1 Il mini linguaggio funzionale

Le caratteristiche principali della programmazione funzionale sono:

- **Non ci sono assegnamenti**. Una variabile non rappresenta una cella di memoria, ma un valore e tale valore non cambia mai.
- Le strutture dati sono *persistenti*: una volta create non si modificano. $sort(l)$ non ordina la lista l ma crea una copia ordinata di l. In altre parole, ci si concentra sulla *trasformazione* di valori immutabili invece che sulla modifica passo-passo dello stato della memoria.
- **Non ci sono side-effect**: l'unico effetto di una chiamata di funzione è la computazione del suo risultato. Di conseguenza si eliminano molti possibili bug e si rende (quasi) irrilevante l'ordine di esecuzione: poiché non ci sono side-effect, un'espressione non cambia valore nel tempo, quindi può essere valutata in qualsiasi momento.
- Usa la *ricorsione* come struttura di controllo principale.
- Le funzioni sono valore first-class:
  - possono essere definite ovunque, anche innestandole dentro altre funzioni;
  - possono essere passate come parametro oppure essere restituite come valore di ritorno (*funzioni higher-order*);
  - esistono operatori per comporre funzioni tra loro. L'abilità di decomporre un problema in sottoproblemi dipende infatti anche dalla capacità di re-incollare le soluzioni ai sottoproblemi.
- L'interprete del linguaggio funzionale funziona un pó come un calcolatore interattivo, che valuta l'espressione inserita restituendone il valore.

Al contrario, la programmazione imperativa si distingue per

- strutture dati modificabili; quindi esiste un concetto di stato/memoria e un programma modifica lo stato/memoria come effetto della sua esecuzione.
- uso di iterazione invece di ricorsione;
- funzioni solo first-order.

1.1 Sintassi

Consideriamo un linguaggio funzionale molto semplice, che corrisponde al nocciolo di ogni linguaggio funzionale. Definiamo i termini del linguaggio con la seguente grammatica, in cui assumiamo che $x \in Var$ vari nell'insieme delle variabili, mentre $n \in Num$ vari nell'insieme dei numeri naturali:

Termini M, N	$::=$	x	variabili
	$\mid$	$n \mid \mathsf{true} \mid \mathsf{false}$	costanti intere e booleane
	$\mid$	$M + M \mid M - M$	operazioni intere
	$\mid$	$\mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M$	condizionale
	$\mid$	$\mathsf{fn}\, x.M$	dichiarazione di funzione
	$\mid$	MM	applicazione di funzione

NOTE:

- le funzioni e i loro argomenti sono nella stessa categoria sintattica, ciò implica in particolare che si possano passare funzioni a funzioni (funzioni higher-order). Anche le funzioni cioè sono valori.

- Le variabili rappresentano VALORI, e non sono modificabili. Non si tratta infatti di locazioni di memoria. Le variabili dei programmi saranno tutte legate (i.e. saranno dei parametri formali): non ha senso infatti che ci siano variabili libere.
- La semantica di questo linguaggio non indica come uno stato, cioè un'associazione tra celle di memoria e loro valori, varia per effetto del programma. La semantica operazionale descriverà invece la computazione del programma in termini di come ridurre un termine (espressione) ad un valore.

Esempi di termini del linguaggio:

- $3 + 2$,
- $\mathsf{fn}\, x.x$ è la funzione identità, $\mathsf{fn}\, x.3$ è una funzione costante, $\mathsf{fn}\, x.x + 1$ e la funzone incremento.
- $\mathsf{fn}\, x.x + 1\ \ 3$,
- $\mathsf{fn}\, x.\mathsf{fn}\, y.y + x$,
- $\mathsf{fn}\, x.2 + x$ e $\mathsf{fn}\, x.(\mathsf{fn}\, y.y + x\ \ 2)$,
- $\mathsf{fn}\, x.(\mathsf{fn}\, y.y + x\ \ 2)\ \ 3$.
- Il termine $M = \mathsf{fn}\, x.\mathsf{fn}\, y.(x\ \ y)$ corrisponde ad una funzione che prende due paramentri (curryficazione) applica il secondo al primo; ad esempio $(M\ \mathsf{fn}\, z.z)\ 5$.
- Ma anche $3 + \mathsf{false}$, opp. $\mathsf{if}\ 2\ \mathsf{then}\ \mathsf{fn}\, x.x + x\ \mathsf{else}\ 0$, opp. $\mathsf{fn}\, x.\mathsf{if}\, x\, \mathsf{then}\ 1\ \mathsf{else}\ x + 2$ sono termini, anche se "poco sensati".

