λ - Calcolo

Per noi, λ-calcolo = calcolo fondazionale per i linguaggi visti fin'ora.

- Sintassi minimale
- Esprimibilità dei comportamenti di interesse

Il λ-calcolo è un calcolo su termini (term calculus)

- insieme di termini
- sistema equazionale su termini

In realtà, tanti λ-calcoli
- $=_\beta$
- $=_\alpha$
- $=_\eta$
- $=_{\beta\eta}$
- $\vdots$

1. λ-calcolo come teoria equazionale
2. λ calcolo come calcolo fondazionale

Def Insieme Λ dei λ-termini:

$$t, s ::= x \mid \lambda x.t \mid t\, s$$

$\searrow$ variabili appartenenti a un insieme fissato X

```
type lam_exp =
  | Var of ident
  | Abs of ident * lam_exp
  | App of lam_exp * lam_exp
```

convenzioni:

$t_1 t_2 \ldots t_n = ((t_1 t_2) \cdots t_n)$

$\lambda x_1 \ldots x_n . t = \lambda x_1 . (\lambda x_2 . \ldots (\lambda x_n . t) ..)$

$\lambda x. x = \lambda x. x$

Es.

$I \triangleq \lambda x. x$

$K \triangleq \lambda x y. x$

$\Omega \triangleq (\lambda x. x x)(\lambda x. x x)$

Def. La relazione $\alpha \subseteq \Lambda \times \Lambda$ è definita così:

$$\lambda x. t \ \alpha \ \lambda y. t\underbrace{[y/x]} \quad \text{con} \quad y \notin FV(t)$$

dobbiamo definire la sostituzione

Def. La sostituzione è la mappa $subst : \Lambda \times \Lambda \times X \to \Lambda$, notazione $t[s/x]$ per $subst(t, s, x)$, definita da:

$$y[s/x] \triangleq \begin{cases} s & \text{se } x = y \\ y & \text{altrimenti} \end{cases}$$

$$(t_1 t_2)[s/x] \triangleq t_1[s/x] \ t_2[s/x]$$

$$(\lambda y. M)[s/x] \triangleq \begin{cases} \lambda y. M & \text{se } x = y \\ \lambda y. M[s/x] & \text{se } y \notin FV(s) \\ \lambda z. M[z/y][s/x] & \text{se } y \in FV(s), \text{ con } z \notin FV(M) \cup FV(s) \end{cases}$$

$(\lambda y. xy)[\Omega/y]$

$\shortparallel$

$\lambda x. x\Omega$

NB. La sostituzione è capture-avoiding

$$\boxed{x+0 = x}$$

Def. La relazione $\alpha \subseteq \Lambda \times \Lambda$ è definita così:

$$\underbrace{\underline{(x+0)} + \underline{(x+0)}}_{= \underbrace{x+x}}$$

$$\lambda x. t \;\alpha\; \lambda y. t[y/x] \quad \text{con} \quad y \notin FV(t)$$

Più precisamente:

$$t \;\alpha\; s \;\stackrel{\Delta}{\Leftrightarrow}\; \exists t', x, y . \begin{cases} t = \lambda x. t' \\ s = \lambda y. t'[y/x] \\ y \notin FV(t) \end{cases}$$

La relazione di α-equivalenza si ottiene estendendo l'azione di α così:

$$\frac{}{x =_\alpha x} \qquad \frac{t \;\alpha\; s}{t =_\alpha s} \qquad \frac{t =_\alpha s}{\lambda x. t =_\alpha \lambda x. s} \qquad \frac{t_1 =_\alpha s_1 \quad t_2 =_\alpha s_2}{t_1 t_2 =_\alpha s_1 s_2}$$

$$\frac{t =_\alpha s}{s =_\alpha t} \qquad \frac{t =_\alpha s \quad s =_\alpha u}{t =_\alpha u}$$

Possiamo definire in modo "rigoroso" il principio di invarianza rispetto alla scelta di variabili legate (parametri formali).

[Ogni nozione su λ-termini che definiamo deve rispettare la α-equivalenza

Teorema 1 La sostituzione rispetta $=_\alpha$:

- $t =_\alpha s \implies t[^u/_x] =_\alpha s[^u/_x]$
- $t =_\alpha s \;\&\; u =_\alpha v \implies t[^u/_x] =_\alpha s[^v/_x]$

Applicazione. Se codifichiamo un linguaggio basato su substitution-model dentro Λ, $\ulcorner\cdot\urcorner : \mathcal{L} \to \Lambda$
in modo tale che tutti i binder siano riducibili a λ-astrazioni, allora possiamo esprimere il principio di invarianza come invarianza rispetto a $=_\alpha$

Il teorema 1 ci dice che il modello di calcolo è coerente con questo principio.

