

INFORMATICA TEORICA

giosumarin

March 2021

Contents

1	Lezione 1	2
1.1	Definizione di funzione	2
1.2	Funzione iniettiva, suriettiva e biettiva	2
2	Lezione 2	3
3	Lezione 3	5
3.1	\mathbb{R} non è numerabile	5
3.2	Cosa è calcolabile?	7
4	Lezione 4	7
4.1	$Dati \sim \mathbb{N}$	7
4.1.1	$DATI \sim \mathbb{N}$	9
5	Lezione 5	9
5.1	Ingredienti della definizione formale di semantica	11
6	Lezione 6	12
6.1	$PROG \sim \mathbb{N}$ su programmi RAM	12
6.2	Come aritmetizzare?	12
6.3	Sistema di calcolo $WHILE$	13
6.3.1	Dimostrazioni induttive su alberi bianchi	14
7	Lezione 7	15
7.1	Esecuzione su una macchina $WHILE$ (intuitivamente)	15
7.2	Definizione formale di semantica di un programma $WHILE$	15
7.3	Potenza computazionale sistema $WHILE$	16
7.3.1	Relazione tra $F(RAM)$ e $F(WHILE)$?	16
7.3.2	Confronto tra due sistemi di calcolo	16
7.4	Concetto di traduttore	17

1 Lezione 1

1.1 Definizione di funzione

Funzione: una legge/regola che ci dice come associare un elemento di A a uno di B .

Definizione globale: $f : A \rightarrow B$: chiamiamo A il dominio della funzione e B il codominio.

Definizione locale: $a \rightarrow^f b$ oppure $f(a) = b$ con b immagine di a rispetto a f e a controimmagine di b rispetto a f .

$f : \mathbb{N} \rightarrow \mathbb{N}$ con $\mathbb{N} = \{0, 1, 2, 3, 4, \dots\}$ e con $\mathbb{N}^+ = \{1, 2, 3, 4, \dots\}$

Globale: $f(n) = \lfloor \sqrt{n} \rfloor$; Locale: $f(5) = \lfloor \sqrt{5} \rfloor$ In una funzione, per definizione, un valore del dominio può portare a uno solo valore di codominio.

1.2 Funzione iniettiva, suriettiva e biettiva

Funzione Iniettiva

$$f : A \rightarrow B \text{ è iniettiva sse } \forall a_1, a_2 \in A \Rightarrow f(a_1) \neq f(a_2)$$

ovvero non ci sono confluenze verso un punto del codominio.

Funzione Suriettiva

$$f : A \rightarrow B \text{ è suriettiva sse } \forall b \in B, \exists a \in A : f(a) = b$$

Definiamo con Im_f l'insieme delle immagini. Quindi

$$\{Im_f = b \in B : \exists a.t.c.f(a) = b\} = \{f(a), a \in A\}$$

Possiamo quindi dire che in generale $Im_f \subseteq B$ ed è suriettiva sse $Im_f = B$, ovvero quando il grafico della funzione copre tutto l'asse y .

Funzione Biettiva Una funzione si dice biettiva quando è sia iniettiva che suriettiva, ovvero

$$\forall b \in B \exists! a \in A : f(a) = b$$

dove con $\exists!$ indichiamo "esiste unico".

Composizione di funzioni Nota: non è commutativo

$$f : A \rightarrow B$$

$$g : B \rightarrow C$$

$$f \text{ composto } g: g \cdot f : A \rightarrow C$$

$$\text{definita come } g \cdot f(a) = g(f(a))$$

Funzione Inversa

$$f : A \rightarrow B \text{ biettiva} \\ f^{-1} : B \rightarrow A \text{ t.c. } f^{-1}(b) = a \leftrightarrow f(a) = b$$

Definiamo

$$i_A : A \rightarrow A \text{ con } i_A(a) = a$$

che ci permette di dare una definizione ulteriore di funzione inversa combinando la funzione identità e la composizione

$$f^{-1} \cdot f = i_A \wedge f \cdot f^{-1} = i_B$$

2 Lezione 2

$$f(a) \downarrow: f \text{ definita } \forall a \in A \text{ si dice che } f \text{ è totale} \\ f(a) \uparrow: \text{ non definita per ogni } a \in A.$$

$f : A \rightarrow B$ è parziale se qualche elemento di A associa un elemento di AB , infatti:

$$Dom_f = \{a \in A : f(a) \downarrow\} \subseteq A \\ Dom_f \subsetneq A \Rightarrow f \text{ parziale (incluso stretto)} \\ Dom_f = A \Rightarrow f \text{ totale}$$

Totalizzare

$$f : A \rightarrow B \text{ parziale} \Rightarrow \tilde{f} : A \rightarrow B \cup \{\perp\} \text{ totale,} \\ \text{Indichiamo } B \cup \{\perp\} \rightarrow B_{\perp} \\ \tilde{f} = \begin{cases} f(a) & \text{se } a \in Dom_f \\ \perp & \text{altrimenti} \end{cases}$$

