Phil 320

Chapter 8: Turing computable functions are recursive

Restrict attention to functions f(x) of one argument.

Suppose f is computed by M. We want to show that f is a recursive function. We do this for all Turing machines at once, using a code number *m* for each Turing machine.

1. Wang coding: contents of tape plus position of cursor

a) Left number (*p*) and right number (*r*).

Example: BBB11B111B1B11BBBB
↑

The *left number*, *p*, is the binary number to the left of the pointer, treating all blanks as 0. In this case, 11011, or 27.
The *right number*, r, is the binary number written backwards from right to left, ending with the scanned square, treating all blanks as 0. In this case, 110101, or 53.

b) Effect of actions by machine. Let p' and r' stand for the new left and right numbers.

Action		*p* odd	*p* even	General
$a = 2$	M moves left	$p' = (p-1)/2$ $r' = 2r + 1$	$p' = p/2$ $r' = 2r$	$p' = \text{quo}(p, 2)$ $r' = 2r + \text{rem}(p, 2)$
		r odd	r even	
$a = 3$	M moves right	$p' = 2p + 1$ $r' = (r-1)/2$	$p' = 2p$ $r' = r/2$	$p' = 2p + \text{rem}(r, 2)$ $r' = \text{quo}(r, 2)$
		scan(*r*) = 0 (r even)	scan(*r*) = 1 (r odd)	
$a = 0$	M prints 0	$p' = p$ $r' = r$	$p' = p$ $r' = r \dot{-} 1$	$p' = p$ $r' = r \dot{-} \text{scan}(r)$
$a = 1$	M prints 1	$p' = p$ $r' = r + 1$	$p' = p$ $r' = r$	$p' = p$ $r' = r + (1 \dot{-} \text{scan}(r))$

Using the above table, $p' = newleft(p, r, a)$ and $r' = newright(p, r, a)$ are p.r. functions (defined by cases).

c) Codes for initial and final positions.

Initial: Cursor points to leftmost of x+1 1's. So $p = 0$ and
$r = \text{start}(x) = 2^{(x+1)} \dot{-} 1$, which is p.r.
Final: If in standard position, cursor points to leftmost of f(x) + 1 1's. So $p = 0$ and $r = 2^{f(x)+1} \dot{-} 1$, which is p.r. Then $f(x) = \text{lg}(r, 2)$.
[This means: largest z such that $2^z \leq r$, and the largest such z is indeed $f(x)$.]

Further, we can define a p.r. function *nonstandard*(p, r) such that the machine is in standard position if and only if *nonstandard*(p, r) = 0.

2. Configuration

Define *triple*(p, q, r) = $2^p\ 3^q\ 5^r$. This single number gives a complete snapshot of the current progress of the Turing machine, since it encodes the left number, right number and current machine state. And given a number n of this form, we can recover p, q and r:

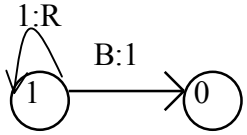
p = left(n) = lo(n, 2); q = state(n) = lo(n, 3); r = right(n) = lo(n, 5)

3. Codes for Turing machines

States: Code q_i as i, as in chapter 4; code the halted state as 0.
Symbols: Code B as 0, 1 as 1
Actions: Code B, 1, L, R as 0, 1, 2, 3 (respectively).

Each Turing machine is first coded as a finite sequence of length 4k, where k is number of states (excluding halted state), and then as a single number. [Basically, use the coding method described in chapter 4, p. 36.]

Example: 1:R



$1q_0, Rq_1$ $\Rightarrow 1,0,3,1$ $\Rightarrow 2^1 3^0 5^3 7^1$

We count the first *entry* in such a sequence as the 0^{th} entry. So each even entry stands for an act; each odd entry stands for the new machine state. To run the machine:
If in state q scanning symbol i, the next act to perform is given by the entry in position $4(q \dot{-} 1) + 2i$, and the new state is given by the entry in position $4(q \dot{-} 1) + 2i + 1$.

So: if m is a code number for a Turing machine, q is the current state and r is the current right number (so the currently scanned symbol is scan(r)), the next act and the new state are:

action(m, q, r) = *entry*(m, $4(q \dot{-} 1) + 2$ scan(r))

newstate(m, q, r) = *entry*(m, $4(q \dot{-} 1) + 2$scan(r) + 1)

4. Defining a p.r. function that tracks the Turing machine computation

We want to define a function that gives the complete current configuration of the machine: p, q and r. We will define a function

Conf(m, x, t)

where m codes the machine M that computes f, x is the argument and t is the stage in the computation of f(x). The function will be defined so that for each stage t up to and including the halting stage,

Conf(m, x, t) = triple(p, q, r),

where p, q, and r are the left number, state and right number at stage t.

If we can show that Conf is p.r., we'll be essentially done. For the state q (which is the 'middle number' encoded by Conf, i.e., the exponent of 3) is positive unless and until M halts, when it becomes 0. So we can find the value of f(x) by using Minimization: it is the value of lg(r, 2) at the least t where q = lo(Conf(m, x, t),3) = 0, provided M is in standard configuration when M halts.

The trick is to define Conf by primitive recursion:

Base clause: Conf(m, x, 0) = triple(0, 1, start(x)) [initially, p = 0, q = 1 and r = start(x)]

Recursion clause: We want to define Conf(m, x, t') in terms of m, x and Conf(m, x, t).

Let c = Conf(m, x, t). Then

i) p = left(c), q = state(c) and r = right(c) are p.r.

ii) a = action(m, q, r) and q^* = newstate(m, q, r) are p.r.

iii) p^* = newleft(p, r, a) and r^* = newright(p, r, a) are p.r.

iv) Conf(m, x, t') = triple(p^*, q^*, r^*) is p.r.

So by composition, Conf(m, x, t') is defined via a p.r. function from (m, x, Conf(m, x, t)). Hence Conf is p.r.

5. Finally: f is recursive.

If t is the stage at which M halts (no prescribed action), then M is in state 0 for the first time at t, and the centre value q of Conf(m, x, t) is 0 for the first time. But before we recover the value of $f(x)$ from the right number r of Conf(m, x, t), we need to determine whether or not M halted in standard position.

Here is a function that tells us precisely whether M has halted in standard position at stage t. This happens just in case [q = 0 and nonstandard(p, r) = 0]. Define

standardhalt(m, x, t) = state(Conf(m, x, t)) + nonstandard(p, r)
where p = left(Conf(m, x, t)) and r = right(Conf(m, x, t)).

Then standardhalt(m, x, t) is p.r., and is either never 0 [if the computation of f(x) does not halt in standard position], or 0 for the first t where the computation halts in standard position. Let

halt(m, x) = {least t such that standardhalt(m, x, t) = 0 if there is such a t
{undefined otherwise.

Then halt is recursive since standardhalt is p.r.

If the machine does halt in standard position at stage t, then its output is given by

value(Conf(m, x, t)) = lg(right(Conf(m, x, t)),2),

which is p.r.

So putting it all together:

$$F(m, x) = \text{value}(\text{Conf}(m, x, \text{halt}(m, x)))$$

is, first of all, recursive by composition. Second, $F(m, x)$ is exactly the value of the function computed by the Turing machine with code m applied to input x (where defined, and otherwise undefined). So for the particular m that codes f, we have

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

which shows that f is recursive and completes the argument.

Upshot:

$$T = A = R$$

where

T = Turing-computable functions
A = abacus-computable functions
R = recursive functions