PROGRAMMAZIONE FUNZIONALE

OPERATORI

- **Memoria**
  $P_{i,j}^{n}: \mathbb{N}^n \rightarrow \mathbb{N}^{j-i+1}$
  Significato: data una memoria di n valori, ritorno una memoria dall'indice i all'indice j.
  Esempio: $P_{1,1}^{2}$ da una memoria di 2 valori, ritorno una memoria di valori dall'indice 1 all'indice 1.
  Esempio: P^{3} da una memoria di 3 valori, ritorno una memoria vuota.
  Esempio: $P_{2,4}^{5}$ da una memoria di 5 valori, ritorno una memoria di valori dall'indice 2 all'indice 4.
- **Flusso**
  $f; g \qquad f: \mathbb{N}^m \rightarrow \mathbb{N}^n \qquad g: \mathbb{N}^n \rightarrow \mathbb{N}^o$
  Significato: l'output della funzione f diventa l'input della funzione g.
  Esempio: $P_{1,2}^{2}; +$ la memoria di 2 elementi diventa l'input della funzione +.
- **Concatenazione**
  $f \wedge g: \mathbb{N}^m \rightarrow \mathbb{N}^{o+p} \qquad f: \mathbb{N}^m \rightarrow \mathbb{N}^o \qquad g: \mathbb{N}^m \rightarrow \mathbb{N}^p$
  Significato: l'input viene dato sia alla funzione f che alla funzione g. L'output delle due funzioni viene concatenato tra di loro.
  Esempio: $P_{1,2}^{2}; + \wedge *$ la memoria di 2 elementi diventa l'input delle funzioni + e *, il cui risultato viene concatenato.
- **Parallelo**
  $f \parallel g: \mathbb{N}^{m+n} \rightarrow \mathbb{N}^{o+p} \quad f: \mathbb{N}^m \rightarrow \mathbb{N}^o \qquad g: \mathbb{N}^n \rightarrow \mathbb{N}^p$
  Significato: i primi m input vengono dati alla funzione f, i restanti vengono dati alla funzione g. Il risultato delle due funzioni viene concatenato.
  Esempio: $P_{1,1}^{3} \parallel +$ dalla memoria di 3 elementi viene estratto il primo valore, il quale verrà concatenato al risultato della funzione + che riceve i restanti input della memoria (ovvero i valori di indice 2 e 3).
- **Esponenziale**
  $\exp(f)$
  Significato: data una funzione f che vuole x input, la memoria viene letta così: i primi x valori della memoria sono gli input di f, il valore x+1 è quante volte deve ciclare la funzione f.
  Esempio: $P_{1,2}^{2}; \exp(+ \wedge 5)$ dalla memoria di 2 elementi viene sommato 5 al valore di indice 1 tante volte quanto è il valore della memoria di indice 2.

FUNZIONI

- S
  Numero input: 1
  Numero output: 1
  Funzione successore. Ritorna il valore di input aumentato di 1.
  Sempre presente.
- +
  Numero input: 2
  Numero output: 1
  Funzione somma. Ritorna la somma dei due valori di input.
  Non sempre presente.
- *
  Numero input: 2
  Numero output: 1
  Funzione prodotto. Ritorna il prodotto dei due valori di input.
  Non sempre presente.
- =
  Numero input: 2
  Numero output: 1
  Funzione equivalenza. Ritorna 1 se i due valori sono uguale, 0 se sono diversi.
  Non sempre presente.
- sg
  Numero input: 1
  Numero output: 1
  Funzione segno. Ritorna 0 se l'input è 0, 1 altrimenti.
  Non sempre presente.
- $\overline{sg}$
  Numero input: 1
  Numero output: 1
  Funzione segno negato. Ritorna 1 se l'input è 0, 0 altrimenti.
  Non sempre presente.

