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

Sommario: 14-15 marzo, 2017

- Funzioni Parziali e 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)

Approfondimenti:

- 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
- M. Minsky, Computation: Finite and Infinite Machines, Chapter 11, pp. 199-216, Prentice-Hall International, 1972.

Funzioni Parziali e Decidibilità

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

[1] $\uparrow$ significa *indefinito*, ovvero ogni programma che esprime la funzione, quando calcolata 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)=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)=true` sse $\mathcal{L}(\mathtt{x}) \neq \mathcal{L}(\mathtt{x})$
  - Equivalenza di grammatiche regolari è decidibile
- **Appartenenza a $\mathcal{L}(\mathtt{G})$ per Libera G** È decidibile
  - esiste decisione $\mathtt{g_G}$, s.t. $\mathtt{g_G}(\mathtt{x}) = \mathtt{true}$ se $\mathtt{x} \in \mathcal{L}(\mathtt{G})$; $\mathtt{g_G}(\mathtt{x}) = \mathtt{false}$ se $\mathtt{x} \notin \mathcal{L}(\mathtt{G})$
- **Terminazione Esecuzione Programma p.** È semi-decidibile se termina, ovvero:
  - esiste semi-decisione g, s.t. `g(p,d)=true` sse `p` termina su input `d`
- **Equivalenza di Programmi.** È semi-decidibile se diversi, ovvero:
  - ....

Lambda Calcolo (Church 1936)

- **Sintassi** (ovviamente, astratta)

$$\Lambda = \mathtt{X} \mid \Pi \mid \lambda\,\mathtt{X}\,.\,\Lambda \mid \Lambda\Lambda$$

  - **Termini:** $\Lambda, \mathtt{g}, \mathtt{t}, \mathtt{t_i}$. Insieme (numerabile) dei termini.
  - **Variabili:** $\mathtt{X}, \mathtt{x}, \mathtt{y}, \mathtt{z}$. Insieme (numerabile) di simboli detti variabili.
  - **Costanti:** $\Pi, \mathtt{c}\ldots$. Insieme (numerabile) di simboli detti costanti (distinguibili: $\mathtt{X} \cap \Pi = \{\}$)
  - **op. Astrazione.** $\lambda\,\mathtt{x}\,.\,\mathtt{t}$ termine ottenuto da $\mathtt{t}$ per *astrazione* (generalizzazione) rispetto alla variabile (libera) $\mathtt{x}$. Il termine è anche detto, funzione nella variabile $\mathtt{x}$.
  - **op. Applicazione.** $\mathtt{t_1}\mathtt{t_2}$ termine ottenuto per *applicazione* del termine $\mathtt{t_1}$ al termine $\mathtt{t_2}$. Il termine è anche detto, applicazione di $\mathtt{t_1}$ a $\mathtt{t_2}$.

- esempi

  $\lambda\mathtt{x}.\lambda\mathtt{y}.\mathtt{x}\,\mathtt{y}$

  $\lambda\mathtt{x}.\,+\mathtt{x}\,5$, oppure $\lambda\mathtt{x}.\,+(\mathtt{x})(5)$, o $\lambda\mathtt{x}.\mathtt{x}+5$ dipende dalla sintassi concreta

Lambda Calcolo: Convenzioni e Definizioni Sintattiche

$$\Lambda = \mathtt{X} \mid \Pi \mid \lambda\,\mathtt{X}\,.\,\Lambda \mid \Lambda\Lambda$$

- **Sintassi Concreta e Variabili Libere**
  - **op. Applicazione.** è associativa a sinistra:
    $\mathtt{x\,y\,x}$ significa $\mathtt{((x\,y)x)}$
  - **op. Applicazione.** ha priorità sull'astrazione:
    $\lambda\mathtt{y.x\,y}$ significa $\lambda\mathtt{y.(x\,y)}$
  - **Variabile Legata, Libera, Occorrente.** Le variabili che occorrono in un termine $\mathtt{t}$ sono raccolte in $\mathtt{Var(t)}$, e si dividono Legate, $\mathtt{BV(t)}$, o Libere, $\mathtt{FV(t)}$, a seconda che siano state astratte o meno.

