6 Arithmetic, computability, and incompleteness

The standard mathematical logic curriculum, as well as looking at some elementary results about formalized theories and their models in general, investigates two particular instances of non-trivial, rigorously formalized, axiomatic systems. First, there's *arithmetic* (a paradigm theory about finite whatnots); and then there is *set theory* (a paradigm theory about infinite whatnots). We consider set theory in the next chapter. This chapter is about arithmetic and related matters. More specifically, we consider three inter-connected topics:

1. The elementary theory of numerical *computable functions*.
2. Formal *theories of arithmetic* and how they represent computable functions.
3. Gödel's epoch-making proof of the *incompleteness* of any sufficiently nice formal theory that can 'do' enough arithmetical computations.

Before turning to some short topic-by-topic overviews, though, it is well worth pausing for a quick general point about why the idea of computability is of such very central concern to formal logic.

6.1 Logic and computability

(a) The aim of regimenting informal arguments and informal theories into formalized versions is to eliminate ambiguities and to make everything entirely determinate and transparently clear (even if it doesn't always seem that way to beginners!). So, for example, we want it to be entirely clear what is and what isn't a formal sentence of a given theory, what is and what isn't an axiom of the theory, and what is and what isn't a formal proof in the theory. We want to be able to settle these things in a way which leaves absolutely no room left for doubt or dispute.

(b) As a step towards sharpening this thought, let's say as an initial rough characterization:

> A property P is *effectively decidable* if and only if there is an *algorithm* (a finite set of instructions for a deterministic computation)

for settling in a finite number of steps, whether a relevant object has property P.

Relatedly, the answer to a question Q is effectively decidable if and only if there is an algorithm which gives the answer, again by a deterministic computation, in a finite number of steps.

To put it only slightly different words, a property P is effectively decidable just when there's a step-by-step mechanical routine for settling whether an object of the relevant kind has property P, such that a suitably programmed deterministic computer could in principle implement the routine (idealizing away from practical constraints of time, etc.). Similarly, the answer to a question Q is effectively decidable just when a suitably programmed computer could deliver the answer (in principle, in a finite time).

Two initial examples from propositional logic: the property of being a tautology is effectively decidable (by a truth-table test); and we can effectively decide what is the main connective of a sentence (by bracket counting).

And the point we made at the outset in (a) now comes to this: we will want it to be effectively decidable e.g. whether a given string of symbols has the property of being a well-formed formula of a certain formal language, whether a formula is an axiom of a given formal theory, and whether an array of formulas is a correctly formed proof of the theory. In other words, we will want to set up a formal deductive theory so that a computer could, in principle, mindlessly check e.g. the credentials of a purported proof by deciding whether each step of the proof is indeed in accordance with the official rules of the theory.

(c) NB: It is one thing to be able to effectively decide whether a purported proof of P really is a proof in a given formal theory T. It is another thing entirely to be able to decide in advance whether P actually *has* a proof in T.

You'll soon enough find out that, e.g., in a properly set up formal theory of arithmetic T we can effectively check whether a supposed proof of P indeed conforms to the rules of the game. But once we are dealing with an even mildly interesting T, there will be no way of deciding in advance whether a T-proof of P exists. Such a theory T is said to be undecidable.

It is of course nice when a theory *is* decidable, i.e. when a computer can tell us whether a given proposition does or doesn't follow from the theory. But few interesting theories are decidable in this sense: so mathematicians aren't going to be put out of business!

(d) Now, in our initial rough definition of the notion of effective decidability, we invoked the idea of what an idealized computer could (in principle) do by implementing some algorithm. This idea surely needs further elaboration.

1. As a preliminary step, we can narrow our focus and just consider the decidability of *arithmetical* properties.

   Why? Because we can always represent facts about finite whatnots like formulas and proofs by using numerical codings. We can then trade in questions about formulas or proofs for questions about their code numbers.

2. And as a second step, we can also trade in questions about the effective decidability of arithmetical *properties* for questions about the algorithmic computability of numerical *functions*.

   Why? Because for any numerical property P we can define a corresponding numerical function (its so-called 'characteristic function') c_P such that if n has the property P, $c_P(n) = 1$ and if n doesn't the have property P, $c_P(n) = 0$. Think of '1' as coding for truth, and '0' for falsehood. Then the question (i) 'can we effectively decide whether a number has the property P?' becomes the question (ii) 'is the numerical function c_P effectively computable by an algorithm?'.

