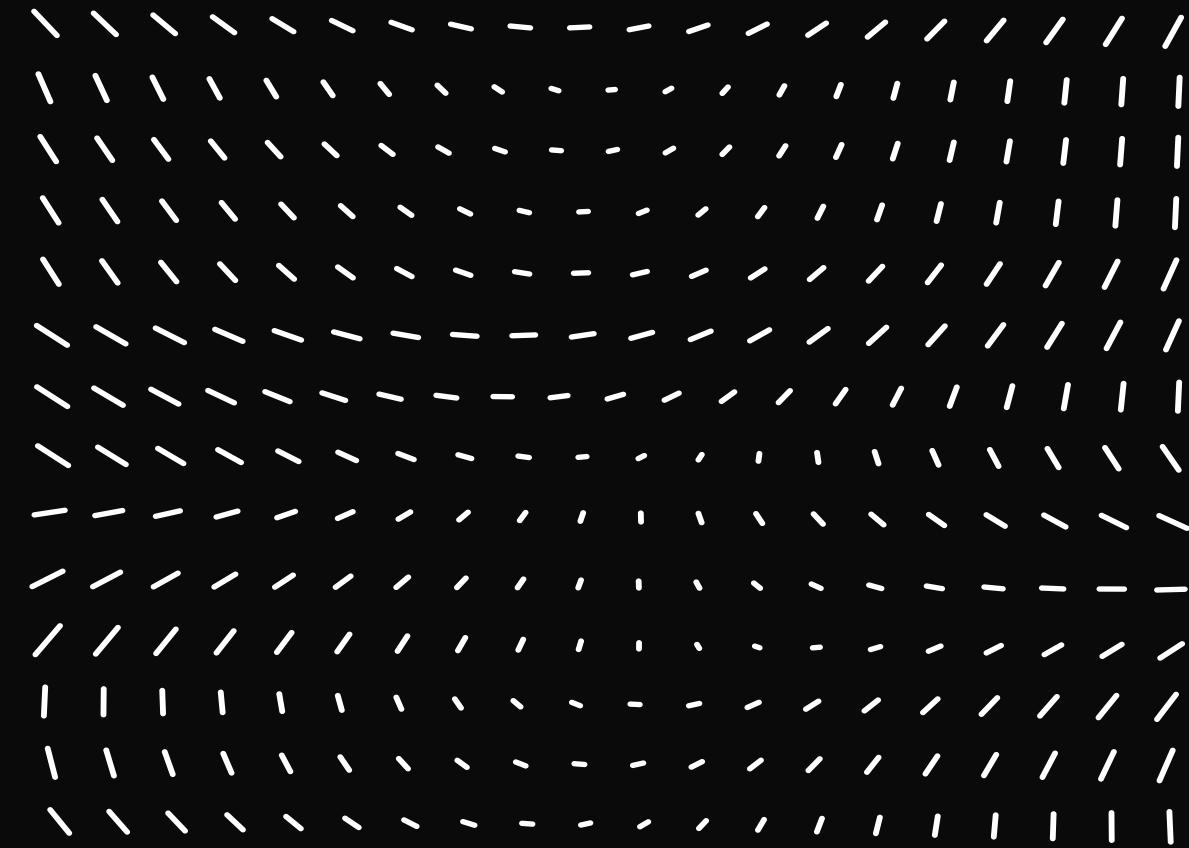


Linguaggi e Computabilità

879276 * A.A. 2022/2023



ES. Dimostrare che L non è regolare $L = \{0^n 1^n \mid n \geq 1\}$

Supponiamo che L sia REG, allora sia $n \in \mathbb{N}$ la costante del PL, sia $w = 0^n 1^n \in L$

$$w = xyz \text{ t.c. } x = 0^{n-1}, y = 0, z = 1^n$$

Allora, per il PL, deve valere $xy^kz \in L \quad \forall k \geq 0$

Ma, per $k=0$ dovrebbe essere $xz \in L$ dove $xz = 0^{n-1} 1^n$ il che è falso

ES. Dimostrare che L non è regolare $L = \{w \in \{0, 1\}^* \mid w = w^R\}$

Supponiamo che L sia REG, allora sia $n \in \mathbb{N}$ la costante del PL, sia $w = 0^n 10^n \in L$

$$w = xyz \text{ t.c. } x = 0^n, y = 0, z = 10^n$$

Allora, per il PL, deve valere $xy^kz \in L \quad \forall k \geq 0$

Ma, per $k=0$ dovrebbe essere $xz \in L$ dove $xz = 0^{n-1} 10^n$ il che è falso

ES. Dimostrare che L non è regolare $L = \{0^n 1^m 2^n \mid n, m \geq 1\}$

Supponiamo che L sia REG, allora sia $n \in \mathbb{N}$ la costante del PL, sia $w = 0^n 1^m 2^n \in L$

$$w = xyz \text{ t.c. } x = 0^{n-1}, y = 0, z = 1^m 2^n$$

Allora, per il PL, deve valere $xy^kz \in L \quad \forall k \geq 0$

Ma, per $k=0$ dovrebbe essere $xz \in L$ dove $xz = 0^{n-1} 1^n 2^n$ il che è falso

