

Lessione I

problema \rightarrow soluzione (naturale)

Soluzioni di natura informatica:

- algoritmo
- programma
- processo

Computer: è l'individuo che compie l'azione
to compute (fare di conto)

Def. algoritmo

- sequenza finita: c_1, c_2, \dots, c_n
- deterministica: c_1, c_2, \dots, c_n
- operazioni ben definite: c_i operazione

Individuo
coordinatore mecc.
calcolatore elettronico

Organisme
di flusso.

- ha un solo inizio
- passa attraverso più fasi
- da un'operazione obbligato una
sola risposta certa \Rightarrow percorso deterministico
- ben definite le condizioni

Lessione II

$$\begin{aligned} D &= \text{Dominio} = \{d_1, \dots, d_n\} & f: D \rightarrow C \\ C &= \text{codominio} = \{c_1, \dots, c_m\} & f(d) = c \end{aligned}$$

Def. funzione corrispondente $f: D \rightarrow C$ uno è un solo elemento $c \in C \Rightarrow f(d) = c$

Def. Sono relazioni di D dove sono compatti tutti gli elementi della funzione

Una funzione può essere:

- Iniettiva:** elementi distinti del dominio hanno $\Rightarrow d_1, d_2 \in D \text{ t.c. } d_1 \neq d_2 \Rightarrow f(d_1) \neq f(d_2) \Leftrightarrow \text{Card}(D) \leq \text{Card}(C)$
- Suriettiva:** ogni elemento del codominio è immagine $\Rightarrow \forall c \in C, \exists d \in D \Leftrightarrow \text{Card}(D) \geq \text{Card}(C)$
- Bisettiva:** funzione è iniettiva e suriettiva $\Leftrightarrow D = \{ \dots \} = C - \{ \dots \}, \{f\} \text{ è invertibile}$

$$\begin{aligned} \text{naturale: } \mathbb{N} &= \{1, 2, 3, \dots\} \\ \text{relativi: } \mathbb{Z} &= \{-1, -2, 0, 1, 2, \dots\} \\ \text{rettangoli: } Q &= \{ \frac{a}{b} \} \end{aligned}$$

Def. In insieme A si dice numerabile se $\text{Card}(A) = \text{Card}(\mathbb{N})$

NO: $\text{Card}(\mathbb{R}) = \text{Card}(\mathbb{N})$ sono etichette e opposte (può costruire una relazione biunivoca)

NO: $\text{Card}(\mathbb{Q}) = \text{Card}(\mathbb{N})$ etichette diagonali e zig-zag \Rightarrow percorso bidirezionale

NO: insieme $G = \{ \text{stringhe binarie infinite} \}$ non è possibile mettere in corrispondenza con $\mathbb{N} \Rightarrow G$ non è numerabile

Sintesi:
Funzione | Relazione | Corrispondenza

Lessione II

Osservazione

- algoritmo di Euclide $\text{MCD}(k_1, k_2) \rightarrow$ corretto, terminazione
- correttezza di un algoritmo: fornisce risultato corretto
- algoritmo soddisfa le terminazioni: garantisce termine (no loop)
- 2 affermazioni vere anche le loro congiunture sono vere

N.B.: se per un problema \exists un algoritmo che lo risolve allora ne esistono infiniti altri

Iterazioni

con $x \rightarrow y$

$$k: x = q_1 r_1$$

$$k+1: r_1 = q_2 r_2$$

$$k+2: r_2 = q_3 r_3$$

$$x = q_1 r_1$$

$$y = r_1 q_2 r_2$$

$$r_1 = r_2 q_3 r_3$$

Dimostrazione: $\{ \text{divisori di } x \in y \} = \{ \text{divisori di } y \}$

Hip: I) d divide $x \in y \Rightarrow d$ divide y e f
II) d divide $y \in x \Rightarrow d$ divide $x \in y$ tesi

Lessione III

tipi di infinito: 1) \mathbb{N} con corrispondenze (\mathbb{Z}, \mathbb{Q}) \Rightarrow numerabile
2) G non può essere messo in corrispondenza \Rightarrow non numerabile

N.B.: siamo relativi alle funzioni e agli algoritmi (sequenze finite, deterministica, ben definito)

Def: una funzione $f: D \rightarrow C$ è detto computabile quando si può scrivere un algoritmo A tale che:

- per ogni funzione computabile si può scrivere un algoritmo. (teoricamente e non operativamente)

Def: una funzione caratteristica ha:

- Domini $= D = \mathbb{N}$ (input)
- Codominio $= C = \{q\}$ "si" o "no" (output)

Def: In insieme si dice decidibile quando esiste funzione caratteristica e computabile

decidibile \Rightarrow computabile $\Rightarrow \exists \text{ op. algoritmi}$

caso: $A \in \text{Generico} \rightarrow \mathbb{N}$
 $\mathbb{N} \rightarrow A \in \text{Generico}$

questo
lezione II

Def: dati insiem A e B si chiama prodotto cartesiano
l'insieme di tutte le due coppe (a, b) dove $a \in A$ e $b \in B$

$$\text{prodotto cartesiano: } A \times B$$

teorema: se A e B numerabili
B è anche $A \times B$
 \Rightarrow vol. rispondenza ottenuta

Def: dato un insieme A, si dice l'insieme delle parti di A, l'insieme di tutti e solo i sottoset di A $\Rightarrow P(A)$

$$\begin{aligned} \text{Card}(A) &= 3 & \text{ricordarsi A stesso} \\ \text{Card}(P(A)) &= 2^{|A|} = 8 & \text{e l'insieme tutto} \end{aligned}$$

gruppo:

$$\begin{aligned} A &= \{1, 2\} \\ P(A) &= \{\emptyset, \{1\}, \{2\}, \{1, 2\}\} = 2^{\text{Card}(A)} \end{aligned}$$

base 2: $a \in A?$ $\begin{cases} \text{si} & \\ \text{no} & \end{cases} \Rightarrow$ viene fatto esaurire

$$\begin{aligned} \text{no. costituenti per ogni elemento} \\ 2^1 = 2 \text{ usa 2 bit per le costituenti} \\ \{1\} = 00 \quad \{2\} = 01 \quad \{1, 2\} = 10 \quad \{\emptyset\} = 11 \end{aligned}$$

$f: \mathbb{N} \rightarrow \mathbb{N}$
f come coppie di valori $\Rightarrow \{ (x, f(x)) \mid x \in \mathbb{N} \} = \{ (x, y) \mid x, y \in \mathbb{N} \}$

Computabilità dicembre 2021

Mario Verdicchio

Università degli Studi di Bergamo

Anno Accademico 2021-2022

Cardinalità di un insieme

- Cardinalità di un insieme:
 - se l'insieme è finito, è il numero dei suoi elementi
 - se l'insieme è infinito, si possono solo fare confronti di cardinalità tra insiemi
- Dati due insiemi A e B:
 - $\text{Card}(A) \leq \text{Card}(B)$ quando esiste una $f: A \rightarrow B$ iniettiva
 - $\text{Card}(A) = \text{Card}(B)$ quando esiste una $f: A \rightarrow B$ biiettiva
 - $\text{Card}(A) < \text{Card}(B)$ quando $\text{Card}(A) \leq \text{Card}(B)$ e $\text{Card}(A) \neq \text{Card}(B)$, ossia esiste una funzione iniettiva da A a B ma non esiste una funzione biiettiva da A a B

Numerabilità

- Un insieme si dice numerabile quando ha la stessa cardinalità di N
- Esempi di insiemi numerabili: Z (interi), Q (razionali), tutti i sottoinsiemi infiniti di N (numeri pari, numeri dispari, numeri primi...)
- Il prodotto cartesiano ($A \times B$) di due insiemi numerabili è anch'esso numerabile (costruire una tabella e visitarne gli elementi con percorsi diagonali)
- Di conseguenza, il prodotto cartesiano di un numero qualunque di insiemi ($A_1 \times A_2 \times \dots \times A_n$) è numerabile

$\mathbb{N}: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, \dots$
 $\uparrow \downarrow \uparrow \downarrow \uparrow \downarrow \uparrow \downarrow \uparrow \downarrow \uparrow \downarrow \uparrow \dots$
A: $a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, \dots$

Un insieme A è numerabile =
tutti i suoi elementi possono essere
messi su una fila

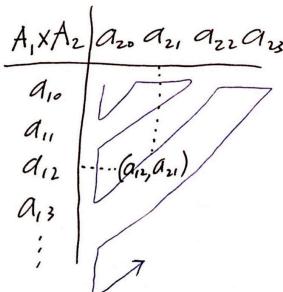
$\mathbb{Z}: 0, -1, 1, -2, 2, -3, 3, -4, 4, -5, 5, -6, 6, \dots$

Ossia possiamo etichettare ogni elemento dell'insieme A con una etichetta di numeri naturali senza mai finirli. Questo vuol dire che \Rightarrow creo una relazione biunivoca tra i due insiemi (biunivoca cardinalità degli insiemi è uguale)

Corrispondenza biunivoca è una funzione che da un insieme va a un altro insieme

percorso e rig-arg = bistrofodico

$\textcircled{6}$	0	1	2	3	4	5	6	$A_1 \times A_2$	A_3
-1	0	-1	-2	-3	-4	-5	-6		
1	0	1	2	3	4	5	6	a_{10}	\checkmark
-2	0	-1/2	-2/2	-3/2	-4/2	-5/2	-6/2	a_{11}	
2	0	1/2	2/2	3/2	4/2	5/2	6/2	a_{12}	
-3	0	-1/3	-2/3	-3/3	-4/3	-5/3	-6/3	a_{13}	
3	0	1/3	2/3	3/3	4/3	5/3	6/3		
-4	0	-1/4	-2/4	-3/4	-4/4	-5/4	-6/4		
4	0	1/4	2/4	3/4	4/4	5/4	6/4		



$$\underbrace{A_1 \times A_2 \times A_3 \times A_4 \times \dots}_{\text{...}}$$

- ogni f appartiene ad $s \Rightarrow$ abbiamo una codifica/corrispondenza iniettiva
- Ponendo lo 0 davanti mi perdo tutte le stringhe che iniziano per 1, quindi gli elementi di S che iniziano per 1 non saranno mai coincidenti, dunque $\text{Card}(S) = \text{Card}(M)$

$$\begin{array}{c}
 S \\
 \in \\
 S = 1011010010000\dots 000\dots \\
 \downarrow \\
 \text{ultimo "1"} \\
 2^0 \quad 2^1 \quad 2^2 \quad 2^3 \quad 2^5 \quad 2^8 \rightarrow 2^0 + 2^1 + 2^2 + 2^3 + 2^5 + 2^8 = 301 \in \mathbb{N} \\
 f(S) = 301 \quad f^{-1}(301) = S \quad f \text{ biunívoca}
 \end{array}$$

$\varphi = (k_1, \dots, k_n)$
 $0 K_1^* 0 K_2^* \dots K_n^* 0000 \dots 00 \dots \in S$
 ↓ $k_i^* = k_{i+1} \text{ "1"}$
 esiste funzione INIETTIVA fda $F \rightarrow S$
 $(\varphi_1 + \varphi_2 \xrightarrow{f} s_1 + s_2)$
 quindi $\text{Card}(F) \leq \text{Card}(S) = \text{Card}(\mathbb{N})$
 essendo F infinito, allora $\text{Card}(F) = \text{Card}(\mathbb{N})$

F infinito implica che $\Rightarrow \text{Card}(f) < \text{Card}(\mathbb{N}^J)$ impossibile !!
 $\Rightarrow \text{Card}(F) = \text{Card}(\mathbb{N}^J)$

Altri insiemi numerabili 1/2

- **S**, l'insieme delle stringhe binarie infinite che da un certo punto in poi contengono solo '0'
 - lo si dimostra mostrando che la codifica binaria costituisce una corrispondenza biunivoca con i numeri naturali
 - **F**, l'insieme di tutte le n-uple di numeri naturali
 - lo si dimostra con la codifica $n^* = (n+1)$ caratteri '1' facendo corrispondere a ciascun elemento di F (k_1, \dots, k_n) la stringa $0, k_1^*, 0, k_2^*, \dots, 0, k_n^*, 0, 0 \dots 0$... che appartiene a S

$$n = 23 \quad (16+4+2+1)$$

$$S_n = 111010 \dots$$

0 1 3 6 9

$$n_8 = \sum_{i=0}^7 2^i \cdot b_i = 2^0 \cdot 1 + 2^1 \cdot 1 + 2^2 \cdot 1 + 2^3 \cdot 1 = 1 + 2 + 4 + 16$$

$$f = (1, L) \\ = 01101100\dots$$

Altri insiemi numerabili 2/2

- Se Σ è l'alfabeto (finito o numerabile) di un linguaggio, le stringhe di questo linguaggio sono n-uple finite di elementi di Σ
 - Come nel caso di F, che è costituito da n-uple finite di elementi di N ed è numerabile, così il linguaggio basato su Σ è anch'esso numerabile
 - Quindi, l'insieme Prog dei programmi (che non sono altro che n-uple dell'alfabeto) è numerabile

- Ciascun carattere che utilizziamo nel codice subisce una codifica ASCII quindi da carattere otteniamo un numero
- L'ordine dei numeri conta, dunque da una stringa otteniamo una terna di valori numerici F

```
while (1)
    print(0);
↑
(w,h,i,l,e,( , ),p,r,i,n,t,( , 0, ))
↑
(119,104,105,108,101,40,49,41,112,114,105,110,116,40,48,41) ∈ F
```

4fc11

$\text{PROG} = \text{insieme dei programmi}$
 $\text{Card}(\text{PROG}) = \text{Card}(F) = \text{Card}(\mathbb{N})$

da aggiungere:
 $;$

G
 $g_0 = 010000111000\dots$
 $g_1 = 1101010011\dots$
 $g_2 = 101010101010\dots$
 $g_3 = 0001000100\dots$
 \vdots
 $\overline{g_h} =$ bit sulla diagonale:
formano una stringa binaria infinita, $g_k \in G$
Complementiamo $g_k \rightarrow \overline{g_k}$
Anche $\overline{g_k} \in G$

$g_{kk} \in g_h \Rightarrow g_{kk} = 0$
 $g_{kk} \in \overline{g_h} \Rightarrow g_{kk} = 1$

g_{kk} è il k-esimo bit di g_k
e anche il k-esimo bit di $\overline{g_k}$
 g_{kk} vale 0 e 1 contemporaneamente
ASSURDO

Insieme non numerabile

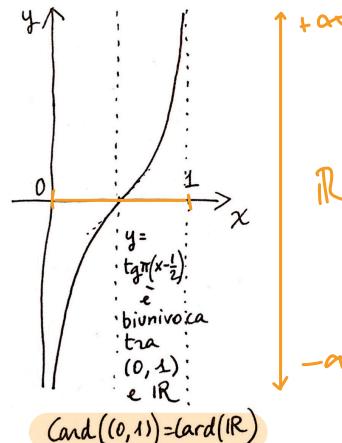
- G, l'insieme delle stringhe binarie infinite, non è numerabile
 - si dimostra per assurdo: se G fosse numerabile, i suoi elementi gli potrebbero essere messi in colonna, costruendo così una matrice binaria infinita a destra e in basso
 - si prenda la diagonale principale di questa matrice e si costruisca la stringa binaria infinita g^* , complementando tutti i suoi bit
 - g^* appartiene a G, quindi è presente nella matrice, su una certa riga
 - il bit di intersezione tra questa riga e la diagonale principale deve essere il complemento di se stesso (perché appartiene contemporaneamente alla diagonale principale e a g^*)
 - ASSURDO, quindi è assurdo supporre che G sia numerabile

Altri insiemi non numerabili

- Le stringhe binarie infinite di G possono essere messe in corrispondenza biunivoca tramite codifica binaria con tutti i numeri reali compresi tra 0 e 1, quindi $\text{Card}(G) = \text{Card}((0,1))$
- Tramite la funzione $y = \tan(\pi(x-1/2))$ il segmento $(0,1)$ può essere messo in corrispondenza biunivoca con tutto R
- Quindi $\text{Card}(G) = \text{Card}(R)$

$$0, \frac{0101011100 \dots}{2^{-2} 2^{-4} 2^{-5} 2^{-6} 2^{-7}} = 0,361$$

$$0, \quad 2^{-2} 2^{-4} 2^{-6} 2^{-7} = 0,3671875\ldots \in (0,1)$$



Dimostrazione x grande

- La tesi è: non esiste una $f:A \rightarrow P(A)$ biiettiva
 Visto che esiste una $f_i:A \rightarrow P(A)$ iniettiva
 $(f_i(a) = \{a\}, \text{ per ogni } a \in A)$, la tesi diventa che non esiste
 una $f:A \rightarrow P(A)$ suriettiva
 - Per assurdo: supponiamo esista una f suriettiva, ossia
 per ogni $A' \subseteq A$, Ossia A' è un elemento di $P(A)$ o sia $A' \in P(A)$
 esiste un $a' \in A$ tale che $f(a') = A'$
 - Ci chiediamo: $a' \in f(a')$? $a' \in f(a') = \{e\}$?
 - Costruiamo $B = \{x \in A : x \notin f(x)\} = \{x \in A : x \notin f(x) = \{x\}\}$ $B \subseteq A$ $B \in P(A)$
 - Essendo f suriettiva, esiste $b \in A$: $f(b) = B$
 - Se ci chiediamo $b \in B$? otteniamo un assurdo per
 entrambe le possibili risposte:
 - Sì: $b \in B$, quindi per come è definito B : $b \notin f(b)$, quindi $b \notin B$
 perché $f(b) = B$
 - No: $b \notin B$, quindi $b \in f(b)$ perché $f(b) = B$, quindi per come è definito B : $b \in B$

$$f: A \rightarrow f(A)$$

$e \xrightarrow{+} \{e\}$

Sect. 6.1 (1)

Insieme delle parti

- Dato un insieme A, l'insieme delle parti di A, indicato con $P(A)$ è l'insieme di tutti e soli i suoi sottoinsiemi
 - Se A è finito, $P(A)$ contiene $2^{\text{Card}(A)}$ elementi
 - Ad es. se $A = \{0; 1; 2\}$
 $P(A) = \{\{0\}; \{1\}; \{2\}; \{0; 1\}; \{1; 2\}; \{0; 2\}; \{\}\}; \{0; 1; 2\}\}$
 - Teorema di Cantor:
se A è un insieme (finito o infinito),
 $\text{Card}(A) < \text{Card}(P(A))$

$\text{Card}(A) \leq \text{Card}(\mathcal{P}(A))$ iniettiva
 $\text{Card}(A) = \text{Card}(\mathcal{P}(A))$ biettiva = Suriettiva + iniettiva
 con il teorema di Cantor
 abbiamo dimostrato che $\exists f$.
 iniettiva ma non biettiva
 $\Rightarrow \text{Card}(A) \leq \text{Card}(\mathcal{P}(A))$ AND
 $\text{Card}(A) \neq \text{Card}(\mathcal{P}(A))$
 $\Rightarrow \text{Card}(A) < \text{Card}(\mathcal{P}(A))$

Conseguenze

Una singola stringa binaria infinita è una descrizione estesa di un sottoinsieme di $\{0, 1\}^{\mathbb{N}}$

Numeri cardinali

- $\text{Card}(N) = \aleph_0$ ("aleph zero")
- $\text{Card}(P(N)) = \text{Card}(R) = \aleph_1$
- $\text{Card}(P(P(N))) = \text{Card}(P(R)) = \aleph_2$
- ...e così via: esistono infiniti (\aleph_0) numeri cardinali

$$\Psi_{\text{comp}} \cup \Psi_{\text{NC}} = \Psi$$

$$\Psi_C \cup \Psi_{NC} = \Psi$$

$$f: \mathbb{N} \rightarrow \mathbb{N}$$

Esistenza delle funzioni non computabili 2/3

- Cardinalità dell'insieme Ψ_C (funzioni computabili $N \rightarrow N$):
 - ogni funzione computabile è computata almeno da un algoritmo, quindi $\text{Card}(\Psi_C) \leq \text{Card}(\text{ALG})$ (ALG : insieme degli algoritmi)
 - per ogni numero naturale n , esiste una funzione computabile f_n tale che $f_n(x) = n$ per ogni x , quindi $\text{Card}(N) \leq \text{Card}(\Psi_C)$
 - dal momento che $\text{Card}(\text{ALG}) = \text{Card}(N)$, abbiamo che la cardinalità di Ψ_C non può che essere la cardinalità di N , cioè \aleph_0

Esistenza delle funzioni non computabili 1/3

- Cardinalità dell'insieme Ψ (funzioni $N \rightarrow N$):
 - ogni $f \in \Psi$ definisce un insieme di coppie $(x, f(x))$, quindi corrisponde a uno e un solo elemento di $P(N \times N)$: esiste quindi una funzione iniettiva da Ψ a $P(N \times N)$
 - $N \times N$ ha la stessa cardinalità di N , cioè \aleph_0
 - $P(N \times N)$ ha quindi la cardinalità di $P(N)$, cioè \aleph_1
 - Ψ ha quindi cardinalità $\leq \aleph_1$
 - ogni elemento di G , ossia ogni stringa binaria infinita può essere vista come una particolare funzione $N \rightarrow N$ (con range $\{0;1\}$): esiste quindi una funzione iniettiva da G a Ψ
 - quindi $\text{Card}(G) = \aleph_1 \leq \text{Card}(\Psi)$
 - unendo i due risultati: $\text{Card}(\Psi) \leq \aleph_1 \leq \text{Card}(\Psi)$, ossia $\text{Card}(\Psi) = \aleph_1$

I

II

⇒

$$\text{I} \quad \text{Card}(\Psi) \leq \text{Card}(P(N \times N)) = \text{Card}(R)$$

$$\text{II} \quad \text{Card}(G) \leq \text{Card}(\Psi)$$

$$\text{Card}(R) \in \text{Card}(\Psi) \leq \text{Card}(R)$$

$$\text{Card}(G) = \text{Card}(R)$$

Esistenza delle funzioni non computabili 3/3

- Cardinalità dell'insieme Ψ_{NC} (funzioni non computabili $N \rightarrow N$):
 - $\Psi = \Psi_C \cup \Psi_{NC}$, inoltre sono chiaramente disgiunti
 - se Ψ_{NC} avesse cardinalità finita oppure pari a \aleph_0 , Ψ avrebbe cardinalità \aleph_0 , perché l'unione di due insiemi numerabili è numerabile: si può infatti enumerare l'unione alternando gli elementi provenienti dai due insiemi
 - dato che la cardinalità di Ψ è \aleph_1 , segue che Ψ_{NC} ha cardinalità \aleph_1

Generalità delle funzioni $N \rightarrow N$

- Le funzioni monoargomentali $N \rightarrow N$ sono un caso particolare delle funzioni a più argomenti $N^n \rightarrow N$
- Non si ha comunque perdita di generalità nel trattare solo le monoargomentali, perché le n-uple possono essere codificate in numeri naturali:
 - data una n-upla (k_1, \dots, k_n) ,
 $t(k_1, \dots, k_n) = 0k_1^*0k_2^*0\dots0k_n^*$
 dove k_j^* è una sequenza di k_j+1 caratteri '1'
 - data una stringa binaria $b = b_0b_1\dots b_m$
 $p(b_0b_1\dots b_m) = \sum_{i=0}^m (2^i b_i)$

Σ^N

Def: Una funzione $f: A \rightarrow B$ è **computabile** quando $f(a) = b$ esiste un algoritmo che dato a e un input restituisce b in output.