So, by those two steps, we do quickly move from e.g. the question whether it is effectively decidable whether a string of symbols is a wff to a corresponding question about whether a certain numerical function is computable.

6.2 Computable functions: an overview

(a) For convenience, we will now use 'S' for the function that maps a number to its successor (where we previously used a prime). Consider, then, the following pairs of equations:

$$x + 0 = x$$
$$x + Sy = S(x + y)$$

$$x \times 0 = 0$$
$$x \times Sy = (x \times y) + x$$

$$x^0 = S0$$
$$x^{Sy} = (x^y \times x)$$

These pairs of equations should be very familiar (at least when written with postfix '+1' instead of prefix 'S'): they in turn define addition, multiplication and exponentiation for the natural numbers.

Take the initial pair of equations. The first of them fixes the result of adding zero to a given number. The second fixes the result of adding the successor of y in terms of the result of adding y. Hence applying and re-applying the two equations, they together tell us how to add $0, S0, SS0, SSS0, \ldots$, i.e. they tell us how to add *any* natural number to a given number x. Similarly, the first of the equations for multiplication fixes the result of multiplying by zero. The second equation fixes the result of multiplying by Sy in terms of the result of multiplying by y and doing an addition. Hence the two pairs of equations together tell us how to multiply a given number x by any of $0, S0, SS0, SSS0, \ldots$. Similarly of course for the pair of equations for exponentiation.

And now note that the six equations taken together not only *define* exponentiation, but they do so by giving us an *algorithm* for computing x^y for any natural numbers x, y – they tell us how to compute x^y by doing repeated multiplications, which we in turn compute by doing repeated additions, which we

compute by repeated applications of the successor function. That is to say, the chain of equations amounts to a set of instructions for a deterministic step-by-step computation which will output the value of x^y in a finite number of steps. Hence, exponentiation is indeed an effectively computable function.

(b) In each of our pairs of equations, the second one fixes the value of the defined function for argument Sy by invoking the value of the *same* function for argument y. A procedure where we evaluate a function for one input by calling the *same* function for some smaller input(s) is standardly termed 'recursive' – and the particularly simple type of procedure we've illustrated three times is called, more precisely, primitive recursion. Now – arm-waving more than a bit! – consider *any* function which can be defined by a chain of equations similar to the chain of equations giving us a definition of exponentiation. Suppose that, starting from trivial functions like the successor function, we can build up the function's definition by using primitive recursions and plugging one function we already know about into another. Such a function is said to be *primitive recursive*.

And generalizing from the case of exponentiation, we have the following observation:

> Any primitive recursive function is similarly effectively computable

(c) So far, so good, However, it is easy to show that

> Not all effectively computable functions are primitive recursive.

A very neat abstract argument proves the point – but I'm not going to give spoilers here! But this raises an obvious question: what further ways of defining functions – in addition to primitive recursion – also give us effectively computable functions?

Here's a pointer. The definition of (say) x^y by primitive recursion in effect tells us to start from x^0, then loop round applying the recursion equation to compute x^1, then x^2, then x^3, ..., keeping going until we reach x^y. In all, we have to loop around y times. In some standard computer languages, implementing this procedure involves using a 'for' loop (which tells us to iterate some procedure, counting as we go, and to do this for cycles numbered 1 to y). In this case, *the number of iterations is given in advance as we enter the loop.* But of course, standard computer languages also have programming structures which implement unbounded searches – they allow open-ended 'do until' loops (or equivalently, 'do while' loops). In other words, they allow some process to be iterated until a given condition is satisfied, *where no prior limit is put on the number of iterations to be executed.*

This suggests that one way of expanding the class of computable functions beyond the primitive recursive functions will be to allow computations employing *open-ended searches*. So let's suppose we do this (there's a standard device for this, but let's not worry about the details now). Functions – more precisely, *total* functions that deliver an output for any numerical input – which can be computed by a chain of applications of primitive recursion and/or open-ended searches are called (simply) *recursive*.

(d) Predictably enough, the next question is: have we *now* got all the effectively computable functions?

