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. Specifications 2
8.2. Finite Structures 6
8.3. Programs as Theories 9
8.4. Undecidability of First-Order Logic 17
8.5. Arithmetic as a Theory 19
8.6. First Incompleteness Theorem 22
8.7. Second Incompleteness Theorem 25
8.8. Defining Halting *(optional)* 28
8.9. Decidable Theories *(optional)* 34
History 37
Exercises 42

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.

How can anyone know that the rules cannot derive it? Not by trying and failing, since failing to find a derivation proves nothing. The way to show that something does not follow from rules is to exhibit a world in which the rules hold and the claim fails. For Wilkie's identity the smallest such world has twelve elements, and its tables for addition, multiplication and powers obey all eleven rules. In 2026 a computer proved, with a SAT solver, that no smaller one exists.

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.

The schoolroom is a small version of a large hope. For two thousand years Euclid's geometry was the model of knowledge: a few axioms, and every truth of the subject derived from them. In 1899 Hilbert rewrote the axioms of geometry so that every assumption was written down. 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.

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

In the counting numbers every statement that can be written down is true or false: that is what the definition of truth in a structure says. Goldbach's conjecture, that every even number greater than 2 is a sum of two primes, can be written as one sentence of first-order logic. Nobody knows whether it is true or false. If there were axioms from which every true statement about the numbers could be derived, then Goldbach's conjecture or its negation would be derivable, and settling it would be a matter of searching.

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

Two questions about theories and programs have been left open so far. They are the questions of this chapter. The first is about searching. The first-order procedure `REFUTE` (Theorem 7.63) 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 second question is about describing. Even a perfect oracle for "does this follow from the theory?" tells us only what follows from the theory. 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? Call the two questions, about searching and about describing, the *search gap* and the *specification gap*.

This chapter shows that neither gap can be closed in general, and that one construction answers both questions: the rules by which programs run can be written down as sentences.

The plan has four steps. First, Section 8.1 makes the specification gap precise: a specification *captures* a structure when it proves exactly the sentences true there, and then a program can decide every question about the structure. Section 8.2 shows that every finite structure can be captured. Second, the search gap. Section 8.3 shows that no program decides whether a program halts, and writes the rules by which programs run as a theory, so that Section 8.4 can carry the result into logic: no program decides first-order validity. Third, the specification gap for the counting numbers. Section 8.5 shows that halting can be expressed in arithmetic, so no program decides arithmetic and no specification captures it, and Section 8.6, the first incompleteness theorem, names, for any sound specification, a true sentence that it does not prove. Fourth, a limit of specifications that does not depend on the structure we meant at all. Section 8.7, the second incompleteness theorem, shows that a consistent specification of arithmetic does not entail its own consistency: some sentences about a specification do not follow from it in first-order logic. The two optional sections prove in full that halting can be expressed (Section 8.8) and ask which infinite structures can be captured (Section 8.9).

8.1. **Specifications.** We describe a system, a structure $\mathfrak{M}$, by a theory Θ, and we want the sentences that follow from Θ to be exactly the sentences true of $\mathfrak{M}$. To say this precisely we need a name for the set of truths of $\mathfrak{M}$, a condition that makes a theory usable, and a meaning for "exactly". One fact

then makes the question sharp: a theory that does the job lets a program decide the truths of $\mathfrak{M}$.

Definition 8.1 (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.

A theory has to be one that someone can use. That asks for one thing more than a set of sentences: a program that recognises its members. Until Section 8.3 makes the word exact, a *program* is anything that could be written in pseudocode: it takes an input, takes steps one at a time, and may or may not stop.

Definition 8.2 (specification)**.** A specification is a theory that a checker can read: given any sentence, it can tell whether the sentence 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.[2]

The examples are the sets one would naturally write down.

- A finite set of sentences is a specification: the program compares the given sentence with each of them.
- The equality axioms of a signature that contains $=$ (Definition 6.36 of Chapter 6 defines them for any signature) are a specification. They are (Eq1), which says that everything is equal to itself, and the instances of (Eq2), which say that equals may be put for equals in a formula. There are infinitely many instances, 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 a theory whose members nobody can recognise is no use as a specification. 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.

The set $\mathrm{Th}(\mathfrak{M})$ is a set of sentences, and so "take all the true ones" looks like the best specification of $\mathfrak{M}$ there could be. Whether it is one depends on whether a program can recognise the true sentences. For the counting numbers it cannot, as Section 8.5 shows. "All the true ones" describes a set; it does not give a checker.

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

Definition 8.3 (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,

$$\Theta \vdash_{\mathrm{NK}} \varphi \quad \text{if and only if} \quad \mathfrak{M} \models \varphi.$$

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.

Remark 8.4 (two senses of "complete")*.* The second half is often called *completeness* of Θ, and the word has been used before. The completeness of NK is a property of the proof system: whatever is true in *every* model of Θ can be derived from Θ. The second half of capturing is a property of the 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 the second half of capturing. The gap between the two senses is the specification gap.

To keep the two senses apart we say "captures", and never "complete", for the property of a theory.

Example 8.5 (the schoolroom)**.** Take the signature with a constant 1 and two-place function symbols $+$, $\cdot$ and $\uparrow$, where $x \uparrow y$ is x^y, and let $\mathbb{P}$ be the positive integers with their usual operations. Let Θ_{HS} be the equality axioms together with the eleven identities on the board, each with $\forall$ in front of its variables. The set Θ_{HS} is a specification, and $\mathbb{P} \models \Theta_{\mathrm{HS}}$, so it proves only truths.

It does not prove every truth. Write W for Wilkie's identity with $\forall x\, \forall y$ in front. The sentence W is true in $\mathbb{P}$, and $\Theta_{\mathrm{HS}} \nvdash_{\mathrm{NK}} W$. The certificate is a structure: 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 x, y.[3] Were W provable from Θ_{HS}, the soundness of NK would make it true in that structure.

Remark 8.6 (at work: a counter-model found by a SAT solver)*.* The twelve-element structure of the introduction, in which the eleven identities hold and Wilkie's identity fails, consists of three tables of a hundred and forty-four entries each. Tables like these are not found by hand. The encoding has one propositional letter for each possible entry of each table, clauses saying that each entry has exactly one value, and clauses for the eleven laws and for the failure of the twelfth. In 2026 a SAT solver listed the smallest counter-models, and said "no" for eleven elements: about seventy-five thousand letters and four and a half million clauses. The "no" came with a checkable proof, 368 megabytes long, which an independent program

[3] 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.

verified.[4] Both kinds of certificate appear here: a structure, which anyone can check by plugging in, certifies that a sentence does not follow; a closed search, written down as a proof, certifies that no smaller structure does.

A specification that misses a truth does not show that no specification captures the structure. The identities true of the positive integers with 1, $+$, $\cdot$ and powers can be decided by a program.[5] So the set of all of them is itself a specification, from which every true identity follows at once. What fails at school is every *finite* list, the eleven identities in particular. For the counting numbers, with $+$, $\times$ and the quantifiers, no specification at all will do, finite or infinite.

Proving that no specification will do means knowing what any specification can do. It can at least produce its theorems, one after another, because a derivation is a finite string that a checker can read.

Lemma 8.7 (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.

