

# 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.

# Primitive recursion

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

# Example: addition

Addition  $add \in \mathbb{N}^2 \rightarrow \mathbb{N}$  satisfies:

$$\begin{cases} add(x_1, 0) & \equiv x_1 \\ add(x_1, x + 1) & \equiv add(x_1, x) + 1 \end{cases}$$

So  $add = \rho^1(f, g)$  where  $\begin{cases} f(x_1) & \triangleq x_1 \\ g(x_1, x_2, x_3) & \triangleq x_3 + 1 \end{cases}$

Note that  $f = \text{proj}_1^1$  and  $g = \text{succ} \circ \text{proj}_3^3$ ; so  $add$  can be built up from basic functions using composition and primitive recursion:  $add = \rho^1(\text{proj}_1^1, \text{succ} \circ \text{proj}_3^3)$ .

# Example: predecessor

Predecessor  $pred \in \mathbb{N} \rightarrow \mathbb{N}$  satisfies:

$$\begin{cases} pred(0) & \equiv 0 \\ pred(x + 1) & \equiv x \end{cases}$$

So  $pred = \rho^0(f, g)$  where  $\begin{cases} f() & \triangleq 0 \\ g(x_1, x_2) & \triangleq x_1 \end{cases}$

Thus  $pred$  can be built up from basic functions using primitive recursion:  $pred = \rho^0(\text{zero}^0, \text{proj}_1^2)$ .

# Example: multiplication

Multiplication  $mult \in \mathbb{N}^2 \rightarrow \mathbb{N}$  satisfies:

$$\begin{cases} mult(x_1, 0) & \equiv 0 \\ mult(x_1, x + 1) & \equiv mult(x_1, x) + x_1 \end{cases}$$

and thus  $mult = \rho^1(\text{zero}^1, add \circ (\text{proj}_3^3, \text{proj}_1^3))$ .

So  $mult$  can be built up from basic functions using composition and primitive recursion (since  $add$  can be).

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

In other words, the set **PRIM** of primitive 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 and primitive recursion.

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

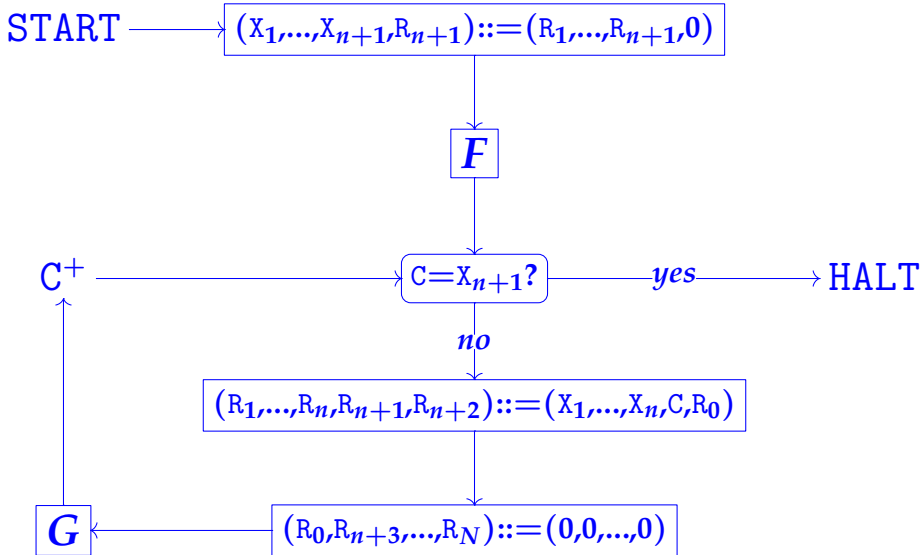
**Theorem.** Every  $f \in \mathbf{PRIM}$  is computable.

**Proof.** Already proved: basic functions are computable; composition preserves computability. So just have to show:

$\rho^n(f, g) \in \mathbb{N}^{n+1} \rightarrow \mathbb{N}$  computable if  $f \in \mathbb{N}^n \rightarrow \mathbb{N}$  and  $g \in \mathbb{N}^{n+2} \rightarrow \mathbb{N}$  are.

Suppose  $f$  and  $g$  are computed by RM programs  $F$  and  $G$  (with our usual I/O conventions). Then the RM specified on the next slide computes  $\rho^n(f, g)$ . (We assume  $X_1, \dots, X_{n+1}, C$  are some registers not mentioned in  $F$  and  $G$ ; and that the latter only use registers  $R_0, \dots, R_N$ , where  $N \geq n + 2$ .)





