IL LINGUAGGIO DI PROGRAMMAZIONE S

I simboli X_1, X_2, … sono chiamati **variabili d'ingresso**, i simboli Z_1, Z_2, … sono chiamati **variabili locali** e il simbolo Y è chiamato **variabile di uscita.** Tutte le variabili di S possono assumere come valori interi non negativi e non viene posto alcun limite ai valori che possono assumere le variabili.
I simboli A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, … sono chiamati **etichette** di S e sono poste a sinistra di una istruzione entro [ ].
Le istruzioni di S sono:

- **Incremento**: $V \leftarrow V+1$
  aumenta di 1 il valore della variabile V
- **Decremento**: $V \leftarrow V-1$
  diminuisce di 1 il valore di V se $V \neq 0$, altrimenti lo lascia invariato
- **Istruzione condizionale**: IF $V \neq 0$ GOTO L
  se $V \neq 0$ esegue l'istruzione con etichetta L, altrimenti esegue l'istruzione successiva

CONVENZIONI:

1) Le variabili locali Z_i e la variabile di uscita Y hanno inizialmente tutte valore 0.
2) Il valore di una variabile sarà spesso indicato dalla lettera minuscola corrispondente. Per esempio y è il valore della variabile Y.

- Un programma è una lista finita di istruzioni.
- Le istruzioni possono essere etichettate oppure no.
- Il calcolo viene effettuato passando in ordine da un'istruzione alla successiva, tranne quando si incontra un'istruzione condizionale che, se la premessa è soddisfatta, ci rinvia mediante l'etichetta, ad un'istruzione specifica.
- Il programma si ferma quando non trova più istruzioni da eseguire.

PROGRAMMI IN S

a) f(x) = { 1 se x = 0;   x se $x \neq 0$

```
1. [A] X ← X – 1
2.     Y ← Y + 1
3.     IF X ≠ 0 GOTO A
```

Esempio di calcolo con X=0
(1; X=0, Y=0)
(2; X=0, Y=0)
(3; X=0, Y=1)
Il programma si ferma con Y=1, mentre X rimane 0.

Esempio di calcolo con X=2
(1; X=2, Y=0)
(2; X=1, Y=0)
(3; X=1, Y=1)
(1; X=1, Y=1)
(2; X=0, Y=1)
(3; X=0, Y=2)
Il programma si ferma con Y=2, mentre il valore di X si è azzerato.

b) f(x) = x

```
1. [A] IF X ≠ 0 GOTO B
2.     Z ← Z + 1
3.     IF Z ≠ 0 GOTO E
4. [B] X ← X – 1
5.     Y ← Y + 1
6.     Z ← Z + 1
7.     IF Z ≠ 0 GOTO A
```

Poiché non è possibile usare solo l’istruzione GOTO E (senza l’IF), introduciamo la variabile locale Z e la coppia di istruzioni
Z ← Z + 1
IF Z ≠ 0 GOTO E
che equivale all’istruzione GOTO E, perché Z ≠ 0 è sempre vera grazie all’istruzione che la precede.
In generale un’espressione GOTO L si chiama **macro** e il segmento di programma che essa racchiude si chiama la sua **espansione macro**.

Esempio di calcolo con X=0
(1; X=0, Y=0, Z=0)
(2; X=0, Y=0, Z=0)
(3; X=0, Y=0, Z=1)
EXIT

Esempio di calcolo con X=2
(1; X=2, Y=0, Z=0)
(4; X=2, Y=0, Z=0)
(5; X=1, Y=0, Z=0)
(6; X=1, Y=1, Z=0)
(7; X=1, Y=1, Z=1)
(1; X=1, Y=1, Z=1)
(4; X=1, Y=1, Z=1)
(5; X=0, Y=1, Z=1)
(6; X=0, Y=2, Z=1)
(7; X=0, Y=2, Z=2)
(1; X=0, Y=2, Z=2)
(2; X=0, Y=2, Z=2)
(3; X=0, Y=2, Z=3)
EXIT

