

Mathematical Logic

Emil Jeřábek

Based on course notes taken by Jindřich Novák

Charles University
Faculty of Mathematics and Physics

Prague
24 October 2025

Course overview

Mathematical logic, in a broad sense, is the investigation of formal logical systems—that typically have a syntactic component operating with expressions such as formulas, and a semantic component that assigns a “meaning” to these expressions interpreted in suitable structures—by mathematical means and methods, similar to, say, abstract algebra (whereas logic had been traditionally a domain of philosophy since antiquity). In a narrower sense, mathematical logic studies formal systems relevant to the foundations of mathematics, such as first-order logic and set theory. It also includes spin-off fields such as the theory of computation.

The purpose of this course is to introduce three (related) basic topics in mathematical logic, each part culminating with one of the major achievements of the field:

1. Classical propositional and first-order logic, leading up to the completeness theorem.
2. Computability, leading up to the undecidability of the halting problem.
3. Theories of arithmetic, leading up to Gödel’s theorems.

These lecture notes also include a list of exercises (purely voluntary): besides helping to get a better acquaintance of the material, many of them present additional side results that may interest dedicated students, but for which there is no room in the main lecture due to time constraints.

A website for this course is maintained at

<https://users.math.cas.cz/~jerabek/teaching/mathlog/mathlog.html>.

It includes basic information about the course, and an updated detailed break-down of topics we covered in each lecture.

Concerning literature, Part 1 seeks to be consistent with

- Lou van den Dries, *Lecture notes on mathematical logic*,

and Part 2 with

- Michael Sipser, *Introduction to the theory of computation*, 2nd ed., Thomson, 2006.

Part 3 does not follow any particular source. Other recommended literature:

- Vítězslav Švejdar, *Logika: neúplnost, složitost a nutnost*, Academia, Praha, 2002 (in Czech).
- René Cori and Daniel Lascar, *Mathematical logic: A course with exercises (Part I and Part II)*, Oxford University Press, 2000.
- Joseph R. Shoenfield, *Mathematical logic*, Addison-Wesley, London, 1967.

Contents

Course overview	iii
1 Syntax and semantics of logic	1
1.1 Propositional logic	1
1.2 Completeness of propositional logic	5
1.3 First-order logic	9
1.4 First-order proof system	13
1.5 Completeness of first-order logic	16
1.6 Consequences of the completeness theorem	19
2 Computability	23
2.1 Turing machines	23
2.2 Universal Turing machines and the halting problem	29
2.3 Computability of logical syntax	32
3 Arithmetic	35
3.1 Robinson and Peano arithmetics	35
3.2 Σ_1 -completeness of $\mathbf{Q}$	37
3.3 Sequence encoding and definability of computation	39
3.4 Undecidability and incompleteness	43
3.5 Unprovability of consistency	45
Exercises (in 2024/25)	49

Part 1

Syntax and semantics of logic

The goal of this part is to prove the completeness theorem for first-order logic (and for propositional logic). We start from the beginning, that is, we introduce the syntax and semantics of propositional and first-order logic and their basic properties. However, since these basics are included among the prerequisites for the course, our treatment of them will be in the form of an extensive *review* of the material to make sure we are all on the same page, and to fix notation and terminology. The presentation will be, therefore, rather terse at times, and we will skip some proofs.

Convention 1.1. Throughout the course, $\mathbb{N}$ denotes the set of natural numbers *including* 0. Ordered pairs and other tuples are denoted using the angle brackets $\langle x, y \rangle$.

Definition 1.2 (Strings). Given an alphabet Σ , the *strings* of a given length n over Σ are elements of Σ^n . The set of all strings over Σ is

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n.$$

Notation 1.3. The length of $w \in \Sigma^*$ is denoted $|w|$; i.e., $|w|$ is the $n \in \mathbb{N}$ such that $w \in \Sigma^n$.

Given $u, v \in \Sigma^*$, their *concatenation* (of length $|u| + |v|$) is denoted $u \smallfrown v$, or simply uv .

The *empty string* (the unique string of length 0) is denoted ε .

Convention 1.4. Even though there is a formal distinction between a symbol $a \in \Sigma$ and the corresponding string of length one from Σ^1 , both will be denoted the same.

1.1 Propositional logic

When formulating propositional logic, there is a somewhat arbitrary choice of which connectives are postulated as basic, and which are introduced as abbreviations. In accordance with van den Dries's lecture notes, we formulate propositional logic in the *De Morgan language* using the connectives $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation), $\top$ (truth, verum), and $\perp$ (falsity, falsum):

Definition 1.5 (Atoms, propositional formulas). Let A be a set of *atoms* (or *propositional variables*). The set Prop_A of *propositional formulas* over A is the smallest subset of Σ^* , where $\Sigma = A \cup \{\wedge, \vee, \neg, \top, \perp, (,)\}$, such that

- (i) $a \in A \implies a \in \text{Prop}_A$,
- (ii) $\varphi, \psi \in \text{Prop}_A \implies (\varphi \wedge \psi), (\varphi \vee \psi), \neg\varphi, \top, \perp \in \text{Prop}_A$.

Remark 1.6. It may not be immediately clear that there exists an object satisfying an inductive definition such as the definition of Prop_A above. (It is clear that if it exists, it is unique, as there can be only one *smallest* set—meaning w.r.t. inclusion—with a given property.) This can be formally proved in several ways:

- (i) Let $\mathcal{F}$ be the collection of all subsets of Σ^* with the given property, and put $\text{Prop}_A = \bigcap \mathcal{F}$. Using the fact that the property $X \in \mathcal{F}$ is defined by a collection of inductive clauses of the form “if $\varphi_0, \varphi_1, \dots \in X$ then $F(\varphi_0, \varphi_1, \dots) \in X$ ”, it is readily seen that $\text{Prop}_A \in \mathcal{F}$, hence it is the smallest set in $\mathcal{F}$.
- (ii) Define a sequence of sets $X_n \subseteq \Sigma^*$, $n \in \mathbb{N}$, such that $X_0 = \emptyset$, and X_{n+1} is the result of all the inductive clauses applied to X_n ; i.e., here,

$$X_{n+1} = A^1 \cup \{(\varphi \wedge \psi), (\varphi \vee \psi), \neg\varphi, \top, \perp : \varphi, \psi \in X_n\}.$$

We can show $X_n \subseteq X_{n+1}$ by induction on n . Put $\text{Prop}_A = \bigcup_{n \in \mathbb{N}} X_n$. Since all the inductive clauses are finitary, we see that Prop_A is closed under them, and it is the smallest set with this property.

We will see other such inductive definitions later; they can all be formalized in a similar manner.

Notation 1.7. We introduce the shorthands $(\varphi \rightarrow \psi) = (\neg\varphi \vee \psi)$, $(\varphi \leftrightarrow \psi) = ((\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi))$.

Convention 1.8. When writing formulas, we will omit outermost brackets, and, in contexts where it does not matter, brackets occurring in repeated conjunctions or disjunctions (e.g., $\varphi \wedge \psi \wedge \omega$).

It is also a common convention that $\wedge$ and $\vee$ bind more strongly than $\rightarrow$ and $\leftrightarrow$, thus, e.g., the expression $\varphi \wedge \psi \rightarrow \chi \vee \omega$ is understood as the formula $((\varphi \wedge \psi) \rightarrow (\chi \vee \omega))$, i.e., $(\neg(\varphi \wedge \psi) \vee (\chi \vee \omega))$.

Remark 1.9. We defined formulas using *infix notation*, in which binary connectives are written in between their two arguments. Logic is sometimes (e.g., in the van den Dries lecture notes) presented using *prefix* (or *Polish*) notation, in which all connectives are placed in front of all their arguments. This leads to a formally simpler development, as all connectives are treated in a uniform way, and there is no need to use *brackets*: e.g., the formula $((p \wedge \neg q) \rightarrow (\neg p \vee r))$ is written $\rightarrow \wedge p \neg q \vee \neg p r$. (In fact, in actual Polish notation as introduced by Łukasiewicz, connectives are denoted by certain capital letters: this formula would be written $CKpNqANpr$.) We opted to stick to the more human-readable infix notation.

Notation 1.10. Given a finite sequence of formulas $\varphi_0, \dots, \varphi_{n-1}$, their repeated conjunction (bracketed in some canonical way, e.g., left-to-right) is denoted

$$\bigwedge_{i < n} \varphi_i = \varphi_0 \wedge \dots \wedge \varphi_{n-1}.$$

Analogous notation is introduced for logical disjunction. For $n = 0$, this is understood as $\bigwedge_{i < 0} \varphi_i = \top$, $\bigvee_{i < 0} \varphi_i = \perp$; for $n = 1$, $\bigwedge_{i < 1} \varphi_i = \bigvee_{i < 1} \varphi_i = \varphi_0$.

Lemma 1.11. (*Unique readability*) For any formula $\varphi \in \text{Prop}_A$, exactly one of the following cases happens:

- (i) $\varphi \in A$ (φ is atomic).
- (ii) $\varphi = (\varphi_0 \wedge \varphi_1)$ for some $\varphi_0, \varphi_1 \in \text{Prop}_A$.
- (iii) $\varphi = (\varphi_0 \vee \varphi_1)$ for some $\varphi_0, \varphi_1 \in \text{Prop}_A$.
- (iv) $\varphi = \neg\varphi_0$ for some $\varphi_0 \in \text{Prop}_A$.
- (v) $\varphi = \top$.
- (vi) $\varphi = \perp$.

Moreover, the formulas φ_0 and φ_1 in (ii)–(iv) are uniquely determined by φ .

Remark 1.12. The syntactic build-up of a formula may be described by a so-called *syntactic tree*: this is an ordered binary tree each of whose nodes is labelled with an occurrence of a symbol in the formula, such that atoms (and constants) are leaf nodes, and a node labelled with a connective has as its the children the arguments of the connective. Lemma 1.11 can be generalized to show that every formula has a unique syntactic tree.

Definition 1.13. A *propositional assignment*, or *truth assignment*, or simply an *assignment*, is a function $\alpha: A \rightarrow \{0, 1\}$. The set of all assignments on A is denoted $\{0, 1\}^A$.

Lemma 1.14 (Formula evaluation). *Any truth assignment α has a unique extension $\hat{\alpha}: \text{Prop}_A \rightarrow \{0, 1\}$ such that for all $\varphi, \psi \in \text{Prop}_A$ and $a \in A$,*

$$\begin{aligned}\hat{\alpha}(a) &= \alpha(a), \\ \hat{\alpha}(\varphi \wedge \psi) &= \min\{\hat{\alpha}(\varphi), \hat{\alpha}(\psi)\}, \\ \hat{\alpha}(\varphi \vee \psi) &= \max\{\hat{\alpha}(\varphi), \hat{\alpha}(\psi)\}, \\ \hat{\alpha}(\neg\varphi) &= 1 - \hat{\alpha}(\varphi), \\ \hat{\alpha}(\top) &= 1, \\ \hat{\alpha}(\perp) &= 0.\end{aligned}$$

Observation 1.15. *For any sequence of formulas $\varphi_0, \dots, \varphi_{n-1}$, we have*

$$\hat{\alpha}\left(\bigvee_{i < n} \varphi_i\right) = 1 \iff \exists i < n \hat{\alpha}(\varphi_i) = 1$$

and

$$\hat{\alpha}\left(\bigwedge_{i < n} \varphi_i\right) = 1 \iff \forall i < n \hat{\alpha}(\varphi_i) = 1. \quad \square$$

Definition 1.16. If $\hat{\alpha}(\varphi) = 1$, we say that α *satisfies* φ , or that α *is a satisfying assignment of* φ ; this is denoted $\alpha \models \varphi$.

A formula φ is a *tautology*, written as $\models \varphi$, if every truth assignment $\alpha: A \rightarrow \{0, 1\}$ satisfies φ .

Dually, φ is *satisfiable* if there exists a truth assignment $\alpha: A \rightarrow \{0, 1\}$ that satisfies φ .

Formulas φ and ψ are *equivalent*, written $\varphi \equiv \psi$, if

$$\forall \alpha \in \{0, 1\}^A \hat{\alpha}(\varphi) = \hat{\alpha}(\psi).$$

Observation 1.17. $\varphi \equiv \psi$ if and only if $\models (\varphi \leftrightarrow \psi)$. $\square$

Definition 1.18 (Entailment). Let $\Gamma \subseteq \text{Prop}_A$ be a set of propositional formulas. We say that a truth assignment α *satisfies* Γ (or that α is a *model of* Γ), written $\alpha \models \Gamma$, if α satisfies every formula $\varphi \in \Gamma$.

Γ *entails* φ , written as $\Gamma \models \varphi$, if for each $\alpha \in \{0, 1\}^A$, whenever $\alpha \models \Gamma$, then $\alpha \models \varphi$.

Definition 1.19 (Boolean function). A *Boolean function* is any function of the form $f: \{0, 1\}^n \rightarrow \{0, 1\}$. We can identify it with $f: \{0, 1\}^A \rightarrow \{0, 1\}$ if $|A| = n$; i.e., $A = \{a_0, \dots, a_{n-1}\}$. A formula $\varphi \in \text{Prop}$ *represents* a Boolean function f if for each $\alpha \in \{0, 1\}^A$ we have $f(\alpha) = \hat{\alpha}(\varphi)$.

Observation 1.20. *Every formula φ represents a unique Boolean function, namely the truth-table function $\text{tt}_\varphi: \{0, 1\}^A \rightarrow \{0, 1\}$ defined by $\text{tt}_\varphi(\alpha) = \hat{\alpha}(\varphi)$.* $\square$

Lemma 1.21. *If A is a finite set of atoms, every Boolean function $f: \{0, 1\}^A \rightarrow \{0, 1\}$ can be represented by a formula.*

Proof. Check that

$$f(\vec{p}) \equiv \bigvee_{\alpha \in f^{-1}(1)} \bigwedge_{i \in A} p_i^{\alpha(i)} \equiv \bigwedge_{\alpha \in f^{-1}(0)} \bigvee_{i \in A} p_i^{1-\alpha(i)},$$

where we write $p^1 = p$, $p^0 = \neg p$. $\square$

This is often expressed by saying that the set of connectives $\{\wedge, \vee, \neg, \top, \perp\}$ is *functionally complete* on $\{0, 1\}$. See Exercises 1–3.

Definition 1.22. A *literal* is an atom or its negation.

A *clause* is a disjunction of a (possibly empty) set of literals.

A *formula in conjunctive normal form*, or a *CNF*, is a conjunction of a (possibly empty) set of clauses.

Dually, a *formula in disjunctive normal form*, or a *DNF*, is disjunction of a set of conjunctions of sets of literals.

Conjunctions of sets of literals are also called *terms*, but we will refrain from this terminology to avoid clash with Definition 1.47 below.

The proof of Lemma 1.21 actually shows:

Corollary 1.23. Every Boolean function can be represented by a CNF and by a DNF. □

Corollary 1.24. Every formula is equivalent to a CNF and to a DNF. □

Remark 1.25. It follows from the proof of Lemma 1.21 that any Boolean function in n variables can be represented by a formula of size $O(2^n n)$ (where the size of a formula is its length as a string). We can improve this to $O(2^n)$ by an inductive construction (see Exercise 7). One may ask if we could do better. The answer is *no*, not by much.

In fact, there are Boolean functions $f: \{0, 1\}^n \rightarrow \{0, 1\}$ such that any formula representing f has size $\Omega(2^n / \log n)$. (Conversely, every Boolean function has a formula of size $O(2^n / \log n)$, but this small improvement takes a lot of work to prove.)

This may be proved by a simple counting argument: a formula φ of size s is a string of length s made of $n + 7$ possible symbols, thus the number of such formulas is $\leq (n + 7)^s$, while there are 2^{2^n} Boolean functions. If all functions can be represented by formulas of size s , then $2^{2^n} \leq (n + 7)^s$, which implies $2^n \leq s \cdot \log(n + 7)$, whence $s \geq 2^n / \log(n + 7)$. The same argument actually shows that *vast majority* of Boolean functions require formulas of size $\Omega(2^n / \log n)$.

Despite that, it is an open problem to construct an explicit sequence of Boolean functions that require formulas of size more than $\Omega(n^c)$ for all constants c . We can construct functions that require formulas of cubic size, but we cannot do any better.

This falls into the field of study known as *circuit complexity*, which is related to various problems in computational complexity.

Propositional logic may seem trivial at first sight, but it is related to many very difficult and very intensively studied areas of mathematics.

Another open problem is the question: *given a formula, how difficult is it to compute whether it is satisfiable?* One obvious way to go about this is to brute-force the solution by trying all possible assignments. This is an inefficient algorithm, however, with computational complexity on the order of 2^n .

Another possibility is to convert the given formula to a DNF and check the satisfiability thereof. Note that it is trivial to check the satisfiability of DNFs, which can be done in polynomial time. The conversion itself, however, requires exponential time to compute (cf. Exercise 6).

Observe that we can easily verify that φ is satisfiable if we are given a satisfying assignment as a witness. Problems like this, where a positive answer has an efficiently checkable witness, are said to belong to the complexity class NP. In fact, satisfiability is a “complete” problem for NP in a suitable sense. The famous question $P \stackrel{?}{=} NP$ in effect asks whether there exists an efficient (i.e., polynomial-time) algorithm for satisfiability testing. It is generally conjectured that this is not the case, and in fact, that every satisfiability-testing algorithm requires time $2^{\Omega(n)}$.

When checking validity or satisfiability of formulas by hand, or converting them to CNF or DNF, or doing other manipulations, it is convenient to have a supply of basic equivalences that we can use. The following lemmas can help.

Lemma 1.26 (Algebraic equivalences). *Conjunction and disjunction are commutative, associative, and idempotent operators up to equivalence. Moreover, $\top$ is a neutral element for conjunction, and a zero*

element for disjunction; dually for $\top$. We also have the lattice absorption and distributivity laws:

$$\begin{array}{ll}
 \varphi \wedge (\psi \wedge \chi) \equiv (\varphi \wedge \psi) \wedge \chi & \varphi \vee (\psi \vee \chi) \equiv (\varphi \vee \psi) \vee \chi \\
 \varphi \wedge \psi \equiv \psi \wedge \varphi & \varphi \vee \psi \equiv \psi \vee \varphi \\
 \varphi \wedge \varphi \equiv \varphi & \varphi \vee \varphi \equiv \varphi \\
 \varphi \wedge (\varphi \vee \psi) \equiv \varphi & \varphi \vee (\varphi \wedge \psi) \equiv \varphi \\
 \varphi \wedge (\psi \vee \chi) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \chi) & \varphi \vee (\psi \wedge \chi) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \chi) \\
 \varphi \wedge \top \equiv \varphi & \varphi \vee \perp \equiv \varphi \\
 \varphi \wedge \perp \equiv \perp & \varphi \vee \top \equiv \top.
 \end{array}$$

Lemma 1.27. *The following so called De Morgan laws hold:*

$$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi, \quad \neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi.$$

(We also have $\neg\neg\varphi \equiv \varphi$.) More generally,

$$\neg \bigvee_{i < n} \varphi_i \equiv \bigwedge_{i < n} \neg\varphi_i, \quad \neg \bigwedge_{i < n} \varphi_i \equiv \bigvee_{i < n} \neg\varphi_i.$$

Lemmas 1.26 and 1.27 show that the quotient structure $\langle \text{Prop}_A, \wedge, \vee, \neg, \perp, \top \rangle / \equiv$ is a Boolean algebra.

When manipulating formulas using Lemmas 1.26 and 1.27, we do not apply them to the whole formula, but to subformulas (= substrings that are themselves formulas; we will not formally introduce this notion and/or propositional substitution). This is justified by the following lemma, whose proof we also omit:

Lemma 1.28 (Substitution of equivalents). *If $\varphi \equiv \varphi'$, and ψ' is obtained from ψ by replacing some subformulas φ with φ' , then $\psi \equiv \psi'$.*

1.2 Completeness of propositional logic

We want to characterize entailment in propositional logic by a proof system: a proof of φ from Γ is a certificate or witness¹ whose existence guarantees that φ indeed follows from Γ , and whose correctness is easily checkable.

Many different kinds of proof systems with different properties are considered in the literature on proof theory and proof complexity, such as Hilbert-style proof systems, sequent calculi, natural deduction, resolution, etc. Furthermore, the selection of axioms and inference rules in proof systems of one type may differ.

We are going to work with a Hilbert-style proof system, also known as a Frege system: this means that a proof is just a sequence of formulas consisting of axioms and of formulas inferred by specified rules of inference. The advantage of such proof systems is that they are very simple to define; their disadvantage is a lack of analogues of more advanced proof-theoretic tools such as cut elimination/proof normalization, but these are outside the scope of this course anyway.

The De Morgan language is rather inconvenient to formulate Hilbert-style proof systems for: on the one hand, it includes many redundant connectives, which means we need many axioms to fix their properties; on the other hand, most natural axioms have the form of an implication, but $\rightarrow$ is not included in the De Morgan language. To make our lives simpler, we will use in this section an alternative language with only $\{\rightarrow, \perp\}$ as the basic connectives (which is readily seen to be functionally complete). A complete proof system using the De Morgan language can be found in the van den Dries lecture notes; alternatively, you can solve Exercise 12.

¹Unlike satisfying assignments as witnesses for satisfiability (cf. Remark 1.25), proofs may in general be much larger than the formulas they prove. Whether propositional proofs can be made polynomial size is closely related to the $\text{NP} \stackrel{?}{=} \text{coNP}$ question.

Definition 1.29. A (*propositional*) *logical axiom* is any instance of one of the axiom schemata

- (A1) $\varphi \rightarrow (\psi \rightarrow \varphi),$
 (A2) $(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi)),$
 (A3) $((\varphi \rightarrow \perp) \rightarrow \perp) \rightarrow \varphi.$

We also consider the schematic inference rule *modus ponens*:

- (MP) From φ and $\varphi \rightarrow \psi$ infer ψ .

Definition 1.30 (Proofs). Let $\varphi \in \text{Prop}_A$ and $\Gamma \subseteq \text{Prop}_A$. A *proof* (or *derivation*) of φ from Γ is a sequence of formulas $\varphi_0, \varphi_1, \dots, \varphi_s$ such that $\varphi_s = \varphi$ and for every $i = 0, \dots, s$ one of the following holds:

- $\varphi_i \in \Gamma$;
- φ_i is a logical axiom (i.e., it is an instance of one of the axiom schemata);
- there are $j, k < i$ such that φ_i is derived from φ_j, φ_k using *modus ponens*.

A formula φ is *provable from* Γ , written $\Gamma \vdash \varphi$, if there exists a proof of φ from Γ .

A proof of φ from the empty set $\emptyset$ is simply called a *proof of* φ , and φ is called *provable*, written $\vdash \varphi$. Γ is *consistent* if $\Gamma \not\vdash \perp$.

Remark 1.31. Definition 1.30 is an inductive definition in thin disguise: we could have just said that the set of formulas provable from Γ (the *deductive closure* of Γ) is the smallest set of formulas that includes Γ and logical axioms, and that is closed under modus ponens. The definition of proofs then follows by unwinding the inductive definition along the lines of Remark 1.6 (ii). However, we want to concentrate on proofs as objects of study, hence we prefer to state the definition explicitly in terms of proofs up front.

We first observe some basic structural properties of provability. (Conditions (i)–(iii) say that $\vdash$ obeys Tarski's definition of an abstract *consequence relation*; condition (iv) say that the consequence relation is *finitary*.)

Observation 1.32. Let $\Gamma, \Delta \subseteq \text{Prop}_A$ and $\varphi, \psi \in \text{Prop}_A$.

- (i) If $\varphi \in \Gamma$, then $\Gamma \vdash \varphi$.
- (ii) If $\Gamma \vdash \varphi$ and $\Gamma \subseteq \Delta$, then $\Delta \vdash \varphi$.
- (iii) If $\Gamma \vdash \varphi$, and $\Delta \vdash \psi$ for each $\psi \in \Gamma$, then $\Delta \vdash \varphi$. (As a special case, if $\Gamma, \psi \vdash \varphi$ and $\Delta \vdash \psi$, then $\Gamma, \Delta \vdash \varphi$.)
- (iv) If $\Gamma \vdash \varphi$, there exists a finite $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \vdash \varphi$.

Proof. Exercise. We point out that (iv) follows from the important fact that any proof contains only finitely many formulas. $\square$

Our present goal is to prove the following theorem:

Theorem 1.33 (Soundness and completeness). For all $\varphi \in \text{Prop}_A$ and $\Gamma \subseteq \text{Prop}_A$,

$$\Gamma \vdash \varphi \iff \Gamma \models \varphi.$$

The soundness theorem is the left-to-right implication (which is the easy part, and will be left as an exercise); the completeness theorem proper is the right-to-left implication.

