9 Expressing and capturing the primitive recursive functions

Addition can be defined in terms of repeated applications of the successor function. Multiplication can be defined in terms of repeated applications of addition. The exponential and factorial functions can be defined, in different ways, in terms of repeated applications of multiplication. There's already a pattern emerging here! And the main task of the last chapter was to get clear about this pattern.

So first we said more about the idea of defining one function in terms of repeated applications of another function. Tidied up, that becomes the idea of *defining a function by primitive recursion* (Defn. 29). Then we explained the idea of giving a definitional chain which defines a function by primitive recursion and/or composition from other functions which we define by primitive recursion and/or composition from other functions, and so on down, until we bottom out with the successor function and other trivia. Tidied up, this gives us the idea of a *primitive recursive function*, i.e. one that can be characterized by such a chain of definitions (Defn. 32).

We noted three key facts:

1. Every p.r. function is effectively computable – moreover it is computable using only 'for' loops, without open-ended searches using 'do until' loops. That's Theorem 22.

2. Conversely, if a numerical function can be computed from the starter pack of simple p.r. functions using only 'for' loops, then it too is primitive recursive. That's Theorem 23.

3. But not every intuitively computable numerical function is primitive recursive. That's Theorem 24.

So the situation is now this. In Chapters 5 and 6, we introduced some *formal* arithmetics with just three functions – successor, addition, multiplication – built in. But we have now reminded ourselves that ordinary *informal* arithmetic talks about many more elementary computable functions like the exponential, the factorial, and so on: and we generalized the sort of way these functions can be defined to specify the whole class of primitive recursive functions. A gulf seems to have opened up, then, between the modesty of the resources of our formal

theories (including the strongest so far, PA) and the richness of the world of p.r. functions (and we know that those aren't even all the computable arithmetical functions).

9.1 Bridging the divide

The aim of this chapter is to show that the theories Q and PA, despite the modesty of their built-in resources, reach a lot further than you might expect. In particular, we arrive at two key results, Theorems 27 and 29. The first will tell us that

> *The language L_A can express all primitive recursive functions.*

And there's more: looking at the proof of that result we find that in fact L_A can express any p.r. function using a Σ_1 wff – i.e. by using a wff of low quantifier complexity. The second of our theorems then tells us that

> *The theory* Q *– and hence any stronger theory like* PA *– can capture any p.r. function.*

Again, the capturing can be done by using Σ_1 wffs.

Now, the ideas involved in proving these two theorems are not particularly difficult. But working through the proofs takes a bit of care and patience. We'll give most of a proof of Theorem 27, but only outline what it takes to prove Theorem 29. And if you are prepared to take these theorems more or less on trust, then it will do little harm to skim and skip.

9.2 L_A can express the factorial function

In this section, as a warm-up exercise, we are going to show that L_A – despite having only successor, addition and multiplication built in – can in fact *express* the factorial function. That is to say, we can construct an L_A wff $\mathsf{F}(\mathsf{x},\mathsf{y})$ such that, for any particular m and n, $\mathsf{F}(\overline{\mathsf{m}},\overline{\mathsf{n}})$ if and only if $n = m!$.

Then in the next section we'll use the same key trick in showing that L_A can express *any* p.r. function at all. But it will be much easier to follow the general argument if you first meet the 'β-function' trick deployed in a simple particular case.

Consider, then, the primitive recursive definition of the factorial function again:

$$0! = 1$$
$$(Sz)! = z! \times Sz$$

Now, think of this definition in the following way: for any x, it tells us how to construct a sequence of numbers $0!, 1!, 2!, \ldots, x!$, where we move from the i-th member of the sequence (counting from zero) to the next by multiplying by Si. Or putting it a bit more abstractly, suppose that for numbers x and y,

1. There is a sequence of numbers $k_0, k_1, \ldots, k_x$ such that: $k_0 = 1$, and if $i < x$ then $k_{Si} = k_i \times Si$, and $k_x = y$.

Then this is equivalent to saying that $y = x!$.

So the question of how to reflect the p.r. definition of the factorial inside L_A can be parlayed into the following question: how can we express facts about *finite sequences of numbers* using the limited resources of L_A?

Use numerical codes! Suppose we can wrap up a finite sequence into a single *code number* c, and then have a two-place *decoding function*, call it simply *decode*, such that if you give *decode* the code c and the index i, the function spits out the i-th member of the sequence which c codes. In other words, suppose that, when c is the code number for the sequence $k_0, k_1, \ldots, k_x$, then $decode(c, i) = k_i$.

