

# Progetto di Programmazione per Aprile 2015

## Funzioni ricorsive

Questo progetto è valido per l'appello di Aprile 2015 del corso di "Programmazione (con laboratorio)"; le date e le modalità di consegna sono indicate sulla pagina web del corso.

### 1 Funzioni ricorsive

Limitatamente al testo di questo progetto, chiameremo *funzione* una qualunque funzione  $f : \mathbf{N}^n \rightarrow \mathbf{N}^m$ , dove  $\mathbf{N} = \{0, 1, 2, \dots\}$  è l'insieme dei numeri naturali, e  $n, m \geq 0$ ; ammetteremo che tale funzione possa essere indefinita su alcuni input (cioè, che si tratti di una "funzione parziale"). I valori  $n$  e  $m$  si chiamano, rispettivamente, *fan-in* e *fan-out* della funzione; notate che possono essere entrambi 0. In particolare, una funzione con fan-in 0 è un vettore costante (e, se il fan-out è 1, è un numero).

Consideriamo le seguenti funzioni ricorsive (che chiameremo "funzioni di base"):

- **Funzione costante  $K_k$ :** per ogni intero  $k \in \mathbf{N}$ , la funzione  $K_k : \mathbf{N}^0 \rightarrow \mathbf{N}^1$  che ha valore costante uguale a  $k$ .
- **Proiezione  $\Pi_i^n$ :** per ogni intero  $n \in \mathbf{N}$  e per ogni  $i = 0, 1, \dots, n-1$ , la funzione  $\Pi_i^n : \mathbf{N}^n \rightarrow \mathbf{N}^1$  che mappa il vettore  $(x_0, x_1, \dots, x_{n-1})$  in  $x_i$ .
- **Multiplexing  $\nabla_t^n$ :** per ogni coppia di interi  $n, t \in \mathbf{N}$ , la funzione  $\nabla_t^n : \mathbf{N}^n \rightarrow \mathbf{N}^{nt}$  che mappa il vettore  $(x_0, x_1, \dots, x_{n-1})$  nel vettore  $(x_0, \dots, x_{n-1}, x_0, \dots, x_{n-1}, \dots)$  ( $t$  volte). Notate, in particolare, che  $\nabla_0^n$  è una funzione che "dimentica" il suo input, mentre  $\nabla_1^n$  è l'identità.
- **Successore  $S$ :** è la funzione  $S : \mathbf{N}^1 \rightarrow \mathbf{N}^1$  che mappa ogni intero  $k$  nell'intero  $k+1$ .

La figura 1 mostra graficamente alcune funzioni di base.

Consideriamo ora il seguente insieme di operatori che permettono di ottenere funzioni nuove da funzioni date:

- **Composizione sequenziale  $\circ$ :** data una funzione  $f : \mathbf{N}^n \rightarrow \mathbf{N}^m$  e una funzione  $g : \mathbf{N}^m \rightarrow \mathbf{N}^p$ , indichiamo  $g \circ f : \mathbf{N}^n \rightarrow \mathbf{N}^p$  la funzione ottenuta applicando prima  $f$  e poi, all'output così ottenuto,  $g$ .

- **Composizione parallela**  $\parallel$ : date  $k$  funzioni  $f_1, \dots, f_k$ , con fan-in  $i_1, \dots, i_k$  e fan-out  $o_1, \dots, o_k$ , la funzione  $f_1 \parallel f_2 \parallel \dots \parallel f_k : \mathbf{N}^{i_1 + \dots + i_k} \rightarrow \mathbf{N}^{o_1 + \dots + o_k}$  è definita come segue: dato un vettore di dimensione  $i_1 + \dots + i_k$ , lo si divide in  $k$  vettori di dimensioni  $i_1, \dots, i_k$ , rispettivamente, e al  $t$ -esimo vettore così ottenuto si applica la funzione  $f_t$ , ottenendo un vettore di dimensione  $o_t$ . A questo punto si prendono i  $k$  vettori ottenuti, di dimensioni  $o_1, \dots, o_k$ , e si concatenano, ottenendo un unico vettore di dimensione  $o_1 + \dots + o_k$ .
- **Ricorsione primitiva**  $R(-, -)$ : sia data una funzione  $h : \mathbf{N}^{1+m+n} \rightarrow \mathbf{N}^m$  e una funzione  $g : \mathbf{N}^n \rightarrow \mathbf{N}^m$ . Definiamo la funzione  $f = R(h, g)$  come una funzione  $f : \mathbf{N}^{1+n} \rightarrow \mathbf{N}^m$  come segue:

$$f(y, x_0, \dots, x_{n-1}) = \begin{cases} g(x_0, \dots, x_{n-1}) & \text{se } y = 0 \\ h(y-1, f(y-1, x_1, \dots, x_{n-1}), x_0, \dots, x_{n-1}) & \text{altrimenti.} \end{cases}$$

