Calcolabilitá e Complessitá

Daniel Biasiotto

May 31, 2022

CONTENTS

- Prof: Stefano Berardi
- PDF Version

1 INFO CORSO

- Testo:
  - Introduction to the theory of Computation
- Link:
  - `https://turingmachinesimulator.com/`

2 TEORIA

2.1 Macchina di Turing

The Church-Turing Thesis Utilizzata per definire un problema risolvibile

- nato per dare una definizione semplice di risolvibilitá
  - un qualsiasi computer puó essere ridefinito con una `MdT` equivalente
- unico accesso alla memoria
  - come un nastro, legge dall'inizio fino alla fine
    - lenta
    - non é una architettura realmente implementabile

Caratterizzata da:

- controllo
  - stati interni
    - che definiscono comportamenti diversi
- memoria/tape
  - testa di lettura
  - alfabeto a scelta
- l'Input é finito
- la fine del Input é marcata con un simbolo speciale

A differenza degli Automi a Stati Finiti puó sovrascrivere, appunta per aiutarsi nell'esecuzione

- una macchina di turing puó simulare qualsiasi macchina a differenza di un DFA

La Macchina di Turing essendo estremamente semplice é ottima per lo studio della Calcolabilitá

2.1.1 *Definizione Formale*

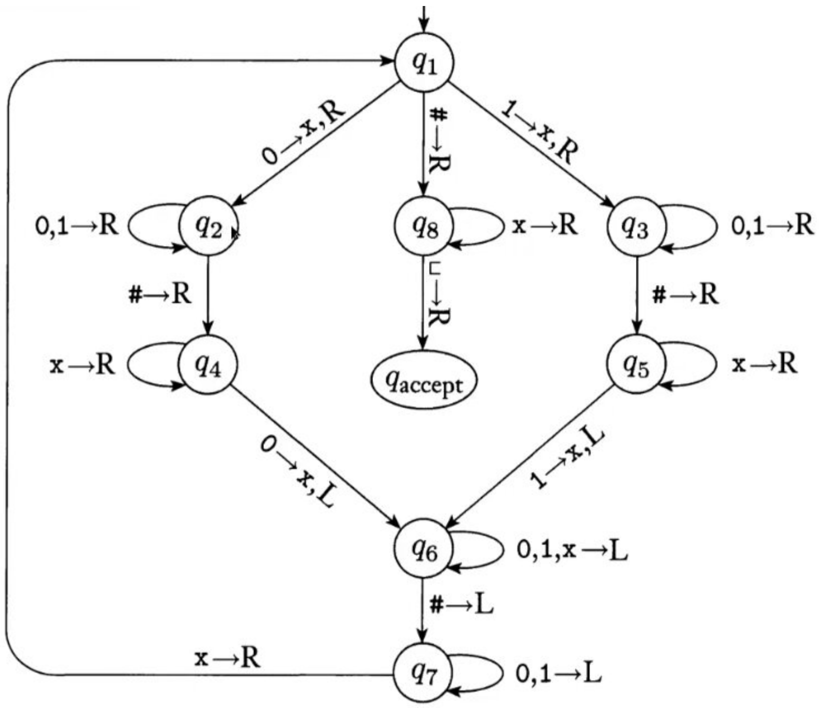
`MT 1936` 7-tupla $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

1. Q: insieme di stati
2. Σ: alfabeto di Input non contenente il simbolo di blank
3. Γ: alfabeto del nastro
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$: funzione di transizione
    - Left (<) / Right (>)
5. q_{accept}

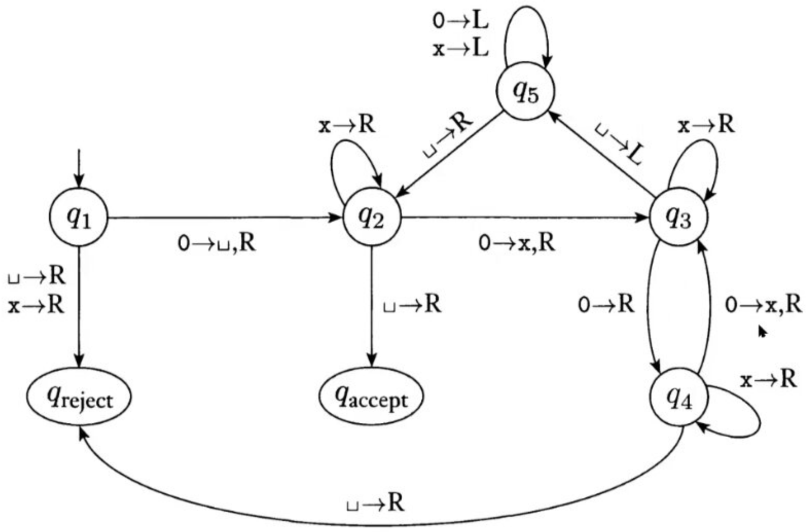
- Lo stato di rifiuto non viene inserito, é sottointeso a ogni Input non riconosciuto una transizione allo stato di rifiuto, bloccante
- Lo stato `S` di `StayPut` é simulabile muovendosi continuamente `L` e `R`, raddoppiando gli stati
- Il nastro infinito a destra e sinistra si simula sulle celle pari e dispari su un nastro infinito solamente a destra