If we can find such a coding scheme, then we can rewrite (1) as follows, talking about a code number c instead of the sequence $k_0, k_1, \ldots, k_x$, and writing $decode(c, i)$ instead of k_i:

2. There is a code number c such that: $decode(c, 0) = 1$, and if $i < x$ then $decode(c, Si) = decode(c, i) \times Si$, and $decode(c, x) = y$.

This way, if a suitable *decode* function can indeed be expressed in L_A, then we can define the factorial in L_A. Great! So can we do this coding trick?

To link up with Gödel's own version of the trick, let's liberalize our notion of coding/decoding just a little to allow decoding functions which take *two* code numbers c and d, and an index number i, as follows:

> A three-place decoding function is a function $decode(c, d, i)$ such that, for *any* finite sequence of natural numbers $k_0, k_1, k_2, \ldots, k_n$ there is a *pair* of code numbers c, d such that, for every $i \leq n$, $decode(c, d, i) = k_i$.

A three-place decoding-function will obviously do just as well as a two-place function to help us express facts about finite sequences.

Even with this liberalization, though, it still isn't at all obvious how to define a decoding function in terms of the functions built into basic arithmetic. But Gödel neatly solved the problem with his β-function. Put

$$\beta(c, d, i) =_{\text{def}} \text{the remainder left when } c \text{ is divided by } d(i + 1) + 1.$$

Then we have

Theorem 26. *For any finite sequence of numbers $k_0, k_1, \ldots, k_n$, we can find a suitable pair of numbers c, d such that for $i \leq n$, $\beta(c, d, i) = k_i$.*

This claim should look intrinsically plausible. As we divide c by $d(i + 1) + 1$, then for different values of i ($0 \leq i \leq n$) we'll get a sequence of $n+1$ remainders. Vary c and d, and the sequence of remainders will vary. The permutations as we vary c and d without limit *appear* to be simply endless. We just need to check,

then, that appearances don't deceive, and we *can* always find a (big enough) c and a (smaller) d which makes the sequence of remainders match a given $n+1$-term sequence of numbers (mathematical completists: see *IGT2*, §15.2, fn. 4 for a proof that this works!)

But now reflect that the concept of a remainder on division can be elementarily defined in terms of multiplication and addition: the remainder when a is divided by b (with $b < a$) is y, when there is some number u (no greater than a) such that $a = b \times u + y$, where $y < b$.

So consider the following open wff:

$$\mathsf{B(c,d,i,y)} =_{\text{def}} (\exists \mathsf{u} \leq \mathsf{c})[\mathsf{c} = \{\mathsf{S(d \times Si) \times u}\} + \mathsf{y} \ \wedge \ \mathsf{y} \leq \mathsf{(d \times Si)}].$$

If you think about it, this expresses our three-place Gödelian β-function in L_A (for remember, we can define '$\leq$' in L_A).

OK: we said that $y = x!$ just in case

1. There is a sequence of numbers $k_0, k_1, \ldots, k_x$ such that: $k_0 = 1$, and if $i < x$ then $k_{Si} = k_i \times Si$, and $k_x = y$.

And we now know we can reformulate this as follows:

2′. There is some pair of code numbers c, d such that: $\beta(c, d, 0) = 1$, and if $i < x$ then $\beta(c, d, Si) = \beta(c, d, i) \times Si$, and $\beta(c, d, x) = y$.

But we've seen that the β-function can be expressed in L_A by the open wff we abbreviated B. So we can render (2′) into L_A as follows:

3. $\exists \mathsf{c} \exists \mathsf{d}\{\mathsf{B(c,d,0,S0)} \wedge$
   $(\forall \mathsf{i} \leq \mathsf{x})[\mathsf{i} \neq \mathsf{x} \rightarrow \exists \mathsf{v} \exists \mathsf{w}\{(\mathsf{B(c,d,i,v)} \wedge \mathsf{B(c,d,Si,w)}) \wedge \mathsf{w = v \times Si}\} \wedge$
   $\mathsf{B(c,d,x,y)}\}$.

Abbreviate all that by '$\mathsf{F(x,y)}$', and we've arrived. For this evidently expresses (2′) which is equivalent to (1) and so expresses the factorial function. Neat!

9.3 L_A can express all p.r. functions

Using the β-function trick again, we can now generalize to show that L_A can express any p.r. function.

We already know from §8.3 the standard strategy for showing that something is true of all p.r. functions. So suppose that the following three propositions are all true:

E1. L_A can express the initial functions. (See Defn. 31.)

E2. If L_A can express the functions g and h, then it can also express a function f defined by composition from g and h. (See Defn. 30.)