EXERCISE 1.1.
- Scrivere un termine che rappresenta una funzione higher order, o un programma che contiene una funzione higher order (e.g. $\mathsf{fn}\, x.x\ \mathsf{fn}\, y.y + 1$).
- Scrivere una funzione che restituisce una funzione (e.g. $\mathsf{fn}\, x.(\mathsf{fn}\, y.y + x)$).
- Scrivere una funzione che prende in input una funzione (e.g. $M = \mathsf{fn}\, x.(x\ 7)$ e il programma $M\ \mathsf{fn}\, y.y + 1$).

□

1.1.1 Le funzioni e le variabili

Questo linguaggio è un sovrainsieme del Lambda-calcolo puro, in cui tutto è una funzione: le variabili rappresentano funzioni, le funzioni prendono come parametri delle funzioni, l'applicazione di una funzione restituisce una funzione. Questo linguaggio è Turing completo (ad esempio anche i valori interi, booleani, e operazioni corrispondenti si rappresentano con particolari funzioni).

Il termine $\mathsf{fn}\, x.M$, detto anche "astrazione", *lega* la variabile x; lo scope di questo legame è il termine M e le occorrenze di x all'interno di M sono dette *legate (bound)* dall'astrazione. Le occorrenze di x che non sono nello scope di un'astrazione si dicono *libere*. Esempi: in $M_1 = x\ y$ e $M_2 = \mathsf{fn}\, y.x\ y$ le occorrenze di x sono libere, mentre in $M_1' = \mathsf{fn}\, x.x$ e $M_2' = \mathsf{fn}\, z.\mathsf{fn}\, x.\mathsf{fn}\, y.x\ (y\ z)$ sono legate. E in $M_3 = (\mathsf{fn}\, x.x)\ x$?

Il nome della variabile legata, i.e., il nome del parametro formale, non è in realtà rilevante. Ad esempio $\mathsf{fn}\, x.x$ e $\mathsf{fn}\, y.y$ rappresentano entrambi la stessa funzione, cioè la funzione identità. Consideriamo quindi equivalenti due termini che differiscono solo per i nomi delle variabili legate (α-*equivalence*). In questo linguaggio quindi un termine rappresenta una classe di equivalenza, e quando c'è bisogno, possiamo sostituirlo con (α-convertirlo in) un termine equivalente ma con variabili legate diverse.

Definition 1.2 (α-equivalenza)**.** I termini che differiscono solo per i nomi delle variabili legate sono intercambiabili in qualsiasi contesto. □

Ad esempio il termine $M_3 = (\mathsf{fn}\, x.x)\ x$ è α-equivalente a $(\mathsf{fn}\, y.y)\ x$.

Definition 1.3 (Variabili libere)**.** Dato un termine M, le variabili libere di M, $\mathrm{fv}(M)$, sono definite induttivamente come segue:

$$\begin{array}{rcl} \mathrm{fv}(x) & = & \{x\} \\ \mathrm{fv}(n) = \mathrm{fv}(\mathsf{true}) = \mathrm{fv}(\mathsf{false}) & = & \emptyset \\ \mathrm{fv}(M+N) = \mathrm{fv}(M-N) & = & \mathrm{fv}(M) \cup \mathrm{fv}(N) \\ \mathrm{fv}(\mathsf{if}\ M_1\ \mathsf{then}\ M_2\ \mathsf{else}\ M_3) & = & \mathrm{fv}(M_1) \cup \mathrm{fv}(M_2) \cup \mathrm{fv}(M_3) \\ \mathrm{fv}(\mathsf{fn}\, x.M) & = & \mathrm{fv}(M) \backslash \{x\} \\ \mathrm{fv}(M\ N) & = & \mathrm{fv}(M) \cup \mathrm{fv}(N) \end{array}$$

□

Un termine senza alcuna variabile libera, si dice *chiuso*. I Programmi sono termini chiusi.

