2 Il lambda calcolo non tipato

Il λ-calcolo e' la teoria che ha ispirato i linguaggi funzionali, nello stesso modo in cui la logica matematica ha ispirato i linguaggi per la programmazione logica. In effetti, il λ-calcolo e' uno strumento teorico estremamente versatile per studiare i linguaggi di programmazione in generale.

Vedremo il λ-calcolo da tre punti di vista: come teoria equazionale, come teoria della semplificazione, come linguaggio di programmazione (descritto mediante una semantica operazionale).

2.1 La teoria equazionale

I termini del λ-calcolo puro (e non tipato) sono descritti dalla seguente BNF

$$M \in \Lambda \quad ::= \quad x \mid M_1 M_2 \mid \lambda x.M$$

Nelle applicazioni si tende ad arricchire il linguaggio con costanti e/o operazioni (di arieta' prefissata)

$$M \in \Lambda \quad ::= \quad x \mid c \mid op(\overline{M}) \mid M_1 M_2 \mid \lambda x.M$$

Per il momento ci limiteremo al λ-calcolo puro, che nonostante la sua semplicita' ha *la stessa potenza* di calcolo delle macchine di Turing. La teoria equazionale del λ-calcolo e' definita dai seguenti assiomi e regole di inferenza per derivare asserti della forma $M = N$

- regole che asseriscono che $=$ e' una relazione di equivalenza (cioe' riflessiva, simmetrica e transitiva)
- regole che asseriscono che $=$ e' una congruenza, cioe'

  $$(\text{app})\ \frac{M_1 = M_1' \qquad M_2 = M_2'}{M_1 M_2 = M_1' M_2'} \qquad\qquad (\xi)\ \frac{M = M'}{(\lambda x.M) = (\lambda x.M')}$$

- l'assioma che permette di ridenominare le variabili legate

  (α) $(\lambda x.M) = (\lambda y.[y/x]M)$ $y \notin \mathrm{FV}(M)$

  questa proprieta' e' tipica delle variabili legate, p.e. nella logica del primo ordine l'analogo di tale assioma asserisce l'equivalenza logica delle formule$\forall x.\phi$ e $\forall x.([y/x]\phi)$

- l'assioma che permette di semplificare l'applicazione di una funzione ad un argomento

  (β) $(\lambda x.M)N = [N/x]M$

  questa e' lunica regola che ha un significato computazionale

- un'altro assioma, che pero' noi non considereremo, asserisce che ogni termine e' una funzione

  (η) $M = (\lambda x.Mx)$ $x \notin \mathrm{FV}(M)$

  tale assioma e' sensato per il λ-calcolo puro, tuttavia se si aggiungono al linguaggio costanti che non corrispondono a funzioni la regola η diventa poco plausibile.

Negli assiomi α e β si e' fatto uso dell'insieme $\mathrm{FV}(M)$ delle variabili libere e dell'operazione $[N/x]M$ di sostituzione. La definizione di queste nozioni richiede un po' di attenzione. poiche' i termini del λ-calcolo contengono variabili legate (al contrario dei termini algebrici). Le soluzioni adottate nel λ-calcolo sono facilmente adattabili a qualsiasi formalismo in cui le variabili possono essere legate.

Definition 2.1 *L'insieme* $\mathrm{FV}(M)$ *e' definitio per induzione sulla struttura di* M

$$\begin{aligned}
\mathrm{FV}(x) &= \{x\} \\
\mathrm{FV}(M_1 M_2) &= \mathrm{FV}(M_1) \cup \mathrm{FV}(M_2) \\
\mathrm{FV}(\lambda x.M) &= \mathrm{FV}(M) - \{x\}
\end{aligned}$$

L'operazione di sostituzione $[N/x]M$ *e' definita per induzione sulla dimensione* $|M|$ *di* M, *o piu' precisamente il numero di* λ *ed applicazioni in* M *(tale numero e' invariante per ridenominazione di variabili)*

$$\begin{aligned}
[N/x]x &= N \\
[N/x]y &= y \ \mathit{se}\ x \not\equiv y \\
[N/x](M_1 M_2) &= ([N/x]M_1 [N/x]M_2) \\
[N/x](\lambda y.M) &= \lambda z.[N/x]([z/y]M) \ \mathit{dove}\ z \notin \mathrm{FV}(\lambda y.M) \cup \mathrm{FV}(N) \cup \{x\}
\end{aligned}$$

