Chapter 5

The Partial Recursive Functions

We have seen in great detail that more and more nested recursion yields more and more functions, *ad infinitum*. And lest one think that there is a concept of "mega-recursion" that captures effectiveness altogether, we have seen how such functions could be effectively indexed (add a new clause for mega-recursion) and then effectively diagonalized to yield an intuitively computable function that is not mega-recursive. The only solution, we saw, is to not leave "targets" everywhere in the Cantorian table of function values for the impending diagonalization to strike. (Think of the children's game Battleship$^{\text{TM}}$. It's easy to win if every ship of your opponent has to sit on the diagonal of the grid!)

The moral is that capturing the total computable functions requires that we think about partial functions that are undefined on some arguments. In computational terms, these undefined values correspond to the program or description of the function going into an "infinite loop" or running off the end of an unbounded search. Infinite loops and partial functions are the price we pay for a computer language capable of computing all intuitively computable total functions. To catch all the flounder, we have to put up with catching some old tires. The "net" of an effective definitional language is too coarse to do the sorting for us.

5.1 Partial Functions

First some notation on partial functions. I know you have probably seen this before, but notation is not standard so choices must be clarified at the outset.

1. A relation R on $A \times B$ is just a subset of $A \times B$.
2. We write $R(a, b)$ to mean $(a, b) \in R$.
3. $dom(R) = \{a \in A | \exists_{b \in B}((a, b) \in R)\}$.

4. $range(R) = \{b \in B | \exists_{a \in A}((a, b) \in R)\}$.
5. A function $f : A \rightarrow B$ is a single-valued relation in $A \times B$.
6. If $x \notin dom(f)$ then we say that $f(x)$ is undefined.
7. A function f is total with respect to some set A (usually understood from context) just in case $dom(f) = A$.
8. A standard convention is to denote total functions with lower-case Latin letters like f, g, h and partial functions with lower-case Greek letters like ϕ, ψ.

The following points are *very important*.

1. We write $\phi(x) \simeq y$ when we mean that $(x, y) \in f$. Thus, *one is not entitled in the theory of partial functions to assume that closed terms like* $f(6)$ *denote*. Since the proof rules governing function symbols and identity presuppose that primitive functions in the logical language are total, this means that the notation $\phi(x) \simeq y$ must be eliminated as shown *before* such rules are applied in a proof.
2. The composition operator is extended to partial functions in an obvious way: if any function in a composition fails to return a value, the whole composition is undefined as well. More precisely, the whole composition
$$C(\psi, \phi_1, \ldots, \phi_n)(\vec{x})$$
is undefined just in case any one of the component applications $\phi_1(x)$ is undefined or they are all defined but
$$\psi(\phi_1(\vec{x}), \ldots, \phi_n(\vec{x}))$$
is undefined.
3. Similarly, the primitive recursion $R(g, f)(n, \vec{x})$ is undefined if any intermediate value involved in the computation of $R(g, f)(n, \vec{x})$ is undefined.

5.2 Minimalization

There are lots of ways to introduce partial yet calculable functions. One such way is to close $Prim$ under the minimalization operator M, where $M(\phi)$ denotes the unique function:
$$(\mu x)\phi(x, \vec{y}) \simeq 0,$$
which returns the least x such that $\phi(x, \vec{y}) \simeq 0$ and $\phi(z, \vec{y})$ is *defined* and nonzero for all $z < x$. Note that whenever an "infinite loop" is hit among the successive computations $\phi(0, \vec{y}), \phi(1, \vec{y}), \phi(2, \vec{y}), \ldots$ the whole minimalization crashes. Thus, we think of minimalization as a stupid, "serial" search that goes off the deep end if any successive computation doesn't halt.
As before, let
$$(\mu x)R(x, \vec{y}) \simeq (\mu x)[\overline{sg}(\chi_R(x, \vec{y}))].$$

5.3 The Zen of Dovetailing

Suppose we want to find a, b such that $R(a, b, \vec{z})$ is true. A dumb way to proceed is to set

$$a(\vec{z}) \simeq (\mu x)[(\mu y)R(x, y, \vec{z})];$$

$$b(\vec{z}) \simeq (\mu y)[(\mu x)R(x, y, \vec{z})].$$

This is dumb, for suppose that $R(1, 1, \vec{z})$ holds but $R(x, 0, \vec{z})$ fails for all x. Then $b(\vec{z})$ is undefined, for the inner search is handed $y = 0$ by the outer search and goes off the deep end waiting to find a matching x.

