

17/04/2020

La gerarchia di Chomsky:

Data una grammatica $G = (V, \Sigma, R, S)$, può essere classificata in base a come sono fatte le sue regole ($\alpha \rightarrow \beta$):

Tipo 0 $\alpha, \beta \in (\Sigma \cup V)^*$

Tipo 1 $\alpha, \beta \in (\Sigma \cup V)^+$
 $|\alpha| \leq |\beta|$ $\alpha = S$ $\beta = \epsilon$
S non appare mai nella parte destra di una regola

Context -
sensitive $\alpha = uAv$ $\beta = uwv$ $\alpha = S$ $\beta = \epsilon$
 $u \in V$ $v, w \in (\Sigma \cup V)^*$ S non appare mai nella parte destra di una regola
 $w \in (\Sigma \cup V)^+$

Tipo 2 $\alpha \in V$ $\beta \in (V \cup \Sigma)^*$

Tipo 3 $\alpha \in V$ $\beta \in \{ \epsilon \} \cup \Sigma \cup \{ \epsilon \}$
 $\uparrow \text{REX}$ $\uparrow (V \cup \Sigma)$
 $\uparrow \text{oppure } (\Sigma \cup V)$

LA MACCHINA DI TURING (Alan Turing, 1936):

→ Definizione di "COMPUTER" nel 1936: persona umana che effettua calcoli con carta e penna.

→ Turing era alla ricerca di un modo per formalizzare il concetto di "COMPUTAZIONE".

→ Domanda fondamentale: Tutti i problemi computazionali sono risolvibili?

→ Turing propose un dispositivo astratto ("macchina") che calcolasse come una persona umana.

Corrispondenze tra persona umana e macchina di Turing:

Persona umana

Macchina di Turing (TM)

Carta

Nastro con capacità infinita composto da celle. Ciascuna cella immagazzina un simbolo di un alfabeto.

Occhi

Testina che si muove lungo il nastro e legge/scrive un simbolo nella cella attualmente puntata dalla testina.

Mente

Unità di controllo con un numero finito di stati interni.

Regole per la computazione

Set di transizioni consentite: per ogni stato e simbolo sotto la testina, viene stabilito cosa la macchina deve fare.

Definizione formale della macchina di Turing:

È una settaglia $(Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$, dove:

Q = insieme finito degli stati interni

Σ = alfabeto degli input ;

$\sqcup \notin \Sigma$ "blank"

Γ = alfabeto del nastro ;

$\Gamma \supseteq \Sigma \cup \{\sqcup\}$ SERVE A INDICARE UNA CELLA VUOTA

$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

$(q, \sigma) \longrightarrow (q', \sigma', m)$

VERCHIO STATO \uparrow SIS'FOLLO LETTO DALLA TESTINA \uparrow NUOVO STATO \uparrow NUOVO SIMBOLLO \uparrow MOSSA DELLA TESTINA
 $(L = \text{LEFT}, R = \text{RIGHT})$

$q_0 \in Q$ = stato iniziale

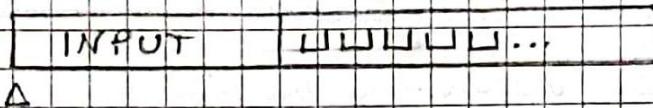
$q_A \in Q$ = stato di accettazione

$q_R \in Q$ = stato di rifiuto ; $q_R \neq q_A$

NB: La macchina di Turing accetta una stringa data in input non appena passa per q_A , a prescindere dal fatto che l'input sia stato consumato intera-

mente o meno. (un discorso analogo vale per lo stato q_r).

La macchina di Turing legge gli input all'interno del nastro. Perciò, la sua configurazione all'inizio di una computazione è la seguente:



→ LA MACCHINA POTREBBE ANCHE SOVRASCRIVERE L'INPUT

Esistono diverse versioni della macchina di Turing. Quella che vedremo noi è la VERSIONE CON NASTRO SEMI-INFINTO, in cui la TM non può muovere la testina a sinistra dell'input.

La CONFIGURAZIONE della TM indica il contenuto attuale del nastro, lo stato interno attuale della macchina e la posizione attuale della testina.

Una configurazione può essere quindi descritta con una stringa che rappresenta il contenuto del nastro (e che è necessariamente finita, dato che il numero di passi di una computazione deve essere finito) e uno stato interno nella posizione che precede la cella che si trova sotto la testina.

Al livello pratico, una configurazione C la vediamo scritta come:

$$C = [u \ q \ v] \quad u, v \in \Gamma^* \quad q \in Q$$

Dove il primo carattere di v indica il simbolo puntato dalla testina.

Definizione:

Una configurazione C_1 porta a una configurazione C_2 se la TM può andare legalmente da C_1 a C_2 con un singolo passo (avendo con una transizione consentita).

Esempio:

$$C_1 = [u a q_i b v] \quad u, v \in \Gamma^* \quad a, b \in \Gamma \quad q_i \in Q$$

$$\delta(q_i, b) = (q_{j,i}, c, \leftarrow) \quad \xrightarrow{\text{LEFT MOVE}} \quad C_2 = [u q_j a c v] \quad c \in \Gamma \\ q_j \in Q$$

Caso speciale 1:

$$C_1 = [q_i \mid b \vee] , \quad \delta(q_i, b) = (q_j, c, L) \xrightarrow{\text{LEFT MOVE
NON RIUSCITA}} C_2 = [q_j \mid c \vee]$$

Caso speciale 2:

$$C_1 = [\mu a q_i] \equiv [\mu a q_i \sqcup]$$

$$\delta(q_i, b) = (q_j, c, R) \xrightarrow{\text{RIGHT MOVE}} C_2 = [\mu a c q_j]$$

CONFIGURAZIONE INIZIALE SULL'INPUT w : $C_0 = [q_0 \mid w]$

CONFIGURAZIONE ACCETTANTE: include q_A

CONFIGURAZIONE RIFIUTANTE: include q_R

Configurazioni di arresto

Definizione:

$M(w)$ accetta ("La macchina M accetta sull'input w ") se esiste una sequenza di configurazioni C_0, C_1, \dots, C_k tali che:

- 1) $C_0 = [q_0 \mid w]$
- 2) C_i produce C_{i+1}
- 3) C_k è una configurazione accettante.

$M(w)$ rifiuta se esiste una sequenza di configurazioni C_0, C_1, \dots, C_k tali che:

- 1) $C_0 = [q_0 \mid w]$
- 2) C_i produce C_{i+1}
- 3) C_k è una configurazione rifiutante.

NB: C'è anche una terza possibilità, in cui la macchina non entra mai né nello stato di accettazione, né in quello di rifiuto.

Definizione:

Un linguaggio riconosciuto da una TM M è:

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

Definizione:

Un linguaggio L è "TURING-RECOGNIZABLE" o "RICOSSIVAMENTE EN_U

MERABILE" (r.e.) se esiste qualche TM M tale che $L(M) = L$.

Osservazione:

Se A è un r.e., $w \notin A$, non è necessariamente vero che $M(w)$ rifiuti! Come abbiamo già accennato, esiste una terza possibilità in cui la macchina entra in un ciclo senza fine (ENDLESS LOOP), senza passare mai né per q_A , né per q_R .

Definizione:

Un linguaggio L è "TURING-DECIDIBILE" o "DECIDIBILE" o "RICORSIVO" se esiste qualche TM M tale che $\forall w \in \Sigma^*$ $M(w)$ accetta se $w \in L$ e $M(w)$ rifiuta se $w \notin L$.

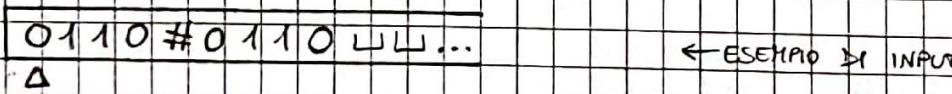
Lemma:

Se A è un linguaggio ricorsivo $\Rightarrow A$ è un r.e.

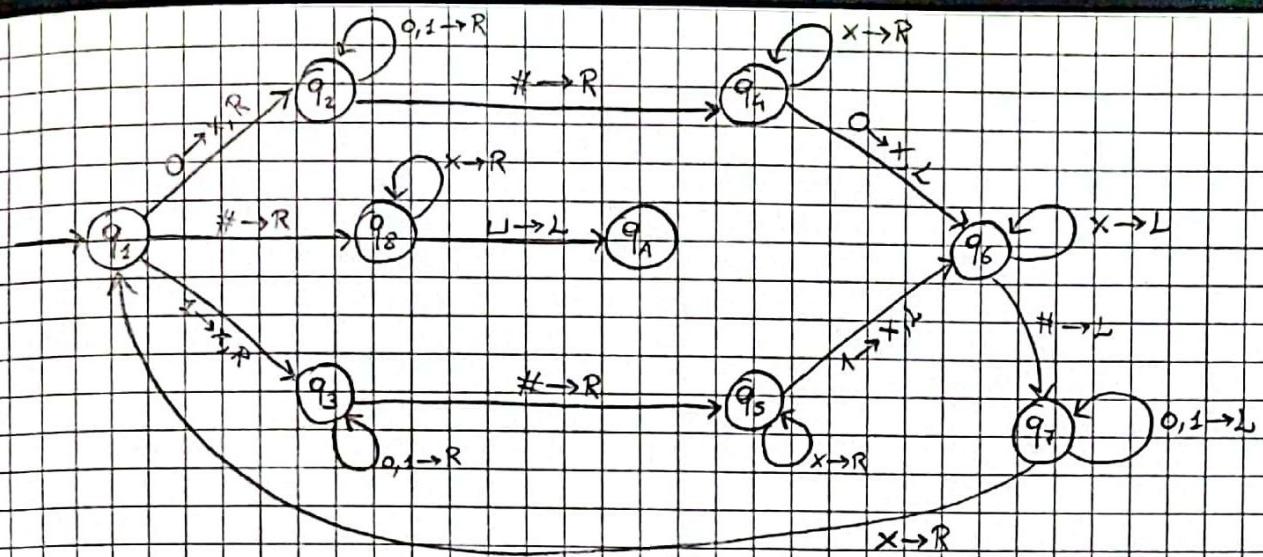
Esercizio:

Dimostrare che $B = \{w\#w \mid w \in \{0,1\}^*\}$ è decidibile.

L'idea è costruire una TM $M = (\mathcal{Q}, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$, $\Sigma = \{0, 1, \#\}$, $\Gamma = \Sigma \cup \{\sqcup, X\}$ che agisca nel seguente modo:



M deve leggere il primo simbolo dell'input, marcarlo con una X , muovere la testina verso destra fino ad arrivare al cancelletto e confrontare il primo simbolo dell'input col primo simbolo dopo il cancelletto, marcando con una X anche quest'ultimo. Se i due simboli sono uguali, M deve muovere la testina verso sinistra finché non trova una X a sinistra del cancelletto, per poi ripetere il procedimento a partire dal simbolo successivo.



→ Ogni freccia non rappresentata porta allo stato q_R .

Osservazioni:

- 1) In generale, una macchina di Turing ^{costruire} significa implementare un algoritmo che risolva un determinato problema.
- 2) Non è necessariamente vero che esiste un unico algoritmo per risolvere un problema o che esista un'unica rappresentazione di TM per schematizzare un particolare algoritmo. Per esempio, si avrebbe potuto risolvere l'esercizio precedente con una macchina di Turing con un totale di 9 stati (compresi q_A, q_R) e con $\Gamma = \Sigma \cup \{L\}$.

→ SIMULATORE DELLE MACCHINE DI TURING ONLINE:

<https://turingmachineimulator.com/>

21/04/2020

Esercizio 1:

Dimostrare che $C = \{0^m \mid m \geq 0\}$ è un linguaggio decidibile.

TM:

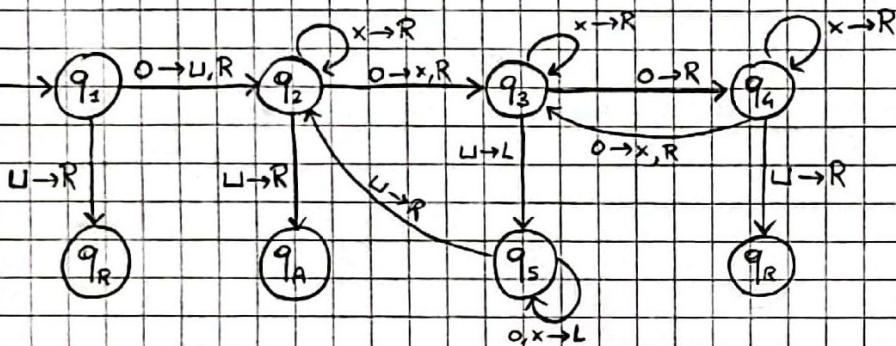
if there is no 0 \rightarrow reject ($\epsilon \notin C$)

while (1):

if there is just one 0 \rightarrow accept

if there is an odd number of 0 \rightarrow reject

cancel half of the 0



Esercizio 2:

Dimostrare che $A = \{a^i b^j c^k \mid i, j, k \geq 1 \wedge i \times j = k\}$ è un linguaggio decidibile.

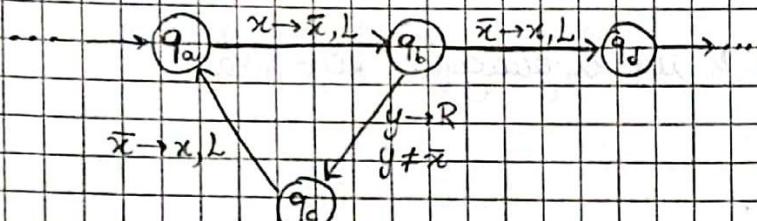
SEQUENZA DI PASSI DA EFFETTUARE:

1. Scansiona l'input da sinistra a destra e rifiuta se $w \notin L(a^+ b^+ c^+)$.
2. Riposiziona la testina sul simbolo più a sinistra dell'input.
3. Marca una a e scala a destra finché non trovi una b non marcata.
4. Spostati tra le b e le c, marcando alternativamente le b e le c, finché le b non sono finite.
5. Ripristina tutte le b marcate.
6. Ripeti dal punto 3. Se tutte le a sono state marcate, accetta se anche tutte le c sono state marcate.

Nella pagina seguente viene mostrato un gadget per riconoscere il simbolo più a sinistra dell'input (e quindi per poter applicare il punto 9).

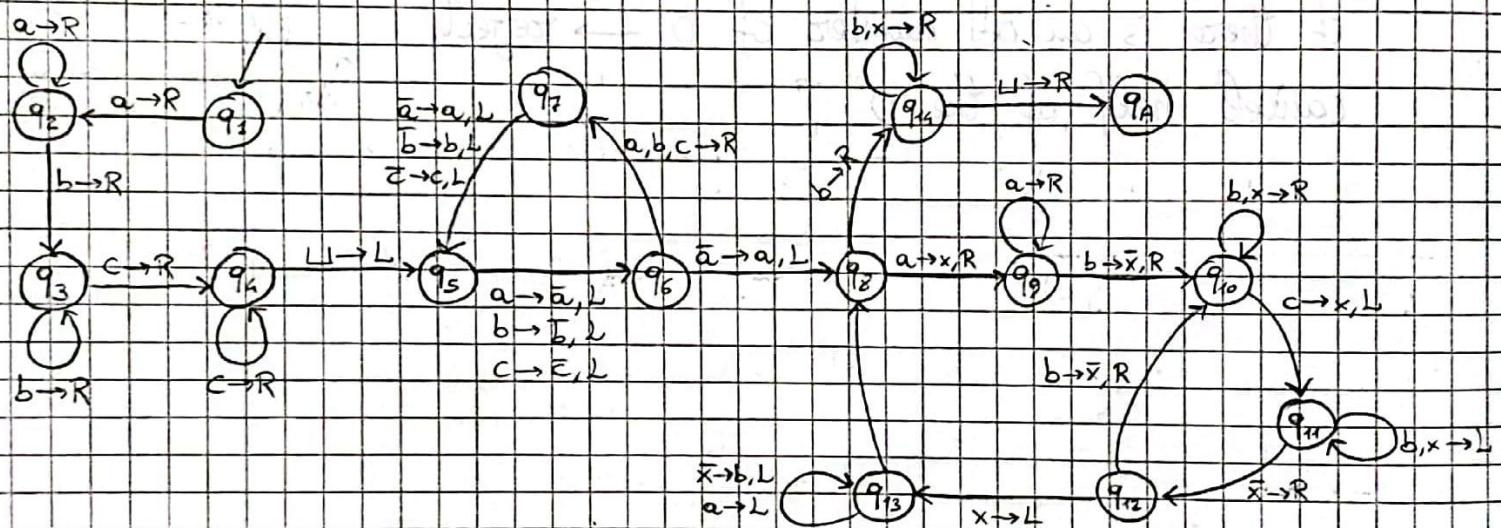
$x \in \Sigma, \bar{x} \notin \Sigma$

SE PER ESEMPIO PARTIAMO DALLA CONFIGURAZIONE $abq_a c$...



$abq_a c$	$\bar{a}q_b c$
$aq_b c$	$q_a abc$
$abq_c c$	$q_b \bar{a} bc$
$aq_b c$	$q_a \bar{a} bc$
$q_b \bar{a} c$	$q_a abc$

Ora siamo in grado di disegnare la TM che riconosce il linguaggio A.



