COMP0009 Logic

Lecture notes · 2026–27

Part 2: Automated reasoning

Chapter 8

Incompleteness

Alexander V. Gheorghiu

Department of Computer Science, University College London

This chapter is issued on its own from the complete lecture notes for COMP0009 Logic. Its section, theorem, exercise and equation numbers are those of the complete notes, so that the two can be read side by side; references to other chapters and parts point into the complete notes.

Contents

8. Incompleteness 1
8.1. Computation as a Theory 2
8.2. Specifications 7
8.3. The First Incompleteness Theorem 12
8.4. The Second Incompleteness Theorem 19
8.5. Defining Halting *(optional)* 25
8.6. Decidable Theories *(optional)* 32
History 35
Exercises 41

8. Incompleteness

Put yourself back in the schoolroom. The teacher writes on the board the rules for adding, multiplying and taking powers of positive whole numbers, eleven of them:[1]

$$\begin{aligned} x+y &= y+x, & x\cdot(y+z) &= x\cdot y + x\cdot z,\\ (x+y)+z &= x+(y+z), & 1^x &= 1,\\ x\cdot 1 &= x, & x^1 &= x,\\ x\cdot y &= y\cdot x, & x^{y+z} &= x^y\cdot x^z,\\ (x\cdot y)\cdot z &= x\cdot(y\cdot z), & (x\cdot y)^z &= x^z\cdot y^z,\\ & & (x^y)^z &= x^{y\cdot z}. \end{aligned}$$

With them you can prove $(x+1)^2 = x^2+2x+1$, and every other identity you were set as homework. A student at the back asks the obvious question. Do these rules settle *every* identity that is true of the positive whole numbers? If an equation between two expressions in x, y, 1, $+$, $\cdot$ and powers holds whatever numbers are put for the letters, can it always be derived from the eleven?

The question is Tarski's, and it waited more than a decade for an answer. The answer is no. Here is an identity, found by Wilkie, in which $A = 1+x$, $B = 1+x+x^2$, $C = 1+x^3$ and $D = 1+x^2+x^4$:

$$(A^y + B^y)^x \cdot (C^x + D^x)^y = (A^x + B^x)^y \cdot (C^y + D^y)^x.$$

It is true for every pair of positive whole numbers x and y. The reason is a hidden common factor: $C = A\cdot(1-x+x^2)$ and $D = B\cdot(1-x+x^2)$, so that $C^x + D^x$ is $(A^x + B^x)$ times $(1-x+x^2)^x$, and in the same way $C^y + D^y$ is $(A^y + B^y)$ times $(1-x+x^2)^y$. Both sides are then the same product, written in a different order. But the factor $1-x+x^2$ has a minus sign in it, and the eleven rules have no subtraction. The identity is true, and the rules cannot derive it.

Failing to find a derivation would prove nothing. To show that a claim does not follow from rules, exhibit a world in which the rules hold and the claim fails. For Wilkie's identity the smallest such world has twelve elements, with tables for addition, multiplication and powers that obey all eleven rules.

That no smaller world exists was settled in 2026 by the method of the last chapter. Each possible entry of each table became a propositional letter, the eleven rules and the failure of Wilkie's identity became clauses, and a SAT solver said "no" for eleven elements, with a proof of 368 megabytes that an independent program checked.[2] These are the two kinds of evidence of the last chapter: a structure, which anyone can check by plugging in, shows that a claim does not follow; a closed search, written down as a proof, shows that no smaller structure does.

Adding Wilkie's identity to the board does not end the matter: no finite list of identities, however long, derives every true one. There is no last patch.

[1] The story of this introduction, as far as Hilbert, is adapted from Gheorghiu (2025).

[2] Subercaseaux and Przybocki (2026). A lemma, checked in the same way, extends the "no" to fewer elements, and the encoding was verified in a proof assistant. Burris and Yeats had conjectured that 12 is the least size; the History tells the rest of the story.

The schoolroom is a small version of a large hope. In 1899 Hilbert wrote down axioms for Euclid's geometry from which every theorem followed by logic alone, and he hoped for the same in arithmetic, the study of the counting numbers $0, 1, 2, \ldots$ with addition and multiplication: axioms from which every truth about the numbers could be derived. Every sentence about the counting numbers is true or false, even when nobody knows which. Goldbach's conjecture, that every even number greater than 2 is a sum of two primes, is one such sentence. With such axioms, settling it would be a matter of searching through derivations.

On 8 September 1930, in Königsberg, Hilbert ended an address with the words *Wir müssen wissen. Wir werden wissen.* (We must know. We shall know.) On the day before, in the same city, a young logician from Vienna, Kurt Gödel, had announced a theorem that showed the hope to be mistaken.

The last chapter left open two questions about theories and programs, and this chapter takes them up.

- *The search gap.* The first-order procedure `REFUTE` certifies every entailment that holds, and runs for ever on some that do not. Is that a fault of the procedure, which a cleverer search would repair?
- *The specification gap.* Even a perfect oracle for "does this follow from the theory?" tells us only what follows from the theory, when what we wanted was what is true of the system it describes. The two agree only when the theory says everything about the system. Can it always be chosen so?

Neither gap can be closed in general, and one construction answers both questions: the rules by which programs run can be written down as sentences. The chapter takes four steps.

- Section 8.1 shows that no program checks whether a combinator halts, writes the rules by which combinators run as a handful of first-order sentences, and concludes that no program decides first-order validity. That settles the search gap.
- Section 8.2 makes the specification gap precise: a specification *captures* a structure when it proves exactly the sentences true there, and that happens exactly when a program can decide those sentences. Computation cannot be captured, and every finite structure can.
- Section 8.3 shows that the counting numbers contain computation, so that no specification captures them, and, the first incompleteness theorem, names for any sound specification a true sentence that it does not prove.
- Section 8.4, the second incompleteness theorem, shows that a consistent specification of arithmetic does not entail its own consistency, a limit that does not mention the counting numbers at all.

Two optional sections follow. Section 8.5 proves in full that halting can be expressed, and Section 8.6 asks which infinite structures can be captured. The main line quotes some of their results but none of their proofs.

8.1. **Computation as a Theory.** The search gap asks whether *any* procedure decides entailment, so we need to say what a procedure is. Chapter 1

has the answer: a procedure is a combinator, since combinators compute everything that any machine computes.[3] We use the rules for a step, the shape of a normal form and the fact that no step is a wrong turn.

Definition 8.1 (halting)**.** A combinator halts when it has a normal form.

Formally, a combinator M *halts* if $M \rightsquigarrow^* N$ for some N in normal form.

Theorem 8.2 (halting is undecidable)**.** *There is no program for checking whether a given combinator halts.*[4]

Proof. Suppose that a program h checks it. Let P be the program that, given a combinator M, asks h whether M given itself as input halts, and does the opposite: it loops for ever if h says yes, and halts if h says no. By Turing completeness P is a combinator, and

$$\begin{aligned} P \text{ given itself halts} \quad &\text{iff} \quad h \text{ says that } P \text{ given itself does not halt} \\ &\text{iff} \quad P \text{ given itself does not halt,} \end{aligned}$$

which is impossible. □

Remark 8.3 (programs reading programs)*.* A combinator takes combinators as its inputs, not strings, so "given a combinator" and "given itself" need a convention. A program reads a combinator M as the string of `K`, `S` and brackets that spells it, that is, as a number, its code $\ulcorner M \urcorner$ (Definition 8.23); and "M given itself" is M applied to the numeral of its own code. With these readings the proof is exact: P, written in pseudocode, is a combinator by Fact 8.24, and h is asked about the combinator $P\,\underline{\ulcorner P \urcorner}$, which a program can write down.

With the theorem in hand we can show that no program decides validity in first-order logic, and hence that the first-order procedure `REFUTE`, which stops exactly on the unsatisfiable clause sets, is as good as any procedure can be: a semi-decision procedure, which can be improved only in speed. The proof writes the combinators as a theory, a set of sentences about combinators in the signature

$$\Sigma_{\mathrm{KS}} = (\{\mathsf{K}, \mathsf{S}\},\ \{\mathsf{ap}\},\ \{\mathsf{Step}, \mathsf{Nf}, \mathsf{Halts}\},\ \mathrm{ar}),$$

with $\mathrm{ar}(\mathsf{ap}) = \mathrm{ar}(\mathsf{Step}) = 2$ and $\mathrm{ar}(\mathsf{Nf}) = \mathrm{ar}(\mathsf{Halts}) = 1$. We write $\mathsf{ap}(x, y)$ as $x\,y$, as application is written, so that every combinator is a closed term; $\mathsf{Step}(x, y)$ says that x steps to y, $\mathsf{Nf}(x)$ that x is in normal form, and $\mathsf{Halts}(x)$ that x halts.

Each rule of Chapter 1 says that if the facts above the bar hold, so does the fact below. Read the bar as an implication, and the rule is a sentence.

[3] Every program in any programming language can be written as a combinator, and the Church–Turing thesis says that every procedure that could be called mechanical can be too. So "no program" means "no combinator" throughout this chapter, and each "no program" is a theorem about combinators. The thesis is used only where a program written in pseudocode is taken to be a combinator (Fact 8.24), and there it saves writing the combinator out. The impossibility results do not rest on it; it says only how far they reach.

[4] This is the fact usually called the undecidability of the *halting problem*. Most accounts credit it to Turing's paper of 1936, which did not state it. The History tells the story.

Steps

(K) $\mathsf{Step}(\mathsf{K}\,x\,y,\ x)$	(app_1) $\mathsf{Step}(x, x') \to \mathsf{Step}(x\,y,\ x'\,y)$
(S) $\mathsf{Step}(\mathsf{S}\,x\,y\,z,\ x\,z\,(y\,z))$	(app_2) $\mathsf{Step}(y, y') \to \mathsf{Step}(x\,y,\ x\,y')$

Normal forms

(N_1) $\mathsf{Nf}(\mathsf{K})$	(N_3) $\mathsf{Nf}(x) \to \mathsf{Nf}(\mathsf{K}\,x)$
(N_2) $\mathsf{Nf}(\mathsf{S})$	(N_4) $\mathsf{Nf}(x) \to \mathsf{Nf}(\mathsf{S}\,x)$
	(N_5) $(\mathsf{Nf}(x) \wedge \mathsf{Nf}(y)) \to \mathsf{Nf}(\mathsf{S}\,x\,y)$

Halting

(H_1) $\mathsf{Nf}(x) \to \mathsf{Halts}(x)$	(H_2) $(\mathsf{Step}(x, y) \wedge \mathsf{Halts}(y)) \to \mathsf{Halts}(x)$

FIGURE 8.1. The theory of computation.

For instance,

$$\frac{P \rightsquigarrow P'}{P\,Q \rightsquigarrow P'\,Q}\ \mathrm{app}_1 \qquad \text{becomes} \qquad \forall x\,\forall x'\,\forall y\,(\mathsf{Step}(x, x') \to \mathsf{Step}(x\,y,\ x'\,y)) :$$

the letters P, P' and Q, which stand for any combinators, become variables bound by $\forall$; the premiss becomes the antecedent; and the conclusion becomes the consequent. Formulas of this shape, with atomic premisses and one atomic conclusion, are called *Horn clauses*.

Definition 8.4 (the theory of computation)**.** The theory of computation is the rules for a step, for normal forms and for halting, written as sentences.

Formally, the *theory of computation* Π is the set of the universal closures of the eleven formulas of Σ_{KS} in Figure 8.1.

The four sentences for steps are Chapter 1's rules, the five for normal forms are the shapes of the shape lemma, and the two for halting say that a normal form halts and that halting passes back along a step. The theory is the same for every program: only the sentence $\mathsf{Halts}(M)$ mentions M.

Example 8.5 (a run, twice)**.** The combinator $\mathsf{K}\,\mathsf{K}\,\mathsf{S}\,\mathsf{S}$ steps to $\mathsf{K}\,\mathsf{S}$ by a K step in the function. By the rules of Chapter 1 the step is

$$\frac{\dfrac{}{\mathsf{K}\,\mathsf{K}\,\mathsf{S} \rightsquigarrow \mathsf{K}}\ \mathsf{K}}{\mathsf{K}\,\mathsf{K}\,\mathsf{S}\,\mathsf{S} \rightsquigarrow \mathsf{K}\,\mathsf{S}}\ \mathrm{app}_1$$

and in NK, from the sentences (K) and (app_1) of Π, it is

$$\frac{\dfrac{(\mathrm{app}_1)}{\mathsf{Step}(\mathsf{K}\,\mathsf{K}\,\mathsf{S},\ \mathsf{K}) \to \mathsf{Step}(\mathsf{K}\,\mathsf{K}\,\mathsf{S}\,\mathsf{S},\ \mathsf{K}\,\mathsf{S})}\ \forall\mathrm{E} \qquad \dfrac{(\mathsf{K})}{\mathsf{Step}(\mathsf{K}\,\mathsf{K}\,\mathsf{S},\ \mathsf{K})}\ \forall\mathrm{E}}{\mathsf{Step}(\mathsf{K}\,\mathsf{K}\,\mathsf{S}\,\mathsf{S},\ \mathsf{K}\,\mathsf{S})}\ \to\mathrm{E}$$

where a bar marked ($\forall$E) stands for as many ($\forall$E) steps as the sentence has quantifiers. Each rule becomes its sentence, instantiated by ($\forall$E) and applied by ($\to$E). The result $\mathsf{K}\,\mathsf{S}$ is in normal form, and the rest of the

derivation of $\mathsf{Halts}(\mathsf{K}\,\mathsf{K}\,\mathsf{S}\,\mathsf{S})$ goes the same way, one line for each rule used:

$\mathsf{Nf}(\mathsf{S})$	(N_2)
$\mathsf{Nf}(\mathsf{K}\,\mathsf{S})$	(N_3), with the line above
$\mathsf{Halts}(\mathsf{K}\,\mathsf{S})$	(H_1), with the line above
$\mathsf{Halts}(\mathsf{K}\,\mathsf{K}\,\mathsf{S}\,\mathsf{S})$	(H_2), with the step and the line above.

The theory Π is meant to describe computation, and the next lemma says that it gets halting right. Parts (i)–(iii) say that Π is strong enough: every step, every normal form and every halting combinator is recorded by a derivation, as in the example. Part (iv) says that Π is not too strong: it never proves that a combinator halts when it does not. So the sentence $\mathsf{Halts}(M)$ follows from Π exactly when M halts, and questions about halting become questions about first-order consequence.

Lemma 8.6 (a program is a theory)**.** *For all combinators M and N:*

(i) if $M \rightsquigarrow N$, then $\Pi \vdash_{\mathrm{NK}} \mathsf{Step}(M, N)$;

(ii) if M is in normal form, then $\Pi \vdash_{\mathrm{NK}} \mathsf{Nf}(M)$;

(iii) if M halts, then $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M)$;

(iv) if $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M)$, then M halts.

So M halts iff $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M)$, iff $\Pi \models \mathsf{Halts}(M)$.

Proof. Parts (i)–(iii) read the rules as sentences, as in the example.

- *(i).* By induction on the rules that stack up to $M \rightsquigarrow N$: each rule is an instance of one of (K), (S), (app_1), (app_2), reached by $(\forall\mathrm{E})$ and applied by $(\rightarrow\mathrm{E})$.
- *(ii).* By induction on M: by the shape lemma, M is K, S, $\mathsf{K}\,N_1$, $\mathsf{S}\,N_1$ or $\mathsf{S}\,N_1\,N_2$ with N_1, N_2 in normal form, and (N_1)–(N_5) cover the five cases.
- *(iii).* Take a run $M = M_0 \rightsquigarrow \cdots \rightsquigarrow M_k$ to a normal form. By (ii) and (H_1), $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M_k)$; by (i) and (H_2), $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M_{i+1})$ gives $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M_i)$, back to $i = 0$.
- *(iv).* Let $\mathfrak{T}$ be the structure whose domain is the set of combinators, in which each closed term means itself, and Step, Nf and Halts are true of the steps, the normal forms and the combinators that halt. Every sentence of Π is true in $\mathfrak{T}$: the four for steps are the rules, the five for normal forms are the shape lemma read from right to left, and a normal form halts, as does anything that steps to a combinator that halts. If $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M)$, then $\mathfrak{T} \models \mathsf{Halts}(M)$, by the soundness of NK, and so M halts.