Alla fine del processo di calcolo il valore di X viene azzerato. Modifichiamo b) in modo che si conservi il valore iniziale di X, introducendo una variabile ausiliaria Z:
c) f(x) = x

1. [A] IF X ≠ 0 GOTO B
2. GOTO C
3. [B] X ← X – 1
4. Y ← Y + 1
5. Z ← Z + 1
6. GOTO A
7. [C] IF Z ≠ 0 GOTO D
8. GOTO E
9. [D] Z ← Z – 1
10. X ← X + 1
11. GOTO C

Osservazioni:
- Si noti l’uso ripetuto della macro GOTO L.
- Il primo ciclo del programma A↻B copia il valore di X sia in Z sia in Y, azzerando X; il secondo ciclo C↻D usa Z per riportare X al suo valore iniziale.
- Se i valori iniziali erano X=m, Y=0, Z=0, al termine del programma avremo X=m, Y=m, Z=0.

Esempio di calcolo con X=1, Y=0, Z=0
(1; X=1, Y=0, Z=0)
(3; X=1, Y=0, Z=0)
(4; X=0, Y=0, Z=0)
(5; X=0, Y=1, Z=0)
(6; X=0, Y=1, Z=1)
(1; X=0, Y=1, Z=1)
(2; X=0, Y=1, Z=1)
(7; X=0, Y=1, Z=1)
(9; X=0, Y=1, Z=1)
(10; X=0, Y=1, Z=0)
(11; X=1, Y=1, Z=0)
(7; X=1, Y=1, Z=0)
EXIT

Il programma c) funziona solo nel caso in cui le variabili Y e Z hanno inizialmente il valore 0. Questo è assicurato se il programma è usato da solo, ma non lo è se viene utilizzato all'interno di un altro programma. Per essere certi che Y e Z abbiano assegnato inizialmente il valore 0 dobbiamo introdurre una macro $V \leftarrow 0$ la cui macro espansione è:

[L] $V \leftarrow V - 1$
    IF $V \neq 0$ GOTO L

A questo punto possiamo anche introdurre la macro $V \leftarrow V'$ che sostituisce al valore della variabile V quello di V', lasciando V' inalterato. Per far questo premettiamo la macro $V \leftarrow 0$ ad una copia del programma c) ottenuto rimpiazzando la variabile X con V' ed Y con V:

1. $V \leftarrow 0$
2. [A] IF $V' \neq 0$ GOTO B
3. GOTO C
4. [B] $V' \leftarrow V' - 1$
5. $V \leftarrow V + 1$
6. $Z \leftarrow Z + 1$
7. GOTO A
8. [C] IF $Z \neq 0$ GOTO D
9. GOTO E
10. [D] $Z \leftarrow Z - 1$
11. $V' \leftarrow V' + 1$
12. GOTO C

Osservazioni:
- Il programma alla fine della sua esecuzione lascia Z con il valore 0. Inoltre essendo Z una variabile locale, non si trova nel programma principale nel quale la macro verrà inserita. Pertanto essendo una nuova variabile, Z sarà inizializzata a 0.
- Non solo le variabili locali, ma anche tutte le etichette devono essere diverse da quelle già usate nel programma principale.
- L'etichetta di uscita (che qui è E) deve essere identica a quella della prima istruzione successiva alla macro nel programma principale.

