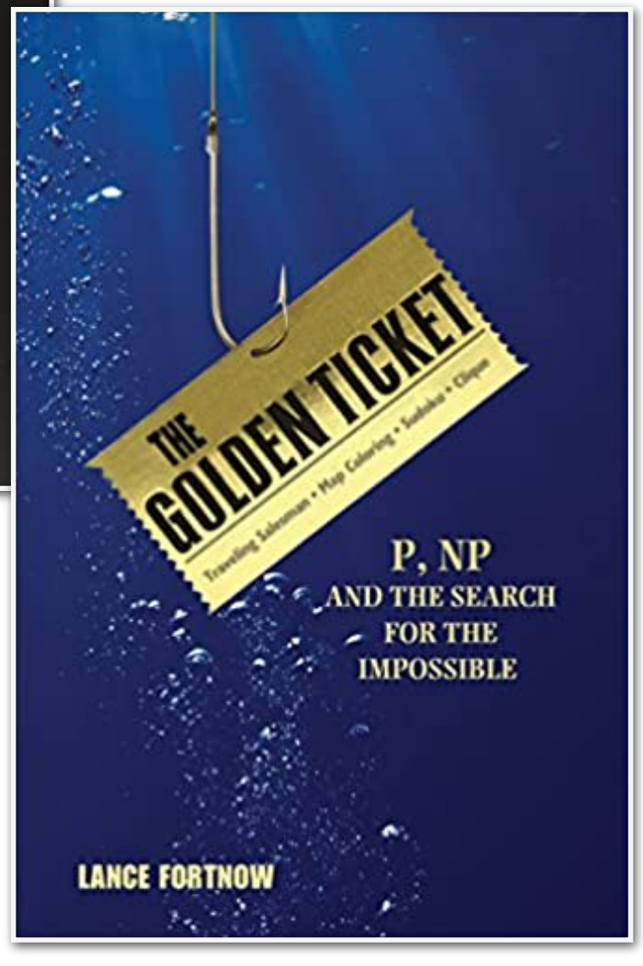
Computabilità, Complessità e Logica

Lezione 11

Libri

Logicomix
Apostolos Doxiadis e Christos Papadimitriou

Storia (a fumetti) dalla crisi dei fondamenti della matematica (fine ‘800), ai problemi di Hilbert fino ad arrivare a Turing e Gödel

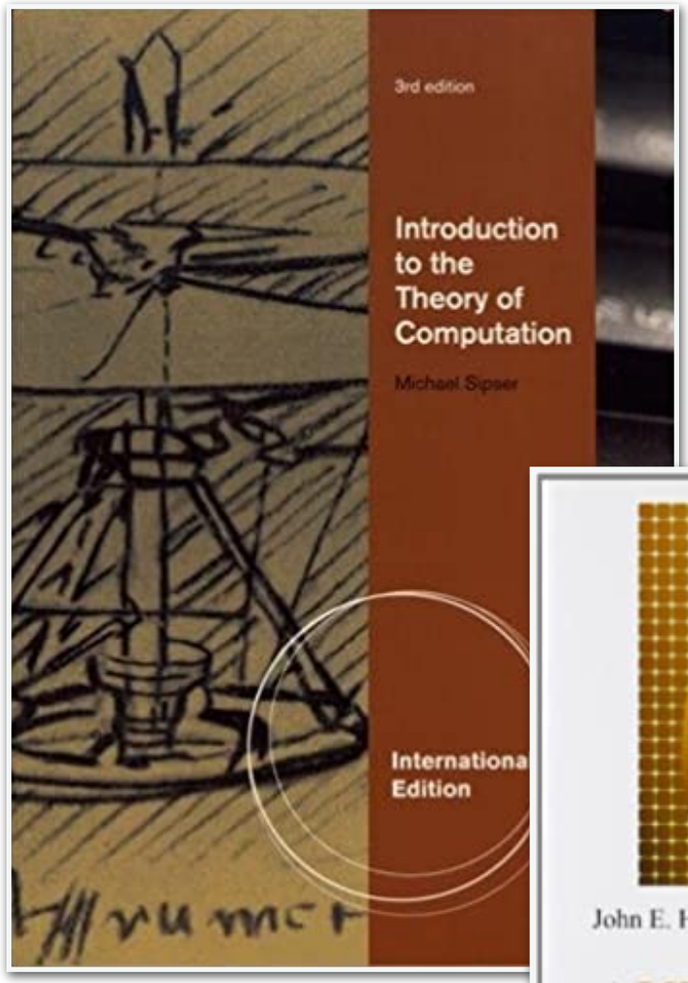


The Golden Ticket:
P, NP, and the Search for the Impossible
Lance Fortnow

Testo divulgativo che esplora la questione P=NP, le implicazioni che questa ha nella vita reale.

