

Teoria della calcolabilità: Studia cosa può essere calcolato da un computer sotto limitazioni di risorse come spazio, tempo e energia.

Algoritmo: Un'insieme finito di istruzioni che operano su dati (discreti) con dimensione finita e rappresentabili in modo finito, le quali ognuna ha effetto limitato

Computazione: L'esecuzione delle istruzioni viene fatto per passi discreti i quali impiegano tempo finito.

Ogni passo dipende solo dai passi precedenti e una porzione finita dei dati. La scelta dello passo deve essere effettuata in modo deterministico.

Non c'è limite di memoria per i dati e al numero di passi di calcolo.

I passi di calcolo sono finiti quando si entra in uno stato finale o stallo.

OSS: Un'eccezione a questa definizione di algoritmo è costituita dalle macchine concorrenti, dove gli input variano nel tempo.

Modello Macchina di Turing (Formalismo per esprimere algoritmi)

Una macchina di Turing (MDT) è una quadrupla $M = (Q, \Sigma, \delta, q_0)$

- $Q \ni q, q', q_i, \dots$ Un'insieme finito di stati

Attenzione: Indicheremo con h lo stato speciale che indica la corretta terminazione della computazione, ma $h \notin Q$.

- $\Sigma \ni \sigma, \sigma', \sigma_i, \dots$ Un'insieme finito di simboli

- $\#$ Carattere bianco, vuoto

- \triangleright Carattere di inizio della memoria, chiamato **aspingente**

OSS: $L, R, - \notin \Sigma$

- $q_0 \in Q$ Stato iniziale

$$- f \subseteq (Q \times \Sigma) \times (Q \cup \{h\}) \times \Sigma \times \{L, R, -\} \rightarrow \text{relazione di transizione}$$

Mantiene il determinismo perché è una relazione la quale associa a un elemento, uno e un solo elemento (Transizione Univoca).

Transizioni finite perché prodotto contiene di insiemi finiti.

$f(q, \Delta) = (q', \sigma, \Delta)$ dove $\sigma = \Delta$ e $\Delta = R$. Questo ci dice che se sono a inizio mem. posso andare solo a destra.

$$\Delta = \{L, R, -\}$$

↓ ↓ ↓
 Sinistra Destra Posizione
 Attuale

Istruzioni finite! \Rightarrow Poiché Q, Σ sono finiti anche f contiene un numero finito di elementi

- Dati: y dati sono $w \in \Sigma^*$

Stringhe ↗ ↘
 chiunque riflessiva e transitiva di Σ rispetto
 alla giusta posizione dei simboli

- Operazione di concatenazione

Siano $I, J \in \Sigma^*$, $I \cdot J = \{w \cdot w' \mid w \in I, w' \in J\}$

$$\Sigma^0 = \{\varepsilon\} \rightarrow \text{Stringa Vuota}$$

$$\Sigma^1 = \Sigma \text{ perché } \Sigma^1 = \Sigma \cdot \Sigma^0 = \Sigma \cdot \{\varepsilon\} = \{a \cdot \varepsilon \mid a \in \Sigma\} = \{a \in \Sigma\} = \Sigma$$

$$\Sigma^{i+1} = \Sigma \cdot \Sigma^i = \{a w \mid a \in \Sigma \text{ e } w \in \Sigma^i\}$$

↴
 carattere aggiunto in testa alla stringa.

$$\Sigma^* = \bigcup_{i \in \mathbb{N}} \Sigma^i$$

Esempio: $\Sigma = \{0,1\}$ $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$

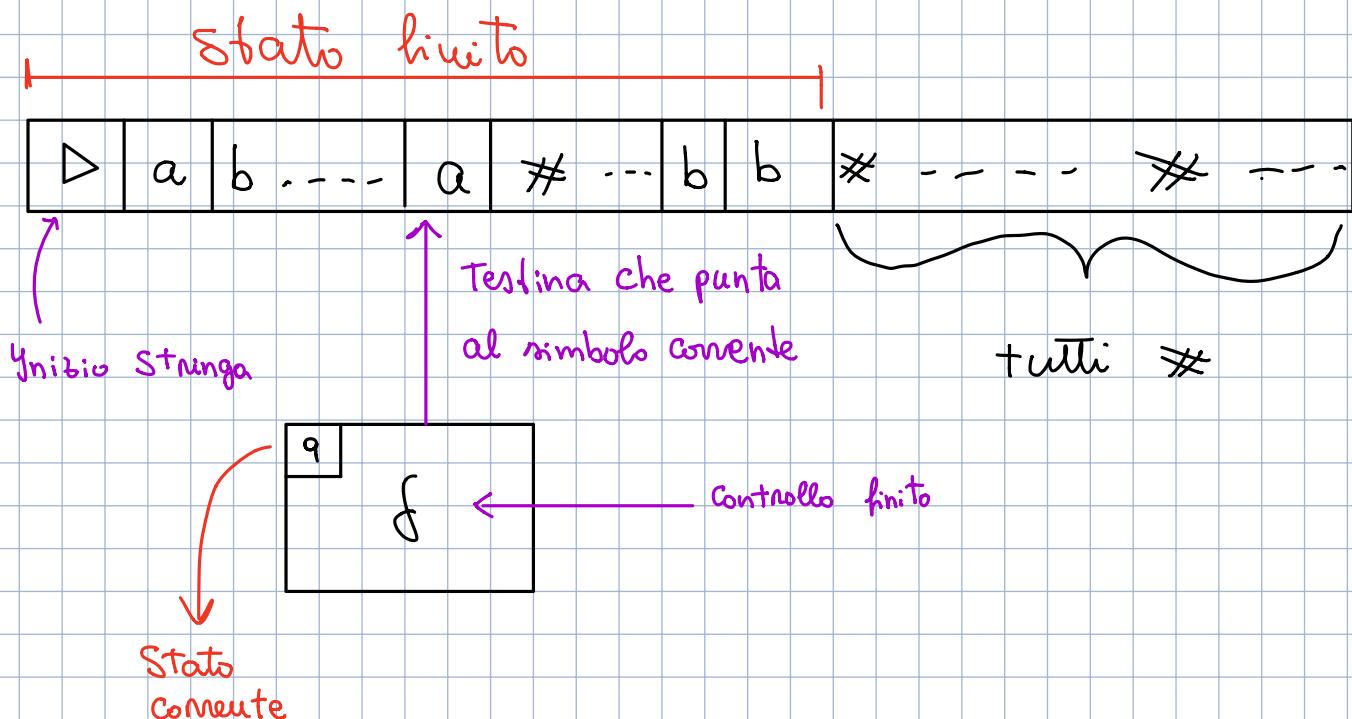
$$\Sigma' = \Sigma = \{0,1\}$$

Proprietà:

- 1) ASSOCIAZIONE: $w, w', w'' \in \Sigma^*$ $w \cdot (w' \cdot w'') = (w \cdot w') \cdot w''$
- 2) IDENTITÀ: $\epsilon \cdot w = w = w \cdot \epsilon$

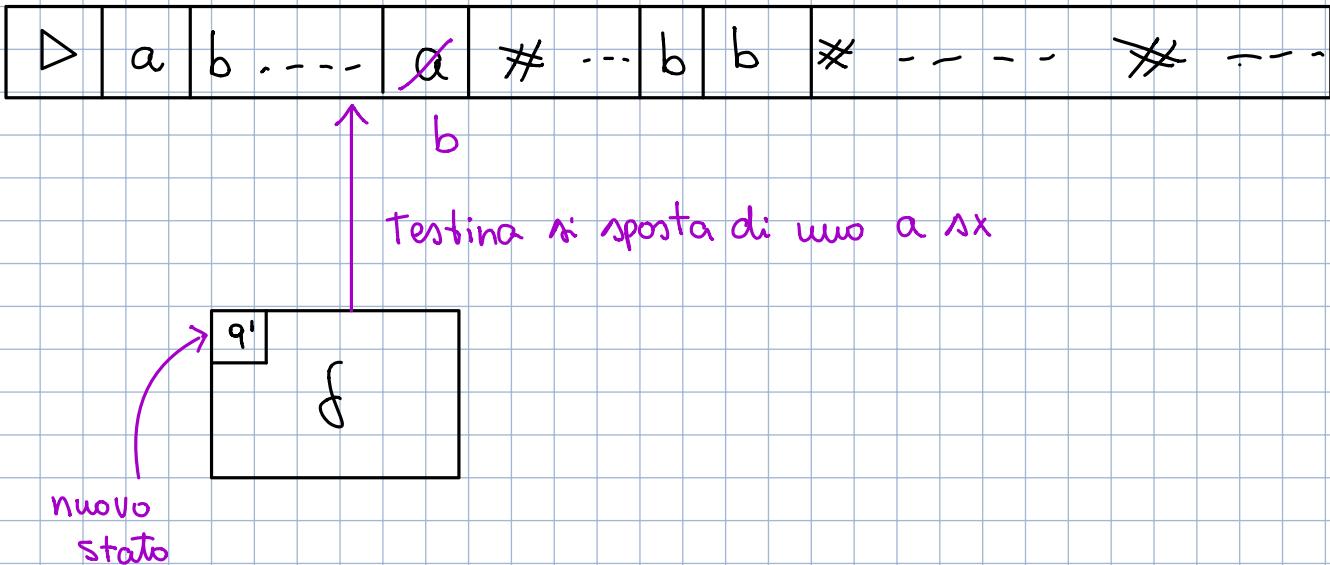
DATI FINITI

- Disegno configurazione istantanea NDT



$$f(q, a) = (q', b, L) \quad \text{Esegue}$$

ottengo:



- Configurazione di una MDT (δ):

$$(q, u \underline{\sigma} v) \in (Q \cup \{h\} \times \Sigma^* \times \Sigma \times \Sigma^F)$$



$$\Sigma^F = \Sigma^* \cdot (\Sigma \setminus \{\#\}) \cup \{\varepsilon\}$$

- $q \rightarrow$ Stato attuale

- $u \rightarrow$ Stringa a sx del carattere corrente

- $\sigma \rightarrow$ Carattere corrente

- $v \rightarrow$ Resto della stringa che termina con un carattere vuoto nullo

Esempio: $(q_0, \triangleright \# a \# a a)$

↓ ↓ ↙
Stato attuale $\underline{y_{init}}$ Simboli

Punto di computazione

$$f \xrightarrow{M} f'$$

- Transizione da una configurazione a un'altra

$$\rightarrow (q, u \underline{a} v) \rightarrow (q', u \underline{b} v) \text{ se } f(q, a) = (q', b, -)$$

2) $(q, u \in \underline{a} \cup r) \rightarrow (q', u \in \underline{b} \cup r)$ se $\delta(q, a) = (q', b, L)$

3a) $(q, u \in \underline{a}) \rightarrow (q', u \in \underline{b} \cup \#)$

3b) $(q, u \in \underline{a} \cup r) \rightarrow (q', u \in \underline{b} \cup r)$

} se $\delta(q, a) = (q', b, R)$

Le pareti dipendono solo dai precedenti, una porzione di dati è in modo deterministico.

Computatione

$(q_0, \triangleright w) \rightarrow^* (q', w')$

u
χ

"
χ'

• Converge $\Leftrightarrow \exists$ computatione t.c. $q' = h$

• Diverge $\Leftrightarrow \forall q, w' \text{ t.c. } (q_0, \triangleright w) \rightarrow^* (q', w') \exists q'', w'' \text{ t.c. } (q', w') \rightarrow (q'', w'')$

Attenzione:

$(h, \# a \#) \leftarrow$ Arresto

$(q_0, \underline{\triangleright}) \leftarrow$ Stallo

Esempi:

Esempio 1.2.4. [Una macchina che non converge per nessun ingresso]

q	σ	δ
q_0	\triangleright	q_0, \triangleright, R
q_0	a	q_0, a, R
q_0	#	$q_0, \#, R$

Un esempio di computazione non terminante della macchina a fianco è
 $(q_0, \triangleright a \# a \#) \rightarrow (q_0, \triangleright a \# a \#) \rightarrow^*$
 $(q_0, \triangleright a \# a \# \dots \# \#) \rightarrow \dots$

↑ le tabelle non rappresentano i vari casi
e come comportarsi in ognuno di essi

q	σ	δ
q_0	\triangleright	q_0, \triangleright, R
q_0		$q_0, , R$
q_0	+	$q_1, , R$
q_1		$q_1, , R$
q_1	#	$q_2, \#, L$
q_2		$h, \#, -$

La computazione per il calcolo di $1 + 2$ è la seguente:

$(q_0, \triangleright | + ||) \rightarrow (q_0, \triangleright | + ||) \rightarrow (q_0, \triangleright | \pm ||)$
 $\rightarrow (q_1, \triangleright | | |) \rightarrow (q_1, \triangleright | | |) \rightarrow (q_1, \triangleright | | | \#)$
 $\rightarrow (q_2, \triangleright | | | \#) \rightarrow (h, \triangleright | | | \#)$

$$\Sigma = \{a\} \quad Q = \{q_0\}$$

$$\Sigma = \{1, +\} \quad Q = \{q_0, q_1\}$$

→ Calcolo somma numeri naturali

q	σ	δ
q_0	\triangleright	q_0, \triangleright, R
q_0	a	q_a, \triangleright, R
q_0	b	q_b, \triangleright, R
q_0	$\#$	$q_2, \#, -$
$q_{a/b}$	a	$q_{a/b}, a, R$
$q_{a/b}$	b	$q_{a/b}, b, R$
$q_{a/b}$	$\#$	$q'_{a/b}, \#, L$
q'_a	\triangleright	q_2, \triangleright, R
q'_a	a	$q_1, \#, L$
q'_a	b	$q_N, b, -$
q'_b	\triangleright	q_2, \triangleright, R
q'_b	a	$q_N, a, -$
q'_b	b	$q_1, \#, L$
q_1	\triangleright	q_0, \triangleright, R
q_1	a	q_1, a, L
q_1	b	q_1, b, L
q_2	$\#$	h, si, R

La sua computazione su aba è la seguente:

$$\begin{aligned}
 (q_0, \geq aba\#) &\rightarrow (q_0, \triangleright aba\#) \rightarrow \\
 (q_a, \triangleright \triangleright ba\#) &\rightarrow (q_a, \triangleright \triangleright ba\#) \rightarrow \\
 (q_a, \triangleright \triangleright ba\#) &\rightarrow (q'_a, \triangleright \triangleright ba\#) \rightarrow \\
 (q_1, \triangleright \triangleright b\#\#) &\rightarrow (q_1, \triangleright \triangleright b\#\#) \rightarrow \\
 (q_0, \triangleright \triangleright b\#\#) &\rightarrow (q_b, \triangleright \triangleright \#) \rightarrow \\
 (q'_b, \triangleright \triangleright \#) &\rightarrow (q_2, \triangleright \triangleright \#) \rightarrow \\
 (h, \triangleright \triangleright \#)
 \end{aligned}$$

Figura 1.1: Una macchina che “decide” se una stringa è palindroma

La seguente computazione si arresta perché la funzione δ non è definita per lo stato q_N ; quindi, non lascia la macchina in una configurazione terminale e riflette il fatto che la stringa ba non è palindroma.

$$\begin{aligned}
 (q_0, \geq ba\#) &\rightarrow (q_0, \triangleright ba\#) \rightarrow (q_b, \triangleright \triangleright a\#) \rightarrow (q_b, \triangleright \triangleright a\#) \rightarrow (q'_b, \triangleright \triangleright a\#) \rightarrow \\
 (q_N, \triangleright \triangleright a\#)
 \end{aligned}$$

Nou c'è limite ai passi di computazione e allo spazio necessario a contenere i dati, come si può vedere nel seguente esempio.

