IL LINGUAGGIO S

Corso di Informatica Teorica

Prof. **Settimo Termini**

Il linguaggio di programmazione S di Davis/Weyuker

Descrizione **informale**

- I simboli X_1 X_2 ... sono chiamati variabili d'ingresso
- I simboli Z_1 Z_2 ... sono chiamati variabili locali
- Il simbolo Y è chiamato variabile d'uscita

Tutte le variabili di $\mathcal{S}$ possono assumere come valori interi non negativi. A volte, per semplicità, si porrà X al posto di X_1 e Z al posto di Z_1.
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 $\mathcal{S}$ e sono poste, a volte, a sinistra di un'istruzione entro [ ] (anche qui, a volte, l'indice 1 verrà tralasciato).

Il linguaggio di programmazione S di Davis/Weyuker

Le istruzioni di $\mathcal{S}$ sono:

nome	istruzione	spiegazione
incremento	$V \quad V+1$	aumenta di 1 il valore della variabile V
decremento	$V \quad V-1$	diminuisci di 1 il valore di V se $V \neq 0$, altrimenti lascialo invariato
istruzione condizionale	IF $V \neq 0$ GOTO L	se $V \neq 0$ esegui l'istruzione con etichetta L, altrimenti esegui l'istruzione successiva

Convenzioni

1. Le variabili locali Z_i e la variabile di uscita Y, inizialmente, hanno tutte valore 0
2. Il valore di una variabile sarà spesso indicato dalla lettera minuscola corrispondente

E ESEMPIO y è il valore di Y

E ESEMPIO z_3 è il valore di Z_3

Convenzioni

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

Se, per sbaglio, abbiamo dato la stessa etichetta a due istruzioni che succede?
Conveniamo che il programma va a cercare la prima istruzione con quell'etichetta.

Programmi in S

ESEMPIO Esempio di programma in $\mathcal{S}$

[A] $X \leftarrow X - 1$
$Y \leftarrow Y + 1$
IF $X \neq 0$ GOTO A

Vogliamo adesso vedere come “funziona” il programma precedente calcolando alcuni valori.
Per poter fare ciò in maniera sistematica introduciamo, si pure per il momento in modo informale, una notazione utile allo scopo.

Programmi in S

1. Numeriamo le istruzioni
2. Compiliamo una lista di n-ple ciascuna delle quali è così formata:

• il primo elemento è il numero dell'istruzione esaminata

• gli elementi successivi sono delle equazioni che ci dicono quali sono in quel momento i valori delle variabili

3. Il passaggio da una n-pla alla successiva è regolato dall'istruzione presa in esame

ESEMPIO Esempi di calcolo

Valore iniziale $X = 0$

$(1;\ X = 0,\ Y = 0)$

$(2;\ X = 0,\ Y = 0)$

$(3;\ X = 0,\ Y = 1)$

e ci fermiamo con $Y = 1$.

X ha mantenuto il valore iniziale 0.

[A]	X $\leftarrow$ X - 1	1.
	Y $\leftarrow$ Y + 1	2.
	IF X $\neq$ 0 GOTO A	3.

E ESEMPIO Esempi di calcolo

Valore iniziale $x = 2$

$(1;\ X = 2,\ Y = 0)$

$(2;\ X = 1,\ Y = 0)$

$(3;\ X = 1,\ Y = 1)$

$(1;\ X = 1,\ Y = 1)$

$(2;\ X = 0,\ Y = 1)$

$(3;\ X = 0,\ Y = 2)$

e ci fermiamo con $Y = 2$.

Il valore iniziale di x è azzerato.

[A]	$\mathrm{X} \leftarrow \mathrm{X} \leftarrow \mathrm{X}$ - 1	1.
	$\mathrm{Y} \leftarrow \mathrm{Y} + 1$	2.
	IF X $\neq$ 0 GOTO A	3.

Programmi in S

Ci si convince facilmente che il comportamento per $x = 2$ si ripete per qualsiasi valore di $x \neq 0$. Concludiamo che il programma calcola la funzione:

$$f(x) = \begin{cases} 1 & se \quad x = 0 \\ x & se \quad x \neq 0 \end{cases}$$

mentre il valore iniziale di x viene azzerato alla fine del programma.

Possiamo pensare di utilizzare il programma precedente per ottenere la funzione identità $f(x) = x$. Per far ciò dobbiamo correggere il comportamento del programma solo per il valore $x = 0$.

Poiché inizialmente la variabile di uscita Y ha il valore 0, possiamo pensare di modificare il programma in modo tale che nel caso $X = 0$ si fermi immediatamente prima di effettuare qualsiasi modifica alle variabili (in particolare a quella di uscita Y). Per $X \neq 0$ può continuare a funzionare come prima.

