

Notes on the Primitive Recursive Functions

Define the **initial primitive recursive functions**:

- ζ , the zero function of arity 0, $\zeta() = 0$
- η , the identity function of arity 1, $\eta(n) = n$; and,
- σ , the successor function of arity 1, $\sigma(n) = n + 1$.

Define the **primitive recursive operations**:

- **Composition**: if f is a primitive recursive function of arity a , and g_1, \dots, g_a are primitive recursive functions of arities r_1, \dots, r_a , and $k \in \mathbf{N}$, then the following is a primitive recursive function of arity k :

$$h(x_1, \dots, x_k) = f(g_1(w_1), \dots, g_a(w_a)),$$

where each w_i is a list of r_i arguments, perhaps with repetition, from x_1, \dots, x_k ; and,

- **Primitive recursion**: if f and g are primitive recursive functions of arity k and $k + 2$, respectively, then there is a primitive recursive function, h , of arity $k+1$ satisfying the following conditions:

$$\begin{aligned} h(0, x_1, \dots, x_k) &= f(x_1, \dots, x_k); \quad \text{and,} \\ h(n + 1, x_1, \dots, x_k) &= g(h(n, x_1, \dots, x_k), n, x_1, \dots, x_k). \end{aligned}$$

Here composition is the natural way to combine functions, and primitive recursion is a restricted kind of recursion in which h with first argument $n + 1$ is defined in terms of h with first argument n , and all the other arguments unchanged.

Define the **primitive recursive functions** to be the smallest class of functions that contains the Initial functions and is closed under Composition and Primitive Recursion. The set of primitive recursive functions is equal to the set of functions computed using bounded iteration (Meyer & Ritchie 1967), i.e. the set of functions definable in the language Bloop from (Hofstadter 1979). Let **PrimRecFns** be the set of primitive recursive functions and let **Primitive Recursive** be the set of primitive recursive sets, i.e., those sets whose characteristic function is primitive recursive:

$$\mathbf{Primitive\ Recursive} = \{S \subseteq \mathbf{N} \mid \chi_S \in \mathbf{PrimRecFns}\}$$

Proposition 1 *The following functions are members of PrimRecFns:*

1. $M_1(x) = \mathbf{if}(x > 0) \mathbf{then}(x - 1) \mathbf{else} 0$
2. $x \ominus y = \mathbf{if}(y \leq x) \mathbf{then}(x - y) \mathbf{else} 0$
3. $+$
4. $*$

$$5. \exp(x, y) = y^x$$

$$6. \text{hyper-exp}(x) = \mathbf{if} (x = 0) \mathbf{then} 1 \mathbf{else} 2^{\text{hyper-exp}(x-1)}$$

Proof:

1. Let $\pi_2^2(x, y) = \eta(\eta(y))$. Define M_1 by primitive recursion:

$$\begin{aligned} M_1(0) &= \zeta() \\ M_1(n+1) &= n = \pi_2^2(M_1(n), n) = \eta(\eta(n)) \end{aligned}$$

2. Define $f_b(x, y) = y \ominus x$ by primitive recursion:

$$\begin{aligned} f_b(0, y) &= y = \pi_1^1(y) \\ f_b(n+1, y) &= M_1(f_b(n, y)) \end{aligned}$$

3.

$$\begin{aligned} 0 + y &= y \\ (n+1) + y &= \sigma(n+y) \end{aligned}$$

4.

$$\begin{aligned} 0 \star y &= 0 \\ (n+1) \star y &= (n \star y) + y \end{aligned}$$

5.

$$\begin{aligned} y^0 &= 1 \\ y^{n+1} &= y^n \star y \end{aligned}$$

6.

$$\begin{aligned} \text{hyper-exp}(0) &= 1 \\ \text{hyper-exp}(n+1) &= 2^{\text{hyper-exp}(n)} \end{aligned}$$

□

Observation: All primitive recursive functions are total, recursive functions.

Def: A **predicate** is a boolean valued function.

Proposition 2 **PrimRecFcns** is closed under definition by cases:

if $h_0, h_1, Q \in \mathbf{PrimRecFcns}$, Q a predicate **then** $f \in \mathbf{PrimRecFcns}$:

$$f(\bar{x}) = \mathbf{if} (Q(\bar{x})) \mathbf{then} h_1(\bar{x}) \mathbf{else} h_0(\bar{x})$$

Proof: $f(\bar{x}) = h_1(\bar{x}) \cdot Q(\bar{x}) + h_0(\bar{x}) \cdot (1 \ominus Q(\bar{x}))$

□

Proposition 3 **PrimRecPreds** is closed under boolean operations.