ESEMPIO

Data una funzione f(x) tale che ritorna 2x se x = 1, altrimenti ritorna 3x. Sono dati come operatori utilizzabili: somma, prodotto, equivalenza, segno e segno negato.

$$f(x) = \begin{cases} 2 & se\ x = 1 \\ 3x & se\ x \neq 1 \end{cases}$$

METODO 1 (Borsato)

Definisco due funzioni t(x) e nt(x)

$$t(x) = 2 \qquad nt(x) = 3x$$

Per prima cosa devo decidere se x è 1 o no, quindi per prima cosa effettuerò l'operazione di equivalenza con la costante 1. Dunque per effettuare questa cosa dovrò riempire la memoria con il valore di input, accodarci la costante 1 e poi applicare l'operatore di equivalenza:

Stato iniziale della memoria	$P_{1,1}^1 \parallel 1 \longrightarrow$	Stato della memoria dopo il concatenamento della costante 1		$= \longrightarrow$	Stato della memoria dopo l'uso dell'operatore equivalenza
x		x	1		=(x, 1)

Però così perdiamo il valore della x… che però ci serve per il caso 3x. Vediamo dunque di sistemare la cosa con l'uso dell'operatore concatenazione (^) il quale ci permetterà di mantenere una copia del valore di x.

Stato iniziale della memoria	$P_{1,1}^1 \text{ ^ } ((P_{1,1}^1 \parallel 1);=) \longrightarrow$	Stato della memoria dopo l'operazione di concatenazione ed equivalenza.	
x		x	=(x, 1)

Come possiamo vedere ora nella memoria abbiamo sia il valore di x, che il valore della sua eguaglianza. Dobbiamo ora decidere a partire da questo dato le due braccia dell'if. Per fare questo sfrutteremo la funzione segno, segno negato ed esponenziale. L'esponenziale lo useremo per ciclare tante volte quanto è il risultato delle funzioni segno e segno negato applicato al risultato dell'equivalenza, in questa maniera ciclerà 0 volte per il caso che noi non vogliamo, e 1 volta (e quindi ci darà il risultato voluto) per il caso che vogliamo.

Per prima cosa dovremmo quindi concatenare a x le funzioni segno e segno negato. Lo faremo così:

Stato della memoria per ottenere i due segni	$(\overline{sg}(P_{1,1}^1) \text{ ^ } sg(P_{1,1}^1)) \longrightarrow$	Stato della memoria dopo l'operazione di concatenazione e segno	
=(x, 1)		$\overline{sg}$(=(x, 1))	sg(=(x, 1))

Ovviamente però a noi interessa mantenere la x per i calcoli futuri, dunque useremo l'operatore || per ottenere questo risultato:

Stato della memoria iniziale		$P_{1,1}^2 \parallel (\overline{sg}(P_{1,1}^1) \wedge sg(P_{1,1}^1)) \longrightarrow$	Stato della memoria dopo le operazioni di applicazioni di segno e segno negato		
x	=(x, 1)		X	$\overline{sg}$(=(x, 1))	sg(=(x, 1))

Ora che abbiamo i valori dei due segni con l'uso dell'esponenzializzazione è un gioco da ragazzi calcolare il risultato con le funzioni t e nt. Innanzitutto vediamo che quando x = 1, allora segno negato da come risultato 0, dunque vuol dire che itererà la funzione t() 1 volta, e la funzione nt() 0 volte.

Al contrario, se invece segno negato vale 1, allora itererà 0 volte la funzione t(), e 1 volta la funzione nt(). In entrambi i casi applicheremo il seguente metodo: prima proveremo a iterare nt(), poi proveremo con t().

