

# INFORMATICA TEORICA

giosumarin

March 2021

## Contents

<b>1</b>	<b>Lezione 1</b>	<b>2</b>
1.1	Definizione di funzione . . . . .	2
1.2	Funzione iniettiva, suriettiva e biettiva . . . . .	2
<b>2</b>	<b>Lezione 2</b>	<b>3</b>
<b>3</b>	<b>Lezione 3</b>	<b>5</b>
3.1	$\mathbb{R}$ non è numerabile . . . . .	5
3.2	Cosa è calcolabile? . . . . .	7
<b>4</b>	<b>Lezione 4</b>	<b>7</b>
4.1	$Dati \sim \mathbb{N}$ . . . . .	7
4.1.1	$DATI \simeq \mathbb{N}$ . . . . .	9
<b>5</b>	<b>Lezione 5</b>	<b>10</b>
5.1	Ingredienti della definizione formale di semantica . . . . .	11
<b>6</b>	<b>Lezione 6</b>	<b>12</b>
6.1	$PROG \sim \mathbb{N}$ su programmi <i>RAM</i> . . . . .	12
6.2	Come aritmetizzare? . . . . .	12
6.3	Sistema di calcolo <i>WHILE</i> . . . . .	13
6.3.1	Dimostrazioni induttive su alberi bianri . . . . .	14
<b>7</b>	<b>Lezione 7</b>	<b>15</b>
7.1	Esecuzione su una macchina <i>WHILE</i> (intuitivamente) . . . . .	15
7.2	Definizione formale di semantica di un programma <i>WHILE</i> . . . . .	15
7.3	Potenza computazionale sistema <i>WHILE</i> . . . . .	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
<b>8</b>	<b>Lezione 8</b>	<b>17</b>
8.1	Costruzione induttiva di <i>comp</i> . . . . .	18
8.2	$F(RAM) \subseteq F(WHILE)$ . . . . .	19

<b>9</b>	<b>Lezione 9</b>	<b>19</b>
9.1	Interprete WHILE di programmi RAM . . . . .	19
9.1.1	Variabili . . . . .	19
9.1.2	Codice interprete in macrowhile . . . . .	20
9.1.3	Conseguenze . . . . .	20
9.1.4	Riflessioni concetto di calcolabilità . . . . .	21
<b>10</b>	<b>Lezione 10</b>	<b>21</b>
10.1	Chiusura di insiemi rispetto ad operazioni . . . . .	21
10.1.1	Chiusura . . . . .	22
10.1.2	Chiusura di un'insieme . . . . .	22
10.1.3	Chiusura di un insieme rispetto a un insieme di operazioni	23
10.1.4	Verso una definizione teorica di calcolabilità . . . . .	23

## 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  $\mathbb{N} = \{0, 1, 2, 3, 4, ..\}$  e con  $\mathbb{N}^+ = \{1, 2, 3, 4, ..\}$

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 comprende tutto l'asse  $y$ .

**Funzione Biiettiva** Una funzione si dice biiettiva 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**

$$f : A \rightarrow B \text{ biiettiva}$$

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

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

$f(a) \downarrow$ :  $f$  definita  $\forall a \in A$  si dice che  $f$  è totale

$f(a) \uparrow$ : non definita per ogni  $a \in A$ .

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

$$A \times b = \{(a, b) : a \in A \wedge b \in B\}$$

**Nota:**  $\times$  non commutativa  $A \times B \neq B \times A$

**Proiettore -iesimo**  $\pi_i : A_1 \times \dots \times A_n \rightarrow A_i$

$$\pi_i(a_1, \dots, a_n) = a_i$$

**Indichiamo A per n volte come**  $A \times \dots \times A = A^n$

### Insieme di funzioni

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

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

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

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

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  $\Rightarrow P \in \text{DATI}^{DATI}_{\perp}$  è una funzione (in un linguaggio).