2.1.2 *Grafi*

STRINGHE UGUALI



STRINGA DI 0 DI LUNGHEZZA 2^N



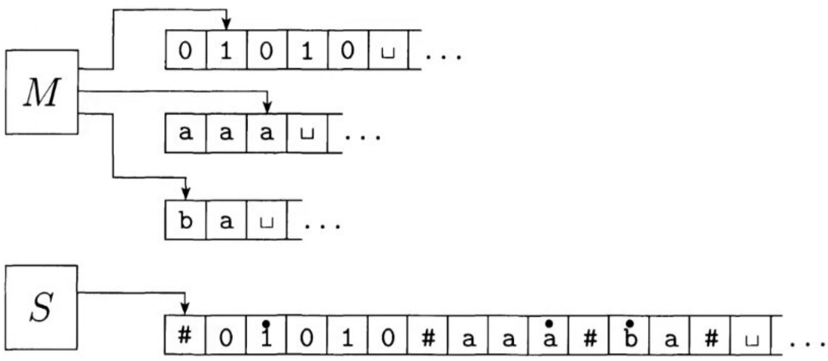
2.1.3 *Macchine Turing a piú registri*

Con piú *tapes* diventa molto piú semplice controllare stringhe, ci si avvicina nel comportamente ad una macchina di Von Neumann Il primo nastro é l'Input, gli altri sono registri $\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k$

- con k simboli / tapes
- S é `StayPut`

Si puó simulare una `MT` a piú nastri con una `MT` ad un nastro solo

- perché? per semplificare la definizione di calcolo
- se un algoritmo a piú nastri é lineare su un nastro sará **quadratico**



2.1.4 *Enumeratori*

Sono TM che enumerano stringhe.

THEOREM Si puó definire un enumeratore E che enumeri un linguaggio L se e solo se esiste una TM M che riconosca (decida positivamente) il linguaggio L PROOF Partiamo a dimostrare che se esiste un tale E allora esiste una M che riconosca il linguaggio enumerato L

Possiamo definire M su input w:

- simula E
- per ogni stringa enumerata da E confrontala con w
- se w é uguale *accetta*, altrimenti continua con la simulazione di E

Da questa costruzione si evince che M accetta solo le stringhe enumerate da E e quindi nel linguaggio L, M riconosce L.

Ora si dimostra l'altra direzione. Se M riconosce il linguaggio L definiamo un enumeratore E:

- ignora l'input
- ripeti per $i = 0, 1, \cdots$
  - esegui per i passi M su $s_1, s_2, s_3, \cdots$
  - se M accetta, stampa la s_j accettata

Questa macchina di turing E simula M su tutte le stringhe s_j che appartengono a Σ^* per i passi di simulazione, non terminando mai. In questa simulazione sostanzialmente si simula in parallelo la macchina M su tutte le stringhe possibili in input (per un numero di passi di computazione sempre maggiore), stampando tutte e sole le s_j accettate da M. Viceversa se una stringa appartiene ad L questa viene accettata in un numero finito di passi da M, e quindi dato abbastanza tempo E la stamperá. Quindi E enumera il linguaggio L.

2.2 Decidibilitá

Per un DFA possiamo definire una TM M che lo simula e verifica l'accettazione o meno dell'Input Decidable - Turing-recognizable

- NFA convertibili
- RegEx convertibili

2.2.1 *Definizioni*

Sia L un linguaggio definito sull'alfabeto Σ, e quindi sottoinsieme di Σ^* Allora $\forall w \in \Sigma^*$:

- Decidibile, esiste una M che decide L
  - $w \in L$: M accetta w
  - $w \notin L$: M non accetta w
- Positivamente Decidibile (*riconoscibile*)
  - $w \in L$: M accetta w
  - $w \notin L$: M non accetta w o non termina
- Negativamente Decidibile
  - $w \in L$: M accetta w o non termina
  - $w \notin L$: M non accetta w

Allora definiamo $\overline{L} = \{w \in \Sigma^* \mid w \notin M\}$ **linguaggio complemento** di L Per i linguaggi complemento si scambiano decidibilitá positiva e decidibilitá negativa:

- L decidibile $\iff$ $\overline{L}$ decidibile
- L positivamente decidibile $\iff$ $\overline{L}$ negativamente decidibile
- L negativamente decidibile $\iff$ $\overline{L}$ positivamente decidibile

Esistono indebolimenti del decisore, ovvero decisori *parziali*

2.2.2 *Teorema di Post*

`4.22` Linguaggio L decidibile $\iff$ é positivamente e negativamente decidibile

- M termina sempre $\forall w \in \Sigma^*$
- M é un decisore che simula M_1 e M_2 in parallelo
  - il primo che termina decide

Riformulando

- un linguaggio é decidibile esattamente quando esso e il suo complemento sono positivamente decidibili