Una definizione alternativa di sostituzione che evita per quanto possibile la ridenominazione delle variabili legate e'

$$\begin{array}{rcl}
[N/x](\lambda x.M) & = & \lambda x.M \\
[N/x](\lambda y.M) & = & \lambda y.[N/x]M \textit{ se } x \not\equiv y \textit{ e } y \notin \mathrm{FV}(N), \textit{ altrimenti} \\
[N/x](\lambda y.M) & = & \lambda z.[N/x]([z/y]M) \textit{ dove } z \notin \mathrm{FV}(\lambda y.M) \cup \mathrm{FV}(N) \cup \{x\}
\end{array}$$

L'ultimo caso nella definizione di sostituzione dice che prima di effettuare la sostituzione di x con N si deve ridenominare la variabile legata y con una *nuova* variabile z. Questa ridenominazione serve ad evitare l'effetto indesiderato che si avrebbe con la definizione naive $[N/x]\lambda y.M = \lambda y.[N/x]M$, ovvero legare le occorenze libere di y in N.
La definizione di sostituzione e' data in modo non-deterministico, poiche' possiamo scegliere qualsiasi $z \notin \mathrm{FV}(\lambda y.M) \cup \mathrm{FV}(N) \cup \{x\}$. E' possibile fissare un criterio di scelta deterministico, p.e. fissando un ordine totale sulle variabili. Nella pratica si tende ad ignorare il problema, questa mancanza di precisione non e' pericolosa purche' si tenga presente che quello che si sta manipolango non sono termini ma classi di equivalenza di termini modulo *α-conversione*. A tal proposito e' opportuno distinguere tre relazioni di congruenza sui λ-termini

- $M \equiv N$ e' l'uguaglianza sintattica
- $M \equiv_\alpha N$ e' l'α-conversione, cioe' la congruenza indotta da assiomi e regole di inferenza rimuovendo la regola β
- $M =_\beta N$ e' la β-conversione, cioe' la congruenza indotta da assiomi e regole di inferenza.

Scriviamo $\equiv_\alpha$ invece di $=_\alpha$ per sottolineare che l'α-conversione e' piu' opportuno considerarla come la corretta definizione di uguaglianza sintattica in presenza di variabili legate.

Proposition 2.2 *Se $M \equiv_\alpha M'$ e $N \equiv_\alpha N'$, allora $\mathrm{FV}(M) = \mathrm{FV}(M')$ e $[N/x]M \equiv_\alpha [N'/x]M'$.*

Lavorando modulo α-conversione e' sempre possibile effettuare una ridenominazione delle variabili legate in M in modo da evitare conflitti con le variabili libere di N, dopodiche' si puo' usare senza pericolo anche una definizione naive di sostituzione $[N/x]M$ (che non effettua la ridenominazione).

2.2 La nozione di riduzione

Introduciamo adesso una relazione $M \to_\beta N$ tra termini, detta β-riduzione, che formalizza l'idea che N e' ottenuto da M effettuando un passo di *semplificazione*.

Definition 2.3 *La relazione $\to_\beta$ su Λ e' definita dalle seguente regole di inferenza:* (β) $(\lambda x.M)N \to_\beta [N/x]M$

$$(\mathrm{app.1})\ \frac{M_1 \to_\beta M_1'}{M_1 M_2 \to_\beta M_1' M_2} \qquad (\mathrm{app.2})\ \frac{M_2 \to_\beta M_2'}{M_1 M_2 \to_\beta M_1 M_2'} \qquad (\xi)\ \frac{M \to_\beta M'}{(\lambda x.M) \to_\beta (\lambda x.M')}$$

Si noti che la relazione $\to_\beta$ e' decidibile (al contrario di $=_\beta$), e per ogni M esiste un numero finito di N t.c. $M \to_\beta N$. una proprieta' fondamentale della β-riduzione e' la **proprieta' Church-Rosser**.

