Partial recursive functions

We saw above that

primitive recursive $\Rightarrow$ computable & total

Since not every computable partial function is total, it is certainly not the case that every computable partial function is primitive recursive.

One intuitively algorithmic method of calculation that can give rise to non-total partial functions is that of searching for the smallest value of a function's argument that produces a given result (zero, say) – since no such value may exist. The corresponding operation on functions is called minimization...

Minimization

Given $f \in Pfn(\mathbb{N}^{n+1}, \mathbb{N})$

define $\mu(f) \in Pfn(\mathbb{N}^n, \mathbb{N})$ by

$$\boxed{\begin{array}{l} \mu(f)(x_1,\dots,x_n) = x \overset{\text{def}}{\Longleftrightarrow} \text{there exist } y_0, y_1, \dots, y_x \\ \quad \text{such that } f(x_1,\dots,x_n,i) = y_i \text{ for } i = 0,1,\dots,x \\ \quad\quad \& \quad y_i > 0 \text{ for } i = 0,1,\dots,x-1 \\ \quad\quad \& \quad y_x = 0 \end{array}}$$

Thus

$$\mu(f)(x_1,\dots,x_n) \equiv \left\{ \begin{array}{l} \text{the least } x \text{ such that} \\ f(x_1,\dots,x_n,x) = 0 \text{ and} \\ f(x_1,\dots,x_n,i) > 0 \text{ for } i < x \end{array} \right.$$

(in particular, $f(x_1,\dots,x_n,i)\downarrow$ for $i<x$)

and in particular

$\mu(f)(x_1,\dots,x_n)\uparrow$ if no x exists satisfying these conditions.

PROPOSITION:

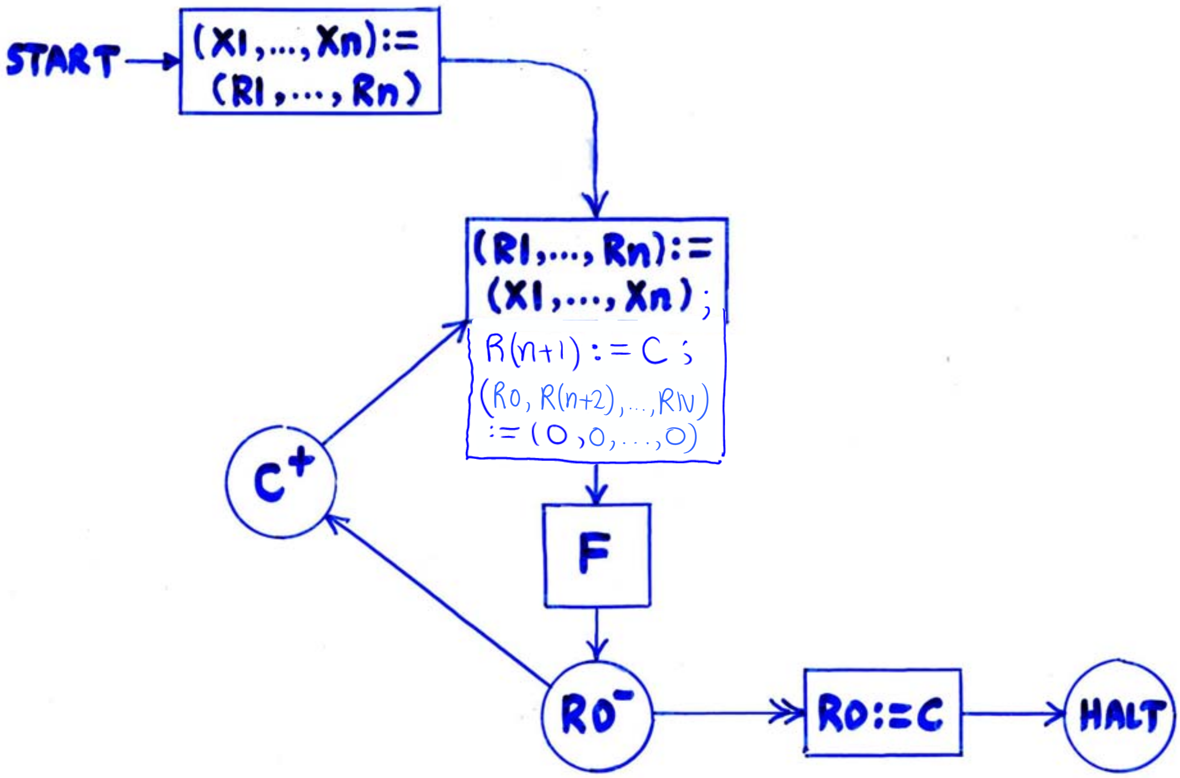
If f is computable, then so is $\mu(f)$.

