

Fondamenti: Alcune Proprietà di Calcolabilità e dei Linguaggi per Funzioni Calcolabili

Sommario: 12-13 marzo, 2019

- Funzioni Parziali ed Esecuzioni Terminanti di Programmi
- Problemi Semidecidibili: Halting, Equivalenza, Ambiguità
- Linguaggi per \mathcal{F} : Lambda Calcolo (Church 1936)
- Logica Combinatoria (Shonfinkel 1924)
- Macchina a Stati (Minsky 1956, Wang 1957)
- Calcolo di Processi Mobili: π -Calcolo (Milner et al. 1992)

Lecture e Approfondimenti:

- [Barendregt90] H.P. Barendregt, Functional Programming and Lambda Calculus, in Handbook of Theoretical Computer Science, vol. B, Chapter 7, pp. 321-363, Elsevier Science Publishers, 1990
- [Minsky72] M. Minsky, Computation: Finite and Infinite Machines, Chapter 11, pp. 199-216, Prentice-Hall International, 1972.
- [Milner92] R. Milner, J. Parrow, D. Walker, A Calculus of Mobile Processes, Information and Computations 100, pp. 1-9, 41-49, 1992.

- **Funzione di Decisione** È una funzione booleana, g , totale, ovvero:
 - $g \in \mathcal{D} \rightarrow \{\text{true}, \text{false}\}$ per due valori $\text{true}, \text{false} \in \mathcal{D}$
 - $\forall x \in \mathcal{D}, g(x) \in \{\text{true}, \text{false}\}$
- **Funzione di semi-Decisione** È una funzione parziale g , ovvero:
 - $g \in \mathcal{D} \rightarrow \{\text{true}\}$ per $\text{true} \in \mathcal{D}$
 - $\forall x \in \mathcal{D}, g(x) = \text{true}$ oppure $g(x) = \uparrow$ ¹
- **Decidibilità** Le funzioni di decisione e semi-decisione, sono utilizzate per descrivere problemi calcolabili con caratteristiche diverse.

¹↑ significa *indefinito*: Ogni programma che esprime la funzione, calcolato su tale valore x è non terminante.

Decidibilità di alcuni, importanti problemi

- **Ambiguità di un grammatica Libera.** È semi-decidibile se ambigua, ovvero:
 - esiste semi-decisione g , s.t. $g(x)=\text{true}$ sse x è ambigua
- **Equivalenza di due grammatiche Libere.** È semi-decidibile se diverse, ovvero:
 - esiste semi-decisione g , s.t. $g(x,y)=\text{true}$ sse $\mathcal{L}(x) \neq \mathcal{L}(y)$
 - Equivalenza di grammatiche regolari è decidibile
- **Appartenenza a $\mathcal{L}(G)$ per Libera G** È decidibile
 - esiste decisione g_G , s.t. $g_G(x) = \begin{cases} \text{true} & x \in \mathcal{L}(G) \\ \text{false} & x \notin \mathcal{L}(G) \end{cases}$
- **Terminazione Esecuzione Programma p .** È semi-decidibile se termina, ovvero:
 - esiste semi-decisione g , s.t. $g(p,d)=\text{true}$ sse p termina su input d
- **Equivalenza di Programmi.** È non decidibile
 - eccetto che per programmi terminanti (dove è semi-decidibile la diversità).

Calcolo di Funzioni: Lambda Calcolo (Church 1936)

- **Sintassi** (ovviamente, astratta)

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

- **Termini:** Λ, g, t, t_1 . Insieme (numerabile) dei termini.
 - **Variabili:** X, x, y, z . Insieme (numerabile) di simboli detti variabili.
 - **Costanti:** Π, c, \dots . Insieme (numerabile) di simboli detti costanti (distinguibili: $X \cap \Pi = \{\}$)
 - **Op. Astrazione:** $\lambda x. t$. Operazione binaria che esprime il valore (funzionale) ottenuto da t per *astrazione* (generalizzazione) rispetto alla variabile (libera) x . Il termine è anche detto, funzione di t nella variabile x .
 - **Op. Applicazione:** $t_1 t_2$. Operazione binaria che esprime il valore ottenuto per *applicazione* del termine t_1 al termine t_2 . Il termine è anche detto, applicazione di t_1 a t_2 . Ha notazione implicita via giustapposizione degli argomenti.
- **Esempi**

$[\@ - ([x], [y])]$
 $[\lambda - ([x], [\lambda - ([y], [\@ - ([x], [y])]])]$ (grafica migliora la lettura!)