- **$\mu$ -ricorsione**  $\mu$ : data una funzione  $f : \mathbf{N}^{1+n} \rightarrow \mathbf{N}$ , definiamo una funzione  $\mu f : \mathbf{N}^n \rightarrow \mathbf{N}$  dove  $\mu f(x_0, \dots, x_{n-1})$  è il più piccolo intero  $y$  tale che  $f(y, x_0, \dots, x_{n-1}) = 0$ , se esso esiste, oppure non è definita nel caso tale intero non esista.

La figura 2 mostra graficamente alcuni operatori. L'operatore di ricorsione primitiva può essere pensato in questi termini: supponete di voler calcolare  $f(y, x_0, \dots, x_{n-1})$ , dove  $f = R(h, g)$ . Costruite una "pila" costituita da  $y - 1$  copie di  $h$ : ogni copia di  $h$  riceve in input il suo livello di appartenenza ( $y$  per l'istanza che si trova più in basso nella pila,  $y - 1$  per quella che la precede ecc., 1 per la prima), l'output prodotto dalla parte della pila che si trova sopra di essa, e una copia degli input  $x_0, \dots, x_{n-1}$ . Sopra la pila di  $h$  si trova una copia di  $g$  che prende gli input  $x_0, \dots, x_{n-1}$  e fornisce l'output al livello più alto della pila.

Una funzione che possa essere ottenuta a partire dalle funzioni di base applicando gli operatori indicati viene chiamata *ricorsiva* (e, in particolare, *primitiva ricorsiva* se non si usa mai la  $\mu$ -ricorsione; in tal caso, la funzione è totale, cioè è sempre definita, perché l'unico operatore che può introdurre funzioni parzialmente definite è la  $\mu$ -ricorsione).

Si può dimostrare che le funzioni calcolabili effettivamente (cioè, calcolabili mediante un algoritmo) sono, a meno di codifiche dell'input e dell'output, tutte e sole le funzioni ricorsive (il fatto che una funzione non sia definita per alcuni input significa che il programma che la calcola su alcuni input entra in loop infinito).

**Esempio 1.** Considerate la funzione

$$f = (\Pi_1^2 \parallel \Pi_0^2) \circ \nabla_2^2.$$

Il fan-in di questa funzione è 2, il fan-out è 2, e

$$f(x, y) = (\Pi_1^2 \parallel \Pi_0^2)(\nabla_2^2(x, y)) = (\Pi_1^2 \parallel \Pi_0^2)(x, y, x, y) = (y, x).$$

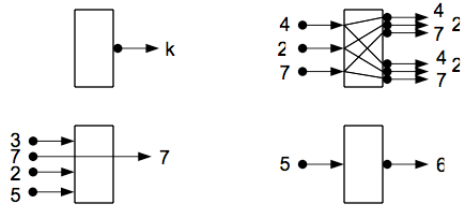


Figura 1: Funzioni di base. Da sinistra a destra, dall'alto in basso: la funzione costante  $K_k$ , il multiplexer  $\nabla_2^3$  (su input  $(4, 2, 7)$ ), la proiezione  $\Pi_1^4$  (su input  $(3, 7, 2, 5)$ ) e il successore  $S$  (su input 5).

Quindi, questa funzione rappresenta lo switch (la funzione che mappa il vettore  $(x, y)$  nel vettore  $(y, x)$ ).

**Esempio 2.** Considerate la funzione

$$f = R(S \circ \Pi_1^3, \Pi_0^1).$$

Dimostrate che ha fan-in 2, fan-out 1 e  $f(x, y) = x + y$ .

**Esempio 3.** Considerate la funzione

$$f = R(K_0 \circ \nabla_0^2, K_1).$$

Dimostrate che ha fan-in e fan-out 1, e che  $f(0) = 1$  mentre  $f(x) = 0$  per ogni  $x \neq 0$ . In altri termini,  $f$  è la funzione che controlla se il suo input è zero; se interpretate, come in C, 0 come “false” e non-zero come “true”, questa è la funzione not.