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

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

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

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

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

**Cardinalità** Indichiamo con  $|A|$  il numero di elementi di  $A$ . Ha senso però solo su insiemi infiniti. Infatti  $|\mathbb{N}| = \text{inf} = |\mathbb{R}|$  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 e 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 e 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 e 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><math>a_{11}</math></u>	$a_{12}$	$a_{13}$	$a_{14}$	...
0.	$a_{21}$	<u><math>a_{22}</math></u>	$a_{23}$	$a_{24}$	...
0.	$a_{31}$	$a_{32}$	<u><math>a_{33}</math></u>	$a_{34}$	...
0.	$a_{41}$	$a_{42}$	$a_{43}$	<u><math>a_{44}</math></u>	...
...	...	...	...	...	...

1 posso sciverso come  $0.\overline{9}$ . 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  $\mathbb{N}$**   $P(\mathbb{N})$  = 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  $P(\mathbb{N} \sim \mathbb{N} \Rightarrow$  listo esaustivamente

<u><math>b_{01}</math></u>	$b_{11}$	$b_{21}$	$b_{31}$	...
$b_{02}$	<u><math>b_{12}</math></u>	$b_{22}$	$b_{32}$	...
$b_{03}$	$b_{13}$	<u><math>b_{23}</math></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.

$$\begin{array}{ccccc} f_0(0) & f_0(1) & f_0(2) & f_0(3) & \dots \\ f_1(0) & f_1(1) & f_1(2) & f_1(3) & \dots \\ f_2(0) & f_2(1) & f_2(2) & f_2(3) & \dots \\ \dots & \dots & \dots & \dots & \dots \end{array}$$

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  $\mathbb{N}$ .

### 3.2 Cosa è calcolabile?

Considerazioni ragionevoli:

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

Quindi  $F(C) \sim PROG \sim \mathbb{N} \not\sim \mathbb{N}_{\perp}^{\mathbb{N}} \sim DATI_{\perp}^{DATI}$ . 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  $\mathbb{Q} \sim \mathbb{N}$  considerando 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  $\langle, \rangle : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}^+$  iniettiva e suriettiva. Abbiamo  $\langle x, y \rangle = n$  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;

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			

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

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

suriettiva perchè riempio fino all'  $n$  voluto e guardo immagine  $\langle x, y \rangle$  corrispondente. Per esempio  $\langle 2, 1 \rangle = 8$ .

**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

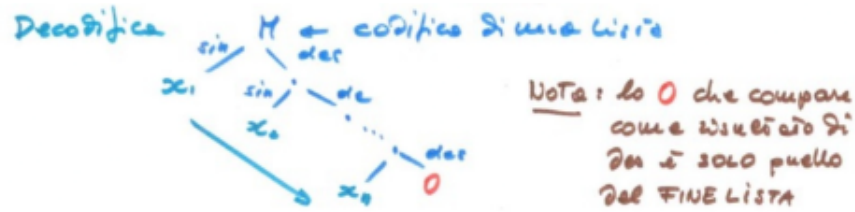
$$\langle x, y \rangle = \langle x + y, 0 \rangle + y = \frac{(x + y)(x + y + 1)}{1} + y + 1.$$

**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  $x = \gamma - y$ ,



Figure 1: Decodifica lista



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

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

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<sub>~</sub> $\mathbb{N}$

**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:

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

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 quanto 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  $\phi_P$  la semantica di  $P$ .

**Linguaggio *RAM***

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

**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?  $S$ =Stato=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 :  $\mathbb{N}$  (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?

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

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

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

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

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

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

**Da programmi a numeri**  $cod(P) = \langle Ar(istr_1), \dots, Ar(istr_m) \rangle$

**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;
- potresti scrivere  $F(RAM) = \{\phi_P : P \in PROG\}$  come  $F(RAM) = \{\phi_i\}_{i \in \mathbb{N}}$ ;
- per il sistema  $RAM$  si ha rigorosamente  $F(RAM) \sim \mathbb{N}^{\mathbb{N}}$ , 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: perché non catturarla matematicamente? (no macchine, no linguaggio, ...).

## 6.3 Sistema di calcolo *WHILE*

Memoria  $x_0, \dots, x_2$  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.

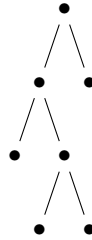
Possiamo quindi dire che un programma *WHILE* è un comando composto.  
 $w - PROG = \{\text{programmi } WHILE\} \leftarrow$  costruiti induttivamente.