Esaminiamo adesso programmi che hanno due variabili di ingresso X_1 e X_2.

d) Funzione somma: $f(x_1, x_2) = x_1 + x_2$

1. $Y \leftarrow X_1$
2. $Z \leftarrow X_2$
3. [B] IF $Z \neq 0$ GOTO A
4. GOTO E
5. [A] $Z \leftarrow Z - 1$
6. $Y \leftarrow Y + 1$
7. GOTO B

Osservazioni:
- Se $X_2 = 0$ allora $Y = X_1$.
  Se invece $X_2 \neq 0$, allora anche $Z \neq 0$ e viene attivato il ciclo B$\circlearrowleft$A che aggiunge, passo passo, ad Y un numero di unità pari al valore di Z; alla fine del calcolo Y varrà $X_1 + X_2$, mentre Z ritornerà a 0 ed X_1 manterrà il suo valore iniziale.
- Il ruolo della variabile ausiliare Z è quello di permettere ad X_2 di mantenere il valore iniziale.
- In questo programma vengono usate due volte la macro $V \leftarrow V'$ e due volte la macro GOTO L.

e) Funzione prodotto: $f(x_1, x_2) = x_1 \cdot x_2$

1. $Z_2 \leftarrow X_2$
2. [B] IF $Z_2 \neq 0$ GOTO A
3. GOTO E
4. [A] $Z_2 \leftarrow Z_2 - 1$
5. $Z_1 \leftarrow X_1 + Y$
6. $Y \leftarrow Z_1$
7. GOTO B

Osservazioni:
- Se $X_2 = 0$ allora il programma esce lasciando $Y = 0$.
- Se invece $X_2 \neq 0$, allora viene attivato il ciclo B$\circlearrowleft$A che somma X_1 a se stesso X_2 volte, producendo così il prodotto tra X_1 e X_2.
- Se $X_1 = 0$ il programma produrrà il risultato corretto, ma attraverso un processo inutilmente lungo.
- All'istruzione 5 abbiamo usato la macro che rappresenta la funzione somma.

f) Funzione parziale: $g(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{se } x_1 \geq x_2 \\ \text{non definito} & \text{se } x_1 < x_2 \end{cases}$

```
1.       Y ← X1
2.       Z ← X2
3.   [C] IF Z ≠ 0 GOTO A
4.       GOTO E
5.   [A] IF Y ≠ 0 GOTO B
6.       GOTO A
7.   [B] Y ← Y – 1
8.       Z ← Z – 1
9.       GOTO C
```

Osservazioni:
- Se $X_1 = m$ e $X_2 = n$, il programma decrementa, attraverso l'etichetta B, entrambi i valori fino a quando uno dei due non si azzera.
- Se prima si azzera il primo valore, cioè se $m < n$, si ha che $Y = 0$ e con le istruzioni 5 e 6 il programma non termina mai.
- Viceversa se $m \geq n$, si ha che $Z = 0$ e il programma termina con $Y = m - n$.

Costruiamo la macro **$W \leftarrow f(V_1, \dots, V_n)$** dove f è una funzione parzialmente calcolabile in S e perciò esiste un programma $\mathcal{P}$ che la calcola. Assumiamo che:
- Tutte le variabili che compaiono in $\mathcal{P}$ sono incluse nella lista: $Y, X_1, \dots, X_n, Z_1, \dots, Z_k$
- Tutte le etichette che compaiono in $\mathcal{P}$ sono incluse nella lista: $E, A_1, \dots, A_l$
- Per ogni istruzione IF $V \neq 0$ GOTO A_i vi è in $\mathcal{P}$ un'istruzione etichettata A_i

Indichiamo la dipendenza di tali variabili in modo esplicito: $\mathcal{P} = \mathcal{P}(Y, X_1, \dots, X_n, Z_1, \dots, Z_k; E, A_1, \dots, A_l)$.
Sostituiamo adesso tutte le variabili con variabili ausiliarie, facendo una traslazione di m indici:
$Q_m = \mathcal{P}(Z_m, Z_{m+1}, \dots, Z_{m+n}, Z_{m+n+1}, \dots, Z_{m+n+k}; E_m, A_{m+1}, \dots, A_{m+l})$.
La macro espansione di $W \leftarrow f(V_1, \dots, V_n)$ è data da:

$$
\begin{array}{ll}
 & Z_m \leftarrow 0 \\
 & Z_{m+1} \leftarrow V_1 \\
 & Z_{m+2} \leftarrow V_2 \\
 & \dots\ \dots\ \dots\ \dots \\
 & Z_{m+n} \leftarrow V_n \\
 & Z_{m+n+1} \leftarrow 0 \\
 & \dots\ \dots\ \dots\ \dots \\
 & Z_{m+n+k} \leftarrow 0 \\
 & Q_m \\
[E_m] & W \leftarrow Z_m
\end{array}
$$

