Logic, Computability and Incompleteness

T = A = R

[optional]

Recursive Functions are Abacus Computable

- Will now adduce further evidence in support of the Church Turing Thesis by seeing that $R \subseteq A$.
- It's interesting and important to know these general results, but you won't be examined on the details of the mappings.
- Since we've already seen that $A \subseteq T$, it will follow that $R \subseteq T$. Finally will see that $T \subseteq R$ and hence that $T = A = R$.
- Need to adopt some conventions. Assume that for an n-place function f^n the arguments appear in registers $1,\ldots,n$ and the output value in box n+1. Also assume that initially all but the first n boxes are empty.

Primitive Recursive Functions are Abacus Computable

Basic plot: first step is to show that all **p.r. functions** are Abacus computable. To do this, need to show that the base functions are Abacus computable:

1) zero function: $\mathbf{z}(x) = 0$. Computed by the vacuous program ↓
2) successor function: $\mathbf{s}(x)$. Initially [1] = x and [2] = 0.

   Then want to add the number in box 1 to box 2 without loss from1, then add one stone to box 2:

$$[1] + [2] \rightarrow 2 \rightarrow (2+) \rightarrow \ldots$$

3) projection functions: $\mathbf{id}^{n}_{m}\,(x_1,\ldots, x_m,\ldots, x_n) = x_m$

   [m] + [n+1] → n+1, with [n+1] = 0 initially and

   [n+1] = x_m at end.

Primitive Recursive Functions are Abacus Computable

- Next need to show that the operations of composition and primitive recursion are Abacus computable:
- Composition:

  $$h^n(x_1,\ldots, x_n) = f^m(g_1(x_1,\ldots, x_n), \ldots, g_m(x_1,\ldots, x_n))$$

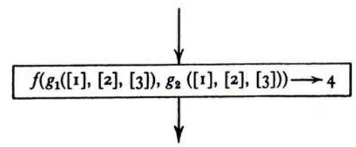
  need to construct a single program

  $$f^m(g_1([1],\ldots,[n]),\ldots, g_n(([1],\ldots,[n])) \rightarrow n+1$$

  out of $m + 1$ given programs (i.e. 1 for f plus m for the gs).

  See flow chart in B&J, p. 80.

Primitive Recursive Functions are Abacus Computable



The structure of the flow chart is shown in Figure 7-2.

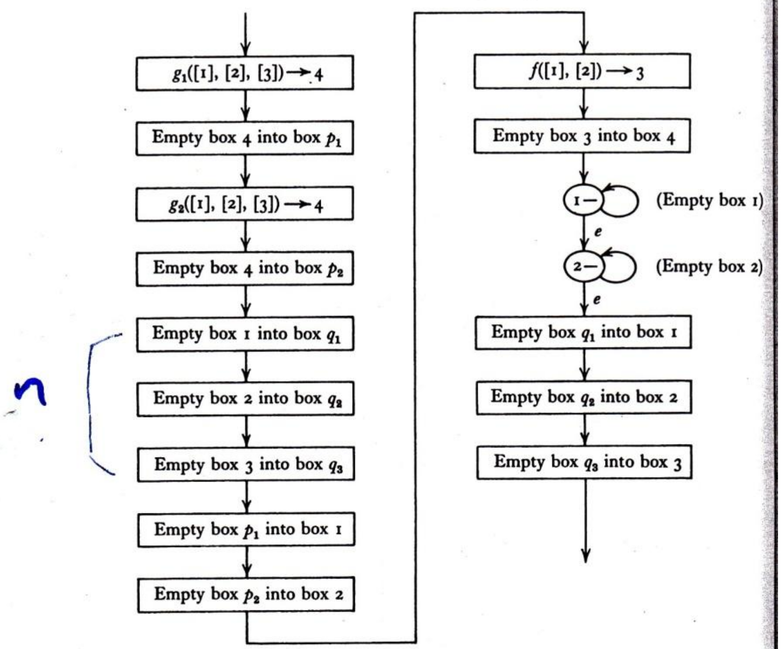


Figure 7-2. Composition of programs in case $m = 2$, $n = 3$.

Primitive Recursive Functions are Abacus Computable

- Primitive recursion:

$$h(x, 0) = f(x)$$

$$h(x, \mathbf{s}(y)) = g(x, y, h(x,y))$$

For Abacus computation, initially [1] = x, [2] = y and all other boxes empty.

start with $f([1]) \rightarrow 2 \quad g([1],[2],[3]) \rightarrow 4$

end with $h([1],[2]) \rightarrow 3$

- See flow chart in B&J, p. 82.