Dunque vediamo il primo caso, quando x = 1:

Stato della memoria iniziale			$(P_{1,1}^1 ; \exp(nt)) \parallel \exp(t) \longrightarrow$	Stato della memoria se x uguale a 1
X	$\overline{sg}$(=(x, 1))	sg(=(x, 1))		2

E il caso quando x non è uguale a 1 :

Stato della memoria iniziale			$(P_{1,1}^1 ; \exp(nt)) \parallel \exp(t) \longrightarrow$	Stato della memoria se x diverso da 1
X	$\overline{sg}$(=(x, 1))	sg(=(x, 1))		3x

In finale quindi il nostro programma funzionale sarà scritto così:

$$P_{1,1}^1 \wedge \left((P_{1,1}^1 \parallel 1); = \right) ; P_{1,1}^2 \parallel \left(\overline{sg}(P_{1,1}^1) \wedge sg(P_{1,1}^1) \right) ; (P_{1,1}^1 ; \exp(nt)) \parallel \exp(t)$$

I passaggi di modifica della memoria del nostro programma saranno quindi i seguenti:

$$x \mapsto x, 1 \mapsto x, = 1 \mapsto x, \overline{sg}(= 1), sg(= 1) \begin{matrix} \nearrow sg \mapsto 2 \\ \searrow 3x, sg \mapsto 3x \end{matrix}$$

Esempio (input = 1):

$$1 \mapsto 1, 1 \mapsto 1, 1 \mapsto 1, 0, 1 \mapsto 1 \mapsto 2$$

Esempio (input = 3):

$$3 \mapsto 3, 1 \mapsto 3, 0 \mapsto 3, 1, 0 \mapsto 9, 0 \mapsto 9$$

METODO 2 (Longiarù)

Definisco due funzioni f1(x) e f2(x)

$$f1(x) = x + 1 \qquad f2(x) = 3x$$

Con questo metodo invece di iterare per decidere il risultato della funzione, useremo il prodotto e la somma. Ovvero, annulleremo uno dei due valori col prodotto e otterremo la soluzione mediante una somma.

Come prima abbiamo bisogno inizialmente di una coppia di valori: x e il suo risultato con l'eguaglianza ad 1.

Stato iniziale della memoria	$\xrightarrow{P^1_{1,1} \text{ ^ } (\ (\ P^1_{1,1} \parallel 1\ );=)}$	Stato della memoria dopo l'operazione di concatenazione ed equivalenza.	
x		x	=(x, 1)

Dopo di questo ci serve ottenere i due risultati che le funzioni f1() e f2() darebbero. Otterremo una coppia di valori che poi moltiplicheremo tra loro

Vediamo prima il codice funzionale per f1():

Stato iniziale della memoria		$\xrightarrow{(P^1_{1,1}\ ;\ f1) \parallel P^1_{1,1}}$	Stato della memoria dopo l'operazione di parallelo ed equivalenza.		$\xrightarrow{*}$	Stato della memoria dopo il prodotto
x	=(x, 1)		f1(x)	=(x, 1)		=(x, 1)*f1(x)

E ora il codice per la funzione f2(), la quale però applicherà la funzione segno negato sul risultato dell'equivalenza:

Stato iniziale della memoria		$\xrightarrow{(P^1_{1,1}\ ;\ f2) \parallel (P^1_{1,1}\ ;\ \overline{sg}\ )}$	Stato della memoria dopo l'operazione di parallelo ed equivalenza.		$\xrightarrow{*}$	Stato della memoria dopo il prodotto
x	=(x, 1)		f2(x)	$\overline{sg}$(=(x, 1))		$\overline{sg}$(=(x, 1))*f2(x)

Dai risultati delle due funzioni applicheremo il prodotto ottenendo il risultato voluto.