La Z_m e le variabili da Z_{m+n+1} a Z_{m+n+k} sono state azzerate perché questo programma potrebbe essere inserito in un ciclo di un altro programma più grande e quindi i valori di queste variabili vanno messi a 0. Grazie a questa macro possiamo scrivere <u>immediatamente</u> dei programmi.

Per esempio un programma che calcola la funzione
$f(x_1, x_2, x_3) = \begin{cases} (x_1 - x_2) + x_3 & \text{se } x_1 \geq x_2 \\ \text{non definita} & \text{se } x_1 < x_2 \end{cases}$
è dato da:
1. $Z \leftarrow X_1 - X_2$
2. $Y \leftarrow Z + X_3$

Se $X_1 - X_2$ non è definita allora la macro $Z \leftarrow X_1 - X_2$ non finirà mai di calcolare e di conseguenza tutto il programma non si fermerà, il che corrisponde al fatto che f non è definita per quei valori.

Costruiamo la macro **IF $P(V_1, \dots, V_n)$ GOTO L** dove p è un predicato calcolabile.
La macro espansione di IF $P(V_1, \dots, V_n)$ GOTO L è data da:

$Z \leftarrow P(V_1, \dots, V_n)$ (questa istruzione è la macro $W \leftarrow f(V_1, \dots, V_n)$)
IF $Z \neq 0$ GOTO L

DESCRIZIONE FORMALE DEL LINGUAGGIO S

Un **enunciato** è un qualsiasi elemento della lista che segue:

- $V \leftarrow V$
- $V \leftarrow V + 1$
- $V \leftarrow V - 1$
- IF $V \neq 0$ GOTO L

Un'**istruzione** è un enunciato oppure un enunciato preceduto da un etichetta racchiusa tra parentesi quadre (**istruzione etichettata**).

Un programma è una lista finita di istruzioni. La **lunghezza** di un programma è la lunghezza di tale lista, cioè è uguale al numero di istruzioni del programma. Il programma di lunghezza 0, detto **programma vuoto**, non contiene nessuna istruzione.
Lo **stato** di un programma $\mathcal{P}$ è una lista di equazioni del tipo $V = m$, dove V è una variabile ed m è un numero. Tale lista contiene esattamente una equazione per ogni variabile che occorre in $\mathcal{P}$.
Indichiamo con σ uno stato di un programma $\mathcal{P}$ e con "<u>valore di V in σ</u>" quel numero q che compare nell'equazione $V = q$ presente in σ. Osserviamo che:

- Non è necessario che lo stato sia raggiunto a partire da uno stato iniziale.
- Non possono essere presenti due equazioni nella stessa variabile.
- Non può mancare un'equazione in una delle variabili di $\mathcal{P}$, ma può essere presente un'equazione in una variabile non presente in $\mathcal{P}$.