The last sentence of the lemma is (iii) and (iv) together, with the soundness and completeness of NK. □

Theorem 8.7 (validity is undecidable)**.** *No program decides, for every sentence of Σ_{KS}, whether it is valid.*

Proof. Write $\bigwedge\Pi$ for the conjunction of the eleven sentences of Π. Given a combinator M, a program can write down the sentence $\bigwedge\Pi \rightarrow \mathsf{Halts}(M)$,

and by Lemma 8.6

$$M \text{ halts} \qquad \text{iff} \qquad \models \bigwedge \Pi \to \mathsf{Halts}(M).$$

A program that decided validity would therefore check whether a given combinator halts, and there is none (Theorem 8.2). □

A larger signature does not help: a program that decided validity for it would decide, in particular, the sentences of Σ_{KS}. The proof is a *reduction*: a computable map from halting questions to questions of the new kind, with the same answers. Every undecidability result of the chapter is proved by one, and the work is always in checking that the answers agree (here, Lemma 8.6).

Corollary 8.8 (other forms of the question)**.** *No program decides, for every input of the following kinds, the question asked:*

(i) given a finite set Γ *of sentences and a sentence* φ*, whether* $\Gamma \vdash_{\mathrm{NK}} \varphi$*;*

(ii) given a finite set of sentences, whether it is satisfiable;

(iii) given a finite set of clauses, whether it is satisfiable.

Proof. Take $\Gamma = \Pi$ and $\varphi = \mathsf{Halts}(M)$ for (i). For (ii) and (iii), $\Pi \cup \{\neg\mathsf{Halts}(M)\}$, a set of clauses once the quantifiers are dropped, is unsatisfiable exactly when M halts. □

In (i) the set Γ is always Π: there is one finite set of sentences whose consequences no program decides.

This settles the search gap. The first-order procedure `REFUTE` stops on every unsatisfiable clause set, and so does `REFUTE-U`. A procedure that also stopped on every satisfiable set would decide satisfiability, and by part (iii) of the corollary there is none. A cleverer search can be faster, but it cannot gain the other half of the answer. The valid sentences can be listed, by running `REFUTE` on the negations of all sentences at once and printing those whose runs close, so they have a semi-decision procedure; but they cannot be decided.[5] Membership can be confirmed; non-membership, in general, cannot.

Remark 8.9 (what remains decidable)*.* None of this makes logic useless in practice. Many questions can still be decided.

- Propositional logic is decided by `DPLL`.
- Whether finitely many sentences have a model with n elements, for a given n, is a propositional question for a SAT solver.
- Many restricted languages have decision procedures of their own, and some of them close this chapter.

No method decides every first-order question, and the tools in use make one of three choices.

- They restrict the language to a part that can be decided.
- They accept a search that may not stop, as `REFUTE` does.
- They keep a person in the loop, as the interactive provers built on type theory do.

[5] Textbooks call such a set *computably enumerable*, or *semi-decidable*.

8.2. **Specifications.** The specification gap asks whether a theory can say everything true of the system it describes, and Π is a first test case. It describes a structure that was there before any theory of it: computation itself. In the language Σ_{KS}, computation is the structure $\mathfrak{T}$ of the proof of Lemma 8.6: its domain is the set of combinators, each closed term means itself, and

$$\begin{aligned} [\![\mathsf{Step}]\!]^{\mathfrak{T}} &= \{(M,N) : M \rightsquigarrow N\}, \\ [\![\mathsf{Nf}]\!]^{\mathfrak{T}} &= \{M : M \text{ is in normal form}\}, \\ [\![\mathsf{Halts}]\!]^{\mathfrak{T}} &= \{M : M \text{ halts}\}. \end{aligned}$$

The structure deserves its name. Every program, in any programming language, can be written as a combinator, and running the program is then running the combinator: that is Turing completeness. So a question about what a program does is a question about $\mathfrak{T}$, and the combinators *express* everything there is to compute.[6]

Does expressing everything mean that Π says everything? The theory Π proves only truths about $\mathfrak{T}$, since $\mathfrak{T}$ is a model of it, and it proves $\mathsf{Halts}(M)$ for every combinator M that halts. But it does not prove everything true there. The Monster Ω of Chapter 1 has no normal form, and

$$\mathfrak{T} \models \neg\mathsf{Halts}(\Omega) \qquad \text{but} \qquad \Pi \nvdash_{\mathrm{NK}} \neg\mathsf{Halts}(\Omega),$$

since in the structure where every combinator halts the sentences of Π are true, and so is $\mathsf{Halts}(\Omega)$ (Exercise 8.3). So a language that can express every computation comes with a theory that leaves open some questions that computation settles.

Could Π be extended to close the gap? In a trivial way it can: the set of all sentences true in $\mathfrak{T}$ proves every truth about $\mathfrak{T}$, each in one line. But a theory is of use only if it can be written down, which here means that a program can recognise its members. Nothing guarantees that for the set of all truths. So the question is whether some theory that *can* be written down says everything true of $\mathfrak{T}$. The answer is no. To prove it we make three notions precise: the set of all truths, a theory that can be written down, and a theory that says everything.

Definition 8.10 (the theory of a structure)**.** The theory of a structure is the set of everything true of it.

Formally, for a Σ-structure $\mathfrak{M}$,

$$\mathrm{Th}(\mathfrak{M}) \coloneqq \{\varphi : \varphi \text{ is a } \Sigma\text{-sentence and } \mathfrak{M} \models \varphi\}.$$

For every sentence φ, exactly one of φ and $\neg\varphi$ is in $\mathrm{Th}(\mathfrak{M})$, since $\mathfrak{M} \models \neg\varphi$ exactly when $\mathfrak{M} \not\models \varphi$: a structure takes a side on every question its language can ask. That is what distinguishes a structure from a theory, which may leave a question open.

[6] Computation is also presented as a structure whose elements are the programs of a language and whose one operation runs one program on another, partial because a run may not end; with the programs taken to be numbers, it is known as Kleene's first algebra. It has programs that behave as K and S, which is how the combinators name its elements. The chapter works with $\mathfrak{T}$, which speaks the language of Π.

To be of use, a theory needs one thing besides its sentences: a program that recognises its members.

Definition 8.11 (specification)**.** A specification is a theory that a checker can read: given a sentence, the checker tells whether it belongs to the theory.

Formally, let Σ be a finite signature. Its sentences are then finite strings of symbols from a finite list, once each variable is written as x followed by digits. A Σ-theory Θ is a *specification* if there is a program that, given a Σ-sentence, always halts and says whether it is in Θ.[7]

The sets one would naturally write down are specifications.

- A finite set of sentences is a specification: the program compares the given sentence with each of them. The theory of computation Π is one.
- The equality axioms of a signature that contains $=$ are a specification. There are infinitely many instances of (Eq2), one for each formula, but a program recognises an instance by its shape.
- In the same way, every finite set of sentences together with all the instances of finitely many schemas is a specification.

The definition asks for a program because the point of a derivation from Θ is that anyone can check it, and checking it includes checking that its open hypotheses are in Θ.

Remark 8.12 (a list of axioms is enough). One could ask for less: that a program merely list the members of Θ, one after another. Nothing would change. If a program lists the sentences $\varphi_0, \varphi_1, \varphi_2, \ldots$, then the sentences $\varphi_n \wedge \cdots \wedge \varphi_n$, with $n+1$ copies of φ_n, form a specification with the same theorems. Each is equivalent to φ_n, and a program recognises one by counting its copies and running the list that far.[8] So "a theory that can be written down" means the same on either reading.

Definition 8.13 (captures)**.** A specification captures a structure when what follows from it is exactly what is true of the structure.

Formally, a specification Θ *captures* a Σ-structure $\mathfrak{M}$ if, for every Σ-sentence φ,

$$\Theta \vdash_{\mathrm{NK}} \varphi \qquad \text{iff} \qquad \mathfrak{M} \models \varphi.$$

The definition has two halves, of very different weight.

- *Only truths:* if $\Theta \vdash_{\mathrm{NK}} \varphi$ then $\mathfrak{M} \models \varphi$. When $\mathfrak{M} \models \Theta$ this half is free, by the soundness of NK.
- *Every truth:* if $\mathfrak{M} \models \varphi$ then $\Theta \vdash_{\mathrm{NK}} \varphi$. This is the half with content: it asks the theory to settle every question that the structure settles.

So Π, a specification since it is finite, does not capture $\mathfrak{T}$: it proves only truths about $\mathfrak{T}$, but not $\neg\mathsf{Halts}(\Omega)$. The set $\mathrm{Th}(\mathfrak{T})$ proves exactly the truths about $\mathfrak{T}$, trivially, and would capture $\mathfrak{T}$ if it were a specification. The question is whether some specification captures $\mathfrak{T}$.

[7] Textbooks call such a set *decidable*, or *recursive*, and a theory with a decidable set of axioms *axiomatizable*.

[8] The trick is due to Craig (1953).

Remark 8.14 (three senses of "complete"). The word "complete" has three uses nearby, and they are easy to confuse.

- *The completeness of* NK is a property of a proof system: whatever is true in *every* model of Θ can be derived from Θ.
- *The second half of capturing*, which many books call the completeness of Θ, is a property of a theory: whatever is true in *the one structure we meant* can be derived from Θ. When every sentence true in $\mathfrak{M}$ is true in every model of Θ, the completeness of NK gives it; the gap between the two is the specification gap. To keep the senses apart we say "captures", and never "complete", for this one.
- *Turing completeness* is a property of a programming language: every program can be written in it. It says that the combinators can *express* every computation, not that a theory of them *proves* every truth about computation.

The last two pull against each other. Because the combinators can express every program, the diagonal program of the halting theorem among them, no specification captures computation (Corollary 8.17).

What a specification makes possible is a mechanical list of its theorems. A program can recognise the members of a specification, and a program can check a derivation, rule by rule. So a program can list the theorems of a specification, one after another, by checking every candidate derivation in turn.[9]

Lemma 8.15 (a specification's theorems can be listed)**.** *For every specification Θ there is a program that runs for ever and prints sentences, such that the sentences it prints are exactly the sentences φ with $\Theta \vdash_{\mathrm{NK}} \varphi$.*

Proof. A derivation of NK is a finite tree of formulas, and can be written out as a string of symbols. The program tries every string.

```
procedure LIST                        -- prints the theorems of Theta
    for each string w, shortest first, then in alphabetical order:
        if w is a derivation of NK,
           each open hypothesis of w is in Theta,
           and the conclusion of w is a sentence:
            print the conclusion of w
```

Each test halts: a derivation is checked node by node against the finitely many rules and their side conditions, and membership of Θ is asked of the program for Θ. So every string is dealt with in finite time. A sentence φ with $\Theta \vdash_{\mathrm{NK}} \varphi$ has a derivation, which is some string, and is printed when `LIST` reaches it. A sentence that is not derivable is never printed. □

Proofs are easy to check and hard to find, and a program that checks every candidate finds them all, eventually. The list says nothing about a sentence that is not on it, however long one waits, and that is the whole difficulty. But a structure takes a side on every question, so a specification that captured it would sooner or later list either φ or $\neg\varphi$, for every sentence φ.

[9] A set that a program can list in this way is called *recursively enumerable*, or *computably enumerable*.

Theorem 8.16 (capturing makes truth decidable)**.** *If a specification Θ captures a structure $\mathfrak{M}$, there is a program that, given a sentence φ, always halts and says correctly whether $\mathfrak{M} \models \varphi$.*

Proof. The program waits for φ or its negation to be printed.

```
procedure DECIDE(phi)                -- phi a sentence
    for each theorem psi of Theta, as LIST prints them:
        if psi is phi:                      answer yes
        if psi is the negation of phi:      answer no
```

It halts: either $\mathfrak{M} \models \varphi$ or $\mathfrak{M} \models \neg\varphi$, and Θ proves every truth about $\mathfrak{M}$, so one of the two is printed. Its answer is correct.

- *If φ is printed*, then $\Theta \vdash_{\mathrm{NK}} \varphi$. Since Θ captures $\mathfrak{M}$, it proves only truths, so $\mathfrak{M} \models \varphi$, and the answer is yes.
- *If $\neg\varphi$ is printed*, then in the same way $\mathfrak{M} \models \neg\varphi$, and the answer is no. □

A specification that captures a structure turns truth into computation. Read backwards, the theorem is a way to show that a structure cannot be captured:

if no program decides the truths of $\mathfrak{M}$, then no specification captures $\mathfrak{M}$.

That answers the question about computation.

Corollary 8.17 (computation cannot be captured)**.** *No specification captures the structure $\mathfrak{T}$ of computation. In particular,* $\mathrm{Th}(\mathfrak{T})$ *is not a specification.*

Proof. For every combinator M, exactly one of $\mathsf{Halts}(M)$ and $\neg\mathsf{Halts}(M)$ is true in $\mathfrak{T}$, according as M halts or not. A program that decided the truths of $\mathfrak{T}$ would therefore check whether a given combinator halts, and there is none (Theorem 8.2). So by Theorem 8.16 no specification captures $\mathfrak{T}$, and since $\mathrm{Th}(\mathfrak{T})$ proves exactly the truths about $\mathfrak{T}$, it is not a specification. □

For Π we can watch the mechanism. The listing of its theorems prints $\mathsf{Halts}(M)$ for every combinator M that halts, and never prints $\neg\mathsf{Halts}(M)$. A specification that captured $\mathfrak{T}$ would also print $\neg\mathsf{Halts}(M)$ for every M that does not halt, and waiting for one of the two would decide halting. So the questions that Π leaves open are not a defect of Π: every specification of computation that tells the truth leaves some truth unproved.

The theorem that capturing makes truth decidable has a converse, and together they say that capturing and deciding are the same thing.

Proposition 8.18 (capturing is deciding)**.** *A structure $\mathfrak{M}$ is captured by some specification if and only if some program, given a sentence φ, always halts and says correctly whether $\mathfrak{M} \models \varphi$.*

Proof. One direction is Theorem 8.16. For the other, suppose a program decides the truths of $\mathfrak{M}$. Then $\mathrm{Th}(\mathfrak{M})$ is a specification, and it captures $\mathfrak{M}$.

- *Only truths.* Whatever it proves is true, by the soundness of NK, since $\mathfrak{M} \models \mathrm{Th}(\mathfrak{M})$.
- *Every truth.* Whatever is true is a member, and so is proved by the derivation consisting of that one hypothesis. □

What a good specification adds is economy: a short list that a person can write, read and argue about, in place of the list of every truth.

Remark 8.19 (deciding with `REFUTE`). The program `DECIDE` searches through derivations, which is slow. The first-order procedure `REFUTE` does the same job faster. Suppose a specification Θ that captures $\mathfrak{M}$ can be written as a finite set of clauses, and let φ be a sentence. Run `REFUTE` on $\Theta \cup \{\neg\varphi\}$ and on $\Theta \cup \{\varphi\}$, taking a step of each run in turn.

- The first run stops exactly when $\Theta \models \varphi$, and the second exactly when $\Theta \models \neg\varphi$.
- One of φ and $\neg\varphi$ is true in $\mathfrak{M}$ and so, because Θ captures $\mathfrak{M}$, is entailed by Θ.

So one of the two runs stops, and which one says whether $\mathfrak{M} \models \varphi$.

Remark 8.20 (finite structures can be captured). Finite structures are the one kind for which the specification gap can always be closed. On a finite structure the definition of truth is an algorithm: each quantifier runs through the domain, so a sentence with k nested quantifiers on n elements takes about n^k cases. So its truths can be decided, and by Proposition 8.18 it can be captured. A finite list will do, and it is the list anyone would write. Give each element d a new name c_d, and write down

- *distinctness*: $c_d \neq c_e$ for each pair of different elements d and e;
- *closure*: $\forall x\,(x = c_{d_1} \vee \cdots \vee x = c_{d_n})$, where $d_1, \ldots, d_n$ are all the elements;
- *the tables*: $f(c_{e_1}, \ldots, c_{e_m}) = c_e$ whenever f sends $e_1, \ldots, e_m$ to e, and $P(c_{e_1}, \ldots, c_{e_m})$ or its negation, for each relation symbol P other than $=$ and each tuple.

These sentences, the *description* $\Delta_{\mathfrak{M}}$ of the structure $\mathfrak{M}$, together with the equality axioms, capture $\mathfrak{M}$ with its elements named. The proof is an induction on sentences: under the closure sentence every $\forall$ becomes a finite conjunction and every $\exists$ a finite disjunction, and what is left are entries of the tables. The closure sentence is the one with no counterpart for an infinite structure. It says where the structure stops, and without it a model may have an extra element of which the tables say nothing (Exercise 8.10).