Esercizio 3:

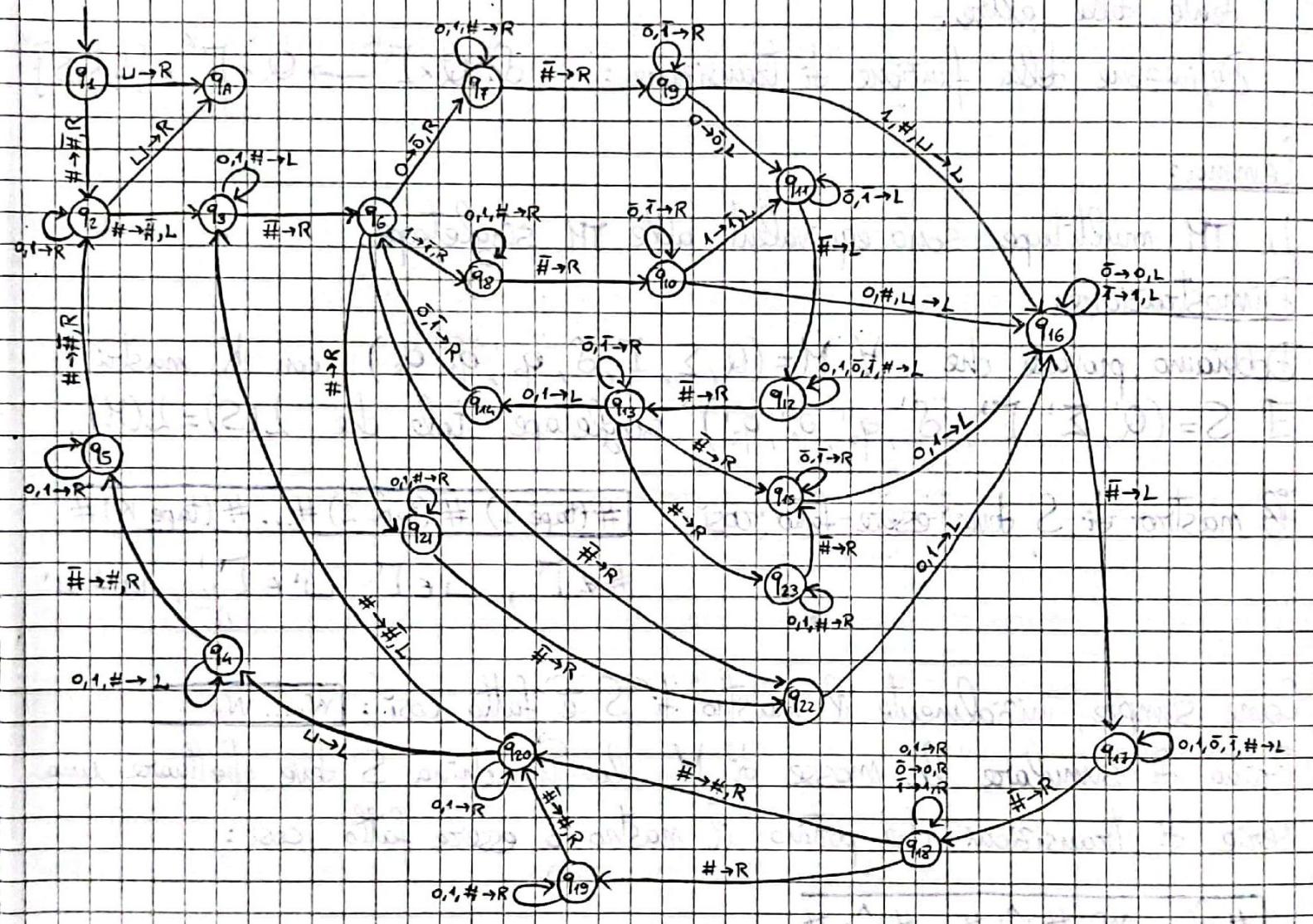
PROBLEMA DELLA DISTINZIONE DEGLI ELEMENTI: Dimostrare che il linguaggio $E = \{ \# x_1 \# x_2 \# \dots \# x_l \mid l \geq 0 \wedge x_i \in \{0, 1\}^* \wedge x_i \neq x_j \text{ se } i \neq j \}$ è decidibile.

SEQUENZA DI PASSI DA EFFETTUARE:

1. Se il simbolo più a sinistra è \sqcup , accetta ($E \in E$).
2. Se il simbolo più a sinistra dell'input non è $\#$, rifiuta.
3. Marca il $\#$ più a sinistra.
4. Scansiona l'input verso destra e marca il primo $\#$ che incontra.
 - 4.1. Se non trovi nessun $\#$, accetta.
5. Confronta le stringhe con il $\#$ marcato sulla sinistra, e rifiuta se sono uguali.
6. Rimuovi la marcatura dal $\#$ più a destra e marca il $\#$ immediatamente successivo sulla destra. Se non trovi nessun $\#$, rimuovi la mar-

catura dal $\bar{#}$ più a sinistra e marca il $\#$ immediatamente successivo sulla destra.

7. Ripeti dal punto 4.



24/10/2020

Varianti della macchina di Turing:

→ VARIANTE "LRS-move": la testina può non muoversi ("stop", S).

Transizione tipo: $\delta(q, a) = (q', b, S)$

Lemma:

Le TM LRS-move sono equivalenti alle TM LR-move.

Dimostrazione:

Una transizione di una TM LRS-move del tipo $\delta(q, a) = (q', b, S)$ può essere scritta come:

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

VE. PRIMA EFFETTUATA PRIMA LA MESSA A DESTRA E Poi QUELLA A SINI $\leftarrow \forall x \in \Gamma \quad \delta(q', x) = (q, x, L)$ ■

→ VARIANTE "multitape": la TM dispone di un numero di nastri fissato e maggiore di 1. L'input si trova sul primo nastro. Per ciascuna mossa, la TM può muovere la testina di ciascun nastro in modo indipendente dalle altre.

Definizione della funzione di transizione: $\delta: Q \times \Gamma^K \rightarrow Q \times \Gamma^K \times \{L, R, S\}$

Lemma:

Le TM multitape sono equivalenti alle TM singletape.

Dimostrazione:

Dobbiamo provare che $\forall M = (Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ con K nastri $\exists S = (Q', \Sigma', \Gamma', \delta', q'_0, q'_A, q'_R)$ singletape tale che $L(S) = L(M)$.

Il nastro di S deve essere fatto così: $\# (\text{tape 1}) \# (\text{tape 2}) \# \dots \# (\text{tape K}) \#$

$\# \in \Gamma$, $l \in \Gamma$, $l' \in \Gamma'$, $l \neq l'$

$\#$ è un simbolo che serve a indicare un nastro vuoto nella macchina M; l' è il vero blank di S

Come sempre, inizialmente il nastro di S è fatto così: $w_1 \dots w_n \leftarrow \text{INPUT STRING } (w)$

Prima di simulare le mosse di M, la macchina S deve effettuare una serie di transizioni che portino il nastro a essere fatto così:

$\# \hat{w}_1 \dots \hat{w}_n \# \hat{l} \# \dots \# \hat{l} \#$

↳ I simboli marcati col cappelletto (^) corrispondono a quelli puntati dalle testine di ciascun nastro di M $\Rightarrow \Gamma' \supseteq \Gamma \cup \{\hat{l}\}$

insieme dei simboli di M con cappelletto

Perciò, per ogni mossa di M, la macchina S deve effettuare tutta una serie di transizioni che aggiettino le varie zone del nastro. Per poter fare ciò, ha bisogno di stati supplementari.

In particolare: $|Q'| \geq |Q| \cdot |\Gamma^K|$

Inoltre, ogni volta che M scrive un simbolo nuovo su una testina, S deve fare spazio tra i due "#" della zona corrispondente, e cioè deve traslare di una posizione verso destra tutti i caratteri successivi con transizioni del tipo $\delta(q_x, x) = (q_x, y, R)$; $\delta(q_y, l') = (q_y, \#, L)$

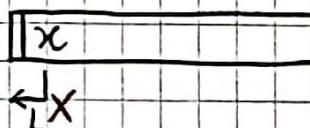
→ VARIANTE "doubly infinite singletape": la testina può andare liberamente anche a sinistra dell'input.

Lemma:

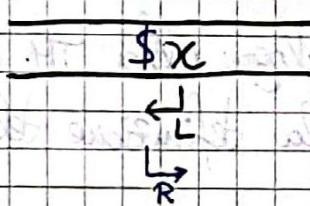
Le TM col nastro doppialemente infinito sono equivalenti alle TM col nastro semi-infinito.

Dimostrazione:

↔) La costruzione di una TM col nastro doppialemente infinito a partire da una TM col nastro semi-infinito è di nuovo banale ma stavolta con un dettaglio tecnico importante:



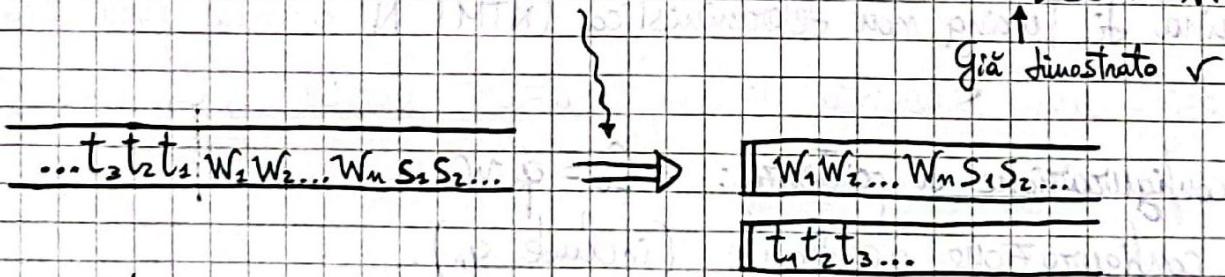
→ NEL NASTRO SEMI-INFINITO LO SPOSTAMENTO VERSO SINISTRA DELLA TESTINA A PARTIRE DAL PRIMO SIMBOLO NON AVVIENE.



→ LA TM COL NASTRO DOPPIAMENTE INFINTO, PER SIMULARE CORRETTEMENTE LA MACCHINA COL NASTRO SEMI-INFINTO, IN QUESTO CASO PARTICOLARE DEVE EFFETTUARE PRIMA UNO SPOSTAMENTO VERSO SINISTRA DELLA TESTINA E Poi UNO SPOSTAMENTO VERSO DESTRA. PER RICONOSCERE L'INIZIO DELLA STRINGA, LA TM CON NASTRO DOPPIAMENTE INFINTO DEVE INIZIALMENTE AGGIUNGERE UN CARATTERE SPECIALE SUBITO A SX.

⇒) Per dimostrare il viceversa, basta definire le seguenti equivalenze:

DOUBLY INFINITE SINGLETAPE \Rightarrow SEMIINFINITE 2-TAPE \Rightarrow SEMIINFINITE SINGLE TAPE



Accortezze da prendere:

→ Nel secondo nastro della TM 2-tape i simboli compaiono in ordine inverso rispetto al nastro della TM doubly infinite singletape; perciò, in questa zona, i movimenti delle testine delle due macchine hanno verso opposto.

→ La TM 2-tape, a differenza di quanto abbiamo già visto, stavolta lavora su un nastro per volta. In particolare, quando nell'altra macchina la testina va da w_1 a t_2 (o viceversa), la TM 2-tape deve cambiare

ra il mastro di riferimento su cui opera. Inoltre, per evitare equivoci, la TM=2-tape deve tenere traccia del mastro su cui leggere / scrivere correntemente, per esempio tramite l'aiuto dei simboli marcati col cappelletto (^).

→ VARIANTE "doubly infinite multtape": la TM dispone di più nastri doppia mente infiniti.

Lemma:

Le TM multtape con nastri doppiamente infiniti sono equivalenti alle TM singletape col mastro semi-infinito.

Macchina di Turing non deterministica:

È la variante più importante della macchina di Turing.

È una settaglia $(Q, \Sigma, \Gamma, \delta, q_0, q_A, q_R)$ del tutto analoga alla TM che conosciamo già, ma con una differenza sostanziale nella definizione della funzione di transizione:

$$\delta: Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{L, R, S\}}$$
$$(q, a) \longrightarrow \{(q_1, b_1, m_1), (q_2, b_2, m_2), \dots\}$$

Una macchina di Turing non deterministica (NTM) N accetta una stringa $w \in \Sigma^*$ se esiste una sequenza di configurazioni C_0, C_1, \dots, C_n tale che:

- 1) C_0 è la configurazione di partenza: $C_0 = q_0 w$
- 2) C_n è la configurazione accettante (include q_A).
- 3) $\forall k \geq 0 \exists$ transizione in δ che si applica a C_k e produce C_{k+1} .

Definizione:

Una NTM N RICONOSCE un linguaggio A $\iff L(N) = A$

Definizione:

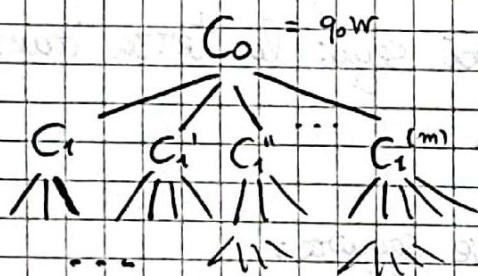
Una NTM N DECIDE un linguaggio A $\iff L(N) = A$ e ogni percorso di computazione di $N(w)$ termina in un numero finito di passi.

Teorema:

Ogni NTM N ha una macchina di Turing deterministica (DTM) D equivalente.

Idea della dimostrazione:

A causa del non-determinismo, Sappiamo che C_0 può produrre una serie di configurazioni derivate diverse:



→ LE CONFIGURAZIONI POSSONO ESSERE SCHEMATIZZATE CON UN ALBERO, IN CUI IL NODO RADICE CORRISPONDE ALLA CONFIGURAZIONE INIZIALE, MENTRE I NODI FOGLIA CORRISPONDONO ALLE CONFIGURAZIONI ACCETTANTI / RIFIUTANTI.

Per provare che, data una NTM N \exists DTM D tale che $L(D) = L(N)$, possiamo costruire una DTM D che effettui una visita dell'albero delle configurazioni della NTM N . Dal corso di Ingegneria degli Algoritmi Sappiamo che esistono due tipi di algoritmi di visita degli alberi:

- VISITA PER PROFONDITÀ
- VISITA PER AMPIEZZA

In particolare, D dovrà effettuare una visita per AMPIEZZA, poiché, in caso contrario, potrebbe correre il rischio di incappare in un percorso di computazione con un numero infinito di passi (che, cioè, non portano mai né allo stato di accettazione né allo stato di rifiuto), non potendo così passare a un percorso di computazione successivo che, magari, avrebbe portato ad accettare la stringa data in input.

Dimostrazione formale:

D ha tre nastri:

- 1)

W	
---	--
- 2)

...

- 3)

...

→ NASTRO CONTENENTE L'INPUT; È "read-only"

→ NASTRO DI LAVORO PER LA SIMULAZIONE DI N

→ NASTRO PER TENERE TRACCA DELLA VISITA IN AMPIEZZA DELL'ALBERO

Definiamo con b il numero massimo delle ramificazioni che hanno le configurazioni all'interno dell'albero (quindi il numero massimo di figli che ha ciascun nodi).

$$b \leq |\mathcal{Q}| \cdot |\mathcal{T}| \cdot 3 \xrightarrow{\text{IFR.2, S7}}$$

Definiamo quindi l'alfabeto per il terzo nastro di \mathcal{D} :

$$\mathcal{I}_b = \{1, 2, \dots, b\}$$

Per esempio, la stringa "231" significa:

"Dalla radice segui la seconda transizione; poi segui la terza transizione; infine segui la prima transizione".

Di seguito viene riportato l'algoritmo che \mathcal{D} deve seguire:

Sull'input w sul nastro 1 (i nastri 2, 3 sono inizialmente vuoti):

1. Copia l'input dal nastro 1 al nastro 2.
2. Usa il nastro 2 per simulare i passi descritti sul nastro 3.
 - 2.1. Se raggiungi q_A , accetta.
 - 2.2. Se raggiungi q_R , vai al punto 3.
 - 2.3. Se tutti i passi sul nastro 3 sono stati simulati, vai al punto 3.
3. Leggi la stringa sul nastro 3 e rimpicciolala con la successiva rispettando lo STRING ORDER in \mathcal{I}_b (ordine in cui viene fatta prima chiudere la lunghezza delle stringhe: $\epsilon < 1 < 2 < \dots < b < 11 < 12 < \dots$).
 - i. Torna al punto 1.

Corollario:

L è un r.e. $\Leftrightarrow \exists$ TM N t.c. N riconosce L ($L(N) = L$).

Teorema:

L è un linguaggio ricorsivo (decidibile) $\Leftrightarrow \exists$ TM N t.c. N decide L (ogni percorso di computazione di N termina in un numero finito di passi).

Dimostrazione:

\Rightarrow L ricorsivo $\xrightarrow{\text{PER DEFINIZIONE}}$ $\exists \text{ DTM } D \text{ t.c. } D \text{ decide } L$

D può essere considerato una NTM con un solo percorso di computazione ✓

\Leftarrow) Poiché N decide L , tutti i suoi percorsi di computazione terminano con un numero finito di passi. Perciò, nella costruzione della DTM D equivalente, basta solo aggiungere un controllo che stabilisca quali percorsi di computazione di N sono stati completati (e quindi hanno portato a una configurazione rifiutante). Ma, in ogni caso, è ovvio che la computazione di D debba terminare con un numero finito di passi, per cui si può concludere che L è un linguaggio ricorsivo. □

Definizione:

Un ENUMERATORE è una TM che ha un mastro speciale P e uno stato speciale q_p ; ogni volta che entra in q_p , "stampa" (genera) la stringa che si trova sul mastro P .

→ Se un enumeratore E non termina mai la sua computazione, allora potrebbe generare infinite stringhe.

Definizione:

L'insieme di stringhe generate da un enumeratore E che parte da un input vuoto è definito come "LINGUAGGIO ENUMERATO DA E ".

Teorema:

Un linguaggio A è r.e. \Leftrightarrow qualche enumeratore E enumera A .

Dimostrazione:

\Leftarrow) Proviamo che, se E enumera A , allora qualche TM M riconosce A . È sufficiente esibire l'algoritmo che deve eseguire la TM M che dobbiamo costruire (vedere pagina seguente).

Sull'input w :

1. Simula Σ . Ogni volta che Σ entra nello stato q_p , confronta w con la stringa sul nastro P .
2. Accetta non appena trovi una corrispondenza tra le due stringhe.

NB: Se $w \notin A \Rightarrow M$ non termina mai la sua computazione, perché rimarrà sempre lì a confrontare invano w con la stringa sul nastro P .

\Rightarrow Proviamo che, se una TM M riconosce A , allora esiste un enumeratore Σ per A .

È sufficiente esibire l'algoritmo che deve eseguire Σ .

Ignorando l'input:

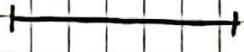
1. for $i=1$ to $+\infty$:

2. Simula M per i passi sulle prime i stringhe di $\Sigma^* = \{s_1, s_2, \dots\}$, dove Σ^* è un insieme ordinato (non impone il criterio di ordinamento).

3. Se qualche computazione accetta, allora copia la stringa corrispondente sul nastro P ed entra nello stato q_p .

NB: Se $w \in A \Rightarrow w \in \Sigma^* \Rightarrow \exists K : w = s_k$

Inoltre, $M(w)$ termina la sua computazione in un numero finito h di passi. Sia $l = \max(h, K)$. Allora, per $i \geq l$, certamente Σ produce la stringa w .



→ Nel 1933, Herbrand e Gödel proposero il modello delle FUNZIONI RI-CORSIVE GENERALI.

→ Nel 1936, Alano Church propose il modello del λ -CALCOLO.

→ Sempre nel 1936, come già sappiamo, Alan Turing propose il modello della MACCHINA DI TURING.