proof Si dimostra prima una direzione e poi l'altra della bi-implicazione

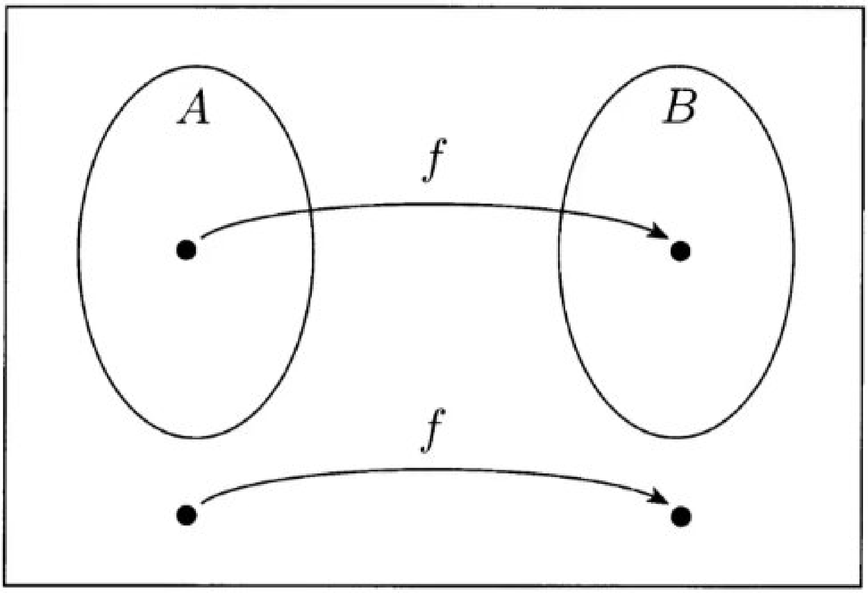
1. $\Rightarrow$
   - Se A é decidibile allora segue direttamente che A e $\overline{A}$ sono positivamente decidibili

  - per definizione di decidibilitá e complemento di un linguaggio

2. $\Leftarrow$
   - Se A e $\overline{A}$ sono positivamente decidibili, definiamo M_1 e M_2, decisori positivi di uno e dell'altro
   - Si definisce M, decisore di A
     - $M =$ Su input w:
       a) Esegui M_1 e M_2 sull'input w in parallelo
       b) Se M_1 accetta, *accept*; se M_2 accetta, *rifiuta*
   - Ogni stringa w appartiene a A o $\overline{A}$
     - Segue che per qualsiasi input una tra M_1 e M_2 deve accettare
   - M termina quando una tra M_1 e M_2 accetta
     - Segue che M termina sempre, quindi é un decisore
   - Inoltre M accetta tutte le $w \in A$ e rifiuta tutte le $w \notin A$, quindi M é un decisore per A
     - A quindi é decidibile in quanto ne esiste un decisore M ■

2.2.3 *Mapping Reducible Language*

Il Linguaggio A é *mapping reducible* al linguaggio B:
$A \leqslant_m B$
se esiste una *funzione computazionale* f tale che:
$w \in L(A) \iff f(w) \in L(B)$



Seguono i corollari:

- Se $A \leqslant_m B$ e B é decidibile $\implies$ A é decidibile
  - si dimostra costruendo la funzione e poi eseguendo B sull'input trasformato
  - stessa dimostrazione per la decidibilitá positiva
- Se $A \leqslant_m B$ e A non é positivamente decidibile $\implies$ B non é positivamente decidibile
  - **proof** Supponendo $A = \overline{A_{TM}}$
    1. $A \leqslant_m B$
    2. $\overline{A} \leqslant_m \overline{B}$
    3. $A_{TM} \leqslant_m \overline{B}$
       * ma se $\overline{B}$ fosse negativamente decidibile (quindi B positivamente decidibile) allora per la proprietá di cui sopra A_{TM} sarebbe negativamente decidibile
       * Ma A_{TM} non puó esserlo, altrimenti sarebbe decidibile per il Teorema di Post: contraddizione. ■

2.2.4 *Macchina di Turing Universale*

U = "Su input $\langle M, w\rangle$, dove M é una TM e w é una stringa"

1. Simula M su w
2. Se M accetta, *accetta*; se M rifiuta, *rifiuta*

Se M cicla, U cicla di conseguenza
La macchina universale é definita a partire da M codificando in un alfabeto binario tutti i simboli di M. La macchina U é definita utilizzando un alfabeto $\Sigma = \{0, 1\}$, quindi un qualsiasi stato o simbolo s di M sará convertibile in una stringa binaria $s^* \in \Sigma^*$ Nelle tape di U tutti i simboli sono delimitati da #.

Queste codifiche sono utilizzate nelle 5 tape di U, definite in questo modo:

1. la funzione di transizione σ di M, questa tape é read-only e qui sono listate tutte le transizioni di M nella forma $q^*, a^*, q'^*, a'^*, m^*$ dove a sono simboli di M e m sono L o R
2. lo stato corrente di M, q^*
3. lo stato accettante di M, q^*_{accept}
4. lo stato di rifiuto di M, q^*_{reject}
5. la tape di simulazione di M

La macchina universale procede leggendo lo stato corrente di M e il simbolo a^* che si trova sotto la testina di lettura di nella tape 5. Quindi scorre le quintuple nella prima tape, se non trova una corrispondenza rifiuta. Se trova una corrispondenza allora sovrascrive la tape 2 con il nuovo stato indicato dalla funzione di transizione e sovrascrive a^* nella tape 5 con la nuova a'^* indicata dalla transizione e aggiungendo un divisore #. fatto questo simula il movimento a destra o a sinistra della testina di M spostandosi nella direzione indicata fino ad un #.