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

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

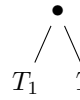
1. dimostro proposizione sugli assegnamenti;
2. suppongo proposizione su  $C$  e la dimostro su  $\textit{while} x_k \neq 0 \textit{do} C$ ;
3. suppongo vera la proposizione su  $C_1, \dots, C_m$  e la dimostro su BEGIN  
 $C_1, \dots, C_m$  END.

### 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  $x_i$ .

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

- 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:

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

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

- *[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  $\mu t$  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

Mostreremo  $F(WHILE) \subseteq F(RAM)$ , mostreremo una traduzione  $comp : W-PROG \rightarrow PROG$ . Per comodità utilizzeremo il linguaggio RAM etichettato: etichetto una riga (istruzione) per fare un salto, ovviamente ho la stessa potenza computazionale di RAM.

In quanto  $W-PROG$  è definito induttivamente  $comp$  può essere definito induttivamente:

1. [BASE] mostro come compilare gli assegnamenti;
2. [PASSO] per ipotesi induttiva assumo dato  $comp(C_1), \dots, comp(C_m)$  e mostro come compilare il comando composto BEGIN  $C_1, \dots, C_m$  END;

3. [PASSO] per ipotesi induttiva assumo dato  $comp(C)$  e mostro come compilare il comando while WHILE  $X_k \neq 0$  DO  $C$ .

### 8.1 Costruzione induttiva di $comp$

In generale tradurremo  $X_k$  con  $R_k$  ovvero variabili di WHILE in registri RAM ( $21 \Rightarrow \infty$ ).

[BASE] **assegnamenti**

- $comp(X_k := 0)$

$$\begin{aligned} \underline{loop} : & \text{ if } R_k == 0 \text{ then goto } \underline{exit} \\ & R_k \leftarrow R_k - 1 \\ & \text{ if } R_{21} == 0 \text{ then goto } \underline{loop} \\ \underline{exit} : & R_k \leftarrow R_k - 1 \end{aligned}$$

Uso  $R_{21}$  perchè sarà sempre uguale a 0, WHILE ha 21 variabili (da 0 a 20).

- $comp(X_k := X_j + / - 1)$ 
  - se  $k == j$ :  $R_k \leftarrow R_k + / - 1$
  - altrimenti:
    1. salvo  $X_j$  in  $R_{22}$ ;
    2. azzero  $R_k$ ;
    3. metto in  $R_j$  e  $R_k$  il contenuto di  $R_{22}$  (un ciclo che fa + 1 al  $R_{22}$  e somma 1 a  $R_k$  e  $R_k$ ;
    4. sommo/sottraggo 1 a  $R_k$ .

[PASSO]

- comando composto: basta prendere la sequenza di comando e creare la sequenza di compilazione dei comandi
- comando while:  $comp(\underline{while} \ x_k \neq 0 \ \underline{do} \ C)$ :

$$\begin{aligned} \underline{loop} : & \text{ if } R_k == 0 \text{ then goto } \underline{exit} \\ & comp(C) \\ & \text{ if } R_{21} == 0 \text{ then goto } \underline{loop} \\ \underline{exit} : & R_k \leftarrow R_k - 1 \end{aligned}$$

## Considerazioni

1. il compilatore è facilmente programmabile;
2. compila ogni sorgente while  $\Rightarrow$  completo;
3. mantiene la semantica:  $\Psi_w = \Phi_{comp(w)} \Rightarrow$  corretto;

quindi  $F(WHILE) \subseteq F(RAM)$ .

### 8.2 $F(RAM) \subseteq F(WHILE)$

Faremo un interprete (compila riga per riga e ha come output l'output del programma),  $I_w$ : interprete scritto in WHILE di programmi scritti in RAM.

- input di  $I_w$ :  $P \in PROG$  e  $x \in \mathbb{N}$ , output:  $\Phi_P(x)$
- input di  $I_w$ :  $codifica(P) = n$  e  $x \in \mathbb{N}$ , output:  $\Phi_n(x) = \Phi_P(x)$
- input di  $I_w$ :  $\langle x, n \rangle$  con  $x$  il dato di input e  $n$  la codifica del programma, output:  $\Phi_n(x) = \Phi_P(x)$

La semantica di  $I_w$ :  $\forall x, n \in \mathbb{N} : \Psi_{I_w}(\langle x, n \rangle) = \Phi_n(x) = \Phi_P(x)$ .