Funzione identità

E ESEMPIO Funzione $f(x) = x$

[A] IF X ≠ 0 GOTO B
Z ← Z + 1
IF Z ≠ 0 GOTO E

[B] X ← X − 1
Y ← Y + 1
Z ← Z + 1
IF Z ≠ 0 GOTO A

E ESEMPIO Esempi di calcolo

Valore iniziale $X = 0$

$(1;\ X = 0,\ Y = 0,\ Z = 0)$

$(2;\ X = 0,\ Y = 0,\ Z = 0)$

$(3;\ X = 0,\ Y = 1,\ Z = 1)$

SI FERMA

(la correzione ha funzionato)

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

E ESEMPIO Esempi di calcolo

Valore iniziale $X = 2$

$(1;\ X = 2,\ Y = 0,\ Z = 0)$

$(4;\ X = 2,\ Y = 0,\ Z = 0)$

$(5;\ X = 1,\ Y = 0,\ Z = 0)$

$(6;\ X = 1,\ Y = 1,\ Z = 0)$

$(7;\ X = 1,\ Y = 1,\ Z = 1)$

$(1;\ X = 1,\ Y = 1,\ Z = 1)$

$(4;\ X = 1,\ Y = 1,\ Z = 1)$

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

(il calcolo continua nella slide seguente)

$(5;\ X = 0,\ Y = 1,\ Z = 1)$
$(6;\ X = 0,\ Y = 2,\ Z = 1)$
$(7;\ X = 0,\ Y = 2,\ Z = 2)$
$(1;\ X = 0,\ Y = 2,\ Z = 2)$
$(2;\ X = 0,\ Y = 2,\ Z = 2)$
$(3;\ X = 0,\ Y = 2,\ Z = 3)$

SI FERMA
Valore di uscita $Y = 2$

Si osservi che a fine calcolo $X = 0$

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

Macro e espansioni macro

? DOMANDA Perché abbiamo introdotto la variabile locale z?

Anche per poter introdurre un'istruzione $(*)$ GOTO L

che come tale non è disponibile nel nostro linguaggio. La coppia di istruzioni

$$(**)\begin{cases} \mathrm{Z} \leftarrow \mathrm{Z}+1 & (1) \\ \mathrm{IF\ Z}\ \neq 0\ \mathrm{GOTO\ L} & (2) \end{cases}$$

si comporta proprio come un'istruzione GOTO L, perché grazie, alla 1, la 2 deve sempre essere eseguita.

Possiamo usare la $(*)$ come un'abbreviazione della coppia di istruzioni $(**)$ anche se, a rigore, non è un'istruzione di $\mathcal{S}$. Un'espressione come la $(*)$ si chiama **macro** ed il segmento di programma che essa riassume si chiama la sua **espansione macro**.

Programmi in S

Entrambi i programmi alla fine del processo di calcolo azzerano il valore di X distruggendo quindi il valore iniziale.
Vogliamo adesso creare una modifica del secondo programma che preservi il valore iniziale della variabile d'ingresso, continuando a calcolare la funzione identità.
Questo lo possiamo facilmente ottenere introducendo una variabile ausiliaria che, dopo aver calcolato Y ci permetta di riportare X al suo valore iniziale.

Un programma che fa ciò è il seguente:

[A] IF X $\neq$ 0 GOTO B
GOTO C

[B] X $\leftarrow$ X $-$ 1
Y $\leftarrow$ Y $+$ 1
Z $\leftarrow$ Z $+$ 1
GOTO A

[C] IF Z $\neq$ 0 GOTO D
GOTO E

[D] Z $\leftarrow$ Z $-$ 1
X $\leftarrow$ X $+$ 1
GOTO C

E ESEMPIO Esempio di calcolo

Valore iniziale $x = 1$

$(1;\ x = 1,\ y = 0,\ z = 0)$

$(3;\ x = 1,\ y = 0,\ z = 0)$

$(4;\ x = 0,\ y = 0,\ z = 0)$

$(5;\ x = 0,\ y = 1,\ z = 0)$

$(6;\ x = 0,\ y = 1,\ z = 1)$

$(1;\ x = 0,\ y = 1,\ z = 1)$

$(2;\ x = 0,\ y = 1,\ z = 1)$

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

(il calcolo continua nella slide seguente)

E ESEMPIO Esempio di calcolo

$(7;\ x = 0,\ y = 1,\ z = 1)$
$(9;\ x = 0,\ y = 1,\ z = 1)$
$(10;\ x = 0,\ y = 1,\ z = 0)$
$(11;\ x = 1,\ y = 1,\ z = 0)$
$(7;\ x = 1,\ y = 1,\ z = 0)$
EXIT

$\mathrm{X} = 1;\ \mathrm{Y} = 1;\ \mathrm{Z} = 0$

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

OSSERVAZIONE Si noti l'uso ripetuto della macro GOTO L

OSSERVAZIONE 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.

OSSERVAZIONE Quindi, se i valori iniziali erano: $\mathrm{X} = m$; $\mathrm{Y} = 0$; $\mathrm{Z} = 0$
Al termine del programma avremo: $\mathrm{X} = m$; $\mathrm{Y} = m$; $\mathrm{Z} = 0$.

