}_T(\underline{\ulcorner \neg D \urcorner}). \tag{VII.18}$$

From (VII.14) and (VII.18) and by Lemma VII.75(b) with $A$ equal to $D$ and $B$ equal to $0 = 1$, we obtain

$$T \vdash \neg D \to \neg\textsc{Con}_T$$

as desired. This completes the proof of the Second Incompleteness Theorem. □

We have established that $T$ cannot prove $\textsc{Con}_T$ — at least assuming that $T$ is a consistent, axiomatizable extension of R. In this case, $\textsc{Con}_T$ is a true statement that is not provable in $T$. Unlike the earlier formula $D$ asserting its own unprovability, Con is a natural statement, with an intuitive meaning that does not depend on self-reference. As such, it is a very satisfying example of the limitation of almost any useful strong theory of arithmetic.

One might still complain that $\textsc{Con}_T$ is metamathematical in nature, not quite purely mathematical in the normal sense of mathematics. There other statements that are independent of PA that are more natural in the sense that

they do not explicitly involve metamathematical concepts. These other independent statements are beyond the scope of the present text, but the interested reader can seek them out by doing an internet search for the "Paris-Harrington theorem", the "hydra theorem", the "Goodstein theorem", and related constructions.

The next obvious question is whether $\text{Con}_T$ is always independent of $T$. The answer is no. To see this, let $T$ be any axiomatizable, consistent extension of R. By the Second Incompleteness Theorem, $T \nvdash \text{Con}_T$. Therefore, if we let $T'$ be the theory $T \cup \{\neg\text{Con}_T\}$, it must be that $T'$ is consistent. Clearly, $T'$ is also axiomatizable. It is in addition reasonable to assume that $T' \vdash \text{Con}_{T'} \to \text{Con}_T$ just because any $T$-proof is a $T'$-proof.[11] Therefore $T' \vdash \neg\text{Con}_{T'}$ just because $T' \vdash \text{Con}_{T'} \to \text{Con}_T$.

On the other hand, the extra condition of $\omega$-consistency is enough to ensure that $\text{Con}_T$ is independent of $T$.

**Theorem VII.77.** *Suppose $T \supseteq \text{R}$ is $\omega$-consistent and axiomatiable and satisfies the Hilbert-Bernays-Löb conditions. Then $\text{Con}_T$ is independent of $T$.*

*Proof.* We already proved that $T \nvdash \text{Con}_T$. So suppose that $T \vdash \neg\text{Con}_T$; that is, that $T \vdash \exists y\, \text{Prf}_T(y, \underline{\ulcorner 0 = 1 \urcorner})$. Since $T$ is consistent and $T \supseteq R$, and since $\text{Prf}_T$ represents $Prf_T$, we have that $T \vdash \neg\text{Prf}_T(\underline{m}, \underline{\ulcorner 0 = 1 \urcorner})$ for every $m \in \mathbb{N}$. This contradicts the $\omega$-consistency of $T$. □

## VII.10 Löb's Theorem

Löb's Theorem is an interesting application of the Second Incompleteness Theorem; loosely speaking, it says that if the provability of $A$ implies the truth of $A$, then $A$ is true. Or more precisely, it states that if $A$ can be proved with the aid of the hypothesis that $A$ has a proof, then $A$ can proved without that hypothesis:

**Theorem VII.78** (Löb's Theorem)**.** *Let $T \supseteq \text{R}$ be consistent and axiomatizable and satisfy the Hilbert-Bernays-Löb conditions. Suppose $T$ proves $\text{Thm}_T(\underline{\ulcorner A \urcorner}) \to A$. Then $T \vdash A$.*

The formula $\text{Thm}_T(\underline{\ulcorner A \urcorner}) \to A$ states that the provability of $A$ implies the truth of $A$; this is known as the *reflection principle* for $A$ relative to $T$. The Second Incompleteness Theorem implies that, under the hypotheses of Löb's Theorem, $T \nvdash \neg A$. This is because otherwise $T \vdash \neg\text{Thm}_T(\underline{\ulcorner A \urcorner})$ and hence, by Lemma VII.75(a), $T \vdash \text{Con}_T$. Löb's Theorem gives the stronger conclusion that $T \vdash A$.