The claim that the recursive functions are indeed just the intuitively computable total functions is *Church's Thesis*, and is very widely believed to be true (or at least, it is taken to be an entirely satisfactory working hypothesis). Why? For a start, there are quasi-empirical reasons: no one has found a function which is incontrovertibly computable by a finite-step deterministic algorithmic procedure but which isn't recursive. But there are also much more principled reasons for accepting the Thesis.

Consider, for example, Turing's approach to the notion of effective computation. He famously aimed to analyse the idea of a step-by-step computation procedure down to its very basics, which led him to the concept of computation by a Turing machine (a minimalist computer). And what we can call *Turing's Thesis* is the claim that the effectively computable (total) functions are just the functions which are computable by some suitably programmed Turing machine.

So do we now have two *rival* claims, Church's and Turing's, about the class of computable functions? Not at all! For it turns out to be quite easy to prove the technical result that a function is recursive if and only if is Turing computable. And so it goes: every other attempt to give an exact characterization of the class of effectively computable functions turns out to locate just the *same* class of functions. That's remarkable, and this is a key theme you will want to explore in a first encounter with the theory of computable functions.

(e) It is fun to find out more about Turing machines, and even to learn to write a few elementary programs (in effect, it is learning to write in a 'machine code'). And there is a beautiful early result that you will soon encounter:

> There is no mechanical decision procedure which can determine whether Turing machine number e, fed a given input n, will ever halt its computation (so there is no general decision procedure which can tell whether Turing machine e in fact computes a total function).

How do we show that? Why does it matter? I leave it to you to read up on the 'undecidability of the halting problem', and its many weighty implications.

6.3 Formal arithmetic: an overview

(i) The elementary theory of computation really is a lovely area, where accessible Big Results come thick and fast! But now we must turn to consider formal theories of arithmetic.

We standardly focus on *First-order Peano Arithmetic*, PA. It will be no surprise to hear that this theory has a first-order language and logic! It has a built-in constant 0 to denote zero, has symbols for the successor, addition and multiplication functions (to keep things looking nice, we still use a prefix S, and infix $+$ and $\times$), and its quantifiers run over the natural numbers. Note, we can form the

sequence of numerals $\mathsf{0}, \mathsf{S0}, \mathsf{SS0}, \mathsf{SSS0}, \ldots$ (we will use $\overline{\mathsf{n}}$ to abbreviate the result of writing n occurrences of S before $\mathsf{0}$, so $\overline{\mathsf{n}}$ denotes n).

PA has the following three pairs of axioms governing the three built-in functions:

$$\forall \mathsf{x}\ \mathsf{0} \neq \mathsf{Sx}$$
$$\forall \mathsf{x} \forall \mathsf{y}(\mathsf{Sx} = \mathsf{Sy} \rightarrow \mathsf{x} = \mathsf{y})$$

$$\forall \mathsf{x}\ \mathsf{x} + \mathsf{0} = \mathsf{x}$$
$$\forall \mathsf{x} \forall \mathsf{y}\ \mathsf{x} + \mathsf{Sy} = \mathsf{S}(\mathsf{x} + \mathsf{y})$$

$$\forall \mathsf{x}\ \mathsf{x} \times \mathsf{0} = \mathsf{0}$$
$$\forall \mathsf{x} \forall \mathsf{y}\ \mathsf{x} \times \mathsf{Sy} = (\mathsf{x} \times \mathsf{y}) + \mathsf{x}$$

The first pair of axioms specifies that distinct numbers have distinct successors, and that the sequence of successors never circles round and ends up with zero again: so the numerals, as we want, must denote a sequence of distinct numbers, zero and all its eventual successors. The other two pairs of axioms formalize the equations defining addition and multiplication which we have met before.

And then, crucially, there is also an arithmetical Induction Rule. Recall, the familiar idea is that, if zero has property P and if a number's having P implies that its successor has P too, then *all* numbers have property P. Formally, in first-order PA, we have the rule:

> If $\varphi(\mathsf{x})$ is a formula with x free, then from $\varphi(\mathsf{0})$ and $\forall \mathsf{x}(\varphi(\mathsf{x}) \rightarrow \varphi(\mathsf{Sx}))$ we can infer $\forall \mathsf{x}\varphi(\mathsf{x})$.

Alternatively but obviously equivalently, we can stipulate that

> Any wff of the form $(\{\varphi(\mathsf{0}) \wedge \forall \mathsf{x}(\varphi(\mathsf{x}) \rightarrow \varphi(\mathsf{Sx}))\} \rightarrow \forall \mathsf{x}\varphi(\mathsf{x}))$ is an axiom.