Definiamo ora l'operazione di sostituzione di una variabile con un termine, necessaria per la definizione della semantica del linguaggio. Indichiamo con $M\{x := N\}$ il termine M in cui la variabile x è stata sostituita con il termine N. Nel seguito useremo una notazione compatta in cui c varia nell'insieme delle costanti intere e booleane, mentre $\mathsf{op}(M_i)_{i\in I}$ varia nell'insieme delle operazioni aritmentiche e booleane.

Definition 1.4 (Sostituzione)**.**

$$\begin{array}{rcl} x\{x := N\} & = & N \\ y\{x := N\} & = & y \\ c\{x := N\} & = & c \\ \mathsf{op}(M_i)_{i\in I}\{x := N\} & = & \mathsf{op}(M_i\{x := N\})_{i\in I} \\ (\mathsf{fn}\, y.M)\{x := N\} & = & \mathsf{fn}\, y.M\{x := N\}\ \mathsf{if}\ y \notin \mathrm{fv}(N) \\ (M_1\ M_2)\{x := N\} & = & (M_1\{x := N\}\ M_2\{x := N\}) \end{array}$$

□

Esempi:

- $(\mathsf{fn}\, x.x)\{x := 3\}$ non è $\mathsf{fn}\, x.(x\{x := 3\}) = \mathsf{fn}\, x.3$ perché in questo modo la sostituzione modificherebbe la semantica del termine. Invece, poiché $\mathsf{fn}\, x.x =^{\alpha} \mathsf{fn}\, z.z$, si ha $(\mathsf{fn}\, z.z)\{x := 3\} = \mathsf{fn}\, z.(z\{x := 3\}) = \mathsf{fn}\, z.z =^{\alpha} \mathsf{fn}\, x.x$.
- $(\mathsf{fn}\, y.x + y)\{x := 3\} = \mathsf{fn}\, y.3 + y$ mentre $(\mathsf{fn}\, y.x + y)\{x := y\} \neq \mathsf{fn}\, y.y + y$, infatti in questo caso cambierebbe la semantica del termine, che diventa la funzione che restituisce il doppio. Il problema è che l'occorrenza di una variabile libera viene sostituita con una occorrenza di una variabile legata (variable capture). La definizione di sostituzione implica invece che bisogna passare attraverso un termine α-equivalente: $(\mathsf{fn}\, y.x + y)\{x := y\} =^{\alpha} (\mathsf{fn}\, z.x + z)\{x := y\} = \mathsf{fn}\, z.(x + z)\{x := y\} = \mathsf{fn}\, z.y + z$.
- $(\mathsf{if}\ x\ \mathsf{then}\ \mathsf{fn}\, y.y\ \mathsf{else}\ \mathsf{fn}\, z.3)\ \{x := \mathsf{true}\} = \mathsf{if}\ \mathsf{true}\ \mathsf{then}\ \mathsf{fn}\, y.y\ \mathsf{else}\ \mathsf{fn}\, z.3$
- $(\mathsf{fn}\, y.y{+}x\ \ x)\{x := 4\} = (\mathsf{fn}\, y.y{+}4\ \ 4)$
- $(\mathsf{fn}\, y.x)\{x := \mathsf{fn}\, z.z\} = \mathsf{fn}\, y.\mathsf{fn}\, z.z$
- $(\mathsf{fn}\, x.x)\{x := y\} = \mathsf{fn}\, x.x$,
- $(\mathsf{fn}\, y.x)\{x := y\} =_{\alpha} (\mathsf{fn}\, z.x)\{x := y\} = \mathsf{fn}\, z.y$

In altre parole $M\{x := N\}$ sostituisce con N *tutte* le occorrenze **libere** di x in M, facendo attenzione che le variabili libere di N non vengano catturate dai binder di M.

1.2 Semantica Operazionale

La semantica operazionale (small-step) specifica il senso di un programma descrivendone il comportamento in termini dei passi della sua esecuzione. Più precisamente, l'esecuzione di un programma è definita in temini di una *relazione di transizione* (opp. riduzione, opp. valutazione) tra termini del linguaggio: $M \longrightarrow M'$. Un altro modo di interpretare la relazione di transizione è quello di pensare ai termini M come agli *stati di una macchina astratta* il cui comportamento è definito dalla relazione $M \longrightarrow M'$, che indica appunto che dallo stato M la macchina effettua un passo di computazione ed evolve nello stato successivo M'.

