COMP0009 Logic — Problem Sheet 7

Part A — Specifications

Alice, Bob and Charlie share a network on which nobody trusts anybody; Mallory attacks it. The network keeps a *table*: who there is, who is whose boss, who is an administrator, which payments have been made. It also keeps a *rulebook*, a set of first-order sentences that every table the network could ever be in must satisfy. Charlie runs one of the computers, and you are going to help him. He has two tools. A *query engine* answers "is this sentence true of today's table?" by looking the table up. A search procedure, `REFUTE`, answers "does this sentence *follow* from the rulebook?", that is, "is it true in every table the rulebook allows?". It finds a proof whenever there is one, and when there is none it may run for ever.

Charlie wants the rulebook to be the last word: a set of sentences from which every truth about the network follows, so that search settles any question at all. In A1 he gets one, for the table of a single day. In A2 the table becomes a ledger that never ends, and the first crack shows. In A3 you look inside the network's scripts, the small programs that lock payments, and the crack turns out to be an old one: nobody can tell in advance whether a script stops. In A4 the rulebook is the arithmetic of the counting numbers, and you will name a true sentence that it cannot prove.

Marks are in square brackets; the whole of Part A is worth 25.

A1. Finite structures.

The network's table on Monday has three users. It says who is whose boss, with a function boss, and who is an administrator, with a relation symbol adm:

user	boss	adm?
Alice	Alice	yes
Bob	Alice	yes
Charlie	Bob	no

Alice is her own boss: boss is a function symbol, so it must give every user a value, Alice included. The table is a *structure* $\mathfrak{M}$: a set of elements (the three users), a value of each function symbol at each element, and a verdict of each relation symbol at each element, with $=$ meaning identity. Truth in $\mathfrak{M}$, written $\mathfrak{M} \models \varphi$, is computed from the table: a $\forall x$ runs x through every user, an $\exists x$ looks for one, and the connectives work as truth tables say. The set of sentences true in $\mathfrak{M}$ is its *theory*, $\mathrm{Th}(\mathfrak{M})$. For every sentence φ, exactly one of φ and $\neg\varphi$ is in $\mathrm{Th}(\mathfrak{M})$: a table takes a side on every question.

A *specification* Θ is a set of sentences that a checker can read: given any sentence, a program says whether it belongs to Θ. (Every finite set is one, and the network's rulebook is one.) A structure in which every sentence of Θ is true is a *model* of Θ, or *satisfies* Θ, and a sentence *follows* from Θ when it is true in every model of Θ. Proofs are derivations in the classical system NK, and $\Theta \vdash_{\mathrm{NK}} \varphi$ says that there is one with conclusion φ and its open hypotheses in Θ. Two facts about NK are used throughout; the notes (Chapter 6) prove them, and here you may take them on trust. *Soundness*: a derivation whose open hypotheses are true in a structure has a conclusion true in it, so NK proves from Θ only what follows from Θ. *Completeness*: whatever follows from Θ has a derivation from Θ.

A specification Θ *captures* a structure $\mathfrak{M}$ when what follows from it is exactly what is true in $\mathfrak{M}$: for every sentence φ,

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

Only truths: whatever Θ proves is true in $\mathfrak{M}$. If $\mathfrak{M} \models \Theta$, this half is free, by the soundness of NK. *Every truth*: whatever is true in $\mathfrak{M}$, Θ proves. The work is in this half.

For a finite table the work can be done, and the specification that does it is the one anyone would write. The *description* of a finite structure gives each element a name, and then lists:

- *distinctness*: $c \neq d$ for each pair of different elements, one sentence per pair;
- *closure*: $\forall x\,(x = c_1 \vee \cdots \vee x = c_n)$, there is nobody else;
- *the function tables*: $f(c) = d$ for every entry of every function's table;
- *the relation tables*: $P(c)$ for every entry that is in the relation, and $\neg P(c)$ for every entry that is not.

The notes (Chapter 8) prove that the description captures the structure with its elements named, once the equality axioms are added (everything equals itself; equals may be put for equals): it proves exactly the sentences true in the structure. And once a specification captures a table, every question about the table is settled by search: list the theorems of the specification, one derivation after another, and wait until φ or $\neg\varphi$ appears.