ES. Dimostrare che L non è regolare $L = \{ 0^n 1^m \mid n \geq m \}$

Supponiamo che L sia REG, allora sia $n \in \mathbb{N}$ la costante del PL, sia $w = 0^n 1^m \in L$

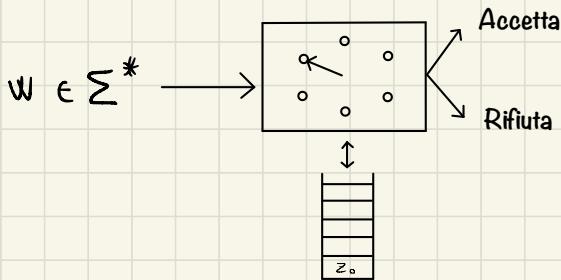
$$w = xyz \quad \text{t.c.} \quad x = \epsilon, y = 0^n, z = 1^m$$

Allora, per il PL, deve valere $xy^kz \in L \quad \forall k \geq 0$

Ma, per $k=0$ dovrebbe essere $xz \in L$ dove $xz = 1^m$ il che è falso

Automati a Pila [Pushdown Automata - PDA]

ϵ -NFA + Pila (stack)



Un automa a pila [PDA] è una settupla: $P = \{ Q, \Sigma, \Gamma, \delta, q_0, z_0, F \}$ dove:

Q : Insieme finito e non vuoto di stati

$q_0 \in Q$: Stato iniziale

Σ : Alfabeto dei simboli di input

$z_0 \in \Gamma$: Simbolo inizialmente presente nella pila

Γ : Alfabeto dei simboli dello stack

$F \subseteq Q$: Insieme degli stati finali

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$$

Il passaggio da uno stato q_0 a uno stato q_1 , avviene così:

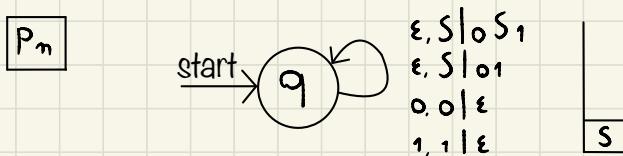
$$q_0 \xrightarrow{\text{Input, pop | push}} q_1$$

L'input viene rimosso dalla stringa

Creazione di un PDA per pila vuota da una CFG

1 Dec 2022

CFG: $S \rightarrow 0S_1 \mid 01$



Esempio di derivazione $S \Rightarrow 0S_1 \Rightarrow 0011$

$(q, 0011, S) \vdash (q, 0011, 0S_1) \vdash (q, 011, S_1)$
 $\vdash (q, 011, 011) \vdash (q, 11, 11) \vdash (q, 1, 1) \vdash (q, \epsilon, \epsilon)$

Automa a Pila Deterministico [DPDA]

14 Dec 2022

I PDA non deterministici accettano tutti e soli i CFL

Un DPDA accetta tutti i linguaggi regolari

Un PDA $P = \{Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\}$ è deterministico se:

1. $|\delta(q, a, x)| \leq 1 \quad \forall q \in Q, \forall a \in \Sigma \cup \{\epsilon\}, \forall x \in \Gamma$

2. Se $|\delta(q, a, x)| \neq 0$ per qualche $a \in \Sigma$ allora $|\delta(q, \epsilon, x)| = 0$

Proprietà di Prefisso [Prefix-free]

Un linguaggio L ha la proprietà del prefisso (cioè è prefix-free) se $\nexists x, y \in L$ tali che $x \neq y$ e x è prefissa di y

