

Strategie 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 sottoespresso le cui valutazione è indispensabile per arrivare al risultato

Strategie di valutazione che non terminano

(alwaysrem (mf 3))

alwaysrem m = λ

mf m = mf m

se valutiamo mf 3 = mf 3 = mf 3

se valutiamo alwaysrem \rightarrow λ

} due strategie

Funzione di ordine superiore

Sono funzioni che prendono altre funzioni come argomento o che restituiscono funzioni come risultato

Esempio

otzero f = f 0

otzero prende come argomento una funzione e restituisce il valore di quest'ultima sul numero 0

Haskell valuta l'espressione (otzero square):

→ A sostituire un nome con l'espressione associata ad esso

→ B sostituire il parametro otale al posto del parametro formale

no C Calcolo di una funzione predefinita

no A $((\lambda f \rightarrow (f_0)) \text{square})$

no B $(\text{square} \ 0)$

no A $((\lambda m \rightarrow (m \cdot m) \ 0)$

no B $(0 \cdot 0)$

no C 0

} funzione in
formato Haskell

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

Esempio COMPOSE: che prende due funzioni (unarie) come esponenti e restituisce la loro composizione

compose $ff = \lambda x \rightarrow (f(fx))$

definiamo anche che $m = m + 1$

e poi risolviamo $((\text{compose} \ \text{square} \ m) \ 3)$

$$m^3 = 9$$

$$\text{square} \ 4 = 16$$

$$f(f(f(x))) \\ \overset{\text{"m" }}{\underset{\text{"m" }}{\overbrace{\quad \quad}}} \quad \overset{3}{\overbrace{\quad \quad}} \quad m^3 = 9$$

$$f(4) = 16 \\ \downarrow \text{square}$$

Possiamo dire $\text{square} \ m = (\text{compose} \ \text{square} \ m)$

CURRYIFICAZIONE

Le nozioni di funzione di ordine superiore ci permette di introdurre quelle di "curryfication"

Curryficare significa trasformare una funzione

$$f: (A_1 \cdot \dots \cdot A_m) \rightarrow B$$

in funzioni di ordine superiore, prendendo funzioni e sostituendone funzioni

$$f_c: A_1 \rightarrow (A_2 \rightarrow (\dots \rightarrow A_m \rightarrow B) \dots))$$

tale che

$$f(e_1, e_2, \dots, e_m) = f_c(e_1)(e_2) \dots (e_m)$$

Nel linguaggio Haskell le funzioni vengono implicitamente curryificate

Potiamo scrivere composite così:

$$\text{composto } f \circ g = (f(g(x)))$$

Si può scrivere da una funzione Curryficate ad una non e viceversa come conseguenze delle definizioni

Esempio:

nel λ -calcolo le funzioni che si introducono hanno un solo argomento e anche esse possono essere curryificate

Induzione sui programmi

L'uso dell'induzione per dimostrare proprietà di programmi permette l'uso di tecniche matematiche per manipolare programmi, per ottimizzarli oppure per dimostrare proprietà su

Le ricorsione è alla base delle potenze computazio
=
nele 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 i numeri più piccoli.

Il principio di induzione è una regola logica che
implicitamente rappresenta anche un metodo di
calcolo nelle prog. funzionale

Inoltre nelle prog. funz. corrisponde alle definizioni
di funzioni numeriche per ricorsione come il "fact"
di Haskell

Vogliamo dimostrare che per ogni input n il
calcolo (la soluzione) di $(\text{fact } n)$ termina

Vogliamo mostrare:

1. la soluzione fact 0 termina
2. per un generico $k > 0$ la soluzione di fact k
termina utilizzando l'ipotesi induttiva ($\text{fact}(k-1)$)

Dimostrazione

1. $\text{fact } 0 \rightarrow \text{if } (m == 0) \text{ then } 1$
 $\qquad \qquad \qquad \text{else } m \cdot (\text{fact}(m-1))$

2. Se $k > 0$

vogliamo $(\text{fact } k) \rightarrow k \cdot (\text{fact}(k-1))$

Per ipotesi induttiva $\text{fact}(k-1)$ è vero perché è

l'insieme di moltiplicazioni e sottrazioni che è
procedura termine

Anche λ -termine poiché è una moltiplicazione
di qualsiasi con qualcosa che termine

λ - calcolo

È un modello computazionale sul quale si basa il
linguaggio funzionale. È un tipo di linguaggio semplice
chiamato λ -calcolo

Basato su

Variebili $(x, y, z \dots)$

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 λ -termini che sono
i programmi e i dati del modello

λ -termini costituiscono il λ -calcolo

Se λ -calcolo è definito delle seguenti grammatiche

$$\Lambda ::= x \mid (\Delta \Delta) \mid \lambda x. \Delta$$

Δ insieme dei λ -termini

X metonimibili e appartenente all'insieme numerabile
di tutte le variabili

Astrazioni servono per creare funzioni anonime
(senza nome) $\lambda x.M$ "per l'input x ritorno del
corpo di M "

Abbreviazioni:

$$(\lambda x_1 \cdot (\lambda x_2 \cdot \dots \cdot (\lambda x_m \cdot M))) \rightarrow (\lambda x_1 x_2 x_3 \dots x_m M)$$

$$(\dots (M_1 M_2) M_3) \dots M_n \rightarrow (M_1 M_2 M_3 \dots M_n)$$

Per esempio

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

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

$\lambda x P$ P ambiente dell'abstrazione

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

variabile limitata al
corpo delle funzioni
(v.a. locale)

Definizione variabile limitata

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

Introducendo

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

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

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

Definizione variabile libera

non si trova nelle
definiz. delle funzioni
ed libere
(v.a. globale)

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

Introducendo

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

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

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

x è una v.a. limitata

Sostituzione

Notazione: $M[L/x]$ termine dove ogni variabile libera x , ovie che non corrispondente un sostituto della funzione, viene rimpiazzato da L

" $M[L/x]$ significa sostituisci tutte le occorrenze libere delle variabili x in M con L .
Lo corso delle funzioni"

Esempio:

$$M = x + 1$$

$$L = 5$$

x = variabile da sostituire

$$M[L/x] = (x+1)[5/x] = 5+1$$

Definizione di sostituzione

1) Se M è una costante ($M=y$) allora

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

sono
definizioni

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

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

3) Se M è una λ -espressione ($M=\lambda y.P$) allora

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

Note bene

Si possono sostituire solo se le variabili da sostituire sono libere

Controesempio

In cui le variabili non sono libere

Funzioni costanti che ritornano entrambe 2 per ogni argomento

$$\lambda x. 2 \quad e \quad \lambda y. 2$$

Sostituiamo x con 2:

$$(\lambda x. 2)[x/2] \rightarrow \lambda x. 2[x/2] \rightarrow \lambda x. x \quad \text{funk identità}$$

$$(\lambda y. 2)[x/2] \rightarrow \lambda y. 2[x/2] \rightarrow \lambda y. x \quad \text{funk che ritorna sempre } x$$

Abbiamo ottenuto 2 funzioni diverse dopo la sostituzione