Partial recursive functions

Aim

A more abstract, machine-independent description of the collection of computable partial functions than provided by register/Turing machines:

> *they form the smallest collection of partial functions containing some basic functions and closed under some fundamental operations for forming new functions from old—composition, primitive recursion and minimization.*

The characterization is due to Kleene (1936), building on work of Gödel and Herbrand.

Basic functions

- Projection functions, $\mathtt{proj}_i^n \in \mathbb{N}^n \to \mathbb{N}$:

$$\mathtt{proj}_i^n(x_1, \ldots, x_n) \triangleq x_i$$

- Constant functions with value 0, $\mathtt{zero}^n \in \mathbb{N}^n \to \mathbb{N}$:

$$\mathtt{zero}^n(x_1, \ldots, x_n) \triangleq 0$$

- Successor function, $\mathtt{succ} \in \mathbb{N} \to \mathbb{N}$:

$$\mathtt{succ}(x) \triangleq x + 1$$

Composition

Composition of $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ with $g_1, \ldots, g_n \in \mathbb{N}^m \rightharpoonup \mathbb{N}$ is the partial function $f \circ [g_1, \ldots, g_n] \in \mathbb{N}^m \rightharpoonup \mathbb{N}$ satisfying for all $x_1, \ldots, x_m \in \mathbb{N}$

$$f \circ [g_1, \ldots, g_n](x_1, \ldots, x_m) \equiv f(g_1(x_1, \ldots, x_m), \ldots, g_n(x_1, \ldots, x_m))$$

where $\equiv$ is "Kleene equivalence" of possibly-undefined expressions: $\mathrm{LHS} \equiv \mathrm{RHS}$ means "either both LHS and RHS are undefined, or they are both defined and are equal."

Primitive recursion

Theorem. Given $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ and $g \in \mathbb{N}^{n+2} \rightharpoonup \mathbb{N}$, there is a unique $h \in \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ satisfying

$$\begin{cases} h(\vec{x}, 0) & \equiv f(\vec{x}) \\ h(\vec{x}, x+1) & \equiv g(\vec{x}, x, h(\vec{x}, x)) \end{cases}$$

for all $\vec{x} \in \mathbb{N}^n$ and $x \in \mathbb{N}$.

We write $\rho^n(f, g)$ for h and call it the partial function defined by primitive recursion from f and g.

Minimization

Given a partial function $f \in \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$, define $\mu^n f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ by

$$\mu^n f(\vec{x}) \triangleq \begin{array}{l}\text{least } x \text{ such that } f(\vec{x}, x) = 0 \text{ and for} \\ \text{each } i = 0, \ldots, x-1,\ f(\vec{x}, i) \text{ is defined} \\ \text{and } > 0 \\ \text{(undefined if there is no such } x\text{)}\end{array}$$

In other words

$$\mu^n f = \{(\vec{x}, x) \in \mathbb{N}^{n+1} \mid \exists y_0, \ldots, y_x \quad (\bigwedge_{i=0}^{x} f(\vec{x}, i) = y_i) \wedge (\bigwedge_{i=0}^{x-1} y_i > 0) \wedge y_x = 0\}$$

Definition. A partial function f is partial recursive ($f \in \mathrm{PR}$) if it can be built up in finitely many steps from the basic functions by use of the operations of composition, primitive recursion and minimization.

In other words, the set PR of partial recursive functions is the smallest set (with respect to subset inclusion) of partial functions containing the basic functions and closed under the operations of composition, primitive recursion and minimization.

Definition. A partial function f is partial recursive ($f \in \mathrm{PR}$) if it can be built up in finitely many steps from the basic functions by use of the operations of composition, primitive recursion and minimization.

Theorem. Every $f \in \mathrm{PR}$ is computable.
Proof. Just have to show:

$\mu^n f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ is computable if $f \in \mathbb{N}^{n+1} \rightharpoonup \mathbb{N}$ is.