Example. On Sunday there were two users, Alice and Bob, with Alice her own boss, Bob's boss Alice, and Alice the only administrator. The description names them alice and bob and lists six sentences:

$$\mathsf{alice} \neq \mathsf{bob}, \qquad \forall x\,(x = \mathsf{alice} \vee x = \mathsf{bob}),$$

$$\mathsf{boss}(\mathsf{alice}) = \mathsf{alice}, \quad \mathsf{boss}(\mathsf{bob}) = \mathsf{alice}, \qquad \mathsf{adm}(\mathsf{alice}), \quad \neg\mathsf{adm}(\mathsf{bob}).$$

Is somebody not an administrator, $\exists x\,\neg\mathsf{adm}(x)$? The sentence is true in Sunday's table, at Bob, so the listing of theorems prints it: a derivation is the sentence $\neg\mathsf{adm}(\mathsf{bob})$ of the description, with one ($\exists$I) bar under it. Had the sentence been false, the listing would have printed its negation instead. One of the two always comes, because the table makes one of them true and the description proves every truth.

(a) Here is Monday's description, with the names alice, bob and $\mathsf{charlie}$, and eight gaps. Fill them in, and say how many sentences the description has. [2]

- distinctness: $\mathsf{alice} \neq \mathsf{bob}$, _____, _____;
- closure: $\forall x\,(x = \mathsf{alice} \vee$ _____ $\vee$ _____$)$;
- the function table: $\mathsf{boss}(\mathsf{alice}) =$ _____, $\mathsf{boss}(\mathsf{bob}) =$ _____, $\mathsf{boss}(\mathsf{charlie}) =$ _____;
- the relation table: $\mathsf{adm}(\mathsf{alice})$, _____, _____.

(b) The half "only truths" costs nothing. Say why: which fact about the sentences of the description, and which fact about NK, give it? [1]

(c) Charlie asks whether anybody is their own boss. To settle it he lists theorems until one of $\psi := \forall x\,(\mathsf{boss}(x) \neq x)$ and its negation appears.

(i) Which of ψ and $\neg\psi$ will the listing print? Say why the other one never appears, however long he waits. [1]

(ii) Here is the derivation that the listing finds, with two gaps: a sentence of the description, and the instance that the ($\forall$E) bar takes. Fill them in. Recall that $t \neq u$ is short for $t = u \to \bot$. [1]

$$\dfrac{\dfrac{\dfrac{[\forall x\,(\mathsf{boss}(x) \neq x)]^1}{\boxed{}}\,\forall\mathrm{E} \qquad \boxed{}}{\bot}\,\to\mathrm{E}}{\neg\forall x\,(\mathsf{boss}(x) \neq x)}\,\to\mathrm{I},1$$

(d) Mallory sends Charlie a table of her own, with a fourth row:

user	boss	adm?
Alice	Alice	yes
Bob	Alice	yes
Charlie	Bob	no
Mallory	Mallory	yes

Which sentences of Monday's description are true in Mallory's table, and which are false? The sentence $\chi := \forall x\,(\mathsf{adm}(x) \to \mathsf{boss}(x) = \mathsf{alice})$, "every administrator answers to Alice", is true on Monday. What does Mallory's table show about χ and the description *without* its closure sentence? [1]

A2. An infinite structure.

Payments on the network are numbered as they arrive: $0, 1, 2, \ldots$ The *ledger* is the structure whose elements are these numbers, with a constant 0 for the first payment and a one-place function next, where $\mathsf{next}(n)$ is $n+1$, the payment after n. Its table cannot be written out, because it never ends. Only sentences can describe the ledger, and Charlie writes two:

(L1) $\forall x\,(\mathsf{next}(x) \neq 0)$ no payment comes before the first;

(L2) $\forall x\,(x \neq 0 \to \exists y\,(\mathsf{next}(y) = x))$ every payment but the first comes after some payment.

Both are true in the ledger. The question is whether what is true in the ledger *follows* from them, that is, holds in every model of them. To show that a sentence does not follow, build a *rogue table*: a model of the given sentences (a table) in which the sentence in question is false, that is, a counter-model. Nothing that follows from the given sentences can fail there.

Example. Does (L2) follow from (L1) alone? To show that it does not, we want a table where (L1) holds and (L2) fails, that is, a payment other than 0 that comes after nobody. Three rows do it. Call the extra rows g and h:

$$\mathsf{next}(0) = g, \qquad \mathsf{next}(g) = g, \qquad \mathsf{next}(h) = g.$$

(L1) holds: no row has 0 in its next column. (L2) fails at h: it is not 0, and no row has h in its next column. So (L1) alone does not give (L2), and Charlie was right to write both.