$$\begin{array}{ll}
\mathtt{FV(x)} = \{\mathtt{x}\} & \mathtt{Var(x)} = \{\mathtt{x}\} \\
\mathtt{FV(c)} = \{\} & \mathtt{Var(c)} = \{\} \\
\mathtt{FV}(\lambda\,\mathtt{x\,.\,t}) = \mathtt{FV(t)} \setminus \{\mathtt{x}\} & \mathtt{Var}(\lambda\,\mathtt{x\,.\,t}) = \mathtt{Var(t)} \cup \{\mathtt{x}\} \\
\mathtt{FV}(\mathtt{t_1 t_2}) = \mathtt{FV}(\mathtt{t_1}) \cup \mathtt{FV}(\mathtt{t_2}) & \mathtt{Var}(\mathtt{t_1 t_2}) = \mathtt{Var}(\mathtt{t_1}) \cup \mathtt{Var}(\mathtt{t_2})
\end{array}$$

$$\mathtt{BV(t)} = \mathtt{Var(t)} \setminus \mathtt{FV(t)}.$$

- esempi
  $\lambda\mathtt{y}.\lambda\mathtt{x}.\mathtt{y}(\mathtt{y}\,\mathtt{x}))$
  Sia $\mathtt{t} \equiv \lambda\mathtt{y}.\lambda\mathtt{x}.\mathtt{xz}$. $\mathtt{Var(t)=\{x,y,z\}}$, $\mathtt{FV(t)=\{z\}}$, $\mathtt{BV(t)=\{x,y\}}$.
  Sia $\mathtt{t} \equiv \lambda\mathtt{y}.\lambda\mathtt{x}.(\lambda\mathtt{x}.\mathtt{x}+5)(\mathtt{x}+\mathtt{y})$. ...

Lambda Calcolo: Semantica

$$\Lambda = \mathtt{X} \mid \Pi \mid \lambda\,\mathtt{X}\,.\,\Lambda \mid \Lambda\Lambda$$

- **Semantica**
  - Una relazione $\rightarrow$ definita da 3 regole ed estesa in una congruenza su Λ.
  - $\alpha -$ **red**.
    $$\lambda\,\mathtt{x}\,.\,\mathtt{t} \rightarrow \lambda\,\mathtt{y}\,.\,\mathtt{t}[\mathtt{y}/\mathtt{x}] \text{ per } \mathtt{y} \notin \mathtt{Var}(\mathtt{t})$$
  - $\beta -$ **red**.
    $$(\lambda\,\mathtt{x}\,.\,\mathtt{t})\mathtt{t}_2 \rightarrow \mathtt{t}[\mathtt{t}_2/\mathtt{x}] \text{ per } \mathtt{FV}(\mathtt{t}_2) \cap \mathtt{BV}(\mathtt{t}) = \{\}$$
  - $\eta -$ **red**.
    $$\lambda\,\mathtt{x}\,.(\mathtt{t}\ \mathtt{x}) \rightarrow \mathtt{t} \text{ per } \mathtt{x} \notin \mathtt{FV}(\mathtt{t})$$
  - Sostituzione. $\mathtt{t}_1[\mathtt{t}_2/\mathtt{x}]$ è l'operazione che rimpiazza, in $\mathtt{t}_1$, ogni occorrenza libera di $\mathtt{x}$ con il termine $\mathtt{t}_2$.
    $$\mathtt{x}[\mathtt{t}/\mathtt{x}] = \mathtt{t}$$
    $$\mathtt{y}[\mathtt{t}/\mathtt{x}] = \mathtt{y} \text{ con } \mathtt{x} \neq \mathtt{y}; \qquad \mathtt{c}[\mathtt{t}/\mathtt{x}] = \mathtt{c}$$
    $$(\lambda\,\mathtt{x}\,.\,\mathtt{t})[\mathtt{t}_2/\mathtt{x}] = \lambda\,\mathtt{x}\,.\,\mathtt{t}; \qquad (\lambda\,\mathtt{y}\,.\,\mathtt{t})[\mathtt{t}_2/\mathtt{x}] = \lambda\,\mathtt{y}\,.\,(\mathtt{t}[\mathtt{t}_2/\mathtt{x}]) \text{ con } \mathtt{x} \neq \mathtt{y}$$
    $$(\mathtt{t}_1\mathtt{t}_2)[\mathtt{t}_3/\mathtt{x}] = (\mathtt{t}_1[\mathtt{t}_3/\mathtt{x}])(\mathtt{t}_2[\mathtt{t}_3/\mathtt{x}])$$
- esempi
  $\lambda\mathtt{y}.\lambda\mathtt{x}.\mathtt{y}(\mathtt{y}\,\mathtt{x}))$