Theorem 8.8 (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
```

[4] 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.

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

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 $\mathfrak{M}$.

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

Proposition 8.9 (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 the theorem that capturing makes truth decidable. 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 8.10 (deciding with `REFUTE`)*.* The program `DECIDE` searches through derivations, which is slow. The first-order procedure `REFUTE` (Theorem 7.63) does the same job faster. It stops on a clause set exactly when the set is unsatisfiable. So if a specification Θ that captures $\mathfrak{M}$ can be written as a finite set of clauses, then for a sentence φ, `REFUTE` stops on $\Theta \cup \{\neg\varphi\}$ exactly when $\Theta \models \varphi$, and on $\Theta \cup \{\varphi\}$ exactly when $\Theta \models \neg\varphi$. Run it on both sets, taking a step of each run in turn. One of φ and $\neg\varphi$ is true in $\mathfrak{M}$ and, because Θ captures $\mathfrak{M}$, is entailed by Θ. So one of the two runs must stop, and which one says whether $\mathfrak{M} \models \varphi$.

8.2. **Finite Structures.** Finite structures are the one kind for which the specification gap can always be closed.

On a finite structure, the definition of truth is itself an algorithm: each quantifier runs through the domain, and a sentence with k nested quantifiers on n elements takes about n^k cases. So the truths of a finite structure, for a finite signature, can be decided. A structure whose truths can be decided is captured by the set of all its truths, by the proposition that capturing is deciding, and so every finite structure can be captured.

That specification is the list of all the truths, which is infinite and of no use to anyone. This section shows that a finite list will do, and that it is the list one would write down anyway: give each element a name, and copy out the tables.

Definition 8.11 (the description of a finite structure)**.** The description of a finite structure names each element and says, in sentences, everything its tables say.

Formally, let Σ be a finite signature that contains $=$, and let $\mathfrak{M}$ be a Σ-structure with the finite domain $\{d_1, \ldots, d_n\}$ that interprets $=$ as identity. Let $\Sigma_{\mathfrak{M}}$ be Σ with a new constant c_d for each element d, and let $\mathfrak{M}^+$ be $\mathfrak{M}$ with each c_d interpreted as d, so that forgetting the new constants gives back $\mathfrak{M}$: $\mathfrak{M}^+ \restriction \Sigma = \mathfrak{M}$. The *description* $\Delta_{\mathfrak{M}}$ of $\mathfrak{M}$ is the set of the following $\Sigma_{\mathfrak{M}}$-sentences.

(i) *Distinctness*: $c_d \neq c_e$ for each pair of different elements d, e, one sentence per pair.

(ii) *Closure*: $\forall x\,(x = c_{d_1} \vee \cdots \vee x = c_{d_n})$.

(iii) *The function tables*: $f(c_{e_1}, \ldots, c_{e_m}) = c_e$ whenever $[\![f]\!]^{\mathfrak{M}}(e_1, \ldots, e_m) = e$, for each function symbol f of Σ (a constant of Σ counting as a function symbol with no arguments).

(iv) *The relation tables*: $P(c_{e_1}, \ldots, c_{e_m})$ if the tuple $(e_1, \ldots, e_m)$ is in $[\![P]\!]^{\mathfrak{M}}$, and $\neg P(c_{e_1}, \ldots, c_{e_m})$ if it is not, for each relation symbol P other than $=$ and each tuple of elements.

The description is finite, since the signature and the domain are. Together with the equality axioms for $\Sigma_{\mathfrak{M}}$, it is therefore a specification.

Theorem 8.12 (finite structures can be captured)**.** *For every finite signature Σ that contains $=$ and every finite Σ-structure $\mathfrak{M}$ that interprets $=$ as identity, the equality axioms for $\Sigma_{\mathfrak{M}}$, together with the description $\Delta_{\mathfrak{M}}$, capture $\mathfrak{M}^+$.*

In words: say who is there, that they are different, that there is nobody else, and every entry of every table; then everything true has been said.

Proof. Under the closure sentence every universal quantifier becomes a finite conjunction and every existential quantifier a finite disjunction, and what remains are entries of the tables. In detail, write Θ for the equality axioms together with $\Delta_{\mathfrak{M}}$. Then $\mathfrak{M}^+ \models \Theta$: $\mathfrak{M}^+ \models \Delta_{\mathfrak{M}}$ by construction, and $\mathfrak{M}^+$ satisfies the equality axioms since it interprets $=$ as identity. So Θ proves only truths, by the soundness of NK. For the other half we show that, for every sentence φ,

$$\Theta \vdash_{\mathrm{NK}} \varphi \text{ if } \mathfrak{M}^+ \models \varphi \qquad \text{and} \qquad \Theta \vdash_{\mathrm{NK}} \neg\varphi \text{ if } \mathfrak{M}^+ \not\models \varphi.$$

The proof is by induction on the number of connectives and quantifiers in φ. The induction can be on this number because writing a constant for a variable does not change it. Two facts about $=$ are used throughout.

- Equality is symmetric and transitive: this follows from (Eq1) and (Eq2) (Exercise 6.8).
- A derivation may replace a term by one it has proved equal to it: this is (Eq2).

First, every closed term t is provably equal to the name of its value: $\Theta \vdash_{\mathrm{NK}} t = c_e$, where e is the value of t in $\mathfrak{M}^+$. The proof is by induction on t.

- *A new constant* c_e. It is its own name, and $c_e = c_e$ is an instance of (Eq1).
- *A term* $f(t_1, \ldots, t_m)$, where f is a function symbol of Σ or a constant of Σ (with $m = 0$). The induction hypothesis gives $t_i = c_{e_i}$ for each i, the function table gives $f(c_{e_1}, \ldots, c_{e_m}) = c_e$, and (Eq2) joins them.

Now the cases of φ.

- *An atomic sentence* $P(t_1, \ldots, t_m)$. Replace each t_i by the name of its value, by the fact just proved and (Eq2). The relation table contains the resulting sentence or its negation, according as $\mathfrak{M}^+$ makes it true or false. An equation $t_1 = t_2$ goes the same way, to $c_d = c_e$, which (Eq1) proves if d is e and distinctness refutes if not.
- $\bot$. It is false in $\mathfrak{M}^+$, and $\neg\bot$, which is $\bot \to \bot$, has a derivation from nothing.
- $\psi \wedge \chi$, $\psi \vee \chi$, $\psi \to \chi$. The induction hypothesis settles ψ and χ, and propositional reasoning in NK settles the compound as the satisfaction clause does. For instance, if $\psi \to \chi$ is false in $\mathfrak{M}^+$, then ψ is true and χ false, so Θ proves ψ and $\neg\chi$, and with them $\neg(\psi \to \chi)$.
- $\forall x\, \psi$, *true in* $\mathfrak{M}^+$. Then each $\psi[c_d/x]$ is true in $\mathfrak{M}^+$, by the substitution lemma, and has fewer connectives and quantifiers than $\forall x\, \psi$, so Θ proves each of them. Take a variable y not in ψ. The closure sentence gives $y = c_{d_1} \vee \cdots \vee y = c_{d_n}$. In the case $y = c_{d_i}$, (Eq2) turns $\psi[c_{d_i}/x]$ into $\psi[y/x]$, so ($\vee$E), applied to the n cases, gives $\psi[y/x]$. Then ($\forall$I) gives $\forall y\, \psi[y/x]$. The rule applies because y occurs in no open hypothesis: all of them are sentences of Θ. Finally, $\forall y\, \psi[y/x]$ is $\forall x\, \psi$ with its bound variable renamed.
- $\forall x\, \psi$, *false in* $\mathfrak{M}^+$. Then some $\psi[c_d/x]$ is false, so Θ proves $\neg\psi[c_d/x]$. Suppose $\forall x\, \psi$. Then ($\forall$E) gives $\psi[c_d/x]$, which with $\neg\psi[c_d/x]$ gives $\bot$ by ($\to$E); and ($\to$I) discharges the supposition. So Θ proves $\neg\forall x\, \psi$.
- $\exists x\, \psi$. The same two arguments with the roles exchanged: if it is true, one instance is proved and ($\exists$I) finishes; if it is false, every instance is refuted, and the closure sentence refutes $\exists x\, \psi$ as it proved a true $\forall x\, \psi$. □

A Σ-sentence is true in $\mathfrak{M}^+$ exactly when it is true in $\mathfrak{M}^+{\restriction}\Sigma = \mathfrak{M}$, so the specification proves exactly the Σ-sentences true in $\mathfrak{M}$.

The closure sentence is the one sentence with no counterpart for an infinite structure, since no finite list of names covers infinitely many elements. The other sentences of $\Delta_{\mathfrak{M}}$ describe the elements that are there. The closure sentence says that there are no others, and so tells every model of it *where the structure stops*. Without it, a model of the remaining sentences may have an extra element, of which the tables say nothing. A sentence true in $\mathfrak{M}$ may then fail at the extra element (Exercise 8.5).

Example 8.13 (the company: a specification of Tuesday's table)**.** Harriet writes the specification that the theorem gives for the company on Tuesday, the structure $\mathfrak{C}$ of Chapter 6. She adds the names thorne, finch, pratt, bell and cole. The tables for ceo and boss give six equations, the first of which is

$$\mathsf{boss}(\mathsf{thorne}) = \mathsf{thorne}.$$

Thorne, the chief executive, is her own boss, because boss is a function symbol and must give every person a value, Thorne included. The rest of the description is

- the ten sentences of distinctness;
- the closure sentence

$$\forall x\,(x = \mathsf{thorne} \vee x = \mathsf{finch} \vee x = \mathsf{pratt} \vee x = \mathsf{bell} \vee x = \mathsf{cole});$$

- the ten entries for Eng and Mgr, such as $\mathsf{Eng}(\mathsf{bell})$ and $\neg\mathsf{Eng}(\mathsf{thorne})$;
- the twenty-five entries for Richer, one for each pair of people, such as $\mathsf{Richer}(\mathsf{thorne}, \mathsf{finch})$ and, the other way round, $\neg\mathsf{Richer}(\mathsf{finch}, \mathsf{thorne})$.

That is fifty-two sentences besides the equality axioms. Call the whole specification Θ_{C}. By the theorem, a derivation from Θ_{C} settles every question about Tuesday's table that the company's language can ask.