2.2.5 *Problemi Decibidili*

$E_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$

- decidibile studiando i percorsi nel grafo delle transizioni

$EQ_{\text{DFA}} = \{\langle A \rangle \mid A \text{ is a DFA and } L(A) = \emptyset\}$

- automa che descrive la differenza simmetrica dei linguaggi
- si riduce a E_{DFA}

$A_{\text{CFG}} = \{\langle G, w \rangle \mid G \text{ is a CFG that generates string } w\}$

- tempo di accettazione 2^n
- non c'é problema di fermata

$E_{\text{CFG}} = \{\langle G \rangle \mid G \text{ is a CFG and } L(G) = \emptyset\}$

2.2.6 *Problemi Indecidibili*

Per molti problemi si utilizza la tecnica della riduzione

- se un problema che sappiamo non decidibile si puó ridurre al problema che stiamo studiando allora anche questo non sará decibidile

EGUAGLIANZA CHOMPSKY $EQ_{\text{CFG}} = \{\langle G, H \rangle \mid G \text{ and } H \text{ are CFGs and } L(G) = L(H)\}$

ACCETTAZIONE 4.11 Problema positivamente decidibile
PROOF Si procede per *diagonalizzazione* utilizzando due `TM` di supporto H e D
$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

- simulabile con una macchina U di Turing universale
  - macchina capace di simulare qualsiasi macchina utilizzando 5 tape
- si osserva l'esecuzione che non termina

Si prova utilizzando la tecnica della *diagonalizzazione* scoperta dal matematico Georg Cantor nel 1873

- iniezione - suriezione (biezione)
  - corrispondenza 1 a 1
- prova che non esiste una enumerazione per un dato insieme di numeri
  - per i Reali si cambia nella ennesima enumerazione la ennesima cifra dopo la virgola
    * si trova cosí un numero che differisce per una cifra da tutti i numeri enumerati
- esistono infinite terne

proof Si definiscono delle `MT` di supporto:

$$H(\langle M, w\rangle) = \begin{cases} \textit{accept} & \text{if } M \text{ accepts } w \\ \textit{reject} & \text{if } M \text{ does not accept } w \end{cases}$$

- supponiamo che `H` esista, e accetti se `M` accetta `w` e rifiuti altrimenti

$$D(\langle M\rangle) = \begin{cases} \text{accept} & \text{if } M \text{ does not accept } \langle M\rangle \\ \text{reject} & \text{if } M \text{ accepts } \langle M\rangle \end{cases}$$

- `D` prende in input una macchina `M` e con un decisore `H` che decide `M` con input la propria descrizione $\langle M\rangle$, accetta se `H` rifiuta e viceversa, continua con altre macchine
  - diagonalizza infinite macchine `M`

Allora si procede diagonalizzando con D applicato a $\langle D\rangle$

$$D(\langle D\rangle) \begin{cases} \text{accept} & \text{if } D \text{ does not accept } \langle D\rangle \\ \text{reject} & \text{if } D \text{ accepts } \langle D\rangle \end{cases}$$

- dovrebbe rifiutare se D accetta
- dovrebbe accettare altrimenti
  - non puó terminare perché per terminare avrebbe bisogno di dare la risposta opposta di se stesso

<u>Abbiamo raggiunto una contraddizione</u> ∎

IMMORTALITÁ `4.23` $\overline{A}_{TM}$ <u>positivamente decidibile</u> $\Longrightarrow$ A_{TM} <u>negativamente decidibile</u> per `T.Post`

- Falso per `4.11`

FERMATA 5.1 Il problema della decisione per L_1 si riduce al problema della decisione per L_2 se sappiamo trasformare un decisore per L_2 in un decisore per L_1

$\text{HALT}_{\text{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ halts on input } w\}$

- $A_{\text{TM}} <_m \text{HALT}_{\text{TM}}$

proof Per contraddizione. Supponiamo esista una `TM` R che decida la fermata, definiamo una `TM` S che decide l'accettazione. Ma l'accettazione non é decidibile. Definiamo S su input w:

- Se R accetta $\langle M, w\rangle$ procedi, altrimenti rifiuta
- Simula M su w, se accetta fa altrettanto, altrimenti rifiuta

$A_{TM} \leqslant_m \text{HALT}_{TM}$ in quanto se R accetta significa che M termina, accettando o rifiutando. Se diverge w non appartiene al linguaggio riconosciuto da M e S puó rifiutare. Per ció S accetta tutte e sole le stringhe in L, ovvero riconosciute da M.

Ma questa é una contraddizione in quanto si dimostra che A_{TM} non é decidibile. ■

DECIBIDILITÁ DEI LINGUAGGI DI CHOMPSKY *Simboli, Produzioni, Terminali* Un linguaggio definibile da una grammatica in forma normale di Chompsky é detto `context-free` Si dimostra che il numero di passi per derivare una stringa di lunghezza n é $2n - 1$

Questo implica che il problema é decidibile, anche se in tempo esponenziale

- si scrivono sulla tape 2 tutte le deduzioni di lunghezza $2n - 1$
- si controlla la correttezza una ad una, se ne si trova una corretta e che corrisponde accettiamo, altrimenti continuiamo, se alche l'ultima non va bene rifiutiamo

Per ridurre la complessitá si utilizza la **programmazione dinamica**

- ci si appunta i risultati intermedi

EMPTYNESS 5.2 Si dimostra per assurdo, se esistesse si potrebbe risolvere l'accettazione

- si riduce a A_{TM}
  - $A_{\text{TM}} <_m E_{\text{TM}}$

proof Per contraddizione. Supponiamo esista una R tale che decida la emptyness, dato una stringa di input w si modifica M per accettare solo questa stringa. Definiamo M, su input x:

- se $x \neq w$ rifiuta

- altrimenti accetta

Questa macchina decide il linguaggio che contiene la sola stringa w.