**Macro while** Per comodità, nella scrittura di  $I_w$  utilizzeremo un WHILE che ingloba alcune macro che possono essere tradotte in WHILE puro.

## 9 Lezione 9

### 9.1 Interprete WHILE di programmi RAM

#### 9.1.1 Variabili

$I_w$  ricrea nelle sue variabili l'ambiente (macchina RAM) in cui esegue  $P$ . Se  $cod(P) = n$  allora  $P$  non userà mai  $R_j$  con  $j > n$ . Posso quindi restringermi a modellare la sequenza  $R_0, \dots, R_{n+2}$ . Il contenuto della memoria in cui si esegue  $P$  può essere considerato in una sola variabile contenente  $\langle a_0, a_1, \dots, a_{n+2} \rangle$ . Avremo quindi la seguente configurazione:

- $X_0 \Leftarrow \langle R_0, \dots, R_{n+2} \rangle$ ;
- $X_1 \Leftarrow L$  con  $L$  program counter;
- $X_2 \Leftarrow x$  ovvero il dato di input;
- $X_3 \Leftarrow n$  ovvero il "listato" del programma  $P$ ;
- $X_4 \Leftarrow$  codice dell'istruzione da eseguire prelevata da  $X_3$  nella posizione  $X_1$ .

### Inizializzazione

1.  $X_1 \leftarrow \text{input}(< x, n >)$
2.  $X_2 := \text{sin}(X_1)$  ovvero il dato di input;
3.  $X_3 := \text{sin}(X_1)$  ovvero il "listato" del programma  $P$ ;
4.  $X_1 := 1$ .

### 9.1.2 Codice interprete in macrowhile

---

```

1: while ( $X_1 \neq 0$ ) do                                ▷  $\text{HALT con } L = 0$ 
2:   if ( $X_1 > \text{length}(X_2)$ ) then  $X_1 := 0$            ▷ Ho finito le istruzioni
3:   else
4:      $X_4 := \text{Pro}(X_1, X_3)$         ▷ Prendo elemento in pos  $X_1$  di  $X_3$  (fetch)
5:     if ( $X_4 \bmod 3 == 0$ ) then                        ▷  $R_k \leftarrow R_k + 1$ 
6:        $X_5 := X_4 / 3$ 
7:        $X_0 := \text{incr}(X_5, X_0)$         ▷ Increment di 1 elem in pos  $X_5$  di  $X_0$ 
8:        $X_1 := X_1 + 1$                     ▷ Incremento program counter
9:     end if
10:    if ( $X_4 \bmod 3 == 1$ ) then                        ▷  $R_k \leftarrow R_k - 1$ 
11:       $X_5 := (X_4 - 1) / 3$ 
12:       $X_0 := \text{decr}(X_5, X_0)$ 
13:       $X_1 := X_1 + 1$ 
14:    end if
15:    if ( $X_4 \bmod 3 == 2$ ) then                        ▷ if  $R_k == 0$  then goto  $m$ 
16:       $X_5 := \text{sin}((X_4 + 1) / 3)$                                 ▷  $k$ 
17:       $X_6 := \text{des}((X_4 + 1) / 3)$                                 ▷  $m$ 
18:      if ( $\text{Pro}(X_5, X_0) == 0$ ) then
19:         $X_1 = X_6$ 
20:      else
21:         $X_1 := X_1 + 1$ 
22:      end if
23:    end if
24:  end if
25: end while
26:  $X_0 := \text{sin}(X_0)$ 

```

---

### 9.1.3 Conseguenze

Posso costruire  $\text{Comp} : \text{PROG} \rightarrow W - \text{PROG}$ :

- $n = \text{cod}(P)$
- $X_1 := < x, n >$

- $I_w$

Programmabile, completo e corretto.

$$\Psi_{comp(P)}(x) = \Psi_{I_w}(< x, n >) = \Phi_n(x) = \Phi_P(x) \text{ OK!}$$

**Teorema di Bohm-Jacopini** Per ogni programma con goto (RAM) ne esiste uno equivalente in linguaggio strutturato (WHILE).