Theorem 2.4 *Se $M \to^* M_1$ e $M \to^* M_2$ allora esiste M_3 t.c. $M_1 \to^* M_3$ e $M_2 \to^* M_3$.*

Si tenga present eche stiamo lavorando modulo α-conversione, quindi volendo essere piu' espliciti dovremo scrivere

$$\begin{tikzcd}
 & M \arrow[dl] & \equiv_\alpha & M' \arrow[dr] & \\
M_1 \arrow[dr] & & & & M_2 \arrow[dl] \\
 & M_3 & \equiv_\alpha & M_3' &
\end{tikzcd}$$

Tale proprieta' ci dice che nonostante la relazione di semplificazione sia non deterministica, due semplificazioni M_1 e M_2 di un termine M possono essere ulteriormente semplificate ad un termine comune M_3. Il seguente risultato e' una conseguenza immediata della proprieta' CR.

Theorem 2.5 *$M \to M'$ implica $M = M'$, inoltre $M_1 = M_2$ sse esiste M_3 t.c. $M_1, M_2 \to^* M_3$.*

Ovvero per dimostrare la β-convertibilita' di due termini e' sufficiente dimostrare che si riducono ad un stesso termine.

2.3 Semantiche operazionali

Se vogliamo pensare al λ-calcolo come un linguaggio funzionale (minimale), la cosa piu' semplice da fare e' dare una semantica operazionale. Considereremo quattro tipi di semantica operazionale corrispondenti a due scelte indipendenti

	call-by-value	call-by-name
a piccoli passi	$\rightarrow_v$	$\rightarrow_n$
a grandi passi	$\Downarrow_v$	$\Downarrow_n$

Tali semantiche sono una versione embrionale di quelle effettivamente utilizzate per descrivere i linguaggi di programmazione. Tali semantiche sono date mediante regole di inferenza. Indicheremo con Λ_0 l'insieme dei λ-termini chiusi (cioe' senza variabili libere), e con Val_0 l'insieme delle λ-astrazioni chiuse. Inoltre chiameremo un $M \in \Lambda_0$ **programma** ed un e $V \in \mathsf{Val}_0$ **valore**.

- $\Downarrow_n \subset \Lambda_0 \times \mathsf{Val}_0$ semantica CBN a grandi passi

  (V) $V \Downarrow_n V$ (app) $\dfrac{M_1 \Downarrow_n \lambda x.P \quad [M_2/x]P \Downarrow_n V}{M_1 M_2 \Downarrow_n V}$

  $\Downarrow_v \subset \Lambda_0 \times \mathsf{Val}_0$ semantica CBV a grandi passi

  (V) $V \Downarrow_v V$ (app) $\dfrac{M_1 \Downarrow_v \lambda x.P \quad M_2 \Downarrow_v V \quad [V/x]P \Downarrow_v V'}{M_1 M_2 \Downarrow_v V'}$

- $\rightarrow_n \subset \Lambda_0 \times \Lambda_0$ semantica CBN a piccoli passi

  (β) $(\lambda x.M)N \rightarrow_n [N/x]M$ (app.1) $\dfrac{M_1 \rightarrow_n M_1'}{M_1 M_2 \rightarrow_n M_1' M_2}$

- $\rightarrow_v \subset \Lambda_0 \times \Lambda_0$ semantica CBV a piccoli passi

  (β_v) $(\lambda x.M)V \rightarrow_v [V/x]M$ (app.1) $\dfrac{M_1 \rightarrow_v M_1'}{M_1 M_2 \rightarrow_v M_1' M_2}$ (app.2) $\dfrac{M_2 \rightarrow_v M_2'}{V M_2 \rightarrow_v V M_2'}$ $V \in \mathsf{Val}_0$

Le principali differenze tra β-riduzione e le semantiche operazionali date sopra sono: le semantiche operazionali sono deterministiche, e non effettuano semplificazioni nel corpo di una λ-astrazione. La seconda caratteristica differenzia le semantiche operazioni dalle strategie di riduzione (che scelgono una tra tutte le possibili semplificazioni possibili). Si osservi che le sostituzioni $[N/x]M$ per definire le semantiche operazionali sono tali che N e' sempre un termine chiuso, quindi non sara' mai necessario effettuare ridenominazioni delle variabili legate in M.