**Esercizio.** Scrivete le seguenti funzioni:

- **or:**  $f : \mathbf{N}^2 \rightarrow \mathbf{N}^1$  dove  $f(x, y)$  vale 0 o 1, e vale 0 solo se  $x = y = 0$ ;
- **and:**  $f : \mathbf{N}^2 \rightarrow \mathbf{N}^1$  dove  $f(x, y)$  vale 0 o 1, e vale 1 solo se  $x \neq 0$  e  $y \neq 0$ ;
- **if:**  $f : \mathbf{N}^3 \rightarrow \mathbf{N}^1$  dove  $f(0, x, y) = y$  e  $f(z, x, y) = x$  per ogni  $z \neq 0$ ;
- **predecessore:**  $f : \mathbf{N}^1 \rightarrow \mathbf{N}^1$  dove  $f(x) = x - 1$  se  $x > 0$ , e  $f(0) = 0$ ;
- **differenza:**  $f : \mathbf{N}^2 \rightarrow \mathbf{N}^1$  dove  $f(x, y) = x - y$  se  $x \geq y$ , e  $f(x, y) = 0$  se  $x < y$ ;
- **prodotto:**  $f : \mathbf{N}^2 \rightarrow \mathbf{N}^1$  dove  $f(x, y) = xy$ ;
- **divisione intera:**  $f : \mathbf{N}^2 \rightarrow \mathbf{N}^1$  dove  $f(x, y) = \lfloor x/y \rfloor$  per ogni  $y > 0$ .

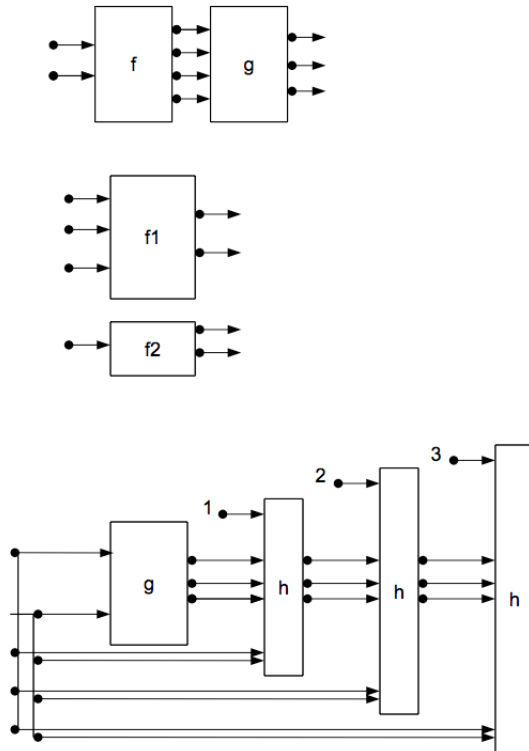


Figura 2: Operatori. Dall'alto in basso: esempio di composizione sequenziale  $g \circ f$  (qui  $f$  ha fan-in 2 e fan-out 4, e  $g$  ha fan-in 4 e fan-out 3, quindi  $g \circ f$  ha fan-in 2 e fan-out 3), di composizione parallela  $f1 || f2$  (dove  $f1$  ha fan-in 3 e fan-out 2, e  $f2$  ha fan-in 1 e fan-out 2, quindi  $f1 || f2$  ha fan-in 4 e fan-out 4), e di ricorsione primitiva: qui  $h$  ha fan-in 6,  $g$  ha fan-in 2 e entrambe hanno fan-out 3; stiamo assumendo di voler valutare  $f = R(h, g)$  in  $(3, x_0, x_1)$ ; i due input  $x_0$  e  $x_1$  sono forniti sia a  $g$  che a ciascuna delle  $h$ , mentre a ciascuna  $h$  è anche fornito in input il "livello" di appartenenza (1, 2, 3).

## 2 Implementazione Java

Realizzate un insieme di classi Java che consentano di fare esperimenti con le funzioni ricorsive. In particolare, dovrete definire una interfaccia `Function` che prevede un metodo che restituisca il fan-in, un metodo che restituisca il fan-out, e un metodo `int[] apply(int[] x)` che corrisponde ad applicare la funzione a un dato vettore. Quest'ultimo metodo deve sollevare una `IllegalArgumentException` se `x` non ha lunghezza pari al fan-in, e altrimenti deve restituire un vettore di lunghezza pari al fan-out.

Le varie funzioni di base saranno classi che implementano l'interfaccia `Function` (prenderanno nel loro costruttore gli eventuali parametri). Così pure gli operatori di composizione.