The section has pulled apart four things that are easy to run together.

- *Expressing.* The combinators express every program: that is Turing completeness, a property of a language.
- *Proving what happens.* The theory Π proves nothing false about $\mathfrak{T}$, and proves $\mathsf{Halts}(M)$ for every M that halts; but it does not prove $\neg\mathsf{Halts}(M)$ for every M that does not.
- *Proving everything, with some theory.* The set $\mathrm{Th}(\mathfrak{M})$ of all truths proves exactly the truths about any structure $\mathfrak{M}$, trivially. It captures $\mathfrak{M}$ only if it is a specification, that is, only if a program can recognise its members.
- *Capturing by a specification.* This is the question that matters, and the answer depends on the structure. Every finite structure can be captured (Remark 8.20). Computation cannot, because capturing would decide its truths, and halting cannot be decided.

The rest of the chapter asks the last question of the counting numbers.

8.3. **The First Incompleteness Theorem.** Computation cannot be captured, but computation is a strange structure, with a partial operation and runs that never end. Surely something as simple as the counting numbers can be captured? They are the structure the founders of modern logic most wanted to capture, and the oldest one in mathematics.

They cannot be, and the reason is that the counting numbers contain computation: the theory Π, and with it everything that can be said about combinators, can be carried inside arithmetic. Combinators can be written as numbers, and stepping, being in normal form and halting as relations between numbers, each expressed by a formula built from + and ×. Then every sentence φ about computation has a translation $[\varphi]$, a sentence about numbers, with

$$\mathfrak{T} \models \varphi \qquad \text{iff} \qquad \mathfrak{N} \models [\varphi],$$

and a program can compute $[\varphi]$ from φ. A specification that captured the counting numbers would let a program decide their truths, and so, through the translation, the truths of computation, which no program can decide (Corollary 8.17). The section fixes the language of the counting numbers and makes this precise in three steps: combinators as numbers, steps as formulas, and the translation. It then names a truth that a given specification misses: the first incompleteness theorem.

The counting numbers. The language is that of the counting numbers in Chapter 6, with equality added, since without it the language could not even say $x + y = y + x$.

Definition 8.21 (the signature of arithmetic, with equality)**.** In this chapter the signature of arithmetic contains $=$, and the counting numbers interpret it as identity.

Formally,

$$\Sigma_{\mathrm{A}} = (\{0\},\ \{s, +, \times\},\ \{<, =\},\ \mathrm{ar}),$$

with $\mathrm{ar}(s) = 1$ and arity 2 for the others. The *standard model of arithmetic* $\mathfrak{N}$, as the structure is traditionally called, has the domain $\mathbb{N} = \{0, 1, 2, \ldots\}$, and interprets

- 0 as zero and s as the successor, $n \mapsto n + 1$;
- $+$ and $\times$ as addition and multiplication;
- $<$ as *less than* and $=$ as identity.

The language has no symbol for 1, or for 7. It does not need one, since it can count.

Definition 8.22 (numeral)**.** The numeral of a number is the term that counts up to it from 0.

Formally, for each number n the *numeral* $\overline{n}$ is the closed term of Σ_{A} defined by recursion on n:

$$\overline{0} \coloneqq 0 \qquad \text{and} \qquad \overline{n+1} \coloneqq s(\overline{n}).$$

So $\overline{3}$ is $s(s(s(0)))$, and in general $\overline{n}$ has n copies of s. Its value in $\mathfrak{N}$ is n, by induction on n.

Unlike a finite structure, the counting numbers cannot be written out as a table, since their domain is infinite: they can only be described by a theory.

The language is small, but it says a great deal. Write $x \mid y$ for $\exists z\,(x \times z = y)$, "$x$ divides y", and

$$\mathit{Prime}(x) \coloneqq \overline{1} < x \wedge \forall y\,(y \mid x \to (y = \overline{1} \vee y = x)),$$
$$\mathit{Even}(x) \coloneqq \exists z\,(x = z + z).$$

With these abbreviations, the sentence $\forall x\,\exists p\,(x < p \wedge \mathit{Prime}(p))$ says that there are infinitely many primes, which is true. The sentence

$$\forall x\,((\mathit{Even}(x) \wedge \overline{2} < x) \to \exists p\,\exists q\,(\mathit{Prime}(p) \wedge \mathit{Prime}(q) \wedge p + q = x))$$

is Goldbach's conjecture. A specification that captured $\mathfrak{N}$ would derive it or its negation, and the same holds for most open problems of number theory, which can be written in this language too.

Combinators as numbers. The first step is to write combinators as numbers, and it goes in three moves. A combinator is a string of the four symbols `K`, `S`, (and), once every application is bracketed; a string of four symbols is a sequence of the digits 1 to 4; and a sequence of digits is a number, written in base five.

Definition 8.23 (code)**.** The code of a combinator is the number its fully bracketed string spells in base five.

Formally, the *code* $\ulcorner M \urcorner$ of a combinator M is the number whose digits in base 5 are the symbols of M written with every application bracketed, read with 1 for `K`, 2 for `S`, 3 for (and 4 for).

Here are three combinators and their codes.

Combinator	String	Digits	Code
K	`K`	1	1
K K	(`KK`)	3 1 1 4	409
S K K	((`SK`)`K`)	3 3 2 1 4 1 4	57734

Different combinators have different codes, and a program can pass between a combinator and its code in either direction. Inputs are numbers too: the input for the number n is Chapter 1's numeral of n, written here with a bar beneath, $\underline{n}$, since the bar above is taken by the numerals of arithmetic. A combinator M *halts on* n if $M\,\underline{n}$ halts, and for numbers e and n we say that *program* e *halts on* n if e is the code of a combinator that halts on n.

Programs are easier to write in pseudocode, and one fact, borrowed from Chapter 1, lets us do so.

Fact 8.24 (programs are combinators). For every program written in pseudocode that takes a number as its input, there is a combinator P such that, for every number n, P halts on n exactly when the pseudocode halts on n.[10]

We call P the *translation* of the pseudocode, and do not distinguish the two: they halt on the same inputs, and the code of the pseudocode is the code of its translation.

[10] Turing completeness is stated for the functions that combinators compute. The form used here, for programs that may run for ever, is the one proved with it, and it is what the Church–Turing thesis says of the pseudocode of this chapter.

Steps as formulas. The second step is to express the relations of computation by formulas of arithmetic.

Definition 8.25 (definable)**.** A relation between numbers is definable if some formula of arithmetic is true of exactly the numbers that stand in it.

Formally, a set S of k-tuples of numbers is *definable* in $\mathfrak{N}$ if there is a formula φ of Σ_{A} whose free variables are among $x_1, \ldots, x_k$ such that, for all numbers $n_1, \ldots, n_k$,

$$\mathfrak{N} \models \varphi[\overline{n_1}/x_1, \ldots, \overline{n_k}/x_k] \qquad \text{iff} \qquad (n_1, \ldots, n_k) \in S.$$

The even numbers are definable, by $\mathit{Even}(x)$, and so are the primes and the pairs (x, y) with $x \mid y$.

Definability is a matter of truth in $\mathfrak{N}$, and says nothing about proofs. A formula that defines a set is true of its members; whether any specification can *prove* it true of them is a different question, about the specification. The notion that asks for proofs has a name of its own, *representability*, and the main line of this chapter never uses it. Section 8.4 needs two facts about what one particular specification proves, and states them.

Why should the relations of computation be definable at all? A combinator is a row of digits, and one step rewrites one stretch of the row and copies the rest. In the step from $\mathsf{S}\,(\mathsf{K}\,\mathsf{K}\,\mathsf{S})$ to $\mathsf{S}\,\mathsf{K}$, for instance, the strings are

$$(\mathtt{S}\ \underbrace{(\mathtt{(KK)S})}_{\mathsf{K}\text{ redex}}\ ) \ \rightsquigarrow\ (\mathtt{S}\ \underbrace{\mathtt{K}}_{P}\ ),$$

where the stretch $((\mathtt{K}\,P)\,Q)$, with $P = \mathtt{K}$ and $Q = \mathtt{S}$, has been replaced by P. So "x steps to y" says something about the digits of x and y at certain positions: that they agree outside one stretch, and inside it have the shape of a redex and of its result. Digits at positions are remainders, and "r is the remainder of a on division by m" is the formula

$$\exists q\,(a = q \times m + r \wedge r < m).$$

That takes care of being a combinator, of one step, and of normal form.

Halting is harder, because it says that *some* run ends, and a run may be as long as you like. A run is a finite table of numbers, one row of digits for each combinator. The one idea needed is that a finite table of numbers can be packed into two numbers a and b, with the entry at position i, counting from 0, the remainder of a on division by $1 + (i+1)\,b$. Then "there is a run that ends" is "there are numbers a and b whose table is a run that ends", which is a statement about remainders again.

Example 8.26 (a table in two numbers)**.** The table with the single row $2, 0, 3$ is packed by $a = 80$ and $b = 2$, with the divisors $1 + b = 3$, $1 + 2b = 5$ and $1 + 3b = 7$:

$$80 = 26 \times 3 + 2, \qquad 80 = 16 \times 5 + 0, \qquad 80 = 11 \times 7 + 3.$$

The remainders are the entries, in order. Section 8.5 proves that every finite table can be packed in this way.

Lemma 8.27 (computation is definable)**.** *There are formulas $\mathit{Comb}(x)$, $\mathit{App}(x, y, z)$, $\mathsf{Step}_{\mathrm{A}}(x, y)$, $\mathsf{Nf}_{\mathrm{A}}(x)$ and $\mathsf{Halts}_{\mathrm{A}}(x)$ of Σ_{A} such that, for every number p and all combinators M and N:*

(i) $\mathfrak{N} \models \mathit{Comb}(\overline{p})$ *iff* p *is the code of a combinator;*
(ii) $\mathfrak{N} \models \mathit{App}(\overline{\ulcorner M \urcorner}, \overline{\ulcorner N \urcorner}, \overline{p})$ *iff* $p = \ulcorner M\, N \urcorner$*;*
(iii) $\mathfrak{N} \models \mathsf{Step}_{\mathrm{A}}(\overline{\ulcorner M \urcorner}, \overline{\ulcorner N \urcorner})$ *iff* $M \rightsquigarrow N$*;*
(iv) $\mathfrak{N} \models \mathsf{Nf}_{\mathrm{A}}(\overline{\ulcorner M \urcorner})$ *iff* M *is in normal form;*
(v) $\mathfrak{N} \models \mathsf{Halts}_{\mathrm{A}}(\overline{\ulcorner M \urcorner})$ *iff* M *halts.*

The proof is in the optional Section 8.5, which writes out the formulas; the main line uses only the lemma. One feature of $\mathsf{Halts}_{\mathrm{A}}$ matters later. A combinator can have many runs, and the formula follows one of them, the *leftmost run*, which at each step rewrites the redex that starts first in the written-out combinator. That run is unique, and it reaches a normal form whenever any run does, so part (v) holds as stated.

The translation. The third step puts the formulas together. A sentence about combinators speaks of terms built from K and S by application, and of the relations Step, Nf and Halts; its translation speaks of their codes.

Definition 8.28 (the translation into arithmetic)**.** The translation of a formula about combinators says the same about their codes.

Formally, for each term t of Σ_{KS} and each variable z not in t, the formula $C_t(z)$, "z is the code of t", is given by

$$\begin{aligned} C_{\mathsf{K}}(z) &\coloneqq z = \overline{1}, \\ C_{\mathsf{S}}(z) &\coloneqq z = \overline{2}, \\ C_x(z) &\coloneqq z = x, \\ C_{t\,u}(z) &\coloneqq \exists v\, \exists w\, \big(C_t(v) \wedge C_u(w) \wedge \mathit{App}(v, w, z)\big), \end{aligned}$$

with v and w new variables. The *translation* $[\varphi]$ of a formula φ of Σ_{KS} is given by

$$\begin{aligned} [\mathsf{Step}(t,u)] &\coloneqq \exists v\, \exists w\, \big(C_t(v) \wedge C_u(w) \wedge \mathsf{Step}_{\mathrm{A}}(v, w)\big), \\ [\mathsf{Nf}(t)] &\coloneqq \exists v\, \big(C_t(v) \wedge \mathsf{Nf}_{\mathrm{A}}(v)\big), \\ [\mathsf{Halts}(t)] &\coloneqq \exists v\, \big(C_t(v) \wedge \mathsf{Halts}_{\mathrm{A}}(v)\big), \\ [\bot] &\coloneqq \bot, \\ [\psi \,\square\, \chi] &\coloneqq [\psi] \,\square\, [\chi] \qquad \text{for } \square \text{ one of } \wedge, \vee, \to, \\ [\forall x\, \psi] &\coloneqq \forall x\, \big(\mathit{Comb}(x) \to [\psi]\big), \\ [\exists x\, \psi] &\coloneqq \exists x\, \big(\mathit{Comb}(x) \wedge [\psi]\big). \end{aligned}$$

The quantifiers are cut down to the codes of combinators, and everything else is said of codes. For instance, $C_{\mathsf{K}\,\mathsf{K}}(z)$ is true in $\mathfrak{N}$ of 409 alone, so $[\mathsf{Halts}(\mathsf{K}\,\mathsf{K})]$ says that $\mathsf{Halts}_{\mathrm{A}}$ is true of 409; and since $\neg\psi$ is $\psi \to \bot$, the translation of $\neg\psi$ is $\neg[\psi]$.

Lemma 8.29 (arithmetic contains computation)**.** *For every sentence* φ *of* Σ_{KS}*,*

$$\mathfrak{T} \models \varphi \qquad \text{iff} \qquad \mathfrak{N} \models [\varphi],$$

and a program, given φ*, computes* $[\varphi]$*.*

Proof. The program follows the equations of the definition. For the rest we prove more, for formulas. Given an assignment s in $\mathfrak{T}$, which gives each

variable a combinator, let $s^\sharp$ give each variable the code of that combinator. We show that $\mathfrak{T}, s \models \varphi$ exactly when $\mathfrak{N}, s^\sharp \models [\varphi]$.

- *Terms.* For each term t, the formula $C_t(z)$ is true in $\mathfrak{N}$, under $s^\sharp$, of exactly one number, the code of the value of t under s. This is by induction on t, with part (ii) of the lemma that computation is definable for an application.
- *Atomic formulas.* By the case of terms and parts (iii)–(v).
- $\bot$ *and the connectives.* The translation leaves them alone, and so does satisfaction.
- *The quantifiers.* By part (i), the numbers of which *Comb* is true are exactly the codes of combinators, so "for every number that is a code" in $\mathfrak{N}$ is "for every combinator" in $\mathfrak{T}$, and the same for "some". □

In words: everything true of computation is true, in translation, of the counting numbers, and nothing else is. In particular the translations of the sentences of Π are all true in $\mathfrak{N}$: arithmetic carries a copy of the theory of computation.

Nothing captures arithmetic. The rest is the argument sketched at the start of this section.

Theorem 8.30 (the truths of arithmetic are undecidable)**.** *No program decides, for every sentence φ of Σ_A, whether $\mathfrak{N} \models \varphi$.*

Proof. A program that decided truth in $\mathfrak{N}$ would decide truth in $\mathfrak{T}$: given φ, it would compute $[\varphi]$ and decide that (Lemma 8.29). In particular, given a combinator M, it would decide $\mathfrak{T} \models \mathsf{Halts}(M)$, that is, whether M halts, and no program does (Theorem 8.2). □

Corollary 8.31 (nothing captures arithmetic)**.** *No specification captures $\mathfrak{N}$. In particular, for every specification Θ with $\mathfrak{N} \models \Theta$, there is a sentence φ with $\mathfrak{N} \models \varphi$ and $\Theta \nvdash_\mathrm{NK} \varphi$.*

Proof. A specification that captured $\mathfrak{N}$ would make the truths of $\mathfrak{N}$ decidable, by Theorem 8.16, and no program decides them, by Theorem 8.30. If $\mathfrak{N} \models \Theta$, then Θ proves only truths, by the soundness of NK; since it does not capture $\mathfrak{N}$, it must miss a truth. □

So $\mathrm{Th}(\mathfrak{N})$, the set of all true sentences of arithmetic, is not a specification. However carefully a list of axioms for arithmetic is chosen, if a program can read it and it proves only truths, it misses some truth.