Proof:

$$\begin{aligned}\neg Q(\bar{x}) &= ((1 \ominus Q(\bar{x}))) \\ Q(\bar{x}) \wedge S(\bar{x}) &= Q(\bar{x}) \cdot S(\bar{x})\end{aligned}$$

□

Prop: The following predicates are **PrimRecPreds**: $==, \leq, <, >, \neq$.

Proof:

$$\begin{aligned}x \leq y &\equiv (1 \ominus (x \ominus y)) \\ x == y &\equiv x \leq y \wedge y \leq x \\ x \neq y &\equiv \neg(x == y) \\ x < y &\equiv x \leq y \wedge x \neq y \\ x > y &\equiv y < x\end{aligned}$$

□

Prop: If $g \in \mathbf{PrimRecFcns}$ then so are:

$$\mathbf{gSum}(n, \bar{y}) = \sum_{i=0}^n g(i, \bar{y}); \quad \mathbf{gProd}(n, \bar{y}) = \prod_{i=0}^n g(i, \bar{y})$$

Proof:

Define \mathbf{gSum} by primitive recursion:

$$\begin{aligned}\mathbf{gSum}(0, \bar{y}) &= g(0, \bar{y}) \\ \mathbf{gSum}(n+1, \bar{y}) &= g(n+1, \bar{y}) + \mathbf{gSum}(n, \bar{y})\end{aligned}$$

\mathbf{gProd} is similar

□

Proposition 4 **PrimRecPreds** is closed under bounded quantification. That is, if $Q \in \mathbf{PrimRecFcns}$ so are:

$$\exists i \leq n(Q(i, \bar{y})); \quad \forall i \leq n(Q(i, \bar{y}))$$

Proof:

$$\forall i \leq n(Q(i, \bar{y})) = \prod_{i=0}^n Q(i, \bar{y})$$

$$\exists i \leq n(Q(i, \bar{y})) = \neg \forall i \leq n(\neg Q(i, \bar{y}))$$

□

Proposition 5 **PrimRecFcns** is closed under bounded minimization. That is, if $Q \in \mathbf{PrimRecPreds}$ then the following is in **PrimRecFcns**.

$$Qmin(x, \bar{y}) = \begin{cases} \text{least } t \leq x & \text{such that } Q(t, \bar{y}) \\ x + 1 & \text{if there is none} \end{cases}$$

Proof: Define $Qmin$ by primitive recursion

Proposition 6 $P, L, R \in \mathbf{PrimRecFcns}$

Proof: Exercise

□

Proposition 7 $\text{Prime}, \text{PF} \in \mathbf{PrimRecFcns}$, where,

$\text{Prime}(x) = \mathbf{if}$ (“ x is prime”) **then** 1 **else** 0

$\text{PF}(n) =$ prime number n , i.e.,

$\text{PF}(0) = 2,$

$\text{PF}(1) = 3,$

$\text{PF}(2) = 5,$

$\text{PF}(3) = 7,$

$\text{PF}(4) = 11,$

$\text{PF}(5) = 13,$

$\text{PF}(6) = 17,$

...

Proof:

$$x|y = \exists z \leq y(x \cdot z == y)$$

$$\text{Prime}(x) \equiv x > 1 \wedge \forall y < x(y|x \rightarrow y == 1)$$

$$\text{NextPrime}(x) \equiv \min t \leq (x + 1)^{x+1} (t > x \wedge \text{Prime}(t))$$

def PF(x)

$a = 2$

x .times do

$a = \text{NextPrime}(a)$

 end

 return(a)

end

□

[Actually $2x$ suffices instead of $(x + 1)^{x+1}$ above.]