Def: insieme delle funzioni computabili

Φ_{comp} o Ψ_c

Def: $\forall f$ computabile \exists elemento un programma che la computa.

$$\text{Card}(\mathbb{E}_{\text{comp}}) \leq \text{Card}(\text{Prog}) = \text{Card}(\mathbb{N})$$

$\forall n \in \mathbb{N} \exists$ una funzione computabile

$$\text{Card}(\mathbb{N}) \leq \text{Card}(\mathbb{E}_{\text{comp}})$$

$$\Sigma_c \cup \Sigma_{nc} = \Sigma = \Psi_c \cup \Psi_{nc} = \Psi$$

$$\text{Card}(\mathbb{N}) \cup \text{Card}(\mathbb{N}) = \text{Card}(\mathbb{N})$$

Riassunto

teorema di Cantor: dato A insieme qualunque
 $\text{Card}(A) < \text{Card}(\mathcal{P}(A))$

Ipotesi delle contraddizioni:

$$\nexists A : \text{Card}(\mathbb{N}) < \text{Card}(A) < \text{Card}(\mathbb{R})$$

dunque:

$$\text{Card}(\mathbb{N}) = \text{Card}(\mathbb{Q}) = \text{Card}(\mathbb{S}) = \text{Card}(\mathbb{F}) = \text{Card}(\text{Prog}) = \text{Card}(\text{Alg}) = \text{Card}(\mathbb{N})$$

$$\text{Card}(\mathbb{Q}) = \text{Card}(\mathbb{R}) = \text{Card}(\mathbb{S})$$

$$\text{Card}(\mathbb{G}) = \text{Card}(\mathcal{P}(\mathbb{N})) = \text{Card}(\mathbb{R})$$

teorema di Cantor: $\text{Card}(\mathbb{N}) < \text{Card}(\mathcal{P}(\mathbb{N})) = \text{Card}(\mathbb{M})$

$$\text{Card}(\mathbb{L}) = \text{Card}(\mathbb{D}) \quad \text{"etichette spiele"}$$

$$\text{Card}(\mathbb{Q}) = \text{Card}(\mathbb{D}) \quad \text{"etichette zig-zag native"}$$

$$\text{Card}(\mathbb{G}) = \text{Card}((\mathbb{N})) \quad \text{"etichette O, codifica"}$$

$$\text{Card}(\mathbb{R}) = \text{Card}((\mathbb{N})) \quad \text{"etichette g=tan(x-y)"}$$

$$\text{Card}(\mathbb{G}) = \text{Card}(\mathcal{P}(\mathbb{N})) \quad \text{"etichette 0,1 (no ci) stringe"}$$

$$\text{Card}(\mathbb{G}) = \text{Card}(\mathbb{D}) \quad \text{"etichette codifica binarie"}$$

$$\text{Card}(\mathbb{F}) = \text{Card}(\mathbb{D}) \quad \text{"un solo o spazio l=k+1"}$$

$$\text{Card}(\text{Prog}) = \text{Card}(\mathbb{N}) \quad \text{"ASCII carattere -F"}$$

$$\text{Card}(\text{Alg}) = \text{Card}(\mathbb{N})$$

$$\text{Card}(\text{Alg}) \subseteq \text{Card}(\text{Prog}) = \text{Card}(\mathbb{F}) = \text{Card}(\mathbb{N})$$

$$\text{Card}(\text{Prog}) = \text{Card}(\text{Prog}) = \text{Card}(\text{Alg}) = \text{Card}(\mathbb{N})$$

$$\text{Card}(\mathbb{N} \times \mathbb{N}) = \text{Card}(\mathbb{N}) \quad \text{"potesse robe di Q"}$$

$$\text{Card}(\mathcal{P}(\mathbb{N} \times \mathbb{N})) = \text{Card}(\mathcal{P}(\mathbb{N}))$$

$$\text{Card}(\Psi) = \text{Card}(\mathbb{N})$$

$$\text{Card}(\Psi) = \text{Card}(\text{Alg}) = \text{Card}(\mathbb{N})$$

$$\text{Card}(\Psi_{nc}) = \text{Card}(\mathbb{N})$$

$$\text{Card}(\{\text{f}, \text{g}\}) = \text{Card}(\mathbb{N})$$

$$f: \mathbb{N} \rightarrow \mathbb{N}$$

$$\cdot \psi, \lambda x, \psi, G$$

Numeralibilità $\rightarrow \text{Card}(A) = \text{Card}(\mathbb{N})$ "conteggio"

Enumerabilità \rightarrow In insieme A è enumerabile quando è il **range** di una f computabile e dato $f: \mathbb{N} \rightarrow A$ numerico A dopo la "posta" degli elementi di A .

$$\begin{aligned} f: \mathbb{N} &\rightarrow A \\ f(1) &= \text{a} \\ f(2) &= \{\text{a}, \text{b}\} \\ f(3) &= \{\text{a}, \text{b}\} \\ f(4) &= \{\text{a}, \text{b}\} \end{aligned}$$

Range = sotto dominio coinvolto nelle funzioni, cioè solo output delle funzioni.

Def: Un insieme enumerabile si dice anche **reciducibile**.

Def: Un insieme A è **decidibile** quando lo sia f_A caratteristica $\Rightarrow f_A(x) = \begin{cases} 0 & \text{se } x \in A \\ 1 & \text{se } x \notin A \end{cases}$

Computabilità

10 gennaio 2022

Mario Verdicchio

Università degli Studi di Bergamo

Anno Accademico 2021-2022

Esercizio 1

- Creare una macchina di Turing (MT) che accetti il seguente linguaggio:
 0^i , con i intero pari

Esercizio 1

- **Creare** una macchina di Turing (MT) che accetti il seguente linguaggio:
 0^i , con i intero pari

Concepire l'algoritmo che la governa e descriverne il funzionamento per mezzo di una tavola di istruzioni

Esercizio 1

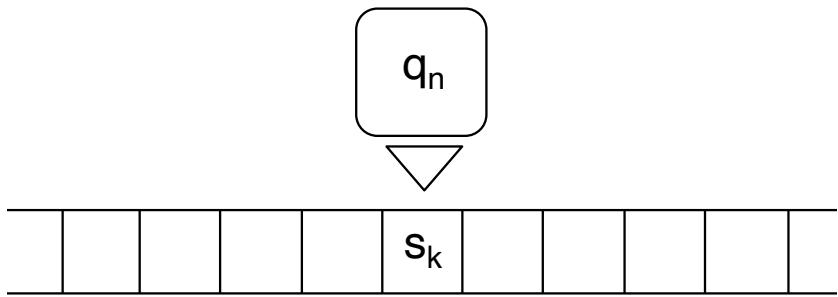
- Creare una macchina di Turing (MT) che **accetti** il seguente linguaggio:
 0^i , con i intero pari

Se la MT trova sul nastro una sequenza di simboli che rispetta la definizione del linguaggio, l'output sul nastro è 0 (= si).

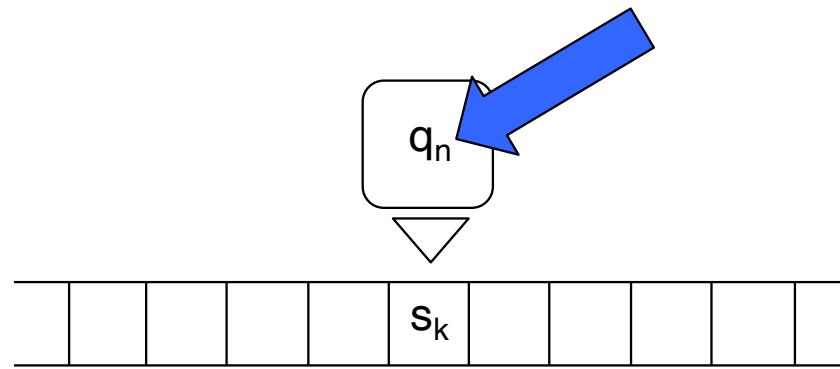
Se la MT trova sul nastro una sequenza di simboli che NON rispetta la definizione del linguaggio, l'output sul nastro è 1 (= no).

Attenzione: sul nastro deve rimanere solo lo 0 oppure l'1. La sequenza di input va cancellata.

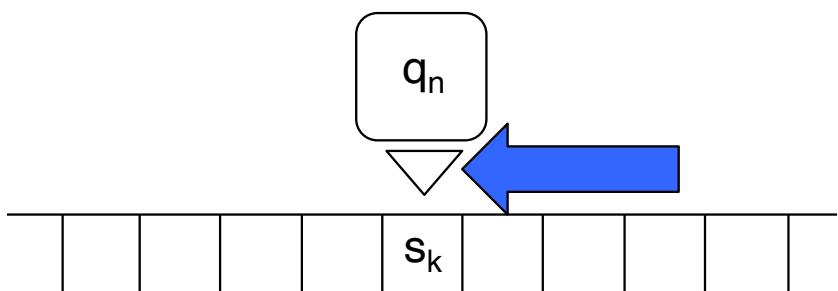
La Macchina di Turing



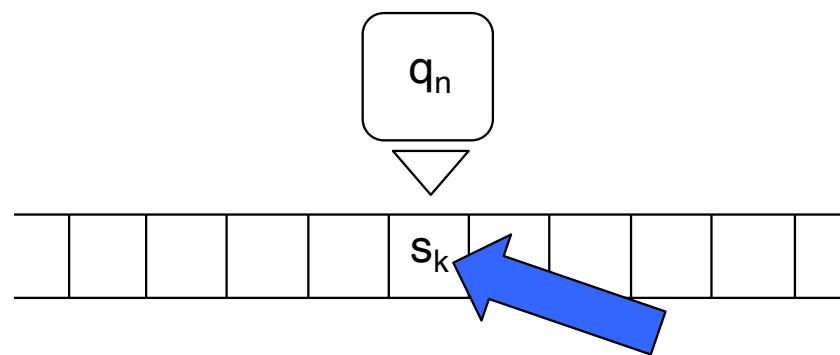
Stato interno



Testina di lettura e scrittura



Simbolo sul nastro biinfinito

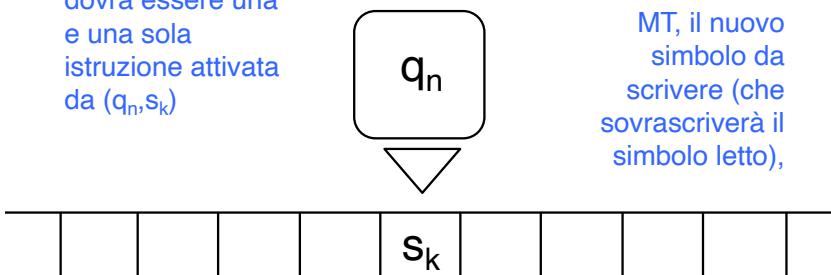


Funzionamento della MT

- La MT esegue l'istruzione all'interno del suo programma che è attivata dalla **configurazione** in cui la MT si trova
- Tale configurazione è determinata da:
 - lo stato interno della MT
 - il simbolo letto dalla testina

Funzionamento della MT

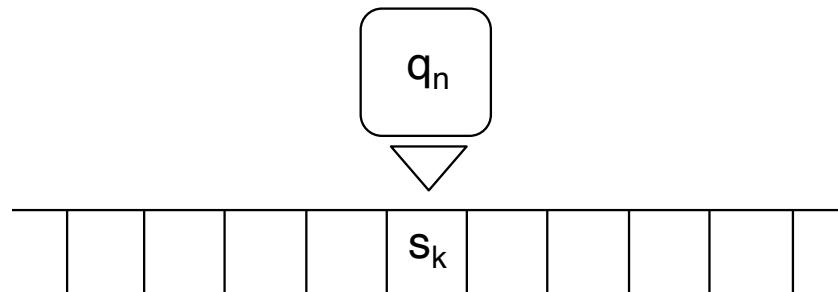
Nel suo programma ci dovrà essere una e una sola istruzione attivata da (q_n, s_k)



Tale istruzione indicherà il nuovo stato interno della MT, il nuovo simbolo da scrivere (che sovrascriverà il simbolo letto),

e il movimento da eseguire (L: left, R: right, C: center)

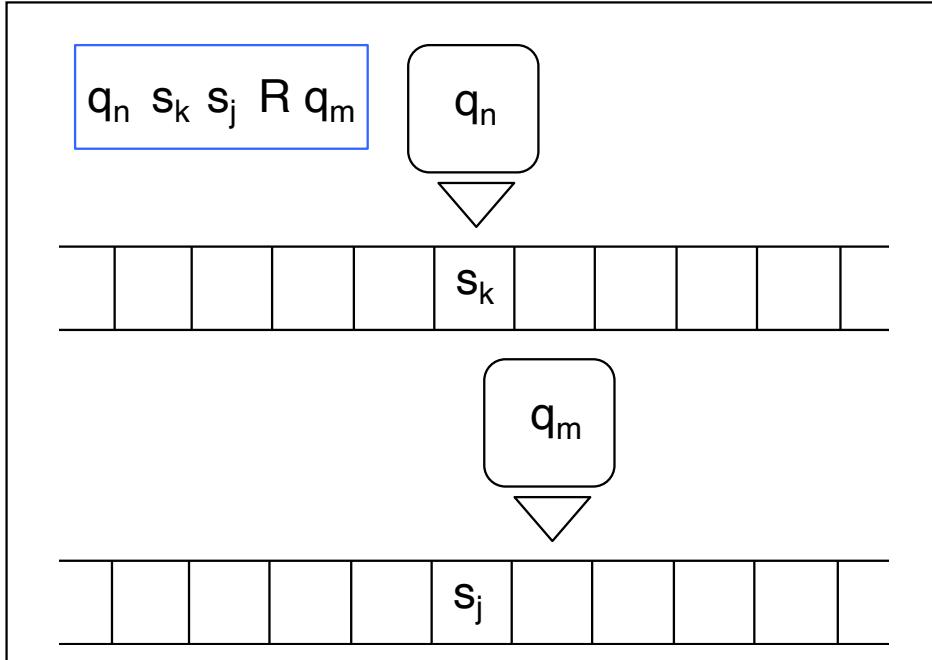
La MT è nella configurazione (q_n, s_k)



configurazione MT

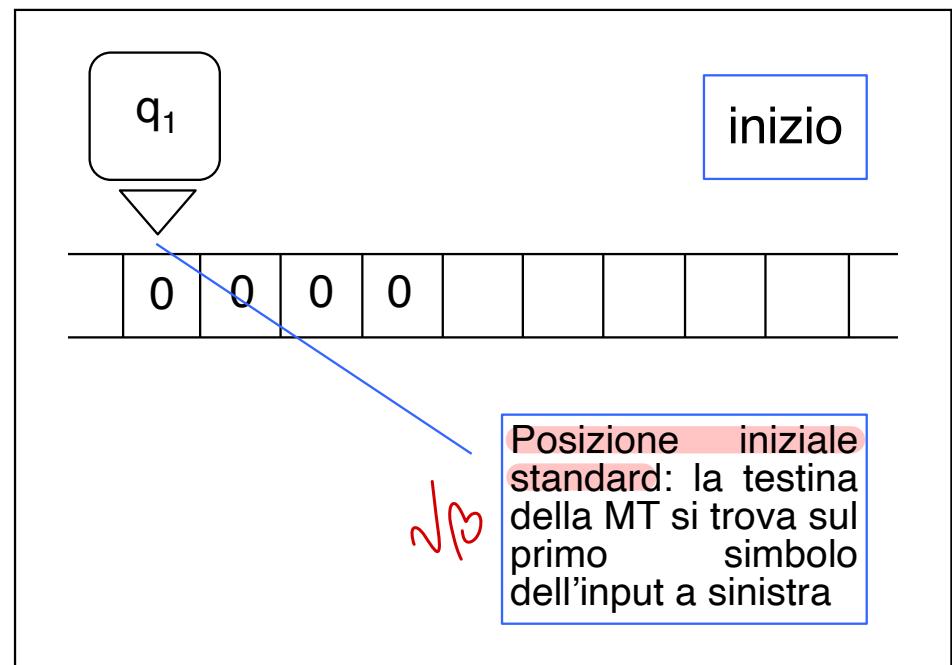
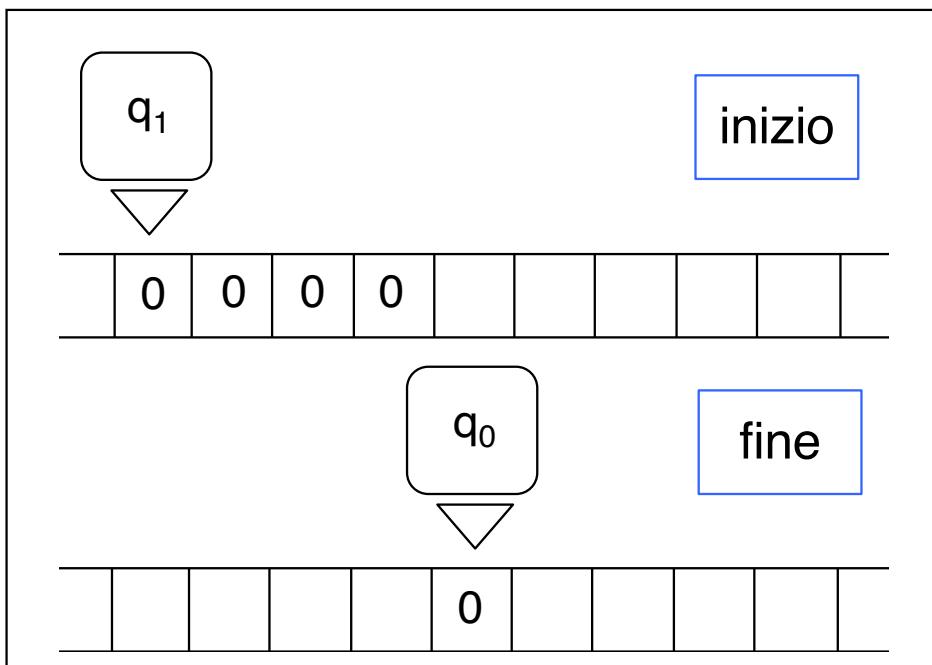
$q_n \quad s_k \quad s_j \quad R \quad q_m$

- Questa istruzione vuole dire:
 - se la MT ha stato interno q_n
 - e se legge sul nastro il simbolo s_k
 - allora scrive sul nastro il simbolo s_j
 - esegue il movimento R
 - e passa allo stato interno q_m

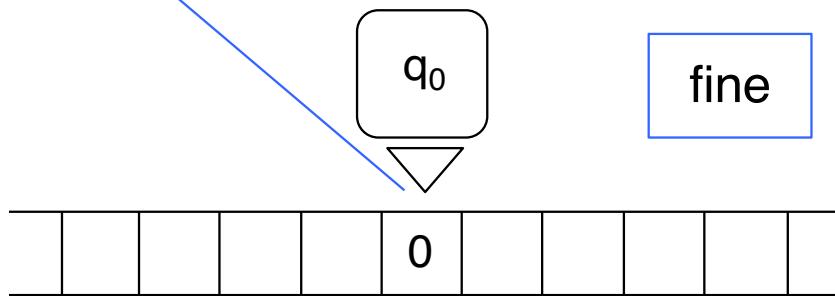


Esercizio 1

- Creare una macchina di Turing (MT) che accetti il seguente linguaggio:
 0^i , con i intero pari

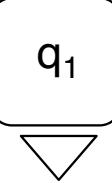


Non esiste una posizione finale standard. La testina può trovarsi ovunque. L'importante è l'output lasciato sul nastro.



fine

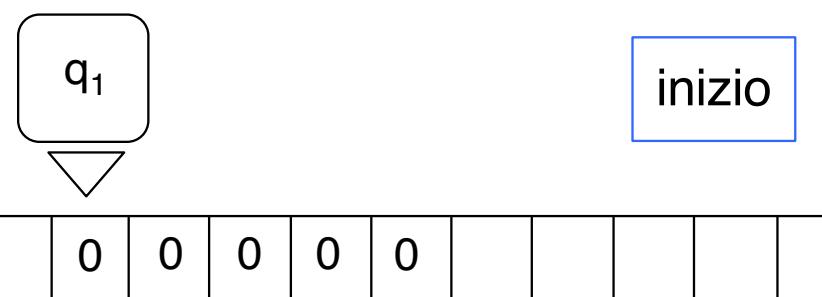
0



inizio

0 1 \$ ^ __

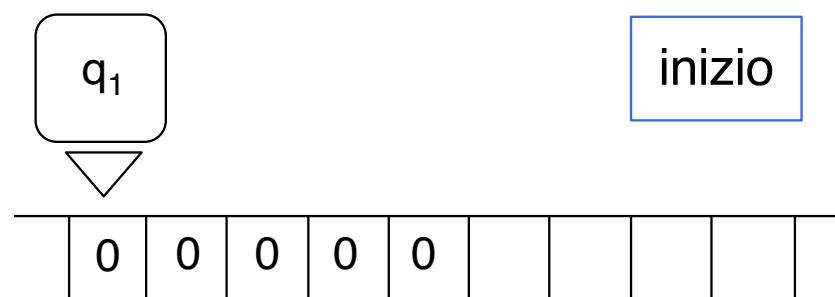
Quali input deve aspettarsi la MT?



inizio

0 0 0 0 0

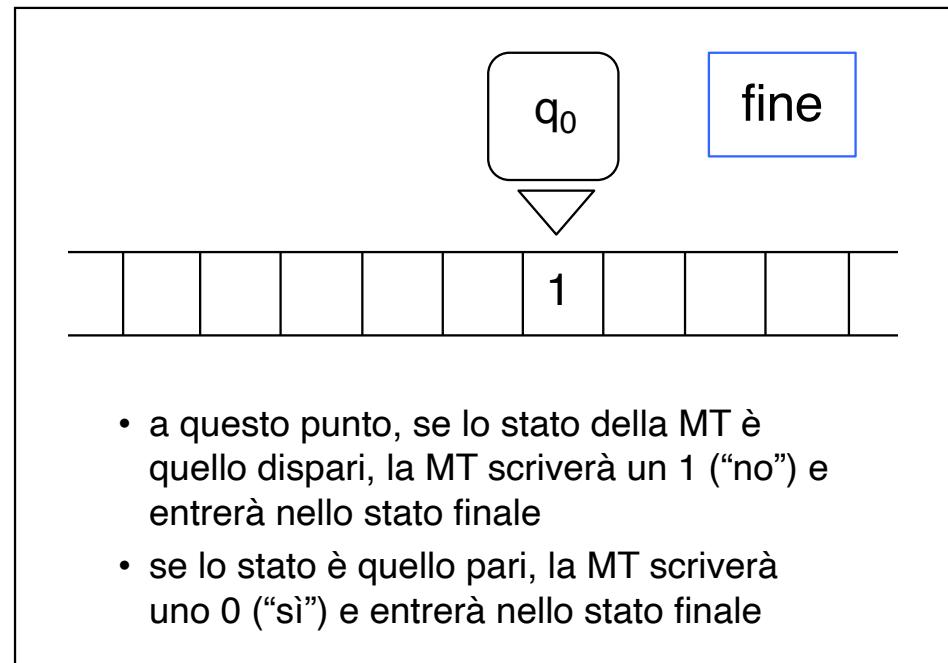
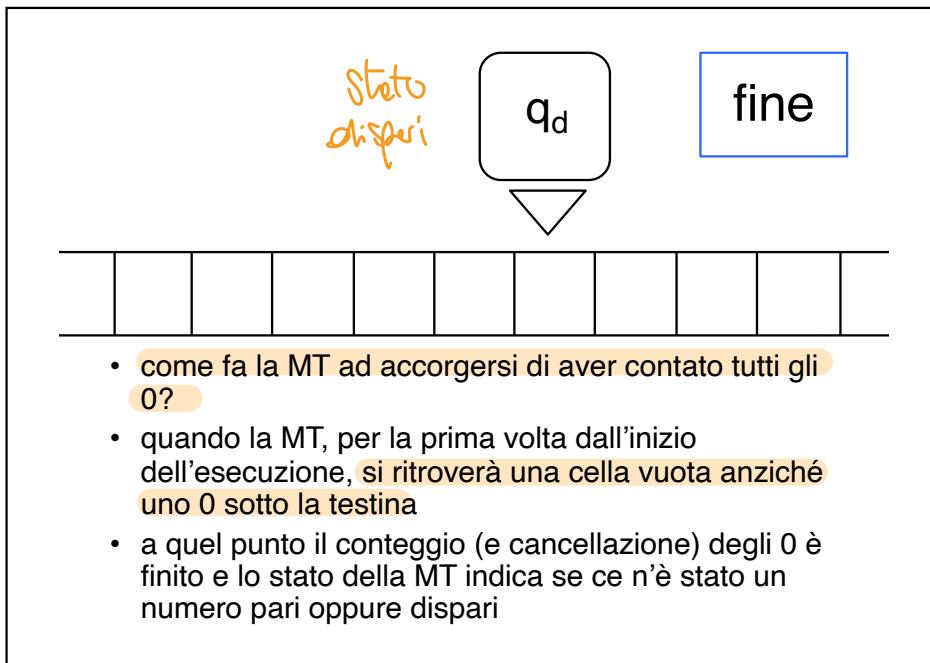
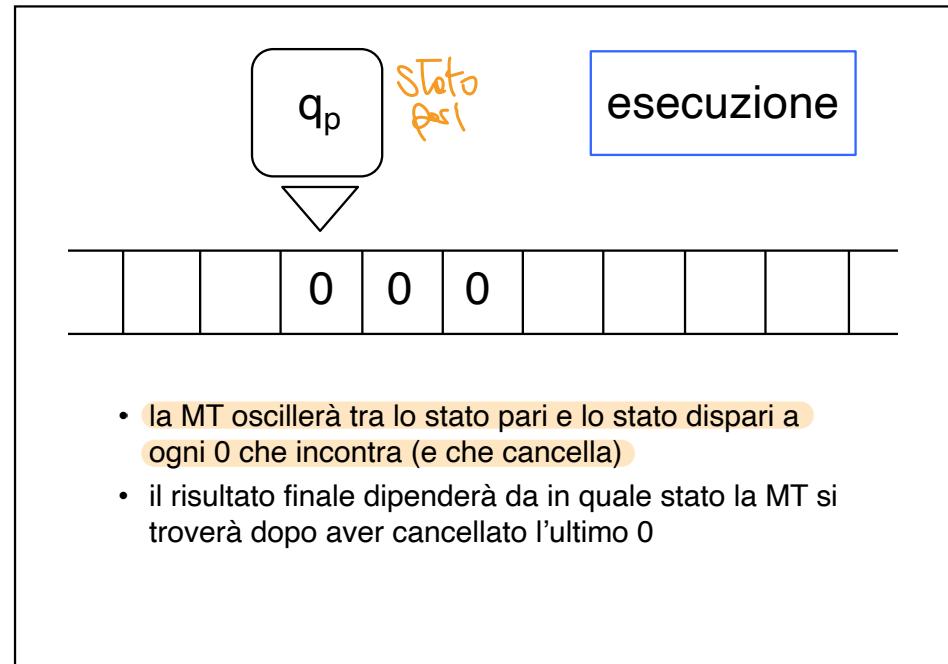
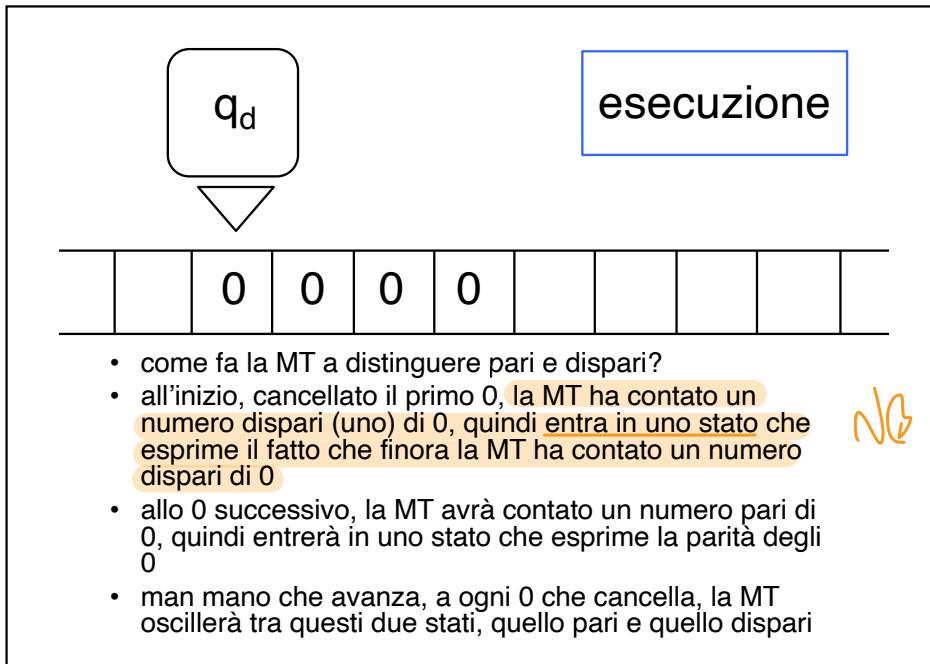
Si tratta di un esercizio teorico, non di testing di software nel mondo reale, quindi va bene ipotizzare che ci siano solo i simboli dell'alfabeto specificato nell'esercizio (e naturalmente celle vuote: [])).