**Finalmente..**  $\mathbb{N}_{\perp}^{\mathbb{N}} \sim \mathbb{N} \sim PROG \sim F(RAM) = F(WHILE)$

- nei sistemi di programmazione RAM e WHILE esistono funzioni non computabili, dimostrato formalmente;
- i sistemi RAM e WHILE, pur profondamente diversi, calcolano le stesse cose.

Usiamo  $comp : W - PROG \rightarrow PROG$  sse  $I_w(\Psi_{I_w}(< x, n >) = \Phi_n(x))$ .

$$U = comp(I_w) \in PROG$$

$$\Phi_U(< x, n >) = \Psi_{I_w}(< x, n >) = \Phi_n(x)$$

Nel sistema di programmazione RAM esiste un programma RAM capace di simulare qualunque altro programma RAM!  $U$  è detto interprete universale (RAM).

#### 9.1.4 Riflessioni concetto di calcolabilità

Due sistemi profondamente diversi che determinano la stessa idea di calcolabilità. Possiamo sublimare il concetto di calcolabile?

Possiamo pensare di definire ciò che è calcolabile a prescindere dalle macchine che usiamo per calcolare?

Possiamo pensare di definire ciò che è calcolabile in termini più astratti, matematici, "lontani dall'informatica"?

## 10 Lezione 10

### 10.1 Chiusura di insiemi rispetto ad operazioni

Dato un insieme  $U$ , si definisce un'operazione su  $U$  una qualunque funzione  $op : U \times \dots \times U \rightarrow U$  con  $U$  ripetuta  $k$  volte in input:  $k$ -arietà dell'operazione.

**Esempi**  $U = \mathbb{N}$

- $+$  :  $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ , somma:  $+(5, 3) = 8$ , operazione binaria;
- $\lfloor \sqrt{\cdot} \rfloor$  :  $\mathbb{N} \rightarrow \mathbb{N}$ , radice troncata, operazione unaria;
- $Pro_t^n$  :  $\mathbb{N} \times \dots \times \mathbb{N} \rightarrow \mathbb{N}$  con  $n$  volte in input, proiezione  $t$ -esima, operazione  $n$ -aria.

### 10.1.1 Chiusura

L'insieme  $A \subseteq U$  è chiuso rispetto all'operazione  $op : U^k \rightarrow U$  sse  $\forall a_1, \dots, a_k \in A : op(a_1, \dots, a_k) \in A$ , ovvero l'operazione non mi "amplia"  $A$ .

#### Esempi

- $+: \mathbb{N}^2 \rightarrow \mathbb{N}$ ,  $PARI \subseteq \mathbb{N}$ .  $PARI$  è chiuso per  $+$ ? sì,  $2k + 2j = 2(k + j)$
- $+: \mathbb{N}^2 \rightarrow \mathbb{N}$ ,  $DISPARI \subseteq \mathbb{N}$ .  $DISPARI$  è chiuso per  $+$ ? no,  $23 + 3 = 6$
- $/: \mathbb{Q}^2 \rightarrow \mathbb{Q}$ ,  $\mathbb{N} \subseteq \mathbb{Q}$ .  $\mathbb{N}$  è chiuso per  $/$ ? no,  $5/2 \notin \mathbb{N}$

Per rispondere si dobbiamo dimostrare che vale, nel nostro caso, per ogni coppia; per rispondere no basta un controesempio.

In generale: se  $\Omega$  è un'insieme di operazioni su  $U$ , allora  $A \subseteq U$  è chiuso rispetto a  $\Omega$  se è chiuso per ogni operazione in  $\Omega$ .

#### Esempi

- $\Omega = \{+, -\}$ ,  $PARI \subseteq \mathbb{N} \rightarrow$  somma sì, prodotto sì ( $2k \cdot 2j = 4kj$ );
- $\Omega = \{+, -\}$ ,  $DISPARI \subseteq \mathbb{N} \rightarrow$  somma no, prodotto sì ( $(2k+1)(2j+1) = 4kj + 2k + 2j + 1$  è dispari).

