STRATEGIA DI VALUTAZIONE: è un algoritmo che quando abbiamo espressioni in cui sono presenti sottoespressioni "riducibili o trasformabili" sceglie quelle da trasformare "ridurre"

Quello che distingue un linguaggio funzionale da un altro sono soprattutto le strategie di valutazione.

Haskell utilizza una strategia chiamata lazy, che riduce il sottotermine la cui valutazione è indispensabile per arrivare al risultato.

STRATEGIE DI VALUTAZIONE CHE NON TERMINANO

(alwayseven (inf 3))

→ se valutiamo inf 3 = inf 3 = inf 3 ...

→ se valutiamo alwayseven → 7

alwayseven n = 7
inf n = inf n

Le funzione di ordine superiore sono funzioni che prendono altre funzioni come argomento o che restituiscono funzioni come risultato

ESEMPIO

atzero f = f 0

atzero prende come argomento una funzione e restituisce il valore di quest'ultime sul numero 0.

Haskell valuta l'espressione (atzero square) : ⇝C CALCOLO DI UNA FUNZ. PREDEFINITA

⇝A ((\f → (f 0)) square)

⇝B (square 0)

⇝A ((\n → (n*n)) 0)

⇝B (0*0)

⇝C 0

⇝A SOSTITUIRE UN NOME CON L'ESPRESSIONE ASSOCIATA AD ESSO

⇝B SOSTITUIRE IL PARAMETRO ATTUALE AL POSTO DEL PARAMETRO FORMALE

Possiamo definire delle funzioni in cui gli argometi e i risultati sono funzioni.

ESEMPIO compose : che prende due funzioni (unarie) come argometi e restituisce la loro composizione.

compose f g = \x -> (f (g x))

$$\underset{\underset{\text{inc}}{\shortparallel}}{f}\big(\overset{\overset{3}{\shortparallel}}{g(x)}\big)$$

definiamo anche inc n = n+1

e poi valutiamo ((compose square inc) 3) = 16

$$\underset{\underset{4}{\shortparallel}}{\text{inc}(3)}$$

$$f(4) = 16$$

↓

square

inc 3 = 4

square 4 = 16

POSSIAMO DIRE square inc = (compose square inc)

CURRYFICAZIONE

La nozione di funzione di ordine superiore ci permette di introdurre quella di "curryficazione".

Curryficare significa trasformare una funzione

$$f : (A_1 \times \ldots \times A_n) \to B$$

in ~~una~~ funzione di ordine superiore, prendono funzioni e restituiscono funzioni

$$f_c : A_1 \to (A_2 \to (\ldots \to A_n \to B) \ldots))$$

tale che

$$f(a_1, a_2, \ldots, a_n) = f_c(a_1)(a_2) \ldots (a_n)$$

Nell' linguaggio Haskell le funzioni vengono implicitamente curryficate

Possiamo scrivere compose

$$\text{compose } f\ g\ x = (f\ (g\ x))$$

SI PUÒ PASSARE DA UNA FUNZIONE CURRYFICATA AD UNA NON E VICEVERSA, COME CONSEGUENZA DELLA DEFINIZIONE

PER ESEMPIO, nel λ-calcolo le funzioni che si introducono hanno un solo argomento e anche esse possono essere curryficate.

INDUZIONE NEI PROGRAMMI

L'uso dell'induzione per dimostrare proprietà di programmi permette l'uso di tecniche matematiche per manipolare programmi, per ottimizzarli opp. per dimostrare proprietà quali la correttezza.

La ricorsione è alla base della potenza computazionale nei linguaggi formali.

In un programma ricorsivo l'output è definito inizialmente per gli elementi di base e poi è definito l'input considerandolo generico e assumendo che valga per numeri più piccoli.

Il principio di induzione è una regola logica che implicitamente rappresenta anche un metodo di calcolo nella prog. funzionale.
Inoltre nella prog. funzionale corrisponde alla definizione di funzioni numeriche per ricorsione come il "fact" di Haskell

"fact 0 ;

fact k = k+1

Vogliamo dimostrare che NOTA [VS]

Per ogni input n il calcolo (la valutazione) di (fact n) termina

Vogliamo mostrare:

1. la valutazione (fact 0) termina ✓
2. per un generico $k>0$, la valutazione di fact k termina, utilizzando l'ipotesi induttiva (fact(k-1)) termini.