(a) Check that (L1) and (L2) are true in the ledger: for each, say in one sentence why, running x through the payments as the query engine would. Then check the third sentence Charlie has not yet written, which is true in the ledger too:

$$\text{(L3)}\quad \forall x\,\forall y\,(\mathsf{next}(x) = \mathsf{next}(y) \to x = y),$$

two different payments are followed by different payments. [1]

(b) Charlie's rulebook is (L1) and (L2). Alice asks whether it guarantees that no payment is followed by itself,

$$\text{(loop)}\quad \forall x\,(\mathsf{next}(x) \neq x),$$

which is true in the ledger. Mallory says that it does not follow, and sends a rogue table of two rows, 0 and a ghost payment g, with its next column missing:

$$\mathsf{next}(0) = ____, \qquad \mathsf{next}(g) = ____.$$

Fill in the column so that (L1) and (L2) hold and (loop) fails, and check all three. Which of (L1), (L2), (L3) does Mallory's table break? [2]

Hint: (L1) leaves only one choice for $\mathsf{next}(0)$*, and then (loop) must fail somewhere.*

(c) Charlie adds (L3) to the rulebook.

(i) Explain why no table with finitely many rows satisfies (L1), (L2) and (L3) together. So Mallory's next rogue table must be infinite. [1]

Hint: by (L3), different rows have different next *values. With n rows, how many different values does the* next *column hold, and which row is then missing from it?*

(ii) Mallory's new rogue table is the ledger itself, with one ghost payment g added, whose next is g. Check (L1), (L2) and (L3) at the ghost, and say where (loop) fails. [1]

(d) Charlie adds (loop) to the rulebook as (L4). Mallory adds two ghosts, g and h, to the ledger. Fill in $\mathsf{next}(g) =$ ____ and $\mathsf{next}(h) =$ ____ so that (L1) to (L4) all hold, and write down a sentence that is true in the ledger and false in Mallory's table. Charlie adds that sentence too. In your own words: what can the rulebook never do here? [1]

A3. Programs as theories.

The network locks each payment behind a *script*, built from the two instructions K and S by writing things side by side (grouping to the left). A script runs by the rules (K), (S), (app_1) and (app_2), and *halts* when some run reaches a *normal form*: K, S, $\mathsf{K}\,N$, $\mathsf{S}\,N$ or $\mathsf{S}\,N\,N'$, with N and N' normal. Every program is such a script: the notes prove that scripts compute everything any machine computes, and you may take that on trust. So "no program can" means "no script can". A number n is handed to a script as its *numeral* $\underline{n}$, and a script M *halts on* n when $M\,\underline{n}$ halts.

A theory says how scripts run. Its signature has the constants K and S, a two-place function symbol for putting side by side, written as application is written, and the relation symbols $\mathsf{Step}(x, y)$ ("x steps to y"), $\mathsf{Nf}(x)$ ("x is a normal form") and $\mathsf{Halts}(x)$. The *theory of computation* Π has eleven sentences, each with $\forall$ in front of its variables. Four say how a script steps, one for each rule, and are named after it:

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

Five give the shapes of a normal form, and two say what halting is:

- *normal forms*: $\mathsf{Nf}(\mathsf{K})$; $\mathsf{Nf}(\mathsf{S})$; $\mathsf{Nf}(x) \to \mathsf{Nf}(\mathsf{K}\,x)$; $\mathsf{Nf}(x) \to \mathsf{Nf}(\mathsf{S}\,x)$; $(\mathsf{Nf}(x) \wedge \mathsf{Nf}(y)) \to \mathsf{Nf}(\mathsf{S}\,x\,y)$;
- *halting*: $\mathsf{Nf}(x) \to \mathsf{Halts}(x)$, *a normal form halts*; and $(\mathsf{Step}(x, y) \wedge \mathsf{Halts}(y)) \to \mathsf{Halts}(x)$, *halting passes back*.

The theory is the same for every script: only the sentence $\mathsf{Halts}(M)$ mentions M. The notes (Chapter 8) prove that *a script M halts exactly when* $\mathsf{Halts}(M)$ *follows from* Π. By the soundness and completeness of NK, that is exactly when $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M)$: a run of M to a normal form can be written as a derivation, as the example shows. So Charlie can hand Π and $\neg\mathsf{Halts}(M)$ to `REFUTE`, which answers `UNSAT` (no structure satisfies them together) exactly when M halts.