Primitive Recursive Functions are Abacus Computable

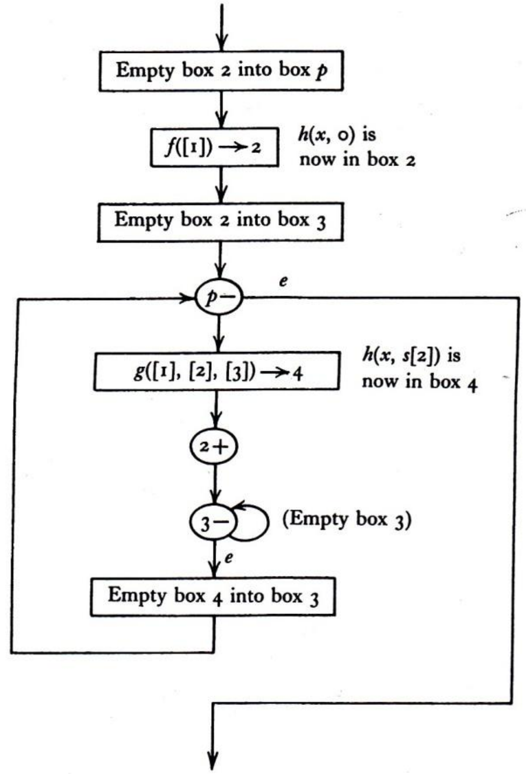


Figure 7-3. Primitive recursion.

Recursive Functions are Abacus Computable

- Finally, need to show that the operation of minimization is Abacus computable:

  For a 2-place function $f(x,y)$

  $\mathbf{Mn}[\,f\,](x) = \{$the least y for which $f(x, y) = 0$

  $= \{$undefined if $f(x, y) = 0$ for no y

  Given the Abacus machine for f, just keep computing $f(x,0)$, $f(x,1)$, $f(x,2)$…. adding a stone to box 2 each time and checking to see if $f(x,y) = 0$.

When/if it reaches a step where $f(x,y) = 0$ it will halt with $[2] = y$.

See flow chart in B&J, p. 84.

Recursive Functions are Abacus Computable

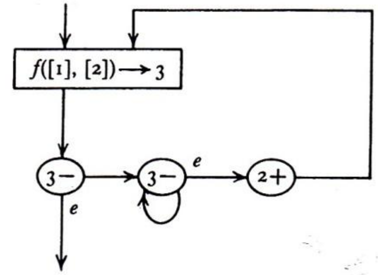


Figure 7-4. Minimization.

Recursive Functions are Abacus Computable

- **Moral of the story**: the cavemen can compute all recursive functions.

Turing Computable Functions are Recursive

- The last step to close the circle of inclusions is to show that $T \subseteq R$ and hence that $T = A = R$.
- **Basic plot**: Begin by characterizing tape configurations in terms of **p.r.** functions.
- Let the right number r be defined as the number in (backwards!) binary notion given by the string of 0's and 1's to the right of and including the currently scanned square.
- Let the left number l be the number in binary notion given by the string of 0's and 1's to the left of the currently scanned square.

Turing Computable Functions are Recursive

- Next specify how r and l change as read/write head moves one square L, R on tape:

1) if TM moves L and l is odd, then

   $l' = (l - 1)/2$ and $r' = 2r + 1$

2) if TM moves L and l is even, then

   $l' = l/2$ and $r' = 2r$

3) if TM moves R and r is odd, then

   $l' = 2l+1$ and $r' = (r - 1)/2$

4) if TM moves R and r is even, then

   $l' = 2l$ and $r' = r/2$

Turing Computable Functions are Recursive

- Initial configuration of TM which computes $f(x_1, x_2)$:

  $l = 0$, $r = s(x_1, x_2) = (2^{(x2 + 1)} \dot{-} 1) \cdot 2^{(x1 + 2)} + (2^{(x1 + 1)} \dot{-} 1)$

  Where $r = s(x_1, x_2)$ is thus a **p.r.** function on the input values.
- Halting configuration: $l = 0$,

  $r = 2^{f(x1,x2) + 1} - 1$ which is the number denoted in binary by a string of $f(x_1, x_2) + 1$ 1's.

  In this manner we can recover the output value $f(x_1, x_2)$ from r in a **p.r.** manner by defining **lo** such that

  $\mathbf{lo}(2^{f(x1,x2) + 1} - 1) = f(x_1, x_2)$

  so $\mathbf{lo}(r) = f(x_1, x_2)$)

Turing Computable Functions are Recursive

- More specifically, **lo**(x) = greatest $w \leq x$ such that $2^w \leq x$. Can be defined primitive recursively through bounded maximization.
- Set of quadruples: The TM program is represented in terms of two 2-place **p.r.** functions a and q, where a maps current state and scanned symbol to overt action, q maps current state and scanned symbol to next state.
- Transitions between configurations: represented in terms of more **p.r.** functions. See B&J chapter 8 for technical details.
- **Moral of the story**: Turing computable functions are recursive. Hence T = A = R, and powerful evidence is garnered in support of the Church-Turing Thesis.