Definiamo i *valori*, cioè un sottoinsieme dei termini del linguaggio che corrispondono ai possibili stati finali, cioè i risultati della valutazione di un termine.

$$\begin{array}{rrll} \textit{Valori } v & ::= & n \mid \mathsf{true} \mid \mathsf{false} & \text{costanti intere e booleane} \\ & \mid & \mathsf{fn}\, x.M & \text{dichiarazione di funzione} \end{array}$$

Nota che la funzione è un valore anche se il suo corpo non è stato completamente valutato.
La relazione di transizione small-step $M \longrightarrow M'$ è definita induttivamente come la più piccola relazione binaria sui termini che soddisfa le seguenti regole e assiomi:

(SUM)
$$\frac{}{n_1 + n_2 \longrightarrow n} \quad n =_{Int} n_1 + n_2$$

(MINUS)
$$\frac{}{n_1 - n_2 \longrightarrow n} \quad n =_{Int} n_1 - n_2$$

(IF-TRUE)
$$\frac{}{\mathsf{if\ true\ then}\ M\ \mathsf{else}\ N \longrightarrow M}$$

(IF-FALSE)
$$\frac{}{\mathsf{if\ false\ then}\ M\ \mathsf{else}\ N \longrightarrow N}$$

(SUM LEFT)
$$\frac{M \longrightarrow M'}{M + N \longrightarrow M' + N}$$

(MINUS LEFT)
$$\frac{M \longrightarrow M'}{M - N \longrightarrow M' - N}$$

(SUM RIGHT)
$$\frac{M \longrightarrow M'}{v + M \longrightarrow v + M'}$$

(MINUS RIGHT)
$$\frac{M \longrightarrow M'}{v - M \longrightarrow v - M'}$$

(IF)
$$\frac{M_1 \longrightarrow M_1'}{\mathsf{if}\ M_1\ \mathsf{then}\ M_2\ \mathsf{else}\ M_3 \longrightarrow \mathsf{if}\ M_1'\ \mathsf{then}\ M_2\ \mathsf{else}\ M_3}$$

(BETA)
$$\frac{}{(\mathsf{fn}\, x.M)\ v \longrightarrow M\{x := v\}}$$

(APP 1)
$$\frac{M \longrightarrow M'}{MN \longrightarrow M'N}$$

(APP 2)
$$\frac{M \longrightarrow M'}{vM \longrightarrow vM'}$$

La scelta del formato delle regole determina una *strategia di valutazione* corrispondente all'ordine di valutazione usato nei liguaggi di programmazione: si valutano gli operandi da sinistra a destra, si valuta prima la guardia del test booleano. Gli assiomi indicano cosa fare quando si raggiunge il termine di questo processo, determinando il vero passo di computazione.

EXERCISE 1.5. la relazione di riduzione data definisce una strategia efficiente per la valutazione del termine if-then-else, che permette cioè di valutare unicamente il ramo scelto dalla valutazione della guardia booleana. Ridefinire la semantica operazionale del linguaggio in modo che adotti una strategia non efficiente per il costrutto if-then-else, valutando entambi i rami del costrutto condizionale. □

Per la parte funzionale le regole che abbiamo dato definiscono una strategia di valutazione **call-by-value**, dove l'argomento di una applicazione viene valutato prima di essere sostituito al parametro formale. In questa strategia, usata dalla maggior parte dei linguaggi di programmazione, gli argomenti vengono

sempre valutati, anche se non sono usati nel corpo della funzione. Diversamente, nella strategia **call-by-name** (opp. lazy) gli argomenti vengo valutati davvero solo quando sono effettivamente usati durante l'esecuzione della funzione. Il linguaggio Haskell usa la strategia lazy, Java usa il passaggio per valore, Scala permette entrambe le possibilità.