Proprietà del Prefisso - esempi

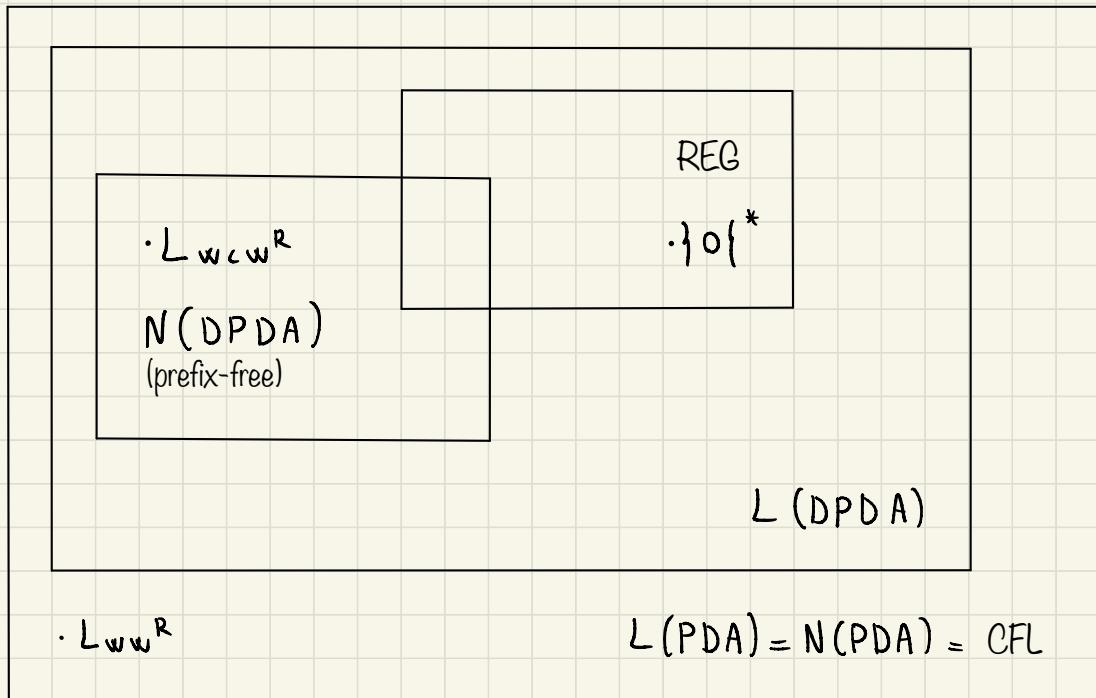
$$L = \{\circ\}^* = \{\epsilon, \circ, \circ\circ, \circ\circ\circ, \dots\}$$

$x \neq y$ e x è prefissa di y . L non è prefix-free.

Potenza Computazionale di un DPDA

Un linguaggio L è $N(P)$ per un DPDA P se e solo se L è prefix-free, ed è $L(P')$ per un DPDA P'

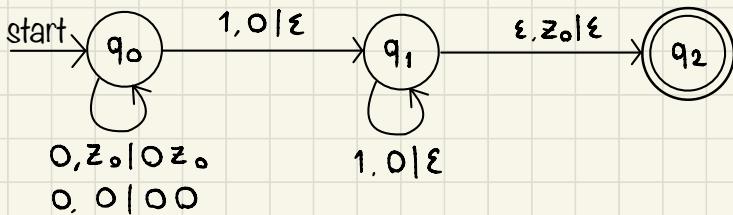
Schema generale della Potenza Computazionale