Esempio 1.2.4. [Una macchina che non converge per nessun ingresso]

q	σ	δ
q_0	\triangleright	q_0, \triangleright, R
q_0	a	q_0, a, R
q_0	$\#$	$q_0, \#, R$

Un esempio di computazione non terminante della macchina a fianco è

$$(q_0, \geq a\#a\#) \rightarrow (q_0, \triangleright a\#a\#) \rightarrow^*$$

$$(q_0, \triangleright a\#a\# \dots \#) \rightarrow \dots$$

Linguaggi di programmazione imperativi (Formalismo per esprimere algoritmi)

Sintassi (Abstracta)

$$E \rightarrow n \mid x \mid E_1 + E_2 \mid E_1 \times E_2 \mid E_1 - E_2$$

↓ ↓
 numero variabile
 ∈ IN ∈ Var
 insieme numerabili: (Associazione biunivoca con l'insieme dei Naturali)

Espressioni Arithmetiche

$$B \rightarrow b \mid E_1 < E_2 \mid \neg B \mid B_1 \vee B_2$$

↓
 booleano { tt, ff }

Espressioni Booleane

$$C \rightarrow \text{skip} \mid x := E \mid C_1 ; C_2 \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \mid$$

↓
 Assegnamento

Comandi

for $x = E_1$, to E_2 do C | while B do C

$$\Sigma = \{a, b\} \quad Q = \{q_0, q_1, q_a, q_b\}$$

→ Controlla se la stringa è palindroma

Memoria : $\sigma : \text{Var} \rightarrow \mathbb{N}$

Stone : Unieme degli σ , $\sigma \in \text{Stone}$

Funzione di aggiornamento :

$-[-/-] : (\text{Var} \rightarrow \mathbb{N}) \times \mathbb{N} \times \text{Var} \rightarrow (\text{Var} \rightarrow \mathbb{N})$

$$\sigma[n/x](y) = \begin{cases} n & \text{se } y = x \\ \sigma(y) & \text{altrimenti} \end{cases}$$

Funzione di valutazione di un'espressione Arithmetica (come eval) :

$\mathcal{E} : \text{exp}_{\text{arit}} \times \text{Stone} \rightarrow \mathbb{N}$

$$\mathcal{E}[n]\sigma = n$$

$$\mathcal{E}[x]\sigma = \sigma(x)$$

$$\mathcal{E}[E_1 + E_2]\sigma = \mathcal{E}[E_1]\sigma \text{ piú } \mathcal{E}[E_2]\sigma$$

$$\mathcal{E}[E_1 \times E_2]\sigma = \mathcal{E}[E_1]\sigma \text{ per } \mathcal{E}[E_2]\sigma$$

$$\mathcal{E}[E_1 - E_2]\sigma = \mathcal{E}[E_1]\sigma \text{ meno } \mathcal{E}[E_2]\sigma$$

con $\text{piú} : \mathbb{N} \rightarrow \mathbb{N}^2$

per $: \mathbb{N} \rightarrow \mathbb{N}^2$

meno : $\mathbb{N} \rightarrow \mathbb{N}^2$

Esempio:

Esempio 1.3.1. Si debba valutare l'espressione⁶

$$x \times 2 - ((y - 7) + 1)$$

nella memoria σ tale che

$$\sigma(x) = 3, \sigma(y) = 5.$$

Abbiamo allora

$$\mathcal{E}[x \times 2 - ((y - 7) + 1)]\sigma =$$

$$\mathcal{E}[x \times 2]\sigma \text{ meno } \mathcal{E}[(y - 7) + 1]\sigma =$$

$$(\mathcal{E}[x]\sigma \text{ per } 2) \text{ meno } \mathcal{E}[(y - 7) + 1]\sigma =$$

$$(\sigma(x) \text{ per } 2) \text{ meno } \mathcal{E}[(y - 7) + 1]\sigma =$$

$$(3 \text{ per } 2) \text{ meno } \mathcal{E}[(y - 7) + 1]\sigma =$$

$$6 \text{ meno } (\mathcal{E}[y - 7]\sigma \text{ piú } 1) =$$

$$6 \text{ meno } ((\mathcal{E}[y]\sigma \text{ meno } 7) \text{ piú } 1) =$$

$$6 \text{ meno } ((\sigma(y) \text{ meno } 7) \text{ piú } 1) =$$

$$6 \text{ meno } ((5 \text{ meno } 7) \text{ piú } 1) =$$

$$6 \text{ meno } (0 \text{ piú } 1) =$$

$$6 \text{ meno } 1 = 5$$

Funzione di valutazione di un'espressione Booleana (come eval) :

$\mathcal{B} : \text{Bexp} \times \text{Stone} \rightarrow \text{bool}$

$$\mathcal{B}[t]\sigma = \text{tt}$$

$$\mathcal{B}[f]\sigma = \text{ff}$$

$$\mathcal{B}[E_1 < E_2]\sigma = \mathcal{E}[E_1]\sigma \text{ minore } \mathcal{E}[E_2]\sigma$$

$$\mathcal{B}[\neg B]\sigma = \text{not } \mathcal{B}[B]\sigma$$

$$\mathcal{B}[B_1 \vee B_2]\sigma = \mathcal{B}[B_1]\sigma \text{ or } \mathcal{B}[B_2]\sigma$$

minore : $\mathbb{N}^2 \rightarrow \text{bool}$

not : $\text{bool} \rightarrow \text{bool}$

Osservazione : Lo stile di definizione seguito per dare la semantica alle espressioni va sotto il nome di Denotazionale, il quale si propone di attribuire una funzione a ciascun costituto del linguaggio.

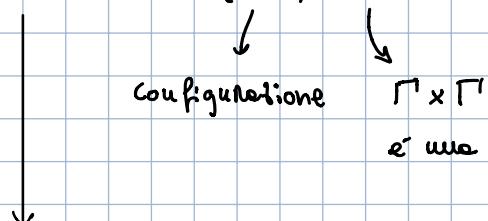
$$\text{Configurazione: } \Gamma = (C, \sigma)$$

↓ ↓
Comando memoria definita

per ogni var che compare
nel comando

Sistema di transizione:

$$(\Gamma, \rightarrow)$$



Una transizione è definita : $\langle C, \sigma \rangle \rightarrow \langle C', \sigma' \rangle$
nel seguente modo

Una computazione è

$$: \langle C, \sigma \rangle \xrightarrow{*} \langle C', \sigma' \rangle$$

definita nel seguente modo

↓

Termina con
succesivo
(converge)

$$\text{se } \langle C, \sigma \rangle \xrightarrow{*} \sigma'$$

↓

Comando
c vuoto!

Nessun comando
de eseguire.

$$\overline{\langle \text{skip}, \sigma \rangle \rightarrow \sigma}$$

$$\overline{\langle x := E, \sigma \rangle \rightarrow \sigma[n/x]}, \text{ se } \mathcal{E}[E]\sigma = n$$

$$\overline{\langle C_1, \sigma \rangle \rightarrow \langle C'_1, \sigma' \rangle}$$

$$\overline{\langle C_1; C_2, \sigma \rangle \rightarrow \langle C'_1; C_2, \sigma' \rangle}$$

⇒ Regole di
inferenza

$$\overline{\langle \text{if } B \text{ then } C_1 \text{ else } C_2, \sigma \rangle \rightarrow \langle C_1, \sigma \rangle}, \text{ se } \mathcal{B}[B]\sigma = tt$$

$$\overline{\langle \text{if } B \text{ then } C_1 \text{ else } C_2, \sigma \rangle \rightarrow \langle C_2, \sigma \rangle}, \text{ se } \mathcal{B}[B]\sigma = ff$$

$$\overline{\langle \text{for } i = E_1 \text{ to } E_2 \text{ do } C, \sigma \rangle \rightarrow \langle i := n_1; C; \text{ for } i = n_1 + 1 \text{ to } n_2 \text{ do } C, \sigma \rangle}$$

se $\mathcal{B}[E_2 < E_1]\sigma = ff \wedge \mathcal{E}[E_1]\sigma = n_1 \wedge \mathcal{E}[E_2]\sigma = n_2$

$$\overline{\langle \text{for } i = E_1 \text{ to } E_2 \text{ do } C, \sigma \rangle \rightarrow \sigma} \quad \text{se } \mathcal{B}[E_2 < E_1]\sigma = tt$$

$$\overline{\langle \text{while } B \text{ do } C, \sigma \rangle \rightarrow \langle \text{if } B \text{ then } C; \text{ while } B \text{ do } C \text{ else skip}, \sigma \rangle}$$

⇒ Azioni:

Esempio di computazione che non converge: $\langle \text{while true do skip}, \sigma \rangle$

$\rightarrow \text{if } tt \text{ then (skip; while true do skip) else skip}, \sigma$

$\rightarrow \langle \text{skip; while true do skip}, \sigma \rangle$

$\rightarrow \langle \text{while tt do skip}, \sigma \rangle$

Calcolabilità

Problema: Nel nostro caso un problema è formulato come:

- 1) Calcolare una funzione.
- 2) Decidere l'appartenenza di un certo dato a un insieme.

T-CALCOLABILE

Dati: Σ alfabeto di M , Σ_0 alfabeto di input, Σ_1 alfabeto di output

con $\#, \triangleright \notin \Sigma_0 \cup \Sigma_1$, $\Sigma_0 \cup \Sigma_1 \subset \Sigma$

Allora $f: \Sigma_0^* \rightarrow \Sigma_1^*$ è calcolabile da $M = (Q, \Sigma, \delta, q_0)$

se $\forall w \in \Sigma_0^\infty : f(w) = w' \Leftrightarrow M(w) \xrightarrow{M} (\#, \triangleright w')$

Quindi f è T-calcolabile se $\exists M$ che la calcola !!

while - Calcolabile

Un comando C calcola $f: Vars \rightarrow \mathbb{N}$ (f è while-calcolabile)

se $\forall \sigma : f(x) = n \Leftrightarrow \langle C, \sigma \rangle \xrightarrow{*} \sigma'$ con $\sigma'(x) = n$

Si noti che x viene utilizzata sia per memorizzare il valore di ingresso,

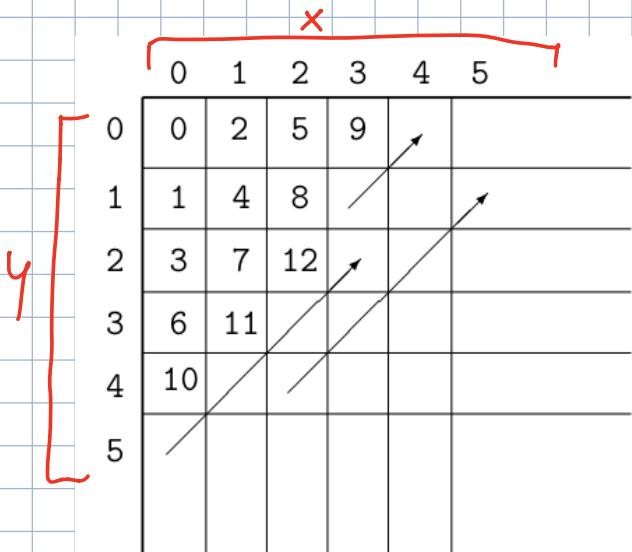
sia di uscita. f è while-calcolabile se esiste una computazione che la calcola!

OSS: Si osservi che la T-calcolabilità e la while-calcolabilità sono invarianti rispetto al modello dei dati.

In particolare se ho dati in un formato A posso codificareli nel formato B in cui opera la macchina, calcolare il risultato in formato B e poi decodificare in formato A.

Questo vale se le codifiche sono funzioni biunivoche e "facili".

Codifica a coda di roulotte



Codifica: $(x, y) \mapsto \frac{1}{2}(x^2 + 2xy + y^2 + 3x + y)$

Decodifica: $n \mapsto (n - \frac{1}{2}k \times (k+1), k - (n - \frac{1}{2}k \times (k+1)))$

$$\text{dove } k = \lfloor \frac{1}{2}(\sqrt{1+8n}-1) \rfloor.$$

Notazioni:

Una funzione è un sottoinsieme di $A \times B$ con A punti di partenza

e B codominio f.c

1) se $f(a) = b$ e $f(a) = c \Rightarrow b=c$ (unicità)

2) $\forall a \in A \exists b \in B$ f.c $f(a) = b$ (se f è totale)

Se non vale il punto 2 \Rightarrow f è parziale

- f converge su a, cioè $f(a) \downarrow \Leftrightarrow \exists b$ f.c $f(a) = b$

- f diverge su a, cioè $f(a) \uparrow \Leftrightarrow \nexists b$ f.c $f(a) = b$

- Dominio di f: $\text{dom}(f) = \{a \mid f(a) \downarrow\}$ \Rightarrow Punti dove f converge

- Immagine di f: $\text{imm}(f) = \{b \mid \exists a \in A \rightarrow f(a) = b\}$

\downarrow
Sottoinsieme di B dove la funzione converge (esiste)

Infine ricordiamo che

- f è iniettiva se e solamente se $\forall a, b \in A. a \neq b$ implica $f(a) \neq f(b)$ (a volte nella terminologia nord-americana si dice che f è uno-a-uno)
- f è surgettiva se e solamente se $\forall b \in B. \exists a \in A$ tale che $f(a) = b$ (ovvero se l'immagine e il codominio di f coincidono)
- f è biunivoca se e solamente se è iniettiva e surgettiva.

Rapporto tra algoritmi e funzioni

f è un insieme potenzialmente infinito di coppie, ma non possiamo ammigrare due f diverse allo stesso insieme, ad ogni insieme (argomento, risultato) possiamo associare una sola funzione f , mentre esistono tanti algoritmi diversi che calcolano la stessa funzione.

Ci poniamo le seguenti domande:

- 1) Quali sono le funzioni calcolabili?
- 2) Quali proprietà hanno?
- 3) Esistono funzioni non calcolabili?
- 4) Sono interessanti?

Applicazioni per descrivere algoritmi:

- 1) Hardware (Macchina di Turing): Ogni volta che devo cambiare programma, devo cambiare macchina
- 2) Software: Ho un interprete, cioè la semantica, fissa. Se cambio programma non devo cambiare macchina. I programmi stanno in memoria e variano come i dati di ingresso.
 - a) Programmi While: Base della programmazione iterativa, semantica operazionale e usati per studio delle compledità
 - b) Funzioni ricursivei: Base della programmazione funzionale.

Funzioni ricorsive primitive (Formalismo per descrivere algoritmi) \Rightarrow Esprime un sottoinsieme delle funz. calcolabili totali

Per formalizzare le funzioni viene usata la λ -notazione.

$\lambda <\text{argomenti}> . <\text{espressione}>$

OSS: Qualsiasi var. che compare nell'expr. ed è stata dichiarata come argomento, si dice sia legata da λ in espressione.

Esempio: $\lambda x . x \rightarrow$ Prende x in input e restituisce x in output

Classe C

La classe C delle funzioni ricorsive primitive è la minima classe di funzioni che obbediscono

Alle seguenti regole di inferenza, regole di sintesi per definire le funzioni.

Attenzione: Ciò è una funzione ricorsiva primitiva se o fa parte degli schemi primitivi di base oppure può essere ottenuta a partire da essi applicando la composizione e la ricorsione primitiva un numero finito di volte.

Schemi primitivi di base:

1) Zero : $\lambda x_1, \dots, x_k. 0$ con $k \geq 0$

2) Successione : $\lambda x. x + 1$

3) Identità : $\lambda x_1, \dots, x_k. x_i$ con $1 \leq i \leq k$

la classe C è chiusa per:

4) Composizione : Se $g_1, \dots, g_k \in C$ sono funzioni in m variabili, ed $h \in C$ è una funzione in k variabili, appartiene a C anche la loro composizione.

$$\lambda x_1, \dots, x_m. h(g_1(x_1, \dots, x_m), \dots, g_k(x_1, \dots, x_m))$$

5) Ricorsione primitiva: Se $h \in C$ è una funzione in $k+1$ variabili, $g \in C$ è una funzione in k variabili, allora appartiene a C anche la funzione f in k variabili definita da:

$$\begin{cases} f(0, x_2, \dots, x_k) &= g(x_2, \dots, x_k) \\ f(x_1 + 1, x_2, \dots, x_k) &= h(x_1, f(x_1, x_2, \dots, x_k), x_2, \dots, x_k) \end{cases}$$

Esempio: Definizione ricorsiva delle somma $\Rightarrow f_5 = \lambda x, y. x + y$.

$$f_1 = \lambda x. x \quad III$$

$$f_2 = \lambda x. x + 1 \quad II$$

$$f_3 = \lambda x_1, x_2, x_3. x_2 \quad III$$

$$f_4 = f_2(f_3(x_1, x_2, x_3)) \quad IV$$

$$\begin{cases} f_5(0, x_2) = f_1(x_2) \\ f_5(x_1 + 1, x_2) = f_4(x_1, f_5(x_1, x_2), x_2) \end{cases} \quad V$$

Regole di Valutazione interna-ristrica:

$$\begin{aligned} f_5(2, 3) &= \\ f_4(1, f_5(1, 3), 3) &= \\ f_4(1, f_4(0, f_5(0, 3), 3), 3) &= \\ f_4(1, f_4(0, f_1(3), 3), 3) &= \\ f_4(1, f_4(0, 3, 3), 3) &= \\ f_4(1, f_2(f_3(0, 3, 3)), 3) &= \\ f_4(1, f_2(3), 3) &= \\ f_4(1, 4, 3) &= \\ f_2(f_3(1, 4, 3)) &= \\ f_2(4) &= \end{aligned}$$

Regole di valutazione esterna:

$$\begin{aligned} f_5(2, 3) &= \\ f_4(1, f_5(1, 3), 3) &= \\ f_2(f_3(1, f_5(1, 3), 3)) &= \\ f_3(1, f_5(1, 3), 3) + 1 &= \\ f_5(1, 3) + 1 &= \\ f_4(0, f_5(0, 3), 3) + 1 &= \\ f_2(f_3(0, f_5(0, 3), 3)) + 1 &= \\ f_3(0, f_5(0, 3), 3) + 1 + 1 &= \\ f_5(0, 3) + 2 &= \\ f_1(3) + 2 &= \\ 3 + 2 &= 5 \end{aligned}$$

