

What follows is a draft of the opening chapters for the planned second edition of my *An Introduction to Gödel's Theorems* (first published by CUP in 2007, with a number of corrected reprints since).

*For those who already know the book* What was 50 pages and 7 chapters – up to and including the first ‘Interlude’ – has become 56 pages and 8 chapters. There has been some significant rearrangement of material (including bringing forward a couple of ideas which had previously been, perhaps unwisely, left to later chapters). The main change/improvement, however, is that the compressed and unnecessarily complex incompleteness argument of what was Ch. 5 (which caused students trouble) has been disentangled. The key result that there are effectively enumerable sets of numbers whose complements aren’t effectively enumerable now gets properly highlighted, with a simpler proof, in the new separate Ch. 3, with the rest of the incompleteness argument in what is now Ch. 6.

*For those who are new to the book* The first edition was published in a series called ‘Cambridge Introductions to Philosophy’. This was a Good Thing because it has meant that a 360 page, larger format, text has been available as a relatively cheap paperback. But this book, unlike others in the series, isn’t *that* elementary. As I said in the Preface, it is for those who want a lot more fine detail than you get in introductory books for a general audience, but who find the rather forbidding presentations in classic texts in mathematical logic too short on explanatory scene-setting. So I hope philosophy students taking an advanced logic course will find the book useful, as will mathematicians who want a more accessible exposition.

*Comments please!* Apart from advertising the second edition (but don’t hold your breath!), the main reason for making these initial chapters available for download from [www.logicmatters.net](http://www.logicmatters.net) is to prompt your comments, corrections, and suggestions: please send them to [peter\\_smith@me.com](mailto:peter_smith@me.com)

For obvious copyright reasons, I won’t be able to make later tranches of the second edition so freely available like this. On the other hand, I’d enormously welcome comments on those later chapters too when they become available. So if you send me any (sensible!) comments on these initial chapters, I’ll put you on the circulation list to let you know when later chapters are restrictedly online for comment: and I’ll consider other requests from academic e-mail addresses to be put on the list. Thanks in advance!

Peter Smith  
1 January 2012

*This page intentionally blank*

# 1 What Gödel's Theorems say

## 1.1 Basic arithmetic

It is child's play to grasp the fundamental notions involved in the arithmetic of addition and multiplication. Starting from zero, there is a sequence of 'counting' numbers, each having exactly one immediate successor. This sequence of numbers – officially, *the natural numbers* – continues without end, never circling back on itself; and there are no 'stray' natural numbers, lurking outside this sequence. Adding  $n$  to  $m$  is the operation of starting from  $m$  in the number sequence and moving  $n$  places along. Multiplying  $m$  by  $n$  is the operation of (starting from zero and) repeatedly adding  $m$ ,  $n$  times. It's as simple as that.

Once these fundamental notions are in place, we can readily define many more arithmetical concepts in terms of them. Thus, for any natural numbers  $m$  and  $n$ ,  $m < n$  iff there is a number  $k \neq 0$  such that  $m + k = n$ .  $m$  is a factor of  $n$  iff  $0 < m$  and there is some number  $k$  such that  $0 < k$  and  $m \times k = n$ .  $m$  is even iff it has 2 as a factor.  $m$  is prime iff  $1 < m$  and  $m$ 's only factors are 1 and itself. And so on.<sup>1</sup>

Using our basic and defined concepts, we can then frame various general claims about the arithmetic of addition and multiplication. There are obvious truths like 'addition is commutative', i.e. for any numbers  $m$  and  $n$ ,  $m + n = n + m$ . There are also some very unobvious claims, yet to be proved, like Goldbach's conjecture that every even number greater than two is the sum of two primes.

That second example illustrates the truism that it is one thing to understand what we'll call *the language of basic arithmetic* (i.e. the language of the addition and multiplication of natural numbers, together with the standard first-order logical apparatus), and it is quite another thing to be able to evaluate claims that can be framed in that simple language.

Still, it is extremely plausible to suppose that, whether the answers are readily available to us or not, questions posed in the language of basic arithmetic do *have* entirely determinate answers. The structure of the natural number sequence, with each number having a unique successor and there being no repetitions, is (surely) simple and clear. The operations of addition and multiplication are (surely) entirely well-defined; their outcomes are fixed by the school-room rules. So what more could be needed to fix the truth or falsity of propositions that – perhaps via a chain of definitions – amount to claims of basic arithmetic?

To put it fancifully: God lays down the number sequence and specifies how the operations of addition and multiplication work. He has then done all he needs

---

<sup>1</sup>'Iff' is, of course, the standard logicians' shorthand for 'if and only if'.

## 1 What Gödel's Theorems say

---

to do to make it the case e.g. that Goldbach's conjecture is true (or false, as the case may be).

Of course, that way of putting it is rather too fanciful for comfort. We may indeed find it compelling to think that the sequence of natural numbers has a definite structure, and that the operations of addition and multiplication are entirely nailed down by the familiar basic rules. But what is the real content of the thought that the truth-values of all basic arithmetic propositions are thereby 'fixed'?

Here's one appealing way of giving non-metaphorical content to that thought. The idea is that we can specify a bundle of fundamental assumptions or *axioms* which somehow pin down the structure of the natural number sequence, and which also characterize addition and multiplication (after all, it is pretty natural to suppose that we *can* give a reasonably simple list of true axioms to encapsulate the fundamental principles so readily grasped by the successful learner of school arithmetic). So now suppose that  $\varphi$  is a proposition which can be formulated in the language of basic arithmetic. Then, the appealing suggestion continues, the assumed truth of our axioms 'fixes' the truth-value of any such  $\varphi$  in the following sense: either  $\varphi$  is logically deducible from the axioms, and so  $\varphi$  is true; or its negation  $\neg\varphi$  is deducible from the axioms, and so  $\varphi$  is false. We may not, of course, actually stumble on a proof one way or the other: but the proposal is that such a proof is always possible, since the axioms contain enough information to enable the truth-value of any basic arithmetical proposition to be deductively extracted by deploying familiar step-by-step logical rules of inference.

Logicians say that a theory  $T$  is (*negation*) *complete* if, for every sentence  $\varphi$  in the language of the theory, either  $\varphi$  or  $\neg\varphi$  is deducible in  $T$ 's proof system. So, put into that jargon, the suggestion we are considering is this: we should be able to specify a reasonably simple bundle of true axioms which, together with some logic, give us a *complete* theory of basic arithmetic: i.e. we could in principle use the theory to prove or disprove any claim which is expressible in the language of basic arithmetic. If that's right, truth in basic arithmetic could just be equated with provability in this complete theory.

It is tempting to say rather more. For what will the axioms of basic arithmetic look like? Here's one candidate: 'For every natural number, there's a unique next one'. This is evidently true: but evident *how*? As a first thought, you might say 'we can just see, using mathematical intuition, that this axiom is true'. But the idea of mathematical intuition is obscure, to say the least. Maybe, on second thoughts, we don't need to appeal to it. Perhaps the axiom is evidently true because it is some kind of definitional triviality. Perhaps it is just part of what we *mean* by talk of the natural numbers that we are dealing with an ordered sequence where each member of the sequence has a unique successor. And, plausibly, other candidate axioms are similarly true by definition.

If those tempting second thoughts are right, then true arithmetical claims are *analytic* in the philosophers' sense of the word: that is to say, the truths of basic arithmetic will all flow deductively from logic plus axioms which are trivially

true-by-definition.<sup>2</sup> This so-called ‘logician’ view would then give us a very neat explanation of the special certainty and the necessary truth of correct claims of basic arithmetic.

## 1.2 Incompleteness

But now, in headline terms, *Gödel’s First Incompleteness Theorem shows that the entirely natural idea that we can completely axiomatize basic arithmetic is wrong.*

Suppose we try to specify a suitable axiomatic theory  $T$  to capture the structure of the natural number sequence and pin down addition and multiplication (and maybe a lot more besides). We want  $T$  to have true axioms and a reliably truth-preserving deductive logic. In that case, everything  $T$  proves must be true, i.e.  $T$  is a *sound* theory. But now Gödel gives us a recipe for coming up with a corresponding sentence  $G_T$ , couched in the language of basic arithmetic, such that – assuming  $T$  really is sound – (i) we can show that  $G_T$  can’t be derived in  $T$ , and yet (ii) we can recognize that  $G_T$  must be true.

This is surely quite astonishing. Somehow, it seems, the truths of basic arithmetic must elude our attempts to pin them down by giving a fixed set of fundamental assumptions from which we can deduce everything else. So how does Gödel show this in his great 1931 paper which presents the Incompleteness Theorems?

Well, note how we can use numbers and numerical propositions to encode facts about all sorts of things. For a trivial example, students in the philosophy department might be numbered off in such a way that the first digit encodes information about whether a student is an undergraduate or postgraduate, the next two digits encode year of admission, and so on. Much more excitingly, Gödel notes that we can use numbers and numerical propositions to encode facts about *theories*, e.g. facts about what can be derived in a theory  $T$ .<sup>3</sup> And what he then did is find a general method that enabled him to take any theory  $T$  strong enough to capture a modest amount of basic arithmetic and construct a corresponding arithmetical sentence  $G_T$  which encodes the claim ‘The sentence  $G_T$  itself is unprovable in theory  $T$ ’. So  $G_T$  is true if and only if  $T$  can’t prove it.

<sup>2</sup>Thus Gottlob Frege, writing in his wonderful *Grundlagen der Arithmetik*, urges us to seek the proof of a mathematical proposition by ‘following it up right back to the primitive truths. If, in carrying out this process, we come only on general logical laws and on definitions, then the truth is an analytic one.’ (Frege, 1884, p. 4)

<sup>3</sup>By the way, it is absolutely standard for logicians to talk of a theory  $T$  as *proving* a sentence  $\varphi$  when there is a logically correct derivation of  $\varphi$  from  $T$ ’s assumptions. But  $T$ ’s assumptions may be contentious or plain false or downright absurd. So,  $T$ ’s proving  $\varphi$  in this logician’s sense does not mean that  $\varphi$  is proved in the sense that it is established as true. It is far too late in the game to kick against the logician’s usage, and in most contexts it is harmless. But our special concern in this book is with the connections and contrasts between being true and being provable in this or that theory  $T$ . So we need to be on our guard. And to help emphasize that proving-in- $T$  is not always proving-as-true, I’ll often talk of ‘deriving’ rather than ‘proving’ sentences when it is the logician’s notion which is in play.

## 1 What Gödel's Theorems say

---

Suppose then that  $T$  is sound. If  $T$  were to prove  $G_T$ ,  $G_T$  would be false, and  $T$  would then prove a falsehood, which it can't do. Hence, if  $T$  is sound,  $G_T$  is unprovable in  $T$ . Which makes  $G_T$  *true*. Hence  $\neg G_T$  is false. And so that too can't be proved by  $T$ , because  $T$  only proves truths. In sum, still assuming  $T$  is sound, neither  $G_T$  nor its negation will be provable in  $T$ . Therefore  $T$  can't be negation complete.

But in fact we don't need to assume that  $T$  is *sound*; we can make do with very significantly less. Gödel's official version of the First Theorem shows that  $T$ 's mere *consistency* is enough to guarantee that a suitably constructed  $G_T$  is true-but-unprovable-in- $T$ . And we only need a little more to show that  $\neg G_T$  is not provable either (we won't pause now to fuss about the needed extra assumption).

We said: the sentence  $G_T$  encodes the claim that that very sentence is unprovable. But doesn't this make  $G_T$  rather uncomfortably reminiscent of the Liar sentence 'This very sentence is false' (which is false if it is true, and true if it is false)? You might well wonder whether Gödel's argument doesn't lead to a cousin of the Liar paradox rather than to a theorem. But not so. As we will see, there really is nothing at all suspect or paradoxical about Gödel's First Theorem as a technical result about formal axiomatized systems (a result which in any case can be proved without appeal to 'self-referential' sentences).

'Hold on! If we locate  $G_T$ , a Gödel sentence for our favourite nicely axiomatized sound theory of arithmetic  $T$ , and can argue that  $G_T$  is true-though-unprovable-in- $T$ , why can't we just patch things up by adding it to  $T$  as a new axiom?' Well, to be sure, if we start off with the sound theory  $T$  (from which we can't deduce  $G_T$ ), and add  $G_T$  as a new axiom, we'll get an expanded sound theory  $U = T + G_T$  from which we *can* quite trivially derive  $G_T$ . But we can now just re-apply Gödel's method to our improved theory  $U$  to find a new true-but-unprovable-in- $U$  arithmetic sentence  $G_U$  that encodes 'I am unprovable in  $U$ '. So  $U$  again is incomplete. Thus  $T$  is not only incomplete but, in a quite crucial sense, is *incompletable*.

Let's emphasize this key point. There's nothing at all mysterious about a theory's failing to be negation complete, plain and simple. Imagine the departmental administrator's 'theory'  $D$  which records some basic facts about the course selections of a group of students: the language of  $D$ , let's suppose, is very limited and can only be used to tell us about who takes what course in what room when. From the 'axioms' of  $D$  we'll be able, let's suppose, to deduce further facts – such as that Jack and Jill take a course together, and that ten people are taking the advanced logic course. But if there's currently no relevant axiom in  $D$  about their classmate Jo, we might not be able to deduce either  $J =$  'Jo takes logic' or  $\neg J =$  'Jo doesn't take logic'. In that case,  $D$  isn't yet a negation-complete story about the course selections of students.

However, that's just boring: for the 'theory' about course selection is no doubt completable (i.e. it can readily be expanded to settle every question that can be posed in its very limited language). By contrast, what gives Gödel's First Theorem its real bite is that it shows that any properly axiomatized and sound

theory of basic arithmetic must *remain* incomplete, however many new true axioms we give it. (And again, we can weaken the soundness condition, and can – more or less – just require consistency for incompleteness.)

### 1.3 More incompleteness

Incompleteness does not just affect theories of basic arithmetic. Consider set theory, for example. Start with the empty set  $\emptyset$ . Form the set  $\{\emptyset\}$  containing  $\emptyset$  as its sole member. Then form the set containing the empty set we started off with plus the set we’ve just constructed. Keep on going, at each stage forming the set of all the sets so far constructed. We get the sequence

$$\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, \{\emptyset, \{\emptyset, \{\emptyset, \{\emptyset\}\}\}, \dots$$

This sequence has the structure of the natural numbers. We can pick out a first member (corresponding to zero); each member has one and only one successor; it never repeats. We can go on to define analogues of addition and multiplication. Now, any standard set theory allows us to define this sequence. So if we could have a negation-complete and sound axiomatized set theory, then we could, in particular, have a negation-complete theory of the fragment of set theory which provides us with an analogue of arithmetic. Adding a simple routine for translating the results for this fragment into the familiar language of basic arithmetic would then give us a complete sound theory of arithmetic. But Gödel’s First Incompleteness Theorem tells us there can’t be such a theory. So there cannot be a sound negation-complete set theory.

The point evidently generalizes: any sound axiomatized mathematical theory  $T$  that can define (an analogue of) the natural-number sequence and replicate enough of the basic arithmetic of addition and multiplication must be incomplete and incomplete.<sup>4</sup>

### 1.4 Some implications?

Gödelian incompleteness immediately defeats what otherwise looked a really rather attractive suggestion about the status of basic arithmetic – namely the logicist idea that it all flows deductively from a simple bunch of definitional truths that articulate the very ideas of the natural numbers, addition and multiplication.

But then, how *do* we manage somehow to latch on to the nature of the unending number sequence and the operations of addition and multiplication in a way that outstrips whatever rules and principles can be captured in definitions? At this point it can begin to seem that we must have a rule-transcending cognitive grasp of the numbers which underlies our ability to recognize certain ‘Gödel

<sup>4</sup>We return to discuss this point more carefully in Section 19.2.

## 1 What Gödel's Theorems say

---

sentences' as correct arithmetical propositions. And if you are tempted to think so, then you may well be further tempted to conclude that minds such as ours, capable of such rule-transcendence, can't be machines (supposing, reasonably enough, that the cognitive operations of anything properly called a machine can be fully captured by rules governing the machine's behaviour).

So already there's apparently a quick route from reflections about Gödel's First Theorem to some conclusions about the nature of arithmetical truth and the nature of the minds that grasp it. Whether those conclusions really follow will emerge later. For the moment, we have an initial idea of what the First Theorem says and why it might matter – enough, I hope, already to entice you to delve further into the story that unfolds in this book.

### 1.5 The unprovability of consistency

If we can derive even a modest amount of basic arithmetic in theory  $T$ , then we'll be able to derive  $0 \neq 1$ .<sup>5</sup> So if  $T$  *also* proves  $0 = 1$ , it is inconsistent. Conversely, if  $T$  is inconsistent, then – since we can derive anything in an inconsistent theory<sup>6</sup> – it can prove  $0 = 1$ . But we said that we can use numerical propositions to encode facts about what can be derived in  $T$ . So there will in particular be a *numerical* proposition  $\text{Con}_T$  that encodes the claim that we can't derive  $0 = 1$  in  $T$ , i.e. encodes in a natural way the claim that  $T$  is consistent.

We know, however, that there is a numerical proposition which encodes the claim that  $\text{G}_T$  is unprovable: we have already said that it is  $\text{G}_T$  itself.

So this means that (half of) the conclusion of Gödel's official First Theorem, namely the claim that if  $T$  is consistent then  $\text{G}_T$  is unprovable, can *itself* be encoded by a numerical proposition, namely  $\text{Con}_T \rightarrow \text{G}_T$ . And now for another wonderful Gödelian insight. It turns out that the informal reasoning that we use, outside  $T$ , to show 'if  $T$  is consistent, then  $\text{G}_T$  is unprovable' is elementary enough to be mirrored by reasoning inside  $T$  (i.e. by reasoning with numerical propositions which encode facts about  $T$ -proofs). Or at least that's true so long as  $T$  satisfies conditions only slightly stronger than the official First Theorem assumes. So, again on modest assumptions, we can derive  $\text{Con}_T \rightarrow \text{G}_T$  inside  $T$ .

But the official First Theorem has already shown that if  $T$  is consistent we can't derive  $\text{G}_T$  in  $T$ . So it immediately follows that if  $T$  is consistent it can't prove  $\text{Con}_T$ . *And that is Gödel's Second Incompleteness Theorem*. Roughly interpreted: nice theories that include enough basic arithmetic can't prove their own consistency.<sup>7</sup>

---

<sup>5</sup>We'll allow ourselves to abbreviate expressions of the form  $\neg\sigma = \tau$  as  $\sigma \neq \tau$ .

<sup>6</sup>There are, to be sure, deviant non-classical logics in which this principle doesn't hold. In this book, however, we aren't going to say much more about them, if only because of considerations of space.