You need to get at least some elementary familiarity with the workings of this theory.

(ii) But why concentrate on first-order PA? We've emphasized in §4.2 that our informal induction principle is most naturally construed as involving a second-order generalization – *for any arithmetical property* P, if zero has P, and if a number which has P always passes it on to its successor, then every number has P. And when Richard Dedekind (1888) and Giuseppe Peano (1889) gave their axioms for what we can call Dedekind-Peano arithmetic, they correspondingly gave a second-order formulation for their versions of the induction principle. Put it this way: Dedekind and Peano's principle quantifies over *all* properties of numbers, while in first-order PA our induction principle rather strikingly only deals with those properties of numbers which can be expressed by open formulas of its restricted language. Why go for the weaker first-order principle?

Well, as we have already noted, first-order logic is in some respects much better behaved than second-order logic (for a start, you can't capture full second-order

logic in a nice complete deductive system). And some would say that second-order logic is really just a bit of set theory in disguise. So, the argument goes, if we want a theory of pure arithmetic, one whose logic can be formalized, we should stick to a first-order formulation just quantifying over numbers. Then something like PA's induction rule (or the suite of axioms of the form we described) is the best we can do.

(iii) But still, even if we have decided to stick to a first-order theory, why restrict ourselves to the impoverished resources of PA, with only three function-expressions built into its language? Why not have an expression for e.g. the exponential functions as well, and add to the theory the two defining axioms for that function? Indeed, why not add expressions for other recursive functions too, and then also include appropriate axioms for *them* in our formal theory?

Good question. The answer is to be found in a neat technical observation first made by Gödel. Once we have successor, addition and multiplication available, plus the usual first-order logical apparatus, we can in fact *already* express any other computable (i.e. recursive) function. To take the simplest sort of case, suppose f is a one-place recursive function: then there will be a two-place expression of PA's language which we can abbreviate $\mathsf{F}(\mathsf{x},\mathsf{y})$ such that $\mathsf{F}(\overline{\mathsf{m}},\overline{\mathsf{n}})$ is true if and only if $f(m) = n$. Moreover, when $f(m) = n$, PA can *prove* $\mathsf{F}(\overline{\mathsf{m}},\overline{\mathsf{n}})$, and when $f(m) \neq n$, PA can *prove* $\neg\mathsf{F}(\overline{\mathsf{m}},\overline{\mathsf{n}})$. In this way, PA as it were already 'knows' about all the recursive functions. Similarly, PA can already express any algorithmically decidable relation.

So PA is expressively a lot richer than you might initially suppose. And indeed, it turns out that even a induction-free subsystem of PA known as Robinson Arithmetic (often called simply Q) can express the recursive functions.

And this key fact puts you in a position to link up your investigations of PA with what you know about computability. For example, there is a fairly straightforward proof that there is no mechanical procedure that a computer could implement which can decide whether a given arithmetic sentence is a theorem of PA (or even a theorem of Q).

(iv) On the other hand, despite its richness, PA is a first-order theory with infinite models, so – applying results from elementary model theory (see the previous chapter) – this first order arithmetic will have non-standard models, i.e. will have models whose domains contain more than a zero and its successors. It is worth knowing at an early stage just something about what some of these non-standard models can look like. And you will also want to further investigate the contrast with second-order versions of arithmetic which are categorical (i.e. don't have non-standard models).

6.4 Towards Gödelian incompleteness

(i) Now for our third related topic: Gödel's incompleteness theorems.

First-order PA, we said, turns out to be a very rich theory. Is it rich enough to settle every question that can be raised in its language? No! In 1931, Kurt

Gödel proved that a theory like PA must be *negation incomplete* – meaning that we can form a sentence G in its language such that PA proves neither G nor ¬G. How does he do the trick?

(ii) It's fun to give an outline sketch, which I hope will intrigue you enough to leave you wanting to find out more! So:

G1. Gödel introduces a *Gödel-numbering scheme* for a formal theory like PA, which is a simple way of coding expressions of PA – and also sequences of expressions of PA – using natural numbers. The code number for an expression (or a sequence of expressions) is its unique *Gödel number*.

