CALCOLI SU STRINGHE DI SIMBOLI

Verranno adesso introdotti due ulteriori linguaggi di programmazione o, meglio:

- una famiglia di linguaggi per la manipolazione di stringhe su un alfabeto di n simboli, un linguaggio per ogni n;
- un linguaggio per descrivere il funzionamento di una macchina di Turing mediante istruzioni.

I primi saranno strettamente modellati su $\mathcal{S}$, il secondo sarà il più possibile simile ad esso.

La necessità di cambiare istruzioni è imposta dal fatto che quelle che sono istruzioni elementari e del tutto naturali nel contesto degli interi sono prive di significato diretto e danno opportune interpretazioni artificiose come istruzioni per manipolare stringhe.

Sia adesso A un alfabeto di n simboli, $A=\{s_1,\dots,s_n\}$ e A^* l'insieme di tutte le stringhe su A compresa la stringa vuota 0 (in termini algebrici A^* è il monoide generato da A).

Per ciascun $n > 0$ verrà quindi introdotto un linguaggio di programmazione $\mathcal{S}_n$.

Tutte le regole sintattiche e le convenzioni per ciascun $\mathcal{S}_n$ sono del tutto analoghe a quelle di $\mathcal{S}$ e non verranno perciò esplicitamente enunciate.

ISTRUZIONI DEI LINGUAGGI $\mathcal{S}_n$

$V \leftarrow \sigma V$ un'istruzione per ciascun simbolo dell'alfabeto A	concatena il simbolo σ alla sinistra della stringa che rappresenta il valore di V
$V \leftarrow V^-$	cancella il simbolo finale della stringa che rappresenta il valore di V se $V=0$ non fare niente
IF V ENDS σ GOTO L un'istruzione per ciascun simbolo dell'alfabeto A e per ogni etichetta L	Se la stringa che rappresenta il valore di V termina col simbolo σ, esegui la prima istruzione etichettata L; altrimenti esegui l'istruzione successiva

Analogamente al caso di $\mathcal{S}$, diremo che una funzione parziale di m variabili su A^* è parzialmente calcolabile in $\mathcal{S}_n$ se esiste un programma in $\mathcal{S}_n$ che la calcola.

Osservazione

Possiamo pensare ai programmi di $\mathcal{S}_n$, per qualche n, o come a pure manipolazioni di stringhe su un alfabeto di n simboli o come a calcoli su numeri che sono rappresentati in base n.
Questo punto di vista è utile quando si vogliono paragonare le capacità di calcolo dei vari $\mathcal{S}_n$ tra loro e con quella di $\mathcal{S}$.

Per poter fare un confronto tra i linguaggi $\mathcal{S}$ ed $\mathcal{S}_n$ è necessario mostrare la calcolabilità in ciascun $\mathcal{S}_n$ delle due funzioni $x+1$ e $x \dot{-} 1$, questa verrà a sua volta semplificata dalla possibilità di usare alcune macro opportune.

IF $V \neq 0$ GOTO L

Espansione :

IF V ENDS S_1 GOTO L
IF V ENDS S_2 GOTO L
. . . .
IF V ENDS S_n GOTO L

$V \leftarrow 0$

Espansione : [A] $V \leftarrow V^-$
IF $V \neq 0$ GOTO A

(Idea: cancella l'ultimo simbolo e continua il procedimento finché non avrai cancellato tutti i simboli)

GOTO L

Espansione :

$Z \leftarrow 0$
$Z \leftarrow s_1 Z$
IF Z ENDS s_1 GOTO L

Il blocco di istruzioni :

IF V ENDS S_1 GOTO B_1
IF V ENDS S_2 GOTO B_2
. . . .
IF V ENDS S_n GOTO B_n

si chiama **FILTRO** e può essere abbreviato in IF V ENDS S_i GOTO B_i $(1 \leq i \leq n)$

$V' \leftarrow V$

Espansione:

```
1          Z ← 0
2          V' ← 0
3    [A]   IF V ENDS s_i GOTO B_i   (1 ≤ i ≤ n)   (FILTRO)
4          GOTO C
5    [B_i] V ← V^-        ⎫
6          V' ← s_i V'    ⎬  i = 1, 2, ..., n
7          Z ← s_i Z      ⎪
8          GOTO A         ⎭
9    [C]   IF Z ENDS s_i GOTO D_i   (1 ≤ i ≤ n)   (FILTRO)
10         GOTO E
11   [D_i] Z ← Z^-        ⎫
12         V ← s_i V      ⎬  i = 1, 2, ..., n
13         GOTO C         ⎭
```

Osservazioni

- Le istruzioni 3 e 9 sono dei filtri e quindi, in realtà blocchi di n istruzioni ciascuno.
- Anche i blocchi di istruzioni 5–8 e 11–13 dovrebbero in realtà essere ripetuti n volte.
- Il programma si basa sul funzionamento successivo dei due cicli $A \rightleftarrows B$ e $C \rightleftarrows D$ (dove B e D indicano, genericamente, quel B_i o D_i che di volta in volta viene chiamato).
  - Il primo ciclo cancella ad ogni passo una lettera di V che ricopia sia in V' sia in Z (che sono state entrambe azzerate)
    Alla fine di questo processo V è cancellata e due sue copie sono presenti in V' e in Z.
  - Il secondo ciclo con un procedimento simile ricopia Z in V.
    Alla fine avremo dunque $Z=0$ e sia V che V' col valore iniziale di V.

Abbiamo adesso gli strumenti per presentare due programmi semplici che calcolano le funzioni $x+1$ e $x \dot{-} 1$ in $\mathcal{S}_n$.

Prima di fare questo però è forse opportuno, anche se non necessario, riprendere il problema della rappresentazione dei numeri in A^*.

Ciò che ci serve è una rappresentazione degli interi uno-a-uno. Quindi la solita rappresentazione decimale che consente ambiguità del tipo

$$007 = 7$$

non è utile da questo punto di vista.

Le rappresentazioni che ci interessano quindi sono del tutto simili alle solite tranne per il punto precedente il che comporterà che in una rappresentazione in base n i simboli elementari rappresenteranno i numeri da 1 a n anziché da 0 ad $n-1$ e lo 0 verrà, a sua volta, identificato con la stringa vuota che indicheremo con lo stesso simbolo 0.