Lambda Calcolo: Programmare

$$\Lambda = \mathtt{X} \mid \Pi \mid \lambda\,\mathtt{X}\,.\,\Lambda \mid \Lambda\Lambda$$

- **Programmi** Sono tutti i termini chiusi, i.e.
$$\mathcal{P} = \{\mathtt{t} \in \Lambda \mid \mathtt{FV}(\mathtt{t}) = \{\}\}$$
- **Aritmetica**
  - $[0] \equiv \lambda\,\mathtt{y}.\lambda\,\mathtt{x}.\mathtt{x}$
  - $[\mathtt{n}+1] \equiv \lambda\,\mathtt{y}.\lambda\,\mathtt{x}.\mathtt{y}([\mathtt{n}]\mathtt{yx})$
  - $\mathtt{succ} \equiv \lambda\,\mathtt{z}.\lambda\,\mathtt{y}.\lambda\,\mathtt{x}.\mathtt{y}(\mathtt{zyx})$
  - `plus, prod, minus, div`
- **Conditional e booleani**
  - $\mathtt{if} \equiv \lambda\,\mathtt{b}.\lambda\,\mathtt{x}.\lambda\,\mathtt{y}.\,\mathtt{b}\,\mathtt{x}\,\mathtt{y}$
  - $\mathtt{true} \equiv \lambda\,\mathtt{x}.\lambda\,\mathtt{y}.\mathtt{x}$
  - $\mathtt{false} \equiv \lambda\,\mathtt{x}.\lambda\,\mathtt{y}.\mathtt{y}$
  - $\mathtt{and}, \mathtt{or}, \mathtt{eq}, \ldots$
- **Recursion-FixedPoint**
  - $\Psi \equiv \lambda\,\mathtt{g}.(\lambda\,\mathtt{x}.\mathtt{g}(\mathtt{xx}))(\lambda\,\mathtt{x}.\mathtt{g}(\mathtt{xx}))$

Lambda Calcolo: Programmare

$$\Lambda = \mathtt{X} \mid \Pi \mid \lambda\,\mathtt{X}\,.\,\Lambda \mid \Lambda\Lambda$$

- **Aritmetica**
- **Programmi**
- **Conditional e costanti booleane**
- **Recursion-FixedPoint**
  - $\Psi \equiv \lambda\,\mathtt{g}.(\lambda\,\mathtt{x}.\mathtt{g}(\mathtt{xx}))(\lambda\,\mathtt{x}.\mathtt{g}(\mathtt{xx}))$
  - $\forall \mathtt{h} \in \mathcal{F}, \forall \mathtt{d} \in \mathcal{D},$
    $$\Psi(\mathtt{h})(\mathtt{d}) = \mathtt{h}(\Psi(\mathtt{h}))(\mathtt{d})$$
  - Sia $\mathtt{g} = \lambda\,\mathtt{x}.\mathtt{t}$ una definizione per una funzione g [2]:
    $$\mathtt{g} \equiv \Psi(\lambda\,\mathtt{g}.\lambda\,\mathtt{x}.\mathtt{t})$$
- Esempio

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

  è una sintassi concreta per il termine

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

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

Logica Combinatoria (Schonfinkel 1924)

