FUNZIONE DI ACKERMANN

Esaminiamo la funzione di Ackermann di due variabili definita da **$A(i, x) = f_i(x)$** che gode delle seguenti proprietà:

- **$A(i+1, x+1) = A(i, A(i+1, x))$**
- **$A(i, 0) = 1$**
- **$A(0, x) = \begin{cases} x+1 & \text{se } x=0 \text{ oppure } x=1 \\ x+2 & \text{se } x>1 \end{cases}$**

TEOREMA: La funzione $A(x, x) = f_x(x)$ non è ricorsiva primitiva.
DIM (per assurdo): Poniamo $A(x, x) = f_x(x) = g(x)$ e supponiamo che g sia ricorsiva primitiva.
Poiché le funzioni ricorsive primitive sono calcolabili da programmi-ciclo, esisterà un m tale che $g \in \mathcal{L}_m$, ma si ha che:

$g \in \mathcal{L}_m \Rightarrow \exists$ k tale che $g(x) \le f_m^{(k)}(x)$	per i teoremi sulla limitazione alla crescita
$f_m^{(k)}(x) < f_{m+1}(x)$	quasi ovunque
$g(x) < f_{m+1}(x)$	quasi ovunque (cioè da un certo valore x in poi)

Dato un intero $n_0 > m+1$ per cui la diseguaglianza è verificata, allora avremo che:
$f_{n0}(n_0) = g(n_0) < f_{m+1}(n_0)$. Ma Poiché $n_0 > m+1$ per il lemma 5 si ha che $f_{n0}(n_0) > f_{m+1}(n_0)$ allora si giunge ad una contraddizione.
Perciò $A(x, x)$ non è ricorsiva primitiva.
Anche $A(x, y)$ non è ricorsiva primitiva, perché se lo fosse stata, allora anche $A(x, x)$ sarebbe stata ricorsiva primitiva.

Procedimento per calcolare A(i, x):

Usiamo una memoria detta "pila" in cui immagazzinare gli argomenti di sinistra, mentre valutiamo quelli di destra. La pila sarà indicata da $(a_1, \dots, a_m)$ in cui a_1 è l'elemento più in alto nella pila e, in generale, a_i sta più in alto di a_{i+1}. Si mette l'argomento di sinistra nella pila e si sviluppa l'espressione a destra iterativamente fino ad arrivare ad espressioni del tipo $A(i, 0)$ o $A(0, x)$ che si sanno calcolare immediatamente. Il valore calcolato sarà usato come argomento di destra, mentre il valore più alto della pila e sarà usato come argomento di sinistra.

Per esempio calcoliamo $A(2, 2)$:

PILA	
	$A(2, 2) = A(1, A(2, 1))$
(1)	$A(2, 1) = A(1, A(2, 0))$
(1, 1)	$A(2, 0) = 1$
(1)	$A(1, 1) = A(0, A(1, 0))$
(0, 1)	$A(1, 0) = 1$
(1)	$A(0, 1) = 2$
///	$A(1, 2) = A(0, A(1, 1))$
(0)	$A(1, 1) = A(0, A(1, 0))$
(0, 0)	$A(1, 0) = 1$
(0)	$A(0, 1) = 2$
///	$A(0, 2) = 4$

Per implementare la pila in questo programma abbiamo bisogno di una variabile L che indica la lunghezza della pila e di una variabile S che contiene i valori che abbiamo conservato nella pila.
L'operazione di inserire un elemento q nella pila è rappresentato da:

$L \leftarrow L + 1$	(aumenta di 1 la lunghezza)
$S \leftarrow \langle q, S\rangle$	(codifica mediante $\langle , \rangle$ il nuovo q)

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

$L \leftarrow L - 1$	(diminuisce di 1 la lunghezza)
$i \leftarrow l(S)$	(prendi l'elemento sinistro di $S = \langle , \rangle$)
$S \leftarrow r(S)$	(rimetti in S ciò che resta, cioè il lato destro di $\langle , \rangle$)

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

A(i, x) =A(i-1, A(i, x-1))	**A(0, x) =**	**x+1**	**se x = {0, 1}**
A(i, 0) = 1		**x+2**	**se x > 1**