Sia dunque un alfabeto A di n lettere $s_1, \dots, s_n$. Assumeremo che gli s_i sono stati posti in un ordine definito (che è quello dato dagli indici) e ci riferiremo sempre a questo ordine.

Consideriamo adesso la stringa:

$$w = s_{i_k} s_{i_{k-1}} \cdots s_{i_1} s_{i_0}$$

associamo a w il numero intero:

$$(*) \qquad x = i_k \cdot n^k + i_{k-1} \cdot n^{k-1} + \ldots + i_1 \cdot n + i_0$$

w sarà allora chiamato la notazione in base n (in A) del numero definito da $(*)$

Osservazione

Mentre la solita notazione in base n che ammette lo 0, ha bisogno di almeno due simboli, la notazione presente funziona anche con un solo simbolo, nel qual caso un numero n è rappresentato da una stringa eguale alla concatenazione di n copie del nostro unico simbolo a disposizione.

Si mostra che tale rappresentazione è unica nel senso che dato il numero x possiamo recuperare gli indici i_ℓ e quindi la stringa w.

Si può anche mostrare che questo può essere fatto usando delle funzioni ricorsive primitive. Esisterà quindi una funzione ricorsiva primitiva $h(m,n,x)$ (dove x è il numero del quale vogliamo recuperare la stringa corrispondente, n è la cardinalità dell'alfabeto A ed m l'indice cercato) tale che:

$$i_m = h(m,n,x) \qquad \text{per } m = 0,1,\ldots,k.$$

Sia $A = \{s_1, \dots, s_n\}$

consideriamo la stringa :

$$w = s_{i_k} s_{i_{k-1}} \dots s_{i_1} s_{i_0}$$

Questa è la rappresentazione in base n del numero :

$$x = i_k \cdot n^k + i_{k-1} n^{k-1} + \dots + i_1 n + i_0$$

Quindi :

- la base è data dalla cardinalità dell' alfabeto
- l'esponente al quale devo elevare n nei vari luoghi dipende dalla posizione del simbolo considerato nella stringa w
- il coefficiente dalla posizione del simbolo nell' ordinamento dato.

LO ZERO È RAPPRESENTATO DALLA STRINGA VUOTA!

Esempi

$$A = \{ \overset{x}{s_1}, \overset{y}{s_2}, \overset{z}{s_3} \} \qquad \text{Card}(A) = 3$$

$$w = \overset{z}{s_3}\overset{z}{s_3}\overset{x}{s_1}\overset{y}{s_2}$$

$$x = 3 \cdot 3^3 + 3 \cdot 3^2 + 1 \cdot 3 + 2$$

Modifica della base decimale

$$A = \{1, 2, \dots, 9, D\}$$

Esempi :

stringa : 763

numero associato : $7 \cdot 10^2 + 6 \cdot 10 + 3 = 763$

stringa : 3D

numero associato : $3 \cdot 10 + 10 = 40$

A questo punto avremmo gli elementi per introdurre una serie di funzioni su A^* e dimostrare la loro ricorsività primitiva.
Queste funzioni ci serviranno in seguito.
Per il momento preferiamo ritornare all'argomento che avevamo lasciato in sospeso e cioè la costruzione di programmi che calcolano

$x+1$ e $x \dot{-} 1$ in S_n.

I due programmi come vedremo tra poco sono estremamente semplici e si capisce immediatamente il meccanismo che sta alla loro base.
L'unico punto da tener presente (e che è quello che genera due cicli diversi nei programmi è il problema del riporto.

Ricordiamo che nel caso dell'addizione, in rappresent. decimale, se sommiamo 1 ad un numero di più cifre che finisce per 9, avremo un riporto che si propagherà fino a quando non incontreremo una cifra inferiore a 9 che lo riassorbirà.
ANALOGAMENTE NELLA NOSTRA RAPPRESENTAZIONE DEI NUMERI AVREMO UN PROBLEMA DI RIPORTO RISPETTIVAMENTE QUANDO VORREMO AGGIUNGERE 1 AD UN NUMERO LA CUI RAPPRESENTAZIONE PRESENTA COME ULTIMA CIFRA L'ULTIMO SIMBOLO NEL NOSTRO ORDINAMENTO E QUANDO VORREMO SOTTRARRE 1 AD UN NUMERO LA CUI RAPPRESENTAZIONE PRESENTA COME ULTIMA CIFRA IL PRIMO SIMBOLO DI A NEL NOSTRO ORDINAMENTO.

$$
\begin{array}{lll}
[B] & \text{IF } X \text{ ENDS } s_i \text{, GOTO } A_i & (1 \le i \le n) \\
 & Y \leftarrow s_1 Y & \\
 & \text{GOTO } E & \\
[A_i] & \left.\begin{array}{l} X \leftarrow X^- \\ Y \leftarrow s_{i+1} Y \\ \text{GOTO } C \end{array}\right\} & 1 \le i < n \\
[A_n] & X \leftarrow X^- & \\
 & Y \leftarrow s_1 Y & \\
 & \text{GOTO } B & \\
[C] & \text{IF } X \text{ ENDS } s_i \text{, GOTO } D_i & (1 \le i \le n) \\
 & \text{GOTO } E & \\
[D_i] & \left.\begin{array}{l} X \leftarrow X^- \\ Y \leftarrow s_i Y \\ \text{GOTO } C \end{array}\right\} & 1 \le i \le n
\end{array}
$$

il caso
$i = n$ è
trattato a parte.

Fig. 2.3. Program which computes $x + 1$ in $\mathscr{S}_n$.