\rightarrow def di chiusura

Definita una op. $\#$

$$\forall x, y \in C, x \# y \in C$$

Meno - limitato:

$$f_7(x, y) = y \quad f_8(x, y) = x$$

$$\begin{cases} \text{pred}(0) = 0 \\ \text{pred}(x+1) = f_8(x, \text{pred}(x)) \end{cases}$$

Funz n.p ottenuta
da la funzione
base identità

$$f_9(x, y, z) = \text{pred}(f_3(x, y, z))$$

$$\begin{cases} f_{10}(0, y) = f_1(y) \\ f_{10}(x+1, y) = f_9(x, f_{10}(x, y), y) \end{cases}$$

↓

$$x \div y = f_{10}(f_7(x, y), f_8(x, y))$$

Somma (Generalizza il successione)

$$\begin{cases} 0+y = y \\ (x+1)+y = (x+y)+1 \end{cases}$$

Prodotto (Generalizza la somma)

$$\begin{cases} 0 \times y = 0 \\ (x+1) \times y = (x \times y) + y \end{cases}$$

Potenza (Generalizza il prodotto)

$$\begin{cases} x^0 = 1 \\ x^{(y+1)} = x \times (x^y) \end{cases}$$

Definizioni:

1) Definizione 1.6.3. Diciamo che la relazione $P(x_1, \dots, x_k) \subseteq \mathbb{N}^k$ è *ricorsiva primitiva* se lo è la sua funzione caratteristica χ_P , definita come

$$\chi_P(x_1, \dots, x_k) = \begin{cases} 1 & \text{se } (x_1, \dots, x_k) \in P \\ 0 & \text{se } (x_1, \dots, x_k) \notin P \end{cases}$$

2) $R = \{x \in \mathbb{N} \mid x \text{ è un numero primo}\}$ è ricorsiva primitiva

3) (unica fattorizzazione) se $p_0 < \dots < p_k \dots$ sono i numeri primi, cioè $R = \{p_0, \dots, p_k \dots\}$, allora per ogni $x \in \mathbb{N}$ esiste un numero *finito* di esponenti $x_i \neq 0$ tali che $x = p_0^{x_0} p_1^{x_1} \dots p_n^{x_n} \dots$;

4) la funzione $(x)_i$ che restituisce l'esponente dell' i -esimo fattore p_i della fattorizzazione di x è ricorsiva primitiva.

Gödelizzazione: $\text{MOT} \leftrightarrow \mathbb{N}$

Si chiama anche

enumerazione (effettiva) canonica

Numerazione effettiva:

F binaria che prende un algoritmo e da un numero. le funz. non dipendono dal significato dei simboli in cui è stato scritto l'algoritmo.

la Gödelizzazione

è \uparrow che ogni seq. di numeri naturali

può essere codificata univocamente come $n = p_0^{n_0+1} \cdot p_1^{n_1+1} \cdots p_k^{n_k+1}$ con p_i numeri primi, ovvero come il prodotto di un numero finito di fattori, e viceversa.

Quindi è possibile andare a rappresentare algoritmi interi come un numero.

Esempio: Data $M = (Q, \Sigma, \delta, q_0)$ con $Q = \{q_0, \dots, q_n\}$ e $\Sigma = \{\sigma_0, \dots, \sigma_n\}$

Ogni quintupla $(q_j, \sigma_j, q_k, \sigma_l, \Delta) \in \delta$ è possibile codificarla come

$$p_0^{j+1} \times p_1^{j+1} \times p_2^{k+1} \times p_3^{l+1} \times p_4^{m_\Delta} \quad \text{con } p_0 < \dots < p_k \text{ numeri primi.}$$

Questo vuol dire che posso associare ad ogni macchina di Turing un numero

e le posso enumerare con un indice i . Anche alle computazioni ecc...

Funzione di Ackermann

La funzione di Ackermann non è definibile mediante schemi di ricorrenza primitiva definiti in precedenza, ma è totale ed ha una definizione intuitivamente accettabilissima.

$$\begin{aligned} A(0, 0, y) &= y \\ A(0, x+1, y) &= A(0, x, y) + 1 \\ A(1, 0, y) &= 0 \\ A(z+2, 0, y) &= 1 \\ A(z+1, x+1, y) &= A(z, A(z+1, x, y), y). \Rightarrow \text{Doppia ricorrenza} \end{aligned}$$

Cresce più velocemente di ogni funzione ricorsiva primitiva ma non è ricorsiva primitiva. Quindi è una funzione totale non ricorsiva primitiva.
Questa funzione calcola una sorta di esponenziale generalizzato.

$$\begin{aligned} A(0, x, y) &= y + x \\ A(1, x, y) &= y \times x \\ A(2, x, y) &= y^x \\ A(3, x, y) &= \underbrace{y \cdot \dots \cdot y}_{x \text{ volte}} \end{aligned}$$

Ottieniamo quindi che non esiste un formalismo capace di esprimere tutte le funzioni totali!

Diagonalizzazione (si può applicare ad un qualunque formalismo che definisce SOLO funzioni totali)

Non esiste un formalismo che esprime tutte e solo le funzioni totali calcolabili

Dim:

Sia A un alfabeto con $\#$ finito di simboli.

- Posso enumerare le funzioni f_n (ad esempio con la lista di Gödel)
- Definisco $g(x) = f_x(x) + 1$ (g è totale) \Rightarrow **Diagonalizzazione** (Stesso x sia per indice che parametro)
- Se cerco g nella lista delle funz. ricorse primitive, non lo trovo perché $\forall n. g(n) \neq f_n(n)$, quindi $g \neq f_n$

Quindi siamo obbligati a considerare anche le funzioni parziali che veniamo indicate con φ e ψ .

Non si applica la diagonalizzazione ad esse perché:

Sia ψ_n la funzione con n -esima definizione (n -esima posizione nella lista) e proviamo a diagonalizzare:

$$\varphi(x) = \psi_x(x) + 1$$

Diciamo che $\varphi(x)$ ha indice i , quindi $\psi_i(x) = \varphi(x) = \psi_x(x) + 1$

$$\text{Se } \psi_i(x) \uparrow \Rightarrow \psi_x(x) \uparrow \Rightarrow \psi_x(x) + 1 \uparrow$$

↓
diverge ↓
diverge ↓
diverge

es: $\text{div}^*(x,y) = \begin{cases} \text{div}(x,y) & \text{se } y \neq 0 \\ 0 & \text{se } y = 0 \end{cases}$

Definito su dove non è def.

OSS: Inoltre non si possono estendere tutte le funzioni parziali a funzioni totali perché non sempre c'è un algoritmo che calcola la funzione estesa.

Funzioni Ricorsive Generali (μ -ricorsive) \Rightarrow insieme delle funzioni calcolabili (che sono parziali e totali)

Viene ammesso lo schema delle fun. ricursive primitive per poter esprimere le funzioni parziali: la classe dei funzioni μ -ricorsive è la minima classe R t.c:

I - V) Per le ric. primitive

VI) Minimizzazione: Se $\varphi(x_1, \dots, x_n, y) \in R$ in $n+1$ variabili, allora

Notazione

la funzione ψ in n variabili è in R se è definita da

$\mu y. I$

indica il minimo $y \in I$ (se c'è)

$$\psi(x_1, \dots, x_n) = \mu y [\varphi(x_1, \dots, x_n, y) = 0 \text{ e converge}]$$

$$\forall z \leq y. \varphi(x_1, \dots, x_n, z) \downarrow$$

IMPORTANTE

Per ogni valore z minore del minimo (y) su cui $\varphi(y) = 0$ la funzione φ deve convergere almeno la funzione ψ può non essere μ -ricorsiva (esco fuori dalla classe)

Questa condizione può essere rimossa se φ è R.P. perché avendo una funz. totale convergerà sicuramente per $z < y$

Questo ci indica che calcolo le funz. sugli argomenti \vec{x} delle φ e una certa y .

Se vale D, il risultato è y , altrimenti deve convergere e vuol avanti incrementando y di 1 e ricalcolando.

Esempio 1

$$\varphi = \lambda x. y. z$$

$$\varphi(y) = \mu y. \varphi(x, y) = 0 \quad \Rightarrow \quad \begin{array}{l} y=0 \\ \text{while } \varphi(y, y) \neq 0 \\ \quad y = y+1 \end{array} \quad \left[\begin{array}{l} \text{Non} \\ \text{termina} \end{array} \right] = \not f y$$

Esempio 2

$$\varphi'(x) = \begin{cases} 0 & \text{se } x \text{ pari} \\ 1 & \text{se } x \text{ dispari} \end{cases} \quad \Rightarrow \quad \begin{array}{l} \varphi(4, 0) = 1 \\ \varphi(4, 1) = 1 \\ \varphi(4, 2) = 0 \end{array} \quad \rightarrow \text{Result!}$$

OSS: Se definisco per comi $\varphi(x) = \begin{cases} \mu y. [\psi < g(x), h(x, y) = 0] & \text{se } \exists y \text{ con } g, h \\ 0 & \text{altrimenti} \end{cases}$ ric. primitive

Ottengo f ric. primitiva perché composizione di funz. ricorse primitive, ed anche totale perché converge sempre. Se pongo dei limiti al numero dei tentativi, dato da $y < g(x)$, non ricade nelle ricorse primitive e non ci sono problemi di parzialità.

Teorema: Una funzione p è μ -ricorsiva se la sua funz. caratteristica χ_p è μ -ricorsiva

Tesi di Church - Turing

Le funzioni (intuitivamente) calcolabili sono tutte e solo le funzioni (Parziali) T-calcolabili.

Risultati: T-calcolabili = while calcolabili = μ -calcolabili

Teorema:

- i) Le funzioni calcolabili sono $\#(\mathbb{N})$; inoltre anche le funzioni calcolabili totali sono $\#(\mathbb{N})$
- ii) esistono funzioni non calcolabili.

Dimostrazione.

- i) Costruisci $\#(\mathbb{N})$ MdT M_i che dal nastro vuoto scrivono $|^i$ e si arrestano (sono le funzioni costanti). Che non siano più di $\#(\mathbb{N})$ segue dal fatto che le MdT si possono enumerare, come fatto intuire a pagina 35.
- ii) Con una costruzione analoga a quella di Cantor (la classe dei sottoinsiemi di \mathbb{N} non è numerabile) si vede che $\{f : \mathbb{N} \rightarrow \mathbb{N}\}$ ha cardinalità $2^{\#(\mathbb{N})}$. \square

i banette
↓

Non sono di più di $\#(\mathbb{N})$

→ Perché le MdT si possono enumerare.

Diagonalizzazione

Se neppureto le funz. calc.
→ in tabella, faccio i complementi delle diag. e non li trovo tra le calcolabili!

Come definito in precedenza con la Gödelizzazione, una numerazione effettiva è una F biunivoca che prende un algoritmo e da un numero, e non dipende dal significato dei simboli in cui è scritto un algoritmo.

Tutti i risultati che seguono sono indipendenti dall'enum. effettiva scelta.

Notazione: φ_i = Funzione (Parziale) che la macchina (algoritmo) M_i calcola. i = indice della macchina (algoritmo)

1) Teorema Padding (lemma): Ogni funzione calcolabile φ_x ha $\#N$ indici.
inf. di indici
 $\forall x \exists i$ può costruire mediante f.p.n un insieme A_x t.c.

$$\forall y \in A_x. \varphi_y = \varphi_x \quad \text{con } \#(A_x) = \#(\mathbb{N}) \quad \text{cioè } \varphi_y(n) = m \Leftrightarrow \varphi_x(n) = m \quad \text{se converge} \\ \varphi_y(n) = \uparrow \Leftrightarrow \varphi_x(n) = \uparrow \quad \text{se diverge}$$

Suggerito: Dato un algoritmo che calcola f , tramite manipolazioni puramente simboliche, genera algoritmi con lo stesso significato (quanti ne voglio).

Dim:

Dato p_i scrivo il prog P_i ; skip poi P ; skip; skip ... posso mettere quanti skip voglio.

Poco generare un n. inf. di programmi che calcolano la stessa funzione

2) Teorema Forma Normale: $\exists U, T : \forall i, x \varphi_i(x) = U(\mu y. T(i, x, y) = \perp)$

\downarrow
Funzione \downarrow
Predicato N.P
Calc totali:
Predicato di libere
N.P Calc totali:
Calc totali:
Existe y t.c il prog. i su input x termina

Dim: Sia $T(i, x, y)$ vero $\Leftrightarrow y$ è la codifica di una computazione

terminante della $M_i(x)$. Cioè, per calcolare T dato i recupera M_i dalla lista e comincia a scandire i valori y . Ma man mano controlla se la computazione terminante è

della forma $M_i(x) = c_0, c_1, \dots, c_n$.

Se lo è allora $c_n = (h, \Delta w)$ e definisci $U(y) = w$

Sia $U(y) = w$

\downarrow
risultato
computazione

$\underline{U(\mu y. T(i, x, y))} = \varphi_i(x)$

\swarrow \searrow
tale y $= w$

$\uparrow \quad \downarrow$
 w

Questo teorema ci dice che fra tutti gli algoritmi che calcolano una determinata funzione ce n'è uno privilegiato, nel senso che ha una forma speciale.

Di conseguenza ogni funzione ha una rappresentazione privilegiata.

OSS: Un'immediata conseguenza del teo. F.N è che ogni funzione calcolata da una MDT ammette una definizione μ -ricorsiva cioè le funzioni T-calcolabili sono μ -ricorsive.

Infatti abbiamo che le funz. μ -calcolabili sono T-calcolabili.

Ottieniamo quindi il seguente teorema:

3) **Teorema:** Una funzione è T-calcolabile \Leftrightarrow è μ -calcolabile

Corollario teorema F.N e il teorema di eq T-calcolabile, μ -calcolabile:

È noto che le funzioni ricorsive "possono" essere rappresentate con un programma FOR

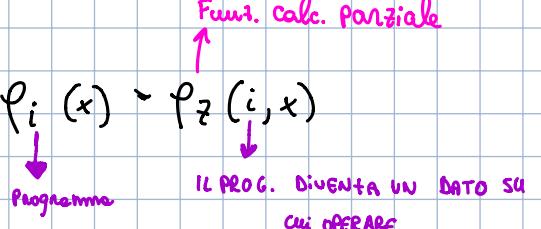
e quelle μ -ricorsive con un programma WHILE ottieniamo la seguente deduzione:

Ogni programma può essere scritto con un WHILE e 2 FOR (in forma normale).

Detto in altri termini:

Ogni funzione calcolabile parziale può essere ottenuta da due funzioni primitive ricorse e una sola applicazione dell'operatore μ .

4) **Teorema di enumerazione:** $\exists z \text{ t.c. } \forall i, x \quad \varphi_i(x) \sim \varphi_z(i, x)$



z è la MDT universale (l'interprete)

Dim:

$$\varphi_i(x) = \bigcup_{y \in \Sigma} (\mu y. T(i, x, y)) \underset{\text{teo. F.N}}{=} \varphi_z(i, x) \quad \text{la } y \text{ è legata.}$$

\downarrow Per C-T la funz. avrà un indice z con argomenti i, x perché non legati \uparrow

Questo teorema ci dice che un formalismo universale, cioè uno che esprime tutte le funzioni calcolabili, è così potente da riuscire ad esprimere l'interprete dei propri programmi.

Esso ci garantisce che esiste una MDT universale M_2 che ha in ingresso uno H_i e il dato x , e si comporta come H_i : su quel dato in par.

5) Teorema del Parametro ($S \vdash n \vdash m$)

Dato una funzione $x \mapsto y$, se fino come parametro $x = z$,
 $z \mapsto y$ può essere calcolata da un'altra funzione

$$\forall n, m > 0 \quad \exists S_n^m \text{ con } m+1$$

Funzione calcolabile totale iniettiva

$$H_i, x, \dots, x_m, y_1, \dots, y_n \vdash \varphi_x(x_1, \dots, x_m, y_1, \dots, y_n) = \varphi_{S(i, x_1, \dots, x_m)}(y_1, \dots, y_n)$$

Parametri:

Caso ($n = m = 1$): $\exists s$ calcolabile, totale, iniettiva | $\forall i, x, y \quad d_y. \varphi_i(x, y) = \varphi_{S(i, x)}(y)$

Scopo: individuare un'altra macchina
 che esegue il mio algoritmo
 originale fornendo la x

Dim intuitiva:

Parametro
 x diventa parametro
 e argomento di S .
 Viene trovata un'altra
 macchina.

1) Dato i premoli H_i , attraverso una f A.P perché le codifiche sono A.P quindi calc. totale.

