Notes on the Primitive Recursive Functions

First please read Section 3 of my article, “Computability and Complexity” in the Stanford Encyclopedia of Philosophy. This defines the primitive recursive functions and gives some of their properties.

After that we continue the development as follows:

Let \textbf{PrimRecFcs} be the set of primitive recursive functions and let \textbf{Primitive Recursive} be the set of primitive recursive sets, i.e., those sets whose characteristic function is primitive recursive:

\[
\text{Primitive Recursive} = \{ S \subseteq \mathbb{N} \mid \chi_S \in \text{PrimRecFcs} \}
\]

**Proposition 1** The following functions are members of \textbf{PrimRecFcs}:

1. \( M_1(x) = \text{if } (x > 0) \text{ then } (x - 1) \text{ else } 0 \)
2. \( x \ominus y = \text{if } (y \leq x) \text{ then } (x - y) \text{ else } 0 \)
3. \( + \)
4. \( * \)
5. \( \exp(x, y) = y^x \)
6. \( \text{hyper-exp}(x) = \text{if } (x = 0) \text{ then } 1 \text{ else } 2^{\text{hyper-exp}(x - 1)} \)

**Proof:**

1. Let \( \pi^2_2(x, y) = \eta(\eta(y)) \). Define \( M_1 \) by primitive recursion:

\[
M_1(0) = \zeta()
M_1(n + 1) = n = \pi^2_2(M_1(n), n) = \eta(\eta(n))
\]

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

\[
f_b(0, y) = y = \pi^1_1(y)
\]
\[
f_b(n + 1, y) = M_1(f_b(n, y))
\]

3. \( 0 + y = y \)
\( (n + 1) + y = \sigma(n + y) \)

4. \( 0 \times y = 0 \)
\( (n + 1) \times y = (n \times y) + y \)
5.

\begin{align*}
y^0 &= 1 \\
y^{n+1} &= y^n \cdot y
\end{align*}

6.

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

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

Def: A predicate is a boolean valued function.

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

if \( h_0, h_1, Q \in \text{PrimRecFns}, Q \) a predicate then \( f \in \text{PrimRecFns} \):

\[
f(\vec{x}) = \text{if } (Q(\vec{x})) \text{ then } h_1(\vec{x}) \text{ else } h_0(\vec{x})
\]

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

Proposition 3 PrimRecPreds is closed under boolean operations.

Proof:

\[
\begin{align*}
\neg Q(\vec{x}) &= ((1 \ominus Q(\vec{x})) \\
Q(\vec{x}) \land S(\vec{x}) &= Q(\vec{x}) \cdot S(\vec{x})
\end{align*}
\]

Prop: The following predicates are PrimRecPreds: \(==, \leq, <, >, \neq\).

Proof:

\[
\begin{align*}
x \leq y &\equiv (1 \ominus (x \ominus y)) \\
x == y &\equiv x \leq y \land y \leq x \\
x \neq y &\equiv \neg (x == y) \\
x < y &\equiv x \leq y \land x \neq y \\
x > y &\equiv y < x
\end{align*}
\]
Prop: If $g \in \text{PrimRecFcns}$ then so are:

$$g\text{Sum}(n, \bar{y}) = \sum_{i=0}^{n} g(i, \bar{y}); \quad g\text{Prod}(n, \bar{y}) = \prod_{i=0}^{n} g(i, \bar{y})$$

Proof:
Define $g\text{Sum}$ by primitive recursion:

$$g\text{Sum}(0, \bar{y}) = g(0, \bar{y})$$
$$g\text{Sum}(n + 1, \bar{y}) = g(n + 1, \bar{y}) + g\text{Sum}(n, \bar{y})$$

$g\text{Prod}$ is similar

□

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

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

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 $Q \in \text{PrimRecPreds}$ then the following is in PrimRecFcns.

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

Proof: Define $Q\text{min}$ by primitive recursion

□

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

Proof: Exercise

□

Proposition 7 Prime, PF $\in \text{PrimRecFcns}$, where,
Prime($x$) = if (“$x$ is prime”) then 1 else 0
PF($n$) = prime number $n$, i.e,
PF(0) = 2,
PF(1) = 3,
PF(2) = 5,
PF(3) = 7,
PF(4) = 11,
PF(5) = 13,
PF(6) = 17,
...