[11] Strictly speaking, we need some extra assumptions. In particular, we want $T$ to prove $\text{Thm}_{T'}(\ulcorner A \urcorner)$ is equivalent to $\text{Thm}_T(\ulcorner \text{Con}_T \to A \urcorner)$. This certainly holds for any straightforward way of formalizing metamathematics in a powerful theory such as PA. See also the comments after the statement of Löb's Theorem.

The proof of Löb's Theorem will proceed by letting $T'$ be the theory $T + \neg A$ and showing that $T'$ is inconsistent. Before giving the proof, we explain how to define the formula $\text{PRF}_{T'}(w, x)$ that represents the $Prf_{T'}$ relation. (We glossed over this point in the footnote on page 282; now we make it precise.) The idea is that $\text{PRF}_{T'}(w, \ulcorner B \urcorner)$ should hold exactly when $\text{PRF}_T(w, \ulcorner A \to B \urcorner)$. More formally, let $\text{IMP}_A(x_1, x_2)$ represent, in R, the computable function $\ulcorner B \urcorner \mapsto \ulcorner A \to B \urcorner$. Then let $\text{PRF}_{T'}(w, x)$ be the formula

$$\text{PRF}_{T'}(w, x) \;:=\; \exists z \, (\text{IMP}_A(x, z) \wedge \text{PRF}_T(w, z)).$$

Then define $\text{THM}_{T'}(x)$ to be the formula $\exists y \, \text{PRF}_{T'}(y, x)$, and define $\text{CON}_{T'}$ to be the sentence $\neg \text{THM}_{T'}(\ulcorner 0 = 1 \urcorner)$.

**Claim.** *Let $T$ be as in the statement of Löb's Theorem.*
(a) $\text{PRF}_{T'}(w, x)$ *represents* $Prf_{T'}$.
(b) *As formalized above, $T'$ satisfies the Hilbert-Bernays-Löb conditions.*
(c) $T' \nvdash \text{CON}_{T'}$.

The proof of the claim is fairly straightforward and is left for the reader as Exercise VII.19. Note that parts (a) and (b) mean that all the constructions for the First and Second Incompleteness Theorems apply to $T'$ without further modification.

*Proof of Löb's Theorem VII.78.* Let $T'$ be the theory $T + \neg A$. It will suffice to show that $T'$ is inconsistent as that implies $T \vdash A$. By the Second Incompleteness Theorem for $T'$, it suffices to prove that $T' \vdash \text{CON}_{T'}$.

By the hypothesis that $T$ proves $\text{THM}_T(\ulcorner A \urcorner) \to A$ and since $T' \vdash \neg A$, we obtain that $T'$ proves $\neg \text{THM}_T(\ulcorner A \urcorner)$. By $\text{HBL}_1$ and $\text{HBL}_3$ we obtain

$$T \;\vdash\; \text{THM}_T(\ulcorner (\neg A \to 0 = 1) \to A \urcorner)$$

and

$$T \;\vdash\; \text{THM}_T(\ulcorner (\neg A \to 0 = 1) \urcorner) \wedge \text{THM}_T(\ulcorner (\neg A \to 0 = 1) \to A \urcorner) \to \text{THM}_T(\ulcorner A \urcorner).$$

Thus since $T' \supseteq T$, the theory $T'$ proves $\neg \text{THM}_T(\ulcorner \neg A \to 0 = 1 \urcorner)$. In other words, $T' \vdash \neg \text{THM}_{T'}(\ulcorner 0 = 1 \urcorner)$, namely, $T'$ proves $\text{CON}_{T'}$. From this, $T'$ is inconsistent and therefore $T \vdash A$. □

# Exercises

**Exercise VII.1.** Prove that Q does not prove $\forall x \, (x \neq Sx)$. [Hint: Do this by constructing a model of $\text{Q} \cup \{\exists x \, (x = Sx)\}$. The universe of the model can be $\mathbb{N} \cup \{\infty\}$ where $\infty$ denotes an additional element. You will need to define the model so that $S\infty$ to equal $\infty$, and also define how addition and multiplication act when one or both arguments are equal $\infty$.]

**Exercise VII.2.** Prove that Q does not prove $\forall x\,(0 + x = x)$ or $\forall x\,(0 \cdot x = 0)$. [Hint: Find a model Q with universe $\mathbb{N} \cup \{\infty_1, \infty_2\}$ in which both $\forall x\,(0 + x = x)$ and $\forall x\,(0 \cdot x = 0)$ are false.] Conclude that Q also does not prove $\forall x\,\forall y\,(x+y = y+x)$ or $\forall x\,\forall y\,(x \cdot y = y \cdot x)$.