<sup>7</sup>That *is* rough. The Second Theorem shows that  $T$  can't prove  $\text{Con}_T$ , which is certainly *one* natural way of expressing  $T$ 's consistency inside  $T$ . But couldn't there be some *other* sentence,  $\text{Con}'_T$ , which also in some good sense expresses  $T$ 's consistency, where  $T$  *does* prove

## 1.6 More implications?

Suppose that there's a genuine issue about whether  $T$  is consistent. Then even before we'd ever heard of Gödel's Second Theorem, we wouldn't have been convinced of its consistency by a derivation of  $\text{Con}_T$  inside  $T$ . For we'd just note that if  $T$  were in fact inconsistent, we'd be able to derive any  $T$ -sentence we like in the theory – including a false statement of its own consistency!

The Second Theorem now shows that we would indeed be right not to trust a theory's announcement of its own consistency. For (assuming  $T$  includes enough arithmetic), if  $T$  entails  $\text{Con}_T$ , then the theory must in fact be *inconsistent*.

However, the real impact of the Second Theorem isn't in the limitations it places on a theory's proving its own consistency. The key point is this. If a nice arithmetical theory  $T$  can't even prove *itself* to be consistent, it certainly can't prove that a *richer* theory  $T^+$  is consistent (since if the richer theory is consistent, then any cut-down part of it is consistent). Hence we can't use 'safe' reasoning of the kind we can encode in ordinary arithmetic to prove that other more 'risky' mathematical theories are in good shape. For example, we can't use unproblematic arithmetical reasoning to convince ourselves of the consistency of set theory (with its postulation of a universe of wildly infinite sets).

And *that* is a very interesting result, for it seems to sabotage what is called Hilbert's Programme, which is precisely the project of trying to defend the wilder reaches of infinitistic mathematics by giving consistency proofs which use only 'safe' methods. A great deal more about this in due course.

## 1.7 What's next?

What we've said so far, of course, has been extremely sketchy and introductory. We must now start to do better. After preliminaries in Chapter 2, we go on in Chapter 3 to introduce the notions of *effective* computability, decidability and enumerability, notions we are going to need in what follows. Then in Chapter 4, we explain more carefully what we mean by talking about an 'axiomatized theory' and prove some elementary results about axiomatized theories in general. In Chapter 5, we introduce some concepts relating specifically to axiomatized theories of arithmetic. Then in Chapters 6 and 7 we prove a pair of neat and relatively easy results – first that any sound and 'sufficiently expressive' axiomatized theory of arithmetic is negation incomplete, and then similarly for any consistent and 'sufficiently strong' axiomatized theory. For reasons that we'll explain in Chapter 8, these informal results fall some way short of Gödel's own First Incompleteness Theorem. But they do provide a very nice introduction to some key ideas that we'll be developing more formally in the ensuing chapters.

---

$\text{Con}'_T$  (and we avoid trouble because  $T$  doesn't prove  $\text{Con}'_T \rightarrow \text{G}_T$ )? We'll return to this question in Sections 25.5 and 28.2.

## 2 Functions and enumerations

We start by fixing some entirely standard notation and terminology for talking about functions (you should know this anyway, quite apart from the use we make of it in coming chapters). We next introduce the useful little notion of a ‘characteristic function’. Then we explain the idea of enumerability and give our first example of a ‘diagonalization argument’ – an absolutely crucial type of argument which will feature repeatedly in this book. Many readers, however, will find themselves able to skip lightly through rather familiar material.

### 2.1 Kinds of function

(a) Functions, and in particular functions from natural numbers to natural numbers, will feature pivotally in everything that follows.

Note though that our concern will be with *total* functions. i.e. functions which map each and every element of their *domain* to some unique corresponding value in their *codomain*.

For certain wider mathematical purposes, especially in the broader theory of computation, the more general idea of a *partial* function can take centre stage. This is a mapping  $f$  which does not necessarily have an output for each argument in its domain (for an obvious though trivial example, consider the reciprocal function  $1/x$  for rational numbers, which is not defined for  $x = 0$ ). However, we won’t need to say much about partial functions in this book, and hence – by default – plain ‘function’ will henceforth always mean ‘total function’.

(b) The conventional notation to indicate that the one-place total function  $f$  maps elements of the domain  $\Delta$  to values in the codomain  $\Gamma$  is, of course,  $f: \Delta \rightarrow \Gamma$ . Let  $f$  be such a function. Then we say:

1. The *range* of  $f$  is  $\{f(x) \mid x \in \Delta\}$ , i.e. the set of elements in  $\Gamma$  that are values of  $f$  for arguments in  $\Delta$ . Note, the range of a function need not be the whole codomain.
2.  $f$  is *surjective* iff the range of  $f$  indeed *is* the whole codomain  $\Gamma$  – i.e. if for every  $y \in \Gamma$  there is some  $x \in \Delta$  such that  $f(x) = y$ . (If you prefer that in English, you can say that such a function is *onto*, since it maps  $\Delta$  onto the whole of  $\Gamma$ .)
3.  $f$  is *injective* iff  $f$  maps different elements of  $\Delta$  to different elements of  $\Gamma$  – i.e. if  $x \neq y$  then  $f(x) \neq f(y)$ . (If you prefer that in English, you can say that such a function is *one-to-one*.)

4.  $f$  is *bijective* if it is both surjective and injective. (In English again,  $f$  is then a *one-one correspondence* between  $\Delta$  and  $\Gamma$ .)<sup>1</sup>

These definitions generalize in natural ways to many-place functions that map two or more objects to values: but we needn't pause over this.

(c) Our special concern, we said, is going to be with numerical functions. It is conventional to use ' $\mathbb{N}$ ' for the set of natural numbers. So  $f: \mathbb{N} \rightarrow \mathbb{N}$  is a one-place total function, defined for all natural numbers, with number values. While  $c: \mathbb{N} \rightarrow \{0, 1\}$ , for example, is a one-place numerical function whose values are restricted to 0 and 1.

' $\mathbb{N}^2$ ' standardly denotes the set of ordered pairs of numbers, so  $f: \mathbb{N} \rightarrow \mathbb{N}^2$  is a one-place function that maps numbers to ordered pairs of numbers. Note,  $g: \mathbb{N}^2 \rightarrow \mathbb{N}$  is another *one*-place function which this time maps an ordered pair (which is one thing!) to a number. If we want to indicate a function like addition which maps *two* numbers to a number, we really need a notation such as  $f: \mathbb{N}, \mathbb{N} \rightarrow \mathbb{N}$ . But again, that's something we needn't pause over here.

## 2.2 Characteristic functions

As well as talking about numerical *functions*, we will also be talking a lot about numerical *properties* and *relations*. But discussion of these can be tied back to discussion of functions using the following idea:

The *characteristic function* of the numerical property  $P$  is the one-place function  $c_P: \mathbb{N} \rightarrow \{0, 1\}$  such that if  $n$  is  $P$ , then  $c_P(n) = 0$ , and if  $n$  isn't  $P$ , then  $c_P(n) = 1$ .

The characteristic function of the two-place numerical relation  $R$  is the two-place function  $c_R: \mathbb{N}, \mathbb{N} \rightarrow \{0, 1\}$  such that if  $m$  is  $R$  to  $n$ , then  $c_R(m, n) = 0$ , and if  $m$  isn't  $R$  to  $n$ , then  $c_R(m, n) = 1$ .

The notion evidently generalizes to many-place relations in the obvious way.

The choice of values for the characteristic function is, of course, entirely arbitrary: any pair of distinct objects would do as the set of values. Our choice is supposed to be reminiscent of the familiar use of 0 and 1, one way round or the other, to stand in for *true* and *false*. And our (somewhat less usual) selection of 0 rather than 1 for *true* is merely for later neatness.

Now, the numerical property  $P$  partitions the numbers into two sets, the set of numbers that have the property and the set of numbers that don't. Its corresponding characteristic function  $c_P$  also partitions the numbers into two sets, the set of numbers the function maps to the value 0, and the set of numbers the function maps to the value 1. And these are of course exactly the *same* partition both times. So in a good sense,  $P$  and its characteristic function  $c_P$

<sup>1</sup>If these notions really *are* new to you, it will help to look at the exercises.

## 2 Functions and enumerations

---

*encapsulate just the same information about a partition.* That's why we can typically move between talk of a property and talk of its characteristic function without loss of relevant information. Similarly, of course, for relations.

We'll be making use of characteristic functions a lot, starting in the next chapter. But the rest of this chapter discusses something else, namely ...

### 2.3 Enumerable sets

Suppose that  $\Sigma$  is some set of items: its members might be natural numbers, computer programs, infinite binary strings, complex numbers or whatever. Then, as a suggestive first shot, we can say:

The set  $\Sigma$  is *enumerable* iff its members can – at least in principle – be listed off in some numerical order (a zero-th, first, second, ...) with every member appearing on the list; repetitions are allowed, and the list may be infinite.

It is tidiest to think of the empty set as the limiting case of an enumerable set: after all, it is enumerated by the empty list.

One issue with this rough definition is that, if we are literally to 'list off' elements of  $\Sigma$ , then we need to be dealing with elements which are either things that themselves can be written down (like finite strings of symbols), or which at least have standard representations that can be written down (in the way that natural numbers have numerals which denote them). That condition will be satisfied in the cases that most interest us in this book; but we really want the idea of enumerability to apply more widely.

A more immediate problem is that it is of course careless to talk about 'listing off' *infinite* sets as if we can complete the job. What we really mean is that any member of  $\Sigma$  will eventually appear on this list, if we go on long enough.

Let's give a more rigorous definition, then, that doesn't presuppose that we have a way of writing down the members of  $\Sigma$ , and doesn't imagine us actually making a potentially infinite list. So officially we'll now say

The set  $\Sigma$  is enumerable iff either  $\Sigma$  is empty or else there is a surjective function  $f: \mathbb{N} \rightarrow \Sigma$  (so  $\Sigma$  is the range of  $f$ : we can say that such a function enumerates  $\Sigma$ ).

This comes to the same as our original informal definition in the cases we are interested in, when we *can* write down the members of  $\Sigma$ :

*Proof* Both definitions trivially cover the case where  $\Sigma$  is empty. So concentrate on the non-empty cases.

Pretend we can list off all the members of  $\Sigma$  in some order, repetitions allowed. Count off the members of the list from zero, and define the function  $f$  as follows:  $f(n)$  = the  $n$ -th member of the list, if the list goes on that far, or  $f(n) = f(0)$  otherwise. Then  $f: \mathbb{N} \rightarrow \Sigma$  is evidently a surjection.

Suppose conversely that  $f: \mathbb{N} \rightarrow \Sigma$  is a surjection. Then, if we successively evaluate  $f$  for the arguments  $0, 1, 2, \dots$  in turn, we get a corresponding list of values  $f(0), f(1), f(2), \dots$  which by hypothesis we can imagine writing down and which eventually contains any given element of  $\Sigma$ , with repetitions allowed.  $\square$

Note, however that our official definition of enumerability *doesn't* put any restriction on the 'niceness' of the enumerating surjection  $f: \mathbb{N} \rightarrow \Sigma$ . It could be a function that a computer could readily handle: but equally, it could – as far as our current definition is concerned – be a 'wild' function making a quite arbitrary association between members of  $\mathbb{N}$  and members of  $\Sigma$ . All that matters for enumerability is that there is *some* function  $f$ , however whacky, which pairs up numbers with elements of  $\Sigma$  in such a way that we don't run out of numbers before we've matched every element.

## 2.4 Enumerating pairs of numbers

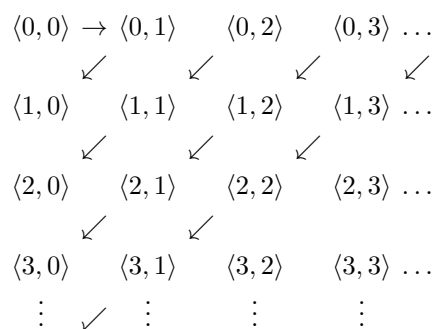
(a) It is immediate that every set of natural numbers  $\Sigma \subseteq \mathbb{N}$ , is enumerable, whether it is finite or infinite (for it is either empty or its members can in principle be listed off).

But if you need an official proof, suppose  $\Sigma$  is non-empty and  $s \in \Sigma$ . Now define the function  $f$  such that  $f(n) = n$  if  $n \in \Sigma$ , and  $f(n) = s$  otherwise. Then  $f$  evidently enumerates  $\Sigma$ .

(b) Here's a rather less obvious example of an enumerable set:

**Theorem 2.1** *The set of ordered pairs of natural numbers  $\langle i, j \rangle$  is enumerable.*

*Proof* The idea is simple. We just arrange the ordered pairs in a systematic array and zig-zag through them, starting for example like this:



This procedure is entirely mechanical, runs through all the pairs, and evidently defines a bijection  $f: \mathbb{N} \rightarrow \mathbb{N}^2$ , a one-one correspondence mapping  $n$  to the  $n$ -th pair on the zig-zag path (so  $f$  is a fortiori a surjection).  $\square$

## 2 Functions and enumerations

---

It is easy to see that ordered pair  $\langle i, j \rangle$  will appear at the location  $pair(i, j) = \{(i + j)^2 + 3i + j\}/2$  on the list (counting from zero). And if you have a taste for puzzles, you can likewise write down two readily calculable functions  $fst(n)$  and  $snd(n)$  which return, respectively the first member  $i$  and the second member  $j$  of the  $n$ -th pair in the zig-zag enumeration.

But we can leave checking those claims as an exercise.

### 2.5 A non-enumerable set: Cantor's theorem

(a) Some sets, however, are 'too big' to enumerate. We can readily find an example and thereby prove *Cantor's Theorem*:<sup>2</sup>

**Theorem 2.2** *There are infinite sets that are not enumerable.*

*Proof* Consider the powerset of  $\mathbb{N}$ , in other words the collection  $\mathcal{P}$  whose members are all the sets of numbers (so  $X \in \mathcal{P}$  iff  $X \subseteq \mathbb{N}$ ).

Suppose for reductio that there is a function  $f: \mathbb{N} \rightarrow \mathcal{P}$  which enumerates  $\mathcal{P}$ , and consider what we'll call the diagonal set  $D \subseteq \mathbb{N}$  such that  $n \in D$  iff  $n \notin f(n)$ .

Since  $D \in \mathcal{P}$  and  $f$  by hypothesis enumerates all the members of  $\mathcal{P}$ , there must be some number  $d$  such that  $f(d) = D$ .

So we have, for all numbers  $n$ ,  $n \in f(d)$  iff  $n \notin f(n)$ . Hence in particular  $d \in f(d)$  iff  $d \notin f(d)$ . Contradiction!

There can therefore be no such enumerating function as  $f$ . Hence the power set  $\mathcal{P}$  cannot be enumerated. In a word, it is *indenumerable*.  $\square$

Which is very neat, though perhaps a little mysterious. For a start, why did we call  $D$  a 'diagonal' set? Let's therefore give a second, rather more intuitive presentation, of the same proof idea: this should make things a bit clearer.

*Another proof* Consider this time the set  $\mathbb{B}$  of infinite binary strings, i.e. the set of unending strings like '0110001010011...'. There's an infinite number of different such strings. Suppose, for reductio, that there is an enumerating function  $f$  which maps the natural numbers onto the strings, for example like this:

$$\begin{array}{l} 0 \rightarrow b_0 : \underline{0}110001010011\dots \\ 1 \rightarrow b_1 : 1\underline{1}00101001101\dots \\ 2 \rightarrow b_2 : 11\underline{0}0101100001\dots \\ 3 \rightarrow b_3 : 000\underline{1}111010101\dots \\ 4 \rightarrow b_4 : 1101\underline{1}11011101\dots \\ \dots \dots \end{array}$$

Go down the diagonal, taking the  $n$ -th digit of the  $n$ -th string  $b_n$  (in our example, this produces 01011...). Now flip each digit, swapping 0s and 1s (in our example,

---

<sup>2</sup>Georg Cantor first established this key result in Cantor (1874), using the Bolzano-Weierstrass theorem. The neater 'diagonal argument' first appears in Cantor (1891).

## 2.5 A non-enumerable set: Cantor's theorem

yielding 10100...). By construction, this 'flipped diagonal' string  $d$  differs from  $b_0$  in the first place; it differs from the next string  $b_1$  in the second place; and so on. So our diagonal construction defines a new string  $d$  that differs from each of the  $b_j$ , contradicting the assumption that our map  $f$  is 'onto' and enumerates *all* the binary strings.

Now, our supposed enumerating function  $f$  was just one example; but the same 'flipped diagonal' construction will plainly work to show that any other candidate map must also fail to enumerate all the strings. So  $\mathbb{B}$  is not enumerable.  $\square$

A moment's reflection shows that this is indeed essentially the same proof.

For take an infinite binary string  $b = \beta_0\beta_1\beta_2\dots$  where each bit  $\beta_i$  is either zero or one. Now this string characterizes a corresponding set  $B$  of natural numbers, where  $n \in B$  if  $\beta_n = 0$  and  $n \notin B$  if  $\beta_n = 1$ . So, for example, the first string  $b_0$  above corresponds to the set of numbers  $\{0, 3, 4, 5, 7, 9, 10, \dots\}$ .

Suppose we enumerate some binary strings  $b_0, b_1, b_2, \dots$ , which respectively represent the sets of numbers  $B_0, B_1, B_2, \dots$ . 'Going down the diagonal' gives us a string representing the set  $K$  such that  $n \in K$  iff  $n \in B_n$ . And 'flipping' gives us the string  $d$  representing the diagonal set  $D = \overline{K}$ , the complement of  $K$  (i.e. the set of numbers not in  $K$ ). So  $n \in D$  iff  $n \notin B_n$ , hence  $D$  differs from each  $B_n$ , so can't appear anywhere in the enumeration.<sup>3</sup> So our proof about  $\mathbb{B}$  is therefore equivalent to the result that any enumeration of sets  $B_n$  leaves out some set of numbers. In other words, as we showed before, the set of sets of natural numbers can't be enumerated.

(b) Let's just add two more quick comments about the second version of our simple but profound proof.

First, an infinite binary string  $b = \beta_0\beta_1\beta_2\dots$  can also be thought of as characterizing a real number  $0 \leq b \leq 1$  in binary digits. So our proof shows that the real numbers in the interval  $[0, 1]$  can't be enumerated (and hence we can't enumerate *all* the reals either).

Second, another way of thinking of an infinite binary string  $b = \beta_0\beta_1\beta_2\dots$  is as characterizing a corresponding function  $f_b$ , i.e. the function that maps each natural number to one of the numbers  $\{0, 1\}$ , where  $f_b(n) = \beta_n$ . So our proof idea also shows that the set of *functions*  $f: \mathbb{N} \rightarrow \{0, 1\}$  can't be enumerated. Put in terms of functions, the trick is to suppose that these functions *can* be enumerated  $f_0, f_1, f_2, \dots$ , define another function by 'going down the diagonal and flipping digits', i.e. define  $\delta(n) = f_n(n) + 1 \pmod{2}$ , and then note that this diagonal function  $\delta$  can't be 'on the list' after all.

