HASKELL

Nicola Fiorillo

Nicola Fiorillo Editions

Indice

Capitolo 1

Lambda calculus

1.1 Introduzione

L'essenza della programmazione funzionale è che un programma è una combinazione di *espressioni*. Espressioni includono valori concreti, variabili e funzioni.
Le *funzioni* sono espressioni a cui è applicato un argomento (input, dominio) e, una volta applicato, può essere *reduced* o *evaluated* ad un valore (output, codominio).
Una funzione è una relazione tra due insiemi; definisce rappresenta quindi una relazione il cui output è sempre prevedibile.
Haskell è un linguaggio a paradigma funzionale *puro*, cioè che gode della proprietà chiamata *referential transparency*. Ciò significa che dati alcuni valori da valutare, viene ritornato lo stesso risultato.

1.2 Definizioni

Anonymous function: $\lambda x.x$
Il punto (.) separa il parametro dal body della funzione.

Alpha equivalence: es. $\lambda x.x$, $\lambda d.d$ e $\lambda z.z$ rappresentano la stessa funzione.

Beta reduction: Quando si applica un argomento ad una funzione, si sostituisce l'input a tutte le istanze della variabile nel corpo della funzione e viene eliminata la testa.
L'applicazione del lambda calculus e associativo da sinistra.

Espressioni riducibili sono chiamate anche *redexes*.
Bounded variable: variabili nel corpo della funzione che hanno riferimento nella testa.

Free variable: variabili nel corpo della funzione che non hanno riferimento nella testa.

Currying: una lambda può avere un solo parametro e può accettare solo un argomento; funzioni che richiedono argomenti multipli sono innestati: $\lambda xy.xy$ risulta $\lambda x.(\lambda y.xy)$

Beta normal form: quando non è più possibile ridurre un'espressione; corridponde quindi ad una espressione completamente valutata o, in Haskell, un programma completamente eseguito.

Combinators: lambda senza free variables (e.g. $\lambda x.x$)

Divergence: quando il processo di riduzione risulta infinito.

Argument: valore passato al parametro della funzione quando la funzione è applicata.

Parameter: variabile che rappresenta un valore all'interno della funzione.

Capitolo 2

Hello, Haskell!

Definire una funzione
`Prelude> let triple x = x * 3` in GHCi
`triple x = x * 3` in file sorgente