Lemma 1.34 (Deduction). For all $\Gamma \subseteq \text{Prop}_A$ and $\varphi, \psi \in \text{Prop}_A$,

$$\Gamma, \psi \vdash \varphi \iff \Gamma \vdash \psi \rightarrow \varphi.$$

Proof. The right-to-left implication is trivial (it follows by a single application of *modus ponens*). We shall now prove the left-to-right implication. By assumption, there is a proof $\varphi_0, \dots, \varphi_n = \varphi$ of φ from $\Gamma \cup \{\psi\}$. We will show $\Gamma \vdash \psi \rightarrow \varphi_i$ by induction on $i \leq n$. We distinguish the ways how φ_i could have been derived:

- Suppose either φ_i is a logical axiom or $\varphi_i \in \Gamma$. Then $\Gamma \vdash \varphi_i$, and we infer $\Gamma \vdash \psi \rightarrow \varphi_i$ using the instance $\varphi_i \rightarrow (\psi \rightarrow \varphi_i)$ of (A1) and (MP).
- Suppose $\varphi_i = \psi$. The following proof shows $\Gamma \vdash \psi \rightarrow \psi$:

$$(\psi \rightarrow ((\psi \rightarrow \psi) \rightarrow \psi)) \rightarrow ((\psi \rightarrow (\psi \rightarrow \psi)) \rightarrow (\psi \rightarrow \psi)) \quad (\text{A2})$$

$$\psi \rightarrow ((\psi \rightarrow \psi) \rightarrow \psi) \quad (\text{A1})$$

$$(\psi \rightarrow (\psi \rightarrow \psi)) \rightarrow (\psi \rightarrow \psi) \quad (\text{MP})$$

$$\psi \rightarrow (\psi \rightarrow \psi) \quad (\text{A1})$$

$$\psi \rightarrow \psi \quad (\text{MP})$$

- Suppose φ_i has been derived by (MP) from φ_j and φ_k for some $j, k < i$, i.e., $\varphi_k = (\varphi_j \rightarrow \varphi_i)$. By the induction hypothesis, $\Gamma \vdash \psi \rightarrow \varphi_j$ and $\Gamma \vdash \psi \rightarrow (\varphi_j \rightarrow \varphi_i)$. We obtain $\Gamma \vdash \psi \rightarrow \varphi_i$ using the instance

$$(\psi \rightarrow (\varphi_j \rightarrow \varphi_i)) \rightarrow ((\psi \rightarrow \varphi_j) \rightarrow (\psi \rightarrow \varphi_i))$$

of (A2) and two applications of (MP). $\square$

Corollary 1.35. $\Gamma \vdash \varphi \iff \Gamma, (\varphi \rightarrow \perp) \vdash \perp$.

Proof.

$\Rightarrow$ A simple application of *modus ponens*.

$\Leftarrow$ Using the deduction lemma,

$$\begin{aligned} \Gamma, \varphi \rightarrow \perp \vdash \perp &\implies \Gamma \vdash (\varphi \rightarrow \perp) \rightarrow \perp \\ &\implies \Gamma \vdash \varphi, \end{aligned}$$

where the last line is obtained via (MP) from $((\varphi \rightarrow \perp) \rightarrow \perp) \rightarrow \varphi$, which is (A3). $\square$

Definition 1.36 (Maximal consistent set). $\Gamma \subseteq \text{Prop}_A$ is *maximal consistent* if Γ is consistent but all $\Gamma \subsetneq \Delta \subseteq \text{Prop}_A$ are inconsistent.

Observation 1.37. A maximal consistent set $\Gamma \subseteq \text{Prop}_A$ is deductively closed, i.e., for any $\varphi \in \text{Prop}_A$, $\Gamma \vdash \varphi \implies \varphi \in \Gamma$.

Proof. $\Gamma \cup \{\varphi\}$ is still consistent, as $\Gamma, \varphi \vdash \perp$ and $\Gamma \vdash \varphi$ implies $\Gamma \vdash \perp$ by Observation 1.32. Thus, $\Gamma \cup \{\varphi\} = \Gamma$ by the maximality of Γ . $\square$

Remark 1.38. One can show (exercise) that maximal consistent sets are exactly the “complete theories”: consistent deductively closed sets $\Gamma \subseteq \text{Prop}_A$ such that $\Gamma \vdash \varphi$ or $\Gamma \vdash \varphi \rightarrow \perp$ for every $\varphi \in \text{Prop}_A$.

Lemma 1.39. Every consistent set $\Gamma \subseteq \text{Prop}_A$ is included in a maximal consistent set $\tilde{\Gamma} \subseteq \text{Prop}_A$.

Proof. We will use Zorn’s Lemma. For the partial order, take all consistent extensions $\Delta \supseteq \Gamma$ ordered by inclusion. The union of any, possibly infinite, chain ($=$ linearly ordered set) C of consistent extensions is a consistent extension, due to Observation 1.32 (iv): if $\bigcup C$ were not consistent, contradiction could be obtained from a finite subset of $\bigcup C$; since C is linearly ordered, such a finite set would be included in some $\Delta \in C$, which would contradict the consistency of Δ . Thus every chain has an upper bound. Consequently, there exists a maximal element, which is a maximal consistent extension of Γ . $\square$

Lemma 1.40. *Every maximal consistent set Γ is satisfiable.*

Proof. Let Γ be maximal consistent and define an assignment $\alpha: A \rightarrow \{0, 1\}$ by

$$\alpha(p) = 1 \iff p \in \Gamma \quad \text{for all } p \in A.$$

We will show that this equivalence holds for all formulas φ , not just for atoms:

$$\hat{\alpha}(\varphi) = 1 \iff \varphi \in \Gamma.$$

It will follow that $\alpha \models \Gamma$. We prove this by induction on the complexity of φ :

- Suppose φ is an atom. Then $\hat{\alpha}(\varphi) = 1 \iff \varphi \in \Gamma$ by definition.
- Suppose $\varphi = \perp$. Then $\hat{\alpha}(\perp) = 0$, and $\perp \notin \Gamma$ because Γ is consistent.
- Suppose $\varphi = (\psi \rightarrow \chi)$. We distinguish three cases:
 - $\hat{\alpha}(\chi) = 1$, thus $\hat{\alpha}(\psi \rightarrow \chi) = 1$.
By the induction hypothesis, $\chi \in \Gamma$, whence $\Gamma \vdash \psi \rightarrow \chi$ using (A1), and $\psi \rightarrow \chi \in \Gamma$ by Observation 1.37.
 - $\hat{\alpha}(\psi) = 0$, thus $\hat{\alpha}(\psi \rightarrow \chi) = 1$.
By the induction hypothesis, $\psi \notin \Gamma$, which implies $\Gamma, \psi \vdash \perp$ by the maximality of Γ , thus $\Gamma, \psi, \chi \rightarrow \perp \vdash \perp$ as well. It follows that $\Gamma, \psi \vdash \chi$ by Corollary 1.35, whence $\Gamma \vdash \psi \rightarrow \chi$ by the deduction lemma, and $\psi \rightarrow \chi \in \Gamma$ by Observation 1.37.
 - $\hat{\alpha}(\psi) = 1$ and $\hat{\alpha}(\chi) = 0$, thus $\hat{\alpha}(\psi \rightarrow \chi) = 0$.
By the induction hypothesis, it follows that $\psi \in \Gamma$ and $\chi \notin \Gamma$, whence $\psi \rightarrow \chi \notin \Gamma$ (otherwise $\Gamma \supseteq \{\psi, \psi \rightarrow \chi\} \vdash \chi$ by (MP), which implies $\chi \in \Gamma$, *quod non*). $\square$

Theorem 1.41 (Propositional completeness theorem). *Let $\Gamma \subseteq \text{Prop}_A$ and $\varphi \in \text{Prop}_A$. Then*

$$\Gamma \vdash \varphi \iff \Gamma \models \varphi.$$

Proof. Soundness ($\Gamma \vdash \varphi$ implies $\Gamma \models \varphi$) is left to the reader as Exercise 8. For the converse implication, assume $\Gamma \not\models \varphi$. Then $\Gamma \cup \{\varphi \rightarrow \perp\}$ is a consistent theory by Corollary 1.35, and we may extend it to a maximal consistent theory $\tilde{\Gamma}$ by Lemma 1.39. There exists an assignment α satisfying $\tilde{\Gamma}$ by Lemma 1.40. Then, in particular, $\alpha \models \Gamma$, and $\hat{\alpha}(\varphi \rightarrow \perp) = 1$ implies $\alpha \not\models \varphi$. Consequently, $\Gamma \not\models \varphi$. $\square$

Theorem 1.42 (Propositional compactness theorem). *Let $\Gamma \subseteq \text{Prop}_A$ and $\varphi \in \text{Prop}_A$.*

- (i) $\Gamma \models \varphi$ iff there exists a finite subset $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models \varphi$.
- (ii) Γ is satisfiable iff all finite $\Gamma_0 \subseteq \Gamma$ are satisfiable.

Proof. (i) follows from Theorem 1.41 and Observation 1.32. (ii): Take $\varphi = \perp$. $\square$

Remark 1.43. The compactness theorem is a purely semantic statement that does not rely on any underlying proof system; it may be proved directly without using the completeness theorem, for example by topological considerations (using the compactness of the space of all assignments; see Exercise 13).

Our proof of the completeness theorem, hence of the compactness theorem, used the axiom of choice in the form of Zorn's lemma. One can show that both statements are (over the Zermelo–Fraenkel set theory) equivalent to the *Boolean prime ideal theorem*, which is weaker than full axiom of choice. This holds also for the completeness and/or compactness theorems of first-order logic below (Theorems 1.86 and 1.88).

The compactness theorem can be used to prove combinatorial statements that ostensibly do not involve any logic, such as the De Bruijn–Erdős theorem below.

Recall that a graph $G = \langle V, E \rangle$ (where V is a set of vertices and $E \subseteq \{\{u, v\} : u, v \in V, u \neq v\}$ is a set of edges) is *k-colourable* if there exists a mapping $c: V \rightarrow [k]$ such that $c(u) \neq c(v)$ for all $\{u, v\} \in E$, where $k \in \mathbb{N}$ and $[k] = \{0, \dots, k-1\}$.

Theorem 1.44 (De Bruijn, Erdős). *For any $k \in \mathbb{N}$, a graph $G = \langle V, E \rangle$ is k -colourable iff all finite subgraphs G_0 of G are k -colourable.*

Proof. Consider a set of atoms

$$A_G = \{p_{u,i} : u \in V, i < k\},$$

where $p_{u,i}$ intuitively means “ $c(u) = i$ ”, and define a set of formulas over A_G by

$$\Gamma_G = \left\{ \bigvee_{i < k} p_{u,i} : u \in V \right\} \cup \left\{ \neg(p_{u,i} \wedge p_{v,i}) : \{u, v\} \in E, i < k \right\}.$$

Then Γ_G is satisfiable iff G is k -colourable:

$\Leftarrow$ Let $c: V \rightarrow [k]$ be a k -colouring of G . Then $\alpha \models \Gamma_G$, where $\alpha(p_{u,i}) = 1$ iff $c(u) = i$.

$\Rightarrow$ Given $\alpha \models \Gamma_G$, we define a k -colouring $c: V \rightarrow [k]$ as follows: for any $u \in V$, there is $i < k$ s.t. $\alpha(p_{u,i}) = 1$; let $c(u)$ be one such i (say, the first one). Then if $\{u, v\} \in E$, then $\alpha \models \neg(p_{u,i} \wedge p_{v,i})$ for each i , thus $c(u) \neq c(v)$.

If all finite subgraphs of G are k -colourable, then every finite $\Gamma_0 \subseteq \Gamma_G$ is satisfiable, as $\Gamma_0 \subseteq \Gamma_{G_0}$ for some finite subgraph G_0 of G . Thus, Γ_G is satisfiable by compactness, whence G is k -colourable. $\square$

1.3 First-order logic

Definition 1.45 (Language). A *language* or *signature* is a collection of relation and function symbols, each of a given arity. Formally, $L = \langle L^r, L^f, \text{ar} \rangle$ where $L^r \cap L^f = \emptyset$ and $\text{ar}: L^r \cup L^f \rightarrow \mathbb{N}$. We also require that $L^r \cup L^f$ is disjoint from $\{\wedge, \vee, \neg, \top, \perp, \forall, \exists, =, (,), \cdot\} \cup \text{Var}$ (see Definition 1.47).

For any $R \in L^r$, $\text{ar}(R) = n$ signifies that R is an n -ary relation symbol. Similarly, $F \in L^f$ with $\text{ar}(F) = n$ is an n -ary function symbol. Nullary function symbols are called *constant symbols*. Nullary relation symbols are rarely used, but they behave essentially as propositional atoms. Relation symbols, especially unary, are also called *predicate symbols*.

Definition 1.46. An L -structure is $\mathcal{A} = \langle A, \{R^{\mathcal{A}} : R \in L^r\}, \{F^{\mathcal{A}} : F \in L^f\} \rangle$ where

- $A \neq \emptyset$ is the domain² (underlying set) of $\mathcal{A}$;
- for any $R \in L^r$ with $\text{ar}(R) = n$, $R^{\mathcal{A}} \subseteq A^n$;
- for any $F \in L^f$ with $\text{ar}(F) = n$, $F^{\mathcal{A}}: A^n \rightarrow A$.

If $s \in L^r \cup L^f$, $s^{\mathcal{A}}$ is also called the *interpretation* of s in $\mathcal{A}$.

Definition 1.47. The set of *variables* is $\text{Var} = \{v_n : n \in \mathbb{N}\}$. We will denote variables with various lowercase letters such as $x, y, z, \dots$

The set Term_L of L -terms is the least set such that every variable is an L -term, and for every n -ary function symbol $F \in L^f$ and any L -terms $t_0, \dots, t_{n-1}$, we have $F(t_0, \dots, t_{n-1}) \in \text{Term}_L$.

An *atomic L -formula* is an expression of the form $R(t_0, \dots, t_{n-1})$ or $t_0 = t_1$, where $R \in L^r$ is an n -ary relation symbol and $t_0, \dots$ are L -terms.

The set Form_L of L -formulas is the least set such that every atomic L -formula is an L -formula, and if φ and ψ are L -formulas, and x is a variable, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\neg\varphi$, $\top$, $\perp$, $\exists x \varphi$, and $\forall x \varphi$ are L -formulas.

When L is understood from the context or immaterial, we will say just *term*, *formula*, etc.

²Sometimes, especially in model theory, $A = \emptyset$ is admitted as well.

Convention 1.48. In practice, we will not follow the above formalities when specifying languages, structures, terms, and formulas. We will typically write a language as a set of symbols, such as $L = \{+, -, \cdot, <\}$, where the nature of the symbols (relation/function, arity) either follows their conventional use, or is understood from the context, and we will write $R \in L$ and $F \in L$ instead of $R \in L^r$ and $F \in L^f$.

Likewise, we will write structures as tuples listing the domain and interpretations of the symbols, such as $\langle \mathbb{Z}, +, -, \cdot, < \rangle$; as also seen here, standard operations on common mathematical structures will be identified just by their symbol (we do not need to write $+\mathbb{Z}$ instead of $+$ if no confusion arises).

Common binary symbols such as $+$ and $<$ will be written in the usual infix notation, e.g., $x + y$ and $x < y$ rather than $+(x, y)$ and $<(x, y)$. Sometimes, applications of function and relation symbols are written without brackets and commas: $Rt_0 \dots t_{n-1}$, $Ft_0 \dots t_{n-1}$ (this is the official definition in some sources); we will at least invariably omit brackets in constants, writing c instead of the $c()$ required by Definition 1.47.

Following common practice, we will write $t \neq s$ for $\neg t = s$; we might do this also with other binary relations, e.g., $t \not\leq s$.

An interpretation of a constant c in a structure $\mathcal{A}$ is formally supposed to be a function $c^{\mathcal{A}}: A^0 \rightarrow A$. Since $|A^0| = 1$, such a function is uniquely specified by just giving its single value; thus, we will identify $c^{\mathcal{A}}$ with an element of A .

Definition 1.49. An occurrence of a variable x in a formula φ is *bound* if it occurs inside a subformula (= a substring that is itself a formula) that starts with $\exists x$ or $\forall x$. Such an occurrence is said to be *within the scope* of the latter bounding quantifier. An occurrence that is not bound is called *free*.

A variable is said to be free in φ if it has a free occurrence in φ (it may or may not also have bound occurrences).

Definition 1.50. A term or a formula is *closed* if it has no free variables; closed formulas are also called *sentences*. A *theory* is a set of sentences.

A formula is called *open* or *quantifier-free* if it has no bound variables.

We mention that the definition of a theory varies in the literature: sometimes, theories are required to be *deductively closed*, i.e., $\varphi \in T$ whenever φ is a sentence such that $T \vdash \varphi$ (see Definition 1.69 below).

Definition 1.51 (Substitution). Let t be a term and x a variable. Given a term s , we define $s(t/x)$ as the result of replacing every occurrence of x in s with t . If φ is a formula, then $\varphi(t/x)$ denotes the result of replacing every *free* occurrence of x in φ with t .

More generally, if $t_0, t_1, \dots, t_{n-1}$ are terms and $x_0, x_1, \dots, x_{n-1}$ are distinct variables, then for any term s , $s(t_0/x_0, \dots, t_{n-1}/x_{n-1})$ is the result of simultaneously replacing every occurrence of each x_i with t_i in s . For a formula φ , the expression $\varphi(t_0/x_0, \dots, t_{n-1}/x_{n-1})$ is defined similarly, substituting just for free occurrences of variables.

Remark 1.52. The set of free variables of a formula φ and the substitution $\varphi(t_0/x_0, \dots, t_{n-1}/x_{n-1})$ can also be defined by induction on φ (exercise).

Notation 1.53. We will write $t(x_0, \dots, x_{n-1})$ or $\varphi(x_0, \dots, x_{n-1})$ to indicate that all variables that occur free in φ or t are among $x_0, x_1, \dots, x_{n-1}$. Then we write $\varphi(t_0, \dots, t_{n-1})$ for $\varphi(t_0/x_0, \dots, t_{n-1}/x_{n-1})$ and likewise for terms. When n is not important or understood from context, we may also write $\varphi(\vec{x})$ for $\varphi(x_0, \dots, x_{n-1})$, etc.

Remark 1.54. Intuitively, the meaning of $\varphi(t/x)$ is “the property expressed by $\varphi(x)$ applied to the element denoted by t ”. However, it does not always work that way. For a simple example, let $\varphi(x)$ be the formula $\exists y y \neq x$, whose meaning is “there is an element distinct from x ”. When we substitute y for x , we obtain the formula $\exists y y \neq y$, which does not mean “there is an element distinct from y ”, but “there is an element distinct from itself”. The problem here is that we substituted the variable y into a context where it is quantified by $\exists y$, which means it locally does not denote the element referred to by the free variable. To avoid such situations, we will only use substitution when it meets the condition below:

Definition 1.55 (Term free for substitution). A term t is *free for x in φ* (or more explicitly, *free for substitution for x in φ*) if no free occurrence of x in φ is in the scope of a quantifier of the form $\exists y$ or $\forall y$ where y occurs in t .

Remark 1.56. Once we define the semantics, it will become possible to formally state and prove that if t is free for x in φ , then $\varphi(t/x)$ indeed has the “ $\varphi(x)$ applied to the element denoted by t ” meaning. In fact, a variant of this property can be stated purely syntactically as the next lemma (this implies the semantic version when we take for s_i and r the constants $\underline{a}$ introduced below and used in the definition of satisfaction).

Lemma 1.57 (Successive substitution). *If $t(x_0, \dots, x_{n-1}, y)$ is free for y in a formula $\varphi(x_0, \dots, x_{n-1}, y)$, then for all terms $\bar{s}, r$, the formula $(\varphi(t/y))(s_0/x_0, \dots, s_{n-1}/x_{n-1}, r/y)$, denoting successive substitution, is the same formula as (i.e., syntactically identical to) the formula $\varphi(s_0, \dots, s_{n-1}, t(s_0, \dots, s_{n-1}, r))$.*

Proof. Exercise 15. □

Definition 1.58 (Constant-symbol language extension). For any L -structure $\mathcal{A}$ and $X \subseteq A$, let $L_X = L \cup \{\underline{a} : a \in X\}$, where each $\underline{a}$ is a new constant symbol distinct from all others and from all symbols of L .

Then $\mathcal{A}_X$ is an L_X -structure with domain A , $s^{\mathcal{A}_X} = s^{\mathcal{A}}$ for all $s \in L$, and $\underline{a}^{\mathcal{A}_X} = a$ for all $a \in X$.

We will later cease underlining constant symbols unless needed for clarity.

Definition 1.59 (Evaluation). If $\mathcal{A}$ is an L -structure, and t is a closed term, then we define $t^{\mathcal{A}} \in A$ by induction on the complexity of t :

- If $t = F(t_0, \dots, t_{n-1})$ then $t^{\mathcal{A}} = F^{\mathcal{A}}(t_0^{\mathcal{A}}, \dots, t_{n-1}^{\mathcal{A}})$.

For any term $t(x_0, \dots, x_{n-1})$, we define $t^{\mathcal{A}} : A^n \rightarrow A$ by $t^{\mathcal{A}}(a_0, \dots, a_{n-1}) = (t(\underline{a}_0, \dots, \underline{a}_{n-1}))^{\mathcal{A}}$.

Definition 1.60 (Satisfaction, model, logical consequence). Let $\mathcal{A}$ be an L -structure. Given an L_A -sentence φ , we define $\mathcal{A} \models \varphi$ by induction on the complexity of φ :

- If φ is $R(t_0, \dots, t_{n-1})$ for some n -ary relation R and closed L_A -terms t_i , then we put $\mathcal{A} \models \varphi$ iff $\langle t_0^{\mathcal{A}}, \dots, t_{n-1}^{\mathcal{A}} \rangle \in R^{\mathcal{A}}$.
- If φ is $t = s$, $\mathcal{A} \models \varphi \iff t^{\mathcal{A}} = s^{\mathcal{A}}$.
- We shall now define behaviour of $\models$ on logical operators and quantifiers:

$$\begin{aligned} \mathcal{A} \models \varphi_0 \wedge \varphi_1 &\iff \mathcal{A} \models \varphi_0 \text{ and } \mathcal{A} \models \varphi_1, \\ \mathcal{A} \models \varphi_0 \vee \varphi_1 &\iff \mathcal{A} \models \varphi_0 \text{ or } \mathcal{A} \models \varphi_1, \\ \mathcal{A} \models \neg \varphi &\iff \mathcal{A} \not\models \varphi, \\ \mathcal{A} \models \top, \\ \mathcal{A} \not\models \perp, \\ \mathcal{A} \models \exists x \varphi &\iff \text{there exists } a \in A \text{ such that } \mathcal{A} \models \varphi(\underline{a}/x), \\ \mathcal{A} \models \forall x \varphi &\iff \text{for all } a \in A, \mathcal{A} \models \varphi(\underline{a}/x). \end{aligned}$$

Observe that in the last two clauses, $\varphi(\underline{a}/x)$ is again an L_A -sentence.

For a not necessarily closed formula $\varphi(x_0, \dots, x_{n-1})$, we write $\mathcal{A} \models \varphi$ if $\mathcal{A} \models \varphi(\underline{a}_0, \dots, \underline{a}_{n-1})$ for all $a_0, \dots, a_{n-1} \in A$; we say that φ *holds in $\mathcal{A}$* , or $\mathcal{A}$ is a *model* of φ .

More generally, if $\Gamma \subseteq \text{Form}_L$, we write $\mathcal{A} \models \Gamma$ if for all $\varphi \in \Gamma$ we have $\mathcal{A} \models \varphi$; we say that $\mathcal{A}$ is a *model* of Γ .

A formula φ is a *logical consequence* of Γ , or Γ *entails* φ , or φ *follows* from Γ , written $\Gamma \models \varphi$, if every model $\mathcal{A}$ of Γ is also a model of φ .

Finally, φ is said to be *logically valid* (or a *first-order tautology*), written $\models \varphi$, if $\emptyset \models \varphi$ (i.e., φ is entailed by $\Gamma = \emptyset$); in other words, if $\mathcal{A} \models \varphi$ for all structures $\mathcal{A}$.

L -formulas φ and ψ are *equivalent*, written $\varphi \equiv \psi$, if for every L -structure $\mathcal{A}$ and every tuple $a_0, \dots, a_{n-1} \in A$, we have

$$\mathcal{A} \models \varphi(\underline{a_0}, \dots, \underline{a_{n-1}}) \iff \mathcal{A} \models \psi(\underline{a_0}, \dots, \underline{a_{n-1}}).$$

In other words, $\varphi \equiv \psi$ iff $\models (\varphi \leftrightarrow \psi)$. More generally, L -theories T and S are *equivalent* if $\mathcal{A} \models T \iff \mathcal{A} \models S$ for all L -structures $\mathcal{A}$; or equivalently, if $T \models \varphi \iff S \models \varphi$ for all L -sentences φ .

Definition 1.61 (Universal closure). For any formula $\varphi(x_0, \dots, x_{n-1})$, its *universal closure* $\varphi^\forall$ is

$$\forall x_0 \dots \forall x_{n-1} \varphi(x_0, \dots, x_{n-1}).$$

In other words, all freely occurring variables of φ are made bound with universal quantifiers.

Remark 1.62. The definition of entailment $\Gamma \models \varphi$ and/or provability $\Gamma \vdash \varphi$ (that we will introduce later) sometimes varies in the literature when Γ contains formulas with free variables. This is the reason why it became standard to define *theories* to consist of sentences only, for which the common definitions agree. On the other hand, there is a general convention that axioms of theories may be written with outer universal quantifiers stripped, thus a formula given as an axiom of a theory really represents its universal closure.

Under our definitions, these distinctions are not important: we observe immediately from the definition that every formula is mutually entailed with its universal closure, as stated in the next lemma, and our proof system will also have this property.