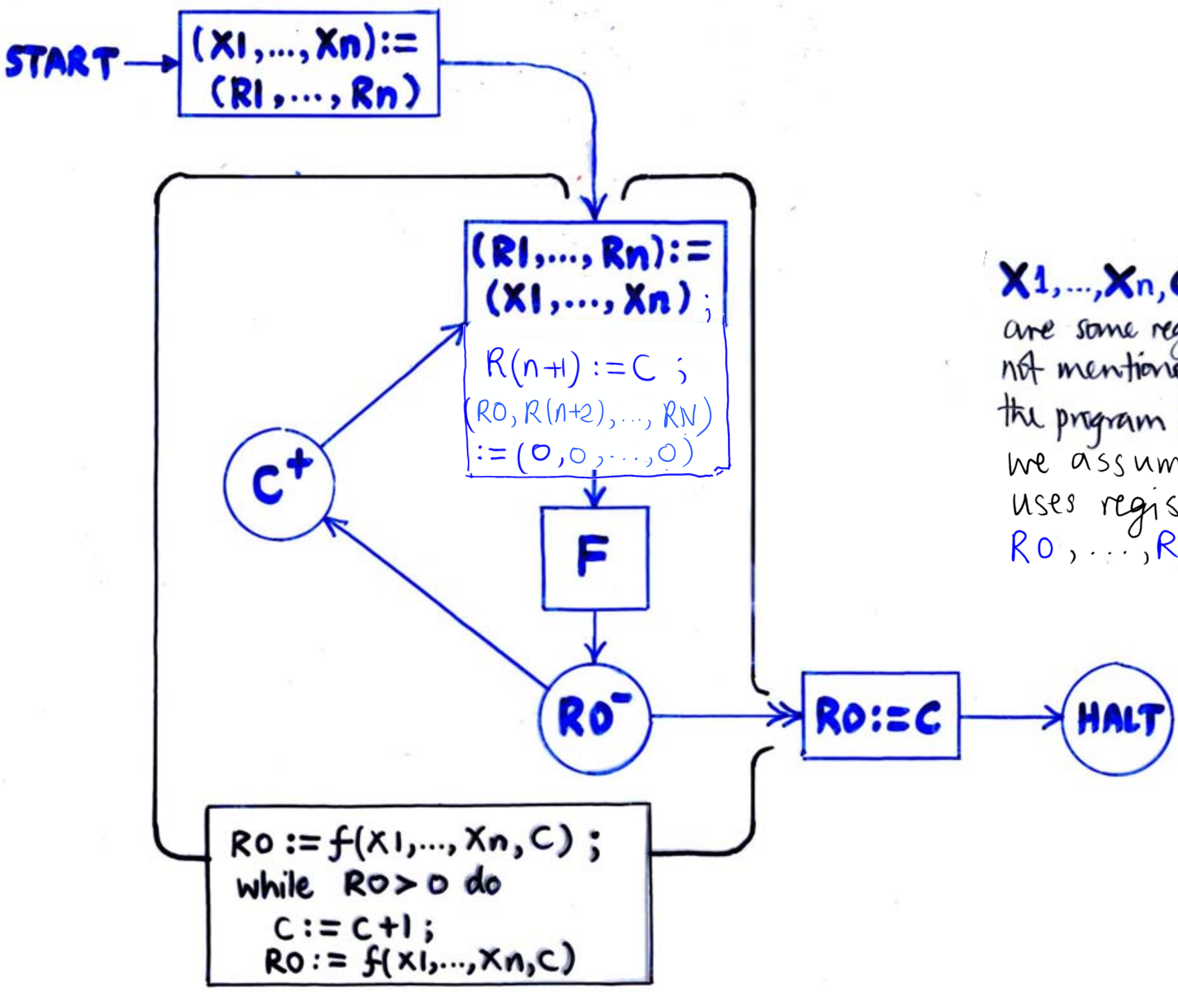
Proof

Given a register machine program

F computing $f(x_1,\dots,x_{n+1})$ in R0 starting with R1, …, R(n+1) set to $x_1,\dots,x_{n+1}$

then the following diagram specifies a register machine program for computing $\mu(f)(x_1,\dots,x_n)$ in R0 starting with R1, …, Rn set to $x_1,\dots,x_n$:





$X_1, \ldots, X_n, C$ are some registers not mentioned in the program **F**, which we assume only uses registers $R0, \ldots, RN$ $(N > n)$.