Example 8.14 (the company, cont'd: deciding by search)**.** Harriet runs the procedure of Remark 8.10 on Θ_{C}. The equality axioms go in as finitely many clauses, and the specification becomes sixty clauses.

Take for φ the sentence $\forall x\,\mathsf{Richer}(\mathsf{boss}(x), x)$, that everybody earns less than their boss. The run with φ closes at level 1, on 5823 ground clauses, by forced moves alone, while the run with $\neg\varphi$ is still open. So the sentence is false: it fails at Thorne, who is her own boss. For the repaired sentence $\forall x\,(x \neq \mathsf{ceo} \to \mathsf{Richer}(\mathsf{boss}(x), x))$ it is the other way round: the run with its negation closes at level 1, on 9069 ground clauses, after five decisions, and the sentence is true. Each run takes under a second.[6]

The cost of such a specification is plain. It is as long as the tables of the structure, since it *is* those tables. It captures that one structure and nothing else. When the system changes, by one new element or one changed entry, the specification is wrong and has to be written again. That is why a system that changes is described not by a table, which says everything about one state alone, but by a theory, which says what must be true of every table the system can produce.

For a system with infinitely many elements no table can be written, and only a theory is left. The rest of the chapter asks whether a theory can ever say everything.

8.3. **Programs as Theories.** The search gap asks about every procedure, not just about `REFUTE`. To answer it we need to say exactly what a procedure is, and Chapter 1 has already given an answer: a procedure is a combinator. A term built from K and S computes by being rewritten, and the theorem on Turing completeness (Theorem 1.18) says that combinators compute everything that any machine computes.[7]

The plan is short to state. Chapter 1's four rules for a step, and its lemma on the shape of a normal form, can be written as a handful of first-order

[6] The script `inc_checks.py` in the course's support folder runs both.

[7] 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.

sentences, the same handful for every program. Then "the program halts" becomes "a certain sentence follows". So whatever cannot be decided about programs cannot be decided about sentences either. Whether a program halts, as we show first, is one such thing. That no program decides it is the source of both of the chapter's answers: Section 8.4 carries it into logic, and Section 8.5 into arithmetic.

The section takes seven things from Chapter 1, and needs nothing else from there.

- *Steps.* A combinator steps by four rules: (K), which takes $\mathsf{K}\,P\,Q$ to P; (S), which takes $\mathsf{S}\,P\,Q\,R$ to $P\,R\,(Q\,R)$; and (app_1) and (app_2), which make a step in the function, or in the argument, a step of the whole application. The formula $M \rightsquigarrow N$ says that M steps to N. A run is a chain of steps, $M \rightsquigarrow^* N$ says that M runs to N, and a combinator that cannot step is in *normal form*.
- *The shape of a normal form* (Lemma 1.7). A combinator is in normal form exactly when it is K, $\mathsf{K}\,N$, S, $\mathsf{S}\,N$ or $\mathsf{S}\,N\,N'$, where N and N' are combinators in normal form.
- *One result at most* (Corollary 1.8). A term runs to at most one normal form.
- *No step is a wrong turn* (Corollary 1.9). If M runs to a normal form N, then every term that M runs to also runs to N.
- *The Monster.* With $\mathsf{I} \coloneqq \mathsf{S}\,\mathsf{K}\,\mathsf{K}$ and $\omega \coloneqq \mathsf{S}\,\mathsf{I}\,\mathsf{I}$, which applies its argument to itself, the Monster is $\Omega \coloneqq \omega\,\omega$, that is, $\mathsf{S}\,\mathsf{I}\,\mathsf{I}\,(\mathsf{S}\,\mathsf{I}\,\mathsf{I})$. It has no normal form: every run of it goes on for ever.
- *The numerals.* With $\mathsf{B} \coloneqq \mathsf{S}\,(\mathsf{K}\,\mathsf{S})\,\mathsf{K}$, the numeral of a number is given by $\underline{0} \coloneqq \mathsf{K}\,\mathsf{I}$ and $\underline{n+1} \coloneqq \mathsf{S}\,\mathsf{B}\,\underline{n}$. The numeral of n applies its first argument n times to its second: $\underline{n}\,f\,x$ runs to $f\,(f\,(\cdots(f\,x)\cdots))$, with n copies of f.
- *Turing completeness* (Theorem 1.18). The functions on numerals that combinators compute are exactly those that a Turing machine computes.

The booleans are recalled where they are used.

Programs. A program's input is a combinator too, and the input for a number is its numeral. Chapter 1 wrote the numerals with a bar over the number. This chapter keeps the bar for the numerals of arithmetic, which are terms of first-order logic, and writes the numerals of the combinators with a bar beneath.

A *program*, in this chapter, is a combinator: a term built from K and S by application alone.

Definition 8.15 (halting)**.** A combinator halts when some run of it ends.

Formally, a combinator M *halts* if $M \rightsquigarrow^* N$ for some combinator N in normal form. A program M *halts on* the number n if the combinator $M\,\underline{n}$ halts.

In words: a program halts on n when, given the numeral of n, it has a result. By the corollary of Chapter 1 that a term has at most one result (Corollary 1.8), the result is then unique.

Halting asks that *some* run ends, not that every run does. The combinator $\mathsf{K}\,\mathsf{K}\,\Omega$, with Ω the Monster, halts: its K step gives K at once. But a run that keeps stepping inside Ω never ends.[8]

Codes. We want a program to take another program as its input. Inputs are numbers, so we number the programs. A combinator is a finite string, and the string we number is the one with every application bracketed, as in the definition of terms in Chapter 1.

Definition 8.16 (the string of a combinator)**.** The string of a combinator is the combinator written out with all its brackets.

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

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

For instance, $\mathrm{str}(\mathsf{K}\,\mathsf{K})$ is $(\mathtt{KK})$, and $\mathrm{str}(\mathsf{S}\,\mathsf{K}\,\mathsf{K})$ is $((\mathtt{SK})\mathtt{K})$.

Definition 8.17 (code)**.** The code of a combinator is the number its string spells in base five.

Formally, the *code* $\ulcorner M \urcorner$ of a combinator M is the number whose digits in base 5 are those of $\mathrm{str}(M)$, read with 1 for $\mathtt{K}$, 2 for $\mathtt{S}$, 3 for (and 4 for).

For instance, $\ulcorner \mathsf{K} \urcorner = 1$ and $\ulcorner \mathsf{K}\,\mathsf{K} \urcorner = 3\cdot 5^3 + 1\cdot 5^2 + 1\cdot 5 + 4 = 409$. Different combinators have different strings, and so different codes. A program can compute the code of a combinator, and the combinator of a code. Base five is a choice: any numbering that a program can compute and undo would serve, and this one is the shortest to describe.

A program given a number as its input may want to ask what the program with that number does. Not every number is a code, so we let the others name a program too, one that never halts.

Definition 8.18 (the program with code e)**.** Every number names a program: the combinator it is the code of, or a program that halts on no input if it is the code of none.

Formally, for a number e, the *program with code* e is

$$\mathrm{prog}(e) \coloneqq \begin{cases} M & \text{if } \ulcorner M \urcorner = e, \\ \mathsf{K}\,\Omega & \text{if } e \text{ is the code of no combinator.} \end{cases}$$

The second line is a convention, and any program that halts on no input would do as well. So $\mathrm{prog}(\ulcorner M \urcorner) = M$ for every combinator M. The second line gives a program that halts on no input: $\mathsf{K}\,\Omega\,\underline{n}$ steps to the Monster Ω, which has no normal form, so by the corollary of Chapter 1 that no step is a wrong turn (Corollary 1.9) it has none either.

[8] The run that always takes the leftmost of the outermost redexes reaches the normal form whenever there is one, so a program halts exactly when that run ends. This is the normalisation theorem for the leftmost order, a theorem of combinatory logic that the course does not prove; it goes back to Curry and Feys (1958). No proof in this chapter depends on it: it explains why the Prolog program of Remark 8.25 gets the right answers. The definitions here are a choice, and other textbooks make others (other numerals, a result required to be a numeral, or λ-terms in place of combinators); none of them changes what can be computed.

What a program cannot decide. We borrow one fact about programs from Chapter 1.

Fact 8.19 (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.[9]

The proof of the fact builds P from the pseudocode by a fixed method, instruction by instruction, and we call this P the *translation* of the pseudocode. Conversely, pseudocode can run a combinator step by step and read off its result; a combinator M answers a yes-or-no question about m and n when $M\,\underline{m}\,\underline{n}$ runs to K (yes) or KI (no), Chapter 1's booleans. So a program may be given in either form. Pseudocode may also take a string as its input, such as a sentence or a derivation: a string over a finite list of symbols is read as a number in a large enough base, just as the string of a combinator is read as its code.

The first thing no program can do is the one that Chapter 1 left open: tell in advance whether a program halts.

Theorem 8.20 (halting is undecidable)**.** *No program, given numbers e and n, always halts and says correctly whether* $\mathrm{prog}(e)$ *halts on n.*[10]

Proof. Suppose, for a contradiction, that a program h does. Here is a program that uses it:

```
procedure P(e)                        -- e a number
    if h says that prog(e) halts on e:
        loop for ever
    else: halt
```

By the fact that programs are combinators, P has a translation, a combinator that halts on exactly the inputs on which P halts. Let p be its code, so that $\mathrm{prog}(p)$ halts on p exactly when P does. Run P on p.

- *If P halts on p,* then h, given p and p, says so, and then P loops for ever on p.
- *If P does not halt on p,* then h says so, and then P halts on p.

Either way P halts on p exactly when it does not, which is impossible. So there is no such h. □

Figure 8.1 draws the proof as a table. Row e records, for each input n, whether $\mathrm{prog}(e)$ halts on n (marked ✓) or runs for ever (marked ∞); the entries shown are made up, but every program has a row. The program P runs down the diagonal and does the opposite of each entry it finds there. So its row differs from row e in column e, for every e, and it is no row of the table. Yet P has a code p, and its row is row p. At the entry in row p and column p, the two would have to differ from each other, which is the contradiction of the proof.

[9] Theorem 1.18 states Turing completeness 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.

[10] 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.

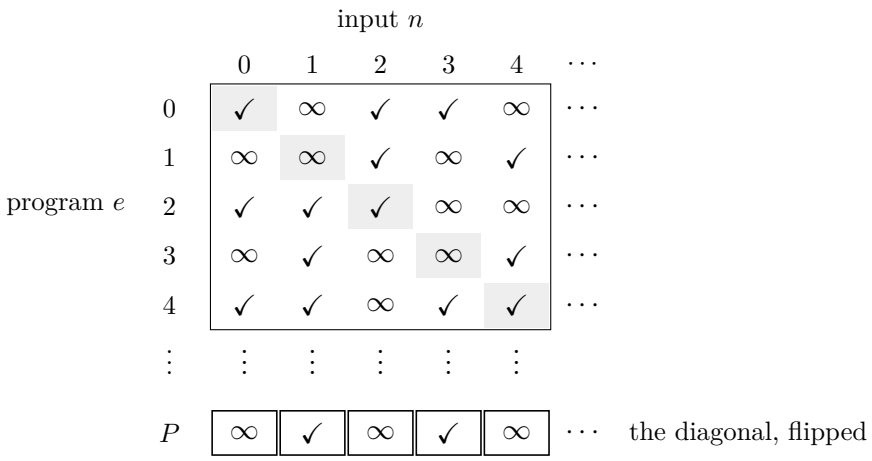


FIGURE 8.1. Halting, as a table.

Computation as a theory. The halting theorem is about combinators. To carry it into logic, we write Chapter 1's rules for a step as sentences. The sentences talk about combinators, so their signature has the constants K and S and a two-place function symbol for application. We write that symbol as application is written, by putting its two arguments side by side and grouping to the left, so that a combinator is also a closed term of the signature. A relation symbol $\mathsf{Step}(x, y)$ will say that x steps to y, a relation symbol $\mathsf{Nf}(x)$ that x is in normal form, and a relation symbol $\mathsf{Halts}(x)$ that x halts. Call this signature Σ_{KS}: the constants K and S, the function symbol for application, the two-place relation symbol Step, and the one-place relation symbols Nf and Halts.

Definition 8.21 (the theory of computation)**.** The theory of computation says, in first-order sentences, how a term steps, which terms are in normal form, and that a term halts when it is in normal form or steps to a term that halts.

Formally, the *theory of computation* Θ_{KS} is the set of the universal closures of the following formulas of Σ_{KS}.

(1) *Steps.*

- $\mathsf{Step}(\mathsf{K}\,x\,y,\ x)$ — the rule (K)
- $\mathsf{Step}(\mathsf{S}\,x\,y\,z,\ x\,z\,(y\,z))$ — the rule (S)
- $\mathsf{Step}(x, x') \to \mathsf{Step}(x\,y,\ x'\,y)$ — a step in the function, (app_1)
- $\mathsf{Step}(y, y') \to \mathsf{Step}(x\,y,\ x\,y')$ — a step in the argument, (app_2)

(2) *Normal forms.*

- $\mathsf{Nf}(\mathsf{K})$ — K alone
- $\mathsf{Nf}(\mathsf{S})$ — S alone
- $\mathsf{Nf}(x) \to \mathsf{Nf}(\mathsf{K}\,x)$ — K with one argument
- $\mathsf{Nf}(x) \to \mathsf{Nf}(\mathsf{S}\,x)$ — S with one argument
- $(\mathsf{Nf}(x) \wedge \mathsf{Nf}(y)) \to \mathsf{Nf}(\mathsf{S}\,x\,y)$ — S with two arguments

(3) *Halting.*

- $\mathsf{Nf}(x) \to \mathsf{Halts}(x)$ a normal form halts
- $(\mathsf{Step}(x, y) \wedge \mathsf{Halts}(y)) \to \mathsf{Halts}(x)$ halting passes back

In words: the first group is Chapter 1's four rules for a step, each bar read as an implication, so that a rule with nothing above the bar is a sentence on its own. The second group is the lemma on the shape of a normal form (Lemma 1.7), for combinators, read from right to left: a term of one of the shapes it lists, with its parts in normal form, is in normal form. The third says that a term halts if it is in normal form already, or if it steps to a term that halts.

For a combinator M, write η_M for the sentence $\mathsf{Halts}(M)$, which says that M halts.

The sentences for normal forms name five shapes. The obvious shortcut, $(\mathsf{Nf}(x) \wedge \mathsf{Nf}(y)) \to \mathsf{Nf}(x\,y)$, is left out, and for a good reason: it would make $\mathsf{K}\,\mathsf{K}\,\mathsf{K}$ normal, since $\mathsf{K}\,\mathsf{K}$ and K are, and $\mathsf{K}\,\mathsf{K}\,\mathsf{K}$ steps to K. Two normal parts make a normal whole only when the whole is not a redex, and the three shapes that are applications, $\mathsf{K}\,x$, $\mathsf{S}\,x$ and $\mathsf{S}\,x\,y$, are exactly the ways of avoiding one.

Two features of the definition are deliberate. The theory has no symbol for "runs to": halting is passed back along a run one step at a time, which keeps every sentence short. And the theory is the same for every program. Only η_M mentions M, because every combinator runs by the same four rules: the program is the term, not a list of instructions.

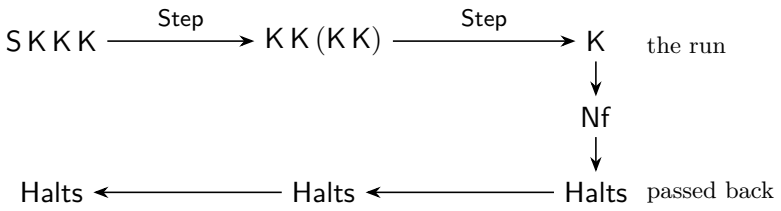
Example 8.22 (the theory at work)**.** The program $\mathsf{I}\,\mathsf{K}$, that is, $\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,\mathsf{K}$, halts in two steps:

$$\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,\mathsf{K} \rightsquigarrow \mathsf{K}\,\mathsf{K}\,(\mathsf{K}\,\mathsf{K}) \rightsquigarrow \mathsf{K}.$$

In any model of Θ_{KS}, its sentences force these lines, one by one:

$\mathsf{Step}(\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,\mathsf{K},\ \mathsf{K}\,\mathsf{K}\,(\mathsf{K}\,\mathsf{K}))$	the S sentence, at K, K, K
$\mathsf{Step}(\mathsf{K}\,\mathsf{K}\,(\mathsf{K}\,\mathsf{K}),\ \mathsf{K})$	the K sentence, at K, $\mathsf{K}\,\mathsf{K}$
$\mathsf{Nf}(\mathsf{K})$	normal forms, the first sentence
$\mathsf{Halts}(\mathsf{K})$	halting, the first sentence
$\mathsf{Halts}(\mathsf{K}\,\mathsf{K}\,(\mathsf{K}\,\mathsf{K}))$	halting, the second, with lines 2 and 4
$\mathsf{Halts}(\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,\mathsf{K})$	halting, the second, with lines 1 and 5.

The last line is $\eta_{\mathsf{I}\,\mathsf{K}}$, so $\Theta_{\mathrm{KS}} \models \eta_{\mathsf{I}\,\mathsf{K}}$. The first two lines are the run, the third recognises its result as a normal form, and the last three pass that result back to its start:



The main lemma of this section says that the sentences describe running in just the sense needed: a program M halts exactly when η_M follows from Θ_{KS}.

Lemma 8.23 (a program is a theory)**.** *For every combinator M, the combinator M halts if and only if $\Theta_{\mathrm{KS}} \models \eta_M$.*

In words: a program halts exactly when the sentence "it halts" follows from the sentences that describe computation.

Proof. The two directions use different structures.

- *If M halts, η_M follows.* Let $\mathfrak{A}$ be a model of Θ_{KS}. Three facts about $\mathfrak{A}$ give the result.
  (1) If $P \rightsquigarrow Q$ then $\mathfrak{A} \models \mathsf{Step}(P, Q)$, by induction on the rules that stack up to the step. A step by the rule (K) or (S) is an instance of the first or the second sentence for steps. A step by (app_1), from $P_1 \rightsquigarrow P_1'$ to $P_1\,P_2 \rightsquigarrow P_1'\,P_2$, follows by the third sentence from $\mathfrak{A} \models \mathsf{Step}(P_1, P_1')$, which the induction hypothesis gives; (app_2) is the same with the fourth sentence.
  (2) If N is a combinator in normal form then $\mathfrak{A} \models \mathsf{Nf}(N)$, by induction on N. By the lemma on the shape of a normal form, N is K, $\mathsf{K}\,N_1$, S, $\mathsf{S}\,N_1$ or $\mathsf{S}\,N_1\,N_2$, with N_1 and N_2 in normal form; the induction hypothesis gives $\mathsf{Nf}(N_1)$ and $\mathsf{Nf}(N_2)$, and the sentence for that shape gives $\mathsf{Nf}(N)$.
  (3) Since M halts, there is a run $M = M_0 \rightsquigarrow M_1 \rightsquigarrow \cdots \rightsquigarrow M_n$ that ends in a normal form M_n. Then $\mathfrak{A} \models \mathsf{Halts}(M_n)$, by the second fact and the first sentence for halting. And if $\mathfrak{A} \models \mathsf{Halts}(M_{i+1})$, then $\mathfrak{A} \models \mathsf{Halts}(M_i)$, by the first fact and the second sentence for halting. Going back along the run from $i = n-1$ to $i = 0$ gives $\mathfrak{A} \models \mathsf{Halts}(M)$, which is η_M.

  As $\mathfrak{A}$ was an arbitrary model of Θ_{KS}, $\Theta_{\mathrm{KS}} \models \eta_M$.
- *If M does not halt, η_M does not follow.* Let $\mathfrak{T}$ be the structure whose domain is the set of combinators, in which K and S mean themselves, application means application, and

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

  Every closed term means itself in $\mathfrak{T}$. Each sentence of Θ_{KS} is true in $\mathfrak{T}$.
  - *Steps.* They are the four rules for a step.
  - *Normal forms.* They hold by the lemma on the shape of a normal form, read from right to left.
  - *Halting.* A combinator in normal form halts, by the run of no steps. If $P \rightsquigarrow Q$ and Q halts, then P halts, by the run that takes that step first.

  Since M does not halt, η_M is false in $\mathfrak{T}$. So $\Theta_{\mathrm{KS}} \not\models \eta_M$. □

Since Θ_{KS} is a set of sentences, the soundness and completeness of NK let us read the lemma with $\Theta_{\mathrm{KS}} \vdash_{\mathrm{NK}} \eta_M$ in place of $\Theta_{\mathrm{KS}} \models \eta_M$: a program halts exactly when a derivation exists.

We call the structure $\mathfrak{T}$ of the second half the term structure. Its domain is the set of closed terms of Σ_{KS} and each of them means itself, as in the term structure of Chapter 6 (Definition 6.64), but its relations are chosen by what the combinators do, not by a set of sentences. It is a Herbrand structure, in the sense of Chapter 7, and it satisfies Θ_{KS}.

The first half of the proof shows that every model of Θ_{KS} makes Halts true of each combinator that halts. It shows nothing about the other combinators, and such a structure may well be more generous: Exercise 8.8 has one in which every combinator halts. The term structure is the least generous, since it makes Halts true of the combinators that halt and of nothing else. That is why the second half uses it.

Example 8.24 (the Monster as a theory)**.** The Monster Ω does not halt: it has no normal form (Chapter 1). So $\Theta_{\mathrm{KS}} \not\models \eta_\Omega$: the term structure is a model of Θ_{KS} and makes η_Ω false. No derivation in NK leads from Θ_{KS} to $\mathsf{Halts}(\Omega)$.

The combinator $\mathsf{K}\,\mathsf{K}\,\Omega$ contains the Monster and still halts. Three lines show that $\Theta_{\mathrm{KS}} \models \eta_{\mathsf{K}\,\mathsf{K}\,\Omega}$: $\mathsf{Step}(\mathsf{K}\,\mathsf{K}\,\Omega,\ \mathsf{K})$, by the K sentence at K and Ω; $\mathsf{Halts}(\mathsf{K})$, as in the example of the theory at work; and $\mathsf{Halts}(\mathsf{K}\,\mathsf{K}\,\Omega)$, by the second sentence for halting. The sentences never look inside Ω.

When M halts, the facts that establish η_M follow its run step by step, and each of them is forced: it is the one literal left in an instance of a sentence of Θ_{KS} once the others are established (Example 8.22 lists them for $\mathsf{I}\,\mathsf{K}$). So a procedure that sets forced literals, as `DPLL` does, can find them by itself. The sentences of Θ_{KS}, and the negation of η_M, each have a clause form of one clause: drop the quantifiers and write each implication as a disjunction. Every one of these clauses has at most one positive literal: they are Horn clauses, the kind a logic program is made of (Remark 8.25). Give $\Theta_{\mathrm{KS}} \cup \{\neg\eta_M\}$ to the first-order procedure `REFUTE` (Theorem 7.63): by that theorem and the lemma that a program is a theory, it returns `UNSAT` exactly when M halts, and on the Monster it runs for ever.

Remark 8.25 (at work: a logic program). The sentences of Θ_{KS} are, almost word for word, a program in Prolog, the logic programming language, with `app` for application:

```
step(app(app(k,X),Y), X).
step(app(app(app(s,X),Y),Z), app(app(X,Z),app(Y,Z))).
step(app(X,Y), app(X1,Y)) :- step(X,X1).
step(app(X,Y), app(X,Y1)) :- step(Y,Y1).
nf(k).  nf(s).  nf(app(k,X)) :- nf(X).  nf(app(s,X)) :- nf(X).
nf(app(app(s,X),Y)) :- nf(X), nf(Y).
halts(X) :- nf(X).
halts(X) :- step(X,Y), halts(Y).
```

`REFUTE` itself is slow on these clauses. For $\mathsf{I}\,\mathsf{K}$, that is $\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,\mathsf{K}$, it must reach level 3, because establishing $\mathsf{Halts}(\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,\mathsf{K})$ uses the second sentence for halting at $\mathsf{S}\,\mathsf{K}\,\mathsf{K}\,\mathsf{K}$ itself, a term of level 3. There are 1446 closed terms of level at most 3, and the S sentence alone has more than three thousand million

ground instances at that level. The levels grow fast because terms are trees. `REFUTE-U`, which chooses the instances by unification (Proposition 7.98), reaches the useful ones sooner: it closes in its second round, at level 1, though even there it grounds about 186,000 clauses.[11]

Prolog answers a query by choosing instances by unification, much as `REFUTE-U` does, and it tries the clauses in the order written. In that order the first step it finds always takes the leftmost of the outermost redexes, so it answers `true` to `halts(M)` exactly when M halts, and on the Monster it never answers. A logic program of eight lines is an interpreter for the combinators.

8.4. **Undecidability of First-Order Logic.** The lemma that a program is a theory turns a question about a program into a question about a sentence, by a map that a program can compute. That is all it takes to move the undecidability of halting from programs to logic, and so to settle the first of the chapter's two questions, the search gap.

Theorem 8.26 (validity is undecidable)**.** *No program decides, for every sentence of the signature Σ_{KS}, whether it is valid, that is, true in every Σ_{KS}-structure.*

Proof. Let θ_{KS} be the conjunction of the finitely many sentences of Θ_{KS}. Given numbers e and n, let $M = \mathrm{prog}(e)$. The sentence $\theta_{\mathrm{KS}} \to \eta_{M\,\underline{n}}$ is valid exactly when $\Theta_{\mathrm{KS}} \models \eta_{M\,\underline{n}}$, and so, by the lemma that a program is a theory, exactly when M halts on n. A program can write the sentence down, given e and n: the conjunction θ_{KS} is the same every time, and $\eta_{M\,\underline{n}}$ is $\mathsf{Halts}(M\,\underline{n})$. If a program decided validity, then running it on $\theta_{\mathrm{KS}} \to \eta_{M\,\underline{n}}$ would decide whether $\mathrm{prog}(e)$ halts on n, which no program does. □

In words: no algorithm decides first-order logic, not even for the sentences that say something as plain as "this program stops". The signature is small, and a larger one does not help: a program for a larger signature would decide, in particular, the sentences of Σ_{KS}, whose validity does not depend on symbols they do not mention.

The proof has a shape that fits any question about programs. To show that no program answers some question, show how a program could turn every question "does $\mathrm{prog}(e)$ halt on n?" into a question of the new kind with the same answer. A program for the new question would then answer every halting question, and no program does. Such a map is called a *reduction*: the one here sent (e, n) to $\theta_{\mathrm{KS}} \to \eta_{M\,\underline{n}}$, where $M = \mathrm{prog}(e)$. The hard part of each such proof is always the same, checking that the answers agree, and here it was the lemma that a program is a theory.

The same reduction gives the other forms of the question at once.

Corollary 8.27 (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 $\Gamma \models \varphi$, or equivalently whether $\Gamma \vdash_{\mathrm{NK}} \varphi$;

[11] The counts are from the scripts `ks_checks.py` and `ks_refute.py` in the course's support folder.

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

Proof. Each part is a reduction from halting. Given numbers e and n, let $M = \mathrm{prog}(e)$. In each case a program computes from e and n an input whose answer says whether M halts on n, and a program that answered the question would therefore decide halting.

- *(i).* Take $\Gamma = \Theta_{\mathrm{KS}}$ and $\varphi = \eta_{M\,\underline{n}}$, by the lemma that a program is a theory. The second form has the same answers as the first, by the soundness and completeness of NK.
- *(ii).* The set $\Theta_{\mathrm{KS}} \cup \{\neg\eta_{M\,\underline{n}}\}$ is unsatisfiable exactly when $\Theta_{\mathrm{KS}} \models \eta_{M\,\underline{n}}$, that is, exactly when M halts on n. So the answer "satisfiable" means "does not halt".
- *(iii).* The clause form of that set, one clause for each of its sentences, is unsatisfiable exactly when the set is. □

In (i) the set Γ never changes: the proof uses the same Θ_{KS} for every program. So there is one finite set of sentences whose consequences no program decides.

This settles the search gap. The first-order procedure certifies every entailment that holds, and may never answer for one that fails; the search gap asked whether a cleverer search could always stop, and part (iii) of the corollary says that none can.

The procedure `REFUTE` stops on every unsatisfiable clause set (Theorem 7.63), and so does `REFUTE-U`, which chooses its instances by unification (Proposition 7.98, from the optional Section 7.8; only its statement is used here). A procedure that also stopped on every satisfiable set would decide satisfiability, and there is no such procedure. Whatever a cleverer search gains, and unification gains a great deal, it cannot gain the other half of the answer. When the procedure runs for ever on a satisfiable set, that is not a defect of the procedure: no procedure could do better.

The valid sentences therefore form a set of a particular kind: one that has a semi-decision procedure (Definition 7.64), and so can be listed, but cannot be decided.[12] They can be *listed*: negate every sentence, put each negation in clause form and run `REFUTE` on all of them at once, interleaving the runs; then print each sentence whose run closes. Every valid sentence is printed sooner or later. But they cannot be *decided*, by the theorem that validity is undecidable. The listing shows where the difficulty lies: a sentence that is not valid is never printed, and no amount of waiting tells it apart from one that has not been printed yet. Membership can be confirmed, and non-membership in general cannot.

Remark 8.28 (what remains decidable). None of this makes logic useless in practice. The theorem is precise about what it rules out. Many questions can still be decided.

- Propositional logic is decided by `DPLL`.

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

- Whether a finite set of sentences is satisfied by a structure with n elements, for a given n, is a propositional question, settled by a SAT solver.[13]
- Many restricted languages have decision procedures of their own, and some of them close this chapter.

What no method does is decide every first-order question. The tools in use make one of three honest 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.

8.5. **Arithmetic as a Theory.** The structure that the founders of modern logic most wanted to capture is the oldest one in mathematics: the counting numbers. We want to know whether some specification captures them. By the theorem that capturing makes truth decidable, it is enough to show that no program decides their truths, and the reduction that worked for validity works again, once halting can be expressed as a sentence about numbers. So what is left is to fix the language, and to express halting in it. 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 8.29 (the signature of arithmetic, with equality)**.** In this chapter the signature of arithmetic contains $=$, and the counting numbers interpret it as identity.

Formally, Σ_{A} has the constant symbol 0, the one-place function symbol s, the two-place function symbols $+$ and $\times$, and the two-place relation symbols $<$ and $=$. 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 8.30 (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.

The structure cannot be written out as a table, since its domain is infinite: it can only be described by a theory.

[13] The encoding has one propositional letter for each possible entry of each table, and clauses saying that each entry has exactly one value and that the sentences hold.

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.

The obvious candidate. Every course on arithmetic starts from the axioms of Peano arithmetic.

Definition 8.31 (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 be applied to.

A program recognises an induction axiom by its shape, so PA is a specification, and $\mathfrak{N} \models \mathrm{PA}$. It proves a great deal: very nearly every theorem of elementary number theory that has ever been proved can be derived in it. The question is whether it, or anything like it, proves every true sentence.

Saying things about programs. To express halting we need to say what it is for a formula to express a relation between numbers. The notion is the plainest one there is.

Definition 8.32 (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 $n_1, \ldots, n_k$,

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

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 8.7 needs two facts about what PA proves, and states them.

From now on a program is referred to by its code, and its input is a number. In this form halting is undecidable, by the theorem of Section 8.3: no program decides, given the numbers e and n, whether $\mathrm{prog}(e)$ halts on n.

Lemma 8.33 (halting is definable)**.** *There is a formula $H(x, y)$ of Σ_{A} such that, for all numbers e and n,* $\mathrm{prog}(e)$ *halts on n if and only if*

$$\mathfrak{N} \models H(\overline{e}, \overline{n}).$$

The idea of the proof. The proof is in the optional Section 8.8, and here is how it goes. A halting run is finite: a list of combinators $M_0 \rightsquigarrow M_1 \rightsquigarrow \cdots \rightsquigarrow M_N$, where M_0 is the program applied to the numeral of its input and M_N is in normal form. Each combinator is a string of the four symbols `K`, `S`, (and), and so, read as in its code, a list of the digits 1 to 4. So a halting run is a finite table of numbers, one row for each combinator.

The one idea of the proof 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 $1 + (i + 1)\, b$. (A run needs a few such pairs, for the lengths of the rows as well as their digits; the sketch speaks of one.) A remainder is expressed by $+$ and $\times$: "r is the remainder of a on division by m" is $\exists q\, (a = q \times m + r \wedge r < m)$.

So $H(x, y)$ can say: *there are numbers N, a and b whose table has, in its first row, the string of the program with code x applied to the numeral of y; in each later row, the string of a combinator one step from the row before; and, in its last row, a normal form.* One step replaces a stretch of the form $((\mathtt{K}\, P)\, Q)$ by P, or one of the form $(((\mathtt{S}\, P)\, Q)\, R)$ by $((P\, R)(Q\, R))$, and a normal form has no such stretch. Each of those conditions talks only about digits at positions in the table, that is, about remainders, that is, about $+$ and $\times$.

Example 8.34 (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 8.8 proves that every finite table can be packed in this way.

Since halting is definable, the undecidability of halting moves from programs to the counting numbers by a reduction, as it moved to validity.

Theorem 8.35 (the truths of arithmetic are undecidable)**.** *No program decides, for every sentence φ of Σ_{A}, whether $\mathfrak{N} \models \varphi$.*

Proof. The program $\mathrm{prog}(e)$ halts on n exactly when $\mathfrak{N} \models H(\overline{e}, \overline{n})$, and a program can write that sentence down, given e and n. A program that decided truth in $\mathfrak{N}$ would therefore decide halting, which no program does. □

Corollary 8.36 (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 the theorem that capturing makes truth decidable. But no program decides them, by the theorem that the truths of arithmetic are undecidable. 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: "take all the true ones" gives no checker. And PA, in particular, misses some truth.

The contrast with school algebra is now exact. The identities true of the positive integers, with 1, $+$, $\cdot$ and powers, can be decided, so they form a specification. Only the finite lists of them fail. For arithmetic, no list a program can read will do, and that is a theorem.

8.6. **First Incompleteness Theorem.** The corollary says that a specification of arithmetic misses a truth, and gives no idea which. We want to name one. The specification is assumed to tell the truth, and the sentence will come from a program that watches the specification at work.

Definition 8.37 (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. It should not 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 $\mathfrak{N} \models \Theta$. In particular, PA is sound, and so is every specification obtained from it by adding true sentences.

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

Definition 8.38 (the diagonal program)**.** Let Θ be a specification in Σ_{A}. The *diagonal program* D_Θ of Θ is the following program. It uses the program `LIST` of Lemma 8.7, which lists the theorems of Θ, and the formula $H(x, y)$ of Lemma 8.33, which is true in $\mathfrak{N}$ of $\overline{e}$ and $\overline{n}$ exactly when $\mathrm{prog}(e)$ halts on n.

