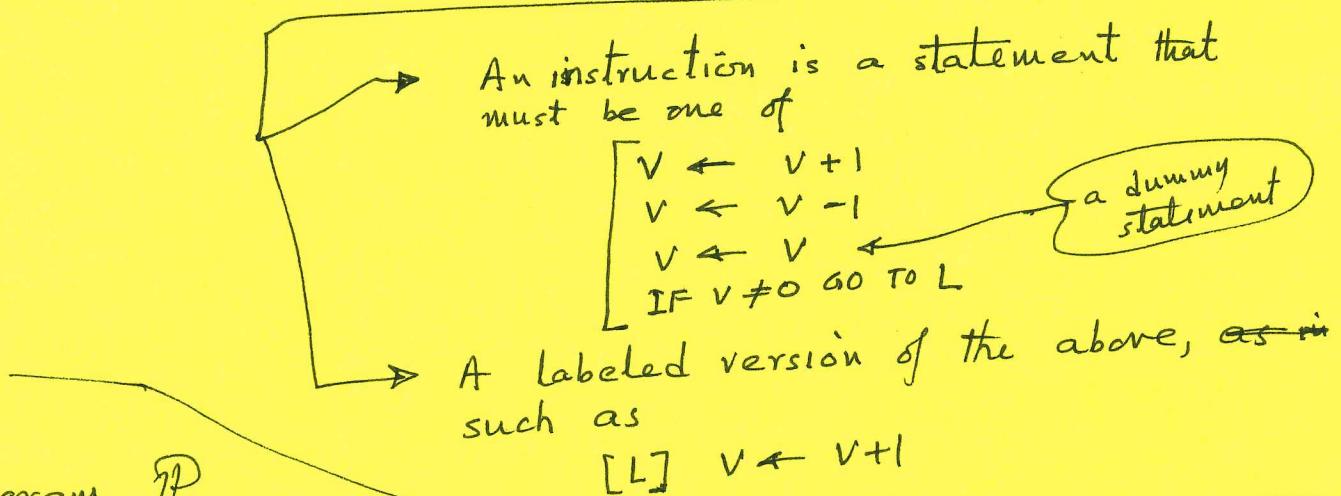


# What Does it Mean to Say That a Function is Partially Computable

First some definitions related to the execution of a program in language S. We will define :

- a program  $\mathcal{P}$
- the state  $\sigma$  of a program  $\mathcal{P}$
- the snapshot  $s = (i, \sigma)$
- successor snapshots
- terminal snapshot
- computation

- A **program**  $\mathcal{P}$  is a list (a finite sequence) of **instructions**,



- A **state**  $\sigma$  of a program  $\mathcal{P}$  is a list of equations of the form  $V = m$ , there being exactly one such equation for each variable  $V$  in  $\mathcal{P}$ .
- The state  $\sigma$  obviously ~~shon~~ may change when an instruction is executed.
- The state  $\sigma$  obviously ~~shon~~ may change when an instruction is executed. So we define a **snapshot**  $s = (i, \sigma)$ , with  $1 \leq i \leq n+1$ , for a program  $\mathcal{P}$  that consists of  $n$  instructions. The state  $\sigma$  in a snapshot  $(i, \sigma)$  is the state of  $\mathcal{P}$  just prior to the execution of the  $i^{\text{th}}$  instruction.
- By the above definition,  $(n+1, \sigma)$  is the **terminal snapshot**.
- If  $(i, \sigma)$  is a non-terminal snapshot, we define the **successor** of  $(i, \sigma)$  to be the snapshot  $(j, \tau)$  that is obtained from  $(i, \sigma)$  by the following rules :
  - $i^{\text{th}}$  instruction is  $V \leftarrow V + 1$  and  $\sigma$  contains  $V = m$ . Then  $j = i+1$  and we obtain  $\tau$  by replacing  $V = m$  by  $V = m + 1$ .
  - $i^{\text{th}}$  instruction is  $V \leftarrow V - 1$  and  $\sigma$  contains  $V = m$  with  $m > 0$ . Then  $j = i+1$  and we obtain  $\tau$  from  $\sigma$  by replacing  $V = m$  by  $V = m - 1$ . If  $m = 0$  in  $\sigma$ ,  $\tau = \sigma$ .
  - $i^{\text{th}}$  instruction is  $V \leftarrow V$ . Now  $j = i+1$  and  $\tau = \sigma$ .
  - $i^{\text{th}}$  instruction is  $\text{IF } V \neq 0 \text{ GO TO } L$ . Then  $\tau = \sigma$ . Regarding  $j$ , if  $\sigma$  contains  $V = 0$ , then  $j = i+1$ . Otherwise,  $j$  is the least number such that the  $j^{\text{th}}$  instruction of  $\mathcal{P}$  carries label  $L$ .
- A **computation** by a program  $\mathcal{P}$  is defined as a **finite** sequence  $s_1, s_2, \dots, s_k$  of snapshots of  $\mathcal{P}$  such that  $s_{i+1}$  is a successor of  $s_i$  for  $i = 1, 2, \dots, k-1$  and that  $s_k$  is a terminal snapshot.