Costruzione di un DPDA - esercizi

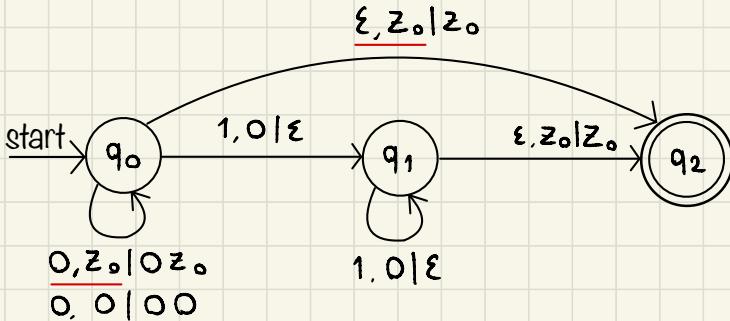
$$L = \{ 0^n 1^n \mid n \geq 1 \}$$

L è prefix-free



$$L = \{ 0^n 1^n \mid n \geq 0 \}$$

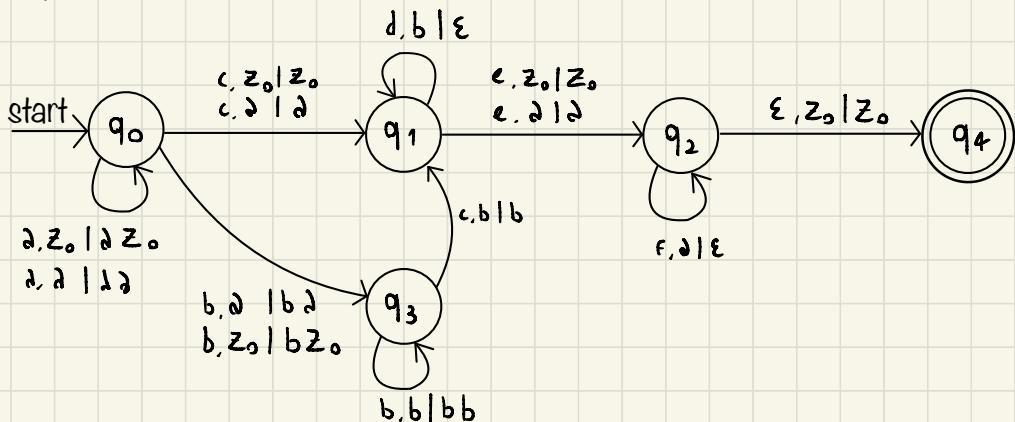
L non è prefix-free



L'automma non è deterministico. È impossibile creare un DPDA per L

$$L = \{ a^n b^m c d^m e f^n \mid n, m \geq 0 \}$$

L è prefix-free

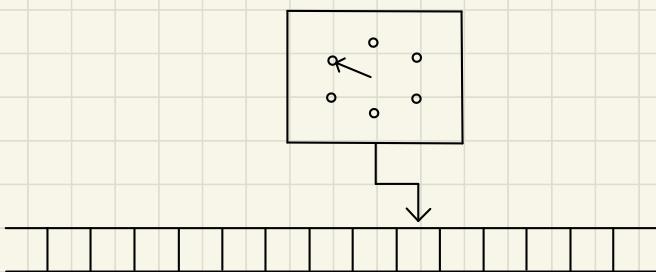


Macchine di Turing

[MdT] [DTM]

Linguaggi ricorsivamente enumerabili

15 Dec 2022



Una Macchina di Turing è una settaglia: $M = \{ Q, \Sigma, \Gamma, \delta, q_0, B, F \}$ dove:

Q : Insieme finito e non vuoto di stati

$q_0 \in Q$: Stato iniziale

$\Sigma \subset \Gamma$: Alfabeto dei simboli di input

$B \in \Gamma \setminus \Sigma$: Simbolo di blank

Γ : Alfabeto dei simboli del nastro

$F \subseteq Q$: Insieme degli stati finali

$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{ L_{left}, R_{right} \}$ funzione parziale di transizione

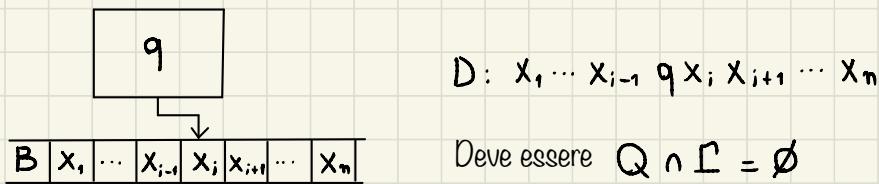
Linguaggi Ricorsivamente Enumerabili [RE]

Linguaggi di tipo 0

Un linguaggio L è Ricorsivamente Enumerabile (RE) se esiste una Macchina di Turing M

tale che $L = L(M)$ $L(M) = \{ w \in \Sigma^* \mid M \text{ accetta } w \}$

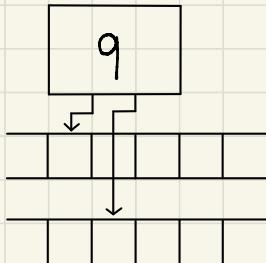
Rappresentazione dello Stato - descrizione istantanea



Macchine di Turing multinastri

10 Jan 2023

Un modello alternativo alla MdT è quello che presenta k nastri con un k fissato



Ogni linguaggio accettato da una MdT multinastro è
Ricorsivamente Enumerabile

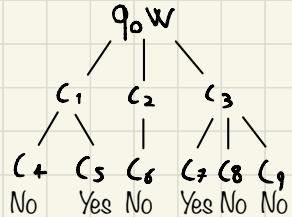
Una MdT con un nastro singolo può simulare una MdT multinastro dividendo il suo nastro in $2k$ "fette".

Macchina di Turing con delta non deterministica [NTM]

$$\delta : Q \times \Gamma \rightarrow P(Q \times \Gamma \times \{\text{L.R.}\}) \quad \text{insieme di stati}$$
$$\delta(q, a) = \{(p_1, b_1, L), (p_2, b_2, R) \dots (p_n, b_n, R)\}$$

Alberi di Derivazione di una MdT non deterministica

Alla computazione di una MdT non deterministica corrisponde un'albero di computazione i cui nodi sono configurazioni della MdT e la cui radice corrisponde alla configurazione data la stringa di input w



La macchina accetta la stringa w se e solo se esiste una computazione che la accetta

Una MdT deterministica con due nastri può simulare una MdT non deterministica, utilizzando il secondo nastro come fosse una coda che contiene configurazioni della MdT non deterministica.

Insieme dei Problemi di decisione - extra

Esistono due classi di complessità, **P** e **NP**

P è la classe dei problemi di decisione dove data una stringa di input la risposta è "sì" o "no"

Sono risolvibili con MdT deterministiche [DTM] in un numero di passi polinomiale

NP è la classe dei problemi di decisione risolvibili con MdT non deterministiche [NTM]

$$P \subseteq NP \quad P \neq NP ? \quad P = NP ?$$

Una DTM può risolvere i problemi di classe NP simulando una NTM in tempo polinomiale?

Se prendo tutte le stringhe che codificano un problema la cui risposta è "sì" e le metto in un insieme, quell'insieme è un linguaggio, perché è un insieme di stringhe.