- **Sintassi** (ovviamente, astratta)

$$\mathcal{C} = \mathtt{X} \mid \Pi \mid \mathtt{S} \mid \mathtt{K} \mid \mathtt{I} \mid \mathcal{C}\mathcal{C}$$

  - **Termini:** $\mathcal{C}, \mathtt{r}, \mathtt{r_i}$. Insieme (numerabile) dei termini.
  - **Variabili:** $\mathtt{X}, \mathtt{x}, \mathtt{y}, \mathtt{z}$. Insieme (numerabile) di simboli detti variabili libere.
  - **Costanti:** $\Pi, \mathtt{c}\ldots$. Insieme (numerabile) di simboli detti costanti (distinguibili: $\mathtt{X} \cap \Pi = \{\}$)
  - **S, K, I** Sono tre identificatori , i.e. $\mathtt{S}, \mathtt{K}, \mathtt{I} \in \Pi$, detti combinatori fondamentali
  - **op. Applicazione.** $\mathtt{r_1 r_2}$ termine ottenuto per *applicazione* del termine $\mathtt{r_1}$ al termine $\mathtt{r_2}$. Il termine è anche detto, applicazione di $\mathtt{r_1}$ a $\mathtt{r_2}$.

- Esempi

  `S(Kx)I`
  `S(S(KS)(S(KK)I))(KI)`

$$\mathcal{C} = \mathtt{X} \mid \Pi \mid \mathtt{S} \mid \mathtt{K} \mid \mathtt{I} \mid \mathcal{C}\mathcal{C}$$

- **Sintassi Concreta e Variabili Libere**
  - **op. Applicazione.** è associativa a sinistra:
    $\mathtt{x}\,\mathtt{y}\,\mathtt{x}$ significa $((\mathtt{x}\,\mathtt{y})\mathtt{x})$
  - **Variabile Libera.** Le variabili che occorrono in un termine sono tutte libere.
- esempi
  `S(Kx)I`
  `S(S(K+)I)(K5)`

Logica Combinatoria: Semantica

$$\mathcal{C} = \mathtt{X} \mid \Pi \mid \mathtt{S} \mid \mathtt{K} \mid \mathtt{I} \mid \mathcal{C}\mathcal{C}$$

- **Semantica**
  - Una relazione $\rightarrow$ definita da 1 regola per combinatore ed estesa in una congruenza su $\mathcal{C}$..
    - $\mathtt{S}\ \mathtt{r}_1\ \mathtt{r}_2\ \mathtt{r} \rightarrow \mathtt{r}_1\ \mathtt{r}\ (\mathtt{r}_2\ \mathtt{r})$
    - $\mathtt{K}\ \mathtt{r}_1\ \mathtt{r}_2 \rightarrow \mathtt{r}_1$
    - $\mathtt{I}\ \mathtt{r} \rightarrow \mathtt{r}$
- esempi

  $\mathtt{S(Kx)I\,3} \rightarrow \ldots$
  $\mathtt{S(S(K+)I)(K\,5)7} \rightarrow \ldots$
  $\mathtt{S(S(KS)(S(KK)I))(KI)3\,5} \rightarrow ..$

Logica Combinatoria: Astrazione/1

$$\mathcal{C} = \mathtt{X} \mid \Pi \mid \mathtt{S} \mid \mathtt{K} \mid \mathtt{I} \mid \mathcal{C}\mathcal{C}$$

- Sia $\mathcal{U}$ un generico calcolo con termini $\mathtt{F}$ per esprimere funzioni e valori, variabili $\mathtt{X}$ per e. parametri [3], applicazione di termini per e. l'applicazione di funzione.[4]

Definition (Astratto, Proprietà di Astrazione)

L'astratto di un termine $\mathtt{u} \in \mathtt{F}$, rispetto ad una variabile $\mathtt{x} \in \mathtt{X}$, è un termine che indichiamo con $\{\mathtt{x}\}\mathtt{u}$ avente la seguente proprietà di astrazione:

$$\forall \mathtt{w} \in \mathtt{F}, \quad (\{\mathtt{x}\}\mathtt{u})\mathtt{w} = \mathtt{u}\,[\mathtt{w}/\mathtt{x}]$$

dove $(\{\mathtt{x}\}\mathtt{u})\mathtt{w}$ è l'applicazione di $\{\mathtt{x}\}\mathtt{u}$ a $\mathtt{w}$, $\mathtt{u}\,[\mathtt{w}/\mathtt{x}]$ è la sostituzione, in $\mathtt{u}$, di $\mathtt{x}$ con $\mathtt{w}$.