Prodotto Cartesiano

$$A \times b = \{(a, b) : a \in A \wedge b \in B\} \\ \text{Nota: } \times \text{ non commutativa } A \times B \neq B \times A \\ \text{Proiettore -iesimo } \pi_i : A_1 \times \dots \times A_n \rightarrow A_i \\ \pi_i(a_1, \dots, a_n) = a_i \\ \text{Indichiamo } A \text{ per } n \text{ volte come } A \times \dots \times A = A^n$$

Insieme di funzioni

$$B^A = \{f : A \rightarrow B\} = \text{insieme delle funzioni da } A \text{ a } B \\ B_{\perp}^A = \{f : A \rightarrow B\} = \text{insieme delle funzioni parziali da } A \text{ a } B$$

Funzione di valutazione Dati A, B e B_{\perp}^A si definisce funzione di valutazione

$$w : B_{\perp}^A \times A \rightarrow B \text{ con } w(f, a) = f(a)$$

Fissando a eseguo un benchmark di funzioni, fissando f creo i punti del grafico di f .

Sistema di calcolo C Abbiamo $P \in \text{PROG}$ che è una sequenza di regole che trasforma un dato di input in un dato di output $\Rightarrow P \in \text{DATI}_{\perp}^{\text{DATI}}$ è una funzione (in un linguaggio).

$$C : \text{DATI}_{\perp}^{\text{DATI}} \rightarrow \text{DATI}_{\perp}$$

dove $C(P, x)$ è la funzione calcolata da P

P è un oggetto semantico/rappresentazione, se faccio girare ho una funzione.

Potenza computazionale di C

$$F(C) = \{C(P, \cdot) : P \in \text{PROG}\} \subseteq \text{DATI}_{\perp}^{\text{DATI}}$$

$$F(C) = \text{DATI}_{\perp}^{\text{DATI}} \Rightarrow \text{informatica può tutto}$$

$$F(C) \subsetneq \text{DATI}_{\perp}^{\text{DATI}} \Rightarrow \text{esistono compiti non automatizzabili}$$

Cardinalità Indichiamo con $|A|$ il numero di elementi di A . Ha senso però solo su insiemi infiniti. Infatti $|\mathbb{N}| = \aleph_0 = |\mathbb{R}|$ risultano equinumerosi, che me ne faccio? In realtà, l'infinito di \mathbb{N} è meno fitto di quello di \mathbb{R} .

Relazione Relazione binaria su $A : R \subseteq A^2$. Elementi $a, b \in A$ sono nella relazione R sse $(a, b) \in R$ che si può anche indicare con aRb .

Relazione di equivalenza sse:

- Riflessiva, $\forall a : aRa$
- Simmetrica, $\forall a, b : aRb = bRa$
- Transitiva, $aRb \wedge bRc \rightarrow aRc$

Relazioni di equivalenza e partizioni $A : R \subseteq A^2$ induce partizione su $A \Rightarrow A_1, A_2, \dots \subseteq A$ t.c.

- $A_i \neq \emptyset$;
- $i \neq j \Rightarrow A_i \cap A_j = \emptyset$;
- $\cup_{i \in I} A_i = A$.

Data $a \in A$ la sua classe di equivalenza è $[a]_R = \{b \in A : aRb\}$.

Si dimostra che:

- Non esistono classi di equivalenza vuote (per riflessività ho almeno dentro me stesso);
- dati $a, b \in A \Rightarrow [a]_R \cap [b]_R = \emptyset$ o $[a]_R = [b]_R$
- $\cup_{a \in A} [a]_R = A$

L'insieme delle classi di equivalenza spezzetta A . L'insieme A visto come partizioni è detto quoziente di A rispetto a R e si indica con A/R .

Cardinalità di insiemi Sia U la classe di tutti gli insiemi. Definisco $\sim \subseteq U^2$ come $A \sim B$ sse esiste biezion tra A e B (associazione 1 a 1 tra elementi di A e B).

Proprietà di \sim :

- riflessiva (uso funzione identità di A (i_A));
- simmetrica: se $A \sim B$ allora $B \sim A$ con la funzione inversa (con biezion esiste per forza);
- transitiva: composizione di biettiva è biettiva.

Se $A \sim B$ i due insiemi sono equinumerosi. Un insieme si dice numerabile sse $A \sim \mathbb{N}$.

3 Lezione 3

Definiamo un insieme non numerabile un insieme a cardinalità infinita ma non "listabili esaustivamente" come \mathbb{N} , sono più fitti e se provo a listare mi perdo qualche elemento.