Es. Definiamo $\mathcal{L}$ come segue:

$$e ::= n \mid b \mid e+e \mid \text{if } e \text{ then } e \text{ else } e$$
$$\mid \text{let } x = e \text{ in } e$$
$$\mid e\,e \mid \text{fun } x \to e$$

Possiamo iniziare a definire una codifica di $\mathcal{L}$ in Λ:

$$\ulcorner n \urcorner \triangleq ?$$
$$\vdots$$
$$\ulcorner \text{let } x = e \text{ in } e' \urcorner = (\lambda x. \ulcorner e' \urcorner) \ulcorner e \urcorner$$
$$\ulcorner e\,e' \urcorner = \ulcorner e \urcorner \ulcorner e' \urcorner$$
$$\ulcorner \text{fun } x \to e \urcorner = \lambda x. \ulcorner e \urcorner$$

Abbiamo che $\ulcorner \text{let } x = e \text{ in } e' \urcorner =_\alpha \ulcorner \text{let } y = e \text{ in } e'[y/x] \urcorner$

$$\ulcorner \text{subst}(e, e', x) \urcorner = \ulcorner e \urcorner [\ulcorner e' \urcorner / x]$$

Def. La <u>relazione</u> $\beta \subseteq \Lambda \times \Lambda$ è definita così:

$$(\lambda x. t)\, s \;\; \beta \;\; t[s/x]$$

La relazioni di β-equivalenza e β-riduzione sono definite così:

- $\dfrac{t \;\beta\; s}{t \to_\beta s} \qquad \dfrac{t \to_\beta s}{\lambda x.t \to_\beta \lambda x.s} \qquad \dfrac{t \to_\beta t'}{t\,s \to_\beta t'\,s} \qquad \dfrac{s \to_\beta s'}{t\,s \to_\beta t\,s'}$

- $\dfrac{}{t \to_\beta^* t} \qquad \dfrac{t \to_\beta s \qquad t \to_\beta^* u}{t \to_\beta^* u}$

- $\dfrac{t \to_\beta^* s}{t =_\beta s} \qquad \dfrac{t =_\beta s}{s =_\beta t}$

$a\, R\, b \qquad b\, R^o\, a$

$x+0 \;\; R \;\; x$

$x \;\; R^o \;\; x+0$

Teorema $=_\beta \; = (\to_\beta \cup \leftarrow_\beta)^*$

↳ uguaglianza per "computazione"

Teorema $\to_\beta$ (e quindi $\to_\beta^*$ e $=_\beta$) rispetta $=_\alpha$

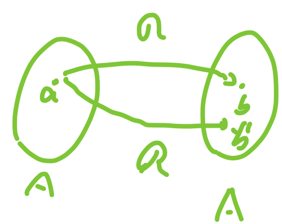
$$t \to_\beta s \;\&\; t =_\alpha t' \implies \exists s'.\; s =_\alpha s' \;\&\; t' \to_\beta s'$$

$R \subseteq A \times A$

$\forall a,b,b' \quad a R b \wedge a R b' \implies b = b'$

$R \subseteq \Delta \qquad R \cup \Delta = \Delta$

$R^\circ ; R \subseteq Id$



Def. Un termine $t \in \Lambda$ è in forma normale se
$$\neg \exists t'.\ t \to_\beta t' \qquad (t \not\to_\beta)$$

Es. · I è in forma normale
· $\lambda y. I$ è <u>una</u> forma normale di $K I$
· Ω <u>non</u> ha forma normale

<u>Domande</u>
- Perché le forme normali sono interessanti?
- Quante forme normali ha un termine?
- Due termini sono β-equivalenti esattamente quando hanno una stessa forma normale?

① Le forme normali sono interessanti perché rappresentano i risultati del calcolo.

Ci si chiede quindi se:

- Un calcolo ha sempre un risultato? No! $\Omega \to_\beta \Omega \to_\beta \cdots$
- Il risultato di un calcolo è necessariamente unico?
- Qual è il rapporto tra $=_\beta$ e "avere lo stesso risultato"

$$\underbrace{t =_{Kl} s \overset{\Delta}{\iff} \exists u . \begin{cases} u \not\to_\beta \\ t \to_\beta^* u \\ s \to_\beta^* u \end{cases}}_{\text{Uguaglianza di Kleene}}$$