QUESTI FORMALISMI SONO EQUIVALENTI!

Tesi di Church-Turing:

Ogni algoritmo, comunque sia fatto, può essere implementato su una macchina di Turing o descritto con il λ -calcolo.

25/04/2020

Per poter effettuare una trattazione sugli algoritmi, introduciamo prima la nozione di PROBLEMA.

In realtà, già dall'inizio del corso abbiamo ricordato i problemi a LINGUAGGI. Facciamo un esempio:

PROBLEMA: il numero m è una potenza di 2?

↪ LINGUAGGIO: $A = \{0^{2^m} \mid m \geq 0\}$

→ Abbiamo già dimostrato che A è decidibile $\Rightarrow \exists$ TM che decide A

Per risolvere il problema sull'istanza m (come numero):

→ Costruisci la rappresentazione binaria di m : $\langle m \rangle_1 = 0^m = \overbrace{00\dots 0}^m$

→ Esegui $M(\langle m \rangle_1) = M(0^m)$ {ACCETTAZIONE $\Rightarrow m$ è una potenza di 2; YES-INSTANCE
RIFIUTO $\Rightarrow m$ non è una potenza di 2; NO-INSTANCE}

Si può definire quindi la seguente equivalenza:

LINGUAGGIO DECIDIBILE \iff PROBLEMA RISOLVIBILE

Problema:

Un certo polinomio univariato (a una sola variabile) ha una radice intera?

$D_1 = \{\langle p \rangle \mid p \text{ è un polinomio univariato con una radice intera}\}$

Lemma:

D_1 è r.c.e.

Dimostrazione:

Nella pagina seguente verrà riportato un algoritmo che deve seguire una macchina di Turing M_1 .

Sull'input $\langle p \rangle$ (p = polinomio univariato sulla variabile x):

1. for $x = 0, +1, -1, +2, -2, +3, -3 \dots$:
2. Valuta $p(x)$.
3. Se $p(x) = 0$, accetta.



N.B.: In tal modo, tutte le istanze NO (ovvero i polinomi che non hanno radici intere) portano M_1 a non concludere mai la sua computazione.

10° problema di Hilbert (1900):

"Definisca un processo per cui può essere determinato con un numero finito di operazioni se un polinomio multivariato (a più variabili) ha una radice intera".

$D_m = \{\langle p \rangle \mid p \text{ è un polinomio multivariato con una radice intera}\}$

Lemma:

D_m è r.e.

Dimostrazione:

La costruzione della TM che riconosce D_m è assolutamente analoga al la precedente, con l'unica differenza che bisogna iterare su tuple f_i interi anziché su semplici numeri.



Teorema:

Ogni radice intera r di un polinomio univariato p soddisfa la relazione $|r| \leq K \frac{C_{\max}}{C_1}$, dove K è il numero di termini di p , C_{\max} è il massimo valore assoluto dei coefficienti di p , e C_1 è il valore assoluto del coefficiente del termine di grado più alto.

Corollario:

$D_1 = \{\langle p \rangle \mid p \text{ è un polinomio univariato con una radice intera}\}$ è un linguaggio DECIDIBILE.

→ Nel 1970, Yury Matijasevič dimostrò che $D_m = \{\langle p \rangle \mid p \text{ è un}$

polinomio multivariato con una radice intera? NON È DECIDIBILE
⇒ il 10° problema di Hilbert NON ha un algoritmo che lo risolva.

Problema:

Un certo grafo non diretto G è连通の?

$$\bar{A} = \{ \langle G \rangle \mid G \text{ è un grafo non diretto connesso} \}$$

Teorema:

\bar{A} è decidibile.

Dimostrazione:

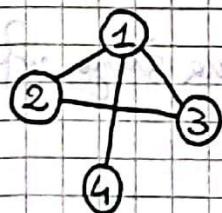
ALGORITMO PER LA MACCHINA M :

Sull'input $\langle G \rangle$, dove $\langle G \rangle$ è la codifica (rappresentazione) di un grafo:

1. Seleziona il primo nodo di G e marcalo.
2. Da ripetere finché non vengono più marcati altri nodi:
 - 3. Per ogni nodo di G , marcalo se è adiacente a un nodo già marcato.
 - 4. Visita tutti i nodi: se sono tutti marcati, accetta; altrimenti, rifiuta.

NB: La codifica di G è un dettaglio a cui bisogna fare attenzione. Possiamo per esempio rappresentare G con due liste, di cui una lista di nodi e una lista di archi.

Esempio:



$$\langle G \rangle = (1, 2, 3, 4) \quad ((1, 2), (1, 3), (2, 3), (1, 4))$$

→ Notiamo anche che, in ogni algoritmo, è specificato quale deve essere il tipo di input da dare alla macchina; qualunque input di forma diversa da quella che ci si aspetta viene a priori rifiutato dalla macchina perché non può far parte del linguaggio che essa riconosce.

→ Oltre al livello ALTO e al livello FORTALE, per descrivere un algoritmo si può utilizzare anche un livello PUNTUALE, che è intermedio tra

gli altri due e che consiste nello specificare con un linguaggio "naturale" tutte le operazioni punto per punto che la macchina deve effettuare.

Problema di accettazione per i DFA:

Un certo DFA accetta una data stringa?

$$A_{DFA} = \{ \langle B, w \rangle \mid B \text{ è un DFA che accetta una stringa } w \}$$

Teorema:

A_{DFA} è decidibile.

Dimostrazione:

ALGORITMO PER LA MACCHINA M:

Sull'input $\langle B, w \rangle$, dove B è un DFA e w è una stringa:

1. Simula B sull'input w.
2. Se la simulazione termina in uno stato di accettazione, accetta; altrimenti, rifiuta.

N.B.: w ha una lunghezza FINITA \Rightarrow M termina la computazione in un numero finito di passi \Rightarrow M DECIDE A_{DFA}

Inoltre, una possibile decodifica per B potrebbe essere:

LISTA PER Q, LISTA PER Σ , LISTA PER S, LISTA PER q_0 , LISTA PER F

Teorema:

Il linguaggio $A_{NFA} = \{ \langle B, w \rangle \mid B \text{ è un NFA che accetta una stringa } w \}$ è decidibile.

Dimostrazione:

ALGORITMO PER LA MACCHINA M:

Sull'input $\langle B, w \rangle$, dove B è un NFA e w è una stringa:

1. Converti B in un DFA C equivalente usando l'algoritmo mostrato nella dimostrazione dell'equivalenza tra DFA e NFA.
2. Simula la TM M (per il linguaggio A_{DFA}) sull'input $\langle C, w \rangle$.
3. Accetta o rifiuta in accordo a ciò che fa M.

Teorema:

Il linguaggio $A_{\text{REX}} = \{\langle R, w \rangle \mid R \text{ è una REX che genera la stringa } w\}$ è decidibile.

Dimostrazione:

ALGORITMO PER LA MACCHINA P:

Sull'input $\langle R, w \rangle$, dove R è una REX e w è una stringa:

1. Converti R in un NFA A equivalente usando l'algoritmo mostrato nel la dimostrazione dell'equivalenza tra REX e NFA.
2. Simula la TM N (per il linguaggio A_{NFA}) sull'input $\langle A, w \rangle$.
3. Accetta o rifiuta in accordo a ciò che fa N . ■

Emptiness Testing Problem per DFA:

Un certo DFA rifiuta qualunque stringa?

$$E_{\text{DFA}} = \{\langle A \rangle \mid A \text{ è un DFA tale che } L(A) = \emptyset\}$$

Teorema:

E_{DFA} è decidibile.

Dimostrazione:

La TM T che andiamo a definire ora sarà analoga a quella che riconosce il linguaggio $\bar{A} = \{\langle G \rangle \mid G \text{ è un grafo non vuoto connesso}\}$:

Sull'input $\langle A \rangle$, dove A è un DFA:

1. Marca lo stato q_0 (stato iniziale di A).
2. Da ripetere finché non vengono più marcati altri stati:
3. Marca ogni stato che ha una transizione entrante in esso e uscente dallo stato appena marcato.
4. Se nessuno stato di accettazione è stato marcato, accetta; altrimenti, rifiuta. ■

Equality Testing Problem per DFA:

Due DFA riconoscono lo stesso linguaggio?

$\text{EQ}_{\text{DFA}} = \{ \langle A, B \rangle \mid A, B \text{ sono DFA e } L(A) = L(B) \}$

Teorema:

EQ_{DFA} è decidibile.

Dimostrazione:

Consideriamo la DIFFERENZA SIMMETRICA tra i linguaggi generati da A, B :

$$L(C) := (L(A) \cap L(B)^c) \cup (L(B) \cap L(A)^c) =$$

$$= (L(A) \cup L(B)) \setminus (L(A) \cap L(B)) = L(A) \Delta L(B)$$

Possiamo dire che $L(A) = L(B) \iff L(A) \Delta L(B) = \emptyset$

Inoltre, poiché i linguaggi regolari sono chiusi rispetto alle operazioni di unione, intersezione e complemento, anche $L(C)$ deve essere un linguaggio regolare. Possiamo quindi sfruttare l'EMPTINESS TESTING PROBLEM sul DFA C .

ALGORITMO PER LA MACCHINA F :

Sull'input $\langle A, B \rangle$, dove A, B sono DFA:

1. Costruisce C , un DFA che riconosce il linguaggio $L(A) \Delta L(B)$.

2. Simula la TM T sull'input $\langle C \rangle$.

3. Accetta o rifiuta in accordo a ciò che fa T .

INFATTI: $T(\langle C \rangle)$ accetta $\iff L(C) = \emptyset \iff L(A) \Delta L(B) = \emptyset \iff L(A) = L(B)$

Teorema:

Il linguaggio $A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ è una CFG che genera } w \}$ è decidibile.

Dimostrazione:

Non va bene, senza ulteriori considerazioni, costruire una TM S che simuli tutti i possibili processi di derivazione di G , poiché alcuni di essi NON terminano con un numero finito di passi e ciò non consentirebbe di provare che A_{CFG} è decidibile. Perciò, è necessario introdurre qualche condizione che costringa la macchina a fermarsi prima e, quindi, a risolvere il problema in un numero finito di passi.

La Forma Normale di Chomsky (CNF) è di aiuto; Sappiamo già che le uniche regole consentite nella CNF sono del tipo:

$$A \rightarrow BC$$

$$A \rightarrow a$$

$$S \rightarrow E$$

dove $a \in \Sigma$, $A, B, C, S \in V$, S è la variabile iniziale

Si dimostra per induzione che, in una CFG G in Forma Normale di Chomsky, una derivazione per una stringa w , con $|w|=n > 0$, ha sempre lunghezza pari a $2n-1$.

Siamo ora in grado di definire un algoritmo che deve seguire la TM S che decide il linguaggio L_{CFG} :

Sull'input $\langle G, w \rangle$, dove G è una CFG e w è una stringa:

1. Converti G in una grammatica equivalente G' in Forma Normale di Chomsky.
2. Se $w = \epsilon$, accetta se la regola $S \rightarrow \epsilon$ è presente in G' ; se non è presente, rifiuta.
3. Qui $|w| > 0$: cerca tutte le derivazioni con $2 \cdot |w| - 1$ passi.
4. Se qualche derivazione genera w , accetta; altrimenti, rifiuta.

Teorema:

Il linguaggio $E_{CFG} = \{\langle G \rangle \mid G \text{ è una CFG tale che } L(G) = \emptyset\}$ è decidibile.

Dimostrazione:

È sufficiente costruire una TM R che, data una CFG G , verifichi se la variabile iniziale S può produrre una stringa di terminali o meno. Anche stavolta ripetiamo dall'algoritmo che abbiamo definito per i grafi connessi:

Sull'input $\langle G \rangle$, dove G è una CFG:

1. Marca tutti i simboli terminali di G .
2. Da ripetere finché non vengono più marcate altre variabili:
3. Marca ogni variabile A per cui \exists regola $A \rightarrow U_1 \dots U_k$ dove tutti gli U_i sono già marcati.
4. Se la variabile iniziale S è marcata, rifiuta; altrimenti, accetta.

28/04/2020

Teorema:

Ogni CFL è decibile.

Dimostrazione:

Se A è un CFL $\Rightarrow \exists$ CFG G tale che $L(G) = A$

Abbiamo già provato che \exists TM S che decide il linguaggio

$$A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ è una CFG e } w \in L(G) \}$$

Perciò è possibile costruire una TM M_A che decide il linguaggio A .

Eccone un algoritmo:

Sull'input w :

1. Anteponi la rappresentazione della grammatica G alla stringa w sul nastro.
2. Simula la TM S sull'input $\langle G, w \rangle$.
3. Accetta o rifiuta in accordo a ciò che fa S .



$$EQ_{\text{CFG}} = \{ \langle G_0, G_1 \rangle \mid G_0, G_1 \text{ sono CFG e } L(G_0) = L(G_1) \}$$

$$EQ_{\text{PDA}} = \{ \langle A, B \rangle \mid A, B \text{ sono PDA e } L(A) = L(B) \}$$

\hookrightarrow Sono linguaggi NON decibili (lo dimostreremo)

Macchina di Turing Universale (UTM):

È una macchina di Turing U che funziona nel seguente modo:

Sull'input $\langle M, w \rangle$, dove M è una TM e w è una stringa:

1. Simula M sull'input w .
2. Se $M(w)$ accetta, allora accetta; se $M(w)$ rifiuta, allora rifiuta.

L'UTM è un vero e proprio computer che lavora su un programma magazzinato in memoria insieme ai dati.

RAM (rappresentata col nastro)

È in pratica l'equivale
nte del nostro computer

- Nel 1936, Alan Turing introdusse per la prima volta l'UTM.
- Nel 1956, Claude Shannon, chiedendosi quale potesse essere l'UTM più piccola possibile, concluse che è possibile costruire UTM con 2 stati e n simboli, o anche UTM con m stati e 2 simboli.
- Nel 1962, Marvin Minsky provò che è possibile realizzare UTM con solo 7 stati e 4 simboli.
- Tra il 1996 e il 2002, Yuri Rogozhin provò che è possibile realizzare UTM con:
 - 15 stati e 2 simboli
 - 9 stati e 3 simboli
 - 6 stati e 4 simboli
 - 5 stati e 5 simboli
 - 4 stati e 6 simboli
 - 3 stati e 9 simboli
 - 2 stati e 18 simboli

→ HA SOLO 22 TRANSIZIONI !

Problema di accettazione per le TM:

Una certa TM accetta una data stringa?

$$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ è una TM che accetta in input una stringa } w\}$$

Lemma:

A_{TM} è un r.e.

Dimostrazione:

La TM che riconosce il linguaggio A_{TM} è proprio la macchina di Turing universale. Sia U un'UTM:

$$U(\langle M, w \rangle) \begin{cases} \text{accetta se } M(w) \text{ accetta} \\ \text{rifiuta se } M(w) \text{ rifiuta} \\ \text{entra in un ciclo senza fine se } M(w) \text{ non termina} \end{cases}$$

Infatti: $\langle M, w \rangle \in A_{\text{TM}} \Leftrightarrow M(w) \text{ accetta} \Leftrightarrow U(\langle M, w \rangle) \text{ accetta}$

NB: A_{TM} NON è decidibile.

Metodo di diagonalizzazione di Cantor (1891):

Due insiemi (finiti o infiniti) hanno la stessa dimensione se esiste una biiezione tra loro.

Cantor riuscì a provare che gli insiemi $\mathbb{N}, 2\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ hanno tutti una corrispondenza biunivoca, e li definì insiemi NUMERABILI.

Egli dimostrò anche che \mathbb{R} NON è un insieme numerabile, e lo fece tramite il suo METODO DI DIAGONALIZZAZIONE:

Supponiamo per assurdo che esiste una funzione biiettiva che lega gli insiemi $[0,1) \subset \mathbb{R}$ e \mathbb{N} . Perciò dovremmo essere in grado di enumerare (elencare) i numeri reali compresi tra 0 e 1:

$$\tau_1 = 0, \underline{0}13542\dots$$

$$\tau_2 = 0, 3\underline{5}4217\dots$$

$$\tau_3 = 0, 213\underline{4}57\dots$$

⋮

Costruiamo adesso un numero reale appartenente a $[0,1)$ la cui prima cifra dopo la virgola è diversa dalla prima cifra dopo la virgola di τ_1 , la seconda cifra dopo la virgola è diversa dalla seconda cifra dopo la virgola di τ_2 , ..., la i -esima cifra dopo la virgola è diversa dalla i -esima cifra dopo la virgola di τ_i , e così via all'infinito:

$$0, \underline{164}\dots$$

È chiaro che tale numero non appartiene alla lista $\{\tau_1, \tau_2, \dots, \tau_i, \dots\}$ ma è altrettanto ovvio che appartiene a $[0,1) \subset \mathbb{R} \Rightarrow$ ASSURDO $\Rightarrow \mathbb{R}$ NON è numerabile.

Teorema A:

L'insieme di tutte le TM è numerabile.

Teorema B:

L'insieme di tutti i linguaggi sopra un certo alfabeto Σ NON è numerabile.

Corollario:

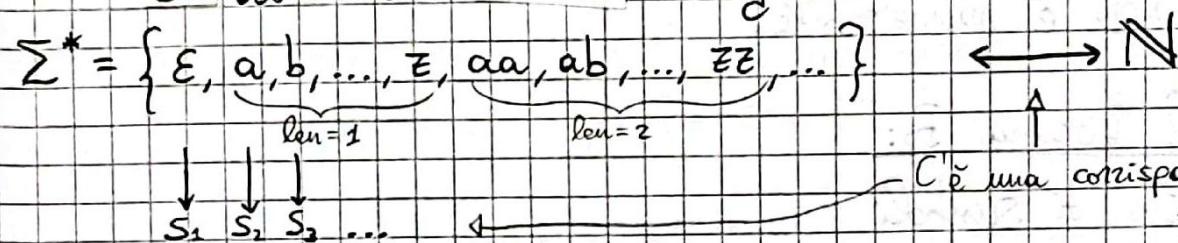
Esistono infiniti linguaggi sopra un certo alfabeto Σ che NON sono r.e.