- Nel Lambda-Calcolo, $\{\mathtt{x}\}\mathtt{u}$ è il termine $\lambda\,\mathtt{x}.\mathtt{u}$ e fa parte di una classe speciale di termini.
- $\mathcal{C}$ non ha una classe speciale di termini.

Proposition (Astratto in $\mathcal{C}$)

$\mathcal{C}$ esprime l'astratto mediante una composizione di applicazioni dei suoi termini, ovvero: $\forall \mathtt{r} \in \mathcal{C}, \forall \mathtt{x} \in \mathtt{X}, \{\mathtt{x}\}\mathtt{r} \in \mathcal{C}$

[3] si assuma per semplicità, che variabili diverse abbiano sempre nomi diversi in uno stesso termine

[4] il calcolo può contenere ulteriori operazioni e strutture

Logica Combinatoria: Astrazione/2

$$\mathcal{C} = \mathtt{X} \mid \Pi \mid \mathtt{S} \mid \mathtt{K} \mid \mathtt{I} \mid \mathcal{C}\mathcal{C}$$

Proposition (Astratto in $\mathcal{C}$)

$\mathcal{C}$ esprime l'astratto mediante una composizione di applicazioni dei suoi termini, ovvero: $\forall \mathtt{r} \in \mathcal{C}, \forall \mathtt{x} \in \mathtt{X}, \{\mathtt{x}\}\mathtt{r} \in \mathcal{C}$

- **Astrazione.** Dato un qualunque termine $\mathtt{r}$ e una qualunque variabile $\mathtt{x}$, mostriamo un metodo [5] per ottenere $\{\mathtt{x}\}\mathtt{r} \in \mathcal{C}$.
  - $\{\mathtt{x}\}\mathtt{x} = \mathtt{I}$
  - $\{\mathtt{x}\}\mathtt{r} = \mathtt{K}\ \mathtt{r}$ se $\mathtt{x} \notin \mathtt{Var}(\mathtt{r})$
  - $\{\mathtt{x}\}(\mathtt{r}_1\ \mathtt{r}_2) = \mathtt{S}\,(\{\mathtt{x}\}\mathtt{r}_1)(\{\mathtt{x}\}\mathtt{r}_2)$
- Esempio

  $\{\mathtt{x}\}(\{\mathtt{y}\}\mathtt{x}) = \ldots$

[5] Si noti che $\{\mathtt{x}\}\mathtt{r}$ può essere espresso con termini equivalenti ma diversi, pertanto esistono altri metodi per calcolare per l'astratto in $\mathcal{C}$

Logica Combinatoria: Programmare con liste

$$\mathcal{C} = \mathtt{X} \mid \Pi \mid \mathtt{S} \mid \mathtt{K} \mid \mathtt{I} \mid \mathcal{C}\mathcal{C}$$

- **Programmi** Sono tutti i termini chiusi, i.e. senza variabili
$$\mathcal{P} = \{\mathtt{r} \in \mathcal{C} \mid \mathtt{Var}(\mathtt{r}) = \{\}\}$$
- **Conditional e booleani**
  - $\mathtt{if} \equiv \{\mathtt{b}\}(\{\mathtt{x}\}(\{\mathtt{y}\}\mathtt{bxy}))$
  - $\mathtt{true} \equiv \mathtt{S}(\mathtt{KK})\mathtt{I}$
  - $\mathtt{false} \equiv \mathtt{KI}$
  - $\mathtt{and}, \mathtt{or}, \mathtt{eq}, \ldots$
- **Coppie e liste**
  - $\mathtt{pair} \equiv \{\mathtt{x}\}(\{\mathtt{y}\}(\{\mathtt{s}\}\mathtt{sxy}))$
  - $\mathtt{first\ r} \equiv \mathtt{r\ true}$
  - $\mathtt{snd\ r} \equiv \mathtt{r\ false}$