Allora S, su input $\langle M, w\rangle$:

- costruisce la M modificata come specificato
- esegue R su M, se R accetta allora rifiuta, e viceversa

In questo modo abbiamo ridotto l'accettazione alla emptyness: R rifiuta se e solo se M accetta w, e quindi il linguaggio L riconosciuto da M non é vuoto. Viceversa se M rifiuta w allora R accetterá in quanto L riconosciuta da M é il linguaggio vuoto. Quindi S decide l'accettazione. Contraddizione in quanto l'accettazione é non decidibile. ■

EQUALITY 5.4 Intesa tra due `MT`

- se sapessi deciderla potrei decidere anche l'`Emptyness`
  - In quanto E_{TM} é considerabile un caso particolare di EQ_{TM}
  - tra una macchiana e la macchina che rifiuta sempre

Anche per i reali:

- calcoli diversi portano anche arrotondamenti diversi, per questo reali rigorosamente uguali possono risultare diversi
- $A_{\text{TM}} <_m EQ_{\text{Real}}$
  - e di conseguenza anche il < e il >

$EQ_{TM} = \{\langle M_1, M_2\rangle \mid L(M_1) = L(M_2)\}$ ■ PROOF Si dimostra per riduzioni:

1. $A_{TM} \leqslant_m \overline{EQ}_{TM}$
   - questo indica che EQ_{TM} non puó essere negativamente decidibile
   - spostiamo al decidibilitá a A_{TM}
2. $\overline{A}_{TM} \leqslant_m EQ_{TM}$
   - questo indica che EQ_{TM} non puó essere positivamente decidibile

Ora basta raggiungere queste conclusioni per chiudere la dimostrazione.

1. Definisco una macchina F che implementa la funzione f che riduce A a $\overline{EQ}$
   - $\langle M, w\rangle \rightarrow^F \langle M_1, M_2\rangle$

- se $L(M_1) \neq L(M_2)$ allora M accetta w
  - M_1 rifiuta sempre
    * $q_0 = q_{reject}$
  - M_2
    * prende x e lo ignora
    * esegue M su w e accetta se M accetta
      · $\begin{cases} M \text{ accetta}: & L(M_2) = \Sigma^* \\ M \text{ non accetta}: & L(M_2) = \emptyset \end{cases}$
  - $L(M_1) \neq L(M_2) \iff M$ accetta w

2. Definisco una Macchina G che implementa la funzione g che riduce $\overline{A}$ a EQ
   - $\langle M, w \rangle \rightarrow^G \langle M_1, M_2 \rangle$
   - se $L(M_1) \neq L(M_2)$ allora M non accetta w
     - M_1 accetta sempre
       * $q_0 = q_{accept}$
     - M_2
       * prende x e lo ignora
       * esegue M su w e accetta se M accetta
         · $\begin{cases} M \text{ accetta}: & L(M_2) = \Sigma^* \\ M \text{ non accetta}: & L(M_2) = \emptyset \end{cases}$
     - $L(M_1) \neq L(M_2) \iff M$ non accetta w ■

CORRISPONDENZA DI POST `PCP - 4.22`

$A_{TM} \leqslant_m PCP$

Questo problema (domino) contiene la Macchina di Turing

- in quanto corrisponde alla visualizzazione della Configurazione di una TM
  - visualizzando la storia del calcolo della macchina

Si definisce un *Modified Post Correspondance Problem*:

$A_{TM} \leqslant_m MPCP \leqslant_m PCP$

Si decide che il primo elemento dell'insieme deve essere utilizzato all'inizio

- sopra abbiamo $n - 1$ passi di calcolo
- sotto abbiamo n passi di calcolo

Questi *domini* rappresentano le funzioni di transizione attraverso le configurazioni della `TM`

- $[\frac{\#qa}{\#rb}]$
  - $\delta(q, a) = (r, b, L)$

- compresi i pezzi dei singoli simboli, che si mantengono da un istante all'altro se non toccati dalla trasformazione di stato
  - $[\frac{1}{1}]$
  - $[\frac{0}{0}]$
  - $[\frac{\sqcup}{\sqcup}]$
  - $[\frac{\#}{\sqcup\#}]$
    * utilizzato quando lo stato deve spostarsi a destra oltre l'ultimo simbolo

Si devono definire dei domino per l'accettazione, che faccia *match*: $[\frac{q_{accept}\#\#}{\#}]$ Per arrivare a questo *accept*: $\forall a \in \Gamma$

- $[\frac{a\, q_{accept}}{q_{accept}}]$
- $[\frac{q_{accept}\, a}{q_{accept}}]$

TASSELLAZIONE – WANG TILES Wikipedia Solo negativamente decidibile

- le tassellazioni aperiodiche sono utilizzate per la sintesi procedurale di texture, heightfields

Si dimostra che WANG non é positivamente decidibile in quanto

- $\overline{\text{HALT}} \leqslant_m \text{WANG}$
- procedendo in maniera non deterministica, il caso di *non-rifiuto* indica che un albero della computazione ha per caso scelto la configurazione corretta per risolvere il problema della tassellazione
- la computazione non deterministica si ferma solo in caso di rifiuto di tutti i rami non deterministici, quindi se la computazione non si ferma si dovrebbe accettare