Dato un programma $\mathcal{P}$ di lunghezza n, la sua **descrizione istantanea** è la coppia $S = (i, \sigma)$ dove σ è uno stato di $\mathcal{P}$, mentre l'indice i indica l'istruzione che sta per essere eseguita con $1 \leq i \leq n+1$.
Una descrizione istantanea di un programma $\mathcal{P}$ di lunghezza n si dice **terminale** se $i = n+1$ (il programma si ferma).
Se $S = (i, \sigma)$ non è terminale allora l'istantanea successiva (j, τ) è così definita:

- Se l'i-esima istruzione di $\mathcal{P}$ è $V \leftarrow V+1$ e σ contiene l'equazione $V = m$, allora $j = i+1$ e τ è ottenuta da σ rimpiazzando l'equazione $V = m$ con $V = m+1$.
- Se l'i-esima istruzione di $\mathcal{P}$ è $V \leftarrow V - 1$ e σ contiene l'equazione $V = m$, allora $j = i+1$ e τ è ottenuta da σ rimpiazzando l'equazione $V = m$ con $V = m - 1$ se $m \neq 0$, altrimenti se $m = 0$ allora $\tau = \sigma$.
- Se l'i-esima istruzione di $\mathcal{P}$ è $V \leftarrow V$, allora $j = i+1$ e $\tau = \sigma$.
- Se l'i-esima istruzione di $\mathcal{P}$ è IF $V \neq 0$ GOTO L allora $\tau = \sigma$ e
  se σ contiene $V = 0$ allora $j = i+1$;
  se σ contiene $V = m$ (con $m \neq 0$) e non esiste nessuna istruzione etichettata L allora $j = n+1$, altrimenti j è il minimo numero tale che la j-esima istruzione è etichettata L (perché potremmo avere più di un'istruzione con la stessa etichetta).

Si chiama **calcolo** di un programma $\mathcal{P}$ una successione di descrizioni istantanee $S_1, S_2, \dots, S_k$ di $\mathcal{P}$ tali che S_{i+1} è l'istantanea successiva a S_i, per ogni $i = 1, 2, \dots, k-1$, ed S_k è terminale.

FUNZIONI CALCOLABILI IN S

Sia $\mathcal{P}$ un qualsiasi programma nel linguaggio S e siano $r_1, \dots, r_m$ m numeri dati. Lo stato iniziale σ di $\mathcal{P}$ è composta dalle equazioni $X_1 = r_1, X_2 = r_2, \dots, X_m = r_m, Y = 0$ e da tante equazioni $V = 0$ quante sono le variabili in $\mathcal{P}$ diverse da $X_1, X_2, \dots, X_m, Y$. La descrizione istantanea iniziale di $\mathcal{P}$ è data da $(1, \sigma)$.

Sono possibili due casi:

1) Esiste una computazione $S_1, \dots, S_k$ di $\mathcal{P}$ che parte dall'istantanea iniziale: indichiamo con $\Psi_{\mathcal{P}}^{(m)}(r_1, \dots, r_m)$ il valore della Y nell'istantanea terminale S_k.
2) Non esiste una tale computazione, cioè $\Psi_{\mathcal{P}}^{(m)}(r_1, \dots, r_m)$ non è definita: in questo caso abbiamo una successione infinita $S_1, S_2, S_3, \dots$ che inizia con l'istantanea iniziale S_1.

Se il programma ha n variabili di ingresso e ne sono specificate solo m (con $m < n$), allora alle rimanenti $n - m$ variabili è assegnato il valore 0.

Dato un programma $\mathcal{P}$ e un intero positivo m, si dice che la funzione $\Psi_{\mathcal{P}}^{(m)}$ è calcolata da $\mathcal{P}$.

Data una funzione parziale g, si dice che è parzialmente calcolabile se esiste un programma $\mathcal{P}$ tale che $g(r_1, \dots, r_m) = \Psi_{\mathcal{P}}^{(m)}(r_1, \dots, r_m)$ per ogni $r_1, \dots, r_m$; sia g che $\Psi_{\mathcal{P}}^{(m)}$ devono essere entrambe definite, altrimenti se una non è definita allora anche l'altra non lo è.

Operatore di minimalizzazione non limitata:

$g = \min_y P(x_1, \dots, x_n, y) = \{$ minimo valore di y per cui il predicato P è vero se un tale g esiste
$\{$ non definito altrimenti

PROPOSIZIONE: Se $P(x_1, \dots, x_n, y)$ è un predicato calcolabile e se $g(x_1, \dots, x_n) = \min_y P(x_1, \dots, x_n, y)$, allora g è una funzione parzialmente calcolabile.