```
procedure D(e)                          -- e a number
    for each theorem phi of Theta, as LIST prints them:
        if phi is  not H(e, e),  e written as a numeral:  halt
```

Its translation, given by the fact that programs are combinators, halts on exactly the inputs on which D_Θ halts. We write d_Θ, or d when Θ is clear, for the code of the translation.

In words: on the input e, the program waits for Θ to prove that $\mathrm{prog}(e)$ does not halt on e, and halts if that ever happens. The interesting input is the program's own code.

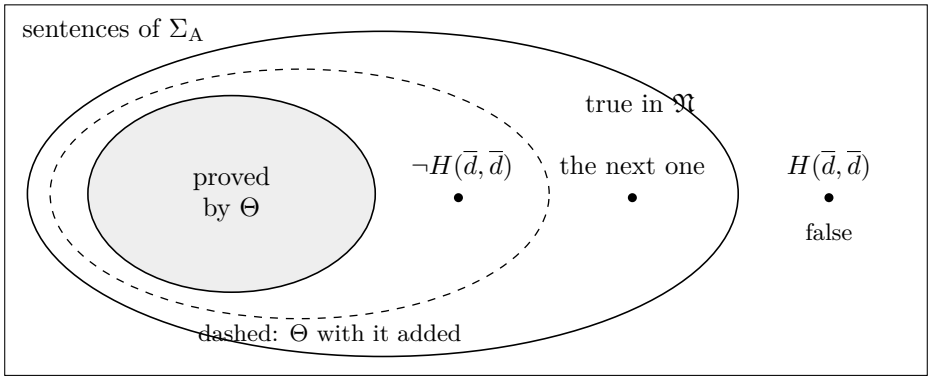


FIGURE 8.2. A sound specification and the truths.

Theorem 8.39 (the first incompleteness theorem)**.** *If Θ is a sound specification in Σ_{A} and d is d_Θ, the code of the translation of its diagonal program, then*

$$\mathfrak{N} \models \neg H(\overline{d},\overline{d}) \qquad \textit{and} \qquad \Theta \nvdash_{\mathrm{NK}} \neg H(\overline{d},\overline{d}).$$

Proof. Write D for the diagonal program of Θ. The program $\mathrm{prog}(d)$ is the translation of D, and halts on exactly the inputs on which D halts, so the definability lemma says: D halts on d if and only if $\mathfrak{N} \models H(\overline{d},\overline{d})$. Consider the run of D on its own code d. The three steps below show, in turn, that the run never ends, that the sentence is therefore true, and that Θ does not prove it.

(1) *D does not halt on d.* If it did, the listing of Θ would have printed $\neg H(\overline{d},\overline{d})$, so Θ would prove it, and as Θ is sound the sentence would be true. By the definability lemma, D would then not halt on d, contrary to the assumption.

(2) *So $\mathfrak{N} \models \neg H(\overline{d},\overline{d})$,* by the definability lemma again, since D does not halt on d.

(3) *And Θ does not prove it,* because if it did, the listing would print it, and D would halt on d, which by the first step it does not. □

The proof used three things: that Θ is sound, that its theorems can be listed, and that halting is definable. It used no particular axioms of arithmetic, and no fact about what any specification proves, beyond what Θ's own soundness gives.

Figure 8.2 draws the theorem. Among the sentences of Σ_{A}, the theorems of a sound Θ lie inside the truths. The sentence $\neg H(\overline{d},\overline{d})$ lies in the gap between them, and its negation is false, so Θ proves neither. Adding the sentence 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 an example shows each.

- *A specification.* The set $\mathrm{Th}(\mathfrak{N})$ proves exactly the truths, so it leaves no truth unproved. No program lists it, since a listing would decide truth in $\mathfrak{N}$ by waiting for φ or $\neg\varphi$, and no program decides truth in $\mathfrak{N}$. So it is not a specification, and its diagonal program cannot be written.
- *Sound.* A specification that contains a sentence and its negation proves every sentence, by ($\to$E) and ($\bot$E), and so it proves $\neg H(\overline{d}, \overline{d})$. It is a specification, and not sound.

Remark 8.40 (what the theorem shows, and what it does not). The theorem is often quoted, and often quoted wrongly, so it is worth being exact.

- *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. The sentence $\neg H(\overline{d}, \overline{d})$ was just proved true, here, by an argument that uses the soundness of Θ, which Θ cannot supply for itself.

Remark 8.41 (non-standard models). The theorem can also be read through structures. The sentence $\neg H(\overline{d}, \overline{d})$ 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. Since Θ is sound, its sentences are all true in $\mathfrak{N}$; a model of a set of truths about $\mathfrak{N}$ that is not a copy of $\mathfrak{N}$ is called a *non-standard model*.

The disagreement has to live somewhere. In $\mathfrak{M}$ the sentence $H(\overline{d}, \overline{d})$ 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 model, namely $\mathfrak{N}$ with c read as a large enough number, so by compactness (Remark 6.66) 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 $\mathfrak{N}$. No set of first-order sentences pins the counting numbers down.[14]

Remark 8.42 (the sentence speaks about itself). Read $\neg H(\overline{d}, \overline{d})$ in $\mathfrak{N}$: it says that D, run on d, never halts. And D, run on d, halts exactly when Θ proves $\neg H(\overline{d}, \overline{d})$. So the sentence 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

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

sentence that says of itself that it is not provable. Here the self-reference costs nothing: it is D run on its own code. The History says more.

Remark 8.43 (the diagonal). Every undecidability result of this chapter came from the same move, 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 of the halting theorem did this to a predictor of halting, and the program D does it to a list of theorems.

The move is older than computers. It is the Monster of the first part of the course, the function that applies its argument to itself and then runs for ever. And the Monster turned out to be Richard's paradox. The History collects four of its appearances in a table.

8.7. **Second Incompleteness Theorem.** The first incompleteness theorem measured a specification against the structure we meant: some truth about $\mathfrak{N}$ does not follow from it. This section finds a limit of another kind, which does not mention $\mathfrak{N}$ at all. Hilbert's programme, the plan to secure mathematics by formal axioms, asked that the axioms should never contradict themselves, a property of derivations and not of truth, and that arithmetic should prove this about itself. We want to know whether a specification can. It cannot: a consistent specification that contains PA does not prove the sentence that says it is consistent, and so, by the completeness of NK, that sentence does not 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. Then we carry the proof of the first theorem out inside PA, which is one more fact.

Consistency is the property of Definition 6.53: 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 8.51).

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

Fact 8.44 (true halting is provable). For all numbers e and n: if $\mathrm{prog}(e)$ halts on n, then $\mathrm{PA} \vdash_{\mathrm{NK}} H(\overline{e}, \overline{n})$.

The reason is that a halting run is a finite calculation. If the program halts, there are particular numbers N, a and b that pack its run, and $H(\overline{e}, \overline{n})$ follows by ($\exists$I) from the claim that these numbers do so. That claim comes down to a check of each of finitely many rows, and each check to facts about particular numbers, such as $\overline{80} = \overline{26} \times \overline{3} + \overline{2}$. A check is finite because every "for all" in H can be written to range only over the numbers below a

given one. 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.[15]

Theorem 8.45 (the first incompleteness theorem, from consistency)**.** *If* Θ *is a consistent specification in* Σ_{A} *with* $\mathrm{PA} \subseteq \Theta$*, and* d *is* d_Θ*, the code of the translation of its diagonal program, then*

$$\mathfrak{N} \models \neg H(\overline{d}, \overline{d}) \qquad \textit{and} \qquad \Theta \nvdash_{\mathrm{NK}} \neg H(\overline{d}, \overline{d}).$$

In words: for a specification that contains PA, consistency is enough in place of soundness.

Proof. Write D for the diagonal program of Θ. The program $\mathrm{prog}(d)$ is the translation of D, and halts on exactly the inputs on which D halts, so both the definability lemma and the fact that true halting is provable may be applied to D and d. We show first that Θ does not prove the sentence, and then that it is true.

(1) Θ *does not prove* $\neg H(\overline{d}, \overline{d})$. Suppose it did. Then the listing of Θ prints the sentence, and D halts on d. By the fact that true halting is provable, PA proves $H(\overline{d}, \overline{d})$, and so does Θ, which contains PA. So Θ proves a sentence and its negation, and therefore $\bot$, which contradicts its consistency.

(2) *The sentence is true.* The program D halts on d only if the listing prints $\neg H(\overline{d}, \overline{d})$, and by the first step it never does. So D does not halt on d, and by the definability lemma $\mathfrak{N} \models \neg H(\overline{d}, \overline{d})$. □

The proof shows the price of the weaker assumption. A consistent specification that is not sound might prove the false sentence $H(\overline{d}, \overline{d})$, and nothing in the argument forbids it. So the theorem cannot say that Θ leaves the question open either way.

Remark 8.46 (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 at once, and acts on whichever comes first. The argument is a little longer, and the main line does without it.[16]

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

Definition 8.47 (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
```