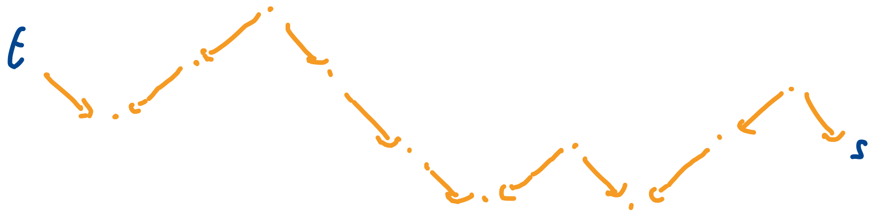
<u>Teorema</u> (Church-Rosser). La relazione $\to_\beta$ è confluente

$$\begin{tikzcd} & t \arrow[dl, "*"'] \arrow[dr, "*"] & \\ t_1 \arrow[dr, dashed, "*"'] & & t_2 \arrow[dl, dashed, "*"] \\ & s & \end{tikzcd}$$

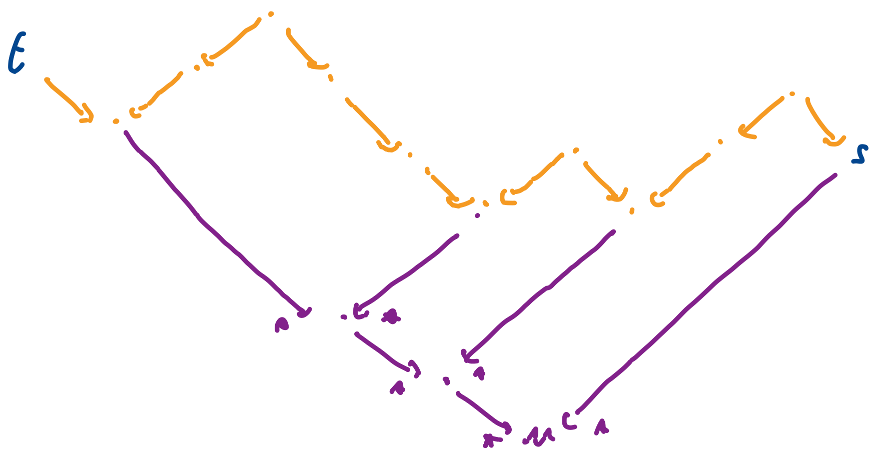
$$t \to_\beta^* t_1 \ \& \ t \to_\beta^* t_2 \implies \exists s .\ t_2 \to_\beta^* s \ \& \ t_1 \to_\beta^* s$$

Corollario. $t =_\beta s$ sse $\exists u.\ t \to_\beta^* u \ \& \ s \to_\beta^* u$

Dim. Se $t =_\beta s$, allora $t \ (\leftarrow_\beta \cup \to_\beta)^* \ s$. quindi:



Applichiamo la confluenza:



Corollario. Se un termine ha una forma normale, questa è necessariamente unica

Osservazioni:

1. Non tutti i termini sono β-equivalenti:

$$K \neq_\beta I$$

2. $=_\lambda \; \cong \; =_\beta$

3. $t =_\beta s$ sse t ed s riducono ad un termine comune

$$\Omega \triangleq (\lambda x. xx)(\lambda x\, xx) \qquad \omega_3 \triangleq (\lambda x. xxx)$$

$$\Omega \to_\beta \Omega \qquad \omega_3\omega_3 \to_\beta \omega_3\omega_3\omega_3 \to_\beta \cdots$$

Quindi: $\Omega \neq_\beta \omega_3\omega_3$

② Forme normali dal punto di vista di $\mathcal{L}$.

$\mathcal{L}$:

- sintassi $\longleftrightarrow$ λ-termini
- programmi $\longleftrightarrow$ λ-termini chiusi
- valori $\longleftrightarrow$ forme normali chiuse
- dinamica $\longleftrightarrow$ $\to_\beta^*$

(teoria equazionale su programmi data da $=_\beta$)

"Teorema". $\forall e, v \in \mathcal{L}$.

$$e \Rightarrow v \quad \text{implica} \quad \ulcorner e \urcorner \to_\beta^* \ulcorner v \urcorner$$

Osservazioni. Abbiamo $\ulcorner v \urcorner \not\to_\beta$?

Abbiamo il viceversa?

Codifica di $\mathcal{L}$ in Λ

Completiamo la codifica di Λ in Λ.