(c) We'll meet another version of the 'diagonalization' trick again at the end of the very next chapter. And then, as we'll see, there's a lot more diagonalization to come later. In fact, it is the pivotal proof idea that we'll be repeatedly using in this book.

---

<sup>3</sup>Maybe we should really call  $K$  the diagonal set, and its complement  $D$  the *anti-diagonal* set. But we won't fuss about that.

## 3 Effective computability

The previous chapter talked about functions rather generally. We now narrow the focus and concentrate more specifically on *effectively computable* functions. Later in the book, we'll want to return to some of the ideas we introduce here and give sharper, technical, treatments of them. But for present purposes, informal intuitive presentations are enough.

We also introduce the crucial related notion of an *effective enumerable* set, i.e. a set that can be enumerated by an effectively computable function.

### 3.1 Effectively computable functions

(a) Familiar school-room arithmetic routines – e.g. for squaring a number or finding the highest common factor of two numbers – give us ways of *effectively computing* the value of some function for a given input: the routines are, we might say, entirely mechanical.

Later, in the logic classroom, we learn new mechanical routines. For example, there's a trivial syntactic computation which takes two wffs and forms their conjunction, and there's an only slightly less trivial procedure for effectively computing the truth value of a propositional calculus wff as a function of the values of its atoms.

What is meant by talking of an *effective* computational procedure? The core idea is that an effective computation involves (1) executing an *algorithm* which (2) *successfully terminates*.

1. Here, an algorithm is a set of step-by-step instructions (instructions which are pinned down in advance of their execution), with each small step clearly specified in every detail (leaving no room for doubt as to what does and what doesn't count as executing the step). More carefully, executing an algorithm (i) involves an entirely determinate sequence of discrete step-by-small-step procedures (where each small step is readily executable by a very limited calculating agent or machine). (ii) There isn't any room left for the exercise of imagination or intuition or fallible human judgement. Further, in order to execute the procedures, (iii) we don't have to resort to outside 'oracles' (i.e. independent sources of information), and (iv) we don't have to resort to random methods (coin tosses). Such algorithmic procedures can be executed by a computing machine. Indeed, we can turn that observation into a helpful slogan: an algorithmic procedure is one that a suitably programmed computer can execute.

2. But there's more: plainly, if an algorithmic procedure is actually to compute a total function – i.e. a function which outputs a value for any relevant input(s) – then the algorithm must *terminate* for every input, and produce the right sort of output. Note, then, it isn't part of the very idea of an algorithm that its execution always terminates: so in general, an algorithm may only compute a partial function.

Putting these two thoughts together, then, we can give an initial definition:

A total function  $f: \Delta \rightarrow \Gamma$  is *effectively computable* iff there is an algorithmic procedure that a suitably programmed computer could use for calculating, in a finite number of steps, the value of the function for any given input from the domain  $\Delta$ .

(The generalization to many-place functions is obvious.)

- (b) But what kind of computer do we have in mind here when we gesture towards a definition by saying that an algorithmic procedure is one that a computer can execute? We need to say something more about the relevant sort of computer's *size and speed*, and about its *architecture*.

A real-life computer is limited in size and speed. There will be some upper bound on the size of the inputs it can handle; there will be an upper bound on the size of the set of instructions it can store; there will be an upper bound on the size of its working memory. And even if we feed in inputs and instructions it can handle, it is of little use to us if the computer won't finish executing its algorithmic procedure for centuries.

Still, we are cheerfully going to abstract from all these 'merely practical' considerations of size and speed – which is why we said nothing about them in explaining what we mean by effective procedures. In other words, we will count a function as being effectively computable if there is a finite set of step-by-step instructions which a computer could in principle use to calculate the function's value for any particular arguments, given time and memory enough. Let's be very clear, then: 'effective' here does *not* mean that the computation must be feasible for us, on existing computers, in real time. So, for example, we still count a function as effectively computable in this broad sense even if on existing computers it might take longer to compute a value for a given input it than we have time left before the heat death of the universe. It is enough that there's an algorithm that works in theory and would deliver a result in the end, if only we had the computational resources to use it and could wait long enough.

'But then,' you might well ask, 'why bother with such a radically idealized notion of computability? If we allow procedures that may not deliver an output in the lifetime of the universe, what good is that? If we are interested in issues of computability, shouldn't we really be concerned not with idealized-computability-in-principle but with some notion of *practicable* computability?'

That's a fair challenge. And modern computer science has much to say about grades of computational complexity and levels of feasibility. However, we will

### 3 Effective computability

---

stick to our ultra-idealized notion of computability. Why? Because later we'll be proving a range of limitative theorems, e.g. about what *can't* be algorithmically computed. By working with a very weak 'in principle' notion of what is required for being computable, our impossibility results will be correspondingly very strong – for a start, they won't depend on any mere contingencies about what is practicable, given the current state of our software and hardware, and given real-world limitations of time or resources. They show that some numerical functions can't be computed mechanically, even on the most generous understanding of that idea.

(c) We've said that we are going to be abstracting from limitations on storage, etc. But you might suspect that this still leaves much to be settled. Doesn't the 'architecture' of a computing device affect what it can compute?

The short answer is that it doesn't (at least, once we are dealing with devices of a certain degree of complexity, which can act as 'general purpose' computers). And intriguingly, some of the central theoretical questions here were the subject of intensive investigation even before the first electronic computers were built. Thus, in the mid 1930s, Alan Turing famously analysed what it is for a numerical function to be step-by-step computable in terms of the capacities of a *Turing machine* (a computer following a program built up from particularly simple basic steps: for explanations and examples, see Chapter 32). Now, a standard Mark I Turing machine has just a single 'tape' or workspace to be used for both storing and manipulating data. But we can readily describe a rather more efficient Mark II machine which has two tapes – one to be used as a main workspace, and a separate one for storing data. Or, more radically, we can consider a computer with unlimited 'Random Access Memory' – that is to say, an idealized version of a modern computer, with an unlimited set of registers in which it can store various items of working data ready to be retrieved into its workspace when needed.<sup>1</sup> The details don't matter here and now. What does matter is that exactly the same functions are computable by algorithms written for Mark I Turing machines, by algorithms written for Mark II machines (or its variants), and by algorithms written for register machines, despite their different architectures.

Indeed, *all* the detailed definitions of algorithmic computability by idealized computers with different architectures that have ever been seriously proposed turn out to be equivalent. In a slogan, *algorithmic computability is architecture independent*.

(d) It's worth pausing over that last major claim, and taking it a bit more slowly, in two stages.

First stage: there's a technical result about the mutual equivalence of various different proposed ways of refining the ideas of effectively computing *numerical* functions (i.e. where the domain and co-domain are natural numbers) – we end up, every time, with the same class of Turing-computable functions. That's a

---

<sup>1</sup>The theoretical treatment of unlimited register machines was first given in Shepherdson and Sturgis (1963); there is a very accessible presentation in the excellent Cutland (1980).

formal mathematical theorem (or rather cluster of theorems, that we have to prove case by case). But it supports the conjecture which Turing famously makes in his classic paper published in 1936:

*Turing's Thesis* The numerical functions that are effectively computable in the informal sense are just those functions that are in fact computable by a suitable Turing machine.

As we'll see, however, Turing machines are rather horrible to work with (essentially, you have to program them at the level of 'machine code'). So you might want to think instead in terms of the numerical functions which are computable – at least when we abstract from limitations of time and memory space – on a modern general purpose computer, using programs written in your favourite general purpose language, C++ perhaps. Then Turing's Thesis is provably equivalent to this: the numerical functions that are effectively computable in the informal sense are just those functions that are in principle computable using algorithms written in C++.

Turing's Thesis – we'll further explore its content in Chapter 35 – correlates an informal notion with a sharp technical analysis. So you might think it isn't the sort of thing for which we can strictly speaking give a *proof* (though see Chapter 36). But be that as it may. Certainly, after more than seventy five years, no successful challenge to Turing's Thesis has ever been mounted. Which means that we can continue to talk informally about effectively computable numerical functions, and yet be very confident that we are referring to a fully determinate class.

(e) Second stage: what about extending the idea of being effectively computable to *non-numerical* functions? Does the idea remain determinate?

Here's a natural suggestion: any computation dealing with sufficiently discrete and distinguishable finite objects  $X$ s can be turned into an equivalent numerical computation via the trick of using simple numerical codes for the different  $X$ s. In other words, by a relatively trivial algorithm, we can map  $X$ s to numbers; we can then do the appropriate core computation on the numbers; and then another trivial algorithm translates the result back into a claim about  $X$ s.<sup>2</sup> So if it is determinate what numerical functions are effectively computable, the same goes for other functions over suitable finite objects.

Fortunately, however, we don't need to pause to assess this line of argument in its fullest generality. For the purposes of this book, the non-numerical computations we are most interested in are cases where the  $X$ s are *expressions* from standard formal languages, or *sequences of expressions*, etc. And in these cases, there's no doubt at all that we can algorithmically map claims about such things to corresponding claims about numbers (see Sections 4.5, 16.1, 16.2). So we can

---

<sup>2</sup>After all, this is how real-world digital computers work: they code up non-numerical data into numbers in binary form, operate on them, and then decode to present the results in some nice human-consumable form.

### 3 Effective computability

---

indeed assume, given Turing's Thesis, that it is also determinate what counts as an effectively computable operation over expressions, etc.

#### 3.2 Effectively decidable properties and sets

(a) We often use algorithmic routines not only to compute *functions* but also to decide whether a *property* holds. For example, there are familiar school-room routines for mechanically testing whether a given number is divisible by nine, or whether it is prime. Later, in the logic room, we learn computational routines for deciding whether a given string of symbols is a wff of the propositional calculus, and for deciding whether such a wff is a tautology.

Inspired by such cases, let's say – as another initial definition – that

A property/relation is *effectively decidable* iff there is an algorithmic procedure that a suitably programmed computer could use to decide, in a finite number of steps, whether the property/relation applies of any appropriate given item(s).

Note that again 'effective' here doesn't been 'practicable' or 'efficient': a property can be effectively decidable even if it would take a vast amount of time or computational resources to settle whether a given object has it. All that is required is that there is some algorithm that in principle would do the job, given world enough and time.

(b) In what follows, we are crucially going to be interested in decidable properties of numbers and decidable relations holding between numbers. And now we can deploy the notion of a characteristic function which we introduced in §2.2, and alternatively say

A numerical property or relation is effectively decidable iff its characteristic function is effectively computable.

Let's just check that our two definitions do indeed tally in the case of numerical properties. So suppose there is an algorithm which decides whether  $n$  is  $P$ , delivering a 'yes'/'no' verdict: just add the instruction 'if the answer is *yes*, output 0: otherwise output 1' and you get an algorithm for computing  $c_P(n)$ . Conversely, suppose you have an algorithm for computing  $c_P(n)$ . Then you can add the line 'if the value is 0 output *yes*: otherwise output *no*' to get an algorithm to decide whether  $n$  is  $P$ .

Since, by Turing's Thesis, the notion of an effectively computable numerical function is determinate, so too is the notion of an effectively decidable numerical property or relation.

So far, so good: what, though, about the idea of a non-numerical effectively decidable property? Can we be sure that *that* notion is determinate?

Again yes, at least in the cases that interest us. For as we said, we can code up finite objects like expressions or sequences of expressions using numbers in

## 3.2 Effectively decidable properties and sets

---

simple algorithmic ways. So the question whether e.g. a certain property of formulae is a decidable one can be uncontentiously translated into the question whether a corresponding property of numbers is a decidable one. Since it is quite determinate what counts as a decidable property of numbers, it then follows that it is quite determinate what counts as a decidable property of formal expressions or sequences of expressions.

(c) Moving from talk of numerical properties to talk of their extensions (sets), we'll also now say that

A set  $\Sigma$  is effectively decidable iff there is an algorithmic procedure that a suitably programmed computer could use to decide, in a finite number of steps, whether any relevant given object is a member of  $\Sigma$ .

For the particular case where  $\Sigma$  is a set of numbers, we can sharpen up this definition in a way parallel to our sharpened account of decidable properties:

A set of natural numbers  $\Sigma \subseteq \mathbb{N}$  is effectively decidable iff the characteristic function of the property of belonging to  $\Sigma$  (i.e. the function  $c_\Sigma$  such that  $c_\Sigma(n) = 0$  if  $n \in \Sigma$  and  $c_\Sigma(n) = 1$  otherwise) is effectively computable.

Given Turing's Thesis, this makes the idea of an effectively decidable set of numbers another sharply determinate one. And again, at least when we are dealing with finite objects that can be nicely coded up by numbers, the idea of effectively decidable sets of them can be made equally determinate.

(d) As a reality check, let's just pause to note a pair of mini-theorems:

**Theorem 3.1** *Any finite set of natural numbers is effectively decidable.*

**Theorem 3.2** *If  $\Sigma$  is an effectively decidable set of numbers, so is its complement  $\bar{\Sigma}$ .*

*Proofs (if needed!)* For the first, note that if  $\Sigma \subseteq \mathbb{N}$  is finite, the characteristic function  $c_\Sigma$  always takes the value 1 except for a finite number of arguments when it goes to 0. Such a function is effectively computable by a brute-force algorithm which checks the input against the finite list of exceptional arguments, outputs 0 if it gets a match and otherwise defaults to 1.

For the second, note that if the characteristic function  $c_\Sigma$  is effectively computable, so (trivially) is the function  $\bar{c}$  defined by  $\bar{c}(n) = 1 - c_\Sigma(n)$ . But  $\bar{c}$  is evidently the characteristic function of  $\bar{\Sigma}$ .  $\square$

Since the complement of a finite set is infinite, those two mini-theorems entail that there are infinite sets of numbers which are decidable. And there are infinite sets with infinite complements that are also decidable, e.g. the set of numbers

### 3 Effective computability

---

divisible by nine, the set of primes. But we'll soon see that there are undecidable sets of numbers too.

#### 3.3 Effective enumerability

(a) We said: a non-empty set  $\Sigma$  is enumerable so long as there is *some* way of listing its elements, or – more carefully – so long as there is *some* surjective function  $f: \mathbb{N} \rightarrow \Sigma$  which enumerates it.

As we noted before, this definition does not require that the listing can be done ‘mechanically’: in other words, the enumerating function  $f$  here can be any arbitrary correlation of numbers with elements of  $\Sigma$  (so long as it is ‘onto’). It need not even be finitely specifiable, let alone be a ‘nice’ computable function.

By contrast, then, we'll say

The set  $\Sigma$  is *effectively enumerable* (e.e.) iff either  $\Sigma$  is empty or else there is an effectively computable function that enumerates it.<sup>3</sup>

By the same reasoning as in §2.3, you can equivalently think of it like this (when the members of  $\Sigma$  are the sort of things you can put on a list):  $\Sigma$  is e.e. just if an (idealized) computer could be programmed to generate a list of  $\Sigma$ 's members such that any member will eventually be mentioned – the list may be empty, or have no end, and may contain repetitions, so long as any item in the set eventually makes an appearance.

(b) Let's have some examples, concentrating for now on numerical ones. First, any finite set of numbers is e.e.: any listing will do, and – being finite – can be stored in an idealized computer and spat out on demand.

For a trivial example of an e.e. infinite set, the evidently computable function  $f(n) = n^2$  effectively enumerates the natural numbers which are perfect squares.

And for an only slightly more complex case imagine an algorithm that takes the natural numbers one at a time in order and applies the well-known mechanical test for being prime, and lists the successes: this procedure generates a never-ending list on which every prime will eventually appear – so the primes are effectively enumerable.

Evidently, that argument about primes generalizes. Here's one version:

**Theorem 3.3** *If  $\Sigma$  is an effectively decidable set of numbers, it is effectively enumerable.*

*Proof* The case where  $\Sigma$  is empty is trivial. So assume  $s \in \Sigma$ , and consider the algorithm which, for input  $n$ , effectively tests which  $n$  is in  $\Sigma$  (by hypothesis,

---

<sup>3</sup>NB: whether a set is effectively enumerable, enumerable but not effectively so, or neither, depends on what functions there *are*, not on which functions we *know* about. Also note that terminology hereabouts isn't entirely stable: some writers use ‘enumerable’ to mean *effectively* enumerable, and use e.g. ‘denumerable’ for the wider notion of enumerability.

### 3.4 Another way of defining e.e. sets of numbers

---

that can be done), and if it gets a ‘yes’ outputs  $n$ , and otherwise outputs  $s$ . This algorithm computes a total surjective function  $f: \mathbb{N} \rightarrow \Sigma$ , so  $\Sigma$  is effectively enumerable.  $\square$

(c) Does the result reverse? If a set  $\Sigma$  is e.e. must it be effectively decidable?

Intuitively, that shouldn’t hold. Suppose we want to decide whether  $s \in \Sigma$  for some suitable  $s$ . If all we know is that there is a computable enumerating function  $f: \mathbb{N} \rightarrow \Sigma$ , then it seems that all we can do is start evaluating  $f(0), f(1), f(2), \dots$ . If eventually we find some  $n$  such that  $f(n) = s$ , that settles it:  $s \in \Sigma$ . But if we haven’t (yet) found such an  $n$ , everything is still to play for: perhaps we’ve just not looked long enough and it will still turn out that  $s \in \Sigma$ , or perhaps  $s \notin \Sigma$ . A finite search along the  $f(n)$ , however long, seems as if it may not settle anything.

However, we do have the following theorem

**Theorem 3.4** *If  $\Sigma$  and also its complement  $\bar{\Sigma}$  are both effectively enumerable sets of numbers, then  $\Sigma$  is effectively decidable.*

*Proof* Suppose  $\Sigma$  is enumerated by the effectively computable function  $f$ , and  $\bar{\Sigma}$  by  $g$ . Here’s how to determine, of any given number  $s$ , which set it is in.

Compute in turn  $f(0), g(0), f(1), g(1), f(2), g(2), \dots$ . Eventually we must get the output value  $s$  (since either  $s \in \Sigma$  or  $s \in \bar{\Sigma}$ ). If we get to some  $m$  such that  $f(m) = s$ , then the algorithm tells us to return the verdict that  $s \in \Sigma$ ; if we get to some  $n$  such that  $g(n) = s$ , then the algorithm tells us to return the verdict that  $s \in \bar{\Sigma}$ , i.e.  $s \notin \Sigma$ .  $\square$

We are, of course, relying here on the ultra-generous notion of decidability-in-principle. We might have to twiddle our thumbs for an immense time to see whether  $s$  is in  $\Sigma$  or  $\bar{\Sigma}$ . Still, our ‘wait and see’ method is guaranteed by our assumptions to produce a result in finite time, in an entirely mechanical way – so this counts as an effectively computable procedure in our official generous sense.