Logica Combinatoria: Programmare con Aritmetica

$$\mathcal{C} = \mathtt{X} \mid \Pi \mid \mathtt{S} \mid \mathtt{K} \mid \mathtt{I} \mid \mathcal{C}\mathcal{C}$$

- **Programmi** Sono tutti i termini chiusi, i.e. senza variabili
$$\mathcal{P} = \{\mathtt{r} \in \mathcal{C} \mid \mathtt{Var}(\mathtt{r}) = \{\}\}$$
- **Conditional e booleani**
- **Coppie e liste**
- **Aritmetica**
  - $[0] \equiv \mathtt{pair\ true\ false}$
  - $[\mathtt{n}+1] \equiv \mathtt{pair\ false}\ [\mathtt{n}]$
  - $\mathtt{zerop\ r} \equiv \mathtt{r\ true}$
  - `succ, plus, prod, minus, div`
- **Recursion-FixedPoint**
  - $\Psi \equiv \ldots$

Logica Combinatoria: Conclusioni (per ora)

$$\mathcal{C} = \mathtt{X} \mid \Pi \mid \mathtt{S} \mid \mathtt{K} \mid \mathtt{I} \mid \mathcal{C}\mathcal{C}$$

- **Programmi** Sono tutti i termini chiusi, i.e. senza variabili
- **Aritmetica e Liste**
- **Conditional e booleani**
- **Recursion-FixedPoint**
- **Un monoide** con applicazione come unica operazione
- **Il più compatto** formalismo di calcolo
- **Il più compatto** linguaggio di programmazione
  - esecutore semplice da realizzare (no sostituzione)
  - valori e operazioni espressi nel linguaggio (inclusa ricorsione) possono essere realizzati come primitive della sua Macchina Astratta
- Espressività: Rendere Primitivi i meccanismi di uso comune (valori, ricorsione,..)

Macchina a Stato (Minsky 1956, Wang 1957)

- **Sintassi** (ovviamente, astratta)

  $\mathcal{Z} = \texttt{Zero}\,\texttt{R} \mid \texttt{Inc}\,\texttt{R} \mid \texttt{DJ0}\,\texttt{R}\,\texttt{N} \mid \texttt{Halt}$

  - **Termini:** $\mathcal{Z}$ Insieme (numerabile) degli Statements, $\texttt{z}, \texttt{z}_\texttt{i}$.
  - **Registri:** $\texttt{R}, \texttt{R}_\texttt{i}$. Insieme finito di Registri a *capacità infinita*.
  - **N:** interi nonegativi
  - **Programmi,** $\mathcal{P}$**:** Ogni sequenza finita di $\mathcal{Z}$ terminante con `Halt`

- Esempi

  $\texttt{Zero}\,\texttt{R}$

  $\texttt{Zero}\,\texttt{R}; \texttt{DJ0}\,\texttt{R}_0\ 4; \texttt{Inc}\,\texttt{R}_1; \texttt{DJ0}\,\texttt{R}\ 1; \texttt{Halt}$

Macchina a Stato: Semantica

$\mathcal{Z} = \texttt{Zero}\,\texttt{R} \mid \texttt{Inc}\,\texttt{R} \mid \texttt{DJ0}\,\texttt{R}\,\texttt{N} \mid \texttt{Halt}$

- Semantica SOS
  - Definizione di **stato**
    - Registri (variabili con nomi predefiniti) rappresentati da una sequenza $\texttt{R}$ della forma: $(\texttt{R}_0, \texttt{n}_0)...(\texttt{R}_\texttt{k}, \texttt{n}_\texttt{k})$
    - Program Counter: Posizione $\texttt{PC}$ del successivo statement da eseguire (a partire da 0).
    - Lo stato è rappresentato dalla coppia $\{\texttt{PC}, \texttt{R}\}$
  - Definizione di **Transizione** $\rightarrow$
    - Esecuzione di Statement $\texttt{z}$ nello stato σ della macchina
      - $\langle \texttt{z}, \sigma \rangle \rightarrow \tau$, indicante ...
      - $\langle \texttt{z}, \sigma \rangle \rightarrow \langle \texttt{z}', \sigma' \rangle$, indicante ...