Riconoscere se una stringa appartiene al linguaggio è equivalente a verificare la risposta del problema codificato in quella stringa

Restrizioni delle MdT

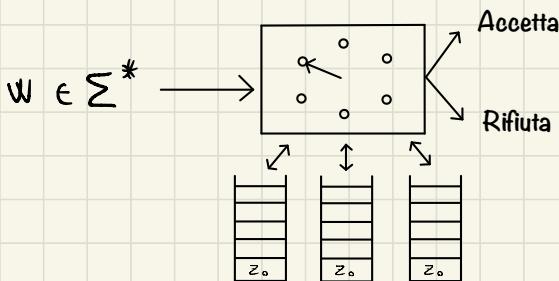
Ogni linguaggio che è accettato da una MdT M_2 è accettato anche da una MdT M_1 , con i seguenti vincoli:

- La testina di M_1 , non va mai a sx della posizione iniziale
- M_1 , non scrive mai B

Le possibili combinazioni di una cella e la sua sottostante sono finite, per cui è possibile creare un alfabeto che le codifichi

(q, U)			
U	x ₀	x ₁	x ₂
D	*	x ₋₁	x ₋₂

Macchine Multi-Stack (= DPDA + k stack)

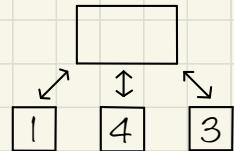


La funzione di transizione è uguale a quella di un DPDA, tenendo conto dei simboli per le k pile

Una Macchina Multi-Stack con 2 pile può accettare tutti i linguaggio RE (simulare una MdT)

Anche limitando l'alfabeto scrivibile sulle pile ad un solo simbolo "x", con k pile si riesce a simulare una MdT. Ogni pila può essere vista come un contatore di cui la macchina non è in grado di sapere il contenuto esatto, ma può controllare se sia uguale o maggiore di 0, ed incrementarne o decrescerne il valore.

Ogni linguaggio RE è accettato da una macchina a 3 contatori
 Ogni linguaggio RE è accettato da una macchina a 2 contatori



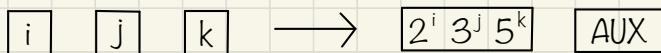
Con 3 contatori posso simulare un DPDA con 2 stack.

Se uno stack del DPDA contiene $x_1, x_2 \dots x_n$ lo posso simulare con un contatore che contiene il numero in base r (dove $r-1$ è la cardinalità dell'alfabeto dello stack) così formato:

$$x_n^{r-1} + \dots + x_2 r + x_1 \quad (\text{intero in base } r)$$

Sostituisco le due pile del DPDA con due contatori, e utilizzo un terzo contatore d'appoggio per svolgere le operazioni (pop, push...)

Con 2 contatori posso simulare una macchina con 3 contatori.



Calcolabilità

Problema Ciao, mondo (Il caratteri)

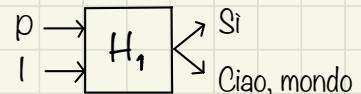
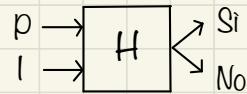
Input: programma P, input I per P

Output: "Si" se P con input I stampa "Ciao, mondo", "No" altrimenti

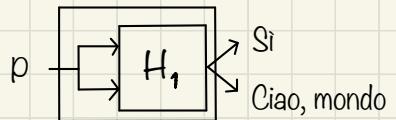
Questo problema è indecidibile

Per assurdo, supponiamo che esista la macchina H

Allora, esiste anche la macchina H_1 che stampa "Si" se H stampa "Si", mentre se H stampa "No", stampa "Ciao, mondo"



Se esiste H_1 , allora esiste H_2 , che prende input $P=I$, legge e copia P , e simula H_1 , usando la copia di P come I



Posso dare H_2 in input ad H_2

In questo caso, H_2 con in input H_2 stamperebbe "Si" se H_2 (prendendo in input se stesso perché $H_2=P=I$) a un certo punto stampa "Ciao, mondo", ma è un paradosso.

Allora H_2 con in input H_2 deve stampare "Ciao, mondo", ma lo stampa se e solo se H_2 con input se stesso non stampa "Ciao, mondo", il che è un ulteriore paradosso.

Si può dunque concludere che H_2 non esiste, ma se H_2 non esiste, allora non esiste neanche H_1 , e se H_1 non esiste, allora non esiste neanche H

Non esiste un programma a cui possa dare un programma P qualsiasi con un input qualsiasi che mi dica se P stamperà o no "Ciao, mondo"

Quanto sono frequenti i problemi indecidibili?