Scripts can be numbered, because a script is a finite string. The *string* of a script is the script written out with all its brackets: $\mathrm{str}(\mathsf{K})$ is `K`, $\mathrm{str}(\mathsf{S})$ is `S`, and $\mathrm{str}(M\,N)$ is $(\,\mathrm{str}(M)\,\mathrm{str}(N)\,)$. The *code* $\ulcorner M \urcorner$ of a script is the number its string spells in base five, with 1 for `K`, 2 for `S`, 3 for (and 4 for). Every number names a script: for a number e, the *script with code e* is

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

The second line is a convention: Ω is a script that never halts, so $\mathsf{K}\,\Omega$ halts on no input.

Example. The script $\mathsf{K}\,(\mathsf{K}\,\mathsf{K})\,\mathsf{S}$ steps to $\mathsf{K}\,\mathsf{K}$, which is a normal form, so it halts. Here is the derivation of $\mathsf{Halts}(\mathsf{K}\,(\mathsf{K}\,\mathsf{K})\,\mathsf{S})$ from Π, one atomic sentence per line. Each line comes from the sentence of Π named beside it by ($\forall$E), with the terms shown, and then by ($\wedge$I) and ($\to$E) with the earlier lines shown.

1.	$\mathsf{Step}(\mathsf{K}\,(\mathsf{K}\,\mathsf{K})\,\mathsf{S},\ \mathsf{K}\,\mathsf{K})$	(K), at $\mathsf{K}\,\mathsf{K}$, S
2.	$\mathsf{Nf}(\mathsf{K})$	normal forms, the first
3.	$\mathsf{Nf}(\mathsf{K}\,\mathsf{K})$	K with one argument, at K; with 2
4.	$\mathsf{Halts}(\mathsf{K}\,\mathsf{K})$	a normal form halts, at $\mathsf{K}\,\mathsf{K}$; with 3
5.	$\mathsf{Halts}(\mathsf{K}\,(\mathsf{K}\,\mathsf{K})\,\mathsf{S})$	halting passes back, at $\mathsf{K}\,(\mathsf{K}\,\mathsf{K})\,\mathsf{S}$, $\mathsf{K}\,\mathsf{K}$; with 1, 4

Line 1 is the run, lines 2 and 3 recognise its result as a normal form, and lines 4 and 5 pass halting back to the start. The string of $\mathsf{S}\,\mathsf{K}$ is `(SK)`, with the digits $3, 2, 1, 4$, so

$$\ulcorner \mathsf{S}\,\mathsf{K} \urcorner = 3 \cdot 5^3 + 2 \cdot 5^2 + 1 \cdot 5 + 4 = 434.$$

Different scripts have different strings, and so different codes. A script can compute the code of a script, and the script of a code.

(a) Write the string of $\mathsf{K}\,\mathsf{S}$ and compute its code. Mallory sends Charlie the number 444 and calls it the code of a script. Write 444 in base five. Is it a code? [1]

(b) Here is a derivation of $\mathsf{Halts}(\mathsf{S}\,(\mathsf{K}\,\mathsf{K})\,\mathsf{K}\,\mathsf{S})$ from Π, set out as in the example, with three lines missing.

1.	$\mathsf{Step}(\mathsf{S}\,(\mathsf{K}\,\mathsf{K})\,\mathsf{K}\,\mathsf{S},\ \mathsf{K}\,\mathsf{K}\,\mathsf{S}\,(\mathsf{K}\,\mathsf{S}))$	(S), at $\mathsf{K}\,\mathsf{K}$, K, S
2.	$\mathsf{Step}(\mathsf{K}\,\mathsf{K}\,\mathsf{S},\ \mathsf{K})$	(K), at K, S
3.	_____	(app_1), at $\mathsf{K}\,\mathsf{K}\,\mathsf{S}$, K, $\mathsf{K}\,\mathsf{S}$; with 2
4.	$\mathsf{Nf}(\mathsf{S})$	normal forms, the second
5.	_____	K with one argument, at S; with 4
6.	$\mathsf{Nf}(\mathsf{K}\,(\mathsf{K}\,\mathsf{S}))$	K with one argument, at $\mathsf{K}\,\mathsf{S}$; with 5
7.	$\mathsf{Halts}(\mathsf{K}\,(\mathsf{K}\,\mathsf{S}))$	a normal form halts; with 6
8.	_____	halting passes back; with 3, 7
9.	$\mathsf{Halts}(\mathsf{S}\,(\mathsf{K}\,\mathsf{K})\,\mathsf{K}\,\mathsf{S})$	halting passes back; with 1, 8