ESISTENZA DI UN DFA EQUIVALENTE 5.3 $A_{TM} <_m \text{REGULAR}_{TM}$

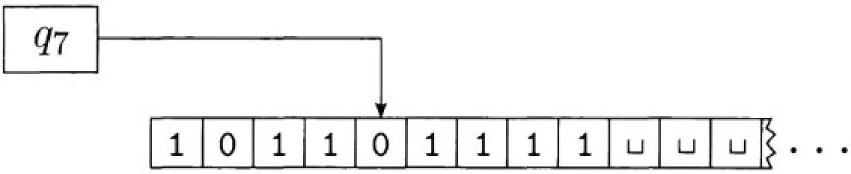


Figure 1: configurazione di $1011q_701111$

2.2.7 *Configurazione di una TM*

2.2.8 *Recap*

`../media/img/decidability.jpg`

- Negativamente Decidibili
  - E_{TM}
  - $\overline{A}_{TM}$
  - ALL_{CFG}
  - WANG
- Decidibili
  - E_{CFG}
  - A_{CFG}

    - $\mathrm{EQ}_{\mathrm{DFA}}$

- Positivamente Decidibili
    - $\overline{E}_{TM}$
    - A_{TM}
    - $\mathrm{HALT}_{\mathrm{TM}}$
    - PCP
        - Corrispondenza di Post
- Né negativamente né positivamente decidibili
    - $\mathrm{REGULAR}_{\mathrm{TM}}$
    - $\mathrm{EQ}_{\mathrm{TM}}$
    - $\mathrm{CONTEXT\text{-}FREE}_{\mathrm{TM}}$
    - $\mathrm{ALL}_{\mathrm{TM}}$
        - se un programma accetta sempre

2.3 Complessitá Temporale

Trattata nel corso di Algoritmi: Complessitá di un algoritmo Per lo studio della complessitá consideriamo la Macchina di Turing (1 registro)

- questo in quanto la complessitá varia anche in base all'architettura

Il tempo di calcolo della macchina M é definito come
$f : \mathbb{N} \to \mathbb{N}$ dove $f(n)$ é il numero massimo di passi compiuti dalla macchina M
Si utilizza la *notazione asintotica* o **big-O Notation**

- O-grande

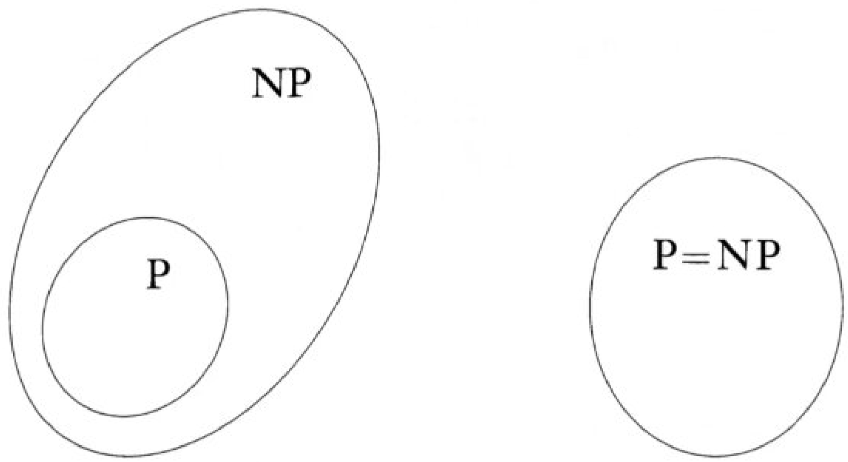
$$\mathrm{TIME} = \{L \mid L \text{ risolvibile da =TM= deterministica in } O(f(n)) \text{ polinomiale}\}$$

$$\mathrm{NTIME} = \{L \mid L \text{ risolvibile da =TM= non deterministica in } O(f(n)) \text{ polinomiale}\}$$

Generalmente:

- P = classe dei linguaggi la cui appartenenza puó essere decisa velocemente
- NP = classe dei linguaggi la cui appartenenza puó essere verificata velocemente

Non si é riuscita a provare l'esistenza di un singolo linguaggio NP che non sia in P
Piú grande problema aperto: $P = NP$



2.3.1 *P*

Teorema `7.8` Sia $t(n)$ una funzione t.c. $t(n) \geqslant n \implies$ qualsiasi macchina *multitape* M con tempo $t(n)$ ha un equivalente $O(t^2(n))$ in una macchina M' *singletape*

- chiaro riprendendo la simulazione di *multitape* in *singletape*
- un passo della simulazione *singletape* impiega al massimo $O(t(n))$ passi

La classe di tempo **Polinomiale** é definito come

$$P = \bigcup_k \text{TIME}(n^k)$$

2.3.2 *Non Determinismo*

Teorema `7.11` Sia $t(n)$ una funzione dove $t(n) > n$. Allora ogni `TM` *singletape* non deterministica con complessitá temporale $t(n)$ ha una equivalente `TM` determinitistica $2^{O(t(n))}$, nel caso di una macchina multiregistro Per una `TM` deterministica a registro singolo si avrá sempre complessitá $2^{O(t(n))2} = 2^{O(t(n))}$

L'esplorazione dell'albero non deterministico é svolto utilizzando *l'ordine lessicografico*

- in profonditá
- questo é posto nell'*address tape* della macchina **deterministica** corrispondente
- a livello n l'albero ha massimo k^n nodi con k numero di possibili figli
- il numero di passi necessari all'esplorazione dell'albero é $2^{O(m)}$