① Come rappresentiamo i numeri? $\rightsquigarrow$ Numerali di Church

$$\ulcorner 0 \urcorner \triangleq \lambda f.\lambda x.\ x$$
$$\ulcorner 1 \urcorner \triangleq \lambda f.\lambda x.\ f\, x$$
$$\ulcorner 2 \urcorner \triangleq \lambda f.\lambda x.\ f\,(f\, x)$$
$$\vdots$$
$$\ulcorner n \urcorner \triangleq \lambda f.\lambda x.\ f^{(n)}\, x$$

$$\begin{cases} f^{(0)}\, x = x \\ f^{(n+1)}\, x = f\,(f^{(n)}\, x) \end{cases}$$

<u>Es</u>. Se definiamo

$$\mathrm{succ} \triangleq \lambda n.\lambda f.\lambda x.\ f\,(n\, x)$$

allora

$$\mathrm{succ}\ \ulcorner n \urcorner =_\beta \ulcorner n+1 \urcorner$$

e

$$\mathrm{succ}\ \ulcorner n \urcorner \to_\beta^* \ulcorner n+1 \urcorner$$

<u>ESERCIZIO</u>. Definire

$$add \triangleq \lambda n.\lambda m.\lambda f.\lambda x.\ n\ f\ (m\ f\ x)$$

$$mult \triangleq \lambda n\ \lambda m.\lambda f.\ n\ (m f)$$

(1) Calcolare (β-ridurre)

$$add\ \ulcorner 2 \urcorner\ \ulcorner 3 \urcorner$$

$$mult\ \ulcorner 2 \urcorner\ \ulcorner 3 \urcorner$$

(2) Notare che $add\ \ulcorner n \urcorner\ \ulcorner m \urcorner \to_\beta^* \ulcorner n + m \urcorner$

$$mult\ \ulcorner n \urcorner\ \ulcorner m \urcorner \to_\beta^* \ulcorner n \cdot m \urcorner$$

Boolean:

$$T \triangleq \lambda xy.x$$

$$F \triangleq \lambda xy.y$$

<u>Esercizio</u>. Studiare

$$\text{and} \triangleq \lambda xy.\, x\, y\, F$$

Far vedere che and codifica la congiunzione Booleana

$$\text{and}\ T\ T \to_{\beta}^{*} T$$
$$\text{and}\ T\ F \to_{\beta}^{*} F$$
$$\text{and}\ F\ T \to_{\beta}^{*} F$$
$$\text{and}\ F\ F \to_{\beta}^{*} F$$

Esercizio.

Definire

$$ite \triangleq \lambda x.x$$

mostrare che

$$ite\ T\ t\ s \to_\beta^* t$$

$$ite\ F\ t\ s \to_\beta^* s$$

Codifica Ricorsione

In λ-calcolo ogni termine ha un punto fisso

Def. Una elemento $x \in A$ è punto fisso di $f : A \to A$ se $f(x) = x$.

Possiamo istanziare questa definizione in λ-calcolo dicendo:

Def. Dati λ-termini t, s, diciamo che s è un punto fisso di t se

$$t\, s =_\beta s$$

In matematica non tutte le funzioni hanno punti fissi.

Es. La funzione $F : \mathbb{N} \to \mathbb{N}$ definita da $F(n) \triangleq n + 1$ non ha punti fissi

Es. La funzione logica $not : bool \to bool$ **non** ha punti fissi:

$$(\neg \exists b.\ not(b) = b)$$

Teorema. In λ-calcolo ogni termine ha un punto fisso

Dim. Definiamo

$$A \triangleq \lambda xy\ y(xxy)$$

$$\Theta \triangleq AA \qquad \text{(Turing's fixed point combinator)}$$

Allora Θt è un punto fisso di t. → termine arbitrario

$$t \to_\beta^* t(\Theta t)$$

(Altra opzione

$$Y \triangleq \lambda f.\ (\lambda x. f(xx))\ (\lambda x. f(xx))$$

↳ Curry's fixed point combinator

$$Y t =_\beta t(Yt)$$
)

Fixed points allow us to solve 'equations'

$$x = F(x)$$

in λ-notazione: $x = (\lambda y . F)\, x$

Possiamo allora dire che ogni equazione della forma

$$x =_\beta F x$$

ha soluzione: basta prendere ΘF

Questo utile per fare ricorsione

Es.

$$\text{fact } n =_\beta \text{ite } (z?\ n)\ \ulcorner 1 \urcorner\ (\text{mult } n\ (\text{fact } (\text{pred } \ulcorner n \urcorner)))$$

test per zero   predecessore

Possiamo astrarre:

$$\text{fact} =_\beta \lambda n.\ \text{ite } (z?\ n)\ \ulcorner 1 \urcorner\ (\text{mult } n\ (\text{fact } (\text{pred } \ulcorner n \urcorner)))$$