DIM: g è calcolata dal seguente programma:

1. [A] IF $P(X_1, \dots, X_n, Y)$ GOTO E
2. $Y \leftarrow Y + 1$
3. GOTO A

Inizialmente la Y ha valore 0. La prima istruzione controlla se il predicato è verificato per 0. Se lo è, il programma si ferma e il valore della variabile d'uscita è proprio 0. Altrimenti si passa all'istruzione 2 che aumenta di 1 il valore di Y e poi all'istruzione 3 che rimanda all'istruzione 1 e il ciclo riprende. Se non esiste alcun valore il programma non si fermerà mai e ciò sta a significare che la funzione non è definita.

TEOREMA: Se h è ottenuta dalle funzioni (parzialmente) calcolabili $f, g_1, \dots, g_k$ mediante composizione, allora h è (parzialmente) calcolabile.

DIM: $h = f(g_1(x_1, \dots, x_n), g_2(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$ è calcolata dal seguente programma:

$Z_1 \leftarrow g_1(X_1, \dots, X_n)$
....
$Z_k \leftarrow g_k(X_1, \dots, X_n)$
$Y \leftarrow f(Z_1, \dots, Z_k)$

TEOREMA: Se h è ottenuta da una funzione calcolabile g mediante

$\{ h(0) = k$
$\{ h(x+1) = g(x, h(x))$

allora h è calcolabile.

DIM: introduciamo la macro $Y \leftarrow k$: $Y \leftarrow Y+1$
$Y \leftarrow Y+1$
… … … …
(k volte)

Un programma che calcola h è:

1. $Y \leftarrow k$
2. [A] IF $X = 0$ GOTO E
3. $Y \leftarrow g(Z, Y)$
4. $Z \leftarrow Z + 1$
5. $X \leftarrow X - 1$
6. GOTO A

FUNZIONI INIZIALI

- $S(x) = x+1$
  Programma: $Y \leftarrow Y + 1$
- $n(x)$ costante 0
  Programma vuoto
- Funzioni di selezione: $u_i^n(x_1, \dots , x_n)$
  Programma: $Y \leftarrow X_i$

Quindi abbiamo che:

- le funzioni iniziali della ricorsività primitiva sono esprimibili col linguaggio S;
- la composizione e la ricorsione possono esprimersi anche col linguaggio S;
- oltre la composizione e la ricorsione abbiamo un nuovo operatore, l'operatore di minimalizzazione non limitata, che è esprimibile col linguaggio S, ma non è ricorsivo primitivo.

Da questo segue che:

- La classe delle funzioni calcolabili è una classe PRC.
- Ogni funzione ricorsiva primitiva è calcolabile.
- L'operatore di minimalizzazione non limitata include nel concetto di calcolabilità anche le funzioni parziali.

RIEPILOGO

<u>Le seguenti formulazioni sono tutte equivalenti</u>

μ-ricorsività (estensione della ricorsività primitiva):

- Composizione
- Ricorsione
- Minimalizzazione

ε-ricorsività (ricorsività generale):

- Sostituzione
- Rimpiazzamento

λ-definibilità:

- Cambiamento delle variabili vincolate
- Eliminazione ed introduzione di λ

Calcolabilità secondo Turing: è un punto di vista completamente diverso!

Come il linguaggio S rappresenta un linguaggio di programmazione che copre l'insieme delle funzioni calcolabili, allo stesso modo la macchina di Turing è un esempio di una macchina, un automa di calcolo, che copre l'intero insieme delle funzioni calcolabili.
Il linguaggio L invece copre un insieme più piccolo di funzioni calcolabili, che si limitano a calcolare solo le funzioni ricorsive primitive.