Dimostrazione del teorema A:

Σ = insieme finito di simboli

$\Sigma^* = \{ \text{stringhe sopra } \Sigma \}$

↳ Possiamo attribuire un ordine a queste stringhe. Supponiamo che questo ordinamento sia lo string order:



C'è una corrispondenza biunivoca con \mathbb{N}
 $\Rightarrow \Sigma^*$ è numerabile

Possiamo decodificare le macchine di Turing con delle stringhe sopra un alfabeto come $\tilde{\Sigma} = \{0, \dots, 9, ;, L, R\}$. Facciamo un esempio:

$$Q = \{q_1, \dots, q_{10}\}$$

$$\Sigma = \{S_1, \dots, S_5\}$$

$$\Gamma = \Sigma \cup \{S_6, \dots, S_{20}\}$$

$$\delta = \{(q_i, S_j) \rightarrow (q_k, S_l, m) \dots\}$$

UNA STRINGA CORRISPONDENTE A QUESTA TM PUÒ ESSERE:

$$"10; 5; 20; 3; 5; 4; 7; L; 10; 13; 9; 11; R \dots L; 1; 10; 9"$$

$\underbrace{\qquad\qquad\qquad}_{\delta}$

$\begin{matrix} \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ 10 & 5 & 20 & 3 & 5 & 4 & 7 \end{matrix}$
 $\underbrace{(q_3, S_5) \rightarrow (q_4, S_7, L)}_{|\delta|=10}$
 $\underbrace{(q_6, S_{13}) \rightarrow (q_9, S_{11}, R)}_{|\Gamma|=20}$
 $\begin{matrix} \uparrow & \uparrow & \uparrow \\ q_1 & q_2 & q_3 \end{matrix}$
 $\begin{matrix} \uparrow & \uparrow & \uparrow \\ q_4 & q_5 & q_6 \end{matrix}$
 $\begin{matrix} \uparrow & \uparrow \\ q_7 & q_8 \end{matrix}$
 $q_9 = q_3$
 $q_{10} = q_{11}$
 $q_{12} = q_{13}$
 $q_{14} = q_{15}$
 $q_{16} = q_{17}$
 $q_{18} = q_{19}$
 $q_{20} = q_{21}$

Poiché le stringhe che decodificano le TM sono un sottoinsieme di $\tilde{\Sigma}^*$, e poiché abbiamo dimostrato che $\tilde{\Sigma}^*$ è numerabile, possiamo concludere che l'insieme di tutte le TM è numerabile. ■

Lemma: ← Serve per dimostrare il teorema 2

L'insieme $BIN = \{w_1 w_2 \dots \mid w_i \in \{0, 1\}\}$ NON è numerabile.

Dimostrazione:

Sfruttiamo il metodo di diagonalizzazione di Cantor.

Supponiamo per assurdo che BIN sia numerabile e stiliamo quindi una lista dei suoi elementi.

$$b_1 = 01100110\dots$$

$$b_2 = 10110011\dots$$

$$b_3 = 011010010\dots$$

:

Costruiamo una stringa infinita il cui i -esimo bit è diverso dall' i -esimo bit di $b_i \forall i \in \mathbb{N}$: $110\dots$

APPARTIENE A BIN MA NON ALLA NOSTRA LISTA!

?

ASSURDO: BIN non è numerabile.

Dimostrazione del teorema B:

Σ = insieme finito di simboli

$\Sigma^* = \{\text{stringhe sopra } \Sigma\}$

\hookrightarrow È un insieme numerabile $\Rightarrow \Sigma^* = \{\sigma_1, \sigma_2, \sigma_3, \dots\}$

$\text{Lang} = \{L \mid L \text{ è un linguaggio sopra } \Sigma\}$

Esiste una corrispondenza biunivoca tra Lang e BIN: consideriamo

$L \in \text{Lang}$; L può essere decodificato con una stringa infinita di bit

$w_1 w_2 w_3 \dots$ in cui $w_i = \begin{cases} 1 & \text{se } \sigma_i \in L \\ 0 & \text{se } \sigma_i \notin L \end{cases}$

$\Rightarrow L$ insieme Lang di tutti i linguaggi sopra Σ NON è numerabile.

Teorema:

Il linguaggio $A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ è una TM che accetta l'input } w \}$ NON è decidibile.

Dimostrazione:

Per assurdo, supponiamo che A_{TM} sia decidibile $\Rightarrow \exists \text{ TM } H$ tale che

$H(\langle M, w \rangle) = \begin{cases} \text{accetta} & \text{se } M(w) \text{ accetta} \\ \text{rifiuta} & \text{se } M(w) \text{ non accetta} \end{cases}$

Ma allora è possibile costruire un TM D' invertendo gli stati q_A, q_R in H:

$D'(\langle M, w \rangle) = \begin{cases} \text{accetta} & \text{se } M(w) \text{ non accetta} \\ \text{rifiuta} & \text{se } M(w) \text{ accetta} \end{cases}$

E quindi è possibile costruire una TM D che, sull'input $\langle M \rangle$, costruisce la stringa $\langle M, \langle M \rangle \rangle$ e simula D su $\langle M, \langle M \rangle \rangle$:

$$D(\langle M \rangle) = \begin{cases} \text{accetta se } M(\langle M \rangle) \text{ non accetta} \\ \text{rifiuta se } M(\langle M \rangle) \text{ accetta} \end{cases}$$

Ma questo significa che:

- Se $D(\langle D \rangle)$ accetta $\Rightarrow D(\langle D \rangle)$ non accetta
- Se $D(\langle D \rangle)$ rifiuta $\Rightarrow D(\langle D \rangle)$ accetta
- Se $D(\langle D \rangle)$ non termina $\Rightarrow D$, H sull'input $\langle D, \langle D \rangle \rangle$ non possono terminare

} TUTTE CONTRADDIZIONI

Il metodo di diagonalizzazione di Cantor c'entra con la dimostrazione appena fatta.

Poiché l'insieme delle TM è numerabile, è possibile costruire una tabella infinita le cui righe sono rappresentate dalle varie TM e le cui colonne sono rappresentate dalle codifiche di tali TM. Vediamo un esempio:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$...
M_1	A	NA	A	NA	
M_2	A	A	A	A	
M_3	NA	NA	NA	NA	
M_4	A	A	NA	NA	
:					

Se supponiamo che \exists TM H che decida tutti gli input $\langle M_i, \langle M_j \rangle \rangle$, allora possiamo costruire la seguente Tabella:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$...	$\langle D \rangle$
M_1	A	R	A	R		
M_2	A	A	A	A		
M_3	R	R	R	R		
M_4	A	A	R	R		
:						
	R	R	A	A		

Poiché anche D (l'opposto di H) è una TM, anch'essa tratta rientra nella lista

Qui si arriva a una contraddizione: dato che D è costruita in modo da essere diversa da tutte le TM della tabella, è impossibile decidere cosa

fare con l'input $\langle D \rangle$.

Quindi non esiste
be proprio esercizi nel
la tabella

ASSURDO

A_{TM} è un r.e. ma NON è decidibile $\Rightarrow A_{TM}^c$ NON è un r.e.

Teorema:

Un linguaggio A è decidibile \Leftrightarrow sia A sia A^c sono r.e.

Dimostrazione:

$\Rightarrow) A$ è decidibile $\Rightarrow A^c$ è decidibile

\Downarrow \Downarrow

A è un r.e. A^c è un r.e.

$\Leftarrow) \text{Se } A, A^c \text{ sono r.e. } \Rightarrow \exists \text{ TM } M_1, M_2 \text{ che riconoscono } A, A^c$

Allora possiamo costruire il seguente decisore N :

Sull'input w :

PER ESEMPIO UTILIZZANDO DUE NASTRI (UNO PER M_1 E UNO PER M_2) E SIMULANDO LE SINGOLE TRANSIZIONI DELLE 2 MACCHINE ALTERNATIVAMENTE

1. Simula $M_1(w)$ e $M_2(w)$ in parallelo.
2. Se $M_1(w)$ accetta, allora accetta; se $M_2(w)$ accetta, allora rifiuta.

N.B.: È sicuro che la computazione di N termina sempre: poiché M_1, M_2 riconoscono linguaggi opposti, data una qualunque stringa, essa verrà sempre accettata o da M_1 oppure da M_2 .

Teorema:

C è un r.e. $\Leftrightarrow \exists$ un linguaggio decidibile D tale che

$$C = \{x \mid \exists y \text{ tale che } \langle x, y \rangle \in D\}.$$

Problema della formata delle TM:

$Halt_{TM} = \{\langle M, w \rangle \mid M \text{ è una TM che termina la computazione sull'input } w\}$

Teorema:

$Halt_{TM}$ è un linguaggio non decidibile.

Dimostrazione:

Abbiamo già provato che A_{TM} non è decidibile.

Per assurdo, supponiamo che $Halt_{TM}$ sia decidibile $\Rightarrow \exists$ TM R tale che

$$R(\langle M, w \rangle) = \begin{cases} \text{accetta se } M(w) \text{ si forma} \\ \text{rifiuta se } M(w) \text{ non si forma} \end{cases}$$

Possiamo quindi costruire la seguente macchina di Turing N :

Sull'input $\langle M, w \rangle$, dove M è una TM e w è una stringa:

1. Simula R sull'input $\langle M, w \rangle$.
2. Se $R(\langle M, w \rangle)$ rifiuta, allora rifiuta. $\leftarrow M(w) \text{ non termina} \Rightarrow \langle M, w \rangle \notin A_M$
3. Se $R(\langle M, w \rangle)$ accetta, allora simula M sull'input w . $\leftarrow M(w) \text{ termina}$
4. Se $M(w)$ accetta, allora accetta; altrimenti, rifiuta.

Abbiamo ottenuto così che N decide il linguaggio $A_M \Rightarrow$ ASSURDO: $Halt_M$ non è decidibile. ■

NB: Questa dimostrazione ha semplificato il problema di accettazione come ~~pro~~ ~~una~~ più "semplice" del problema della fermata ($A_M \leq Halt_M$).

Questo non è altro che il processo di RIDUZIONE tra problemi.

→ Il fatto che $Halt_M$, A_M non sono linguaggi decidibili implica che non possiamo mettere in piedi un programma P che accetti in input un altro programma P' e decida sempre ~~se~~ \uparrow in un tempo finito se P' termina l'esecuzione o meno, oppure se accetta una determinata stringa o meno. In altre parole, i problemi di accettazione e di fermata non sono in generale risolvibili in un tempo finito.

09/05/2020

Teorema:

Il linguaggio $E_M = \{\langle M \rangle \mid M \text{ è una TM tale che } L(M) = \emptyset\}$ NON è decidibile.

Dimostrazione:

Sfruttiamo anche qui un processo di riduzione tra problemi ($A_M \leq E_M$).

Per assurdo, assumiamo che \exists TM R che decide il linguaggio E_M .

Consideriamo una qualunque codifica $\langle M, w \rangle$, e da essa costruiamo

una macchina M' che lavori nel seguente modo:

Sull'input z:

1. Se $z \neq w$, allora rifiuta.
2. Se $z = w$, allora esegui M sull'input w.
3. Accetta o rifiuta in accordo a ciò che fa M.

A questo punto per M' ci sono due possibilità:

$$\rightarrow L(M') = \{w\} \iff M(w) \text{ accetta} \iff R(\langle M' \rangle) \text{ rifiuta}$$
$$\rightarrow L(M') = \emptyset \iff M(w) \text{ non accetta} \iff R(\langle M' \rangle) \text{ accetta}$$

In tal modo, è possibile costruire una TM S con il seguente algoritmo:

Sull'input $\langle M, w \rangle$, dove M è una TM e w è una stringa:

1. Costruisci $\langle M' \rangle$ a partire da M, w.
2. Esegui R sull'input $\langle M' \rangle$.
3. Se $R(\langle M' \rangle)$ accetta, allora rifiuta. Se $R(\langle M' \rangle)$ rifiuta, allora accetta.

Abbiamo così ottenuto che S decide il linguaggio $A_M \Rightarrow$ ASSURDO

$\Rightarrow E_M$ NON è decidibile. ■

Teorema:

Il linguaggio $REGULAR_M = \{\langle M \rangle \mid M \text{ è una TM tale che } L(M) \text{ è regolare}\}$
NON è decidibile.

Dimostrazione:

Sfruttiamo di nuovo il fatto che $A_M \leq^* REGULAR_M$

Per assurdo, assumiamo che \exists TM R che decide $REGULAR_M$.

Consideriamo una qualunque codifica $\langle M, w \rangle$ e facciamo un lavoro del tutto analogo alla dimostrazione precedente. In particolare, costruiamo una TM S con il seguente algoritmo:

Sull'input $\langle M, w \rangle$, dove M è una TM e w è una stringa:

1. Costruisca la codifica della seguente TM M_2 :

"Sull'input x :

1. Se x ha la forma $0^m 1^n$, allora accetta.
 2. Esegui M sull'input w e accetta se $M(w)$ accetta".
2. Esegui R sull'input $\langle M_2 \rangle$.
3. Se $R(\langle M_2 \rangle)$ accetta, allora accetta. Altrimenti, rifiuta.

In particolare, per M_2 ci sono due possibilità:

$$\rightarrow L(M_2) = \{0^m 1^n \mid m, n \geq 0\} \iff M(w) \text{ non accetta} \iff R(\langle M_2 \rangle) \text{ rifiuta}$$

\hookrightarrow NON È REGOLARE $\hookrightarrow \langle M, w \rangle \notin A_T$

$$\rightarrow L(M_2) = \sum^* \iff M(w) \text{ accetta} \iff R(\langle M_2 \rangle) \text{ accetta}$$

\hookrightarrow È REGOLARE $\hookrightarrow \langle M, w \rangle \in A_T$

In tal modo, abbiamo che $S(\langle M, w \rangle)$ accetta se $R(\langle M_2 \rangle)$ accetta se $\langle M, w \rangle \in A_T$; $S(\langle M, w \rangle)$ rifiuta in caso contrario.

In altre parole, abbiamo ottenuto che S decide il linguaggio A_T
 \Rightarrow ASSURDO \Rightarrow $REGULAR_{TM}$ NON è decidibile. ■

Teorema:

Il linguaggio $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ sono TM e } L(M_1) = L(M_2)\}$
NON è decidibile.

Dimostrazione:

Stavolta sfruttiamo il fatto che $E_{TM} \leq^* EQ_{TM}$.

Per assecco, assumiamo che $\exists_{TM} R$ che decide EQ_{TM} .

Costruiamo una TM S che lavori nel seguente modo:

Sull'input $\langle M \rangle$, dove M è una TM:

1. Costruisce una codifica $\langle M_3 \rangle$ su una TM M_3 tale che $L(M_3) = \emptyset$.
2. Esegui R sull'input $\langle M, M_3 \rangle$.
3. Se $R(\langle M, M_3 \rangle)$ accetta, allora accetta. Altrimenti rifiuta.

Poiché R accetta $\Leftrightarrow L(M) = L(M_3) \Leftrightarrow L(M) = \emptyset$, e rifiuta negli altri casi, e poiché S accetta o rifiuta in questo accordo a ciò che fa R , abbiamo ottenuto che S decide il linguaggio $E_M \Rightarrow$ ASSURDO $\Rightarrow EQ_M$ NON è decidibile.

Teorema di Rice:

Sia P una qualunque proprietà non banale di un linguaggio riconosciuto da una TM. Allora il problema di determinare se il linguaggio di una data TM ha la proprietà P è NON decidibile.

NB: P è una proprietà non banale vuol dire che:

- $\rightarrow P \neq \emptyset$ (\exists almeno una TM T tale che $L(T)$ gode della proprietà P)
- $\rightarrow P^c \neq \emptyset$ (\exists almeno una TM T' tale che $L(T')$ non gode della proprietà P)

Dimostrazione:

Sfruttiamo una volta per tutte la riduzione $A_M \leq P$.

Per assurdo, assumiamo che la proprietà P sia decidibile $\Rightarrow \exists$ TM R che decide P .

Sia T_0 una TM tale che $L(T_0) = \emptyset$. Senza perdita di generalità, possiamo supporre che $T_0 \notin P$.

Poiché P non è banale, \exists TM T_1 tale che $T_1 \in P$.

Costruiamo una TM S che lavori nel seguente modo:

Sull'input $\langle M, w \rangle$, dove M è una TM e w è una stringa:

1. A partire da $\langle M, w \rangle$, costruisci una catifera della seguente TM M_w : "Sull'input x :

1. Simula M su w . Se $M(w)$ non accetta, allora rifiuta.

2. Simula T_1 su x . Se $T_1(x)$ accetta, allora accetta."

2. Esegui R sull'input $\langle M_w \rangle$.

3. Se $R(\langle M_w \rangle)$ accetta, allora accetta. Altrimenti, rifiuta.

A questo punto ci sono due possibilità:

$$\rightarrow w \in L(M) \Rightarrow M(w) \text{ accetta} \Rightarrow L(M_w) = L(T_1)$$

$$T_1 \in P \Rightarrow M_w \in P \Rightarrow R(\langle M_w \rangle) \text{ accetta} \Rightarrow S(\langle M, w \rangle) \text{ accetta}$$

$$\rightarrow w \notin L(M) \Rightarrow L(M_w) = \emptyset = L(T_0)$$

$$T_0 \notin P \Rightarrow M_w \notin P \Rightarrow R(\langle M_w \rangle) \text{ rifiuta} \Rightarrow S(\langle M, w \rangle) \text{ rifiuta}$$

Abbiamo così ottenuto che S decide il linguaggio $A_M \Rightarrow$ ASSURDO

\Rightarrow La proprietà P è non decidibile. ■

Storia della computazione:

Dati una TM M e un input w , la STORIA DELLA COMPUTAZIONE è una sequenza di configurazioni C_1, C_2, \dots, C_k tale che:

$\rightarrow C_i$ produce C_{i+1} con una transizione legale.

$$\rightarrow C_1 = q_0 w$$

$\rightarrow C_k =$ configurazione accettante (e in tal caso si parlerebbe di STORIA DELLA COMPUTAZIONE ACCETTANTE) oppure

$C_k =$ configurazione rifiutante (e in tal caso si parlerebbe di STORIA DELLA COMPUTAZIONE RIFIUTANTE). =

Se una computazione non termina mai, allora non esiste una storia della computazione.

In particolare:

\rightarrow Le DTM hanno al più una storia della computazione.

\rightarrow Le NTM potrebbero avere più storie della computazione diverse.

Automa linearmente limitato (LBA):

Un LBA è una macchina di Turing con un singolo mastro e una testina a cui non è consentito muoversi al di fuori della porzione del mastro che inizialmente conteneva l'input.

Nel caso particolare in cui l'input è la stringa vuota, il mastro è costituito

to da un'unica cella contenente il carattere \sqcup blank (\sqcup).

Gli LBA hanno una potenza espressiva minore delle TM; ciò è testimoniato dal seguente teorema:

Teorema:

Il linguaggio $A_{LBA} = \{ \langle M, w \rangle \mid M \text{ è un LBA che accetta la stringa } w \}$ è decidibile.