E3. If L_A can express the functions g and h, then it can also express a function f defined by primitive recursion from g and h. (See Defn. 29.)

Then by the argument of §8.3, those assumptions will be enough to establish our desired general result. So how can we prove (E1) to (E3)?

Proof of E1. Just look at cases. The successor function $Sx = y$ is of course expressed by the open wff $\mathsf{Sx} = \mathsf{y}$.

The zero function, $Z(x) = 0$ is expressed by the wff $\mathsf{Z}(\mathsf{x}, \mathsf{y}) =_{\text{def}} (\mathsf{x} = \mathsf{x} \wedge \mathsf{y} = \mathsf{0})$.

Finally, the three-place function $I^3_2(x, y, z) = y$, to take just one example of an identity function, is expressed by the wff $\mathsf{I}^3_2(\mathsf{x}, \mathsf{y}, \mathsf{z}, \mathsf{u}) =_{\text{def}} \mathsf{y} = \mathsf{u}$ (or we could use $(\mathsf{x} = \mathsf{x} \wedge \mathsf{y} = \mathsf{u} \wedge \mathsf{z} = \mathsf{z})$ if we'd like x and z actually to appear in the wff). Likewise for all the other identity functions. □

Proof of E2. Suppose, to take a simple example, that g and h are one-place functions, expressed by the wffs $\mathsf{G}(\mathsf{x}, \mathsf{y})$ and $\mathsf{H}(\mathsf{x}, \mathsf{y})$ respectively. Then, the function $f(x) = h(g(x))$ is evidently expressed by the wff $\exists \mathsf{z}(\mathsf{G}(\mathsf{x}, \mathsf{z}) \wedge \mathsf{H}(\mathsf{z}, \mathsf{y}))$.

For suppose $g(m) = k$ and $h(k) = n$, so $f(m) = n$. Then by hypothesis $\mathsf{G}(\overline{\mathsf{m}}, \overline{\mathsf{k}})$ and $\mathsf{H}(\overline{\mathsf{k}}, \overline{\mathsf{n}})$ will be true, and hence $\exists \mathsf{z}(\mathsf{G}(\overline{\mathsf{m}}, \mathsf{z}) \wedge \mathsf{H}(\mathsf{z}, \overline{\mathsf{n}}))$ is true, as required. Conversely, suppose $\exists \mathsf{z}(\mathsf{G}(\overline{\mathsf{m}}, \mathsf{z}) \wedge \mathsf{H}(\mathsf{z}, \overline{\mathsf{n}}))$ is true. Then since the quantifiers run over numbers, $(\mathsf{G}(\overline{\mathsf{m}}, \overline{\mathsf{k}}) \wedge \mathsf{H}(\overline{\mathsf{k}}, \overline{\mathsf{n}}))$ must be true for some k. So we'll have $g(m) = k$ and $h(k) = n$, and hence $f(m) = h(g(m)) = n$ as required.

Other cases where g and/or h are multi-place functions can be handled similarly. □

Proof of E3. The tricky case! We need to show that we can use the same β-function trick and prove more generally that, if the function f is defined by recursion from functions g and h which are already expressible in L_A, then f is also expressible in L_A.

We are assuming that

$$f(\vec{x}, 0) = g(\vec{x})$$
$$f(\vec{x}, Sy) = h(\vec{x}, y, f(\vec{x}, y)).$$

(Remember, $\vec{x}$ indicates some variables being 'carried along for the ride', that don't change in the course of the recursion that defines $f(\vec{x}, Sy)$ in terms of $f(\vec{x}, y)$.)

This definition amounts to fixing the value of $f(\vec{x}, y) = z$ thus:

1. There is a sequence of numbers $k_0, k_1, \ldots, k_y$ such that: $k_0 = g(\vec{x})$, and if $i < y$ then $k_{Is} = h(\vec{x}, u, k_i)$, and $k_y = z$.

So using a three-place β-function again, that comes to

2. There is some c, d, such that: $\beta(c, d, 0) = g(\vec{x})$, and if $i < y$ then $\beta(c, d, Si) = h(\vec{x}, i, \beta(c, d, i))$, and $\beta(c, d, y) = z$.

Suppose we can already express the n-place function g by a $(n + 1)$-variable expression G, and the $(n+2)$-variable function h by the $(n+3)$-variable expression H. Then – using '$\vec{\mathsf{x}}$' to indicate a suitable sequence of n variables – (2) can be rendered into L_A by

3. $\exists c \exists d\{\exists k[B(c,d,0,k) \wedge G(\vec{x},k)] \wedge$
$(\forall i \leq y)[i \neq y \rightarrow \exists v \exists w\{(B(c,d,i,v) \wedge B(c,d,Si,w)) \wedge H(\vec{x},i,v,w)\}] \wedge$
$B(c,d,y,z)\}$.