1. fact 0 = 1 <u>TERMINA</u> \n -> if (n==0) then 1 else n*(fact(n-1))
2. se $k>0$
   valutiamo (fact k) -> k*(fact (k-1))

PER L'IPOTESI INDUTTIVA fact(k-1) È VERA perché è l'insieme di moltiplicazioni e sottrazione che è procedure che termina

ANCHE fact k termina perché è una molt. di qualcosa che termina

λ-CALCOLO

È un modello computazionale sul quale si base il linguaggio funzionale

È UN TIPO DI LINGUAGGIO SEMPLICE CHIAMATO PARADIGMICO

BASATO SU:

VARIABILE $(x, y, z \ldots)$

ASTRAZIONE O FUNZIONE ANONIMA $\lambda x . M$ dove M è un termine e x è una variabile

APPLICAZIONE: MN dove M ed N sono termini

TUTTI QUESTI RAPPRESENTANO I LAMBDA-TERMINI che sono i programmi e i dati del modello

LAMBDA TERMINI COSTITUISCONO IL λ-CALCOLO

IL λ-calcolo è definito dalla seguente grammatica:

$$\Lambda ::= X \mid (\Lambda \Lambda) \mid \lambda X . \Lambda$$

Λ : insieme dei λ-termini

X : METAVARIABILE e APPARTIENE ALL'INSIEME NUMERABILE DI TUTTE LE VARIABILI

ASTRAZIONE: serve per creare funzioni anonimme (senza nome)

$\lambda x . M$ preso l'input x ritorna il valore del corpo M

PER ESEMPIO

$\lambda x . (x(\lambda y . yx))$

$\downarrow$ ABBR.

$\lambda x . x(\lambda y . yx)$

Abbreviazioni:

$(\lambda x_1 . (\lambda x_2 . \ldots . (\lambda x_n . M))) \to (\lambda x_1 x_2 x_3 \ldots x_n . M)$

$(\ldots (M_1 M_2) M_3) \ldots M_n) \to (M_1 M_2 M_3 \ldots M_n)$

$\lambda x.P$ $\quad$ P AMBIENTE DELL'ABLAZIONE

Nel termine, una variabile può essere limitata se rappresenta l'argomento del corpo della funzione, libera altrimenti.

DEF. VARIABILE LIMITATA

Definiamo $BV(M)$ l'insieme delle variabili limitate di M

Introducendo:

$$BV(x) = \Phi \qquad \text{insieme vuoto}$$

$$BV(PQ) = BV(P) \cup BV(Q)$$

$$BV(\lambda x.P) = \{x\} \cup BV(P)$$

DEF. VARIABILE LIBERA

Definiamo $FV(M)$ l'insieme delle variabili libere di M

Introduciamo

$$FV(x) = \{x\}$$

$$FV(PQ) = FV(P) \cup FV(Q)$$

$$FV(\lambda x.P) = FV(P) \setminus \{x\}$$

SOSTITUZIONE

Notazione $M[L \setminus x]$ termine dove ogni variabile libera x, ossia che non rappresenta un argomento della funzione, viene rimpiazzata da L

DEF. DI SOSTI

DEF. DI SOSTITUZIONE

1) Se M è una variabile $(M = y)$ allore

$$y[L/x] = \begin{cases} L & \text{se } x = y \\ y & \text{se } x \neq y \end{cases}$$

2) Se M è un'applicazione $M = PQ$ allore

$$PQ[L/x] = P[L/x]Q[L/x]$$

3) Se M è una labda astrazione $(M = \lambda y.P)$ allore

$$\lambda y.P[L \backslash x] = \begin{cases} \lambda y.P & \text{se } x = y \\ \lambda y.(P[L \backslash x]) & \text{se } x \neq y \end{cases}$$

N.B.

Si possono sostiture solo se le variabili da sostiture sono libere

Controesempio in cui le variabili non sono variabili libere.

FUNZIONI COSTANTI CHE RITORNANO ENTRAMBE z Ogni ARGOMENTO

$(\lambda x. z)$ e $(\lambda y. z)$

SOSTITUIAMO $[x/z]$ x a z (x per le variabili libere z)

$(\lambda x. z)[x/z] \to \lambda x. z[x/z] \to \lambda x. x$ FUNZ. IDENTITÀ

$(\lambda y. z)[x/z] \to \lambda y. z[x/z] \to \lambda y. x$ FUNZ. CHE RITORNA SEMPRE X

ABBIAMO OTTENUTO DUE FUNZIONI DIVERSE DOPO LA SOSTITUZIONE