### 10.1.2 Chiusura di un'insieme

**Problema:** Sia  $A \subseteq U$  e  $op : U^k \rightarrow U$ . Qual è il più piccolo sottoinsieme di  $U$  che contiene  $A$  e sia chiuso per  $op$ ? In pratica voglio allargare  $A$  per renderlo chiuso.

#### Risposte ovvie:

- se  $A$  è chiuso per  $op$  allora  $A$  stesso;
- sicuramente  $U$  soddisfa le due richieste ma non è sempre il più piccolo.

**Esempio** Sia  $A = \{2, 3\} \subseteq \mathbb{N}$  e  $+: \mathbb{N}^2 \rightarrow \mathbb{N}$ , qual è il più piccolo insieme che soddisfa le proprietà sopra? Sicuramente  $\mathbb{N}$  ma non è il più piccolo: sicuramente non ci servono 0 e 1. Non basta nemmeno aggiungere  $\{4, 5, 6\}$  perchè devo soddisfare la chiusura anche sui valori che aggiungo all'insieme che sto cercando.

**Teorema:** Sia  $A \subseteq U$  e  $op : U^k \rightarrow U$ . Il più piccolo sottoinsieme di  $U$  contenente  $A$  è chiuso rispetto a  $op$  si ottiene calcolando la chiusura rispetto a  $op$ , cioè l'insieme  $A^{op}$  definito induttivamente come:

1.  $\forall a \in A \Rightarrow a \in A^{op}$ ;
2.  $\forall a_1, \dots, a_k \in A^{op} \Rightarrow op(a_1, \dots, a_k) \in A^{op}$ ;

3. nient'altro sta in  $A^{op}$ .

Possiamo dare la seguente definizione più operativa:

1. metti in  $A^{op}$  tutti gli element di  $A$ ;
2. applica  $op$  a una  $k$ -tupla di elementi in  $A^{op}$ ;
3. aggiungo il riultato in  $A^{op}$  se non è già presente;
4. reitero i punti 2 e 3 finche  $A^{op}$  cresce;
5. output  $A^{op}$ .

**Esempio**  $A = \{2, 3\}$ ,  $A \subseteq \mathbb{N}$  e voglio trovare  $A^+$ :

1.  $A^+ \leftarrow A$ ;
2.  $2 + 3 = 5 \notin A^+ \Rightarrow A^+ = \{2, 3, 5\}$ ;
3.  $2 + 2, 3 + 3, 5 + 5, \dots \notin A^+ \Rightarrow A^+ = \{2, 3, 4, 5, 6, 10, \dots\}$ ;
4. ....
5. output:  $A^+ = \mathbb{N} \setminus \{0, 1\}$ .

### 10.1.3 Chiusura di un insieme rispetto a un insieme di operazioni

$A^\Omega$  si ottiene generalizzando il processo per un'operazione induttivamente:

1.  $\forall a \in A \Rightarrow a \in A^\Omega$ ;
2.  $\forall i \in \{1, \dots, t\} \forall a_1, \dots, a_k \in A^\Omega \Rightarrow op_i(a_1, \dots, a_k) \in A^\Omega$ ;
3. nient'altro sta in  $A^\Omega$ .

### 10.1.4 Verso una definizione teorica di calcolabilità

**Astratta** : che astraeda qualunque connotato informatico;

**Roadmap** :

1.  $ELEM$ : insieme di funzioni che qualunque idea di calcolabile si voglia proporre deve considerare calcolabili;  $ELE$  non può esaurire il concetto di calcolabilità  $\Rightarrow$  ampliare;
2.  $\Omega$ : insieme di operazioni su funzioni che costruiscono funzioni. Le  $op$  in  $\Omega$  sono banalmente implementabili  $\Rightarrow$  se le applico a funzioni calcolabili ottengo nuove funzioni calcolabili;
3.  $ELEM^\Omega = P$  = classe delle funzioni ricorsive parziali.  $P \rightarrow$  la nostra idea astratta della classe delle funzioni calcolabili.

## 11 Lezione 11