Abbreviate this defined wff as $\varphi(\vec{x},y,z)$; it is then evident that φ will serve to express the p.r. defined function f. Which gives us the desired result E3. □

So, we've shown how to establish each of the claims E1, E2 and E3. We therefore get our desired

Theorem 27. *The language L_A can express all primitive recursive functions.*

Proof. For any p.r. function f, there is a sequence of functions $f_0, f_1, f_2, \ldots, f_k$ where each f_j is either an initial function or is constructed out of previous functions by composition or recursion, and $f_k = f$. Corresponding to that sequence of functions we can write down a sequence of L_A wffs which express those functions. We write down the E1 expression corresponding to an initial function. If f_j comes from two previous functions by composition, we use an existential quantifier construction as in E2 to write down a wff built out of the wffs expressing the two previous functions. And if f_j comes from two previous functions by recursion, we use the β-function trick and write down a (3)-style expression built out of the wffs expressing the two previous functions. □

9.4 Canonical wffs for expressing p.r. functions are Σ_1

Let's say that

Defn. 36. *An L_A wff* canonically expresses *the p.r. function f if it recapitulates a definitional chain for f by being constructed in the manner described in the proof of Theorem 27.*

We can express a given p.r. function f by other wffs too (if only by adding redundant clauses): but it is the canonical ones from which we can read off a full definition for f which will interest us the most.

Now, a canonical wff which reflects a full definition of f is built up starting from wffs expressing initial wffs. Those starter wffs are Δ_0 wffs – see the Proof for E1 – and hence Σ_1.

Suppose g and h are one-place functions, expressed by the Σ_1 wffs $G(x,y)$ and $H(x,y)$ respectively. The function $f(x) = h(g(x))$ is expressed by the wff $\exists z(G(x,z) \wedge H(z,y))$ – as in the Proof for E2 – which is Σ_1 too. For that is equivalent to a wff with the existential quantifiers pulled from the front of the Σ_1 wffs G and H out to the very front of the new wff. Similarly for other cases of composition.

Finally, look at the case where f is defined from g and h by primitive recursion, where g and h are p.r. functions which can be expressed by Σ_1 wffs. Then f can be expressed by a wff of the kind (3) – as in our proof of E3. And this too is Σ_1. For B is Δ_0: and (3) is equivalent to what we get when we drag all the existential

quantifiers buried at the front of each of B, G and H to the very front of the wff. (Yes, dragging existentials past a universal is usually wicked! – but the only universal here is a bounded universal, which is 'really' just a tame conjunction, and simple tricks allow us to get the existentials all at the front. For details, see *IGT*2.)

So in fact our recipe for building a canonical wff stage by stage in fact takes us from Σ_1 wffs to Σ_1 wffs. Which yields the stronger

Theorem 28. *L_A can express any p.r. function f by a Σ_1 wff which recapitulates a full definitional chain for f.*

9.5 Q can capture all p.r. functions

We have shown that the language of the theory Q can *express* all p.r. functions using Σ_1 wffs. We now want to show that this theory (and hence any stronger one) can also *capture* all those functions using Σ_1 wffs.

But hold on! We haven't yet said what it is for a theory to capture a function. So we first need to explain that.

Recall our earlier account of what it is to capture a property or relation. In particular, recall

Defn 18. *The theory T captures the two-place numerical relation R by the open wff $\varphi(\mathsf{x},\mathsf{y})$ iff, for any m, n,*

i. if m has the relation R to n, then $T \vdash \varphi(\overline{\mathsf{m}},\overline{\mathsf{n}})$,

ii. if m does not have the relation R to n, then $T \vdash \neg\varphi((\overline{\mathsf{m}},\overline{\mathsf{n}})$.

Thinking of $f(m) = n$ as stating a two-place relation between m and n we might expect the definition for capturing a function to have the same shape:

Defn. 37. *The theory T captures the one-place function f by the open wff $\psi(\mathsf{x},\mathsf{y})$ iff, for any m, n,*

i. if $f(m) = n$, then $T \vdash \psi(\overline{\mathsf{m}},\overline{\mathsf{n}})$,

ii. if $f(m) \neq n$, then $T \vdash \neg\psi(\overline{\mathsf{m}},\overline{\mathsf{n}})$.