2) Scrivere x, y sul nastro H_i (posso farlo con una punt. A.P)

Date queste 2 op. ho definito un algoritmo, quindi ho un indice $J = S(i, x)$ per la terza di C-T. Abbiamo dim. che $\exists s$ calc. tot. che funziona come ci aspettavamo.

Per l'iniettività?

Se ho già trovato J , da J genero diversi indici i di macchine che calcolano la stessa funzione (padding lemma).

Lo uso per trovare un indice $h >$ di tutti quelli già visti.

Questo indica che la funzione è strettamente crescente, quindi invertibile.

Questo teorema è importante perché alle basi della valutazione parziale ed è uno strumento molto utile nel dimostrazioni nell'teoria della calcolabilità.

6) Teorema di esprimività

Un formalismo è Turing equivalente se:

- ha un algoritmo universale
- vale il teorema del parametro.

7) TEOREMA DI RICORSIONE

$\forall f$ calcolabile Totale $\exists n$ t.c $p_n = \varphi_{f(n)}$

Dim.

$$\exists s \varphi_{s(i,u)}(z) = \varphi_i(u,z) = \psi(u,z) = \begin{cases} \varphi_{\varphi_u(u)}(z) & u \text{-esima macchina} \\ \text{indefinito} & \text{altrimenti} \end{cases}$$

calcolabile

$\varphi_s(z) = \varphi_i(u,z)$ TEO.
PARAMETRO
C.T
ci dice che ha un indice i

$$d(u) = \lambda u . s(i,u)$$

Nou dipende dalla f, ed è calc. tot. iniettiva

ne prendo uno

$$f_r(x) \stackrel{(2)}{=} f(d(x)) \rightarrow \varphi_r(r) \downarrow \Rightarrow \varphi_{d(r)}(z) = \varphi_{\varphi_r(r)}(z)$$

C.T
ci dice che ha un indice r

$$n \stackrel{(3)}{=} d(r)$$

Punto fisso

$$\varphi(n) \stackrel{(3)}{=} \varphi_{d(r)} \stackrel{(1)}{=} \varphi_{\varphi_r(r)} = \varphi_{\varphi(d(r))} \stackrel{(3)}{=} \varphi_{\varphi(n)}$$

e iniett.
di d

OSS: I punti fissi sono infiniti numerabili!

Questo teorema ci dice che la funzione f "trasforma" programmi in programmi e la trasformazione effettuata non cambia la funzione calcolata ovvero trasforma p_n in $p_{f(n)}$ mantenendo la semantica.

OSS:

- Il punto fisso è calcolabile mediante una funz. tot. iniettiva f a partire dall' indice di p;
- Ci sono $\#(N)$ punti fissi di f

Un insieme ricorsivo è ricursivamente enumerabile

I è un insieme ricorsivo $\Leftrightarrow \chi_I(x) = \begin{cases} 1 & \text{se } x \in I \\ 0 & \text{se } x \notin I \end{cases}$ è calc. totale

↓
chiamati anche
decidibili → Rappresentabili anche tramite funzioni calcolabili totali.

Ci dice che un insieme è ricorsivo se è possibile costruire un algoritmo che in un tempo finito sia in grado, dato un qualunque numero naturale, se esso appartiene o meno all'insieme.

Rappresentano, come le funzioni calcolabili i problemi

↑
insolubili
o
semidecidibili

J è un insieme ricursivamente enumerabile $\Leftrightarrow \exists i. J = \text{dom}(\varphi_i)$ oppure $J = \emptyset$

↓
chiamati anche semidecidibili

↓
Indice che determina una funzione ricorsiva

Ci dice che un insieme è ricorsivo enumerabile se dato un elemento a , è possibile stabilire in un numero finito di passi se esso appartenga o meno ad esso.

Se tuttavia l'elemento non appartiene all'insieme, allora non c'è assicurata la possibilità di verificare la non appartenenza in un numero finito di passi (semidecidibilità).

Si nota che l'insieme degli insiemi Ricorsivi è insieme degli insiemi N.E

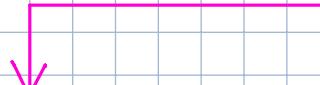
Proprietà:

1) Se I è ricorsivo allora I è ricorsivo enumerabile, ma non è vero il contrario!

Dim:

$$\varphi_i(x) = \begin{cases} \rightarrow \text{se } \chi_I(x)=1 \\ \uparrow \text{altrimenti} \end{cases}$$

D.M



$$U = \left\{ x \mid \varphi_x(x) \downarrow \right\} \text{ è l.e ma non n}$$

Dim:

Per dimostrare X_U calc. tot. cioè $U \in N$

$$f(x) = \begin{cases} \varphi_x(x) + 1 & \text{se } X_U(x) = 1 \equiv x \in U \\ 0 & \text{se } X_U(x) = 0 \equiv x \notin U \end{cases}$$

e calc. tot.

f ha un indice cioè ha un algo che la calcola? NO!

Qualunque indice lo prende, non va bene

es:

$$\text{Se } n \quad f_n(n) = \varphi(n) + 1 \quad \text{se } n \in U \quad \Rightarrow \text{Non succede mai}$$

di approssimazione

$$\text{se } n \quad f_n(n) \uparrow \text{ e } f(n) = 0 \quad \text{se } n \notin U \quad \Rightarrow \text{Non succede nemmeno se } n \notin U$$

Quindi qualiasi n prende uno è ok.

\bar{U} può essere l.e? No! Prop 2!

$$2) I \text{ e } \bar{I} \text{ sono l.e} \Leftrightarrow I \text{ e } \bar{I} \text{ sono n}$$

Dim:

$$\exists \Psi_I \in \varphi_{\bar{I}} \text{ t.c. } \text{dom}(\varphi_I) = I, \text{ dom}(\varphi_{\bar{I}}) = \bar{I}$$

Voglio sapere se $X_I(x) \underset{+}{\leq} 1$ in tempo finito

$$\varphi_I(x) \rightarrow \text{paro}, \text{ se femina } x \in \text{dom}(\varphi_I) = I \Rightarrow X_I(x) = 1$$

$$\text{altrimenti } \varphi_{\bar{I}}(x) \rightarrow \text{paro}, \text{ se femina } x \in \text{dom}(\varphi_{\bar{I}}) = \bar{I} \Rightarrow X_{\bar{I}}(x) = 1$$

TEOREMA: $J \text{ è n.e} \Leftrightarrow \begin{cases} J = \emptyset \\ \exists f \text{ calc. tot. t.c. } \text{imm}(f) = J \end{cases}$

Dtm:

1° caso banale \Rightarrow la funzione caratteristica da sempre 0

2° caso

Sia $J = \text{dom}(\varphi_i)$ dalla φ_i contiene f

1) Poiché $J \neq \emptyset$, trova $\bar{n} \in \text{dom}(\varphi_i)$

	0	1	2	3	4	5
0	0	2	5	9		
1	1	4	8	13		
2	3	7	12			
3	6	11				
4						
5						

Ho scelto
corrisponde al $\bar{n} \in \text{dom}(\varphi_i)$

Ho trovato \bar{n} con codice di columba

2) Definire f in modo tale che se qualcosa non termina dopo un certo num pari ecc. - restituisce \bar{n}

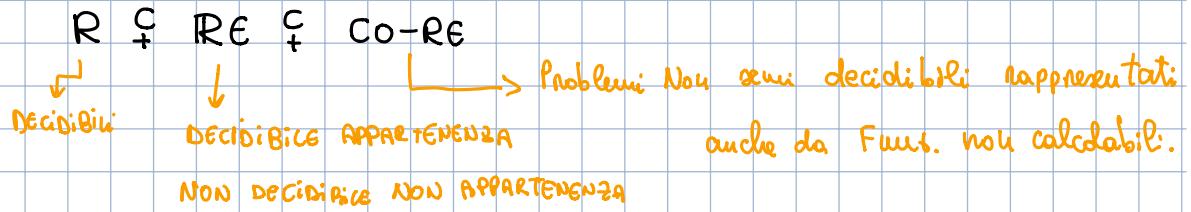
	0	1	2	3	4	5
0	0	2	5	9		
1	1	4	8	13		
2	3	7	12			
3	6	11				
4						
5						

$\rightarrow = \langle 3, 11 \rangle$

$\left\langle m, n \right\rangle$
*pari argomento

Calcola $\varphi_i(n)$ per m pari $\begin{cases} \downarrow \text{pari } f(\langle m, n \rangle) = n \\ \not\downarrow \text{pari } f(\langle m, n \rangle) = \bar{n} \end{cases}$

OSS: Ottieniamo quindi la seguente relazione



K e Riduzioni

$$U = \left\{ x \mid \varphi_x(x) \downarrow \right\} = \text{insieme degli algoritmi applicati a se stessi e convergenti} \Rightarrow \bar{U} \text{ e.r. perché } \bar{U} = \text{dom}(\varphi_x)$$

ma non è ricorsivo

Dim:

Per dimostrare \bar{U} calc. tot. cioè $\bar{U} \in \Sigma_1^0$

$$f(x) = \begin{cases} \varphi_x(x)+1 & \text{se } \varphi_x(x) \downarrow \equiv x \in U \\ 0 & \text{se } \varphi_x(x) \uparrow \equiv x \notin U \end{cases}$$

\downarrow
è calc. tot.

f ha un indice cioè ha un algo che la calcola? NO!

Qualunque indice io prendo, non va bene

es.

Se $n \in \bar{U}$, $f_n(n) = \varphi(n) = f_n(n)+1 \Rightarrow$ Non succede mai

Bisogna ragionare

se $n \notin \bar{U}$, $f_n(n) \uparrow$ e $f(n) = 0$ quindi $\varphi_n(n) \neq f(n) = 0 \Rightarrow$ Non succede nemmeno nel nullo

Quindi: non esiste algoritmo per decidere se $x \in \bar{U}$ o no \Rightarrow Problema insolubile

\bar{U} può essere e.r.? No! Prop 2! $\Rightarrow \bar{U} \in \text{Co-RE}$

PROBLEMA DELLA FERMATA

$$U_f = \left\{ \langle i, x \rangle \mid \varphi_i(x) \downarrow \right\} \text{ ovvero } U_f = \left\{ (x, y) \mid \exists z. T(y, x, z) \right\}$$

\downarrow
Prog. argomento

questo insieme non è r. Se lo formasse avremmo che $\langle x, x \rangle \in U_f$

$\Leftrightarrow x \in \bar{U}$, cioè se U_f fosse ricorsivo, lo sarebbe anche \bar{U} .

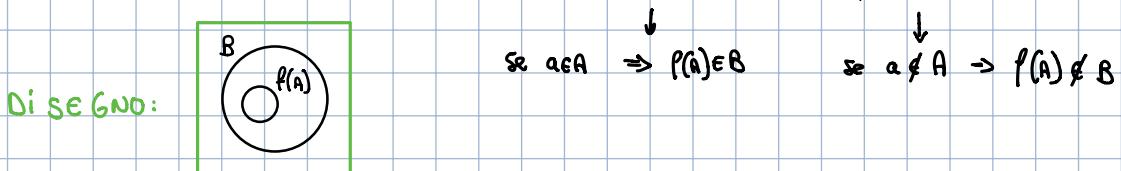
la tecnica usata per dimostrare che \mathcal{U}_0 non è ricorsivo, quindi per collegare \mathcal{U}_0 e \mathcal{U} si chiama **Riducibilità**.

Riducibilità

Una riduzione è una particolare funzione f che trasforma un problema (insieme o classe) A in un altro problema B , in modo da mantenere inalterata la caratteristica principale.

Def. 1 A si riduce a B secondo la riduzione f ($A \leq_f B$) tutte e sole le volte che

$$a \in A \Leftrightarrow f(a) \in B, \text{ ovvero } f(A) \subseteq B \text{ e } f(\bar{A}) \subseteq \bar{B}$$



$$\text{Proprietà: } A \leq_f B \Leftrightarrow \bar{A} \leq_f \bar{B}$$

$$\text{Dim: } x \in \bar{A} \Leftrightarrow x \notin A \Leftrightarrow f(x) \notin B \Leftrightarrow f(x) \in \bar{B}$$

Def. 2: Si definisce una relazione di riduzione dove F è una particolare classe di funz.

$$A \leq_F B \Leftrightarrow \exists f \in F \text{ t.c. } A \leq_f B$$

Ci interessano solo quelle riduzioni \leq_F che danno origine a classi di probl.

Omogenee.

Def. 3: Siano \mathcal{D} ed \mathcal{E} due classi di problemi con $\mathcal{D} \subseteq \mathcal{E}$. Una relazione di riduzione

$$\leq_F \text{ classifica } \mathcal{D} \text{ ed } \mathcal{E} \Leftrightarrow \text{per ogni problema } A, B, C$$

- i) $A \leq_F A$ (Riflessiva)
- ii) $A \leq_F B, B \leq_F C \Rightarrow A \leq_F C$ (Transitività)
- iii) $A \leq_F B, B \in \mathcal{D} \Rightarrow A \in \mathcal{D}$ (\mathcal{D} ideale = chiuso all'ingiù per riduzione)
- iv) $A \leq_F B, B \in \mathcal{E} \Rightarrow A \in \mathcal{E}$ (\mathcal{E} ideale = chiuso all'ingiù per riduzione)

- i) $id \in F$ (F ha identità)
- ii) $f, g \in F \Rightarrow f \circ g \in F$ (F chiusa per composizione)
- iii) $f \in F, B \in \mathcal{D} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{D}$
- iv) $f \in F, B \in \mathcal{E} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{E}$

Ogni punto i è a sinistra.
è vero \Leftrightarrow vale il punto
i a destra.

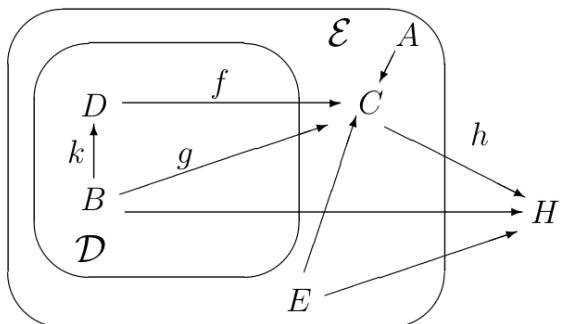
Def. 4: Se \leq_F classifica D ed E, HA, B, H problemi

- i) $A \equiv B$ se $A \leqslant_F B$ e $B \leqslant_F A$
 (si dice anche che $\{B \mid A \equiv_F B\}$ è il *grado* di A , o anche che A è equivalente a B rispetto a \leqslant_F)²⁰

ii) H è \leqslant_F -arduo per \mathcal{E} se $\forall A \in \mathcal{E}. A \leqslant_F H$ ²¹ (**problema Aduo**)

iii) H è \leqslant_F -completo per \mathcal{E} se H è \leqslant_F -arduo per \mathcal{E} e $H \in \mathcal{E}$

\Rightarrow A e B stanno quando se si riducono l'uno all'altro.



Tutti i problemi $\in \mathcal{E}$ si riducono a C e $C \in \mathcal{E}$

Tutti i problemi $\in \mathcal{E}_1$ riducibili ad H , ma $H \notin \mathcal{E}$

Il disegno di sopra esemplifica quanto scritto. Il problema C è completo per \mathcal{E} e a esso si riducono sia B e D di \mathcal{D} , sia A e E di \mathcal{E} ; non tutte le riduzioni sono state disegnate (come frecce), tuttavia si noti che $g = f \circ k$ e allo stesso modo si compongono tutte le frecce disegnate o meno; infine H è un problema arduo per \mathcal{E} , ma non completo: tutti i problemi di \mathcal{E} si riducono ad H , ma $H \notin \mathcal{E}$.

Se un problema è completo per una classe \mathcal{E} e appartiene ad una sottoclasse \mathcal{D} , allora le due classi coincidono.

Proprietà: Se \leq_F classifica D ed E , $D \subseteq E$ e C è completo per E , allora

Dim:

Sia $C \in D$ e $A \in E$. Per completezza $A \leq_E C$ e $A \in D$ per la condizione

(iii) di \leq_F che classifica D ed E . Quindi $E \subseteq D$ e le ten.

Proprietà: Se A è completo per \mathcal{E} , $A \leq_F B$, e $B \in \mathcal{E}$, allora B è completo per \mathcal{E} .

Dimostrazione. $\forall D \in \mathcal{E}$, $D \leqslant_F A$ per completezza, ma \leqslant_F -classifica \mathcal{D} ed \mathcal{E} e allora $D \leqslant_F A$ e $A \leqslant_F B$ implicano $D \leqslant_F B$ e quindi B è arduo e, poiché appartiene a \mathcal{E} , è completo. \square

Un problema completo per Σ gioca un ruolo rilevantissimo, in quanto "rappresenta le difficoltà" massime dei problemi in Σ . Infatti, è facile vedere che il grado di un problema A completo per Σ è il grado max di Σ in \leq_F .