DEFINITION:

A **partial recursive function** is a partial function that can be built up from the basic functions by repeated use of the operations of composition, primitive recursion and minimization.

In other words, the set $\boxed{PR}$ of partial recursive functions is the smallest set of partial functions containing the basic functions and closed under the operations of composition, primitive recursion and minimization.

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

EXAMPLES of minimization

Ex.1 The everywhere undefined function is partial recursive. For if $f(x,y) \overset{\text{def}}{=} 1$, then $\mu(f)(x)\uparrow$, for all x;
and $f = suc \circ zero^2$; so $\mu(f) = \mu(suc \circ zero^2)$ is partial recursive.

Ex.2

$$\left.\begin{array}{l} d(x,y) \overset{\text{def}}{=} \text{integer part of } x/y \text{ (undefined if } y=0) \\ m(x,y) \overset{\text{def}}{=} \text{remainder when } x \text{ is divided by } y \end{array}\right\} \text{are partial recursive.}$$

For note that

$$d(x,y) \equiv \text{least } z \text{ such that } x < y\cdot(z+1) \qquad (\text{thus } d(x,0)\uparrow).$$

Now

$$ge(x,y) \overset{\text{def}}{=} \begin{cases} 0 & \text{if } x<y \\ 1 & \text{if } x \geq y \end{cases}$$ is primitive recursive, since

$ge(x,y) = ifzero(y \dot{-} x, 1, 0)$ (and we saw above that ifzero & $\dot{-}$ are in PRIM).

So $f(x,y,z) \overset{\text{def}}{=} ge(x, y.(z+1))$ is also in PRIM (since multiplication is)

Then $d = \mu(f)$ is partial recursive.

Finally, note that $m(x,y) \equiv x \dot{-} y.d(x,y)$, so m is also in PR.

N.B.
The function d in Ex. 2 is not total recursive because it is undefined when its second argument is 0.
Thus

$$\mathrm{div}(x,y) \stackrel{\text{def}}{=} \begin{cases} \text{integer part of } x/y & \text{if } y>0 \\ 0 & \text{if } y=0 \end{cases}$$

is (total) recursive: $\mathrm{div} = \mathrm{ifzero} \circ (\mathrm{proj}_2^2, \mathrm{zero}^2, d)$.

In fact, div is primitive recursive (exercise: prove this).

Every $f \in \mathrm{PRIM}$ satisfies
$f \in \mathrm{PR}$ & f is total

BUT converse is false:

$\boxed{\text{there are total recursive functions which are not primitive recursive}}$

Here is a sketch of the proof of this, making use of our next major result – the Theorem on page 114 ...

Proof (sketch)

First, (eg $\rho^1(\mathrm{proj}^1_1, \mathrm{suc} \circ \mathrm{proj}^3_3)$ is a formal description for $\mathrm{add}(x,y) \stackrel{\text{def}}{=} x+y$)

code $\boxed{\text{formal descriptions}}$ of primitive recursive functions as numbers so that

$$e(x,y) \stackrel{\text{def}}{=} \begin{cases} f_x(y) & \text{if } x \text{ is code of a formal description of a unary prim. rec. function, } f_x \text{ say} \\ 0 & \text{if } x \text{ is not the code of a formal description of a unary prim. rec. function} \end{cases}$$

is a computable function.

Next,

[this is a similar diagonalization trick to that in the proof that not all functions are computable

consider $e'(x) \stackrel{\text{def}}{=} e(x,x)+1$

CLAIM: $e' \in PR$, but $e' \notin PRIM$.

Next we make use of the Theorem to be proved below (p114), namely that PR coincides with the collection of computable functions. Since e is computable, this Theorem implies that it is in PR — and hence so is e' since

$$e' = \mathrm{suc} \circ (e \circ (\mathrm{proj}^1_1, \mathrm{proj}^1_1)).$$

To see that $e' \notin PRIM$, suppose the contrary and derive a contradiction: if $e' \in PRIM$, then it would have a formal description, and hence $e' = f_x$ for some code x. Then

$$\begin{aligned} f_x(x) &= e'(x) && \text{since } e' = f_x \\ &= e(x,x)+1 && \text{by definition of } e' \\ &= f_x(x)+1 && \text{by definition of } e \end{aligned}$$

which is impossible! $\square$

Here is a more explicit example of a non-primitive-recursive member of PR:

Ackermann's function

FACT : there is a total function $ack \in Fun(\mathbb{N} \times \mathbb{N}, \mathbb{N})$ satisfying

$$\begin{cases} ack(0, y) = y+1 \\ ack(x+1, 0) = ack(x, 1) \\ ack(x+1, y+1) = ack(x, ack(x+1, y)) \end{cases}$$

ack is recursive, but not primitive recursive.