Naming the missed truth. The corollary says that a specification misses a truth, but gives no idea which. For the theory Π the missed truth was a fact about a program that does not halt, $\neg\mathsf{Halts}(\Omega)$, and for arithmetic it will be the translation of one. We assume that the specification tells the truth, and write a program that watches it at work.

Definition 8.32 (sound)**.** A specification Θ in Σ_A is *sound* if $\Theta \vdash_\mathrm{NK} \varphi$ implies $\mathfrak{N} \models \varphi$, for every sentence φ.

Here the word names a property of a specification, not to be confused with the soundness of NK, a property of the proof system: whatever NK derives from true premisses is true. The two are connected: by the soundness

of NK, a specification Θ is sound exactly when $\mathfrak{N} \models \Theta$. A sound specification has the "only truths" half of capturing $\mathfrak{N}$ (Definition 8.13); the first incompleteness theorem is about the other half.

The program that watches the specification waits for it to say that some program does not halt on its own code.

Definition 8.33 (the diagonal program)**.** Let Θ be a specification in Σ_{A}. The *diagonal program* D_Θ of Θ is the following program, which uses the program `LIST` of Lemma 8.15 to list the theorems of Θ:

```
procedure D(e)                          -- e a number
    if e is the code of a combinator M:
        for each theorem phi of Theta, as LIST prints them:
            if phi is  [not Halts(M e)],  e as a numeral:  halt
```

We write d for its code, and γ_Θ, or γ, for the sentence $[\neg\mathsf{Halts}(D_\Theta\,\underline{d})]$.

On the input e, the program waits for Θ to prove that program e does not halt on e, and halts if that ever happens. The interesting input is the program's own code, and the sentence γ says, in arithmetic, that D run on its own code does not halt.

Theorem 8.34 (the first incompleteness theorem)**.** *If Θ is a sound specification in Σ_{A}, then*

$$\mathfrak{N} \models \gamma_\Theta \qquad \textit{and} \qquad \Theta \nvdash_{\mathrm{NK}} \gamma_\Theta.$$

Proof. Write D for the diagonal program of Θ and d for its code, and run D on d:

$\Theta \vdash_{\mathrm{NK}} \gamma$	iff	$D\,\underline{d}$ halts	the definition of D, Lemma 8.15
	iff	$\mathfrak{T} \not\models \neg\mathsf{Halts}(D\,\underline{d})$	
	iff	$\mathfrak{N} \not\models \gamma$	Lemma 8.29.

So Θ proves the sentence exactly when the sentence is false. Since Θ is sound, it proves nothing false, so it does not prove the sentence; and then, by the same chain, the sentence is true. □

The proof used three things: that Θ proves only truths, the half of capturing that soundness gives; that its theorems can be listed (Lemma 8.15); and that arithmetic contains computation (Lemma 8.29). It used no particular axioms of arithmetic. Where Corollary 8.31 says only that some truth is missed, the diagonal program points at one: the translation of a fact about a program that never halts, as $\neg\mathsf{Halts}(\Omega)$ was for Π.

Figure 8.2 draws the theorem. Among the sentences of Σ_{A}, the theorems of a sound Θ lie inside the truths. The sentence γ lies in the gap between them, and its negation is false, so Θ proves neither. Adding γ to Θ widens the theorems to take it in, but the wider specification is again sound and has its own diagonal program, whose sentence lies in the new gap. There is no last patch.

The theorem has two hypotheses on Θ: that it is a specification, and that it is sound. Both are needed, and one example for each shows why.

- *A specification.* The set $\mathrm{Th}(\mathfrak{N})$ proves exactly the truths, so it leaves no truth unproved. It is not a specification (Corollary 8.31), so it has no program `LIST`, and its diagonal program cannot be written.

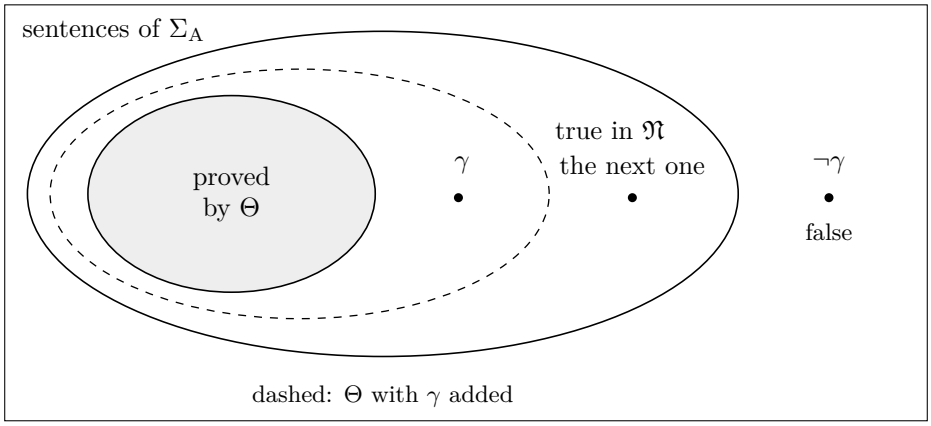

FIGURE 8.2. A sound specification and the truths.

- *Sound.* A specification that contains a sentence and its negation proves every sentence, by ($\rightarrow$E) and ($\bot$E), and so it proves its own γ. It is a specification, and not sound.

Remark 8.35 (what the theorem shows, and what it does not). The theorem is often quoted, and often quoted wrongly.

- *It shows* that no specification of arithmetic that tells the truth tells the whole truth. However the axioms are chosen, provided a checker can read them and they prove nothing false, some true sentence of arithmetic does not follow from them, and the proof says which.
- *It does not show* that there are truths that nobody can know. We have just proved γ true, by an argument that uses the soundness of Θ, which Θ cannot supply for itself.

Remark 8.36 (the sentence speaks about itself). The first line of the proof's chain reads: Θ proves γ exactly when $D\,\underline{d}$ halts. Read in $\mathfrak{N}$, the sentence γ says that $D\,\underline{d}$ does not halt, and so it says, of itself, "Θ does not prove me". That is, in another dress, the sentence with which the incompleteness theorem was first proved. There the self-reference was built inside arithmetic: formulas were coded as numbers, and substituting a formula's own number into it gave a sentence that says of itself that it is not provable. Here the self-reference costs nothing: it is D run on its own code.

It is the same move as in the halting theorem, the one essential idea of the subject: apply a thing to its own description, and arrange for it to do the opposite of what it predicts about itself. The program P did this to a predictor of halting, and the program D does it to a list of theorems; both are a program run on its own code. The move is older than computers: it is the Monster $\omega\,\omega$ of the first part of the course, the function that applies its argument to itself and then runs for ever. The History lines up four of its appearances in a table.

Remark 8.37 (the schoolroom). The schoolroom of the introduction asked this section's question of a smaller structure, and of only part of its language.

Take the signature with a constant 1 and two-place function symbols $+$, $\cdot$ and $\uparrow$, where $x \uparrow y$ is x^y; let $\mathbb{P}$ be the positive integers with their usual operations; and let Θ_{HS} be the equality axioms together with the eleven identities on the board, each with $\forall$ in front of its variables. Then Θ_{HS} is a specification, and $\mathbb{P} \models \Theta_{\mathrm{HS}}$, so it proves only truths. It does not prove every truth: Wilkie's identity, with $\forall x\, \forall y$ in front, is true in $\mathbb{P}$ and not provable from Θ_{HS}. The certificate is a structure of twelve elements, with tables for 1, $+$, $\cdot$ and $\uparrow$, in which all eleven identities hold and Wilkie's identity fails at one pair of elements.[11]

A specification that misses a truth does not show that no specification does better. The identities true of $\mathbb{P}$ can be decided by a program,[12] so the set of all of them is a specification that proves exactly the true identities. It captures the *equational theory* of $\mathbb{P}$, its true identities, which no finite list does. It does not capture $\mathbb{P}$, since capturing (Definition 8.13) asks about every sentence, with its connectives and quantifiers.

Nothing captures $\mathbb{P}$ in that sense. Let each positive integer n stand for the counting number $n - 1$. Then 0, s, $+$ and $\times$ can be defined from 1, $+$ and $\cdot$; for instance, the counting numbers for which $x + y = z$ are those for which $x + y = z + 1$ holds in $\mathbb{P}$. So a program that decided the truths of $\mathbb{P}$ would decide those of $\mathfrak{N}$, and none does (Theorem 8.30). By Theorem 8.16, no specification captures $\mathbb{P}$. At school, the fragment can be captured and the whole structure cannot.

8.4. **The Second Incompleteness Theorem.** We know now that neither computation nor the counting numbers can be captured. For the counting numbers the route was roundabout: it went through another structure, computation, which they happen to contain, and through truth in $\mathfrak{N}$, which no specification reaches. Is there a more direct limit, one that mentions no structure at all?

There is, and it is the form in which Hilbert's programme put the question. Take a specification strong enough to do a little arithmetic, such as Peano arithmetic below. Because it is a specification, a program lists its theorems; because it can do a little arithmetic, it can talk about programs, and so about the program that lists its own theorems. In particular it can state that this program never prints $\bot$, which is to say that the specification is consistent. The second incompleteness theorem says that, if the specification is consistent, it cannot prove that sentence.

The theorem matters for two reasons. The first is what its sentence says: consistency is what Hilbert's programme, the plan to secure mathematics by formal axioms, most wanted the axioms to establish about themselves. The second, and more to the point, is that the theorem has nothing to do with any intended structure. Consistency is a property of derivations, not of truth. So the limit is a limit of being specifiable as such: whatever structure we had in mind, a specification that can reason about programs cannot

[11] The structure is one of the family described in Subercaseaux and Przybocki (2026, §7), with particular values given to the cells that the template leaves as parameters; the script `inc_wilkie.py` in the course's support folder checks the eleven identities in it and finds the one pair at which Wilkie's identity fails.

[12] Macintyre (1981); see Burris and Yeats (2004) for the history.

certify its own consistency, and, by the completeness of NK, its consistency does not even follow from it.

The route has two steps. The first theorem assumed that Θ is sound, a strong assumption: to know that a theory proves nothing false about the counting numbers, one has to know which sentences about them are true, and that is what no specification can tell us. So we first replace soundness by consistency, at the cost of one fact about what PA proves: every true sentence about halting. Then we carry the proof of the first theorem out inside PA. That step rests on three facts about what PA proves of the list of theorems of Θ. We state them exactly and sketch why each holds; the full proofs are long, and we take them on loan.

Peano arithmetic. Both steps need a specification that can reason about programs, and the natural one is the specification every course on arithmetic starts from.

Definition 8.38 (Peano arithmetic)**.** *Peano arithmetic*, PA, is the set of sentences of Σ_{A} consisting of the equality axioms and the following sentences, each with $\forall$ in front of its free variables:

(1) $s(x) \neq 0$ and $s(x) = s(y) \to x = y$;
(2) $x + 0 = x$ and $x + s(y) = s(x + y)$;
(3) $x \times 0 = 0$ and $x \times s(y) = x \times y + x$;
(4) $x < y \leftrightarrow \exists z\,(s(z) + x = y)$;
(5) for every formula φ with a free variable x, the *induction axiom* $(\varphi[0/x] \wedge \forall x\,(\varphi \to \varphi[s(x)/x])) \to \forall x\,\varphi$.

The first group describes the successor, the second and third say how $+$ and $\times$ compute, and the fourth defines the order. The fifth is proof by induction, with one axiom for each property it might apply to. A program recognises an induction axiom by its shape, so PA is a specification. Since $\mathfrak{N} \models \mathrm{PA}$, it is sound, and so is every specification obtained from it by adding true sentences. It proves a great deal: very nearly every theorem of elementary number theory that has ever been proved can be derived in it. By the first incompleteness theorem it misses some truth all the same.

For arithmetic, PA plays the part that Π played for computation: a short theory, true of the structure we meant, that proves every true fact about halting (Fact 8.39 below) and not every truth.

Consistency, which is to replace soundness, is the property used in the proof of completeness: a set of sentences Θ is consistent if $\Theta \nvdash_{\mathrm{NK}} \bot$. In words: the theory never leads to a contradiction. Equivalently, there is no sentence φ with both $\Theta \vdash_{\mathrm{NK}} \varphi$ and $\Theta \vdash_{\mathrm{NK}} \neg\varphi$, since ($\to$E) turns such a pair into $\bot$ and ($\bot$E) turns $\bot$ into anything. The definition mentions derivations alone, and no structure.

A sound specification is consistent, since $\mathfrak{N} \not\models \bot$. The converse fails (Remark 8.49).

To trade soundness for consistency, we need one fact about what PA can prove: that it can check a finished calculation.

Fact 8.39 (true halting is provable). For every combinator M: if M halts, then $\mathrm{PA} \vdash_{\mathrm{NK}} [\mathsf{Halts}(M)]$.

The reason is that a halting run is a finite calculation. If M halts, there are particular numbers N, a and b that pack its run, and $[\mathsf{Halts}(M)]$ follows by ($\exists$I) from the claim that these numbers do so. That claim comes down to checking each of finitely many rows. Each check is finite because every "for all" in $\mathsf{Halts}_{\mathrm{A}}$ can be written to range only over the numbers below a given one, and so it comes down to facts about particular numbers, such as $\overline{80} = \overline{26} \times \overline{3} + \overline{2}$. The specification PA proves each such fact by calculating with its equations, as a pupil checks a long division. We take the details on loan. The fact is the arithmetical counterpart of part (iii) of the lemma that a program is a theory (Lemma 8.6): PA, like Π, proves every true fact about halting. What neither proves is every true fact about *not* halting.[13]

So every specification that contains PA proves $[\mathsf{Halts}(M)]$ for every combinator M that halts. The step from soundness to consistency uses nothing else about Θ, and the theorem asks for no more.

Theorem 8.40 (the first incompleteness theorem, from consistency)**.** *Let Θ be a consistent specification in Σ_{A} that proves $[\mathsf{Halts}(M)]$ for every combinator M that halts. Then*

$$\mathfrak{N} \models \gamma_\Theta \qquad \textit{and} \qquad \Theta \nvdash_{\mathrm{NK}} \gamma_\Theta.$$

Proof. Write D for the diagonal program of Θ and d for its code. As in the proof of the first incompleteness theorem (Theorem 8.34),

$$\Theta \vdash_{\mathrm{NK}} \gamma \qquad \text{iff} \qquad D\,\underline{d} \text{ halts.}$$

We show first that Θ does not prove the sentence, and then that it is true.

(1) Θ *does not prove* γ. Suppose it did. Then $D\,\underline{d}$ halts, so Θ proves $[\mathsf{Halts}(D\,\underline{d})]$, by the hypothesis. The sentence γ is the negation of that one, so Θ proves a sentence and its negation, and therefore $\bot$, which contradicts its consistency.

(2) *The sentence is true.* By the first step and the display, $D\,\underline{d}$ does not halt, so $\mathfrak{T} \models \neg\mathsf{Halts}(D\,\underline{d})$, and $\mathfrak{N} \models \gamma$ by the lemma that arithmetic contains computation (Lemma 8.29). □

In words: for a specification that proves every true sentence about halting, and so for every specification that contains PA, consistency is enough in place of soundness. The proof shows the price of the weaker assumption. A consistent specification that is not sound might prove the false sentence $\neg\gamma$, and nothing in the argument forbids it. So the theorem cannot say that Θ leaves the question open either way.

Remark 8.41 (a question left open either way). A cleverer program does give a sentence that a consistent Θ containing PA neither proves nor refutes. It searches the listing for a proof of a sentence and for a proof of its negation

[13] The fact is called the Σ_1-completeness of arithmetic. It applies to $\mathsf{Halts}_{\mathrm{A}}$ written with every universal quantifier bounded, as it can be: the optional Section 8.5 shows how, and the main line uses only that it can be done. The fact then holds even for the part of PA whose only induction axiom is the one for the formula $x = 0 \vee \exists y\,(x = s(y))$, which is needed to pass from $\varphi(\overline{0}), \ldots, \varphi(\overline{n})$ to $\forall i \leq \overline{n}\,\varphi(i)$. With that sentence as an axiom in place of induction, the part is Robinson's arithmetic, for which the fact is proved in Open Logic Project (2026, §35.11).

at once, and acts on whichever comes first. The argument is a little longer, and the main line does without it.[14]

Consistency as a sentence. Whether Θ is consistent is a question about a program: the one that waits for $\bot$.