Si consiglia anche il blog di Richard J Lipton:
Gödel’s lost letter and P=NP
https://rjlipton.wpcomstaging.com

Libri



Introduction to the Theory of Computation
Michael Sipser

Testo classico su automi, linguaggi e computabilità. Molti degli esercizi del tutorato sono presi da questo testo.

Automi, linguaggi e calcolabilità
John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman

Testo che si focalizza sulle questioni di automi e linguaggi partendo dai linguaggi regolari per andare ai linguaggi context free (che non abbiamo visto, una classe intermedia), context dependant (altra classe Intermedia) fino ad arrivare a macchine di Turing e questioni di decidibilità

Classi oltre PSPACE

- Vi sono infinite classi oltre PSPACE mettendo restrizioni diverse su tempo e spazio
- Mettendo una restrizione esponenziale sul tempo otteniamo due classi:

$$\mathrm{EXPTIME} = \bigcup_{k \in \mathbb{N}} \mathrm{TIME}\left(2^{n^k}\right)$$

linguaggi decisi da macchine di Turing deterministiche in tempo esponenziale

$$\mathrm{NEXPTIME} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}\left(2^{n^k}\right)$$

Linguaggi decisi da macchine di Turing non deterministiche in tempo esponenziale

Classi oltre PSPACE

- Mettendo invece una restrizione esponenziale sullo spazio otteniamo la classe:

$$\mathrm{EXPSPACE} = \bigcup_{k \in \mathbb{N}} \mathrm{SPACE}\left(2^{n^k}\right)$$

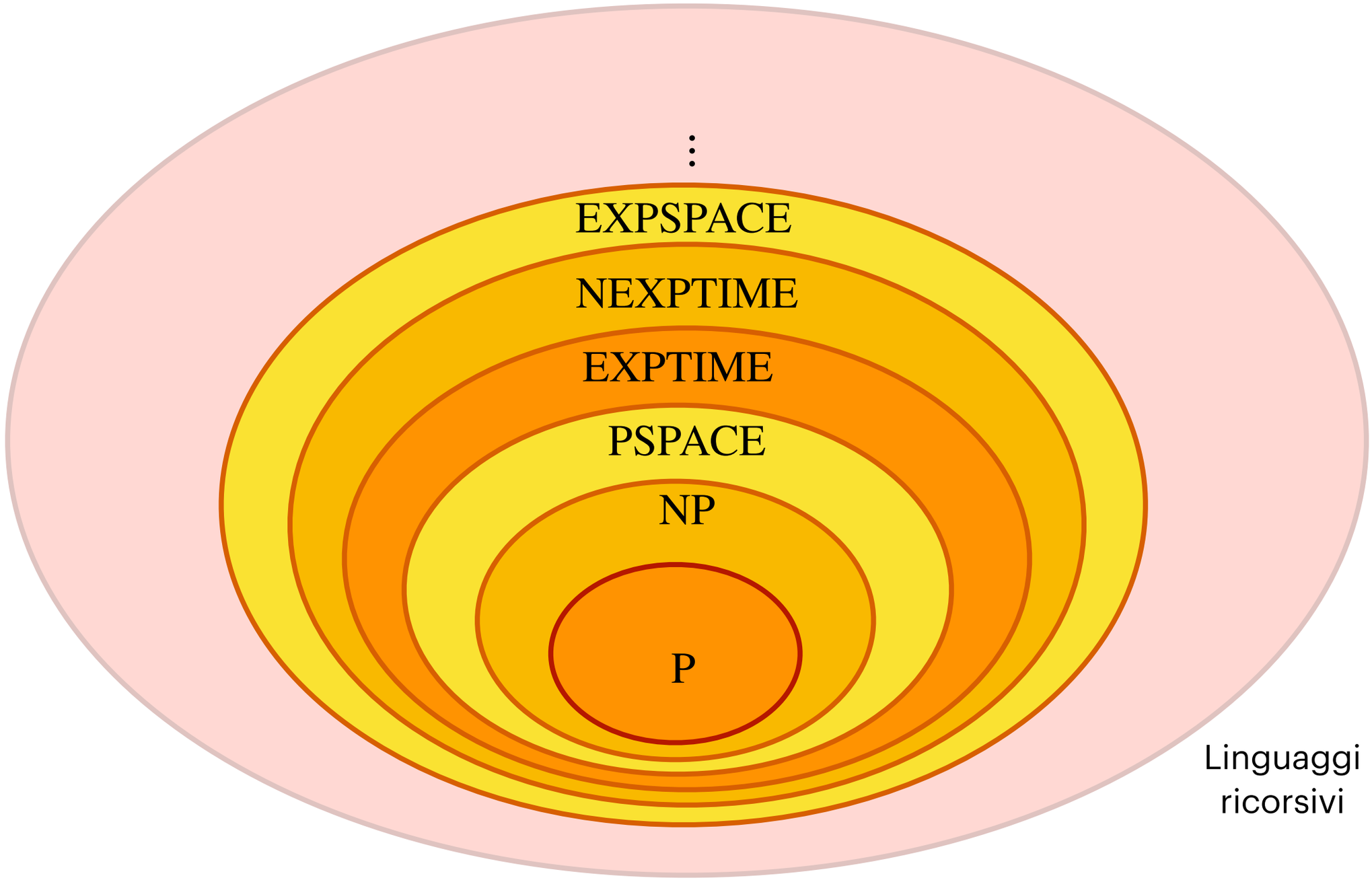
  linguaggi decisi da macchine di Turing deterministiche in spazio esponenziale