(i) Fill in the three lines. Which lines are the run of the script, and at which normal form does it halt? [2]

(ii) Draw line 3 as a tree of NK whose leaves are a sentence of Π and line 2. [1]

(c) Charlie would also like Π to derive $\neg\mathsf{Halts}(M)$ whenever a script M never halts. Find a structure that satisfies Π in which $\mathsf{Halts}(M)$ is true for every script M. What follows about deriving $\neg\mathsf{Halts}(M)$ from Π? And what does `REFUTE` do when it is given Π and $\neg\mathsf{Halts}(M)$ for a script M that never halts? [1]

Hint: one element, and every relation true of everything.

(d) Mallory offers Charlie a script h that, she says, settles the question that `REFUTE` cannot: given numbers e and n, it always halts, and answers *yes* if $\mathrm{prog}(e)$ halts on n, and *no* if it does not. Charlie writes a program that uses it:

```
procedure P(e)                          -- e a number
    ask h whether prog(e) halts on e
    if h says yes:  ______
    if h says no:   ______
```

Every program is a script, so P has a translation, a script that halts on exactly the inputs on which P halts; call its code p, so that $\mathrm{prog}(p)$ is that translation, and run P on p.

(i) Fill in the two blanks so that P does the *opposite* of what h predicts. Then write out the two cases: suppose P halts on p, and suppose it does not. What follows about Mallory's h? [1]

(ii) Charlie concludes that no program decides whether a sentence follows from a specification: "if one did, I could feed it Π and $\mathsf{Halts}(M\,\underline{n})$". Finish his argument in two or three sentences, and say what it means for `REFUTE`: can a cleverer search answer *no* whenever the answer is no? [1]

A4. The incompleteness theorems.

The ledger of A2 was the counting numbers with next alone. Now the specification speaks of the numbers with 0, the next number s, $+$, $\times$, $<$ and $=$. The structure is $\mathfrak{N}$, the set $\{0, 1, 2, ...\}$ with these symbols given their usual meanings, traditionally called the *standard model* of arithmetic. The number n is named by the *numeral* $\overline{n}$, which is s written n times round 0; so $\overline{3}$ is $s(s(s(0)))$.

This numeral is a term of arithmetic. The numeral $\underline{n}$ of A3 is a different thing, a script. Both stand for the number n, one in sentences about numbers and the other as an input to a script, and both appear below: $H(\overline{e}, \overline{n})$ is a sentence, and prog(e) halts on n when prog$(e)\,\underline{n}$ halts.

Charlie takes the specification everybody takes, *Peano arithmetic* PA: the equality axioms and, with $\forall$ in front of each,

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

A specification Θ for arithmetic is *sound* if every sentence it proves is true in $\mathfrak{N}$. By the soundness of NK, Θ is sound exactly when $\mathfrak{N} \models \Theta$.

You need two facts, which the notes (Chapter 8) prove and you may use without proof. First, halting is definable in arithmetic: there is a formula $H(x, y)$ of this language such that, for all numbers e and n,

$$\text{prog}(e) \text{ halts on } n \quad \text{if and only if} \quad \mathfrak{N} \models H(\overline{e}, \overline{n}).$$

Second, for every specification Θ a program lists its theorems, one after another, by checking every string in turn to see whether it is a derivation from Θ (A1 used this). From here on you work on your own, with the method of A3(d): *run a program on its own code, and arrange for it to do the opposite.*

(a) Say why PA is a specification, though it has infinitely many sentences, and why it is sound. [1]

(b) Let Θ be any sound specification for arithmetic. Charlie writes the *diagonal program* of Θ:

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

Every program is a script, so D has a translation, a script that halts on exactly the inputs on which D halts; let d be its code, so that $\text{prog}(d)$ is the translation of D. In words, D on the input e waits for Θ to prove that $\text{prog}(e)$ does not halt on e, and halts if that ever happens. *The first incompleteness theorem* says that $\neg H(\overline{d}, \overline{d})$ is true and Θ does not prove it. Here is its argument in three steps, with six gaps. Fill them in. [3]

1. *D does not halt on d.* Suppose it did. Then the listing of Θ printed the sentence _____, so Θ proves it. Since Θ is _____, the sentence is true in $\mathfrak{N}$. By the fact about H, that means that D _____ on d, which contradicts the supposition.
2. *So $\neg H(\overline{d}, \overline{d})$ is true in $\mathfrak{N}$,* because, by the fact about H again, $H(\overline{d}, \overline{d})$ is true exactly when _____.
3. *And Θ does not prove it.* If it did, the listing would print it, and then D would _____ on d, which by step ___ it does not.