Definition 8.42 (the consistency sentence)**.** The consistency sentence of a specification says that the program that waits for a contradiction never stops.

Formally, let Θ be a specification in Σ_{A}, let C_Θ be the program

```
procedure C(e)                          -- the input is ignored
    for each theorem phi of Theta, as LIST prints them:
        if phi is bottom:  halt
```

and let C be the combinator that runs it on 0. The *consistency sentence* of Θ is

$$\mathrm{Con}_\Theta \coloneqq [\neg \mathsf{Halts}(C)].$$

The program C_Θ ignores its input, so

Θ is consistent	iff	C does not halt	Lemma 8.15
	iff	$\mathfrak{N} \models \mathrm{Con}_\Theta$	Lemma 8.29.

The sentence Con_Θ thus says "Θ is consistent" in the language of arithmetic. Like $\neg\mathsf{Halts}(\Omega)$ for Π and γ for Θ, it is the translation of a fact about a program that does not halt. Hilbert's programme hoped for a derivation of such a sentence from axioms everyone accepts.

Remark 8.43 (the program, not only what it does). The definition speaks of C_Θ itself, and not of any combinator that halts on the same inputs. When Θ is consistent, C_Θ halts on no input, like every combinator that never halts. For some of those, such as the translation of "loop for ever", PA proves that they never halt, and with such a combinator in place of C the sentence would be provable. The sentence must speak of the program that actually searches the theorems of Θ.

Provability as a sentence. The second step of the route carries the proof of the first theorem from consistency out inside PA. That proof is an argument about numbers and programs: if Θ is consistent, then the diagonal program of Θ does not halt on its own code. To run it inside PA, arithmetic has to say not only that Θ proves $\bot$ but that it proves any given sentence.

Definition 8.44 (the provability sentence)**.** The provability sentence of φ says that the program that waits for φ stops.

Formally, let Θ be a specification in Σ_{A}, let φ be a sentence of Σ_{A}, and let W_φ be the program

```
procedure W(e)                          -- the input is ignored
    for each theorem psi of Theta, as LIST prints them:
        if psi is phi:  halt
```

[14] The trick is due to Rosser (1936); Gödel's original theorem assumed a condition between consistency and soundness, called ω-consistency.

The *provability sentence* of φ is

$$\mathrm{Prov}_\Theta(\varphi) \coloneqq [\mathsf{Halts}(W_\varphi\,\underline{0})].$$

In words, $\mathrm{Prov}_\Theta(\varphi)$ says "Θ proves φ": by Lemmas 8.15 and 8.29, $\mathfrak{N} \models \mathrm{Prov}_\Theta(\varphi)$ exactly when $\Theta \vdash_{\mathrm{NK}} \varphi$. The program C_Θ of the consistency sentence is $W_\bot$, and the translation of a negation is the negation of the translation, so Con_Θ is $\neg\mathrm{Prov}_\Theta(\bot)$.

The sentence $[\mathsf{Halts}(D\,\underline{d})]$ also says "Θ proves γ", through the diagonal program. Write h for it, so that γ is $\neg h$. Three facts about these sentences carry the proof of the first theorem into PA. Each is a true sentence, and each says something that proof used; what is needed is that PA proves it.

Fact 8.45 (what arithmetic proves about provability). Let Θ be a specification in Σ_A with $\mathrm{PA} \subseteq \Theta$. Then PA proves each of the following.

(1) *The diagonal program does what it says:* $h \to \mathrm{Prov}_\Theta(\gamma)$.
(2) *The theorems are closed under* ($\to$E)*:* $\mathrm{Prov}_\Theta(\varphi) \wedge \mathrm{Prov}_\Theta(\neg\varphi) \to \mathrm{Prov}_\Theta(\bot)$, for every sentence φ.
(3) *True halting is provably provable:* $[\mathsf{Halts}(M)] \to \mathrm{Prov}_\Theta([\mathsf{Halts}(M)])$, for every combinator M.

Each part is true for a reason that PA can follow, and the work is in checking that it can. In each the reasoning is an induction inside PA, on a number about which PA knows nothing in advance: a row of a run, a string in the list, a run of any length. This is where the induction axioms of PA do work that the fact that true halting is provable (Fact 8.39) does not need.[15]

- *(i).* The formula $\mathsf{Halts}_\mathrm{A}$ follows the leftmost run, and PA proves that this run is unique: two tables with the same first row agree row by row, by induction on the row number. So h speaks of one definite run of D on its own code, and $\mathrm{Prov}_\Theta(\gamma)$ of one definite run of W_γ on 0. Each run carries out the search of `LIST`, one string at a time, and stops at the first theorem that equals γ, which for D on its own code is the sentence it waits for. Following each run by induction on the row number, with the invariant that every row is a stage of the search, PA proves that each halts exactly when some string is a derivation of γ from Θ. So the two halt together.
- *(ii).* If `LIST` prints φ and $\neg\varphi$, it has met derivations of both. The tree that joins them by ($\to$E) is a derivation of $\bot$ with its open hypotheses in Θ, and `LIST` meets it later, since it tries every string. The joined string is computed from the two by arithmetic on their codes, and checking it comes down to checking its parts. That the search reaches it is again an induction along the run of $W_\bot$.
- *(iii).* The reason for the fact that true halting is provable (Fact 8.39) is a recipe: from numbers that pack a run of M, it writes down a derivation of $[\mathsf{Halts}(M)]$ from PA, one calculation for each row. The recipe is a

[15] Parts (ii) and (iii) are this chapter's forms of what are called the *derivability conditions*, and (iii) is known as the formalised Σ_1-completeness of arithmetic. Open Logic Project (2026, §37.6) states them for a proof predicate written directly in arithmetic, and asks the reader to take them on faith; Hájek and Pudlák (1993, Chapter I) prove them in full.

program, and part (iii) says that PA proves that the recipe works for every run, including runs whose numbers it knows nothing about except that they pack a run. That takes induction, on the length of the run and on the size of the numbers checked. It is also where PA $\subseteq \Theta$ is used: the derivations that the recipe writes have their open hypotheses in PA, so they are derivations from Θ.

With the three facts, the proof of the first theorem from consistency can be carried out inside PA.

Lemma 8.46 (arithmetic knows the first theorem)**.** *If Θ is a specification in Σ_{A} with* PA $\subseteq \Theta$*, then*

$$\mathrm{PA} \vdash_{\mathrm{NK}} \mathrm{Con}_\Theta \to \gamma_\Theta.$$

Proof. From PA and the hypothesis h, the parts of Fact 8.45 give, in turn,

$\mathrm{Prov}_\Theta(\gamma)$	part (i)
$\mathrm{Prov}_\Theta(h)$	part (iii), with $M = D\,\underline{d}$
$\mathrm{Prov}_\Theta(\bot)$	part (ii), with $\varphi = h$, since γ is $\neg h$.

These are the three moves of the first step of the proof of the theorem from consistency, said in arithmetic: $D\,\underline{d}$ halts, so Θ proves γ; it then also proves that $D\,\underline{d}$ halts; and a sentence with its negation gives $\bot$. The sentence Con_Θ is $\neg\mathrm{Prov}_\Theta(\bot)$, so ($\to$E) derives $\bot$ from PA, h and Con_Θ. Discharging h by ($\to$I) gives $\neg h$, which is γ, and discharging Con_Θ gives $\mathrm{Con}_\Theta \to \gamma$. □

Theorem 8.47 (the second incompleteness theorem)**.** *If Θ is a consistent specification in Σ_{A} with* PA $\subseteq \Theta$*, then $\Theta \nvdash_{\mathrm{NK}} \mathrm{Con}_\Theta$.*

Proof. Suppose $\Theta \vdash_{\mathrm{NK}} \mathrm{Con}_\Theta$. Since PA $\subseteq \Theta$, the lemma that arithmetic knows the first theorem (Lemma 8.46) gives $\Theta \vdash_{\mathrm{NK}} \mathrm{Con}_\Theta \to \gamma_\Theta$, and ($\to$E) gives $\Theta \vdash_{\mathrm{NK}} \gamma_\Theta$. Since Θ contains PA, it proves $[\mathsf{Halts}(M)]$ for every combinator M that halts (Fact 8.39), so this contradicts the first incompleteness theorem from consistency (Theorem 8.40). So Θ does not prove Con_Θ. □

In words: a consistent specification of arithmetic cannot prove its own consistency. By the completeness of NK, it follows that $\Theta \not\models \mathrm{Con}_\Theta$: some model of Θ makes Con_Θ false. So this is a limit on what a specification entails, and not only on how well it fits $\mathfrak{N}$. Unlike the theorem from consistency, it needs PA $\subseteq \Theta$ in earnest, and not only the true sentences about halting: the argument runs inside PA, with its induction.

Remark 8.48 (what this did to Hilbert's programme). The programme asked for two things about arithmetic: axioms that prove every truth, and a proof, by means everyone accepts, that the axioms never lead to a contradiction. The first incompleteness theorem rules out the first of these. The second incompleteness theorem rules out the second, in the form intended. The means everyone accepts are, at the very most, those of PA, and the finitary reasoning the programme had in mind is usually taken to be far weaker. Even PA cannot prove its own consistency, let alone that of anything stronger.

How completely this ends the programme depends on what "finitary" means, and that is still argued over. The consistency of PA can be proved by an induction that PA cannot carry out (the History tells the story), and

whether that induction is finitary is a matter of judgement, not of proof. Nor does $\mathfrak{N}$ settle the question. It satisfies PA, so PA is consistent; but that certificate is an infinite structure, which is what the programme set out to do without, and the axioms cannot issue it for themselves.

Remark 8.49 (consistent but not sound). The specification PA is consistent, since it is sound. So, by the theorem, PA does not prove $\mathrm{Con}_{\mathrm{PA}}$. It follows that PA together with $\neg\mathrm{Con}_{\mathrm{PA}}$ is consistent. A derivation of $\bot$ from it would give, by $(\to\mathrm{I})$, a derivation of $\neg\neg\mathrm{Con}_{\mathrm{PA}}$ from PA, and then, since NK is classical, one of $\mathrm{Con}_{\mathrm{PA}}$, which the theorem rules out.

But PA with $\neg\mathrm{Con}_{\mathrm{PA}}$ added is not sound, since $\neg\mathrm{Con}_{\mathrm{PA}}$ says that PA is inconsistent, which is false. A consistent theory can prove something false about the numbers, and even something false about itself.

8.5. **Defining Halting** *(optional).* This section proves part (v) of the lemma that computation is definable (Lemma 8.27), which the main line used with only a sketch of its proof, and shows that the formula can be written with every universal quantifier bounded, the form that Fact 8.39 (true halting is provable) needs. Parts (i)–(iv) come from the same pieces. The main line uses these results, not their proofs.

Rather than present the formula and then check it, we work it out by unfolding what it has to say. We want a formula $\mathsf{Halts}_{\mathrm{A}}(x)$ that is true of the code of a combinator M exactly when M halts. By definition, M halts when it runs to a normal form, and M runs to a combinator when a finite chain of steps leads from one to the other. So $\mathsf{Halts}_{\mathrm{A}}$ has to say: *there are k and combinators $M_0, \ldots, M_k$ with $M_0 = M$, with $M_i \rightsquigarrow M_{i+1}$ for each $i < k$, and with M_k in normal form.* In symbols, pretending for a moment that we may quantify over combinators, and over lists of them of any length,

$$\exists k\, \exists M_0 \cdots \exists M_k\, \big(M_0 = M \wedge \forall i < k\, (M_i \rightsquigarrow M_{i+1}) \wedge \mathsf{Nf}(M_k)\big).$$

Four things in this line are not yet arithmetic, and the section deals with them in turn.

(a) The list $M_0, \ldots, M_k$ has a length that is not fixed in advance, and a formula has only a fixed number of quantifiers.

(b) Each M_i is a combinator, not a number.

(c) $M_i \rightsquigarrow M_{i+1}$ is a step, a relation between combinators.

(d) $M_0 = M$ and "M_k is in normal form" say how the run starts and how it ends.

Each becomes a statement about digits at positions, and so, in the end, about remainders, which $+$ and $\times$ express.

(a) A list as two numbers. The list has to be packed into a fixed number of numbers. Write $\mathrm{rem}(a, m)$ for the remainder of a on division by m, for $m \geq 1$. The packing uses the function

$$\beta(a, b, i) \coloneqq \mathrm{rem}\big(a,\ 1 + (i+1)\, b\big),$$

called Gödel's *β-function.* One fact from elementary number theory is needed, and we prove it.

Lemma 8.50 (the Chinese remainder theorem)**.** *Suppose that*

- *$m_0, \ldots, m_n$ are positive integers, no two of which have a common factor greater than* 1*, and*
- *$k_0, \ldots, k_n$ are numbers with $k_i < m_i$ for each i.*

Then there is a number a with $\mathrm{rem}(a, m_i) = k_i$ *for every $i \leq n$.*

In words: when the divisors share no factor, the remainders can be chosen independently of one another, and some number has all of them. (With the divisors 2 and 4 they cannot: no number leaves remainder 0 on division by 2 and 1 on division by 4.)

Proof. The idea is to count: there are exactly as many numbers below the product of the m_i as there are lists of remainders. Let $P = m_0 m_1 \cdots m_n$, and send each number $a < P$ to its list of remainders $(\mathrm{rem}(a, m_0), \ldots, \mathrm{rem}(a, m_n))$. Two different numbers $a < a' < P$ have different lists. For if the lists were equal, every m_i would divide $a' - a$. Then their product P would divide $a' - a$ too. To see this, write P as a product of powers of different primes. Take one of them, p^k. The prime p divides some m_i, since a prime that divides a product divides one of its factors. It divides no other m_l, so all k factors p of P come from that one m_i. So p^k divides m_i, and therefore divides $a' - a$. A number divisible by the powers of different primes is divisible by their product, by the uniqueness of factorisation into primes, so P divides $a' - a$. But $0 < a' - a < P$, so P cannot divide $a' - a$. So the map is one-to-one. There are P numbers below P, and P possible lists of remainders, so every list is the list of some a, and in particular the list $(k_0, \ldots, k_n)$. □

Lemma 8.51 (packing a sequence)**.** *For every finite sequence of numbers $k_0, \ldots, k_n$ there are numbers a and b such that $\beta(a, b, i) = k_i$ for every $i \leq n$.*

In words: every finite list of numbers, however long, can be stored in two numbers, and read back one entry at a time by taking remainders.

Proof. Let j be the largest of $n, k_0, \ldots, k_n$, let $b = j!$, and let $m_i = 1 + (i+1)\, b$ for $i \leq n$. We check the two hypotheses of the Chinese remainder theorem.

- $k_i < m_i$. We have $k_i \leq j \leq j! = b < m_i$, since $j \leq j!$ for every number j (for $j = 0$ because $0! = 1$).
- *No two of the m_i have a common factor.* Suppose a prime p divides m_i and m_l with $i < l$. Then p divides their difference $(l - i)\, b$. It does not divide b: if it did, it would divide $m_i - 1$, which is a multiple of b. It would also divide m_i, and so their difference 1. So p divides $l - i$. But $1 \leq l - i \leq n \leq j$, so $l - i$ divides $j! = b$, and then p divides b after all, which is a contradiction.

The Chinese remainder theorem now gives a with $\mathrm{rem}(a, m_i) = k_i$, which says $\beta(a, b, i) = k_i$. □

Example 8.52 (packing $2, 0, 3$)**.** For the sequence $2, 0, 3$ the proof takes $j = 3$, $b = 3! = 6$ and the divisors 7, 13, 19; the Chinese remainder theorem gives $a = 611$, and indeed $611 = 87 \times 7 + 2 = 47 \times 13 + 0 = 32 \times 19 + 3$. The proof's b is generous: the smallest pair that packs the sequence is $a = 80$, $b = 2$.

The function β is definable: $\beta(a,b,i) = v$ exactly when $\mathfrak{N}$ makes true

$$B(a,b,i,v) \coloneqq \exists q \, \big(a = q \times s(s(i) \times b) + v \wedge v < s(s(i) \times b)\big),$$

since $s(s(i) \times b)$ is $1 + (i+1)\, b$.

When $\beta(a,b,i)$ appears inside a formula as if it were a term, as in $\beta(a,b,i) = \overline{1}$, the formula abbreviates one that says "there is a v with $B(a,b,i,v)$ and $v = \overline{1}$". Since β is a function, nothing is lost.