11 Jan 2023

$$f_P: \mathbb{N} \rightarrow \mathbb{N} \quad \text{esistono } |\mathbb{N}|^{|\mathbb{N}|} \text{ funzioni così definite. } |\mathbb{N}| = \aleph_0$$

Un $P \in \text{Programmi}$, sequenza di bit, può essere codificato come un numero naturale
Esistono quindi \aleph_0 Programmi

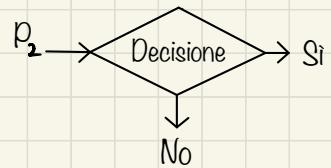
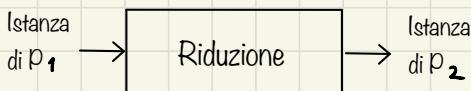
Per l'ipotesi del continuo, i Programmi sono molti meno rispetto alle funzioni
Sono dunque molti di più i problemi (funzioni) indecidibili rispetto a quelli decidibili

Riduzione tra problemi

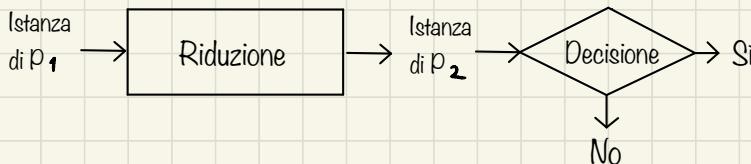
Sia P_2 un problema indecidibile, e vogliamo dimostrare che anche un altro problema P_2' lo sia

Se P_2 fosse decidibile, allora \exists un algoritmo che lo decide

Prendiamo una struttura di questo tipo:



Se la risposta di P_1 è "Si", la risposta di P_2 su quell'istanza è "Si", se la risposta di P_1 è "No", lo sarà anche per l'istanza di P_2 , se la stringa non è formata bene (non codifica correttamente un'istanza di P_1), lo stesso varrà per l'istanza di P_2



Se esistesse questa struttura, esisterebbe dunque una struttura in grado di ricevere istanze di P_1 e restituire "Si" oppure "No", ma essa non esiste. Dunque se è possibile effettuare la riduzione, allora segue che non esiste una struttura in grado di restituire "Si" o "No" ad un'istanza di P_2 , quindi P_2 non è decidibile

Enumerazione delle stringhe binarie

13 Jan 2023

Stringhe binarie	ϵ	0	1	00	01	10	11	...
Numeri naturali	1	2	3	4	5	6	7	...

Questa funzione di associazione è una biiezione

Stringa $w \rightarrow 1w$ (letto come numero naturale in base binaria)
Indice $n = 1w \rightarrow w$

Codifica in Binario di una Macchina di Turing

Data M una MdT: $(Q, \{0, 1\}, \Gamma, \delta, q_1, B, \{q_2\})$ dove

$$\bullet Q = \{q_1, q_2 \dots q_r\}$$

$$\bullet \Gamma = \{x_1, x_2 \dots x_s\} \quad \text{con } X_1 = 0, X_2 = 1, X_3 = B$$

Indichiamo L come D_1 ed R come D_2

Codifica di $\delta(q_i, x_j) = (q_k, X_\ell, D_m)$

$$\Rightarrow (q_i, x_j, q_k, X_\ell, D_m) \Rightarrow (i, j, k, \ell, m)$$

$$\Rightarrow 0^i 10^j 10^k 10^\ell 10^m \quad \text{Usando ! come separatore}$$

Codifica di δ con i casi $C_1, C_2 \dots C_n$

$$\Rightarrow C_1 || C_2 || \dots || C_n \quad \text{Usando || come separatore}$$

Esempio di codifica di MdT $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$

$$\underbrace{0100100010100}_{\delta(q_1, X_1)} \underbrace{|| 0001010100100}_{\delta(q_1, X_2)} \underbrace{|| 0001000100010010}_{\delta(q_1, X_3)} \\ \delta(q_1, X_1) = (q_3, X_1, R) \quad \delta(q_1, X_2) = (q_1, X_2, R) \quad \delta(q_1, X_3) = (q_3, X_2, L)$$

Se si può codificare M in una stringa binaria w, w potrà essere numerata come descritto prima. Dunque se $\text{cod}(M) = w$, e $w = w_i$, si può dire che $M = M_i$

Date le regole di codifica di una MdT, è chiaro che non tutte le stringhe binarie codificano una MdT. A tutte queste stringhe che non codificano una MdT, viene assegnata una (la stessa) MdT fittizia che non accetta alcuna stringa.

È dunque possibile definire una relazione biunivoca come per le stringhe binarie:

Numeri naturali	1	2	3	4	5	6	7	...
Stringhe binarie	w_1	w_2	w_3	w_4	w_5	w_6	w_7	...
MdT	M_1	M_2	M_3	M_4	M_5	M_6	M_7	...

Linguaggio di Diagonalizzazione [Ld]

Il Linguaggio di Diagonalizzazione è l'insieme delle stringhe binarie w_i tali che $w_i \notin L(M_i)$

$$L_d = \{ w_i \in \{0,1\}^* \mid w_i \notin L(M_i) \} \quad \text{Autoreferenzialità}$$

È dunque l'insieme delle MdT codificate da una stringa binaria, le quali non accettano se stesse come input

L_d non è un linguaggio RE, cioè non esiste una MdT che lo accetta

		w_j	→				
		1	2	3	4	5	...
M_i	1	0	1	1	0	...	
	2	1	1	0	0	...	
	3	0	0	1	1	...	
	4	0	1	0	1	...	
	5	
	...						

$$D_{ij} = 1 \quad \text{se } w_j \in L(M_i) \\ D_{ij} = 0 \quad \text{altrimenti}$$

Ogni riga della matrice è dunque un vettore binario che rappresenta la funzione caratteristica del linguaggio accettato dalla MdT.