Macro assegnazione

Il comportamento del programma ci induce a prenderlo in considerazione per introdurre una macro V $\leftarrow$ V' che sostituisca il valore della variabile V con quello di V', lasciando V' inalterato.
Il comportamento "impeccabile" è tale, però, solo nel caso in cui le variabili Y e Z abbiano inizialmente il valore 0. Questo è automaticamente assicurato se il programma viene usato da solo. Non lo è invece se viene utilizzato all'interno di un programma più grosso.
Per essere certi che Y e Z abbiano assegnato il valore 0 prima che entri in funzione il programma precedente dobbiamo introdurre una macro V $\leftarrow$ 0 prima di esso.

Macro azzeramento

La macro espansione di V $\leftarrow$ 0 è la seguente:

[L] V $\leftarrow$ V $-$ 1

IF X $\neq$ 0 GOTO L

Il programma che si ottiene premettendo la macro V $\leftarrow$ 0 ad una copia del programma precedente ottenuto rimpiazzando la variabile X con V' ed Y con V ci da proprio quello che cercavamo: una macro espansione di V $\leftarrow$ V'.

IL LINGUAGGIO S

```
[A]   IF X ≠ 0 GOTO B
      GOTO C
[B]   X ← X − 1
      Y ← Y + 1
      Z ← Z + 1
      GOTO A
[C]   IF Z ≠ 0 GOTO D
      GOTO E
      Z ← Z − 1
      X ← X + 1
      GOTO C
```

```
      V ← 0
[A]   IF V ≠ 0 GOTO B
      GOTO C
[B]   V' ← V' − 1
      V ← V + 1
      Z ← Z + 1
      GOTO A
[C]   IF Z ≠ 0 GOTO D
      GOTO E
      Z ← Z − 1
      V' ← V' + 1
      GOTO C
```

Macro azzeramento

? DOMANDA Come mai non abbiamo introdotto anche una macro Z $\leftarrow$ 0 per la variabile locale Z che è utilizzata?

Il programma è tale da lasciare Z con il valore 0 alla fine della sua esecuzione e quindi non c'è alcun problema per ciò che accadrà dopo, anche se è parte di un **ciclo**. Ma per prima?
Essendo una variabile locale, Z non deve occorrere già nel programma principale nel quale la macro in oggetto verrà inserita. Essendo una nuova variabile, Z sarà, pertanto, automaticamente inizializzata a 0.

Macro azzeramento

OSSERVAZIONE Non solo le variabili locali ma anche tutte le etichette devono essere diverse da quelle già usate nel programma principale.

OSSERVAZIONE L'etichetta di uscita (che qui è E) deve essere identica a quella della prima istruzione successiva alla macro del programma principale (e se tale istruzione non è già etichettata dobbiamo etichettarla).

Programmi in S

Esaminiamo adesso il programma che segue, che ha due variabili di ingresso

$$
\begin{array}{ll}
 & Y \leftarrow X_1 \\
 & Z \leftarrow X_2 \\
[B] & \text{IF } Z \neq 0 \text{ GOTO A} \\
 & \text{GOTO E} \\
[A] & Z \leftarrow Z - 1 \\
 & Y \leftarrow Y + 1 \\
 & \text{GOTO B}
\end{array}
$$

? DOMANDA **Cosa fa questo programma?**

Programmi in S

Il programma calcola la somma. infatti:

- Se $X_2 = 0$ allora $Y = X_1$
- Se $X_2 \neq 0$ (e quindi, in base alla seconda istruzione, anche $Z \neq 0$) viene attivato il ciclo B-A il cui scopo è quello di aggiungere, passo passo, ad Y un numero di unità pari al valore di Z (e di X_2).

Alla fine del calcolo quindi Y varrà $X_1 + X_2$ mentre Z sarà ritornato a 0 ed X_1 avrà mantenuto il suo valore iniziale X_1.

Qual è il ruolo della variabile ausiliaria Z?
Proprio quello di permettere ad X_2 di mantenere il valore iniziale.

Programmi in S

? DOMANDA **Come mai questo programma è tanto più semplice del programma "copia", nonostante la funzione somma sembri essere un po' più complessa dell'identità?**

Perché abbiamo usato pesantemente le macro:

- due volte V ← V'
- due volte GOTO L

Fornendo le rispettive espansioni otterremmo un programma lungo più del doppio di "copia".

Moltiplicazione

Costruiamo adesso un programma che calcola la **moltiplicazione**.

L'idea base è quella di sfruttare il programma per la somma che già conosciamo.

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

Moltiplicazione

OSSERVAZIONE Le prime tre istruzioni ci garantiscono che se $X_2 = 0$ allora il prodotto sarà 0, qualunque sia l'altro fattore

OSSERVAZIONE Garantito ciò, la seconda parte del programma, attraverso il ciclo A-B, produce $X_1 \cdot X_2$ sommando X_1 a se stesso X_2 volte.

OSSERVAZIONE Se $X_1 = 0$ il programma produrrà il risultato corretto, pur se attraverso un processo inutilmente lungo.

OSSERVAZIONE Anche qui abbiamo fatto uso di macro tutte le volte che abbiamo potuto farlo. In particolare, l'istruzione 5 è ammissibile perché abbiamo già mostrato un programma che fa la somma.

Esempio di funzione parziale

Consideriamo adesso il programma che segue.