But (for technical reasons we are not going to fuss about here) it turns out to be useful to add a further requirement

iii. $T \vdash \exists!\mathsf{y}\psi(\overline{\mathsf{m}},\mathsf{y})$.

In other words, T also 'knows' that ψ is functional (associates a number to just one value).

Our target, then, is the following theorem:

Theorem 29. *The theory* Q *can capture any p.r. function by a Σ_1 wff.*

Proof outline. There's more than one route to this theorem. But the conceptually simplest is to use again the same overall strategy we used in proving that Q's language can express every p.r. function. Suppose then that we can prove

C1. Q can capture the initial functions.

C2. If Q can capture the functions g and h, then it can also capture a function f defined by composition from g and h.

C3. If Q can capture the functions g and h, then it can also capture a function f defined by primitive recursion from g and h.

Then it follows that Q can capture any p.r. function.

So how do we prove C1? We just check that the formulas which we said in §9.3 *express* the initial functions in fact serve to *capture* the initial functions in Q.

How do we prove C2? Suppose g and h are one-place functions, captured by the wffs $\mathsf{G}(\mathsf{x},\mathsf{y})$ and $\mathsf{H}(\mathsf{x},\mathsf{y})$ respectively. Then we prove that the function $f(x) = h(g(x))$ is captured by the wff $\exists\mathsf{z}(\mathsf{G}(\mathsf{x},\mathsf{z}) \wedge \mathsf{H}(\mathsf{z},\mathsf{y}))$. We can generalize the result.

And how do we prove C3? This is the tedious case that takes hard work, done in gory detail *IGT2*! We need to show that (a tweaked version of) formula B not only expresses but captures Gödel's β-function. And then we use that fact to prove that if the n-place function g is captured by a $(n+1)$-variable expression G, and the $(n+2)$-variable function h by the $(n+3)$-variable expression H, then a wff built to the pattern of (3) in §9.3 captures the function f defined by primitive recursion from g and h. Not surprisingly, details get messy (not difficult, just messy).

So take a definitional chain for defining a p.r. function. Follow the step-by-step instructions implicit in §9.3 about how to build up a canonical wff which in effect recapitulates that recipe, tweaking the treatment of the β-function. You'll get a wff captures the function in Q (and in any stronger theory which contains the language of basic arithmetic). Moreover the wff in question will again be Σ_1 by the same argument as before. □

9.6 Expressing/capturing properties and relations

Just a brief coda, linking what we've done in this chapter with the last section of the previous chapter.

We said in Defn. 34 that the characteristic function c_P of a monadic numerical property P is defined by setting $c_P(m) = 0$ if m is P and $c_P(m) = 1$ otherwise. And a property P is said to be p.r. decidable if its characteristic function is p.r.

Now, suppose that P is p.r.; then c_P is a p.r. function. So L_A can express c_P by a two-place Σ_1 wff $\mathsf{c}_P(\mathsf{x},\mathsf{y})$. So if m is P, i.e. $c_P(m) = 0$, then $\mathsf{c}_P(\overline{\mathsf{m}},\mathsf{0})$ is true. And if m is not P, i.e. $c_P(m) \neq 0$, then $\mathsf{c}_P(\overline{\mathsf{m}},\mathsf{0})$ is not true. Hence, by the definition of expressing-a-property, the wff $\mathsf{c}_P(\mathsf{x},\mathsf{0})$ serves to express the p.r. property P. The point generalizes from monadic properties to many-place relations. So as an easy corollary of Theorem 28 we get:

Theorem 30. *L_A can express all p.r. decidable properties and relations, again using Σ_1 wffs.*

Similarly, suppose again that the monadic property P is p.r. so c_P is a p.r. function. So Q can capture c_P by a two-place Σ_1 wff $\mathsf{c}_P(\mathsf{x}, \mathsf{y})$. So if m is P, i.e. $c_P(m) = 0$, then $\mathsf{Q} \vdash \mathsf{c}_P(\overline{\mathsf{m}}, \mathsf{0})$. And if m is not P, i.e. $c_P(m) \neq 0$, then $\mathsf{Q} \vdash \neg\mathsf{c}_P(\overline{\mathsf{m}}, \mathsf{0})$. Hence, by the definition of capturing-a-property, the wff $\mathsf{c}_P(\mathsf{x}, \mathsf{0})$ serves to capture the p.r. property P in Q. The point trivially generalizes from monadic properties to many-place relations. So as an easy corollary of Theorem 29 we get:

Theorem 31. Q *can capture all p.r. decidable properties and relations, again using* Σ_1 *wffs.*