[15] The fact is called the Σ_1-completeness of arithmetic. It applies to H written with every universal quantifier bounded, as it can be: the optional Section 8.8 shows how, and the main line uses only that it can be done. The fact then holds even for a fragment of PA without induction, and is proved in Open Logic Project (2026, §35.11).

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

and let c be the code of its translation. The *consistency sentence* of Θ is

$$\mathrm{Con}_\Theta := \neg H(\overline{c}, \overline{0}).$$

The specification Θ is consistent exactly when C_Θ never halts, whatever its input, and so, by the definability lemma, exactly when $\mathfrak{N} \models \mathrm{Con}_\Theta$. In words, Con_Θ says "Θ is consistent" in the language of arithmetic. The programme hoped for a derivation of such a sentence from axioms everyone accepts.

The definition asks for the translation of C_Θ, and not for 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 code in place of c the sentence would be provable. The sentence must speak of the program that actually searches the theorems of Θ.

The second incompleteness theorem needs one more fact on loan, and of the facts the chapter borrows it is the longest to check. The proof of the first incompleteness theorem from consistency is an argument about numbers and programs. In particular it shows that if Θ is consistent, then the diagonal program of Θ does not halt on its own code d. That argument can be carried out inside PA.

Fact 8.48 (arithmetic knows the first theorem)*.* If Θ is a specification in Σ_A with $\mathrm{PA} \subseteq \Theta$, and d is d_Θ, the code of the translation of its diagonal program, then

$$\mathrm{PA} \vdash_\mathrm{NK} \mathrm{Con}_\Theta \to \neg H(\overline{d}, \overline{d}).$$

Written with Con_Θ in place of "Θ is consistent", the argument uses only the definitions of the programs D_Θ and C_Θ, the fact that true halting is provable, and the step from a sentence and its negation to $\bot$. What has to be checked is that PA can prove each of these about the formula H. That check is long and is omitted by most textbooks.[17]

Theorem 8.49 (the second incompleteness theorem)**.** *If Θ is a consistent specification in Σ_A with $\mathrm{PA} \subseteq \Theta$, then $\Theta \nvdash_\mathrm{NK} \mathrm{Con}_\Theta$.*

In words: a consistent specification of arithmetic cannot prove its own consistency.

Proof. Let d be d_Θ, and suppose $\Theta \vdash_\mathrm{NK} \mathrm{Con}_\Theta$. Since $\mathrm{PA} \subseteq \Theta$, the fact that arithmetic knows the first theorem gives $\Theta \vdash_\mathrm{NK} \mathrm{Con}_\Theta \to \neg H(\overline{d}, \overline{d})$, and ($\to$E) gives $\Theta \vdash_\mathrm{NK} \neg H(\overline{d}, \overline{d})$. That contradicts the first incompleteness theorem from consistency, since Θ is a consistent specification with $\mathrm{PA} \subseteq \Theta$. So Θ does not prove Con_Θ. □

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 $\mathfrak{N}$.

[17] The properties of provability involved are called the *derivability conditions*; Open Logic Project (2026, §37.6) states them and also takes them on faith.

Remark 8.50 (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 in which it was 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. The consistency of PA is not in doubt: $\mathfrak{N}$ satisfies it. But that certificate is the structure itself, and the axioms cannot issue it for themselves.

Remark 8.51 (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 believe something false about the numbers, and even something false about itself.

8.8. **Defining Halting** *(optional).* This section is optional. It proves the lemma that halting is definable in $\mathfrak{N}$, which the main line used with only a sketch of its proof, and it shows that the formula can be written with every universal quantifier bounded, the form the fact that true halting is provable needs. The main line uses these results, not their proofs.

The proof has two parts. The first is a fact about numbers: a finite sequence of numbers can be packed into two numbers, from which each entry is recovered as a remainder. The second lays the run out as a table of strings, one for each combinator of the run, and uses the packing to write a formula that says "there is a halting run".

Each part comes down to one picture. The first: the list $2, 0, 3$ is stored in the single number 80, as its remainders on division by 3, 5 and 7 ($80 = 26\times 3+2 = 16\times 5+0 = 11\times 7+3$), and any finite list can be stored in this way. The second: a step of a run copies most of one row of the table into the next, and changes one stretch of it (Figure 8.3). A reader who wants only the idea of the proof can stop here; the rest of the section checks that both pictures can be written as formulas of arithmetic.

Packing a sequence. 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 8.52 (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.*

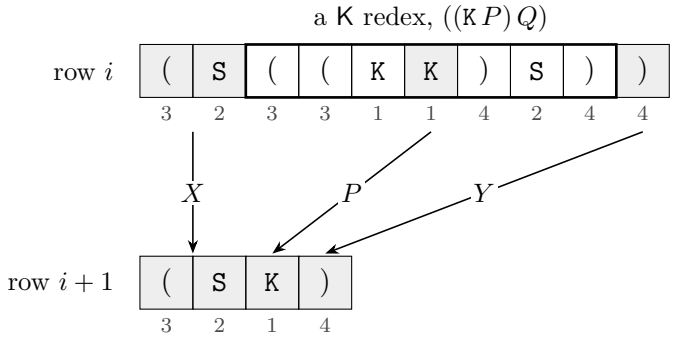


FIGURE 8.3. A step is the replacement of a stretch.

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 8.53 (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: any 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 $1 \le l - i \le n \le j$, 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 8.54 (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 formula. 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.

These abbreviations 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, H has unbounded quantifiers only in its leading block of $\exists$. That is the form needed by the fact that true halting is provable.

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 $\forall i \le N$ and $\exists i \le N$ in the same way; and $\exists p, q \le N \, \varphi$ for $\exists p \le N \, \exists q \le N \, \varphi$. In a sum such as $p + 3$, the number 3 stands for its numeral $\overline{3}$.

A run as a table. 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 $i \le N$, of:

- $\mathit{Len}(i) \le W$;
- 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.

Stretches. 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:

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

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:

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

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

Steps. A step rewrites one part of a combinator that is a redex, and leaves the rest alone. In the string, that is the replacement of 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 their lengths, and p for the position where the redex starts, which is ℓ_X. A `K` redex at p is then described by

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

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

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

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 8.16 writes the first and lays out the second.