inizio

0 0 0 0 0

- la MT deve capire se gli 0 sul nastro sono pari o dispari
- visto che l'input deve essere cancellato, ogni volta che la MT incontra uno 0, lo cancella e va a destra



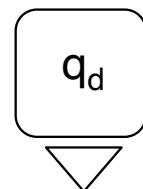
Scrittura del programma

- Ora esprimiamo queste diverse configurazioni e azioni della MT in termini di istruzioni del suo programma
- Le istruzioni hanno la struttura:
 - stato attuale
 - simbolo letto
 - simbolo scritto
 - movimento
 - nuovo stato

$q_{in} \ S_k \ S_j \ R \ q_{in}$

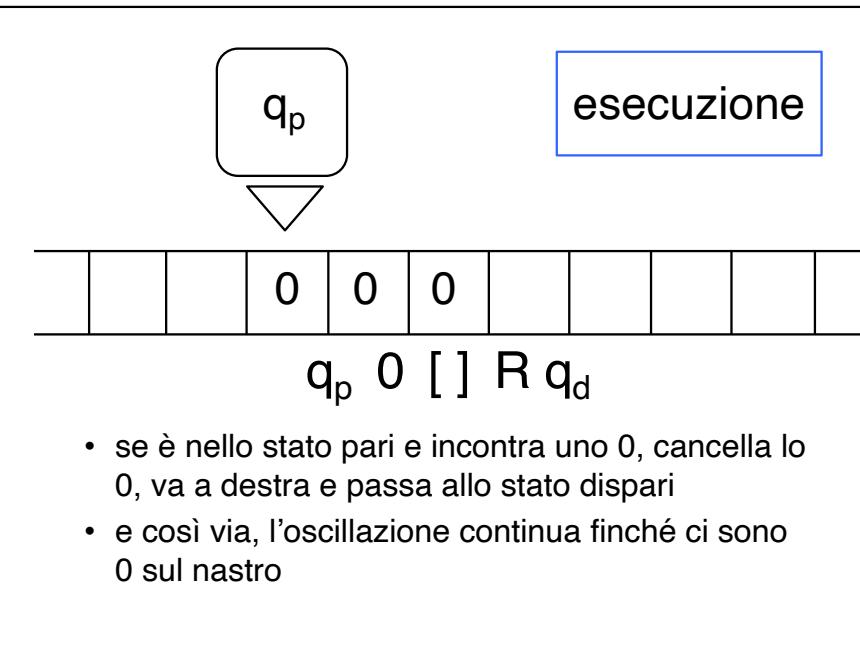


- nello stato iniziale, la MT legge uno 0,, cancella lo 0, va a destra e va nello stato dispari

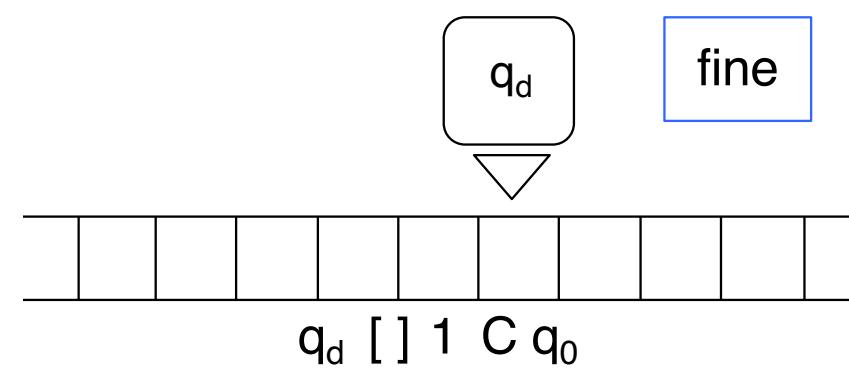


esecuzione

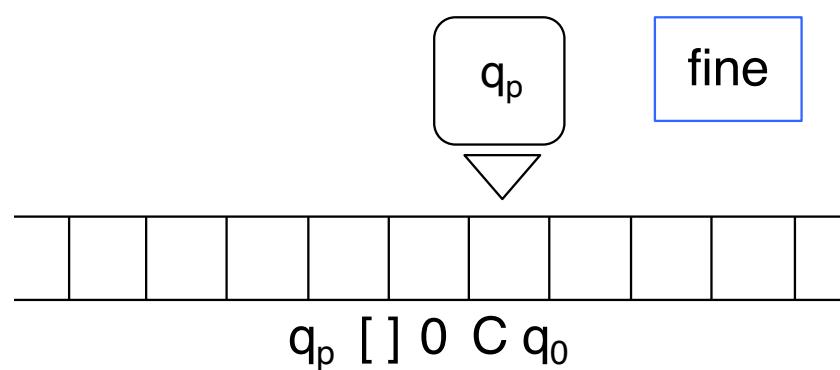
- se è nello stato dispari e incontra uno 0, cancella lo 0, va a destra e la MT passa allo stato pari



- se è nello stato pari e incontra uno 0, cancella lo 0, va a destra e passa allo stato dispari
- e così via, l'oscillazione continua finché ci sono 0 sul nastro



- se gli 0 finiscono e la MT è nello stato dispari, scrive un 1 e entra nello stato finale, ossia termina
- a questo punto il movimento non è importante, ma scegliamo di non farle cambiare posizione (C)



- se gli 0 finiscono e la MT è nello stato pari, scrive uno 0 per accettare l'input come appartenente al linguaggio, e termina

Il programma della MT

$q_0 \ 0 \ [] R \ q_d$
 $q_d \ 0 \ [] R \ q_p$
 $q_p \ 0 \ [] R \ q_d$
 $q_d \ [] 1 \ C \ q_1$
 $q_p \ [] 0 \ C \ q_1$

Condizioni al contorno

- È sempre utile farsi domande sulle condizioni al contorno, ovvero sia i casi estremi
- Come si deve comportare la MT se fin da subito non trova alcuno 0?
- Se considerate “zero” un numero pari, allora se non ci sono zeri, cioè c’è la stringa vuota sul nastro, la MT dovrà scrivere uno 0 di accettazione

condizione
limite

Aggiunta di istruzione

$q_0 \ 0 \ [] \ R \ q_d$

$q_d \ 0 \ [] \ R \ q_p$

$q_p \ 0 \ [] \ R \ q_d$

$q_d \ [] \ 1 \ C \ q_1$

$q_p \ [] \ 0 \ C \ q_1$

$q_0 \ [] \ 0 \ C \ q_1$

Caratteristiche di un programma di una MT

- La nuova istruzione può essere aggiunta in qualunque posizione
- L'ordine con cui sono scritte le istruzioni non conta
- Viene comunque eseguita la sola istruzione attivata dall'attuale configurazione della MT
- L'istruzione attivata deve essere unica, altrimenti si perde il determinismo della MT

Determinismo di una MT

$q_0 \ 0 \ [] \ R \ q_d$

$q_d \ 0 \ [] \ R \ q_p$

$q_p \ 0 \ [] \ R \ q_d$

$q_d \ [] \ 1 \ C \ q_1$

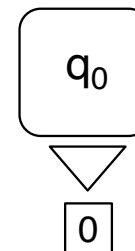
$q_p \ [] \ 0 \ C \ q_1$

$q_0 \ [] \ 0 \ C \ q_1$

tutte le configurazioni devono essere diverse tra loro

Determinismo di una MT

??



$q_0 \ 0 \ [] \ R \ q_d$

$q_d \ 0 \ [] \ R \ q_p$

$q_p \ 0 \ [] \ R \ q_d$

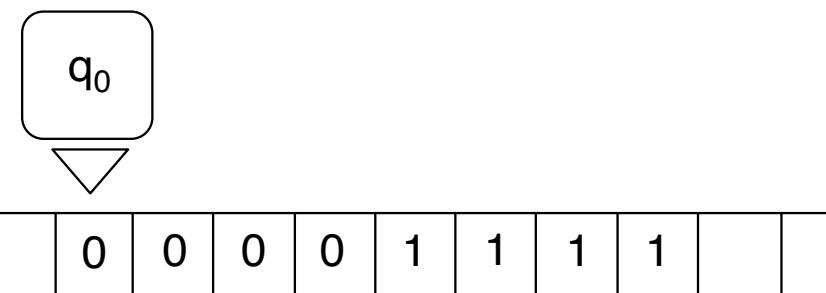
$q_d \ [] \ 1 \ C \ q_1$

$q_p \ [] \ 0 \ C \ q_1$

$q_0 \ [] \ 0 \ C \ q_1$

Esercizio 2

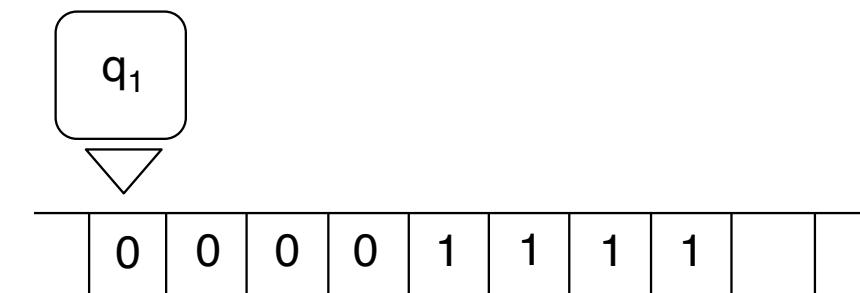
- Creare una macchina di Turing (MT) che accetti il seguente linguaggio:
 $0^i 1^i$, con $i \geq 0$



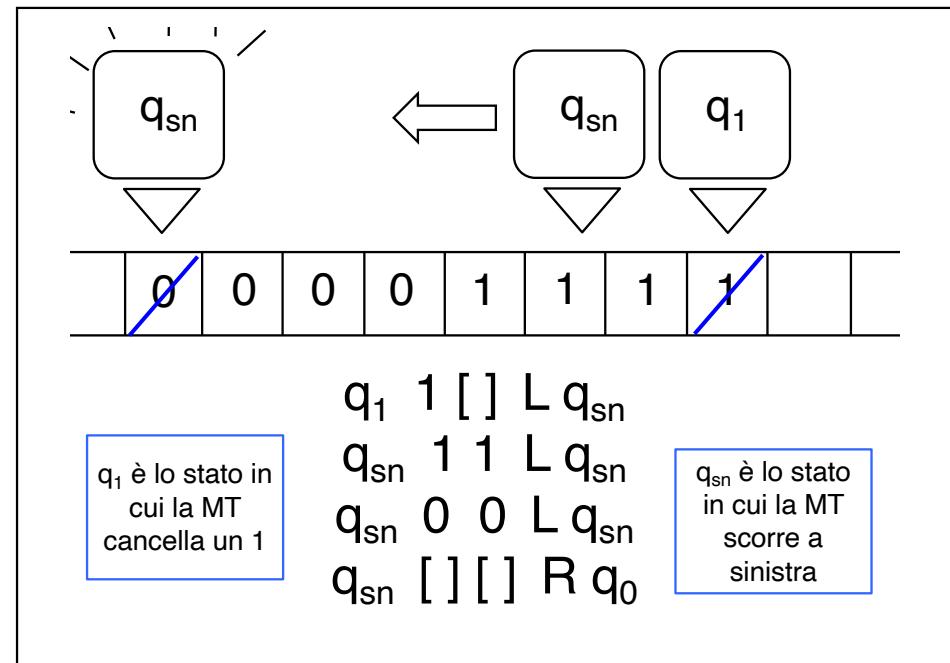
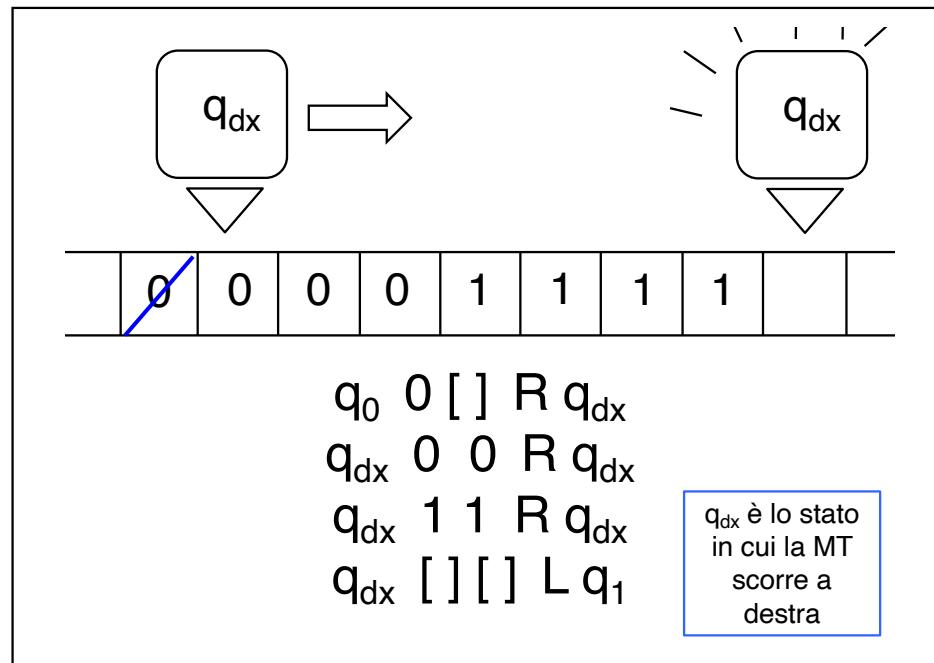
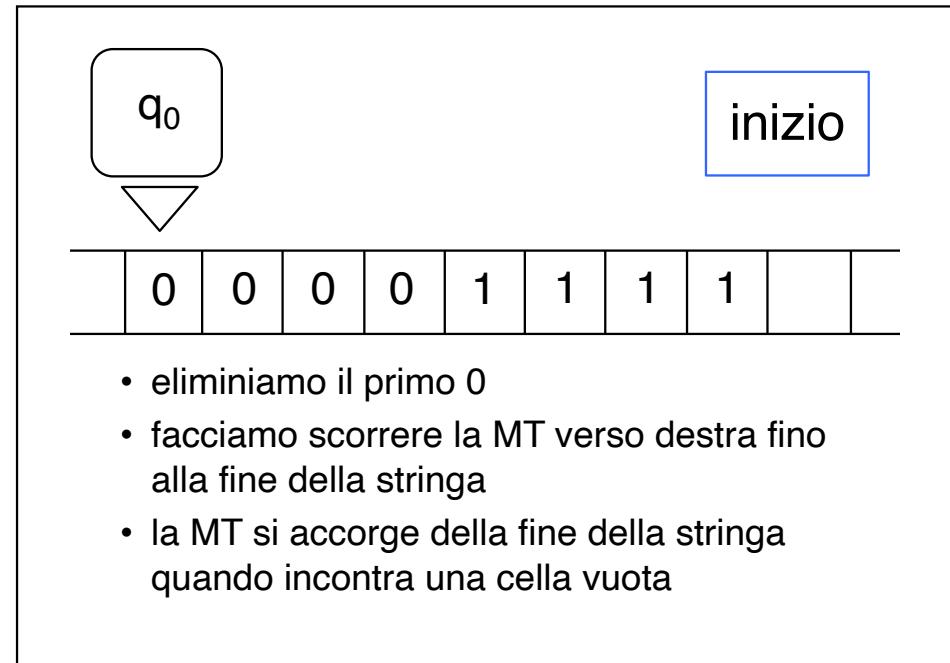
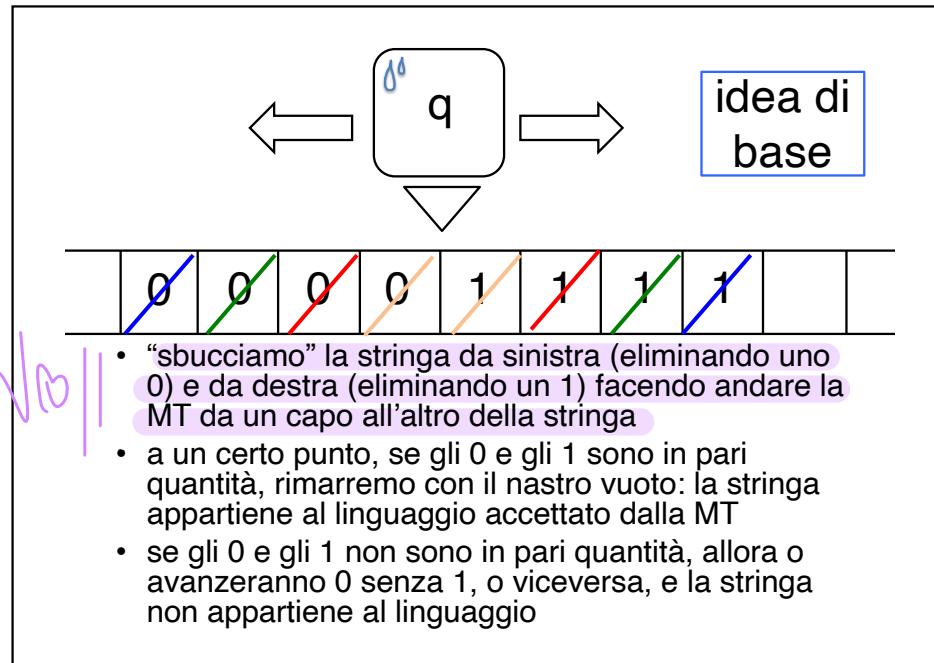
- un'idea che sorge spontanea è contare quanti 0 ci sono, contare quanti 1 ci sono e vedere se le due quantità coincidono
- purtroppo le MT NON sono in grado di contare perché non hanno una memoria interna (se non quella contenente il loro programma)

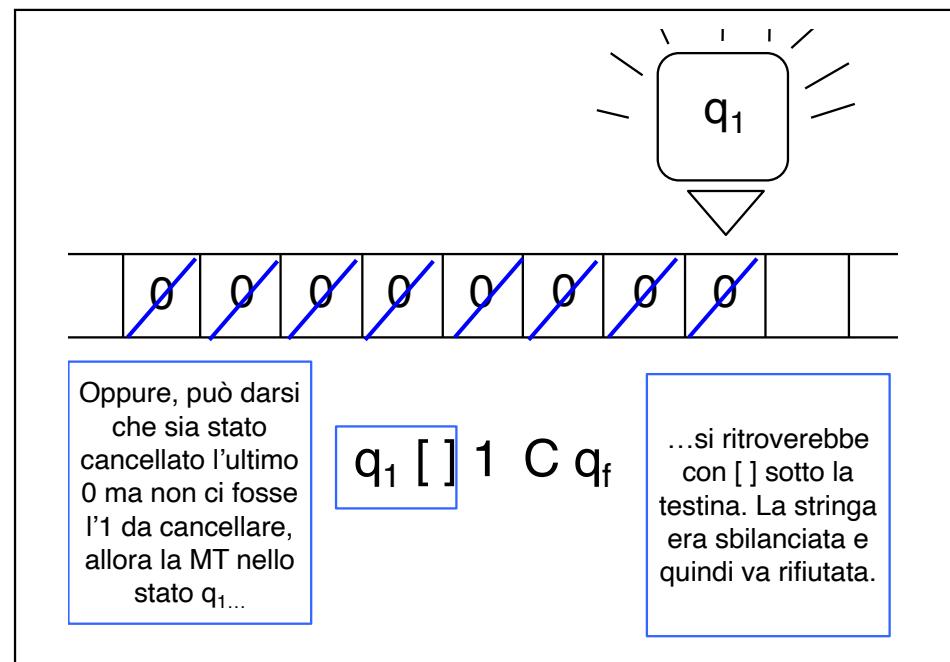
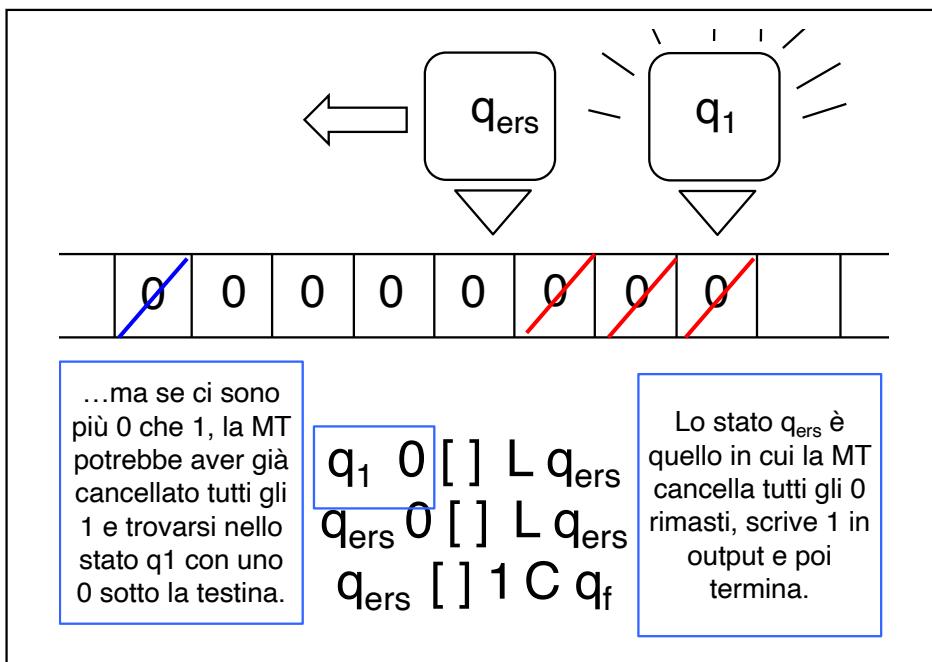
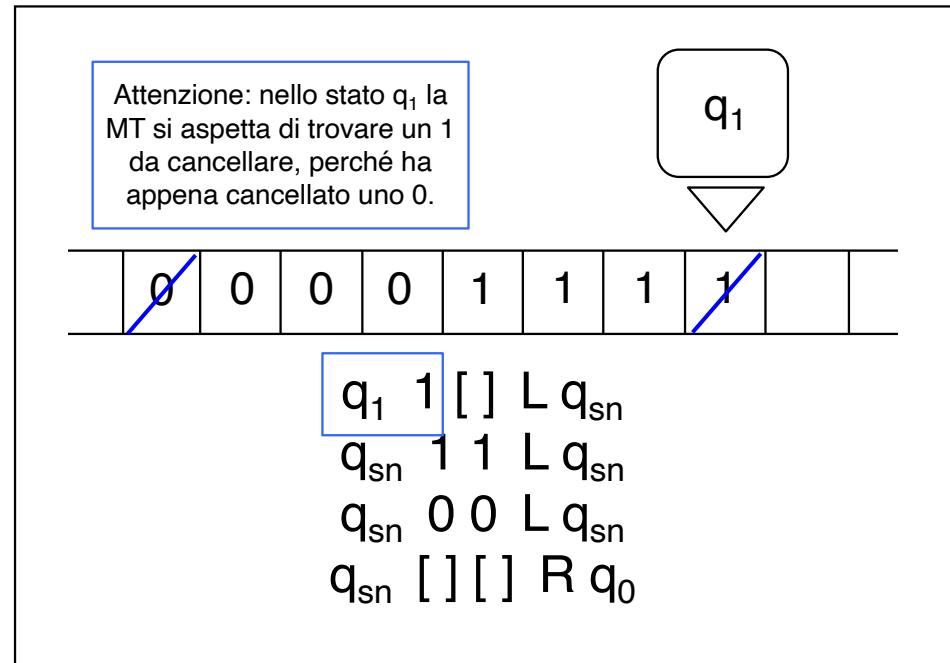
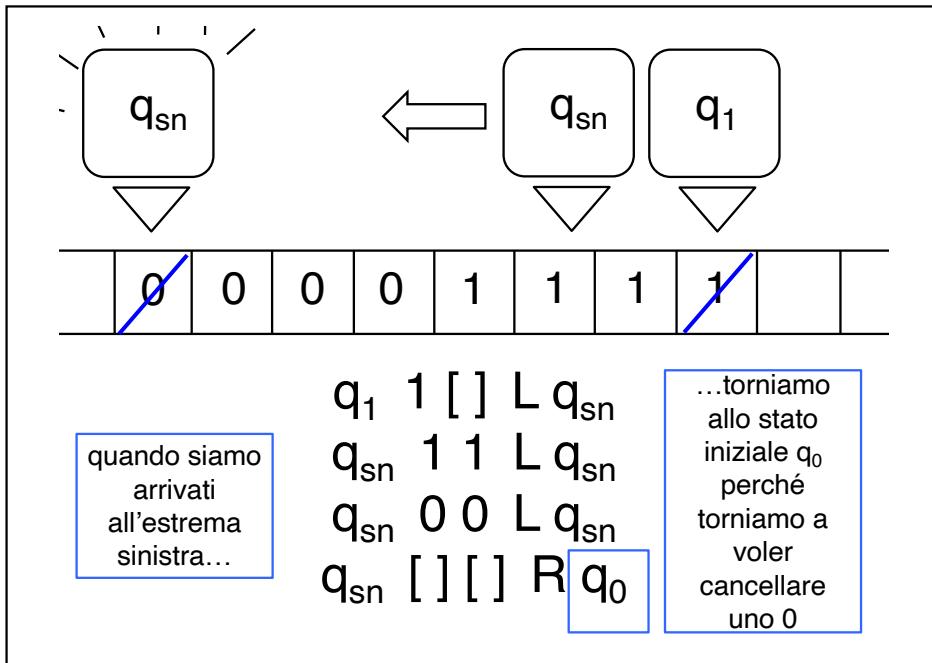
Il linguaggio