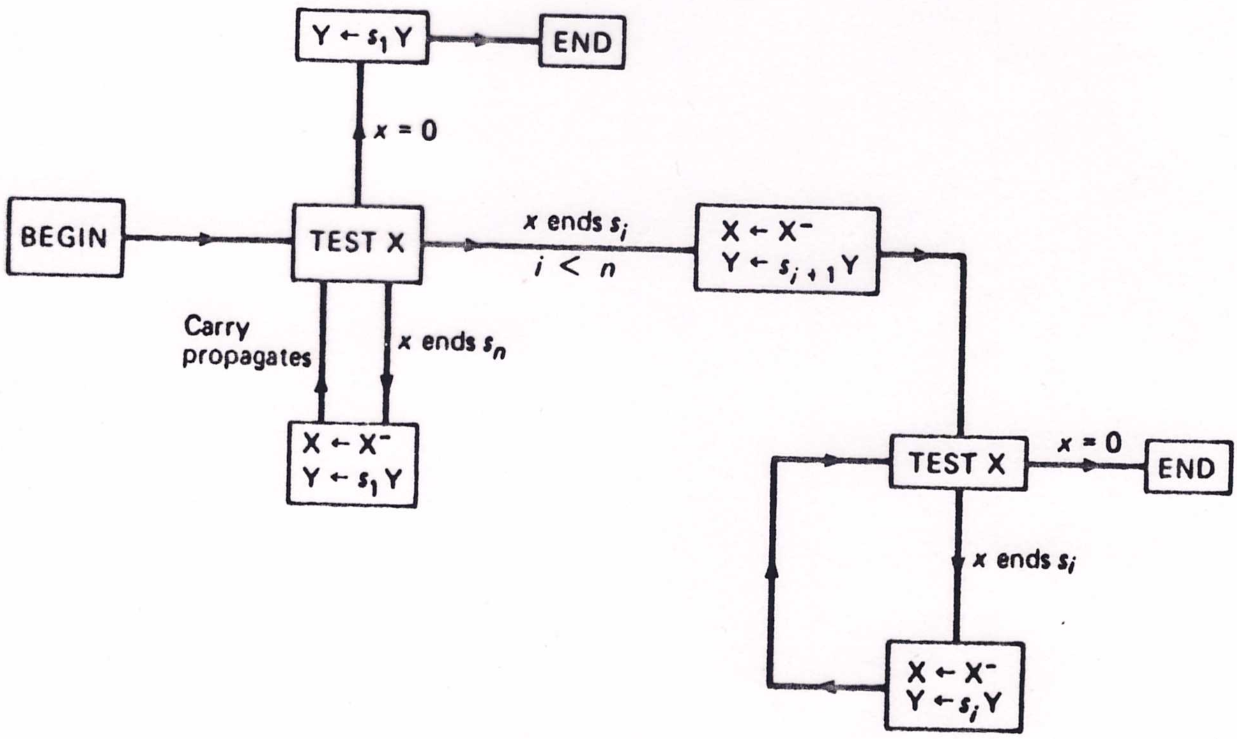


Fig. 2.2. Flow chart for computing $x + 1$ in $\mathscr{S}_n$.

[B] IF X ENDS s_i GOTO A_i $\quad (1 \leq i \leq n)$

GOTO E

[A_i] $X \leftarrow X^-$

$Y \leftarrow s_{i-1} Y$ $\quad \Big\} \; 1 < i \leq n$

GOTO C

[A_1] $X \leftarrow X^-$

IF $X \neq 0$ GOTO C_2

GOTO E

[C_2] $Y \leftarrow s_n Y$

GOTO B

[C] IF X ENDS s_i GOTO D_i $\quad (1 \leq i \leq n)$

GOTO E

[D_i] $X \leftarrow X^-$

$Y \leftarrow s_i Y$ $\quad \Big\} \; 1 \leq i \leq n$

GOTO C

il caso $i = 1$ è trattato a parte.

Fig. 2.5. Program which computes $x \dot{-} 1$ in $\mathscr{S}_n$.

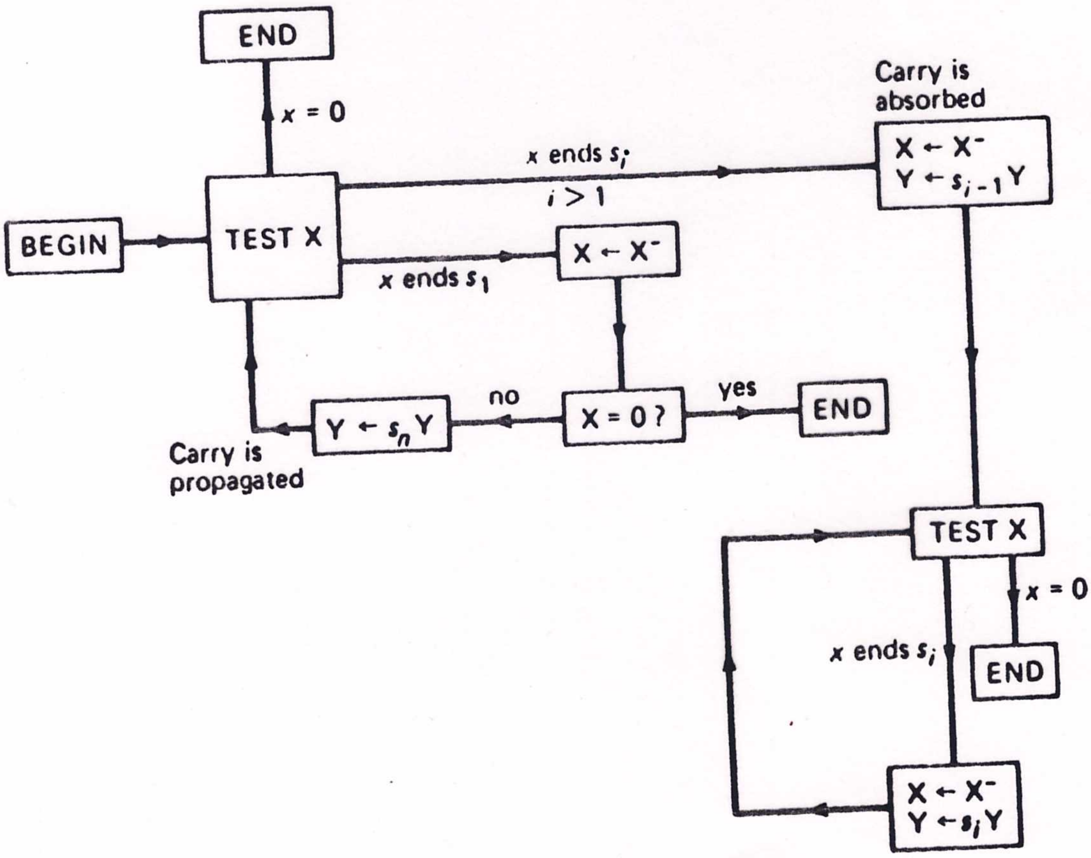


Fig. 2.4. Flow chart for computing $x \dot{-} 1$ in $\mathscr{S}_n$.