Proviamo valgono le seguenti affermazioni:

- se $B \leq_F A$ allora B al più ha il (o meglio al più appartiene al) grado di A , cioè è più facile o altrettanto difficile di A ;
- se $A \leq_F B$ allora B ha almeno il grado di A , cioè è di difficoltà maggiore o uguale a quella di A .

$$\Rightarrow \text{grado}(B) \leq \text{grado}(A)$$

$$\Rightarrow \text{grado}(B) \geq \text{grado}(A)$$

Classificare R ed $R.E$

Definiamo un insieme $R.E.C = \{ \varphi_x \mid \text{dom}(\varphi_x) = \mathbb{N} \}$ che rappresenta la classe delle funz. calcolabili totali.

Def. A è riducibile a B ($A \leq_{\text{rec}} B$) $\Leftrightarrow \exists f: \mathbb{N} \rightarrow \mathbb{N}$ calc. tot t.c $x \in A \Leftrightarrow f(x) \in B$

Vediamo che queste relazioni di riduzione conservano la R ed $R.E$.

Teorema: la relazione di riduzione \leq_{rec} classifica R ed $R.E$

Dimostrazione. Sappiamo già che $R \subseteq R.E$ grazie alla proprietà 1.10.3(i). Possiamo allora usare il lemma 1.10.11 per dimostrare la tesi. Facciamo allora vedere che tutte le ipotesi del lemma sono soddisfatte:

- | | | |
|---|---------------------------------------|--|
| i) $\text{id} \in F$ | $(F \text{ ha identità})$ | i) Facile, dalla definizione di funzione μ -ricorsiva. |
| ii) $f, g \in F \Rightarrow f \circ g \in F$ | $(F \text{ chiusa per composizione})$ | ii) Ovvio perché la composizione conserva la totalità. |
| iii) $f \in F, B \in \mathcal{D} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{D}$ | | iii) La funzione caratteristica di $\{x \mid f(x) \in B\}$ è $\chi_B \circ f$, che è calcolabile totale perché f e χ_B sono entrambe calcolabili totali. |
| iv) $f \in F, B \in \mathcal{E} \Rightarrow \{x \mid f(x) \in B\} \in \mathcal{E}$ | | iv) Analoga al punto precedente, con la semi-caratteristica di B . □ |

Il fatto che \leq_{rec} classifichi R e $R.E$ può essere visto come la capacità che hanno le funz. calc. totali di separare i problemi ricorribili da quelli ricorribilmente enumerabili: ciò avviene in base al tempo necessario per decidere un problema. Se il problema è ricorrioso allora ovunque le

Risposta in tempo finito, altrimenti il tempo necessario è infinito.

Quelche basta trovare un problema che sia $\leq_{\text{rec-completo}}$ per R per poter vedere quali problemi sono decidibili e quali no.

Ancora più interessante è trovare un problema $\leq_{\text{rec-completo}}$ per R.E.: sappiamo

quali sono i problemi al più semplici decidibili e quali nemmeno semidecidibili.

Infatti basta ridurre il probl. da studiare a quello completo e sappiamo che è ricorsivamente enumerabile oppure ridurre il problema completo a quello da studiare e sappiamo che quest'ultimo, alla meglio è ricorsivamente enumerabile. Infatti, come notato nella digressione:

- Se $A \leq_{\text{rec}} B$ e B è ricorsivamente enumerabile ($B \in \text{RE}$), allora A è ricorsivamente enumerabile (e forse anche ricorsivo).
- Se $A \leq_{\text{rec}} B$ e A non è ricorsivamente enumerabile ($A \notin \text{RE}$), allora B non è ricorsivamente enumerabile (e meno che meno ricorsivo).

Quelche se A è ricorsivo il fatto che uno lo riduce a B non ci consente di dedurre alcunché sulla natura di B , il quale potrebbe essere n.o r.e. o nemmeno r.e. Analogamente nel caso di A r.e.

$\bar{K} \leq_{\text{rec}} K$? No, perché significherebbe che \bar{K} è al massimo R.E. $\Rightarrow K$ e \bar{K} sono ricorsivi

$K \leq_{\text{rec}} \bar{K}$? No, perché se $K \leq_{\text{rec}} \bar{K} \Rightarrow \bar{K} \leq_{\text{rec}} K$ che abbiano dim. non vero.

Teorema: K è RE-completo, ovvero $\leq_{\text{rec-completo}}$ per RE

Dim: Dimostrazione. Dobbiamo dimostrare che se $A \in \text{RE}$ allora $A \leq_{\text{rec}} K$. Per definizione A è il dominio di una funzione calcolabile ψ , cioè $A = \{x \mid \psi(x) \downarrow\}$. A partire da ψ si definisce una funzione ψ' a due variabili di cui ignora la seconda, cioè sia $\psi'(x, y) = \psi(x)$, che è a sua volta una funzione calcolabile e quindi avrà un indice, diciamo i ; in simboli $\psi' = \varphi_i$. Allora, per il teorema del parametro $\psi'(x, y) = \varphi_i(x, y) = \varphi_{s(i, x)}(y)$, con s calcolabile, iniettiva e totale. Posso riscrivere la definizione di A come segue:

$$\begin{aligned} A &= \{x \mid \psi(x) \downarrow\} \\ &= \{x \mid \psi'(x, y) \downarrow\} \\ &= \{x \mid \varphi_i(x, y) \downarrow\} \\ &= \{x \mid \varphi_{s(i, x)}(y) \downarrow\} \text{ per il teorema del parametro} \\ &= \{x \mid \varphi_{s(i, x)}(s(i, x)) \downarrow\} \text{ ponendo } y = s(i, x) \\ &= \{x \mid s(i, x) \in K\} \end{aligned}$$

quindi $x \in A$ se e solamente se $f(x) \in K$, con $f(x) = \lambda x. s(i, x)$, che è totale, calcolabile e iniettiva perché la $s(i, x)$ lo è (si veda la nota a piè di pagina 56: prendiamo i tale che $\psi'(x, y) = \varphi_i(x, y) = \varphi_{s(i, x)}(y)$ da cui $f = \lambda x. s(i, x)$). \square

Questo vuol dire che ogni problema \Rightarrow in RE è difficile al massimo

Come K ,

cioè ogni problema $A \in \text{RE}$

e t.c. $A \leq_{\text{rec}} K$

Esercizio di riduzione: ($U \leq_{rec} TOT$)

Vediamo il seguente esercizio, affatto banale, con cui si mostra che l'insieme degli indici delle funzioni calcolabili totali, cioè rec , è indecidibile. Dimostriamo che

$$K = \{x \mid \varphi_x(x) \downarrow\} \leq_{rec} \{x \mid \varphi_x \in rec\} = TOT^{22}$$

Definiamo ora questa funzione: $\psi(x, y) = \begin{cases} 1 & \text{se } x \in K \\ \text{indef} & \text{altrimenti} \end{cases}$

La nostra ψ è calcolabile parziale: il programma P_x calcola $\varphi_x(x)$ e se è quando questa converge, restituisce 1 per ogni y . Per il teorema $s-m-n$ esiste f calcolabile totale iniettiva tale che $\varphi_{f(x)}(y) = \psi(x, y)$. (Per costruire la f si ricordi la nota a pié di pagina 56: si scelga i tale che $\psi(x, y) = \varphi_i(x, y) = \varphi_{s(i,x)}(y)$ da cui $f = \lambda x. s(i, x)$). Adesso

$$x \in K \Rightarrow \varphi_{f(x)} = \psi(x, y) = 1 \Rightarrow \varphi_{f(x)} \text{ totale} \Rightarrow f(x) \in TOT$$

$$x \notin K \Rightarrow \varphi_{f(x)} = 1 \text{ indefinito} \Rightarrow \varphi_{f(x)} \text{ non è totale} \Rightarrow f(x) \notin TOT.$$

Di conseguenza, TOT è ben che ci vada ricorsivamente enumerabile.

$$TOT = \{x \mid \varphi_x \in REC\}$$

↑
Sintassi

Lemma: A è un insieme di indici che rappresentano funzioni se e solo se

$$\forall x, y. \text{ se } x \in A \text{ e } P_x = \varphi_y \Rightarrow y \in A$$

Adesso studiamo quelli: insiem di indici A che neppure le funzioni e tali per cui

$$U \leq_{rec} A \text{ oppure } U \leq_{rec} \bar{A}$$

Teorema: Sia A un insieme di indici che neppure rappresentano le funzioni tali che

$$\emptyset \neq A \neq \mathbb{N}$$

>Allora $U \leq_{rec} A$ oppure (ponono essere vere entrambe) $U \leq_{rec} \bar{A}$

Non rappresentano funzioni

Dim:

Dimostrazione. Prendi i_0 tale che $\varphi_{i_0}(y)$ sia ovunque indefinita. Supponiamo che $i_0 \in \bar{A}$ e dimostriamo $K \leq_{rec} A$ (se $i_0 \in A$ si procede in modo simmetrico). Poichè $A \neq \emptyset$ scegli $i_1 \in A$. Hai $\varphi_{i_0} \neq \varphi_{i_1}$ perché A è un insieme di indici che rappresentano le funzioni. Definiamo adesso la seguente funzione che è calcolabile:

$$\psi(x, y) = \varphi_{f(x)}(y) = \begin{cases} \varphi_{i_1}(y) & \text{se } x \in K \\ \text{indefinita} = \varphi_{i_0}(y) & \text{altrimenti} \end{cases}$$

dove, usando il teorema $s-m-n$, abbiamo determinato la f funzione calcolabile totale iniettiva (come suggerito nella nota a pié di pagina 56: sia i tale che $\psi(x, y) = \varphi_i(x, y) = \varphi_{s(i,x)}(y)$, allora si pone $f = \lambda x. s(i, x)$). Allora

$$x \in K \text{ implica } \varphi_{f(x)} = \varphi_{i_1} \text{ implica } f(x) \in A$$

perchè $i_1 \in A$ e A è un insieme di indici che rappresentano le funzioni e quindi anche $f(x) \in A$. Viceversa, dato che $i_0 \in \bar{A}$,

$$x \notin K \text{ implica } \varphi_{f(x)} = \varphi_{i_0} \text{ implica } f(x) \in \bar{A} \text{ (implica } f(x) \notin A\text{).}$$

(*)

Oss: Esistono insiem B (per esempio TOT) t.c. $U \leq_{rec} B$ e anche $U \leq_{rec} \bar{B}$,

Cioè esistono f e g calcolabili totali iniettive t.c.

$$x \in K \Leftrightarrow f(x) \in B \text{ e } x \in U \Leftrightarrow g(x) \in \bar{B}$$

Teorema di Rice: Sia A una classe di funzioni calcolabili.

L'insieme $A = \{n \mid \varphi_n \in A\}$ è non vuoto $\Leftrightarrow A \neq \emptyset$

Oppure A è la classe di tutte le funzioni calcolabili.

D.m.:

Dimostrazione. Si noti che A è un insieme di *indici* mentre \mathcal{A} è una classe di *funzioni* (anche se la lettera è la stessa, il carattere è *diverso!* a indicare che i primi sono *sintassi*, mentre i secondi sono *semantica*).

La dimostrazione è immediata per i casi banali, cioè quando $A = \emptyset$ e quando \mathcal{A} è la classe di *tutte* le funzioni calcolabili.

Negli altri casi, basta applicare il teorema 1.10.19, poiché A è un insieme di indici che rappresentano le funzioni, il quale non è vuoto, perché \mathcal{A} contiene almeno una funzione, né coincide con \mathbb{N} , perché \mathcal{A} non contiene tutte le funzioni calcolabili. \square

Teorema: K non è un insieme di indici che rappr. funzioni.

D.m.:

Non deve valere $\forall x \in K \quad \phi_x = \phi_y \Rightarrow y \in K$

Definiamo $\psi(x, y) = \begin{cases} 42 & \text{se } x = y \\ \text{indefinita} & \text{altrimenti} \end{cases}$

Per C-T e per il teorema del parametro ho $\psi(x, y) = \phi_i(x, y) = \phi_{s(i,x)}(y) = \phi_{f(x)}(y)$

\exists un punto fisso per $f(x) \Rightarrow \phi_{f(x)}(y) = \phi_x(y)$

Per $\psi(p, p) = \phi_p(p) = 42 \Rightarrow p \in K$

Per il padding lemma, \exists un indice $z \neq p \mid \phi_z = \phi_p$

Però $\psi(p, q) = \phi_z(p) = \phi_p(z)$ è indefinita. Quindi $q \notin K$ e K non è un i.i.r.f.

Considerazioni:

Questo risultato si ripercuote sulle proprietà che si possono dimostrare sui programmi: ogni metodo di prova si scontra con il problema della fermata. Ci sono però varie tecniche per aggirare il problema, ad esempio l'**analisi statica** del codice, dove si analizza il **testo** del programma, per raccogliere informazioni su come vengono usati durante l'esecuzione i vari oggetti (variabili, chiamate...), per esempio se vengono rispettati i tipi, se si inizializzano...

Si ha successo, con questo tipo di analisi, perché il **programma è approssimato in modo sicuro**: ciò che viene predetto è una sovra-approssimazione di ciò che succederà davvero. Per esempio, può succedere di dire che tra i valori assegnati ad una variabile **int** c'è una **String** senza che ciò accada a runtime, ma non capiterà mai di dire che tutti i valori assegnati sono **int** se a runtime a tale variabile viene assegnata una **String**.

A questa famiglia appartengono vari strumenti, spesso incorporati nei compilatori: type-checker, analizzatori data-flow e control-flow...

Applicazione Un'applicazione del teorema di Rice è che $K_1 = \{x \mid \text{dom}(\phi_x) \neq \emptyset\}$, cioè l'insieme degli indici delle funzioni definite in almeno un punto **non è ricorsivo**, sebbene sia ricorsivamente enumerabile.

Inoltre K , K_0 e K_1 si riducono l'un l'altro.

Fine della Calcolabilità

Teoria della complessità

Si occupa di come poniamo calcolare.

Problemi decidibili: Conosciamo una funzione f calcolabile totale che ci permette di dire se un elemento appartiene o meno a un certo insieme ($x \in I$).

Cerchiamo una f che stima il tempo necessario per determinare se $x \in I$

↓
calcolo tot.

risorse

cioè cerchiamo $f(|x|) \geq$ tempo / spazio di calcolo necessario a decidere $x \in I$

Vogliamo f come la minima
delle funzioni che maggiorano la
quantità di risorse per calcolare $x \in I$

↓
taglia = numero di ingredi
di una funzione

↑ Maggiore numero di risorse
utilizzate

Vogliamo la valutazione $f(|x|)$ al CASO PERTINO $\forall x$

Def:

Problemi complessi = Problemi che appartengono ad una classe e che sono i più difficili da risolvere con quelle limitazioni spazio / temporali.

OSS: f determina una classe di complessità = insieme di tutti quei problemi nei quali f maggiora la stima tempo / spazio

↓

le classi di complessità sono invarianti da : $\begin{cases} \text{tipo di modello di calcolo} \\ \text{rappresentazione dei dati} \end{cases}$

Avere delle classi di complessità ci permette anche di definire una gerarchia fra esse



$\log\text{SPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE} = \text{NSPACE} \subseteq \text{R} \subseteq \text{R.E}$

$\log \text{space}$ =insieme delle funzioni che richiedono spazio logaritmico rispetto alla taglia

P = Problemi risolvibili in tempo polinomiale deterministico

NP = Problemi risolvibili in tempo polinomiale non deterministico

PSPACE = Problemi risolvibili in spazio polinomiale deterministico

NSPACE = Problemi risolvibili in spazio polinomiale non deterministico

MDT a k -nastri

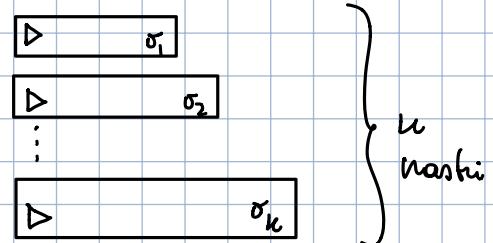
$$M = (Q, \Sigma, \delta, q_0)$$

$$f: Q \times \Sigma^k \rightarrow Q \cup \{ \text{Si}, \text{No} \} \times (\Sigma \times \{ L, R, - \})^k$$

Cioè

$$\delta(q, \sigma_1, \dots, \sigma_k) = (q', (\sigma'_1, \Delta_1), (\sigma'_2, \Delta_2), \dots, (\sigma'_k, \Delta_k))$$

dividiamo lo stato h



$$\gamma = (q, u_1 \sigma_1 v_1, u_2 \sigma_2 v_2, \dots, u_k \sigma_k v_k)$$

config.