  - m profonditá dell'albero

RAGGIUNGIBILITÁ $\text{PATH} = \{\langle G, s, t\rangle \mid G$ é diretto con un cammino da s a $t\}$ La soluzione banale non deterministica ha $2^{O(t(n))}$ esponenziale

Con un algoritmo marcando i nodi man mano che vengono scoperti si raggiunge complessitá polinomiale

- rappresentando il grafo con liste di adiacenza la si puó stimare $O(n)$ nel numero di archi

ALGORITMO DI EUCLIDE RELPRIME, il `MCD` tra due numeri Relativamente Primi é 1 $\text{MCD}(x, y) = \text{MCD}(x \mod (y), y)$ quindi procediamo: $(x, y) \to (x \mod y, y) \to (y, x \mod y) \to \cdots \to (x, 0)$ $\text{MCD}(x, 0) = x$

I passi sono eseguiti $min(2\log_2 x, 2\log_2 y)$ ovvero proporzionali al numero di cifre nella rappresentazione binaria: $O(n)$ quindi polinomiale

GRAMMATICHE DI CHOMPSKY Per migliorare la complessitá si cerca di derivare tutte le sottostringhe di lunghezza crescente della stringa di input

- si memorizzano le soluzioni delle sottostringhe
  - per ogni sottostringa la si divide in sottostringhe e si guarda la soluzione delle sottostringhe
  - in una rappresentazione matriciale la soluzione si trova nella riga precedente
- ogni controllo richiede $O(1)$ in quanto le sottostringhe sono sempre riconducibile ai siboli terminali

Con questo algoritmo si raggiunge $O(n^3)$

2.3.3 NP

Un linguaggio é `NP` $\iff$ é deciso da un algoritmo non deterministico polinomiale Un $M : O(n^k)$ `NTM` equivale a $M' : 2^{O(n^k)}$ `TM`

- da tempo polinomiale a tempo esponenziale

$$\text{NP} = \bigcup_k \text{NTIME}(n^k)$$

Un linguaggio é `NP` se dispone di un *verificatore* in tempo polinomiale, detto allora *polinomialmente verificabile*

Def `7.18` Un **verificatore** é una macchina di turing V tale che per un linguaggio A:

- $A = \{w \mid V \text{ accepts } \langle w, c\rangle \text{ for some string } c\}$

  - w riguarda i dati del problema
  - c riguarda le istruzioni della `TM`, un candidato di soluzione o almeno ci é legato in qualche maniera
    * potrebbe essere anche il cammino della macchina non deterministica
    * la *address tape* nella simulazione deterministica di una macchina non deterministica
- si misura il tempo di un verificatore solo in funzione della lunghezza di w
  - un verificatore polinomiale esegue in tempo polinomiale secondo la lunghezza di w

Prova `7.20` Il determinismo con certificato c utilizzando V é convertito in non determinismo trovando il c in maniera non deterministica di lunghezza massima n^k (dove questo é il polinomio di complessitá)

Si dimostra quindi che le due definizioni sono equivalenti in quanto é sempre possibile convertire un V polinomiale in una M polinomiale non deterministica e viceversa.

NP-COMPLETO **definition** Un linguaggio B é NP-completo se soddisfa le seguenti condizioni:

1. $B \in NP$
2. $\forall A \in NP, A <_P B$
   - A si riduce in tempo polinomiale a B

Ci sono quindi due possibilitá che si escludono l'un l'altra:

- $P = NP$
- Tutti i problemi NP-completi non sono polinomiali

La classe NP-completo descrive i problemi piú difficili in NP

TEOREMA DI COOK-LEVIN Problemi in NP la cui complessitá é legata a quella dell'intera classe sono detti NP-completi Il problema della soddisfatibilitá (*satisfiability problem*) fa parte di questa classe

- Una formula booleana é soddisfacibile se qualche assegnamento di 0 e di 1 fa si che la formula risulti 1
- SAT $= \{\langle \phi \mid \phi \rangle$ é una formula booleana soddisfacibile $\}$

`7.27` **theorem** SAT $\in$ P $\iff$ P $=$ NP

Questo teorema é implicato da `7.37`: **theorem** SAT é NP-completo

corollary 3SAT é NP-completo

- CNF-SAT $\leqslant_P$ 3-SAT $\leqslant_P$ CLIQUE

NB - Per provare la NP-completessa si procede da SAT al problema in particolare

HAMILTON'S PATH Percorso che percorre tutti il grafo a partire da p arrivando in t senza ripetizioni. Si percorre il grafo non deterministicamente

- si scartano tutti i rami in cui il primo nodo non é p o t non é l'ultimo
- si scartano i rami in cui ci sono ripetizioni

Non conosciuto algoritmo in P
3SAT $\leqslant_P$ HamPath

COMPOSITENESS Composites $= \{x \mid x = pq$ for integers $p, q > 1\}$ Un numero composto é un numero non primo. Esiste un algoritmo polinomiale per verificare se un numero é composto o meno ma non per trovare la sua scomposizione (o almeno non lo si é trovato) Quindi: Composites $\in$ NP $\wedge$ Composites $\in$ P

CLIQUE 7.32 Grafo non orientato, fornito un k

- si richiede un sottografo in cui 2 qualunque nodi distinti sono connessi di un arco