G2. We can then define relations like *Prf*, where $Prf(m, n)$ holds if and only if m is the Gödel number of a PA-proof of the sentence with code number n. So *Prf* is a numerical relation which, so to speak, 'arithmetizes' the syntactic relation between a sequence of expressions (proof) and a particular sentence (conclusion).

G3. There's a procedure for computing, given numbers m and n, whether $Prf(m, n)$ holds. Informally, we just decode m (that's an algorithmic procedure). Now check whether the resulting sequence of expressions – if there is one – is a well-constructed PA-proof according to the rules of the game (proof-checking is another algorithmic procedure). If that sequence is a proof, check whether it ends with a sentence with the code number n (that's another algorithmic procedure).

G4. Since PA can express any algorithmically decidable *relation*, there will in particular be a formal expression in the language of PA which we can abbreviate $\mathsf{Prf}(\mathsf{x}, \mathsf{y})$ which expresses the relation *Prf*. This means that $\mathsf{Prf}(\overline{\mathsf{m}}, \overline{\mathsf{n}})$ is true if and only if m codes for a PA proof of the sentence with Gödel number n.

G5. Now define $\mathsf{Prov}(\mathsf{x})$ to be the expression $\exists \mathsf{z}\mathsf{Prf}(\mathsf{z}, \mathsf{x})$. Then $\mathsf{Prov}(\overline{\mathsf{n}})$, i.e. $\exists \mathsf{z}\mathsf{Prf}(\mathsf{z}, \overline{\mathsf{n}})$, is true if and only if some number Gödel-numbers a PA-proof of the wff with Gödel-number n, i.e. is true just if the wff with code number n is a theorem of PA. Therefore $\mathsf{Prov}(\mathsf{x})$ is naturally called a *provability predicate*.

G6. Next, with only a little bit of cunning, we construct a *Gödel sentence* G in the language of PA with the following property: G is true if and only if $\neg\mathsf{Prov}(\overline{\mathsf{g}})$ is true, where g is the code number of G.

Don't worry for the moment about how we do this construction (it is surprisingly easy). Just note that G is true on interpretation if and only if the sentence with Gödel number g is not a PA-theorem, i.e. if and only if G is not a PA-theorem.

In short, G *is true if and only if it isn't a* PA*-theorem*. So, rather stretching a point, it is rather as if G 'says' *I am unprovable in* PA.

G7. Now, suppose G were provable in PA. Then, since G is true if and only if it isn't a PA-theorem, G would be false. So PA would have a false theorem. Hence assuming PA is *sound* and only has true theorems, then it can't prove G. Hence, since it is not provable, G is indeed true. Which means that ¬G is false. Hence, still assuming PA is sound, it can't prove ¬G either.

So, in sum, assuming PA is sound, it can't prove either of G or ¬G. As announced, PA is negation incomplete.

Wonderful!

(iii) Now the argument generalizes to other nicely axiomatized sound theories T which can express enough arithmetical truths. We can use the same sort of cunning construction to find a true G_T such that T can prove neither G_T nor $\neg\mathsf{G}_T$. Let's be really clear: this doesn't, repeat *doesn't*, say that G_T is 'absolutely unprovable', whatever that could mean. It just says that G_T and its negation are unprovable-in-T.

Ok, you might well ask, why don't we simply 'repair the gap' in T by adding the true sentence G_T as a new axiom? Well, consider the theory $U = T + \mathsf{G}_T$ (to use an obvious notation). Then (i) U is still sound, since the old T-axioms are true and the added new axiom is true. (ii) U is still a nicely axiomatized formal theory given that T is. (iii) U can still express enough arithmetic. So we can find a sentence G_U such that U can prove neither G_U nor $\neg\mathsf{G}_U$.

And so it goes. Keep throwing more and more additional true axioms at T and our theory will remain negation-incomplete (unless it stops counting as nicely axiomatized). So here's the key take-away message: any sound nicely axiomatized theory T which can express enough arithmetic will not just be incomplete but in a good sense T will be *incompletable*.

(iv) Now, we haven't quite arrived at what's usually called the First Incompleteness Theorem. For that, we need an extra step Gödel took, which enables us to drop the semantic assumption that we are dealing with a *sound* theory T for a weaker consistency requirement. But I'll leave you to explore the (not very difficult) details, and also to find out about the Second Theorem. It really is time to start reading!