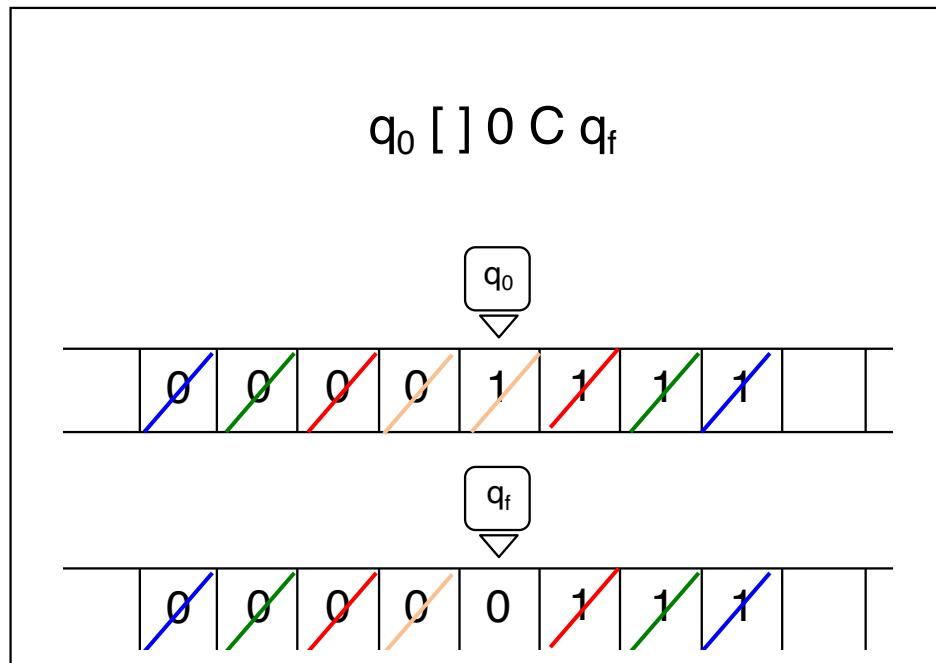
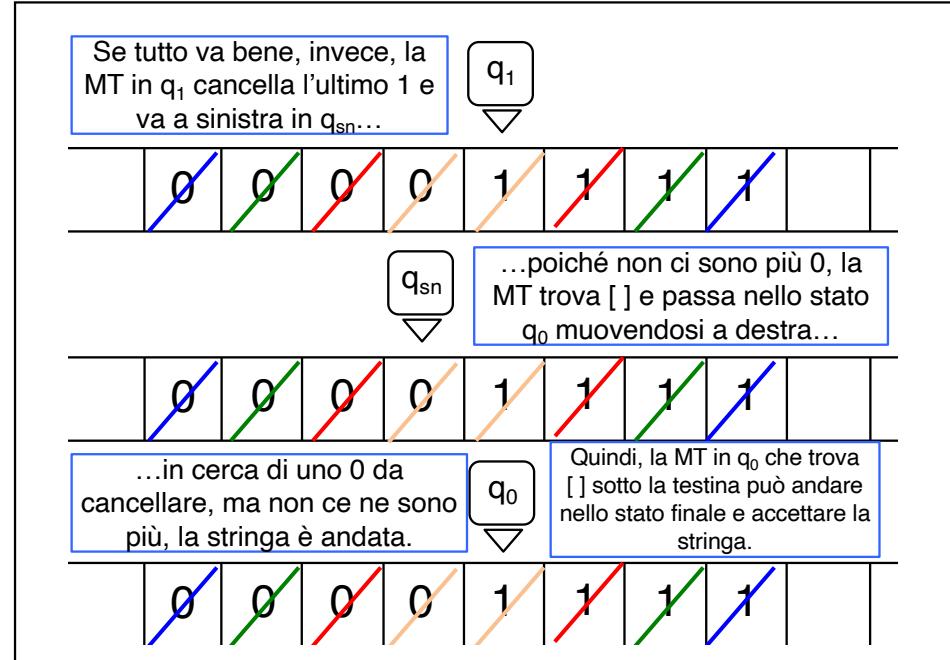
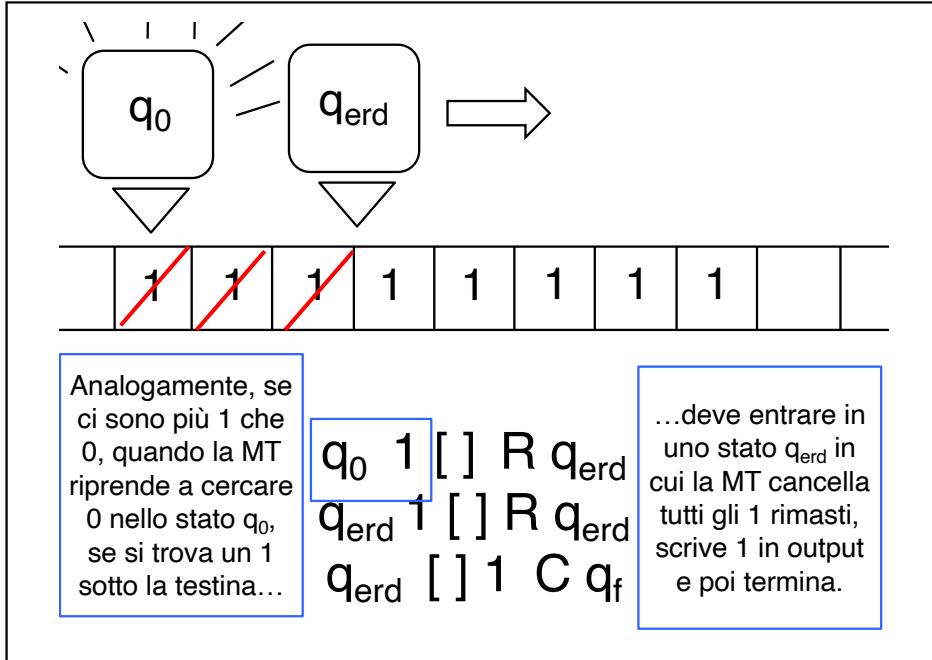
- Il linguaggio:
 $0^i 1^i$, con $i \geq 0$
è costituito da stringhe che hanno 0 a sinistra e 1 a destra nella stessa quantità
- 00001111 fa parte del linguaggio
- 0001111 no
- per semplicità, ipotizziamo che in input arrivino stringhe $0^i 1^j$, e programmiamo la nostra MT per controllare se $i=j$ o meno
- ciò vuol dire che non dobbiamo preoccuparci di input come 0000101111



- non possiamo, come abbiamo fatto nell'esercizio precedente, usare gli stati interni della macchina per tenere traccia della situazione
- prima, qualunque fosse la lunghezza della stringa, avevamo bisogno solo di due stati (pari, dispari)
- ora, se volessimo contare gli 0, avremmo bisogno di tanti stati quanti 0 arrivano in input
- questa soluzione non è possibile perché il programma è fisso, e non può adattarsi alla dimensione dell'input in arrivo







Aggreghiamo le istruzioni

$q_0 \ 0 \ [\] R \ q_{dx}$	$q_1 \ [\] 1 \ C \ q_f$
$q_{dx} \ 0 \ 0 \ R \ q_{dx}$	$q_0 \ 1 \ [\] R \ q_{erd}$
$q_{dx} \ 1 \ 1 \ R \ q_{dx}$	$q_{erd} \ 1 \ [\] R \ q_{erd}$
$q_{dx} \ [\] [\] L \ q_1$	$q_{erd} \ [\] 1 \ C \ q_f$
$q_1 \ 1 \ [\] L \ q_{sn}$	$q_0 \ [\] 0 \ C \ q_f$
$q_{sn} \ 1 \ 1 \ L \ q_{sn}$	
$q_{sn} \ 0 \ 0 \ L \ q_{sn}$	
$q_{sn} \ [\] [\] R \ q_0$	
$q_1 \ 0 \ [\] L \ q_{ers}$	
$q_{ers} \ 0 \ [\] L \ q_{ers}$	
$q_{ers} \ [\] 1 \ C \ q_f$	

Elimina lo 0 e scorri a destra

$q_0 \ 0 [] R q_{dx}$
 $q_{dx} \ 0 \ 0 \ R q_{dx}$
 $q_{dx} \ 1 \ 1 \ R q_{dx}$
 $q_{dx} \ [] [] L q_1$
 $q_1 \ 1 [] L q_{sn}$
 $q_{sn} \ 1 \ 1 \ L q_{sn}$
 $q_{sn} \ 0 \ 0 \ L q_{sn}$
 $q_{sn} \ [] [] R q_0$
 $q_1 \ 0 [] L q_{ers}$
 $q_{ers} \ 0 [] L q_{ers}$
 $q_{ers} \ [] 1 C q_f$

$q_1 \ [] 1 C q_f$
 $q_0 \ 1 [] R q_{erd}$
 $q_{erd} \ 1 [] R q_{erd}$
 $q_{erd} \ [] 1 C q_f$
 $q_0 \ [] 0 C q_f$

Arrivato all'estrema destra,
cancella l'1 e vai a sinistra

$q_0 \ 0 [] R q_{dx}$
 $q_{dx} \ 0 \ 0 \ R q_{dx}$
 $q_{dx} \ 1 \ 1 \ R q_{dx}$
 $q_{dx} \ [] [] L q_1$
 $q_1 \ 1 [] L q_{sn}$
 $q_{sn} \ 1 \ 1 \ L q_{sn}$
 $q_{sn} \ 0 \ 0 \ L q_{sn}$
 $q_{sn} \ [] [] R q_0$
 $q_1 \ 0 [] L q_{ers}$
 $q_{ers} \ 0 [] L q_{ers}$
 $q_{ers} \ [] 1 C q_f$

$q_1 \ [] 1 C q_f$
 $q_0 \ 1 [] R q_{erd}$
 $q_{erd} \ 1 [] R q_{erd}$
 $q_{erd} \ [] 1 C q_f$
 $q_0 \ [] 0 C q_f$

Arrivato all'estrema sinistra, torna a
destra e riparti dall'inizio

$q_0 \ 0 [] R q_{dx}$
 $q_{dx} \ 0 \ 0 \ R q_{dx}$
 $q_{dx} \ 1 \ 1 \ R q_{dx}$
 $q_{dx} \ [] [] L q_1$
 $q_1 \ 1 [] L q_{sn}$
 $q_{sn} \ 1 \ 1 \ L q_{sn}$
 $q_{sn} \ 0 \ 0 \ L q_{sn}$
 $q_{sn} \ [] [] R q_0$
 $q_1 \ 0 [] L q_{ers}$
 $q_{ers} \ 0 [] L q_{ers}$
 $q_{ers} \ [] 1 C q_f$

$q_1 \ [] 1 C q_f$
 $q_0 \ 1 [] R q_{erd}$
 $q_{erd} \ 1 [] R q_{erd}$
 $q_{erd} \ [] 1 C q_f$
 $q_0 \ [] 0 C q_f$

Ci sono più 0 che 1, non va bene

$q_0 \ 0 [] R q_{dx}$
 $q_{dx} \ 0 \ 0 \ R q_{dx}$
 $q_{dx} \ 1 \ 1 \ R q_{dx}$
 $q_{dx} \ [] [] L q_1$
 $q_1 \ 1 [] L q_{sn}$
 $q_{sn} \ 1 \ 1 \ L q_{sn}$
 $q_{sn} \ 0 \ 0 \ L q_{sn}$
 $q_{sn} \ [] [] R q_0$
 $q_1 \ 0 [] L q_{ers}$
 $q_{ers} \ 0 [] L q_{ers}$
 $q_{ers} \ [] 1 C q_f$

$q_1 \ [] 1 C q_f$
 $q_0 \ 1 [] R q_{erd}$
 $q_{erd} \ 1 [] R q_{erd}$
 $q_{erd} \ [] 1 C q_f$
 $q_0 \ [] 0 C q_f$

Ci sono più 1 che 0, non va bene

$q_0 \ 0 [] R q_{dx}$
 $q_{dx} \ 0 \ 0 \ R q_{dx}$
 $q_{dx} \ 1 \ 1 \ R q_{dx}$
 $q_{dx} \ [] [] L q_1$
 $q_1 \ 1 [] L q_{sn}$
 $q_{sn} \ 1 \ 1 \ L q_{sn}$
 $q_{sn} \ 0 \ 0 \ L q_{sn}$
 $q_{sn} \ [] [] R q_0$
 $q_1 \ 0 [] L q_{ers}$
 $q_{ers} \ 0 [] L q_{ers}$
 $q_{ers} \ [] 1 C q_f$

$q_1 \ [] 1 C q_f$
 $q_0 \ 1 [] R q_{erd}$
 $q_{erd} \ 1 [] R q_{erd}$
 $q_{erd} \ [] 1 C q_f$
 $q_0 \ [] 0 C q_f$

Ci sono tanti 0 quanti 1, va bene

$q_0 \ 0 [] R q_{dx}$
 $q_{dx} \ 0 \ 0 \ R q_{dx}$
 $q_{dx} \ 1 \ 1 \ R q_{dx}$
 $q_{dx} \ [] [] L q_1$
 $q_1 \ 1 [] L q_{sn}$
 $q_{sn} \ 1 \ 1 \ L q_{sn}$
 $q_{sn} \ 0 \ 0 \ L q_{sn}$
 $q_{sn} \ [] [] R q_0$
 $q_1 \ 0 [] L q_{ers}$
 $q_{ers} \ 0 [] L q_{ers}$
 $q_{ers} \ [] 1 C q_f$
 $q_1 \ [] 1 C q_f$
 $q_0 \ 1 [] R q_{erd}$
 $q_{erd} \ 1 [] R q_{erd}$
 $q_{erd} \ [] 1 C q_f$
 $q_0 \ [] 0 C q_f$

Esercizio 3

- Creare una macchina di Turing (MT) che accetti il seguente linguaggio:
 $0^i 1^i$, con $i \geq 0$
- Abbandoniamo l'ipotesi che in input arrivino stringhe $0^i 1^j$
- Modificare la MT dell'esercizio 2 in modo che possa rifiutare anche input come 0000101111, in cui gli 0 e gli 1 sono mischiati

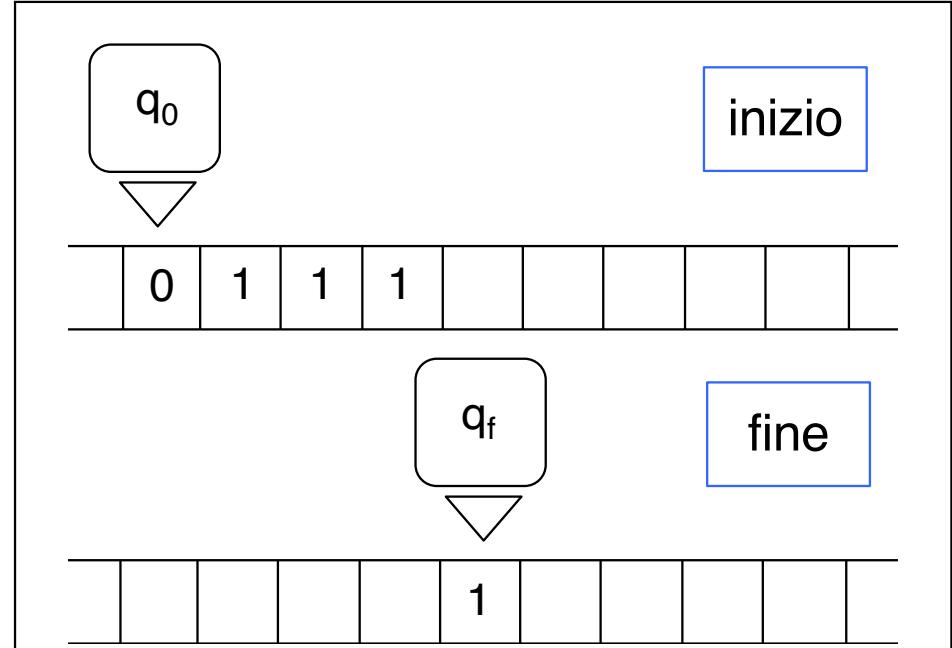
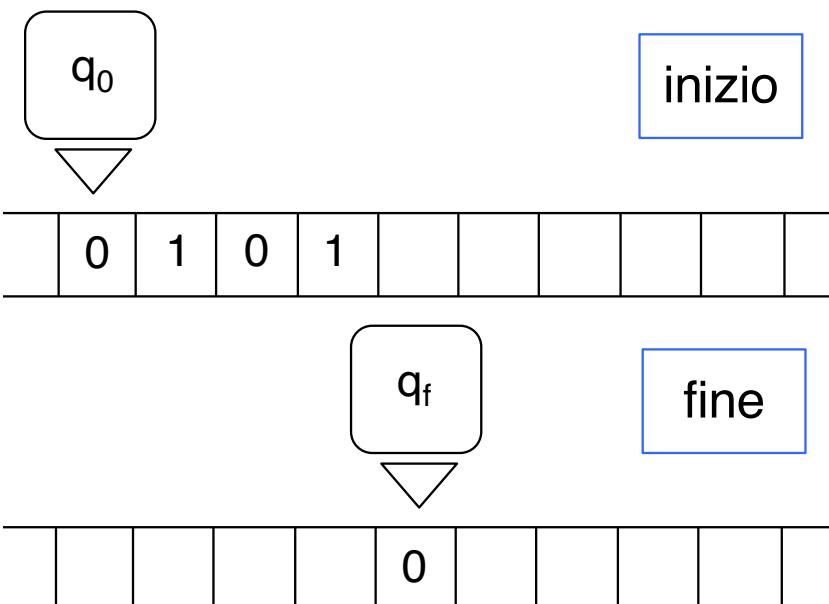
Computabilità

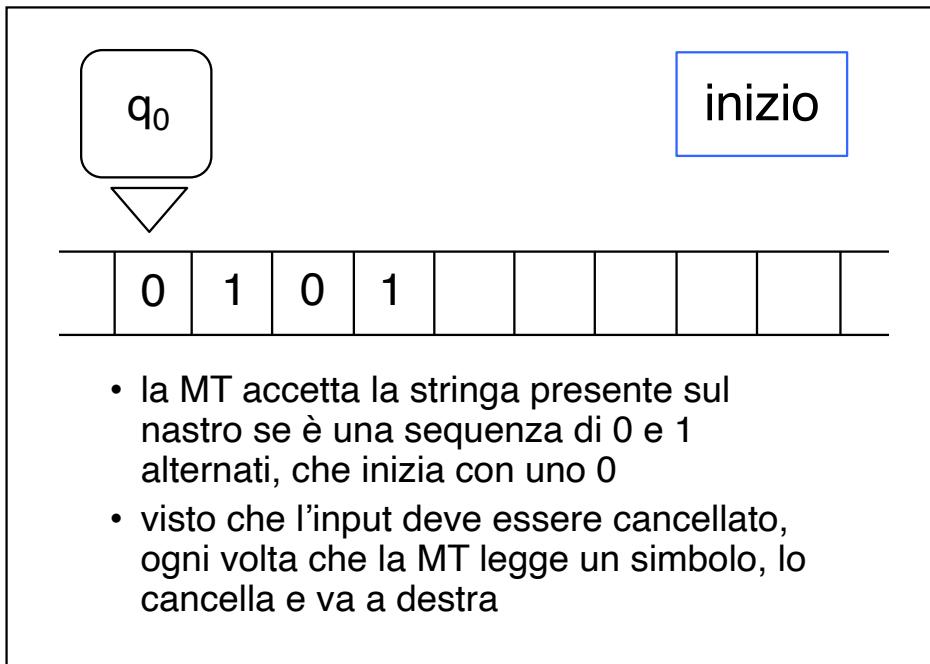
11 gennaio 2022

Mario Verdicchio
Università degli Studi di Bergamo
Anno Accademico 2021-2022

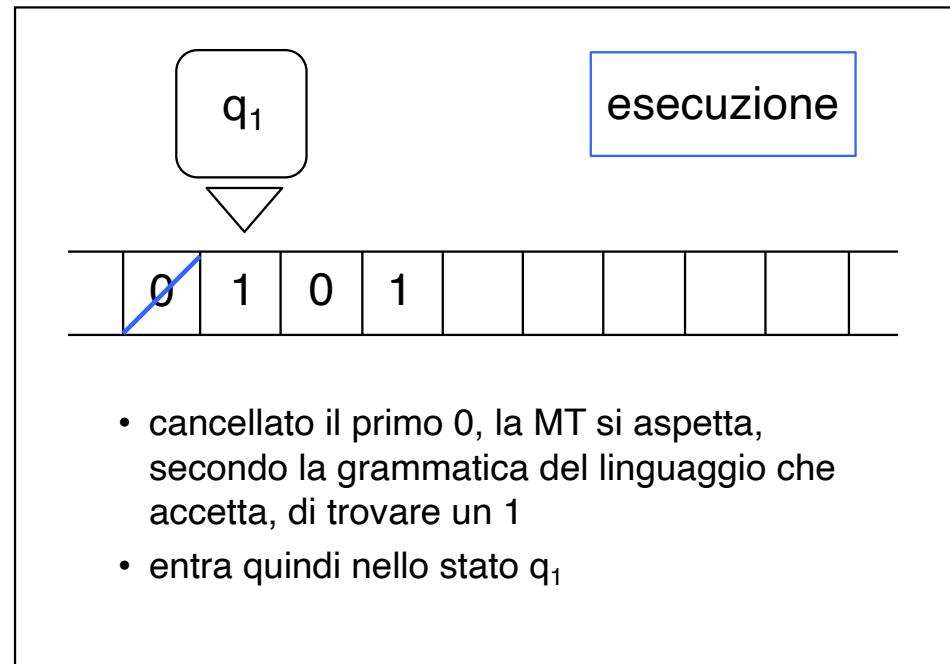
Esercizio

- Creare una macchina di Turing (MT) che accetti il seguente linguaggio:
 - $(01)^i$ (con $i > 0$)

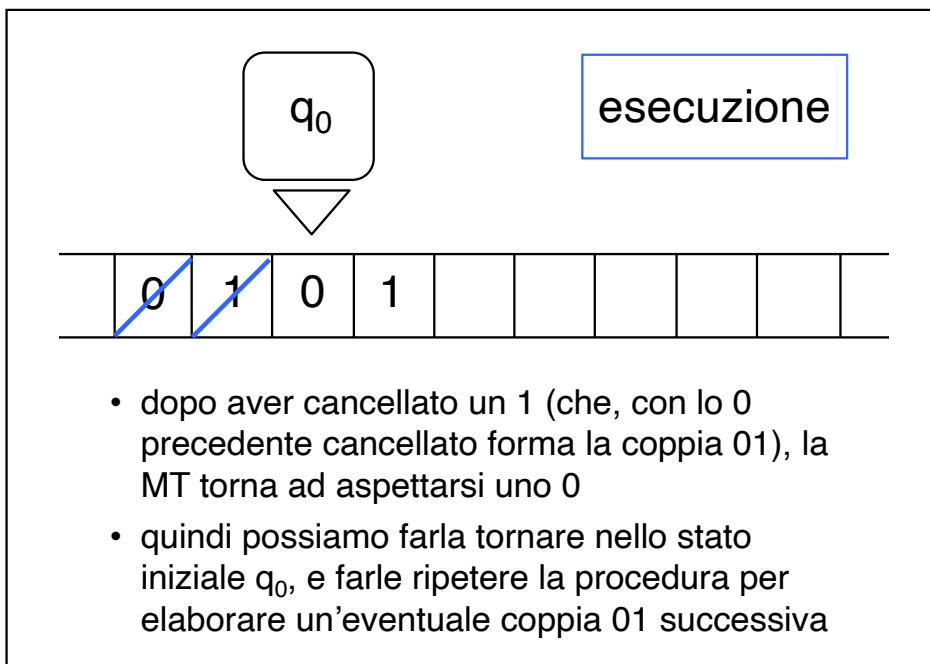




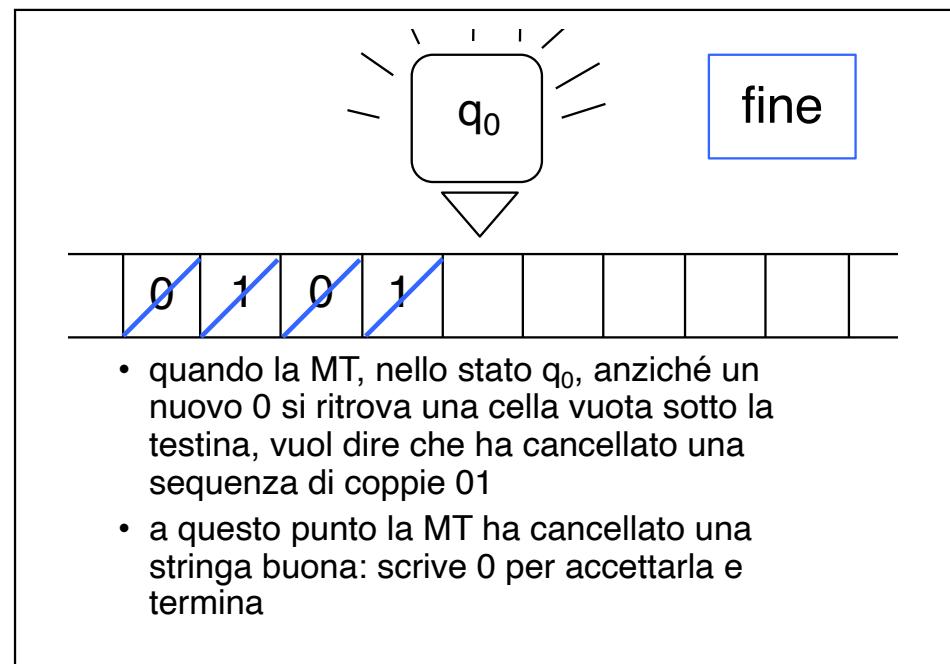
- la MT accetta la stringa presente sul nastro se è una sequenza di 0 e 1 alternati, che inizia con uno 0
- visto che l'input deve essere cancellato, ogni volta che la MT legge un simbolo, lo cancella e va a destra