Lemma 1.63. For any L -formula φ , $\varphi \models \varphi^\forall$ and $\varphi^\forall \models \varphi$. □

Lemma 1.64. If $\varphi \equiv \varphi'$, and ψ' is obtained from a formula ψ by replacing some occurrences of φ as subformulas with φ' , then $\psi \equiv \psi'$.

Proof. By induction on the complexity of ψ . □

Definition 1.65. A formula $\varphi(x_0, \dots, x_{n-1})$ is in *prenex normal form* if it has the form

$$Q_0 y_0 \dots Q_{m-1} y_{m-1} \theta(x_0, \dots, x_{n-1}, y_0, \dots, y_{m-1})$$

where $Q_i \in \{\exists, \forall\}$, the formula θ is open, and the y_i are pairwise distinct variables, distinct from the x_j s.

Lemma 1.66. Let Q be either the existential or the universal quantifier. Then

$$\begin{aligned} \neg \exists x \varphi &\equiv \forall x \neg \varphi, \\ \neg \forall x \varphi &\equiv \exists x \neg \varphi, \\ Qx \varphi &\equiv Qy \varphi(y/x) && \text{if } y \text{ is free for } x \text{ in } \varphi, \\ \left. \begin{aligned} (\varphi \wedge Qx \psi) &\equiv Qx (\varphi \wedge \psi) \\ (\varphi \vee Qx \psi) &\equiv Qx (\varphi \vee \psi) \end{aligned} \right\} && \text{if } x \text{ is not free in } \varphi. \end{aligned}$$

The first two equivalences are called the De Morgan rules for quantifiers.

Lemma 1.67. Every formula is equivalent to a formula in prenex normal form.

Proof. Apply Lemma 1.66 systematically to bring all quantifiers out, renaming them in case of clashes. □

1.4 First-order proof system

Definition 1.68. We consider the following list of axioms and rules for first-order logic:

Propositional axioms and rules of inference

Same as in Definition 1.29.

Axioms of equality

$$\begin{aligned} x &= x, \\ x = y \wedge x = z &\rightarrow y = z, \\ \left(\bigwedge_{i < n} x_i = y_i \right) &\rightarrow (R(\vec{x}) \rightarrow R(\vec{y})), \\ \left(\bigwedge_{i < n} x_i = y_i \right) &\rightarrow F(\vec{x}) = F(\vec{y}) \end{aligned}$$

for each n -ary relation symbol R and n -ary function symbol F .

Quantifier axioms and rules

Supposing t is free for x in φ :

$$\begin{aligned} \forall x \varphi &\rightarrow \varphi(t/x), \\ \varphi(t/x) &\rightarrow \exists x \varphi. \end{aligned}$$

Supposing x is not free in ψ :

$$\begin{aligned} \text{From } \psi \rightarrow \varphi \text{ infer } \psi &\rightarrow \forall x \varphi. \quad (\text{universal generalization, } (\forall\text{Gen})) \\ \text{From } \varphi \rightarrow \psi \text{ infer } \exists x \varphi &\rightarrow \psi. \quad (\text{existential generalization, } (\exists\text{Gen})) \end{aligned}$$

Definition 1.69 (Provability). Let $\Gamma \subseteq \text{Form}_L$ and $\varphi \in \text{Form}_L$. Then φ is *provable* from Γ , written $\Gamma \vdash \varphi$, if there exists a sequence of formulas $\varphi_0, \dots, \varphi_n$ (called a *proof* or *derivation* of φ from Γ) such that $\varphi_n = \varphi$, and for each $i \leq n$, one of the following holds:

- $\varphi_i \in \Gamma$;
- φ_i is a logical axiom;
- φ_i is derived by a rule of inference from some of the φ_j , $j < i$.

If $\Gamma = \emptyset$, we just say that φ is *provable*, and write $\vdash \varphi$.

Remark 1.70. Since we base our proof system on the propositional one from Definition 1.29, we again need to adjust the language so that we use the propositional connectives $\{\rightarrow, \perp\}$ instead of $\{\wedge, \vee, \neg, \top, \perp\}$. In particular, we should, strictly speaking, replace the conjunctions appearing in the axioms of equality with equivalent $\{\rightarrow, \perp\}$ -formulas; since the conjunctions only occur as antecedents of implications, we can do it elegantly by rewriting $(\bigwedge_{i < n} x_i = y_i) \rightarrow (R(\vec{x}) \rightarrow R(\vec{y}))$ as

$$x_0 = y_0 \rightarrow (x_1 = y_1 \rightarrow (\dots \rightarrow (x_{n-1} = y_{n-1} \rightarrow (R(\vec{x}) \rightarrow R(\vec{y}))) \dots)),$$

and similarly for the other formulas.

Alternatively, we could base our proof system on a propositional system that directly uses the De Morgan connectives. In a way, since we already proved the propositional completeness theorem, the choice of the propositional part of our calculus, including what set of basic connectives it employs, does not matter much. We may essentially treat the propositional part of the proof system as a “black box”: the exact selection of axioms and rules is not important as long as they are complete for propositional logic. As the next lemma shows, we can allow *any* propositional tautologies as axioms, and any propositionally valid rules as rules of inference. But in order not to confuse things further, we will formally stick with $\{\rightarrow, \perp\}$ as the set of basic connectives in this and the next section.

Lemma 1.71. *If $\varphi(p_0, \dots, p_{n-1})$ is a propositional tautology, then*

$$\vdash \varphi(\psi_0, \dots, \psi_{n-1})$$

for any first-order formulas $\psi_0, \dots, \psi_{n-1}$. More generally, if

$$\varphi_0(\vec{p}), \dots, \varphi_{m-1}(\vec{p}) \models \varphi(\vec{p})$$

for propositional formulas $\varphi_0, \dots, \varphi_{m-1}, \varphi$, then

$$\varphi_0(\vec{\psi}), \dots, \varphi_{m-1}(\vec{\psi}) \vdash \varphi(\vec{\psi}).$$

We will say that $\varphi(\vec{\psi})$ follows from $\varphi_0(\vec{\psi}), \dots, \varphi_{m-1}(\vec{\psi})$ by propositional reasoning.

Proof. By Theorem 1.41, we may fix a propositional proof $\chi_0(\vec{p}), \dots, \chi_s(\vec{p})$ of $\varphi(\vec{p})$ from $\{\varphi_i(\vec{p}) : i < m\}$. (If the proof involves other atoms than $\vec{p}$, we either substitute them with e.g. $\perp$, or we extend the $\vec{\psi}$ list. This is not important.) Then $\chi_0(\vec{\psi}), \dots, \chi_s(\vec{\psi})$ is a first-order proof of $\varphi(\vec{\psi})$ from $\{\varphi_i(\vec{\psi}) : i < m\}$. $\square$

Remark 1.72. Using ($\forall$ Gen) and propositional reasoning, it is easy to derive a simpler version of the ($\forall$ Gen) rule: $\varphi \vdash \forall x \varphi$. This implies that our proof system shares with the notion of entailment the property that any formula is equiderivable with its universal closure, as we promised earlier:

Lemma 1.73. *For any L -formula φ , $\varphi \vdash \varphi^\forall$ and $\varphi^\forall \vdash \varphi$. $\square$*

Remark 1.74. As in propositional logic, our basic tool for constructing proofs will be the deduction lemma. We have to be slightly careful with its formulation: as we just observed, we have $\varphi \vdash \forall x \varphi$, but in general $\not\vdash \varphi \rightarrow \forall x \varphi$ as this formula is not logically valid. In order to fend off such examples, we will require that the principal formula in the deduction lemma is a sentence.

Lemma 1.75 (Deduction). *If Γ is a set of L -formulas, ψ is an L -formula, and φ is an L -sentence, then*

$$\Gamma, \varphi \vdash \psi \iff \Gamma \vdash \varphi \rightarrow \psi.$$

Proof. The right-to-left implication is a trivial application of (MP). For the left-to-right implication, let $\varphi_0, \dots, \varphi_n = \psi$ be a proof of ψ from $\Gamma \cup \{\varphi\}$. We show $\Gamma \vdash \varphi \rightarrow \varphi_i$ by induction on i :

- (i) Suppose that $\varphi_i \in \Gamma \cup \{\varphi\}$, or φ_i is an axiom, or φ_i is derived by (MP). Then the proof is identical to that of Lemma 1.34.
- (ii) Suppose φ_i is derived by the existential generalization rule; i.e., $\varphi_i = \exists x \alpha \rightarrow \beta$ is derived from $\varphi_j = \alpha \rightarrow \beta$, $j < i$, where x is not free in β . By the induction hypothesis, $\Gamma \vdash \varphi \rightarrow (\alpha \rightarrow \beta)$, and thus

$\Gamma \vdash \varphi \rightarrow (\alpha \rightarrow \beta)$	induction hypothesis
$\vdash \alpha \rightarrow (\varphi \rightarrow \beta)$	propositional reasoning
$\vdash \exists x \alpha \rightarrow (\varphi \rightarrow \beta)$	($\exists$ Gen)
$\vdash \varphi \rightarrow (\exists x \alpha \rightarrow \beta)$	propositional reasoning.

We can use ($\exists$ Gen) because φ is a sentence, thus x is not free in $\varphi \rightarrow \beta$.

- (iii) Finally, suppose φ_i is derived by universal generalization; i.e., that $\varphi_i = \beta \rightarrow \forall x \alpha$ is derived from $\varphi_j = \beta \rightarrow \alpha$. Similarly to the existential case, we can derive

$\Gamma \vdash \varphi \rightarrow (\beta \rightarrow \alpha)$	induction hypothesis
$\vdash \varphi \wedge \beta \rightarrow \alpha$	propositional reasoning
$\vdash \varphi \wedge \beta \rightarrow \forall x \alpha$	($\forall$ Gen)
$\vdash \varphi \rightarrow (\beta \rightarrow \forall x \alpha)$	propositional reasoning. $\square$

Our main goal in Part 1 is to prove that the first-order proof system we have defined adequately captures logical consequence. Let us start with the easy part:

Theorem 1.76 (Soundness theorem). *Let $\Gamma \subseteq \text{Form}_L$ and $\varphi \in \text{Form}_L$. Then*

$$\Gamma \vdash \varphi \implies \Gamma \models \varphi.$$

Proof. We fix a proof, say $\varphi_0, \dots, \varphi_n$, of φ from Γ , and let $\mathcal{A}$ be an L -structure such that $\mathcal{A} \models \Gamma$. We will show $\mathcal{A} \models \varphi_i$ by induction on i , and hence $\mathcal{A} \models \varphi$. In essence, we are proving that satisfaction is preserved under the rules of inference. As before, we consider the various ways φ_i could have been derived from some φ_j , $j < i$, and we analyse each case individually:

Derived propositionally. This case includes $\varphi_i \in \Gamma$, φ_i being a logical axiom, and being derived by (MP). We leave this to the reader as an exercise.

Derived by universal generalization. By the definition of ($\forall$ Gen), we need to verify that

$$\mathcal{A} \models \underbrace{\beta(\vec{x}) \rightarrow \alpha(\vec{x}, y)}_{\varphi_j} \quad \text{implies} \quad \mathcal{A} \models \underbrace{\beta(\vec{x}) \rightarrow \forall y \alpha(\vec{x}, y)}_{\varphi_i},$$

where we indicated explicitly the free variables. Note that y does not occur free in β by assumptions of the ($\forall$ Gen) rule. Assuming

$$\mathcal{A} \models \beta(\vec{x}) \rightarrow \alpha(\vec{x}, y),$$

we will show

$$\mathcal{A} \models \beta(\vec{x}) \rightarrow \forall y \alpha(\vec{x}, y)$$

using the definition of satisfaction: let $\vec{a} \in A$ be such that $\mathcal{A} \models \beta(\vec{a})$; then we need to check $\mathcal{A} \models \forall y \alpha(\vec{a}, y)$.

Let $b \in A$: we have $\mathcal{A} \models \beta(\vec{a}) \rightarrow \alpha(\vec{a}, b)$, thus $\mathcal{A} \models \alpha(\vec{a}, b)$. This means $\mathcal{A} \models \forall y \alpha(\vec{a}, y)$ as b was arbitrary.

Derived by existential generalization. Suppose $\mathcal{A} \models \alpha(\vec{x}, y) \rightarrow \beta(\vec{x})$; we need to show that $\mathcal{A} \models \exists y \alpha(\vec{x}, y) \rightarrow \beta(\vec{x})$. This can be proved by a similar argument as for ($\forall$ Gen).

Axiom of equality. This follows easily. For example, assume φ_i is the axiom

$$x_0 = y_0 \wedge \dots \wedge x_{n-1} = y_{n-1} \rightarrow F(x_0, \dots, x_{n-1}) = F(y_0, \dots, y_{n-1}).$$

For every $\vec{a}, \vec{b} \in A$, if $\mathcal{A} \models \bigwedge_{i < n} a_i = b_i$, then $a_0 = b_0$ and $\dots$ and $a_{n-1} = b_{n-1}$, thus $F^{\mathcal{A}}(\vec{a}) = F^{\mathcal{A}}(\vec{b})$, i.e., $\mathcal{A} \models F(\vec{a}) = F(\vec{b})$.

Quantifier axiom. Consider an axiom φ_i of the form $\alpha(t/y) \rightarrow \exists y \alpha(\vec{x}, y)$. Recall that to postulate this axiom, we require that t be free for y in α . Indicating explicitly the free variables, φ is

$$\alpha(\vec{x}, t(\vec{x}, y)/y) \rightarrow \exists y \alpha(\vec{x}, y).$$

We need to show that this holds in any structure $\mathcal{A}$. Let $\vec{a}, b \in A$ be such that

$$\mathcal{A} \models \alpha(\vec{x}, t(\vec{x}, y)/y)(\vec{a}/\vec{x}, b/y).$$

By Lemma 1.57, this means

$$\mathcal{A} \models \alpha(\vec{a}, t(\vec{a}, b)).$$

Putting $c = t^{\mathcal{A}}(\vec{a}, b)$, we obtain $\mathcal{A} \models \alpha(\vec{a}, c)$ (Exercise 16). It follows that $\mathcal{A} \models \exists y \alpha(\vec{a}, y)$ by the definition of satisfaction.

The argument for axioms of the form $\forall y \alpha \rightarrow \alpha(t/y)$ is completely analogous. $\square$

1.5 Completeness of first-order logic

We aim to prove the completeness theorem:

Theorem 1.77. *If Γ is a set of L -formulas and φ is an L -formula, then*

$$\Gamma \models \varphi \implies \Gamma \vdash \varphi.$$

Definition 1.78. Let T be an L -theory. Then T is said to be

- *consistent* if $T \not\vdash \perp$;
- *complete* if for all L -sentences φ , we have $T \vdash \varphi$ or $T \vdash \neg\varphi$;
- *Henkin* if every existential statement has a witness: i.e., for every L -formula $\varphi(x)$, there is a constant c (called the *Henkin constant* for φ) such that

$$T \vdash \exists x \varphi(x) \rightarrow \varphi(c).$$

Remark 1.79. Since our proof systems uses $\{\rightarrow, \perp\}$ as basic connectives, $\neg\varphi$ in the definition of complete theories formally denotes $(\varphi \rightarrow \perp)$.

A complete theory is essentially the same thing as a maximal consistent set of sentences (a first-order version of Definition 1.36; cf. Remark 1.38). We use the former instead of the latter as it is an notion of independent interest outside the context of the proof of the completeness theorem (e.g., it will be of central importance in Part 3). In this connection, we note for future reference that an L -sentence φ such that $T \not\vdash \varphi$ and $T \not\vdash \neg\varphi$ is called *independent of T* (or *undecidable in T* , though this usage somewhat clashes with the notion of algorithmic undecidability that we will see later in Definitions 2.6 and 2.32). Sentences φ such that $T \vdash \neg\varphi$ are called *refutable* in T .

An outline of our proof of Theorem 1.77 is as follows:

- Reduce it to proving that any consistent theory T has a model.
- If $T \not\vdash \perp$, there is a complete theory $\tilde{T} \supseteq T$, $\tilde{T} \not\vdash \perp$, in the same language.
- If $T \not\vdash \perp$, there is a Henkin theory $T_H \supseteq T$, $T_H \not\vdash \perp$ (in an expanded language).
- If $T \not\vdash \perp$ is complete and Henkin, there is a structure $\mathcal{A}$ such that $\mathcal{A} \models T$.

We shall proceed with the details. We start with the last point, which explains the motivation for introducing Henkin theories. But let us first observe that even though the definition of Henkin theories only provides Henkin constants that witness existential sentences, we also obtain suitable Henkin constants witnessing universal sentences:

Lemma 1.80. *If T is a Henkin L -theory, then for every L -formula $\varphi(x)$, there is a constant c such that $T \vdash \varphi(c) \rightarrow \forall x \varphi(x)$.*

Proof. By assumption, there is a constant c such that $T \vdash \exists x \neg\varphi(x) \rightarrow \neg\varphi(c)$. Then we have

$T \vdash \neg\varphi(x) \rightarrow \exists x \neg\varphi(x)$	axiom
$\vdash \exists x \neg\varphi(x) \rightarrow \neg\varphi(c)$	Henkin assumption
$\vdash \neg\varphi(x) \rightarrow \neg\varphi(c)$	propositional reasoning
$\vdash \varphi(c) \rightarrow \varphi(x)$	more propositional reasoning
$\vdash \varphi(c) \rightarrow \forall x \varphi(x)$	($\forall$ Gen).

□

Lemma 1.81. *If T is a complete and consistent Henkin theory, then T has a model.*

Proof. Let CT stand for the collection of all closed L -terms. We define an equivalence relation on CT by $t \sim s$ iff $T \vdash t = s$. The axioms of equality ensure that $\sim$ is an equivalence relation, hence we may define the quotient set $A = CT/\sim$. If t is a closed term, let $[t]$ denote the equivalence class of t .

We define an L -structure $\mathcal{A}$ with underlying set A by

$$\begin{aligned} F^{\mathcal{A}}([t_0], \dots, [t_{n-1}]) &= [F(t_0, \dots, t_{n-1})], \\ \langle [t_0], \dots, [t_{n-1}] \rangle \in R^{\mathcal{A}} &\iff T \vdash R(t_0 \dots t_{n-1}) \end{aligned}$$

for each n -ary function symbol F , and n -ary relation symbol R . In order to make sure that $F^{\mathcal{A}}$ and $R^{\mathcal{A}}$ are well-defined, we need to check that the definitions are independent of the choice of representatives of the equivalence classes: i.e., if $[t_0] = [s_0], \dots, [t_{n-1}] = [s_{n-1}]$, then $[F(\vec{t})] = [F(\vec{s})]$, and $T \vdash R(\vec{t}) \iff T \vdash R(\vec{s})$. This follows from the equality axioms.

We can show $t^{\mathcal{A}} = [t]$ for each $t \in CT$ by induction on the complexity of t .

We claim that

$$\mathcal{A} \models \varphi \iff T \vdash \varphi$$

for all sentences φ , which implies $\mathcal{A} \models T$. We proceed by induction on the complexity of φ :

Atomic formula. Suppose φ is $R(t_0, \dots, t_{n-1})$. Then

$$\begin{aligned} \mathcal{A} \models R(t_0 \dots t_{n-1}) &\iff \langle t_0^{\mathcal{A}}, \dots, t_{n-1}^{\mathcal{A}} \rangle \in R^{\mathcal{A}} \\ &\iff \langle [t_0], \dots, [t_{n-1}] \rangle \in R^{\mathcal{A}} \\ &\iff T \vdash R(t_0, \dots, t_{n-1}). \end{aligned}$$

The same argument also applies with $=$ in place of R .

Falsum. Suppose φ is $\perp$. Then $T \not\vdash \perp$ by consistency, and $\mathcal{A} \not\models \perp$ by definition.

Implication. Suppose φ is $\varphi_0 \rightarrow \varphi_1$. Then

$$\begin{aligned} \mathcal{A} \models \varphi_0 \rightarrow \varphi_1 &\iff \mathcal{A} \not\models \varphi_0 \text{ or } \mathcal{A} \models \varphi_1 \\ &\iff T \vdash \neg \varphi_0 \text{ or } T \vdash \varphi_1 && \text{induction hypothesis} \\ &\iff T \vdash \varphi_0 \rightarrow \varphi_1. \end{aligned}$$

For the last equivalence, “ $\Rightarrow$ ” follows by propositional reasoning (we have $\neg \varphi_0 \vdash \varphi_0 \rightarrow \varphi_1$ and $\varphi_1 \vdash \varphi_0 \rightarrow \varphi_1$). To show “ $\Leftarrow$ ”, assume $T \vdash \varphi_0 \rightarrow \varphi_1$. If $T \vdash \varphi_0$, we obtain $T \vdash \varphi_1$ by (MP); otherwise, $T \vdash \neg \varphi_0$ by the completeness of T .

Existential quantification. Suppose φ is $\exists x \psi(x)$. Then

$$\begin{aligned} \mathcal{A} \models \exists x \psi(x) &\iff (\exists t \in CT) \mathcal{A} \models \psi([t]) \\ &\iff (\exists t \in CT) \mathcal{A} \models \psi(t) && \text{using } t^{\mathcal{A}} = [t] = [t]^{\mathcal{A}} \\ &\iff (\exists t \in CT) T \vdash \psi(t) && \text{induction hypothesis} \\ &\iff T \vdash \exists x \psi(x). \end{aligned}$$

Concerning the last equivalence, for “ $\Rightarrow$ ”, use the axiom $\psi(t) \rightarrow \exists x \psi(x)$; for “ $\Leftarrow$ ”, T is Henkin whence there exists a constant c such that $T \vdash \exists x \psi(x) \rightarrow \psi(c)$.

Universal quantification. Suppose φ is $\forall x \psi(x)$. We compute

$$\begin{aligned} \mathcal{A} \models \forall x \psi(x) &\iff (\forall t \in CT) \mathcal{A} \models \psi([t]) \\ &\iff (\forall t \in CT) \mathcal{A} \models \psi(t) \\ &\iff (\forall t \in CT) T \vdash \psi(t) \\ &\iff T \vdash \forall x \psi(x) \end{aligned}$$

similarly to the existential case. □

Lemma 1.82. *If T is a consistent L -theory, then there exists a complete consistent L -theory $\tilde{T}$ extending it; i.e., $\tilde{T} \supseteq T$.*

Proof. As in the propositional case, we use Zorn's lemma to show that there exists a maximal consistent L -theory $\tilde{T}$ such that $\tilde{T} \supseteq T$ (recall that the union of a chain of consistent theories is consistent).

To see that $\tilde{T}$ is complete, let φ be a sentence such that $\tilde{T} \not\vdash \varphi$; we will show $\tilde{T}$ proves $\neg\varphi$, i.e., $\varphi \rightarrow \perp$. By the maximality of $\tilde{T}$, the theory $\tilde{T} \cup \{\varphi\}$ is inconsistent: $\tilde{T}, \varphi \vdash \perp$. Thus, $\tilde{T} \vdash \varphi \rightarrow \perp$ by the deduction theorem. $\square$

Lemma 1.83 (Constants). *Let T be an L -theory, $\varphi(x)$ an L -formula, and c a constant symbol such that $c \notin L$. Then*

$$T \vdash \varphi(c) \quad \text{implies} \quad T \vdash \varphi(x).$$

Proof. Let $\varphi_0, \dots, \varphi_n$ be a proof of $\varphi(c)$ in T , and y be a variable that does not occur in the proof. Then $\varphi_0(y/c), \dots, \varphi_n(y/c)$ is still a valid proof of $(\varphi(c/x))(y/c) = \varphi(y/x)$ from T . (The meaning of “ (y/c) ” is that we replace each occurrence of c with y ; this is not formally a substitution according to Definition 1.51 as c is not a variable.) Thus, T proves $\varphi(y/x)$; we may infer $\forall y \varphi(y/x)$ using $(\forall\text{Gen})$, and then $\varphi(x)$ using the axiom $\forall y \varphi(y/x) \rightarrow \underbrace{(\varphi(y/x))(x/y)}_{\varphi(x)}$. $\square$

Lemma 1.84. *If T is a consistent L -theory, $c \notin L$ a constant symbol, and $\varphi(x)$ an L -formula, then the following theory is consistent:*

$$T \cup \{\exists x \varphi(x) \rightarrow \varphi(c)\}.$$

Proof. If $T, \exists x \varphi(x) \rightarrow \varphi(c) \vdash \perp$, let y be a variable not occurring in $\varphi(x)$. Then

$$\begin{array}{ll} T \vdash (\exists x \varphi(x) \rightarrow \varphi(c)) \rightarrow \perp & \text{deduction theorem} \\ T \vdash (\exists x \varphi(x) \rightarrow \varphi(y)) \rightarrow \perp & \text{lemma on constants} \\ T \vdash \exists y (\exists x \varphi(x) \rightarrow \varphi(y)) \rightarrow \perp & (\exists\text{Gen}). \end{array}$$

But $\vdash \exists y (\exists x \varphi(x) \rightarrow \varphi(y))$ (Exercise 18), hence $T \vdash \perp$, which is a contradiction. $\square$

Lemma 1.85. *Let T be a consistent L -theory; then there exists a language $L_H \supseteq L$ and a consistent Henkin L_H -theory $T_H \supseteq T$.*

Proof. We construct the language L_H and the theory T_H inductively as follows³:

$$\begin{array}{ll} L_0 = L & L_{n+1} = L_n \cup \{c_\varphi : \varphi(x) \text{ is an } L_n\text{-formula}\} \\ T_0 = T & T_{n+1} = T_n \cup \{\exists x \varphi(x) \rightarrow \varphi(c_\varphi) : \varphi(x) \text{ is an } L_n\text{-formula}\} \\ L_H = \bigcup_{n \in \mathbb{N}} L_n & T_H = \bigcup_{n \in \mathbb{N}} T_n \end{array}$$

Clearly, T_H is an L_H -theory such that $T_H \supseteq T$. If $\varphi(x)$ is an L_H -formula, then $\varphi(x)$ is an L_n -formula for some $n \in \mathbb{N}$, hence $T_H \supseteq T_n$ includes the Henkin axiom $\exists x \varphi(x) \rightarrow \varphi(c_\varphi)$. Thus, T_H is a Henkin theory.