Now, if every e.e. set of numbers in fact has an effectively enumerable complement, then every e.e. set of numbers is decidable after all, contrary to our argument above. But as we’ll see in §3.5, some e.e. sets have complements which are not e.e., and there are indeed undecidable e.e. sets of numbers.

### 3.4 Another way of defining e.e. sets of numbers

(a) We said that an algorithmic procedure is one that a suitably programmed computer can execute (abstracting from limitations of time or memory space). To fix ideas, concentrate for this section on algorithms written in your favourite general purpose language, C++ as it might be.

Here now is a new definition: the *numerical domain* of such an algorithm  $\Pi$  is the set of natural numbers  $n$  such that, when the algorithm  $\Pi$  is applied to the

### 3 Effective computability

---

number  $n$  as input, then  $\Pi$  will eventually terminate (in principle, given world enough and time) and deliver some number as output.

Some algorithms have empty numerical domains (perhaps they always crash or get stuck in a loop when fed a single number as input). Some algorithms have  $\mathbb{N}$ , the whole set of natural numbers, as their numerical domain (e.g. a correct algorithm that maps  $n$  to the  $n$ -th prime). And yet other algorithms have some but not all numbers in their domain (exercise: sketch an algorithm which gives the output 0 for all even number inputs but goes into an infinite loop for any odd number input, and hence has just the even numbers for its numerical domain).

Rather surprisingly, however, whatever the algorithm  $\Pi$ , its numerical domain will *always* be an effectively enumerable set of numbers. Indeed we have

**Theorem 3.5**  *$W$  is an effectively enumerable set of numbers if and only if it is the numerical domain of some algorithm  $\Pi$ .*

*Proof of the ‘only if’ direction* Suppose  $W$  is an e.e. set of numbers. Then either (i)  $W$  is empty, or (ii) there is an effectively computable function  $f$  which enumerates  $W$ .

In case (i), choose any algorithm  $\Pi$  that never produces any output and we are done.

In case (ii) there must be some algorithm which computes the enumerating function  $f$ . So we can use that to construct the following composite algorithmic procedure  $\Pi$ . Given number  $n$  as input, compute the values of  $f(0), f(1), f(2), \dots$  in turn: keep going on and on unless and until one of those values turns out to be  $n$  – and if it does, stop and output the number 0. Then the numerical domain of  $\Pi$  is obviously  $W$  (for  $\Pi$  will terminate just when fed an  $n \in W$ ).

*Proof of the ‘if’ direction* Suppose that  $W$  is the numerical domain of some step-by-step algorithmic procedure  $\Pi$ . We’ll introduce the shorthand notation  $\Pi:n$  to refer to a computation using  $\Pi$  on input  $n$ . Then consider the following zig-zag effective procedure:

Do the initial step of  $\Pi:0$ ; then the initial step of  $\Pi:1$  and the next step of  $\Pi:0$ ; then the initial step of  $\Pi:2$  and the next steps of  $\Pi:1$  and  $\Pi:0$ ; then the initial step of  $\Pi:3$  and the next steps of  $\Pi:2$ ,  $\Pi:1$  and  $\Pi:0$ ; and keep on going, zig-zagging through (skipping computations of  $\Pi:n$  if they have already halted). And whenever we get to a step where the computation of some  $\Pi:n$  halts with some numerical output, write down  $n$ .

This effective procedure eventually executes every  $\Pi:n$  to the point where it halts (if it ever does), so must potentially generate a list of all the  $n$  in  $W$ . Hence  $W$  is e.e. □

(b) We supposed we were working within some general-purpose programming language like C++ in which we can regiment instructions a version of any algorithm that is supposed to operate on numbers. Of course, it is a significant

### 3.5 The Basic Theorem about e.e. sets

assumption here that there *are* general purpose languages, apt for regimenting instructions for all kinds of numerical algorithms. But even a passing familiarity with modern computing practice should make our assumption seem entirely reasonable.<sup>4</sup>

Now start listing off in some ‘alphabetical order’ all the possible strings of symbols of our chosen programming language, and retain just those that obey the rules for being a series of syntactically well-formed program instructions in that language. This gives us an effectively generated list off versions of all the possible algorithms  $\Pi_0, \Pi_1, \Pi_2, \dots$  as stated in our language (most of them will be garbage of course, useless algorithms which ‘crash’). Let the numerical domain of  $\Pi_e$  be  $W_e$ . Then, by our last theorem, every effectively enumerable set of numbers is  $W_e$  for some index  $e$ . Which implies

**Theorem 3.6** *The set  $\mathcal{W}$  of effectively enumerable sets of natural numbers is itself enumerable.*

For take the function  $f: \mathbb{N} \rightarrow \mathcal{W}$  defined by  $f(e) = W_e$ . This enumerates  $\mathcal{W}$ .<sup>5</sup>

And this immediately entails

**Theorem 3.7** *Some sets of numbers are not effectively enumerable, and hence not effectively decidable.*

*Proof* We already know that the powerset  $\mathcal{P}$  of  $\mathbb{N}$  – i.e. the collection of *all* sets of numbers *can’t* be enumerated (see the first proof of Theorem 2.2). But we’ve just seen that  $\mathcal{W}$ , the set of effectively enumerable sets of numbers, *can* be enumerated. So  $\mathcal{W} \neq \mathcal{P}$ . But trivially,  $\mathcal{W} \subseteq \mathcal{P}$ . So there must be members of  $\mathcal{P}$  which aren’t in  $\mathcal{W}$ , i.e. sets of numbers which aren’t effectively enumerable. Theorem 3.3 entails that these sets, a fortiori, aren’t effectively decidable.  $\square$

### 3.5 The Basic Theorem about e.e. sets

With the simple but crucial Theorem 3.5 to hand, we can now prove what deserves to be called the *Basic Theorem* about effectively enumerable sets of numbers:

**Theorem 3.8** *There is an effectively enumerable set of numbers  $K$  such that its complement  $\overline{K}$  is not effectively enumerable.*

*Proof* We use another diagonal construction. Put  $K =_{\text{def}} \{e \mid e \in W_e\}$  and the result follows.<sup>6</sup>

*$\overline{K}$  is not effectively enumerable* By definition, for any  $e$ ,  $e \in \overline{K}$  if and only if  $e \notin W_e$ . Hence,  $\overline{K}$  cannot be identical to any of the  $W_e$  (since  $e$  is in one but

<sup>4</sup>Much later, when we return to discuss Turing’s Thesis, we’ll say more in defence of the assumption.

<sup>5</sup>Question: is  $f$  an effective enumeration?

<sup>6</sup>Compare the analogous construction in our comments on Cantor’s Theorem 2.2.

### 3 Effective computability

---

not the other). Therefore  $\overline{K}$  isn't one of the effectively enumerable sets (since the  $W_e$  are all of them).

*K is effectively enumerable* We use the same argument idea that we used in proving Theorem 3.5.

Each  $W_e$  is, by definition, the numerical domain of a corresponding algorithm  $\Pi_e$ . So now consider the following zig-zag effective procedure:

Do the initial step of  $\Pi_0:0$ ; then do the initial step of  $\Pi_1:1$  and the next step of  $\Pi_0:0$ ; then the initial step of  $\Pi_2:2$  and the next steps of  $\Pi_1:1$  and  $\Pi_0:0$ ; then the initial step of  $\Pi_3:3$  and the next steps of  $\Pi_2:2$ ,  $\Pi_1:1$  and  $\Pi_0:0$ ; and keep on going, zig-zagging through (skipping computations of  $\Pi_e:e$  if they have already halted). Whenever we get to a step where the computation of some  $\Pi_e:e$  halts with some output, write down  $e$ .

This effective procedure eventually executes every  $\Pi_e:e$  as far as you like, so must potentially generate a list of all the  $e$  such in  $\Pi_e:e$  gives an output, i.e. all the  $e$  such that  $e$  is in the numerical domain of  $\Pi_e$ , i.e. all the  $e$  such that  $e \in W_e$ . Hence the procedure effectively enumerates  $K$ .  $\square$

As an immediate corollary we have

**Theorem 3.9** *There is an effectively enumerable set of numbers which is not decidable*

*Proof* Take  $K$  again, an e.e. set with a non-e.e. complement. If  $K$  were decidable, its complement  $\overline{K}$  would be decidable too, by mini-Theorem 3.2. But then  $\overline{K}$  would be e.e., by mini-Theorem 3.3. Contradiction.  $\square$

## 4 Axiomatized formal theories

Gödel's Incompleteness Theorems tell us about the limits of theories of arithmetic. Or rather, more carefully, they tell us about the limits of *axiomatized formal theories* of arithmetic. But what exactly does that mean? This chapter starts exploring the idea and proves some elementary results about axiomatized formal theories in general.

### 4.1 Formalization as an ideal

Rather than just dive into a series of definitions, it is well worth pausing to remind ourselves of why we *care* about formalized theories.

Let's get back to basics. In elementary logic classes, we are drilled in translating arguments into an appropriate formal language and then constructing formal deductions of putative conclusions from given premisses.

Why bother with formal languages? Because everyday language is replete with redundancies and ambiguities, not to mention sentences which simply lack clear truth-conditions. So, in assessing complex arguments, it helps to regiment them into a suitable artificial language which is expressly designed to be free from obscurities, and where surface form reveals logical structure.

Why bother with formal deductions? Because everyday arguments often involve suppressed premisses and inferential fallacies. It is only too easy to cheat. Setting out arguments as formal deductions in one style or another enforces honesty: we have to keep a tally of the premisses we invoke, and of exactly what inferential moves we are using. And honesty is the best policy. For suppose things go well with a particular formal deduction. Suppose we get from the given premisses to some target conclusion by small inference steps each one of which is obviously valid (no suppressed premisses are smuggled in, and there are no suspect inferential moves). Our honest toil then buys us the right to confidence that our premisses really do entail the desired conclusion.

Granted, outside the logic classroom we almost never set out deductive arguments in fully formalized versions. No matter. We have glimpsed a first ideal – arguments presented in an entirely perspicuous formal language with maximal clarity and with everything entirely open and above board, leaving no room for misunderstanding, and with all the arguments' commitments systematically and frankly acknowledged.<sup>1</sup>

---

<sup>1</sup>For an early and very clear statement of this ideal, see Frege (1882), where he explains the point of the first modern formal system of logic – albeit with a horrible notation – presented in his *Begriffsschrift* (i.e. *Conceptual Notation*) of 1879.

## 4 Axiomatized formal theories

---

Old-fashioned presentations of Euclidean geometry illustrate the pursuit of a related second ideal – the axiomatized theory. Like beginning logic students, school students used to be drilled in providing deductions, though the deductions were framed in ordinary geometric language. The game is to establish a whole body of theorems about (say) triangles inscribed in circles, by deriving them from simpler results, which had earlier been derived from still simpler theorems that could ultimately be established by appeal to some small stock of fundamental principles or axioms. And the aim of this enterprise? By setting out the derivations of our various theorems in a laborious step-by-step style – where each small move is warranted by simple inferences from propositions that have already been proved – we develop a unified body of results that we can be confident must hold if the initial Euclidean axioms are true.

On the surface, school geometry perhaps doesn't seem very deep: yet making all its fundamental assumptions fully explicit is surprisingly difficult. And giving a set of axioms invites further enquiry into what might happen if we tinker with these assumptions in various ways – leading, as is now familiar, to investigations of non-Euclidean geometries.

Old-style axiomatized geometry was presented informally. These days, many mathematical theories are presented axiomatically in a more formal way from the outset. For example, set theory is typically presented by laying down some axioms expressed in a partially formalised language and exploring their deductive consequences. We want to discover exactly what is guaranteed by the fundamental principles embodied in the axioms. And we are again interested in exploring what happens if we change the axioms and construct alternative set theories.

However, even the most tough-minded mathematics texts which explore axiomatized theories are in fact written in an informal mix of ordinary language and mathematical symbolism. Proofs are very rarely spelt out in every formal detail, and so their presentation falls short of the logical ideal of full formalization. But we will hope that nothing stands in the way of our more informally presented mathematical proofs being sharpened up into fully formalized ones – i.e. we hope that they *could* be set out in a strictly regimented formal language of the kind that logicians describe, with absolutely every inferential move made fully explicit and checked as being in accord with some overtly acknowledged rule of inference, with all the proofs ultimately starting from our explicitly given axioms. True, the extra effort of laying out everything in this kind of detail will almost never be worth the cost in time and ink. In mathematical practice we use enough formalization to convince ourselves that our results don't depend on illicit smuggled premisses or on dubious inference moves, and leave it at that – our motto is 'sufficient unto the day is the rigour thereof'.<sup>2</sup> But still, it *is* absolutely essential for good mathematics to achieve precision and to avoid the use of unexamined inference rules or unacknowledged assumptions. So, putting

---

<sup>2</sup>'Most mathematical investigation is concerned not with the analysis of the complete process of reasoning, but with the presentation of such an abstract of the proof as is sufficient to convince a properly instructed mind.' (Russell and Whitehead, 1910–13, vol. 1, p. 3)

together the logician's aim of perfect clarity and honest inference with the mathematician's project of regimenting a theory into a tidily axiomatized form, we can see the point of the notion of an *axiomatized formal theory* as a composite ideal.

Note, we are not saying that mathematicians ought really always to work inside fully formalized theories. Mathematics is hard enough even when done using the usual strategy of employing just as much rigour as seems appropriate to the case in hand.<sup>3</sup> And in any case, as mathematicians (and some philosophical commentators) are apt to stress, there is a lot more to mathematical practice than striving towards the logical ideal. For a start, we typically aim for proofs which are not merely correct but *explanatory* – i.e. proofs which not only show that some proposition must be true, but in some sense make it clear *why* it is true. However, such observations don't affect our present point, which is that the business of formalization just takes to the limit features that we expect to find in good proofs anyway, i.e. precise clarity and lack of inferential gaps.

## 4.2 Formalized languages

Putting together the ideal of formal precision and the ideal of regimentation into an axiomatic system, we have arrived at the concept of an *axiomatized formal theory*, which comprises a formalized *language*, a set of formulae from the language which we treat as *axioms* for the theory, and a *deductive system* for proof-building, so that we can derive theorems from the axioms.

In this section, we'll say just a bit more about the idea of a properly formalized language – though we'll be very brisk, as we don't want to get bogged down in details. Our main concern is to emphasize points about effective decidability which aren't usually highlighted in introductory presentations.

(a) But first let's stress that we are normally interested in *interpreted* languages – i.e. we are concerned not merely with patterns of symbols but with expressions which have an intended significance. After all, our formalized proofs are ideally supposed to be just that, i.e. *proofs* with content, which show things to be true. Agreed, we'll often be very interested in certain features of proofs that can be assessed independently of their significance (for example, we'll want to know whether a putative proof does obey the formal syntactic rules of a given deductive system). But it is one thing to set aside their semantics for some purposes; it is another thing entirely to drain formal proofs of all semantic significance.

We will think of a formal language  $L$ , therefore, as being a pair  $\langle \mathcal{L}, \mathcal{I} \rangle$ , where  $\mathcal{L}$  is a syntactically defined system of expressions and  $\mathcal{I}$  gives the interpretation of these expressions. In the next chapter, we'll give an account of the syntax

---

<sup>3</sup>See Lakatos (1976) for a wonderful exploration of how mathematics evolves. This gives some real sense of how regimenting proofs in order to clarify their assumptions – the process which formalization idealizes – is just one phase in the complex process that leads to the growth of mathematical knowledge.

## 4 Axiomatized formal theories

---

and semantics of the particular language  $L_A$ , a formal counterpart of what we called ‘the language of basic arithmetic’ in §1.1. But for the moment, we’ll stick to generalities.

(b) Start with  $L$ ’s syntactic component  $\mathcal{L}$ . We can here assume that this is based on a finite alphabet of symbols (for we can always construct e.g. an unending supply of variables from a finite base by standard tricks like using repeated primes to yield ‘ $x$ ’, ‘ $x'$ ’, ‘ $x''$ ’, etc.). Then

1. We first need to specify which symbols or finite strings of symbols make up  $\mathcal{L}$ ’s *non-logical vocabulary*, e.g. the individual constants (names), predicates, and function-signs of  $L$ .
2. We also need to settle which symbols or strings of symbols make up  $\mathcal{L}$ ’s *logical vocabulary*: typically this will comprise variables, symbols for connectives and quantifiers, the identity sign, and bracketing devices.
3. Crucially, we need syntactic construction rules to determine which finite sequences of logical and non-logical vocabulary constitute the well-formed formulae of  $\mathcal{L}$  – its *wffs*, for short. For technical purposes it can be useful to allow wffs with free variables; but of course, our main interest will be in the  $\mathcal{L}$ -*sentences*, i.e. the closed wffs without variables dangling free.

This should all be very familiar.<sup>4</sup>

Now, given that the whole point of using a formalized language is to make everything as clear and determinate as possible, we plainly do not want it to be a disputable matter whether a given symbol or cluster of symbols is e.g. a constant or one-place predicate or two-place function of a system  $\mathcal{L}$ . Nor, crucially, do we want disputes about whether a given string of symbols is an  $\mathcal{L}$ -wff or, more specifically, is an  $\mathcal{L}$ -sentence.

So, whatever the fine details, for a properly formalized syntax  $\mathcal{L}$ , there should be clear and objective procedures, agreed on all sides, for *effectively deciding* whether a putative constant-symbol really is a constant, a putative one-place predicate is indeed one, etc. Likewise we need to be able to effectively decide whether a string of symbols is an  $\mathcal{L}$ -wff/ $\mathcal{L}$ -sentence. It goes almost without saying that the formal languages familiar from elementary (or not-so-elementary) logic standardly have this feature.<sup>5</sup>

(c) Let’s move on, then, to the interpretation  $\mathcal{I}$ . The prime aim is, of course, to fix the content of each closed  $\mathcal{L}$ -wff, i.e. each  $\mathcal{L}$ -sentence. And standardly, we fix

---

<sup>4</sup>If it *isn’t*, any elementary but sufficiently formal logic textbook will help – see e.g. Leary (2000, Ch. 1) or – at a more leisurely pace – Chiswell and Hodges (2007).

<sup>5</sup>If you have done some model theory, then you’ll have encountered an exception: for there it is a useful dodge to construct ‘languages’ in an extended sense which have one constant for every element of a domain, even when the domain has a very large cardinality. But these are languages-as-formal-objects-we-can-theorize-about, not languages which we could in principle master and use to express theories.

the content of formal sentences by giving truth-conditions, i.e. by saying what it would take for a given sentence to be true.

However, we can't, in the general case, do this in a manageable way just by giving a list associating  $\mathcal{L}$ -sentences with truth-conditions (for the simple reason that there will be an unlimited number of sentences). We'll therefore aim for a 'compositional semantics', which tells us how to systematically work out the truth-condition of any  $\mathcal{L}$ -sentence in terms of the semantic significance of the expressions which it contains.