**Exercise VII.3.**
(a) Prove that PA proves $\forall x\,(x \neq Sx)$. Compare to Exercise VII.1.
(b) Prove that PA proves $\forall x\,(x \neq SSx)$.

**Exercise VII.4.** Prove that PA proves $\forall x\,\forall y\,(x + y = y + x)$. This will require three steps, each using induction:
(a) Prove that PA proves $0 + x = x + 0$. (You can use Example VII.5.)
(b) Prove that PA proves $Sx + y = S(x + y)$. (Use induction on $y$.)
(c) Prove that PA proves $x + y = y + x$. (Use induction.)

**Exercise VII.5.** Let PA $\setminus \{Q_3\}$ be the theory with the same axioms as PA except omitting the axioms $Q_3$. Prove that PA $\setminus \{Q_3\}$ logically implies $Q_3$. Therefore, $Q_3$ may be omitted from the list of axioms for PA, without changing the theory. [Hint: Use induction on the formula $x \neq 0 \to \exists y\,(Sy = x)$.]

**Exercise VII.6.** Consider replacing the definition of $E_A$ in Equation VII.2 with

$$E_A(x_1) \;:=\; \forall y\,[A_{SelfSub}(x_1, y) \to A(y)].$$

Revise the second proof of Theorem VII.24 so that it works with this definition of $E_A$.

**Exercise VII.7.** Prove the following claims from Example VII.53.
(a) $A_P(x_1, y)$ represents the predecessor function $P$ in R.
(b) $\mathrm{R} \nvdash \forall y\,(Sy \neq 0)$. Therefore, $A_{G_P}$ does not represent the function $P$ in R.

**Exercise VII.8.** Give a direct proof that a relation $S$ is representable in R if and only if its characteristic function $\chi_S$ is representable in R. By a "direct" proof, we mean a proof that uses the definitions of R and representability, and does not depend on Theorems V.17 and VII.20.

**Exercise VII.9.** Regular minimization was used to define the function $2^{\lfloor \log_2 x \rfloor}$, but was not elsewhere used for the definition of the Gödel $\beta$ function. Show how to eliminate the use of regular minimization in the definitions used for the Gödel $\beta$ function by reworking the definition to use bounded quantification in place of regular minimization.[12]

**Exercise VII.10.** Prove that the sequence concatenation function $^\frown$

$$\langle\!\langle n_1, \ldots, n_k \rangle\!\rangle^\frown \langle\!\langle m_1, \ldots, n_\ell \rangle\!\rangle \;:=\; \langle\!\langle n_1, \ldots, n_k, m_1, \ldots, m_\ell \rangle\!\rangle.$$

is representable. Give a proof using the techniques of Sections VII.6 and VII.7 including Gödel sequence coding, but not using Theorem VII.20.

[12] This result is usually stated as "exponentiation is $\Delta_0$-definable", see Gaifman-Dimitracopoulus [1980]. The gist is that addition, multiplication, Boolean operators and bounded quantification suffice to form a formula that represents the graph of the exponentiation function in R, and hence defines the graph of the exponentiation function in $\mathcal{N}$.

**Exercise VII.11.** If $g$ is a $k$-ary function and $h$ is a $(k+2)$-ary function, then a $(k+1)$-ary function $f$ can be defined by *primitive recursion* from $g$ and $h$ as

$$\begin{aligned} f(\vec{n}, 0) &= g(\vec{n}) \\ f(\vec{n}, m+1) &= h(\vec{n}, m, f(\vec{n}, m)). \end{aligned}$$

Suppose $g$ and $h$ are representable. Give a direct proof that $f$ is representable. By a "direct" proof is meant a proof using Gödel sequence coding, and not using Theorem VII.20.

**Definition VII.79.** Let $T \supseteq \mathrm{R}$ be consistent, and $S$ be a $k$-ary relation. Define that $T$ *semirepresents* $S$ provided that there is a formula $A_S(x_1, \ldots, x_k)$ such that for all $n_1, \ldots, n_k \in \mathbb{N}$,

(a) If $S(n_1, \ldots, n_k)$ is true, then $T \vdash A_S(\underline{n_1}, \ldots, \underline{n_k})$; and
(b) If $S(n_1, \ldots, n_k)$ is false, then $T \nvdash A_S(\underline{n_1}, \ldots, \underline{n_k})$.