Macchina a Stato: Computazione

$\mathcal{Z} = \texttt{Zero}\,\texttt{R} \mid \texttt{Inc}\,\texttt{R} \mid \texttt{DJO}\,\texttt{R}\,\texttt{N} \mid \texttt{Halt}$

- Semantica SOS
  - Definizione di **stato**
  - Definizione di **Transizione**
  - Definizione di **Computazione** di un programma di $\texttt{p} \in \mathcal{P}$:
    - sequenza componente registro degli stati attraversati nell'esecuzione di $\texttt{p}$.

$$\langle \texttt{z}, \{\texttt{n}, \sigma\}\rangle \rightarrow \langle \texttt{z}_1, \{\texttt{n}_1, \sigma_1\}\rangle \rightarrow \langle \texttt{z}_2, \{\texttt{n}_2, \sigma_2\}\rangle \rightarrow \ldots$$

    - La semantica di un programma è la sua computazione:

$$\texttt{Sem}(\texttt{p}) \equiv \sigma, \sigma_1, \sigma_2, \ldots,.$$

- **Notazione**.
  - $\{\texttt{n}, (\texttt{R}_1, \texttt{n}_1), \cdots, (\texttt{R}_\texttt{k}, \texttt{n}_\texttt{k})\}$ stato con $\texttt{PC} \equiv \texttt{n}$ e k registri
  - σ, τ (anche con pedici) sono componente registro dello stato
  - $\sigma(\texttt{R}_\texttt{i}) = 0$ quando $\sigma = (\texttt{R}_1, \texttt{n}_1), \cdots, (\texttt{R}_\texttt{i}, 0), \cdots, (\texttt{R}_\texttt{k}, \texttt{n}_\texttt{k})$,
  - $\sigma(\texttt{R}_\texttt{i}) \neq 0$ quando $\sigma = (\texttt{R}_1, \texttt{n}_1), \cdots, (\texttt{R}_\texttt{i}, \texttt{n}_\texttt{i} \neq 0), \cdots, (\texttt{R}_\texttt{k}, \texttt{n}_\texttt{k})$,
  - $\sigma[\texttt{R}_\texttt{i} \leftarrow 0] = (\texttt{R}_1, \texttt{n}_1), \cdots, (\texttt{R}_\texttt{i}, 0), \cdots, (\texttt{R}_\texttt{k}, \texttt{n}_\texttt{k})$
  - $\sigma[\texttt{R}_\texttt{i} \leftarrow \texttt{R}_\texttt{i} + 1] = (\texttt{R}_1, \texttt{n}_1), \cdots, (\texttt{R}_\texttt{i}, \texttt{n}_\texttt{i} + 1), \cdots, (\texttt{R}_\texttt{k}, \texttt{n}_\texttt{k})$
    quando $\sigma = (\texttt{R}_1, \texttt{n}_1), \cdots, (\texttt{R}_\texttt{i}, \texttt{n}_\texttt{i}), \cdots, (\texttt{R}_\texttt{k}, \texttt{n}_\texttt{k})$
  - $\texttt{P}[\texttt{n}]$ n-esimo statement del programma P
  - $\#\texttt{P}$ numero di statements del programma (posizionati da 0 a $\#\texttt{P} - 1$)