Define

$$\text{Seq}(a_0, a_1, \dots, a_n) \equiv 2^{a_0+1} 3^{a_1+1} \dots \text{PF}(n)^{a_n+1}$$

Sequence numbers let us uniquely encode an **arbitrary finite sequence of natural numbers** as a single natural number:

Proposition 8 *The following functions are in PrimRecFns:*

$$\text{IsSeq}(S) \equiv \text{“}S \text{ is a Sequence number”}$$

$$\text{length}(\text{Seq}(a_0, a_1, \dots, a_n)) \equiv n + 1$$

$$\text{Item}(\text{Seq}(a_0, a_1, \dots, a_n), i) \equiv a_i$$

Proof:

$$\text{Good}(x, S) \equiv \forall y < S((y < x \wedge \text{PF}(y)|S) \vee (y \geq x \wedge \text{PF}(y) \nmid S))$$

$$\text{IsSeq}(S) \equiv \exists x < S(\text{Good}(x, S))$$

$$\text{length}(S) \equiv \min x < S(\text{Good}(x, S))$$

$$\text{Item}(S, i) \equiv \min y < S(\text{IsSeq}(S) \wedge \text{PF}(i)^{y+1}|S \\ \wedge \text{PF}(i)^{y+2} \nmid S)$$

□

Kleene’s COMP Theorem: Let $\text{COMP}(n, x, c, y)$ mean – with appropriate coding – that c is a valid halting computation of Turing machine M_n on input x and its output is y . COMP is a Primitive Recursive predicate.

Corollary: [Normal form for Partial Recursive Functions]

For all n , the n th partial recursive function, $M_n(\cdot)$ can be written as follows:

$$M_n(x) = R(\min z : \text{COMP}(n, x, L(z), R(z)))$$

Perhaps not that interesting details of the proof of Kleene’s COMP Theorem: Our encoding will make c a sequence number, $c = \text{Seq}(\text{ID}_0, \text{ID}_1, \dots, \text{ID}_t)$. Each ID_i is a sequence number of symbols,

$$\text{ID}_i = \text{Seq}(\triangleright, a_1, \dots, a_{i-1}, [\sigma, a_i], a_{i+1}, \dots, a_r),$$

where — as in our definition of VALCOMP in class — this represents an instantaneous description (ID) of a Turing machine in state σ looking at symbol a_i . Notice that we make the symbol in square i $[\sigma, a_i]$ indicating that M_n is in state σ and its head is examining square i which contains the symbol a_i .¹

A simple way to encode the Turing machine M_n , is with n a sequence number as follows:

$$n = \text{Seq}(\langle s, q \rangle, I_1, I_2, \dots, I_q)$$

Here $\text{Item}(n, 0) = \langle s, q \rangle$ means that s is the number of tape symbols and q is the number of states. Let $\Sigma = \{\alpha_0, \dots, \alpha_{s-1}\}$ and let $Q = \{\sigma_0, \dots, \sigma_q\}$. We will assume that $\sigma_0 = h$ is the halting state, and σ_1 is the start state. Similarly, we may assume that $\alpha_0 = 0, \alpha_1 = 1, \alpha_2 = \sqcup, \alpha_3 = \triangleright$. Each I_j is the list of instructions for state q_j :

$$I_j = \text{Seq}(I_{j,0}, \dots, I_{j,s-1})$$

where $I_{j,k} = \langle j', k', d \rangle$ is the triple² indicating that in state σ_j looking at symbol α_k , M_n goes into state $\sigma_{j'}$, writes symbol $\alpha_{k'}$, and moves in direction d , where 0 indicates left, 1 indicates stay where you are, and 2 indicates right. The state σ_0 will always be “ h ”, and “ σ_1 ” will be the start state.

We will encode the symbols of an ID via the number $i, 0 \leq i < s$ for symbol α_i . The pair $[\sigma_j, \alpha_i]$ will be encoded by the number $s + \langle j, i \rangle$.

To express $\text{NEXT}(n, \text{ID}, \text{ID}')$, we need to do the following:

1. Find the following values: s, q , the number of tape symbols and states of M_n ; i, i', j, j', k, k' , the positions of the head, the states, and the symbols scanned, in ID, ID' ; and $I = I_{j,k} = \langle j', w, d \rangle$, M_n ’s instruction on reading symbol α_k in state σ_j , indicating that it should change to state $\sigma_{j'}$, write symbol α_w , and move in direction d .
2. Assert that the above values are consistent with ID, ID' .
3. Assert that ID' follows from ID by executing instruction I .