Dimostrazione:

Costruiamo una UTM L che simuli un LBA M sull'input w . Ci sono tre possibilità:

- $M(w)$ accetta $\Rightarrow L(\langle M, w \rangle)$ può tranquillamente accettare.
- $M(w)$ rifiuta $\Rightarrow L(\langle M, w \rangle)$ può tranquillamente rifiutare.
- $M(w)$ non termina mai $\Rightarrow L$, per terminare, avrebbe bisogno di ~~trovare~~ un criterio per stabilire che $M(w)$ mai si ferma e, quindi, non accetta.

In quest'ultimo caso, la sequenza di configurazioni di M è infinita.

Ma, poiché M , in quanto LBA, ha a disposizione una porzione limitata di nastro, può in realtà assumere un numero finito di configurazioni diverse.

In particolare, se $q = |Q|$, $g = |\Gamma|$, $n = |w| = \text{numero di celle disponibili}$, il numero di configurazioni possibili è $q \cdot n \cdot g^n$.

Se $M(w)$ non termina e la sua sequenza di configurazioni è infinita (e, quindi, ha lunghezza maggiore di $q \cdot n \cdot g^n$), per il principio della piccionaia una configurazione C_i si presenta più volte; ciò implica che M è entrato in un loop e C_i (come le altre configurazioni) ~~si~~ ^è seguito viene ripercorsa all'infinito, in un ciclo senza fine. Questa è una situazione che può essere accertata dalla UTM L che, a quel punto, può rifiutare l'input $\langle M, w \rangle$ senza pericolo di errori.

Nella pagina seguente viene riportato l'algoritmo che deve seguire L .

Sull'input $\langle M, w \rangle$, dove M è un LBA e w è una stringa:

1. Simula M su w per $q \cdot n \cdot g^n$ passi oppure finché M non termina.
2. Se $M(w)$ accetta, allora accetta. In qualunque altro caso, rifiuta.

In definitiva, L decide il linguaggio A_{LBA} . □

Teorema:

Il linguaggio $E_{\text{LBA}} = \{\langle M \rangle \mid M \text{ è un LBA tale che } L(M) = \emptyset\}$ NON è decidibile.

Idea della dimostrazione:

Il linguaggio A_{TM} "si riduce via storia della computazione" a E_{LBA} .

Consideriamo il linguaggio $L(B) = \{x \mid x \text{ è una storia della computazione accettante per } M(w)\}$, dove w è un qualunque input dell'LBA M }, e consideriamo una macchina B tale che $B(x)$ accetta $\iff x \in L(B)$.
 x , essendo una computazione della storia accettante, può essere rappresentato sul nastro così: $C_0 \# C_1 \# C_2 \# \dots \# C_k$

In particolare B , poiché non ha bisogno di uscire con la testina al di fuori dell'input, può essere senza problemi un LBA.

Vediamo l'algoritmo che deve seguire B :

Sull'input x :

1. Controlla se x è una serie di stringhe separate da '#'.
2. Controlla che C_0 sia la configurazione di partenza di $M(w)$.
3. Controlla che C_k includa q_A .
4. $\forall i \mid 0 \leq i \leq k-1$ controlla che $C_i \rightarrow C_{i+1}$.
5. Se tutti i controlli sono stati superati, accetta. Altrimenti, rifiuta.

A questo punto bisognerebbe ridurre A_{TM} a E_{LBA} "via storia della computazione".

Dimostrazione formale:

Per assurdo, assumiamo che il linguaggio E_{LBA} sia decidibile da una TM R .

Costruiamo una TM S che lavori nel seguente modo:

Sull'input $\langle M, w \rangle$, dove M è una TM e w è una stringa:

1. A partire da $\langle M, w \rangle$, costruisci una codifica di un LBA B il quale può verificare se una storia della configurazione accettante è valida.
2. Esegui R sull'input $\langle B \rangle$.
3. Se $R(\langle B \rangle)$ accetta, allora rifiuta. Altrimenti, accetta.

Abbiamo così ottenuto che:

$$\langle M, w \rangle \in A_{\text{TM}} \iff M(w) \text{ accetta} \iff \exists \text{ storia della configurazione accettante per } M(w) \iff L(B) \neq \emptyset \iff R(\langle B \rangle) \text{ rifiuta} \iff S(\langle M, w \rangle) \text{ accetta}$$

D'altra parte, $S(\langle M, w \rangle)$ rifiuta negli altri casi (ovvero se $\langle M, w \rangle \notin A_{\text{TM}}$)
Di conseguenza, S decide il linguaggio $A_{\text{TM}} \Rightarrow \text{ASSURDO}$
 $\Rightarrow E_{\text{LBA}}$ non è decidibile. ■

Teorema:

Il linguaggio $All_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ è una CFG con un alfabeto di terminali } \Sigma \text{ tale che } L(G) = \Sigma^* \}$ è non decidibile.

RICHIAMO: E_{CFG} è decidibile.

Idea della dimostrazione:

Il linguaggio A_{TM} "si riduce tramite storia della computazione" a All_{CFG} .

L'idea è, a partire da una codifica $\langle M, w \rangle$, derivare una CFG G che generi tutte le stringhe che non codificano storie della computazione accettanti per $M(w)$, per poi sfruttare G per provare che A_{TM} "si riduce tramite storia della computazione" a All_{CFG} .

Dimostrazione formale:

Per assurdo, assumiamo che All_{CFG} sia decidibile $\Rightarrow \exists \text{ TM } R$ che decide il linguaggio All_{CFG} .

Consideriamo una codifica $\langle M, w \rangle$, dove M è una TM e w è una stringa, e da essa costruiamo una CFG G che genera:

- 1) Qualunque stringa che non sia composta da configurazioni separate da '#'.
- 2) Qualunque stringa che non cominci con la configurazione di partenza per $M(w)$.
- 3) Qualunque stringa che non termini con una configurazione accettante per $M(w)$.
- 4) Qualunque stringa che contenga due configurazioni consecutive illegali per M .

Tuttavia, è particolarmente difficile costruire una grammatica fatta in questo modo. Perciò, è conveniente sfruttare l'equivalenza tra CFG e PDA e, a partire dalla codifica $\langle M, w \rangle$, costruire un PDA P che riconosca:

- 1) Qualunque stringa che non sia composta da configurazioni separate da '#'.
2) Qualunque stringa che non cominci con la configurazione di partenza per $M(w)$.
3) Qualunque stringa che non termini con una configurazione accettante per $M(w)$.
4) Qualunque stringa che contenga due configurazioni consecutive illegali per M .

È tutto sommato facile fare in modo che P accetti le stringhe di tipo (1), (2), (3). È invece più delicato trattare con un PDA le stringhe di tipo (4).

In particolare, si presentano due problemi:

A) P deve confrontare due configurazioni consecutive e, poiché quando legge i simboli dell'input li consuma una volta per tutte senza possibilità di tornare indietro, ha necessità di memorizzare nello stack la prima delle due configurazioni. Ma lo stack è un oggetto LIFO, quindi i caratteri salvati in esso possono essere letti solo in ordine contrario, il che renderebbe farraginoso il loro confronto con i caratteri della seconda configurazione.

SOLUZIONE: scrivere la stringa che codifichi un'eventuale storia della computazione nel seguente modo:

$$\# C_1 \# C_2^R \# C_3 \# C_n^R \# \dots$$

B) Una qualunque configurazione intermedia C_i deve essere confrontata sia con C_{i-1} , sia con C_{i+1} (nel caso in cui $C_{i-1} \rightarrow C_i$ sia una transizione legale). Tuttavia, non è possibile salvare sullo stack i caratteri di C_i con i caratteri di C_{i-1} , i quali già si trovano sullo stack. D'altra parte, nel momento in cui il confronto fra C_{i-1} e C_i è terminato,

i simboli che codificano C_i sono stati già consumati all'interno dell'input, il che renderebbe impossibile il confronto tra C_i e C_{i+1} .

SOLUZIONE: sfruttare il non determinismo, facendo simulare in parallelo a P tutti i confronti tra la configurazione C_i e la configurazione C_{i+1} $\forall i \mid 0 \leq i \leq K-1$. Ogni percorso di computazione, ad esempio, potrà effettuare un unico confronto tra due configurazioni consecutive, partendo quindi con uno stack che conteggi tanti '#' quante sono le configurazioni da saltare all'interno dell'input prima di arrivare a quelle che si vogliono esaminare.

Adesso siamo finalmente in grado di costruire una TM S che abbia il seguente algoritmo:

Sull'input $\langle M, w \rangle$, dove M è una TM e w è una stringa:

1. A partire da $\langle M, w \rangle$, costruisci \exists una descrizione $\langle P \rangle$ di un PDA P che riconosca tutte le stringhe che NON codifichino storie della computazione accettante per $M(w)$.
2. A partire da $\langle P \rangle$, costruisci una codifica $\langle G \rangle$ di una CFG G equivalente a P .
3. Esegui R sull'input $\langle G \rangle$.
4. Se $R(\langle G \rangle)$ accetta, allora rifiuta; altrimenti, accetta.

Abbiamo così ottenuto che:

$$\begin{aligned} \langle M, w \rangle \in \text{Acc}_M &\iff M(w) \text{ accetta} \iff \exists \text{ storia della computazione accettante } x \text{ per } M(w) \\ &\iff P(x) \text{ rifiuta} \iff x \notin L(G) \iff L(G) \neq \Sigma^* \iff \end{aligned}$$
$$\iff \langle G \rangle \notin \text{All}_{\text{CFG}} \iff R(\langle G \rangle) \text{ rifiuta} \iff S(\langle M, w \rangle) \text{ accetta}$$

D'altra parte, $S(\langle M, w \rangle)$ rifiuta negli altri casi (ovvero se $\langle M, w \rangle \notin \text{Acc}_M$).

Di conseguenza, S decide il linguaggio $\text{Acc}_M \implies \text{ASSURDO}$

$\implies \text{All}_{\text{CFG}}$ NON è decidibile.

Teorema:

Il linguaggio $EQ_{CFG} = \{ \langle G, H \rangle \mid G, H \text{ sono CFG tali che } L(G) = L(H) \}$ è NON decidibile.

Dimostrazione:

Sfruttiamo il fatto che $ALL_{CFG} \leq EQ_{CFG}$.

Per assurdo, assumiamo che EQ_{CFG} sia decidibile da una TM R .

Costruiamo una TM S che lavori nel seguente modo:

• Sull'input $\langle G \rangle$, dove G è una CFG:

1. Costruisce una codifica $\langle H \rangle$, dove H è una CFG tale che $L(H) = \Sigma^*$.
2. Esegui R sull'input $\langle G, H \rangle$.
3. Se $R(\langle G, H \rangle)$ accetta, allora accetta; altrimenti, rifiuta.

Abbiamo così ottenuto che S decide il linguaggio $ALL_{CFG} \Rightarrow$ ASSURDO

$\Rightarrow EQ_{CFG}$ NON è decidibile. ■

Teorema:

EQ_{CFG}^c è un TC.e.

TM che accetta se l'input non contiene due grammatiche; altrimenti le due grammatiche vengono convertite in CNF e, alle prime CFG vengono fatte generare stringhe di lunghezza in via via crescente (con 2n+1 passi) e, in parallelo, alle seconda CFG vengono fatte generare tutte le stringhe possibili con 2n+1 passi; se NON c'è corrispondenza, la TM accetta, altrimenti no.

Corollario:

EQ_{CFG} NON è un TC.e.

10/05/2020

Il problema di corrispondenza di Post (PCP):

È un esempio di problema indecidibile non collegato alla teoria degli automi.

ISTANZA: Un insieme di tessere del domino:

STRINGA SUPERIORE
STRINGA INFERIORE

DOMANDA: Esiste una sequenza di tessere del domino (con possibili ripetizioni) tale che la concatenazione delle stringhe superiori coincide con la concatenazione delle stringhe inferiori?

Vediamo un paio di esempi nella pagina seguente.

Esempio 1:

$$\left\{ \begin{array}{|c|c|c|c|} \hline b & a & ca & abc \\ \hline ca & ab & a & c \\ \hline \end{array} \right\} \in \text{PCP}$$

Infatti, una possibile sequenza è:

a	b	ca	a	abc
ab	ca	a	ab	c

Esempio 2:

$$\left\{ \begin{array}{|c|c|c|} \hline abc & ca & abb \\ \hline ab & a & ca \\ \hline \end{array} \right\} \notin \text{PCP}$$

È facile convincersi che questa, invece, è un'istanza NO poiché, per ogni tessera del domino, la stringa superiore ha lunghezza strettamente maggiore della stringa inferiore.

NB: Le istanze SÌ e le istanze NO particolarmente evidenti come quella dell'esempio 2 sono facilmente risolvibili. Ma, in generale, per le istanze NO non si riesce a trovare una dimostrazione formale che testimoni che esse non appartengono a PCP. Questo è il motivo informale per cui PCP è un problema indecidibile.

Per arrivare alla dimostrazione formale dell'indecidibilità di PCP, intraprenderemo il problema PCP modificato (MPCP):

$\text{MPCP} = \{ \langle P, J \rangle \mid P \text{ è un'istanza di PCP con cui è possibile creare una corrispondenza tra stringhe superiori e stringhe inferiori con una concatenazione di tessere che inizia dalla tessera speciale } J \}$

Dobbiamo provare che MPCP " \leq " PCP, ovvero che da un'istanza $\langle P, J \rangle$ di MPCP sia sempre possibile derivare un'istanza $\langle P' \rangle$ di PCP equivalente. In altre parole:

$$\langle P, J \rangle \in \text{MPCP} \iff \langle P' \rangle \in \text{PCP}$$

Per farlo, introduciamo delle definizioni:

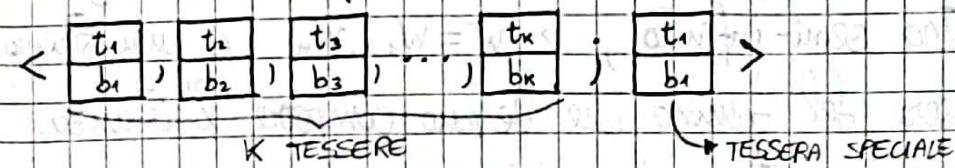
Sia Σ^* l'insieme di stringhe sopra un certo alfabeto Σ e sia $\bullet \notin \Sigma$. Data una stringa SE Σ^* (per esempio $s = \text{"hello"}$):

$$s = \bullet \cdot h \cdot e \cdot l \cdot l \cdot o$$

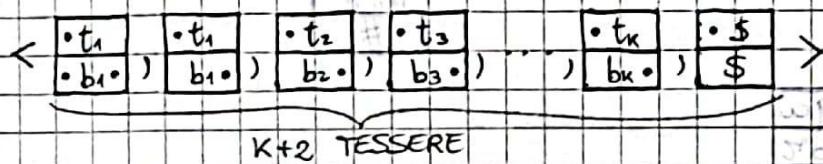
$s = "h \cdot e \cdot l \cdot l \cdot o \cdot"$

$\cdot s = "\cdot h \cdot e \cdot l \cdot l \cdot o \cdot"$

Ora, data una qualunque istanza $\langle P, J \rangle$:



possiamo ricavare un'istanza $\langle P' \rangle$ con due tessere in più:



Siamo ora in grado di dimostrare che $\langle P, J \rangle \in MPCP \iff \langle P' \rangle \in PCP$:

$\Rightarrow \langle P, J \rangle \in MPCP$ vuol dire che, a partire dall'istanza $\langle P, J \rangle$, è possibile costruire una sequenza di tessere in cui la concatenazione delle stringhe superiori è uguale alla concatenazione delle stringhe inferiori, a partire dalla tessera speciale:

$t_1 \dots t_j \dots t_m$
$b_1 \dots b_j \dots b_m$

Allora sarà sicuramente possibile costruire una sequenza con la stessa caratteristica nell'istanza $\langle P' \rangle$:

$\cdot t_1 \dots t_j \dots t_m \cdot \$$
$\cdot b_1 \dots b_j \dots b_m \cdot \$$

\Leftarrow Nell'istanza $\langle P' \rangle$, per avere una corrispondenza tra stringhe superiori e stringhe inferiori, la sequenza deve necessariamente iniziare con la tessera

$\cdot t_1$
$\cdot b_1 \cdot$

 (che è l'unica con parte superiore e parte inferiore che iniziano con lo stesso carattere) e deve necessariamente concludersi con la tessera

$\cdot \$$
$\cdot \$$

 (che è l'unica con parte superiore e parte inferiore che terminano con lo stesso carattere). Per il resto, è immediato trovare le tessere

corrispondenti nell'istanza $\langle P, J \rangle$ per creare la sequenza equivalente di tessere che abbia la parte superiore e la parte inferiore che matchino.

Teorema:

MPCP è non decidibile.

Dimostrazione:

Sfruttiamo il fatto che $\text{A}_{\text{TM}} \leq \text{MPCP}$.

Consideriamo una codifica $\langle M, w \rangle$, dove M è una macchina di Turing LR-

-move con un singolo nastro semi-infinito, e $w = w_1..w_n$ è una stringa.

Vediamo quali sono le tessere del dominio che devono comporre l'istanza $\langle P, J \rangle$ che si può ottenere a partire da $\langle M, w \rangle$:

$\begin{array}{|c|} \hline \# \\ \hline \# q_0 w_1 .. w_n \# \\ \hline \end{array}$

→ tessera speciale
di partenza (d)

$\forall a, b \in \Gamma \quad \forall q, r \in Q \quad q \neq q_r$
 $\delta(q, a) \rightarrow (r, b, R)$

$\begin{array}{|c|} \hline q_A \\ \hline \# \\ \hline \end{array}$

→ tessera
finale

$\forall a, b, c \in \Gamma \quad \forall q, r \in Q \quad q \neq q_r$
 $\delta(q, a) \rightarrow (r, b, L)$

$\begin{array}{|c|} \hline q_A \\ \hline b \\ \hline r \\ \hline \end{array}$

mosse a
destra

$\begin{array}{|c|} \hline c \\ \hline q_A \\ \hline r \\ \hline c \\ \hline b \\ \hline \end{array}$

mosse a
sinistra

$\begin{array}{|c|} \hline \# \\ \hline q_A \\ \hline \# \\ \hline r \\ \hline b \\ \hline \end{array}$

mosse a sinistra
a partire dalla prima cella del nastro

$\forall a \in \Gamma$

$\begin{array}{|c|} \hline a \\ \hline a \\ \hline \end{array}$

serve a riportare i simboli che non sono
coinvolti in un passo della computazione

$\begin{array}{|c|} \hline a \\ \hline q_A \\ \hline \end{array}$

serve per poter arrivare a un match finale
delle stringhe nel momento in cui si entra nello
stato q_A

$\begin{array}{|c|} \hline q_A \\ \hline a \\ \hline q_A \\ \hline \end{array}$

ALL YOU CAN
EAT

$\begin{array}{|c|} \hline \# \\ \hline \# \\ \hline \end{array}$

serve a passare sopra al simbolo '*' che
separa una configurazione dall'altra

$\begin{array}{|c|} \hline \# \\ \hline \square \\ \hline \# \\ \hline \end{array}$

espansione del nastro
verso destra

Per assurdo, supponiamo che MPCP sia decidibile $\Rightarrow \exists \text{ TM } R$ che decide MPCP.
Allora possiamo costruire una TM S con il seguente algoritmo:

Sull'input $\langle M, w \rangle$, dove M è una TM e w è una stringa:
1. Costruisci l'istanza $\langle P, J \rangle$ di MPCP corrispondente a $\langle M, w \rangle$.