3.1 \mathbb{R} non è numerabile

Proviamo a dimostrare che non c'è biezion tra \mathbb{N} e \mathbb{R} :

1. dimostro che $\mathbb{R} \sim [0, 1]$, ovvero che $[0, 1]$ è fitto come \mathbb{R} ;
2. dimostro che $\mathbb{N} \not\sim [0, 1]$
3. $\mathbb{N} \not\sim [0, 1] \Rightarrow \mathbb{N} \not\sim \mathbb{R}$

$\mathbb{R} \sim [0, 1]$

- scelgo un punto su $[0, 1]$
- proietto sulla semicirconferenza centrata in $\frac{1}{2}$
- traccio linea tra $\frac{1}{2}$ e il punto proiettato

La funzione è iniettiva in quanto ogni punto crea un punto diverso (cambia l'angolo); è anche suriettiva tramite l'operazione inversa. Possiamo quindi dire che $\mathbb{R} \sim [0, 1]$.

$\mathbb{N} \not\sim [0, 1]$ Dimostrazione per assurdo: $\mathbb{N} \sim [0, 1]$, quindi $[0, 1]$ è listabile.

0.	<u>a_{11}</u>	a_{12}	a_{13}	a_{14}	...
0.	a_{21}	<u>a_{22}</u>	a_{23}	a_{24}	...
0.	a_{31}	a_{32}	<u>a_{33}</u>	a_{34}	...
0.	a_{41}	a_{42}	a_{43}	<u>a_{44}</u>	...
...

1 posso sciverso come $0.\bar{9}$. Costruiamo ora $0, c_1, c_2, \dots, c_i, \dots$

$$c_i = \begin{cases} a_{ii} + 1 & \text{se } a_{ii} < 9 \\ a_{ii} - 1 & \text{se } a_{ii} = 9 \end{cases}$$

c non è nessuno della lista perchè differisce per la i -esima componente, differisce dal primo perchè $c_1 \neq a_{11}$, dal secondo perchè $c_2 \neq a_{22}$ e così via. Possiamo quindi dire che $\mathbb{N} \not\sim [0, 1]$.

Quindi $\mathbb{N} \not\sim \mathbb{R}$, di conseguenza \mathbb{R} non è numerabile ed è un'insieme continuo: tutti gli insiemi equinumerosi a \mathbb{R} si dicono insiemi continui.

Insieme delle parti di \mathbb{N} $P(\mathbb{N} = \text{sottoinsiemi di } \mathbb{N}) \not\sim$ dimostrato per diagonalizzazione. Creo elenco di sottoinsiemi e trovo un sottoinsieme di \mathbb{N} che non c'è nell'elenco.

$\mathbb{N} \Rightarrow$ **1 2 3 4 5 6 ...**
 $A \Rightarrow$ **1 1 0 1 1 0 ...**
dove $1 \Rightarrow \in A$ e $0 \Rightarrow \notin A$

Per assurdo $P(\mathbb{N} \sim \mathbb{N} \Rightarrow \text{listo esaustivamente})$

<u>b_{01}</u>	b_{11}	b_{21}	b_{31}	...
\bar{b}_{02}	<u>b_{12}</u>	b_{22}	b_{32}	...
b_{03}	b_{13}	<u>b_{23}</u>	b_{33}	...
...

Considero ora il sottoinsieme di \mathbb{N} rappresentato dal vettore $\overline{b_{01}b_{12}b_{23}} \dots$ dove overline rappresenta il negato. Questo vettore è un sottoinsieme di $P(\mathbb{N})$ che non appartiene a \mathbb{N} .

$\mathbb{N}^{\mathbb{N}}$ Insieme non numerabile $\mathbb{N}^{\mathbb{N}} = \{f : \mathbb{N} \rightarrow \mathbb{N}\}$.

Anche in questo caso procedo per diagonalizzazione per ipotesi assurda. Metto sulle colonne i valori di N e sulle righe le funzioni.

<u>$f_0(0)$</u>	$f_0(1)$	$f_0(2)$	$f_0(3)$...
$f_1(0)$	<u>$f_1(1)$</u>	$f_1(2)$	$f_1(3)$...
$f_2(0)$	$f_2(1)$	<u>$f_2(2)$</u>	$f_2(3)$...
...

Definisco $\phi : \mathbb{N} \rightarrow \mathbb{N}$ con $\phi(n) = f_n(n) + 1$. $\phi \in \mathbb{N}^{\mathbb{N}}$ e dovrebbe stare nella lista esaustiva ma non c'è quindi è un insieme continuo come l'insieme delle parti di \mathbb{N} .

Table 1: Rappresentazione delle coppie di Cantor

		y			
		0	1	2	3
x	0	1	3	6	10
	1	2	5	9	
	2	4	8		
	3	7			

3.2 Cosa è calcolabile?

Considerazioni ragionevoli:

- $PROG \sim \mathbb{N}$, cosidero la digitalizzazione di un programma, è un numero espresso in binario
- $DATI \sim \mathbb{N}$, come sopra

Quindi $F(C) \sim PROG \sim \mathbb{N} \sim \mathbb{N}_\perp^{\mathbb{N}} \sim DATI_\perp^{DATI}$. Esistono funzioni non calcolabili, pochi programmi e tante funzioni.