Suppose f is computed by RM program F (with our usual I/O conventions). Then the RM specified on the next slide computes $\mu^n f$. (We assume $\mathtt{X}_1, \ldots, \mathtt{X}_n, \mathtt{C}$ are some registers not mentioned in F; and that the latter only uses registers $\mathtt{R}_0, \ldots, \mathtt{R}_N$, where $N \geq n+1$.)

Computable = partial recursive

Theorem. Not only is every $f \in \mathrm{PR}$ computable, but conversely, every computable partial function is partial recursive.

Proof (sketch). Let $f \in \mathbb{N}^n \rightharpoonup \mathbb{N}$ be computed by RM M with $N \geq n$ registers, say. Recall how we coded instantaneous configurations $c = (\ell, r_0, \ldots, r_N)$ of M as numbers $\ulcorner [\ell, r_0, \ldots, r_N] \urcorner$. It is possible to construct primitive recursive functions $lab, val_0, next_M \in \mathbb{N} \to \mathbb{N}$ satisfying

$$\begin{aligned} lab(\ulcorner [\ell, r_0, \ldots, r_N] \urcorner) &= \ell \\ val_0(\ulcorner [\ell, r_0, \ldots, r_N] \urcorner) &= r_0 \\ next_M(\ulcorner [\ell, r_0, \ldots, r_N] \urcorner) &= \text{code of } M\text{'s next configuration} \end{aligned}$$

(Showing that $next_M \in \mathrm{PRIM}$ is tricky—proof omitted.)

Proof sketch, cont.

Writing $\vec{x}$ for $x_1, \ldots, x_n$, let $config_M(\vec{x}, t)$ be the code of M's configuration after t steps, starting with initial register values $\mathtt{R}_0 = 0, \mathtt{R}_1 = x_1, \ldots, \mathtt{R}_n = x_n, \mathtt{R}_{n+1} = 0, \ldots, \mathtt{R}_N = 0$. It's in PRIM because:

$$\begin{cases} config_M(\vec{x}, 0) & = \ulcorner [0, 0, \vec{x}, \vec{0}] \urcorner \\ config_M(\vec{x}, t+1) & = next_M(config_M(\vec{x}, t)) \end{cases}$$

Proof sketch, cont.

Writing $\vec{x}$ for $x_1, \ldots, x_n$, let $config_M(\vec{x}, t)$ be the code of M's configuration after t steps, starting with initial register values $\mathtt{R}_0 = 0, \mathtt{R}_1 = x_1, \ldots, \mathtt{R}_n = x_n, \mathtt{R}_{n+1} = 0, \ldots, \mathtt{R}_N = 0$. It's in PRIM because:

$$\begin{cases} config_M(\vec{x}, 0) & = \ulcorner [0, 0, \vec{x}, \vec{0}] \urcorner \\ config_M(\vec{x}, t+1) & = next_M(config_M(\vec{x}, t)) \end{cases}$$

Can assume M has a single `HALT` as last instruction, Ith say (and no erroneous halts). Let $halt_M(\vec{x})$ be the number of steps M takes to halt when started with initial register values $\vec{x}$ (undefined if M does not halt). It satisfies

$$halt_M(\vec{x}) \equiv \text{least } t \text{ such that } I - lab(config_M(\vec{x}, t)) = 0$$

and hence is in PR (because $lab, config_M, I - (\ ) \in \mathrm{PRIM}$).

Proof sketch, cont.