(c) Charlie patches the specification: Θ' is Θ with $\neg H(\overline{d}, \overline{d})$ added. Is Θ' sound? Does the argument of (b) apply to it, and with which sentence? Why is that sentence not the same one? [1]

(d) Some questions can still be decided. Here are five that Charlie asks of the network. For each, say whether some program answers it correctly on every input, always halting; and if not, whether a program at least answers *yes* on every input whose answer is yes, as `REFUTE` does. Give a reason, pointing at the question of Part A that settles it. [1]

question	decided?
Is this sentence true of today's table, which has finitely many users?	
Do the constraints on the network's on/off switches allow this setting? (A propositional question.)	
Does this first-order sentence follow from the specification?	
Does this script halt on this input?	
Is this sentence of arithmetic true of the counting numbers?	

Two more theorems close the story, and you may take both on trust. A specification Θ is *consistent* if $\Theta \nvdash_{\text{NK}} \bot$. A sound specification is consistent, since $\bot$ is false in $\mathfrak{N}$. The notes (Chapter 8) prove *the first incompleteness theorem, from consistency*: for a consistent specification Θ that proves every true sentence saying that a program halts, consistency alone is enough for the argument of (b). Every specification with $\text{PA} \subseteq \Theta$ proves those sentences, by a fact about PA that the notes take on trust. Consistency can itself be said in arithmetic: the sentence Con_Θ says that the program that waits for the listing to print $\bot$ never halts. From three more facts about what PA proves of the listing, which they state exactly and take on trust, the notes then prove *the second incompleteness theorem*:

if Θ is a consistent specification with $\text{PA} \subseteq \Theta$, then $\quad \Theta \nvdash_{\text{NK}} \text{Con}_\Theta$.

> Charlie cannot even get his rulebook to certify that it never contradicts itself. The certificate is the structure $\mathfrak{N}$, and the specification cannot issue it for itself.

Charlie's wall is real. Compilers meet A3 every day. A proof assistant must know that every function it accepts terminates, since a function that never returns could be used as a proof of anything. Lean, Agda and Rocq therefore ask for a termination argument, and when the checker cannot find one, Lean makes you write `partial` in front of the definition: an admission that the question is undecidable in general. Rice showed in 1953 that the same goes for every property of what a program computes that some programs have and others lack: none is decidable. School algebra is A2 in miniature: eleven identities true of the positive whole numbers, an identity found by Wilkie in 1981 that they do not derive, and a rogue table with twelve elements in which the eleven hold and Wilkie's fails. In 2026 Subercaseaux and Przybocki proved with a SAT solver that no smaller rogue table exists, and Hajdari and Niederhauser, with a SAT encoding of their own, reached the same answer. Goldbach's conjecture, that every even number above 2 is a sum of two primes, is one sentence of A4's language; nobody knows whether PA proves it, refutes it, or neither. And the diagonal argument of A4 has itself been checked by machine: both incompleteness theorems were verified in the proof assistant Isabelle by Paulson in 2014, every step checked by a program that is itself a script of A3's kind.

Part B — Practice

Practice, for when you want more of the same; nothing here is marked. The drills go from reading and computing, through building rogue tables and deriving halting, to deciding and justifying with no scaffolding. The last two, B13 and B14, go beyond this week: try them only if you are curious. In the drills, P, Q and R are relation symbols, f a one-place function symbol, c and d names, and x, y, z, u, v, w variables. What you need:

- A *specification* Θ is a set of sentences that a program can recognise. It *captures* a structure $\mathfrak{M}$ when $\Theta \vdash_{\mathrm{NK}} \varphi$ exactly when $\mathfrak{M} \models \varphi$, for every sentence φ: *only truths* (free, by the soundness of NK, when $\mathfrak{M} \models \Theta$) and *every truth*.
- The *description* of a finite structure: a name for each element; distinctness ($c \neq d$ for each pair); closure ($\forall x\,(x = c_1 \vee \cdots \vee x = c_n)$); every entry of every function table ($f(c) = d$); every entry of every relation table ($P(c)$ or $\neg P(c)$). With the equality axioms it captures the structure with its elements named, and so proves exactly the sentences true in the structure.
- *Deciding by search*: when Θ captures $\mathfrak{M}$, list the theorems of Θ and wait for φ or $\neg\varphi$. One of them comes.
- To show that φ does *not* follow from Θ, build a *rogue table*: a model of Θ in which φ is false (a counter-model).
- A *script* is built from K and S; it steps by (K), (S), (app_1), (app_2), and *halts* when some run reaches a normal form (K, S, $\mathsf{K}\,N$, $\mathsf{S}\,N$, $\mathsf{S}\,N\,N'$). Its *string* is the script with all its brackets; its *code* is the string read in base five ($\mathtt{K}, \mathtt{S}, (,)$ as $1, 2, 3, 4$). The script with code e is $\mathrm{prog}(e)$, or $\mathsf{K}\,\Omega$ if e is the code of no script.
- The *theory of computation* Π: the four step rules as sentences about Step, the five shapes as sentences about Nf, and $\mathsf{Nf}(x) \to \mathsf{Halts}(x)$, $(\mathsf{Step}(x, y) \wedge \mathsf{Halts}(y)) \to \mathsf{Halts}(x)$. A script M halts exactly when $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M)$: a run is a derivation, one atomic sentence per line. Π never derives $\neg\mathsf{Halts}(M)$.
- *Halting is definable in arithmetic*: there is a formula $H(x, y)$ with $\mathfrak{N} \models H(\overline{e}, \overline{n})$ exactly when $\mathrm{prog}(e)$ halts on n. A specification Θ for arithmetic is *sound* if it proves only sentences true in $\mathfrak{N}$, that is, if $\mathfrak{N} \models \Theta$; Peano arithmetic PA is sound.

- *The diagonal*: no program decides halting, because a program could run the decider on its own code and do the opposite.
- *The first incompleteness theorem*: a sound specification Θ for arithmetic does not prove the true sentence $\neg H(\overline{d}, \overline{d})$, where d is the code of the translation of the program D that waits for Θ to prove it. *From consistency*: the same holds for every consistent Θ with PA $\subseteq \Theta$.
- *The second incompleteness theorem*: if Θ is a consistent specification with PA $\subseteq \Theta$, then $\Theta \nvdash_{\mathrm{NK}} \mathrm{Con}_\Theta$, where Con_Θ says that the program that waits for Θ to prove $\bot$ never halts.

B1. Reading. Which of these sets of sentences can a program recognise, that is, which are specifications? Give a reason for each.

(i) The two sentences (L1) and (L2) of A2.
(ii) The set of all sentences true in Monday's table of A1.
(iii) The set of all sentences of arithmetic true in $\mathfrak{N}$.
(iv) The set of sentences $\mathsf{Halts}(M)$ for the scripts M that halt.
(v) The set of sentences that follow from the theory of computation Π.

B2. Strings and codes.

(i) Write the strings of $\mathsf{S}\,(\mathsf{K}\,\mathsf{S})$ and of $\mathsf{K}\,\mathsf{K}\,\mathsf{K}$. Why are they different, though both have three letters?
(ii) Compute the code of $\mathsf{K}\,\mathsf{S}\,\mathsf{K}$.
(iii) Which of the numbers 1, 3, 409, 440 and 2199 are codes, and of which scripts?

B3. A structure $\mathfrak{M}$ has two elements, c and d, a function f with $f(c) = d$ and $f(d) = d$, and a relation symbol P true of c alone. Write its description, part by part, and count the sentences. Which sentence would you add if a third element e joined, and how many sentences would the description then have?

B4. Deciding by search, in the structure of B3. For each sentence say which of it and its negation the listing of theorems prints, and name the entry of the description that a derivation starts from.

(i) $\forall x\,(P(x) \to \neg P(f(x)))$
(ii) $\exists x\,(f(x) = x)$
(iii) $\forall x\,\exists y\,(f(y) = x)$

B5. Halting as a derivation. For each script M, give a derivation of $\mathsf{Halts}(M)$ from Π, set out as in A3(b): one atomic sentence per line, with the sentence of Π, the terms and the earlier lines beside each.