Stato iniziale della memoria	$\xrightarrow{\left(\left((P^1_{1,1}\ ;\ f1) \parallel P^1_{1,1}\right)\ ;*\right) \text{ ^ } \left(\left((P^1_{1,1}\ ;\ f2) \parallel (P^1_{1,1}\ ;\ \overline{sg}\ )\right)\ ;*\right)}$	Stato della memoria dopo il calcolo di f1 e f2	$\xrightarrow{+}$	Stato finale
x		=(x,1)*f1(x)		ris*
=(x, 1)		$\overline{sg}$(=(x,1))*f2(x)		

*ris = risultato finale di f(x)

In finale quindi il nostro programma sarà scritto così:

$$P_{1,1}^1 \text{ ^ } \left(\left(P_{1,1}^1 \parallel 1 \right); = \right) ; \left(\left(\left(P_{1,1}^1 ; f1 \right) \parallel P_{1,1}^1 \right) ; * \right) \text{ ^ } \left(\left(\left(P_{1,1}^1 ; f2 \right) \parallel \left(P_{1,1}^1 ; \overline{sg} \right) \right) ; * \right) ; +$$

I passaggi di modifica della memoria del nostro programma saranno quindi i seguenti:

$$x \mapsto x,1 \mapsto x,=1 \mapsto \left(=1 * f1(x)\right) + \left(\overline{sg}(=1) * f2(x)\right)$$

Esempio (x = 1):

$$1 \mapsto 1,1 \mapsto 1,1 \mapsto \left(1 * (1+1)\right) + \left(0 * (3*1)\right) \mapsto (1*2) + (0*3) \mapsto 2 + 0 \mapsto 2$$

Esempio (x = 3):

$$3 \mapsto 3,1 \mapsto 3,0 \mapsto \left(0 * (3+1)\right) + \left(1 * (3*3)\right) \mapsto (0*4) + (1*9) \mapsto 0 + 9 \mapsto 9$$

ALTRI ESEMPI

DEFINIRE LA SOMMA

L'operatore somma non è sempre definito. Per definirlo useremo l'esponenzializzazione e la funzione successore.

Stato della memoria iniziale		$\exp(S)$ $\longrightarrow$	Stato della memoria dopo la somma
a	b		a + b

Come potete vedere la somma non è altro che un ciclo iterativo dove il successore viene applicato su a per b volte.

Dunque possiamo definire la somma così:

$$+ =_{def} \exp(S)$$

Esempio (0 + 5):

$$0,5 \mapsto 1 \mapsto 2 \mapsto 3 \mapsto 4 \mapsto 5$$

Esempio (5 + 2):

$$5,2 \mapsto 6 \mapsto 7$$

DEFINIRE IL PRODOTTO

L'operatore prodotto può essere definito sfruttando l'operatore somma (che abbiamo definito qui sopra) e l'esponenzializzazione per il numero di volte che deve iterare

Sappiamo che il prodotto di un valore per 0 da 0, dobbiamo dunque sfruttare questa informazione a nostro vantaggio con l'aiuto dell'esponenzializzazione, che vedremo ora:

Stato iniziale della memoria		$0 \parallel P^2_{1,2} \longrightarrow$	Stato finale della memoria		
p	q		0	p	q

L'aggiunta dello 0 è importante per il seguente motivo: è il valore minimo che il prodotto da come risultato. Questo significa che il risultato del prodotto sarà sempre presente nella prima cella della nostra area di memoria.

p e q invece verranno usati rispettivamente così: p come addendo da aggiungere a 0 ad ogni passo iterativo e q sarà il numero di iterazioni. Questo ci porta quindi al prossimo passo: l'uso dell'operatore + insieme all'esponenzializzazione:

Valori iniziali nella memoria prima dell'esponenzializzazione			$\exp(+ \text{ ^ } P^2_{2,2}) \longrightarrow$	Valori presenti nella memoria dopo l'esponenzializzazione		$P^2_{1,1} \longrightarrow$	Stato finale della memoria
0	p	q		p * q	q		p * q

L'esponenzializzazione come vedete itera applicando la funzione somma ai primi due valori della memoria, ritorna 1 risultato, al quale viene collegato il secondo valore presente in memoria.