START

$(X_1, \dots, X_{n+1}, R_{n+1}) ::= (R_1, \dots, R_{n+1}, 0)$

$F$

$C^+$

$C = X_{n+1}?$

*yes*

HALT

*no*

$(R_1, \dots, R_n, R_{n+1}, R_{n+2}) ::= (X_1, \dots, X_n, C, R_0)$

$G$

$(R_0, R_{n+3}, \dots, R_N) ::= (0, 0, \dots, 0)$

*while  $C < X_{n+1}$  do  $(R_0, C) ::= (g(X_1, \dots, X_n, C, R_0), C+1)$*

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

Every  $f \in \text{PRIM}$  is a total function, because:

- ▶ all the basic functions are total
- ▶ if  $f, g_1, \dots, g_n$  are total, then so is  $f \circ (g_1, \dots, g_n)$  [why?]
- ▶ if  $f$  and  $g$  are total, then so is  $\rho^n(f, g)$  [why?]

so we need something more in order to characterise all computable partial 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.

# Minimization

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

$\mu^n f \in \mathbb{N}^n \rightarrow \mathbb{N}$  by

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

In other words

$$\mu^n f = \{(\vec{x}, x) \in \mathbb{N}^{n+1} \mid \exists y_0, \dots, y_x$$

$$\left( \bigwedge_{i=0}^x f(\vec{x}, i) = y_i \right) \wedge \left( \bigwedge_{i=0}^{x-1} y_i > 0 \right) \wedge y_x = 0 \}$$

# 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

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

$$\equiv \mu^2 f(x_1, x_2)$$

where  $f \in \mathbb{N}^3 \rightarrow \mathbb{N}$  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 **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.



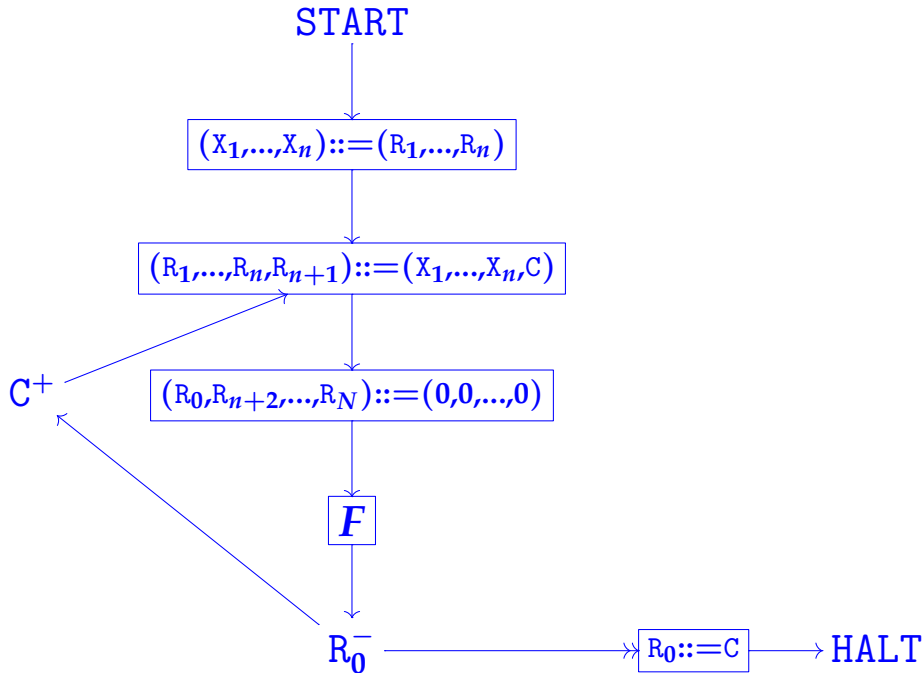
**Definition.** A partial function  $f$  is **partial recursive** ( $f \in \mathbf{PR}$ ) if it can be built up in finitely many steps from the basic functions by use of the operations of composition, primitive recursion and minimization.

**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  $X_1, \dots, X_n, C$  are some registers not mentioned in  $F$ ; and that the latter only uses registers  $R_0, \dots, R_N$ , where  $N \geq n + 1$ .)



START

$(X_1, \dots, X_n) ::= (R_1, \dots, R_n)$

$(R_1, \dots, R_n, R_{n+1}) ::= (X_1, \dots, X_n, C)$

$(R_0, R_{n+2}, \dots, R_N) ::= (0, 0, \dots, 0)$

$F$

$R_0^-$

$R_0 ::= C$

HALT

$C^+$

$R_0 := f(X_1, \dots, X_n, C);$   
while  $R_0 > 0$  do  
 $C := C + 1;$   
 $R_0 := f(X_1, \dots, X_n, C)$

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

$$lab(\ulcorner [\ell, r_0, \dots, r_N] \urcorner) = \ell$$

$$val_0(\ulcorner [\ell, r_0, \dots, r_N] \urcorner) = r_0$$

$$next_M(\ulcorner [\ell, r_0, \dots, r_N] \urcorner) = \text{code of } M\text{'s next configuration}$$

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

L8 (see Cutland, p95 et seq.) 

