3 Programmazione nel calcolo lambda non tipizzato

Uno dei fatti sorprendenti sul calcolo lambda non tipizzato è che possiamo usarlo per codificare dati, come numeri booleani e naturali, nonché programmi che operare sui dati. Questo può essere fatto esclusivamente all'interno del calcolo lambda, senza aggiungendo qualsiasi sintassi o assiomi aggiuntivi.

Avremo spesso occasione di dare nomi a particolari termini lambda; noi

di solito usa lettere in grassetto per tali nomi.

3.1 Booleani

Iniziamo definendo due termini lambda per codificare i valori di verità "veri" e "Falso":

T = λxy.x

F = λxy.y

Sia e sia il termine λab.abF. Verifica quanto segue:

e TT →→ β T

e TF →→ β F

e FT →→ βF

e FF →→ β F

Si noti che T e F sono forme normali, quindi possiamo davvero dire che un termine come

e TT restituisce T. Lo diciamo e codifica la funzione booleana “e”. Esso

resta inteso che questa codifica è rispetto alla codifica particolare di "vero" e "Falso". Non lo affermiamo e MN restituisce qualcosa di significativo se M o N sono termini diversi da T e F.

Per inciso, non c'è nulla di unico nel termine λab.abF. È uno dei tanti

possibili modi di codificare la funzione “e”. Un'altra possibilità è λab.bab.

Esercizio 5. Trova i termini lambda o e non che codificano le funzioni booleane "or"

e non". Riesci a trovare più di un termine?

Inoltre, definiamo il termine se allora else = λx.x. Questo termine si comporta come un funzione "se-allora-altrimenti" - in particolare, abbiamo

se allora altrimenti TMN →→β M

se allora altrimenti FMN →→β N

per tutti i termini lambda M, N.

3.2 Numeri naturali

Se f e x sono termini lambda e n > 0 un numero naturale, scrivi f

nx per il termine

f(f(. . .(fx). . .)), dove f ricorre n volte. Per ogni numero naturale n, definiamo

un termine lambda n, chiamato n-esimo numero della Chiesa, come n = λfx.fnx. Ecco le

primi numeri della Chiesa:

0 = λfx.x

1 = λfx.fx

2 = λfx.f(fx)

3 = λfx.f(f(fx))

4 = λfx.f(f(f(f(x))

. . .

Questo particolare modo di codificare i numeri naturali si deve ad Alonzo Church,

che fu anche l'inventore del calcolo lambda. Si noti che 0 in effetti è lo stesso

termine come F; quindi, quando interpretiamo un termine lambda, dovremmo saperlo in anticipo

se interpretare il risultato come un booleano o un numero.

La funzione successore può essere definita come segue: succ = λnfx.f(nfx). Che cosa

questo termine calcola se applicato a un numero?

succ n = (λnfx.f(nfx))(λfx.f nx)

→β λfx.f((λfx.fnx)fx)

→β λfx.nx)

= λfx.f n+1x

= n + 1

Pertanto, abbiamo dimostrato che il termine succ codifica effettivamente la funzione successore, quando applicato a un numero. Ecco le possibili definizioni di addizione e

moltiplicazione:

aggiungi = λnmfx.nf(mfx)

mult = λnmf.n(mf).

Esercizio 6. (a) Valuta manualmente i termini lambda aggiungi 2 3 e muti 2 3.

(b) Dimostrare che somma n m →→β n + m, per tutti i numeri naturali n, m.

(c) Dimostrare che moltiplica n m →→β n · m, per tutti i numeri naturali n, m.

Definizione. Supponiamo f : N

k → N è una funzione k-ari sui numeri naturali, e

che M è un termine lambda. Diciamo che M (in senso numerico) rappresenta f se per tutti

n1, . . . , nk ∈ N,

M n1 . . . nk →→β f(n1, . . . , nk).

Questa definizione rende esplicito cosa significa essere una “codifica”. Possiamo dire, per

ad esempio, che il termine add = λnmfx.nf(mfx) rappresenta la funzione di addizione. La definizione si generalizza facilmente a funzioni booleane o funzioni di altro

tipi di dati.

Spesso utile è la funzione èzero da numeri naturali a booleani, che è

definito da

iszero (0) = vero

iszero (n) = falso, se n diverso 0.

Convincetevi che il seguente termine è una rappresentazione di questa funzione:

iszero = λnxy.n(λz.y)x.

Esercizio 7. Trova i termini lambda che rappresentano ciascuna delle seguenti funzioni:

(a) f(n) = (n + 3)2

,

(b) f(n) =

vero se n è pari,

falso se n è dispari,

(c) exp (n, m) = n

m,

(d) pred (n) = n - 1.

Nota: la parte (d) non è facile. In effetti, Church ha creduto per un po' che fosse impossibile, finché il suo studente Kleene non ha trovato una soluzione. (In effetti, Kleene ha detto di aver trovato

la soluzione pur avendo tirato i denti del giudizio, quindi il suo trucco per definire il la funzione predecessore è a volte indicata come il "trucco dei denti del giudizio".)

Abbiamo visto come codificare alcune semplici funzioni booleane e aritmetiche. Tuttavia, non abbiamo ancora un metodo sistematico per costruire tali funzioni. Che cosa abbiamo bisogno è un meccanismo per definire funzioni più complicate da semplici

quelli. Si consideri ad esempio la funzione fattoriale, definita da:

0! = 1

n! = n · (n − 1)!, se n diverso 0.

La codifica di tali funzioni nel calcolo lambda è l'argomento del prossimo

sezione. È legato al concetto di punto fisso.