4 Lezione 4

4.1 *Dati* $\sim \mathbb{N}$

Forniamo una legge che:

- associ biunivocamente dati a numeri e viceversa;
- consente di operare direttamente per operare sui corrispettivi dati; che ci consenta di dire, senza perdita di generalizzazione, che i nostri programmi lavorano sui numeri.

Per fare ciò, passiamo attraverso un risultato matematico sulla cardinalità di insiemi. $\mathbb{N} \times \mathbb{N} \sim \mathbb{N}^+ \Rightarrow \mathbb{N} \times \mathbb{N} \sim \mathbb{N}$, da cui si può ottenere $\mathbb{Q} \sim \mathbb{N}$ consierando che possiamo vedere le frazioni $\in \mathbb{Q}$ come coppie di numeratore e denominatore ovvero $\mathbb{N} \times \mathbb{N}$.

Funzione coppia di Cantor Definiamo $<, >: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}^+$ iniettiva e suriettiva. Abbiamo $< x, y > = n$ con $sin: \mathbb{N}^+ \rightarrow \mathbb{N}$ e $des: \mathbb{N}^+ \rightarrow \mathbb{N}$. Per il "ritorno" abbiamo quindi che $sin(n) = x$ e $des(n) = y$.

Consideriamo una rappresentazione grafica come in Tabella 4.1, riempita con i numeri $\in \mathbb{N}^+$ seguendo la diagonale. Cantor è iniettiva perchè le coordinate di punti diverse individuano celle diverse che vengono riempite successivamente; suriettiva perchè riempio fino all' n voluto e guardo immagine $< x, y >$ corrispondente. Per esempio $< 2, 1 > = 8$.

Table 2: Rappresentazione analitica di cantor, la coppia $\langle x, y \rangle$ si trova sulla diagonale della riga $x + y$

		y			
		y	...
x	x	$\langle x, y \rangle$	
		
	$x + y$...			
	...				

Forma analitica di Cantor Come vediamo nella Tabella 4.1 troviamo il valore della coppia $\langle x, y \rangle$ sulla diagonale che inizia in $\langle x + y, 0 \rangle$.

1. $\langle x, y \rangle = \langle x + y, 0 \rangle + y$
2. trovo la coppia $\langle z, 0 \rangle = \sum_{i=1}^z \frac{z(z+1)}{2} + 1$

Il punto 2 è dato dal fatto che un generico valore nella colonna 0 è dato dalla somma degli indici fino a quello cercato +1, vediamo per esempio nella Tabella 4.1 che il valore 7 nella riga 3 è calcolabile come $3 + 2 + 1 + 0$ a cui aggiungiamo ancora 1. Unendo i due punti troviamo che

$$\langle x, y \rangle = \langle x + y, 0 \rangle + y = \frac{(x + y)(x + y + 1)}{1} + y + 1.$$

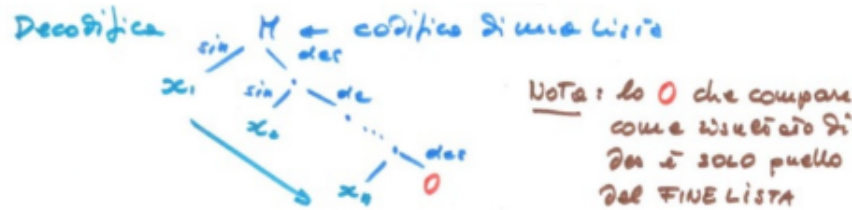
Come tornare a \mathbb{N}^+ e \mathbb{N}^+ Vogliamo capire come trovare sinistra e destra partendo da n .

1. trovare le coordinate $\langle \gamma, 0 \rangle$ del punto iniziale della diagonale dove si trova n ;
2. $y = n - \langle \gamma, 0 \rangle$ e $x = \gamma - y$,

Per il punto 1 possiamo dire che $\gamma = \max\{z \in \mathbb{N} : \langle z, 0 \rangle \leq n\}$, quindi

$$\begin{aligned} \langle z, 0 \rangle \leq n &\Rightarrow \frac{z(z+1)}{2} + 1 \leq n \\ &\Rightarrow z^2 + z + 2 - 2n \leq 0 \Rightarrow \text{eq 2° grado} \\ &\Rightarrow z_{1,2} = \frac{-1 \mp \sqrt{8n-7}}{2} \Rightarrow \text{solo } \leq 0 \\ &\Rightarrow \frac{-1 - \sqrt{8n-7}}{2} \leq z \leq \frac{-1 + \sqrt{8n-7}}{2} \\ &\Rightarrow \text{intero più grande} \Rightarrow \gamma = \lfloor \frac{-1 + \sqrt{8n-7}}{2} \rfloor; \end{aligned}$$

Figure 1: Decodifica lista



troviamo infine che $des(n) = y = n - \langle \gamma, 0 \rangle$ e $sin(n) = x = \gamma - y$.

