Coding Turing Machines

Key functions coding tape behavior:

$$\begin{aligned}
\text{strt}(x) &= 2^{x+1} \dot{-} 1 \\
\text{scan}(r) &= \text{rem}(2, r) \\
\text{newleft}_0(l, r) &= l \\
\text{newrght}_0(l, r) &= r \dot{-} \text{scan}(r) \\
\text{newleft}_1(l, r) &= l \\
\text{newrght}_1(l, r) &= r + 1 \dot{-} \text{scan}(r) \\
\text{newleft}_L(l, r) &= \text{quo}(l, 2) \\
\text{newrght}_L(l, r) &= 2r + \text{rem}(l, 2) \\
\text{newleft}_R(l, r) &= 2l + \text{rem}(r, 2) \\
\text{newrght}_R(l, r) &= \text{quo}(r, 2)
\end{aligned}$$

If we code B as 0, 1 as 1, L as 2, and R as 3, then we have:

$$\text{newleft}(l, r, a) = \begin{cases} l & \text{, if } a = 0, 1 \\ \text{quo}(l, 2) & \text{, if } a = 2 \\ 2l + \text{rem}(r, 2) & \text{, if } a = 3 \end{cases}$$

$$\text{newrght}(l, r, a) = \begin{cases} r \dot{-} \text{scan}(r) & \text{, if } a = 0 \\ r + 1 \dot{-} \text{scan}(r) & \text{, if } a = 1 \\ 2r + \text{rem}(l, 2) & \text{, if } a = 2 \\ \text{quo}(r, 2) & \text{, if } a = 3 \end{cases}$$

These are p.r. because of the cases stuff.

The value represented by the right number in a halted standard configuration:

$$\text{valu}(r) = \text{lg}(r, 2)$$

And we are in a standard halting position iff:

$$l = 0 \wedge r = 2^{\text{lg}(r,2)+1} \dot{-} 1$$

Let std(l, r) be the characteristic function of the relation just exhibited.

We code the machine itself as a sequence of quadruples. We code the *halted* state as 0. So our adder (which no longer calculates addition) was:

$$S_1BRS_2, S_11BS_1, S_2B1S_3, S_21RS_2, S_3BBS_0, S_31LS_3$$

Note the fixed order in which these are written: Increasing by state, with the instruction for B before that for 1. So we may simplify to:

$$RS_2, BS_1, 1S_3, RS_2, BS_0, LS_3$$

This is then coded as the sequence:

$$3, 2, 0, 1, 1, 3, 3, 2, 0, 4, 2, 3$$

which is itself coded as:

$$2^{12}3^3 5^2 7^0 11^1 13^1 17^3 19^3 23^2 29^0 31^4 37^2 41^3$$

This fairly large number is thus the code of our adder.

We now need to see how to "read" the instuctions for a machine from its code number. The *action* to perform when in state q scanning the symbol i is found in the $4(q \dot{-} 1) + 2i^{\text{th}}$ position; the state to go into is found in the next posiiton. So:

$$\begin{aligned} \text{actn}(m, q, r) &= \text{ent}(m, 4(q \dot{-} 1) + 2 \times \text{scan}(r)) \\ \text{newstat}(m, q, r) &= \text{ent}(m, 4(q \dot{-} 1) + 2 \times \text{scan}(r) + 1) \end{aligned}$$

A configuration of a machine—left number, state, and right number—can be coded by a triple, in the obvious way:

$$\text{config}(l, q, r) = 2^l 3^q 5^r$$

So we can easily recover the elements of the configuration from it:

$$\begin{aligned} \text{left}(c) &= \text{lo}(c, 2) \\ \text{stat}(c) &= \text{lo}(c, 3) \\ \text{rght}(c) &= \text{lo}(c, 5) \end{aligned}$$

We now code the sequence of configurations of the machine:

$$\begin{aligned} \text{init}(m, x) &= \text{config}(0, 1, \text{strt}(x)) \\ \text{nextleft}(m, c) &= \text{newleft}[\text{left}(c), \text{rght}(c), \text{actn}(m, \text{stat}(c), \text{rght}(c))] \\ \text{nextrght}(m, c) &= \text{newrght}[\text{left}(c), \text{rght}(c), \text{actn}(m, \text{stat}(c), \text{rght}(c))] \\ \text{nextstat}(m, c) &= \text{newstat}[m, \text{stat}(c), \text{rght}(c)] \\ \text{newconf}(m, c) &= \text{config}(\text{nextleft}(m, c), \text{nextstat}(m, c), \text{nextrght}(m, c)] \end{aligned}$$

The 'evolution' of the machine's configutation is then described by:

$$\begin{aligned} \text{conf}(m, x, 0) &= \text{init}(m, x) \\ \text{conf}(m, x, t') &= \text{newconf}(m, x, \text{conf}(m, x, t)) \end{aligned}$$

We are halted at step t if we are in state 0 and so we are halted in standard position if:

$$\text{halted}(m, x, t) \equiv \text{stat}(\text{conf}(m, x, t)) = 0 \wedge \text{std}[\text{left}(\text{conf}(m, x, t)), \text{rght}(\text{conf}(m, x, t))]$$

Let stdh(m, x, t) be the characteristic function of halted(m, x, t). Then, if we *are* halted in standard position, the output of the machine is given by:

$$\text{out}(m, x, t) = \text{valu}(\text{rght}(\text{conf}(m, x, t)))$$

To this point, everything has been primitive recursive.

The step at which the machine halts in standard configuration, if any, is given by:

$$\mathbf{halt}(m, x) = \mu t[\mathbf{stdh}(m, x, t)]$$

This function is recursive, but *not* primitive recursive, since it uses *unbounded* minimization.

So, if we set:

$$F(m, x) = \mathbf{out}(m, x, \mathbf{halt}(m, x))$$

then F is a (partial) recursive function, and its value, when it is defined, is the output of the machine with index m when started on input x; it is undefined if that machine does not halt on that input. That is, if $f(x)$ is the function calculated by the machine with index m, then:

$$f(x) = F(m, x)$$

so $f(x)$ is recursive.