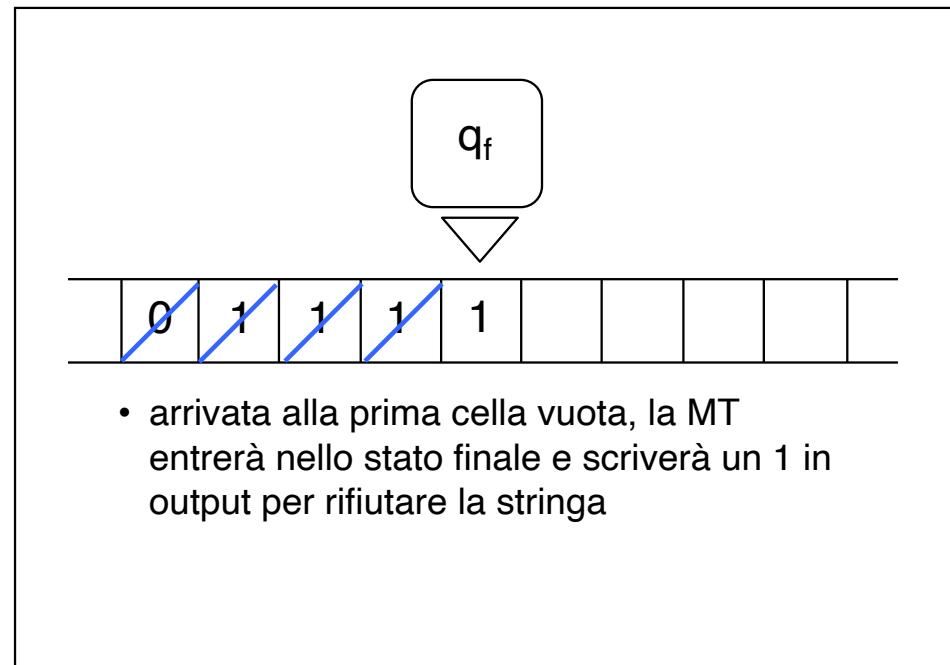
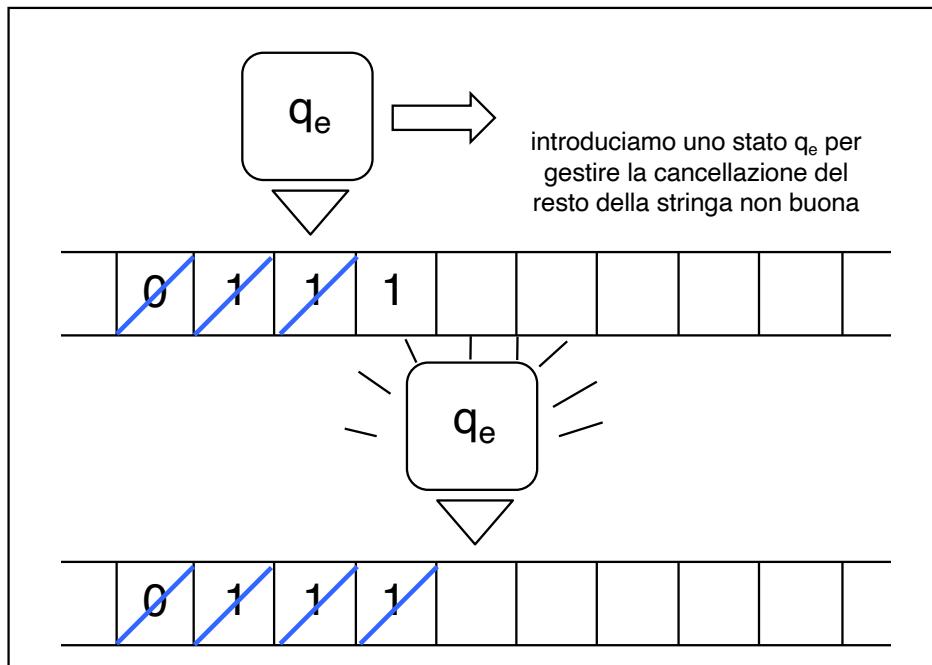
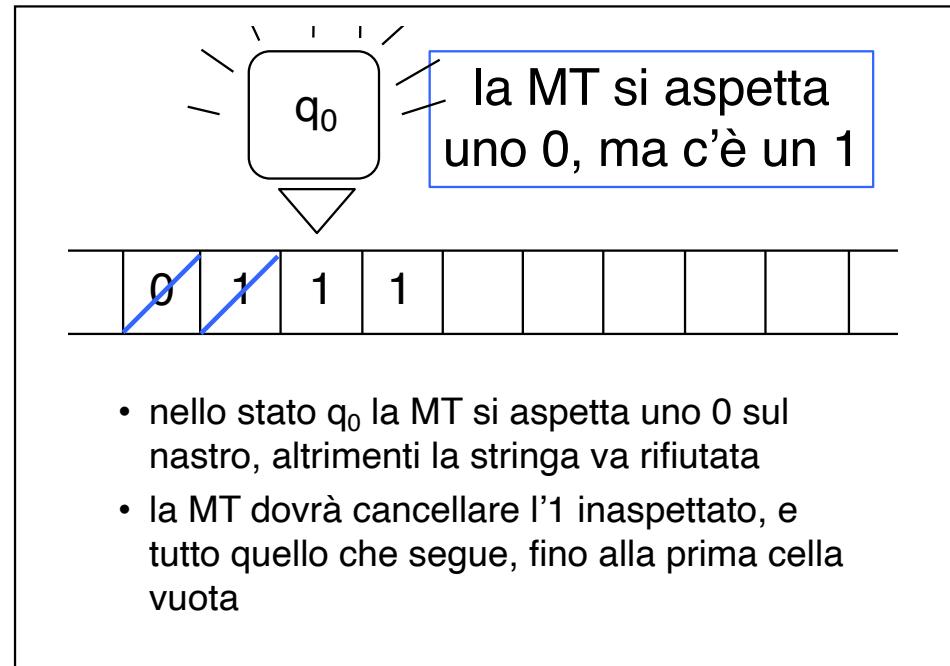
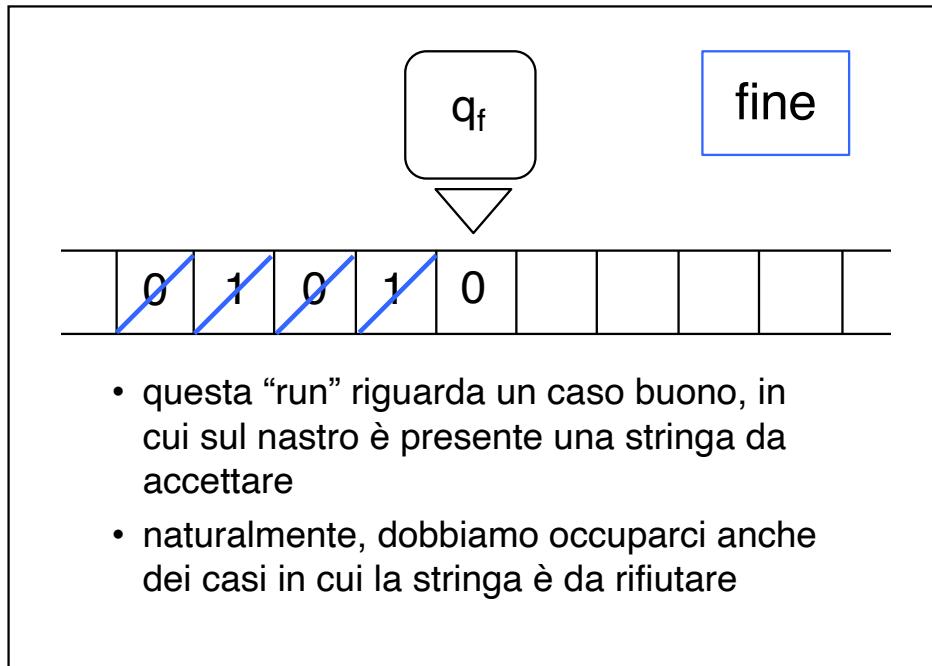
- cancellato il primo 0, la MT si aspetta, secondo la grammatica del linguaggio che accetta, di trovare un 1
- entra quindi nello stato q_1

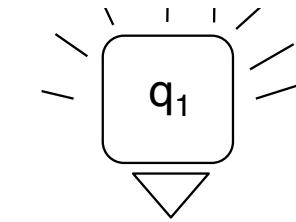


- dopo aver cancellato un 1 (che, con lo 0 precedente cancellato forma la coppia 01), la MT torna ad aspettarsi uno 0
- quindi possiamo farla tornare nello stato iniziale q_0 , e farle ripetere la procedura per elaborare un'eventuale coppia 01 successiva



- quando la MT, nello stato q_0 , anziché un nuovo 0 si ritrova una cella vuota sotto la testina, vuol dire che ha cancellato una sequenza di coppie 01
- a questo punto la MT ha cancellato una stringa buona: scrive 0 per accettarla e termina

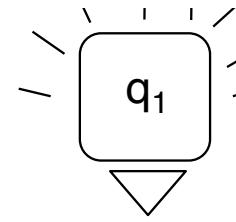




la MT si aspetta un 1, ma c'è uno 0



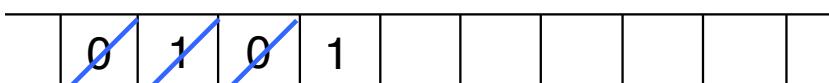
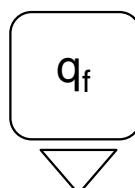
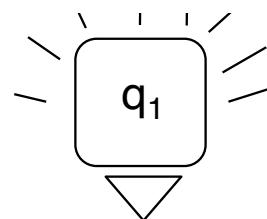
- analogamente, se la MT nello stato q_1 , ossia che si aspetta un 1 sotto la testina, trova uno 0, si procederà allo stesso modo di prima: cancellare tutto scorrendo a destra e poi terminare con la scrittura di un 1



la MT si aspetta un 1, ma non c'è nulla



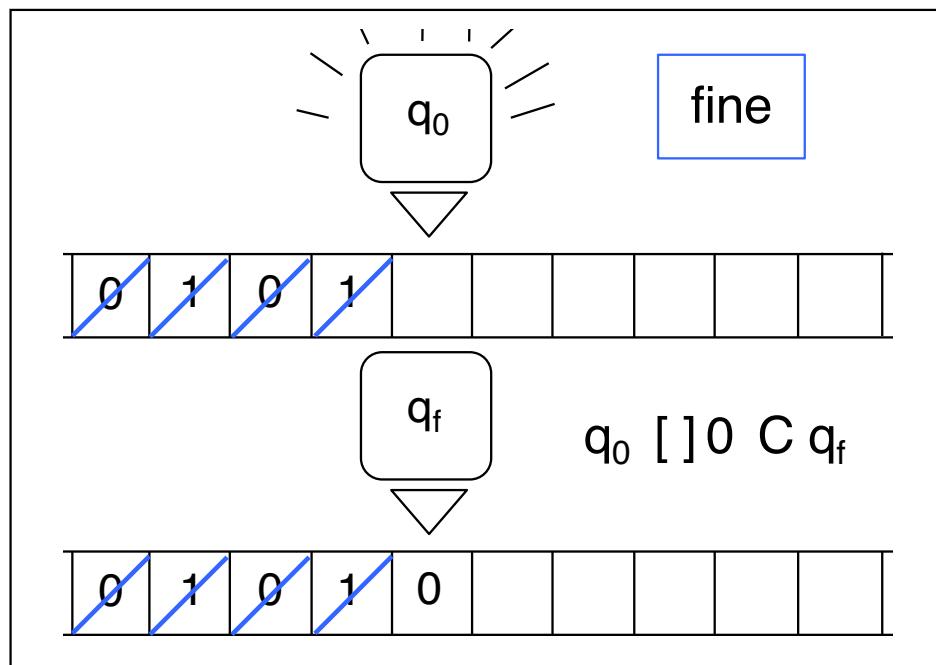
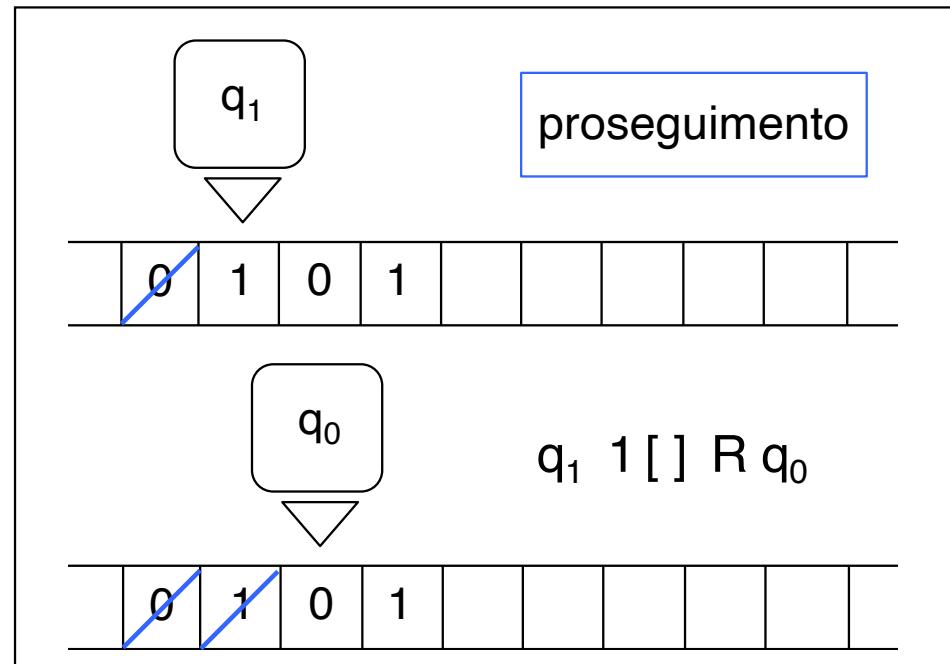
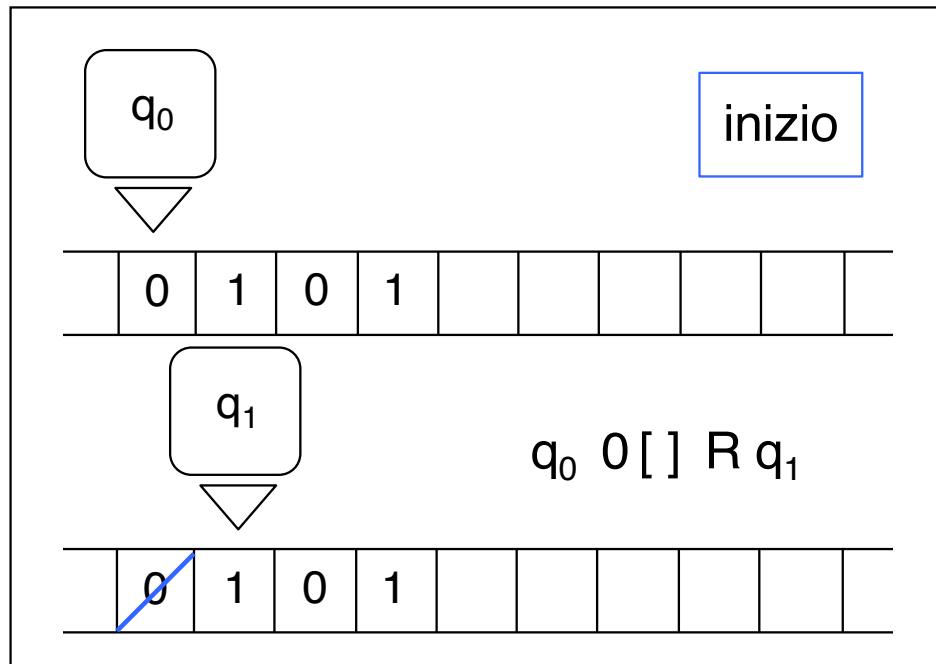
- questo è un altro possibile caso negativo: la stringa finisce con uno 0 e non c'è l'1 che lo accompagna
- in questo caso non serve cancellare ulteriormente, basta scrivere 1 e terminare



Scrittura del programma

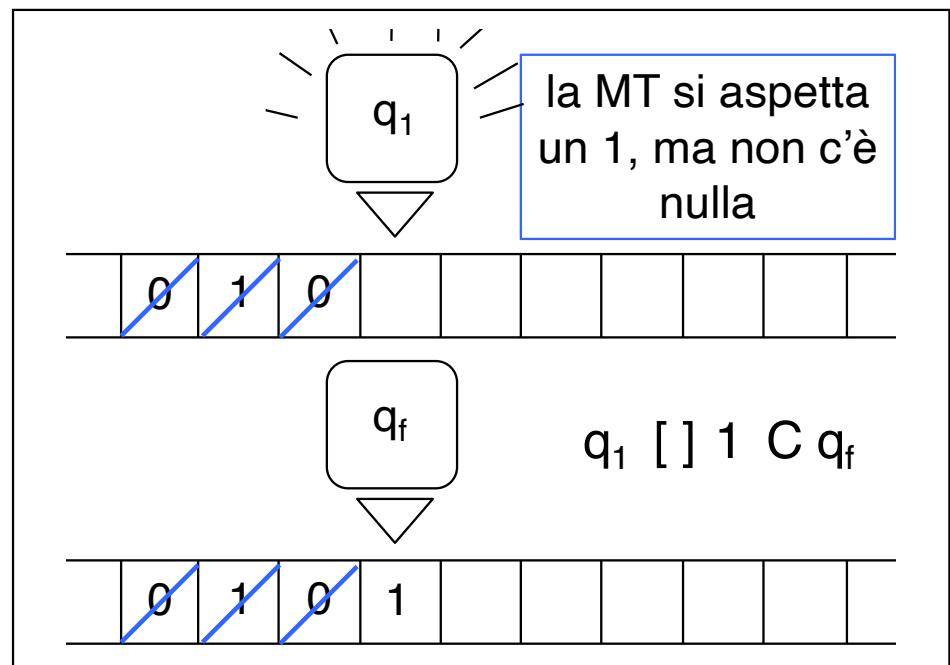
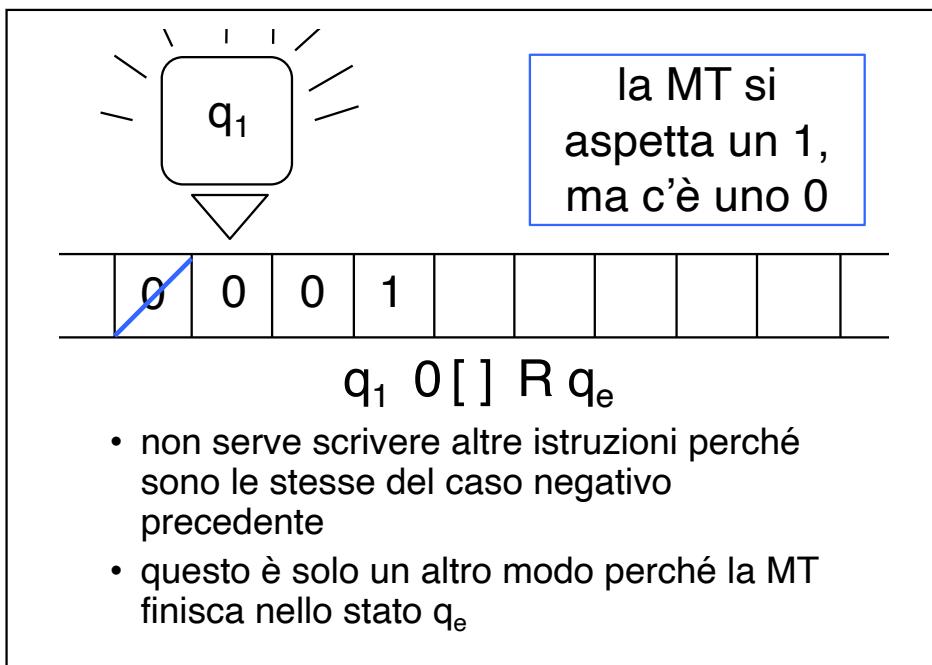
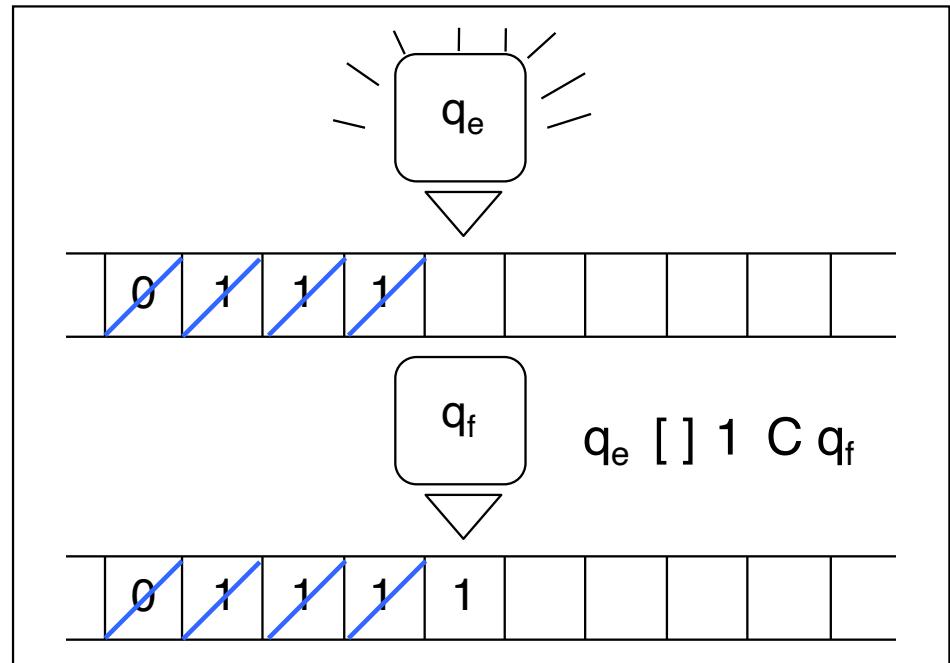
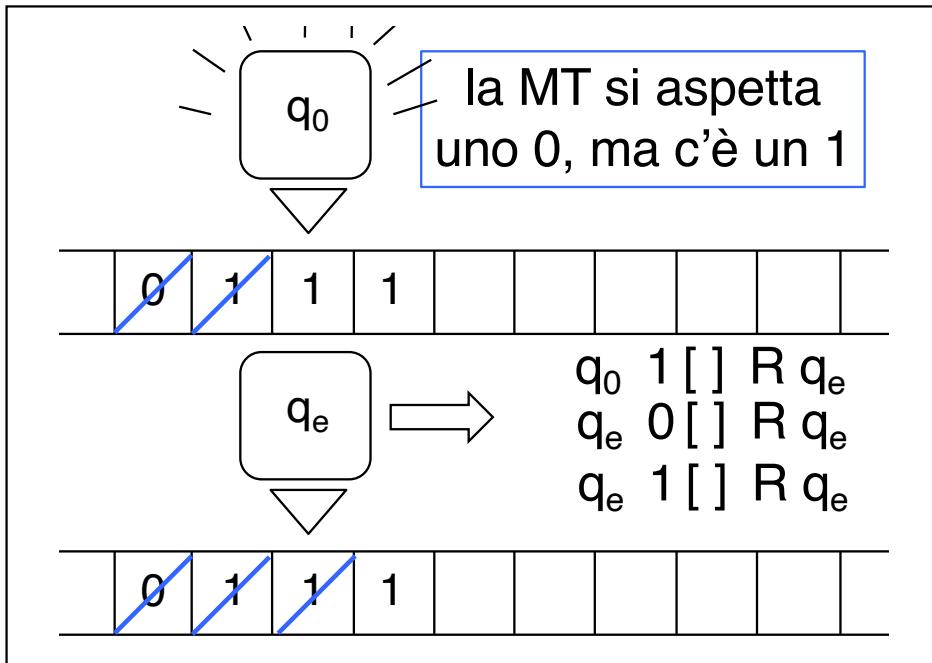
- Ora esprimiamo queste diverse configurazioni e azioni della MT in termini di istruzioni del suo programma
- Ricordiamo che le istruzioni hanno la seguente forma:

$q_{\text{attuale}} \ S_{\text{letto}} \ S_{\text{scritto}} \ M_{\text{movimento}} \ q_{\text{nuovo}}$



Istruzioni per l'accettazione
di stringhe del linguaggio

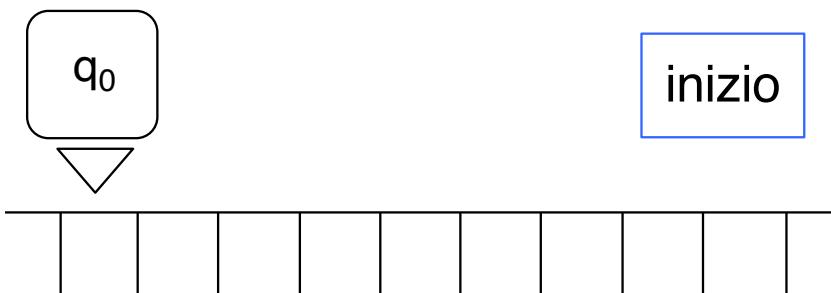
$q_0 \ 0[] R q_1$
 $q_1 \ 1[] R q_0$
 $q_0 [] 0 C q_f$



Mettiamo assieme le istruzioni

$q_0 \ 0[] \ R \ q_1$	Parsing regolare di coppie 01
$q_1 \ 1[] \ R \ q_0$	
$q_0 \ []0 \ C \ q_f$	Accettazione della stringa
$q_0 \ 1[] \ R \ q_e$	Scoperta di un 1 irregolare
$q_e \ 0[] \ R \ q_e$	Cancellazione del resto della stringa irregolare e rifiuto della stringa
$q_e \ 1[] \ R \ q_e$	
$q_e \ []1 \ C \ q_f$	Scoperta di uno 0 irregolare
$q_1 \ 0[] \ R \ q_e$	
$q_1 \ []1 \ C \ q_f$	Stringa terminata irregolarmente

Come si comporta la MT se fin da subito non trova niente sul nastro?