	$Y \leftarrow X_1$	1.
	$Z \leftarrow X_2$	2.
[C]	IF $Z \neq 0$ GOTO A	3.
	GOTO E	4.
[A]	IF $Y \neq 0$ GOTO B	5.
	GOTO A	6.
[B]	$Y \leftarrow Y - 1$	7.
	$Z \leftarrow Z - 1$	8.
	GOTO C	9.

Esempio di funzione parziale

Se iniziamo con i valori $X_1 = m$ e $X_2 = n$, allora avremo che se $m \geq n$ il programma calcola $m - n$. Infatti Z assumerà il valore 0 quando Y ha assunto il valore $m - n$ e la quarta istruzione farà fermare il programma.
Se invece $m < n$, Y assumerà il valore 0 quando Z è ancora > 0 e quindi la terza istruzione rinvierà ad A, la A tuttavia non rimanda a B perché la premessa **non** è soddisfatta e si passa all'istruzione successiva 6 che rimanda a sua volta alla 5.
Il calcolo **non termina mai**. Il programma calcola la funzione **parziale**

$$g(x_1, x_2) = \begin{cases} x_1 - x_2 & se \quad x_1 \geq x_2 \\ \uparrow & se \quad x_1 < x_2 \end{cases}$$

Il simbolo $\uparrow$ indica che la funzione non è definita

Due macro

Usiamo adesso i risultati precedenti per introdurre le due ***macro*** di tipo molto generale che seguono:

(*) $\quad W \leftarrow f(V_1, ..., V_n)$

dove f è una funzione parzialmente calcolabile in S

(**) IF $P(V_1, ..., V_n)$ GOTO L

dove P è un predicato calcolabile.

Due macro

Lo scopo che ci proponiamo nel fare ciò è del tutto evidente; quello che vogliamo avere è una sorta di generalizzazione, rispettivamente, delle istruzioni di ***assegnazione:***

$$W \leftarrow f(V_1, ..., V_n)$$

e di ***salto condizionato***

IF $P(V_1, ..., V_n)$ GOTO L

Macro

Verifichiamo - sotto l'ipotesi che f sia una funzione parzialmente calcolabile in S - che la macro (*) è accettabile (cioè che esiste una sua macro espansione in S).

Non c'è alcun dubbio che così debba essere perchè se f è parzialmente calcolabile in esisterà un programma $\mathscr{P}$ che la calcola. L'unico problema che si pone, dunque, è quello di cambiare opportunamente le variabili usate nel programma $\mathscr{P}$ quando lo utilizziamo inserendolo nella macro espansione di (*) in modo da evitare errori dovuti a connessioni non corrette o identificazioni non volute.
Detto in altro modo, dobbiamo collegare in modo corretto le variabili presenti nel programma $\mathscr{P}$ che calcola f con le variabili generiche V_1, ... , V_n che compaiono in f

Macro

Ammettiamo allora che:

- tutte le variabili che compaiono in $\mathscr{P}$ siano incluse nella lista:

  $Y, X_1, \ldots, X_n, Z_1, \ldots, Z_k$

- tutte le etichette che compaiono in $\mathscr{P}$ siano incluse nella lista $E, A_1, \ldots, A_l$.

Assumiamo ancora che per ogni istruzione del tipo

IF $V \neq 0$ GOTO A_1

vi sia in $\mathscr{P}$ un'istruzione etichettata A_1

Tutte queste assunzioni se non automaticamente verificate nel programma $\mathscr{P}$ possono essere facilmente imposte o aggiunte.

Macro

Indichiamo adesso la dipendenza da tali variabili nel programma in modo esplicito

$$\mathscr{P} = \mathscr{P}(\mathrm{Y}, \mathrm{X}_1, \ldots, \mathrm{X}_n, \mathrm{Z}_1, \ldots, \mathrm{Z}_k, \mathrm{E}, \mathrm{A}_1, \ldots, \mathrm{A}_l.$$

Sostituiamo adesso tutte le variabili con variabili ausiliarie (facendo, in particolare, una traslazione di m indici, partiamo dalla variabile ausiliaria di indice (posto) m) avremo che:

$$\mathscr{Q}_m = \mathscr{P}(\mathrm{Z}_m, \mathrm{Z}_{m+1}, \ldots, \mathrm{Z}_{m+n}, \mathrm{Z}_{m+n+1}, \ldots, \mathrm{Z}_{m+n+k}; \mathrm{E}_m, \mathrm{A}_{m+1}, \ldots, \mathrm{A}_{m+l})$$

Adesso siamo pronti a scrivere la nostra macro espansione.