We write $\forall i < N \, \varphi$ for $\forall i \, (i < N \to \varphi)$, $\exists i < N \, \varphi$ for $\exists i \, (i < N \wedge \varphi)$, and $\forall i \leq N$ and $\exists i \leq N$ in the same way; and $\exists p, q \leq N \, \varphi$ for $\exists p \leq N \, \exists q \leq N \, \varphi$. In a sum such as $p + 3$, the number 3 stands for its numeral $\overline{3}$.

The abbreviations for β bring in quantifiers. Every one of them, wherever it stands, can be bounded by a, since a quotient or a remainder of a is at most a. Every other quantifier below ranges over the rows of a table or the positions in a row, and is bounded by the number of rows or the length of a row. Written so, $\mathsf{Halts}_{\mathrm{A}}$ has unbounded quantifiers only in its leading block of $\exists$. That is the form needed by Fact 8.39.

(b) Combinators as rows. Each combinator of the run is read as its string, a row of digits.

Definition 8.53 (the string of a combinator)**.** The string of a combinator is the combinator written out with all its brackets, the string that its code spells.

Formally, $\mathrm{str}(M)$ is defined by recursion on the shape of M:

$$\begin{aligned} \mathrm{str}(\mathsf{K}) &\coloneqq \mathtt{K}, \\ \mathrm{str}(\mathsf{S}) &\coloneqq \mathtt{S}, \\ \mathrm{str}(M_1\, M_2) &\coloneqq (\,\mathrm{str}(M_1)\,\mathrm{str}(M_2)\,). \end{aligned}$$

Let $M_0 \rightsquigarrow M_1 \rightsquigarrow \cdots \rightsquigarrow M_N$ be a run. Row i of its table is $\mathrm{str}(M_i)$, written in the digits used for codes: 1 for `K`, 2 for `S`, 3 for (and 4 for). The rows have different lengths, so the table is described by packed sequences of three kinds:

$$\begin{aligned} \mathit{Len}(i) &\coloneqq \beta(a_1, b_1, i), \\ \mathit{Dg}(i,j) &\coloneqq \beta(a_2, b_2, i \times W + j), \\ \mathit{Ct}(i,j) &\coloneqq \beta(a_3, b_3, i \times s(W) + j), \end{aligned}$$

for the length of row i, the digit at position j of row i, and the bracket count of row i before position j, where W is a number at least as large as every length. The bracket count is the number of opening brackets among the first j digits, less the number of closing ones.

The clause *Rows* says that the numbers mean what was just said. It is the conjunction, for all $i \leq N$, of:

- $\mathit{Len}(i) \leq W$;
- for every $j < \mathit{Len}(i)$, the digit $\mathit{Dg}(i,j)$ is one of $\overline{1}$, $\overline{2}$, $\overline{3}$, $\overline{4}$;
- $\mathit{Ct}(i,0) = 0$;
- for every $j < \mathit{Len}(i)$, the count goes up by one at a digit $\overline{3}$, down by one at a digit $\overline{4}$ (written $\mathit{Ct}(i,j) = s(\mathit{Ct}(i, s(j)))$), and stays the same at the others.

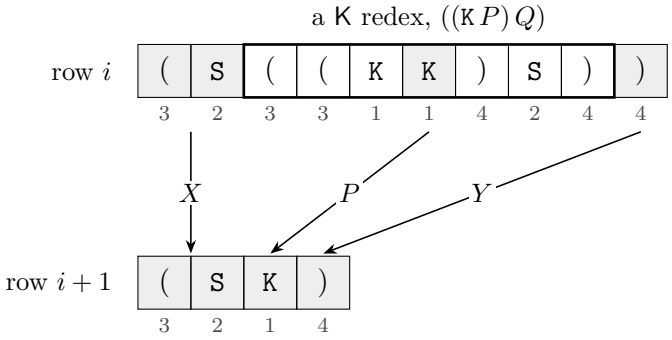

FIGURE 8.3. A step is the replacement of a stretch.

(c) One step. A step copies most of one row into the next and changes one stretch of it (Figure 8.3). A *stretch* of row i is the part from a position p up to, but not including, a position q. In the string of a combinator, the stretches that are themselves strings of combinators are the strings of its parts. Call them the *term stretches*: a single `K` or `S`, or an opening bracket together with everything up to the closing bracket that matches it. The matching bracket is where the count first comes back to its value at p:

$$\begin{aligned} T(i,p,q) \coloneqq\ & \big(q = s(p) \wedge (Dg(i,p) = \overline{1} \vee Dg(i,p) = \overline{2})\big) \\ & \vee \big(Dg(i,p) = \overline{3} \wedge p < q \wedge Ct(i,q) = Ct(i,p) \\ & \qquad \wedge \forall r < q\,(p < r \to Ct(i,p) < Ct(i,r))\big). \end{aligned}$$

A row is the string of a combinator exactly when it is one term stretch and every opening bracket in it opens a stretch made of two term stretches and a closing bracket:

$$\begin{aligned} Wf(i) \coloneqq\ & T(i,0,Len(i)) \\ & \wedge \forall p < Len(i)\,\big(Dg(i,p) = \overline{3} \to \exists u,v \le Len(i)\, Pair(i,p,u,v)\big), \end{aligned}$$

where $Pair(i,p,u,v)$ is $T(i,s(p),u) \wedge T(i,u,v) \wedge Dg(i,v) = \overline{4}$: after the bracket at p come two term stretches, from $p+1$ to u and from u to v, and then a closing bracket.

With term stretches in hand, the step itself can be described. A step rewrites one part of a combinator that is a redex, and leaves the rest alone. In the string, it replaces one stretch:

Redex	**Row** i	**Row** $i+1$
K	$X\ ((\mathtt{K}\,P)\,Q)\ Y$	$X\ P\ Y$
S	$X\ (((\mathtt{S}\,P)\,Q)\,R)\ Y$	$X\ ((P\,R)(Q\,R))\ Y$

Here P, Q and R are term stretches, and X and Y are the rest of the row, unchanged. Figure 8.3 shows one K step written this way, from (S((KK)S)) to (SK), with $X =$ (S, $P =$ K, $Q =$ S and $Y =$). The new row is made of three copies from the old one, and nothing else.

Write ℓ_X, ℓ_P, ℓ_Q and so on for the lengths of these pieces, and p for the position where the redex starts, which is ℓ_X. A K redex at p is then

described by

$$\begin{aligned} \mathit{Red}_{\mathsf{K}}(i,p,\ell_P,\ell_Q) \coloneqq\ & \mathit{Dg}(i,p)=\overline{3} \wedge \mathit{Dg}(i,p+1)=\overline{3} \wedge \mathit{Dg}(i,p+2)=\overline{1} \\ & \wedge T(i,p+3,p+3+\ell_P) \wedge \mathit{Dg}(i,p+3+\ell_P)=\overline{4} \\ & \wedge T(i,p+4+\ell_P,p+4+\ell_P+\ell_Q) \\ & \wedge \mathit{Dg}(i,p+4+\ell_P+\ell_Q)=\overline{4} \\ & \wedge p+4+\ell_P+\ell_Q < \mathit{Len}(i). \end{aligned}$$

A copy of a stretch is described by

$$\mathit{Cp}(i,p,i',p',\ell) \coloneqq \forall j < \ell\ (\mathit{Dg}(i',p'+j)=\mathit{Dg}(i,p+j)),$$

which says that the ℓ digits of row i' from p' are those of row i from p. A K step from row i to row $i+1$ is

$$\begin{aligned} \mathit{Next}_{\mathsf{K}}(i) \coloneqq\ & \exists p,\ell_P,\ell_Q,\ell_Y \le \mathit{Len}(i)\ (\mathit{Red}_{\mathsf{K}}(i,p,\ell_P,\ell_Q) \\ & \wedge \mathit{Len}(i)=p+\ell_P+\ell_Q+\ell_Y+5 \wedge \mathit{Len}(s(i))=p+\ell_P+\ell_Y \\ & \wedge \mathit{Cp}(i,0,s(i),0,p) \wedge \mathit{Cp}(i,p+3,s(i),p,\ell_P) \\ & \wedge \mathit{Cp}(i,p+\ell_P+\ell_Q+5,s(i),p+\ell_P,\ell_Y)). \end{aligned}$$

The three copies are X, P and Y, each put in its place in the new row. The formulas $\mathit{Red}_{\mathsf{S}}(i,p,\ell_P,\ell_Q,\ell_R)$ and $\mathit{Next}_{\mathsf{S}}(i)$ are written in the same way from the S line of the table above, with seven fixed digits in row i, six in row $i+1$, six copies (X, P, Q, R twice and Y), and the bound $p+6+\ell_P+\ell_Q+\ell_R < \mathit{Len}(i)$. This bound, like the last line of the K formula, keeps every position inside the row, where the digits are fixed by *Rows*. We do not write the S formulas out here; Exercise 8.15 writes the first and lays out the second.

Let $\mathit{Next}(i)$ be $\mathit{Next}_{\mathsf{K}}(i) \vee \mathit{Next}_{\mathsf{S}}(i)$. A redex starts at position r of row i when

$$\begin{aligned} \mathit{Redex}(i,r) \coloneqq\ & \exists \ell_P,\ell_Q \le \mathit{Len}(i)\ \mathit{Red}_{\mathsf{K}}(i,r,\ell_P,\ell_Q) \\ & \vee \exists \ell_P,\ell_Q,\ell_R \le \mathit{Len}(i)\ \mathit{Red}_{\mathsf{S}}(i,r,\ell_P,\ell_Q,\ell_R), \end{aligned}$$

and row i has no redex when

$$\mathit{Normal}(i) \coloneqq \forall r \le \mathit{Len}(i)\ \neg \mathit{Redex}(i,r).$$

A combinator can have many runs, since a step may rewrite any of its redexes, and $\mathsf{Halts}_{\mathrm{A}}$ will follow only one. The *leftmost run* rewrites, at each step, the redex whose stretch starts first in the row. A term stretch is fixed by the position where it starts, so different redexes start at different positions; the leftmost run therefore takes one definite step at each stage, and is unique. A leftmost step from row i is a step whose redex starts at a position p before which no redex starts. Let $\mathit{Next}^{\mathrm{L}}_{\mathsf{K}}(i)$ be $\mathit{Next}_{\mathsf{K}}(i)$ with the clause

$$\forall r < p\ \neg \mathit{Redex}(i,r)$$

added next to $\mathit{Red}_{\mathsf{K}}(i,p,\ell_P,\ell_Q)$, let $\mathit{Next}^{\mathrm{L}}_{\mathsf{S}}(i)$ be obtained from $\mathit{Next}_{\mathsf{S}}(i)$ in the same way, and let $\mathit{Next}^{\mathrm{L}}(i)$ be their disjunction.

Nothing is lost by following the leftmost run. A redex that lies inside another starts after it, so the redex that starts first is the leftmost of the outermost redexes. By the normalisation theorem for combinators, stated in Chapter 1 and taken on loan there, the run that always rewrites that

redex reaches a normal form whenever the combinator has one. We use this theorem only as a fact about combinators, to see what the formula is true of; nothing is proved about it inside PA.

Lemma 8.54 (a step is the replacement of a stretch)**.** *For all combinators M and M':*

(i) $M \rightsquigarrow M'$ *exactly when* $\mathrm{str}(M')$ *is obtained from* $\mathrm{str}(M)$ *by one of the two replacements of the table, with P, Q and R term stretches;*

(ii) M *is in normal form exactly when* $\mathrm{str}(M)$ *has no stretch of either form in the column "Row i" of the table, with P, Q and R term stretches.*

In words: a step can be done on the string, by cut and paste, without ever reading the combinator as a tree.

Proof. We show first that the term stretches of $\mathrm{str}(M)$ are exactly the strings of the parts of M. Two facts about the bracket count are needed, both proved by induction on a combinator N.

- Along $\mathrm{str}(N)$ the count never falls below its value at the start of $\mathrm{str}(N)$.
- At the end of $\mathrm{str}(N)$ the count is back to that value, and, when N is an application, it is not back to it any earlier.

The claim is proved by induction on M. If M is K or S, its string has one symbol and M has one part. If M is $M_1\,M_2$, its parts are M itself and the parts of M_1 and of M_2, and $\mathrm{str}(M)$ is an opening bracket, $\mathrm{str}(M_1)$, $\mathrm{str}(M_2)$ and a closing bracket. The whole string is the string of M, and it is a term stretch by the second fact. By the induction hypothesis, the term stretches that lie inside $\mathrm{str}(M_1)$ or inside $\mathrm{str}(M_2)$ are the strings of the parts of M_1 and M_2. There are no other term stretches.

- *One that begins at the first bracket of* $\mathrm{str}(M)$ ends where the count first comes back to its value at the start, which by the second fact is the end of $\mathrm{str}(M)$. So it is the whole string.
- *One that begins with a bracket inside* $\mathrm{str}(M_1)$ ends inside $\mathrm{str}(M_1)$. Let c be the count just before that bracket. By the first fact, c is at least the count b at the start of $\mathrm{str}(M_1)$. After the bracket the count is $c+1$, and at the end of $\mathrm{str}(M_1)$ it is back to b. Since it moves by one at a time, it comes back to c by then.
- *One that begins with a bracket inside* $\mathrm{str}(M_2)$ ends inside $\mathrm{str}(M_2)$, in the same way.

A term stretch that is a single `K` or `S` lies inside $\mathrm{str}(M_1)$ or $\mathrm{str}(M_2)$, so the list is complete.

It follows that a stretch of either form in the column "Row i", with P, Q and R term stretches, is exactly the string of a part of M that is a redex. The two claims of the lemma follow.

- *(i).* A step rewrites one part of M that is a redex, by (K) or (S). The rules (app_1) and (app_2) carry the rewriting out to the whole term, each putting back one pair of brackets around it and leaving the rest of the string as it was. So the string of the result is the string of M with that stretch replaced, as in the table. Conversely, each replacement of the table rewrites the string of one redex part in this way, and so gives the

string of the result of a step. Since different combinators have different strings, M' is that result.

- *(ii).* A combinator is in normal form exactly when it has no part that is a redex. □

(d) The ends of the run. The last row must be in normal form, which is $\mathit{Normal}(N)$; the first must be the string of the combinator with code x. The digits of row 0 are read off by a fourth packed sequence: $\mathit{Val}(j) \coloneqq \beta(a_4, b_4, j)$ is the number that the first j digits of row 0 spell in base 5. Let $\mathit{Start}(x)$ be the conjunction of

- $\mathit{Wf}(0)$, which says that row 0 is the string of a combinator;
- $\mathit{Val}(0) = 0$, $\mathit{Val}(\mathit{Len}(0)) = x$, and $\mathit{Val}(s(j)) = \overline{5} \times \mathit{Val}(j) + \mathit{Dg}(0, j)$ for every $j < \mathit{Len}(0)$, which say that its digits spell x.

Putting it together. Let $\mathsf{Halts}_{\mathrm{A}}(x)$ be

$$\exists N\, \exists W\, \exists a_1\, \exists b_1 \cdots \exists a_4\, \exists b_4$$
$$\big(\mathit{Rows} \wedge \mathit{Start}(x) \wedge \forall i < N\ \mathit{Next}^{\mathrm{L}}(i) \wedge \mathit{Normal}(N)\big).$$

The formula $\mathsf{Halts}_{\mathrm{A}}(x)$ says that there is a table of $N + 1$ rows whose first row is the combinator with code x, whose later rows are each one leftmost step from the row before, and whose last row is in normal form. Each piece of the line we started from has its counterpart:

- *there are k and $M_0, \ldots, M_k$* became $\exists N\, \exists W\, \exists a_1\, \exists b_1 \cdots \exists a_4\, \exists b_4$: the length of the run and the numbers that pack its table;
- *each M_i is a combinator* became *Rows*, with *Wf*;
- $M_0 = M$ became $\mathit{Start}(x)$;
- $M_i \rightsquigarrow M_{i+1}$ *for each $i < k$* became $\forall i < N\ \mathit{Next}^{\mathrm{L}}(i)$, with the step chosen as the leftmost one;
- M_k *is in normal form* became $\mathit{Normal}(N)$.