Indichiamo con $\longrightarrow^*$ la chiusura riflessiva e transitiva di $\longrightarrow$, cioè una sequenza fatta di un numero finito (maggiore o uguale a 0) di passi di riduzione, i.e., $M \longrightarrow^* M'$ se $M' = M$ opp. $M \longrightarrow M_1 \longrightarrow \ldots \longrightarrow M_k \longrightarrow M'$ per qualche $M_1, ..., M_k$.

EXAMPLE 1.6. $((\mathsf{fn}\, x.\mathsf{fn}\, y.x{+}y)\ \ 2)\ \ (\mathsf{if}\,\mathsf{true}\,\mathsf{then}\ 5\ \mathsf{else}\ 1) \longrightarrow (\mathsf{fn}\, y.2{+}y)\ (\mathsf{if}\,\mathsf{true}\,\mathsf{then}\ 5\ \mathsf{else}\ 1) \longrightarrow (\mathsf{fn}\, y.2{+}y)\ 5 \longrightarrow 2+5 \longrightarrow 7$ e questa computazione è derivabile nel modo seguente:

$$\dfrac{\dfrac{}{\mathsf{fn}\, x.\mathsf{fn}\, y.x{+}y\ \ 2 \longrightarrow \mathsf{fn}\, y.2{+}y}\ \mathit{Beta}}{((\mathsf{fn}\, x.\mathsf{fn}\, y.x{+}y)\ \ 2)\ (\mathsf{if}\,\mathsf{true}\,\mathsf{then}\ 5\ \mathsf{else}\ 1) \longrightarrow (\mathsf{fn}\, y.2{+}y)\ (\mathsf{if}\,\mathsf{true}\,\mathsf{then}\ 5\ \mathsf{else}\ 1)}\ \mathit{App}1$$

$$\dfrac{\dfrac{}{\mathsf{if}\,\mathsf{true}\,\mathsf{then}\ 5\ \mathsf{else}\ 1 \longrightarrow 5}\ \mathit{IfTrue}}{(\mathsf{fn}\, y.2{+}y)\ (\mathsf{if}\,\mathsf{true}\,\mathsf{then}\ 5\ \mathsf{else}\ 1) \longrightarrow (\mathsf{fn}\, y.2{+}y)\ 5}\ \mathit{App}2$$

$$\dfrac{}{(\mathsf{fn}\, y.2{+}y)\ 5 \longrightarrow 2+5}\ \mathit{Beta} \qquad \dfrac{}{2+5 \longrightarrow 7}\ \mathit{Sum}$$

□

Proposition 1.7. *La valutazione, cioè l'esecuzione del programma, è deterministica. Se $M \longrightarrow M'$ e $M \longrightarrow M''$ allora $M' = M''$.* □

EXERCISE 1.8. Dimostrare la proposizione precedente per induzione sulla struttura del termine M. □

Un termine che non si riduce a nulla e non è un valore si chiama termine *stuck* e rappresenta un errore di programmazione: e.g. $5 + \mathsf{false}$ oppure $(\mathsf{fn}\, x.\mathsf{if}\ x\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false})\ (\mathsf{if}\,\mathsf{false}\,\mathsf{then}\ \mathsf{fn}\, y.\mathsf{true}\ \mathsf{else}\ 4) \longrightarrow (\mathsf{fn}\, x.\mathsf{if}\ x\ \mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false})\ 4 \longrightarrow \mathsf{if}\ 4\,\mathsf{then}\ \mathsf{true}\ \mathsf{else}\ \mathsf{false} \not\longrightarrow$.

Non tutti i termini ammettono una computazione finita: $\Omega = (\mathsf{fn}\, x.x\, x)\ \mathsf{fn}\, x.x\, x$ riduce in un passo a se stesso, quindi *diverge* e non raggiunge mai una forma normale, cioè un valore finale.

EXERCISE 1.9. Descrivere la valutazione del termine $((\mathsf{fn}\, x.3)\ (\mathsf{fn}\, y.y))\ \ ((\mathsf{fn}\, z.\mathsf{if}\ z\,\mathsf{then}\,1\,\mathsf{else}\,0)\ (\mathsf{false}))$. Modificare le regole di valutazione in modo tale che, mantenendo una strategia call-by-value, il termine precedente evolva in un termine stuck in meno passi di riduzione. Scrivere le regole di valutazione della strategia call-by-name e valutare il termine precedente secondo questa strategia. □