What does such a compositional semantics  $\mathcal{I}$  look like? Suppose, for example, that  $\mathcal{L}$  has the usual syntax of a first-order language (for the moment, without function signs).<sup>6</sup> Then

1. We initially need to specify the domain of quantification – e.g. as the set of people.
2. We give interpretations for the non-logical vocabulary by assigning *values* in the domain to constants, and giving *satisfaction conditions* for predicates. For example,
  - i. the value of 'm' is Socrates; the value of 'n' is Plato; ...
  - ii. a thing satisfies 'F' iff it is wise; an ordered pair of things satisfies 'L' iff the first of them loves the second; ...

Then we have the obvious rules for assigning truth-conditions to atomic sentences, so that e.g. 'Fm' is true just in case the value of 'm' satisfies 'F' (i.e. iff Socrates is wise); 'Lmn' is true just in case the ordered pair ⟨value of 'm', value of 'n'⟩ satisfies 'L' (i.e. iff Socrates loves Plato); and so on.<sup>7</sup>

3. Now we need to deal with the logical vocabulary. There are the usual rules for assigning truth-conditions to sentences built up out of simpler ones using the propositional connectives. That leaves the quantifiers to deal with. Take the existential case. If every object in the domain is named

<sup>6</sup>'First-order' means that the quantifiers run over the objects in the domain: compare 'second-order' logic which also allows a second sort of quantifier that runs over properties of those objects. For more, see Section 23.1.

<sup>7</sup>Note that something satisfies 'F' according to  $\mathcal{I}$  iff it is wise, hence iff it is in the set of wise people. Call that set associated with 'F' its *extension*. Then 'Fm' is true on interpretation  $\mathcal{I}$  iff the value of 'm' is in the extension of 'F'.

Pursuing this idea, we can give a basically equivalent semantic story that deals with one-place predicates by assigning them subsets of the domain as extensions rather than by giving satisfaction conditions; similarly two-place predicates will be assigned sets of ordered pairs of elements of the domain, and so forth. Which is the way logic texts more usually tell the official semantic story, and for a very good reason. In logic, we are interested in finding the valid inferences, i.e. those which are such that, on *any* possible interpretation of the relevant sentences, if the premisses are true, the conclusion is true. Logicians therefore need to be able to *generalize* about all possible interpretations. Describing interpretations set-theoretically gives us a mathematically clean way of doing this generalizing work. However, in specifying a *particular* interpretation  $\mathcal{I}$  for a given  $\mathcal{L}$  we don't need to put it in such overly set-theoretic terms. So we won't.

## 4 Axiomatized formal theories

---

by some term, then it is easy. ‘ $\exists xFx$ ’ is true just if, for some term ‘ $c$ ’, ‘ $Fc$ ’ comes out true. In the general case, though, we can’t assume that every object has a name, and we have to box more cleverly. A standard trick is, in effect, to say ‘ $\exists xFx$ ’ is true just if there is some way of assigning an object from the domain of quantification as a value to ‘ $x$ ’ – treating that now as a temporary name – so that ‘ $Fx$ ’ comes out true (so, on this approach, we do need to allow wffs with free variables).

This should be basically quite familiar. And we don’t need to go into more details here – especially because, in the central case which concerns us when the domain of quantification is the natural numbers, there *is* a canonical numeral to denote each number, so the semantics of quantification is easily dealt with.

For now, the point we want to highlight is this: given the aims of formalization, a compositional semantics needs to yield an unambiguous truth-condition for each sentence *and do it in an effective way*. The usual accounts of the semantics of the standard formal languages of logic do indeed have this feature of effectively generating unique readings for sentences. (Careful! The claim is only that the semantics effectively tells us the conditions under which a given sentence is true: it doesn’t at all follow that there is an effective way of telling whether those conditions hold and the sentence *is* true.)

### 4.3 Axiomatized formal theories

Now for the idea of an axiomatized formal theory, built in a formalized language (normally, of course, an interpreted formalized language). Once more, it is issues about decidability which need to be emphasised.

(a) First, some wffs of our theory’s language are to be selected as (*non-logical*) *axioms*, i.e. as the fundamental non-logical assumptions of our theory. Of course, we’ll normally want these axioms to be sentences which are true on interpretation: but that needn’t be built into the very notion of an axiomatized theory.

Since the fundamental aim of the axiomatization game is to see what follows from a bunch of axioms, we certainly don’t want it to be a matter for dispute whether a given proof does or doesn’t appeal only to axioms in the chosen set. Given a purported derivation of some result, there should be an absolutely clear procedure for settling whether the input premisses are genuinely to be found among the official axioms. In other words, *for an axiomatized formal theory, we must be able to effectively decide whether a given wff is an axiom or not*.

That doesn’t, by the way, rule out theories with infinitely many axioms. We might want to say ‘every wff of such-and-such a form is an axiom’ (where there is an unlimited number of instances): that’s permissible so long as it is still effectively decidable what counts as an instance of that form.

(b) Second, an axiomatized formal theory needs some deductive apparatus, i.e. some sort of formal *proof system*. And we’ll take proof derivations always to be

*finite* arrays of wffs, arrays which are built up in ways that conform to the rules of the relevant proof system.<sup>8</sup>

We'll take it that the core idea of a proof system is once more very familiar from elementary logic. The differences between various equivalent systems of proof presentation – e.g. old-style linear proof systems which use logical axioms vs. different styles of natural deduction proofs vs. tableau (or ‘tree’) proofs – don't essentially matter. What is crucial, of course, is the strength of the overall system we adopt. We will predominantly be working with some version of standard first-order logic with identity. But whatever system we adopt, we need to be able to specify it in a way which enables us to settle, without room for dispute, what counts as a well-formed derivation.

In other words, *we require the property of being a well-formed proof from premisses  $\varphi_1, \varphi_2, \dots, \varphi_n$  to conclusion  $\psi$  in the theory's proof system to be an effectively decidable one.* The whole point of formalizing proofs is to set out the deductive structure of an argument with absolute determinacy; so we don't want it to be a disputable or subjective question whether the inference moves in a putative proof do or do not conform to the rules for proof-building for the formal system in use. Hence there should be a clear and effective procedure for deciding whether a given array of symbols counts as a well-constructed derivation according to the relevant proof system.<sup>9</sup>

Again be careful! The claim here is only that it should be effectively decidable whether an array of wffs presented as a well-constructed derivation really *is* a proper derivation. This is *not* to say that we can always decide in advance whether a derivation from given premisses exists to be discovered. Even in familiar first-order quantificational logic, for example, it is not in general decidable in advance whether there exists a proof from certain premisses to a given conclusion (we'll be proving this undecidability result later, in Section 31.5).

(c) Given an axiomatized formal theory  $T$ , then, it is decidable which wffs are its logical or non-logical axioms and which arrays of wffs conform to the derivation rules of  $T$ 's proof system. It will therefore also be decidable which arrays of wffs are axiomatic  $T$ -proofs – i.e. which arrays are properly constructed proofs, all of whose premisses are indeed  $T$ -axioms.

<sup>8</sup>We are not going to put any finite upper bound on the permissible length of proofs. So you might well ask: why not allow infinite arrays to count as proofs too? And indeed, there is some interest in theorizing about infinite proofs. For example, there are proof systems including the so-called  $\omega$ -rule, which says that from the infinite array of premisses  $\varphi(0), \varphi(1), \varphi(2), \dots, \varphi(n), \dots$  we can infer  $\forall x\varphi(x)$  where the quantifier runs over all natural numbers.

But do note that finite minds can't really take in the infinite number of separate premisses in an application of the  $\omega$ -rule: that's an impossible task. Hence, in so far as the business of formalization is primarily concerned to regiment and formalize the practices of ordinary mathematicians, albeit in an idealized way, it's natural at least to start by restricting ourselves to finite proofs, even if we don't put any contingent bound on the length of proofs.

<sup>9</sup>When did the idea clearly emerge that properties like being a wff or an axiom or a proof *ought* to be decidable? It was arguably already implicit in Hilbert's conception of rigorous proof. But Richard Zach has suggested that an early source for the *explicit* deployment of the idea is von Neumann (1927).

## 4 Axiomatized formal theories

---

So, to summarize these constraints on decidability:

$T$  is an (interpreted) axiomatized formal theory just if (i)  $T$  is couched in an (interpreted) formalized language  $\langle \mathcal{L}, \mathcal{I} \rangle$ , such that it is effectively decidable what counts as a wff/sentence of  $\mathcal{L}$ , and what the truth-condition of any sentence is, etc., (ii) it is effectively decidable which  $\mathcal{L}$ -wffs are axioms of  $T$ , and (iii)  $T$  uses a proof system such that it is effectively decidable whether an array of  $\mathcal{L}$ -wffs counts as conforming to the proof-building rules, and hence (iv) it is effectively decidable whether an array of  $\mathcal{L}$ -wffs counts as a proof from  $T$ 's axioms.

(d) The point perhaps needs highlighting. Logic books often define a theory very generously to be *any set of sentences*, or to be *any set of sentences which are consequences of some set of sentences*  $\Sigma$ . In this book, however, when we talk of axiomatized theories, we are really only going to be interested in (*effectively*) *axiomatized formal theories* in the sense of theories where it is *effectively decidable* what's in a set of axioms  $\Sigma$ . For it is only theories of this kind to which Gödel's theorems can be applied.

### 4.4 More definitions

Here are six standard definitions, specifically to do with theories (we've already met some of these notions more informally):

1. Given a derivation of the *sentence*  $\varphi$  from the axioms of the theory  $T$  using the background logical proof system, we will say that  $\varphi$  is a *theorem* of the theory. Using the standard abbreviatory symbol, we write:  $T \vdash \varphi$ .
2. A theory  $T$  is *sound* iff every theorem of  $T$  is true (i.e. true on the interpretation built into  $T$ 's language). Soundness is, of course, normally a matter of having true axioms and a truth-preserving proof system.
3. A theory  $T$  is *effectively decidable* iff the property of being a theorem of  $T$  is an effectively decidable property – i.e. iff there is a mechanical procedure for determining, for any given sentence  $\varphi$  of  $T$ 's language, whether  $T \vdash \varphi$ .
4. Assume now that  $T$  has a standard negation connective ' $\neg$ '. A theory  $T$  *decides* the sentence  $\varphi$  iff either  $T \vdash \varphi$  or  $T \vdash \neg\varphi$ . A theory  $T$  *correctly decides*  $\varphi$  just when, if  $\varphi$  is true (on the interpretation built into  $T$ 's language),  $T \vdash \varphi$ , and if  $\varphi$  is false,  $T \vdash \neg\varphi$ .
5. A theory  $T$  is *negation complete* iff  $T$  decides every sentence  $\varphi$  of its language (i.e. for every sentence  $\varphi$ , either  $T \vdash \varphi$  or  $T \vdash \neg\varphi$ ).
6.  $T$  is *inconsistent* iff for some sentence  $\varphi$ , we have both  $T \vdash \varphi$  and  $T \vdash \neg\varphi$ .

Note, it is convenient to restrict the theorems to the derivable *sentences*, wffs without free variables; but nothing really hangs on this.

Here's a very elementary toy example to illustrate some of these definitions. Consider a trivial pair of theories,  $T_1$  and  $T_2$ , whose shared language consists of the interpreted propositional atoms 'p', 'q', 'r' together with all the wffs that can be constructed from them using the familiar propositional connectives, whose shared underlying logic is a standard natural deduction system for propositional logic, and whose sets of axioms are respectively  $\{\neg p\}$  and  $\{\neg p, 'q', \neg r\}$ .  $T_1$  and  $T_2$  are then both axiomatized formal theories. For it is mechanically decidable what is a wff of the theory, and whether a purported proof is a proof from the given axioms. Both theories are consistent. Moreover, both are decidable theories; just use the truth-table test to determine whether a candidate theorem really follows from the axioms.

However, note that although  $T_1$  is a *decidable theory* that doesn't mean  $T_1$  *decides every wff*; it doesn't decide e.g. the wff  $(q \wedge r)$ , since  $T_1$ 's sole axiom doesn't entail either  $(q \wedge r)$  or  $\neg(q \wedge r)$ . To stress the point: it is one thing to have a general way of mechanically deciding what is a theorem; it is another thing for a theory to be negation complete, i.e. to have the resources to prove or disprove every wff.

By contrast,  $T_2$  *is* negation complete: any wff constructed from the three atoms using the truth-functional connectives has its truth-value decided, and the true ones can be proved and the false ones disproved.

Our toy example illustrates another crucial terminological point. You will be familiar with the idea of a deductive system being '(semantically) complete' or 'complete with respect to its standard semantics'. For example, a natural deduction system for propositional logic is said to be semantically complete when every inference which is semantically valid (i.e. truth-table valid) can be shown to be valid by a proof in the deductive system. But a theory's having a semantically complete logic is one thing, being a negation-complete theory is something else entirely. For example,  $T_1$  by hypothesis has a complete truth-functional *logic*, but is not a complete *theory*. For a more interesting example, we'll soon meet a formal arithmetic which we label 'Q'. This theory uses a standard quantificational deductive logic, which again is a (semantically) complete *logic*: but we can easily show that Q is not a (negation) complete *theory*.<sup>10</sup>

<sup>10</sup>Putting it symbolically may help. To say that a theory  $T$  with the set of axioms  $\Sigma$  is (negation) complete is to say that, for any sentence  $\varphi$ ,

$$\text{either } \Sigma \vdash \varphi \text{ or } \Sigma \vdash \neg\varphi;$$

while to say that a logic is (semantically) complete is to say that for any set of wffs  $\Sigma$  and any sentence  $\varphi$ ,

$$\text{if } \Sigma \models \varphi \text{ then } \Sigma \vdash \varphi,$$

where ' $\vdash$ ' signifies the relation of formal deducibility, and ' $\models$ ' signifies the relation of semantic consequence. As it happens, the first proof of the semantic completeness of a proof system for quantificational logic was also due to Gödel, and the result is often referred to as 'Gödel's Completeness Theorem' (Gödel, 1929). The topic of *that* theorem is therefore evidently not to be confused with the topic of his (First) Incompleteness Theorem: the semantic completeness

## 4 Axiomatized formal theories

---

Do watch out for this potentially dangerous double use of the term ‘complete’;<sup>11</sup> beware too of the use of ‘decidable’ and ‘decides’ for two not unconnected but significantly different ideas. These dual usages are now entirely entrenched: you just have to learn to live with them.

### 4.5 The effective enumerability of theorems

Deploying our notion of effective enumerability, we can now state and prove the following portmanteau theorem (the last claim is the crucial part):

**Theorem 4.1** *If  $T$  is an axiomatized formal theory then (i) the set of wffs of  $T$ , (i') the set of sentences of  $T$ , (ii) the set of proofs constructible in  $T$ , and (iii) the set of theorems of  $T$ , can each be effectively enumerated.*

*Proof sketch for (i)* By hypothesis,  $T$  has a formalized language with a finite basic alphabet.

But that implies we can give an algorithm for mechanically enumerating all the possible finite strings of symbols formed from a finite alphabet. For example, put the symbols of the finite alphabet into some order. Then start by listing all the strings of length 1, followed by all those of length 2 in ‘alphabetical order’, followed by all those of length 3 in ‘alphabetical order’, and so on and so forth.

By the definition of a formalized language, however, there is a mechanical procedure for as deciding we go along which of these symbol strings count as wffs. So, putting these mechanical procedures together, as we ploddingly enumerate all the possible strings we can throw away the non-wffs that turn up, leaving us with an effective enumeration of all the wffs.  $\square$

*Proof sketch for (i')* As for (i), replacing ‘wff’ by ‘sentence’.  $\square$

*Proof sketch for (ii)* Assume that  $T$ -proofs are linear sequences of wffs. Just as we can effectively enumerate all the possible wffs, so we can effectively enumerate all the possible finite sequences of wffs in some ‘alphabetical order’. One brute-force way is to start effectively enumerating all possible strings of symbols, and throw away any that isn't a sequence of wffs. By the definition of an axiomatized theory, there is then an algorithmic recipe for deciding which of these sequences of wffs are well-formed derivations from axioms of the theory. So as we go along

---

of a proof system for quantificational logic is one thing, the negation incompleteness of certain theories of arithmetic quite a different thing.

<sup>11</sup>The double use isn't a case of terminological perversity, though – even though logicians can be guilty of that! For there's the following parallel. A (negation) complete theory is one such that, if you add as a new axiom some proposition that can't already be derived in the theory, then the theory becomes useless by virtue of becoming inconsistent. Likewise a (semantically) complete deductive system is one such that, if you add a new logical axiom or new rule of inference that can't already be derived, then the logic becomes useless by virtue of warranting arguments that aren't semantically valid.

## 4.6 Negation-complete theories are decidable

---

we can mechanically select out these proof sequences from the other sequences of wffs, to give us an effective enumeration of all the possible proofs. (If  $T$ -proofs are more complex arrays of wffs – as in tree systems – then the construction of an effective enumeration of the arrays needs to be correspondingly more complex: but the core proof-idea remains the same.)  $\square$

*Proof sketch for (iii)* Start effectively enumerating the well-constructed proofs again. But this time, just record their conclusions when they pass the mechanical test for being closed sentences. This mechanically generated list now contains all and only the theorems of the theory.  $\square$

One comment on this (compare §3.3(c)). Be very clear that to say that the theorems of a formal axiomatized theory can be mechanically *enumerated* is not to say that the theory is *decidable*. It is one thing to have a mechanical method which is bound to generate any theorem eventually; it is quite another thing to have a mechanical method which, given an arbitrary wff  $\varphi$ , can determine – without going on for ever – whether  $\varphi$  will ever turn up on the list of theorems. Most interesting axiomatized theories are, as we'll see, *not* decidable.

## 4.6 Negation-complete theories are decidable

Despite that last point, however, we do have the following important result in the rather special case of negation-complete theories (compare Theorem 3.4):

**Theorem 4.2** *Any consistent, axiomatized, negation-complete formal theory  $T$  is decidable.*<sup>12</sup>

*Proof* We know from Theorem 4.1 that there's an algorithm for effectively enumerating the theorems of  $T$ . So to decide whether the sentence  $\varphi$  of  $T$ 's language is a  $T$ -theorem, start effectively listing the theorems, and do this until either  $\varphi$  or  $\neg\varphi$  turns up and then stop. If  $\varphi$  turns up, declare it to be a theorem. If  $\neg\varphi$  turns up, declare that  $\varphi$  is *not* a theorem.