Possiamo a questo punto presentare i primi risultati del confronto della capacità di calcolo dei vari $\mathcal{S}$ e $\mathcal{S}_n$.

Ricordiamo che il punto di vista assunto per poter fare il confronto è che tutti i calcoli sono calcoli numerici e le stringhe sono solo un modo di rappresentare i numeri in una base eguale alla cardinalità dell'alfabeto.

Proposizione

Una funzione è parzialmente calcolabile (in $\mathcal{S}$) se e solo se è parzialmente calcolabile in $\mathcal{S}_1$.

Dimostrazione

Se guardiamo ad $\mathcal{S}$ e ad $\mathcal{S}_1$ dal punto di vista "numerico", i due linguaggi si comportano nello stesso identico modo data la corrispondenza esistente tra le istruzioni corrispondenti

$$(*) \qquad V \leftarrow s_1 V \quad \longleftrightarrow \quad V \leftarrow V+1$$

$$(**) \qquad V \leftarrow V^- \quad \longleftrightarrow \quad V \leftarrow V-1$$

Le (**) diminuiscono entrambe, nei rispettivi linguaggi, di una unità il valore di V.
Analogamente, le (*) aumentano di una unità il valore di V.

Le istruzioni di salto sono anch'esse identiche perché "V ENDS s_1", essendoci un solo simbolo, è equivalente a dire che $V \neq 0$.

I due linguaggi sono quindi identici.

Mostriamo adesso che:

<u>Proposizione</u> Se una funzione f è parzialmente calcolabile, allora è parzialmente calcolabile in $\mathcal{S}_n$ per ogni n.

<u>Dimostrazione</u>

Sia $\mathcal{P}$ il programma nel linguaggio $\mathcal{S}$ che calcola f.

Trasformiamo il programma $\mathcal{P}$ in $\mathcal{S}$ in un programma $\mathcal{P}_{\mathcal{S}_n}$ rimpiazzando le istruzioni di $\mathcal{P}$ con corrispondenti macro in $\mathcal{S}_n$ nel modo seguente:

	in $\mathcal{S}$			in $\mathcal{S}_n$
Istruz.:	$V \leftarrow V+1$	$\longrightarrow$	macro:	$V \leftarrow V+1$
"	$V \leftarrow V-1$	$\longrightarrow$	"	$V \leftarrow V \dot{-} 1$
"	IF $V \neq 0$ GOTO L	$\longrightarrow$	"	IF $V \neq 0$ GOTO L

Il nuovo programma $\mathcal{P}_{\mathcal{S}_n}$ in $\mathcal{S}_n$ sarà molto più lungo di $\mathcal{P}$, una volta sostituita ogni macro dalla sua macroespansione, ma ovviamente calcolerà in $\mathcal{S}_n$ la stessa funzione che $\mathcal{P}$ calcolava in $\mathcal{S}$, perché non fa altro che simulare $\mathcal{P}$ in $\mathcal{S}_n$.

Passiamo adesso ad introdurre il linguaggio di POST-TURING

Questo è un modo di esprimere il funzionamento di una macchina di Turing, come noi l'abbiamo già introdotta, usando istruzioni del tipo incontrato finora e programmi come liste di istruzioni.

Si rinvia alla discussione già fatta per tutte le motivazioni e dettagli di altro genere.

Passiamo alla presentazione del nostro linguaggio:

IL LINGUAGGIO DI POST-TURING $\mathcal{T}$

ISTRUZIONE	INTERPRETAZIONE
PRINT σ	RIMPIAZZA CON σ IL SIMBOLO PRESENTE SUL QUADRATO ESAMINATO
IF σ GOTO L	VAI ALLA PRIMA ISTRUZIONE ETICHETTATA L SE IL SIMBOLO ESAMINATO È σ ALTRIMENTI PASSA ALL'ISTRUZIONE SUCCESSIVA
RIGHT	ESAMINA IL QUADRATO IMMEDIATAMENTE A DESTRA DI QUELLO ESAMINATO
LEFT	ESAMINA IL QUADRATO IMMEDIATAMENTE A SINISTRA DI QUELLO ESAMINATO

Supponiamo di avere a che fare con funzioni su stringhe dell'alfabeto $A = \{s_1, \dots, s_n\}$.

- Un simbolo aggiuntivo s_0 o B servirà per separare i diversi "ingressi".
- Tutti i quadrati del nastro sono Blank, tranne un numero finito.
- Il quadrato esaminato sarà indicato da una freccia che lo punta:

$$B\ s_1\ B\ \underset{\uparrow}{s_2}\ s_1\ B$$

- Per configurazione di nastro si intende il contenuto del nastro più l'indicazione del quadrato esaminato.
- Per calcolare una funzione parziale di m variabili su A^*, $f(x_1, \dots, x_m)$ dobbiamo porre le variabili iniziali nel nastro e questo sarà fatto usando la seguente configurazione iniziale di nastro:

$$\underset{\uparrow}{B}\ x_1\ B\ x_2\ B \dots B\ x_m\ B$$

- Si osservi che con la nostra rappresentazione dei numeri (a differenza di quanto avveniva con la rappresentazione usata nel corso della descrizione informale delle MdT) lo zero verrà rappresentato da un Blank e quindi non verranno identificati valori iniziali eguali a zero di una n-upla se non specificando dall'esterno quante sono le variabili d'ingresso

Definizione

Sia $f(x_1, ..., x_m)$ una funzione parziale di m variabili sull'alfabeto $\{s_1, ..., s_n\}$. Allora si dice che il programma $\mathcal{P}$ nel linguaggio di Post-Turing <u>CALCOLA</u> f se posto nella configurazione iniziale

$$\underset{\uparrow}{B}\, x_1\, B \dots B\, x_m$$

prima o poi si fermerà se e solo se $f(x_1, ..., x_m)$ è definita e se, quando si ferma, può leggersi il valore $f(x_1, ..., x_m)$ (ignorando eventuali altri simboli ausiliari che sono stati usati nel corso del calcolo differenti da $s_1, ..., s_n$) nel nastro.