Proposition 2.6 *Sia $x \in \{v, n\}$, cioe' CBV o CBN, allora*

- *$M \rightarrow_x N$ implica $M \rightarrow_\beta N$*
- *$M \Downarrow_x V$ sse $M \rightarrow_x^* V \in \mathsf{Val}_0$*

Nota che i valori sono invarianti per β-riduzione, cioe' $M \rightarrow_\beta N$ e $M \in \mathsf{Val}_0$ implicano $N \in \mathsf{Val}_0$. E' possibile correlare la semantica operazionale alla β-conversione. Per far cio' e' necessario introdurre la nozione di osservazione ed equivalenza osservazionale. Per semplicita' considereremo solo le semantiche operazionali a grandi passi.

Definition 2.7 *Data una semantica operazione $\Downarrow_x \subset \Lambda_0 \times \mathsf{Val}_0$, si dice che "$M$* **termina**" *$M \Downarrow_x$ sse $\exists V{:}\mathsf{Val}_0.M \Downarrow_x V$. La* **equivalenza osservazionale** *$\approx_x \subset \Lambda \times \Lambda$ e' data da $M \approx_x N$ sse $C[M] \Downarrow_x \leftrightarrow C[N] \Downarrow_x$ per ogni contesto $C[\_]$ con un buco t.c. $C[M]$ e $C[N]$ sono programmi. In modo analogo si definisce il* **preordine osservazionale** *$M \leq_x N$ sse $C[M] \Downarrow_x \supset C[N] \Downarrow_x$ per ogni contesto $C[\_]$ con un buco t.c. $C[M]$ e $C[N]$ sono programmi.*

Il seguente risultato dice che la β-conversione e' *sound* per dimostrare l'equivalenza osservazionale $\approx_n$.

Theorem 2.8 *$M =_\beta N$ implica $M \approx_n N$.*

Un risultato simile per $\approx_v$ si ottiene indebolendo la teoria equazionale, ovvero sostituendo la regola β con

$$(\beta_v) \quad (\lambda x.M)V = [V/x]M \quad V \text{ valore}$$

Dove un valore (che puo' avere anche variabili libere) e' una variabile o una λ-astrazione. Invece le due equivalenza osservazionali non sono fra loro confrontabili, p.e.:

- $\Omega \approx_v K\ I\ \Omega \not\approx_v I$
- $\Omega \not\approx_n K\ I\ \Omega \approx_n I$

dove $I \equiv \lambda x.x$, $K \equiv \lambda x, y.x$ e $\Omega \equiv (\lambda x.xx)(\lambda x.xx)$ e' l'esempio tipico di programma che non termina, infatti $\Omega \rightarrow_x \Omega$.

2.4 Relazione con la Logica Combinatoria

In questa sezione facciamo vedere che il $\lambda\beta$-calcolo corrisponde alla teoria $CL + \epsilon$ (assiomatizzata nella logica del primo ordine nella Sezione 1.5.2). Tale assiomatizzazione e' dovuta ad Albert Meyer. La corrispondenza e' stabilita mediante due traduzioni tra termini:

- $\_^\lambda : CL \to \lambda\beta$

  $x^\lambda \triangleq x$
  $K^\lambda \triangleq (\lambda x, y.x)$
  $S^\lambda \triangleq (\lambda x, y, z.x\ z\ (y\ z))$
  $(M\ N)^\lambda \triangleq M^\lambda N^\lambda$

- $\_^{cl} : \lambda \to CL + \epsilon$

  $x^{cl} \triangleq x$
  $(\lambda x.M)^{cl} \triangleq \epsilon\ (\lambda^* x.M^{cl})$
  $(M\ N)^{cl} \triangleq M^{cl} N^{cl}$