It is beyond the scope of this course to prove that $ack \notin PRIM$.
(Roughly speaking, $ack \notin PRIM$ because as x & y increase, $ack(x, y)$ grows faster than any primitive recursive function possibly can grow.)
One way to see that $ack \in PR$ is to design a register machine to compute ack (exercise), and then appeal to the Theorem we are about to prove that states that PR coincides with the set of computable functions.
The proof that the register machine for ack always halts (i.e. that ack is total) is non-trivial. (It can be done by "well-founded induction" on pairs $(x, y) \in \mathbb{N}^2$ ordered lexicographically: $(x_1, y_1) < (x_2, y_2) \Leftrightarrow (x_1 < x_2 \text{ or } (x_1 = x_2 \;\&\; y_1 < y_2))$.)

THEOREM:

A partial function is (register machine) computable if & only if it is partial recursive.

We have already proved that the collection of computable partial functions contains the basic functions and is closed under the operations of composition, primitive recursion and minimization, and hence contains all partial recursive functions.

So it remains to see that

$$f \text{ computable} \Rightarrow f \text{ partial recursive}$$

Proof of (f computable $\Rightarrow f \in PR$)

If $f \in Pfn(\mathbb{N}^n, \mathbb{N})$ is computable, there is a register machine M which when started with $R1, \ldots, Rn$ set to $x_1, \ldots, x_n$ (and all other registers set to 0), halts if & only if $f(x_1, \ldots, x_n)\downarrow$, and in that case $R0$ contains this value.

Suppose the registers of M are $R0, R1, R2, \ldots, Rn, R(n+1), \ldots, Rm$ (for some $m \geq n$).

Suppose M's program has instructions labelled $L0, L2, \ldots, LI$ (some $I \geq 0$), and without loss of generality assume that the only HALT instruction is the last one (LI) and that there are no erroneous halts (i.e. the only labels referred to in increment/decrement instructions lie in the range $L0, \ldots, LI$).

The <u>state</u> of M at any stage in its computation can be specified by the code $[\ell, r_0, r_1, \ldots, r_m]$ of a list of length $m+2$, where

ℓ = current instruction (so $0 \leq \ell \leq I$)

r_j = current contents of Rj $(j = 0, 1, \ldots, m)$.

The proof that f is partial recursive depends upon the following lemmas:

LEMMA 1:

There are primitive recursive functions $lab, val_0, val_1, \ldots, val_m \in Fun(\mathbb{N}, \mathbb{N})$ satisfying

$$\begin{cases} lab([\ell, r_0, \ldots, r_m]) = \ell \\ val_j([\ell, r_0, \ldots, r_m]) = r_j \end{cases} \quad \left(\begin{array}{l} \text{for all } j = 0, \ldots, m \text{ and} \\ \text{all } (\ell, r_0, \ldots, r_m) \in \mathbb{N}^{m+2} \end{array}\right)$$

Thus lab gives the label of a state of M, whilst val_j gives the value held in R_j.

LEMMA 2:

There is a primitive recursive function $next \in Fun(\mathbb{N}, \mathbb{N})$ which gives the next state of M in terms of the current one.

The proof of these lemmas uses the following property of our coding of lists of numbers as numbers ...

PROPOSITION:

The functions

$mklist^n \in Fun(\mathbb{N}^n, \mathbb{N})$

$hd, tl \in Fun(\mathbb{N}, \mathbb{N})$

defined by

$$mklist^n(x_1, \ldots, x_n) \stackrel{def}{=} [x_1, \ldots, x_n]$$

$$hd(x) \stackrel{def}{=} \begin{cases} x_1 & \text{if } x = [x_1, \ldots, x_n] \text{ for some } n > 0 \ \& \ x_1, \ldots, x_n \\ 0 & \text{if } x = [nil] = 0 \end{cases}$$

$$tl(x) \stackrel{def}{=} \begin{cases} [x_2, \ldots, x_n] & \text{if } x = [x_1, \ldots, x_n] \text{ for some } n > 0 \ \& \ x_1, \ldots, x_n \\ 0 & \text{if } x = [nil] = 0 \end{cases}$$

are all primitive recursive.