Abbiamo quindi dimostrato $\mathbb{N} \times \mathbb{N} \sim \mathbb{N}^+$, per dimostrare $\mathbb{N} \times \mathbb{N} \sim \mathbb{N}$ basta semplicemente definire una nuova funzione, ovvero

$$[,] : \mathbb{N} \times \mathbb{N} \sim \mathbb{N} \text{ t.c. } [x, y] = \langle x, y \rangle - 1$$

e possiamo notare che , mostra che \mathbb{Q} è numerabile.

4.1.1 DATI_~ \mathbb{N}

Liste di interi Codifichiamo x_1, \dots, x_n in $\langle x_1, \dots, x_n \rangle$. Ricordiamo che le liste non hanno lunghezza nota, quindi metto uno 0 a fine lista per capire che sono arrivato alla fine.

Codifica: $1, 2, 5 \Rightarrow \langle 1, 2, 5, 0 \rangle \Rightarrow \langle 1, \langle 2, \langle 5, 0 \rangle \rangle \rangle \Rightarrow \langle 1, \langle 2, 16 \rangle \rangle \Rightarrow \langle 1, 188 \rangle \Rightarrow 18144$.

Decodifica: Creo albero a partire da n , a sinistra troverò i vari x ordinati con in cima quello di indice inferiore e a destra o un sottoalbero o uno 0. Quando trovo 0 a destra mi fermo. Un esempio è mostrato in Figura 4.1.1.

Strutture dati derivanti Array(lunghezza nota):

$$x_1, \dots, x_n \Rightarrow [x_1, \dots, x_n] = [x_1, \dots, [x_{n-1}, x_n]] \dots]$$

Matrici:

$$\begin{matrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{matrix} \Rightarrow \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \Rightarrow [[a_{11}, a_{12}], [a_{11}, a_{12}]]$$

Grafi: utilizzando le liste di adiacenza o le matrici di adiacenza.

5 Lezione 5

Sistema di calcolo *RAM*: macchina *RAM* + linguaggio *RAM* (assembly semplificato). Consente di definire rigorosamente:

- $PROG \sim \mathbb{N}$

- $C(P, _) RAM(P, _)$, semantica dei programmi
- $F(RAM)$, potenza computazionale

Forse l'idea di potenza computazionale fornita in prima istanza ($F(RAM)$) è stringente in quando la macchina RAM è molto semplice, successivamente introdurremo la macchina $WHILE$ (JVM) e confronteremo le loro potenze computazionali. Se avremo $F(_)$:

- diverse \Rightarrow ciò che è computabile dipende dallo strumento
- uguale \Rightarrow computabilità (tesi di Church)? posso calcolare stessi insiemi di funzioni?

Sistema di calcolo RAM La macchina RAM è composta da:

- L , program counter, indica indirizzo della prossima istruzione da eseguire;
- P , programma, formato da istruzioni;
- R , memoria, insieme di registri e ogni cella può contenere un numero $\in \mathbb{N}$, dove R_1 contiene l'input e R_0 l'output.

La terminazione è data da $L = 0$.

Output: $\phi_P(x) = contenuto(R_o)$ o \perp in caso di loop, indichiamo con ϕ_P la semantica di P .

Linguaggio RAM

- $R_k \leftarrow R_k + 1$
- $R_k \leftarrow R_k - 1; x \dot{-} y = x - y \text{ if } x \geq y \text{ else } 0$
- *if* $R_k = 0$ *then goto* m ; $m = \{1, |P|\}$; $|P| = \text{numero di istruzioni di } P$

Semantica Operazionale Ovvero specificare il significato di ogni istruzione, e quindi dei programmi, specificando l'effetto che quell'istruzione ha sui registri della macchina.

Come descrivo l'effetto di un'istruzione? $S = \text{Stato} = \text{foto della macchina}$. Prendo S prima e dopo l'esecuzione di un'istruzione. S_{init}, S_1, S_{fin} , P induce una sequenza di stati.

La semantica di P :

$$\phi_P = \begin{cases} y & \text{se finisce} \\ \perp & \text{altrimenti} \end{cases}$$

5.1 Ingredienti della definizione formale di semantica

Stato

$$S : \{L, R\} \rightarrow \mathbb{N}$$

$$Stati = \mathbb{N}^{\{L, R\}}$$

$S(R_k)$: contenuto del registro R_k quando la macchina è nello stato S

stato finale : $S(L) = 0 \Rightarrow \mathbf{HALT}$

dato : $\mathbb{N}(\text{infatti } DATI \sim \mathbb{N})$

Inizializzazione Dato il dato n prepara la macchina nello stato iniziale:

- $S_{init}(L) = 1$
- $S_{init}(R_1) = n$
- $\forall i \neq 1 : S_{init}(R_i) = 0$

Programmi $PROG = \{\text{programmi RAM}\}, P \in PROG; |P| = \# \text{istruzioni}$

Esecuzione Dinamica del programma \Rightarrow funzione stato prossimo.