If other words, $S(n_1, \ldots, n_k)$ is true if and only if $T \vdash A_S(\underline{n_1}, \ldots, \underline{n_k})$. Prove that $S$ is semirepresented by $T$ if and only if $S$ is semidecidable.

**Exercise VII.12.** Let $T \supseteq R$ be consistent and axiomatizable. Prove that the theory $T = \{A : A \in T\}$ and the set $\{A : \neg A \in T\}$ are computably inseparable. [Hint: Use a proof by contradiction and a self-referential formula.]

**Exercise VII.13.** Let $A$ be an arbitrary $L_{\mathrm{PA}}$-sentence. Prove that PA does not prove $\neg \mathrm{THM}_{\mathrm{PA}}(\underline{\ulcorner A \urcorner})$.

**Exercise VII.14.** Let $E$ be a self-referential formula such that R proves $E \leftrightarrow \mathrm{THM}_{\mathrm{PA}}(\underline{\ulcorner \neg E \urcorner})$, so that $E$ states "I am refutable by PA." Is $E$ true or false? Justify your answer. (For this and the next exercises, "true" or "false" means true or false in the standard model of the integers.)

**Exercise VII.15.** Let $F$ be a self-referential formula such that R proves $F \leftrightarrow \mathrm{THM}_{\mathrm{PA}}(\underline{\ulcorner F \urcorner})$, so that $F$ states that "I am PA provable". Is $F$ true or false? Justify your answer. A sentence $F$ with this property is called a "Henkin sentence". [Hint: Use Löb's Theorem.]

**Exercise VII.16.** Let $G$ be a self-referential formula such that R proves $G \leftrightarrow \neg \mathrm{THM}_{\mathrm{PA}}(\underline{\ulcorner \neg G \urcorner})$, so that $G$ states that "PA cannot refute my negation". Is $G$ true or false? Justify your answer. [Hint: Use Löb's Theorem, or the previous exercise.]

**Exercise VII.17.** Let $H$ be a self-referential formula such that R proves

$$H \leftrightarrow \neg \mathrm{THM}_{\mathrm{PA}}(\underline{\ulcorner H \urcorner}) \wedge \neg \mathrm{THM}_{\mathrm{PA}}(\underline{\ulcorner \neg H \urcorner}),$$

so that $H$ states that "I am independent of PA". Is $H$ true or false? Justify your answer.

**Exercise VII.18.** Let $J$ be a self-referential formula such that R proves

$$J \leftrightarrow \mathrm{THM}_{\mathrm{PA}}(\underline{\ulcorner J \urcorner}) \vee \mathrm{THM}_{\mathrm{PA}}(\underline{\ulcorner \neg J \urcorner}),$$

so that $J$ states that "I am not independent of PA". Is $J$ true or false? Justify your answer. [Hint: Use Löb's Theorem.]

**Exercise VII.19.** Prove the claim stated on page VII.10 before the proof of Löb's Theorem VII.78.

**Exercise VII.20.** Suppose $T \supseteq R$ is consistent, axiomatizable and satisfies the Hilbert-Bernays-Löb conditions. Let $D$ be a self-referential formula such that $\mathrm{R} \vdash D \leftrightarrow \neg \mathrm{THM}_T(\ulcorner D \urcorner)$.

(a) Prove $T \vdash D \leftrightarrow \mathrm{CON}_T$.
(b) Suppose that $E$ is another sentence such that $\mathrm{R} \vdash E \leftrightarrow \neg \mathrm{THM}_T(\ulcorner E \urcorner)$. Prove that $T \vdash D \leftrightarrow E$.
(c) Show that there is a theory $T$ such that $T \vdash \neg D$. (This shows that the assumption of $\omega$-consistency in Theorem VII.71 cannot be weakened to the assumption of consistency.)