$\gamma \rightarrow \gamma'$ è come prima me devo far

muovere k nastri

$$(q, w_1, \dots, w_k) \xrightarrow{n} (q', w'_1, \dots, w'_k)$$

Come prima, $M(x) = (q_0, \overbrace{\Delta x, \Delta \dots \Delta}^{k-1})$

Complexità in tempo deterministico

M richiede tempo (deterministico) t per risolvere il caso $x \in I$

$$\Leftrightarrow M(x) \xrightarrow{t} (\text{Si}/\text{No}, w)$$

Intendo che determinare con precisione il numero di passi potrebbe risultare difficile e costoso, ci si accontenta di una stima superiore $f(|x|)$ con $|x| = \text{taglia}$.

DEF: M decide I in tempo (deterministico) f se $\forall x \in I$ il tempo t richiesto

da M per risolvere x è tale che $t \leq f(|x|)$

La classe di completezza in tempo (d) f è

$$\text{TIME}(f) = \{ I \mid \exists M \text{ che decide } I \text{ in tempo (deterministico) } f \}$$

Termino (riduzione dei nastri)

Data M a k -nastri che decide I in tempo (f)

$\exists M'$ a 1 nastro che decide I in tempo (f) in $O(f^2)$

Dim:

$$\mathcal{F} = (q, \triangleright w_1 \sigma_1 u_1, \triangleright w_2 \sigma_2 u_2, \dots, \triangleright w_k \sigma_k u_k)$$

Config generale macchina M



$$(q, \triangleright \triangleright' w_1 \bar{\sigma}_1 u_1 \triangleleft' \triangleright' w_2 \bar{\sigma}_2 u_2 \triangleleft' \dots w_k \bar{\sigma}_k u_k \triangleleft') \text{ Config generale macchina } M'$$

cioè racchiudiamo ciascun nastro $w_i \sigma_i u_i$ tra due nuove parentesi \triangleright' e \triangleleft' e usiamo $\#\Sigma$ nuovi simboli $\bar{\sigma}_i$ per ricordarci di qual era la posizione della testina sull' i -esimo nastro.

Per cominciare, dobbiamo partire dalla config. iniziale di M

$$(q_0, \triangleright x, \triangleright, -, \triangleright) \xrightarrow{O(n)} (q'_0, \triangleright \triangleright' x \triangleleft' (\triangleright' \triangleleft')^{k-1}) \text{, per qualche } q.$$

config iniziale di M CONFIG INIZIALE M'

Adesso osserviamo che una macchina non può toccare un numero di caselle maggiore del numero dei passi che compie.

Quindi la lunghezza totale del nastro $\leq K = k \times ((f(n)+2) + 1)$

↓
dovuto alle parentesi $\triangleright' \triangleleft'$ ↓
dovuto al simbolo \triangleright

M' per simulare una mossa di M tiene il dato di ingresso da rx a dx e viceversa, 2 volte:

1) Determina i simboli correnti (scorrere il nastro) (Prima volta)

$4k = 2 \text{ volte } rx$
 $\text{e } dx$

2) Scrivere i nuovi simboli $= 2K$ (Seconda volta)

$k = \text{andare a fine nastro}$

$2k = \text{spostare simboli verso dx}$

Poiché le altre costanti sono irrelevanti, poniamo concludere che ogni passo di computazione costa $O(f(n))$. Evidendo che il numero di passi è $O(f(n))$, M' decide I in $O(f^2)$

Teorema di accelerazione lineare

Se $I \in \text{Time}(f) \Rightarrow \forall \epsilon \in \mathbb{R}^+ \text{ si ha } I \in \text{Time}\left(\epsilon \cdot f(n) + n + 2\right)$

Questo teorema ci dice che posso accelerare l'algoritmo linearmente quanto voglio.

Quindi se $f = c \cdot n$ posso eliminare c e mettere $\epsilon = \frac{1}{c}$

Dim:

Prendo m simboli $[\sigma_1, \sigma_2, \dots, \sigma_m]$ e li considero come unico carattere
Quindi posso codificare gli stati in una tripla $[q, \sigma_1 \sigma_2 \dots \sigma_m, u]$ con

u che indica la posizione del cursore

$[q, \sigma_1 \sigma_2 \dots \sigma_m, u] \in \Sigma'$ cioè i simboli della nuova macchina

M' in 6 passi simula m passi di M , questo perché nei primi 6 passi M' va a rx , poi a dx , poi ancora a dx e infine infine al suo carattere corrente
in modo da raccolgono i simboli che M può visitare con m mosse.

Infatti M con m mosse può spostarsi nella stringa di rx , dx o stare nella corrente.

Quindi M' simula le m mosse di M muovendosi a rx/dx del blocco corrente
ma può visitare solo O quello rx O quello dx , non entrambi perché si sposta di 1 singolo blocco.
(Può andare nei blocchi adiacenti in un singolo passo di computazione).

M' farà $n+2$ passi per condurre l'input $+ \left\lceil 6 \cdot \left(\frac{f(n)}{m} \right) \right\rceil$ passi

Quindi posso prendere come $\left\lceil m = \frac{6}{\epsilon} \right\rceil$

OSS: Si ottiene quindi che P (classe dei problemi decidibili in tempo polinomiale deterministico)

è:

$$P = \bigcup_{k \geq 1} \text{Time}(n^k)$$

MDT 1/0 a k-nastri

Se $f(q, \sigma_1, \dots, \sigma_k) = (q', (\sigma'_1, D_1), \dots, (\sigma'_k, D_k))$

- $\sigma'_1 = \sigma_1 \Rightarrow$ Il primo nastro contiene l'input (solo lettura)
- $D_k = R$ oppure ($\text{se } D_k = - \Rightarrow \sigma'_k = \sigma_k$) \rightarrow Ultimo nastro contiene l'output (solo scrittura)
- $\sigma_1 = *$ $\Rightarrow D_1 \in \{L, -\}$ \rightarrow Quando finisce di leggere il nastro d'inizio posso solo tornare indietro o stare fermo.
(leggi solo un bianco)

Teorema

$\forall M$ a k-nastri che decide I in tempo (d) \Leftrightarrow

$\exists M'$ a $k+2$ nastri che decide I in tempo $c \cdot d$

Dim.

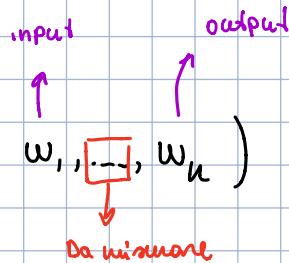
Nel primo nastro ci ha solo l'inizio, e nell'ultimo c'è la fine
nei k nastri interni svolge le operazioni

Complexità in SPAZIO

M di tipo 1/0 richiede il seguente spazio

$$\sum_{i=2}^{k-1} w_i$$

$$M(x) \xrightarrow{*} (\text{Si/No}, w_1, \boxed{w_k}, w_k)$$



M decide in spazio(d) $\Leftrightarrow \forall x. M$ richiede spazio $\leq f(|x|)$

$\text{SPACE}(f) = \{ I \mid \exists M \text{ che decide } I \text{ in spazio } f \}$



CLASSE

di complessità

TEOREMA (Riduzione lineare dello spazio)

$I \in \text{SPACE}(f) \rightarrow \forall \epsilon \in \mathbb{R}, I \in \text{SPACE}(2 + \epsilon \cdot f(n))$

$$\text{PSPACE} = \bigcup_{u \geq 1} \text{SPACE}(n^u)$$

$$\text{LOGSPACE} = \bigcup_{k \geq 1} \text{SPACE}(k \cdot \log(n))$$

SPazi LOGARITMICI

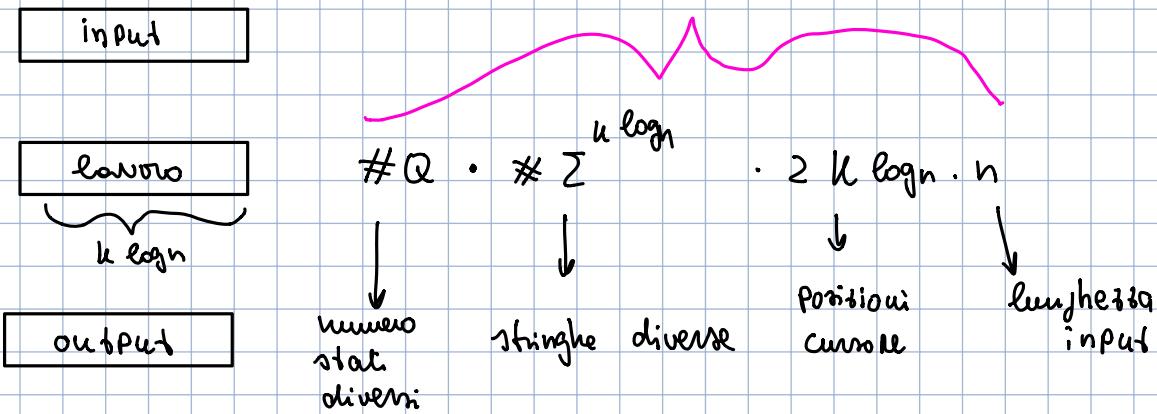
teorema

$\text{LOGSPACE} \subseteq \text{P}$ ma non sappiamo se $\text{LOGSPACE} = \text{P}$ o $\text{LOGSPACE} \neq \text{P}$

Dim:

Macchina a 3 nastri:

numero di config diverse della macchina



Devo trovare un p.t.c.

$$n^p \geq \#Q \cdot * \sum^{k \log n} \cdot 2 k \log n \cdot n$$

$$\log n^p \geq \log(\#Q) \cdot \log(* \sum^{k \log n}) \cdot 2 k \log(\log n) \cdot \log(n)$$

$$P \cdot \log n \geq \log(\#Q) + K \log(n) \log(\#\Sigma) + 2U \log(\log n) + \log(n)$$

eliminiamo $\log(\#Q)$ perché è una costante e $\log(\log n)$ perché molto piccolo.

poi semplifico per $\log n$

$$P \geq \#\Sigma + 1$$

Ottieniamo che basta usare $P \geq \#\Sigma$

Quindi scopriamo che:

OSS: lo spazio limita il tempo. Questo perché se fanno troppe volte sopra la stessa configurazione vado in ciclo.

Determinismo e non Determinismo

Un caso deterministico genera completamente lo spazio delle potentiali soluzioni (Brute Force) → Cerco la soluzione tra quelle possibili

In caso non deterministico ho un albero di scelte non deterministiche cioè vengono scelte con una determinata possibilità, ad esempio con un dado (Guess and Try) → Viene data una soluzione e controllo se funziona

OSS: Un entrambi: can controllo che sia una soluzione

NDT non deterministiche

$N = (Q, \Sigma, \Delta, q_0)$ con le sole condizioni delle NDT deterministiche
↓
Relazione (non più una funzione) di transizione

$$\Delta \subseteq (Q \times \Sigma) \times (Q \times \{S, No\} \times \Sigma \times \{L, R, -\})$$

$(q, \omega \sigma \nu)$

configurazione standard

$(q_0, \Delta x) \xrightarrow{*} (q, \omega)$

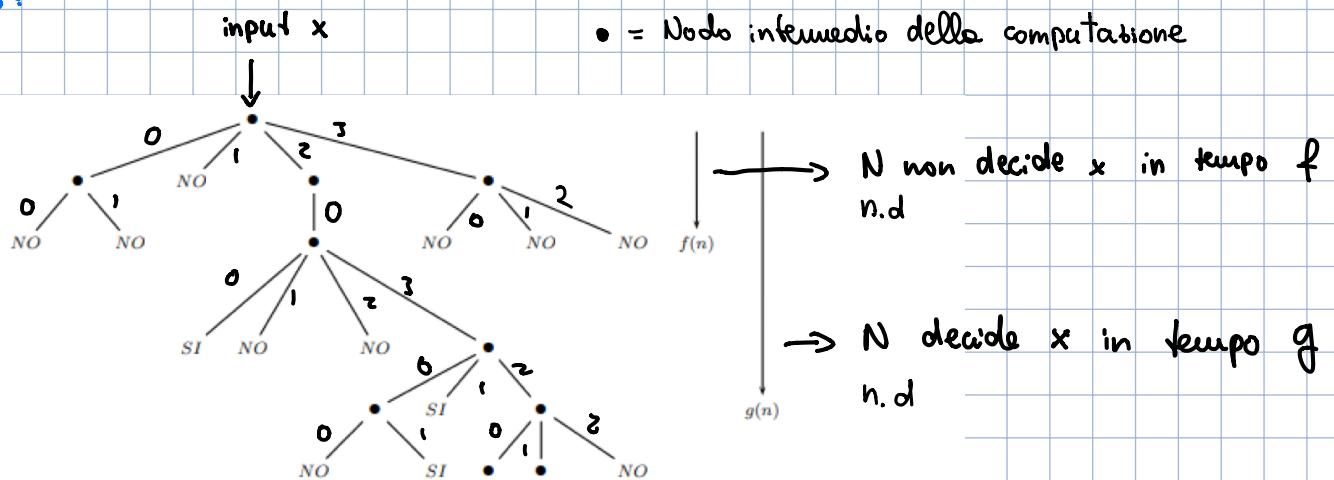
computazione Standard, semplicemente viene scelta in modo non deterministico

OSS:

N decide $I \Leftrightarrow \forall x \in I \exists N. N(x) \xrightarrow{*} (S_i, \omega, \dots \omega_k)$

Cioè esiste almeno una computazione che dà la soluzione

Esempio:



Numerando gli archi di decisione identifico una computazione tramite la successione di numeri. Data una certa σ ho tante quintuple in Δ , le enumero e le ordino in qualche modo. Prendo la n_1 scelta dallo stato iniziale, la n_2 scelta dallo stato risultante...

Nell'esempio, una scelta Si è identificata dalla computazione $(2, 0, 0)$, un'altra da $(2, 0, 3, 1)$.

Tempo e Spazio non Deterministici

Tempo

N decide I in tempo non deterministico $f(I \times I)$

\Leftrightarrow a) N decide I

b) $\forall x \in I \exists t | N(x) \xrightarrow{N}^t (S_i, \omega) \text{ e } t \leq f(I \times I)$

in modo
non deterministico

$NTIME(f) = \{ I \mid \exists N \text{ che decide } I \text{ in tempo n.d. } f \}$

$$NP = \bigcup_{n \geq 1} N\text{Time}(n^u)$$

Teorema: Se $I \in N\text{Time}(f) \Rightarrow I \in \text{Time}(c^f)$ con $c > 1$ che dipende soltanto dalla macchina in $\text{Time}(c^f)$, cioè ho una perdita esponentiale.

$$\text{cioè: } N\text{Time}(f(n)) \subseteq \text{Time}(c^{f(n)})$$

Dimi:

$$\text{grado}(q, \sigma) = \# \{ (q', \sigma', \Delta') \mid ((q, \sigma), (q', \sigma', \Delta')) \in \Delta \}$$

$$\text{Diciamo poi che } d = \max \left\{ \underset{\downarrow}{\text{grado}}(q, \sigma) \mid q \in Q, \sigma \in \Sigma \right\}$$

Numero
di livelli:
albero

Definito ciò:

- 1) Ordiniamo Δ ottenendo così che ogni computazione è una seq. di scelte lunga t , indicabili con una sequenza di numeri naturali minori di t nell'intervallo $[0 \dots d-1]$
 - 2) Costruisco una nuova macchina M che simula N per ogni computazione (c_0, c_1, \dots, c_t)
 - a) Nasco di lavoro in più che contiene le stecche di scelte computazioni già fatte
 - b) Ripercorre le computazioni e verifica le varie computazioni:
 - 1) se termina, bene, ho finito
 - 2) se non termina, prende la prossima computazione (c_{t+1}) in base al

es: computazione 00, 01, 02 --, 10, 11, 12, --, 20, 21 -- ecc.
- In poche parole "spazzolo" i livelli dell'albero e mi tieno sul nastro di lavoro quelle già fatte.

Qual'è il numero massimo di computazioni sul nastro di lavoro? = $d^{f(n)}$

SPAZIO

N decide I in spazio non deterministico $f(|x|) \Leftrightarrow$

a) N decide I

b) $\forall x \in I$

$$\exists w_1, \dots, w_n \text{ t.c } N(x) \xrightarrow[N]{*} (s_i, w_1, \dots, w_n) \text{ e } \sum_{2 \leq i \leq n} |w_i| \leq f(|x|)$$

$$NSPACE(f) = \left\{ I \mid \exists N \text{ decide } I \text{ in spazio n.d. } f \right\}$$

$$NPSPACE = \bigcup_{n \geq 1} NSPACE(n^k)$$

Teorema di Savitch

$$NSPACE = PSPACE$$

Funzioni di misura appropriate

$f : \mathbb{N} \rightarrow \mathbb{N}$ totale è appropriata \Leftrightarrow

1) f è monotona crescente

2) $\exists M. \forall x \in \Sigma^* M(x) \downarrow$ con risultato $\diamond, \diamond \notin \Sigma$

carattere lungo tanti caratteri
speciale quanto $f(|x|)$

Condizioni:

$M(x)$ converge $\xrightarrow{\quad} a) \text{In tempo } O(f(|x|) + |x|)$

$\xrightarrow{\quad} b) \text{In spazio } O(f(|x|))$

Esempi: $n^k, \lfloor \log n \rfloor, n!, n^n \dots$

Inoltre se f, g appropriate, lo sono anche

$f(g), f^g, f \cdot g$ ecc...