- Apparentemente non ci siamo occupati di questo caso, perché lo prendiamo in considerazione per la prima volta ora
- In realtà, c'è già un'istruzione che viene attivata da questa configurazione

Condizioni al contorno

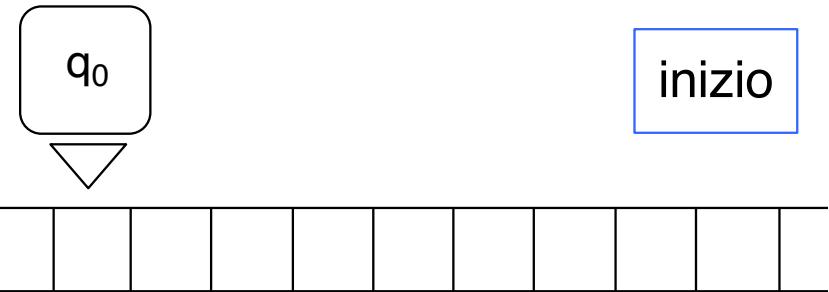
- Interroghiamoci sulle condizioni al contorno, ovvero i casi estremi
- Ecco due domande importanti, simili ma distinte:
 - Come si deve comportare la MT se fin da subito non trova niente sul nastro?
 - Come si comporta la MT se fin da subito non trova niente sul nastro?

$q_0 \ 0[] \ R \ q_1$
$q_1 \ 1[] \ R \ q_0$
$q_0 \ []0 \ C \ q_f$
$q_0 \ 1[] \ R \ q_e$
$q_e \ 0[] \ R \ q_e$
$q_e \ 1[] \ R \ q_e$
$q_e \ []1 \ C \ q_f$
$q_1 \ 0[] \ R \ q_e$
$q_1 \ []1 \ C \ q_f$

Accettazione della stringa

Abbiamo scritto questa istruzione per la situazione in cui, dopo aver cancellato un certo numero di coppie 01, la MT si ritrova con una cella vuota sotto la testina, e quindi può accettare la stringa appena cancellata.

Come si comporta la MT se fin da subito non trova niente sul nastro?



- Quindi, in questa situazione, la MT, con questo programma, accetterebbe la stringa vuota, scrivendo uno 0 sul nastro e terminando

Bene così?

No.

- Creare una macchina di Turing (MT) che accetti il seguente linguaggio:
 - $-(01)^i$ (con $i > 0$)
- Secondo la specifica del linguaggio, le stringhe accettate devono contenere almeno una coppia 01
- Ossia, la stringa vuota deve essere rifiutata

Come fare?

$q_0 \ 0 [] \ R \ q_1$ _____
 $q_1 \ 1 [] \ R \ q_0$ _____
 $q_0 \ [] \ 0 \ C \ q_f$ _____
 $q_0 \ 1 [] \ R \ q_e$ _____
 $q_e \ 0 [] \ R \ q_e$ _____
 $q_e \ 1 [] \ R \ q_e$ _____
 $q_e \ [] \ 1 \ C \ q_f$ _____
 $q_1 \ 0 [] \ R \ q_e$ _____
 $q_1 \ [] \ 1 \ C \ q_f$ _____

Parsing regolare di coppie 01

Accettazione della stringa

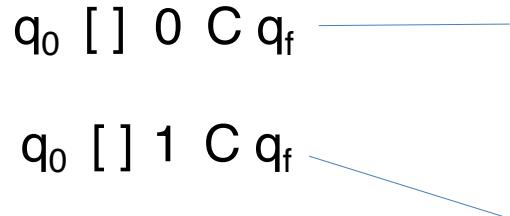
Scoperta di un 1 irregolare

Cancellazione del resto della stringa irregolare e rifiuto della stringa

Scoperta di uno 0 irregolare

Stringa terminata irregolarmente

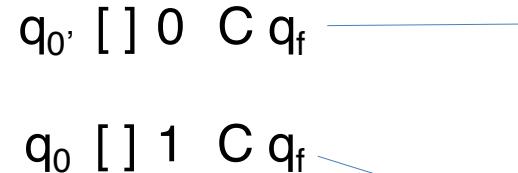
È come se avessimo bisogno di usare q_0 in due modi diversi



Accettazione della
stringa DOPO AVER
CANCELLATO
ALMENO UNA COPPIA
01

Rifiuto della stringa
PERCHÉ NON C'È
NEMMENO UNA
COPPIA 01

Usiamo due stati diversi



Accettazione della
stringa DOPO AVER
CANCELLATO
ALMENO UNA COPPIA
01

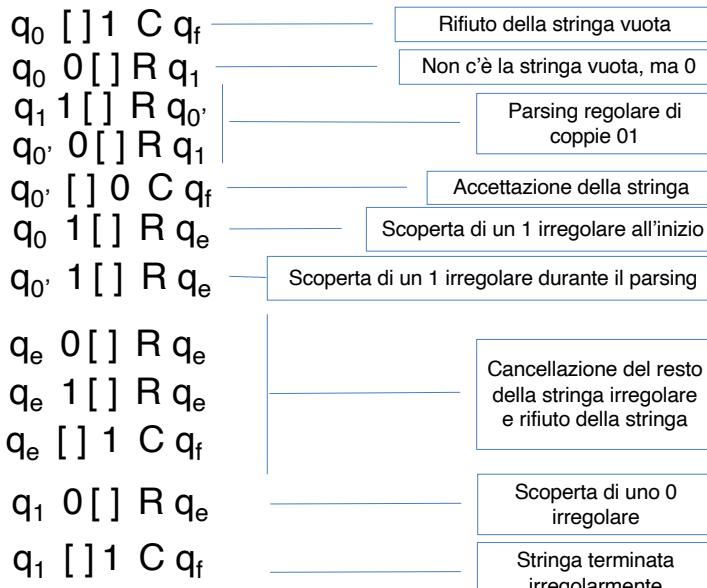
Rifiuto della stringa
PERCHÉ NON C'È
NEMMENO UNA
COPPIA 01

q_0 : stato iniziale, in cui la MT si aspetta uno 0

q_0' : stato intermedio, in cui la MT si aspetta uno 0

Come usare q_0 e q_0'

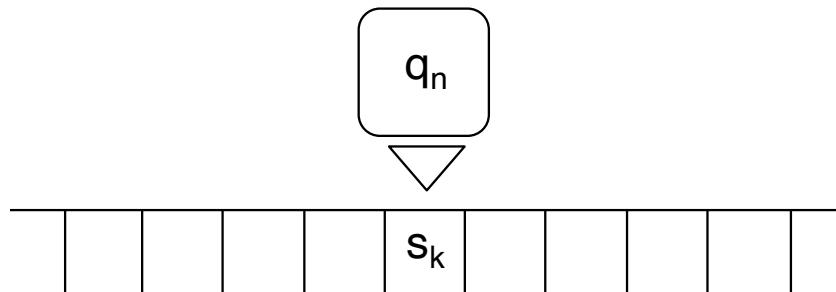
- q_0 : stato iniziale
- q_0' : stato intermedio } la MT si aspetta uno 0
- La MT inizia nello stato q_0 in cui può rifiutare la stringa vuota, oppure leggere e cancellare uno 0
- Se la MT legge e cancella uno 0, in futuro non tornerà più nello stato iniziale q_0 bensì andrà nello stato q_0' quando si aspetta di leggere uno 0 sul nastro



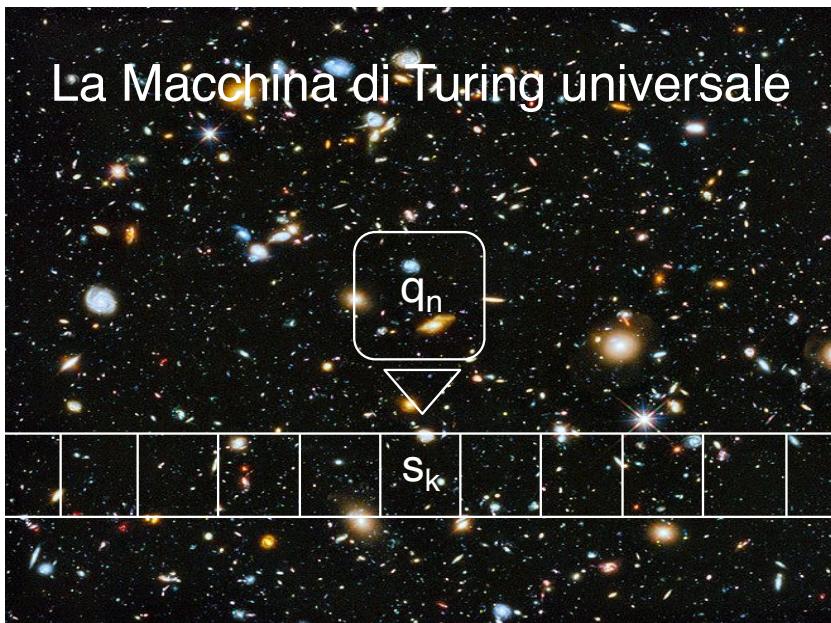
$q_0 \ 1 \ [] \ R \ q_e$ Scoperta di un 1 irregolare all'inizio
 $q_{0'} \ 1 \ [] \ R \ q_e$ Scoperta di un 1 irregolare durante il parsing

- Da notare questo “sdoppiamento”
- Prima c’era un solo stato in cui la MT si aspetta uno 0, ossia q_0
- Ora, invece, con la distinzione dell’inizio (q_0) dal parsing ($q_{0'}$) dobbiamo occuparci dell’1 irregolare due volte, per coprire entrambi gli stati

La Macchina di Turing



La Macchina di Turing universale



La Macchina di Turing universale

- Nota anche con l’acronimo MTU, è una Macchina di Turing in grado di simulare il comportamento di ogni altra MT
- Il concetto è facile da formulare, ma esiste davvero una MTU?
- Ancor prima di entrare nei dettagli della dimostrazione della sua esistenza, noi sappiamo già che la MTU esiste

MT = macchine di turing

MTU = macchina di turing universale

La MTU esiste 1/n

- Il ragionamento che facciamo è controfattuale (a un certo punto facciamo un'ipotesi che va contro la realtà dei fatti e arriviamo ad un assurdo, il che conferma che tale ipotesi è errata)
- I computer che noi tutti usiamo esistono, e questi computer sono macchine calcolatrici che riescono a simulare la funzionalità di qualunque altra macchina calcolatrice

La MTU esiste 2/n

- Ad esempio, con il software giusto, il vostro computer può diventare:
 - la macchina Skype
 - la macchina iTunes
 - la macchina calcolatrice
 - la macchina Word
 - la macchina interprete Java
 - la macchina .NET
 - la macchina compilatore Python
 - la macchina Photoshop
 - la macchina Firefox
 - etc. etc.
- Di fatto, qualunque funzione computabile, se scritta sotto forma di programma, può essere eseguita dal computer

|| No

La MTU esiste 3/n

- Il vostro computer è, di fatto, una macchina computazionale universale
- La sua universalità deriva dal fatto che in memoria possiamo mettere non solo i dati da elaborare, ma anche le istruzioni con cui elaborare questi dati
- Questa è l'idea del computer di tipo "stored program" è attribuita a Von Neumann, ma...



John Von Neumann (1903 – 1957)

...la vera storia non è chiarissima.

Von Neumann, genio ungherese (a 22 anni aveva già una laurea in ingegneria chimica a Zurigo e un dottorato in matematica a Budapest) sfuggito ai nazisti e rifugiatosi negli USA, presentò l'idea nel 1945.

Molti storici, però, affermano che il lavoro avesse preso forte ispirazione da un lavoro di Turing del 1936. Inoltre, due collaboratori di Von Neumann per il progetto Manhattan (bomba atomica), Eckert e Mauchly, avevano elaborato simili idee in maniera indipendente.

Per questi motivi, la sola attribuzione dell'idea dello stored-program a Von Neumann appare inappropriata a numerosi studiosi.

La sua passione per la bomba atomica gli fu fatale: morì a 54 anni di cancro.

La MTU esiste 4/n

- Ora facciamo l'ipotesi controfattuale: forse i computer di oggi, grazie alla loro universalità, non sono basati sul modello concettuale della Macchina di Turing?
- D'altro canto, ogni volta che dobbiamo risolvere un problema, dobbiamo concepirne una nuova: sembra che ci siano numerose MT specializzate, ma non una MT universale

La MTU esiste 6/n

- A questo punto abbiamo due scelte:
 - convincerci che i computer di oggi fanno più di quello che una MT può fare
 - cercare una MTU che funga da modello teorico dei computer di oggi
- La prima scelta è una vera sfida teorica e anche tecnologica, finora mai vinta
- La seconda è sostenuta dalla seguente dimostrazione

Visto che i computer sono in grado di eseguire qualsiasi calcolo sono delle MTU?

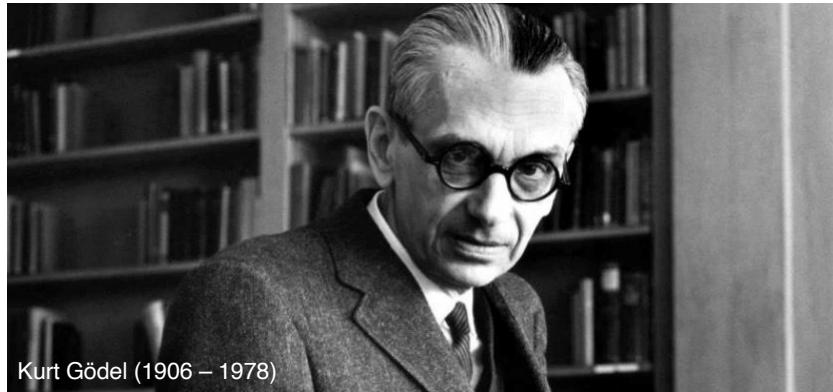
La MTU esiste 5/n

- Questo, però, va contro l'assunto che la Macchina di Turing sia un modello astratto della computazione in generale
- Molti hanno cercato di sfidare questo assunto, che non è stato dimostrato e che non può essere formalmente dimostrato perché:
 - la MT è formalmente ben definita, il concetto di "computazione" no
 - a tal punto che per definire formalmente "computazione" si fa riferimento alla MT, anche per le computazioni d'avanguardia come quelle quantistiche (con le MT non-deterministiche, su cui torneremo poi)

Gödelizzazione

- Prima di iniziare la dimostrazione dobbiamo introdurre il concetto di gödelizzazione
- Il suo nome deriva dal cognome del logico matematico Kurt Gödel
- Tralasciando definizioni formali complesse e non particolarmente utili in questo contesto, possiamo intendere la gödelizzazione come un processo di codifica degli elementi di un linguaggio
- Per codifica intendiamo il suo senso più stretto: una corrispondenza biunivoca con un sottoinsieme dei numeri naturali

| Def
codifica



Kurt Gödel (1906 – 1978)

L'austriaco Kurt Gödel è conosciuto soprattutto per la sua dimostrazione dell'incompletezza dei teoremi. Nel 1931 pubblica risultati fondamentali dimostrando che in ogni sistema matematico basato su assiomi e regole di inferenza ci sono proposizioni che non possono essere dimostrate o smentite dentro gli assiomi del sistema. In particolare, la consistenza degli assiomi non può essere dimostrata. Questo risultato segna la fine di decenni di tentativi di formalizzazione della matematica in termini di assiomi da parte di numerosi studiosi, tra cui Bertrand Russell e David Hilbert. I risultati di Gödel sono un punto di riferimento per la matematica del XX secolo e implicano che un computer non potrà mai essere programmato per rispondere a tutti i quesiti matematici. Da sempre affetto da disturbi psichici, morì di fame in ospedale, convinto che ci fossero persone intenzionate ad avvelenarlo.

Enumerazione delle MT

- Per l'unicità della scomposizione dei numeri in fattori primi, a ogni MT corrisponde uno e un solo gödeliano $g(MT)$, e a un numero naturale può corrispondere al massimo una sola MT (attenzione: non tutti i numeri sono gödeliani di MT, ad esempio i numeri dispari non lo sono)
- Si induce in questo modo una enumerazione di MT, con le MT ordinate secondo l'ordine dei corrispondenti gödeliani in \mathbb{N} :
 $MT_i < MT_j \Leftrightarrow g(MT_i) < g(MT_j)$

Gödelizzazione delle MT

- Creiamo una funzione $d()$ che associa a ciascun simbolo un numero dispari (es.: $d(C)=5$)

$$\begin{array}{ccccccccc} L & R & C & s_0 & q_0 & s_1 & q_1 & s_2 & q_2 \dots \\ 1 & 3 & 5 & 7 & 9 & 11 & 13 & 15 & 17 \dots \end{array}$$

- All'istruzione $I: q_1 s_0 s_2 R q_3$ associamo il numero

$$g(I) = 2^{d(q1)} * 3^{d(s0)} * 5^{d(s2)} * 7^{d(R)} * 11^{d(q3)}$$

- alla macchina $MT=\{I_1; I_2; \dots; I_n\}$ associamo il numero

$$g(MT) = 2^{g(I1)} * 3^{g(I2)} * \dots * p_n^{g(In)}$$

dove p_n è l' n -esimo numero primo

casella
 $d: C \rightarrow \mathbb{N}_{\text{dispari}}$

$g: I/MT \rightarrow \mathbb{N}$

Considerazioni 1/3

- Occorre una precisazione sull'univocità della corrispondenza tra MT e numeri naturali
- L'ordine con cui le istruzioni di una MT sono elencate
 - non conta per la funzione svolta dalla MT:
 - $\{I_1; I_2; I_3; I_4\}$ e $\{I_3; I_1; I_4; I_2\}$ sono la stessa MT
 - conta per calcolare il suo gödeliano
 - $2^{g(I1)} * 3^{g(I2)} * 5^{g(I3)} * 7^{g(I4)} \neq 2^{g(I3)} * 3^{g(I1)} * 5^{g(I4)} * 7^{g(I2)}$

Considerazioni 2/3

- Abbiamo due scelte:
 - introdurre una regola nell'insieme delle MT: le loro istruzioni vanno scritte in ordine alfabetico, così non ci saranno "doppioni" di MT che hanno le stesse istruzioni ma in ordine diverso
 - considerare la tavola di una MT, così come è scritta come la definizione univoca di una MT, quindi considerare diversa la MT che ha $\{I_1; I_2; I_3; I_4\}$ come tavola dalla MT che ha $\{I_3; I_1; I_4; I_2\}$
- In questo contesto, vanno bene entrambe

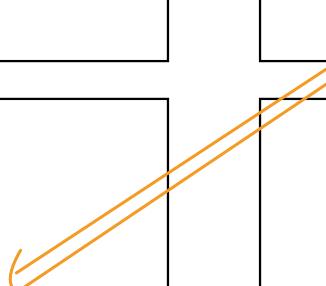
Considerazioni 3/3

- Attenzione: in questo procedimento abbiamo codificato sia singole istruzioni,
$$g(I) = 2^{d(q1)} * 3^{d(s0)} * 5^{d(s2)} * 7^{d(R)} * 11^{d(q3)}$$
sia intere MT
$$g(MT) = 2^{g(I1)} * 3^{g(I2)} * \dots * p_n^{g(I_n)}$$
- C'è il rischio che i numeri che corrispondono a istruzioni e numeri che corrispondono a macchine si confondano tra loro?

Gödelizzazione e MTU

- La gödelizzazione è il metodo per assimilare programmi e dati: in fondo, sono tutti espressi da numeri
- A questo punto, l'idea alla base dell'implementazione della MTU è chiara:
 - sul suo nastro metteremo in input il codice della MT che deve simulare e i dati da elaborare
 - in maniera analoga, nella memoria del mio computer ci sono sia il programma Powerpoint, sia il file di queste slide

sul nastro =
codice + dati

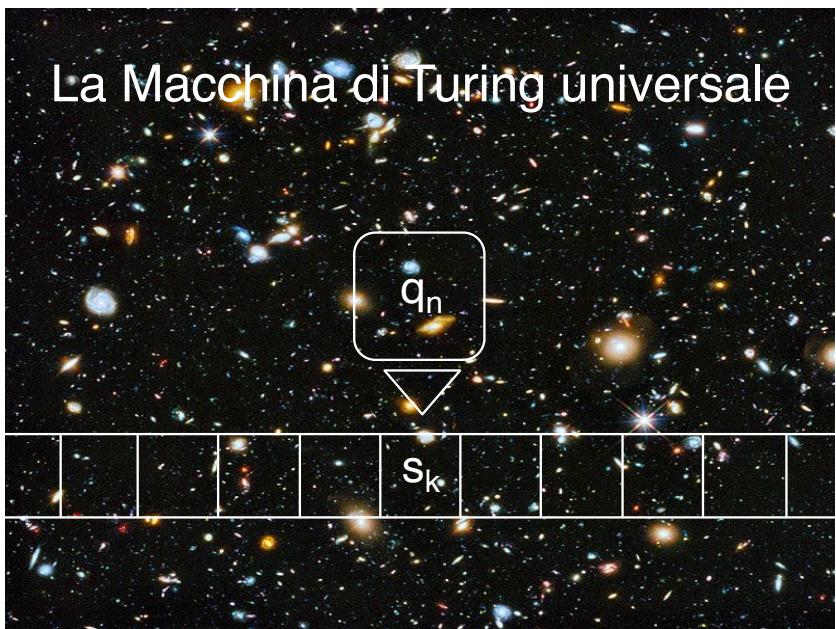


Funzionamento della MTU

- In input sul nastro:
 - codifica della tavola della MT da simulare
 - input su cui eseguire la tavola
- La MTU decodifica (secondo le regole precedentemente descritte) la tavola e scrive le istruzioni della MT sul nastro
- La parte di nastro con il programma diventa la tavola da eseguire
- Una volta raggiunto lo stato finale della MT, la MTU cancella tutto sul nastro tranne il risultato in output

Hanno
Gödelizzazione

La Macchina di Turing universale



Funzione parziale

- Una funzione $f: D \rightarrow C$ si dice parziale quando per alcuni elementi x in D non è definita, ossia non esiste alcun elemento y in C tale che $y = f(x)$
- In tal caso si scrive $f(x) = \perp$
- Il sottoinsieme di D di tutti e soli gli elementi x per cui esiste una $y = f(x)$ si chiama campo di esistenza della funzione

Def

Esistenza delle funzioni parziali (1/3)

- ALG, l'insieme degli algoritmi, è numerabile
- Visto che gli algoritmi sono stringhe di lunghezza finita di un linguaggio, possono essere messi in ordine (componendo l'ordinamento in base alla lunghezza delle stringhe con il criterio di ordinamento lessicografico), e quindi possiamo considerare ALG enumerabile, e quindi esiste una funzione $\phi: N \rightarrow ALG$ totale e computabile che genera tutti gli algoritmi in ordine

Una funzione definita per ogni elemento del dominio

$Card(ALG) = Card(\mathbb{N})$

ordinamento
lunghezza + alfabetico

"posizione"
 $F: \mathbb{N} \rightarrow ALG$

Esistenza delle funzioni parziali (2/3)

- Supponiamo (per assurdo) che gli algoritmi in ALG computino tutti funzioni totali: ossia dando in input i numeri naturali, la funzione ϕ genera una sequenza di algoritmi A_i , ciascuno dei quali computa una funzione θ_i totale
 - Definiamo la funzione $f(x) = \theta_x(x) + 1$
 - Essendo ϕ computabile, la ricerca della x -esima funzione θ_x è un procedimento algoritmico, inoltre θ_x è totale, quindi f è computabile e totale
 - Essendo computabile e totale, f è nella lista delle θ_i
 - ϕ computabile $\Rightarrow f(x) = \theta_x$
 - θ_x totale $\rightarrow f(x) = \theta_{x+1}$ computabile + totale
 - $f(x)$ computabile e totale \in lista θ_i

"enumerabile"
 $\phi: \mathbb{N} \rightarrow ALG$
Ai compute gli tot.