It remains to show that T_H is consistent. It suffices to show that $T_n \not\vdash \perp$ for all $n \in \mathbb{N}$. We do this by induction on n . For the base case, $T_0 = T$ is consistent by assumption.

Let us show the induction step for $n + 1$. Assume that $T_n \not\vdash \perp$, and suppose $T_{n+1} \vdash \perp$ towards a contradiction. Then

$$T_n \cup \{\exists x \varphi_i(x) \rightarrow \varphi_i(c_{\varphi_i}) : i < m\} \vdash \perp$$

for some $m \in \mathbb{N}$ and some L_n -formulas φ_i , $i < m$. But this theory is consistent by m applications of Lemma 1.84 (more formally, we should prove this by induction on m). This is a contradiction. $\square$

³Note that once $\varphi(x)$ is an L_n -formula, it is also an L_m -formula for every $m \geq n$, hence we introduce (wastefully) a new constant c_φ in each L_m , $m > n$, even though this is not indicated in the notation. That is, we should formally distinguish the constants by writing something like $c_{\varphi,n}$ for the constants newly introduced in L_{n+1} ; alternatively, we could define L_{n+1} so that we add the Henkin constants only for formulas that do not already have them. We will not bother with these technicalities as they do not really matter.

Theorem 1.86 (Completeness). *Let Γ be a set of L -formulas and φ an L -formula. Then*

$$\Gamma \models \varphi \quad \text{implies} \quad \Gamma \vdash \varphi.$$

Proof. Assume $\Gamma \not\models \varphi$. Then $\Gamma^\forall \not\models \varphi^\forall$ by Lemma 1.73, thus the theory $T = \Gamma^\forall \cup \{\neg\varphi^\forall\}$ is consistent. By Lemma 1.85, T may be extended to a consistent Henkin L_H -theory T_H , which in turn may be extended to a consistent complete L_H -theory $\tilde{T}$ by Lemma 1.82. $\tilde{T}$ remains a Henkin theory.

Since $\tilde{T}$ is a consistent complete Henkin theory, it has a model $\mathcal{A}_H \models \tilde{T}$ by Lemma 1.81. Observe $\mathcal{A}_H$ is, in particular, an L_H -structure.

Let $\mathcal{A}$ be the L -reduct of $\mathcal{A}_H$; i.e., we forget about the interpretations of symbols outside of L .

Then $\mathcal{A} \models T$, whence $\mathcal{A} \models \Gamma^\forall$ and $\mathcal{A} \not\models \varphi^\forall$. It follows that $\mathcal{A} \models \Gamma$ and $\mathcal{A} \not\models \varphi$ (Lemma 1.63), which proves $\Gamma \not\models \varphi$. $\square$

Before we forget the proof of the completeness theorem, let us observe that it also gives an upper bound on the minimal cardinality of models of consistent theories (the cardinality of an L -structure is understood to be the cardinality of its underlying set):

Theorem 1.87 (Downward Löwenheim–Skolem theorem). *Let T be an L -theory and $\kappa \geq |L|$ an infinite cardinal. If T has a model, then it has a model $\mathcal{A} \models T$ such that $|A| \leq \kappa$.*

Proof. Let us estimate the size of the model of T constructed in the proof of Theorem 1.86. Since L -formulas in a language L of cardinality $|L| \leq \kappa$ are finite strings made of $\leq \kappa$ many possible symbols, there are at most $\kappa^{<\omega} = \kappa$ many L -formulas. It follows by induction on n that the languages L_n from the proof of Lemma 1.85 satisfy $|L_n| \leq \kappa$: for the induction step, we have

$$|L_{n+1}| \leq \underbrace{|L_n|}_{\leq \kappa} + \underbrace{|\{c_\varphi : \varphi(x) \text{ is an } L_n\text{-formula}\}|}_{\leq \kappa} \leq \kappa.$$

This implies $|L_H| \leq \kappa$, and in particular, there are $\leq \kappa$ closed L_H -terms. Thus, the model of the Henkin completion of T in language L_H defined in the proof of Lemma 1.81 has cardinality at most κ . $\square$

1.6 Consequences of the completeness theorem

Theorem 1.88 (Compactness). *Let Γ be a set of L -formulas.*

- (i) $\Gamma \models \varphi$ iff there is a finite subset $\Gamma_0 \subseteq \Gamma$ such that $\Gamma_0 \models \varphi$.
- (ii) Γ has a model iff every finite $\Gamma_0 \subseteq \Gamma$ has a model.

Proof.

- (i) “ $\Rightarrow$ ” is trivial; for “ $\Leftarrow$ ”, $\Gamma \models \varphi$ implies $\Gamma \vdash \varphi$ by the Completeness Theorem. By definition, there is a proof $\varphi_0, \dots, \varphi_n$ of φ from Γ . Let $\Gamma_0 = \Gamma \cap \{\varphi_0, \dots, \varphi_n\}$. Then $\Gamma_0 \subseteq \Gamma$ is finite and $\Gamma_0 \vdash \varphi$, thus $\Gamma_0 \models \varphi$.
- (ii) We apply (i) with $\varphi = \perp$. $\square$

Definition 1.89. Let $\mathcal{A}$ and $\mathcal{B}$ be L -structures for some language L . An *isomorphism* of $\mathcal{A}$ to $\mathcal{B}$ is a bijection $f: A \rightarrow B$ such that

- $R^{\mathcal{A}}(a_0, \dots, a_{n-1}) \iff R^{\mathcal{B}}(f(a_0), \dots, f(a_{n-1}))$ for all n -ary relations $R \in L$ and $a_0, \dots, a_{n-1} \in A$;
- $F^{\mathcal{B}}(f(a_0), \dots, f(a_{n-1})) = f(F^{\mathcal{A}}(a_0, \dots, a_{n-1}))$ for n -ary functions $F \in L$ and $a_0, \dots, a_{n-1} \in A$.

We write $f: \mathcal{A} \simeq \mathcal{B}$ if f is an isomorphism of $\mathcal{A}$ to $\mathcal{B}$. We say $\mathcal{A}$ and $\mathcal{B}$ are *isomorphic*, written $\mathcal{A} \simeq \mathcal{B}$, if there exists an isomorphism $f: \mathcal{A} \simeq \mathcal{B}$.

Lemma 1.90. Let $f: \mathcal{A} \simeq \mathcal{B}$ and $a_0, \dots, a_{n-1} \in A$. Write $f(\vec{a})$ for the tuple $f(a_0), \dots, f(a_{n-1})$.

- (i) $t^{\mathcal{B}}(f(\vec{a})) = f(t^{\mathcal{A}}(\vec{a}))$ for each term $t(\vec{x})$.
- (ii) $\mathcal{A} \models \varphi(\vec{a}) \iff \mathcal{B} \models \varphi(f(\vec{a}))$ for each formula $\varphi(\vec{x})$.

Proof. By induction on the complexity of t and φ . □

Definition 1.91. If $\mathcal{A}$ is an L -structure, the (complete) theory of $\mathcal{A}$ is $\text{Th}(\mathcal{A}) = \{\varphi : \mathcal{A} \models \varphi\}$.

The standard model of arithmetic is $\mathbb{N} = \langle \mathbb{N}, 0, 1, +, \cdot, < \rangle$; its theory $\text{Th}(\mathbb{N})$ is called the *true arithmetic*. Models of arithmetic not isomorphic to $\mathbb{N}$ are called *nonstandard*.

Example 1.92. There exists a nonstandard model of true arithmetic. We can prove this using the compactness theorem as follows. We extend the language of arithmetic L to $L' = L \cup \{c\}$ and put

$$T = \text{Th}(\mathbb{N}) \cup \{c > \bar{n} : n \in \mathbb{N}\}, \quad \text{where } \bar{n} = \underbrace{1 + \dots + 1}_n.$$

Every finite $T_0 \subseteq T$ has a model: if n is the largest such that $c > \bar{n}$ occurs in T_0 , then $\langle \mathbb{N}, n+1 \rangle \models T_0$. The compactness theorem then implies T has a model $\mathcal{M}$.

$\mathcal{M}$ is a model of $\text{Th}(\mathbb{N})$, not isomorphic to $\mathbb{N}$. Note that $\mathbb{N}$ is embedded in $\mathcal{M}$ as an initial segment via the inclusion $n \in \mathbb{N} \mapsto \bar{n}^{\mathcal{M}}$; these are called the *standard* elements of $\mathcal{M}$, and we will pretend that $\mathbb{N}$ is outright a subset of $\mathcal{M}$. The axioms of T ensure that $c^{\mathcal{M}}$ is a nonstandard element.

The full structure of $\mathcal{M}$ is very complicated, but we can at least understand how $\langle \mathcal{M}, < \rangle$ looks like:

The ordering of $\mathcal{M}$ is discrete, thus each nonstandard element $a \in \mathcal{M}$ belongs to a convex subset $\{a + n : n \in \mathbb{Z}\}$ order-isomorphic to $\mathbb{Z}$. These are equivalence classes of the equivalence relation $a \sim b \iff |a - b| \in \mathbb{N}$. The induced order on these equivalence classes, i.e., the quotient structure $\langle \mathcal{M}, < \rangle / \sim$, is a dense linear order with a least element $\mathbb{N}$ and without a largest element: e.g., if a and b belong to different classes, then the class of $\lfloor (a + b)/2 \rfloor$ is strictly between the classes of a and b .

The compactness theorem is also useful for proving undefinability results:

Example 1.93. Let $L = \{P(x)\}$. There is no L -sentence φ such that for every finite L -structure $\mathcal{M}$,

$$\mathcal{M} \models \varphi \iff |P^{\mathcal{M}}| > |\mathcal{M} \setminus P^{\mathcal{M}}|.$$

Assume for contradiction that φ is such a sentence, and define

$$T = \left\{ \exists \vec{x} \left(\bigwedge_{i < j < n} x_i \neq x_j \wedge \bigwedge_{i < n} P(x_i) \right), \exists \vec{x} \left(\bigwedge_{i < j < n} x_i \neq x_j \wedge \bigwedge_{i < n} \neg P(x_i) \right) \right\}_{n \in \mathbb{N}}.$$

Every finite $T_0 \subseteq T$ is consistent with φ and with $\neg\varphi$. By compactness, both $T \cup \{\varphi\}$ and $T \cup \{\neg\varphi\}$ have models: $\mathcal{A} \models T \cup \{\varphi\}$ and $\mathcal{B} \models T \cup \{\neg\varphi\}$. We may assume both $\mathcal{A}$ and $\mathcal{B}$ to be countable by the downward Löwenheim–Skolem theorem.

Then writing $\mathcal{A} = \langle A, P^{\mathcal{A}} \rangle$, we have that $A, P^{\mathcal{A}}, A \setminus P^{\mathcal{A}}$ are countably infinite and the same for $\mathcal{B}$. Thus, $\mathcal{A}$ is isomorphic to $\mathcal{B}$, but $\mathcal{A} \models \varphi$, $\mathcal{B} \not\models \varphi$. This is a contradiction.

Example 1.94. No sentence can define the class of connected graphs.

We can usefully generalize the construction in Example 1.92 to all first-order theories:

Theorem 1.95 (Löwenheim–Skolem theorem). Let T be an L -theory and $\kappa \geq |L|$ an infinite cardinal. Let us assume that T either has an infinite model, or it has arbitrarily large⁴ finite models. Then T has a model of cardinality κ .

⁴I.e., for every $n \in \mathbb{N}$, T has a model of cardinality at least n .

Proof. The basic idea is to employ κ many constants to ensure that any model has size $\geq \kappa$, and apply the downward L-S theorem.

Let $L_\kappa = L \cup \{c_\alpha : \alpha < \kappa\}$ be new constants and $T_\kappa = T \cup \{c_\alpha \neq c_\beta : \alpha < \beta < \kappa\}$. Let us check that every finite $T' \subseteq T_\kappa$ has a model. T' only mentions the new constants c_α for $\alpha \in I$, where $I \subseteq \kappa$ is finite. Let $\mathcal{A} \models T$ be such that $|A| \geq |I|$; pick distinct $c_\alpha^A \in A$ for $\alpha \in I$, and pick arbitrary $c_\alpha^A \in A$ for $\alpha \notin I$. Then

$$\langle \mathcal{A}, c_\alpha^A : \alpha < \kappa \rangle \models T'.$$

By compactness, T_κ has a model $\mathcal{A}$. Since $|L_\kappa| = \kappa$, we may assume $|A| \leq \kappa$ by the downward Löwenheim–Skolem theorem. Because $\mathcal{A} \models T_\kappa$, the interpretations $\{c_\alpha^A : \alpha < \kappa\}$ are pairwise distinct. Thus, $|A| \geq \kappa$, whence $|A| = \kappa$. $\square$

In Example 1.92, we constructed a nonstandard model of arithmetic. The Löwenheim–Skolem theorem tells us that there are such models of arbitrary cardinality.

We are also going to extract a useful general statement from the argument in Example 1.93:

Definition 1.96. Let T be an L -theory and $\kappa \geq \aleph_0$ a cardinal. Then T is κ -categorical if all $\mathcal{A} \models T$ of cardinality κ are isomorphic.

Theorem 1.97 (Vaught’s test). *Let $\kappa \geq |L|$ be an infinite cardinal. If T is a κ -categorical L -theory without finite models, then T is complete.*

Proof. Assume for contradiction that T is not complete, and fix a sentence φ such that $T \cup \{\varphi\}$ and $T \cup \{\neg\varphi\}$ are consistent. Then there exist models $\mathcal{A} \models T \cup \{\varphi\}$, $\mathcal{B} \models T \cup \{\neg\varphi\}$, $|A| = |B| = \kappa$ by the Löwenheim–Skolem theorem. We have $\mathcal{A} \models \varphi$ and $\mathcal{B} \not\models \varphi$, thus $\mathcal{A}$ and $\mathcal{B}$ are not isomorphic. $\square$

Example 1.98. The theory DLO of dense linear orders without endpoints has language $L = \{<\}$ and the following axioms:

$$\begin{aligned} x &\not< x, \\ x < y \wedge y < z &\rightarrow x < z, \\ x < y \vee x = y &\vee y < x, \\ \forall x \forall y (x < y &\rightarrow \exists z (x < z \wedge z < y)), \\ \forall x \exists y x &< y, \\ \forall x \exists y y &< x. \end{aligned}$$

This theory is $\aleph_0$ -categorical, hence complete. This can be shown by a back-and-forth argument: given two countable models $\mathcal{A} = \langle A, < \rangle$ and $\mathcal{B} = \langle B, < \rangle$ of DLO, we enumerate them as $A = \{a_n : n \in \mathbb{N}\}$ and $B = \{b_n : n \in \mathbb{N}\}$. We construct an isomorphism between them as a union of a chain of finite partial isomorphisms, starting from the empty partial mapping, and extending it with one element at a time, alternately adding the first unused element from $\{a_n : n \in \mathbb{N}\}$ to the domain, or the first unused element from $\{b_n : n \in \mathbb{N}\}$ to the codomain. We use density and the other axioms of DLO to make sure that a suitable image or preimage exists for each element as it is being added. We leave the details to the reader (who might have very well seen this famous argument before, anyway).

Part 2

Computability

We wish to formalize the notion of an effective algorithm. There are several motivations for this:

First, it is intrinsically interesting as effective computability seems to be a fundamental concept for which we would like to have a formal counterpart.

Second, it allows us to mathematically formulate and answer questions about computability of particular problems or about general properties of computable problems. If a problem *is* computable, we can show this just by exhibiting an algorithm, for which an intuitive understanding of the concept suffices. However, if we want to prove that some problem is *not* computable, we need a precise definition so that we can argue about the collection of all algorithms.

A specific problem suggested by Part 1 is the so-called *Entscheidungsproblem*¹:

Is there an algorithm that decides whether a given first-order sentence φ is logically valid?

We know that validity of propositional formulas is algorithmically decidable by trying all assignments (see also Remark 1.25). For first-order sentences, we have a “one-sided” algorithm: we may systematically enumerate all possible proofs. The completeness theorem ensures that if a sentence *is* valid, we will (in principle) find its proof sooner or later; however, if a sentence is *not* valid, this algorithm will run forever and never halt. We will eventually show that the Entscheidungsproblem is not decidable; for that, we will need to develop a formal definition of algorithms.

Third, effective algorithms and related concepts are an important tool for investigation of first-order theories of arithmetic, as we will see in Part 3.

2.1 Turing machines

Many different formal *models of computation* have been proposed:

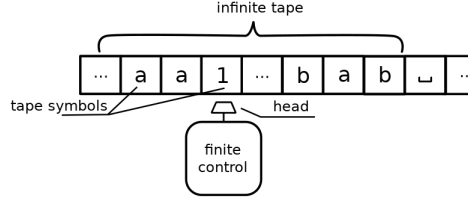
- Turing machines
- General recursive functions
- λ -calculus
- Random-access machines
- ...

However, all of these turn out to be equivalent. This leads to the so-called *Church–Turing thesis*, which posits that a problem is effectively computable in the informal sense iff it is computable by a Turing machine.

Turing machines are what we will use as our formal model. Intuitively speaking, this is an abstract model of a simple physical device consisting of an internal logic circuitry that can be in finitely many

¹Which literally just means “decision problem” in German, but it was borrowed to English with a more specific meaning.

states, with access to a *tape* divided into discrete *cells*, each of which can hold one symbol. The tape provides the machine with input, and it is subsequently used as a working memory. We assume the tape has a beginning, but it is (at least potentially) infinite in the other direction, and the machine, therefore, may not run out of memory to write into. The machine can scan one cell of the tape using a reading-writing *head* that can move along the tape.



What the machine does at any given moment is determined by its current state and the input symbol it is scanning at that moment: the machine can write a new symbol on the tape, switch to a different state, and move left or right², as specified by the *transition function*. There are three special states: the *initial state* that the machine is in when the computation starts, and the *accepting* and *rejecting* states that terminate the computation, indicating a YES/NO answer.

The input of the machine is a string of symbols (e.g., digits, letters, or other symbols) from the *input alphabet*; for convenience, the machine may write on the tape symbols from a larger alphabet called the *tape alphabet* during the computation. The tape alphabet also includes the *blank symbol* $\sqcup$ that denotes “empty” cells; this is the only symbol that may occur infinitely many times on the tape. When the computation begins, all cells after the actual input string contain the blank symbol.

The formal definition follows:

Definition 2.1. A *Turing machine* is a septuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$ where

- the set of states Q is a finite set that contains the *initial state* q_0 , the *accepting state* q_{acc} , and the *rejecting state* q_{rej} such that $q_{\text{acc}} \neq q_{\text{rej}}$;
- the *input alphabet* Σ is a finite nonempty set such that $\sqcup \notin \Sigma$;
- the *tape alphabet* Γ is a finite set such that $\Gamma \supseteq \Sigma \cup \{\sqcup\}$;
- and the *transition function* is a function $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.

This defines a Turing machine as a syntactic object, but what we really need is to define how it computes.

Definition 2.2. A *configuration* of a Turing machine M is $\langle q, h, u \rangle$ where $q \in Q$ is the *current state*, $h \in \omega$ is the *head position*, and $u \in \Gamma^\omega$ is the *tape content*. We denote the i th cell of u as u_i . A configuration $\langle q, h, u \rangle$ is *accepting* if $q = q_{\text{acc}}$; *rejecting* if $q = q_{\text{rej}}$; and *halting* if it is accepting or rejecting.

The *initial configuration* corresponding to an input $x \in \Sigma^*$ is $\langle q_0, 0, x \sqcup^\omega \rangle$ (i.e., the machine is in the initial state, and the head is at the beginning of the tape, whose content is the input string followed by infinitely many blanks).

A nonhalting configuration $\langle q, h, u \rangle$ yields a configuration $\langle q', h', u' \rangle$ defined as follows. Let $a = u_h$ be the current symbol, and $\delta(q, a) = \langle q', a', t \rangle$. Then $h' \in \omega$ and $u' \in \Gamma^\omega$ are defined by

$$h' = \begin{cases} h + 1 & \text{if } t = R, \\ \max\{h - 1, 0\} & \text{if } t = L, \end{cases} \quad u'_i = \begin{cases} a' & \text{for } i = h, \\ u_i & \text{for } i \in \omega, i \neq h. \end{cases}$$

²The machine may not stay put in place. We could allow that, but a moment's reflection shows that it would be pointless: if the head does not move, the output of the transition function (the new state and new symbol written on the tape) completely determines what happens next. Thus, we can just keep applying the transition function until the head *does* move (or the computation terminates or enters an infinite loop); i.e., we can modify the machine by collapsing steps when the head does not move with the following step. However, this argument does not work for multi-tape Turing machines (see below), hence their heads *are* allowed not to move.

We observe that any nonhalting configuration yields exactly one new configuration; we view this as performing one step of the computation.

Definition 2.3. A *run* of a Turing machine M on input $x \in \Sigma^*$ is a sequence of configurations $C_0, \dots, C_t$ where C_0 is the initial configuration on input x , and C_i yields C_{i+1} for each $i < t$.

M *accepts*, resp. *rejects*, x , if there is a run $C_0, \dots, C_t$ of M on x where C_t is an accepting, resp. rejecting, configuration.

Remark 2.4. As can be seen from the definitions above, the values of the transition function $\delta(q, a)$ for $q \in \{q_{\text{acc}}, q_{\text{rej}}\}$ are irrelevant, as the machine always halts in such states anyway. Thus, we could have defined δ as only a function $(Q \setminus \{q_{\text{acc}}, q_{\text{rej}}\}) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$. We keep the domain to be all of Q for consistency with Sipser's book.

Definition 2.5. A *decision problem* (or *language*; not to be confused with first-order languages) is any subset $L \subseteq \Sigma^*$. That is to say, it is a collection of possible inputs for a Turing machine.

For historical reasons, computability is replete with lots of parallel terminology (even the field itself was rebranded from *recursion theory* to *computability* in recent decades). We could of course choose a particular set of terms and stick to it, but it is important to at least be aware of the synonyms as they are all used in common sources.

Definition 2.6 (Decidability). A Turing machine M is said to *decide*, or *compute*, a decision problem L if for every input $x \in \Sigma^*$:

$$x \in L \implies M \text{ accepts } x; \quad x \notin L \implies M \text{ rejects } x.$$

A decision problem L is *decidable* (or *computable*, or *recursive*) if there exists a Turing machine M that decides L .

Definition 2.7 (Semidecidability). A Turing machine M is said to *recognize* (or *semidecide*) a decision problem L if for every input $x \in \Sigma^*$ we have

$$x \in L \iff M \text{ accepts } x.$$

L is said to be *semidecidable* (or *recognizable*, *computably enumerable*, *recursively enumerable*, or *partially decidable*, abbreviated *c.e.* or *r.e.*) if L is recognized by some Turing machine M . The *language of M* is

$$L(M) = \{x \in \Sigma^* : M \text{ accepts } x\}.$$

Remark 2.8.

- Every Turing machine recognizes exactly one language, viz. $L(M)$. That is, a language L is recognized by a Turing machine M iff $L = L(M)$.
- M decides L iff M recognizes L (i.e., $L = L(M)$) and M halts on every input $x \in \Sigma^*$.

That is, the difference between recognizing and deciding a language is that a decider must reject every input $x \notin L$ in finitely many steps, whereas a recognizer may run forever on such inputs and never halt.

Observation 2.9. Every decidable language is semidecidable. □

Decision problems formalize the notion of computational tasks that admit a YES/NO answer. However, not all problems we might consider computing by an algorithm are of this kind. We will also work with more general problems where the solution can be an arbitrary string; these are called *function problems*³:

³Even more generally, we could consider problems that admit more than one valid solution; these are called *search problems*, and are important in computational complexity, but we will not see them in this course.

Definition 2.10. A *partial function* $f: X \rightarrow Y$ is a function $f: X' \rightarrow Y$ where $X' \subseteq X$; i.e., f is possibly defined only on a portion of X and not necessarily everywhere. The *domain* of f is $\text{dom}(f) = X'$. In this context, a function is said to be *total* if $\text{dom}(f) = X$; i.e., if f obeys the usual definition of a function $X \rightarrow Y$.

A *function problem* is partial function $f: \Sigma^* \rightarrow \Sigma^*$.

Definition 2.11 (Function-problem computation). A Turing machine M is said to *output* $y \in \Sigma^*$ on input $x \in \Sigma^*$ if there is an accepting run $C_0, \dots, C_t$ of M on input x such that $C_t = \langle q_{\text{acc}}, h, y \cup \sqcup^\omega \rangle$.