- Esiste una gerarchia infinita di classi. Per esempio possiamo definire per $m \in \mathbb{N}$

$$m - \mathrm{EXPTIME} = \bigcup_{k \in \mathbb{N}} \mathrm{TIME}\left(2^{2^{\vdots^{2^{n^k}}}} \right\} \text{altezza } m\Bigg)$$

Classi di complessità

Diagramma aggiornato



Riduzioni (ancora)

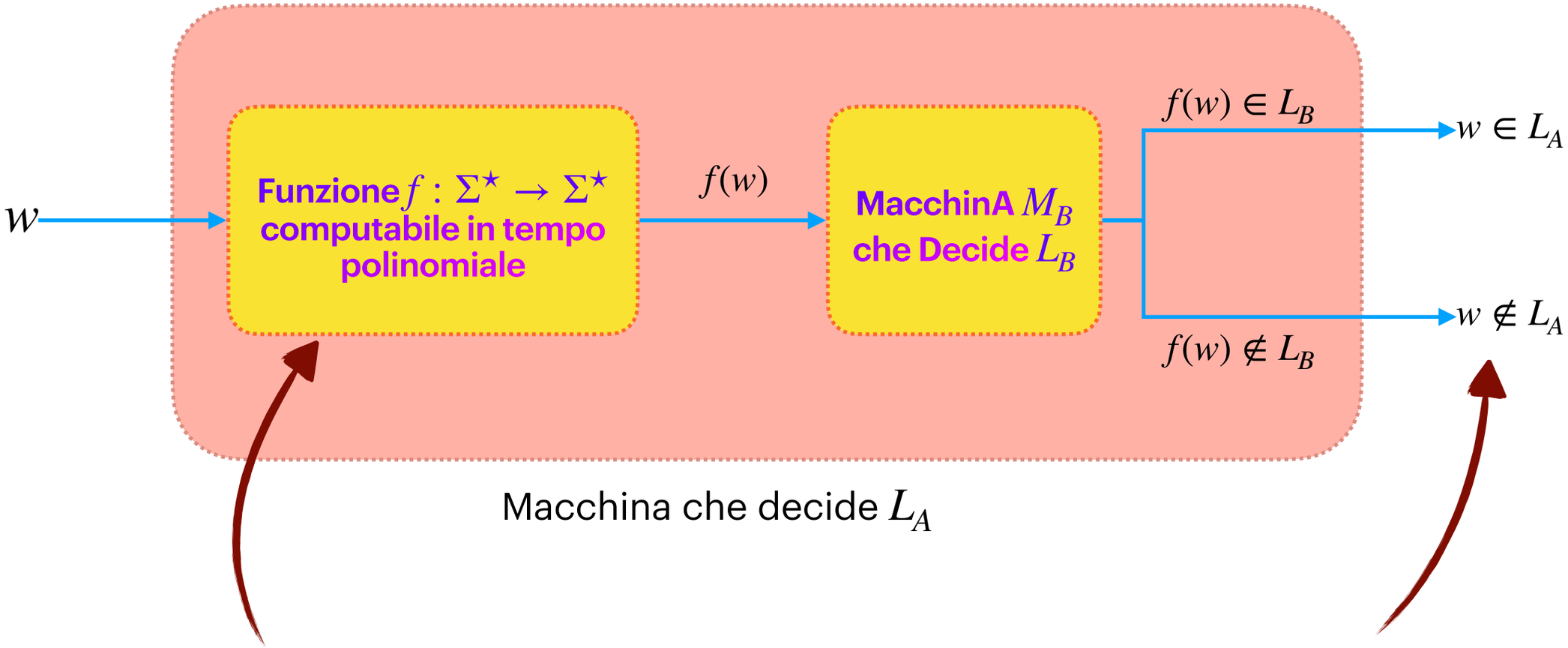
- Abbiamo visto come possiamo fare una riduzione tra linguaggi usando una funzione computabile
- Se mettiamo restrizioni aggiuntive sulla funzione computabile possiamo ottenere altre tipologie di riduzione
- In particolare ci interesseranno le riduzioni **in tempo polinomiale**.
- Rispetto alle riduzioni già viste in questo caso non solo la funzione deve essere computabile, ma deve esserlo in modo efficiente da una MdT deterministica