The other formulas of the lemma come from the same pieces. The formula $\mathit{Comb}(x)$ says that there is a table of one row with *Rows* and $\mathit{Start}(x)$; the formula $\mathsf{Nf}_{\mathrm{A}}(x)$ adds $\mathit{Normal}(0)$; and $\mathsf{Step}_{\mathrm{A}}(x, y)$ asks for a table of two rows, the first spelling x, the second spelling y, with $\mathit{Next}(0)$. For $\mathit{App}(x, y, z)$, the string of $M\,N$ is an opening bracket, the string of M, the string of N and a closing bracket, so

$$\ulcorner M\,N \urcorner = 3 \cdot 5^{\ell+m+1} + \ulcorner M \urcorner \cdot 5^{m+1} + \ulcorner N \urcorner \cdot 5 + 4,$$

where ℓ and m are the numbers of digits of $\ulcorner M \urcorner$ and $\ulcorner N \urcorner$; the powers of 5 form a sequence $1, 5, 25, \ldots$, which is packed and unpacked like any other.

Proof of part (v) of the lemma that computation is definable. Let M be a combinator and e its code; parts (i)–(iv) are proved in the same way, from the same pieces.

- *If M halts, the formula is true of $\overline{e}$.* Take the leftmost run $M = M_0 \rightsquigarrow \cdots$ of M. Since M has a normal form, the normalisation theorem for combinators says that this run reaches one, say M_N. Let W be the largest of the lengths of the strings $\mathrm{str}(M_i)$, and pack the lengths, the

digits, the bracket counts and the values of the first digits of row 0 by the packing lemma (Lemma 8.51), in the orders stated. The resulting numbers make every clause true:

- *Rows*, by the choice of the sequences;
- *Start*, since row 0 is the string of M;
- each $\mathit{Next}^{\mathrm{L}}(i)$, and $\mathit{Normal}(N)$, by the lemma that a step is the replacement of a stretch (Lemma 8.54): the stretch replaced in row i is the string of the redex that the leftmost run rewrites, and no redex starts before it.

- *If the formula is true of $\overline{e}$, then M halts.* Take numbers that make the clauses true for $\overline{e}$. By *Start* and *Rows*, row 0 is the string of a combinator whose digits spell e, so it is $\mathrm{str}(M)$. By induction on $i \leq N$, row i is the string of a combinator M_i with $M \rightsquigarrow^* M_i$: for the step from i to $i+1$, the clause $\mathit{Next}^{\mathrm{L}}(i)$ says, among other things, that row $i+1$ is row i with a stretch replaced as in the table. The stretch replaced is the string of a part of M_i, and it is replaced by the string of a combinator, so row $i+1$ is the string of a combinator M_{i+1}. By the lemma that a step is the replacement of a stretch, $M_i \rightsquigarrow M_{i+1}$. By $\mathit{Normal}(N)$ and the same lemma, M_N is in normal form. So M halts, which is what was to be shown. □

The definitions of this section can be tested against the combinators themselves, and have been.[16]

8.6. **Decidable Theories** *(optional).* Nothing later depends on this section. It asks which structures, other than finite ones, can be captured, and proves one case in full. It ends with a remark on the structures that satisfy every truth of arithmetic and are not $\mathfrak{N}$.

By Proposition 8.18, a structure can be captured exactly when its truths can be decided, so the question becomes a search for decision procedures. The main tool in that search is quantifier elimination.

Definition 8.55 (quantifier elimination)**.** A set of sentences Θ *admits quantifier elimination* if for every formula φ there is a formula ψ such that

- ψ has no quantifiers,
- ψ has no free variables other than those of φ, and
- $\Theta \models \forall \bar{x}\,(\varphi \leftrightarrow \psi)$, where $\bar{x}$ lists the free variables of φ.

In words: every question that Θ can be asked, however many quantifiers it has, is equivalent under Θ to one without any. When φ is a sentence, ψ is a sentence without quantifiers, built from atomic sentences about the constants alone, and so it is often easy to settle.

The first example is the rationals with their order, an infinite structure captured by a short list of the properties of its order.

[16] The script `ks_strings.py` in the course's support folder checks that *Wf* holds of exactly the strings of combinators, among all 349,524 strings of at most nine symbols; that the term stretches are exactly the strings of the parts, for all 550 combinators with at most five leaves; that the replacements of the table are exactly the steps, and a string with no redex stretch exactly a normal form, for all 3238 combinators with at most six leaves.

Definition 8.56 (dense linear order)**.** The specification DLO (for *dense linear order*) consists of the equality axioms for the signature $\{<, =\}$ and the following sentences.

- *Irreflexive:* $\forall x\, \neg(x < x)$.
- *Transitive:* $\forall x\, \forall y\, \forall z\, ((x < y \wedge y < z) \to x < z)$.
- *Linear:* $\forall x\, \forall y\, (x < y \vee x = y \vee y < x)$.
- *Dense:* $\forall x\, \forall y\, (x < y \to \exists z\, (x < z \wedge z < y))$.
- *No end points:* $\forall x\, \exists y\, (y < x)$ and $\forall x\, \exists y\, (x < y)$.

The rationals $(\mathbb{Q}, <)$ make every one of these sentences true.

Theorem 8.57 (a dense order is captured)**.** *The specification* DLO *admits quantifier elimination, and it captures* $(\mathbb{Q}, <)$*.*

In words: six sentences and the equality axioms say everything true about the order of the rationals.

Proof. By the completeness of NK it is enough to work in an arbitrary structure that satisfies DLO, so the equivalences below are claims about every such structure. We eliminate quantifiers by induction on φ. The connectives are immediate, $\forall x$ is $\neg\exists x\, \neg$, and the only case with content is $\exists x\, \psi$ with ψ already free of quantifiers. The case is settled in four moves.

(1) Write ψ in disjunctive normal form and move $\exists x$ inside the disjunction, so that it remains to treat $\exists x\, (\lambda_1 \wedge \cdots \wedge \lambda_k)$ with literals λ_i.

(2) Remove negations: $\neg(u < v)$ is equivalent to $v < u \vee u = v$, and $u \neq v$ to $u < v \vee v < u$, by the order axioms; distribute again. Conjuncts that do not mention x move outside the quantifier.

(3) If $x < x$ is a conjunct, the conjunction is false. A conjunct $x = x$ can be dropped. If $x = y$ or $y = x$ is a conjunct for a variable y other than x, then $\exists x\, (x = y \wedge \chi)$ is equivalent to $\chi[y/x]$.

(4) What remains is $\exists x\, (y_1 < x \wedge \cdots \wedge y_l < x \wedge x < z_1 \wedge \cdots \wedge x < z_r)$, and it is equivalent to the conjunction of all $y_i < z_j$. In one direction this is transitivity. In the other, take the largest y_i and the smallest z_j, which exist since the order is linear. By hypothesis the first is below the second, and density gives an x between them. If there are no y_i, or no z_j, the absence of a least or greatest element gives x, and the conjunction is empty and true.

For a sentence, the quantifier-free equivalent has no variables and no constants, so it is built from $\bot$ by the connectives and is provably true or provably false. So, for every sentence φ, the specification DLO proves φ or proves $\neg\varphi$. As $(\mathbb{Q}, <)$ satisfies DLO, the specification proves only truths about $(\mathbb{Q}, <)$, by soundness. So a true sentence φ is proved, since otherwise $\neg\varphi$ would be proved and true. Hence DLO captures $(\mathbb{Q}, <)$. □

The proof is an algorithm: each move rewrites a formula, and the rewriting ends. So it decides the truths of $(\mathbb{Q}, <)$ directly, without the listing of theorems.

Quantifier elimination reaches further than orders.

Remark 8.58 (addition without multiplication). The counting numbers with 0, s, $+$ and $<$ but without $\times$ can also be captured, by the axioms of PA that do not mention $\times$, with induction for the formulas of the smaller language. The proof is again by quantifier elimination, after one addition: for each k and r, a symbol for "x leaves remainder r on division by k", a relation that can already be written with $+$ alone.[17] The definable sets of numbers are then easy to describe: each is, beyond some point, periodic, like the even numbers.

Remark 8.59 (the real numbers). The real numbers with 0, 1, $+$, $\times$ and $<$ can be captured too, by the axioms of an ordered field in which every positive number has a square root and every polynomial of odd degree has a root. The proof is once more by quantifier elimination, now of polynomial inequalities.[18] It follows that the sets of reals definable in one variable are finite unions of points and intervals. The counting numbers are not such a set, so they cannot be defined among the reals. The reals contain them, and yet are the tamer of the two.

Remark 8.60 (why multiplication is the dangerous addition). In none of the three infinite structures of this section can a finite sequence be packed into numbers and unpacked again by a formula. With $+$ and $\times$ together on the counting numbers, the β-function does exactly that, and with it every halting run is definable. The undecidability of arithmetic then follows from the lemma that arithmetic contains computation (Lemma 8.29) alone.

Even the simplest sentences of arithmetic are affected. No program decides whether a polynomial equation with integer coefficients has a solution in the counting numbers. That question is asked by a sentence whose only quantifiers are existential ones in front of an equation.

Remark 8.61 (non-standard models). The first incompleteness theorem (Theorem 8.34) can also be read through structures. The sentence γ does not follow from Θ, so by the completeness of NK some model $\mathfrak{M}$ of Θ makes it false. In $\mathfrak{N}$ it is true, so $\mathfrak{M}$ is not $\mathfrak{N}$, nor a copy of it: it satisfies every sentence of Θ and still disagrees with $\mathfrak{N}$ about whether D halts on its own code. Both are models of Θ, since Θ is sound, and a model of Θ that is not a copy of $\mathfrak{N}$ is called a *non-standard model of* Θ. This one is not a model of $\mathrm{Th}(\mathfrak{N})$, since it makes the true sentence γ false.

The disagreement has to live somewhere. In $\mathfrak{M}$ the sentence $\neg\gamma$ is true, and it claims that a halting run exists. When Θ contains PA one can show, though we do not here, that the numerals $\overline{0}, \overline{1}, \overline{2}, \ldots$ of $\mathfrak{M}$ behave exactly as the counting numbers do, so no numeral codes such a run. The run is coded by an element beyond every numeral, an element that no closed term names, and read in $\mathfrak{M}$ it is an infinitely long "run" that no machine performs.

Non-standard models are not only the price of incompleteness. Even $\mathrm{Th}(\mathfrak{N})$, which leaves no truth unproved, has them. Add a new constant c and, for every n, the sentence $\overline{n} < c$. Every finite part of this set has a

[17] Presburger (1930). The procedure is expensive: every decision procedure for this structure takes, on some sentences of length n, a number of steps that grows like $2^{2^{cn}}$ (Fischer and Rabin, 1974).

[18] Tarski (1951); the result was obtained by 1930.

Structure or question	**Captured?**	**Truths decidable?**
a finite structure	yes, by its table	yes, by evaluation
propositional validity	(no structure)	yes, by `DPLL`
$(\mathbb{Q}, <)$	yes, by DLO	yes, by quantifier elimination
$(\mathbb{N}, 0, s, +, <)$	yes	yes, at great cost
$(\mathbb{R}, 0, 1, +, \times, <)$	yes	yes, at great cost
$(\mathbb{N}, 0, s, +, \times, <)$	no	no
first-order validity	(no structure)	no; listable by `REFUTE`

FIGURE 8.4. What can be captured and decided.

model, namely $\mathfrak{N}$ with c read as a large enough number, so by compactness the whole set has one, and in it c names an element beyond every numeral. Forget c, and the result satisfies every sentence true in $\mathfrak{N}$ without being a copy of $\mathfrak{N}$: a non-standard model of $\mathrm{Th}(\mathfrak{N})$ itself. No set of first-order sentences pins the counting numbers down.[19]

Figure 8.4 summarises the chapter's answers.

History. The question of this chapter was asked as a programme. Hilbert's *Foundations of Geometry* of 1899 set out axioms for Euclid's geometry from which every theorem followed by logic alone, with nothing taken from the diagrams. In his address to the International Congress of Mathematicians in Paris in 1900, the second of his problems asked for a proof that the axioms of arithmetic (for Hilbert, those of the real numbers) are consistent.[20] In the 1920s the request grew into *Hilbert's programme*: to write the whole of mathematics in formal systems, and to prove, by elementary reasoning about finite strings of symbols, that those systems are consistent and that they settle every question.

The programme was answered in Königsberg in September 1930, at two meetings a day apart. On 7 September a conference on the epistemology of the exact sciences held a round-table discussion on the foundations of mathematics. There Kurt Gödel, aged twenty-four, remarked that there are true sentences of arithmetic that a consistent formal system cannot prove. Almost nobody took it in. John von Neumann did, and spoke with him afterwards. On 8 September Hilbert addressed the Society of German Scientists and Physicians in the same city, and closed with the words *Wir müssen wissen. Wir werden wissen.*[21]

Von Neumann, who had heard the first incompleteness theorem at Königsberg, found the second independently and wrote to Gödel on 20 November

[19] Skolem built the first such model in 1934.

[20] Hilbert (1899), Hilbert (1900).

[21] Hilbert (1930); the words are the epitaph on his grave in Göttingen. On the two days see Dawson Jr. (1997).

1930 to say so. Gödel had announced both a few weeks earlier. The first was published in 1931, together with a sketch of the second.[22]

Gödel's proof is not the one of this chapter. He coded formulas and derivations by numbers. He showed that the relation "p is the number of a derivation of the sentence with number f" can be expressed in the formal system itself, with its true instances provable there. Then, by substituting a formula's own number into it, he built a sentence that says of itself that it is not derivable: the sentence of Remark 8.36, reached by coding instead of by a program. The general form of this step, the *fixed-point lemma*, was stated by Carnap in 1934.

From consistency alone Gödel's argument shows that the sentence is not derivable; to show that its negation is not derivable either, he assumed ω-consistency, a condition between consistency and soundness. Rosser removed that assumption in 1936, by the cleverer sentence of Remark 8.41.[23]

The second theorem did not leave the consistency of arithmetic in doubt. In 1936 Gentzen, whose natural deduction this course uses, proved PA consistent by an induction longer than any that PA itself can carry out, as the second theorem says he must.[24]

Computation came next. In 1936 Church showed that no effectively calculable function decides whether a term of his λ-calculus has a normal form (the halting theorem, Theorem 8.2, is the same fact for its cousin, the combinators). In a short note of the same year he deduced that no mechanical procedure decides first-order provability, and so, by the completeness theorem, validity.[25] Turing reached the same conclusion independently and by a different route, and the next year showed that his machines compute exactly the functions that Church's calculus defines, the fact behind the Turing completeness of the combinators.[26]

The first of Turing's papers defines his machines, proves by a diagonal argument that no machine decides whether a machine is "circle-free", and deduces that none decides whether a machine ever prints a given symbol. In its eleventh section it writes, for each machine, a formula of first-order logic that is derivable exactly when the machine prints that symbol: an ancestor of the construction of Section 8.1, with machines in place of combinators and printing in place of halting. The paper does not state the undecidability of the halting problem, and does not use the word "halting". The self-referential argument for halting in its now standard form is in Kleene's *Introduction to Metamathematics* of 1952, and the name "halting problem" first appears in print in Davis's *Computability and Unsolvability* of 1958. But Turing's printing problem and the halting problem are equivalent, and the credit given to Turing is fair in substance if not in letter.[27]

What "formal system" meant was settled only by the machines. In 1963, and again in 1964, Gödel wrote that Turing's analysis gave "a precise and

[22] Gödel (1931).
[23] Rosser (1936).
[24] Gentzen (1936).
[25] Church (1936a,b).
[26] Turing (1937a), Turing (1937b).
[27] Hamkins and Nenu (2025); Kleene (1952), p. 382; Davis (1958), p. 70.

unquestionably adequate definition of the general notion of formal system" (in 1964, "concept" in place of "notion"), and that the incompleteness theorems therefore hold for every consistent formal system that contains a little arithmetic.[28]

The decidable cases came first. Presburger showed in 1929 that the counting numbers with addition alone can be captured and decided. By 1930 Tarski had a decision procedure for the real numbers with addition and multiplication, though it was published only after the war. In 1974 it was shown that every procedure for Presburger's structure is enormously expensive.[29] The last word on the undecidable side came in 1970, when Matiyasevich completed the work of Davis, Putnam and Julia Robinson and answered the tenth of Hilbert's problems of 1900: no procedure decides whether a polynomial equation with integer coefficients has a solution in whole numbers.[30]

The schoolroom problem has a history of its own. The eleven identities are in Dedekind's *Was sind und was sollen die Zahlen?* of 1888. Tarski asked in the 1960s whether they derive every true identity; the question is in print in a paper of 1969 with Doner.[31] Wilkie found his identity in 1981, and the paper describing it was published in 2000.[32] Gurevič gave the first counter-model, with 59 elements, in 1985, and proved in 1990 that no finite list of identities derives every true one.[33] Smaller counter-models followed, down to a twelve-element one of Burris and Yeats, who conjectured that twelve is the least. A computer search by Zhang in 2005 ruled out fewer than eleven.[34] Meanwhile Macintyre had shown in 1981 that the set of all true identities is decidable.[35] In 2026 Subercaseaux and Przybocki proved the conjecture with a SAT solver, as the introduction describes, and classified the twelve-element counter-models: up to isomorphism there are 8 957 952 of them.[36] At about the same time, and with a SAT encoding of their own, Hajdari and Niederhauser also ruled out eleven elements and found twelve-element counter-models other than that of Burris and Yeats.[37]

Every undecidability result of the chapter is a diagonal argument, and the diagonal is older than all of them. Figure 8.5 lines up four of its appearances. Each starts from a list, applies each entry of the list to its own index, and twists the result, so that the twisted diagonal differs from every entry of the list. Figure 8.6 collects the dates.