Realizzate una classe, di nome `Functions`, che contenga solo campi statici finali di tipo `Function` che corrispondono a varie funzioni. Ad esempio, la funzione `switch` descritta nell'esempio 1 della sezione precedente (la cui formula era  $(\Pi_1^2 \parallel \Pi_0^2) \circ \nabla_2^2$ ) potrebbe essere definita così<sup>1</sup>:

```
public final static Function SWITCH =
    new SequentialComposition(
        new Parallel(new Function[] {new Projection(2, 1), new Projection(2, 0)}),
        new Multiplex(2, 2)
    );
```

## 3 Estensioni

Il progetto è volutamente specificato in modo minimale, poiché le consegne saranno valutate, oltre che in base alla correttezza della realizzazione, anche sulla base del numero di funzionalità aggiuntive introdotte. Ecco due esempi.

**Gödelizzazione.** Fissato un  $n > 0$ , una funzione di *gödelizzazione delle n-tuple* è una qualunque funzione biettiva  $f : \mathbf{N}^n \rightarrow \mathbf{N}^1$ . Ad esempio, per  $n = 2$  un'esempio di funzione di gödelizzazione è rappresentato dalla cosiddetta *funzione coppia di Cantor*  $\langle -, - \rangle$ :

	0	1	2	3	4	...
0	0	1	3	6	10	
1	2	4	7	11		
2	5	8				
3	9					

che mappa  $\langle 0, 0 \rangle = 0$ ,  $\langle 0, 1 \rangle = 1$ ,  $\langle 1, 0 \rangle = 2$  ecc. Questa si può generalizzare a vettori di lunghezza maggiore di 2, ponendo  $\langle x_1, x_2, \dots, x_n \rangle = \langle x_1, \langle x_2, \dots, x_n \rangle \rangle$ .

Naturalmente, ogni funzione di gödelizzazione  $f : \mathbf{N}^n \rightarrow \mathbf{N}^1$  ammette una funzione inversa di *degödelizzazione*  $f^{-1} : \mathbf{N}^1 \rightarrow \mathbf{N}^n$ .

<sup>1</sup>La sintassi `new X[] { ... }` permette di creare al volo un array di tipo `X` i cui elementi sono quelli specificati fra parentesi graffe.

Potreste definire un'interfaccia per le funzioni di gödelizzazione e degödelizzazione, di cui la funzione di Cantor sarebbe un'implementazione. Fatto questo, considerate la seguente domanda: date due Function  $f$  e  $g$  con lo stesso fan-in  $n$  e lo stesso fan-out  $m$ , data una funzione di gödelizzazione  $[-, \dots, -] : \mathbf{N}^n \rightarrow \mathbf{N}^1$ , e dato un  $k$ , è vero che le due funzioni coincidono sulle prime  $k$   $n$ -ple? Cioè, è vero che, per ogni  $i = 0, 1, \dots, k - 1$  si ha  $f([i]^{-1}) = g([i]^{-1})$ ?

Implementate un metodo che risponda a questa domanda. Fate attenzione, nella documentazione del metodo, a specificare cosa succede se  $f$  o  $g$  non sono funzioni totali (cioè, se c'è qualche input su cui non sono definite).

**Rappresentazione grafica.** Data una funzione  $f : \mathbf{N}^1 \rightarrow \mathbf{N}^1$  e dato un intervallo di naturali  $[a, b] = \{a, a + 1, \dots, b\}$ , considerate i valori assunti da  $f$  sugli elementi dell'intervallo. Se  $M$  è il massimo valore assunto da  $f$ , potete rappresentare la funzione come una matrice di caratteri con  $M + 1$  righe (numerate dal basso in alto, da 0 a  $M$ ) e con  $b - a + 1$  colonne (numerate da sinistra a destra, da  $a$  a  $b$ ) in cui la matrice è riempita con spazi, tranne gli elementi della forma  $(i, f(i))$  (per  $i = a, \dots, b$ ) che sono riempiti con asterischi. Alternativamente, invece di una matrice, potete usare una stringa ottenuta concatenando uno dopo l'altro i caratteri della matrice (dall'alto in basso, da sinistra a destra) e mettendo degli a-capo alla fine di ogni riga.

Questo si può generalizzare per funzioni della forma  $f : \mathbf{N}^1 \rightarrow \mathbf{N}^n$ , scegliendo  $n$  diversi caratteri per ognuno degli indici in output (e stabilendo cosa fare quando, per qualche input  $i$ , il vettore  $f(i)$  risulta avere delle ripetizioni).

Implementate un metodo `toString(int a, int b)` che consenta di avere il grafico di una data funzione con fan-in 1 nell'intervallo  $[a, b]$ .