M *computes* a function problem $f: \Sigma^* \rightarrow \Sigma^*$ if $L(M) = \text{dom}(f)$ and for each $x \in \text{dom}(f)$, M outputs $f(x)$ on input x .

$f: \Sigma^* \rightarrow \Sigma^*$ is a *partial computable function* (or *partial recursive function*) if there is a Turing machine M that computes it. A partial computable function f that is total (i.e., defined everywhere on Σ^*) is called simply a *computable function* (or *recursive function*).

Example 2.12. Consider the language

$$\text{PALINDROMES} = \{w \in \{a, b\}^* : w = w^R\},$$

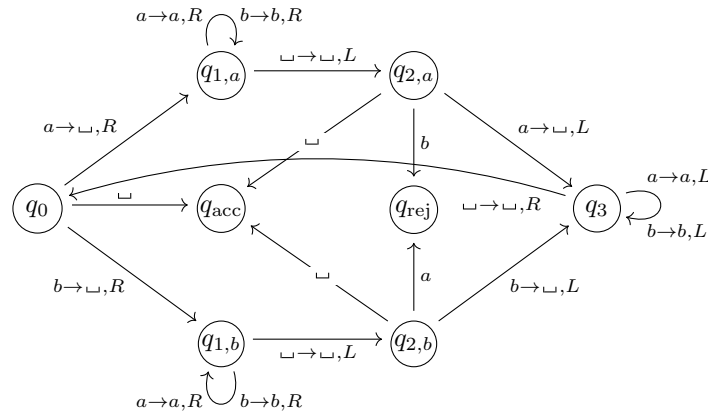
where the string reversal operator R is defined by $(w_0 \dots w_{n-1})^R = w_{n-1} \dots w_0$. In other words, the language consists of words over a 2-letter alphabet $\{a, b\}$ that read the same in both directions; e.g., bab or $abba$. We will now design a Turing machine that decides whether a given word is a palindrome.

A simple algorithm is to repeatedly check that both symbols at the ends of the string are the same and cross them out, until we either detect an inconsistency or end up with a string of length ≤ 1 . A Turing machine cannot operate at both ends simultaneously, but we can achieve something similar by moving the head back and forth: the machine removes the left-most symbol and “remembers” it in its internal state, moves to the right end, checks that the last symbol agrees with the remembered one and removes it, and rewinds back to the left.

Formally, we define the machine as

$$\begin{aligned} M &= \langle Q, \{a, b\}, \{a, b, \sqcup\}, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle, \\ Q &= \{q_0, q_{\text{acc}}, q_{\text{rej}}, q_{1,a}, q_{1,b}, q_{2,a}, q_{2,b}, q_3\}, \end{aligned}$$

where the transition function δ is given by the following diagram:



Remark 2.13. Programming Turing machines down to an explicit listing of the transition function table can be a tedious endeavour that requires a lot of determination and patience, while the result is not very illuminating and obscures the ideas behind the algorithm. The purpose of Example 2.12 is to present during the lecture at least once a complete Turing machine with all the bells and whistles that computes something sensible to show that it can be done, but from now on we will rather describe Turing machines using an informal pseudo-code, assuming that the reader can imagine how to translate it to a formal presentation if required.

If desired—to get a better feeling for what can be implemented on Turing machines and how, or just for fun—there are a number of online Turing machines simulators one can play with, e.g., <https://turingmachinesimulator.com>. The palindrome machine from Example 2.12 can be found at <https://turingmachinesimulator.com/shared/slylqbjruc>.

Remark 2.14. One can find many variant definitions of Turing machines in the literature. Some of the common modifications include:

- A two-sided infinite tape, or different conventions for handling attempts to move past the end of the tape.
- Different halting conditions: e.g., using a partial transition function (the machine halts when δ is undefined in the current configuration), or allowing more than one accepting/rejecting state.
- Restrict the tape alphabet to be just $\Sigma \cup \{\sqcup\}$.
- *Multi-tape Turing machines*: the machine has k tapes (where k is a fixed number that's part of the specification of the machine) including an *input tape* (usually read-only), several *work tapes*, and—if we are interested in function problems—an *output tape* (usually write-only); each tape has its own head that can move (or stay put) independently.
- *Nondeterministic, alternating, randomized, or quantum machines*: these are considered to be different models of computation rather than simple variants. They generally yield the same class of computable problems as ordinary Turing machines, but they may be able to solve some problems more efficiently, whence they are important in computational complexity.

Lemma 2.15. *Any problem computable on a k -tape Turing machine is computable on a single-tape Turing machine.*

Proof sketch. Represent the content of all k tapes (including markers indicating head positions) on one tape using a new tape alphabet $(\Gamma \times \{0, 1\})^k \cup \{\sqcup\}$. To simulate one step of the original machine, sweep the tape to locate the head positions of the simulated tapes and read the symbols at these positions, remembering them in the state of the new machine. When this information is collected, the machine knows what to do in the next step (write new symbols, move tape heads); it traverses the tape again to implement these changes. $\square$

We remark that even though multi-tape Turing machines are equivalent to single-tape machines in the sense above, it is often easier or more efficient to solve a problem on a multi-tape machine. For example, to decide PALINDROMES on a two-tape Turing machine, we can simply copy the string to a work tape and then traverse the two copies in opposite directions. Multi-tape Turing machines are the standard model of computation in computational complexity as they tend to correspond more closely in terms of efficiency to an informal notion of algorithms.

Multi-tape Turing machines are also sometimes handy for simplifying proofs, such as the next important lemma:

Lemma 2.16. *A language $L \subseteq \Sigma^*$ is decidable iff L and $\Sigma^* \setminus L$ are semidecidable.*

Proof.

$\Rightarrow$ Suppose L is decidable; then so is $\Sigma^* \setminus L$ (we may take a Turing machine deciding L and swap the accepting and rejecting states). It follows $\Sigma^* \setminus L$ is semidecidable, while L itself is semidecidable trivially.

$\Leftarrow$ Let M_0 recognize (semidecide) L , and M_1 recognize $\Sigma^* \setminus L$. A 2-tape Turing machine M described by the following pseudo-code decides L :

- (1) Copy the input onto the second tape.
- (2) Run M_0 and M_1 in parallel on the two tapes.
- (3) If M_0 accepts, then ACCEPT. If M_1 accepts, then REJECT.

Observe that M has to eventually halt by definition of language recognition: any input x belongs to either Σ^* or its complement $\Sigma^* \setminus L$. Hence x is accepted by exactly one of M_0 and M_1 , and the algorithm halts in finite time with the correct response. $\square$

Remark 2.17. Let us convince ourselves that any Turing machine with input alphabet of size $|\Sigma| \geq 2$ can be simulated by one with tape alphabet $\Gamma = \Sigma \cup \{\sqcup\}$. Given a machine

$$M = \langle \Sigma, \Gamma, Q, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle,$$

we fix k such that $|\Gamma| \leq (|\Sigma| + 1)^k$, and an injective encoding $e: \Gamma \rightarrow (\Sigma \cup \{\sqcup\})^k$. We may assume $e(\sqcup) = \sqcup^k$. We simulate M using k -tuples of symbols from $\Sigma \cup \{\sqcup\}$ to represent each symbol on the tape using e . To simulate one step, we read and remember the k -tuple of symbols to the right of the head position, and apply the original transition function: we write a new k -tuple of symbols, and move the head k positions to the left or to the right.

We need to expand the input to the tape encoding at the beginning of the simulation. This may be done one cell at a time: we take a symbol a that had not been encoded yet, shift the content of the tape to the right of the symbol by $k - 1$ positions to make room, go back to write the encoding $e(a)$, and repeat until we get to the end of the original input. If we care about the output string, we have to similarly decode it after the simulation halts.

Remark 2.18. We introduced computability of problems (sets, functions) on finite strings, but in other accounts, the primary definition of computability is often stated for problems on nonnegative integers (in particular, this is standard in the set-up of general recursive functions). We want to consider such a definition as well: besides the fact that there are various intrinsically interesting computational problems dealing with integers, we will need it as a tool for investigation of first-order theories of arithmetic in the third part of this course.

We can accommodate integers in our set-up by encoding them with suitable finite strings (as we can do with other objects that we might want to compute with, such as finite graphs). Perhaps the simplest encoding is the *unary* representation, where $n \in \mathbb{N}$ is represented by the string $1^n := \underbrace{11 \dots 1}_n$ (possibly

using other symbol in place of 1). This requires a string of length n . A more efficient representation is to express integers in *binary*, *decimal*, or more generally, *base- k positional notation* for some $k \geq 2$: $n \in \mathbb{N}$ is written as $a_{t-1} \dots a_0 \in \{0, \dots, k-1\}^t$, where $n = \sum_{i < t} a_i k^i$. The length of this representation is $\lceil \log_k(n+1) \rceil$.

While this may be the most natural choice, it has the drawback that distinct strings may represent the same integer due to leading 0s: e.g., $000101 = 101$. For some purposes it may be more convenient to use a representation that is a *bijection* between $\mathbb{N}$ and Σ^* : e.g., when we will arithmetize Turing machines and their computation, we will need to assign each string a distinct number. An elegant way to accomplish this is to use the following variant of the positional notation:

Definition 2.19. Let $k \geq 1$, and assume Σ is the finite alphabet $\{1, 2, 3, \dots, k\}$. The *bijective base- k numeration* (also called *dyadic* in the case $k = 2$) is the following encoding:

$$\lceil - \rceil: \Sigma^* \rightarrow \mathbb{N}, \quad \lceil a_0 \dots a_{t-1} \rceil = \sum_{i < t} a_i k^i.$$

(Up to the choice of writing direction this is the same formula as for normal base- k notation, but crucially, we use digits $\{1, \dots, k\}$ rather than $\{0, \dots, k-1\}$. Note that for $k = 1$, we obtain the unary encoding described above.)

We also call $\lceil w \rceil$ the *Gödel*⁴ number of w .

Lemma 2.20. For any $k \geq 1$, $\lceil - \rceil$ is a bijection between $\mathbb{N}$ and $\{1, \dots, k\}^*$.

Proof. Exercise. $\square$

⁴This is not really the encoding originally used by Gödel. In general, this term refers to any encoding of finite objects (e.g., graphs, Turing machines, ...) by natural numbers.

We also note that the dyadic representation of n can be constructed by taking the binary representation of $n + 1$, stripping the leading digit 1, and adding 1 to each remaining digit.

Definition 2.21. We say that $L \subseteq \mathbb{N}$ is *(semi)decidable* if its dyadic encoding $\{w \in \{1, 2\}^* : \ulcorner w \urcorner \in L\}$ is (semi)decidable.

We say $F: \mathbb{N} \rightarrow \mathbb{N}$ is *computable* if $G: \{1, 2\}^* \rightarrow \{1, 2\}^*$ is computable, where G is uniquely determined by $F(\ulcorner w \urcorner) = \ulcorner G(w) \urcorner$. In other words, G is given by the following commutative diagram:

$$\begin{array}{ccc} \Sigma^* & \xrightarrow{\ulcorner - \urcorner} & \mathbb{N} \\ \downarrow G & \circlearrowright & \downarrow F \\ \Sigma^* & \xrightarrow{\ulcorner - \urcorner} & \mathbb{N} \end{array}$$

Exercise 25 shows that the choice of representation of natural numbers does not actually affect what sets or functions on $\mathbb{N}$ are computable.

It is also convenient to extend the definition of computability to k -ary relations and functions:

Definition 2.22. We say $R \subseteq (\Sigma^*)^k$ is *(semi-)decidable* if

$$\{w_0 \# w_1 \# \cdots \# w_{k-1} : \langle w_0, \dots, w_{k-1} \rangle \in R\} \subseteq (\Sigma \cup \{\#\})^*$$

is (semi-)decidable, where $\# \notin \Sigma$ is a new separator symbol.

Similarly, $F: (\Sigma^*)^k \rightarrow \Sigma^*$ is *computable* if the partial function $G: (\Sigma \cup \{\#\})^* \rightarrow \Sigma^*$ such that $G(w_0 \# w_1 \# \cdots \# w_{k-1}) = F(w_0, \dots, w_{k-1})$ is computable.

By combining this with Definition 2.21, we can also define (semi-)decidability of relations $R \subseteq \mathbb{N}^k$ and computability of partial functions $f: \mathbb{N}^k \rightarrow \mathbb{N}$.

We remark that a more elegant way of defining computability of k -ary relations and functions might be to use a variant of multi-tape Turing machines with k input tapes.

2.2 Universal Turing machines and the halting problem

We don't need a different computer for every task that we want to get done: we just need one *universal* computer that can run arbitrary programs that it reads as data. This paradigm applies to Turing machines, too.

Warning. While we continue to use angle brackets $\langle - \rangle$ to denote finite sequences, in computability theory they are also commonly used for strings encoding finite objects (analogues of Gödel numbers, but with the result being a string rather than a number). In particular, they will denote strings encoding Turing machines.

Theorem 2.23. Let Σ be an alphabet with at least two symbols. Then there exists a universal Turing machine U_Σ with the following property:

For every Turing machine M on the same alphabet Σ , there is a code $\langle M \rangle \in \Sigma^*$ such that

$$U_\Sigma(\langle M \rangle x) \simeq M(x) \quad \text{for each } x \in \Sigma^*.$$

Here, “ $\simeq$ ” means that U_Σ accepts $\langle M \rangle x$ iff M accepts x , and it rejects $\langle M \rangle x$ iff M rejects x . Moreover, U_Σ outputs $y \in \Sigma^*$ on input $\langle M \rangle x$ iff M outputs y on input x .

Proof. Let $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}} \rangle$ be a Turing machine on the alphabet Σ , where we assume $\Gamma = \Sigma \cup \{\sqcup\}$ (see Remark 2.17). We fix an enumeration $Q = \{q_i : i < s\}$, where q_0 the initial state as indicated above, $q_1 = q_{\text{acc}}$, and $q_2 = q_{\text{rej}}$. We also fix an enumeration $\Gamma = \{a_j : j < k\}$.

The code $\langle M \rangle$ of M will describe the transition function $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$. For convenience, we will use some auxiliary extra symbols ($\#$, $\#\#$, 0 , 1 , L , R) to define $\langle M \rangle$ and the operation of U_Σ (also, $\langle M \rangle$ may include blanks). Officially, U_Σ is required to have input alphabet Σ , and in particular, we should make $\langle M \rangle \in \Sigma^*$; we achieve this by encoding the expanded alphabet $\Sigma' = \Sigma \cup \{\sqcup, \#, \#\#, 0, 1, L, R\}$ by c -tuples of symbols from Σ for a suitable c , similarly to Remark 2.17. (This is where we use the assumption $|\Sigma| \geq 2$.)

We define

$$\begin{aligned} \langle M \rangle = & \#\# \langle \delta(q_0, a_0) \rangle \# \langle \delta(q_0, a_1) \rangle \# \cdots \# \langle \delta(q_0, a_{k-1}) \rangle \\ & \# \langle \delta(q_1, a_0) \rangle \# \langle \delta(q_1, a_1) \rangle \# \cdots \# \langle \delta(q_1, a_{k-1}) \rangle \\ & \vdots \\ & \# \langle \delta(q_{s-1}, a_0) \rangle \# \cdots \# \langle \delta(q_{s-1}, a_{k-1}) \rangle \#\# \end{aligned}$$

where if $\delta(q_i, a_j) = \langle q_{i'}, a_{j'}, t \rangle \in Q \times \Sigma \times \{L, R\}$, we define the encoding of $\delta(q_i, a_j)$ as

$$\langle \delta(q_i, a_j) \rangle = a_{j'} t 0^{i'}.$$

Here, $0^{i'}$ denotes a string of zeroes of length i' .

We shall now describe the operation of the universal Turing machine U_Σ . It maintains on its tape a representation of the current configuration of M ; it works in an endless loop where on each iteration, it simulates the effects of one step of M . It will be convenient for this purpose to represent a configuration of M as

$$u_0 \dots u_{h-1} \langle M \rangle u_h u_{h+1} \dots$$

where $u_0 \dots u_{h-1}$ is the content of the simulated tape to the left of the head position h , and $u_h u_{h+1} \dots$ the rest of the tape from the head position onward. During the simulation, some parts of $\langle M \rangle$ will be modified a little; we will still refer to it as $\langle M \rangle$. In particular, we need to indicate the current state q_i of M : we do this by replacing the $\#$ in front of the entry $\langle \delta(q_i, a_0) \rangle$ of the encoded transition function table (i.e., the beginning of the row of the table corresponding to q_i) with the symbol $\#\#$.

Note that the encoding of Σ' by Σ^c is applied only to the $\langle M \rangle$ part of the configurations; the symbols u_i of the simulated tape will be written literally.

During the simulation, the head will be kept inside $\langle M \rangle$, except possibly venturing one step outside to read/write the current symbol of the simulated machine, or to move the simulated head. We can rewind the tape to the left or right end of $\langle M \rangle$ at any time, because these can be recognized by the substring $\#\#$ (or $\#\#\#$); this works even after the encoding of Σ' by Σ^c because we never go far away from $\langle M \rangle$, and therefore we cannot lose track of whether we are currently seeing an encoded symbol of Σ' or an unencoded symbol of Σ .

Let us now describe the operation of U_Σ using pseudo-code. In the beginning of the simulation, U_Σ starts with $\langle M \rangle x$ on the input tape, which is almost a valid representation of the initial configuration of M on input x —we only need to mark the row of the transition function table corresponding to the initial state q_0 :

(00) Move right and replace the second $\#$ with $\#\#$.

Next comes the main loop of U_Σ , simulating one step of the computation of M . Suppose that the tape contains a representation of a configuration as above. We first have to locate the entry of the transition table corresponding to the current state q_i and the symbol a_j under the head of the simulated machine:

- (01) Move past the right end of $\langle M \rangle$.
- (02) Read and remember $u_h = a_j$.
- (03) Locate the $\#\#$ symbol, and replace it with $\#$.
- (04) Repeat j times: move right towards the next $\#$ symbol.

The head is now at the $\#$ symbol in front of the string $\langle \delta(q_i, a_j) \rangle = a_{j'} t 0^{i'}$.

- (05) Read and remember $a_{j'}$ and t .

We now have to mark the table row corresponding to the new state $q_{i'}$ with $\#$. Unlike $a_{j'}$ or t , we cannot just read i' and remember in the state of U_Σ , because the number of states of M may be arbitrarily large (it is not bounded by a constant). We proceed as follows: we convert the $0^{i'}$ in $\langle \delta(q_i, a_j) \rangle$ to $1^{i'}$, mark the row corresponding to q_0 , and then use a loop that converts the 1s back to 0s one by one, each time moving the marker to the next row.

However, we must take care to abort the simulation if the new state is halting. Note that we *can* count up to 2 in the state of U_Σ , thus we can check if $i' = 1$ or $i' = 2$, even if we cannot remember an arbitrarily large i' . (This is the reason we fixed the accepting and rejecting states to be q_1 and q_2 , resp.)

- (06) Move right to the next $\#$, replacing 0 with 1 as we go.
- (07) If the total number of 0s was 1, then ACCEPT.
- (08) If the total number of 0s was 2, then REJECT.
- (09) Locate the left end of $\langle M \rangle$.
- (10) Change the second $\#$ to $\#$.

Now comes the loop for moving the marker. Note that each row of the table has $k = |\Gamma| = |\Sigma| + 1$ entries, which is a constant that we can count up to in the state of U_Σ .

- (11) Move right to locate the first 1.
- (12) If none is found before the end of $\langle M \rangle$, go to (18).
- (13) Change the 1 to 0.
- (14) Locate the $\#$ symbol and change it to $\#$.
- (15) Move to the k th $\#$ to the right.
- (16) Change it to $\#$.
- (17) Move to the beginning of $\langle M \rangle$, and go to (11).

We have now placed $\#$ correctly to mark the new state. We have yet to update the symbol under the simulated head, and move the head. Note that upon exiting the loop above, the head of U_Σ is past the right end of $\langle M \rangle$, i.e., at the position of u_h .

- (18) If $t = R$:
- (19) Shift $\langle M \rangle$ to the right (overwriting u_h).
- (20) Write $a_{j'}$ in the free space to the left of it.
- (21) If $t = L$:
- (22) Write $a_{j'}$.
- (23) Locate the left end of $\langle M \rangle$.
- (24) If it is at the beginning of the tape, go to (01).
- (25) Remember the symbol $u_{h-1} = a_{j''}$ to the left of it.
- (26) Shift $\langle M \rangle$ to the left (overwriting u_{h-1}).
- (27) Write $a_{j''}$ in the free space to the right.
- (28) Go to (01).

This finishes the simulation for decision problems. If we care about computation of functions, U_Σ cannot literally halt in step (07): it must first clean up the tape (i.e., remove $\langle M \rangle$ and shift $u_h u_{h+1} \dots$ accordingly) so that its output is the same as the output of M . $\square$

Definition 2.24. The *halting problem* for a given alphabet Σ is the language

$$A_\Sigma = \{ \langle M \rangle x : M \text{ accepts } x \} \subseteq \Sigma^*.$$

Theorem 2.25. The halting problem A_Σ is semidecidable, but not decidable.

Proof. A_Σ is recognized by the universal Turing machine U_Σ . To see it is not decidable, we assume towards a contradiction that A_Σ is decided by some Turing machine H . This means that H accepts the pair $\langle M \rangle x$ if M accepts x , and rejects it otherwise.

We define a new Turing machine D on the input alphabet Σ that works as follows:

- (1) Duplicate the input x to xx .
- (2) Run H .
- (3) If H accepts, then REJECT; if H rejects, then ACCEPT.

We run D on input $\langle D \rangle$, and obtain a contradiction:

$$\begin{aligned}
 D \text{ accepts } \langle D \rangle &\iff H \text{ rejects } \langle D \rangle \langle D \rangle && \text{definition of } D \\
 &\iff \langle D \rangle \langle D \rangle \notin A_\Sigma && H \text{ decides } A_\Sigma \\
 &\iff D \text{ does not accept } \langle D \rangle && \text{definition of } A_\Sigma. \quad \square
 \end{aligned}$$

Remark 2.26. D stands for “diagonal machine”, as the proof of Theorem 2.25 is a variant of Cantor’s diagonal argument. We imagine the infinite matrix indexed by strings where rows enumerate codes $\langle M \rangle$ of Turing machines, columns enumerate inputs x , and a 0/1 entry in the matrix indicates whether M accepts x or not. The machine D computes the diagonal of the matrix with 0/1 flipped, but this clearly cannot agree with any row of the matrix.

Remark 2.27. The undecidability of the halting problem is a fundamental result of the theory of computation. It implies the undecidability of many other problems concerning the behaviour of Turing machines by means of *reductions*: the general idea is that if we assume for contradiction that problem X is computable, we can use an algorithm solving X as a subprogram to build an algorithm that decides the halting problem (which is impossible). The full extent of the idea of “using A as a subprogram to compute B ” leads to so-called *Turing reductions*. We will not define them formally as this requires a modification of the Turing machine model; rather, we will introduce a more strict notion of *many-one reductions*, which is much easier to define.

It might seem that the halting problem is not robustly defined as it depends on the—rather arbitrary—encoding of Turing machines we devised in the proof of Theorem 2.23. One should not worry about this: in fact, all “reasonable” encodings of Turing machines yield halting problems of the same complexity (i.e., reducible to each other). Better yet, as we will see later in Part 3, we can use reductions from the halting problem to prove undecidability of “pre-existing” problems of independent interest that do not mention Turing machines at all, in particular, the Entscheidungsproblem. In this way, the undecidability of the halting problem is not just an important result on its own, but also a useful tool: once we identified *one* undecidable problem, we can show the undecidability of many other.

Definition 2.28. Let $A, B \subseteq \Sigma^*$. We say A is *many-one reducible* (or *mapping reducible*) to B , written $A \leq_m B$, if there is a computable $f: \Sigma^* \rightarrow \Sigma^*$ such that for all $x \in \Sigma^*$ we have

$$x \in A \iff f(x) \in B.$$

We say A and B are *many-one equivalent*, written $A \equiv_m B$, if $A \leq_m B$ and $B \leq_m A$.

Lemma 2.29. If $A \leq_m B$ and B is (semi)decidable, then A is (semi)decidable.

Proof. If M (semi)decides B , and M' computes f , then A is (semi)decided by the Turing machine that simulates M' to compute $f(x)$ and then simulates M . $\square$

Example 2.30. We defined the “halting problem” to be A_Σ in accordance with Sipser’s book, but it would be more logical to call A_Σ the “acceptance problem”, and reserve the name “halting problem” for $H_\Sigma = \{ \langle M \rangle x : M \text{ halts on input } x \}$ (which is indeed the traditional definition). Fortunately, this does not make a significant difference: it is a simple exercise to prove that $H_\Sigma \equiv_m A_\Sigma$.

