Presentiamo adesso un procedimento per calcolare $A(i,x)$ che bene si presta ad essere programmato (Ricordiamo che, in passato, avevamo usato il procedimento di calcolo di A per introdurre la nozione di funzione (ε-) ricorsiva.

USIAMO UNA PARTICOLARE MEMORIA DETTA "PILA" IN CUI IMMAGAZZINARE GLI ARGOMENTI DI SINISTRA MENTRE STIAMO VALUTANDO QUELLI DI DESTRA.

LA PILA SARÀ INDICATA DA $(a_1,\dots,a_m)$ E SI CONVERRÀ CHE a_1 È L'ELEMENTO PIÙ IN ALTO NELLA PILA E, IN GENERALE, a_i STA PIÙ IN ALTO DI a_{i+1}.

Vogliamo adesso calcolare $A(2,2) = A(1, A(2,1))$;
Poniamo allora 1 nella pila e calcoliamo $A(2,1)$:

$A(i,0)=1$
$A(0,x)=\begin{cases} x+1 & \text{se } x=a_1 \\ x+2 & \text{s.n.} \end{cases}$

$A(i+1,x+1) = A(i, A(i+1,x))$

(1)	$A(2,2) = A(1, A(2,1))$
(1,1)	$A(2,1) = A(1, A(2,0))$
(1)	$A(2,0) = 1$
(0,1)	$A(1,1) = A(0, A(1,0))$
(1)	$A(1,0) = 1$
(—)	$A(0,1) = 2$
(0)	$A(1,2) = A(0, A(1,1))$
(0,0)	$A(1,1) = A(0, A(1,0))$
(0)	$A(1,0) = 1$
(—)	$A(0,1) = 2$
	$A(0,2) = 4$

Il procedimento si può separare nei seguenti passi:

1. – Si mette l'argomento di sinistra nella pila e si sviluppa l'espressione a destra iterativamente fino ad arrivare ad espressioni del tipo $A(i,0)$ e $A(0,x)$.
2. – Le espressioni $A(i,0)$ e $A(0,x)$ si sanno calcolare immediatamente, per cui, successivamente:
3. – Si prende il valore più in alto della pila e si opera nuovamente come in 1 sulla nuova espressione

Per avere nel nostro programma questo nuovo oggetto che è la pila abbiamo bisogno di due variabili L ed S la prima delle quali ci informa nella lunghezza della pila stessa e la seconda contiene i valori che abbiamo conservato nella pila.

L'operazione di porre q nella pila è rappresentata da:

$L \leftarrow L+1$ (aumenta di 1 la lunghezza)

$S \leftarrow \langle q, S \rangle$ (codifica mediante la $\langle , \rangle$ il nuovo q)

L'operazione inversa di prendere il primo elemento della pila è invece rappresentata da:

$L \leftarrow L-1$ (diminuisci di 1 la lunghezza)

$i \leftarrow \ell(S)$ (prendi l'elemento sinistro di $S = \langle , \rangle$ che è quello voluto)

$S \leftarrow r(S)$ (rimetti in S ciò che resta, cioè il lato destro di $\langle , \rangle$)

PROGRAMMA CHE CALCOLA LA FUNZIONE DI ACKERMANN $A(i,n)$

Definizione $A(i,n) = A(i-1, A(i,n-1))$

con $A(i,0) = 1$

e $A(0,n) = \begin{cases} n+1 & \text{se } n = 0,1 \\ n+2 & \text{se } n > 1 \end{cases}$

```
1.  [A]  IF X≠0 GOTO D
2.       X ← 1
3.  [B]  IF L=0 GOTO G
4.       L ← L-1
5.       i ← ℓ(S)
6.       S ← r(S)
7.       GOTO A
8.  [C]  X ← X+2
9.       GOTO B
10. [D]  IF i≠0 GOTO F
11.      IF X≠1 GOTO C
12.      X ← 2
13.      GOTO B
14. [F]  X ← X-1
15.      L ← L+1
16.      S ← <i-1, S>
17.      GOTO A
18. [G]  Y ← X
```

se $n=0$ calcola $A(i,0) = 1$

Se la pila non è vuota prende il primo valore dalla pila

Calcola $A(0,n) = n+2$ per $n>1$ che è il caso in cui viene attivata..