It is a curious historical fact that while medieval Western philosophy was employed by the Church to come up with static demonstrations of timeless Church dogmas, Zen masters in medieval Japan made their way in the world by giving fencing advice to samurai warriors. Since samurai wore thin armor and carried lethal blades (capable of slicing a human body from shoulder to hip in one stroke) it was a matter of the utmost importance to strike all of one's enemies first on the battlefield. Only fractions of seconds separated life from death, so in a field of expert enemies, it would be a (truly) fatal error to focus on one enemy to the exclusion of the others. This fatal focus of attention on an isolated feature of battle was called a "suki". Buddhists recognized "suki" as a special symptom of the general human weakness of trying to conceptualize and partition reality. There is a Zen koan about the "thousand armed" bodhisattva (bodhisattva = "all but dissertation buddha") Kannon. If Kannon uses only one of her arms to perform each task to completion, to the exclusion of all others (serial computation), then she/he is no more powerful than a human; but if she avoids "suki" and works on all problems with all arms at once (parallel computation) she can achieve marvels.

We will use our n-ary encodings to achieve the same result. Instead of searching for the first component and then for the second, we search for the code number of a pair whose components satisfy the relation. Then we return the desired component of the number we find.

$$a(\vec{z}) \simeq ((\mu w)R((w)_0, (w)_1, \vec{z}))_0;$$

$$b(\vec{z}) \simeq ((\mu w)R((w)_0, (w)_1, \vec{z}))_1.$$

Searching in parallel by minimalization over a coded tuple is called *dovetailing* because infinite searches are interleaved together to avoid "suki". This is the basic programming technique in recursive function theory and is involved in almost every interesting construction. We will employ it shortly.

5.4 The Partial Recursive Functions

Let $Part$ denote the least set X such that

1. the basic functions o, s, p_k^i are all in X;

2. X is closed under the extended notions of composition, primitive recursion, and minimalization.

Although composition and primitive recursion produce total functions from total functions, once minimalization is applied we end up with partial functions in the mix, so we have to use our extended notions of composition and primitive recursive to deal with them.

Sadly, the standard way of talking is screwed up, because the partial recursive functions include total functions, which are then the total partial recursive functions. This silly decision is softened by calling the total partial recursive functions simply the total recursive functions or simply the recursive functions. Let Tot denote the total recursive functions. It would have been better to call the partial recursive functions the recursive functions and the total ones the total recursive functions, but the damage of putting the qualifier on the broader class is already done. Where is Aristotle when you need him?

The following is immediate (why?) and allows us to retain all our earlier work!

Proposition 5.1

1. $Prim \subseteq Part$.

2. If $Prim$ is closed under an operator, then so are $Part$ and Tot.

5.5 Indexing the Partial Recursive Functions

One might suppose it would make more sense to index Tot instead of $Part$, since the intuitively effective, *total* functions were the ones we initially wanted to capture. But this proposal raises a dilemma. If the indexing were intuitively effective, then we could diagonalize (because the functions are total) and produce a total effective function that is not total recursive, undermining our hope to have captured all the intuitively effective total functions. But if the indexing is not effective, we can't use it to construct intuitively effective functions. We escape the dilemma by indexing the partial recursive functions. Then if we have indeed caught all the intuitively effective functions, it follows that we can't effectively sort the total indices from the partial ones. That's how we will escape from the diagonal argument.