E' facile dimostrare le seguenti proprieta' delle traduzioni, che ci dicono che esse sono *interpretazioni relative* tra teorie:

- Se $M = N$ e' derivabile in CL, allora $M^\lambda = N^\lambda$ e' derivabile nel $\lambda\beta$-calcolo
- Se $M = N$ e' derivabile nel $\lambda\beta$-calcolo, allora $M^{cl} = N^{cl}$ e' derivabile in $CL + \epsilon$

Per dimostrare che la seconda implicazione e' un se e solo se, basta costruire un opportuno modello di $CL + \epsilon$ che corrisponde al $\lambda\beta$-calcolo. Tale modello non e' altro che l'**open term model** del $\lambda\beta$-calcolo, ovvero la struttura applicativa $(D, \cdot)$ t.c.

- D e' l'insieme delle classi di equivalenza $[M]$ modulo $=_\beta$ di λ-termini $M \in \Lambda$
- $[M] \cdot [N] = [M\ N]$
- l'interpretazione delle costanti di $CL + \epsilon$ e' $K = [\lambda x, y.x]$, $K = [\lambda x, y, z.x\ z\ (y\ z)]$, $\epsilon = [\lambda x, y.x\ y]$

E' abbastanza facile dimostrare che questa struttura applicativa e' un modello di $CL + \epsilon$ (l'unico assioma delicato e' l'estensionalita per ϵ), e che inoltre l'interpretazione $[\![M^{cl}]\!]_\rho = [M]$, dove $M \in \Lambda$ e $\rho(x) = [x]$. Questo ci permette di concludere che $CL + \epsilon \vdash M^{cl} = N^{cl}$ implica $Th(D) \vdash M^{cl} = N^{cl}$, e quindi $M =_\beta N$.

3 Lambda calcoli applicati

Per studiare i linguaggi di programmazione mediante il λ-calcolo e molto conveniente estendere l'insieme dei termini. Se Σ e' una segnatura per una teoria algebrica single-sorted, cioe' Σ_n e' l'insieme dei simboli di operazione n-arie, allora vi sono due modi estendere l'insieme dei termini:

- $M \in \Lambda_\Sigma ::= \ldots \mid op(M_{i:n})$ dove $op \in \Sigma_n$, questo e' il modo piu' ovvio di combinare le regole di formazione dei termini algebrici e dei λ-termini, oppure
- $M \in \Lambda_\Sigma ::= \ldots \mid c_{op}$ dove $op \in \Sigma_n$, cioe' trattare una operazione come una costante higher-order.
- Le due estensioni si possono tradurre l'una nell'altra, infatti la costante c_{op} la si puo' identificare con $\lambda vetx.op(\overline{x})$, dove $\overline{x}$ e' una n-upla di variabili distinte, e $op(\overline{M})$ equivale a $c_{op}\ \overline{M}$.

Sebbene le costanti higher-order permettano una descrizione piu' concisa della sintassi, cio' puo' creare delle complicazioni nel descrivere la semantica operazionale, e soprattutto la nozione di valore. Quindi prenderemo il primo approccio come quello preferenziale. Nel resto della sezione esemplificheremo come estendere la β-conversione, -riduzione e semantica operazionale (per brevita ci limitiamo a considerare la semantica CBV a grandi passi). Parleremo di $\beta\delta$-conversione, dove le δ-regole sono quelle che descrivono il comportamente delle operazioni/costanti aggiuntive.

3.1 Estensione per i booleani

- Le operazioni da aggiungere, con la loro arieta', sono $true, false$: 0 e if: 3
- Le regole equazionali da aggiungere sono

  (if.true) $if(true, M, N) = M$ (if.false) $if(false, M, N) = N$

  per ottenere le regole di riduzione le equazioni vanno orientate da sinistra a desstra
- Il sottoinsieme dei valori (per la semantica operazionale CBV) e' dato dalla BNF

  $V \in \mathsf{Val} ::= x \mid \lambda x.M \mid true \mid false$
- Le regole per la semantica operazionale CBV a grandi passi sono