SI DICE CHE $\mathcal{P}$ CALCOLA <u>STRETTAMENTE</u> f SE SONO SODDISFATTE ANCHE LE DUE CONDIZIONI:

- Nessuna istruzione di $\mathcal{P}$ fa menzione di simboli (ausiliari) diversi da $s_0, s_1, ..., s_n$
- Quando $\mathcal{P}$ si ferma, la configurazione di nastro è del tipo:

$$\underset{\uparrow}{B}\, y\, B$$

dove y non contiene Blank.

Vogliamo adesso mostrare che

SE $f(x_1, \ldots, x_m)$ È PARZIALMENTE CALCOLABILE IN $\mathcal{S}_n$ ALLORA È CALCOLABILE STRETTAMENTE DA UN PROGRAMMA DI POST-TURING.

Il metodo che useremo sarà ovviamente quello di simulare il programma di $\mathcal{S}_n$ mediante un programma di Post-Turing.

Per prima cosa scriviamo alcune <u>macro</u> in $\mathcal{T}$:

GOTO L

```
IF s0 GOTO L
IF s1 GOTO L
-----
IF sn GOTO L
```

RIGHT TO NEXT BLANK

```
[A] RIGHT
    IF B GOTO E
    GOTO A
```

LEFT TO NEXT BLANK

```
[A] LEFT
    IF B GOTO E
    GOTO A
```

ERASE A BLOCK

```
[A] RIGHT
    IF B GOTO E
    PRINT B
    GOTO A
```

MOVE BLOCK RIGHT

(ponendo la testina a destra del blocco).

$[C]$ LEFT

Filtro → IF s_i GOTO A_i $(0 \le i \le n)$

$$\left.\begin{array}{ll} [A_i] & \text{RIGHT} \\ & \text{PRINT } s_i \\ & \text{LEFT} \\ & \text{GOTO } C \end{array}\right\} i = 1, 2, \dots, n$$

$[A_0]$ RIGHT

PRINT B

LEFT

L'effetto di questa macro è quello di spostare a destra un blocco di simboli (che non contiene Blank all'interno) a partire da una configurazione iniziale :

… B ☐☐☐☐☐☐ B …
↑

Convenzione

Un numero tra parentesi quadre [ ] posto dopo una macro indica che la macro deve essere ripetuta un egual numero di volte.

Ciò che adesso dobbiamo fare è simulare le tre istruzioni del linguaggio $\mathcal{S}_n$ in $\mathcal{T}$:

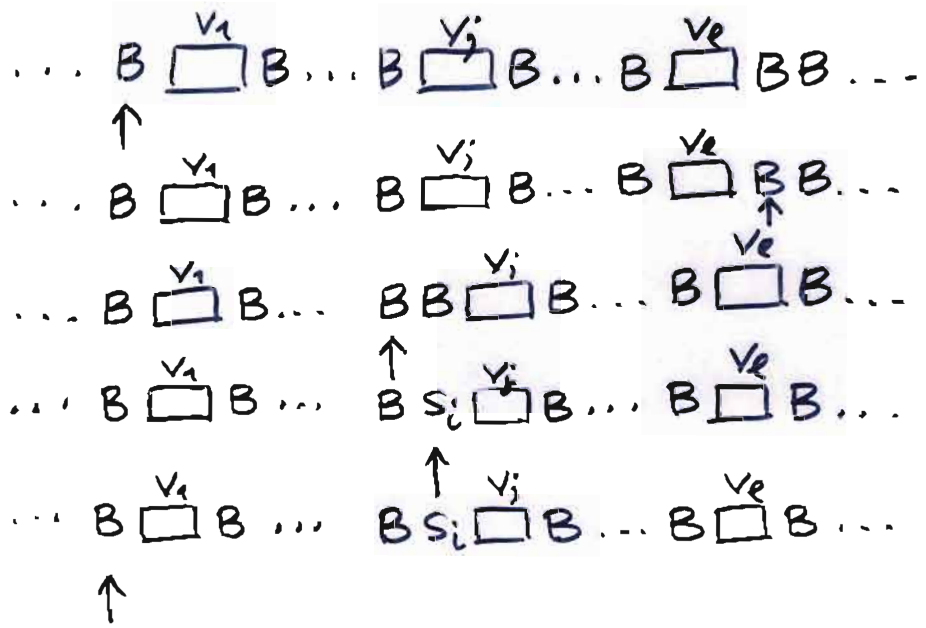
- $V_j \leftarrow s_i V_j$
- $V_j \leftarrow V_j^-$
- IF V_j ENDS s_i GOTO L

Istruzione $V_j \leftarrow s_i V_j$ (in $\mathcal{S}_n$)

Sua simulazione in $\mathcal{T}$:

I passi da compiere sono:

- andare sul Blank che segue il valore di V_ℓ (ultima variabile)
- spostare i valori delle variabili $V_j, \dots, V_\ell$ di un posto a destra
- stampare s_i nel nuovo Blank che si è creato
- ritornare nella posizione iniziale a sinistra del primo valore (sul Blank che precede il valore di V_1)



Il programma è:

```
RIGHT TO NEXT BLANK [ℓ]
MOVE BLOCK RIGHT [ℓ-j+1]
RIGHT
PRINT s_i
LEFT TO NEXT BLANK [j-1]
```

Istruzione $V_j \leftarrow V_j^-$ (in $\mathcal{S}_n$)

Sua simulazione in $\mathcal{T}$:

I punti da rilevare sono due:

- dobbiamo considerare a parte il caso in cui V_j è la parola vuota
- il meccanismo di muovere un blocco a destra automaticamente cancella i valori precedentemente scritti e quindi possiamo utilizzare opportunamente quest'istruzione per cancellare il simbolo finale di V_j.