Proof of (f computable $\Rightarrow f \in PR$), cont.

First, note that

$$\mathrm{state}(x_1,\ldots,x_n,t) \stackrel{\text{def}}{=} \begin{cases} \text{State of } M \text{ at } t^{\text{th}} \text{ step,} \\ \text{starting with } R1=x_1,\ldots,Rn=x_n \\ \text{\& all other registers} = 0 \end{cases}$$

is primitive recursive, because

$$\begin{cases} \mathrm{state}(x_1,\ldots,x_n,0) = [0,0,x_1,\ldots,x_n,0,\ldots,0] \\ \mathrm{state}(x_1,\ldots,x_n,t+1) = \mathrm{next}(\mathrm{state}(x_1,\ldots,x_n,t)) \end{cases}$$

so that

$$\mathrm{state} = \rho^n(\mathrm{mklist}^{m+2} \circ (\mathrm{zero}^n, \mathrm{zero}^n, \mathrm{proj}_1^n, \ldots, \mathrm{proj}_n^n, \mathrm{zero}^n, \ldots, \mathrm{zero}^n), \mathrm{next} \circ \mathrm{proj}_{n+2}^{n+2})$$

with mklist^{m+2}, $\mathrm{next} \in \mathrm{PRIM}$.

Now:

$$f(x_1,\ldots,x_n) \equiv \mathrm{val}_0(\mathrm{state}(x_1,\ldots,x_n,\mathrm{halt}(x_1,\ldots,x_n)))$$

where

$$\mathrm{halt}(x_1,\ldots,x_n) \stackrel{\text{def}}{=} \begin{cases} \text{number of steps taken to halt} \\ \text{(\& undefined if never halt)} \end{cases}$$

$$\equiv \text{least } t \text{ such that } \mathrm{lab}(\mathrm{state}(x_1,\ldots,x_n,t)) = I$$

$$\equiv \mu(h)(x_1,\ldots,x_n)$$

where $h(x_1,\ldots,x_n,t) \stackrel{\text{def}}{=} I \dot{-} \mathrm{lab}(\mathrm{state}(x_1,\ldots,x_n,t))$.

Since lab, state & $\dot{-}$ are in PRIM, so is h and hence $\mathrm{halt} = \mu(h)$ is in PR.

Thus $f = \mathrm{val}_0 \circ (\mathrm{state} \circ (\mathrm{proj}_1^n, \ldots, \mathrm{proj}_n^n, \mu(h)))$ is also in PR.

$\square$

To complete the proof of the Theorem, we have to prove the Proposition and Lemmas 1 & 2.

Proof of the Proposition

Since $\text{mklist}^n(x_1, \ldots, x_n) = \langle x_1, \langle x_2, \ldots \langle x_n, 0 \rangle \cdots \rangle \rangle$,
to see that $\text{mklist}^n \in \text{PRIM}$, it suffices to show that $\langle x, y \rangle = 2^x(2y+1)$ is primitive recursive — which follows from the fact that multiplication and exponentiation are in PRIM.

The proof that hd and tl are in PRIM requires more effort. We get to their primitive recursivity via that of a number of intermediate functions:

(i) $\text{mod}_2(x) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } x \text{ even} \\ 1 & \text{if } x \text{ odd} \end{cases}$ is primitive recursive, because

it satisfies $\begin{cases} \text{mod}_2(0) = 0 \\ \text{mod}_2(x+1) = \text{ifzero}(\text{mod}_2(x), 1, 0) \end{cases}$.

(ii) $\text{half}(x) \stackrel{\text{def}}{=}$ integer part of $x/2$ is primitive recursive by (i), since

it satisfies $\begin{cases} \text{half}(0) = 0 \\ \text{half}(x+1) = \text{ifzero}(\text{mod}_2(x), \text{half}(x), \text{half}(x)+1) \end{cases}$.

