Computation

In general, a partial function $f$ on a set $S^n$ is a function whose domain is a subset of $S^n$.

If a partial function on $S^n$ has the domain $S^n$, 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

$$g(r_1, r_2, \ldots, r_m) = \varphi(m)(r_1, r_2, \ldots, r_m)$$

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 $Y, X_1, \ldots, X_n, Z_1, \ldots, Z_k$,
- 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 $A_i$.

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

We write:

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

In particular, we will use:

$$Q_m = \varphi(Z_m, Z_{m+1}, \ldots, Z_{m+n}, Z_{m+n+1}, \ldots, Z_{m+n+k}; E_m, A_{m+1}, \ldots, A_{m+l})$$

for a given value $m$.

Macros

Whenever we expand a macro, the number $m$ has to be chosen large enough so that none of the variables or labels in $Q_m$ 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

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

**Example:**

\[
\begin{align*}
Z & ← X_1 - X_2 \\
Y & ← Z + X_3 \\
\end{align*}
\]

This program computes \( f(x_1, x_2, x_3) \), where
\[
\begin{align*}
f(x_1, x_2, x_3) &= (x_1 - x_2) + x_3, \text{ if } x_1 \geq x_2 \\
&= \uparrow, \text{ if } x_1 < x_2 \\
\end{align*}
\]

Macros

Now let us introduce macros of the form

\[
\text{IF } P(V_1, \ldots, V_n) \text{ GOTO } L,
\]

where \( P(x_1, \ldots, x_n) \) is a computable predicate.

This will be based on the convention that

\[
\text{TRUE} = 1, \quad \text{FALSE} = 0.
\]

According to this convention, predicates are just total functions whose values are always either 0 or 1.

**Example:**

How can we expand the macro \( \text{IF } V=0 \text{ GOTO } L \) ?

\( V = 0 \) corresponds to the following predicate \( P(x) \):

\[
P(x) = \text{TRUE}, \text{ if } x = 0 \\
\quad = \text{FALSE}, \text{ otherwise}
\]

This can be computed by the following program:

\[
\begin{align*}
\text{IF } &Y=0 \text{ GOTO } E \\
&Y ← Y+1
\end{align*}
\]