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

• Sintassi Concreta e Variabili Libere

- **Applicazione.** È associativa a sinistra:
 $x y x$ significa $((x y) x)$
- **Applicazione.** Ha priorità sull'Astrazione:
 $\lambda y. x y$ significa $\lambda y. (x y)$
- **Variabile Legata, Libera, Occorrenze.** Le variabili che occorrono in un termine t sono raccolte in $\text{Var}(t)$, e si dividono Legate, $\text{BV}(t)$, o Libere, $\text{FV}(t)$, a seconda che siano state astratte o meno.

$t \in \Lambda$	$\text{BV}(t)$	$\text{FV}(t)$
$x \in X$	$\{\}$	$\{x\}$
$c \in \Pi$	$\{\}$	$\{\}$
$\lambda x. t_1$	$\{x\} \cup \text{BV}(t_1)$	$\text{FV}(t_1) \setminus \{x\}$
$t_1 t_2$	$\text{BV}(t_1) \cup \text{BV}(t_2)$	$\text{FV}(t_1) \cup \text{FV}(t_2)$
$\text{Var}(t) = \text{BV}(t) \cup \text{FV}(t)$		

• Esempi

Sia $t \equiv \lambda y. \lambda x. y(y x x)$. Possiamo omettere le parentesi?

Sia $t \equiv \lambda y. \lambda x. x z$. Abbiamo: $\text{Var}(t) = \{x, y, z\}$, $\text{FV}(t) = \{z\}$, $\text{BV}(t) = \{x, y\}$.

Sia $t \equiv \lambda y. \lambda x. (\lambda x. x + 5)(x + y)$. Abbiamo: ... 2 simboli di costante

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

• Semantica

- Una relazione \rightarrow definita da 3 regole ed estesa in una congruenza su Λ .

- α – red.

$$\lambda x. t \rightarrow_{\alpha} \lambda y. t[y/x] \quad \text{per } y \notin \text{Var}(t)$$

- β – red.

$$(\lambda x. t)t_2 \rightarrow_{\beta} t[t_2/x] \quad \text{per } \text{FV}(t_2) \cap \text{BV}(t) = \{\}$$

- η – red.

$$\lambda x. (t x) \rightarrow_{\eta} t \quad \text{per } x \notin \text{FV}(t)$$

- Sostituzione. $t_1[t_2/x]$ è l'operazione che rimpiazza, in t_1 , ogni occorrenza libera di x con il termine t_2 .

$$x[t/x] = t$$

$$y[t/x] = y \text{ con } x \neq y; \quad c[t/x] = c$$

$$(\lambda x. t)[t_2/x] = \lambda x. t; \quad (\lambda y. t)[t_2/x] = \lambda y. (t[t_2/x]) \text{ con } x \neq y$$

$$(t_1 t_2)[t_3/x] = (t_1[t_3/x])(t_2[t_3/x])$$

• Esempi

Sia $t \equiv \lambda y. \lambda x. y(y(x))$. Quali regole sono applicabili a t ?

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

- **Valutazione, Computazione** di $t \in \Lambda$

- Una sequenza:

$$t_0 \rightarrow_{r_1} t_1 \quad \dots \quad \rightarrow_{r_n} t_n$$

tale che:

- $(t \equiv t_0) \wedge (n \geq 0)$
- annotazioni $r_i \in \{\alpha, \beta, \eta\}$
- $t_n \rightarrow_{r_{n+1}} t_{n+1}$ solo se $r_{n+1} \equiv \alpha$

- **Esempi**

Sia $t \equiv \lambda x. (\lambda y. x y) (\lambda x. y x)$. Vediamo una valutazione di t :

$$t \rightarrow_{\eta} \lambda x. x (\lambda x. y x) \rightarrow_{\eta} \lambda x. x y$$

Non è unica:

$$t \rightarrow_{\beta} \lambda x. x (\lambda x. y x) \rightarrow_{\eta} \lambda x. x y$$

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

- **Programmi** Sono tutti i termini chiusi, i.e.

$$\mathcal{P} = \{t \in \Lambda \mid FV(t) = \{\}\}$$

- **Aritmetica**

- $[0] \equiv \lambda y. \lambda x. x$
- $[n + 1] \equiv \lambda y. \lambda x. y([n]yx)$
- $\text{succ} \equiv \lambda z. \lambda y. \lambda x. y(zyx)$
- plus, prod, minus, div, ...

- **Conditional e booleans**

- $\text{if} \equiv \lambda b. \lambda x. \lambda y. bxy$
- $\text{true} \equiv \lambda x. \lambda y. x$
- $\text{false} \equiv \lambda x. \lambda y. y$
- zero, and, or, eq, ...

