Computable Functions
What does it exactly mean when we say that a program computes a function? We would like to find a precise definition for this.
Let \( P \) be any program in the language \( L \) and let \( r_1, \ldots, r_m \) be \( m \) given numbers.
We form the state \( \sigma \) of \( P \) which consists of the equations
\[
X_1 = r_1, \; X_2 = r_2, \; \ldots, \; X_m = r_m, \; Y = 0
\]
and the equations \( V = 0 \) for all other variables \( V \) in \( P \).
We call this the initial state.

Moreover, we call the snapshot \((1, \sigma)\) the initial snapshot.
When running the program, there are two possible outcomes:

Case 1:
There is a computation \( s_1, s_2, \ldots, s_k \) of \( P \) beginning with the initial snapshot.
Then we write \( \psi_P(m)(r_1, r_2, \ldots, r_m) \) for the value of the variable \( Y \) at the terminal snapshot \( s_k \).

Case 2:
There is no such computation; i.e., there is an infinite sequence \( s_1, s_2, s_3, \ldots \) beginning with the initial snapshot.
In this case, \( \psi_P(m)(r_1, r_2, \ldots, r_m) \) is undefined.

In general, a partial function \( f \) on a set \( S^m \) is a function whose domain is a subset of \( S^m \).
If a partial function on \( S^m \) has the domain \( S^m \), then it is called total.
Computation

A given partial function $g$ (of one or more variables) is said to be **partially computable** if it is computed by some program.

This is the case if there is a program $\varphi$ such that $g(r_1, r_2, \ldots, r_m) = \psi^{(m)}(r_1, r_2, \ldots, r_m)$ for all $r_1, r_2, \ldots, r_m$.

This means not only that both sides have the same value when they are **defined**, but also that when either side of the equation is **undefined**, the other one is as well.

A function is said to be **computable** if it is both partially computable and total.

 Macros

So far we have used macros in an informal way. Let us now develop a more precise definition of them.

Let $f(x_1, x_2, \ldots, x_n)$ be a partially computable function, and $\varphi$ be a program that computes $f$.

Let us assume that

- the variables in $\varphi$ are named $Y, X_1, \ldots, X_n, Z_1, \ldots, Z_n$,
- the labels in $\varphi$ are named $E, A_1, \ldots, A_l$, and
- for each instruction of $\varphi$ of the form

  $$\text{IF } V \neq 0 \text{ GOTO } A_i$$

  there is in $\varphi$ an instruction labeled $A_i$.

Whenever we expand a macro, the number $m$ has to be chosen **large enough** so that none of the variables or labels in $\varphi_m$ occur in the main program.

Note that in the expansion the output and local variables are set to zero, although at the start of the main program they are set to zero anyway.

This is necessary because the macro expansion may be part of a loop in the main program.

Macros

We can modify any program of the language $L$ to comply with these assumptions.

We write:

$\varphi = \varphi(Y, X_1, \ldots, X_n, Z_1, \ldots, Z_n; E, A_1, \ldots, A_l)$

In particular, we will use:

$\varphi_m = \varphi(Z_{m+1}, Z_{m+2}, \ldots, Z_{m+n}; E_{m}, A_{m+1}, \ldots, A_{m+l})$

for a given value $m$. 

Macros

Now we want to use macros of the form:

$Z_m \leftarrow 0$

$Z_{m+1} \leftarrow V_1$

$Z_{m+2} \leftarrow V_2$

in our programs, where $W$, $V_1, \ldots, V_n$ can be any variables; $W$ could be among $V_1, \ldots, V_n$.

We expand the macro as follows:

$Z_{m+n+1} \leftarrow 0$

$Z_{m+n+2} \leftarrow 0$

$\vdots$

$Z_{m+n+k} \leftarrow 0$

$Q_m \leftarrow [E_{m}] W \leftarrow Z_m$

Macros

Obviously, if $f(V_1, \ldots, V_n)$ is undefined ($\uparrow$), the program $Q_m$ will never terminate.

So if $f$ is not total, and the macro $W \leftarrow f(V_1, \ldots, V_n)$ is encountered when $V_1, \ldots, V_n$ have values for which $f$ is not defined, the main program will never terminate.

Example:

$Z \leftarrow X_1 - X_2$

$Y \leftarrow Z + X_3$

This program computes $f(x_1, x_2, x_3)$, where

$f(x_1, x_2, x_3) = \begin{cases} (x_1 - x_2) + x_3, & \text{if } x_1 \geq x_2 \\ \uparrow, & \text{if } x_1 < x_2 \end{cases}$
Macros

Now let us introduce macros of the form
IF \( P(V_1, \ldots, V_n) \) GOTO L ,
where \( P(x_1, \ldots, x_n) \) is a computable predicate.
This will be based on the convention that
TRUE = 1, FALSE = 0.
According to this convention, predicates are just total
functions whose values are always either 0 or 1.

Macros

Let \( P(x_1, \ldots, x_n) \) be a computable predicate.
Then we expand the macro
IF \( P(V_1, \ldots, V_n) \) GOTO L
to
\[ Z \leftarrow P(V_1, \ldots, V_n) \]
IF \( Z \neq 0 \) GOTO L
As usual, the variable \( Z \) has to be chosen to create
no conflicts with the main program.

Example:

How can we expand the macro
IF \( V=0 \) GOTO L ?
\( V = 0 \) corresponds to the following predicate \( P(x) \):
\[ P(x) = \begin{cases} 
\text{TRUE} & \text{if } x = 0 \\
\text{FALSE} & \text{otherwise} 
\end{cases} \]
This can be computed by the following program:
IF \( X \neq 0 \) GOTO E
\( Y \leftarrow Y+1 \)

(Partially) Computable Functions

By introducing macros, we have seen that it is
possible to compute complex functions with our very
simple programming language \( L \).
Notice that macros do not change the specification of
the language, but they just simplify writing down
programs.
As you know, we can always replace macros with
actual code.
So what are the limitations of the language \( L \)?
In order to find out, we need to do some
mathematics.

Composition

Let us combine computable functions in such a way
that the output of one becomes an input to another.
For example, we could combine the functions \( f \) and \( g \)
to obtain a new function \( h \):
\[ h(x) = f(g(x)) \]

Let us now take a more general view:

Definition: Let \( f \) be a function of \( k \) variables and let
\( g_1, \ldots, g_k \) be functions of \( n \) variables. Let
\[ h(x_1, \ldots, x_n) = f(g_1(x_1, \ldots, x_n), \ldots, g_k(x_1, \ldots, x_n)) \]
Then \( h \) is said to be obtained from \( f \) and \( g_1, \ldots, g_k \)
by composition.

Theorem 1.1: If \( h \) is obtained from the (partially)
computable functions \( f, g_1, \ldots, g_k \) by composition,
then \( h \) is (partially) computable.

Proof: The following program obviously computes \( h \):
\[ Z_1 \leftarrow g_1(X_1, \ldots, X_n) \]
\[ \vdots \]
\[ Z_k \leftarrow g_k(X_1, \ldots, X_n) \]
\[ Y \leftarrow f(Z_1, \ldots, Z_k) \]
If \( f, g_1, \ldots, g_k \) are not only partially computable but are
also total, then so is \( h \).