Calcola $A(0,1)$

Calcola in generale $A(i,n) = A(i-1, A(i,n-1))$

- IL GRUPPO DI ISTRUZIONI 1-2 calcola $A(i,0)=1$ oppure rinvia al GRUPPO [D] E SUCCESSIVE (CHE POTREMMO CHIAMARE "GRUPPO SMISTAMENTO")
- INFATTI, [D] 10-11-12 SELEZIONA I CASI:
  - $(x=0, i\neq 0) \rightarrow$ [F] (calcolo generale)
  - $(x\neq 0, i=0, x\neq 1) \rightarrow$ [C] (calcolo $A(0,x)$ per $x>1$)
  - $(x=1, i=0) \rightarrow$ (calcolo di $A(0,1)$).
- IL GRUPPO [B] 3-4-5-6 prende il primo valore della pila (infatti viene attivato o dopo il calcolo di $A(0,x)$ o dopo quello di $A(0,1)$)
- IL GRUPPO [F] 14-15-16-17 sviluppa, in generale, $A(i,x)$ e, infatti viene attivato solo nel caso generale ($i\neq 0, x\neq 0$)

40bis

Sono possibili quattro casi:

1. $A(i,0)=1$ realizzato dalle istruzioni $1\rightarrow 2\rightarrow 3\rightarrow 18$
2. $A(0,1)=2$ " da $1\rightarrow 10\rightarrow 11\rightarrow 12\rightarrow 13\rightarrow 3\rightarrow 18$
3. $A(0,x)=x+2$ per $x>1$: $A\rightarrow D\rightarrow C$ (se non ci sono niente nella pila)
   $1\rightarrow 10\rightarrow 11\rightarrow 8\rightarrow 9\rightarrow 3\rightarrow 18$
4. $A(i,x)=A(i-1, A(i,x-1))$ con $x, i\neq 0$ oppure $i=0$ e $x>1$

   ciclo: $A\rightarrow D\rightarrow F\rightarrow A$ che calcola tale valore in generale

   ciclo: $A\rightarrow D\rightarrow C\rightarrow B\rightarrow A$ che considera il caso $i=0$ e $x>1$ per cui $A(0,x)=x+2$ e che va a prendere il nuovo valore dalla pila

41.

Osservazioni

1. L'esibizione del programma di $\mathcal{S}$ per calcolare $A(i,x)$ non era necessaria per mostrare che A è calcolabile da un punto di vista intuitivo, perché la definizione stessa mostra in maniera più che sufficiente l'esistenza di un procedimento algoritmico, per quanto complicato, per calcolare i suoi valori.

2. Quindi la dimostrazione dell'insufficienza della ricorsività primitiva a ricoprire il campo della calcolabilità si ha già con la sola dimostrazione della non ricorsività primitiva di $A(i,x)$.

3. La costruzione del programma è invece necessaria per mostrare che $A(i,x)$ è <u>calcolabile</u> in $\mathcal{S}$, cioè è necessaria se si vuole mostrare che $\mathcal{S}$ è sufficientemente potente da calcolare anche questa funzione che <u>non</u> è ricorsiva primitiva.

4. Una volta data <u>una</u> dimostrazione di calcolabilità di $A(i,x)$ in <u>un</u> formalismo non c'è ovviamente alcun bisogno di ripetere la dimostrazione in altri formalismi dimostrabilmente equivalenti.

Limitiamoci adesso ad enunciare, senza dare la dimostrazione, un teorema "inverso" al teorema della limitazione:

TEOREMA

Sia P un programma ciclo che calcola la funzione $g(x_1, \ldots, x_m)$; esista una costante k tale che

$$T_P(x_1, \ldots, x_m) \leq f_n^{(k)}(\max(x_1, \ldots, x_m))$$

con $n \geq 2$.

Allora $g \in \mathcal{L}_n$

LINGUAGGI @CICLO

IN L_1 POSSO STABILIRE SE DUE DIVERSI PROGRAMMI @ALCOLANO LA STESSA FUNZIONE.

IN L_2 (E, A FORTIORI, IN OGNI L_n, con $n>2$) QUESTO NON E' POSSIBILE.