Macro

La macro espansione di $W \leftarrow f(V_1, ..., V_n)$ (dove $V_1, ..., V_n$, W sono variabili qualsiasi)

è data dal programma $\mathscr{Q}_m$ a cui abbiamo premesso una serie di istruzioni di assegnazione che effettuano il cambiamento di variabili che abbiamo discusso prima e alla fine del quale c'è un'istruzione che assegna a W il valore assunto dalla variabile Z_n che è il nome che abbiamo dato alla variabile di uscita di $\mathscr{Q}_m$. Quest'ultima istruzione avrà l'etichetta E_m

Macro

$$
\begin{aligned}
\mathrm{Z_m} &\leftarrow 0 \\
\mathrm{Z_{m+1}} &\leftarrow \mathrm{V_1} \\
\mathrm{Z_{m+2}} &\leftarrow \mathrm{V_2} \\
&\dots \\
\mathrm{Z_{m+n}} &\leftarrow \mathrm{V_n} \\
\mathrm{Z_{m+n+1}} &\leftarrow 0 \\
&\dots \\
\mathrm{Z_{m+n+k}} &\leftarrow 0 \\
&\mathscr{Q}_{\mathrm{m}} \\
[\mathrm{E_m}] \quad \mathrm{W} &\leftarrow \mathrm{Z_m}
\end{aligned}
$$

programma macro espansione di $\boldsymbol{W \leftarrow f(V_1, \dots, V_n)}$

Due macro

OSSERVAZIONE.

Mentre è chiaro cosa abbiamo fatto con le variabili
$Z_{m+1}, \ldots, Z_{m+n}, Z_{m+n+1}, \ldots, Z_{m+n+k}$
(poichè $V_{1, \ldots,} V_n$ sono i nuovi nomi delle n variabili d'ingresso $X_{1, \ldots,} X_n$, abbiamo assegnato ad esse i valori che devono assumere per calcolare $f(V_{1, \ldots,} V_n)$)
non è immediatamente evidente perché sono state esplicitamente azzerate le variabili Z_m e $Z_{m+n+1}, \ldots, Z_{m+n+k}$ dal momento che abbiamo fatto la convenzione che le variabili ausiliarie sono tutte inizialmente azzerate.

Macro

OSSERVAZIONE.

Il motivo molto banale è che la macro espansione di $W \leftarrow f(V_1, ..., V_n)$ non sarà normalmente e solitamente usata da sola ma farà parte di programmi più grandi ed è possibile che faccia parte di cicli nei quali vengono usate anche tali variabili e quindi è necessario assicurarsi, per un corretto funzionamento che tali variabili siano azzerate prima dell'esecuzione del programma $\mathscr{Q}_m$

Commento

Il poter disporre di macro del tipo $W \leftarrow f(V_1, ..., V_n)$ è molto utile per potere scrivere immediatamente dei programmi.

Consideriamo ad esempio le funzione $f(x_1, x_2, x_3) = (x_1 - x_2) + x_3$ (nella quale le parentesi svolgono un ruolo) e che, a rigore sarebbe più corretto scrivere come:

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

Macro

Un programma che calcola questa funzione si scrive immediatamente usando due macro

$$\begin{aligned} Z &\leftarrow X_1 \text{ - } X_2 \\ Y &\leftarrow Z + X_3 \end{aligned}$$

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 mai, cosa che corrisponde al fatto che la nostra funzione f non è definita per quei valori delle variabili di ingresso. Altrimenti, si passerà alla seconda macro e il programma si fermerà dando il valore corretto.

Seconda macro

Passiamo adesso alla seconda macro.

(**) IF $P(V_{1, ..., } V_n)$ GOTO L

vogliamo mostrare che essa è ammissibile in S, cioè esiste una sua macro espansione in S sotto l'ipotesi che il predicato P sia calcolabile.

Scriviamo la (**) nel modo seguente

$Z \leftarrow P(V_{1, ..., } V_n)$

IF $Z \neq 0$ GOTO L

La verifica è immediata perchè la prima istruzione è la macro di cui abbiamo appena mostrato l'ammissibilità e la seconda è una delle istruzioni base del linguaggio.

Completiamo adesso la descrizione del linguaggio S.

Innanzitutto una distinzione terminologica

Un ***enunciato*** è un qualsiasi elemento della lista seguente:

$V \leftarrow V + 1$

$V \leftarrow V - 1$

$V \leftarrow V$

IF $V \neq 0$ GOTO L

una ***istruzione*** è o un enunciato (chiamato anche istruzione non etichettata) o un enunciato preceduto da un etichetta racchiusa entro parentesi quadre [ ], detta anche istruzione etichettata.

Si osservi che nella lista precedente, al terzo posto c'è un'istruzione nuova, un'istruzione che non fa nulla. Vedremo poi perchè viene introdotta.

Continuiamo:

Un PROGRAMMA è una ***lista finita di istruzioni***
La lunghezza di un programma è la lunghezza di tale lista, ossia è uguale al numero di istruzioni del programma.
Così come, prima, tra gli enunciati possibili abbiamo assunto anche l'istruzione $V \leftarrow V$, così - tra i programmi possibili - ammetteremo anche il programma di lunghezza 0 che non ha nessuna istruzione e sarà chiamato PROGRAMMA VUOTO

Domanda: Quale funzione calcola il programma vuoto?
(RISPONDETE VOI)

Diamo adesso una descrizione formale di ciò che avviene nel corso della computazione effettuata da un generico programma di S.