$$\sigma : stati \times PROG \rightarrow stati_{\perp}; \sigma(S, P) = S'$$

Lo stato che segue lo stato S dopo l'esecuzione di un'istruzione di P :

- dipende dall'istruzione che devo eseguire
 - l'istruzione dipende da $S(L)$
1. se $S(L) = 0$ allora $S' = \perp$
 2. se $S(L) > |P|$ allora $S'(L) = 0$ e $\forall i : S'(R_i) = S(R_i)$
 3. se $1 \leq S(L) \leq |P|$: considero l'istruzione $S(L) -esima$:
 - $R_k \leftarrow R_k + /-1$:
 - $S'(R_k) = S(R_k) + /-1$
 - $S'(L) = S(L) + 1$
 - $\forall i \neq k : S'(R_i) = S(R_i)$
 - *if* $R_k = 0$ *then goto* m :
 - $S'(R_i) = S(R_i)$
 - $S'(L) = m$ *if* $S(R_k) == 0$ *else* $S(L) + 1$

Semantica di P

$$\phi_P : \mathbb{N} \rightarrow \mathbb{N}_{\perp}$$

$$\phi_P(n) = \begin{cases} y & \text{se } S_m(L) = 0 \text{ e } S_m(R_0) = y \\ \perp & \text{se va in loop} \end{cases}$$

Potenza computazionale di RAM $F(RAM) = \{f \in \mathbb{N}_{\perp}^{\mathbb{N}} : \exists P \in PROG, \phi_P = f\} = \{\phi_P : P \in PROG\} \subseteq \mathbb{N}_{\perp}^{\mathbb{N}}$.

Incluso stretto per intuizione.

6 Lezione 6

6.1 $PROG \sim \mathbb{N}$ su programmi RAM

Come codificare programmi in numeri e ritorno biunivocamente Appliciamo a ogni istruzione del programma un'aritmetizzazione e poi uniamo i vari numeri generati per creare un singolo numero tramite la codifica utilizzata con la lista di numeri + Cantor. Per il ritorno, sappiamo decodificare la lista finale; se l'aritmetizzazione (Ar) è invertibile allora da n posso ricostruire univocamente il sorgente.

Ci MANca solo il passaggio fatto da Ar : da istruzioni a numeri e viceversa. Questo passaggio si dice aritmetizzare o godelizzare.

6.2 Come aritmetizzare?

$Ar : \text{istruzione} \rightarrow \mathbb{N} \text{ t.c. } Ar(istr = n) \leftrightarrow Ar^{-1}(n) = istr$

$Ar(R_k \leftarrow R_k + 1) = 3k$

$Ar(R_k \leftarrow R_k \dot{-} 1) = 3k + 1$

$Ar(\text{if } R_k = 0 \text{ then goto } m = 3 < k, m > -1 \text{ come fare } +2$

Com'è fatto Ar^{-1} Utilizzo il resto della divisione per 3, quindi $\|n\|_3$ (n modulo 3):

- 0: $n = 3k \Rightarrow R_{\frac{n}{3}} \leftarrow R_{\frac{n}{3}} + 1;$
- 1: $n = 3k + 1 \Rightarrow R_{\frac{n-1}{3}} \leftarrow R_{\frac{n-1}{3}} \dot{-} 1;$
- 2: $n = < k, m > -1 \Rightarrow < k, m > = \frac{n+1}{3} \Rightarrow \text{if } R_{\sin \frac{n+1}{3}} = 0 \text{ then goto } R_{\text{des} \frac{n+1}{3}}.$

Da programmi a numeri $\text{cod}(P) = < Ar(istr_1), \dots, Ar(istr_m) >$

Da numeri a programmi Come la decodifica destro/sinistro con le parti sinistra che "subiscono" Ar^{-1} , anche in questo caso ci fermiamo quando troviamo lo 0 nel lato destro.

Osservazioni

- i numeri diventano linguaggio di programmazione;
- potrei scrivere $F(RAM) = \{\phi_P : P \in PROG\}$ come $F(RAM) = \{\phi_i\}_{i \in \mathbb{N}}$;

- per il sistema RAM si ha rigorosamente $F(RAM) \sim \mathbb{N} \approx \mathbb{N}_+^{\mathbb{N}}$, quindi alcuni problemi non sono automatizzabili;
- forse, considerando un sistema di calcolo C più sofisticato ma comunque rigorosamente trattabile come RAM , potremmo dare un'idea formale di "ciò che è calcolabile automaticamente" come $F(C)$ che sia più ampia di $F(RAM)$;
- se dimostriamo che $F(C) = F(RAM)$ allora cambiare tecnologia non cambia ciò che è calcolabile \Rightarrow la calcolabilità è intrinseca ai problemi: perchè non catturarla matematicamente? (no macchine, no linguaggio, ...).