- **Recursion-FixedPoint**

- $\Psi \equiv \lambda g. (\lambda x. g(xx))(\lambda x. g(xx))$

$$\Lambda = X \mid \Pi \mid \lambda X. \Lambda \mid \Lambda \Lambda$$

- **Aritmetica**
- **Programmi**
- **Conditional e costanti booleane**
- **Recursion-FixedPoint**

- $\Psi \equiv \lambda g. (\lambda x. g(xx)) (\lambda x. g(xx))$

- $\forall h \in \mathcal{F}, \forall d \in \mathcal{D},$

$$\Psi(h)(d) = h(\Psi(h))(d) \quad (\text{vedi esercizio L5.7})$$

- Sia $g = \lambda x. t$ una definizione per una funzione g ²:

$$g \equiv \Psi(\lambda g. \lambda x. t)$$

- **Esempi**

$$\text{fact} = \lambda x. \text{if } (\text{eq } x \text{ [0]}) \text{ [1]} (\text{prod } x \text{ (fact (minus } x \text{ [1]))})$$

è una sintassi concreta per il termine

$$\Psi(\lambda \text{fact}. \lambda x. \text{if } (\text{eq } x \text{ [0]}) \text{ [1]} (\text{prod } x \text{ (fact (minus } x \text{ [1]))}))$$

²(possibilmente ricorsiva, i.e. g occorre libera in t)

- **Recursion-FixedPoint**

- $\Psi \equiv \lambda g. (\lambda x. g(xx)) (\lambda x. g(xx))$
- $\forall h \in \mathcal{F}, \forall d \in \mathcal{D},$
 $\Psi(h)(d) = h(\Psi(h))(d)$ (vedi esercizio L5.7)

- **Esempi**

`fact = $\lambda x. \text{if } (\text{eq } x \text{ [0]}) \text{ [1]} (\text{prod } x \text{ (fact (minus } x \text{ [1]))})$`
è una sintassi concreta per il termine

`$\Psi(\lambda \text{ fact. } \lambda x. \text{if } (\text{eq } x \text{ [0]}) \text{ [1]} (\text{prod } x \text{ (fact (minus } x \text{ [1]))}))$`

Calcoliamo: `fact[0]` ovvero: $\Psi(h)[0]$, dove $h \equiv \lambda \text{ fact. } \lambda x. \text{if}..$

$\Psi(h)[0] = h(\Psi(h))[0] \rightarrow_{\beta} (\lambda x. \text{if}..)[0] \rightarrow_{\beta}$
 $\rightarrow_{\beta} \text{if } (\text{eq } [0] \text{ [0]}) \text{ [1]} (\text{prod } x \text{ ((}\Psi(h)\text{)) (minus } x \text{ [1]))})$
 $\rightarrow_{\beta} \dots \rightarrow_{\beta} [1]$ (vedi. esercizio L5.10)

- 1 (a) Si fornisca una grammatica per la sintassi astratta del λ -Calcolo in accordo alla notazione utilizzata nell'albero astratto $[\lambda - ([x], [\lambda - ([y], [@ - ([x], [y])])])]$. (b) Si mostri poi, che tale albero è ottenuto dalla grammatica data.
- 2 (a) Si fornisca una grammatica per la sintassi concreta del λ -Calcolo in accordo alle proprietà date per associatività e precedenza tra operatori. (b) Si mostri poi, la sintassi concreta dell'albero astratto $[\lambda - ([x], [\lambda - ([y], [@ - ([x], [y])])])]$. (c) Si mostri, infine il parse tree del termine ottenuto al punto (b) precedente.
- 3 Si completino i calcoli indicati con '...' nelle slides precedenti.
- 4 Si mostri la sequenza di α -red applicate per ridurre il termine dato sotto, ad un termine contenente sempre, identificatori diversi per variabili diverse:
$$\lambda x. \lambda y. (\lambda x. y (\lambda y. x) x) (\lambda y. x (\lambda x. y x) y)$$
- 5 (a) Si mostri la valutazione di $\lambda x. \lambda y. (\lambda x. y (\lambda y. x) x) (\lambda y. x (\lambda x. y x) y)$. (b) Nell'ipotesi di aver usato α -red nella valutazione fornita se ne giustifichi l'uso.
- 6 Si mostri la valutazione di $(\lambda x. (\lambda y. \lambda x. x y) (\lambda y. y x)) y$
- 7 Si dimostri che: $\forall F, \Psi F = F(\Psi F)$
- 8 (a) Si scriva, in Lambda-Calcolo, un programma per la funzione plus introdotta nell'aritmetica data per tale linguaggio. (b) Si mostri la computazione di `plus[2][1]`
- 9 Si scriva, in Lambda-Calcolo, un programma per la funzione zerop che calcola true quando applicata a [0], false altrimenti. (b) Si mostri la computazione di `zerop[2]`.