2. Esegui R sull'input $\langle P, J \rangle$.

3. Se $R(\langle P, J \rangle)$ accetta, allora accetta; altrimenti, rifiuta.

In questo modo, S decide il linguaggio $\text{A}_{\text{TM}} \Rightarrow \text{ASSURDO}$
→ MPCP NON è decidibile.

Corollario:

PCP è NON decidibile.

Dimostrazione:

Sfruttiamo il fatto che MPCP " \leq " PCP.

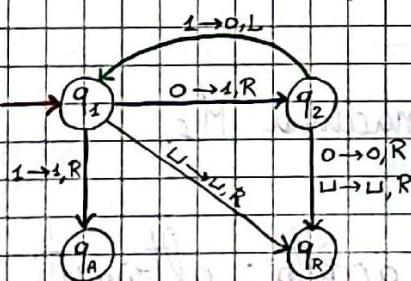
Per assurdo, supponiamo che PCP sia decidibile $\Rightarrow \exists \text{ TM } R$ che decide PCP
Allora possiamo costruire una TM T con il seguente algoritmo:

Sull'input $\langle P, d \rangle$, dove P è un'istanza di PCP e d è una tessera:

1. Costruisce l'istanza $\langle P' \rangle$ di PCP corrispondente a $\langle P, d \rangle$.
2. Esegui R sull'input $\langle P' \rangle$.
3. Se $R(\langle P' \rangle)$ accetta, allora accetta; altrimenti, rifiuta.

In questo modo, T decide MPCP \Rightarrow ASSURDO \Rightarrow PCP NON è decidibile.

Facciamo un esempio di costruzione di un MPCP a partire da una TM M :



SUPPOSIANO CHE L'INPUT SIA
 $w=01$

Le tessere del domino sono:

#	$q_1 0$	$0 q_2 1$	$1 q_2 1$	$1 q_2 1$	$# q_2 1$	$# q_2 1$
$\# q_1 0 1 \#$	$1 q_2$	$0 q_2 0$	$q_1 1 0$	$q_1 1 0$	$q_1 1 0$	$\# q_1 0$

$q_1 1$	0	1	L	#	#
$1 q_A$	0	1	L	#	#

$q_A 0$	$q_A 1$	$q_A L$	$0 q_A$	$1 q_A$	$L q_A$	$q_A \# \#$
q_A	q_A	q_A	q_A	q_A	q_A	#

La sequenza delle tessere che permette di matchare la stringa superiore con quella inferiore è:

$\# q_1 0 1 \# 1 q_2 1 \# q_1 1 0 \# 1 q_1 0 \# q_1 0 \# q_1 \# \#$

$\# q_1 0 1 \# 1 q_2 1 \# q_1 1 0 \# 1 q_1 0 \# q_1 0 \# q_1 \# \#$

PARTE CHE SERVE PER FAR MATCHARE LA STRINGA SUPERIORE CON LA STRINGA INFERIORE

STORIA DELLA COMBINAZIONE $W = 1(01)$

Definizione:

Una funzione $f: \Sigma^* \rightarrow \Gamma^*$ è una FUNZIONE CALCOLABILE se qualche TM M , su ogni input $w \in \Sigma^*$, termina con l'output $f(w) \in \Gamma^*$ sul nastro.

Definizione:

Un linguaggio $A \subseteq \Sigma^*$ si dice "MAPPING REDUCIBLE" o "MANY-ONE REDUCIBLE" a un linguaggio $B \subseteq \Gamma^*$ ($A \leq_m B$) se esiste una funzione calcolabile $f: \Sigma^* \rightarrow \Gamma^*$ tale che $\forall w \in \Sigma^* \quad w \in A \Leftrightarrow f(w) \in B$.

Teorema:

Se $A \leq_m B$ e B è decidibile \Rightarrow anche A è decidibile.

Dimostrazione:

Sia M_B un decisore per il linguaggio B , e sia M_f una TM che testimonia il fatto che A è riducibile a B tramite una funzione $f: \Sigma^* \rightarrow \Gamma^*$.

Allora possiamo costruire un'altra TM M_A con il seguente algoritmo:

Sull'input w :

1. Calcola $f(w)$ eseguendo la macchina M_f .
2. Esegui M_B sull'input $f(w)$.
3. Se $M_B(f(w))$ accetta, allora accetta; altrimenti, rifiuta.

M_A decide il linguaggio $A \Rightarrow A$ è decidibile. ■

Corollario:

Se $A \leq_m B$ e A è NON decidibile \Rightarrow anche B è NON decidibile.



A questo punto, possiamo dimostrare ad esempio che $\text{A}_{\text{TM}} \leq_m \text{Halt}_{\text{TM}}$ esistendo una TM F che computi una riduzione tra i due linguaggi. L'algoritmo di F è il seguente:

Sull'input $\langle M, w \rangle$:

1. Se $\langle M, w \rangle$ non è una codifica di una TM e di una stringa, allora p...

mi sul nastro di output la codifica $\langle E, w \rangle$, dove E è una TM che non termina mai; dopodiché fermati.

→ È UN PASSO CHE POTREMO SOTTINTENDERE

2. Costruisce la codifica $\langle M', w \rangle$, dove M' è una TM col seguente algoritmo:

"Sull' input w :

1. Esegui M sull' input w .

2. Se $M(w)$ accetta, allora fermati.

3. Se $M(w)$ rifiuta, allora entra in un ciclo senza fine".

3. Poni sul nastro di output la codifica $\langle M', w \rangle$; dopodiché fermati.

Teorema:

Se $A \leq_m B$ e B è un r.e. \Rightarrow anche A è un r.e.

La dimostrazione di questo teorema è identica a quella sulla decibilità di A mostrata nella pagina precedente.

Corollario:

Se $A \leq_m B$ e A non è un r.e. \Rightarrow anche B non è un r.e.

Lemma:

$$A \leq_m B \iff A^c \leq_m B^c$$

Sfruttiamo questo lemma per provare che i linguaggi EQ_{TM} , EQ_{TM}^c non sono r.e. Per farlo, dimostriamo che:

$$\bullet A_{TM} \leq_m EQ_{TM}^c$$

$$\bullet A_{TM} \leq_m EQ_{TM}$$

In particolare, dobbiamo costruire due TM molto simili (differiscono solo in un punto) che facciano il seguente lavoro:

Sull' input $\langle M, w \rangle$, dove M è una TM e w è una stringa:

1. Costruisce M_1 col seguente algoritmo:

"Su qualunque input: [rifiuta | accetta]".

2. Costruisce M_2 col seguente algoritmo:

"Su qualunque input: esegui $M(w)$ e accetta se $M(w)$ accetta".

3. Poni sul nastro di output la codifica $\langle M_1, M_2 \rangle$ e fermati.

Sappiamo che A_{TM} è un r.e. ma è NON decidibile \Rightarrow

$\Rightarrow A_{TM}^c$ NON è un r.e. Per il Lemma:

- $A_{TM} \leq_m^c EQ_{TM}^c \Rightarrow A_{TM}^c \leq_m^c EQ_{TM}^c \Rightarrow EQ_{TM}^c$ NON è un r.e.
- $A_{TM} \leq_m^c EQ_{TM} \Rightarrow A_{TM}^c \leq_m^c EQ_{TM}^c \Rightarrow EQ_{TM}^c$ NON è un r.e.

12/05/2020

La macchina di Turing "SELF":

SELF è una TM che ignora l'input, scrive sul nastro la sua stessa descrizione $\langle \text{SELF} \rangle$ e poi si ferma.

Il suo funzionamento è il seguente:

Questa è B

STAMPA DUE COPIE DI CIÒ CHE SEGUE, LA SECONDA TRA VIRGOLETTE

"STAMPA DUE COPIE DI CIÒ CHE SEGUE, LA SECONDA TRA VIRGOLETTE"

Possiamo dire che $\text{SELF} = A \cdot B$

Questa è prototipo da A

↑
CONCATENAZIONE DI DUE TM: PRIMA ESEGUI A, Poi ESEGUI B

Lemma:

Esiste una funzione calcolabile $q: \Sigma^* \rightarrow \Sigma^*$ tale che, se $w \in \Sigma^*$, allora $q(w) = \langle P_w \rangle$, dove P_w è una TM col seguente algoritmo:
Su qualunque input:

1. Cancella il nastro (= ignora l'input).

2. Scrivi w sul nastro e fermati.

Corollario:

Esiste una TM Q col seguente algoritmo:

Sull'input w :

1. Sul nastro riempiezza w con $\langle P_w \rangle$

2. Fermati.

$\text{SELF} = A \cdot B$, dove:

$$A = P_{\langle B \rangle}$$

$\leftarrow A$ è la TM che stampa la codifica di B sul nastro

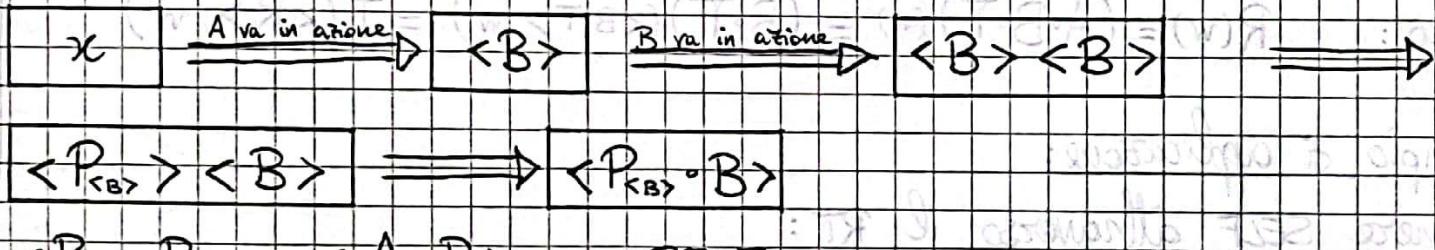
B = "Sull'input $\langle M \rangle$, dove M è una porzione di TM:

1. Fai una copia di $\langle M \rangle$.
2. Esegui Q sulla copia più a sinistra di $\langle M \rangle$ per rimpiazzarla con $\langle P_{\langle M \rangle} \rangle$.
3. Collega $\langle P_{\langle M \rangle} \rangle$ e $\langle M \rangle$ per ottenere $\langle P_{\langle M \rangle} \circ \langle M \rangle \rangle$.
4. Fermati".

Vediamo la TM SELF all'opera:

$$\text{SELF}(x) = A \cdot B(x) = P_{\langle B \rangle} \circ B(x)$$

COSA C'È SCRITTO SUL NASTRO PASSO DOPO PASSO:



Teorema della ricorsione (RT):

Sia T una TM che calcola una funzione $t: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. Allora esiste una TM R che calcola una funzione $r: \Sigma^* \rightarrow \Sigma^*$ tale che, $\forall w \in \Sigma^*, r(w) = t(\langle R \rangle, w)$.

Dimostrazione:

Intruduciamo due macchine di Turing:

P'_x = "Sull'input w :

$$Q'(x) = \langle P'_x \rangle$$

1. Sposta a sinistra di w .

2. Stampa x a sinistra di w ".

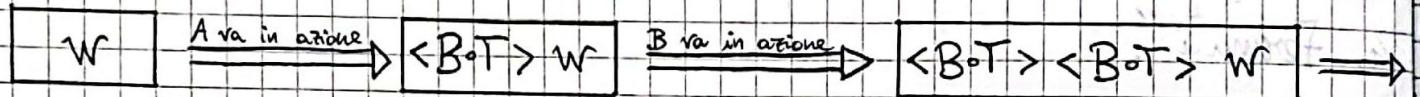
Costruiamo una TM $R = A \cdot B \cdot T$, dove:

$$A = P'_{\langle B \circ T \rangle}$$

$B =$ "Sull'input $\langle M, w \rangle$, dove M è un frammento di TM e w è una stringa:

1. Fai una copia di $\langle M \rangle$ a sinistra dell'input.
2. Esegui Q' sulla copia più a sinistra di $\langle M \rangle$ per rimpiazzarla con $\langle P'_{\langle M \rangle} \rangle$.
3. Collega $\langle P'_{\langle M \rangle} \rangle$ e $\langle M \rangle$ per ottenere $\langle P'_{\langle M \rangle} \circ M \rangle$.
4. Dai in output $\langle P'_{\langle M \rangle} \circ M, w \rangle$ ".

COSA C'È SCRITTO SUL NASTRO PASSO DOPO PASSO:



$$\langle P'_{\langle B \cdot T \rangle} \rangle \langle B \cdot T \rangle w \Rightarrow \langle P'_{\langle B \cdot T \rangle} \circ B \cdot T \rangle w$$

$$\langle P'_{\langle B \cdot T \rangle} \circ B \cdot T \rangle = \langle A \cdot B \cdot T \rangle = \langle R \rangle$$

$$\text{Perciò: } R(w) = (A \cdot B \cdot T)(w) = (B \cdot T)(\langle B \cdot T \rangle, w) = T(\langle R \rangle, w)$$

Esempio di applicazione:

Ottenerà SELF attraverso il RT:

$T =$ "Sull'input $\langle M, w \rangle$:

1. Cancella w dal nastro.
2. Termina con l'output $\langle M \rangle$ ".

Utilizzando T , possiamo costruire la macchina R tale che:

$$R(w) = T(\langle R \rangle, w) = \langle R \rangle \implies R \in \text{SELF}$$

Di conseguenza, in modo più semplice, è possibile scrivere l'algoritmo di SELF nel seguente modo:

Su qualunque input:

1. Ottieni, attraverso il RT, la tua stessa descrizione $\langle \text{SELF} \rangle$ sul nastro.
2. Fermati.



Dimostriamo adesso che A_{TM} è non decidibile mediante il RT.

Per assurdo, assumiamo che esiste una TM H che decide A_{TM} .

Allora possiamo costruire una TM B con il seguente algoritmo:

Sull'input w:

1. Ottieni, attraverso il RT, la tua stessa descrizione $\langle B \rangle$.
2. Esegui H sull'input $\langle B, w \rangle$.
3. Se $H(\langle B, w \rangle)$ accetta, allora rifiuta; altrimenti, accetta.

In questo modo abbiamo ottenuto che:

- $B(w)$ accetta $\Rightarrow H(\langle B, w \rangle)$ rifiuta $\Rightarrow \langle B, w \rangle \notin A_{\text{TM}} \Rightarrow$
 $\Rightarrow B(w)$ NON accetta
- $B(w)$ rifiuta $\Rightarrow H(\langle B, w \rangle)$ accetta $\Rightarrow \langle B, w \rangle \in A_{\text{TM}} \Rightarrow$
 $\Rightarrow B(w)$ accetta
- \Rightarrow ASSURDO $\Rightarrow A_{\text{TM}}$ non è decidibile.

Definizione:

Se M è una TM, la lunghezza di $\langle M \rangle$ ($| \langle M \rangle |$) è il numero di simboli in $\langle M \rangle$.

Definizione:

Una TM M si dice MINIMALE se non c'è nessuna TM equivalente a M che abbia una descrizione di lunghezza minore.

Teorema:

Il linguaggio $\text{Min}_{\text{TM}} = \{ \langle M \rangle \mid M \text{ è una TM minimale} \}$ NON è r.c.e.

Dimostrazione:

Per assurdo, supponiamo che un enumeratore E enumerate Min_{TM} .

Allora possiamo costruire una TM C con il seguente algoritmo:

Sull'input w:

1. Ottieni, attraverso il RT, la tua stessa descrizione $\langle C \rangle$.
2. Esegui E finché non viene stampata una macchina D con $| \langle D \rangle | > | \langle C \rangle |$
3. Simula D sull'input w.

Perciò, $C(w) = D(w) \Rightarrow C$ è equivalente a D .

$|<D>| > |<C>| \Rightarrow D$ non è minimale $\Rightarrow D \notin \text{Min}_m$

Ma D viene stampata da E che enumera il linguaggio Min_m
 \Rightarrow ASSURDO $\Rightarrow \text{Min}_m$ NON è un R.E.

Teorema del punto fisso per le funzioni calcolabili:

Sia $t: \Sigma^* \rightarrow \Sigma^*$ una funzione calcolabile. Allora esiste una TM F per cui $t(<F>) = G$ descrive una TM equivalente a F .

NB: Se una stringa non è una codifica corretta di una TM, possiamo assumere che la stringa descriva una TM che rifiuta sempre senza essere eseguita.

Dimostrazione:

Basta mostrare l'algoritmo che deve seguire la TM F :

Sull'input w :

1. Ottieni, tramite il RT, la tua stessa descrizione $<F>$.
2. Calcola $t(<F>) = <G>$.
3. Esegui G sull'input w .

È evidente che G, F sono equivalenti.

Definizione:

Un ORACOLO per un linguaggio B è un dispositivo esterno in grado di determinare se una qualunque stringa w fa parte o meno di B (anche nel caso in cui B non sia un R.E.).

Definizione:

Una TM CON ORACOLO è una TM che ha la capacità aggiuntiva di interrogare un ~~real~~ oracolo.

$M^B :=$ TM M con un oracolo per il linguaggio B .

→ Avere un oracolo per un determinato problema può rendere "decidibili"

anche altri problemi.

Esempio:

T^{dec} può decidere il linguaggio E_M (che sappiamo non essere decidibile). Vediamo come:

$T^{\text{dec}} =$ "Sull'input $\langle M \rangle$, dove M è una TM:

1. Costruisci la seguente TM N :

$N =$ "Su qualunque input:

1. Esegui M in parallelo su tutte le stringhe in Σ^* (effettuando più ma un passo ~~per~~ su tutte le stringhe di lunghezza pari a 1^{zero} , poi due passi su tutte le stringhe di lunghezza minore o uguale a 2, e così via, come in una visita in ampiezza di un albero).