(i) $\mathsf{K}\,\mathsf{S}\,(\mathsf{K}\,\mathsf{K})$ (ii) $\mathsf{S}\,\mathsf{K}\,\mathsf{S}\,\mathsf{K}$ (iii) $\mathsf{K}\,(\mathsf{K}\,\mathsf{K}\,\mathsf{S})$

Hint: in (iii) the step happens inside the argument.

B6. What Π leaves open.

(i) Show that Π does not derive $\mathsf{Step}(\mathsf{K}, \mathsf{K})$: find a structure that satisfies Π in which it is false.
(ii) Show that Π does not derive $\neg\mathsf{Step}(\mathsf{K}, \mathsf{K})$ either.

(iii) So Π settles neither $\mathsf{Step}(\mathsf{K},\mathsf{K})$ nor its negation. Why does that not spoil the fact that M halts exactly when $\Pi \vdash_{\mathrm{NK}} \mathsf{Halts}(M)$?

B7. A rogue table. The whole numbers $\ldots, -2, -1, 0, 1, 2, \ldots$ with $f(x) = x + 1$ satisfy two sentences:

$$\text{(r1)}\ \ \forall x\,(f(x) \neq x) \qquad\qquad \text{(r2)}\ \ \forall x\,\exists y\,(f(y) = x).$$

The sentence $\forall x\,(f(f(x)) \neq x)$ is true of them too. Show that it does not follow from (r1) and (r2): fill in a two-row table, $f(c) =$ ____, $f(d) =$ ____, check both sentences, and say where the sentence fails.

B8. PA calculates. Show that $\overline{2}+\overline{2} = \overline{4}$, that is, $s(s(0))+s(s(0)) = s(s(s(s(0))))$, follows from the two axioms for $+$ by rewriting the left side, one axiom at a time, until it becomes the right side. Name the axiom used at each step.

B9. Definable sets. A set of numbers, or of pairs, is *definable* if some formula of arithmetic is true in $\mathfrak{N}$ of exactly its members. Write a formula for each, with every quantifier bracketed, using only 0, s, $+$, $\times$, $<$ and $=$.

(i) x is odd.
(ii) $x \leq y$.
(iii) x is a square.
(iv) x and y have a common divisor greater than 1.

B10. A reduction. Show that no program decides, given the code of a script, whether that script halts on *every* input.

Hint: from e and n, with $M = \mathrm{prog}(e)$, a program can compute the code of $\mathsf{K}\,(M\,\underline{n})$. What does $\mathsf{K}\,(M\,\underline{n})\,\underline{m}$ do, for any m?

B11. Build a rogue table, on your own. Let Θ be the specification with the two sentences (L1) $\forall x\,(\mathsf{next}(x) \neq 0)$ and (L3) $\forall x\,\forall y\,(\mathsf{next}(x) = \mathsf{next}(y) \to x = y)$. Show that (L2), $\forall x\,(x \neq 0 \to \exists y\,(\mathsf{next}(y) = x))$, does not follow from Θ. Then show that your rogue table could not have been finite.

B12. Decide and justify. For each set, say whether a program can *list* it (print exactly its members, one after another, running for ever) and whether a program can *decide* it (given a candidate, always halt and say whether it is a member).

(i) The sentences that follow from the theory of computation Π.
(ii) The sentences that follow from the description of Monday's table (A1) and the equality axioms.
(iii) The scripts that do *not* halt.

B13. *(For the curious.)* The optional section "Defining Halting" of the notes (Chapter 8) packs a finite list of numbers into two numbers a and b: the ith entry is the remainder of a on division by $1 + (i+1)\,b$. Find a and b that pack the list $3, 1$: first say why $b = 1$ and $b = 2$ cannot work, then find the smallest b that can, and an a for it. Then write "the ith entry is v" as a formula of arithmetic in a, b, i and v.

B14. *(For the curious.)* The rationals with $<$ are an infinite structure that a short specification *does* capture: the order is irreflexive, transitive and linear, between any two

points there is a third, and there is no least or greatest point. The optional section "Decidable Theories" of the notes (Chapter 8) shows that under these sentences every formula is equivalent to one without quantifiers.

(i) Say which quantifier-free formula in y and z is equivalent to $\exists x\,(y < x \wedge x < z)$, and why.

(ii) Do the same for $\exists x\,(x < y)$.

(iii) Why does (ii) fail for the counting numbers with $<$? What does that say about capturing them with these sentences?