Chiameremo stato di un programma $\mathscr{P}$ 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 $\mathscr{P}$

Indicheremo lo stato di un programma $\mathscr{P}$ con la lettera $\boldsymbol{\sigma}$

Chiameremo valore di V in $\boldsymbol{\sigma}$ quel numero q che compare nell'equazione V=q presente in $\boldsymbol{\sigma}$(q è unico).

Osservazioni

In base alla definizione di stato di un programma che abbiamo dato:

- 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 di una delle variabili di $\mathscr{P}$

MA

- Può essere presente un'equazione in una variabile non presente in $\mathscr{P}$

Supponiamo adesso di avere un programma P ed uno stato sigma di $\mathscr{P}$

Per poter dire che cosa avverrà abbiamo bisogno di conoscere l'istruzione che sta per essere eseguita.

Abbiamo dunque bisogno di una nuova nozione quella di ***descrizione istantanea***.

Descrizione istantanea di un programma

Dato un programma $\mathscr{P}$ di lunghezza n, la sua descrizione istantanea è la coppia $s = (i, \sigma)$ dove l'indice i, compreso tra 1 ed n+1 ($1 \leq i \leq n+1$), indica l'istruzione che sta per essere eseguita e σ è uno stato di $\mathscr{P}$.

Quando i assume il valore n+1 il programma si ferma.

Il valore di una variabile V in $s = (i, \sigma)$ è il valore di V in σ.

Una descrizione istantanea di un programma P di lunghezza n si dice terminale se i=n+1

Dobbiamo adesso considerare i casi in cui la descrizione istantanea non è terminale.

Descrizione istantanea di un programma

Se $s = (i, \sigma)$ non è terminale, allora la descrizione istantanea successiva (j, τ) è così definita:

dove σ è uno stato di $\mathscr{P}$ e l'indice i, compreso tra 1 ed n+1 ($1 \leq i \leq n+1$). indica l'istruzione che sta per essere eseguita.

σ

Quando i assume il valore n+1 il programma si ferma.
Il valore di una variabile V in $s = (i, \sigma)$ è il valore di V in σ.

Una descrizione istantanea di un programma $\mathscr{P}$ di lunghezza n si dice terminale se i=n+1

Descrizione istantanea di un programma

Se $s = (i, \sigma)$ non è terminale, allora la descrizione istantanea successiva (j, τ) è così definita:

1. **SE** l'i-esima istruzione di $\mathscr{P}$ 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 $V= m+1$.

2. **SE** l'i-esima istruzione di $\mathscr{P}$ 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 $m \neq 0$. Nel caso in cui è $m = 0$, $\tau = \sigma$

2. **SE** l'i-esima istruzione di P è $V \leftarrow V$ **ALLORA** $j=i+1$ e $\tau = \sigma$

Descrizione istantanea di un programma

Infine

4. **SE** l'i-esima istruzione di $\mathscr{P}$ è IF V ≠ 0 GOTO L **ALLORA** $\tau = \sigma$ e

SE σ contiene l'equazione V=0 **ALLORA** j=i+1

SE σ contiene l'equazione V=m, con m ≠ 0 **ALLORA**

SE 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

Infatti: potremmo avere più di una istruzione con la stessa etichetta, ma in questo modo indichiamo quale sarà l'istruzione alla quale siamo rinviati.

Calcolo di un programma

Si chiama ***calcolo di un programma*** $\mathscr{P}$ una successione di ***descrizioni istantanee*** $s_1, s_2, ..., s_k$ di $\mathscr{P}$ tali che

s_{i+1} è l'istantanea successiva a s_i, per ogni i= 1, ..., k-1, ed

s_k è terminale

Osservazioni

In base alle definizioni che abbiamo dato, permettiamo a ciascun programma di essere usato con un qualsiasi numero di variabili di ingresso, QUINDI:

Se il programma $\mathscr{P}$ ha n variabili d'ingresso e noi ne specifichiamo solo m ($< n$) allora alle rimanenti variabili n-m è assegnato il valore 0.

Se invece sono specificati più valori di ingresso delle variabili del programma allora i valori in eccesso sono ignorati.

Funzioni calcolabili in S

Sia adesso $\mathscr{P}$ ***un qualsiasi programma*** nel linguaggio S e siano $r_1, r_2, ..., r_m$ m numeri dati.

Lo stato iniziale di $\mathscr{P}$ è quello che consiste delle equazioni

$$X_1 = r_1, \quad X_2 = r_2, \quad ..., \quad X_m = r_m \quad , Y = 0$$

e da tante equazioni $V = 0$ quante sono le variabili V in $\mathscr{P}$ diverse da X_1, X_2, X_m, Y.

Sia $\boldsymbol{\sigma}$ tale stato. La descrizione istantanea ***iniziale*** di $\mathscr{P}$ è data da $(1, \boldsymbol{\sigma})$

Dobbiamo adesso esaminare i due casi possibili a seconda che esista o no una computazione di $\mathscr{P}$ a partire dallo stato iniziale $(1, \sigma)$.