Teorema di Gerarchia

Data f appropriata:

- Time($f(n)$) \subseteq Time($f(2^{n+1})^3$)
- Time($f(n)$) \subseteq SPACE($f(n) \cdot \log(f(n))$)

Esempio:

$$\{x \mid \varphi_x(x) \downarrow \text{entro } f(|x|) \text{ passi}\} \begin{cases} \notin \text{Time}(f) \\ \in \text{time}(f^3) \end{cases}$$

Teorema:

$$P \subseteq EXP = \bigcup_{k \geq 1} \text{Time}(2^{n^k})$$

Dim:

$$P \subseteq \text{Time}(2^n) \subseteq \text{Time}(f(2^{(2^{n+1})^3})) \subseteq \bigcup_{k \geq 1} \text{Time}(2^{n^k})$$

A riportare:

- SPACE(f) \subseteq NSPACE(f)
- TIME(f) \subseteq NTIME(f)
- NSPACE(f) \subseteq TIME($K^{\log n + f(n)}$)
- LogSPACE \subseteq P
- LogSPACE \subseteq PSPACE
- PSPACE = NSPACE
- NP \subseteq EXP

Teorema

$\forall g$ calcolabile $\exists I$ problema | $I \in \text{Time}(f(n)) \wedge I \notin \text{Time}(g(n))$, con $f > g$ quasi ovunque

Questo teorema ci dice che la gerarchia non è sup. limitata, quindi esistono sempre problemi più difficili di altri.

Teorema di accelerazione [Blum]

$\forall h$ calc. tot. $\exists I : \forall M$ che decide I in tempo f

$\exists M'$ che decide I in tempo $f' \mid f'(n) > h(f(n))$

quasi ovunque

Ci dice che puoi accelerare f' quanto vuoi; cioè non esiste mai un algoritmo ottimo per risolvere un problema.

Teorema della lacuna

$\exists f$ calc. tot. : $\text{Time}(f) = \text{Time}(2^f)$

Ci dice che c'è un insieme di programmi che sono altrettanto veloci su una macchina lenta che su una macchina nuova.

Cioè che tu abbia f risorse o 2^f risorse non fa differenza.

OSS: Sembra che ci sia una contrapposizione con il teo. di gerarchia, ma così non è perché non c'è richiesta una f appropriata nel teorema della lacuna. Questo ci mostra che non è possibile fare a meno delle funzioni di misura appropriate!

Teori di Cook - Karp

La teori di Cook - Karp ci dice che :

$\text{P} = \text{Problemi Trattabili}$

$\text{NP} = \text{Problemi Intrattabili}$

Inoltre P e NP sono clami robusti cioè resistono al cambio di modello e alle rapp. dei dati

Riduzione efficiente

Definizione:

$I \leq_{\text{logspace}} J$ $\Leftrightarrow (x \in I \Leftrightarrow f(x) \in J \text{ con } f \in \text{logspace})$

Teorema

\leq_{LS} classifica P e NP

Dim:

- 1) $I_d \in \text{logSPACE} \Rightarrow$ banale, prendo una macchina a k-nasti che prende il nastro di ingresso e lo risolve in uscita
- 2) $f, g \in \text{logspace} \rightarrow g \circ f \in \text{logSPACE}$

Si fa partire M' in modo da richiedere a M i dati di lavoro e usarli all'occorrenza sostituendoli sempre su una sola casella.

- 3) Se $J \in P$ e $I \leq_{\text{LS}} J \Rightarrow I \in P$

Se I si riduce per un logaritmo a J e J lo risolve in tempo logaritmico, allora anche I lo risolve in tempo logaritmico

- 4) Se $J \in NP$ e $I \leq_{\text{LS}} J \Rightarrow I \in NP$

Analogo al punto precedente

Espresioni Booleane

$$B = tt \mid ff \mid x \mid \neg B \mid B_1 \vee B_2 \mid B_1 \wedge B_2$$

$$\nu: x' \rightarrow \{ tt, ff \} \text{ con } x' \in h(x) \}$$

↓

è un anegamento booleano

B chiuso cioè $\nu(B) = B'$, B non ha var. libere

Soddisfacibilità: Un anegamento booleano V che rende vera un'espressione booleana

- $[[tt]]_V = tt$
- $[[ff]]_V = ff$
- $[[\neg B]]_V = \text{not } [[B]]_V$
- $[[B_1 \wedge B_2]]_V = [[B_1]]_V \text{ and } [[B_2]]_V$
- $[[B_1 \vee B_2]]_V = [[B_1]]_V \text{ or } [[B_2]]_V$

Forma Normale Congiuntiva: $B = \bigwedge_{i=1}^n D_i$ e $D_i = \bigvee_{j=1}^m \ell_j$ con ℓ_j letterali

$\ell \in \{ tt, ff, x, \neg x \}$

Forma normale Disgiuntiva: $B = \bigvee_{i=1}^n D_i$ e $D_i = \bigwedge_{j=1}^m \ell_j$ con ℓ_j letterali

Un'oltre $\forall B \exists B'$ in forma normale . $\forall V [[B]]_V \Leftrightarrow [[B']]_V$

Questo si ottiene avendo, da B con $O(n)$ simboli, un B' con $O(2^n)$ simboli

Esempio:

$$(x_1 \wedge y_1) \vee (\neg x_2 \wedge tt) =$$

$$= (x_1 \vee (\neg x_2 \wedge tt)) \wedge (y_1 \vee (\neg x_2 \wedge tt)) \rightarrow \text{(la dim. dell'espr. cresce)}$$

Problema SAT

Anche chiamato problema della soddisfabilità.

Ci chiediamo se dato B , $\exists V . ([B]_V = \text{tf})$

Data un expr booleana, esiste un assegnamento booleano che la soddisfa?

SAT \in NP

Dim:

Costruisco una NDT non deterministica che verifica se l'assegnamento esiste o meno.

Supponendo n variabili, allora ho n scelte all'inizio, per ognuna delle quali ho due scelte: assegnare tt o ff.

Faccio l'assegnamento per ognuna e ottengo un cammino.

Vedremo invece che la verifica avviene in tempo polinomiale andando a verificare l'assegnamento dato.

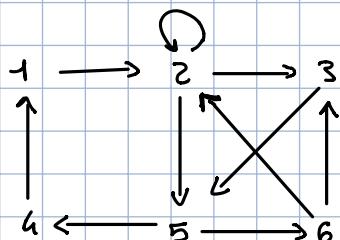
Problema HAM \in NP

Problema del cammino hamiltoniano.

Ci chiediamo se dato un grafo orientato $G = (N, A)$

\exists un cammino che tocca tutti i nodi una e una sola volta

Esempio:



4, 1, 2, 5, 6, 3 = Camino Hamiltoniano

$\text{HAM} \leq_{\text{LS}} \text{SAT}$

Dim:

Devo definire $f \in \text{logspace}$.

G ha un cammino hamiltoniano $\Leftrightarrow \exists V. \underbrace{[f(G)]}_{B = \text{Espressione Booleana}} = tt$

$B = \text{Espressione Booleana}$

Supponiamo che $G = (N, A)$ ha n nodi, allora $B = f(g)$ ha n^2 variabili

$X_{i,j}$: ciascuna rappresentazione che nell'i-erimo posto di un cammino appare il nodo j
 \downarrow
 $1 \leq i, j \leq n$

Sia $\pi: [1, n] \rightarrow [1, n] \mid (\pi(i), \pi(i+1)) \in A$ una permutazione

con $\pi(i) = \text{nodo in posizione } i$

Allora π deve essere una funzione

1) Soggettiva, un nodo non può essere mappato due volte

$$\neg(x_{ij} \wedge x_{kj}) \text{ con } i \neq k \Rightarrow \neg x_{ij} \vee \neg x_{kj} \text{ con } i \neq k$$

\downarrow
Applico De Morgan per

ottenere un disgiunto

2) Totale, ogni nodo deve apparire nel cammino

$$x_{1j} \vee x_{2j} \vee x_{3j} \vee \dots \vee x_{nj}$$

3) Soggettiva, in ogni posizione deve apparire un nodo, se non rimarrebbe uno spazio vuoto

$$x_{i1} \vee x_{i2} \vee \dots \vee x_{in}$$

4) Non posso avere 2 nodi nella stessa posizione

$$\neg(x_{ij} \wedge x_{ik}) \text{ con } j \neq k \Rightarrow \neg x_{ij} \vee \neg x_{ik} \text{ con } j \neq k$$

↓
Applico De Morgan Per
ottenere un disgiunto

Ho ben definito la permutazione, manca da definire $(\pi(i), \pi(i+1)) \in A$

5) $\forall (i, j) \notin A \rightarrow \neg x_{u,i} \vee \neg x_{k+1,j} \quad \forall k. 1 \leq k \leq n-1$

Arco da i a j

che non è nel cammino HAM

Adesso devo dimostrare che se c'è un cammino hamiltoniano allora esiste un assegnamento booleano V che soddisfa.

Cioè $\exists V. [C \not\models (G)] = tt \iff G \text{ è HAM}$

Dimostra i due versi dell' implicazione

1) \Leftarrow

a) $\forall j \exists i \text{ t.c } V(x_{i,j}) = tt$

b) $\forall i \exists j \text{ t.c } V(x_{i,j}) = tt$

c) f individua un cammino (grazie alle prop. 5)

2) \Rightarrow

Se ho un cammino composto da $(\pi(1), \dots, \pi(n))$, defino $V(x_{ij}) = \begin{cases} tt & \text{se } \pi(j) = i \\ ff & \text{se } \pi(j) \neq i \end{cases}$

Con ciò ho definito una permutazione che rispetta che gli archi siano ben rappresentati.

Mi manca da dimostrare che $f \in \text{logspace}$

Costruisco un MDT I/O con $\Sigma = \{ \text{tt}, \text{ff}, \top, \perp, \vee, \wedge, (,) \} \cup \{ 0, 1 \}$
 le variabili sono le coppie (i, j) con i, j rapp. in binario $O(\log n)$
 l'input è la successione di anelli bit richiesti

Operazioni:

1) Scrive n (numero di nodi) sul nastro di lavoro

2) Scrive le clausole da 1-4 sull'output

Usa 3 nastri di lavoro ulteriori per memorizzare i 3 indici

i, j, k che compaiono nelle clausole

3) Infine scrive sul nastro di output la clausola 5

Scorrerai il nastro di input anche questa volta per verificare la clausola.

Perché opera in logspace?

Perché tra n che i, j sono rappresentati in binario quindi
 richiedono $\log n$ spazio.

E neudo che ho 5 nastri, avrò $\sum_{i=2}^5 |w_i| = 5 \log n$ cioè

$f \in \text{SPACE}(5 \cdot \log n)$

Problema della CRICCA

Dato un grafo $G = (N, A)$ ci chiediamo se

$\exists C \subseteq N \text{ t.c. } \forall i, j \in C \text{ con } i, j \in A$

Una cicca C è di ordine k se contiene k nodi

la cicca è un grafo completo \Rightarrow ogni nodo è collegato agli altri transit

$SAT \leq_{LS} CRICCA$

Un anello.

Dim:

$\exists V . [[B]]_V = tt \Leftrightarrow f(B) = G(N, A)$ ha n-cicca

con $B = \bigwedge_{i=1}^n C_i$

Ottieniamo che:

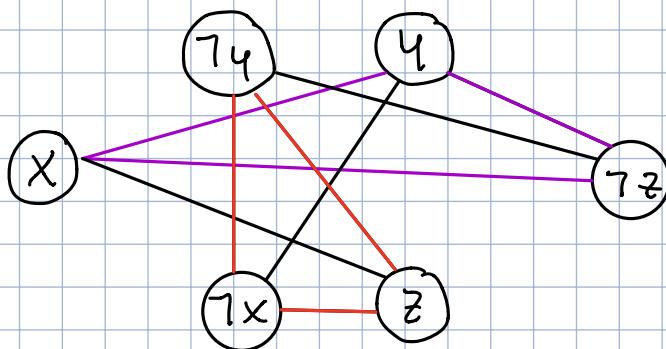
N è l'insieme delle occurrenze dei letterali in B

$$A = \{ (i, j) \mid i \in C_k \Rightarrow (j \notin C_k \wedge i \neq \neg j) \}$$

Cioè metto un arco tra 2 nodi solo se non sono all'interno dello stesso letterale né se sono uno il negato dell'altro

Esempio:

$$(x \vee \neg y) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg z)$$

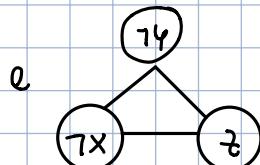
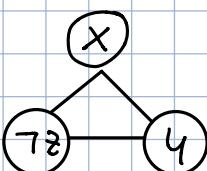


Connetto i nodi che

\Rightarrow non appaiono nello stesso

letterale e che non è
né uno negato!

Gli archi colorati formano una cicca di grado 3 perché formano un grafo completo con i nodi, cioè ogni nodo è connesso a un altro nodo con un arco.



Da ciò ottengo che se c'è una cicca c'è un argomento booleano e viceversa. Quindi ho una **riduzione**.

Appartiene a LOGSPACE perché posso usare lo stesso meccanismo della rappn. binaria e mantenendo gli indici sui nastri di lavoro.

CIRCUITI BOOLEANI

Una funzione booleana $f: \{\text{tt}, \text{ff}\}^n \rightarrow \{\text{tt}, \text{ff}\}$ si realizza mediante un circuito booleano.

Un circuito booleano è un grafo diretto aciclico, con:

- $i \in N$ nodi, chiamati porte
- $(i, j) \in A$ archi

Inoltre:

1) le porte hanno 0, 1, 2 ingressi e una sorta $s(i) = \{\text{tt}, \text{ff}, \top, \perp, \vee, \wedge\}$

2) le porte possono essere di 3 tipi:

- a) **Ingressi**: Sono le porte con $s(i) \in \{\text{tt}, \text{ff}\} \cup X$ e nessun arco entrante
- b) **Uscite**: Sono le porte con il massimo dell'ordinamento parziale e non ha archi uscenti

c) **Nodi intermedi**:

- Se $s(i) = \top$ allora ha 1 uscita e 1 entrata
- Se $s(i) = \{\vee, \wedge\}$ allora ha 1 uscita e 2 entrate

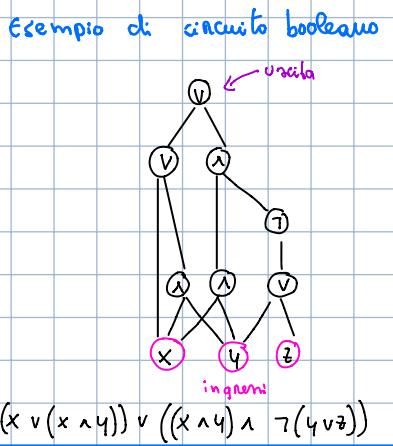
Inoltre:

Per calcolare l'uscita ho bisogno di una funzione di valutazione che assegna i valori di verità agli ingressi, definiamo la semantica

$$\begin{aligned}[i]_V &= \text{tt} \text{ se } s(i) = \text{tt} \\ [i]_V &= f \text{ se } s(i) = \text{ff} \\ [i]_V &= V(x) \text{ se } s(i) = x\end{aligned}$$

$$\begin{aligned}[i]_V &= \text{not}([j]_V) \text{ se } (j, i) \in A \text{ e } s(i) = \neg \\ [i]_V &= [j]_V \text{ and } [h]_B \text{ se } (j, i), (h, i) \in A \text{ e } s(i) = \wedge \\ [i]_V &= [j]_V \text{ or } [h]_B \text{ se } (j, i), (h, i) \in A \text{ e } s(i) = \vee\end{aligned}$$

$$(i, j) \in A$$



Circuit SAT problem

Dato un circuito booleano $C = (N, A)$ ci chiediamo:

Esiste una valutazione booleana che soddisfa il circuito booleano C ?

$$\exists V . [C]_V = tt \text{?} \quad \text{con } [C]_V = [n]_V \text{ con } n \text{ porte di uscita}$$

Caratteristiche:

- 1) Circuit SAT E NP
- 2) Il certificato (Circuit Value) ha costo polinomiale perché tempo sul nastro di input la rappresentazione del grafo, cioè l'unione delle coppie che formano gli archi e la sorta di ogni nodo (Porta)
- 3) Circuit SAT \leq_{LS} SAT

Dim:

Dato $C = (N, A)$ definiamo $f \in \text{logspace}$ t.c

$$\exists V . [[C]]_V = tt \Leftrightarrow \exists V' \subseteq V . [[f(C)]]_{V'} = tt$$

$$\text{Se } [[x]]_V = tt \rightarrow [[x]]_{V'} = tt$$

le variabili $f(c)$ includono tutte le variabili di $C \cup \{ g \text{ variabile per ogni porta di } C \}$

Costruiamo i congiunti di $f(c)$:

Nodo del circuito

- Se g è la porta di uscita \Rightarrow genera un congiunto g variabile
- Se $s(i) = tt$ o $ff \Rightarrow$ genera un congiunto i nel primo caso, $\neg i$ nel secondo caso
- Se $s(i) = x \Rightarrow (i \Leftrightarrow x)$ cioè i è vero se e soltanto se x è vero, cioè $(i \Rightarrow x) \wedge (x \Rightarrow i)$ cioè $(\neg i \vee x) \wedge (\neg x \vee i)$
- Se $s(i) = \wedge \Rightarrow (i \text{ è vero} \Leftrightarrow h \wedge k)$ con $(h, i), (k, i) \in A$, espandendola diventa $\neg(i \vee h) \wedge (\neg i \vee k) \wedge (\neg h \vee \neg k \vee i)$
- Se $s(i) = \vee \Rightarrow (i \text{ è vero} \Leftrightarrow h \vee k)$ con $(h, i), (k, i) \in A$, espandendola diventa $(\neg i \vee h \vee k) \wedge (\neg h \vee i) \wedge (\neg k \vee i)$
- Se $s(g) = \gamma \Rightarrow (h, g) \in A$ allora $g = \gamma h$ cioè $(\gamma g \vee \gamma h) \wedge (g \vee h)$

Tabella di computazione

Dato un problema $I \in P \Leftrightarrow \exists M \mid \forall x \in I . M(x) \xrightarrow{t} (s/\text{no}, w)$
 con $t \leq |x|^k$

In questa MDT inizia con la config $\Delta a_1 \dots a_n$ e dopo i passi
 sarà nelle config $\triangleright a_1 \dots \underline{a_i} \dots a_n$.