6.3 Sistema di calcolo **WHILE**

Memoria x_0, \dots, x_20 con x_0 output e x_1 input. Le variabili contengono numeri arbitrariamente grandi e di conseguenza in una singola variabile posso salvare, per esempio con Cator, più di un semplice numero.

Non abbiamo un program counter in quanto il linguaggio **WHILE** è strutturato.

Linguaggio WHILE Sintassi induttiva: ho delle fasi semplici, con passi induttivi faccio cose più complicate.

- $[BASE]$: comando assegnamento: $x_k = 0$, $x_k = x_j + 1$, $x_k = x_j - 1$;
- $[PASSI]$: comando while: while $x_k \neq 0$ do C , con C che può essere:
 - comando di assegnamento
 - comando while
 - comando composto
- $[PASSI]$: comando composto: BEGIN C_1, \dots, C_m END, con C come sopra.

Possiamo quindi dire che un programma **WHILE** è un comando composto.

$w - PROG = \{\text{programmi WHILE}\} \leftarrow$ costruiti induttivamente.

Semantica di w : $\psi_w : \mathbb{N} \rightarrow \mathbb{N}$

$w - \mathbf{PROG}$ Per dimostrare una proposizione su $w - PROG$:

1. dimostro proposizione sugli assegnamenti;
2. suppongo proposizione su C e la dimostro su $while\ x_k \neq 0\ do\ C$;
3. suppongo vera la proposizione su C_1, \dots, C_m e la dimostro su BEGIN C_1, \dots, C_m END.

6.3.1 Dimostrazioni induttive su alberi bianri



Dividiamo i nodi in nodi interni e foglie.

1. $[BASE]$: \bullet è un albero binario



2. $[PASSO]$: se T_1 e T_2 sono alberi binari allora $T_1 \quad T_2$ è un albero binario
3. nient'altro è un albero binario

su ogni albero binario, il numero di nodi interni è minore di 1 rispetto alle foglie = P Per induzione:

1. $[BASE]$: \bullet , 1 foglia e 0 nodi interni \Rightarrow **VERO**
2. $[PASSO]$: suppongo vero P su T_1 e T_2 , ovvero suppongo vero:
 - T_1 : foglie F_1 , nodi interni $F_1 - 1$
 - T_2 : foglie F_2 , nodi interni $F_2 - 1$



Dimostro vero P per: $T_1 \quad T_2$ ovvero $F_1 + F_2$ foglie, e $F_1 - 1 + F_2 - 1 + 1 = F_1 + F_2 - 1$ nodi interni \Rightarrow **VERO**

Depth

1. $[BASE]$: $depth(\bullet) = 0$



2. $[BASE]$: $depth(T_1 \quad T_2) = 1 + \max(depth(T_1) + depth(T_2))$

7 Lezione 7

7.1 Esecuzione su una macchina WHILE (intuitivamente)

1. inizializzazione: imposto le variabili x_0, \dots, x_{20} come $0, n, 0, \dots, 0$;
2. esecuzione: eseguo le istruzioni del programma w (non ho bisogno del program counter);
3. terminazione: quando le istruzioni di w terminano oppure ho un loop;
4. output: contenuto di x_0 , quindi $\Psi(n) = cont(x_0) / \perp$.

Semantica del programma $w \quad \Psi_w : \mathbb{N} \rightarrow \mathbb{R}_\perp$.

7.2 Definizione formale di semantica di un programma WHILE

- Stato: foto della macchina in un tempo t ovvero una tupla/vettore di 21 componenti: (c_0, \dots, c_{20}) con c_i contenuto della cella x_i .

$$W - Stati = \mathbb{N}^{21}, \text{ stato } \underline{x} \in \mathbb{N}^{21} \text{ (vettore } \underline{x} \text{)}$$

- Dati: \mathbb{N}
- Inizializzazione: $w-in : \mathbb{N} \rightarrow \mathbb{N}^{21}$ con $w-in(n) = (0, n, 0, \dots, 0)$
- Semantica operazione:

$$[]() : w-comandi \times w-stati \rightarrow w-stati_\perp; [C](\underline{x}) = \underline{y}$$

con C comando, \underline{y} è lo stato prossimo partendo da \underline{x} eseguendo C

Posso definire intuitivamente $[C](\underline{x})$ sulla struttura induttiva del comando C :

- $[BASE]$ Assegnamenti:

$$[x_k = 0](\underline{x}) = \underline{y} \text{ con } y_i = \begin{cases} x_i & \text{se } i \neq k \\ 0 & \text{se } i = k \end{cases}$$

$$[x_k = x_j + / \dot{-} 1](\underline{x}) = \underline{y} \text{ con } y_i = \begin{cases} x_i & \text{se } i \neq k \\ x_j + / \dot{-} 1 & \text{se } i = k \end{cases}$$

- $[PASSO]$ Comando composto: $[BEGIN \ C_1, \dots, C_m \ END]$, conosco $[C_i]$ per ipotesi induttiva, ovvero conosco cosa fa ogni singolo comando.