Summary. A program is a combinator, and no program decides whether a program halts: a predictor of halting, run on its own code, would contradict itself. Chapter 1's rules for running a combinator, read as sentences, make

[28]The note added in 1963 to the English translation of the 1931 paper, and the postscript of 1964 to his Princeton lectures of 1934; both are in Gödel (1986).

[29]Presburger (1930), Tarski (1951), Fischer and Rabin (1974).

[30]Matiyasevich (1970).

[31]Doner and Tarski (1969).

[32]Wilkie (2000).

[33]Gurevič (1985, 1990).

[34]Burris and Yeats (2004), Zhang (2005).

[35]Macintyre (1981).

[36]Subercaseaux and Przybocki (2026).

[37]Hajdari and Niederhauser (2026).

Who	**The list**	**Each applied to itself**	**The twist**	**What follows**
Cantor, 1891	infinite sequences of 0s and 1s	the nth digit of the nth sequence	change it	the list misses a sequence
Russell, 1901	all sets	is the set a member of itself?	take those that are not	the sets not members of themselves form no set
halting	all programs	does the program halt on its own code?	do the opposite	no program decides halting
the program D	the theorems of Θ	does Θ say program e does not halt on e?	halt exactly then	Θ misses a truth

FIGURE 8.5. One argument, four times.

a finite theory Π, the same for every program: a derivation by the rules is a derivation in NK from Π, and the combinators themselves form a model of Π, so "the program halts" follows from Π exactly when it does. So first-order validity is undecidable, even for the consequences of that one set. The first-order procedure `REFUTE` certifies every entailment that holds and runs for ever on some that do not, and that is as good as any procedure can be.

A specification is a theory that a program can recognise. It captures a structure when the sentences that follow from it are exactly those true in the structure. If a specification captures a structure, a program decides the structure's truths by listing theorems until the sentence or its negation appears. The converse holds too, so a structure can be captured exactly when its truths can be decided. Computation, whose truths include which combinators halt, cannot be captured. Once its elements have names, every finite structure in a finite signature is captured by finitely many sentences and the equality axioms: name the elements, say that they are different and that there are no others, and copy the tables.

The counting numbers contain computation: combinators can be coded as numbers, and steps, normal forms and halting defined with $+$ and $\times$, so every sentence about computation translates into a sentence about numbers with the same truth value. So the truths about the counting numbers are undecidable, and no specification captures them. For each sound specification, the program that waits for it to say "this program does not halt on itself" gives a true sentence the specification does not prove. For a specification that proves every true sentence about halting, as one that contains Peano arithmetic does, consistency is enough in place of soundness. And a consistent specification that contains Peano arithmetic cannot prove its own

Who	When	What
Dedekind	1888	the eleven identities
Hilbert	1899; 1900	axioms for geometry; the consistency of arithmetic asked for
Presburger	1929	addition alone is decidable
Tarski	by 1930; 1948	the real numbers are decidable
Gödel	1930; 1931	the incompleteness theorems
Church	1936	having a normal form is undecidable
Church; Turing	1936	first-order validity is undecidable
Rosser	1936	a sentence neither proved nor refuted, from consistency alone
Gentzen	1936	the consistency of PA, from outside it
Kleene; Davis	1952; 1958	the halting argument; the name
Matiyasevich	1970	polynomial equations are undecidable
Wilkie	1981	a true identity the eleven do not derive
Gurevič	1985; 1990	a counter-model; no finite list suffices
Subercaseaux, Przybocki	2026	the least counter-model has twelve elements; all of them classified
Hajdari, Niederhauser	2026	no eleven-element counter-model, by a second encoding

FIGURE 8.6. Incompleteness: who did what.

consistency; by completeness, it does not even entail it. The first theorem does not show that some truths can never be known: the true sentence it names was proved true, by an argument that uses the soundness the specification cannot supply for itself (Remark 8.35).

This part of the course began with the wish never to be surprised by a system one has specified. Its three chapters have said where that wish can be met. For a system with finitely many elements, a finite list of sentences can say everything, and a program can check every claim. For a question in propositional logic, a solver can settle it, quickly in practice. For a first-order question, a search finds every proof there is, and may search for ever when there is none. For the counting numbers, the simplest infinite system there is, no specification says everything, and no program decides what is true. That is not a failure to know: it is a theorem, which is a way of knowing, and it is what Hilbert's "we shall know" turned out to mean.

HILBERT'S PROGRAMME — STATUS.

Item	Status after this chapter
1. Consistency	• **holds**: semantically, a sound specification of arithmetic is consistent, because $\mathfrak{N}$ satisfies it × **fails**: as a finitary proof, no consistent specification containing PA proves its own consistency
2. Completeness	× **fails**: no specification captures $\mathfrak{N}$; the shortfall is in the specification, and NK itself stays complete
3. Decision procedure	× **fails**: first-order validity and truth in $\mathfrak{N}$ are undecidable, and the first-order procedure's half answer is the best any procedure can give

• holds ○ open, or settled in part × fails

Hilbert asked three questions of a theory: whether it is consistent, whether it derives every truth (for NK, every truth in every model of the theory; for arithmetic, every truth about $\mathfrak{N}$), and whether its consequences can be decided. For first-order logic with the rules of NK, each has an answer; for the counting numbers, none has the answer he wanted. What remains decidable is still large: every finite structure, propositional logic, and the structures of the optional Section 8.6.

Exercises. The exercises run from practice to proof.

- Exercises 8.1 to 8.5 are about programs, the theory of computation and reductions;
- Exercises 8.6 to 8.10 are about specifications and finite structures;
- Exercises 8.11 to 8.13 are about arithmetic and the incompleteness theorems;
- Exercises 8.14 to 8.16 go with the optional sections, and are optional too.

Exercise 8.1. The combinator $\mathsf{K}\,(\mathsf{I}\,\mathsf{K})$, that is, $\mathsf{K}\,(\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,\mathsf{K})$, halts. Write its run, and the lines by which the rules of Π derive that it halts, as in Example 8.5. Which sentences for steps are used, and where?

Exercise 8.2. Let X be a set of combinators that contains K and S and contains $\mathsf{K}\,N$, $\mathsf{S}\,N$ and $\mathsf{S}\,N\,N'$ whenever it contains N and N'. Show that X contains every combinator in normal form. Show that the set of combinators in normal form is itself such a set. So it is the smallest one, which is why $\mathfrak{T}$ interprets Nf by it.

Exercise 8.3. Let $\mathfrak{G}$ be the structure whose domain is the set of combinators, with K, S and application as in the structure $\mathfrak{T}$ of the proof of Lemma 8.6, in which Step, Nf and Halts are true of everything. Show that $\mathfrak{G}$ satisfies Π and makes $\mathsf{Halts}(\Omega)$ true. Why does this not contradict the lemma that a program is a theory (Lemma 8.6)? Why does the proof of part (iv) of that lemma use $\mathfrak{T}$, and not just any model of Π?

Exercise 8.4. Compute the code of $\mathsf{I} = \mathsf{S}\,\mathsf{K}\,\mathsf{K}$. Which numbers below 500 are codes, and of which combinators?

Exercise 8.5. Show that no program decides, given the code of a program, whether that program halts on the input 0. (Find a reduction: from e and n, compute the code of a program that ignores its input and does what program e does on n.)

Exercise 8.6. Say which of the following sets of sentences are specifications, with a reason for each.

(a) The eleven identities of Remark 8.37, each with $\forall$ in front of its variables.
(b) The induction axioms of PA.
(c) The set of all sentences true in a structure with five elements.
(d) The set of all sentences of Σ_{A} true in $\mathfrak{N}$.
(e) The set of sentences of Σ_{A} with fewer than a hundred symbols that are true in $\mathfrak{N}$.

Exercise 8.7. Let $A = 1 + x$, $B = 1 + x + x^2$, $C = 1 + x^3$, $D = 1 + x^2 + x^4$ and $E = 1 - x + x^2$.

(a) Show that $C = A \cdot E$ and $D = B \cdot E$.
(b) Deduce that $(A^y + B^y)^x \cdot (C^x + D^x)^y = (A^x + B^x)^y \cdot (C^y + D^y)^x$ for all positive integers x and y.

(c) Check that $E \geq 1$ for every positive integer x, so that every quantity in the argument is a positive integer. Which step of the argument cannot be imitated with the eleven identities, and why?

Exercise 8.8. Let $\mathfrak{M}$ be a structure whose truths are decided by a program. Prove in full that $\mathrm{Th}(\mathfrak{M})$ is a specification and that it captures $\mathfrak{M}$. Which half of capturing uses the soundness of NK?

Exercise 8.9. The structure $\mathfrak{B}$ has the domain $\{a, b, c\}$, and its signature has three symbols.

- A one-place function symbol f, with $f(a) = b$, $f(b) = c$ and $f(c) = c$.
- A one-place relation symbol P, true of a alone.
- The relation symbol $=$, interpreted as identity.

(a) Write the description $\Delta_{\mathfrak{B}}$ of $\mathfrak{B}$ (Remark 8.20), and count its sentences.

(b) The sentence $\forall x\,(P(x) \to \neg P(f(x)))$ is true in $\mathfrak{B}$. Following the idea in Remark 8.20, describe a derivation of it from the equality axioms and $\Delta_{\mathfrak{B}}$: which instances are proved, and how are they combined?

Exercise 8.10. Take the equality axioms and the description $\Delta_{\mathfrak{B}}$ of Exercise 8.9, and remove the closure sentence. Find a model of what is left in which the sentence $\forall x\,(f(x) = x \to x = c_c)$ is false, although it is true in $\mathfrak{B}$. What does the example show about the closure sentence?

Exercise 8.11. Write in Σ_{A}, using *Prime* and $x \mid y$ as abbreviations and with every quantifier's scope bracketed:

(a) there are infinitely many pairs of primes that differ by 2;

(b) every number is the sum of four squares;

(c) every number greater than 1 is divisible by a prime.

Exercise 8.12. Let Θ be a specification in Σ_{A} and γ the sentence γ_Θ of its diagonal program.

(a) Which step of the proof of the first incompleteness theorem fails if Θ is not assumed sound?

(b) Suppose that Θ is sound, and let Θ' be Θ with γ added. Show that Θ' is a sound specification. Does the first incompleteness theorem apply to Θ', and with which sentence?

Exercise 8.13. Add a fifth row to the table of diagonal arguments in the History, for Cantor's theorem that no function f from a set A to the set of its subsets is onto: the list, each applied to itself, the twist, and what follows.

Exercise 8.14. *(Optional.)* Find numbers a and b with $\beta(a, b, i)$ equal to 1, 0, 1 for $i = 0, 1, 2$, first by the method of the proof of the packing lemma, and then a smaller pair by trial.

Exercise 8.15. *(Optional.)* Write out the formula $\mathit{Red}_{\mathsf{S}}(i, p, \ell_P, \ell_Q, \ell_R)$ that Section 8.5 leaves to the reader, in the style of $\mathit{Red}_{\mathsf{K}}$. Then say where, in row $i + 1$ of an S step, the six fixed digits stand, and where each of the six copies begins.

Exercise 8.16. *(Optional.)* Eliminate the quantifier from $\exists x\,(y < x \wedge x < z \wedge x \neq w)$ in DLO, following the four moves of the proof, and simplify the result.

References.

Stanley Burris and Karen Yeats. The saga of the high school identities. *Algebra Universalis*, 52:325–342, 2004.

Alonzo Church. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1(1):40–41, 1936a.

Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936b.

William Craig. On axiomatizability within a system. *Journal of Symbolic Logic*, 18 (1):30–32, 1953.

Martin Davis. *Computability and Unsolvability*. McGraw-Hill, New York, 1958.

John W. Dawson Jr. *Logical Dilemmas: The Life and Work of Kurt Gödel*. A K Peters, Wellesley, MA, 1997.

John Doner and Alfred Tarski. An extended arithmetic of ordinal numbers. *Fundamenta Mathematicae*, 65:95–127, 1969.

Michael J. Fischer and Michael O. Rabin. Super-exponential complexity of Presburger arithmetic. In *Complexity of Computation*, volume 7 of *SIAM–AMS Proceedings*, pages 27–41. American Mathematical Society, 1974.

Gerhard Gentzen. Die Widerspruchsfreiheit der reinen Zahlentheorie. *Mathematische Annalen*, 112:493–565, 1936.

Alexander V. Gheorghiu. High school algebra and the limits of AI. *Mathematics Today*, 61(5), October 2025. Graham Hoare Prize 2025.

Kurt Gödel. Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198, 1931.

Kurt Gödel. *Collected Works, Volume I: Publications 1929–1936*. Oxford University Press, Oxford, 1986. Edited by Solomon Feferman and others.

R. Gurevič. Equational theory of positive numbers with exponentiation. *Proceedings of the American Mathematical Society*, 94:135–141, 1985.

R. Gurevič. Equational theory of positive numbers with exponentiation is not finitely axiomatizable. *Annals of Pure and Applied Logic*, 49:1–30, 1990.

Agon Hajdari and Johannes Niederhauser. SATisfying the high school identities but not Wilkie's identity. arXiv:2608.16406, 2026.

Petr Hájek and Pavel Pudlák. *Metamathematics of First-Order Arithmetic*. Perspectives in Mathematical Logic. Springer, Berlin, 1993.

Joel David Hamkins and Theodor Nenu. Did Turing prove the undecidability of the halting problem? arXiv:2407.00680v3, 2025.

David Hilbert. *Grundlagen der Geometrie*. Teubner, Leipzig, 1899.

David Hilbert. Mathematische Probleme. Lecture to the International Congress of Mathematicians, Paris. Published in Nachrichten der Königlichen Gesellschaft der Wissenschaften zu Göttingen, 253–297, 1900.

David Hilbert. Naturerkennen und Logik. *Die Naturwissenschaften*, 18:959–963, 1930.

Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland, Amsterdam, 1952.

Angus Macintyre. The laws of exponentiation. In *Model Theory and Arithmetic*, volume 890 of *Lecture Notes in Mathematics*, pages 185–197. Springer, 1981.

Yuri V. Matiyasevich. Enumerable sets are Diophantine. *Soviet Mathematics Doklady*, 11:354–357, 1970.

Open Logic Project. The Open Logic Text. `https://openlogicproject.org`, 2026. Release 9620cc7, 12 July 2026. Licensed CC BY 4.0.

Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I Congrès des Mathématiciens des Pays Slaves, Warszawa 1929*, pages 92–101, 1930.

J. Barkley Rosser. Extensions of some theorems of Gödel and Church. *Journal of Symbolic Logic*, 1(3):87–91, 1936.

Bernardo Subercaseaux and Benjamin Przybocki. A SAT attack on Tarski's high school algebra problem. arXiv:2608.08421v2, 2026.

Alfred Tarski. *A Decision Method for Elementary Algebra and Geometry*. University of California Press, Berkeley, second edition, 1951. First issued as a RAND report, 1948.

Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937a.

Alan M. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2(4): 153–163, 1937b.

Alex J. Wilkie. On exponentiation – a solution to Tarski's high school algebra problem. *Quaderni di Matematica*, 6:107–129, 2000. Manuscript circulated 1981.

Jian Zhang. Computer search for counterexamples to Wilkie's identity. In *Automated Deduction – CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 441–451. Springer, 2005.