Ma allora fact è una soluzione dell'equazione

$$f =_\beta \underbrace{\lambda f.\ \lambda n.\ \text{ite } (z?\ n)\ \ulcorner 1 \urcorner\ (\text{mult } n\ (f\ (\text{pred } \ulcorner n \urcorner)))}_{F}$$

ma possiamo allora definire

$$\text{fact} \triangleq \Theta F$$

Es. Calcoliamo fact [2]

In generale, ogni funzione

$$\text{let rec } f\ x = t$$

può essere codificata come

$$\Theta\,(\lambda f.\lambda x.\ t)$$

In particolare, si ha

$$\Theta\,(\lambda f.\lambda x.t)\ v \longrightarrow_{\beta}^{*} (\lambda f.\lambda x.t)\,(\Theta\,(\lambda f.\lambda x.t))\ v$$

$$\longrightarrow_{\beta}^{*} t\,[{}^{\Theta(\lambda f.\lambda x.t)}/_{f}\,,\ {}^{v}/_{x}]$$

<u>Teorema</u>. Tutte e sole le funzioni computabili sono esprimibili nel λ-calcolo

Call-by-Name & Call-by-Value

$\mathcal{L}$

- sintassi $\longleftrightarrow$ λ-termini
- programmi $\longleftrightarrow$ λ-termini chiusi
- valori $\longleftrightarrow$ forme normali chiuse
- dinamica $\longleftrightarrow$ $\to_\beta^*$

(teoria equazionale su programmi data da $=_\beta$)

"Teorema". $\forall e, v \in \mathcal{L}$.

$e \Rightarrow v$ implica $\ulcorner e \urcorner \to_\beta^* \ulcorner v \urcorner$

Osservazioni.

- Abbiamo $\ulcorner v \urcorner \not\to_\beta$? No!
- Possiamo usare λ-calc per capire esecuzioni programmi? No!
- Possiamo vedere λ-calcolo come una sorta di low-level language! No!

Problemi

① I linguaggi sono lazy / weak. Non si riduce il corpo delle funzioni prima della loro chiamata.

fun $x \to x + 0$

$$\frac{t \to_\beta s}{\lambda x.t \to_\beta \lambda x.s}$$

② I linguaggi usano riduzione deterministica, $\to_\beta$ non lo è

$(\lambda x.\ \text{ite}\ x\ \ulcorner 0 \urcorner\ \ulcorner 1 \urcorner)\ (\text{and}\ (\xi ?\ \ulcorner 0 \urcorner)\ (\xi ?\ \ulcorner 1 \urcorner))$

(ma comunque confluente)

Soluzione

① Definiamo il concetto di: valore

$$v ::= x \mid \lambda x.t$$

NB. Valori chiusi sono funzioni

ed eliminiamo la regola

$$\frac{t \to_\beta s}{\lambda x.t \to_\beta \lambda x.s}$$

$$(\lambda x.t)s \to_{\beta w} t[s/x]$$

$$\frac{t \to_{\beta w} t'}{ts \to_{\beta w} t's} \qquad \frac{s \to_{\beta w} s'}{ts \to_{\beta w} ts'}$$

(ogni tanto si parla di β-weak reduction)

Teorema. ① Ogni valore è in forma βw-normale

$$v \not\to_{\beta w}$$

② Dato un termine chiuso (programma) t, se

$$t \to^*_{\beta w} s \not\to_{\beta w}$$

allora s è un valore

$$\mathcal{L} \xrightarrow{\ulcorner \cdot \urcorner} \lambda\text{-Calcolo}$$

Programmi	$\xrightarrow{\ulcorner \cdot \urcorner}$	Termini chiusi
Valori	$\xrightarrow{\ulcorner \cdot \urcorner}$	Valori
Esecuzione	$\xrightarrow{\ulcorner \, \urcorner}$	$\xrightarrow{*}_{\beta w}$

$p \Downarrow v$ implica $\ulcorner p \urcorner \xrightarrow{*}_{\beta w} \ulcorner v \urcorner$

② Fissiamo una strategia che determinizza βw.

2.1 Call-by-Name

$$(\lambda x.t)\, s \rightarrow_{\beta_N} t[s/x] \qquad \frac{t \rightarrow_{\beta_N} t'}{t\, s \rightarrow_{\beta_N} t'\, s}$$

2.2. Call-by-Value

$$(\lambda x.t)\, v \rightarrow_{\beta_V} t[v/x] \qquad \frac{t \rightarrow_{\beta_V} t'}{t\, s \rightarrow_{\beta_V} t'\, s} \qquad \frac{s \rightarrow_{\beta_V} s'}{v\, s \rightarrow_{\beta_V} v\, s'}$$

passaggio parametro solo quando si raggiungono valori