To index $Part$, we simply add a clause for minimalization to our earlier indexing of $Prim$. The arity of the function minimalized should be one plus the arity k of the function we want to obtain. To reflect the fact that the indexed functions may end up being partial, we write ϕ_x^k instead of f_x^k.

$$
\begin{aligned}
&k = 0 \Rightarrow \phi_x^k = x \\
&k > 0 \wedge \\
&\quad lh(x) = 0 \Rightarrow \phi_x^k = C(o, p_k^1); \\
&\quad lh(x) = 1 \Rightarrow \phi_x^k = C(s, p_k^1); \\
&\quad lh(x) = 2 \Rightarrow \phi_x^k = p_k^{\min((x)_0, k)}; \\
&\quad lh(x) = 3 \Rightarrow \phi_x^k = C(\phi_{(x)_0}^{lh((x)_1)}, \phi_{((x)_1)_0}^k, \ldots, \phi_{((x)_1)_{lh((x)_1) \dot{-} 1}}^k); \\
&\quad lh(x) = 4 \Rightarrow \phi_x^k = R(\phi_{(x)_0}^{k \dot{-} 1}, \phi_{(x)_1}^{k+1}); \\
&\quad lh(x) = 5 \Rightarrow \phi_x^k = M(\phi_{(x)_0}^{k+1}); \\
&\quad lh(x) > 5 \Rightarrow \phi_x^k = C(o, p_k^1).
\end{aligned}
$$

5.6 The "Universal Machine" Theorem

We now have an indexing ϕ_n^k of $Part$. But n and k are parameters to ϕ_n^k, not arguments. In the case of $Prim$, we provided an effective indexing f_n^k of $Prim$ such that

$$g(n, \vec{x}) = f_n^k(\vec{x})$$

is not primitive recursive. So it would be of no small interest if in this case there were to exist a *universal function* ψ in $Part$ such that

$$\psi(n, \langle \vec{x} \rangle) = \phi_n^k(\vec{x}) \text{ if } \phi_n^k(\vec{x}) \text{ is defined;}$$

$$\psi(n, \langle \vec{x} \rangle) \text{ is undefined if } \phi_n^k(\vec{x}) \text{ is undefined.}$$

Notice that the k parameter is dropped because the code number of the argument list effectively "tells" the program how many inputs to expect. It will turn out that the existence of such a function is of broad significance for the theory of computability. It is much more useful for later work to first define a primitive recursive relation

$$U(n, t, y, i)$$

such that for each n, t, y, and k-ary $\vec{x}$,

$$\exists t \; U(n, t, y, \langle \vec{x} \rangle) \iff \phi_n^k(\vec{x}) \simeq y.$$

Think of t as a "resource bound" on computation so that, intuitively,

$$U(n, t, y, \langle \vec{x} \rangle) \iff \begin{array}{l} n \text{ halts with output } y \text{ on the } k \text{ inputs } \vec{x} \text{ after using} \\ \text{no more than quantity } t \text{ of computational resources.} \end{array}$$

The universal relation is often called the *Kleene predicate*. The existential quantifier is now intuitive. A halting computation halts under some finite bound on resources. Computations that never halt use more and more resources. (This intuitive gloss works better for more "computery" formalisms like Turing and Urm machines than it does for partial recursion).

Then we can recover the desired universal function y by dovetailing the search for the output y with the search for a suitable runtime bound t and then returning the component that represents the output.

$$\psi(n, \langle\vec{x}\rangle) \simeq ((\mu z)U(n, (z)_0, (z)_1, \langle\vec{x}\rangle))_1.$$

Since a runtime bound and output exist only if the computation halts with that output, and since the predicate will be shown to be primitive recursive, the minimization is guaranteed to find the pair and return the correct output.

It remains only to exhibit a primitive recursive decision procedure for $U(n, t, y, \langle\vec{x}\rangle)$. In our definition, the "resource bound" will concern the sizes of code numbers of tuples of outputs of intermediate computations. The tuples will be "horizontal" (synchronic) in the case of composition and "vertical" (diachronic) in the case of recursion and minimalization. The resource bound will allow us to bound quantifiers in our primitive recursive derivation. We define the parametrized family by simultaneous recursion and course-of-values recursion. You might be worried about stuffing variable numbers of arguments, but recursive calls to U take care of it! Watch closely how that works. (I guess you will have to when you do the exercise). Strictly speaking, this is a course-of-values recursion (on which variable?), so aren't you happy we already know that course-of-values recursion is a primitive recursive operator?