Funzioni calcolabili in S

Nel caso in cui esiste una computazione $s_1, s_2, ..., s_k$ del programma $\mathscr{P}$ che parte dalla configurazione iniziale indichiamo con il simbolo:

$$\Psi^{(m)}_{\mathscr{P}}(r_1, r_2, ..., r_m)$$

il valore assunto dalla variabile Y del programma $\mathscr{P}$ nella configurazione istantanea terminale s_k.
(il perchè usiamo questa simbologia "barocca", è evidente)

Funzioni calcolabili in S

Nel caso in cui non esista una tale computazione - cioè nel caso in cui a partire dalla configurazione istantanea iniziale si passa a successive configurazioni istantanee senza arrivare mai a una configurazione istantanea terminale - allora diciamo che questo "strano oggetto"

$$\Psi_{\mathscr{P}}^{(m)}(r_1, r_2, ..., r_m)$$

non è definito.

Dato un programma, perciò, possiamo associarvi una funzione - da esso calcolata - che è proprio lo strano oggetto appena scritto. Essa non è definita se il programma non termina e assume il valore assunto dalla variabile di uscita Y quando il programma si è fermato.

Osservazioni

Data, viceversa, una funzione parziale g, si dice che g è parzialmente calcolabile in S, se esiste un programma $\mathscr{P}$ tale che

$$g(r_1, r_2, ..., r_m) = \Psi^{(m)}_{\mathscr{P}}(r_1, r_2, ..., r_m)$$

per ogni m-upla $(r_1, r_2, ..., r_m)$

L'uguaglianza è da intendere nel senso che o entrambi i lati dell'equazione sono definiti (e in questo caso hanno lo stesso valore) oppure sono entrambi non definiti.

Classi di funzioni che sono calcolabili in S

Ci chiediamo:

Che rapporto esiste tra le funzioni S-calcolabili e altre classi di funzioni che abbiamo introdotto in precedenza?

La domanda riguarda le ***funzioni Turing calcolabili***, le ***funzioni ricorsive primitive*** e le ***funzioni μ-ricorsive***.

Mostreremo adesso che le funzioni ricorsive primitive sono calcolabili in S.
Subito dopo mostreremo che anche le funzioni μ-ricorsive sono S-calcolabili.

Non abbiamo ancora tutti gli strumenti per dimostrare che lo sono anche le funzioni Turing calcolabili (lo vedremo in seguito).

Funzioni ricorsive primitive

Vogliamo mostrare adesso che le funzioni ricorsive primitive sono tutte S-calcolabili.

Ricordiamo in primo luogo la definizione:

La classe delle funzioni ricorsive primitive è quella che si ottiene a partire dalle funzioni iniziali (successore, costante 0 e funzioni di scelta) per chiusura sotto le operazioni di composizione e ricorsione.

Mostriamo in primo luogo che le funzioni iniziali sono calcolabili in S

Calcolabilità in S delle funzioni ricorsive primitive

funzione successore

$$s(\mathrm{n}) = \mathrm{x}{+}1$$

Programma che la calcola:

$$\mathrm{Y} \leftarrow \mathrm{X}{+}1$$

(NOTARE CHE LA PRECEDENTE è UNA MACRO!)

Calcolabilità in S delle funzioni ricorsive primitive

funzione <u>costante 0</u>

E' calcolata dal programma vuoto.

OVVIAMENTE potremmo scrivere tanti altri programmi che la calcolano

Calcolabilità in S delle funzioni ricorsive primitive

funzioni di <u>scelta</u> o <u>selezione</u>

$$u_i^{(n)}(x_1, ..., x_n) = x_i$$

Programmi che le calcolano:

$$Y \leftarrow X_i$$

NB ANCHE QUESTE SONO MACRO!

Mostriamo adesso che le operazioni di composizione e ricorsione conservano la calcolabilità in S

Calcolabilità in S delle funzioni ricorsive primitive

Teorema. ***Se la funzione*** h ***è ottenuta dalle funzioni*** f, g_1 , ..., g_k ***(parzialmente) calcolabili in S mediante composizione allora*** h ***è parzialmente calcolabile in S.***

Dimostrazione: h ***è calcolata dal programma seguente:***

$$
\begin{aligned}
\mathrm{Z}_1 &\leftarrow g_1(\mathrm{X}_1, \ldots, \mathrm{X}_n) \\
&\ldots \\
\mathrm{Z}_k &\leftarrow g_k(\mathrm{X}_1, \ldots, \mathrm{X}_n) \\
\mathrm{Y} &\leftarrow f(\mathrm{Z}_1, \ldots, \mathrm{Z}_k)
\end{aligned}
$$

Calcolabilità in S delle funzioni ricorsive primitive

<u>Teorema.</u> Se la funzione h è ottenuta dalla funzione g, calcolabile in S, mediante le equazioni di ricorsione

$$\begin{cases} h\,(0) = \mathrm{k} \\ h\,(\mathrm{x}+1) = \mathrm{g}(\mathrm{x},\, \mathrm{h}(\mathrm{x})) \end{cases}$$

allora h è calcolabile in S.