1.	[A] IF X ≠ 0 GOTO D	
2.	X ← 1	se x = 0 calcola A(i, 0) = 1
3.	[B] IF L = 0 GOTO G	
4.	L ← L – 1	se la pila non è vuota prende il primo valore dalla pila
5.	i ← l(S)	
6.	S ← r(S)	
7.	GOTO A	
8.	[C] X ← X + 2	calcola A(0, x) = x+2 per x > 1 che è il caso in cui viene attivata
9.	GOTO B	
10.	[D] IF i ≠ 0 GOTO F	
11.	IF X ≠ 1 GOTO C	
12.	X ← 2	calcola A(0, 1)
13.	GOTO B	
14.	[F] X ← X – 1	calcola in generale A(i, x) = A(i-1, A(i, x-1))
15.	L ← L + 1	
16.	S ← <i-1, S>	
17.	GOTO A	
18.	[G] Y ← X	

Sono possibili quattro casi:

1) A(i, 0) = 1
   istruzioni 1→2→3→18
2) A(0, 1) = 2
   istruzioni 1→10→11→12→13→3→18
3) A(0, x) = x+2 per x > 1
   istruzioni 1→10→11→8→9→3→18
4) A(i, x) = A(i-1, A(i, x-1)) con x, i ≠ 0 oppure i=0 e x>1
   ciclo A→D→F→A — che calcola tale valore in generale
   ciclo A→D→C→B→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 della pila

Il gruppo di istruzioni 1 – 2 calcola A(i, 0) = 1 oppure rinvia al gruppo [D] e successive.
Il gruppo [D] 10 – 11 – 12 seleziona i casi con x ≠ 0 e con:
- i ≠ 0 → [F] (calcolo generale)
- i = 0 e x ≠ 1 → [C] (calcolo di A(0, x) per x > 1)
- i = 0 e x = 1 → (calcolo di A(0, 1))

Il gruppo [B] 3 – 4 – 5 – 6 prende il primo valore della pila, infatti viene attivato dopo il calcolo di A(0, x) o A(0, 1).
Il gruppo [F] 14 – 15 – 16 – 17 sviluppa in generale A(i, x), infatti viene attivato solo nel caso in cui i ≠ 0 e x ≠ 0.

Una caratteristica positiva del linguaggio LOOP è quella di non avere istruzioni di salto.
Aggiungiamo alle istruzioni di L la seguente coppia di istruzioni

WHILE V ≠ 0 DO
… … … … … … …
END

che funziona in modo simile alla coppia LOOP – END.
Mentre V ≠ 0 il blocco di istruzioni compreso fra WHILE ed END deve essere ripetuto.
A differenza del caso LOOP – END, il valore di v può variare nel corso del calcolo ed eventualmente non divenire mai zero. In questo modo non abbiamo un controllo a priori nel numero di volte che il blocco di istruzioni compreso fra WHILE ed END verrà eseguito.
Questo linguaggio L + WHILE sarà chiamato W. Tutte le funzioni calcolabili da programmi di W sono anche calcolabili in S:

WHILE V ≠ 0 DO		[A] IF V = 0 GOTO B
P	equivale a	P
END		GOTO A
		[B] ………..

TEOREMA: Ogni funzione f parzialmente calcolabile può essere calcolata da un programma del linguaggio W.
DIM: Assumiamo che f sia una funzione di n variabili $x_1, \dots, x_n$ che può essere scritta come
$f(x_1, \dots, x_n) = l(\min_z[g(x_1, \dots, x_n, z) = 0])$ dove l è una delle funzioni coppia (ricorsiva primitiva), $\min_z$ è l'operatore di minimalizzazione non limitata e g è una funzione ricorsiva primitiva.
La funzione f è calcolata da seguente programma:

1. $Z \leftarrow 0$
2. $V \leftarrow g(X_1, \dots, X_n, 0)$
3. WHILE V ≠ 0 DO
4. $Z \leftarrow Z+1$
5. $V \leftarrow g(X_1, \dots, X_n, Z)$
6. END
7. $Y \leftarrow l(Z)$

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