$$\begin{aligned} U(n,t,y,i) \quad = \quad & [lh(i) = 0 \wedge y = n] \vee \\ & [lh(i) > 0 \wedge lh(n) = 0 \wedge y = 0] \vee \\ & \ldots \text{ some other disjuncts} \ldots \\ & [lh(i) > 0 \wedge lh(n) = 3 \wedge (\exists_{z \leq t})[lh(z) = lh((n)_1) \wedge \\ & (\forall_{w \leq lh((n)_1))} U(((n)_1)_w, t, (z)_w, i) \wedge U(d(n)_0, t, y, z)] \vee \\ & \ldots \text{ more disjuncts} \ldots \end{aligned}$$

Exercise 5.1 *Finish it. Also, where can you place this definition in the Grzegorczyk hierarchy? Draw on work you have already done and explain as simply as possible (i.e., in a way that you might remember).*

Observe that the definition of this relation is elementary. Thus, we have shown:

Proposition 5.2 ***(Kleene Normal Form)*** *There exists an elementary relation U such that for each n, n-ary x,*

$$\phi_n^k(\vec{x}) \simeq ((\mu z)U(n, (z)_0, (z)_1, \langle\vec{x}\rangle))_1.$$

This is remarkable. It says that all the power of primitive recursion after Grzegorczyk class E_3 is superfluous for arriving at all the partial recursive functions. In fact, E_2 suffices. That is the solution to Hilbert's 10^{th} question. The following is a less informative corollary.

Proposition 5.3 ***(Universal Machine Theorem)*** *There exists a u such that for each n, k, and k-ary x,*

$$\phi_n^k(\vec{x}) = \phi_u^2(n, \langle \vec{x} \rangle).$$

Proof. Just set $\phi_u^2(n, \langle \vec{x} \rangle) = ((\mu z)U(n, (z)_0, (z)_1, \langle \vec{x} \rangle))_1$, which is justified by the fact that the right-hand side is a partial recursive derivation tree. $\dashv$

5.7 The *s-m-n* Theorem

Despite the nearly great name, the *s-m-n* theorem is a pretty tame fact. It says that you can effectively "stuff" arguments to obtain programs that act as though the arguments were already received. But it will turn out that together with the universal theorem, the *s-m-n* property codifies everything about our indexing that is essential for computability theory. So by simply assuming the universal and *s-m-n* theorems as axioms, we can ditch all the fussy details once and for all! But not until we prove it and show that it has that pivotal significance.

Proposition 5.4 s-m-n ***Theorem*** *There exists a primitive recursive function $s_m^n(n, \vec{x})$, where $\vec{x}$ is m-ary, such that for each n-ary $\vec{y}$,*

$$\phi^n_{s_m^n(i,\vec{x})}(\vec{y}) \simeq \phi_i^{m+n}(\vec{x}, \vec{y}).$$

How to prove it? Well, it's pretty obvious we want $s_m^n(i, \vec{x})$ to return an index of the y such that:

$$\psi(\vec{y}) \simeq \phi_i^{m+n}(c_{x_1}(p_n^1)(\vec{y}), \ldots, c_{x_m}(p_n^1)(\vec{y}), p_n^1(\vec{y}), \ldots, p_n^n(\vec{y})).$$

This involves crossing from index i in the $\phi_{\_}^{m+n}$ indexing to some index j in the $\phi_{\_}^n$ indexing.
First, we have to define a primitive recursive function *proj* such that for all n-ary $\vec{x}$,

$$\phi^n_{proj(i,n)}(\vec{x}) \simeq x_i.$$

But that's easy, since our coding doesn't care about arity:

$$\begin{aligned} proj(i, n) &= proj(i) \\ &= \langle i, 0 \rangle. \end{aligned}$$

Next, we need to be able to compute the index of a constant function from the constant. Easy, but annoying.

Exercise 5.2 *Define a primitive recursive function const such that for all x,*

$$\phi^n_{const(n)}(x) \simeq n.$$

Hint: you could use the answer to exercise 3.

And we have to write a primitive recursive program that knows how to return the index of a composition from the indices of the functions involved.

Exercise 5.3 *Define a primitive recursive function comp such that for all j, and m-ary i,*

$$\phi^k_{comp(j,\vec{i})} = C(\phi^k_{i_1}, \ldots, \phi^k_{i_m}).$$

Exercise 5.4 *Now it's easy to use the above to define $s^n_m(i, \vec{x})$.*