Non si sa se esistono algoritmi polinomiali P
Clique $= \{\langle G, k\rangle \mid G$ is an undirected graph with a k-clique$\}$
É NP-completo
proof Data ϕ una formula con k clausole del tipo

- $\phi = (a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$

Si definisce la riduzione f per cui Clique $<_P$ 3SAT

- f genera la stringa $\langle G, k\rangle$, dove G é un grafo non orientato
- i nodi di G sono raggruppati in k triplette $t_1, \ldots, t_k$
- gli archi di G connettono tutti i nodi tranne:
  1. nodi della stessa tripletta
  2. due nodi contraddittori, come x_1 e $\overline{x_1}$

Si dimostra che $\phi \in$ 3SAT $\iff$ $G \in$ k-Clique Quindi 3SAT $<_P$ Clique ■

SUBSET-SUM 7.56 Subset-Sum $= \{\langle S, t\rangle \mid S = \{s_1, \ldots, s_n\}$ dove esistono $\{y_1, \ldots, y_m\} \subseteq S$ tali che $\sum y_i = t\}$

Si dimostra facilmente che questo é NP definendone un verificatore polinomiale oppure una `TM` non deterministica polinomiale che lo definisca.

Subset-Sum é NP-completo

La prova procede per riduzione polinomiale da 3SAT a Subset-Sum, convertendo elementi e strutture del problema che rappresentano variabili e clausole booleane.

2.4 Complessitá Spaziale

`8.1` **definition** Data la `TM` M che termina sempre. Si dice *complessitá spaziale* di M la funzione $f : N \to N$, dove $f(n)$ é il massimo numero di celle di nastro che la M passa su un qualsiasi input di lunghezza n

2.4.1 Classi

`8.2` **definition** Data $f : N \to R^+$. Le *classi di complessitá spaziale* $\textsc{space}(f(n))$ e $\textsc{nspace}(f(n))$, sono definiti come:

- $\textsc{space}(f(n)) = \{L \mid L$ é decidibile da una TM deterministica in spazio $O(f(n))\}$
- $\textsc{nspace}(f(n)) = \{L \mid L$ é decidibile da una TM non deterministica in spazio $O(f(n))\}$

definition PSPACE é la classe di linguaggi che sono decidibili in spazio polinomiale da una `TM` deterministica

- $$\textsc{pspace} = \bigcup_k \textsc{space}(n^k)$$

Da `8.5` segue che PSPACE = NPSPACE

In sommario:

- P $\subseteq$ NP $\subseteq$ PSPACE = NPSPACE $\subseteq$ EXPTIME

Questo perché:

- NP $\subseteq$ NPSPACE in quanto una macchina in $f(n)$ passi puó esplorare al massimo $f(n)$ celle di memoria
- PSPACE $\subseteq$ EXPTIME, una machina in PSPACE puó eseguire passi senza ripetersi al massimo
  - $$f(n) \cdot 2^{O(f(n))} = \bigcup_k \textsc{time}(2^{n^k})$$

  , dopo di che é in loop

../media/img/complexity-classes.jpg

Qualsiasi di queste inclusioni potrebbero essere eguaglianze, ma non sono state trovate prove a riguardo.

Inoltre si definiscono le classi sottolineari:

$$L = \bigcup_k \textsc{space}(\log n)$$

$$NL = \bigcup_k \textsc{nspace}(\log n)$$

E si dimostra: $\textsc{l} \subseteq \textsc{nl} \subseteq \textsc{p} \subseteq \textsc{np} \subseteq \textsc{pspace} = \textsc{npspace} \subseteq \textsc{exptime}$

2.4.2 Teorema di Savitch

8.5 Per qualsiasi funzione $f : N \to R^+$, dove $f(n) \geqslant n$,

- $\textsc{nspace}(f(n)) \subseteq \textsc{space}(f^2(n))$

Il passaggio da non determinismo a determinismo per il tempo é piú impegnativo che per lo spazio, lo spazio é piú potente in quanto puó essere riutilizzato, al contrario del tempo.

- l'equivalente deterministico di una macchina non deterministica polinomiale ha:
  - Tempo $2^{O(n^k)}$

  - Spazio $O(n^2)$

Da questo teorema segue che PSPACE $=$ NPSPACE in quanto il quadrato di un polinomiale é ancora polinomiale.

2.4.3 GG

Gioco Generalizzato della Geografia

- il gioco consiste nel spostarsi in un grafo i cui nodi sono nomi di cittá
- gli archi vanno da un cittá il cui nome finisce con una certa lettere a un nodo/cittá che inizia per data lettera
- ci sono due giocatori che partono da una data cittá
- a turno scelgono un arco da percorrere, perde chi non puó scegliere un arco entrante in un nodo giá visitato

Si dimostra che GG é PSPACE definendo una funzione ricorsiva detta di Von Neumann VonN(a, X, g) una volta fissato il grafo G

- vero se esiste una strategia vincente a partire da a per il giocatore g, che porta quindi ad una configurazione in cui non esiste una mossa b per il giocatore $\neg g$ che non violi le regole

Altro risultato della teoria é che GG é PSPACE-completo, quindi se si scoprisse un algoritmo in tempo polinomiale che risolva GG questo dimostrerebbe che:

- P = NP = PSPACE = NPSPACE.

In quanto per il teorema di `Savitch` NPSPACE = PSPACE.
Questa ipotesi é ritenuta improbabile, anche se non si puó escludere.