Macchina a Stato: Transizioni

$$\mathcal{Z} = \texttt{Zero}\,\texttt{R} \mid \texttt{Inc}\,\texttt{R} \mid \texttt{DJ0}\,\texttt{R}\,\texttt{N} \mid \texttt{Halt}$$

$$\langle \texttt{Halt}, \{\texttt{n}, \sigma\}\rangle \rightarrow \{\texttt{n}, \sigma\}$$

$$\frac{0 \leq \texttt{n} < \#\texttt{P}, \quad \texttt{z}_\texttt{n} = \texttt{P}[\texttt{n}]}{\langle \texttt{Zero}\,\texttt{R}_\texttt{i}, \{\texttt{n}, \sigma\}\rangle \rightarrow \langle \texttt{z}_\texttt{n}, \{\texttt{n}+1, \sigma[\texttt{R}_\texttt{i} \leftarrow 0]\}\rangle}$$

$$\frac{0 \leq \texttt{n} < \#\texttt{P}, \quad \texttt{z}_\texttt{n} = \texttt{P}[\texttt{n}]}{\langle \texttt{Inc}\,\texttt{R}_\texttt{i}, \{\texttt{n}, \sigma\}\rangle \rightarrow \langle \texttt{z}_\texttt{n}, \{\texttt{n}+1, \sigma[\texttt{R}_\texttt{i} \leftarrow \texttt{R}_\texttt{i}+1]\}\rangle}$$

$$\frac{0 \leq \texttt{m} < \#\texttt{P}, \quad \texttt{z}_\texttt{m} = \texttt{P}[\texttt{m}], \quad \sigma(\texttt{R}_\texttt{i}) = 0}{\langle \texttt{DJ0}\,\texttt{R}_\texttt{i}\,\texttt{m}, \{\texttt{n}, \sigma\}\rangle \rightarrow \langle \texttt{z}_\texttt{m}, \{\texttt{m}+1, \sigma\}\rangle} \qquad \frac{0 \leq \texttt{n} < \#\texttt{P}, \quad \texttt{z}_\texttt{n} = \texttt{P}[\texttt{n}], \quad \sigma(\texttt{R}_\texttt{i}) \neq 0}{\langle \texttt{DJ0}\,\texttt{R}_\texttt{i}\,\texttt{m}, \{\texttt{n}, \sigma\}\rangle \rightarrow \langle \texttt{z}_\texttt{n}, \{\texttt{n}+1, \sigma[\texttt{R}_\texttt{i} \leftarrow \texttt{R}_\texttt{i}-1]\}\rangle}$$

Esercizi.

Esercizio

Si dia una semantica SOS per il Lambda Calcolo

Soluzione

Esercizi.

Esercizio

Si completino i calcoli indicati con '...' nelle slides precedenti.

Soluzione

Esercizi.

Esercizio

Si mostri la sequenza di $\alpha - \texttt{red}$ *applicate per ridurre il termine dato ad un termine contenente sempre, identificatori diversi per variabili diverse:* $\lambda\,\mathtt{x}.\lambda\,\mathtt{y}.(\lambda\,\mathtt{x}.\mathtt{y}(\lambda\,\mathtt{y}.\mathtt{x})\mathtt{x})(\lambda\,\mathtt{y}.\mathtt{x}(\lambda\,\mathtt{x}.\mathtt{yx})\mathtt{y})$

Soluzione

Esercizi.

Esercizio

Si dimostri che: $\forall \mathtt{F}, \quad \Psi\,\mathtt{F} = \mathtt{F}(\Psi\,\mathtt{F})$

Soluzione

Esercizi.

Esercizio

Si scriva, in Lambda-Calcolo, un programma per la funzione `prod` *introdotta nell'aritmetica data per tale linguaggio.*

Soluzione

Esercizi.

Esercizio

Si scriva, in Lambda-Calcolo, un programma per la funzione `zerop` *che calcola il test su zero nell'aritmetica data per tale linguaggio.*

Soluzione

Esercizi.

Esercizio

Si dia una semantica SOS per la Logica Combinatoria

Soluzione

Esercizi.

Esercizio

Si dimostri che `S K K = I`

Soluzione

Esercizi.

Esercizio

Si scriva, in Logica Combinatoria, un programma per la funzione `plus` *introdotta nell'aritmetica data per tale linguaggio.*

Soluzione

Esercizi.

Esercizio

Si scriva, in Logica Combinatoria, un programma per la funzione `minus` *introdotta nell'aritmetica data per tale linguaggio.*

Soluzione