6.5 Main recommendations on arithmetic, etc.

I hope those arm-waving overviews were enough to pique your interest. But if you want a more expansive overview of the territory, then you can very usefully look at one of

1. Robert Rogers, *Mathematical Logic and Formalized Theories* (North-Holland, 1971), Chapter VIII, 'Incompleteness, Undecidability' (still quite discursive, very clear).
2. Robert S. Wolf, *A Tour Through Mathematical Logic* (Mathematical Association of America, 2005), Chapter 3, 'Recursion Theory and Computability'; and Chapter 4, 'Gödel's Incompleteness Theorems' (more

detailed, requiring more of the reader, though some students do really like this book).

But now turning to textbooks, how to approach the area? Gödel's 1931 proof of his incompleteness theorem actually uses only facts about the primitive recursive functions. As we noted, these functions are only a subclass of the effectively computable numerical functions. A more general treatment of computable functions was developed a few years later (by Gödel, Turing and others), and this in turn throws more light on the incompleteness phenomenon. So there's a choice to be made. Do you look at things in roughly the historical order, first introducing just the primitive recursive functions, explaining how they get represented in theories of formal arithmetic, and then learning how to prove initial versions of Gödel's incompleteness theorem – and only *then* move on to deal with the general theory of computable functions? Or do you explore the general theory of computation first, only turning to the incompleteness theorems later?

My own Gödel books take the first route. But I also recommend alternatives taking the second route. First, then, there is

3. Peter Smith, *Gödel Without (Too Many) Tears** (Logic Matters, 2020): freely downloadable from logicmatters.net/igt. This is a very short book – just 130 pages – which, after some general introductory chapters, and a little about formal arithmetic, explains the idea of primitive recursive functions, explains the arithmetization of syntax, and then proves Gödel's First Theorem pretty much as Gödel did, with a minimum of fuss. There follow a few chapters on closely related matters and on the Second Theorem.

GWT is, I hope, very clear and accessible, and it perhaps gives all you need for a first foray into this area if you don't want (yet) to tangle with the general theory of computation. However, you might well prefer to jump straight into one of the following:

4. Peter Smith, *An Introduction to Gödel's Theorems** (2nd edition CUP, 2013: also now downloadable from logicmatters.net/igt), is three times the length of *GWT* and ranges more widely. It starts by informally exploring various ideas such as effective computability, and then it proves two correspondingly informal versions of the first incompleteness theorem. The next part of the book gets down to work talking about formal arithmetics, developing some of the theory of primitive recursive functions, and explaining the 'arithmetization of syntax'. Then it establishes more formal versions of Gödel's first incompleteness theorem and goes on discuss the second theorem, all in more detail than *GWT*.

   The last part of the book then widens out the discussion to explore the idea of recursive functions more generally, discussing Turing machines and the Church-Turing thesis, and giving further proofs of incompleteness

(e.g. deriving it from the 'recursive unsolvability' of the halting problem for Turing machines).

5. Richard Epstein and Walter Carnielli, *Computability: Computable Functions, Logic, and the Foundations of Mathematics* (Wadsworth 2nd edn. 2000: Advanced Reasoning Forum 3rd edn. 2008) is an excellent introductory book on the standard basics, particularly clearly and attractively done. Part I, on 'Fundamentals', covers some background material, e.g. on the idea of countable sets (many readers will be able to speed-read through these initial chapters). Part II, on 'Computable Functions', comes at them two ways: first via Turing Machine computability, and then via primitive recursive and then partial recursive functions, ending with a proof that the two approaches define the same class of effectively computable functions. Part III, 'Logic and Arithmetic', turns to formal theories of arithmetic and the way that the representable functions in a formal arithmetic like Robinson's Q turn out to be the recursive ones. Formal arithmetic is then shown to be undecidable, and Gödelian incompleteness derived. The shorter Part IV has a chapter on Church's Thesis (with more discussion than is often the case), and finally a chapter on constructive mathematics. There are many interesting historical asides along the way.

Those two books should be very accessible to those without much mathematical background: but even more experienced mathematicians should appreciate the careful introductory orientation which they provide. Then next, taking us half-a-step up in mathematical sophistication, we arrive at a quite delightful book:

6. George Boolos and Richard Jeffrey, *Computability and Logic* (CUP 3rd edn. 1990). This is a modern classic, wonderfully lucid and engaging, admired by generations of readers. Indeed, looking at it again in revising this Guide, I couldn't resist some re-reading! It starts with a exploration of Turing machines, 'abacus computable' functions, and recursive functions (showing that different definitions of computability end up characterizing the same class of functions). And then it moves on discuss logic and formal arithmetic (with interesting discussions ranging beyond what is covered in my book or E&C).