$$[C_m](\dots([C_2]([C_1](\underline{x})))\dots) = \underline{y} = [C_1] \cdot \dots \cdot [C_m](\underline{x})$$

- *[PASSO]*: Comando while: $[WHILE\ x_k \neq 0\ DO\ C]$, conosco $[C_i]$ come sopra.

$$[C](\dots((([C]([C](\underline{x})))\dots)) = \underline{y}$$

Quante volte applico C ? Tante volte quanto serve per azzerare x_k dello stato risultante durante l'iterazione del comando C ovvero

$$\begin{cases} [C]^e(\underline{x}) \text{ con } e = \mu t(k\text{-esima componente di } [C]^{(t)}(\underline{x}) = 0) \\ \perp \text{ altrimenti} \end{cases}$$

dove μt è il minor numero di volte.

Semantica di w e $w\text{-prog}$ $\Psi_w(n) = \mathbf{Pr}(0, [w](w\text{-in}(n)))$; proiezione 0-esima ci restituisce l'output applicando $[w]$ allo stato prodotto dall'inizializzazione con $x_1 = n$.

7.3 Potenza computazionale sistema WHILE

$$F(WHILE) = \{f \in \mathbb{N}_\perp^\mathbb{N} : \exists w \in w\text{-PROG}, f = \Psi_w\} = \{\Psi_w : w \in w\text{-PROG}\}$$

7.3.1 Relazione tra $F(RAM)$ e $F(WHILE)$?

Che relazione esiste tra $F(WHILE)$ e $F(RAM) = \{\phi_P : P \in PROG\}$?

- $F(RAM) \subsetneq F(WHILE)$, sarebbe anche comprensibile vista la semplicità della macchina RAM ;
- Insiemi con intersezioni o disgiunti, sarebbe preoccupante perchè il concetto di calcolabile dipenderebbe dalla macchina;
- $F(WHILE) \subsetneq F(RAM)$, sarebbe sorprendente poichè $WHILE$ sembra più sofisticata di RAM ;
- $F(RAM) = F(WHILE) \Rightarrow$ calcolabile non dipende dalla tecnologia

7.3.2 Confronto tra due sistemi di calcolo

Poniamo di avere due sistemi di calcolo C_1 e C_2 con i relativi programmi $C_1\text{-PROG}$ e $C_2\text{-PROG}$.

$$F(C_1) = \{f \in \mathbb{N}_\perp^\mathbb{N} : f = \Psi_{P_1} \text{ per qualche } P_1 \text{ e } C_1\text{-PROG}\} = \{\Psi_{P_1} : P_1 \in C_1\text{-PROG}\}$$

$$F(C_2) = \{f \in \mathbb{N}_\perp^\mathbb{N} : f = \phi_{P_2} \text{ per qualche } P_2 \text{ e } C_2\text{-PROG}\} = \{\phi_{P_2} : P_2 \in C_2\text{-PROG}\}$$

Come mostriamo che $F(C_1) \subseteq F(C_2)$ ovvero che il primo non supera il secondo?

Dimostro che

$$\forall f \in F(C_1) \Rightarrow f \in F(C_2)$$

ovvero che per ogni elemento del primo insieme allora è anche nel secondo insieme (dimostrazione di inclusione).

Risolviamo questo problema con un traduttore, prende un programma in un linguaggio e lo traduce in un altro linguaggio; per esempio quando compilo un programma in $C++$ creo un assembly, questo implica che $C++$ è al più potente come assembly. Matematicamente possiamo descrivere un traduttore come

$$\exists P_1 \in C_1-PROG : f = \Psi_{P_1} \Rightarrow \exists P_2 \in C_2-PROG : f = \Phi_{P_2},$$

per ogni programma nel primo sistema ne esiste uno equivalente a un programma del secondo sistema.

7.4 Concetto di traduttore

Dati i sistemi C_1 e C_2 , una traduzione da C_1 a C_2 è una funzione $T : C_1-PROG \rightarrow C_2-PROG$ con le seguenti proprietà:

- [Programmabile]: è programabile;
- [Completa]: traduce ogni C_1-PROG in C_2-PROG ;
- [Corretta]: mantiene la semantica $\forall P \in C_1-PROG : \phi_{T(P)} = \Psi_P$, è la formalizzazione del concetto di compilatore (nota che $\phi_{T(P)}$ è l'oggetto e Psi_P il sorgente).

Se esiste $T : C_1-PROG \rightarrow C_2-PROG$ allora $F(C_1) \subseteq F(C_2)$. Dimostrazione:

$$f \in F(C_1) \Rightarrow \exists P \in C_1-PROG : f = \Psi_P$$

a P applico T , ottengo

$$T(P) \in C_2-PROG \text{ (completezza), con } \phi_{T(P)} = \Psi_P = f \text{ (correttezza)}$$

esiste dunque un $PROG$ in C_2-PROG per f , per cui $f \in F(C_2)$.

8 Lezione 8