The following formula expresses $\text{NEXT}(n, \text{ID}, \text{ID}')$. Note that it describes a Primitive-Recursive predicate by the fact that **Primitive Recursive** is closed under bounded quantification and boolean operations.

$$\begin{aligned} \text{NEXT} \equiv & (\exists q, s, I < n)(\exists i < \text{length}(\text{ID}))(\exists i' < \text{length}(\text{ID}'))(\exists j, j' \leq q)(\exists k, k', w < s)(\exists d \leq 2) \\ & \left[\text{Item}(n, 0) = \langle s, q \rangle \wedge \text{Item}(\text{ID}, i) = s + \langle j, k \rangle \wedge \text{Item}(\text{ID}', i') = s + \langle j', k' \rangle \wedge \right. \\ & \left. j > 0 \wedge I = \text{Item}(\text{Item}(n, j), k) \wedge \text{Follows}(I, \text{ID}, \text{ID}') \right] \end{aligned}$$

The nitty-gritty definition of $\text{Follows}(I, \text{ID}, \text{ID}')$ is just a case analysis, including the fact that if we try to move right, off the current string, then we add a $\sqcup = \sigma_2$ to the end:

¹The natural number that encodes $[\sigma, a_i]$ will be $s + \langle \sigma, a_i \rangle$, where $s = |\Sigma|$ is the number of tape symbols of M_n .

²The triple, $\langle a, b, c \rangle$, is equal to $\langle a, \langle b, c \rangle \rangle$.

$$\begin{aligned}
\text{Follows} &\equiv I = \langle j', \langle w, d \rangle \rangle \wedge \\
&d = 0 \rightarrow (i' + 1 = i \wedge k' = \text{Item}(\text{ID}, i') \wedge w = \text{Item}(\text{ID}', i)) \wedge \\
&d = 1 \rightarrow (i' = i \wedge k' = w) \wedge \\
&d = 2 \rightarrow (i' = i + 1 \wedge w = \text{Item}(\text{ID}', i) \wedge \\
&\quad (k' = \text{Item}(\text{ID}, i') \vee (i' = \text{length}(\text{ID}) \wedge k' = 2))) \wedge \\
&(\forall \ell < \text{length}(\text{ID}))(\ell = i \vee \ell = i' \vee \text{Item}(\text{ID}, \ell) = \text{Item}(\text{ID}', \ell)) \wedge \text{Item}(\text{ID}', i) \wedge \\
&\text{length}(\text{ID}) = \text{length}(\text{ID}') \vee (d = 2 \wedge i + 1 = \text{length}(\text{ID}) \wedge i + 2 = \text{length}(\text{ID}'))
\end{aligned}$$

Now that we have written NEXT, the remainder of COMP is not difficult:

$$\begin{aligned}
\text{COMP} &\equiv \text{IsSeq}(c) \wedge \text{START}(\text{Item}(c, 0), x) \wedge \text{END}(\text{Item}(c, \text{length}(c) - 1)) \wedge \\
&(\forall i < \text{length}(c))\text{NEXT}(n, \text{Item}(c, i), \text{Item}(c, i + 1))
\end{aligned}$$

Here, $\text{START}(\text{ID}, x)$ says that ID is a correct starting instantaneous description of M_n on input x . Similarly, $\text{END}(\text{ID}, y)$, says that ID is a halting ID , with output y . These are written as follows:

$$\text{START}(\text{ID}, x) \equiv \text{Value}(\text{ID}, 1, x); \quad \text{END}(\text{ID}, y) \equiv \text{Value}(\text{ID}, 0, y),$$

where $\text{Value}(\text{ID}, j, x)$ means that ID consists of $([j, 3], x_1, \dots, x_n)$, i.e., state σ_j looking at symbol $\alpha_3 = \triangleright$, with the binary expansion of x on the rest of the tape,

$$\begin{aligned}
\text{Value}(\text{ID}, j, x) &\equiv \text{Item}(\text{ID}, 0) = s + \langle j, 3 \rangle \wedge (\exists \ell < x)(2 \uparrow \ell \leq x < 2 \uparrow (\ell + 1) \wedge \\
&(\forall i \leq \ell)(\exists y, z \leq x)(\exists d \leq 1)(\text{Item}(\text{ID}, i + 1) = d \wedge \\
&\quad x = y + (d + 2z)(2 \uparrow (\ell - j)) \quad \wedge \quad y < 2 \uparrow (\ell - j)
\end{aligned}$$

□