   There are in fact two later editions – heavily revised and considerably expanded – with John Burgess as a third author. But I know that I am not the only one to think that these later versions (good though they are) do lose something of the original book's famed elegance and individuality and distinctive flavour. Still, whichever edition comes to hand, do read it! – you will learn a great deal in an enjoyable way.

One comment: none of these books – including my longer one – gives a full proof of Gödel's Second Incompleteness Theorem. The guiding idea is easy enough, but there is very tedious work to be done in implementing it. If you *really* want all the details, see one of the relevant recommendations in Part III.

6.6 Some parallel/additional reading

I should mention a more elementary book which might well appeal to some for its debunking of myths about the wider significance of Gödelian incompleteness:

7. Torkel Franzén, *Gödel's Theorem: An Incomplete Guide to its Use and Abuse* (A. K. Peters, 2005). John Dawson (who we'll meet again below) writes "Among the many expositions of Gödel's incompleteness theorems written for non-specialists, this book stands apart. With exceptional clarity, Franzén gives careful, non-technical explanations both of what those theorems say and, more importantly, what they do not. No other book aims, as his does, to address in detail the misunderstandings and abuses of the incompleteness theorems that are so rife in popular discussions of their significance. As an antidote to the many spurious appeals to incompleteness in theological, anti-mechanist and post-modernist debates, it is a valuable addition to the literature." Invaluable, in fact!

And next, here's a group of three books at about the same level as those mentioned in the previous section. First, the Open Logic Project now has a good volume on our topics:

8. Jeremy Avigad and Richard Zach, *Incompleteness and Computability: An Open Introduction to Gödel's Theorems*, tinyurl.com/icomp-open.

   Chapters 1 to 5 are on computability and Gödel, covering a good deal in just 120 *very* sparsely printed pages. Avigad and Zach are admirably clear as far as they go – though inevitably, given the length, they have to go pretty briskly. But this could be enough for those who want a short first introduction. And others could well find this very useful revision material, highlighting some basic main themes.

Still, I'd certainly recommend taking a slower tour through more of the sights by following the recommendations in the previous section, or by reading the following excellent book that could well have been an alternative main recommendation:

9. Herbert E. Enderton's relatively short book *Computability Theory: An Introduction to Recusion Theory* (Associated Press, 2011). This is written with attractive zip and lightness of touch (this is a notably more relaxed book than his earlier *Logic*). The first chapter is on the informal Computability Concept. There are then chapters on general recursive functions and on register machines (showing that the register-computable functions are exactly the recursive ones), and a chapter on recursive enumerability. Chapter 5 makes 'Connections to Logic' (including proving Tarski's theorem on the undefinability of arithmetical truth and a semantic incompleteness theorem). The final two chapters push on to say something about 'Degrees of Unsolvability' and 'Polynomial-time Computability'. This is all very nicely and accessibly done.

This book, then, makes an excellent alternative to Epstein & Carnielli in particular: it is, however, a little more abstract and sophisticated, which why I have on balance recommended E&C for many readers. The more mathematical might well prefer Enderton. (By the way, staying with Enderton, I should mention that Chapter 3 of his earlier *A Mathematical Introduction to Logic* (Academic Press 1972, 2002) gives a good brisk treatment of different strengths of formal theories of arithmetic, and then proves the incompleteness theorem first for a formal arithmetic with exponentiation and then – after touching on other issues – shows how to use the β-function trick to extend the theorem to apply to arithmetic without exponentiation. Not the best place to start, but this chapter too could be very useful revision material.)

Thirdly, I have already warmly recommended the following book for its coverage of first-order logic:

10. Christopher Leary and Lars Kristiansen's *A Friendly Introduction to Mathematical Logic**, tinyurl.com/friendlylogic. Chapters 4 to 7 now give a very illuminating double treatment of matters related to incompleteness (you don't have to have read the previous chapters in this book to follow the later ones, other than noting the arithmetical system N introduced in their §2.8). In headline terms that you'll only come fully to understand in retrospect:

    a) L&K's first approach doesn't go overtly via computability. Instead of showing that certain syntactic properties are primitive recursive and showing that all primitive recursive properties can be 'represented' in theories like N (as I do in *IGT*), L&K rely on more directly showing that some key syntactic properties can be represented. This representation result then leads to, inter alia, the incompleteness theorem.

    b) L&K follow this, however, with a general discussion of computability, and then use the introductory results they obtain to prove various further theorems, including incompleteness again.