Calcolabilità in S delle funzioni ricorsive primitive

Dimostrazione. *h è calcolata dal programma seguente:*

$$
\begin{array}{ll}
 & \mathrm{Y} \leftarrow k \\
[\mathrm{A}] & \mathrm{IF\ X} = 0 \ \mathrm{GOTO\ E} \\
 & \mathrm{Y} \leftarrow g(\mathrm{Z},\ \mathrm{Y}) \\
 & \mathrm{Z} \leftarrow \mathrm{Z}{+}1 \\
 & \mathrm{X} \leftarrow \mathrm{X}{-}\ 1 \\
 & \mathrm{GOTO\ A}
\end{array}
$$

dove la macro $\mathrm{Y} \leftarrow k$ è facilmente espansa da:

$$
\left.
\begin{array}{l}
\mathrm{Y} \leftarrow \mathrm{Y}{+}\ 1 \\
\ldots \\
\mathrm{Y} \leftarrow \mathrm{Y}{+}\ 1
\end{array}
\right\} \quad \text{k volte}
$$

Calcolabilità in S delle funzioni ricorsive primitive

Operazione di ricorsione su funzioni di più variabili. Si può facilmente costruire un programma simile a quello precedente (con una notazione leggermente più pesante) anche per le funzioni di più variabili.

CONCLUSIONE

Possiamo quindi concludere che tutte le funzioni ricorsive primitive sono S-calcolabili. Al di là dell'interesse specifico di tale proprietà, osserviamo che abbiamo uno strumento molto potente per costruire delle macro. Quando sappiamo che una funzione è ricorsiva primitiva possiamo subito utilizzarla come macro senza dovere prima scrivere esplicitamente un programma di S.

Funzioni μ-ricorsive

Vogliamo mostrare adesso che anche le funzioni μ-ricorsive sono calcolabili in S.
Questo ci permetterà di concludere che la classe delle funzioni μ-ricorsive è contenuta in S.
Se riuscissimo a mostrare anche il viceversa e cioè che tutte le funzioni S-calcolabili sono μ-ricorsive, avremmo mostrato l'equivalenza tra due modelli di computo che partono da intuizioni molto diverse della nozione di calcolabilità.
In questo modo avremmo dato un sostegno alla Tesi di Church- Turing.
Ricordiamo in primo luogo la definizione:

La classe delle funzioni μ-ricorsive è quella che si ottiene a partire dalle funzioni iniziali per chiusura sotto le operazioni di composizione, ricorsione e minimalizzazione non limitata.

Funzioni μ-ricorsive

Quindi le funzioni μ -ricorsive sono una classe chiusa in modo ricorsivo primitivo che è una estensione propria delle funzioni ricorsive primitive

Noi abbiamo già mostrato che le ***funzioni ricorsive primitive*** sono ***calcolabili in S***, l'unica cosa che dobbiamo ancora fare, dunque, è verificare che l'operatore di ***minimalizzazione non limitata*** è ***calcolabile in S***.

In questo modo avremo completato la dimostrazione che le ***funzioni μ -ricorsive*** sono ***calcolabili in S.***

Minimalizzazione non limitata

Ricordiamo la definizione di minimalizzazione non limitata

$$g = \min_{y} P(x_1, x_2, ..., x_m, y) = \begin{cases} \textit{minimo valore di } y \textit{ per cui } P \textit{ è vero } \underline{\textit{se tale } y \textit{ esiste}} \\ \uparrow \quad \textit{(non definito), altrimenti} \end{cases}$$

Vogliamo dimostrare la

Proposizione. Se $P(x_1, x_2, ..., x_m, y)$ è un predicato computabile e se

$$g\,(x_1, x_2, ..., x_m, y) = \min_y P(x_1, x_2, ..., x_m, y)$$

allora g è una funzione parzialmente calcolabile

Minimalizzazione non limitata

Dimostrazione. g è calcolata dal semplice programma che segue

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

Infatti, ricordando le convenzioni fatte, Y ha inizialmente il valore 0. La prima istruzione verifica se il predicato P è soddisfatto per 0. Se lo è allora il programma si ferma ed il valore della variabile di uscita è proprio 0, perchè la seconda istruzione non ha avuto modo di modificare il valore della variabile di uscita Y.
Altrimenti si passa alla seconda istruzione che aumenta di 1 il valore di Y, poi alla terza istruzione che rimanda alla prima e il ciclo riprende. Se non esiste alcun valore che soddisfa il predicato, il programma non si fermerà mai, cosa che corrisponde correttamente al fatto che la funzione g non è definita.

Cosa è ancora in sospeso?

Molte cose tra cui:

- Dimostrare che le funzioni S-calcolabili sono mu-ricorsive
- Trovare che relazioni intercorrono tra la Turing calcolabilità, la calcolabilità in S e le funzioni mu ricorsive

MA PRIMA

cercheremo di vedere - nella prossima lezione - se anche nel caso del modello di computo fornito dal linguaggio S sussistano risultati (spiacevolmente) negativi come quelli trovati nel contesto della Turing calcolabilità (non Turing calcolabilità della funzione produttività e teorema della fermata per le MdT) e - successivamente - mostreremo un risultato positivo (esistenza di un programma ***universale***)