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.

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

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

In other words

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

Example of minimization

integer part of x_1/x_2 $\equiv$ least x_3 such that
(undefined if $x_2{=}0$) $x_1 < x_2(x_3 + 1)$

Example of minimization

$$\begin{array}{rcl}\text{integer part of } x_1/x_2 & \equiv & \text{least } x_3 \text{ such that} \\ \text{(undefined if } x_2{=}0\text{)} & & x_1 < x_2(x_3+1) \\ & & \\ & \equiv & \mu^2 f(x_1,x_2)\end{array}$$

where $f \in \mathbb{N}^3 \to \mathbb{N}$ is

$$f(x_1,x_2,x_3) \triangleq \begin{cases} 1 & \text{if } x_1 \geq x_2(x_3+1) \\ 0 & \text{if } x_1 < x_2(x_3+1) \end{cases}$$

Definition. A partial function f is partial recursive ($f \in \mathbf{PR}$) 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 $\mathbf{PR}$ 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.

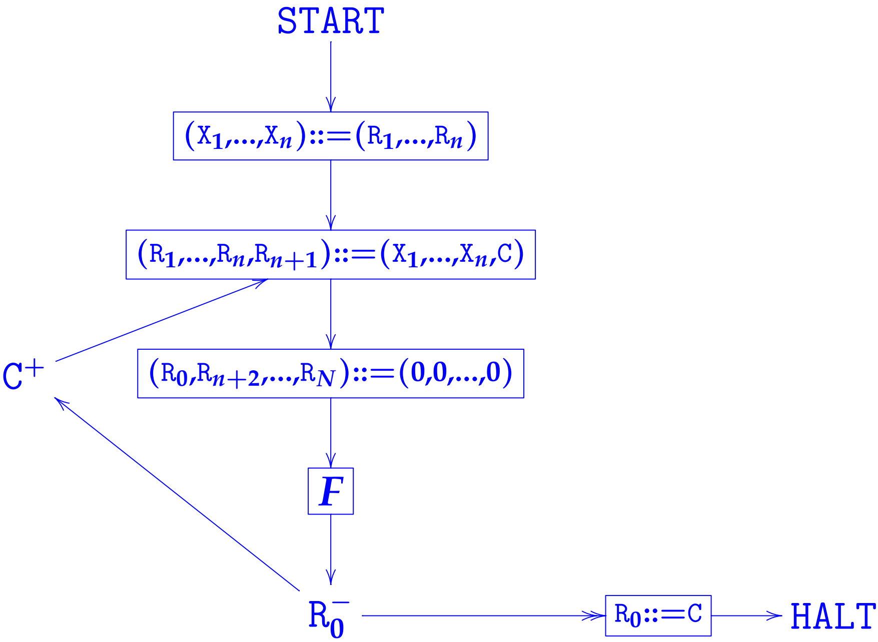
Definition. A partial function f is partial recursive ($f \in \mathbf{PR}$) if it can be built up in finitely many steps from the basic functions by use of the operations of composition, primitive recursion and minimization.

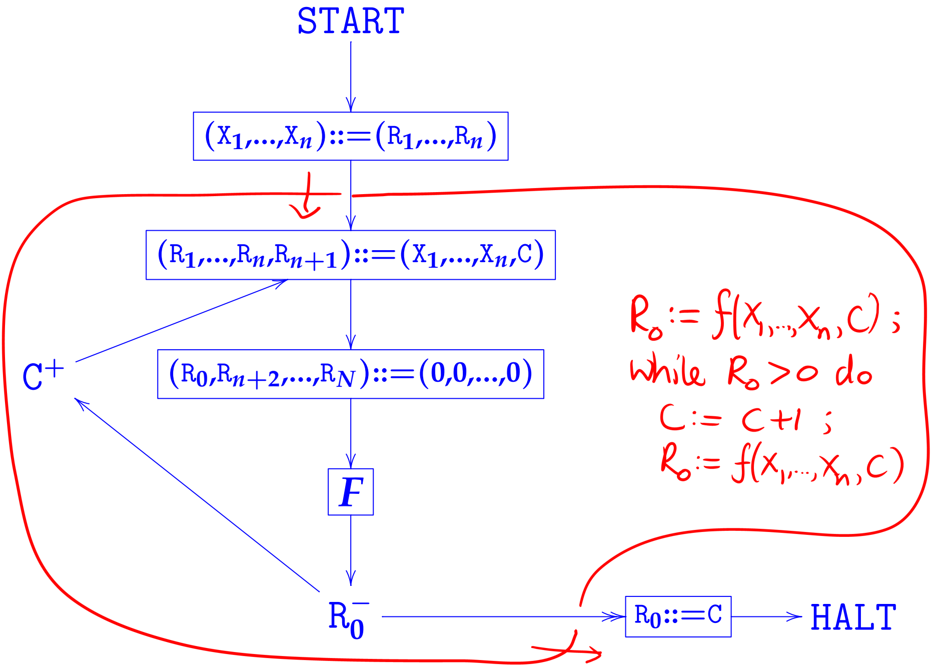
Theorem. Every $f \in \mathbf{PR}$ is computable.

Proof. Just have to show:

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


START
$(\mathtt{X}_1,\ldots,\mathtt{X}_n) ::= (\mathtt{R}_1,\ldots,\mathtt{R}_n)$
$(\mathtt{R}_1,\ldots,\mathtt{R}_n,\mathtt{R}_{n+1}) ::= (\mathtt{X}_1,\ldots,\mathtt{X}_n,\mathtt{C})$
$(\mathtt{R}_0,\mathtt{R}_{n+2},\ldots,\mathtt{R}_N) ::= (0,0,\ldots,0)$
F
$\mathtt{C}^+$
$\mathtt{R}_0^-$
$\mathtt{R}_0 ::= \mathtt{C}$
HALT



$R_0 := f(x_1,\ldots,x_n,C)$;
while $R_0 > 0$ do
  $C := C+1$;
  $R_0 := f(x_1,\ldots,x_n,C)$

Computable = partial recursive

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

Proof (sketch). Let f be computed by RM M. 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 $next_M \in \mathbf{PRIM}$ is tricky—proof omitted.)

Proof sketch, cont.

Let $\mathit{config}_M(\vec{x}, t)$ be the code of M's configuration after t steps, starting with initial register values $\vec{x}$. It's in **PRIM** because:

$$\begin{cases} \mathit{config}_M(\vec{x}, 0) & = \ulcorner [0, \vec{x}] \urcorner \\ \mathit{config}_M(\vec{x}, t+1) & = \mathit{next}_M(\mathit{config}_M(\vec{x}, t)) \end{cases}$$

Proof sketch, cont.

Let $config_M(\vec{x}, t)$ be the code of M's configuration after t steps, starting with initial register values $\vec{x}$. It's in $\mathbf{PRIM}$ because:

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

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 $\mathbf{PR}$ (because $lab, config_M, I - (\ ) \in \mathbf{PRIM}$).

Proof sketch, cont.

Let $config_M(\vec{x}, t)$ be the code of M's configuration after t steps, starting with initial register values $\vec{x}$. It's in $\mathbf{PRIM}$ because:

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

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

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

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

So $f \in \mathbf{PR}$, because $f(\vec{x}) \equiv val_0(config_M(\vec{x}, halt_M(\vec{x})))$.

Definition. A partial function f is partial recursive ($f \in \mathbf{PR}$) 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 $\mathbf{PR}$ 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

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

- 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.