Nelle righe della matrice appaiono tutti i linguaggi RE

Voglio dunque dimostrare che Ld non appare in nessuna riga della matrice

Considero la diagonale della matrice e la complemento

Otengo: **1000...** e chiamo questo linguaggio Ld

Questo perché, sulla diagonale sono indicate con **O** le MdT che non accettano la loro stessa codifica, e con **I** quelle che l'accettano. Complementando la diagonale si ottiene dunque Ld

Il vettore risultato non compare come riga da nessuna parte nella matrice perché, essendo il complemento della diagonale, ha almeno un elemento diverso rispetto a ogni riga.

- Può essere la prima riga? No, purché il primo elemento è diverso
- Può essere la seconda riga? No, perché il secondo elemento è diverso
- Può essere la terza riga? No, perché il terzo elemento è diverso

Dimostrazione per assurdo

Per assurdo, se \exists MdT che accetta Ld , allora compare nell'enumerazione $M_1, M_2 \dots M_n$

Sia M_i la MdT che accetta Ld . Ci chiediamo se w_i appartiene o meno a Ld

1) Se $w_i \in Ld$, allora M_i accetta w_i

Contraddizione

Per definizione di Ld , w_i non può appartenere a Ld

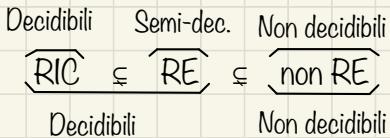
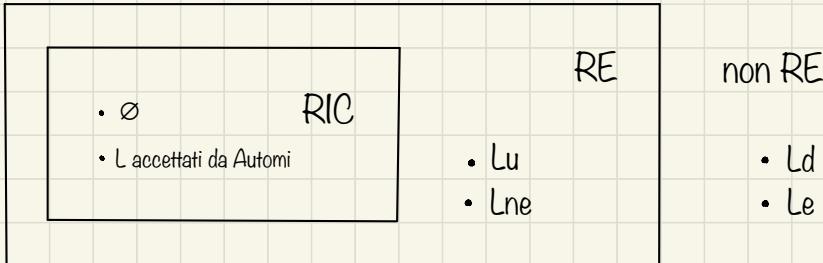
2) Se $w_i \notin Ld$, allora M_i non accetta w_i

Contraddizione

Ma per definizione di Ld , se M_i non accetta w_i , allora $w_i \in Ld$

Entrambi i casi generano una contraddizione, per cui \nexists MdT che accetta Ld

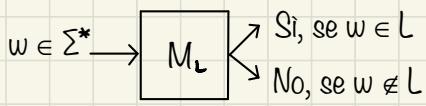
Schema Generale dei Linguaggi RE e non RE



Ci sono due diversi sistemi di denominazione dei linguaggi

Linguaggi Ricorsivi [RIC]

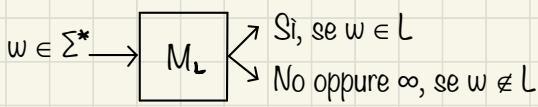
L linguaggio ricorsivo, allora \exists MdT M_L tale che



La MdT si ferma e produce un risultato qualsiasi sia la stringa di input

Linguaggi RE non Ricorsivi

L linguaggio RE non-ricorsivo, allora \exists MdT M_L tale che



La MdT potrebbe continuare all'infinito se la stringa di input non appartiene al linguaggio

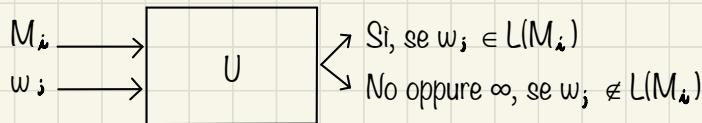
Halting Problem

È indecidibile sapere se una determinata MdT si fermerà oppure no dato un certo input

Macchina di Turing Universale

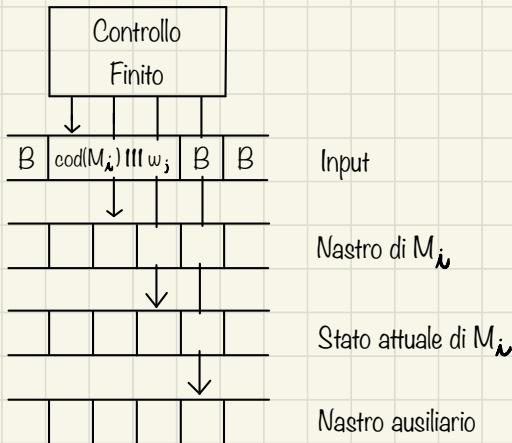
[U]

14 Jan 2023



La MdT universale è un simulatore di MdT che preso in input una MdT M_i e una stringa w_j si comporta esattamente come farebbe M_i con in input w_j .