IL LINGUAGGIO WHILE

Una caratteristica positiva del linguaggio LOOP è quella di non avere istruzioni di salto.
Vediamo se è possibile arricchirlo rinunciando però a non perdere il potere espressivo di $\mathcal{S}$.

Aggiungiamo alle istruzioni di L la seguente coppia di istruzioni, che funziona in modo simile a quello della coppia LOOP-END:

```
WHILE V≠0 DO
  ---
  - - - - - -
END
```

(Questo linguaggio L + WHILE sarà chiamato W)

Quindi mentre V≠0 il blocco di istruzioni compreso tra WHILE ed END deve essere ripetuto.
Si osservi che questa volta, a differenza del caso LOOP-END, il valore di V può variare nel corso del calcolo ed eventualmente non divenire mai zero.
In questo caso, quindi, non abbiamo un controllo a priori sul numero di volte che il blocco di istruzioni compreso tra WHILE ed END verrà eseguito.

Si osservi ancora che

il seguente di programma

```
WHILE V≠0 DO
  P
END
```

può essere simulato da

```
[A] IF V=0 GOTO B
  P'
  GOTO A
[B] ————
```

Quindi tutte le funzioni calcolabili da programmi di W sono calcolabili in $\mathcal{S}$.

Richiamiamo adesso un teorema nella rappresentazione delle funzioni ricorsive (non ancora dimostrato) che ci permetterà di dimostrare che vale anche la proprietà inversa.

Ogni funzione parzialmente computabile f di n variabili $x_1, \ldots, x_n$ può essere scritta come:

$$(*) \quad f(x_1,\ldots,x_n) = l(\min_z [g(x_1,\ldots,x_n,z)=0])$$

dove g è una funzione ricorsiva primitiva, l una delle funzioni (ricorsive primitive) coppia e $\min_z$ l'operatore di minimalizzazione non limitata.

TEOREMA: Ogni funzione parzialmente calcolabile può essere calcolata da un programma del linguaggio W.

Dimostrazione

La nostra funzione f sia rappresentata come in (*). Allora essa è calcolata dal seguente programma in W:

```
1.    Z ← 0
2.    V ← g(X1,...,Xn,0)
3.    WHILE V ≠ 0 DO
4.        Z ← Z+1
5.        V ← g(X1,...,Xn,Z)
6.    END
7     Y ← l(Z)
```

Osservazioni

1. Le macro 2, 5, 7 sono ammissibili in L (e quindi in W) perché g ed l sono ricorsive primitive.
2. In base al teorema precedente non solo sono sempre evitabili le istruzioni di salto ma è sufficiente usare una sola istruzione WHILE.

EQUIV. TRA $\mathcal{S}$ e WHILE

Se f è calc. in W allora lo è anche in $\mathcal{S}$.

Simuliamo in $\mathcal{S}$ le istruzioni di W.

Meglio:

- simuliamo <u>solo</u> l'istruzione WHILE (partendo da quella più interna se è nidificata)
- Infatti il sottoprogramma contenuto nel WHILE-END più interno conterrà <u>SOLO</u> istruzioni LOOP e calcolerà quindi funzioni ricorsive primitive (calcolabili in $\mathcal{S}$)

Se f è calcolabile in S allora lo è anche in W

Non possiamo, facilmente, usare la tecnica della simulazione delle singole istruzioni perché l'istruzione di salto si può presentare intrecciata con altre istruzioni di salto in modo da non riuscire a individuare un potenziale blocco di istruzioni da inserire in un ipotetico WHILE-END che potrebbe simulare l'istruzione di salto.

Allora? Sfruttiamo la caratterizzazione delle funzioni S-calcolabili, fornita dal teorema della forma normale.

OSSERVAZIONE

L'istruzione WHILE (in W) può essere sfruttata UNA SOLA VOLTA

DOMANDA

Analogamente, in S, potremmo fare in modo da usare UNA SOLA VOLTA l'istruzione di salto?

RISPOSTA

NO! (perché?)

- W è sufficientemente ricco da potere ancora calcolare le funzioni ricorsive primitive anche se togliamo WHILE

- La forza di S risiede tutta nell'istruzione di salto, se la togliamo rimangono solo le istruzioni di incremento e decremento.