Potrei immaginare di mettere tutte queste configurationi in una matrice
 dove la riga i rappresenta l' i -esimo passo della computazione.

\triangleright	a_1	\dots	a_n
\triangleright	<u>a_1</u>	\dots	a_n
:			
\triangleright	$a_1 \dots \underline{a_i} \dots a_n$		
:			
\triangleright	a_1	\dots	<u>a_n</u>

Potremmo dare a questa tabella computazionale un formato standard.

Tabella di computazione Standard

Una matrice quadrata $T[i, j]$ con $1 \leq i, j \leq |x|^k$ se $M(x)$ termina
 in $|x|^k - 2$ passi

Condizioni:

- La macchina termina in meno di $|x|^k - 2$ passi, come detto
- Preso la riga i , essa comincia con il respingente e termina con tutti i caratteri bianchi.
 Tutte le caselle non significative di una riga, quindi, sono riempite con $\#$
 Siccome la lunghezza della riga è $|x|^k$, ma la macchina termina in $|x|^k - 2$ passi, non avrà mai il cursore in ultima posizione perché il tempo limita lo spazio.
- Supponiamo il cursore in una posizione, posso codificare lo stato nell'alfabeto
 L'alfabeto contiene $\sigma_q \in (\Sigma \times Q \times \{h\})$ che registra che nella configuratione i -esima il cursore si trova nella posizione j , si legge σ e lo stato è q . Basta prendere $\Sigma \times Q$ nuovi simboli.
- All'inizio il cursore è sul primo carattere subito dopo il respingente. In più i passaggi sul respingente "indietrovant", cioè i due passi obbligati successivi di andare sul respingente e tornare a destra, vengono condensati in un singolo passo. Così facendo, il cursore non si troverà mai sul respingente. C'è un'eccezione, nel caso seguente
- Quando $T[i, j] = \text{si/no}$, allora sposta il cursore fino alla seconda colonna (con massimo $O(|x|^k)$ passi), introducendo uno stato ausiliario di finto arresto. Lo stato di accettazione, se raggiunto, è quindi sempre in $T(l, 2)$ per qualche $l \leq |x|^k$.
 Si ammette che il cursore passi sopra il simbolo \triangleright quando lo stato sia q_{SI} (abbreviazione di $\sigma_{q_{SI}}$), con il vincolo che non debba mai toccare il \triangleright più a sinistra (che rappresenta l'inizio del nastro).
- Se σ_{SI} oppure σ_{NO} appaiono sulla riga $p < |x|^k$ e nella seconda colonna, allora tutte le righe di indice q con $p \leq q \leq |x|^k$ sono uguali alla p -esima

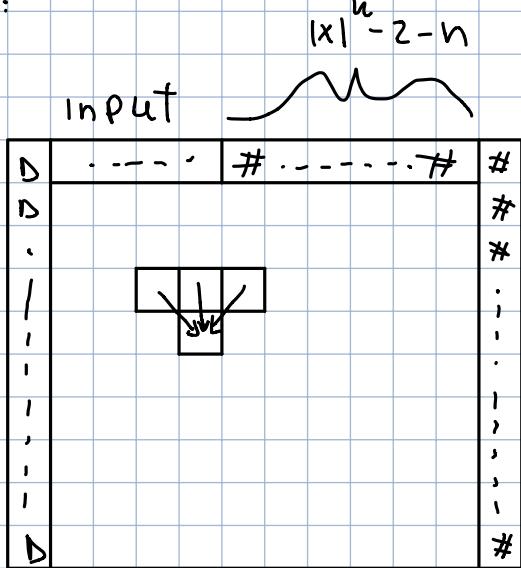
ultima colonna

→ Salvo la pos.
 del cursore

faccio in modo da
 → Non avere il cursore
 sul respingente

la condizione di terminazione con successo: M accetta $x \Leftrightarrow \exists i \mid T(i, 2) = \sigma_{SI} (= T(|x|^k, 2))$

Ynoline:



$T(i, j)$ dipende solo da $T(i-1, j), T(i-1, j-1), T(i-1, j+1)$ determinato da f .

Questo perché la macchina può fare 1 passo solo

cioè: $T_{ij} = f(T(i-1, j), T(i-1, j-1), T(i-1, j+1))$
e f dipende solo da f .

Esempio:

Esempio MdT che verifica se una stringa è palindroma. Esempio: la stringa abba, 16 passi per verificare, quindi 18 righe e colonne

	1	2	3	4	5	6	7	...	17	18
1	\triangleright	a_{q_0}	b	b	a	#	#	...	#	#
2	\triangleright	\triangleright	b_{q_A}	b	a	#	#	...	#	#
3	\triangleright	\triangleright	b	b_{q_A}	a	#	#	...	#	#
4	\triangleright	\triangleright	b	b	a_{q_A}	#	#	...	#	#
5	\triangleright	\triangleright	b	b	a	$#_{q_A}$	#	...	#	#
6	\triangleright	\triangleright	b	b	a'_{q_A}	#	#	...	#	#
7	\triangleright	\triangleright	b	b_{q_1}	#	#	#	...	#	#
8	\triangleright	\triangleright	b_{q_1}	b	#	#	#	...	#	#
9	\triangleright	\triangleright	b_{q_0}	b	#	#	#	...	#	#
10	\triangleright	\triangleright	\triangleright	b_{q_B}	#	#	#	...	#	#
11	\triangleright	\triangleright	\triangleright	b	$#_{q_B}$	#	#	...	#	#
12	\triangleright	\triangleright	\triangleright	b'_{q_B}	#	#	#	...	#	#
13	\triangleright	\triangleright	\triangleright	# $_{q_0}$	#	#	#	...	#	#
14	\triangleright	\triangleright	\triangleright	\triangleright_{SI}	#	#	#	...	#	#
15	\triangleright	\triangleright_{SI}	\triangleright	#	#	#	#	...	#	#
16	\triangleright	\triangleright_{SI}	\triangleright	#	#	#	#	...	#	#
17	\triangleright	\triangleright_{SI}	\triangleright	#	#	#	#	...	#	#
18	\triangleright	\triangleright_{SI}	\triangleright	#	#	#	#	...	#	#

Sposto nella
seconda
colonna

Config di acc.

Padding
↓

righe uguali
a quelle di accettazione

Padding

Trasformare la tabella di computazione in un circuito

1) Ogni simbolo $\sigma \in \Sigma^l = (\Sigma \times Q \times h)$ viene associato a una stringa di bit, necessario di $m = \lceil \log \# \Sigma \rceil$ di bit quindi:

$$\sigma \longleftrightarrow (s, s_1, \dots, s_m)$$

s
bit

Le Δ e $\#$ possono codificarsi come $\Delta = 1 \dots 1$ e $\# = 000 \dots 0$

2) Ciascuna riga della tabella T diventa una seq. di bit

$$T_{i,j} \xrightarrow{f} (s_{i,j,1}, s_{i,j,2}, \dots, s_{i,j,m})$$

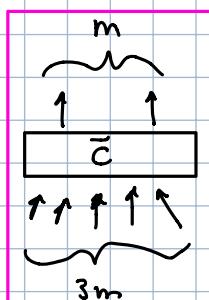
Quindi

$$T \longleftrightarrow S_{i,j,l} = F_f \begin{pmatrix} s_{i-1,j-1,1} & s_{i-1,j-1,2} & \dots & s_{i-1,j-1,m} \\ s_{i-1,j,1} & s_{i-1,j,2} & \dots & s_{i-1,j,m} \\ s_{i-1,j+1,1} & s_{i-1,j+1,2} & \dots & s_{i-1,j+1,m} \end{pmatrix}$$

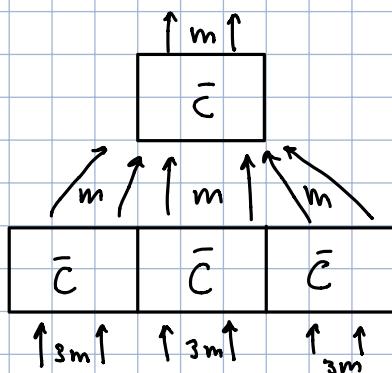
$$F = (F_1, F_2, \dots, F_m) \xrightarrow{f} G$$

F dipende
solo da f
ed è una funz.
booleana

costruire un circuito:

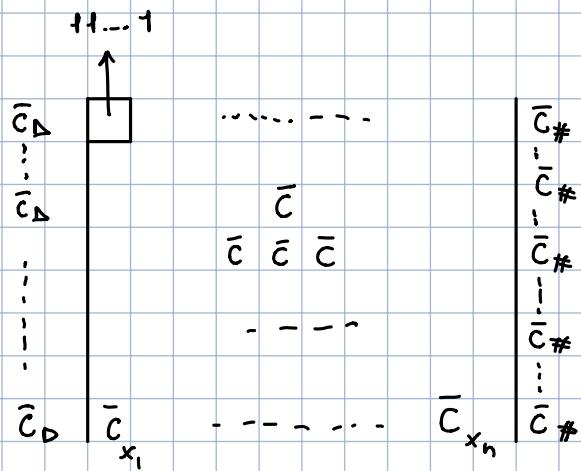


e ottengo



OSSERVAZIONI:

- 1) La costituzione di \bar{C} non dipende dall'input, dipende da f e i 3 m
- 2) Non dipendendo da $x \rightarrow$ il costo di \bar{C} è costante
- 3) Valgono per i circuiti corrispondenti al Δ e al $\#$



$$x \in I \Leftrightarrow [[f(I)]]_{\emptyset} = 11\dots1 \quad \text{e} \quad f \in \text{logSPACE}$$

Dim $f \in \text{logSPACE}$

Ho bisogno di n^u circuiti perché ho bisogno di $\binom{n^u}{(n^u-1), (n^u-2)}$ \bar{C}

e l'indice i, j, k

Quindi ho u nastri di lavoro.

Se gli indici li rappresento in binario ho bisogno di

$$\sum_{p=1}^u u \cdot \log n$$

Quindi ho che Circuit Value è PP-completo.

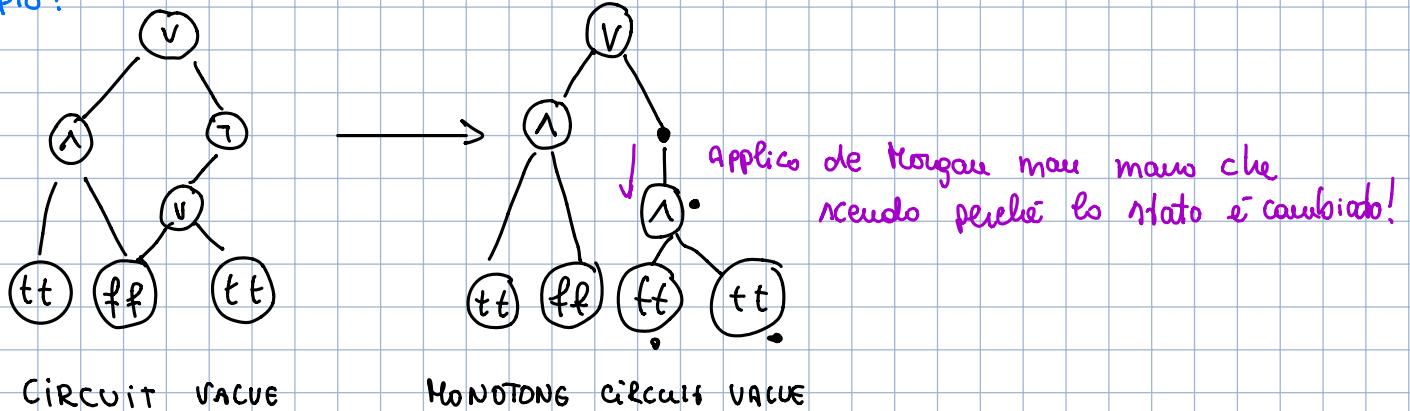
Monotone circuit Value

Circuito dove non compone il Not (\neg) .

OSS: Il not non mantiene l'informazione a differenza dell' and e or

CIRCUIT VALUE \leq_S MONOTONE CIRCUITS VALUE

Esempio:



Quindi MONOTONE circuits VALUE è P - completo

Grammatiche libere dal contesto

Grammatice $G = (N, \Sigma, P, S)$

N alfabeto

Σ simboli terminali

P produzioni, $P = \{ A \rightarrow \alpha \mid A \in N, \alpha \in (N \cup \Sigma)^+ \}$, cioè α non vuota

S $\in N$ simbolo distinto

$$L(G) = \{ w \in \Sigma^+ \mid S \xrightarrow{*} w \}$$

Esempio: $S \xrightarrow{*} () | (s) | ss$ genera il linguaggio del tipo $S \xrightarrow{*} (s) \xrightarrow{*} (ss) \xrightarrow{*} (())$

Il problema, dato G , $L(G) = \emptyset$? Problema P-completo

SAT è NP-completo

Sapendo che CIRCUIT SAT $\leq_{\text{L}}^{\text{S}}$ SAT, quindi vogliamo dimostrare che Circuit-SAT sia NP-completo.

Prendo $I \in \text{NP}$ e costituito

$$f \in \text{logspace} \mid x \in I \Leftrightarrow f(x) = c \text{ e } \exists v \mid [c]_v = tt \\ f(x) \in \text{Circuit SAT}$$

$$T_{i,j} = \cup (T_{i-1,j-1}, T_{i-1,j}, T_{i-1,j+1}, \text{scelta})$$

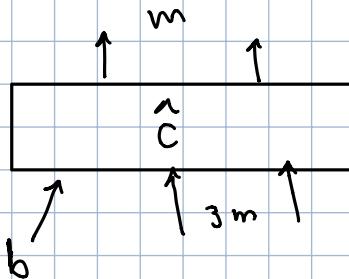
↳ la scelta divenuta un b

N con grado p $\exists N'$ con grado 2

La computazione è un insieme di scelte $(b_0, \dots, b_{(n-j)^k})$ dove $b_i \in \{0,1\}$

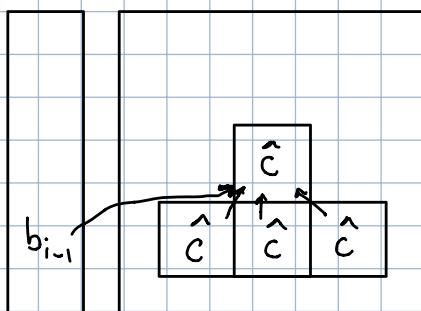
$T_{i,j}$ divenuta un nuovo circuito che ha $3m$ entrate, m uscite e

\rightarrow bit



Il costo del circuito è costante e dipende da Δ e N'

Il bit b_i è la computazione che risolve $x \in I$



Altri Problemi NP-completi: HAM, CRICCA, COMMUNO VIAGGIATORE ecc....