Why does this work as a decision procedure? Well first, by hypothesis,  $T$  is negation complete, so either  $\varphi$  is a  $T$ -theorem or  $\neg\varphi$  is. So it is guaranteed that in a finite number of steps either  $\varphi$  or  $\neg\varphi$  will be produced in our enumeration of the theorems, and hence our 'do until' procedure terminates. And second, if  $\varphi$  is produced,  $\varphi$  is a theorem of course, while if  $\neg\varphi$  is produced, we can conclude that  $\varphi$  is not a theorem, since the theory is assumed to be consistent.

Hence, in this case, there *is* a dumbly mechanical procedure for deciding whether  $\varphi$  is a theorem.  $\square$

---

<sup>12</sup>By the way, it is trivial that an *inconsistent* axiomatized theory with a classical logic is decidable. For if  $T$  is inconsistent, every wff of  $T$ 's language is a theorem by the classical principle *ex contradictione quodlibet*. So all we have to do to determine whether  $\varphi$  is a  $T$ -theorem is to decide whether  $\varphi$  is a wff of  $T$ 's language, which by hypothesis you can if  $T$  is an axiomatized formal theory.

## 5 Capturing numerical properties

The previous chapter concerned axiomatized formal theories in general. This chapter introduces some key concepts we need in describing formal arithmetics in particular, notably the concepts of *expressing* and *capturing* numerical properties and functions. But we need to start with ...

### 5.1 Three remarks on notation

(a) Gödel's First Incompleteness Theorem is about the limitations of axiomatized formal theories of arithmetic: if a theory  $T$  is consistent and satisfies some other fairly minimal constraints, we can find arithmetical truths that can't be derived in  $T$ . Evidently, in discussing Gödel's result, it will be *very* important to be clear about when we are working 'inside' some specified formal theory  $T$  and when we are talking informally 'outside' that particular theory (e.g. in order to establish truths that  $T$  can't prove).

However, we do want our informal talk to be compact and perspicuous. Hence we will tend to borrow the standard logical notation from our formal languages for use in augmenting mathematical English (so, for example, we might write ' $\forall x \forall y (x + y = y + x)$ ' as a compact way of expressing the 'ordinary' arithmetic truth that the order in which you sum numbers doesn't matter).

Equally, we will want our formal wffs to be readable. Hence we will tend to use notation in building our formal languages that is already familiar from informal mathematics (so, for example, if we want to express the addition function in a formalized theory of arithmetic, we will use the usual sign '+', rather than some unhelpfully anonymous two-place function symbol like ' $f_3^2$ ').

This two-way borrowing of notation inevitably makes expressions of informal everyday arithmetic and their formal counterparts look very similar. And while context alone should make it pretty clear which is which, it is best to have a way of explicitly marking the distinction. To that end, *we will adopt the convention of using our ordinary type-face (mostly in italics) for informal mathematics, and using a sans-serif font for expressions in our formal languages.* Thus compare

$$\begin{array}{ll} 1 + 2 = 3 & 1 + 2 = 3 \\ \forall x \forall y (x + y = y + x) & \forall x \forall y (x + y = y + x) \\ \forall x \exists y y = Sx & \forall x \exists y y = Sx \end{array}$$

The expressions on the left will belong to our mathematicians'/logicians' augmented English (borrowing ' $S$ ' to mean 'the successor of'); the expressions on

the right are wffs – or abbreviations for wffs – of one of our formal languages, with the symbols chosen to be reminiscent of their intended interpretations.

(b) In addition to *italic symbols* for informal mathematics and **sans-serif symbols** for formal wffs, we also need another layer of symbols. For example, we need a compact way of generalizing about formal expressions, as when we defined negation completeness in Section 4.4 by saying that for any sentence  $\varphi$ , the theory  $T$  entails either  $\varphi$  or its negation  $\neg\varphi$ . We’ll standardly use Greek letters for this kind of ‘metalinguistic’ duty. So note that Greek letters will never belong to our formal languages themselves: these symbols belong to logicians’ augmented English.

What exactly is going on, then, when we are talking about a formal language  $L$  and say e.g. that the negation of  $\varphi$  is  $\neg\varphi$ , when we are apparently mixing a symbol from augmented English with a symbol from  $L$ ? Answer: there are hidden quotation marks, and ‘ $\neg\varphi$ ’ is to be read as meaning ‘the expression that consists of the negation sign “ $\neg$ ” followed by  $\varphi$ ’.

(c) Sometimes, when being *very* pedantic, logicians use so-called Quine-quotes when writing mixed expressions which contain both formal and metalinguistic symbols (thus:  $\ulcorner\neg\varphi\urcorner$ ). But this is excessive. We are not going to bother, and no one will get confused by our more casual (and entirely normal) practice. In any case, we’ll want to use corner-quotes later for a different purpose.

We’ll be pretty relaxed about ordinary quotation marks too. We’ve so far been rather punctilious about using them when mentioning, as opposed to using, wffs and other formal expressions. But from now on, we will normally drop them other than around single symbols. Again, no confusion should ensue.

Finally, we will also be relaxed about dropping unnecessary brackets in formal expressions (and we’ll cheerfully change the shape of pairs of brackets, and even occasionally insert redundant ones, when that aids readability).

## 5.2 The language $L_A$

Now to business. There is no single language which could reasonably be called *the* language for formal arithmetic: rather, there is quite a variety of different languages, apt for framing theories of different strengths.

However, the core theories of arithmetic which we’ll be discussing most are framed in the interpreted language  $L_A = \langle \mathcal{L}_A, \mathcal{I}_A \rangle$ , which is a formalized version of what we called ‘the language of basic arithmetic’ in Section 1.1. So let’s concentrate for the moment on characterizing this simple language.

(a) *Syntax*  $\mathcal{L}_A$  has a standard first-order syntax, with one built-in two-place predicate and three built-in function expressions. Thus,

1. The *logical* vocabulary of  $\mathcal{L}_A$  comprises the usual connectives and brackets, an inexhaustible supply of variables (including, let’s suppose, ‘a’ to

## 5 Capturing numerical properties

---

‘d’, ‘u’ to ‘z’), and the usual first-order quantifiers. It also has the identity symbol ‘=’ (which will be the sole atomic predicate). The finer details here, however, are not critical.

2. The *non-logical* vocabulary of  $\mathcal{L}_A$  is  $\{0, S, +, \times\}$ , where
  - i. ‘0’ is a constant;
  - ii. ‘S’ is a one-place function-expression (read ‘the successor of’);
  - iii. ‘+’ and ‘ $\times$ ’ are two-place function-expressions.

For readability, we’ll allow ourselves to write e.g.  $(a + b)$  and  $(a \times b)$  rather than  $+(a, b)$  and  $\times(a, b)$ .

3. A *term* of  $\mathcal{L}_A$  is an expression that you can build up from ‘0’ and/or variables using the successor function ‘S’, addition and multiplication – as in  $SSS0$ ,  $(S0 + x)$ ,  $(SSS0 \times (Sx + y))$ , and so on. Putting it more carefully,
  - i. ‘0’ is a term, as is any variable.
  - ii. If  $\sigma$  and  $\tau$  are terms, so are  $S\sigma$ ,  $(\sigma + \tau)$ ,  $(\sigma \times \tau)$ .
  - iii. Nothing else is a term.

The *closed* terms are the variable-free terms, such as the numerals we introduce next.

4. A (*standard*) *numeral* of  $\mathcal{L}_A$  is a term built up from our single constant ‘0’ using just the successor function, i.e. they are expressions of the form  $SS\dots S0$  with zero or more occurrences of ‘S’.<sup>1</sup> We’ll abbreviate the numerals  $S0$ ,  $SS0$ ,  $SSS0$ , etc. by ‘1’, ‘2’, ‘3’, etc. Further, when we want to generalize, we’ll write e.g. ‘ $\bar{n}$ ’ to indicate the standard numeral  $SS\dots S0$  with  $n$  occurrences of ‘S’. (Overlining is conventional, and helpfully distinguishes numerals from variables.)
5. Since the only predicate built into  $\mathcal{L}_A$  is the identity sign, the only possible *atomic wffs* have the form  $\sigma = \tau$ , where again  $\sigma$  and  $\tau$  are terms. Then the *wffs* are formed from atomic wffs in the entirely standard way, by using connectives and quantifiers. Note, for convenience we are allowing wffs with free variables.

(b) *Semantics* The basic features of the interpretation  $\mathcal{I}_A$  are just as you would expect. The intended domain of quantification is the natural numbers. ‘0’ denotes

---

<sup>1</sup>In using ‘S’ rather than ‘s’, we depart from the normal logical practice which we follow elsewhere of using upper-case letters for predicates and lower-case letters for functions: but this particular departure is sanctioned by aesthetics and common usage.

A very common alternative convention you should know about even though we won’t employ it here is to use a postfixed prime as the symbol for the successor function; in that notation the standard numerals are then  $0$ ,  $0'$ ,  $0''$ ,  $0'''$ ,  $\dots$ .

0; and we interpret ‘S’, ‘+’, ‘×’ as the successor (i.e. add one) function, addition and multiplication. And because of the special circumstance that every number in the domain has a numeral that denotes it, the semantic rules are particularly simple (e.g. we don’t need to do the trick of using variables as temporary names that we mentioned in §4.2(c)). We can run the story like this:

1. First note that  $\mathcal{I}_A$  assigns values to closed terms as follows:
  - i. The value of ‘0’ is zero. Or in an obvious shorthand,  $val[0] = 0$ .
  - ii. If  $\tau$  is a closed term, then  $val[S\tau] = val[\tau] + 1$ .
  - iii. If  $\sigma$  and  $\tau$  are closed terms, then  $val[(\sigma + \tau)] = val[\sigma] + val[\tau]$ , and  $val[(\sigma \times \tau)] = val[\sigma] \times val[\tau]$ .

It immediately follows from (i) and (ii), by the way, that numerals have the values that they should have, i.e. for all  $n$ ,  $val[\bar{n}] = n$ .

2. The atomic sentences of  $\mathcal{L}_A$  must all have the form  $\sigma = \tau$ , where  $\sigma$  and  $\tau$  are terms. And given the standard reading of the identity relation, it is immediate that

A sentence of the form  $\sigma = \tau$  is true iff  $val[\sigma] = val[\tau]$ .

3. Molecular sentences built up using the truth-functional connectives are treated in the familiar ways. And then, because every number  $n$  has a numeral  $\bar{n}$  to pick it out, we can put the rule for the existential quantifier very simply like this (without in fact explicitly mentioning the domain of quantification):

A sentence of the form  $\exists\xi\varphi(\xi)$  (where ‘ $\xi$ ’ can be any variable) is true iff, for some number  $n$ ,  $\varphi(\bar{n})$  is true.

Similarly

A sentence of the form  $\forall\xi\varphi(\xi)$  is true iff, for any  $n$ ,  $\varphi(\bar{n})$  is true.

And then it is easy to see that  $\mathcal{I}_A$  will, as we want, effectively assign a unique truth-condition to every  $\mathcal{L}_A$  sentence.

### 5.3 A quick remark about truth

The semantics  $\mathcal{I}_A$  entails that the sentence  $(1 + 2) = 3$ , i.e.  $(S0 + SS0) = SSS0$ , is true just so long as one plus two is three. Likewise the sentence  $\exists v 4 = (v \times 2)$ , i.e.  $\exists v SSSS0 = (v \times SS0)$ , is true just so long as there is some number such that four is twice that number (i.e. so long as four is even). But, by any normal arithmetical standards, one plus two *is* three, and four *is* even. So by the same workaday standards, those two  $\mathcal{L}_A$ -sentences are indeed true.

## 5 Capturing numerical properties

---

Later, when we come to present Gödel's Theorems, we'll describe how to take an arithmetical theory  $T$  built in the language  $L_A$ , and construct a sentence  $G_T$  which turns out to be 'true but unprovable-in- $T$ '. And while the sentence in question is a bit exotic, there is nothing in the least exotic about the notion of truth being applied to it here either: it is the very same workaday notion we've just so simply explained.  $\mathcal{I}_A$  explicitly defines what it takes for *any*  $L_A$ -sentence, however complex, to be true in this humdrum sense.

Now there are, to be sure, philosophers who will say that no  $L_A$ -sentence *is* strictly speaking true in the humdrum sense – because they are equally prepared to say that, speaking really strictly, one plus two *isn't* three and four *isn't* even.<sup>2</sup> Such common-or-garden arithmetic claims, they aver, presuppose the existence of numbers as mysterious kinds of objects in some Platonic heaven, and they are sceptical about the literal existence of such things. In the view of many of these philosophers, arithmetical entities should be thought of as useful *fictions*: and, at least when we are on our very best behaviour, we really ought not to claim that one plus two equals three but only that *in the arithmetical fiction* one plus two equals three.

We can't, however, tangle with this rather popular philosophical view here: and fortunately we needn't do so, for the issues it raises are quite orthogonal to our main concerns in this book. Fictionalists about arithmetic can systematically read our talk of various  $L_A$  sentences being true in their favoured way – i.e. as talk 'within the arithmetical fiction'. It won't significantly affect the proofs and arguments that follow.

### 5.4 Expressing numerical properties and functions

A competent formal theory of arithmetic should surely be able to talk about a lot more than just the successor function, addition and multiplication. But 'talk about' *how*?

(a) Let's assume for the moment that we are dealing with a theory built in the rather minimal language  $L_A$ . So, for a first example, consider  $L_A$ -sentences of the type

$$1. \exists v(2 \times v = \bar{n}).$$

For  $n = 4$ , for example, this unpacks into ' $\exists v(SS0 \times v = SSSS0)$ '. Abbreviate such a wff by  $\psi(\bar{n})$ . Then it is obvious that, for any  $n$ ,

- if  $n$  is even, then  $\psi(\bar{n})$  is true,
- if  $n$  isn't even, then  $\neg\psi(\bar{n})$  is true,

where we mean, of course, true on the arithmetic interpretation built into  $L_A$ .

---

<sup>2</sup>See e.g. Field (1989, Ch. 1) and Balaguer (1998) for discussion.

## 5.4 Expressing numerical properties and functions

---

So consider the corresponding open wff<sup>3</sup>  $\psi(x)$  with one free variable, i.e.

$$1'. \exists v(2 \times v = x).$$

This is, as the logicians say, satisfied by the number  $n$  just when  $\psi(\bar{n})$  is true, i.e. just when  $n$  is even. Or to put it another way,  $\psi(x)$  has the set of even numbers as its extension. Which means that our open wff expresses the property *even*, at least in the sense of being true of the right objects, i.e. having the right extension.

Another example:  $n$  has the property of being prime iff it is greater than one, and its only factors are one and itself. Or equivalently,  $n$  is prime just in case it is not 1, and of any two numbers that multiply to give  $n$ , one of them must be 1. So consider wffs of the type

$$2. (\bar{n} \neq 1 \wedge \forall u \forall v (u \times v = \bar{n} \rightarrow (u = 1 \vee v = 1)))$$

(where we use  $\alpha \neq \beta$  for  $\neg\alpha = \beta$ ). Abbreviate such a wff by  $\chi(\bar{n})$ . Then  $\chi(\bar{n})$  holds just in case  $n$  is prime, i.e. for every  $n$ ,

if  $n$  is prime, then  $\chi(\bar{n})$  is true,  
if  $n$  isn't prime, then  $\neg\chi(\bar{n})$  is true.

The corresponding open wff  $\chi(x)$

$$2'. (x \neq 1 \wedge \forall u \forall v (u \times v = x \rightarrow (u = 1 \vee v = 1)))$$

is therefore satisfied by exactly the prime numbers. In other words,  $\chi(x)$  expresses the property *prime*, again in the sense of having the right extension.

In this sort of way, a formal language like  $L_A$  with minimal basic resources can in fact come to express a whole variety of arithmetical properties by means of complex open wffs with the right extensions. And our examples motivate the following official definition that in fact applies to *any* language  $L$  in which we can form the standard numerals:

A numerical property  $P$  is *expressed* by the open wff  $\varphi(x)$  with one free variable in an arithmetical language  $L$  iff, for every  $n$ ,  
if  $n$  has the property  $P$ , then  $\varphi(\bar{n})$  is true,<sup>4</sup>  
if  $n$  does not have the property  $P$ , then  $\neg\varphi(\bar{n})$  is true.

‘True’ of course continues to mean true on the given interpretation built into  $L$ .

(b) We can now extend our definition in the obvious way to cover relations. Note, for example, that in a language like  $L_A$

$$3. \psi(\bar{m}, \bar{n}) =_{\text{def}} \exists v (v + \bar{m} = \bar{n})$$

---

<sup>3</sup>Usage varies: in this book, an open wff is one which isn't closed, i.e. which has at least one free variable, though it might have other bound variables.

<sup>4</sup>Do we need to spell it out?  $\varphi(\bar{n})$  is of course the result of substituting the numeral for  $n$  for each occurrence of ‘ $x$ ’ in  $\varphi(x)$ .

## 5 Capturing numerical properties

---

is true just in case  $m \leq n$ . And so it is natural to say that the corresponding expression

$$3'. \psi(x, y) =_{\text{def}} \exists v(v + x = y)$$

expresses the relation *less-than-or-equal-to*, in the sense of getting the extension right. Generalizing again:

A two-place numerical relation  $R$  is expressed by the open wff  $\varphi(x, y)$  with two free variables in an arithmetical language  $L$  iff, for any  $m, n$ ,

if  $m$  has the relation  $R$  to  $n$ , then  $\varphi(\bar{m}, \bar{n})$  is true,

if  $m$  does not have the relation  $R$  to  $n$ , then  $\neg\varphi(\bar{m}, \bar{n})$  is true.

Likewise for many-place relations.<sup>5</sup>

(c) Let's highlight again that 'expressing' in our sense is just a matter of getting the extension right. Suppose  $\varphi(x)$  expresses the property  $P$  in  $L$ , and let  $\theta$  be any true  $L$ -sentence. Then whenever  $\varphi(\bar{n})$  is true so is  $\varphi(\bar{n}) \wedge \theta$ . And whenever  $\varphi(\bar{n})$  is false so is  $\varphi(\bar{n}) \wedge \theta$ . Which means that  $\varphi'(x) =_{\text{def}} \varphi(x) \wedge \theta$  also expresses  $P$  – irrespective of what  $\theta$  means.

Hence, we might say,  $\varphi(x)$ 's expressing  $P$  in our sense can really only be a necessary condition for its expressing that property in the more intuitive sense of having the right meaning. However, the intuitive notion is murky and notoriously difficult to analyse (even if we can usually recognize wffs which 'express the right meaning' when we meet them). By contrast, our notion is sharply defined. And, blunt instrument though it is, it will serve us perfectly well for most purposes.