Il programma è

```
     RIGHT TO NEXT BLANK [j]    ← si posiziona a destra della j-esima variabile
     LEFT                       ← torna a sinistra
     IF B GOTO C                ← controlla se è la stringa vuota
     MOVE BLOCK RIGHT [j]
     RIGHT
     GOTO E
[C]  LEFT TO NEXT BLANK [j-1]
```

Esaminiamo bene il comportamento di
MOVE BLOCK RIGHT

Siamo nella situazione:

$B\ \overset{V_j}{\boxed{\quad\quad}}\ B$ (**non** siamo sul Blank a destra di V_j!)
↑

Dare a questo punto un'istruzione di muoversi a destra significa cancellare il simbolo su cui punta la freccia perché:

$B\ \boxed{\quad}\ s_i$
↑

$B\,B\ \boxed{\quad}$
↑

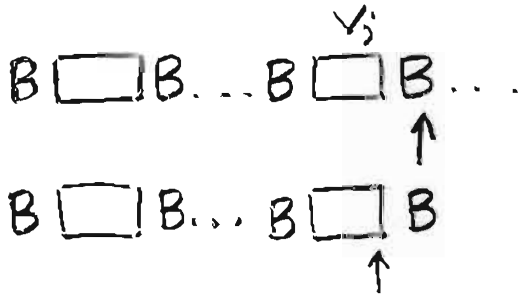
Istruzione IF V_j ENDS s_i GOTO L (in $\mathcal{S}_n$)

Sua simulazione in $\mathcal{T}$:

```
     RIGHT TO NEXT BLANK [j]
     LEFT
     IF s_i GOTO C
     GOTO D
[C]  LEFT TO NEXT BLANK [j]
     GOTO L
[D]  RIGHT
     LEFT TO NEXT BLANK [j]
```

Quali sono i passi essenziali da compiere?

- raggiungere V_j
- esaminare il suo simbolo finale.
- nel caso in cui tale simbolo finale è s_i andare all'istruzione C
- in caso contrario ritornare all'istruzione di partenza.



Avendo adesso a disposizione una simulazione in $\mathcal{T}$ dei tre tipi di istruzione di $\mathcal{S}_n$ possiamo rapidamente giungere alla conclusione.

Sia P un programma in $\mathcal{S}_n$ che calcola f. P faccia uso di k variabili locali, e quindi usi le $m+k+1$ variabili:

$$X_1, \dots, X_m, Z_1, \dots, Z_k, Y$$

Poniamo $\ell = m+k+1$ e scriviamo queste variabili come

$$V_1, \dots, V_\ell$$

Costruiamo adesso un programma di Post-Turing Q che simula P passo passo. Dobbiamo solo fare in modo che dopo ogni passo e quindi anche alla fine la configurazione di nastro sia:

$$\dots \underset{\uparrow}{B}\, x_1 B \dots B x_m B z_1 B \dots B z_k B\, y\, B \dots$$

Possiamo facilmente estrarre da questa stringa il valore di $f(x_1, \dots, x_m)$ che è dato da y.

Se vogliamo, come abbiamo esplicitamente chiesto prima, che il programma calcoli <u>strettamente</u> f, allora dobbiamo cancellare dal nastro i valori di tutte le variabili tranne y.

Questo, ricordando che tutte le variabili coinvolte sono ℓ, è facilmente ottenibile mediante l'istruzione seguente (che deve essere posta alla fine del programma precedentemente considerato):

ERASE A BLOCK $[\ell-1]$.

Dopo l'esecuzione di quest'ultima istruzione la configurazione di nastro sarà:

$$\dots \underset{\uparrow}{B}\, y\, B \dots$$

il che mostra che il programma in $\mathcal{T}$ calcola f strettamente.

Cominciamo a tirare alcune conclusioni:

Abbiamo mostrato che:

f parzialmente calcolabile (in $\mathcal{S}$)

$\Downarrow$

f parzialmente calcolabile in $\mathcal{S}_n$

$\Downarrow$

f calcolabile (strettamente) da un programma di Post-Turing

Vogliamo adesso mostrare che

f calcolabile da un programma di Post-Turing

$\Downarrow$

f parzialmente calcolabile (in $\mathcal{S}$)

Risultato che ci permette di chiudere il ciclo.

Per poter fare ciò abbiamo bisogno di alcune funzioni sulle stringhe che andremo adesso ad esaminare.

TENIAMO PRESENTE LA TABELLINA

CON LE ISTRUZIONI DI $\mathcal{S}_n$ e di $\mathcal{T}$

$\mathcal{S}_n$	$\mathcal{T}$
$V \leftarrow \sigma V$	PRINT σ
$V \leftarrow V^-$	IF σ GOTO L
IF V ENDS σ GOTO L	RIGHT
	LEFT

Consideriamo un alfabeto $A = \{s_1, \ldots, s_n\}$ e ricordiamo che le stringhe su A verranno sempre interpretate come rappresentazioni in base n di numeri interi.

Consideriamo l'operazione di concatenazione così definita:

$$\begin{cases} CONCAT_n^{(1)}(u) = u \\ CONCAT_n^{(m+1)}(u_1, \ldots, u_m, u_{m+1}) = CONCAT_n^{(m)}(u_1, \ldots, u_m)\, u_{m+1} \end{cases}$$

dove n è la cardinalità dell'alfabeto ed $m \geq 1$.

Dal punto di vista di A^* tale operazione non consiste in altro che nel porre due stringhe di simboli una accanto all'altra in modo da produrne una terza.

Tuttavia, se interpretiamo le stringhe come rappresentazioni numeriche allora si può vedere anche l'operazione CONCAT come una strana funzione numerica che produce diversi valori numerici, al variare di A, anche sulle stesse stringhe.

Mostreremo tra poco che CONCAT vista come funzione numerica è ricorsiva primitiva.

Presenteremo tra poco anche altre funzioni le quali compiono delle operazioni assolutamente elementari (e, senza alcun dubbio, costruttive) viste come operazioni su stringhe di simboli.

Se invece vediamo tali funzioni (come CONCAT) come funzioni numeriche allora richiede un minimo di cura il mostrare che esse sono calcolabili in $\mathcal{S}$ (o ricorsive primitive).

Mostreremo in dettaglio la ricorsività primitiva solo delle prime due funzioni.