2. Se M accetta una qualunque di queste stringhe, allora accetta".

2. Chiama l'oracolo per determinare se $\langle N, \text{input qualsiasi} \rangle \in A_{\text{TM}}$.

3. Se l'oracolo risponde di NO, allora accetta. Se invece l'oracolo risponde di SÌ, allora rifiuta".

Possiamo quindi dire che E_M È DECIDIBILE RELATIVAMENTE AD A_{TM} .

Definizione:

Un linguaggio A si dice "TURING-REDUCIBLE" a un linguaggio B ($A \leq_T B$) se A è decidibile relativamente a B .

Teorema:

Se $A \leq_T B$ e B è decidibile \Rightarrow anche A è decidibile.

Dimostrazione:

B è decidibile \Rightarrow possiamo prendere come oracolo per B una normalissima TM $\Rightarrow A$ è decidibile relativamente a una TM $\Rightarrow A$ è decidibile



L'ingegneria informatica si basa sullo studio di come un'INFORMAZIONE sia:

• IMMAGGGINATA

• TRASMESSA

• ANALIZZATA

• TRASFORMATA

Tuttavia, è complicato fare una vera e propria definizione di INFORMAZIONE. Possiamo solo dire che il concetto di informatione è correlato a quello di DATO (valore numerico che quantifica certi processi) e a quello di CONOSCENZA.

Quello che ci chiediamo è: è possibile fare una misura QUANTITATIVA di un' informazione?

Esempio:

Quale delle seguenti stringhe porta più informazione?

S_1 : 01

S_1 : $(01)^m \rightarrow$ QUANTITÀ BASSA DI INFORMAZIONE

DESCRIZIONE DELLA STRINGA: "Ripeti: 01..."

S_2 : 01101010001010001010001000001010000

S_2 : Numeri primi \rightarrow QUANTITÀ MEDIA DI INFORMAZIONE

DESCRIZIONE DELLA STRINGA: "Se l'indice della posizione è un numero composto $\rightarrow 0$. Se l'indice della posizione è un numero primo $\rightarrow 1$ "

S_3 : 100000101110101101001010010000011101

S_3 : Sequenza casuale \rightarrow QUANTITÀ ALTA DI INFORMAZIONE

DESCRIZIONE DELLA STRINGA: "10000010111..."

È chiaro che una stringa possa avere più di una descrizione. Possiamo assumere che la quantità di informazione di una stringa è proporzionale alla sua descrizione più breve.

15/05/2020

Introduciamo un metodo per definire la quantità di informazione presente in una stringa: l'approccio basato sulla teoria della computazione (COMPLESSITÀ DI KOLMOGOROV).

Un primo modo per descrivere questo metodo è:

Data una stringa x , la quantità di informazione che essa contiene

È proporzionale a $|M|$, dove $M(x) = x$. In altre parole, la quantità di informazione che x porta è proporzionale alla complessità della TM atta a stampare x partendo dal nastro vuoto.

Il problema di questo specifico approccio è che è molto dispendioso, perché le stringhe di bit casuali, per essere stampate, richiederebbero all'interno uno stato diverso per ogni singolo bit. Kolmogorov trovò una soluzione più efficiente nel 1963.

Definizione ("descrizione" di $x \in \{0,1\}^*$):

È la codifica di una TM M e di un input $w \in \{0,1\}^*$ ($\langle M, w \rangle$) tale che $M(w) = x$.

È una definizione più vantaggiosa perché, ad esempio, permette di passare alla TM M un input $w = x$ nel caso in cui x sia una stringa casuale, casicché la lunghezza della codifica $\langle M, w \rangle$ dipende soprattutto dalla lunghezza dell'input; è chiaro che, per x composto da un pattern che si ripete, basta che w sia uguale al pattern e poi ci pensa M a replicarlo. E così via.

Ora concentriamoci più in dettaglio sul concetto di CODIFICA di qualche oggetto: sull'alfabeto $\{0,1\}$:

$$\rightarrow x \in \{0,1\}^* \implies \langle x \rangle = x \implies |\langle x \rangle| = |x|$$

$\rightarrow x, y \in \{0,1\}^*$ $\implies \langle x, y \rangle$ deve essere in modo tale che sia possibile distinguere la fine della stringa x in modo univoco utilizzando solo i bit 0, 1. Vediamo due possibilità:

i) $\langle x, y \rangle = \tilde{x}01y$, dove \tilde{x} è la stringa x coi bit raddoppiati.

ESEMPIO: $x = 001$, $y = 101 \implies \langle x, y \rangle = 000011\boxed{01}101$

In generale, $|\langle x, y \rangle| = 2|x| + |y| + 2$

ii) $\langle x, y \rangle = [|x|, x, y]$, in cui, prima della stringa x , viene specificata la lunghezza della stessa. Comunque sia, i bit che indicano la lunghezza di x vanno raddoppiati e seguiti dal separatore 01, sempre

per evitare ambiguità.

$$\text{Perciò, } |\langle x, y \rangle| = 2 \log_2 |x| + |x| + |y| + 2$$

Definizione:

La DESCRIZIONE MINIMALE $d(x)$ di una stringa $x \in \{0,1\}^*$ è la stringa $\langle M, w \rangle$ più corta che rappresenta una descrizione per x (se le stringhe $\langle M, w \rangle$ sono più di una, allora $d(x)$ è la prima nell'ordine lessicografico).

Definizione:

La COMPLESSITÀ DI KOLMOGOROV di una stringa $x \in \{0,1\}^*$ è $K(x) = |d(x)|$.

Teorema:

Esiste una costante c tale che $\forall x \in \{0,1\}^*$

$$K(x) \leq |x| + c$$

Dimostrazione:

Sia M una TM che si ferma non appena viene eseguita:

$$M(x) = x \quad \forall x \in \{0,1\}^*$$

$$\Rightarrow K(x) = |d(x)| \leq |\langle M, x \rangle| \leq \overbrace{2|\langle M \rangle| + 2}^c + |x| = |x| + c$$

Teorema:

Esiste una costante c tale che $\forall x \in \{0,1\}^*$

$$K(xx) \leq K(x) + c$$

Dimostrazione:

Costuiamo una TM M col seguente algoritmo:

Sull'input $\langle N, w \rangle$, dove N è una TM e w è una stringa:

1. Esegui N su w finché non termina con l'output x .
2. Fai una copia di x sul mastro.
3. Termina.

→ Possiamo supporre che $\langle N, w \rangle$ sia la descrizione minima di x

$$\text{Quindi: } K(xx) = |d(xx)| \leq |\langle M, d(x) \rangle| \leq \overbrace{2|\langle M \rangle| + 2}^c + |d(x)| = K(x) + c$$

Applicando la stessa dimostrazione, si può provare più in generale che

$$K(x^n) \leq K(x) + \log_2(n^2) + c$$

TERMINGHE CHE DERIVA DAGLI M STATI IN PIÙ CHE LA TM HA COSTRUIRE ELEVA AVERE RI-ESITO A H PER EFFETTUARE LE COPIE DI

Teorema:

Esiste una costante c tale che $\forall x, y \in \{0,1\}^*$ $K(xy) \leq 2K(x) + K(y) + c$

Dimostrazione:

Costruiamo una TM M col seguente algoritmo:

Sull'input $\langle\langle N_1, w_1 \rangle, \langle N_2, w_2 \rangle\rangle$, dove N_1, N_2 sono TM e w_1, w_2 sono stringhe

1. Esegui N_1 su w_1 e lascia il suo output x sul nastro.

2. Esegui N_2 su w_2 e lascia il suo output y alla destra di x .

3. Termina.

$$N_1(w_1) = x \quad N_2(w_2) = y$$

$$\text{Quindi: } K(xy) = |\hat{d}(xy)| \leq |\langle M, \langle \hat{d}(x), \hat{d}(y) \rangle \rangle| \leq 2|\langle M \rangle| + 2 + 2|\hat{d}(x)| + 2 + |\hat{d}(y)| = 2K(x) + K(y) + c \quad \blacksquare$$

Allo stesso identico modo si può dimostrare che \exists costante c tale che $\forall x, y \in \{0,1\}^*$ $K(xy) \leq 2\log_2(K(x)) + K(x) + K(y) + c$.

Teorema:

Non esiste alcuna costante c tale che $\forall x, y \in \{0,1\}^*$

$$K(xy) \leq K(x) + K(y) + c$$

Definizione:

Un LINGUAGGIO DI DESCRIZIONE è una funzione calcolabile $p: \Sigma^* \rightarrow \Sigma^*$.

Esempio:

$p \in \text{Python}$; $p(\text{"Programma in Python"}) = \text{"Risultato del programma"}$

Definizione:

$\hat{d}_p(x) := \text{"DESCRIZIONE MINIMALE"} \text{ DI } x \text{ RISPETTO A } p = \text{la prima stringa } s \text{ nello string order tale che } p(s) = x$.

Definizione:

La complessità di Kolmogorov rispetto a un linguaggio di descrizione p è $K_p(x) := |\hat{d}_p(x)|$.

Teorema di invarianta:

Per ogni linguaggio di descrizione p esiste una costante C_p tale che
 $\forall x \in \{0,1\}^* \quad K(x) \leq K_p(x) + C_p$ (LA COMPLESSITÀ DI KOLMOGOROV È OTTIMALE).

Dimostrazione:

p funzione calcolabile $\Rightarrow \exists \text{TM } P$ che calcola p .

Allora possiamo costruire una TM M col seguente algoritmo:

Sull'input w :

1. Esegui P su w e dai in output $p(w)$.

Quindi: $K(x) = |\tilde{d}(x)| \leq |\langle M, \tilde{d}_p(x) \rangle| \leq 2|\langle M \rangle| + 2 + |\tilde{d}_p(x)| = K_p(x) + C_p$

Definizione:

Una stringa $x \in \{0,1\}^*$ è c -COMPRIHIBILE se $K(x) \leq |x| - c$.
COMPRIMIBILE DI c BIT

Definizione:

Una stringa x che NON è c -comprimibile si dice INCOMPRESSIBILE PER c .

Definizione:

Una stringa incompressibile per 1 è INCOMPRESSIBILE.

Teorema:

Per ogni $m \in \mathbb{N}$ esiste almeno una stringa incompressibile di lunghezza m .

Dimostrazione:

Fissato $n > 0$, il numero di stringhe di lunghezza n è 2^n .

Il numero delle loro descrizioni di lunghezza $< n$ è al massimo:

$$1 + 2 + 2^2 + \dots + 2^{n-1} = \sum_{i=0}^{n-1} 2^i = 2^n - 1$$

\Rightarrow almeno una stringa di lunghezza n è incompressibile.

Corollario:

Almeno $2^n - 2^{n-c+1} + 1$ stringhe di lunghezza n sono incompressibili per c .

Teorema: Esiste una TM che, data una stringa, sia in grado di calcolarne la complessità di Kolmogorov.

La complessità di Kolmogorov $K: \{0,1\}^* \rightarrow \mathbb{N}$ è NON calcolabile.

Dimostrazione:

Per assurdo, assumiamo che K sia calcolabile $\Rightarrow \exists \text{ TM } KM$ che calcola K .

Allora possiamo costruire una TM M col seguente algoritmo:

Per qualunque input:

1. Ottieni, mediante il RT, la tua stessa descrizione $\langle M \rangle$.
2. Calcola $|\langle M \rangle|$.
3. Enumera tutte le stringhe $x \in \{0,1\}^*$ nello string order e, nel frattempo:
 - 4. Esegui KM su x per ottenere $K(x)$.
 - 5. Se $K(x) > |\langle M \rangle|$, allora metti x sul mastro e fermati.

Poiché M stampa una stringa x per qualunque input, abbiamo che $M(\varepsilon) = x$.

$\Rightarrow K(x) \leq |\langle M, \varepsilon \rangle| = |\langle M \rangle| \rightarrow$ IN CONTRADDIZIONE CON CIÒ CHE È PREVISTO DALL'ALGORITMO

\Rightarrow ASSURDO \Rightarrow la complessità di Kolmogorov NON è calcolabile.

In particolare, una TM che tenta di calcolare la complessità di Kolmogorov di una data stringa x potrebbe lavorare nel seguente modo:

Sull'input x , dove x è una stringa in $\{0,1\}^*$:

1. Enumera tutte le stringhe $y \in \{0,1\}^*$ nello string order e, nel frattempo:
 - 2. Se $y = \langle N, w \rangle$, allora esegui N sull'input w .
 - 3. Se $N(w) = x$, allora dai in output $|N, w| = K(x)$.

\Rightarrow Il problema di questa costruzione è che $N(w)$ potrebbe non terminare, per cui l'algoritmo potrebbe non interrompere mai la sua esecuzione!

Teorema:

Se il linguaggio $Halt_{\text{TM}}$ fosse decidibile, allora la complessità di Kolmogorov sarebbe calcolabile.

Teorema:

Se la complessità di Kolmogorov fosse calcolabile, allora il linguaggio $Halt_{TM}$ sarebbe decidibile.

Teorema:

Il linguaggio $J = \{x \in \{0,1\}^* \mid x \text{ è incompressibile}\}$ è NON decidibile.

Idea della dimostrazione:

Per assurdo, supponiamo che J sia decidibile.

Allora, possiamo costruire una TM il cui algoritmo consiste nel generare la prima stringa incompressibile nell'ordine lessicografico che abbia una lunghezza maggiore dell'algoritmo stesso.

Qui si avverrebbe a un assurdo fatto che, essendo la stringa più lunga dell'algoritmo che la genera, vuol dire che essa è in realtà compressibile. Perciò, per forza di cose, J è NON decidibile. \square

Le macchine Busy Beaver (Tibor Rado - 1962):

Sono delle macchine di Turing con:

- Un solo mastro doppiaamente infinito.
- $\Sigma = \{0\}$ $\Gamma = \{0,1\}$, $0 \equiv \sqcup$
- Il mastro inizialmente sempre vuoto (con soli 0).
- La testina LR-move.

Definizione:

Una macchina BB- n è una macchina Busy Beaver che ha n stati non terminali ($|Q \setminus F| = n$, dove $F = \{h\}$, h = stato di arresto).

Lemma:

$\forall n > 0$ ci sono $(4n+4)^{2n}$ macchine BB- n .

Dimostrazione:

Due macchine sono diverse se hanno tabelle di transizione diverse.
Le transizioni sono del tipo $S: (Q \setminus F) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$

$$\text{Quindi le possibilità sono } (|Q| \cdot |\Gamma| \cdot |\{L, R\}|)^{|\text{Q.F.} \cdot |\Gamma|} = \\ = ((2n+1) \cdot 2 \cdot 2)^{m \cdot 2} = (4m+4)^{2m}$$

Il BB-n contest:

Trovare la macchina BB-n che termini lasciando sul mastro il maggior numero possibile di 1 (tale numero è il "punteggio" (score) della macchina).

Definizione (funzione di Busy Beaver):

È una funzione $\Sigma: \mathbb{N} \rightarrow \mathbb{N}$ tale che

$$\Sigma(n) = \max \{ \text{punteggi delle macchine BB-n che si fermano} \}$$

Teorema:

La funzione Σ non è calcolabile.

Dimostrazione:

Σ cresce più velocemente di qualunque funzione calcolabile.

Corollario:

Il linguaggio $BB = \{ \langle M \rangle \mid M \text{ è una BB-n con punteggio pari a } \Sigma(n) \}$ è non decidibile.

Definizione (step function di Busy Beaver):

È una funzione $S: \mathbb{N} \rightarrow \mathbb{N}$ tale che

$$S(n) = \max \{ \text{numeri di mosse delle macchine BB-n che si fermano} \}$$

Teorema:

La funzione S non è calcolabile.

Dimostrazione:

$S(n) \geq \Sigma(n) \Rightarrow S$ cresce più velocemente di ogni funzione calcolabile.

Nella pagina seguente, viene riportato tutto ciò che sappiamo riguardo queste due funzioni.

n	$\Sigma(m)$	$S(m)$
0	0	0
1	1	1
2	4	6
3	6	21
4	13	107
5	> 4098	$> 47\ 176\ 870$
6	$\geq 3.5 \cdot 10^{18267}$	$\geq 7.4 \cdot 10^{36534}$
...		
1919		$\Sigma(1919)$ E $S(1919)$ NON POSSONO ESSERE DETERMINATI

22/05/2020

Introduciamo adesso la LOGICA MATEMATICA, che è quella branca della matematica che studia i fondamenti della matematica stessa, e vediamo qual è la sua relazione con la teoria della computabilità.

Definizione (logica del primo ordine):

È un insieme di regole che servono a costruire affermazioni che sono valide in ogni contesto (tautologicamente vere).

Un esempio è costituito dalla TAUTOLOGIE PROPOSIZIONALI:

→ PRINCIPIO DEL TERZO ESCLUSO : $A \lor \text{non } A \quad (A \vee \bar{A})$

→ PRINCIPIO DELLA CONSISTENZA: $\text{non} (A \text{ e non } A) \quad (\overline{A \wedge \bar{A}})$

Vediamo altri esempi di regole:

• MODUS PONENS: Se A è valido e $(A \text{ implica } B)$ è valido, allora B è valido $(A \wedge (A \rightarrow B)) \rightarrow B$

• ELIMINAZIONE DEI QUANTIFICATORI: A è valido per tutti gli $x \rightarrow A(x)$ è valido

• MANIPOLAZIONE DELLE REGOLE CON I QUANTIFICATORI: Se "non $(A(x)$ per tutti gli $x)$ " è valida, allora "esiste una x tale che non $A(x)$ " è valido.

...

In poche parole, la logica del primo ordine costituisce una serie di ASSIOMI sui quali sarà possibile costruire dei teoremi.

Facciamo ora un esempio pratico.

Gli ASSIOMI DI PEANO per i numeri non negativi:

- ESISTE UN VALORE CHIAMATO "ZERO": esiste un \mathbb{Z} tale che, per tutti gli x , $S(x)$ (il successore di x ; è una funzione) non è \mathbb{Z} .
- OGNI INTERO HA AL PIÙ UN PREDECESSORE: per ogni x, y , se $S(x) = S(y)$, allora $x = y$.

dal partire da questi due assiomi, siamo in grado di definire tutti i numeri interi non negativi:

$$S(x) \equiv x+1 ;$$

$$0 \equiv \mathbb{Z}$$

$$1 \equiv S(0)$$

$$2 \equiv S(1) \equiv S(S(0))$$

$$3 \equiv S(S(S(0)))$$

...

Definizione:

Una FORMULA è una stringa ben fondata sull'alfabeto $\{\wedge, \vee, \neg, (), [], \forall, \exists, x, \underbrace{R_1, \dots, R_x}_{\text{SIMBOLI DI RELAZIONE}}\}$.