(d) Let's say that a two-place numerical relation  $R$  is *functional* iff for every  $m$  there is one and only one  $n$  such that  $Rmn$ . Then evidently any one-place numerical function  $f$  has an associated two-place functional relation  $R_f$  such

---

<sup>5</sup>A pernicky footnote for very-well-brought-up logicians. We could have taken the canonical way of expressing a monadic property to be not a complete wff  $\varphi(x)$  with a free variable but a predicative expression  $\varphi(\xi)$  – where ' $\xi$ ' here isn't a variable but a metalinguistic *place-holder*, marking a *gap* to be filled by a term (i.e. by a name or variable). Similarly, we could have taken the canonical way of expressing a two-place relation to be a doubly gappy predicative expression  $\varphi(\xi, \zeta)$ , etc.

Now, there are technical and philosophical reasons for rather liking the gappy notation to express properties and relations. However, it is certainly the default informal mathematical practice to prefer to use complete expressions with free variables rather than expressions with place-holders which mark gaps. Sticking to this practice, as we do in this book, therefore makes for a more familiar-looking notation and hence aids readability. (Trust me! – I did at one stage try writing this book systematically using Greek letters as place-holders, and some passages would look quite unnecessarily repellent to the ordinary mathematician's eye.)

Still, there's a wrinkle. Just once, in Section 13.3, we'll want to talk about the expressive power of a theory whose language lacks quantifiers and variables, so in particular lacks expressions with free variables. In that special context, you'll have to treat any implicit reference to expressions of the form  $\varphi(x, y)$  as a cheerful abuse of notation, with the apparent variables really functioning as place-holders, so *there* we really do mean what we should better write as  $\varphi(\xi, \zeta)$  and so on.

## 5.5 Capturing numerical properties and functions

---

that  $f(m) = n$  just in case  $R_f mn$ . And if, as is standard, you say the extension of a one-place function  $f$  is the set of pairs  $\langle m, n \rangle$  such that  $f(m) = n$ , then  $f$  has the *same* extension as its corresponding functional relation  $R_f$ .

If we are interested, then, in expressing functions in the same sense of getting their extensions right, then expressing a one-place function  $f$  and expressing the corresponding two-place functional relation  $R_f$  will come to just the same.<sup>6</sup> Which motivates the following definition:

A one-place numerical function  $f$  is expressed by the open wff  $\varphi(x, y)$  in an arithmetical language  $L$  just if, for any  $m, n$ ,  
     if  $f(m) = n$ , then  $\varphi(\bar{m}, \bar{n})$  is true,  
     if  $f(m) \neq n$ , then  $\neg\varphi(\bar{m}, \bar{n})$  is true.

Likewise for many-place functions.

(e) We should perhaps confirm that  $L$  can express a property  $P$  iff it can express  $P$ 's characteristic function  $c_P$ . But we have

1. If  $\varphi(x)$  expresses  $P$ , then  $(\varphi(x) \wedge y = 0) \vee (\neg\varphi(x) \wedge y = 1)$  expresses  $c_P$ .
2. If  $\psi(x, y)$  expresses  $c_P$ , then  $\psi(x, 0)$  expresses  $P$ .

*Proof* For (1), suppose  $\varphi(x)$  expresses  $P$ . Then if  $c_P(m) = 0$ , then  $m$  is  $P$ , so by our supposition  $\varphi(\bar{m})$  must be true, so  $(\varphi(\bar{m}) \wedge 0 = 0) \vee (\neg\varphi(\bar{m}) \wedge 0 = 1)$  is true.

And if  $c_P(m) \neq 0$ , then  $m$  is not  $P$ , so  $\neg\varphi(\bar{m})$  must be true, so both disjuncts of  $(\varphi(\bar{m}) \wedge 0 = 0) \vee (\neg\varphi(\bar{m}) \wedge 0 = 1)$  are false, so the negation of the whole is true.

Similarly for the remaining cases  $c_P(m) = 1$ ,  $c_P(m) \neq 1$ . And the four cases together verify (1).

(2) is trivial, so we are done. □

## 5.5 Capturing numerical properties and functions

(a) Of course, we don't merely want various numerical properties, relations and functions to be *expressible* in the language of a formal theory of arithmetic. We also want to be able to use the theory to *prove* facts about which numbers have which properties or stand in which relations (more carefully: we want formal derivations which will be proofs in the intuitive sense if can we take it that the axioms are indeed secure truths). We likewise want to be able to use the theory to *prove* facts about the values of various functions for given inputs.

---

<sup>6</sup>Some identify a function with its extension, and also identify a relation with *its* extension. Arguably that's a mistake. But if you take that line, then you will and say that a function  $f$  and its corresponding functional relation  $R_f$  are in fact the very *same* thing. However, we aren't assuming that here.

## 5 Capturing numerical properties

---

Now, it is a banal observation that to establish facts about *individual* numbers typically requires less sophisticated proof-techniques than proving general truths about *all* numbers. So let's focus here on the relatively unambitious task of case-by-case proving that particular numbers have or lack a certain property. This level of task is reflected in the following general definition concerning formal provability:

The theory  $T$  captures the property  $P$  by the open wff  $\varphi(x)$  iff, for any  $n$ ,

- if  $n$  has the property  $P$ , then  $T \vdash \varphi(\bar{n})$ ,
- if  $n$  does not have the property  $P$ , then  $T \vdash \neg\varphi(\bar{n})$ .

For example, in theories of arithmetic  $T$  with very modest axioms, the wff  $\psi(x) =_{\text{def}} \exists v(2 \times v = x)$  not only expresses but captures the property *even*. In other words, for each even  $n$ ,  $T$  can prove  $\psi(\bar{n})$ , and for each odd  $n$ ,  $T$  can prove  $\neg\psi(\bar{n})$ . Likewise, in the same theories, the wff  $\chi(x)$  from the previous section not only expresses but captures the property *prime*.

We can extend the notion of 'capturing' to the case of relations in the entirely predictable way:

The theory  $T$  captures the two-place relation  $R$  by the open wff  $\varphi(x, y)$  iff, for any  $m, n$ ,

- if  $m$  has the relation  $R$  to  $n$ , then  $T \vdash \varphi(\bar{m}, \bar{n})$ ,
- if  $m$  does not have the relation  $R$  to  $n$ , then  $T \vdash \neg\varphi(\bar{m}, \bar{n})$ .

Likewise for many-place relations.

(b) What about 'capturing' a function in the same sense of getting the extension right? Given the tightness of the link between a function and the corresponding functional relation, the obvious thing to say is:

A one-place numerical function  $f$  is captured by the open wff  $\varphi(x, y)$  in theory  $T$  just if, for any  $m, n$ ,

- if  $f(m) = n$ , then  $T \vdash \varphi(\bar{m}, \bar{n})$ ,
- if  $f(m) \neq n$ , then  $T \vdash \neg\varphi(\bar{m}, \bar{n})$ .

The generalization to many-place functions is immediate. (Later, we will return to consider ways we might want to strengthen the notion of capturing a function: but for the moment this initial definition will give us what we want.)

(c) We should add a comment which parallels the point we made about 'expressing'. Suppose  $\varphi(x)$  captures the property  $P$  in  $T$ , and let  $\theta$  be any  $T$ -theorem. Then whenever  $T \vdash \varphi(\bar{n})$ , then  $T \vdash \varphi(\bar{n}) \wedge \theta$ . And whenever  $T \vdash \neg\varphi(\bar{n})$ , then  $T \vdash \neg(\varphi(\bar{n}) \wedge \theta)$ . Which means that  $\varphi'(x) =_{\text{def}} \varphi(x) \wedge \theta$  also captures  $P$  – irrespective of  $\theta$ 's content.

Hence, we might say,  $\varphi(x)$ 's capturing  $P$  in our initial sense is just a necessary condition for its capturing that property in the more intuitive sense of proving

## 5.6 Expressing vs. capturing: keeping the distinction clear

wffs with the right meaning. Likewise for the idea of capturing a function (even if we adopt the later stronger definitions). But again the intuitive notion is murky, and our sharply defined notions will serve our purposes.

(d) Again, we should perhaps confirm that  $T$  can capture a property  $P$  iff it can capture  $P$ 's characteristic function  $c_P$ . But, assuming  $T \vdash 0 \neq 1$  and has a modicum of propositional logic, we have

1. If  $\varphi(x)$  captures  $P$  in  $T$ , then  $(\varphi(x) \wedge y = 0) \vee (\neg\varphi(x) \wedge y = 1)$  captures  $c_P$ .
2. If  $\psi(x, y)$  captures  $c_P$  in  $T$ , then  $\psi(x, 0)$  captures  $P$ .

*Proof sketch* Suppose  $\varphi(x)$  captures  $P$  in  $T$ . If  $c_P(m) = 0$ , then  $m$  is  $P$ , so by our supposition  $T \vdash \varphi(\bar{m})$ , hence  $T \vdash (\varphi(\bar{m}) \wedge 0 = 0) \vee (\neg\varphi(\bar{m}) \wedge 0 = 1)$ .

And if  $c_P(m) \neq 0$ , then  $m$  is not  $P$ , so  $T \vdash \neg\varphi(\bar{m})$ . Hence  $T \vdash \neg(\varphi(\bar{m}) \wedge 0 = 0)$  and also  $T \vdash \neg(\neg\varphi(\bar{m}) \wedge 0 = 1)$ . Hence  $T \vdash \neg(\varphi(\bar{m}) \wedge 0 = 0) \vee (\neg\varphi(\bar{m}) \wedge 0 = 1)$ .

Exercise: now complete the proof of (1) and (2). ☒

## 5.6 Expressing vs. capturing: keeping the distinction clear

The jargon we've used here in talking of a theory's 'capturing' a numerical property or numerical function is frankly a bit deviant. But terminology here varies very widely anyway. Perhaps most commonly these days, logicians talk of  $P$  being 'represented' by a wff  $\varphi(x)$  satisfying our conditions for capture. But I'm unapologetic: 'capture' is very helpfully mnemonic for 'case-by-case prove'.

But whatever your favoured jargon, the key thing is to be absolutely clear about the distinction we need to mark – so let's highlight it again. Sticking to the case of properties,

1. whether  $P$  is *expressible* in a given theory just depends on the richness of that theory's *language*;
2. whether a property  $P$  can be *captured* by the theory depends on the richness of its *axioms* and *proof system*.<sup>7</sup>

---

<sup>7</sup>'Expresses' for properties and relations is used in our way by e.g. Smullyan (1992, p. 19). As alternatives, we find e.g. 'arithmetically defines' (Boolos et al., 2002, p. 199), or simply 'defines' (Leary 2000, p. 130; Enderton 2002, p. 205).

Gödel originally talked of a numerical relation being 'decidable' (*entscheidungsdefinit*) when it is captured by an arithmetical wff (Gödel, 1931, p. 176). As later alternatives to our 'captures' we find 'numeralwise expresses' (Kleene 1952, p. 195; Fisher 1982, p. 112), and also simply 'expresses'(!) again (Mendelson, 1997, p. 170), 'formally defines' (Tourelakis, 2003, p. 180) and plain 'defines' (Boolos et al., 2002, p. 207). At least 'binumerate' – (Smoryński 1977, p. 838; Lindström 2003, p. 9) – won't cause confusion. But as noted, 'represents' (although it is perhaps too close for comfort to 'expresses') seems the most common choice in recent texts: see e.g. Leary (2000, p. 129), Enderton (2002, p. 205), Cooper (2004, p. 56).

And when it comes to talking of expressing vs capturing for functions, there's the same terminological confusion, though this time compounded by the fact that (as we noted) that

## 5 Capturing numerical properties

---

Note that expressibility does not imply capturability: indeed, we will prove later that – for any respectable theory of arithmetic  $T$  – there are numerical properties that are expressible in  $T$ 's language but not capturable by  $T$  (see e.g. Section 22.4). However, there *is* a link in the other direction. Suppose  $T$  is a *sound* theory of arithmetic, i.e. one whose theorems are all true on the given arithmetic interpretation of its language. Hence if  $T \vdash \varphi(\bar{n})$ , then  $\varphi(\bar{n})$  is true. And if  $T \vdash \neg\varphi(\bar{n})$ , then  $\neg\varphi(\bar{n})$  is true. Which immediately entails that *if  $\varphi(x)$  captures  $P$  in the sound theory  $T$ , then  $\varphi(x)$  expresses  $P$ .*

---

there are also stronger definitions of the idea of capturing functions that, for certain purposes, it will later be useful to have in play,

The moral is plain: *when reading other discussions, always very carefully check the local definitions of the jargon!*

## 6 The truths of arithmetic

In Chapter 4, we proved that the theorems of any effectively axiomatized theory *can* be effectively enumerated. In this chapter, we prove by contrast that the truths of any language which is sufficiently expressive of arithmetic *can't* be effectively enumerated (we will explain in just a moment what 'sufficiently expressive' means). As we'll see, it immediately follows that a sound axiomatized theory with a sufficiently expressive language can't be negation complete!

### 6.1 Sufficiently expressive languages

Recall: a one-place numerical function  $f$  can be expressed in language  $L$  just when there is an open  $L$ -wff  $\varphi$  such that  $\varphi(\bar{m}, \bar{n})$  is true iff  $f(m) = n$  (Section 5.4). We will now say that

An interpreted formal language  $L$  is *sufficiently expressive* iff (i) it can express every effectively computable one-place numerical function, and (ii) it can form wffs which quantify over numbers.

Clause (ii) means, of course, that  $L$  must have quantifiers. But – unless the domain of  $L$ 's built-in interpretation is already just the natural numbers –  $L$  will also need to be able to form a predicate we'll abbreviate  $\text{Nat}(x)$  which picks out (whatever plays the role of) the numbers in  $L$ 's domain. E.g. we need ' $\exists x(\text{Nat}(x) \wedge \varphi(x))$ ' to be available to say that some number satisfies the condition expressed by  $\varphi$ .

As we've just announced, we are going to show that sound axiomatized theories with sufficiently expressive languages can't be negation complete. But of course, that wouldn't be an interesting result if a theory's having a sufficiently expressive language were a peculiarly tough condition to meet. But it isn't. Much later in this book, in Section 31.1, we'll show that even  $L_A$ , the language of basic arithmetic, is sufficiently expressive.

However, we can't yet establish this claim about  $L_A$ : doing that would obviously require having a general theory of computable functions, and we haven't got one yet. For the moment, then, we'll just *assume* that theories with sufficiently expressive languages are worth thinking about, and see what follows.<sup>1</sup>

---

<sup>1</sup>Though perhaps we can do rather better than mere assumption even at this point, for consider the following line of argument. Suppose we have a suitably programmed general purpose computer  $M$  which computes  $f$ . Now imagine that we use numerical coding to associate programs and descriptions of the  $M$ 's memory states, outputs etc. with numbers. Then we can encode claims about  $M$ 's performance as it evaluates  $f(m)$ . There will, in particular, be a

## 6.2 The truths of a sufficiently expressive language

It now quickly follows, as we initially announced, that

**Theorem 6.1** *The set of truths of a sufficiently expressive language  $L$  is not effectively enumerable.*

*Proof* Take the argument in stages. (i) The Basic Theorem about e.e. sets of numbers (Theorem 3.8) tells us that there a set  $K$  which is e.e. but whose complement  $\overline{K}$  isn't. Suppose the effectively computable function  $f$  enumerates it, so  $n \in K$  iff  $\exists x f(x) = n$ , with the variable running over numbers.

(ii) Since  $f$  is effectively computable, then in any given sufficiently expressive arithmetical language  $L$  there will be some wff of  $L$  which expresses  $f$ : let's abbreviate that wff  $F(x, y)$ . Then  $f(m) = n$  just when  $F(\overline{m}, \overline{n})$  is true.

(iii) By definition, a sufficiently expressive language can form wffs which quantify over numbers. So  $\exists x f(x) = n$  just when  $\exists x(\text{Nat}(x) \wedge F(x, \overline{n}))$  is true (where  $\text{Nat}(x)$  stands in for whatever  $L$ -predicate might be needed to explicitly restrict  $L$ 's quantifiers to numbers).

(iv) So from (i) and (iii) we have

$n \in K$  if and only if  $\exists x(\text{Nat}(x) \wedge F(x, \overline{n}))$  is true; and therefore,  
 $n \in \overline{K}$  if and only if  $\neg \exists x(\text{Nat}(x) \wedge F(x, \overline{n}))$  is true.

(v) Now suppose for a moment that the set  $\mathcal{T}$  of true sentences of  $L$  is effectively enumerable. Then, given a description of the expression  $F$ , we could run through the supposed effective enumeration of  $\mathcal{T}$ , and whenever we come across a truth of the type  $\neg \exists x(\text{Nat}(x) \wedge F(x, \overline{n}))$  for some  $n$  – and it will be effectively decidable if a wff has that particular syntactic form – list the number  $n$ . That procedure would give us an effectively generated list of all the members of  $\overline{K}$ .

(vi) But by hypothesis  $\overline{K}$  is *not* effectively enumerable. So  $\mathcal{T}$  can't be effectively enumerable after all. Which is what we wanted to show.  $\square$

## 6.3 Unaxiomatizability

Here's a first easy corollary. We need a simple definition:

A set of wffs  $\Sigma$  is *axiomatizable* iff there is an effectively axiomatized formal theory  $T$  such that, for any wff  $\varphi$ ,  $\varphi \in \Sigma$  if and only if  $T \vdash \varphi$  (i.e.  $\Sigma$  is the set of  $T$ -theorems).

Then it is immediate that

---

statement  $\varphi(\overline{m}, \overline{n})$  in an arithmetical language  $L$  which encodes the claim that  $M$  gives output  $n$  on input  $m$ , so  $\varphi(\overline{m}, \overline{n})$  is true just when  $f(m) = n$ . Hence  $\varphi$  expresses  $f$ , and  $L$  is sufficiently expressive, given  $L$  is rich enough to have the resources to code up descriptions of the behaviour of programs in general purpose computers. However, it should seem fairly plausible that such coding needn't take *very* much arithmetical language – as we'll indeed confirm in later chapters.

**Theorem 6.2** *The set  $\mathcal{T}$  of true sentences of a sufficiently expressive language  $L$  is not axiomatizable.*

*Proof* Suppose otherwise, i.e. suppose that  $T$  is an axiomatized formal theory, framed in a sufficiently expressive language  $L$ , such that the  $T$ -theorems are just the truths  $\mathcal{T}$  expressible in that language. Then, because it is properly axiomatized,  $T$ 's theorems could be effectively enumerated (by the last part of Theorem 4.1). That is to say, the truths  $\mathcal{T}$  of  $L$  could be effectively enumerated, contrary to Theorem 6.1. Hence there can be no such theory as  $T$ .  $\square$

## 6.4 An incompleteness theorem

Suppose we build an axiomatized formal theory  $T$  in a sufficiently expressive language  $L$ . Then because it is axiomatized,  $T$ 's theorems *can* be effectively enumerated. On the other hand, because  $T$ 's language is sufficiently expressive, the truths expressible in its language *cannot* be effectively enumerated. There is therefore a mismatch between the truths and the  $T$ -theorems here.