2.3 Computability of logical syntax

We are particularly interested in computing problems associated to logical theories. If L is a finite first-order language, then L -terms, L -formulas, and other syntactic objects such as proofs can be manipulated by Turing machines as they are just finite strings. A minor issue is that as defined in Definition 1.45, formulas are strings over an *infinite* alphabet, as there are infinitely many variables. This is easy to fix: we may, e.g., denote variables v_n by strings of the form $v1001101$, where 1001101 is the representation of n in binary. It is then easy to show that basic features of logical syntax are computable:

Lemma 2.31. *The following sets and functions are computable for a fixed finite language L :*

- (i) *The set of L -terms.*
- (ii) *The set of L -formulas.*
- (iii) *$\{\langle \varphi, x \rangle : x \text{ is a free variable of a formula } \varphi\}$.*
- (iv) *The substitution function: given a formula φ , a variable x and a term t , compute $\varphi(t/x)$.*
- (v) *$\{\langle \Gamma, \varphi, \pi \rangle : \pi \text{ is a proof of } \varphi \in \text{Form}_L \text{ from a finite set } \Gamma \subseteq \text{Form}_L\}$.*

Proof. Exercise 28. □

The established terminology concerning decidability of first-order theories is a bit confusing; or rather, it reflects an alternative definition of theories that requires them to be *deductively closed* sets of sentences:

Definition 2.32. If T is a theory in a finite language L , then T is said to be *decidable* if $\text{Thm}(T) = \{\varphi : \varphi \text{ is an } L\text{-sentence}, T \vdash \varphi\}$ is a decidable set of strings.

A theory T in a finite language L is said to be *recursively axiomatized*, or *computably axiomatized*, if T —considered as a set of axioms (i.e., without the requirement of deductive closure)—is decidable. A theory T is *recursively* (or *computably*) *axiomatizable* if it is equivalent to a recursively axiomatized theory.

Lemma 2.33.

- (i) *Every recursively axiomatizable theory is semidecidable.*
- (ii) *Every complete, recursively axiomatizable theory is decidable.*

Proof.

(i): We may assume T is recursively axiomatized. Given a sentence φ , we exhaustively enumerate all pairs $\langle \Gamma, \pi \rangle$. We accept if π is a proof of φ from Γ and all $\psi \in \Gamma$ are in T .

(ii): $\text{Thm}(T)$ is semidecidable by (i). Then

$$\Sigma^* \setminus \text{Thm}(T) = \{\varphi : \varphi \text{ is not an } L\text{-sentence}\} \cup \{\varphi : T \vdash \neg\varphi\}$$

is also semidecidable and, therefore, $\text{Thm}(T)$ is decidable by Lemma 2.16. □

Remark 2.34. Conversely, every semidecidable theory is recursively axiomatizable; in particular, a theory is “semidecidably axiomatizable” iff it is recursively axiomatizable, hence there is no need to introduce this notion separately. This can be shown by so-called *Craig’s trick*, whose idea is that each sentence has many equivalent sentences that can be used to additionally encode arbitrary data. See Exercise 30. It is also good to mention that every consistent decidable theory has a complete consistent decidable extension (Exercise 31).

One more useful general observation is that adding finitely many axioms cannot increase the complexity of a theory; in particular, if an extension of a theory by finitely many axioms is undecidable, the theory itself is undecidable:

Lemma 2.35. *Let T be an L -theory, where L is a finite language, and X a finite set of L -sentences. Then $\text{Thm}(T + X) \leq_m \text{Thm}(T)$.*

Proof. $T + X \vdash \varphi$ iff $T \vdash (\alpha \rightarrow \varphi)$, where $\alpha = \bigwedge_{\psi \in X} \psi$. □

Part 3

Arithmetic

We are interested in this part in properties of the structure of natural numbers $\langle \mathbb{N}, +, \cdot, \dots \rangle$ and related theories that encompass elementary integer arithmetic:

- Can we present the theory of $\mathbb{N}$ by an effective proof system (say, as a recursively axiomatized first-order theory)?
- Can we decide whether a given number-theoretic statement (i.e., a first-order sentence over $\mathbb{N}$) is true?
- Can we prove the consistency of the theory of elementary arithmetic by finitary methods?

(Here, “finitary” means roughly that we are only allowed to reason with natural numbers, finite strings, and other finite objects, but not infinite sets. Of course, the theory of $\mathbb{N}$ is consistent as it has, by definition, a model. But this argument requires infinite sets such as $\mathbb{N}$ and the satisfaction relation on $\mathbb{N}$.) We will see that the answers to all of these are negative.

These questions, particularly the last one, arose in connection to the foundations of mathematics at the turn of the 19th/20th century. During the 19th century, the focus of mathematics shifted from the study of “concrete” objects such as natural and real numbers or geometric shapes towards more abstract reasoning, such as abstract algebra (group theory, field theory) and, especially, *set theory*. This opened a realm of very powerful methods that were, however, met with suspicion and skepticism of many mathematicians,¹ especially after the discovery of various *paradoxes* (such as Russell’s paradox) that showed that naïve set theory in the form initiated by Cantor was untenable, and put in doubt the consistency of infinitary methods at large.

Hilbert’s program sought to put the theory of abstract, infinitary mathematics on a firm footing by formally proving its consistency by proof-theoretic means in an uncontroversial finitary theory, such as a theory of arithmetic.

Proof theoretic analysis of foundational theories is not, on the whole, an unreasonable idea, and some progress had been made early on: notably, Presburger provided a complete axiomatization of the theory of $\langle \mathbb{N}, 0, 1, +, \leq \rangle$, and proved it consistent and decidable by proof-theoretic means; later, Tarski did the same for theories of $\langle \mathbb{R}, 0, 1, +, \cdot, \leq \rangle$ and elementary geometry. However, Hilbert’s program as such was all but killed by *Gödel’s theorems*, establishing that any effectively presented theory capable of expressing basic integer arithmetic is incomplete, and cannot even prove its *own* consistency, let alone the consistency of a considerably stronger theory.

3.1 Robinson and Peano arithmetics

The idea of an axiomatic definition of natural numbers ties in with the general axiomatic approach in mathematics, including axiomatic definitions of other basic structures such as $\langle \mathbb{R}, 0, 1, +, \cdot, \leq \rangle$ being the

¹Cf. the oft-quoted comment attributed to P. Gordan, “this is not mathematics; this is theology,” on Hilbert’s solution of Gordan’s problem in invariant theory, involving what we now know as Hilbert’s basis theorem.

unique completely ordered field up to isomorphism. An early axiomatic description of $\mathbb{N}$, which actually defines it uniquely up to isomorphism, is given by the *Dedekind–Peano axioms*. Up to some differences in terminology, they essentially postulate that there is a natural number $0 \in \mathbb{N}$ and a function $S: \mathbb{N} \rightarrow \mathbb{N}$ (the *successor* function, normally denoted $x + 1$) such that:

- (i) $\forall x, y \in \mathbb{N} (S(x) = S(y) \rightarrow x = y)$.
- (ii) $\forall x \in \mathbb{N} S(x) \neq 0$.
- (iii) Any set $X \subseteq \mathbb{N}$ that contains 0 and is closed under S equals $\mathbb{N}$.

While the first two axioms are straightforward, the last axiom (the *induction* principle) is not expressible by a first-order sentence as it quantifies over *subsets* of $\mathbb{N}$ rather than just its elements. (One can do this in *second-order* logic.) One might think we could amend this by considering a structure with domain $\mathcal{P}(\mathbb{N})$, but this does not really work either because for any theory in such a language, there is no way of enforcing that its model includes *all* subsets of $\mathbb{N}$ using first-order axioms.

After all, we know from Part 1 that any first-order theory of arithmetic will have nonstandard models of arbitrarily large cardinality, hence there is no way it could define $\mathbb{N}$ up to isomorphism; but we may still hope to capture all first-order sentences valid in $\mathbb{N}$ by a nice explicit set of natural axioms (as is possible for $\langle \mathbb{N}, 0, 1, +, \leq \rangle$ and $\langle \mathbb{R}, 0, 1, +, \cdot, \leq \rangle$). A natural attempt at such an axiomatization is *Peano arithmetic* that postulates the induction axiom for sets definable by first-order formulas, which seems to be all one can do in first-order logic.

Definition 3.1 (Robinson and Peano arithmetics). The *language of arithmetic*² is $L_{PA} = \{0, S, +, \cdot, \leq\}$. *Robinson's arithmetic* $\mathbf{Q}$ is the L_{PA} -theory with axioms

- (Q1) $S(x) = S(y) \rightarrow x = y$,
- (Q2) $S(x) \neq 0$,
- (Q3) $x \neq 0 \rightarrow \exists y S(y) = x$,
- (Q4) $x + 0 = x$,
- (Q5) $x + S(y) = S(x + y)$,
- (Q6) $x \cdot 0 = 0$,
- (Q7) $x \cdot S(y) = x \cdot y + x$,
- (Q8) $x \leq y \leftrightarrow \exists z z + x = y$.

Peano arithmetic $\mathbf{PA}$ is $\mathbf{Q}$ extended with the schema of induction

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

for all formulas φ . We allow φ to have other free variable besides x ; these are called the *parameters* of the induction axiom. Thus, more formally, the universal closure of

$$\varphi(0/x) \wedge \forall x (\varphi \rightarrow \varphi(S(x)/x)) \rightarrow \forall x \varphi$$

is an axiom of $\mathbf{PA}$ for every L_{PA} -formula φ .

Recall that the *standard model of arithmetic* is $\mathbb{N} = \langle \mathbb{N}, 0, 1, +, \cdot, < \rangle$; its theory $\mathbf{Th}(\mathbb{N})$ is called the *true arithmetic*. An L_{PA} -sentence φ is called *true* if $\mathbb{N} \models \varphi$, and *false* otherwise; an L_{PA} -theory T is called *sound* if $\mathbb{N} \models T$.

Observation 3.2. *PA is sound.* □

²The Dedekind–Peano axioms could make do with just 0 and S as they define $\mathbb{N}$ up to isomorphism; common arithmetical operations such as $+$, $\cdot$ are then definable using second-order recursion. However, this is impossible in first-order logic—the first-order theory of $\langle \mathbb{N}, 0, S \rangle$ is expressively very poor. We thus need to explicitly include $+$ and $\cdot$ in the language.

PA is a first-order version of the Dedekind–Peano axioms, and may look as a plausible candidate for a complete axiomatization of true arithmetic. However, we will prove that it is in fact incomplete, and it cannot be made complete by adding any semidecidable set of axioms; furthermore, this holds already for extensions of the rudimentary theory $\mathbf{Q}$. This is the content of Gödel’s first incompleteness theorem.

In contrast to PA, Robinson’s arithmetic $\mathbf{Q}$ is a very weak base theory, and it is not a reasonable approximation of the theory of $\mathbb{N}$ on its own (it cannot prove even basic identities such as $\forall x 0 + x = x$; cf. Exercises 34 and 35). It is introduced in a utilitarian way as a minimal-ish theory for which (or rather, for whose arbitrary extensions) we can prove Gödel’s theorem. Here, the weakness of $\mathbf{Q}$ becomes its strength—the weaker the base theory is, the more broadly applicable Gödel’s theorem is, and the easier it is to verify its assumptions in a given application. It is, in particular, technically convenient that $\mathbf{Q}$ is (unlike PA) finitely axiomatized.

We will derive the *incompleteness* of extensions of $\mathbf{Q}$ from their *undecidability* (cf. Lemma 2.33). Towards that goal, we will show that we can “represent” semidecidable sets $X \subseteq \mathbb{N}$ in the theory, and then undecidability of the theory will follow by reduction from the halting problem. We do this in two steps:

- Semidecidable sets are definable by so-called Σ_1 formulas in $\mathbb{N}$.
 - The main technical ingredient here is that $L_{\mathbf{PA}}$ is expressive enough to define encoding of finite sequences.
- True Σ_1 sentences are provable in $\mathbf{Q}$.

We start with the second bullet point.

3.2 Σ_1 -completeness of $\mathbf{Q}$

Definition 3.3. *Bounded quantifiers* are the abbreviations

$$\exists x \leq t \varphi \equiv \exists x (x \leq t \wedge \varphi), \quad \forall x \leq t \varphi \equiv \forall x (x \leq t \rightarrow \varphi),$$

where t is a term not containing x .

A formula is *bounded*, or Δ_0 , if all its quantifiers are bounded.

A formula $\varphi(\vec{x})$ is Σ_1 if it has the form $\exists \vec{y} \theta(\vec{x}, \vec{y})$ where θ is bounded.

Remark 3.4. The motivation for the definition is that a bounded quantifier $\exists x \leq t \dots$ or $\forall x \leq t \dots$ only quantifies over a finite set $\{0, \dots, t\}$, hence we can verify its truth by checking all the cases one by one. This may not be literally true if t depends on other variables, as then—if we are in a nonstandard model $\mathcal{M}$ —the interval $\{x \in M : x \leq^{\mathcal{M}} t\}$ may actually be an infinite set when viewed from outside of the model. However, if the value of t is a standard number, this argument works, even in a very weak theory like $\mathbf{Q}$. This is the intuition behind the fact that all true Σ_1 sentences are provable in $\mathbf{Q}$ that we are aiming to prove.

Definition 3.5. The *numeral* representing $n \in \mathbb{N}$ is the closed term

$$\bar{n} = \underbrace{S(S(\dots(S(0))\dots))}_{n \text{ times}}.$$

Formally, we define $\bar{n}$ by induction (in the meta-theory) as $\bar{0} = 0$ and $\overline{n+1} = S(\bar{n})$. It follows immediately from the definition that

$$\bar{n}^{\mathbb{N}} = n.$$

Remark 3.6. The next key lemma expresses that $\mathbf{Q}$ can evaluate arithmetic operations on standard numbers, and bounded quantifiers with standard bounds. We note that essentially all our subsequent results about $\mathbf{Q}$ (Σ_1 -completeness, the undecidability and incompleteness theorems) only rely on this lemma rather than any other properties of $\mathbf{Q}$; that is, they hold when $\mathbf{Q}$ is replaced with the theory axiomatized by the sentences listed in the statement of Lemma 3.7 (this is a variant of the theory known in the literature as *Robinson’s theory* $\mathbf{R}$).

Lemma 3.7. *Let $n, m \in \mathbb{N}$.*

- (i) $\mathbf{Q} \vdash \bar{n} + \bar{m} = \overline{n + m}$.
- (ii) $\mathbf{Q} \vdash \bar{n} \cdot \bar{m} = \overline{nm}$.
- (iii) *If $n \neq m$, then $\mathbf{Q} \vdash \bar{n} \neq \bar{m}$.*
- (iv) $\mathbf{Q} \vdash \forall x (x \leq \bar{n} \leftrightarrow x = \bar{0} \vee x = \bar{1} \vee \dots \vee x = \bar{n})$.

Proof.

- (i) By induction (in the meta-theory!) on m . The base case $m = 0$ is clear:

$$\mathbf{Q} \vdash \bar{n} + 0 \stackrel{\text{Q4}}{=} \bar{n} = \overline{n + 0}.$$

For the induction step $m \mapsto m + 1$:

$$\mathbf{Q} \vdash \bar{n} + \overline{m + 1} \stackrel{\text{def}}{=} \bar{n} + S(\bar{m}) \stackrel{\text{Q5}}{=} S(\bar{n} + \bar{m}) \stackrel{\text{i.h.}}{=} S(\overline{n + m}) \stackrel{\text{def}}{=} \overline{n + m + 1}.$$

- (ii) Again, the proof is by meta-induction on m . The base case $m = 0$ is **Q6**. For the induction step, $\mathbf{Q}$ proves

$$\bar{n} \cdot \overline{m + 1} \stackrel{\text{def}}{=} \bar{n} \cdot S(\bar{m}) \stackrel{\text{Q7}}{=} \bar{n} \cdot \bar{m} + \bar{n} \stackrel{\text{i.h.}}{=} \overline{nm} + \bar{n} \stackrel{(i)}{=} \underbrace{\overline{nm + n}}_{n(m+1)}.$$

- (iii) By meta-induction on $\min\{n, m\}$. First suppose $m = 0 < n$. Then

$$\mathbf{Q} \vdash \bar{n} \stackrel{\text{def}}{=} S(\overline{n - 1}) \stackrel{\text{Q2}}{\neq} 0.$$

Similarly if $m > 0 = n$. Finally, if $n, m > 0$, we have

$$\begin{array}{ll} \mathbf{Q} \vdash \bar{n} = \bar{m} \rightarrow \overline{n - 1} = \overline{m - 1} & \text{Q1,} \\ \mathbf{Q} \vdash \overline{n - 1} \neq \overline{m - 1} & \text{induction hypothesis.} \end{array}$$

- (iv) ($\leftarrow$) If $m \leq n$, then

$$\begin{array}{ll} \mathbf{Q} \vdash \bar{n} = \overline{n - m} + \bar{m} & \text{by (i),} \\ \vdash \bar{m} \leq \bar{n} & \text{by Q8.} \end{array}$$

- ($\rightarrow$) By meta-induction on n :

Base case $n = 0$. Let us reason in $\mathbf{Q}$. If $x \leq 0$, then $z + x = 0$ for some z by **Q8**. By **Q3**, either $x = 0$ and we are done, or $x = S(y)$ for some y . But then $0 = z + S(y) = S(z + y)$ by **Q5**, contradicting **Q2**.

Induction step $n \mapsto n + 1$. Reason in $\mathbf{Q}$, and assume $x \leq \overline{n + 1}$. Again, we have $\overline{n + 1} = z + x$ for some z by **Q8**, and either $x = 0$ (in which case we are done) or $x = S(y)$ for some y .

In the latter case, $S(z + y) = \overline{n + 1} = S(\bar{n})$ by **Q5**, thus **Q1** implies $z + y = \bar{n}$, i.e., $y \leq \bar{n}$ by **Q8**.

Then y is 0 or $\bar{1}$ or $\dots$ or $\bar{n}$ by the induction hypothesis, thus x is $\bar{1}$ or $\bar{2}$ or $\dots$ or $\overline{n + 1}$. $\square$

Corollary 3.8. *If t is a closed $L_{\mathbf{PA}}$ -term, and $t^{\mathbb{N}} = n$, then $\mathbf{Q} \vdash t = \bar{n}$.*

Proof. By induction on the complexity of t using (i) and (ii) of Lemma 3.7. $\square$

Lemma 3.9. *Let θ be a Δ_0 sentence. Then*

$$\begin{aligned}\mathbb{N} \models \theta &\implies \mathbb{Q} \vdash \theta, \\ \mathbb{N} \not\models \theta &\implies \mathbb{Q} \vdash \neg\theta.\end{aligned}$$

Proof. By induction on the complexity of θ :

Atomic formulas. Suppose θ is $t = s$ or $t \leq s$ for some closed terms t and s (they have to be closed as θ is a sentence). Let $n = t^{\mathbb{N}}$ and $m = s^{\mathbb{N}}$. By Corollary 3.8, $\mathbb{Q} \vdash t = \bar{n}$ and $\mathbb{Q} \vdash s = \bar{m}$. Moreover, using Lemma 3.7,

$$\begin{aligned}\mathbb{N} \models t = s &\implies n = m \implies \mathbb{Q} \vdash \bar{n} = \bar{m}, \\ \mathbb{N} \not\models t = s &\implies n \neq m \implies \mathbb{Q} \vdash \bar{n} \neq \bar{m} && \text{by (iii),} \\ \mathbb{N} \models t \leq s &\implies n \leq m \implies \mathbb{Q} \vdash \bar{n} \leq \bar{m} && \text{by (iv),} \\ \mathbb{N} \models t \not\leq s &\implies n \not\leq m \implies \mathbb{Q} \vdash \bar{n} \not\leq \bar{m} && \text{by (iv) and (iii).}\end{aligned}$$

Conjunction, Disjunction, Negation. This case is left as an exercise.

Universal quantification. Suppose $\theta = \forall x \leq t \theta_0(x)$ for some closed term t . As before, we have $\mathbb{Q} \vdash t = \bar{n}$, where $n = t^{\mathbb{N}}$.

- Suppose $\mathbb{N} \models \theta$. It follows that for each $m \leq n$, we have $\mathbb{N} \models \theta_0(\bar{m})$, thus $\mathbb{Q} \vdash \theta_0(\bar{m})$ by the induction hypothesis. Moreover,

$$\mathbb{Q} \vdash x \leq t \rightarrow x = \bar{0} \vee x = \bar{1} \vee \dots \vee x = \bar{n}$$

by (iv), thus

$$\mathbb{Q} \vdash x \leq t \rightarrow \theta_0(x),$$

whence $\mathbb{Q} \vdash \forall x \leq t \theta_0(x)$.

- Suppose $\mathbb{N} \not\models \theta$. Then there is $m \leq n$ such that $\mathbb{N} \not\models \theta_0(\bar{m})$; whence by the induction hypothesis, $\mathbb{Q} \vdash \neg\theta_0(\bar{m})$. Moreover, $\mathbb{Q} \vdash \bar{m} \leq \bar{n} = t$ by (iv), hence $\mathbb{Q} \vdash \neg\forall x \leq t \theta_0(x)$.

Existential quantification. This is analogous to universal quantification; details are left as an exercise. $\square$

Theorem 3.10 (Σ_1 -completeness). *Every true Σ_1 sentence φ is provable in Robinson's arithmetic $\mathbb{Q}$.*

Proof. Let $\varphi = \exists x_0, \dots, x_{k-1} \theta(\vec{x})$ be a Σ_1 sentence, where $\theta \in \Delta_0$. Then $\mathbb{N} \models \varphi$ implies there are some $n_0, \dots, n_{k-1} \in \mathbb{N}$ such that $\mathbb{N} \models \theta(\vec{n})$, which implies $\mathbb{Q} \vdash \theta(\bar{n}_0, \dots, \bar{n}_{k-1})$ by the previous lemma, whence $\mathbb{Q} \vdash \exists \vec{x} \theta(\vec{x})$. $\square$

3.3 Sequence encoding and definability of computation

Our next goal is to express Turing computation by formulas in the language of arithmetic. We will forget about $\mathbb{Q}$ for the moment: we will work exclusively with true arithmetic $\text{Th}(\mathbb{N})$ in this section, which is much more convenient, as we do not have to verify the formal provability of anything. The results will be linked back to $\mathbb{Q}$ by means of its Σ_1 -completeness: we will make sure to express everything relevant by Σ_1 formulas.

In order to make our life simpler, we will allow certain definable functions to appear inside Δ_0 and Σ_1 formulas, with complexity low enough so that they can be eliminated.

Definition 3.11. An $R \subseteq \mathbb{N}^k$ is a Δ_0 relation if it is definable in $\mathbb{N}$ by a Δ_0 formula $\theta(\vec{x})$, i.e.,

$$R(\vec{n}) \iff \mathbb{N} \models \theta(\vec{n})$$

for all $\vec{n} \in \mathbb{N}^k$.

A Δ_0 function is a partial function $f: \mathbb{N}^k \rightarrow \mathbb{N}$ such that

- f is bounded by an L_{PA} -term t (i.e., a polynomial with coefficients from $\mathbb{N}$): $f(\vec{x}) \leq t(\vec{n})$ for all $\vec{n} \in \text{dom}(f)$; and
- the graph of f , i.e., $\{(\vec{n}, m) \in \mathbb{N}^{k+1} : f(\vec{n}) = m\}$, is a Δ_0 relation.

Example 3.12. The function $x \dot{-} y = \max\{x - y, 0\}$ is a Δ_0 function, as $x \dot{-} y \leq x$, and $x \dot{-} y = z$ iff $x = y + z \vee (x \leq y \wedge z = 0)$.

The functions $\lfloor x/y \rfloor$ and $\text{rem}(x, y) = x - y\lfloor x/y \rfloor$ for $y > 0$ (i.e., $\text{rem}(x, y)$ is the unique r such that $0 \leq r < y$ and $x \equiv r \pmod{y}$) are Δ_0 functions: they are again bounded by x , and their graphs are Δ_0 -definable by

$$\begin{aligned} \lfloor x/y \rfloor = z &\iff yz \leq x \wedge x < y(z+1), \\ \text{rem}(x, y) = z &\iff z < y \wedge \exists u \leq x \, x = yu + z. \end{aligned}$$

Definition 3.13. If $L \supseteq L_{\text{PA}}$, an L -formula θ is a $\Delta_0(L)$ formula if all quantifiers in θ are bounded. A $\Sigma_1(L)$ formula is an L -formula of the form $\exists \vec{y} \theta(\vec{x}, \vec{y})$, where θ is a $\Delta_0(L)$ formula.

(Thus, the difference between Δ_0 and $\Delta_0(L)$ formulas is that the latter may use the extra relations and functions from L in atomic formulas, and in the case of functions, in quantifier bounds; likewise for $\Sigma_1(L)$ formulas.)

We can now formally state that Δ_0 relations and functions can be eliminated in the following way:

Lemma 3.14. Let $L \supseteq L_{\text{PA}}$, and $\mathbb{N}(L)$ be an L -structure expanding the standard model $\mathbb{N}$ such that the interpretations of relations and functions from $L \setminus L_{\text{PA}}$ in $\mathbb{N}(L)$ are Δ_0 relations and Δ_0 functions, respectively.

Then any $\Delta_0(L)$ formula is equivalent in $\mathbb{N}(L)$ to a Δ_0 formula, and any $\Sigma_1(L)$ formula is equivalent in $\mathbb{N}(L)$ to a Σ_1 formula.