1) $\boxed{\text{Funzione lunghezza } |u|}$

Osserviamo che il numero

$$\sum_{j=0}^{x} n^j = n^x + n^{x-1} + \ldots + n + 1$$

ha la rappresentazione in base n:

$$W = \underbrace{s_1 \cdot s_1 \cdot \ldots \cdot s_1}_{\substack{x+1 \\ \text{volte}}} = s_1^{[x+1]}$$

Tale numero è pertanto il più piccolo numero la cui rappresentazione in base n è lunga $x+1$.

Possiamo allora definire $|u|$ come il più piccolo x minore o eguale ad u tale che $\sum_{j=0}^{x} n^j > u$,

cioè $|u| = \min_{x \le u} \left[\sum_{j=0}^{x} n^j > u \right]$

e quindi $|u|$ è ricorsiva primitiva.

Ritorniamo adesso alla funzione CONCAT.

2) $g(u,v) = CONCAT_n(u,v)$

Sia $\underline{u}$ la stringa che rappresenta u nell'alfabeto $A = \{s_1, \dots, s_n\}$ di n simboli e $\underline{v}$ la stringa che rappresenta v.

Abbiamo che:

$\underline{u} = s_{i_k} s_{i_{k-1}} \dots s_{i_1} s_{i_0}$; $u = i_k \cdot n^k + i_{k-1} \cdot n^{k-1} + \dots + i_1 n + i_0$

$\underline{v} = s_{j_\ell} s_{j_{\ell-1}} \dots s_{j_1} s_{j_0}$; $v = j_\ell \cdot n^\ell + j_{\ell-1} \cdot n^{\ell-1} + \dots + j_1 n + j_0$

$$\underline{u}\,\underline{v} = CONCAT_n(\underline{u},\underline{v}) = s_{i_k} \cdot s_{i_{k-1}} \dots s_{i_1} s_{i_0} \mid s_{j_\ell} s_{j_{\ell-1}} \dots s_{j_1} s_{j_0}$$

e $CONCAT_n(u,v) =$

$$= \underbrace{i_k \cdot n^{k+\ell+1} + i_{k-1} \cdot n^{k+\ell} + \dots + i_1 \cdot n^{\ell+2} + i_0 \cdot n^{\ell+1}}_{n^{\ell+1}(i_k \cdot n^k + i_{k-1} \cdot n^{k-1} + \dots + i_0)} + \underbrace{j_\ell \cdot n^\ell + \dots + j_1 n + j_0}_{v} =$$

$$= n^{(\ell+1)} \cdot u + v \qquad \text{poiché } \ell+1 = |v|$$

$$= u \cdot n^{|v|} + v$$

Poiché le operazioni che intervengono sono tutte ricorsive primitive concludiamo che anche $CONCAT(u,v)$ lo è.

Usando, adesso, la proprietà precedente, l'ipotesi di induzione e l'operazione di composizione si mostra immediatamente che è primitiva ricorsiva anche l'operazione di concatenazione di m stringhe:

$$\mathrm{CONCAT}_n^{(m)}(u_1, \dots, u_m)$$

Elenchiamo adesso altre quattro funzioni la cui ricorsività primitiva, come funzioni numeriche, può dimostrarsi con tecniche simili alle precedenti e che qui omettiamo.

- $\mathrm{RTEND}_n(w)$ (right end : termine destro) : data una stringa w, produce il simbolo più a destra della parola stessa.

- $\mathrm{LTEND}_n(w)$ (left end) : analogamente, fornisce il simbolo più a sinistra di una stringa.

- $\mathrm{RTRUNC}_n(w)$ (right truncation)
  Data una stringa w produce ciò che rimane dopo aver tagliato il simbolo più a destra.

- $\mathrm{LTRUNC}_n(w)$ (left truncation)
  analogamente, fornisce ciò che rimane dopo aver tagliato il simbolo più a sinistra.

Siano adesso A e $\tilde{A}$ due alfabeti $A \subset \tilde{A}$ di n ed l simboli, rispettivamente, per cui sarà $1 \leq n < l$.

Introduciamo adesso le due operazioni

$$\mathrm{UPCHANGE}_{n,l}(x)$$

e $$\mathrm{DOWNCHANGE}_{n,l}(x)$$

il cui effetto è quello di associare al numero x il numero che si ottiene leggendo come stringa di $\tilde{A}^*$ la stringa che rappresenta x in A^* nel primo caso

e di associare ad x il numero che si ottiene leggendo come stringa di A^* la stringa che rappresenta x in $\tilde{A}^*$ nel secondo caso.

DOMANDA: Il processo descritto è fattibile?

Nel caso della funzione UPCHANGE sicuramente sì perché possiamo sempre <u>leggere</u> in $\tilde{A}^*$ una stringa di A^*.

Nel caso della funzione DOWNCHANGE il processo descritto <u>non</u> si può effettuare in generale perché una generica stringa di $\tilde{A}^*$ può contenere simboli che non appartengono ad A.

CONVENIAMO allora che in questo secondo caso il processo descritto viene effettuato sulla stringa che si ottiene cancellando i simboli di $\tilde{A} \setminus A$.

Mostriamo adesso esibendo dei programmi di $\mathcal{S}$ che UPCHANGE e DOWNCHANGE sono calcolabili.

PROGRAMMA CHE CALCOLA $UPCHANGE_{n,l}$:

$(n < l)$

```
1   [A]  IF X=0 GOTO E
2        Z ← LTEND_n (X)
3        X ← LTRUNC_n (X)
4        Y ← l·Y + Z
5        GOTO A
```

L'istruzione 2 prende il simbolo più a sinistra di X che viene usato dalla 4.

L'istruzione 3 permette di riesaminare come variabile d'ingresso al ciclo successivo ciò che è rimasto.