→ Le variabili si possono rappresentare così: $x_1 \equiv x$ $x_2 \equiv xx$ $x_3 \equiv xxx$.

Definizione:

Una FORMULA ATOMICA DI ARITÀ K è una formula del tipo $R_j(\overbrace{x_1, \dots, x_k}^K)$

In particolare, se Φ' , Φ_1 , Φ_2 sono formule, un'altra stringa Φ è anch'essa una formula se è vera una delle seguenti affermazioni.

- Φ è una formula atomica
- $\Phi = \Phi_1 \wedge \Phi_2$
- $\Phi = \exists x [\Phi']$
- $\Phi = \bar{\Phi}'$
- $\Phi = \Phi_1 \vee \Phi_2$
- $\Phi = \forall x [\Phi']$

→ Una variabile non quantificata (senza \exists, \forall) è detta VARIABILE LIBERA.

→ Una formula senza variabili libere è detta SENTENZA.

Esempi:

- $\forall x_1 [R_1(x_1, x_2) \wedge R_3(x_3)]$ NON è una sentenza, VARIABILI LIBERE
- $\forall x_1 \exists x_2 \forall x_3 [R_1(x_1, x_2) \wedge R_3(x_3)]$ è una sentenza.

Definizione (universo):

È un insieme di valori che possono essere assegnati alle variabili.

Definizione (modello):

È un universo unito a un'assegnazione di determinate relazioni sopra gli elementi dell'universo ai simboli di relazione.

Esempio di modello:

$$U = \mathbb{N}$$

$$R_1 = \text{"minore o uguale a"}$$

→ È UNA RELAZIONE DI ARTÀ 2

ad questo modello appartengono le coppie di numeri come $(2, 2)$, $(2, 3)$, $(3, 5)$ ma NON $(5, 4)$, ad esempio.

D'ora in poi considereremo $R_1 \equiv \leq$, ovvero ci faccia comodo.

Definizione (linguaggio di un modello):

È una collezione di formule che usano solo i simboli di relazione assegnati dal modello e con la giusta arit.

In particolare, ogni sentenza di un linguaggio di un modello può essere vera o falsa in quel modello.

Esempio:

SENTENZA: $\forall x, y [R_2(x, y) \vee R_2(y, x)]$

Se assumiamo che $U = \mathbb{N}$, $R_2 \equiv \leq$, allora la sentenza è VERA.

Se assumiamo che $U = \mathbb{N}$, $R_2 \equiv <$, allora la sentenza è FALSA.

Facciamo altri esempi, assumendo che $U = \mathbb{N}$:

• ESISTENZA DI INFINITI NUMERI PRIMI (EUCLIDE, ~300 a.C.)

$$\forall q \exists p \forall x, y [p > q \wedge (x, y > 1 \rightarrow xy \neq p)]$$

VERO

• ULTIMO TEOREMA DI FERMAT •

$$\forall a, b, c, n [(a, b, c > 0 \wedge n > 2) \rightarrow a^n + b^n \neq c^n]$$

VERO

• CONGETTURA DEI NUMERI PRIMI GEMELLI

$$\forall q \exists p \forall x, y [p > q \wedge (x, y > 1 \rightarrow xy \neq p \wedge xy \neq p+2)]$$

Lo non sappiamo ancora se questa sentenza sia vera o falsa, ma la cosa certa è: essa è vera, o è falsa.

Definizione:

Se M è un modello, la TEORIA di M $\text{Th}(M)$ è l'insieme delle sentenze nel linguaggio del modello che sono vere in quel modello.

{

→ QUESTO È STRETTAMENTE CORRELATO A CIÒ CHE ABBIAMO FATTO FINORA: PROVARE A STABILIRE SE UNA DATA STRINGA APPARTIENE O NO A UN LINGUAGGIO.

Teorema:

$\text{Th}(\mathbb{N}, +)$ è decidibile.

Idea della dimostrazione:

Mostrare che $\text{Th}(\mathbb{N}, +) \leq_m \text{A}_{\text{DFA}}$ DECIDIBILE

In pratica, data una formula Φ , vogliamo costruire uno DFA che accetta la stringa vuota \emptyset se e solo se Φ è una sentenza vera.

Per dimostrarlo effettivamente il teorema, abbiamo bisogno di nuove nozioni che adesso introduciamo.

Inizialmente consideriamo i seguenti alfabeti:

$$\Sigma = \{\square\}$$

$$\Sigma_1 = \{\square, \blacksquare\}$$

$$\Sigma_i = \left\{ \begin{array}{c} b_1 \\ \vdots \\ b_i \end{array} \mid b_1, \dots, b_i \in \{0, 1\} \right\} \quad \forall i > 0$$

Lemme:

\exists DFA P con alfabeto di input Σ_3 che accetta sequenze in Σ_3^*

$$\begin{array}{|c|c|} \hline a_1 & b_1 \\ \hline a_2 & b_2 \\ \hline \vdots & \vdots \\ \hline a_i & b_i \\ \hline \end{array} \quad \begin{array}{|c|} \hline z_1 \\ \hline \end{array}$$

$$(a_1 b_1 \dots z_1) + (a_2 b_2 \dots z_2) = (a_3 b_3 \dots z_3)$$

→ NUMERI CODIFICATI DA STRINGHE BINARIE

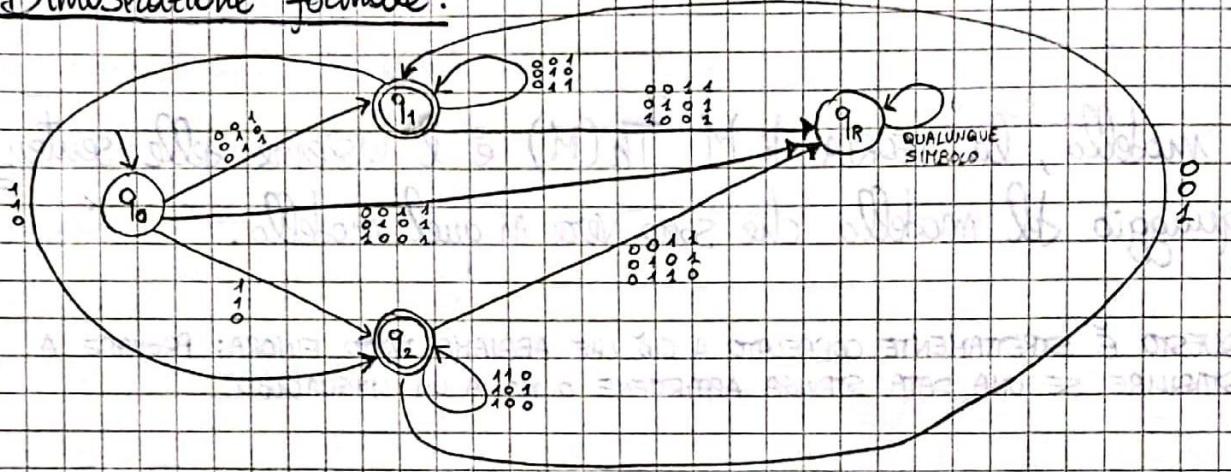
Idea della dimostrazione:

IDEA #1: Lavorare alla rovescia dall'ultimo simbolo della sequenza

(cosa possibile poiché i linguaggi regolari sono chiusi rispetto all'operazione di rovesciamento).

IDEA #2: Tenere traccia dei riporti delle somme di bit negli stati interni.

Dimostrazione formale:



Consideriamo adesso le seguenti formule:

$\Phi_0 := \Phi = Q_1 x_1 Q_2 x_2 \dots Q_K x_K [\Psi]$, dove i Q_i sono quantificatori e Ψ ha solo variabili libere

$\Phi_1 := Q_2 x_2 \dots Q_K x_K [\Psi]$

$\Phi_i := Q_{i+1} x_{i+1} \dots Q_K x_K [\Psi]$

$\Phi_K := \Psi$

Lemma:

\exists DFA A_K che accetta qualunque sequenza in Σ^* le cui righe corrispondono a K valori a_1, \dots, a_K che, quando assegnati alle variabili x_1, \dots, x_K soddisfano Ψ .

Dimostrazione:

Procediamo per induzione.

PASSO BASE:

Ψ è una formula atomica $\Rightarrow \Psi = R_s(x, y, z)$ NEL CASO CHE CI INTERESSA $\Rightarrow \Psi = "x+y=z"$ $\Rightarrow A_K$ è esattamente l'automa che abbiamo costruito per il lemma precedente.

PASSO INDUTTIVO:

Se Ψ NON è una formula atomica, si può comunque esprimere in formule più semplici per cui le quali si può costruire un DFA.

- $\Psi = \bar{\Psi}_1 \rightsquigarrow$ I linguaggi regolari sono chiusi rispetto al complemento ✓
- $\Psi = \Psi_1 \wedge \Psi_2 \rightsquigarrow$ I linguaggi regolari sono chiusi rispetto all'intersezione ✓
- $\Psi = \Psi_1 \vee \Psi_2 \rightsquigarrow$ I linguaggi regolari sono chiusi rispetto all'unione ✓

Non consideriamo i casi in cui Ψ è esprimibile con formule più semplici accompagnate da quantificatori poiché abbiamo inizialmente imposto che Ψ contiene solo variabili libere.

Lemma:

Supponiamo che \exists DFA A_{i+1} che accetta qualunque sequenza di Σ_{i+1}^* le cui righe corrispondono a valori delle variabili libere $x_1 \dots x_{i+1}$ che soddisfano Φ_{i+1} .

Se $Q_{i+1} = \mathbb{I}$, allora \exists DFA A_i che accetta qualunque sequenza di Σ_i^* le cui righe corrispondono a valori delle variabili libere $x_1 \dots x_i$ che soddisfano Φ_i .

Dimostrazione:

A partire dai simboli

a_1	b_1
\vdots	\vdots
a_i	b_i

$\in \Sigma_{i+1}^*$, dobbiamo lavorare con i corrispondenti simboli

a_1	b_1
\vdots	\vdots
a_i	b_i

ottenuti semplicemente eliminando l'ultima riga.

Il DFA A_i che andremo a costruire sarà perfettamente identico al DFA A_{i+1} che, per ipotesi, abbiamo già, solo che dovrà basarsi sull'alfabeto Σ_i anziché su Σ_{i+1} . Per fare ciò, dobbiamo badare a due accorgimenti:

→ fondamentalmente la componente $i+1$ dei simboli di Σ_{i+1} deve essere ignorata dal nuovo DFA. Infatti, può succedere che, nel DFA A_{i+1} , ci siano due transizioni a partire da uno stato q_j che arrivino in due stati differenti e sono rispettivamente marcate con i simboli

a_1	a_1
\vdots	\vdots
a_i	a_i

per cui, nel momento in cui vengono sostituiti con i simboli corrispondenti di Σ_i , che sono perfettamente identici, si crea non-determinismo.

Perciò, quel che abbiamo ottenuto è un NFA che accetta la sequenza data in input se almeno uno dei due percorsi di computazione è accettante.

e questo è perfettamente coerente col fatto che, per ipotesi, $Q_{its} = \exists$
 che è il quantificatore che permette di accettare la formula nel momento
 in cui c'è almeno un valore per cui l'affermazione vale.

→ Nel DFA A_{its} , a partire dallo stato iniziale, può partire una transizione
 marcatà col simbolo $\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$ che, nel nuovo automa divorrebbe $\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$

il quale, rappresentando solo degli zeri all'inizio di i stringhe numeri
 che, è del tutto plenastico; perciò, è possibile sostituire tale simbolo con la stringa vuota ϵ , confermando semplicemente il non-determinismo del nuovo automa.

Grazie all'equivalenza tra DFA e NFA, possiamo concludere che l'asserito è dimostrato. ■

Lemma:

Supponiamo che \exists DFA A_{its} che accetta qualunque sequenza di \sum_{its}^* le cui righe corrispondono a valori delle variabili libere $x_1 \dots x_{its}$ che soddisfano Φ_{its} .
 Se $Q_{its} = \forall$, allora \exists DFA A_i che accetta qualunque sequenza di \sum_i^* le cui righe corrispondono a valori delle variabili libere $x_1 \dots x_i$ che soddisfano Φ_i .

Dimostrazione:

$$\forall x_i [\Phi_i] \equiv \exists x_i [\bar{\Phi}_i]$$

⇒ Abbiamo ricordato la dimostrazione a quella del Lemma precedente. ■

Alla luce di tutti questi Lemmi, è possibile costruire un DFA che accetti $\epsilon \iff$ la formula $\Phi = Q_1 x_1 Q_2 x_2 \dots Q_k x_k$ è una sentenza vera.
 In definitiva, abbiamo provato che $\text{Th}(\mathbb{N}, +) \leq_m \text{A}_{DFA}$ \implies
 $\text{Th}(\mathbb{N}, +)$ è decidibile.

Teorema:

$\text{Th}(\mathbb{N}, +, \times)$ è NON decidibile.

Idea della dimostrazione:

Mostrare che $c_{\text{TM}} \leq_m \text{Th}(\mathbb{N}, +, \times)$

Per dimostrare effettivamente il teorema, abbiamo avvalerci del seguente asserto

Lemme:

Siano M una TM e w una stringa. Allora esiste una formula $\Phi_{M,w}$ sul linguaggio di $(\mathbb{N}, +, \times)$ che contiene una sola variabile libera x , e la sentenza $\exists x [\Phi_{M,w}] \iff M(w) \text{ accetta.}$

Idea della dimostrazione:

$\langle M \rangle$ è una stringa binaria $\implies \langle M \rangle$ è un numero in \mathbb{N}

w è una stringa binaria $\implies w$ è un numero in \mathbb{N} .

La storia della computazione è decodificata da una stringa binaria \implies
 \implies anch'essa è un numero in \mathbb{N} .

Tutti i controlli sulla validità della storia della computazione di $M(w)$ possono essere fatti tramite le operazioni di somma e prodotto sui numeri $\langle M \rangle, w, \langle \text{storia della computazione} \rangle$ CHE FOI SAREBBERE LA VARIABILE LIBERA X CITATA DAL LEMMA

Informalmente, abbiamo concluso che il modello $(\mathbb{N}, +, \times)$ è abbastanza potente da codificare una storia della computazione di una TM, per cui si ha che $c_{\text{TM}} \leq_m \text{Th}(\mathbb{N}, +, \times) \implies \text{Th}(\mathbb{N}, +, \times)$ è non decidibile. \square

Definizione:

Una DIMOSTRAZIONE FORMALE di una sentenza ψ è una sequenza di istuzioni S_1, \dots, S_k , dove $S_k = \psi$ e ogni altro S_i o è un assioma oppure può essere derivato dalle sentenze precedenti utilizzando le regole della logica del primo ordine.

~ LA CORRETTEZZA DI UNA DIMOSTRAZIONE FORMALE È DECIDIBILE.

Definizione:

Un SISTEMA FORMALE è composto da alcuni assiomi e dalle regole della logica del primo ordine.

NB: Sistema formale \neq Modello

LIVELLO SINTATTICO:

guarda dagli assiomi cosa si può ricavare e cosa no

LIVELLO SEMANTICO:

guarda la verità o falsità di certe affermazioni in un certo universo

Definizione:

Un sistema formale si dice:

→ **CONSISTENTE** se non esiste alcuna sentenza P tale che sia P che $\neg P$ possono essere derivate (dagli assiomi).

→ **COMPLETO** se, per qualunque sentenza P , o P oppure $\neg P$ può essere derivato.

→ **ROBUSTO** se qualunque sentenza P che può essere derivata è vera in qualsiasi modello del sistema formale.

NB: Robustezza \Rightarrow Consistenza

Consistenza $\not\Rightarrow$ Robustezza

Teorema di completezza di Gödel:

Qualunque insieme di assiomi consistente ha un modello.

Teorema:

Nel sistema formale $(N, +, \times)$, la collezione di sentenze dimostrabili è...

Dimostrazione:

Basta esibire una TM col seguente algoritmo:

Sull'input Ψ , dove Ψ è una sentenza:

1. Genera tutte le stringhe nello string order e, nel frattempo:
2. Scarta ogni stringa che non codifica una dimostrazione formale.
3. Se è stata trovata una dimostrazione formale per Ψ , accetta.

Teorema:

Alcune sentenze in $\text{Th}(N, +, \times)$ non sono derivabili.

Dimostrazione:

Per assurdo, assumiamo che tutte le sentenze siano derivabili. Allora

è possibile costruire una TM D col seguente algoritmo:

Sull'input Ψ , dove Ψ è una sentenza:

1. Esegui l'algoritmo P (mostrato nella dimostrazione del precedente teorema) in parallelo sugli input $\Psi, \bar{\Psi}$.
2. Se P accetta Ψ , allora accetta.
3. Se P accetta $\bar{\Psi}$, allora rifiuta.

In questo modo, abbiamo ottenuto che D decide $\text{Th}(\mathbb{N}, +, \times)$ \Rightarrow
 \Rightarrow ASSURDO \Rightarrow alcune sentenze in $\text{Th}(\mathbb{N}, +, \times)$ non sono decidibili. ■

Teorema:

Nella teoria $\text{Th}(\mathbb{N}, +, \times)$ NON è possibile dimostrare un'affermazione analoga a "Questa sentenza non può essere dimostrata" nonostante sia vera.

Dimostrazione:

Costruiamo una TM S col seguente algoritmo:

Su qualunque input:

1. Ottieni, attraverso il RT, la tua stessa descrizione $\langle S \rangle$.
2. Costruisci la sentenza $\Psi_0 = \exists x[\varphi_{S,0}]$.
3. Esegui il solito algoritmo P sull'input Ψ_0 .
4. Se P accetta Ψ_0 , allora accetta.

In questo modo, Ψ_0 è vera $\Leftrightarrow S$ non accetta 0

\rightarrow Se P trova una dimostrazione per $\Psi_0 \Rightarrow S(0)$ accetta $\Rightarrow \Psi_0$ è falsa
 $\Rightarrow (\mathbb{N}, +, \times)$ è inconsistente \Rightarrow assurdo

\rightarrow Se P non trova una dimostrazione per $\Psi_0 \Rightarrow S(0)$ non accetta $\Rightarrow \Psi_0$ è vera

Primo Teorema di incompletezza di Gödel:

Nessun sistema formale sufficientemente potente può essere sia completo che robusto.

Teorema di Rosser:

Nessun sistema formale sufficientemente potente può essere sia completo che consistente.

Secondo teorema di incompletezza di Gödel:

Nessun sistema formale sufficientemente potente che sia consistente può dimostrare la sua stessa consistenza.

23/05/2020

COMPLESSITÀ COMPUTAZIONALE

In questo mese LA

+