Now suppose that  $T$  is also a *sound* theory, i.e. its theorems are all true. The mismatch between the truths and the  $T$ -provable sentences must then be due to there being truths which  $T$  can't prove. Suppose  $\varphi$  is one of these. Then  $T$  doesn't prove  $\varphi$ . And since  $\neg\varphi$  is false,  $T$  doesn't prove that either. Hence we our first version of an incompleteness theorem:

**Theorem 6.3** *If  $T$  is a sound axiomatized theory whose language is sufficiently expressive, then  $T$  cannot be negation complete.*

But we announced – though of course at this stage we can't yet prove – that even the minimal language  $L_A$  is sufficiently expressive, so any more inclusive language will be too: *hence a huge number of interesting theories which can express some arithmetic must be incomplete if sound.*

Astonishing! We have reached an arithmetical incompleteness theorem already. And note, by the way, that we can't patch  $T$  up and make it complete by adding more true axioms and/or a richer truth-preserving logic to get another properly axiomatized theory  $T'$ . For  $T'$  will still be sound, still have a sufficiently expressive language, and hence still be incomplete. So we could equally well call Theorem 6.3 an *incompleteness* theorem.

The great mathematician Paul Erdős had the fantasy of The Book in which God keeps the neatest and most elegant proofs of mathematical theorems. Our sequence of proofs of Theorems 3.8 and 6.1 and now of our first incompleteness result Theorem 6.3 surely belongs in The Book.<sup>2</sup>

<sup>2</sup>For more on Erdős's conceit of proofs from The Book, see Aigner and Ziegler (2004).

## 7 Sufficiently strong arithmetics

Theorem 6.3, our first shot at an incompleteness theorem, applies to sound theories. But we have already remarked in Section 1.2 that Gödel’s arguments show that we don’t need to assume soundness to prove incompleteness. In this chapter we see how to argue from *consistency* to incompleteness.

But if we are going to weaken one assumption (from soundness to mere consistency) we’ll need to strengthen another assumption: we’ll now consider theories that don’t just *express* enough but which can *capture*, i.e. *prove*, enough.

Starting in Chapter 9, we’ll begin examining various formal theories of arithmetic ‘from the bottom up’, in the sense of first setting down the axioms of the theories and then exploring what the different theories are capable of proving. For the moment, however, we are continuing to proceed the other way about. In the previous chapter, we considered theories that have sufficiently expressive languages, and so can express what we’d like any arithmetic to be able to express. Now we introduce the companion concept of a *sufficiently strong* theory, which is one that by definition can prove what we’d like any moderately competent theory of arithmetic to be able to prove about decidable properties of numbers. We then establish some easy but deep results about such theories.

### 7.1 The idea of a ‘sufficiently strong’ theory

Suppose that  $P$  is some effectively decidable property of numbers, i.e. one for which there is a mechanical procedure for deciding, given a natural number  $n$ , whether  $n$  has property  $P$  or not.

Now, when we construct a formal theory of the arithmetic of the natural numbers, we will surely want deductions inside our theory to be able to track, case by case, any mechanical calculation that we can already perform informally. We don’t want going formal to *diminish* our ability to determine whether  $n$  has this property  $P$ . As we stressed in Section 4.1, formalization aims at regimenting what we can already do: it isn’t supposed to hobble our efforts. So while we might have some passing interest in more limited theories, we will naturally aim for a formal theory  $T$  which at least (a) is able to frame some open wff  $\varphi(x)$  which expresses the decidable property  $P$ , and (b) is such that if  $n$  has property  $P$ ,  $T \vdash \varphi(\bar{n})$ , and if  $n$  does not have property  $P$ ,  $T \vdash \neg\varphi(\bar{n})$ . In short, we want  $T$  to capture  $P$  (in the sense of Section 5.5).

The suggestion therefore is that, if  $P$  is any effectively decidable property of numbers, we ideally want a competent theory of arithmetic  $T$  to be able to capture  $P$ . Which motivates the following definition:

A formal theory of arithmetic  $T$  is *sufficiently strong* iff it captures all effectively decidable numerical properties.

And it seems a reasonable and desirable condition on a formal theory of the arithmetic of the natural numbers that it be sufficiently strong.<sup>1</sup>

Much later (in Section 31.1), when we've done some more investigation into the general idea of effective decidability, we'll finally be in a position to warrant the claim that some simple, intuitively sound, and (by then) very familiar theories built in  $L_A$  do indeed meet this condition. We will thereby show that the condition of being 'sufficiently strong' is actually easily met. But we can't establish that now: this chapter just supposes that there *are* such theories and derives some consequences.

## 7.2 An undecidability theorem

A trivial way for a theory  $T$  to be sufficiently strong (i.e. to prove lots of wffs about properties of individual numbers) is by being inconsistent (i.e. by proving *every* wff about individual numbers). It goes without saying, however, that we are interested in *consistent* theories.

We also like to get *decidable* theories when we can, i.e. theories for which there is an algorithm for determining whether a given wff is a theorem (see Section 4.4). But, sadly, we have the following key result:<sup>2</sup>

**Theorem 7.1** *No consistent, sufficiently strong, axiomatized formal theory of arithmetic is decidable.*

*Proof* We suppose  $T$  is a consistent and sufficiently strong axiomatized theory yet also decidable, and derive a contradiction.

By hypothesis,  $T$ 's language can frame open wffs with 'x' free. These will be effectively enumerable:  $\varphi_0(x), \varphi_1(x), \varphi_2(x), \dots$ . For by Theorem 4.1 we know that the complete set of wffs of  $T$  can be effectively enumerated. It will then be a mechanical business to select out the ones with just 'x' free (there are standard mechanical rules for determining whether a variable is free or bound).

Now let's fix on the following definition:

$n$  has the property  $D$  if and only if  $T \vdash \neg\varphi_n(\bar{n})$ .

Note that the construction here links the subscripted index with the standard numeral to be substituted for the variable in  $\neg\varphi_n(x)$ . So this is a cousin of the

<sup>1</sup>Why is being 'sufficiently expressive' defined in terms of expressing *functions*, and being 'sufficiently strong' defined in terms of capturing *properties*? No deep reason at all. In fact, given capturing properties goes with capturing their characteristic functions, we could have defined be sufficiently strong by means of a condition on functions too.

<sup>2</sup>The undecidability of arithmetic was first shown by Church (1936b). For a neater proof, see Tarski et al. (1953, pp. 46–49). The informal proof as given here is due to Timothy Smiley, who was presenting it in Cambridge lectures in the 1960s. The first published version of it which I know is in Hunter (1971, pp. 224–225).

## 7 Sufficiently strong arithmetics

---

‘diagonal’ constructions which we encountered in proving Theorem 2.2 and again in proving Theorem 3.8.

We next show that the supposition that  $T$  is a decidable theory entails that the ‘diagonal’ property  $D$  is an effectively decidable property of numbers. For given any number  $n$ , it will be a mechanical matter to enumerate the open wffs until the  $n$ -th one,  $\varphi_n(x)$ , is produced. Then it is a mechanical matter to form the numeral  $\bar{n}$ , substitute it for the variable and prefix a negation sign. Now we just apply the supposed mechanical procedure for deciding whether a sentence is a  $T$ -theorem to test whether the wff  $\neg\varphi_n(\bar{n})$  is a theorem. So, on our current assumptions, there is an algorithm for deciding whether  $n$  has the property  $D$ .

Since, by hypothesis, the theory  $T$  is sufficiently strong, it can capture all decidable numerical properties: so it follows, in particular, that  $D$  is capturable by some open wff. This wff must of course occur somewhere in our enumeration of the  $\varphi(x)$ . Let’s suppose the  $d$ -th wff does the trick: that is to say, property  $D$  is captured by  $\varphi_d(x)$ .

It is now entirely routine to get out a contradiction. For, by definition, to say that  $\varphi_d(x)$  captures  $D$  means that for any  $n$ ,

- if  $n$  has the property  $D$ ,  $T \vdash \varphi_d(\bar{n})$ ,
- if  $n$  doesn’t have the property  $D$ ,  $T \vdash \neg\varphi_d(\bar{n})$ .

So taking in particular the case  $n = d$ , we have

- i. if  $d$  has the property  $D$ ,  $T \vdash \varphi_d(\bar{d})$ ,
- ii. if  $d$  doesn’t have the property  $D$ ,  $T \vdash \neg\varphi_d(\bar{d})$ .

But note that our initial definition of the property  $D$  implies:

- iii.  $d$  has the property  $D$  if and only if  $T \vdash \neg\varphi_d(\bar{d})$ .

From (ii) and (iii), it follows that whether  $d$  has property  $D$  or not, the wff  $\neg\varphi_d(\bar{d})$  is a theorem either way. So by (iii) again,  $d$  does have property  $D$ , hence by (i) the wff  $\varphi_d(\bar{d})$  must be a theorem too. So a wff and its negation are both theorems of  $T$ . Therefore  $T$  is inconsistent, contradicting our initial assumption that  $T$  is consistent.

In sum, the supposition that  $T$  is a consistent and sufficiently strong axiomatized formal theory of arithmetic *and* decidable leads to contradiction.  $\square$

There’s an old hope (which goes back to Leibniz) that can be put in modern terms like this: we might one day be able to mechanize mathematical reasoning to the point that a suitably primed computer could solve all mathematical problems in a domain by deciding theoremhood in an appropriate formal theory. What we’ve just shown is that this is a false hope: as soon as a theory is strong enough to capture the results of boringly mechanical reasoning about decidable properties of individual numbers, it must itself cease to be decidable.

## 7.3 Another incompleteness theorem

Now let's put together Theorem 4.2, *Any consistent, axiomatized, negation-complete formal theory is decidable*, and Theorem 7.1, *No consistent, sufficiently strong, axiomatized formal theory of arithmetic is decidable*. These, of course, immediately entail

**Theorem 7.2** *A consistent, sufficiently strong, axiomatized formal theory of arithmetic cannot be negation complete.*

That is to say, for any c.s.s.a. (consistent, sufficiently strong, axiomatized) theory of arithmetic, there will be a pair of sentences  $\varphi$  and  $\neg\varphi$  in its language, neither of which is a theorem. But one of the pair must be true on the given interpretation of  $T$ 's language. Therefore, for any c.s.s.a. theory of arithmetic  $T$ , there are true sentences of its language which  $T$  cannot decide.

And adding in new axioms won't help. To re-play the sort of argument we gave in Section 1.2, suppose  $T$  is a c.s.s.a. theory of arithmetic, and suppose  $\varphi$  is a true sentence of arithmetic that  $T$  can't prove or disprove. The theory  $T^+$  which you get by adding  $\varphi$  as a new axiom to  $T$  will, of course, now trivially prove  $\varphi$ , so we've plugged that gap. But note that  $T^+$  is consistent (for if  $T^+$ , i.e.  $T + \varphi$ , were inconsistent, then  $T \vdash \neg\varphi$  contrary to hypothesis). And  $T^+$  is sufficiently strong (since it can still prove everything  $T$  can prove). It is still decidable which wffs are axioms of  $T^+$ , so the theory still counts as a properly axiomatized formal theory. So  $T^+$  is another c.s.s.a. theory and Theorem 7.2 applies: so there is a wff  $\varphi^+$  (distinct from  $\varphi$ , of course) which is again true-on-interpretation but which  $T^+$  cannot decide (and if  $T^+$  can't prove either  $\varphi^+$  or  $\neg\varphi^+$ , then neither can the weaker  $T$ ). In sum, the c.s.s.a. theory  $T$  is therefore not only incomplete but also in a good sense incompletable.<sup>3</sup>

Which is another proof for The Book.

---

<sup>3</sup>Perhaps we should note that, while the informal incompleteness argument of Chapter 6 depended on assuming that there are general-purpose programming languages in which we can specify (the equivalent of) any numerical algorithm, the argument of this chapter doesn't require that assumption. On the other hand, our previous incompleteness result didn't make play with the idea of theories strong enough to capture all decidable numerical properties, whereas our new incompleteness result does. What we gain on the roundabouts we lose on the swings.

## 8 Interlude: Taking stock

### 8.1 Comparing incompleteness arguments

Our informal incompleteness results, Theorems 6.3 and 7.2, aren't the same as Gödel's own theorems. But they are close cousins, and they seem quite terrific results to arrive at so very quickly.

Or are they? Everything depends, for a start, on whether the ideas of a 'sufficiently expressive' arithmetic language and a 'sufficiently strong' theory of arithmetic are in good order. Now, as we've already briefly indicated in Section 3.1, there are a number of standard, well-understood, ways of formally refining the intuitive notions of effective computability and effective decidability, ways that turn out to locate the same entirely definite and well-defined class of numerical functions and properties. Hence the ideas of a 'sufficiently expressive' language (which expresses all computable one-place functions) and a 'sufficiently strong' theory (which captures all decidable properties of numbers) can also be made perfectly clear.

But by itself, that claim doesn't take us very far. For it leaves wide open the possibility that a language expressing all computable functions or a theory that captures all decidable properties has to be very rich indeed. However, we announced right back in Section 1.2 that Gödel's own arguments rule out complete theories even of the truths of basic arithmetic. Hence, if our easy Theorems are to have the full reach of Gödel's work, we'll really have to show (as we promised) that the language of basic arithmetic is sufficiently expressive, and that a theory built in that language can be sufficiently strong.

In sum, if something like our argument for Theorem 6.3 is to be used to establish a variant of one of Gödel's own results, then it needs to be augmented with (i) a general treatment of the class of computable functions, *and* (ii) a proof that (as we claimed) even  $L_A$  can express at least the one-place computable functions. And if something like our argument for Theorem 7.2 is to be used, it needs to be augmented by (iii) a proof that (as we claimed) common-or-garden theories couched in  $L_A$  can be sufficiently strong.

But even with (i), (ii) and (iii) in play, there would still remain a significant difference between our easy theorems and Gödel's arguments. For our lines of argument don't yet give us any specific examples of unprovable truths. By contrast, Gödel's proof tells us how to take a consistent theory  $T$  and actually construct a true but unprovable-in- $T$  sentence (the one that encodes 'I am unprovable in  $T$ '). *Moreover, Gödel does this without needing the general treatment in (i) and without needing all of (ii)/(iii) either.*

There is a significant gap, then, between our two intriguing, quickly-derived, but informal theorems and the industrial-strength results that Gödel proves. So, while what we have shown so far is highly suggestive, it is time to start turning to Gödel's own arguments. But before we press on, let's highlight two very important general lessons we can already learn from our informal theorems:

- A. Arguments for incompleteness come in (at least) two flavours. First, we can combine the premiss (a) that we are dealing with a *sound* theory with the premiss (b) that our theory's language is *expressively* rich enough. Or second, we can weaken one assumption and beef up the other: in other words, we can use the weaker premiss (a') that we are dealing with a *consistent* theory but add the stronger premiss (b') that our theory can *prove* enough facts. We'll see that Gödel's arguments too come in these two flavours.
- B. Arguments for incompleteness don't have to depend on the construction of Gödel sentences that somehow say of themselves that they are unprovable. Neither of our informal proofs do.

## 8.2 A road-map

So now we turn to Gödel's proofs. And to avoid getting lost in what follows, it will help to have in mind an overall road-map of the route we are taking:

1. We begin by describing some standard formal systems of arithmetic, in particular the benchmark PA, so-called 'First-order Peano Arithmetic', and an important subsystem Q, 'Robinson Arithmetic'. (Chapters 9–11)
2. These systems are framed in  $L_A$ , the language of basic arithmetic. So they only have successor, addition and multiplication as 'built-in' functions. But we go on to describe the large family of 'primitive recursive' functions, properties and relations (which includes all familiar arithmetical functions like the factorial and exponential, and familiar arithmetic properties like being prime, and relations like one number being the square of another). And we then show that Q and PA can not only express but capture all the primitive recursive functions, properties and relations – a major theorem that was, in essence, first proved by Gödel. (Chapters 12–14)
3. We next turn to Gödel's simple but crucial innovation – the idea of systematically associating expressions of a formal arithmetic with numerical codes. Any sensibly systematic scheme of 'Gödel numbering' will do: but Gödel's original style of numbering has a certain naturalness, and makes it tolerably straightforward to prove arithmetical results about the codings. With a coding scheme in place, we can reflect properties and relations of strings of symbols of PA (to concentrate on that theory) by properties

and relations of their Gödel numbers. For example, we can define the numerical properties *Term* and *Wff* which hold of a number when it is the code number for a symbol sequence which is, respectively, a term or a wff of PA. And we can, crucially, define the numerical relation  $Prf(m, n)$  which holds when  $m$  codes for an array of wffs that is a PA proof, and  $n$  codes the closed wff that is thereby proved. This project of coding up various syntactic relationships is often referred to as *the arithmetization of syntax*. And what Gödel showed is that – given a sane system of Gödel numbering – these and a large family of related arithmetical properties and relations are primitive recursive.

4. Next – the really exciting bit! – we use the fact that relations like  $Prf$  are expressible in PA to construct a ‘Gödel sentence’.  $G$  will be true when there is no number that is the Gödel number of a PA proof of the wff that results from a certain construction – where the wff that results is none other than  $G$  itself. So  $G$  is true just if it is unprovable in PA. Given PA is sound and only proves truths,  $G$  can’t be provable; hence  $G$  is true; hence  $\neg G$  is false, and so is also unprovable in PA. In sum, given PA is sound, it cannot decide  $G$ . Further, it turns out that we can drop the *semantic* assumption that PA is sound. Using the fact that PA can capture relations like  $Prf$  (as well as merely express them), we can still show that  $G$  is undecidable while just making a *syntactic* assumption. (Chapter 17)
5. Finally, we note that the true-but-unprovable sentence  $G$  for PA is generated by a method that can be applied to any other arithmetic that satisfies some modest conditions. In particular, adding  $G$  as a new axiom to PA just gives us a revised theory for which we can generate a new true-but-unprovable wff  $G'$ . Throwing in  $G'$  as a further axiom then gives us another theory for which we can generate yet another true-but-unprovable wff. And so it goes. PA is therefore not only incomplete but incompletable. In fact, *any* properly axiomatized consistent theory that contains the weak theory  $Q$  is incompletable. (Chapter 18)

Just one comment. This summary makes Gödel’s formal proofs in terms of *primitive recursive* properties and relations sound rather different from our informal proofs using the idea of theories which express/capture *decidable* properties or relations. But a link can be made when we note that the primitive recursive properties and relations are in fact a large subclass of the intuitively decidable properties and relations. Moreover, showing that  $Q$  and PA can express/capture all primitive recursive properties and relations takes us most of the way to showing that those theories are sufficiently expressive/sufficiently strong.

However, we’ll leave exploring this link until much later. Only after we have travelled the original Gödelian route (which doesn’t presuppose a *general* account of computability) will we return to consider how to formalize the arguments of Chapters 6 and 7 (a task which *does* presuppose such a general account).