**Exercise VII.21.** Let $T \supseteq \mathrm{R}$ be a consistent, axiomatizable theory. This exercise asks you to carry Rosser's construction of an independent sentence for $T$. A *Rosser proof* of a formula $A$ is defined to be a proof $P$ of $A$ such that there is no proof $P'$ of $\neg A$ with $\ulcorner P' \urcorner < \ulcorner P \urcorner$. More formally, let $\mathrm{NEG}(x_1, x_2)$ represent the (computable) mapping $\ulcorner A \urcorner \mapsto \ulcorner \neg A \urcorner$ in R, and define $\mathrm{RPRF}_T(w, x)$ to be the formula

$$\mathrm{RPRF}_T(w, x) \;:=\; \mathrm{PRF}_T(w, x) \wedge \forall v \, \forall y \, (v < w \wedge \mathrm{NEG}(x, y) \rightarrow \neg \mathrm{PRF}_T(v, y)).$$

Let $\mathrm{RTHM}_T(x)$ be the formula $\neg \exists w \, \mathrm{PRF}_T(w, x)$. Finally, let $D_R$ be a self-referential formula such that R proves $D_R \leftrightarrow \neg \mathrm{RTHM}_T(\ulcorner D_R \urcorner)$. Also, let $A$ be an arbitrary formula. Prove:

(a) There is a Rosser $T$-proof of $A$ if and only if there is a $T$-proof of $A$.
(b) If $T \vdash A$, then $T \vdash \mathrm{RTHM}_T(\ulcorner A \urcorner)$.
    [Hint: You will need to use $T \supseteq R$ and Axiom $\mathrm{R}'_{\leq}$.]
(c) If $T \vdash \neg A$, then $T \vdash \neg \mathrm{RTHM}_T(\ulcorner A \urcorner)$.
(d) Prove that $T \nvdash D_R$.
(e) Prove that $T \nvdash \neg D_R$.
(f) Conclude that $D_R$ is independent of $T$.

# Bibliography


[1] G. S. Boolos, J. P. Burgess, and R. C. Jeffrey, *Computability and Logic*, Cambridge University Press, second ed., 2007.

[2] S. R. Buss, *Bounded Arithmetic*, Bibliopolis, Naples, Italy, 1986. Revision of 1985 Princeton University Ph.D. thesis.

[3] C. C. Chang and H. J. Keisler, *Model Theory*, Dover Publications, 2012. 3rd Edition.

[4] S. A. Cook, *Feasibly constructive proofs and the propositional calculus*, in Proceedings of the Seventh Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, 1975, pp. 83–97.

[5] H. Enderton, *A Mathematical Introduction to Logic*, Academic Press, second ed., 2001.

[6] G. Frege, *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*, Halle, 1879. English translation by Stefan Bauer-MengelBerg, with an introduction by van Heijenoort, in [21], pages 1-82.

[7] H. Gaifman and C. Dimitracopoulos, *Fragments of Peano's arithmetic and the MRDP theorem*, in Logic and Algorithmic: An International Symposium held in honour of Ernst Specker, Monographie #30 de L'Enseignement Mathématique, 1980, pp. 187–206.

[8] D. Hilbert and W. Ackermann, *Grundzüge der theoretischen Logic*, Springer Verlag, first ed., 1928.

[9] P. G. Hinman, *Fundamentals of Mathematical Logic*, AK Peters/CRC Press, 2008.

[10] R. E. Hodel, *An Introduction to Mathematical Logic*, Dover, 1995. Reprinting with corrections of first edition, PWS Publishing, 1995.

[11] Y. I. Manin, *A Course in Mathematical Logic for Mathematicians*, Springer, 2010. Second edition.

[12] D. Marker, *Model Theory: An Introduction*, Springer, 2002.

[13] E. Mendelson, *Introduction to Mathematical Logic*, Chapman and Hall/CRC, sixth ed., 2015.

[14] J. D. Monk, *Mathematical Logic*, Springer, 1976.

[15] E. Nelson, *Predicative Arithmetic*, Princeton University Press, 1986.

[16] P. Pudlák, *Cuts, consistency statements and interpretation*, Journal of Symbolic Logic, 50 (1985), pp. 423–441.

[17] J. Robinson, *Definability and decision problems in arithmetic*, Journal of Symbolic Logic, 44 (1949), pp. 98–114.

[18] J. R. Shoenfield, *Mathematical Logic*, Addison-Wesley, 1967.

[19] A. Tarski, *A Decision Method for Elementary Algebra and Geometry*, RAND Corporation, 1948.

[20] A. Turing, *On computable numbers, with an application to the entscheidungsproblem*, Proceedings of the London Mathematical Society, Series 2, 42 (1936), p. 230–265.

[21] J. van Heijenoort, ed., *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*, Harvard University Press, 1967.

[22] A. N. Whitehead and B. Russell, *Principia Mathematica*, vol. 1, Cambridge University Press, 1910.

# Index