(iii)
$$f(x,y) \stackrel{\text{def}}{=} \begin{cases} x/2^y & \text{if } x > 0 \ \& \ 2^y \text{ divides } x \\ 0 & \text{otherwise} \end{cases}$$

is primitive recursive by (i) & (ii), since it satisfies

$$\begin{cases} f(x,0) = x \\ f(x,y+1) = \text{ifzero}(\text{mod}_2(f(x,y)), \text{half}(f(x,y)), 0) \end{cases}.$$

Combining (ii), (iii), (iv), and the fact (cf. page 98) that bounded summations preserve the property of primitive recursiveness, we have that hd and tl are in PRIM because ...

$$hd(x) = \begin{cases} \text{largest } y \text{ such that } 2^y \text{ divides } x & , \text{if } x>0 \\ 0 & , \text{if } x=0 \end{cases}$$

$$= \sum_{y<x} \mathrm{ifzero}(f(x,y+1),0,1)$$

and

$$tl(x) = \mathrm{half}(f(x, hd(x)))$$

$\square$ Proposition

Proof of Lemma 1

This follows immediately from the Proposition, because

$$lab = hd$$

and $$val_j = hd \circ (\underbrace{tl \circ \cdots \circ tl}_{j+1}).$$

$\square$ Lemma 1

Proof of Lemma 2

By examining M's program, we can define four $(I+1)$-tuples of numbers

$$(a_0,\ldots,a_I),\ (b_0,\ldots,b_I),\ (c_0,\ldots,c_I),\ \text{and}\ (d_0,\ldots,d_I)$$

as follows:

- for each $i = 0,\ldots,I-1$
  if i^{th} instruction is an increment, say $L_i : R_j^+ \to L_k$,
  then define $a_i = j$, $b_i = 0$, $c_i = k$, and $d_i = k$,
  else the instruction is a decrement, say $L_i : R_j^- \to L_k, L_\ell$,
  and define $a_i = j$, $b_i = 1$, $c_i = k$, and $d_i = \ell$
- define $a_I = 0$, $b_I = 0$, $c_i = I$, and $d_i = I$.

Summary:

If i^{th} instruction is,	then $(a_i, b_i, c_i, d_i) \overset{def}{=}$
$L_i : R_j^+ \to L_k$	$(j, 0, k, k)$
$L_i : R_j^- \to L_k, L_\ell$	$(j, 1, k, \ell)$
L_I : HALT	$(0, 0, I, I)$

$$next_\ell(x) \overset{def}{=} \sum_{i=0}^{I} ifzero(val_{a_i}(x), d_i, c_i) . eq(i, lab(x))$$

$$next_j(x) \overset{def}{=} \sum_{i=0}^{I-1} f_{ij}(x) . eq(i, lab(x)) + val_j(x) . eq(I, lab(x))$$

where

$$f_{ij}(x) \overset{def}{=} \Big(b_i (val_j(x) \dot{-} 1) + (1 - b_i)(val_j(x) + 1) \Big) . eq(a_i, j) + val_j(x) . (1 - eq(a_i, j))$$

Note that the equality test function $eq(x, y) \overset{def}{=} \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases}$

is primitive recursive, since $eq(x,y) = ifzero(x \dot{-} y, ifzero(y \dot{-} x, 1, 0), 0)$.
Using it, we can define primitive recursive functions $next_\ell$ and $next_j$ as follows:

$$next_\ell(x) \overset{def}{=} \sum_{i=0}^{I} ifzero(val_{a_i}(x), d_i, c_i) . eq(i, lab(x))$$

$$next_j(x) \overset{def}{=} \sum_{i=0}^{I-1} f_{ij}(x) . eq(i, lab(x)) + val_j(x) . eq(I, lab(x))$$

where

$$f_{ij}(x) \overset{def}{=} \Big(b_i . pred(val_j(x)) + (1 - b_i) . suc(val_j(x)) \Big) . eq(a_i, j) + val_j(x) . (1 - eq(a_i, j))$$

(and recall that $suc(x) = x + 1$, $pred(x) = x \dot{-} 1$).

By choice of the constants a_i, b_i, c_i, d_i $(i = 0, \ldots, I)$, it follows that given a state $[\ell, r_0, \ldots, r_m]$ of M,

$next_\ell([\ell, r_0, \ldots, r_m])$ = number of the instruction in the next state

$next_j([\ell, r_0, \ldots, r_m])$ = contents of R_j in the next state

(provided $0 \leq j \leq m$).

Therefore the next-state function is given by

$$next(x) = [\, nextl(x), next_0(x), \ldots, next_m(x) \,]$$

i.e.

$$next = mklist^{m+2} \circ (\, nextl, next_0, \ldots, next_m)$$

and hence it is primitive recursive since $nextl$, $next_i$, and (by the Proposition) $mklist^{m+2}$ are all in PRIM.

$\square$ Lemma 2