L'istruzione 4 al primo passo non fa che prendere il simbolo e metterlo nella variabile d'uscita, nei passi successivi concatena il nuovo simbolo fornitogli da Z ai simboli che già stavano in Y, usando la formula della Concatenazione vista come operazione numerica:

$$CONCAT_n(u,v) = u \cdot n^{|v|} + v$$

(qui, ad ogni passo, $|v| = 1$)

PROGRAMMA CHE CALCOLA DOWNCHANGE$_{n,\ell}$

$(n < \ell)$

```
[A]  IF X=0 GOTO E
     Z ← LFEND_ℓ (X)
     X ← LTRUNC_ℓ (X)
     IF Z > n GOTO A
     Y ← n·Y + Z
     GOTO A
```

Questo programma funziona esattamente come il primo tutte le volte che vengono incontrati simboli il cui indice è non superiore ad n. In quest'ultimo caso la quarta istruzione rinvia ad A senza aver permesso alcuna operazione sulla variabile di uscita e questo corrisponde all'operazione di cancellare il simbolo prima di operare la trasformazione di base.

PROPOSIZIONE

Se la funzione parziale $f(x_1, \dots, x_m)$ è calcolata da un programma di POST-TURING allora f è parzialmente calcolabile.

Dimostrazione

Sia $\mathcal{P}$ un programma di POST-TURING che calcola f. Vogliamo costruire un programma $\mathcal{Q}$ di $\mathcal{S}$ che la calcola.

Per prima cosa costruiamo il CORPO del programma, cioè una simulazione di $\mathcal{P}$ in $\mathcal{S}$.

Vedremo successivamente come "preparare" le variabili di ingresso in modo da renderle utilizzabili dal CORPO del programma.

Sia allora f una funzione parziale di m variabili su $A^* = \{s_1, \dots, s_n\}^*$.

Il programma $\mathcal{P}$ userà sicuramente l'ulteriore simbolo B ed anche altri simboli se vogliamo evitare di fare l'ipotesi aggiuntiva che $\mathcal{P}$ calcoli f strettamente.

Elenchiamo tutti i simboli usati da $\mathcal{P}$:

$$\underbrace{s_1, \dots, s_n}_{\text{elementi di A}}, \underbrace{s_{n+1}, \dots, s_r}_{\text{eventuali simboli ausiliari}}, \underbrace{s_{r+1} (= B)}_{\text{Blank}}.$$

Il programma $\mathcal{Q}$ (di $\mathcal{S}$) che simulerà $\mathcal{P}$ leggerà le stringhe come rappresentazioni di numeri in base $r+1$ ed opererà di conseguenza.

Per rappresentare "numericamente" la configurazione del nastro e delle azioni da compiere verranno introdotte le tre variabili

$$L \, , \; H \, , \; R$$

a cui verranno assegnati i seguenti valori numerici:

- ad H verrà assegnato il valore numerico del simbolo esaminato dalla testina (cioè il suo indice, il suo numero d'ordine, nella lista dei simboli).

- ad L verrà assegnato come valore il numero che rappresenta in base $r+1$ la stringa w formata da tutti i simboli che si trovano a sinistra della testina a partire dal primo simbolo sinistro diverso dal Blank (tale stringa è finita)

- ad R, in modo analogo, verrà assegnato come valore il numero che rappresenta in base $r+1$ la stringa w formata da tutti i simboli che si trovano a destra della testina fino all'ultimo simbolo destro diverso dal Blank (anche tale stringa è finita).

Date le convenzioni precedenti è possibile simulare le istruzioni di $\mathcal{T}$ in $\mathcal{S}$ nel modo seguente:

$\mathcal{T}$	$\mathcal{S}$
IF s_i GOTO L	IF $H = i$ GOTO L
PRINT s_i	$H \leftarrow i$
RIGHT	$L \leftarrow CONCAT_{n+1}(L, H)$ $H \leftarrow LTEND_{n+1}(R)$ $R \leftarrow LTRUNC_{n+1}(R)$
LEFT	$R \leftarrow CONCAT_{n+1}(H, R)$ $H \leftarrow RTEND_{n+1}(L)$ $L \leftarrow RTRUNC_{n+1}(L)$

Il corpo del programma Q che deve simulare $\mathcal{P}$ sarà adesso ottenuto rimpiazzando ciascuna istruzione di $\mathcal{P}$ (che sarà una delle istruzioni base di $\mathcal{T}$) con la corrispondente simulazione in $\mathcal{S}$.

Passiamo adesso al problema della <u>preparazione</u> dei dati iniziali in una forma opportuna per la loro elaborazione da parte del <u>CORPO</u> del programma.

f è una funzione di m-variabili su $A^* = \{s_1, \ldots, s_n\}^*$. Quindi i valori iniziali di $x_1, \ldots, x_n$ saranno numeri che rappresentano le stringhe di ingresso in base n.
La funzione UPCHANGE ci permetterà di cambiare base in modo da fare opportunamente elaborare queste stringhe dal nostro programma (dal <u>CORPO</u> del programma).
Infine dobbiamo comunicare al CORPO del programma la configurazione iniziale del nastro che è : $B x_1 B x_2 B \ldots B x_m$
Tutto ciò viene realizzato dalle seguenti istruzioni :

$$L \leftarrow 0$$
$$H \leftarrow r+1$$
$$Z_1 \leftarrow UPCHANGE_{n,r+1}(X_1)$$

- - - - - - - - - - - - - - -

$$Z_m \leftarrow UPCHANGE_{n,r+1}(X_m)$$

$$R \leftarrow CONCAT_{r+1}(Z_1, r+1, Z_2, r+1, \ldots, r+1, Z_m)$$

Come facciamo adesso a recuperare il valore della variabile d'uscita quando la macchina (o il programma) si fermerà?

Ricordando che il valore di $f(x_1, \dots, x_m)$ viene letto, per definizione, dal nastro nella configurazione finale, ignorando tutti i simboli ausiliari è chiaro che ciò che bisogna fare quando il programma si ferma è "leggere in base n" la stringa formata da tutti i simboli di $A = \{s_1, \dots, s_n\}$ presenti sul nastro.

Ciò viene realizzato dalle due istruzioni:

$$Z \leftarrow CONCAT_{r+1}(L, H, R)$$

$$Y \leftarrow DOWNCHANGE_{n, r+1}(Z).$$

Abbiamo così mostrato che se f è calcolata da un programma di POST-TURING allora è anche parzialmente calcolabile in $\mathcal{S}$ (cioè esiste un programma di $\mathcal{S}$ che la calcola).