- When there is a computation by  $\mathcal{P}$  — in the sense defined on the other side of this scroll — we denote the final value of  $y$  by

$$\psi_{\mathcal{P}}^{(m)}(r_1, r_2, \dots, r_m)$$

initial values  
for m input  
variables

We say  $\psi_{\mathcal{P}}^{(m)}$  is computed by  $\mathcal{P}$ .

Note that  $\psi_{\mathcal{P}}^{(m)}()$  is merely an expressive notation for the value of the output variable  $y$ . The notation tells us that  $y$  gets its values through the program  $\mathcal{P}$  and with values specified for  $m$  input variables. In and of itself,  $\psi_{\mathcal{P}}^{(m)}$  is not an attempt to define a function.

- A given partial function

$$g(r_1, r_2, \dots, r_m)$$

note the distinction we are making between a partial function and a partially computable function

- is said to be partially computable if there exists a program  $\mathcal{P}$  such that

$$g(r_1, r_2, \dots, r_m) = \psi_{\mathcal{P}}^{(m)}(r_1, r_2, \dots, r_m)$$

Both sides must remain undefined at exactly for the same arguments. (The RHS is undefined when  $\mathcal{P}$  gets trapped in a loop. In this case, there will be no terminal state.)

## MORE ON MACROS

- Let  $f(x_1, x_2, \dots, x_n)$  be some partially computable function computed by a program  $\mathcal{P}$ . Let's say we expect to call  $f(x_1, \dots, x_n)$  repeatedly in a larger program. So we would like to call  $f(x_1, \dots, x_n)$  as a macro.
- To create a macro expansion for  $f(x_1, \dots, x_n)$ , let's first write the program  $\mathcal{P}$  that straightforwardly implements  $f$ .
- Let all the variables of  $\mathcal{P}$  be  $y, x_1, x_2, \dots, x_n, z_1, \dots, z_k$  and let all the labels of  $\mathcal{P}$  be  $E, A_1, A_2, \dots, A_L$ .
- To make explicit the variable names and the label names used in  $\mathcal{P}$ , we can denote the program by

$$\mathcal{P}(y, x_1, \dots, x_n, z_1, \dots, z_k; E, A_1, \dots, A_L)$$

- To convert this code into a macro expansion, our main concern is with name collisions with the variable and label names in the calling program. To address that issue, we choose a high enough value for the index  $m$  and rewrite  $\mathcal{P}$  so that it becomes

$$Q_m = \mathcal{P}(z_m, z_{m+1}, \dots, z_{m+n}, z_{m+n+1}, \dots, z_{m+n+k}; E_m, A_{m+1}, \dots, A_{m+k})$$

- We can now use  $Q_m$  in a **macro call**:

$$W \leftarrow f(v_1, \dots, v_n)$$

where the **macro expansion** is

$$\begin{aligned} z_m &\leftarrow 0 \\ z_{m+1} &\leftarrow v_1 \\ \vdots & \\ z_{m+n} &\leftarrow v_n \\ z_{m+n+1} &\leftarrow 0 \\ \vdots & \\ z_{m+n+k} &\leftarrow 0 \end{aligned}$$

$$[E_m] W \leftarrow z_m$$

This gives precise meaning to converting a program into a macro expansion