Esistenza delle funzioni parziali (3/3)

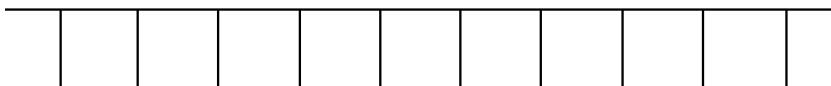
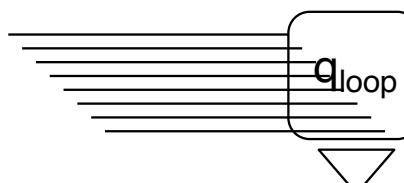
- Essendo computabile e totale, f è nella lista delle θ_i
- Chiamiamo k la posizione della funzione f nella lista
- Calcoliamo $f(k)$:
 - per la definizione di f , $f(k) = \theta_k(k) + 1$
 - poiché f è la k -esima funzione nella lista, $f(k) = \theta_k(k)$
 - si ottiene $\theta_k(k) + 1 = \theta_k(k)$, ossia $1 = 0$, che è assurdo
- L'unica via d'uscita dall'assurdo è ipotizzare che esistano funzioni parziali nella lista, e che quindi θ_k non sia definita in k

$$f(k) = \theta_k(k) + 1$$

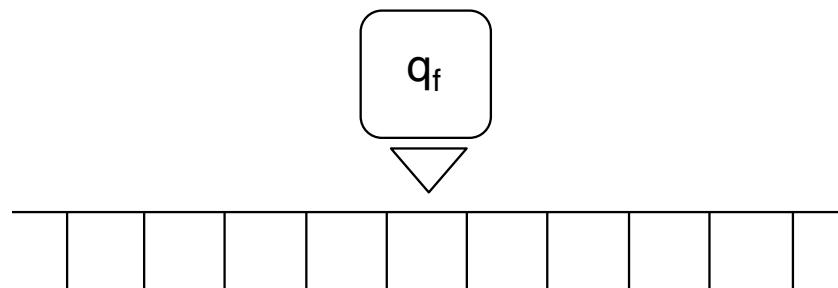
k -esima funzione $\Rightarrow f(k) = \theta_k(k)$

$$\theta_k(k) + 1 = \theta_k(k) \Rightarrow 0 = 1$$

Macchina di Turing che computa una funzione parziale? (B)



Macchina di Turing che computa una funzione parziale? (A)



Dunque il computer:

- a) fa in più rispetto ad una MT? Non dimostrabile
b) esiste una MTU generalizzabile?

computer è una macchina computazionale universale

le parole di tutto ciò



stored program
codice + dati

GENERALIZZAZIONE

$$d(1) : C \rightarrow \mathbb{N}$$

$$f(1) : I / MT \rightarrow \mathbb{N}$$

- HOSTO:
- codifica MT
 - input MT

MTV: decodifica MT e usa input sul nexto rimane solo output

MT e dati (I) sono numeri sul nexto.

funzioni parziali

- $f(x) = 1$ non esiste immagine
- campo di esistenza delle f.
- con immagine
- ALG enumerabile
- $f : \mathbb{N} \rightarrow \text{ALG}$ totale e computabile

\rightarrow ALG = funzioni totali + funzioni parziali

- MT compute una f parziale:
- output inesistente (niente nexto)
 - MT in loop, computazione non finisce mai

Computabilità

12 e 13 gennaio 2022

Mario Verdicchio
Università degli Studi di Bergamo
Anno Accademico 2021-2022

La macchina H

- Sia C_{MT} la codifica del codice di una MT
- Si suppone che esista la seguente macchina:
$$H(C_{MT}, I) = \begin{cases} 0, & \text{se } MT(I) \text{ termina} \\ 1, & \text{se } MT(I) \text{ non termina} \end{cases}$$
- Sulla base della macchina H, costruiamo la macchina H'

HALTING PROBLEM

Problema dell'arresto

- Dato il codice di una MT e un input I, la computazione di MT su I termina in tempo finito?
Indecidibile
- Questo problema non è decidibile, ossia non è risolvibile tramite una MT
Seugna
- Lo dimostriamo per assurdo, supponendo che esista una macchina H che riesca, in tempo finito, a darci sempre una risposta (positiva o negativa)

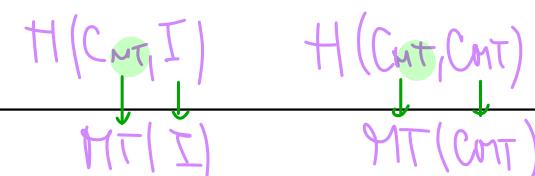
Decidibile \Rightarrow sempre computabile
Indecidibile \Rightarrow non sempre computabile

La macchina H'

Diamo la della
MT a MT stessa

Diamo il codice
del software al
software stesso

- Si costruisce la seguente macchina:
$$H'(C_{MT}) = \begin{cases} 0, & \text{se } MT(C_{MT}) \text{ termina, ossia } H(C_{MT}, C_{MT})=0 \\ 1, & \text{se } MT(C_{MT}) \text{ non termina, } H(C_{MT}, C_{MT})=1 \end{cases}$$
- La macchina H' computa una funzione che è un caso particolare della funzione computata da H (C_{MT} è un valore particolare di I)



La macchina Z

- Sulla base di H' , si costruisce la seguente macchina:

$$Z(C_{MT}) = \begin{cases} \text{non termina, se } H'(C_{MT}) = 0 \\ 0, \text{ se } H'(C_{MT}) = 1 \end{cases}$$

Fa l'opposto di H'

- Per la non-terminazione basta aggiungere istruzioni che facciano sì che, se H' restituisce 0 in output, la testina di Z vada, ad esempio, a destra all'infinito

Tavolo -> numero attraverso la godelizzazione

Controesempio?

```
while (1) {
    print (0);
}
```

1) non abbiamo input
2) non esiste una macchina per tutti i linguaggi in generale

NO: non è un controesempio perché si tratta di un caso specifico risolvibile algoritmicamente.
Quello che non esiste è una MT che possa fare l'analisi di terminazione o meno per qualunque codice C_{MT} e input I .

Non esiste una macchina generale che stabilisca la non terminazione per qualunque software e qualunque input

Il paradosso

- Z è una MT
- Possiamo dunque codificarla $\Rightarrow C_Z$ (godeliano, tavola tradotta in codice numerico)
- Diamo a Z il codice di Z stesso $\Rightarrow Z(C_Z)$ e abbiamo due casi possibili:

- Proviamo a calcolare $Z(C_Z)$

- a) $Z(C_Z)$ non termina $\Leftrightarrow H'(C_{MT}) = 0 \Leftrightarrow Z(C_Z)$ termina
- b) $Z(C_Z) = 0 \Leftrightarrow H'(C_{MT}) = 1 \Leftrightarrow Z(C_Z)$ non termina

- In ogni caso, si arriva a un assurdo che ci costringe a respingere l'assunzione che la macchina H esista
- La macchina H non può esistere, ossia il problema dell'arresto non è decidibile

↳ $Z(C_T) = 1 \Rightarrow H'(C_T) = 0 \Rightarrow MT(C_T) = 0$
 $H(C_T, C_T) = 0 \Rightarrow T(C_T) = 0$

$$H'(C_T) = H(C_T, C_T)$$

$$T(C_T)$$

La computabilità non possiede una definizione formale.
Ulteriore definizione di computabilità

Funzioni base e operazioni

Funzioni elementari la cui definizione è immediata

- Funzioni base:
 - funzione zero, $Z: N \rightarrow N$, $Z(x) = 0$ per ogni x
 - funzione successore, $s: N \rightarrow N$, $s(x) =$ il successore di x nellenumerazione di N
 - proiezioni, $P^n_i: N^n \rightarrow N$, $P^n_i(x_1, \dots, x_n) = x_i$ Scelgo il componente i -esimo

- Operazione di composizione:

- date $\chi: N^k \rightarrow N$ e $\psi_1, \dots, \psi_k: N^n \rightarrow N$
- si ottiene per composizione la funzione $\phi: N^n \rightarrow N$ quando $\phi(x_1, \dots, x_n) = \chi(\psi_1(x_1, \dots, x_n), \dots, \psi_k(x_1, \dots, x_n))$

- Operazione di ricorsione:

- date $\chi: N^n \rightarrow N$ e $\psi: N^{n+2} \rightarrow N$
- si ottiene per ricorsione la funzione $\phi: N^{n+1} \rightarrow N$ quando

$$\begin{cases} \phi(x_1, \dots, x_n, 0) = \chi(x_1, \dots, x_n) \\ \phi(x_1, \dots, x_n, s(y)) = \psi(x_1, \dots, x_n, y, \phi(x_1, \dots, x_n, y)) \end{cases}$$

Vengono definite ulteriori funzioni che sono la composizione di funzioni base

Operazioni somma, sottrazione.... non vengono utilizzate perché stiamo definendo le operazioni base

Funzioni ricorsive primitive (RP)

RP

- Si dice **ricorsiva primitiva** una funzione che si ottiene dalle funzioni base applicando un numero finito di volte le operazioni di composizione e/o ricorsione
- RP è l'insieme di tutte e sole le funzioni ricorsive primitive
- Una **derivazione in RP** è una sequenza di funzioni, ciascuna delle quali è una funzione base, oppure è stata ottenuta dalle funzioni precedenti tramite composizione o ricorsione
- L'ultima funzione in una derivazione in RP si dice RP-derivabile
- Una funzione è in RP \Leftrightarrow è RP-derivabile
- Esempio di derivazione dell'addizione in RP:
 - $\phi_1(x) = P^1_1(x)$ [base] Funzione identità
 - $\phi_2(x) = s(x)$ [base]
 - $\phi_3(x,y,z) = P^3_3(x,y,z)$ [base] Prende 3 input e restituisce il terzo
 - $\phi_4(x,y,z) = s(P^3_3(x,y,z))$ [composizione di 2 e 3]
 - $\phi_5(x,0) = P^1_1(x)$ Caso base
 - $\phi_5(x,s(y)) = s(P^3_3(x,y,\phi_5(x,y)))$ [ricorsione di 1 e 4]

Caso generale

RP derivabile

Derivazione in
RP

Somma è funzione ricorsiva si
riesce ad ottenerne dalle
funzioni base tramite
composizione e ricorsione

Ripercorro il passo ricorsivo fino a quando non arrivo al caso base

$$\begin{aligned} \bullet \quad \phi_5(x,0) &= P^1_1(x) \\ \rightarrow \phi_5(x,s(y)) &= s(P^3_3(x,y,\phi_5(x,y))) \end{aligned}$$

$$\bullet \quad \phi_5(x,0) = x = x + 0 = x$$

$$\bullet \quad \phi_5(x,3) = \phi_5(x,s(2)) = s(P^3_3(x,2,\phi_5(x,2))) = \\ s(\phi_5(x,2)) = s(s(\phi_5(x,1))) = s(s(s(\phi_5(x,0)))) = \\ s(s(s(x))) = x + 3$$

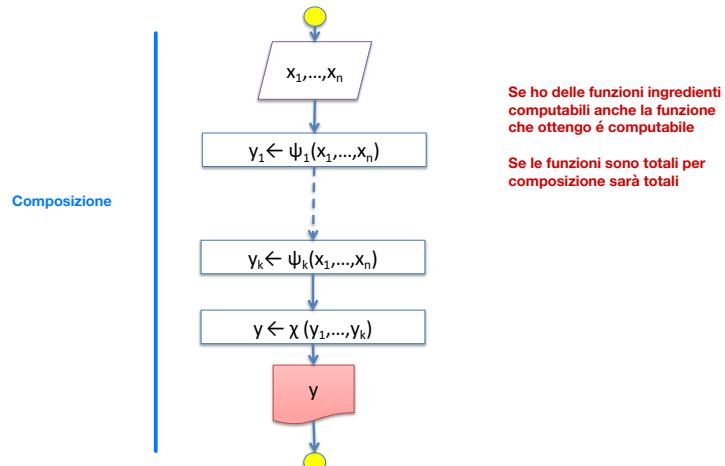
ϕ_5 si comporta come +

dal nostro punto di vista, DEFINIAMO +
come ϕ_5 dimostrando che + è in RP, cioè è una
funzione ricorsiva primitiva

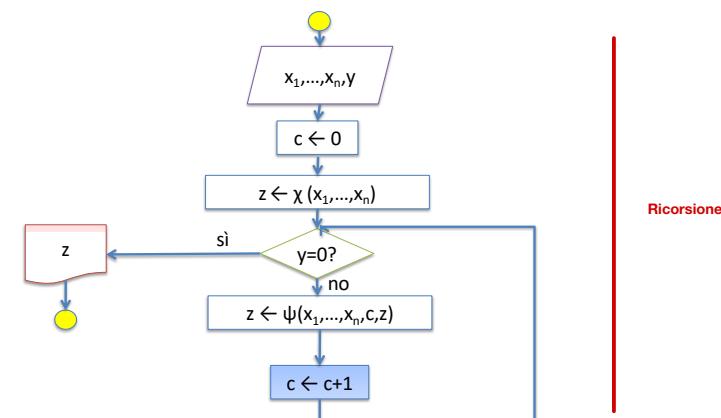
$$S(\phi_5(x_{11})) = S(S(P^3_3|x_{11}|, \phi_5(x_{11}))) = S(S(\phi(x_{11}))) = S(S(S(\phi(x_{10})))) = \\ = S(S(S(x)))$$

3 successore
di 2

La composizione conserva la computabilità e la totalità delle funzioni



La ricorsione conserva la computabilità e la totalità delle funzioni



Le funzioni RP

- Le funzioni base sono computabili e totali
- Le operazioni conservano tali caratteristiche delle funzioni a cui sono applicate
- Perciò le funzioni RP sono computabili e totali

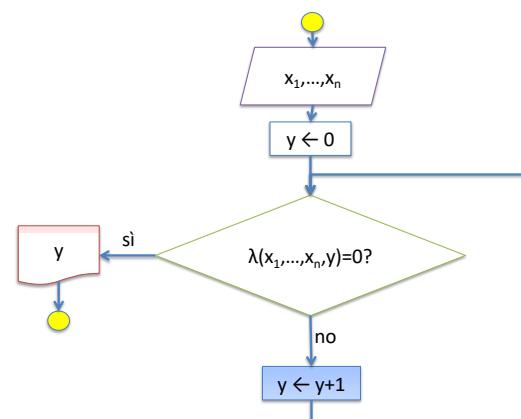
Nuova operazione: minimizzazione

- Operazione di minimizzazione:
 - data $\lambda:N^{n+1} \rightarrow N$ Prende una tupla $N + 1$ e restituisce un singolo numero
 - si ottiene per minimizzazione la funzione $\phi:N^n \rightarrow N$ quando
$$\phi(x_1, \dots, x_n) = \mu y (\lambda(x_1, \dots, x_n, y) = 0),$$
dove $\mu y(P)$ vuol dire "il più piccolo y tale che P "
- Funzioni definite tramite la minimizzazione possono essere parziali, perché, dati x_1, \dots, x_n , non è garantita l'esistenza di un y tale che $\lambda(x_1, \dots, x_n, y) = 0$

Funzioni computabili totali non RP

- Le funzioni RP (ad 1 argomento, senza perdere in generalità) ϕ_i sono enumerabili perché possiamo costruire una sequenza di funzioni ottenute da quelle base con 0, 1, 2, 3, ... applicazioni delle operazioni di composizione e ricorsione
- Data tale sequenza, definiamo $f(x) = \phi_x(x)+1$
- f è computabile e totale
- Se supponiamo che f sia nella sequenza, ovvero che $f \equiv \phi_k$, calcoliamo $f(k)$
- Per la definizione di f , $f(k) = \phi_k(k)+1$
- Per la posizione di f nella sequenza, $f(k) = \phi_k(k)$
- Si conclude che $\phi_k(k)+1 = \phi_k(k)$, ossia, visto che ϕ_k è totale e quindi $\phi_k(k)$ è un valore definito, che $1 = 0$
- Dobbiamo respingere l'ipotesi che f sia nella sequenza, ossia dobbiamo ammettere che esistono funzioni computabili totali che NON sono in RP

La minimizzazione conserva la computabilità delle funzioni



Funzioni ricorsive generali (RG)

- Si dice **ricorsiva generale** una funzione che si ottiene dalle funzioni base applicando un numero finito di volte le operazioni di composizione, ricorsione, e minimizzazione
- RG è l'insieme di tutte e sole le funzioni ricorsive generali
- Una **derivazione** in RG è una sequenza di funzioni, ciascuna delle quali è una funzione base, oppure è stata ottenuta dalle funzioni precedenti tramite composizione, ricorsione, o minimizzazione
- L'ultima funzione in una derivazione in RG si dice RG-derivabile
- Una **funzione** è in RG \Leftrightarrow è RG-derivabile
- Esempio di funzione in RG:

$\phi(x,y) = \begin{cases} x/y, & \text{se } x \text{ è multiplo di } y \\ \perp, & \text{altrimenti} \end{cases}$

Divisione che emette il risultato quando non ha resto

- $\phi_1(x) = pr(x)$ [funzione predecessore, definita per ricorsione da $z(x)$ e $P^1_1(x)$]
 - $\phi_2(x,y) = x-y$ [differenza simmetrica, definita per ricorsione da $P^1_1(x)$ e $pr(x)$]
 - $\phi_3(x,y) = dist(x,y) = x-y + y-x$ [distanza, definita per composizione da $+ e \perp$]
 - $\phi_4(x,y) = \mu z(dist(z,y), x) = 0$ [definita per minimizzazione da una composizione di $*$ e $dist$]
- ϕ_4 è la funzione ϕ , ed è stata derivata da funzioni RP con operazioni di composizione, ricorsione, e minimizzazione, quindi è in RG

Enumerazione delle RG

- In questo caso non si conclude nulla sull'appartenenza di $f(x) = \phi_x(x)+1$ a RG o meno, perché l'uguaglianza $\phi_k(k)+1 = \phi_k(k)$ non porta ad alcun assurdo
- Semplicemente, dobbiamo ammettere che ϕ_k , se appartiene alla sequenza delle RG, non sia definita in k , il che è compatibile con la parzialità delle RG
- Il tutto è compatibile anche con l'ipotesi che f NON sia nella sequenza delle RG

Dimostrazione

- $\phi(x,y) = x/y$, se x è multiplo di y
 \perp , altrimenti

1. $\phi_1(x) = pr(x)$ [funzione predecessore, definita per ricorsione da $z(x)$ e $P^1_1(x)$]

$pr(0) = z(x) = \perp$ Caso base

$pr(s(x)) = P^1_1(x) = x$ Caso generale

$pr(x)$ è in RP (E QUINDI anche in RG)

2. $\phi_2(x,y) = x-y$ [differenza simmetrica, definita per ricorsione da $P^1_1(x)$ e $pr(x)$]

$x-y = P^1_1(x) = x$

$x-y = pr(x-y)$

$x-y$ è in RP (E QUINDI anche in RG)

3. $\phi_3(x,y) = dist(x,y) = x-y + y-x$ [distanza, definita per composizione da $+ e \perp$]

$dist$ è in RP (E QUINDI anche in RG)

4. $\phi_4(x,y) = \mu z(dist(z,y), x) = 0$ [definita per minimizzazione da una composizione di $*$ e $dist$]

quindi, ϕ_4 appartiene a RG (ϕ_4 coincide con la funzione ϕ)

$x-y = z(x) = x$

$x-y = +(x, (x-y))$ [anche al moltiplicazione è in RP]

$x-y = x-4$

Nella differenza simmetrica
se il primo termine è più
piccolo del secondo = 0

$$|x-y| = k-4$$

x è il più piccolo e tale che
 y distanza fra $x-y$ e $x = 0$

Teorema (Turing, 1937)

Se e solo se

- Una funzione f è RG $\Leftrightarrow f$ è T-computabile
- (\Rightarrow)
 - si dimostra che le funzioni base sono T-computabili
 - e che le operazioni di composizione, ricorsione, e minimizzazione conservano la T-computabilità

La funzione zero è T-computabile

- La seguente macchina di Turing MTz infatti computa la funzione zero:

$q_1 \mid s_0 D q_1$
 $q_1 s_0 \mid C q_0$

Una barra singola = 0

Le funzioni proiezione sono T-computabili

- La seguente macchina di Turing MT p^3_1 , ad esempio, computa la funzione $P^3_1(x,y,z) = x$:

$q_1 \mid \mid D q_1$
 $q_1 s_0 s_0 D q_2$
 $q_2 \mid s_0 D q_2$
 $q_2 s_0 s_0 D q_3$
 $q_3 \mid s_0 D q_3$
 $q_3 s_0 s_0 C q_0$

Prendo una terna e restituisco il primo elemento

I numeri sono separati da una cella in mezzo sul nastro

La funzione successore è T-computabile

- La seguente macchina di Turing MTs infatti computa la funzione successore:

$q_1 \mid \mid D q_1$
 $q_1 s_0 \mid C q_0$

Aggiungo una barra

La composizione conserva la T-computabilità

- $\phi(x) = x(\psi(x))$
- Se x e ψ sono T-computabili, lo è anche ϕ ?
- Ossia: se esistono MT_x e MT_ψ , si riesce a costruire MT_ϕ ?
- Si: basta aggiungere le istruzioni di MT_x a quelle di MT_ψ , con i seguenti accorgimenti:
 - rinominare gli stati interni di MT_x per evitare sovrapposizione con gli stati di MT_ψ
 - aggiungere istruzioni che portino la testina della macchina nella posizione standard dopo che sono state eseguite le istruzioni di MT_ψ e prima che vengano eseguite quelle di MT_x

Primo esegue un programma e poi l'altro

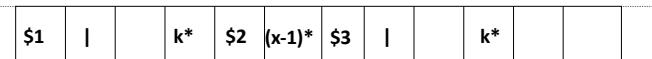
La ricorsione conserva la T-computabilità

$$\begin{cases} \phi(0) = k \\ \phi(s(x)) = \psi(x, \phi(x)) \end{cases}$$

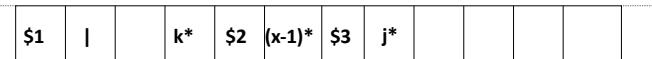
- Se ψ è T-computabile, lo è anche ϕ ?
- Ossia: avendo a disposizione MT_ψ , si riesce a costruire MT_ϕ ?
- Sì: viene mostrato nelle seguenti slide
- L'alfabeto include i simboli $\$, \$1, \$2, \3 che sono usati come separatori

La ricorsione con MT

- MT_ϕ esegue il programma di MT_ψ sulla parte a destra di $\$3$:



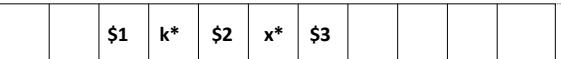
ottenendo:



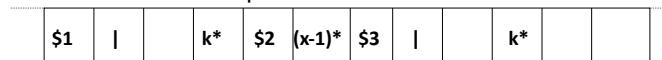
dove j^* è la codifica di $j = \psi(0, k) = \psi(0, \phi(0)) = \phi(1)$

La ricorsione con MT

- Per calcolare $\phi(x)$, MT_ϕ inizia scrivendo sul nastro (k^* rappresenta $k+1$ barre):



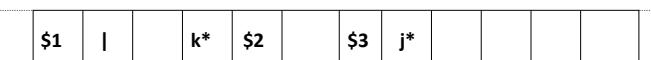
- MT_ϕ cancella una barra da x^*
- Se ci sono zero barre tra $\$2$ e $\$3$, l'output è tra $\$1$ e $\$2$ (perché $\phi(0) = k$)
- Altrimenti MT_ϕ configura il nastro così:



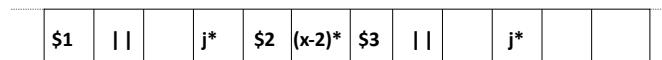
Codifica di $(0,k)$

La ricorsione con MT

- MT_ϕ cancella una barra tra $\$2$ e $\$3$, se non ci sono più barre vuol dire che $x=1$ e quindi l'output è $j = \psi(0, k) = \phi(1)$, che si trova a destra di $\$3$:



- Altrimenti nuova configurazione e si ripete:



- Eseguendo MT_ψ a destra di $\$3$ si calcola $\psi(1, \phi(1)) = \phi(2)$...e così via fino a $\phi(x)$

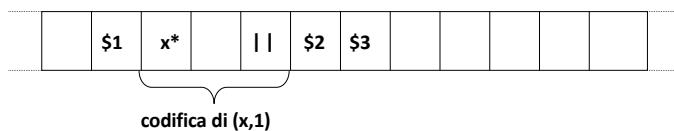
La minimizzazione conserva la T-computabilità

- $\phi(x) =$ il più piccolo $y: \lambda(x,y)=0$
- Dobbiamo costruire MT_ϕ usando MT_λ
- MT_ϕ procede eseguendo iterativamente il codice di MT_λ per calcolare $\lambda(x,0), \lambda(x,1), \lambda(x,2), \dots, \lambda(x,y)$ e restituisce in output il primo y per cui $\lambda(x,y)=0$
- Se tale y non esiste MT_ϕ non si ferma mai (e infatti ϕ non è definita per quella x)

Minimizzazione introduce la parzialità

La minimizzazione con MT

- Altrimenti MT_ϕ riconfigura il nastro così:

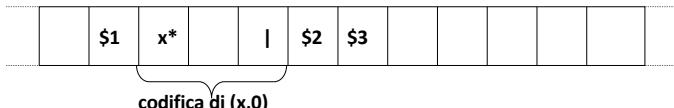


e ripete tutto per calcolare $\lambda(x,1)$

- ...e così via fino a trovare (eventualmente) il primo y per cui $\lambda(x,y)=0$

La minimizzazione con MT

- MT_ϕ configura inizialmente il nastro così:

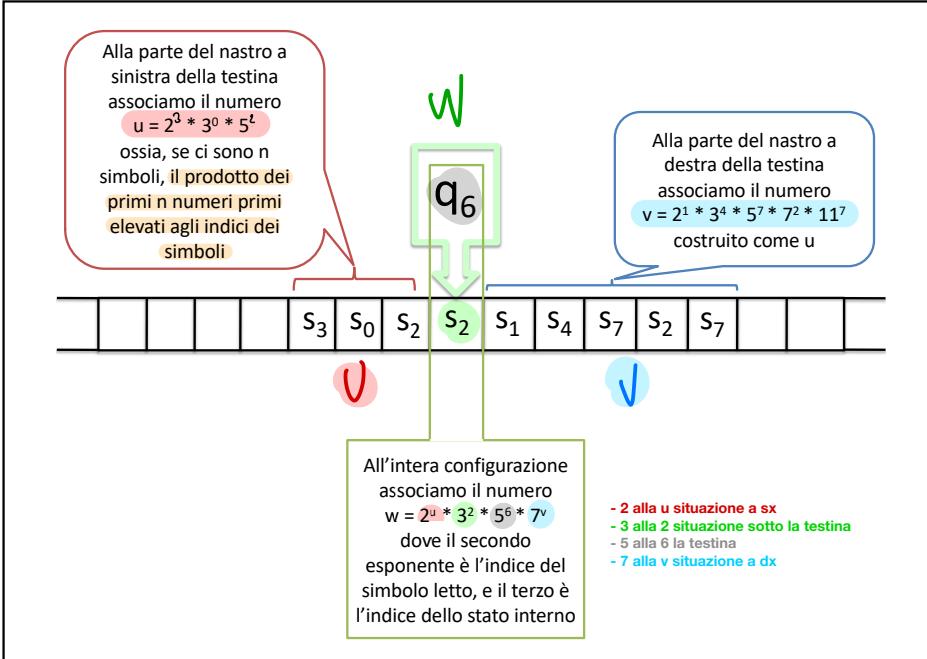


- Poi copia i dati tra \$1 e \$2 nello spazio tra \$2 e \$3 e lì usa il codice di MT_λ per calcolare $\lambda(x,0)$
- Se il risultato è zero cancella tutto il resto e lo lascia come output

Gödelizzazione delle configurazioni di una MT

- Prima di procedere alla dimostrazione (sommaria) di (\Leftarrow) , introduciamo delle codifiche elaborate da Gödel che permettono di esprimere le computazioni delle MT come applicazioni di funzioni matematiche

- tanti numeri primi quanti sono nella sezione
- Ciascuno viene elevato all'indice del simbolo nella cella a cui corrisponde
- E tutti moltiplicati tra di loro



Possiamo vedere il programma della MT come una funzione

Il programma di MT come funzione

- L'indice w è associato alla configurazione della MT in maniera univoca, grazie all'unicità della scomposizione di un numero in fattori primi
- Per mezzo delle sue istruzioni, la MT passa da una configurazione all'altra in modo deterministico: da w la MT non può fare altro che passare a w' (a meno che w non sia una configurazione finale)
- Ad ogni MT possiamo quindi associare una funzione:
 $\rho_{MT}(w) = \begin{cases} w' & \text{se } w \text{ è la codifica di una configurazione non finale} \\ w & \text{altrimenti} \end{cases}$

Il determinismo di MT garantisce che ρ_{MT} sia una funzione
Si può dimostrare che qualunque sia la MT, ρ_{MT} è in RP

La computazione di un numero finito di passi con MT come funzione

- A partire da ρ_{MT} , definiamo θ_{MT} per ricorsione:

$$\begin{cases} \theta_{MT}(w, 0) = w \\ \theta_{MT}(w, s(z)) = \rho_{MT}(\theta_{MT}(w, z)) \end{cases}$$
- Se w è la codifica di una configurazione w , $\theta_{MT}(w, z)$ è la codifica della configurazione che si ottiene dopo z istruzioni a partire da w
- Anche θ_{MT} è RP, perché definita per ricorsione da una funzione RP

f è T-computabile $\Rightarrow f$ è RG

- Sia MT_f la MT che computa f
- Dato l'input x , lo si codifichi sul nastro di MT_f
- La computazione di MT_f è data dalla funzione RP $\theta_{MT_f}(w, z)$, e con un'operazione di minimizzazione si ottiene (se esiste) il minimo numero z_0 di passi per cui, a partire dalla codifica della configurazione di partenza w_1 , si ottiene una configurazione finale
- Calcolando $\theta_{MT_f}(w_1, z_0)$ si ottiene la codifica della configurazione finale, e invertendo la gödelizzazione, si ricava la codifica dell'output sul nastro
- Decodificando l'output, si ottiene $f(x)$
- f è stata ottenuta componendo tutti i calcoli di cui sopra, fatti con funzioni RP e con una minimizzazione su una funzione RP, quindi f è RG

- parto dalla MT ce computa una certa funzione
- mappa il suo comportamento su delle funzioni che sono in RP
- cerco il numero minimo di passi per ottenere una configurazione finale
- dalla configurazione finale ottengo i simboli presenti sul nastro
- decodifico questi simboli e ottengo il risultato finale della computazione

In conclusione:

- Teorema di Turing:** È stato dimostrato
Una funzione f è RG \Leftrightarrow f è T-computabile
- Tesi di Church:**
Una funzione è computabile \Leftrightarrow essa è RG
- Tesi di Church-Turing:**
Una funzione è computabile \Leftrightarrow essa è T-computabile

Tesi => perché il concetto di computabilità non è rigorosamente formulato o definito. Quindi non è stato dimostrato perché è una proposta

Teorema di Turing: f è RG \Leftrightarrow f è T-computabile

- \Rightarrow
- f base sono T-computabili
 - (da dx e sx) operazioni ricorsione, composizione, minimizzazione conservano la T-computabilità

T-computabile: \exists una MT che computa la funzione/operazione

- f zero è T-computabile : σ al head
- f accrescere è T-computabile : +1 all'input
- f proiezione è T-computabile : rimane solo input selezionato
- Composizione conserva T-computabilità: $\phi(x)=x(\psi(x))$ MT_x e MT_y esistono e per costituire MT_φ aggiungo le istruzioni MT_x e MT_y (seguire prima MT_y di MT_x e evitare conflitti dati)
- Ricorsione conserva T-computabilità
- Minimizzazione conserva T-computabilità: costituisco MT_φ facendo MT_x se $y \neq MT_\phi$ non si ferme mai (f parziale $f(x)=\perp$)

(da dx e sx) \cdot $O_{MT}(w,t)$ codifica delle conf che si ottiene dopo t istruzioni a partire da w

Riassunto

Def: un problema **ben** è decidibile o indecidibile
quando non è risolvibile tramite una MT.

=>

Funzioni base:

- zero $z: \mathbb{N} \rightarrow \mathbb{N}$
- successore $s: \mathbb{N} \rightarrow \mathbb{N}$
- proiezione $f^i: \mathbb{N}^n \rightarrow \mathbb{N}$

Operazioni:

- composizione
- ricorsione

RP: sono f ottenibili dalle F base applicando

un n° finito di volte **composizione e ricorsione**

- derivaione in RP: liste operazioni su F base
- l'ultima riga di una derivaione in RP: RP-derivabile
- f in RP \Leftrightarrow è RP-derivabile
- le f. RP ed i suoi argomenti ϕ sono enumerabili
- $\exists f$ che sono computabili e totali ma non RP

RG:

sono f ottenibili dalle F base applicando un n° finito di volte **ricorsione, composizione, minimizzazione**

- derivaione in RG: sequenza funzioni
- l'ultima funzione RG-derivabile
- f in RG \Leftrightarrow RG-derivabile
- se $\exists f$ RP applica ricorsione, composizione, minimizzazione è una F RG
- RG è enumerabile $\Rightarrow f \in \omega \cap RG$

Dato:

- codifica MT : C_{MT}
- Input delle MT : I

Non è possibile calcolare il tempo di computazione di una MT con il suo I tramite una MT (prob. esecutabile)

PROBLEMA DI ARRESTO
HALTING PROBLEM

OBS:

- le F. base sono computabili e totali
- le operazioni preservano tali caratter.
- \Rightarrow le f. RP sono computabili e totali

Minimizzazione: new equazione

- $\mu y P = 0$ "il più piccolo y t.c. P=0"
- f definite tramite minimizzate, possono usare parziali

Def: codifica \rightarrow computazione delle MT \Rightarrow come applicazione di f matematiche

$$W = 1 \cdot 2^1 \cdot 3^2 \cdot 4^3 \cdot 5^4 \cdot 6^5 \cdot 7^6 \cdot 8^7 \cdot 9^8 \cdot 10^9 \cdot \dots$$

- conf N Universo
- da conf e conf 1, varie deterministiche

OBS: ad ogni MT posiamo associare una f

$$Q_{MT}(w) = \begin{cases} w & \text{se } w \in \text{dom codifica non finale} \\ w & \text{altrimenti} \end{cases}$$

NP: $\forall M \in \omega \quad Q_{MT}(w) \in RP$

1. Perché quella di Church è una tesi e non un teorema?

- a) Perché non esiste una definizione precisa del concetto di ricorsività generale.
- b) Perché non esiste una definizione precisa del concetto di computabilità.
- c) Perché non esiste una definizione precisa del concetto di ricorsività primitiva.
- d) Nessuna delle precedenti

2. Quale dei seguenti insiemi è numerabile?

- a) $\{0; 1\}$
- b) $\{\}$
- c) L'insieme di tutte le funzioni $N \rightarrow N$
- d) Nessuna delle precedenti

3. Dati due insiemi A e B, $\text{Card}(A) = \text{Card}(B)$ vuol dire che

- a) esiste una funzione iniettiva da A a B ma non ne esiste una suriettiva
- b) esiste una funzione suriettiva da A a B ma non ne esiste una iniettiva
- c) almeno una funzione suriettiva da A a B è anche iniettiva
- d) Nessuna delle precedenti

4. Se A è un insieme finito, $\text{Card}(\wp(A)) = 2^{\text{Card}(A)}$. Perché la base della potenza è 2?

- a) Perché nei computer si usa la logica binaria.
- b) Perché *tertium non datur*.
- c) Non si sa perché, ma finora l'equazione non è stata mai smentita.
- d) Nessuna delle precedenti

5. Il teorema di Cantor in simboli matematici è

- a) $\text{Card}(\wp(A)) = 2^{\text{Card}(A)}$
- b) $\text{Card}(\wp(A)) < \text{Card}(A)$
- c) $(\text{Card}(A) = \aleph_0) \Rightarrow (\text{Card}(\wp(A)) = \aleph_1)$
- d) Nessuna delle precedenti

6. Quale delle seguenti non è una funzione base?

- a) zero
- b) successore
- c) somma
- d) Nessuna delle precedenti: sono tutte funzioni base

7. Quale delle seguenti affermazioni riguardanti le MT è vera?

- a) Le MT sono un modello astratto di un calcolatore, solo se tale calcolatore è elettronico
- b) Le MT sono un modello astratto di un calcolatore, solo se tale calcolatore è meccanico
- c) Le MT sono un modello astratto di un calcolatore, solo se tale calcolatore è deterministico
- d) Nessuna delle precedenti

$\chi \cdot \delta / 10$

numerabile:

$$\text{Card}(A) = \text{Card}(N)$$

$$f: N \rightarrow N \quad \psi_N \quad \text{Card}(\psi_N) = \text{Card}(R)$$

corrispondenze
biunivoca =
iniettiva + suriettiva

teorema di Cantor:

$$\text{Card}(\wp(A)) > \text{Card}(A)$$

f base:

- zero
- successore
- proiezione

8. Perché la minimizzazione introduce la parzialità nelle funzioni ricorsive?

- a) Perché a volte implica una divisione per zero, che non è definita
- b) Perché a volte implica una differenza con risultato negativo, che non è definito in N
- c) Perché a volte la condizione su cui si basa non si verifica mai
- d) Nessuna delle precedenti

$\log(P=0)$

- un stato iniziale
- con finali diverse
- più finali

9. Una MT è deterministica. Questo vuole dire che nella sua tavola...

- a) ci deve essere una istruzione con uno stato iniziale
- b) ci deve essere una istruzione con uno stato finale
- c) ci possono essere due istruzioni con la medesima configurazione
- d) Nessuna delle precedenti

$\chi: 10/10 + 1$

$\text{Card}(I) <$
 ~~$\text{Card}(I) <$~~
 $\text{Card}(R)$

10. Secondo l'ipotesi del continuo...

- a) c'è un insieme infinito (numerabile) di insiemi con cardinalità intermedia tra \aleph_0 e \aleph_1
- b) non esiste un insieme con cardinalità intermedia tra \aleph_0 e \aleph_1
- c) c'è un insieme infinito (non numerabile) di insiemi di cardinalità intermedia tra \aleph_0 e \aleph_1
- d) Nessuna delle precedenti

Parte B - Domanda aperta

Dimostrare che l'*halting problem* non è decidibile.

problema dell'arresto

Halting problem: dato il codice di una MT e le sue relativa input, se tempo di esecuzione della MT rispettando I è finito sarebbe un problema risolvibile (decidibile), ovvero non è risolvibile tramite MT.

Dimostrazione: supponiamo per assurdo che esiste una macchina H che sia in grado di dare una risposta ($t, 0-1$) in un tempo finito.

- Sia C_H le codice del codice di una MT e input I
- Definiamo la seguente macchina $H(C_H, I)$ come:

$$H(C_H, I) = \begin{cases} 0 & \text{se } MT(I) \text{ termina} \\ 1 & \text{se } MT(I) \text{ non termina} \end{cases}$$

- Definiamo la $H'(C_H)$ come: Un caso particolare delle F compute da H

$$H'(C_H) = \begin{cases} 0 & \text{se } H(C_H, C_H) = 0, \text{ ovvero } MT(C_H) \text{ termina} \\ 1 & \text{se } H(C_H, C_H) = 1, \text{ ovvero } MT(C_H) \text{ non termina} \end{cases}$$

- Infine definiamo $Z(C_H)$ come:

$$Z(C_H) = \begin{cases} 0 & \text{Se } H'(C_H) = 1 \\ 1 & \text{Se } H'(C_H) = 0 \end{cases}$$

Proviamo a calcolare $Z(C_H)$, ho 2 possibilità:

- a) $Z(C_H) = 0$ "termina" $\Leftrightarrow H'(C_H) = 1 \Leftrightarrow H(C_H, C_H) = 1$, ovvero $Z(C_H) = 1$ "non termina" ~~risultato~~
- b) $Z(C_H) = 1$ "non termina" $\Leftrightarrow H'(C_H) = 0 \Leftrightarrow H(C_H, C_H) = 0$, ovvero $Z(C_H) = 0$ "termina" ~~risultato~~

In ogni caso mi affido a un logorio, dunque siamo costretti a respingere l'affermazione.

In conclusione non esiste una macchina H.

1. Quale dei seguenti insiemi **NON** è numerabile? $\neq \text{Card}(\mathbb{N})$

- a) L'insieme delle funzioni $N \rightarrow N$ computabili
- b) L'insieme delle stringhe binarie infinite che contengono solo "0" da un certo punto in poi
- c) L'insieme delle stringhe binarie infinite che contengono solo "1" da un certo punto in poi
- ~~d) L'insieme delle funzioni $N \rightarrow N$ non computabili~~

2. Dati due insiemi A e B, $\text{Card}(A) < \text{Card}(B)$ vuol dire che

- ~~a) esiste una funzione iniettiva da A a B ma non una suriettiva~~
- b) esiste una funzione suriettiva da B ad A ma non iniettiva
- c) esiste una funzione iniettiva da B ad A ma non una suriettiva
- d) esiste una funzione suriettiva da A a B ma non iniettiva

3. Se A è un insieme finito, $\text{Card}(\wp(A)) = 2^{\text{Card}(A)}$. Perché la base della potenza è 2?

- a) Perché nei computer si usa la logica binaria.
- b) Perché *tertium non datur*.

~~c) Perché dato un elemento a e un sottoinsieme I di A, 2 sono i casi: $a \in I$ oppure $a \notin I$.~~

- d) Non si sa perché, ma finora l'equazione non è stata mai smentita.

4. L'*halting problem* è **indecidibile**, ossia

- ~~a) una MT non riesce mai a decidere se un certo programma con un certo input termina~~
- ~~b) una MT non riesce sempre a decidere se un certo programma con un certo input termina~~
- c) una MT riesce sempre a decidere se un certo programma con un certo input termina
- d) una MT non riesce mai a eseguire un halt

5. Il teorema di Cantor in simboli matematici è

- a) $\text{Card}(\wp(N)) = \aleph_0$
- b) $\text{Card}(\wp(N)) = \aleph_1$
- ~~c) $\text{Card}(\wp(A)) > \text{Card}(A)$~~
- d) $\text{Card}(\wp(A)) = 2^{\text{Card}(A)}$

6. Quale delle seguenti affermazioni riguardanti le MT è vera?

- a) Le MT sono elettroniche
- b) Le MT sono meccaniche
- ~~c) Le MT possono essere elettroniche~~
- d) Le MT non possono essere meccaniche

7. Quale delle seguenti operazioni introduce la parzialità nelle funzioni ricorsive?

- a) La ricorsione
- b) La composizione
- c) La sottrazione
- ~~d) La minimizzazione~~

iniettive:

$$\begin{aligned} &\forall a_1, a_2 \in A \text{ con } a_1 \neq a_2, \\ &\exists b_1, b_2 \in B: f(a_1) \neq f(a_2) \\ &\text{Card}(A) < \text{Card}(B) \end{aligned}$$

8. Perché quella di Church è una tesi e non un teorema?

- a) Perché siamo ancora in attesa di una dimostrazione rigorosa.
- ~~b) Perché non esiste una definizione generale del concetto di computabilità.~~
- c) Perché non esiste una definizione generale del concetto di ricorsività generale.
- d) Perché Church voleva distinguersi da Turing.

9. Una MT è deterministica. Questo vuole dire che nella sua tavola...

- ~~a) ci deve essere una istruzione con uno stato finale~~
- ~~b) ci deve essere una istruzione con uno stato iniziale~~
- c) ci possono essere due istruzioni con la medesima configurazione
- d) ci devono essere due istruzioni con la medesima configurazione

10. Secondo l'ipotesi del continuo...

- a) esistono infiniti punti (numerabili) tra 0 e 1
- b) esistono infiniti punti (non numerabili) tra 0 e 1
- c) esiste un insieme con cardinalità intermedia tra \aleph_0 e \aleph_1
- ~~d) non esiste un insieme con cardinalità intermedia tra \aleph_0 e \aleph_1~~

Parte B – Domanda aperta $\frac{7}{10}$

Enunciare e dimostrare il teorema di Cantor.

Teorema: Sia A un insieme (finito o infinito) allora:

$$\text{Card}(A) < \text{Card}(\wp(A))$$

Quanto significa che:

esiste $f: A \rightarrow \wp(A)$ iniettiva $\Rightarrow \text{Card}(A) \leq \text{Card}(\wp(A))$ e
non esiste $f: A \rightarrow \wp(A)$ bisettiva $\Rightarrow \text{Card}(A) = \text{Card}(\wp(A))$

che f bisettiva è una funzione iniettiva e suriettiva e dunque eguale a dirsi che:

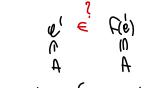
non esiste $f: A \rightarrow \wp(A)$ suriettiva $\Rightarrow \text{Card}(A) < \text{Card}(\wp(A))$

Osservazione: supponiamo esiste una $f: A \rightarrow \wp(A)$ suriettiva:

~~x esistono~~

cioè per ogni $A' \subseteq A$ con $A' \in \wp(A)$ esiste $a \in A$ t.c. $f(a) = A'$

ci chiediamo $a \in f(a)$?



costruiamo $B = \{x \in A : x \notin f(x)\}$ essendo f suriettiva, esiste $b \in A$ t.c. $f(b) = B$

ci chiediamo $b \in B$?

• se $b \in B$ allora $b \notin f(b)$ ma per come è definito B $b \notin f(b) \notin B$

contradiction

• se $b \notin B$ allora $b \notin f(b)$ ma per come è definito B $b \in B$

contradiction

Quindi non esiste una $f: A \rightarrow \wp(A)$ suriettiva.

Parte A – Questionario 12/20

1. Quale dei seguenti insiemi è numerabile?
 a) L'insieme delle funzioni computabili
 b) L'insieme delle funzioni non computabili X
 c) L'insieme di tutte le funzioni $N \rightarrow N$
 d) L'insieme di tutte le funzioni $N^2 \rightarrow N$

$$\begin{aligned} \text{Card}(\Psi) &= \text{Card}(\mathbb{R}) \\ \text{Card}(\Psi_c) &= \text{Card}(\mathbb{Q}) \\ \text{Card}(\Psi_{\text{nc}}) &= \text{Card}(\mathbb{R}) \end{aligned}$$

2. Se per un problema esiste un algoritmo che lo risolve, allora
 a) quell'algoritmo è l'unico a risolverlo
X b) esiste un insieme numerabile di algoritmi che lo risolvono
 c) esiste un insieme non numerabile di algoritmi che lo risolvono
 d) esiste un insieme finito di algoritmi che lo risolvono

3. In che senso il nastro della Macchina di Turing (MT) è infinito?
 a) La MT è un concetto teorico che non ha nessun riscontro nella realtà
 b) Per computare una funzione serve una memoria infinita
X c) Supponiamo di non esaurire mai la memoria della macchina
X d) Supponiamo di avere un insieme numerabile di celle di memoria

4. L'halting problem è indecidibile, ossia
 a) una MT non riesce mai a decidere se un certo programma con un certo input termina
X b) non esiste una MT che possa computare, dato un qualsiasi programma e un qualsiasi input, se l'esecuzione terminerà o meno
 c) non si riesce a formulare il problema dell'arresto sotto forma di funzione matematica
 d) se dato da risolvere a una MT, questa non terminerà mai

5. Il teorema di Cantor in simboli matematici è
X a) $\text{Card}(A) < \text{Card}(\wp(A))$
 b) $\text{Card}(\wp(N)) = \aleph_0$
 c) $\text{Card}(\wp(N)) = \aleph_1$
 d) $\text{Card}(\wp(A)) = 2^{\text{Card}(A)}$

6. La Gödelizzazione delle MT è di fatto...
X a) una codifica delle MT
 b) un restrinzione dell'insieme delle MT
 c) una trasformazione delle MT in funzioni matematiche
 d) una costruzione dell'insieme delle parti delle MT

7. Quale delle seguenti affermazioni è falsa?
X a) Se una funzione è in RG, allora è in RP.
 b) Se una funzione è in RG, allora potrebbe essere parziale.
 c) Se una funzione è basata sulla minimizzazione, allora è in RG.
X d) Se una funzione è in RP, allora è sicuramente totale.

8. Quale delle seguenti funzioni non è una funzione base?

- X a) L'identità
X b) La somma
 c) La funzione zero
 d) $P^3_3(x,y,z)$

9. Devo dimostrare che $\text{Card}(A) < \text{Card}(B)$, con A e B infiniti. Quale dei seguenti passi è parte della dimostrazione?

- X a) Dimostro che esiste una $f: A \rightarrow B$ suriettiva
X b) Dimostro che esiste una $f: A \rightarrow B$ iniettiva
 c) Dimostro che esiste una $f: A \rightarrow B$ biunivoca
X d) Dimostro che non esiste una $f: A \rightarrow B$ suriettiva

10. Quali dei seguenti comportamenti posso programmare in una MT che computa $f(x) = \perp$?

- a) La MT scrive 0 sul nastro e termina.
 b) La MT scrive 1 sul nastro e termina.
 c) La MT termina subito appena inizia la computazione.
X d) La MT cancella tutto il nastro e termina.

- loop MT
 - nelle sol. NESTIO

Parte B – Domanda aperta 4/10

Scrivere il codice della Macchina di Turing che computa la seguente funzione $f: N \rightarrow N^2$

$$f(x) = \begin{cases} \perp & \text{se } x = 0 \\ (0, x) & \text{se } x \neq 0 \end{cases}$$

La codifica dei numeri naturali è la solita: n viene rappresentato con $n+1$ barre verticali "|". Usare "s₀" per indicare la cella vuota, " q_0 " per lo stato iniziale, " q_f " per lo stato finale.

q_0	S_0	S_0	C	q_{0x}	
q_0			0	q_{0x}	
q_{0x}	S_0	S_0	C	q_{0x}	controllo F particolare
q_{0x}			0	q_{0x}	
q_{0x}			0	q_{0x}	stabilimento e cancella (0x)
q_{0x}			0	q_{0x}	
q_{0x}	S_0	S_0	S	q_{0x}	
q_{0x}			S	q_{0x}	
q_{0x}	S_0	S_0	S	q_{000}	
q_{000}	S_0		C	q_f	

Dimostrazione definitiva di Euclide

corretto, probabilmente

dividendo	divisore	quotiente	resto
x :	q	q	$r \Rightarrow x = q \cdot q + r$
y :	r	q'	$r' \Rightarrow y = r \cdot q' + r'$

Ora 4) $\{ \text{divisori di } x \text{ e } y \} = \{ \text{divisori di } y \text{ e } r \}$

Hip: d divide sia x sia y

- a) d divide y
- b) d divide $q \cdot q$
- c) d divide x
- d) d divide $x - q \cdot q$
- e) d divide $r = x - q \cdot q$

$\Rightarrow d$ divide $r \wedge d$ divide y (e AND v)

d divide sia r sia y

Ora 5) $\{ \text{divisori di } x \text{ e } r \} = \{ \text{divisori di } y \text{ e } r \}$

Hip: d divide sia y sia r

- a) d divide y
- b) d divide $q \cdot q$
- c) d divide r
- d) d divide $q \cdot q + r$
- e) d divide $x = q \cdot q + r$

$\Rightarrow (e \text{ AND } v) \Rightarrow d$ divide sia x sia y