Let $\mathit{Next}(i)$ be $\mathit{Next}_{\mathsf{K}}(i) \vee \mathit{Next}_{\mathsf{S}}(i)$, and let $\mathit{Normal}(i)$ say that row i has no redex:

$$\begin{aligned} \mathit{Normal}(i) \coloneqq \neg\exists p, \ell_P, \ell_Q \le \mathit{Len}(i)\ &\mathit{Red}_{\mathsf{K}}(i, p, \ell_P, \ell_Q) \\ &\wedge \neg\exists p, \ell_P, \ell_Q, \ell_R \le \mathit{Len}(i)\ \mathit{Red}_{\mathsf{S}}(i, p, \ell_P, \ell_Q, \ell_R). \end{aligned}$$

Lemma 8.55 (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. □

The first row. The first row is the string of $M\,\underline{n}$, where $M = \mathrm{prog}(e)$. It is an opening bracket, the digits of e in base 5, the string of the numeral $\underline{n}$, and a closing bracket. By the definition of the numerals, the string of $\underline{n}$ is

$$U^n\ Z\ )^n, \qquad \text{where} \quad U = (\texttt{(S((S(KS))K))} \quad \text{and} \quad Z = (\texttt{K((SK)K)}),$$

that is, n copies of the fourteen symbols of U, the ten of Z, and n closing brackets: $15n + 10$ symbols in all. The digits of e 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 e spell in base 5. Let $\mathit{Start}(x, y)$ be $\exists L \le \mathit{Len}(0)$ of the conjunction of:

- $\mathit{Wf}(0)$, $\mathit{Len}(0) = L{+}\overline{15}{\times}y{+}\overline{12}$, $\mathit{Dg}(0,0) = \overline{3}$ and $\mathit{Dg}(0, L{+}\overline{15}{\times}y{+}\overline{11}) = \overline{4}$;
- $\mathit{Val}(0) = 0$, $\mathit{Val}(L) = x$, and $\mathit{Val}(s(j)) = \overline{5} \times \mathit{Val}(j) + \mathit{Dg}(0, s(j))$ for every $j < L$;
- $T(0, 1, s(L))$;
- for each $r < 14$, $\forall q < y\ \mathit{Dg}(0, L + 1 + \overline{14} \times q + \overline{r}) = \overline{u_r}$, where u_r is the digit of the symbol at position r of U, counting from 0;
- for each $r < 10$, $\mathit{Dg}(0, L + 1 + \overline{14} \times y + \overline{r}) = \overline{z_r}$, where z_r is the digit of the symbol at position r of Z;
- $\forall q < y\ \mathit{Dg}(0, L + \overline{11} + \overline{14} \times y + q) = \overline{4}$.

The first item fixes the outer brackets and asks the row to be the string of a combinator; the second and third say that positions 1 to L hold a term stretch whose digits spell x; the last three lay out the numeral.

The formula. Finally let $H(x, y)$ be

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

In words: $H(x, y)$ says that there is a table of $N + 1$ rows whose first row is $\mathrm{prog}(x)$ applied to the numeral of y, whose later rows are each one step from the row before, and whose last row is in normal form.

Proof of the lemma that halting is definable. Let e and n be numbers.

- *If* $\mathrm{prog}(e)$ *halts on* n*, the formula is true.* Then e is the code of a combinator M, since the program of every other number is $\mathsf{K}\,\Omega$, which halts on no input. Take a run $M\,\underline{n} = M_0 \rightsquigarrow \cdots \rightsquigarrow M_N$ that ends in a normal form. 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 e by the packing lemma, in the orders stated. The resulting numbers make every clause true:
  - *Rows*, by the choice of the sequences;
  - *Start*, by the string of $M\,\underline{n}$;
  - each $\mathit{Next}(i)$, and $\mathit{Normal}(N)$, by the lemma that a step is the replacement of a stretch.
- *If the formula is true,* $\mathrm{prog}(e)$ *halts on* n*.* Take numbers that make the clauses true for $\overline{e}$ and $\overline{n}$. By *Start* and *Rows*, row 0 is the string of a combinator: an application whose function has the string spelled by the digits of e, and whose argument is $\underline{n}$. So e is the code of a combinator M, and row 0 is $\mathrm{str}(M\,\underline{n})$. By induction on $i \le N$, row i is the string of a combinator M_i with $M\,\underline{n} \rightsquigarrow^* M_i$: for the step from i to $i + 1$, the clause $\mathit{Next}(i)$ says 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 $M\,\underline{n}$ halts, which is what was to be shown. □

The definitions of this section can be tested against the combinators themselves, and have been.[18]

8.9. **Decidable Theories** *(optional).* This section is optional: nothing later depends on it. It asks which structures, other than finite ones, can be captured, and proves one case in full.

By the theorem that capturing makes truth decidable and its converse, a structure can be captured exactly when its truths can be decided. So the question becomes a search for decision procedures. The main tool for finding them is the following.

[18] 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; and that the numeral of n has the string $U^n Z\,)^n$.

Definition 8.56 (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 rationals with their order are an infinite structure that can be captured, by a short list of the properties of their order.

Definition 8.57 (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 8.58 (a dense order is captured)**.** *The specification* DLO *admits quantifier elimination, and it captures* $(\mathbb{Q}, <)$.

In words: an infinite structure can sometimes be captured by a handful of sentences. 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 8.59 (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 helpful 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.[19] The definable sets of numbers are then easy to describe: each is, beyond some point, periodic, like the even numbers.

Remark 8.60 (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.[20] 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 8.61 (why multiplication is the dangerous addition). The three infinite structures of this section have one thing in common. In none of them 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 definability lemma 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.

Figure 8.4 summarises the chapter's answers.

[19] 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).

[20] 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.

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.[21] 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.*[22]

Von Neumann, who had heard the first incompleteness theorem at Königsberg, found the second independently and wrote to Gödel on 20 November 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.[23]

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 8.42, reached by coding instead

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

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

[23] Gödel (1931).

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 8.46.[24]

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.[25]

Computation came next. In 1936 Church showed that no effectively calculable function decides whether a term of his λ-calculus has a normal form (Exercise 8.10 asks the same of the combinators, its cousin). 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.[26] 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.[27]

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 8.3, 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.[28]

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 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.[29]

The decidable cases came first. Presburger showed in 1929 that the counting numbers with addition alone can be captured and decided. By 1930

[24]Rosser (1936).

[25]Gentzen (1936).

[26]Church (1936a,b).

[27]Turing (1937a), Turing (1937b).

[28]Hamkins and Nenu (2025); Kleene (1952), p. 382; Davis (1958), p. 70.

[29]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).

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.[30] 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.[31]

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.[32] Wilkie found his identity in 1981, and the paper describing it was published in 2000.[33] 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.[34] Smaller counter-models followed, down to the twelve-element ones of Burris and Yeats, who conjectured that twelve is the least. A computer search by Zhang in 2005 ruled out fewer than eleven.[35] Meanwhile Macintyre had shown in 1981 that the set of all true identities is decidable.[36] In 2026 Subercaseaux and Przybocki proved the conjecture with a SAT solver, as Remark 8.6 describes.[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 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. Every finite structure in a finite signature, once each of its elements has a name, 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.

A program is a combinator, and no program decides whether a program halts: a predictor of halting, run against itself on its own code, would contradict itself. The rules by which combinators run can be written as a finite set of first-order sentences, the same for every program, from which "the program halts" follows exactly when it does. So first-order validity is undecidable, even for the consequences of that one set. The first-order procedure

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

[31] Matiyasevich (1970).

[32] Doner and Tarski (1969).

[33] Wilkie (2000).

[34] Gurevič (1985, 1990).

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

[36] Macintyre (1981).

[37] Subercaseaux and Przybocki (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.

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

FIGURE 8.6. Incompleteness: who did what.

`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.

Halting can also be expressed in arithmetic, 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 contains Peano arithmetic, consistency is enough in place of soundness. And a consistent specification that contains Peano arithmetic cannot prove its own 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 8.40).

This part of the course began with the wish never to be surprised by a system one has specified. Its three chapters have said exactly 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 exactly what Hilbert's "we shall know" turned out to mean.

Hilbert's programme — status.

Item	Status after this chapter
1. Consistency	○ **in part**: a sound specification of arithmetic is consistent, because $\mathfrak{N}$ satisfies it; but a consistent specification containing PA cannot prove 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 8.9.

Exercises. The exercises run from practice to proof.

- Exercises 8.1 to 8.5 are about specifications and finite structures;
- Exercises 8.6 to 8.11 are about programs as theories and reductions;
- Exercises 8.12 to 8.14 are about arithmetic and the incompleteness theorems;
- Exercises 8.15 to 8.17 go with the optional sections, and are optional too.

Exercise 8.1. Say which of the following sets of sentences are specifications, with a reason for each.

(a) The eleven identities of Example 8.5, 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 8.2. 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 8.3. 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 8.4. 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 $\mathfrak{B}$ (Definition 8.11), and count its sentences.
(b) The sentence $\forall x\,(P(x) \to \neg P(f(x)))$ is true in $\mathfrak{B}$. Following the proof that finite structures can be captured, describe a derivation of it from the equality axioms and $\Delta_{\mathfrak{B}}$: which instances are proved, and how are they combined?

Exercise 8.5. Take the equality axioms and the description $\Delta_{\mathfrak{B}}$ of Exercise 8.4, 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 8.6. 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 that the sentences of Θ_{KS} force in any model of Θ_{KS}, as in Example 8.22. Which sentences for steps are used, and where?

Exercise 8.7. 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 the term structure interprets Nf by it.

Exercise 8.8. Let $\mathfrak{G}$ be the structure whose domain is the set of combinators, with K, S and application as in the term structure, in which Step, Nf and Halts are true of everything. Show that $\mathfrak{G}$ satisfies Θ_{KS} and makes η_Ω true. Why does this not contradict the lemma that a program is a theory? Why does the proof of that lemma need the term structure, and not just any model of Θ_{KS}, to show that η_M does not follow?

Exercise 8.9. Compute the code of $\mathsf{I} = \mathsf{S}\,\mathsf{K}\,\mathsf{K}$. Which numbers below 500 are codes, and of which combinators?

Exercise 8.10. Show that no program decides, given the code of a combinator M, whether M has a normal form. (This is, in effect, the first problem ever shown to be undecidable; the History tells the story.)

Exercise 8.11. 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 the program $\mathrm{prog}(e)$ does on n.)

Exercise 8.12. 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 8.13. Let Θ be a specification in Σ_{A} and d the code d_Θ of the translation 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 $\neg H(\overline{d}, \overline{d})$ added. Show that Θ' is a sound specification. Does the first incompleteness theorem apply to Θ', and with which sentence?

Exercise 8.14. 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 8.15. *(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 8.16. *(Optional.)* Write out the formula $Red_{\mathsf{S}}(i, p, \ell_P, \ell_Q, \ell_R)$ that Section 8.8 leaves to the reader, in the style of Red_{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 8.17. *(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.

Haskell B. Curry and Robert Feys. *Combinatory Logic, Volume I.* North-Holland, Amsterdam, 1958.

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.

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.