Notare che la stringa di input della MdT U è una sola: $\text{cod}(M_i) \mid\mid w_j$



La MdT U è una MdT a 4 nastri.

Per prima cosa, copia sul secondo nastro tutto quello che trova da $\mid\mid$ in poi fino ai B , scrive sul terzo nastro lo stato iniziale di M_i , dopodiché inizia la simulazione.

Legge dal primo nastro i casi della Delta di M_i e si muove sul secondo nastro di conseguenza. Così per il resto della stringa, seguendo il comportamento di M_i .

Linguaggio Universale

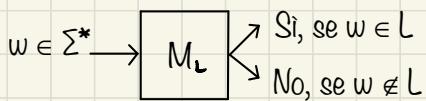
[Lu]

Il Linguaggio Universale è il linguaggio accettato dalla MdT universale.

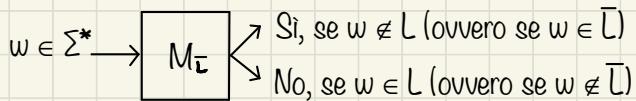
$$Lu = \{ (M_i, w_j) \mid w_j \in L(M_i) \} = \{ \text{cod}(M_i) \mid\mid w_j \in \{0, 1\}^* \mid w_j \in L(M_i) \}$$

Complemento di un Linguaggio RIC

Data una MdT M_L che accetta un linguaggio ricorsivo L , che si comporta in questo modo:



Costruiamo la MdT $M_{\bar{L}}$ che si comporta in questo modo:

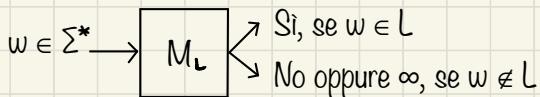


$M_{\bar{L}}$ è dunque una macchina che accetta il complemento di L , e si ferma sempre.

Quindi anche il complemento di L ricorsivo, è un linguaggio \bar{L} ricorsivo.

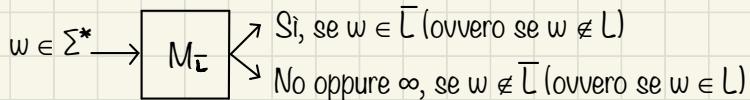
Complemento di un Linguaggio RE non Ricorsivo

Data una MdT M_L che accetta un linguaggio RE L , che si comporta in questo modo:

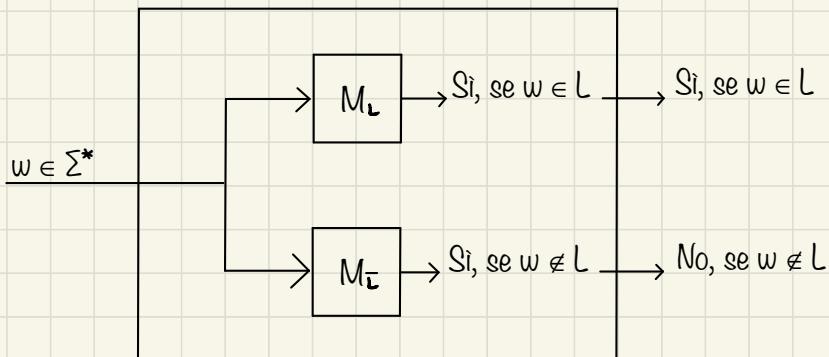


Supponiamo che anche \bar{L} sia RE ma non ricorsivo.

Costruiamo la MdT $M_{\bar{L}}$ che si comporta in questo modo:



Costruiamo quindi una MdT M tale che:



M simula entrambe le macchine M_L e $M_{\bar{L}}$ e guarda quale delle due finisce prima.

Ma questo è impossibile, perché in questo caso esisterebbe una macchina M in grado di terminare sempre e riconoscere il linguaggio RE L.

Si conclude che il complemento di un linguaggio RE non RIC L è un linguaggio non RE.

Insiemi di MdT o Algoritmi

$$Le = \{ \text{MdT } M \mid L(M) = \emptyset \} \quad Lne = \{ \text{MdT } M \mid L(M) \neq \emptyset \}$$

Le e Lne sono insiemi di MdT. Ma, condicando MdT sotto forma di stringhe binarie,

Le e Lne sono insiemi di stringhe binarie, quindi linguaggi.

Posso dunque considerare linguaggi degli insiemi di MdT con particolari caratteristiche.

Le è un linguaggio RE ma non RIC, ma $Le = \overline{Lne}$, quindi Lne è un linguaggio non RE

Proprietà dei linguaggi RE

Una proprietà dei linguaggi RE è un insieme di linguaggi RE, cioè un insieme di MdT

Una proprietà è banale (trivial) se è l'insieme vuoto oppure se è tutto RE

Teorema di Rice

Ogni proprietà non banale di linguaggi RE è indecidibile