## Proof sketch, cont.

Writing  $\vec{x}$  for  $x_1, \dots, x_n$ , let  $config_M(\vec{x}, t)$  be the code of  $M$ 's configuration after  $t$  steps, starting with initial register values  $R_0 = \mathbf{0}, R_1 = x_1, \dots, R_n = x_n, R_{n+1} = \mathbf{0}, \dots, R_N = \mathbf{0}$ . 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  $x_1, \dots, x_n$ , let  $config_M(\vec{x}, t)$  be the code of  $M$ 's configuration after  $t$  steps, starting with initial register values  $R_0 = 0, R_1 = x_1, \dots, R_n = x_n, R_{n+1} = 0, \dots, R_N = 0$ . 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,  $I$ th 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  $lab, config_M, I - ( ) \in \text{PRIM}$ ).

## Proof sketch, cont.

Writing  $\vec{x}$  for  $x_1, \dots, x_n$ , let  $config_M(\vec{x}, t)$  be the code of  $M$ 's configuration after  $t$  steps, starting with initial register values  $R_0 = 0, R_1 = x_1, \dots, R_n = x_n, R_{n+1} = 0, \dots, R_N = 0$ . 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,  $I$ th 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  $lab, config_M, I - ( ) \in \text{PRIM}$ ).

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

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

In other words, the set **PRIM** of primitive 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 and primitive recursion.

FACT : every  $f \in \mathbf{PRIM}$  is a total function



**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  $ack \in \mathbb{N}^2 \rightarrow \mathbb{N}$  satisfying

$$ack(0, x_2) = x_2 + 1$$

$$ack(x_1 + 1, 0) = ack(x_1, 1)$$

$$ack(x_1 + 1, x_2 + 1) = ack(x_1, ack(x_1 + 1, x_2))$$

# Ackermann's function

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

$$ack(0, x_2) = x_2 + 1$$

$$ack(x_1 + 1, 0) = ack(x_1, 1)$$

$$ack(x_1 + 1, x_2 + 1) = ack(x_1, ack(x_1 + 1, x_2))$$

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

OCaml version 4.00.1

```
# let rec ack (x : int)(y : int) : int =
  match x ,y with
    0 , y -> y+1
  | x , 0 -> ack (x-1) 1
  | x ,y -> ack (x-1) (ack x (y-1));;
val ack : int -> int -> int = <fun>
# ack 0 0;;
- : int = 1
# ack 1 1;;
- : int = 3
# ack 2 2;;
- : int = 7
# ack 3 3;;
- : int = 61
# ack 4 4;;
Stack overflow during evaluation (looping recursion?).
#
```

OCaml version 4.00.1

```
# let rec ack (x : int)(y : int) : int =  
  match x ,y with  
    0 , y -> y+1  
  | x , 0 -> ack (x-1) 1  
  | x ,y -> ack (x-1) (ack x (y-1));;  
val ack : int -> int -> int = <fun>
```

```
# ack 0 0;;
```

```
- : int = 1
```

```
# ack 1 1;;
```

```
- : int = 3
```

```
# ack 2 2;;
```

```
- : int = 7
```

```
# ack 3 3;;
```

```
- : int = 61
```

```
# ack 4 4;;
```

```
Stack overflow during evaluation (looping recursion?).
```

```
#
```

$$(\text{ack } 4 \ 4 = 2^{2^{2^{2^{2^2}}}} - 3)$$

# Ackermann's function

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

$$ack(0, x_2) = x_2 + 1$$

$$ack(x_1 + 1, 0) = ack(x_1, 1)$$

$$ack(x_1 + 1, x_2 + 1) = ack(x_1, ack(x_1 + 1, x_2))$$

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

# Ackermann's function

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

$$ack(0, x_2) = x_2 + 1$$

$$ack(x_1 + 1, 0) = ack(x_1, 1)$$

$$ack(x_1 + 1, x_2 + 1) = ack(x_1, ack(x_1 + 1, x_2))$$

In fact, writing  $a_x$  for  $ack(x, -) \in \mathbb{N} \rightarrow \mathbb{N}$ , one has

$$a_{x+1}(y) = \underbrace{(a_x \circ \dots \circ a_x)}_{\text{compose } y \text{ times}}(1)$$

← this is an e.g. of  
a prim-rec. definition  
"of higher type"