Proof:

\[
x|y = \exists z \leq y (x \cdot z = y)
\]
Prime($x$) \equiv x > 1 \land \forall y < x (y|x \rightarrow y = 1)
NextPrime($x$) \equiv \min t \leq (x + 1)^{x+1} (t > x \land \text{Prime}(t))

def PF($x$)
\[
a = 2
x\text{.times do}
\[
a = \text{NextPrime}(a)
\]
end
return($a$)
end

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

Let
\[
\text{Seq}(a_0, a_1, \ldots, a_n) \equiv 2^{a_0+1}3^{a_1+1} \ldots \text{PF}(n)^{a_n+1}
\]

Proposition 8 The following functions are in PrimRecFns:

IsSeq($S$) \equiv “$S$ is a Sequence number”
\[
\text{length(Seq}(a_0, a_1, \ldots, a_n)) \equiv n + 1
\]
Item(Seq($a_0, a_1, \ldots, a_n), i) \equiv a_i
Proof:

Good\( (x, S) \equiv \forall y < S((y < x \land PF(y)|S) \lor (y \geq x \land PF(y) \not|S)) \)

IsSeq\( (S) \equiv \exists x < S(Good(x, S)) \)

length\( (S) \equiv \min x < S(Good(x, S)) \)

Item\( (S, i) \equiv \min y < S(IsSeq(S) \land PF(i)^y+1|S \land PF(i)^y+2 \not|S) \)

\(\square\)

With sequence numbers, we can think of any natural number as an arbitrary finite sequence of natural numbers.

**Kleene’s COMP Theorem:** Let 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.

**Proof:** Our encoding will make \( c \) a sequence number, \( c = \text{Seq}(\text{ID}_0, \text{ID}_1, \ldots, \text{ID}_t) \). Each \( \text{ID}_i \) is a sequence number of symbols,

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

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

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

\[
n = \text{Seq}(\langle s, q \rangle, I_1, I_2, \ldots, I_q)
\]

Here Item\( (n, 0) = \langle s, q \rangle \) means that \( s \) is the number of tape symbols and \( q \) is the number of states. Let \( \Sigma = \{\alpha_0, \ldots, \alpha_{s-1}\} \) and let \( Q = \{\sigma_0, \ldots, \sigma_q\} \). We will assume that \( \sigma_0 = h \) is the halting state, and \( \sigma_1 \) 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 \( q_j \):

\[
I_j = \text{Seq}(I_{j,0}, \ldots, I_{j,s-1})
\]

where \( I_{j,k} = \langle j', k', d \rangle \) is the triple\(^1\) indicating that in state \( \sigma_j \) looking at symbol \( \alpha_k \), \( M_n \) 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 \( \sigma_0 \) will always be “\( h \)”, and “\( \sigma_1 \)” will be the start state.

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

To express 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 \( \text{ID}, \text{ID}' \); and \( I = I_{j,k} = \langle j', w, d \rangle \), \( M_n \)'s instruction on reading symbol \( \alpha_k \) in state \( \sigma_j \), indicating that it should change to state \( \sigma_j' \), write symbol \( \alpha_w \), and move in direction \( d \).

2. Assert that the above values are consistent with \( \text{ID}, \text{ID}' \).

\(^1\)The 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 \( M_n \).

\(^2\)The triple, \( \langle a, b, c \rangle \), is equal to \( \langle a, \langle b, c \rangle \rangle \).
3. Assert that ID' follows from ID by executing instruction I.

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

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

The nitty-gritty definition of Follows(I, ID, ID') is just a case analysis, including the fact that if we try to move right, off the current string, then we add a □ = σ₂ to the end:

\[
\text{Follows} \equiv I = \langle j', \langle w, d \rangle \rangle \land \\
    d = 0 \rightarrow (i' + 1 = i \land k' = \text{Item}(ID, i') \land w = \text{Item}(ID', i)) \land \\
    d = 1 \rightarrow (i' = i \land k' = w) \land \\
    d = 2 \rightarrow (i' = i + 1 \land w = \text{Item}(ID', i) \land \\
    (k' = \text{Item}(ID, i') \lor (i' = \text{length}(ID) \land k' = 2))) \land \\
    (\forall \ell < \text{length}(ID))(\ell = i \lor \ell = i' \lor \text{Item}(ID, \ell) = \text{Item}(ID', \ell)) \land \text{Item}(ID', i) \land \\
    \text{length}(ID) = \text{length}(ID') \lor (d = 2 \land i + 1 = \text{length}(ID) \land i + 2 = \text{length}(ID'))
\]

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

\[
\text{COMP} \equiv \text{IsSeq}(c) \land \text{START}(\text{Item}(c, 0), x) \land \text{END}(\text{Item}(c, \text{length}(c) - 1)) \land \\
    (\forall i < \text{length}(c))\text{NEXT}(n, \text{Item}(c, i), \text{Item}(c, i + 1))
\]

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

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

where Value(ID, j, x) means that ID consists of \((j, 3], x_1, \ldots, x_n\), i.e., state \(σ_j\) looking at symbol \(α_3 = \triangleright\), with the binary expansion of \(x\) on the rest of the tape,

\[
\text{Value}(ID, j, x) \equiv \text{Item}(ID, 0) = s + \langle j, 3 \rangle \land (\exists \ell < x)(2 \uparrow \ell \leq x < 2 \uparrow (\ell + 1) \land \\
    (\forall i \leq \ell)(\exists y, z \leq x)(\exists d \leq 1)(\text{Item}(ID, i + 1) = d \land \\
    x = y + (d + 2z)(2 \uparrow (\ell - j)) \land y < 2 \uparrow (\ell - j)) \land \\
    y < 2 \uparrow (\ell - j)
\]

□