  (V) $V \Downarrow_v V$ (app) $\dfrac{M_1 \Downarrow_v \lambda x.P \quad M_2 \Downarrow_v V \quad [V/x]P \Downarrow_v V'}{M_1 M_2 \Downarrow_v V'}$

  (if.1) $\dfrac{P \Downarrow_v true \quad M_1 \Downarrow_v V}{if(P, M_1, M_2) \Downarrow_v V}$ (if.2) $\dfrac{P \Downarrow_v false \quad M_2 \Downarrow_v V}{if(P, M_1, M_2) \Downarrow_v V}$

  ovvero quelle per il λ-calcolo puro (ma per l'insieme dei valori esteso), a cui vanno aggiunte le due regole per vautare un if.

Si osservi che nei λ-calcoli applicati si possono avere dei programmi (cioe' termini chiusi) che non sono valori, ma che non sono ulteriormente riducibili. Per esempio $true\ false$ o $if(I, true, I)$. Secondo la definizione che abbiamo dato di terminazione, tali programmi non hanno un valore, e quindi non terminano. Pero' volendo essere piu' precisi si dovrebbe distinguere tra programmi che non terminano (come Ω) e programmi che si bloccano in una situazione anomala (come $true\ false$). Questo tipo di distinzione e' importante quando si considerano i *type systems* e la loro capacita' di individuare a tempo di compilazione la possibilita' di errori a tempo di esecuzione.

3.2 Estensione per i naturali

- Le operazioni da aggiungere (oltre a quelle per i booleani) sono $zero$: 0 e $succ, pred, is0$: 1
- Le regole equazionali da aggiungere sono

  (is0.true) $is0(zero) = true$ (is0.false) $is0(succ(M)) = false$ (pred) $pred(succ(M)) = M$

  anche in questo caso le riduzioni vanno orientate da sinistra a destra
- Il sottoinsieme dei valori (per la semantica operazionale CBV) e' dato dalla BNF

  $V \in \mathsf{Val} ::= \ldots \mid succ^k(zero)$ con $k \geq 0$
- Le regole da aggiungere per la semantica operazionale CBV a grandi passi sono

  (is0.1) $\dfrac{M \Downarrow_v zero}{is0(M) \Downarrow_v true}$ (is0.2) $\dfrac{M \Downarrow_v succ^{k+1}(zero)}{is0(M) \Downarrow_v false}$ (pred) $\dfrac{M \Downarrow_v succ^{k+1}(zero)}{pred(M) \Downarrow_v succ^k(zero)}$

Anche in questo caso si hanno programmi che che corrispondono a condizioni di errore, in particolare $pred(zero)$ e' della stessa natura della divisione per zero (quindi non individuabile a tempo di compilazione).

3.3 Estensione con operatore di punto fisso

L'operatore di punto fisso fornisce un modo semplice di trattare le definizioni rocorsive. Infatti se $x = P$ e' una definizione ricorsiva (dove x puo' occorre in P, allora il valore a cui x viene legato lo si puo' esprimere come $fix(\lambda x.P)$.

- L'unica operazionE da aggiungereE' fix: 1
- La regola equazionale da aggiungere e'

  (fix) $fix(M) = M\ fix(M)$

  anche in questo caso le riduzioni vanno orientate da sinistra a destra, anche se cio' puo' sembrare controintuitivo.
- Il sottoinsieme dei valori (per la semantica operazionale CBV) rimane invariato

- La regola da aggiungere per la semantica operazionale CBV a grandi passi e'

$$(\mathrm{fix})\ \frac{M\Downarrow_v \lambda x.P \quad [fix(\lambda x.P)/x]P\Downarrow_v V}{fix(M)\Downarrow_v V}$$

Sebbene nel λ-calcolo sia possibile definire termini chiusi che si comportano come operatore di punto fisso, p.e.

$$\begin{array}{rcl} (\lambda f.V\ V) & \text{dove} & V \equiv (\lambda x.f\ x\ x) \\ Z\ Z & \text{dove} & Z \equiv \lambda f, x.f\ (x\ x\ f) \end{array}$$

in pratica risulta piu' conveniente introdurre fix come un operatore primitivo.