    This is all presented with the same admirable clarity as the first part of the book on FOL.

There are, of course, many other more-or-less introductory treatments covering aspects of computability and/or incompleteness, and we will return to the topic in Part III of this Guide. For now, I will mention just three further, and rather more individual, books.

First, of the relevant texts in American Mathematical Society's 'Student Mathematical Library', by far the best is

11. A. Shen and N. K. Vereshchagin, *Computable Functions*, (AMA, 2003). This is a lovely, elegant, little book, which can be recommended for giving a differently-structured quick tour through some of the Big Ideas. Well worth reading as a follow-up to a more conventional text.

Next we come to a stand-out book that you should certainly tackle at some point (though I rather suspect that many readers will appreciate it more if they come to it after reading one or more of the main recommendations in the previous section):

12. Raymond Smullyan, *Gödel's Incompleteness Theorems*, Oxford Logic Guides 19 (Clarendon Press, 1992). This is delightully short – under 140 pages – proving some rather beautiful, slightly abstract, versions of the incompleteness theorems. This is a modern classic which anyone with a taste for mathematical elegance will find very rewarding.

To introduce the third book, the first thing to say is that it presupposes *very* little knowledge about sets, despite the title. If you are familiar with the idea that the natural *numbers* can be identified with (implemented as) *finite sets* in a standard way, and with a few other low-level ideas, then you can dive in without further ado to

13. Melvin Fitting's, *Incompleteness in the Land of Sets** (College Publications, 2007). This is a very engaging read, approaching the incompleteness theorem and related results in an unusual but illuminating way. From the book's blurb: "Russell's paradox arises when we consider those sets that do not belong to themselves. The collection of such sets cannot constitute a set. Step back a bit. Logical formulas define sets (in a standard model). Formulas, being mathematical objects, can be thought of as sets themselves – mathematics reduces to set theory. Consider those formulas that do not belong to the set they define. The collection of such formulas is not definable by a formula, by the same argument that Russell used. This quickly gives Tarski's result on the undefinability of truth. Variations on the same idea yield the famous results of Gödel, Church, Rosser, and Post.

    This book gives a full presentation of the basic incompleteness and undecidability theorems of mathematical logic in the framework of set theory. Corresponding results for arithmetic follow easily, and are also given. Gödel numbering is generally avoided, except when an explicit connection is made between set theory and arithmetic. The book assumes little technical background from the reader. One needs mathematical ability [and] a general familiarity with formal logic ..."

And, finally, if only because I've been asked about it such a large number of times, I suppose I should end by also mentioning the (in)famous

14. Douglas Hofstadter, *Gödel, Escher, Bach** (Penguin, first published 1979). When students enquire about this, I helpfully say that it is the sort of book that you will probably really like if you like this kind of book, and you won't if you don't. It is, to say the very least, quirky, idiosyncratic and entirely distinctive. However, as I far as I recall, the parts of the

book which touch on techie logical things are in fact pretty reliable and won't lead you astray.

Which is a great deal more than can be said about many popularizing treatments of Gödel's theorems!

6.7 A little history?

If you haven't already done so, *do* read

15. Richard Epstein's brisk and very helpful 28 page 'Computability and Undecidability – A Timeline' which is printed at the very end of Epstein & Carnielli, listed in §6.5.

This will really give you the headline news you initially need. Enthusiasts can find a little more detail about the early days in e.g. Rod Adams's *An Early History of Recursive Functions and Computability** (Docent Press, 2011). But it is a lot more interesting to read

16. John Dawson, *Logical Dilemmas: The Life and Work of Kurt Gödel* (A. K. Peters, 1997).

Not, perhaps, as lively as the Fefermans' biography of Tarski which I mentioned in §5.4 – but then Gödel was such a very different man. Fascinating, though! (As far as getting any logical insights goes, you can ignore the third-rate book by Stephen Budiansky *Journey to the Edge of Reason: The Life of Kurt Gödel*, OUP 2021).