Proof. It suffices to prove the result for $\Delta_0(L)$ formulas. We can replace each $(L \setminus L_{\text{PA}})$ -relation symbol with its Δ_0 definition. We have to be slightly more careful with functions, as they may appear nested in terms. However, they can be successively eliminated inside out: if $f(\vec{s})$ is an occurrence of an $(L \setminus L_{\text{PA}})$ -function f inside an atomic subformula $\psi(f(\vec{s}), \dots)$, where $\vec{s}$ are L_{PA} -terms, then we rewrite it as $\exists u \leq t(\vec{s}) (\theta(\vec{s}, u) \wedge \psi(u, \dots))$, where t is a bounding L_{PA} -term for f , and θ is a Δ_0 definition of the graph of f ; likewise if $f(\vec{s})$ appears in a quantifier bound $Qx \leq t'(f(\vec{s}), \dots) \psi(x, \dots)$. Each such replacement decreases the number of occurrences of the new function symbols, hence we will eventually eliminate all of them. $\square$

Let us now go back to L_{PA} -definability of computation. The main tool we are missing yet is *encoding of finite sequences*: we need to be able to express that x is accepted by a Turing machine M iff there exists a *sequence* of configurations of M with certain properties, and so on.

It is straightforward to encode sequences of fixed length, which can be reduced to ordered pairs:

Lemma 3.15. The L_{PA} -term $[x, y] = (x + y)^2 + x$ is a pairing function: i.e.,

$$\mathbb{N} \models \forall x, y, x', y' ([x, y] = [x', y'] \rightarrow x = y \wedge x' = y').$$

Proof. We have $(x + y)^2 \leq [x, y] < (x + y + 1)^2$. Thus, if $[x, y] = [x', y']$, then $(x + y)^2 = (x' + y')^2$, which implies $x + y = x' + y'$, which implies $x = x'$, whence $y = y'$. $\square$

It is much less obvious how to encode sequences of variable length. There are several strategies how to accomplish that; we will use an elegant definition due to Gödel based on the idea that a number x can encode the sequence $\langle \text{rem}(x, m_0), \text{rem}(x, m_1), \dots \rangle$ for suitable moduli $m_0, m_1, \dots$. (Some alternative sequence encoding schemes are introduced in Exercises 37–41.)

What makes Gödel's encoding work is the *Chinese remainder theorem*: this is a well-known result in elementary number theory/algebra, but we include a short proof for completeness.

Lemma 3.16 (Chinese remainder theorem). *If $m_0, m_1, \dots, m_{k-1}$ are pairwise coprime natural numbers, then for every $x_0, x_1, \dots, x_{k-1} \in \mathbb{N}$, there exists $x \in \mathbb{N}$ such that*

$$x \equiv x_i \pmod{m_i} \quad \text{for all } i < k.$$

Proof. Put $m = \prod_{i < k} m_i$, and consider the abelian group homomorphism $f: \mathbb{Z}/m\mathbb{Z} \rightarrow \prod_{i < k} \mathbb{Z}/m_i\mathbb{Z}$ given by $f(x) = \langle x \bmod m_i : i < k \rangle$. We see that f is injective: $x \in \ker(f)$ only if x is divisible by all m_i , which—in view of their being pairwise coprime—means that x is divisible by m . But $\mathbb{Z}/m\mathbb{Z}$ and $\prod_{i < k} \mathbb{Z}/m_i\mathbb{Z}$ are finite sets of the same size, viz. m , hence the injectivity of f implies that it is surjective. $\square$

In order to apply the Chinese remainder theorem, we also need a suitable definable sequence of pairwise coprime moduli:

Lemma 3.17. *If $1, \dots, k \mid m$, then $\{1 + im : i \leq k\}$ are pairwise coprime.*

Proof. If $i < j \leq k$ and $p \mid 1 + im, 1 + jm$ is prime, then $p \mid (i - j)m$ implies $p \mid m^2$, whence $p \mid m$ and thus $p \mid 1$: a contradiction. $\square$

Definition 3.18. Gödel's β -function is $\beta(x, m, i) = \text{rem}(x, 1 + (i + 1)m)$.

Theorem 3.19 (Sequence encoding). *$\beta(x, m, i)$ is a Δ_0 function such that for any $k \in \mathbb{N}$ and any $x_0, \dots, x_{k-1} \in \mathbb{N}$, there are $x, m \in \mathbb{N}$ that encode this sequence via β in the following sense:*

$$\beta(x, m, i) = x_i \quad \text{for all } i < k.$$

Proof. We have $\beta(x, m, i) \leq x$, and the graph of β is definable by the Δ_0 formula

$$\beta(x, m, i) = y \iff \exists q \leq x \ x = y + q \cdot S(S(i) \cdot m).$$

Thus, β is a Δ_0 function.

Let k and $x_0, \dots, x_{k-1}$ be given. Fix some m such that $1, \dots, k \mid m$ and $m \geq x_i$ for all $i < k$. Since $1 + m, \dots, 1 + km$ are coprime by Lemma 3.17, the Chinese remainder theorem tells us there is some x such that

$$x \equiv x_i \pmod{1 + (1 + i)m} \quad \text{for all } i < k.$$

Also $x_i < 1 + (1 + i)m$, thus $x_i = \text{rem}(x, 1 + (1 + i)m) = \beta(x, m, i)$. $\square$

Example 3.20. Sequence encoding allows us to express recursive definitions in the language of arithmetic. For instance, (the graph of) the powering function x^y is Σ_1 -definable as

$$x^y = z \iff \exists x, m \ (\beta(x, m, 0) = 1 \wedge \beta(x, m, y) = z \wedge \forall i < y \ \beta(x, m, i + 1) = x \cdot \beta(x, m, i)).$$

As defined, Gödel's β -function requires a pair of numbers x, m to encode a sequence $x_0, \dots, x_{k-1}$, and even so it does not determine the length of the sequence (i.e., k). Thus, we introduce a slightly more convenient variant of the function that uses the number $[[x, m], k]$ as the code:

Definition 3.21. We introduce Δ_0 functions seq and len by

$$\begin{aligned} \text{seq}(w, i) = y &\iff \exists x, m, k \leq w \ (w = [[x, m], k] \wedge i < k \wedge \beta(x, m, i) = y), \\ \text{len}(w) = k &\iff \exists x, m \leq w \ w = [[x, m], k]. \end{aligned}$$

The intention is that w encodes a sequence of length $\text{len}(w)$ whose i th entry is $\text{seq}(w, i)$ for $i < \text{len}(w)$. (The functions as defined are partial— w only codes a sequence if it is of the form $[[x, m], k]$ for some x, m, k . This will not be a concern.)

Theorem 3.22. *Every semidecidable set of natural numbers $X \subseteq \mathbb{N}$ is Σ_1 -definable; i.e., there exists a Σ_1 formula $\sigma(x)$ such that*

$$n \in X \iff \mathbb{N} \models \sigma(\bar{n}) \quad \text{for all } n \in \mathbb{N}.$$

Proof. Fix a Turing machine $M = \langle Q, \Sigma, \Gamma, q_0, q_{\text{acc}}, q_{\text{rej}}, \delta \rangle$ that semidecides X , or more precisely, the set of strings $\{w \in \{1, 2\}^* : \ulcorner w \urcorner \in X\}$ (see Definition 2.21). Thus, $\Sigma = \{1, 2\}$; we assume w.l.o.g. that the elements of Q and Γ are natural numbers as well (in particular, we identify $\sqcup$ with some natural number $\neq 1, 2$).

We represent configurations of M by natural numbers using sequence encoding. We cannot literally follow Definition 2.2 as we cannot encode infinite sequences, thus we represent a configuration as (a code of) a sequence $\langle q, h, w_0, \dots, w_s \rangle$ where $q \in Q$ is the current state, h is the head position, w_i is the content of i th cell of the tape, and $s \geq h$ is such that $w_i = \sqcup$ for all $i > s$. Note that the representation of a given configuration is non-unique, because the representation may use arbitrarily large s , and regardless of that, a given finite sequence can be coded by infinitely many different numbers.

Working with this representation, we will present formulas $\text{Initial}(u, x)$ expressing “ u is the initial configuration on input x ”, $\text{Accepting}(u)$ expressing “ u is an accepting configuration”, and $\text{Yields}(u, v)$ expressing “ u yields v ”. Then we can define X by the formula

$$\begin{aligned} \sigma(x) = \exists w [& \text{len}(w) \geq 1 \wedge \text{Initial}(\text{seq}(w, 0), x) \\ & \wedge \text{Accepting}(\text{seq}(w, \text{len}(w) \div 1)) \\ & \wedge \forall i < \text{len}(w) \div 1 \text{ Yields}(\text{seq}(w, i), \text{seq}(w, i + 1))] \end{aligned}$$

expressing Definition 2.3. Note that the Δ_0 functions seq , len , and $\div$ can be eliminated by Lemma 3.14; thus, we can write $\sigma(x)$ as a Σ_1 formula as long as Initial and Accepting are Σ_1 formulas (whose initial existential quantifiers can be prenexed out of the square bracket), and Yields is a Δ_0 formula.

It remains to define the formulas Initial , Accepting , and Yields with the properties above. Again, we can use Δ_0 functions such as seq and len freely.

We can define

$$\begin{aligned} \text{Accepting}(u) &\equiv \text{seq}(u, 0) = q_{\text{acc}}, \\ \text{Yields}(u, v) &\equiv \bigvee_{\substack{\langle q, a \rangle \in Q \times \Gamma \\ \delta(q, a) = \langle q', a', t \rangle}} \text{Next}_{q, a, q', a', t}(u, v), \end{aligned}$$

where $\text{Next}_{q, a, q', a', t}(u, v)$ denotes

$$\begin{aligned} \text{seq}(u, 0) &= q \wedge \text{seq}(u, \text{seq}(u, 1) + 2) = a \\ \wedge \text{len}(v) &\geq \max\{\text{len}(u), \text{seq}(v, 1) + 3\} \\ \wedge \forall i < \text{len}(v) \text{ seq}(v, i) &= \begin{cases} q' & i = 0, \\ \text{seq}(u, 1) + 1 & i = 1 \wedge t = R, \\ \text{seq}(u, 1) \div 1 & i = 1 \wedge t = L, \\ a' & i = \text{seq}(u, 1) + 2, \\ \text{seq}(u, i) & 2 \leq i < \text{len}(u) \wedge i \neq \text{seq}(u, 1) + 2, \\ \sqcup & i \geq \text{len}(u) \end{cases} \end{aligned}$$

(the last expression can be written using disjunctions and conjunctions as there are only a fixed number of cases).

The main problem with description of the initial configuration is to check that the digits $a_i \in \{1, 2\}$ on the tape form the bijective base-2 representation

$$x = \sum_{j < k} 2^j a_j$$

of the given input $x \in \mathbb{N}$. In order to do this, we use an auxiliary sequence w with values

$$\text{seq}(w, i) = \sum_{j < k-i} 2^j a_{i+j}$$

for $i \leq k$, which satisfies the backwards recurrence

$$\text{seq}(w, k) = 0, \quad \text{seq}(w, i) = 2 \text{seq}(w, i + 1) + a_i,$$

and we check that $\text{seq}(w, 0) = x$. Below, $a_i = \text{seq}(u, i + 2)$ and $k = \text{len}(u) - 2 = \text{len}(w) - 1$:

$$\begin{aligned} \text{Initial}(u, x) \equiv & \text{seq}(u, 0) = q_0 \wedge \text{seq}(u, 1) = 0 \\ & \wedge \exists w \left(\text{len}(w) \geq 1 \wedge \text{len}(u) \geq \max\{\text{len}(w) + 1, 3\} \right. \\ & \quad \wedge \text{seq}(w, 0) = x \\ & \quad \wedge \text{seq}(w, \text{len}(w) \div 1) = 0 \\ & \quad \wedge \forall i < \text{len}(w) \div 1 \left(1 \leq \text{seq}(u, i + 2) \leq 2 \right. \\ & \quad \quad \left. \wedge \text{seq}(w, i) = 2 \text{seq}(w, i + 1) + \text{seq}(u, i + 2) \right) \\ & \quad \left. \wedge \forall i < \text{len}(u) \left(\text{len}(w) + 1 \leq i \rightarrow \text{seq}(u, i) = \perp \right) \right) \end{aligned}$$

□

We mention that conversely, any Σ_1 -definable subset of $\mathbb{N}$ is semidecidable (Exercise 36).

3.4 Undecidability and incompleteness

We have now everything in place to prove the main results of Part 3, but we need one more technical assumption.

Definition 3.23. An L_{PA} -theory T is Σ_1 -*sound* if all Σ_1 sentences σ provable in T are true; i.e.,

$$T \vdash \sigma \implies \mathbb{N} \models \sigma.$$

Observe that any sound theory (Definition 3.1) such as $\mathbf{Q}$ or $\mathbf{PA}$ is Σ_1 -sound, and any Σ_1 -sound theory is consistent.

Although the following statement is often lumped together with Gödel's first incompleteness theorem, it is more properly called the *undecidability theorem*:

Theorem 3.24 (Kleene's undecidability theorem). *Every Σ_1 -sound theory $T \supseteq \mathbf{Q}$ is undecidable.*

Proof. Let $X \subseteq \mathbb{N}$ be an undecidable but semidecidable set, which exists by Theorem 2.25 and Definition 2.21, and let $\sigma(x)$ be a Σ_1 -definition of X , which exists by Theorem 3.22. Then $n \mapsto \sigma(\bar{n})$ is a computable function (cf. Lemma 2.31) that provides a many-one reduction of X to $\text{Thm}(T)$, as

$$n \in X \iff \mathbb{N} \models \sigma(\bar{n}) \iff T \vdash \sigma(\bar{n}).$$

In the second equivalence, " $\implies$ " follows from the Σ_1 -completeness of $\mathbf{Q} \subseteq T$, and " $\impliedby$ " from the Σ_1 -soundness of T . Thus, $\text{Thm}(T)$ is undecidable by Lemma 2.29. □

Theorem 3.25 (Gödel's first incompleteness theorem).

Every Σ_1 -sound, recursively axiomatizable theory $T \supseteq \mathbf{Q}$ is incomplete.

Proof. If T were complete, then T would be decidable by Lemma 2.33, contradicting Theorem 3.24. □

Let us recall our motivating problem from the beginning of Part 2:

Definition 3.26. The *Entscheidungsproblem* for a given finite language L is

$$\{\varphi : \varphi \text{ is an } L\text{-sentence, } \models \varphi\} = \text{Thm}(\emptyset)$$

(where $\emptyset$ denotes the L -theory with an empty set of nonlogical axioms).

Theorem 3.27 (Church). *The Entscheidungsproblem for L_{PA} is undecidable.*

Proof. By Theorem 3.25 and Lemma 2.35 applied with $T = \emptyset$ and $X = Q$. □

Remark 3.28. While we will not go into the details, it is good to mention that the undecidability and incompleteness theorems above can be sharpened in various ways:

- (i) The assumption of Σ_1 -soundness in Theorems 3.24 and 3.25 may be simplified to plain consistency; this is the *Gödel–Rosser theorem*. Explicitly, every consistent extension of Q is undecidable, and therefore incomplete if recursively axiomatizable.

Even better, in view of Lemma 2.35, we see that a theory T in a language $L \supseteq L_{PA}$ is undecidable, and thus incomplete if recursively axiomatizable, whenever it is consistent with Q (i.e., $T + Q$ is consistent).

A useful tool for proving this is the concept of *representation* of computable sets and functions, see Exercises 42–43, upgrading the notion of definability in $\mathbb{N}$ from Theorem 3.22.

- (ii) The results from the previous point even hold when Q is replaced with the weaker theory R mentioned in Remark 3.6. (Since R is not finitely axiomatizable, the fact that mere consistency of $T + R$ suffices for undecidability is not automatic from Lemma 2.35; this is a nontrivial result of Cobham.)
- (iii) We are interested not just in theories in the language of arithmetic, but for example in foundational theories in the language of set theory such as ZFC . The Gödel–Kleene–Rosser theorems can be applied to ZFC in the following way: given an L_{PA} -sentence φ , we define its translation φ^ω into the language of set theory by restricting all quantifiers to ω and replacing arithmetical functions and relations such as $+$, $\cdot$, $\leq$ with their set-theoretical definitions. (We need to expand compound terms similarly to the proof of Lemma 3.14.) Then $T = \{\varphi : ZFC \vdash \varphi^\omega\}$ is an extension of Q (or even PA), which is consistent if ZFC is, whence T is undecidable. The function $\varphi \mapsto \varphi^\omega$ is computable, and it provides a many-one reduction of $\text{Thm}(T)$ to $\text{Thm}(ZFC)$, hence ZFC is undecidable as well. Moreover, T is semidecidable, thus recursively axiomatizable by Remark 2.34; thus it is incomplete. If φ is a sentence independent of T (i.e., neither provable nor refutable), then φ^ω is a sentence independent of ZFC .

This approach can be generalized using the notion of (*relative*) *interpretation*. The details are somewhat technical, but the basic idea is as follows. A *translation* $*$ of a language L_0 to a language L_1 is specified by an L_1 -formula $\delta_*(x)$ that defines the domain of the objects of L_1 (e.g., “ $x \in \omega$ ” in the interpretation of PA in ZFC above), an L_1 -formula $x =^* y$, L_1 -formulas $R^*(\vec{x})$ for each L_0 -relation $R(\vec{x})$, and L_1 -formulas $F^*(\vec{x}, y)$ for each L_0 -function F (representing the graph of F). Given an L_0 -sentence φ , we define its translation φ^* by restricting all quantifiers with δ_* , and replacing $=$ and all relation and function symbols with their $*$ -ed definitions (again, we need to quantify over intermediate results to expand compound terms). If the languages are finite, then $\varphi \mapsto \varphi^*$ is a computable function. (Sometimes even more complicated translations are considered: e.g., we may represent objects of L_0 by k -tuples of objects of L_1 for a fixed $k \geq 1$, thus each variable translates to a k -tuple of variables. This is useful e.g. to interpret plane geometry in $\text{Th}(\mathbb{R})$.)

Then $*$ is an *interpretation* of an L_0 -theory T_0 in an L_1 -theory T_1 if it is a translation of L_0 to L_1 such that T_1 proves $\exists x \delta_*(x)$ and the translations of axioms of T_0 and axioms of equality. It follows that $T_0 \vdash \varphi \implies T_1 \vdash \varphi^*$ for all L_0 -sentences φ . Thus, if L_0 is finite, $\varphi \mapsto \varphi^*$ provides a many-one reduction of $\text{Thm}(T)$ to $\text{Thm}(T_1)$, where $T = \{\varphi : T_1 \vdash \varphi^*\} \supseteq T_0$.

We obtain the following generalization of the undecidability and incompleteness theorems: if T is a consistent theory such that Q (or just R) is interpretable in T , then T is undecidable, and it is incomplete if recursively axiomatizable.

- (iv) (Szmielew, Tarski) As a ready-made application of the previous point to set theories, the *adjunctive set theory* (AST) with axioms

$$\begin{aligned} \exists z \forall t \, t \notin z, \\ \forall x \forall y \, \exists z \forall t \, (t \in z \leftrightarrow t \in x \vee t = y) \end{aligned}$$

(postulating that $\emptyset$ and $x \cup \{y\}$ exist) interprets **Q**. Thus, every theory consistent with AST is undecidable, and it is incomplete if recursively axiomatizable.

The even weaker *Vaught set theory* (VS), axiomatized by the schema

$$\forall x_0, \dots, x_{n-1} \, \exists z \forall t \, \left(t \in z \leftrightarrow \bigvee_{i < n} t = x_i \right)$$

for $n \in \mathbb{N}$ (including $n = 0$, which gives the axiom of empty set), interprets Robinson's theory **R**. Thus, again, every theory consistent with VS is undecidable, and it is incomplete if recursively axiomatizable.

- (v) Replacing **Q** with AST in the proof of Theorem 3.27, we see that the Entscheidungsproblem is undecidable for a language with just one binary relation symbol (and therefore for a language with a binary function, or with a relation or function symbol of higher arity).

In fact, an exact characterization is known: the Entscheidungsproblem for a finite language L is undecidable iff L contains at least one at least binary symbol (relation or function), or at least two unary functions.

3.5 Unprovability of consistency

While Theorem 3.25 shows that any sufficiently strong theory (and in particular, any theory that could serve as the foundation of mathematics) is incomplete, it does not exhibit an explicit statement independent of the theory. This is remedied by Gödel's *second* incompleteness theorem:

Theorem 3.29. *If T is a consistent recursively axiomatized theory extending PA, then T does not prove the sentence Con_T expressing its own consistency.*

In a way, this is still not entirely satisfactory, as the true but unprovable sentence Con_T it provides is not something that would be recognized as an arithmetical result by a number theorist; rather, it is a statement of logic encoded in the first-order language of arithmetic. Nevertheless, it is a statement with an intuitively clear concrete meaning, and one whose unprovability is of independent interest (for one thing, it dooms Hilbert's program to failure).

We are not going to prove Theorem 3.29 as there is not enough time in the course to do it properly; however, we will outline the main ideas that go into it and explain where the difficulties are.

Before we can even start to think about the proof, the first difficulty is the actual *statement* of the theorem: what is Con_T , really?

Since the consistency of T means that T does not prove $\perp$, a natural definition of Con_T is

$$\text{Con}_T = \neg \text{Pr}_T(\ulcorner \perp \urcorner)$$

where $\text{Pr}_T(x)$ is a *provability predicate* for T : a formula expressing “(the sentence encoded by) x is provable in T ”. But this just shifts the problem to what is Pr_T .

We know from Theorem 3.22 that $\text{Thm}(T)$ is definable in $\mathbb{N}$ by a Σ_1 formula, and we may consider taking any such formula as Pr_T . But a moment's reflection shows that even though this is a useful and perhaps desirable property for Pr_T to have, it is far from sufficient to establish Theorem 3.29: e.g., if $\sigma(x)$ is any Σ_1 formula that defines $\text{Thm}(T)$, then $\sigma(x) \wedge x \neq \ulcorner \perp \urcorner$ is also a Σ_1 formula that defines $\text{Thm}(T)$ (as T is, in fact, consistent), and the consistency of the latter provability predicate is trivially provable.

Rather than relying on the “extensional” definability of $\text{Thm}(T)$ by Pr_T , we should construct an “intensional” definition that closely mimics the actual definition of provability from Section 1.4. Going in a top-down fashion, we put

$$\text{Pr}_T(x) = \exists p \text{ Proof}_T(p, x)$$

where $\text{Proof}_T(p, x)$ is supposed to describe “ p is a proof of x ”. In turn, we can define $\text{Proof}_T(p, x)$ using sequence encoding as

$$\begin{aligned} \text{len}(p) > 0 \wedge \text{seq}(p, \text{len}(p) \div 1) = x \\ \wedge \forall i < \text{len}(p) \left(\tau(\text{seq}(p, i)) \right. \\ \quad \vee \text{AxP}(\text{seq}(p, i)) \vee \text{AxQ}(\text{seq}(p, i)) \vee \text{AxE}(\text{seq}(p, i)) \\ \quad \vee \exists j, k < i \text{ MP}(\text{seq}(p, i), \text{seq}(p, j), \text{seq}(p, k)) \\ \quad \left. \vee \exists j < i \text{ Gen}(\text{seq}(p, i), \text{seq}(p, j)) \right), \end{aligned}$$

where $\tau(x)$ is fixed Σ_1 formula that defines the set of axioms of T ; $\text{AxP}(x)$, $\text{AxQ}(x)$, and $\text{AxE}(x)$ mean “ x is a propositional axiom”, “ x is a quantifier axiom”, and “ x is an axiom of equality”, respectively; $\text{MP}(x, y, z)$ means “ x is inferred from y and z by modus ponens”, and $\text{Gen}(x, y)$ “ x is inferred from y by a generalization rule”.

We proceed to define the formulas AxP etc. in a similar fashion; in the process, we will need to define formulas expressing things like “ x is a term”, “ x is a formula”, “ w is the result of substitution of free occurrences of variable x for a term y in a formula z ”. It is reasonably obvious that we can express all such things using sequence encoding to mimic the definitions from the real world, but it should be equally obvious that carrying this out down to the last iota is quite tedious.

Note that our definition of Pr_T and Con_T depends on the choice of the formula τ describing the axiom set of T , not just on T itself; thus, we should indicate this in the notation more properly by writing Pr_τ and Con_τ . But we will not bother with this.

With some effort, it can be checked that Pr_T is (provably equivalent to) a Σ_1 formula, and by construction, it defines $\text{Thm}(T)$ in $\mathbb{N}$. But it has many other natural properties as well. In particular, the proof of Theorem 3.29 essentially relies on the following statement, which is, in fact, the most difficult and most technical part of the proof.

Theorem 3.30. *If Pr_T is the provability predicate of a recursively axiomatized theory $T \supseteq \text{PA}$ constructed as outlined above, then Pr_T satisfies the following Hilbert–Bernays–Löb derivability conditions for all sentences φ and ψ :*

- (D1) *If T proves φ , then T proves $\text{Pr}_T(\ulcorner \varphi \urcorner)$.*
- (D2) *T proves $\text{Pr}_T(\ulcorner \varphi \urcorner) \rightarrow \text{Pr}_T(\ulcorner \text{Pr}_T(\ulcorner \varphi \urcorner) \urcorner)$.*
- (D3) *T proves $\text{Pr}_T(\ulcorner \varphi \rightarrow \psi \urcorner) \wedge \text{Pr}_T(\ulcorner \varphi \urcorner) \rightarrow \text{Pr}_T(\ulcorner \psi \urcorner)$.*

Proof sketch.

(D1) follows from the Σ_1 -completeness of $T \supseteq \text{Q}$: if T proves φ , then $\text{Pr}_T(\ulcorner \varphi \urcorner)$ is a true Σ_1 sentence.