Si noti il fatto che P è sopraelevato a 2. Questo perché la memoria nel momento in cui viene avviata exp è già ridotta di un valore (q è il numero di iterazioni), rimangono solo 0 e p, che sono i valori da dare come input alla funzione da iterare; però succederebbe che dopo la prima iterazioni avremmo una memoria composta di un unico valore, ovvero il risultato della prima iterazione di exp, che non è sufficiente come numero di input per l'operatore +, per questo con l'operatore ^ colleghiamo al risultato di ogni iterazioni di exp il secondo valore della memoria.

Quando finiscono le iterazioni avremmo il risultato desiderato nella prima cella di memoria.

Il prodotto è quindi definibile così:

$$* =_{def} \left(0 \parallel P^2_{1,2}\right); \exp(+ \text{ ^ } P^2_{2,2}); P^2_{1,1}$$

Esempio (2 * 3):

$$2,3 \mapsto 0,2,3 \mapsto 0,2 \mapsto 2,2 \mapsto 4,2 \mapsto 6,2 \mapsto 6$$

Esempio (5 * 1):

$$5,1 \mapsto 0,5,1 \mapsto 0,5 \mapsto 5,5 \mapsto 5$$

Esempio (0 * 4):

$$0,3 \mapsto 0,0,3 \mapsto 0,0 \mapsto 0,0 \mapsto 0,0 \mapsto 0,0 \mapsto 0$$

DEFINIRE LA SOTTRAZIONE

Per definire la sottrazione prima dobbiamo definire la funzione P, ovvero la funzione precedente. La definiremo in maniera simile a quella del prodotto, ovvero useremo l'esponenzializzazione, in altre parole

itereremo tante volte quanto è il numero da diminuire di 1 valore, e tenendoci un contatore del suo valore precedente, torneremo poi quello.

Per prima cosa aggiungiamo nella nostra memoria i due valori che faranno da contatori:

Stato della memoria iniziale	$0 \parallel 0 \parallel P^1_{1,1} \longrightarrow$	Stato della memoria dopo l'aggiunta delle aree di memoria		
x		0	0	x

A questo punto possiamo applicare l'esponenzializzazione, la quale aumenterà di un unità il valore in prima posizione, rimuoverà il valore in seconda posizione e ci attaccherà in coda il valore originale del numero in prima posizione, ed infine estrae il nostro risultato che si troverà nella seconda cella di memoria:

Valori iniziali nella memoria prima dell'esponenzializzazione			$\exp((S \parallel P^1) \wedge P^2_{1,1}) \longrightarrow$	Valori presenti nella memoria dopo l'esponenzializzazione		$P^2_{2,2} \longrightarrow$	Stato finale della memoria
0	0	x		x	x-1		x-1

In questo modo abbiamo definito la funzione precedente:

$$p(x) = \left(0 \parallel 0 \parallel P^1_{1,1}\right); \exp\left((S \parallel P^1) \wedge P^2_{1,1}\right); P^2_{2,2}$$

La sottrazione è adesso facile da definire: si tratta solo di iterare tante volte quanto è il valore del secondo input dell'operatore. E' quindi definibile così:

$$-=_{def} \exp\left(p\left(P^1_{1,1}\right)\right)$$

Si ricorda che con questa definizione che la sottrazione tra 0 e 1 da come risultato 0, visto che lavoriamo solo sui numeri naturali.

Esempio (5 – 3):

$5,2 \mapsto$

$0,0,5 \mapsto 0,0 \mapsto 1,0 \mapsto 2,1 \mapsto 3,2 \mapsto 4,3 \mapsto 5,4 \mapsto 4$

$\mapsto 3$

$0,0,4 \mapsto 0,0 \mapsto 1,0 \mapsto 2,1 \mapsto 3,2 \mapsto 4,3$

Esempio (1 – 1):

$1,1 \mapsto \mapsto 0$

$0,0,1 \mapsto 0,0 \mapsto 1,0$

Esempio (0 – 1):

$0,1 \mapsto \mapsto 0$

$0,0,0 \mapsto 0$