Writing $\vec{x}$ for $x_1, \ldots, x_n$, let $config_M(\vec{x}, t)$ be the code of M's configuration after t steps, starting with initial register values $\mathtt{R}_0 = 0, \mathtt{R}_1 = x_1, \ldots, \mathtt{R}_n = x_n, \mathtt{R}_{n+1} = 0, \ldots, \mathtt{R}_N = 0$. It's in PRIM because:

$$\begin{cases} config_M(\vec{x}, 0) & = \ulcorner [0, 0, \vec{x}, \vec{0}] \urcorner \\ config_M(\vec{x}, t+1) & = next_M(config_M(\vec{x}, t)) \end{cases}$$

Can assume M has a single $\mathtt{HALT}$ as last instruction, Ith say (and no erroneous halts). Let $halt_M(\vec{x})$ be the number of steps M takes to halt when started with initial register values $\vec{x}$ (undefined if M does not halt). It satisfies

$$halt_M(\vec{x}) \equiv \text{least } t \text{ such that } I - lab(config_M(\vec{x}, t)) = 0$$

and hence is in PR (because $lab, config_M, I - (\ ) \in \mathrm{PRIM}$).
So $f \in \mathrm{PR}$, because $f(\vec{x}) \equiv val_0(config_M(\vec{x}, halt_M(\vec{x})))$.

Definition. A partial function f is partial recursive ($f \in \mathrm{PR}$) if it can be built up in finitely many steps from the basic functions by use of the operations of composition, primitive recursion and minimization.

The members of PR that are total are called recursive functions.

Fact: there are recursive functions that are not primitive recursive. For example...

Ackermann's function

There is a (unique) function $ack \in \mathbb{N}^2 \to \mathbb{N}$ satisfying

$$\begin{aligned} ack(0, x_2) &= x_2 + 1 \\ ack(x_1 + 1, 0) &= ack(x_1, 1) \\ ack(x_1 + 1, x_2 + 1) &= ack(x_1, ack(x_1 + 1, x_2)) \end{aligned}$$

Ackermann's function

There is a (unique) function $ack \in \mathbb{N}^2 \to \mathbb{N}$ satisfying

$$\begin{aligned} ack(0, x_2) &= x_2 + 1 \\ ack(x_1 + 1, 0) &= ack(x_1, 1) \\ ack(x_1 + 1, x_2 + 1) &= ack(x_1, ack(x_1 + 1, x_2)) \end{aligned}$$

- ack is computable, hence recursive [proof: exercise].

Ackermann's function

There is a (unique) function $ack \in \mathbb{N}^2 \to \mathbb{N}$ satisfying

$$\begin{aligned} ack(0, x_2) &= x_2 + 1 \\ ack(x_1 + 1, 0) &= ack(x_1, 1) \\ ack(x_1 + 1, x_2 + 1) &= ack(x_1, ack(x_1 + 1, x_2)) \end{aligned}$$

- ack is computable, hence recursive [proof: exercise].
- **Fact:** ack grows faster than any primitive recursive function $f \in \mathbb{N}^2 \to \mathbb{N}$: $\exists N_f \; \forall x_1, x_2 > N_f \; (f(x_1, x_2) < ack(x_1, x_2))$.
  Hence ack is not primitive recursive.

Diagonalization

We can also construct a function that is recursive but not primitive recursive by diagonalization.

Diagonalization

We can also construct a function that is recursive but not primitive recursive by diagonalization.

We can fix a coding of primitive recursive functions by numbers. That is define a surjective function $p : \mathbb{N} \to \mathrm{PRIM}$ in such a way that the function $\chi : \mathbb{N}^2 \to \mathbb{N}$ defined by

$$\chi(n, x) = p(n)(x)$$

is computable.

Diagonalization

We can also construct a function that is recursive but not primitive recursive by diagonalization.

We can fix a coding of primitive recursive functions by numbers. That is define a surjective function $p : \mathbb{N} \to \mathrm{PRIM}$ in such a way that the function $\chi : \mathbb{N}^2 \to \mathbb{N}$ defined by

$$\chi(n, x) = p(n)(x)$$

is computable.

Then $d : \mathbb{N}^2 \to \mathbb{N}$ defined by

$$d(x) = \chi(x, x) + 1$$

is computable but not primitive recursive.