(D3) is a formalization of the closure of provability under modus ponens, and it is fairly straightforward: we take the proofs of $\varphi \rightarrow \psi$ and φ , concatenate them (as sequences), add ψ at the end, and argue that this is a proof of ψ .

(D2) is the most difficult part. It is, in effect, a formalization of (D1) inside T itself as a meta-theory; since (D1) was proved by appealing to Σ_1 -completeness, (D2) can be proved as a special case of *formalized Σ_1 -completeness*³: T proves

$$\sigma(u) \rightarrow \text{Pr}_T(\ulcorner \sigma(\dot{u}) \urcorner)$$

for every Σ_1 formula $\sigma(x)$ (here, the deceptively simple notation $\ulcorner \sigma(\dot{u}) \urcorner$ hides the rather complicated operation that given a number u , we construct (the Gödel number of) the closed term $\bar{u}$, and substitute

³We apply formalized Σ_1 -completeness with σ being the sentence $\text{Pr}_T(\ulcorner \varphi \urcorner)$, but the version with free variables is needed for the inductive proof.

it into σ for x , taking the Gödel number of the resulting sentence). This can be shown by induction on the complexity of σ , but we have to basically formalize the whole proof of Lemma 3.9 in T . $\square$

The original proof of Gödel's *first* incompleteness theorem did not rely on results from computability theory as we have done (Gödel did not have a general definition of computability), but it proceeded by syntactic manipulations using the provability predicate. The rôle played by the diagonal argument in the proof of Theorem 2.25 was taken by the so-called *Gödel's diagonal lemma*:

Lemma 3.31. *For every L_{PA} -formula $\varphi(x)$, there is an L_{PA} -sentence α such that Q proves $\alpha \leftrightarrow \varphi(\overline{\alpha})$.*

Proof. Exercise 44. $\square$

Using the diagonal lemma, we can construct *Gödel's sentence* ν that satisfies

$$Q \vdash \nu \leftrightarrow \neg \text{Pr}_T(\overline{\nu})$$

(i.e., ν says “I am unprovable in T ”). If T is consistent, then ν is unprovable:

$$T \vdash \nu \implies T \vdash \text{Pr}_T(\overline{\nu}) \implies T \vdash \neg \nu \implies T \vdash \perp$$

using Σ_1 -completeness (i.e., (D1)) and the definition of ν . (Thus also, if T is consistent, then $\mathbb{N} \models \neg \text{Pr}_T(\overline{\nu})$, hence $\mathbb{N} \models \nu$, hence $T \not\vdash \neg \nu$ if T is Σ_1 -sound. This provides an alternative proof of Theorem 3.25.)

Now, the proof of Gödel's second incompleteness theorem is essentially a formalization of this argument in T itself. More precisely, we have:

Theorem 3.32. *Let T be a consistent extension of Q , and Pr_T a provability predicate for T that satisfies the Hilbert–Bernays–Löb derivability conditions. Then $T \not\vdash \neg \text{Pr}_T(\overline{\perp})$.*

Proof. Let ν be as above. Since T proves $\text{Pr}_T(\overline{\nu}) \rightarrow (\nu \rightarrow \perp)$, it proves

$$\begin{aligned} & \text{Pr}_T(\overline{\text{Pr}_T(\overline{\nu}) \rightarrow (\nu \rightarrow \perp)}), \\ & \text{Pr}_T(\overline{\text{Pr}_T(\overline{\nu})}) \rightarrow \text{Pr}_T(\overline{\nu \rightarrow \perp}), \\ & \text{Pr}_T(\overline{\text{Pr}_T(\overline{\nu})}) \rightarrow (\text{Pr}_T(\overline{\nu}) \rightarrow \text{Pr}_T(\overline{\perp})) \end{aligned}$$

by applying (D1) and (D3) twice. Using also (D2), we obtain

$$T \vdash \text{Pr}_T(\overline{\nu}) \rightarrow \text{Pr}_T(\overline{\perp}).$$

Assuming for contradiction that $T \vdash \neg \text{Pr}_T(\overline{\perp})$, we infer $T \vdash \neg \text{Pr}_T(\overline{\nu})$, whence $T \vdash \nu$ by the definition of ν , and $T \vdash \text{Pr}_T(\overline{\nu})$ by (D1), thus T is inconsistent. $\square$

Remark 3.33. (For those familiar with modal logic.) The proof of Theorem 3.32 does not overtly use any quantifiers; it essentially looks like a proof in a propositional modal logic where Pr_T plays the rôle of the necessity modal operator $\Box$. Elaboration of this idea leads to so-called *provability logic*.

Remark 3.34. As with Gödel's first incompleteness theorem, we can apply the second incompleteness theorem to theories in other languages than L_{PA} by means of interpretations. But surprisingly, this idea can be used to significantly improve the statement of the incompleteness theorem even for theories of arithmetic, by exploiting the fact that Q interprets some weak fragments of PA such as induction for Δ_0 formulas (“bounded arithmetic”), that are nevertheless strong enough to carry out some form of the proof of Theorem 3.30. We can obtain the following elegant formulation that applies to theories in any language, and brings the base theory down from PA to Q , even though Q is too weak to directly prove the Hilbert–Bernays–Löb derivability conditions:

Theorem 3.35 (Pudlák). *If T is a consistent recursively axiomatized theory, then T cannot interpret $Q + \text{Con}_T$.*

Exercises (in 2024/25)

We have seen in the lecture that the De Morgan language $\{\wedge, \vee, \neg, \top, \perp\}$ is functionally complete, and specifically, that every Boolean function can be represented by a CNF or DNF of size $O(2^n)$.

1. Prove that $\{\vee, \neg\}$, $\{\rightarrow, \perp\}$, and $\{\uparrow\}$ are functionally complete, where $x \uparrow y$ denotes the Sheffer stroke $\neg(x \wedge y)$.

2. Prove that $\{\rightarrow\}$, $\{\wedge, \vee, \top, \perp\}$, and $\{\leftrightarrow, \top, \perp\}$ are not functionally complete.

[Hint: Find a nontrivial property of Boolean functions which is preserved by composition, and holds for functions in the given basis.]

3. For any Boolean function $f: \{0, 1\}^n \rightarrow \{0, 1\}$, the following are equivalent:

(i) $\{f\}$ is functionally complete.

(ii) $f(0, \dots, 0) = 1$, $f(1, \dots, 1) = 0$, and there exists an assignment $\langle e_0, \dots, e_{n-1} \rangle \in \{0, 1\}^n$ such that $f(e_0, \dots, e_{n-1}) = f(\neg e_0, \dots, \neg e_{n-1})$.

[Hint: For (ii) $\rightarrow$ (i), look at functions obtained from f by identifying some of the variables.]

4. For any $n \in \mathbb{N}$, the *parity* function $\bigoplus_{i < n} x_i: \{0, 1\}^n \rightarrow \{0, 1\}$ is defined as $(\sum_{i < n} x_i) \bmod 2$. Show that any DNF or CNF representing $\bigoplus_{i < n} x_i$ has size $\Omega(2^n)$. [Hint: What terms of the form $\bigwedge_{i \in I} x_i^{e_i}$ can imply one of $\bigoplus_{i < n} x_i = 0$ or $\bigoplus_{i < n} x_i = 1$? Here, $I \subseteq [n]$, $e_i \in \{0, 1\}$, $x^1 = x$, $x^0 = \neg x$.]

5. There are formulas representing $\bigoplus_{i < n} x_i$ of size $O(n^c)$ for some constant c .

[Hint: Consider a balanced tree of binary parities. You may get it down to $c = 2$.]

6. Any DNF equivalent to the CNF $\bigwedge_{i < n} (x_i \vee y_i)$ has size $\Omega(2^n)$.

7. Every Boolean function $f: \{0, 1\}^n \rightarrow \{0, 1\}$ can be represented by a formula of size $O(2^n)$.

[Hint: Inductively express a formula in $n + 1$ variables as a combination of formulas in n variables.]

8. Prove the propositional soundness theorem: for all $\Gamma \subseteq \text{Prop}_A$ and $\varphi \in \text{Prop}_A$, if $\Gamma \vdash \varphi$, then $\Gamma \models \varphi$.

9. Let $\Gamma, \Delta \subseteq \text{Prop}_A$ and $\varphi, \psi \in \text{Prop}_A$. Show that if $\Gamma \vdash \varphi$ and $\Delta, \varphi \vdash \psi$, then $\Gamma, \Delta \vdash \psi$.

10. For every $\varphi \in \text{Prop}_A$, its *De Morgan dual* $\varphi^d \in \text{Prop}_A$ is obtained by exchanging $\wedge$ with $\vee$, and $\top$ with $\perp$ inside φ . Formally, we define φ^d by induction on the complexity of φ :

$$\begin{aligned} a^d &= a, & a &\in A, & (\neg\varphi)^d &= \neg(\varphi^d), \\ \top^d &= \perp, & & & \perp^d &= \top, \\ (\varphi \wedge \psi)^d &= (\varphi^d \vee \psi^d), & & & (\varphi \vee \psi)^d &= (\varphi^d \wedge \psi^d). \end{aligned}$$

Show that for all assignments $v: A \rightarrow \{0, 1\}$, $v(\varphi^d) = v_{\neg}(\neg\varphi)$, where $v_{\neg}: A \rightarrow \{0, 1\}$ is the assignment defined by $v_{\neg}(a) = 1 - v(a)$ for each $a \in A$.

11. Let $\varphi, \psi \in \text{Prop}_A$.

(i) $\varphi \equiv \psi$ if and only if $\varphi^d \equiv \psi^d$.

(ii) $\varphi \models \psi$ if and only if $\psi^d \models \varphi^d$.

In the lecture, we have proved completeness of a proof system using connectives $\{\rightarrow, \perp\}$. A complete system using the De Morgan language $\{\wedge, \vee, \neg, \top, \perp\}$ is given in the van den Dries lecture notes, but the next exercise shows how to construct one mechanically.

12. For any $\{\rightarrow, \perp\}$ -formula φ , let φ^* denote the De Morgan formula such that $p^* = p$ for atoms p , $\perp^* = \perp$, and $(\varphi \rightarrow \psi)^* = (\neg\varphi^* \vee \psi^*)$. Similarly, given a De Morgan formula ψ , let $\psi^\#$ be its translation to a $\{\rightarrow, \perp\}$ -formula using fixed $\{\rightarrow, \perp\}$ -translations of all De Morgan connectives. Let $\vdash_0$ denote a sound and complete Hilbert-style proof system for $\{\rightarrow, \perp\}$ -formulas such as the one given in the lecture, and let $\vdash_1$ be the Hilbert-style proof system in the De Morgan language that has inference rule schemata $\varphi_1^*, \dots, \varphi_k^* / \varphi_0^*$ for each rule schema $\varphi_1, \dots, \varphi_k / \varphi_0$ of $\vdash_0$ (where axioms are treated as rules with $k = 0$), and axiom schemata $\neg c(\varphi_0, \dots, \varphi_{k-1}) \vee c^{\#*}(\varphi_0, \dots, \varphi_{k-1})$, $\neg c^{\#*}(\varphi_0, \dots, \varphi_{k-1}) \vee c(\varphi_0, \dots, \varphi_{k-1})$ for each k -ary De Morgan connective c . Then $\vdash_1$ is a sound and complete proof system in the De Morgan language. [Hint: You will need to show $\vdash_1 \neg\psi \vee \psi^\#$, $\vdash_1 \neg\psi^{\#*} \vee \psi$ for all De Morgan formulas ψ .]

13. (If you are familiar with topology.) Give a direct proof of the propositional compactness theorem, not using the completeness theorem.

[Hint: Consider the product topology on the set $\{0, 1\}^A$ of all assignments.]

14. A set of propositional or first-order sentences S is *independent* if S is not equivalent to S' for any proper subset $S' \subsetneq S$.

(i) S is independent iff $S \setminus \{\varphi\} \not\models \varphi$ for all $\varphi \in S$.

(ii) Show that every countable theory T has an independent axiomatization, i.e., an independent set of sentences S equivalent to T . [Hint: Try to generalize the fact that $\{\varphi, \psi\} \equiv \{\varphi, \psi \vee \neg\varphi\}$.]

(Uncountable theories have independent axiomatizations as well by a theorem of I. Reznikoff, but this is more difficult to prove.)

15. Prove Lemma 1.57: if a term $t(x_0, \dots, x_{n-1}, y)$ is free for y in a formula $\varphi(x_0, \dots, x_{n-1}, y)$, then for all terms $s_0, \dots, s_{n-1}, r$, the formula $(\varphi(t/y))(s_0/x_0, \dots, s_{n-1}/x_{n-1}, r/y)$ is syntactically identical to the formula $\varphi(s_0/x_0, \dots, s_{n-1}/x_{n-1}, t(s_0/x_0, \dots, s_{n-1}/x_{n-1}, r/y)/y)$.

16. Let $\mathcal{A}$ be an L -structure, t a closed L -term such that $t^{\mathcal{A}} = a \in A$, and $\varphi(x)$ an L -formula. Show that $\mathcal{A} \models \varphi(t)$ iff $\mathcal{A} \models \varphi(a)$.

17. Consider a modification of the first-order proof system given in the lecture such that the axioms of equality are replaced with the axiom $x = x$ and the axiom schema $t = s \wedge \varphi(t/x) \rightarrow \varphi(s/x)$ for all formulas φ and terms t, s free for x in φ . Show that this is equivalent to the original proof system.

18. For any formula $\varphi(x)$ and variable y free for x in φ , show that the formula $\exists y (\exists x \varphi(x) \rightarrow \varphi(y))$ is provable.

19. Using Vaught's test, show the completeness of the theory of a successor: it has a language with one unary function symbol s , and axioms $s(x) = s(y) \rightarrow x = y$, $\forall x \exists y s(y) = x$, and $s^n(x) \neq x$ for each $n \in \mathbb{N}_{>0}$, where s^n denotes the n -fold iteration of s (i.e., $s^0(x)$ is x , and s^{n+1} is $s(s^n(x))$).

20. For each $n \in \mathbb{N}$, let P_n denote the path graph of length n , i.e., the structure $\langle [n], E_n \rangle$, where $[n] = \{0, \dots, n-1\}$ and $E_n = \{\langle i, j \rangle \in [n]^2 : |i - j| = 1\}$. Show that there is no sentence φ such that for all $n \in \mathbb{N}$, $P_n \models \varphi$ iff n is odd. [Hint: Adapt the previous exercise.]

21. Fix a field F . The theory of vector spaces over F has a language consisting of the language $\{+, -, 0\}$ of abelian groups and unary functions $a \cdot x$ for each $a \in F$; it has the usual algebraic axioms (axioms of abelian groups, $ab \cdot x = a \cdot (b \cdot x)$, $1 \cdot x = x$, $(a+b) \cdot x = a \cdot x + b \cdot x$, $a \cdot (x+y) = a \cdot x + a \cdot y$). Show that the theory of infinite vector spaces over F (i.e., with additional axioms $\exists x_0 \dots \exists x_n \bigwedge_{i < j} x_i \neq x_j$ for $n \in \mathbb{N}$) is complete and κ -categorical for all infinite $\kappa > |F|$. [Hint: Every vector space has a basis.]

22. An *atom* in a Boolean algebra $\mathbf{A} = \langle A, 0, 1, \wedge, \vee, -, \leq \rangle$ is an element $a \in A$ such that $a > 0$, but $0 < x < a$ for no $x \in A$; $\mathbf{A}$ is *atomless* if $0 \neq 1$ and $\mathbf{A}$ has no atoms. Show that the theory of atomless Boolean algebras is $\aleph_0$ -categorical, hence complete.

[Hint: Construct an isomorphism between two countable atomless Boolean algebras $\mathbf{A}$ and $\mathbf{B}$ by a back-and-forth argument, as a union of a sequence of isomorphisms between finite subalgebras. It might help to observe that if $\mathbf{A}_0$ is a finite subalgebra of $\mathbf{A}$, and $\mathbf{A}_1$ is the algebra generated by $A_0 \cup \{b\}$ for some $b \in A$, then each atom of $\mathbf{A}_0$ either remains an atom in $\mathbf{A}_1$, or splits into two atoms.]

23. Show that the functions $+: \mathbb{N}^2 \rightarrow \mathbb{N}$ and $\cdot: \mathbb{N}^2 \rightarrow \mathbb{N}$ are computable when the input and output are represented in unary.

24. The same when the input and output are represented in binary.

25. Show that there are computable functions converting natural numbers from one representation to another (unary, ordinary base- k , bijective base- k , considering also different k s).

26. Fix an alphabet Σ .

(i) The following functions are computable: the constant function ε ; the functions $s_a: \Sigma^* \rightarrow \Sigma^*$ for $a \in \Sigma$, defined by $s_a(x) = x \smallfrown a$; the projections $\pi_i^n: (\Sigma^*)^n \rightarrow \Sigma^*$, $\pi_i^n(x_0, \dots, x_{n-1}) = x_i$.

(ii) If $f: (\Sigma^*)^n \rightarrow \Sigma^*$ and $g_i: (\Sigma^*)^m \rightarrow \Sigma^*$, $i < n$, are computable functions, their composition $h: (\Sigma^*)^m \rightarrow \Sigma^*$, $h(\vec{x}) = f(g_0(\vec{x}), \dots, g_{n-1}(\vec{x}))$, is computable.

(iii) If $f_\varepsilon: (\Sigma^*)^n \rightarrow \Sigma^*$ and $f_a: (\Sigma^*)^{n+2} \rightarrow \Sigma^*$, $a \in \Sigma$, are computable, the function $h: (\Sigma^*)^{n+1} \rightarrow \Sigma^*$ defined from them by the recursion

$$\begin{aligned} h(\vec{x}, \varepsilon) &= f_\varepsilon(\vec{x}), \\ h(\vec{x}, y \smallfrown a) &= f_a(\vec{x}, y, h(\vec{x}, y)) \end{aligned}$$

is computable.

Functions in the smallest class that contains the functions from (i) and that is closed under the operations (ii) and (iii) are called *primitive recursive*. (Usually, the definition of primitive recursive functions is stated for functions $\mathbb{N}^n \rightarrow \mathbb{N}$, corresponding to our definition with $|\Sigma| = 1$ and the integers represented in unary. Our more general definition is equivalent up to the bijective base- $|\Sigma|$ numeration.)

27. The set of well bracketed strings over the alphabet $\Sigma = \{(,)_i : i < k\}$ is the smallest set of strings such that the empty string ε is well bracketed, and if x and y are well bracketed and $i < k$, then xy and $(,)_i$ are well bracketed. E.g., $(3(1)_1(2()_0)2(1)_1)3(2)_2$ is well bracketed. Show that the set of well bracketed strings is decidable.

The next exercise is to prove Lemma 2.31:

28. Let L be a finite first-order language. Show that the following sets and functions are computable:

(i) The set of L -terms.

(ii) The set of L -formulas.

(iii) The set of pairs $\langle \varphi, x \rangle$ where x is a free variable of an L -formula φ .

(iv) The substitution function: given an L -formula φ , a variable x , and an L -term t , compute $\varphi(t/x)$.

(v) The set of triples $\langle \Gamma, \varphi, \pi \rangle$ where π is a proof of an L -formula φ from a finite set of L -formulas Γ .

29. A language $X \subseteq \Sigma^*$ is semidecidable iff it can be represented as $\exists w \in \Sigma'^* P(x, w)$ for a finite alphabet Σ' (which we might take to be Σ itself if $|\Sigma| \geq 2$) and a decidable predicate P .

[Hint: Consider a description of an accepting run of a Turing machine, or—if you are already familiar with the section on arithmetic—a Σ_1 -formula that defines X in $\mathbb{N}$.]

30. (Craig's trick.) Every semidecidable theory is recursively axiomatizable. [Hint: Express $\text{Thm}(T)$ as $\exists w P(\varphi, w)$ with P decidable. Given φ and w , devise a sentence equivalent to φ that encodes w .]

31. Show that every decidable consistent theory T has a decidable complete extension.

[Hint: Consider a completion procedure that enumerates sentences φ one by one, and extends the current list of axioms with φ or $\neg\varphi$, whichever maintains consistency with T .]

32. Prove $\mathbf{Q} \vdash \forall x (x \leq \bar{n} \vee \bar{n} \leq x)$ for each $n \in \mathbb{N}$.

33. $\mathbf{Q}$ proves $x \cdot y = 0 \rightarrow x = 0 \vee y = 0$, and more generally, $x \cdot y = \bar{n} \rightarrow x = 0 \vee y \leq \bar{n}$ for each $n \in \mathbb{N}$.

34. The standard model $\mathbb{N}$ extends to an L_{PA} -structure $\mathbb{N}^\infty$ with domain $\mathbb{N} \cup \{\infty\}$, $\infty \notin \mathbb{N}$, so that $\mathbb{N}^\infty \models Q$. Moreover, we are free to choose $(0 \cdot \infty)^{\mathbb{N}^\infty}$ in an arbitrary way (while the rest of the model is uniquely determined by the axioms of Q). Conclude that Q does not prove any of the formulas $S(x) \not\leq x$, $x \cdot y = y \cdot x$, or $0 \cdot x \neq 1$.

35. Q does not prove $x + y = y + x$ or $0 + (x + y) = (0 + x) + y$.
[Hint: Modify the previous exercise to a model with two “infinities”.]

36. All Σ_1 -definable sets are semidecidable.

In the lecture, we developed an encoding of sequences in the language of arithmetic using Gödel’s β -function. In the next three exercises, you will devise an alternative sequence encoding scheme due to E. Nelson, as simplified by P. Pudlák.

37. The set $\{x : \exists n \in \mathbb{N} x = 2^n\}$ of powers of 2 is definable by a Δ_0 formula, not using the 2^n function.
[Hint: Consider the divisors of x .]

38. Consider an encoding of finite sets $X \subseteq \mathbb{N}$ by pairs $[r, w]$ where the binary expansion of w is a concatenation of binary expansions of elements of X , and the binary expansion of r acts as a “ruler” such that the positions of 1s mark where the individual elements of X start in w . Show that the predicate “ x is in the set encoded by $[r, w]$ ” is Δ_0 -definable.

39. Construct a Δ_0 encoding of finite sequences based on Exercise 38.

As yet another alternative, we will look at a representation of binary strings introduced by A. A. Markov Jr., who attributes it to J. Nielsen. The idea of using it for encoding strings in weak theories of arithmetic is due to J. Murwanashyaka; the extension to sequences of integers is due to A. Visser.

40. Let $\langle \text{SL}_2(\mathbb{N}), I, \cdot \rangle$ denote the monoid of non-negative integer matrices $\begin{pmatrix} a & b \\ c & d \end{pmatrix} \in \mathbb{N}^{2 \times 2}$ of determinant 1, with $\cdot$ being matrix multiplication and $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Put $A_0 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ and $A_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$.

(i) Given $i = 0, 1$, which $M \in \text{SL}_2(\mathbb{N})$ are of the form NA_i for $N \in \text{SL}_2(\mathbb{N})$? [Hint: Focus on comparisons between the entries of M .]

(ii) Using (i), show that each $M \in \text{SL}_2(\mathbb{N}) \setminus \{I\}$ can be written in a unique way as NA_0 or NA_1 with $N \in \text{SL}_2(\mathbb{N})$.

(iii) Conclude that $\text{SL}_2(\mathbb{N}) \simeq \langle \{0, 1\}^*, \varepsilon, \cdot \rangle$.

41. Develop a Δ_0 encoding of finite sequences based on the previous exercise. [Hint: You may represent $\{n_0, \dots, n_{k-1}\}$ by $A_0^{n_0} \cdots A_1 A_0^{n_{k-1}-1} A_1$, using $A_0^n = \begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix}$. Then encode sequences by sets.]

A formula $\varphi(x)$ *represents* a set $X \subseteq \mathbb{N}$ in a theory T if $T \vdash \varphi(\bar{n})$ for all $n \in X$, and $T \vdash \neg \varphi(x)$ for all $n \in \mathbb{N} \setminus X$.

A formula $\varphi(x, y)$ represents in T a partial function $f: \mathbb{N} \rightarrow \mathbb{N}$ if $T \vdash \forall y (\varphi(\bar{n}, y) \leftrightarrow y = \bar{m})$ for all $n, m \in \mathbb{N}$ such that $f(n) = m$.

42. All decidable sets are Σ_1 -representable in Q .

[Hint: Starting with Σ_1 -definitions of X and $\mathbb{N} \setminus X$, write a Σ_1 formula expressing “there is a witness for $x \in X$ smaller than any witness for $x \notin X$ ”. Use Exercise 32 to show that it works.]

43. All partial computable functions are Σ_1 -representable in Q .

[Hint: Using a Σ_1 -definition of the graph of f , adapt the witness comparison argument from Exercise 42.]

44. Prove Gödel’s diagonal lemma (Lemma 3.31): for every formula $\varphi(x)$, there exists a sentence α such that $Q \vdash \alpha \leftrightarrow \varphi(\ulcorner \alpha \urcorner)$.

[Hint: Using representability of a suitable computable function, construct a formula $\psi(x)$ such that $Q \vdash \psi(\ulcorner \chi \urcorner) \leftrightarrow \varphi(\ulcorner \chi(\ulcorner \chi \urcorner) \urcorner)$ for all $\chi(x)$.]