Riduzioni in tempo polinomiale

- Diciamo che $f : \Sigma^\star \to \Sigma^\star$ è una **riduzione in tempo polinomiale** di un linguaggio L_A a un linguaggio L_B se:
  - f è una funzione computabile in tempo polinomiale
  - Per ogni $w \in \Sigma^\star$ abbiamo $w \in L_A \iff f(w) \in L_B$
- Se esiste tale funzione f diciamo che L_A si riduce a L_B che indicheremo come $L_A \leq_P L_B$

Riduzioni in tempo polinomiale

Una rappresentazione grafica di come $L_A \leq_P L_B$ (L_A si riduce in tempo polinomiale a L_B)



Problemi completi: intuizione

- Vogliamo individuare i problemi di decisione (linguaggi) che sono "i più difficili" per una certa classe $\mathscr{C}$ di linguaggi
- Intuitivamente questo significa che se siamo in grado di risolvere un problema completo per $\mathscr{C}$ allora siamo in grado di risolvere tutti i problemi in $\mathscr{C}$
- Un modo per stabilire come decidere un linguaggio usando una MdT che ne decide un altro è vedere se esiste una riduzione in tempo polinomiale

Problemi completi

- Con la nozione di riduzione possiamo cercare linguaggi che sono completi per una classe di linguaggi $\mathscr{C}$
- Un linguaggio L è **completo** per $\mathscr{C}$ se rispetta queste due condizioni:
  - $L \in \mathscr{C}$
  - Per tutti gli $L' \in \mathscr{C}$ abbiamo $L' \leq_P L$

Problemi completi

- La seconda condizione (ovvero che tutti i linguaggi di $\mathscr{C}$ sono riducibili a L) ci dice che L è **hard** o **difficile** per $\mathscr{C}$ (indicato anche con $\mathscr{C}$-hard o $\mathscr{C}$-difficile).

- Se L è $\mathscr{C}$-hard possiamo usarlo per risolvere tutti i problemi in $\mathscr{C}$, la completezza si ha quando L è $\mathscr{C}$-difficile e $L \in \mathscr{C}$

- Non è assicurato che ogni classe $\mathscr{C}$ abbia problemi completi (e alcune classi non ne hanno)

- Tutti i problemi $\mathscr{C}$-completi sono riducibili tra di loro (in un certo senso sono “ugualmente difficili”)

Problemi NP-completi

- In particolare se scegliamo $\mathscr{C} =$ NP ci possiamo chiedere:
  - Esistono problemi NP-completi?
  - Quali sono dei problemi NP-completi?
- Le risposte a queste domande sono:
  - Sì, NP ammette problemi/linguaggi completi
  - Centinaia di problemi, molto che hanno anche applicazioni reali, sono NP-completi

Problemi NP-completi

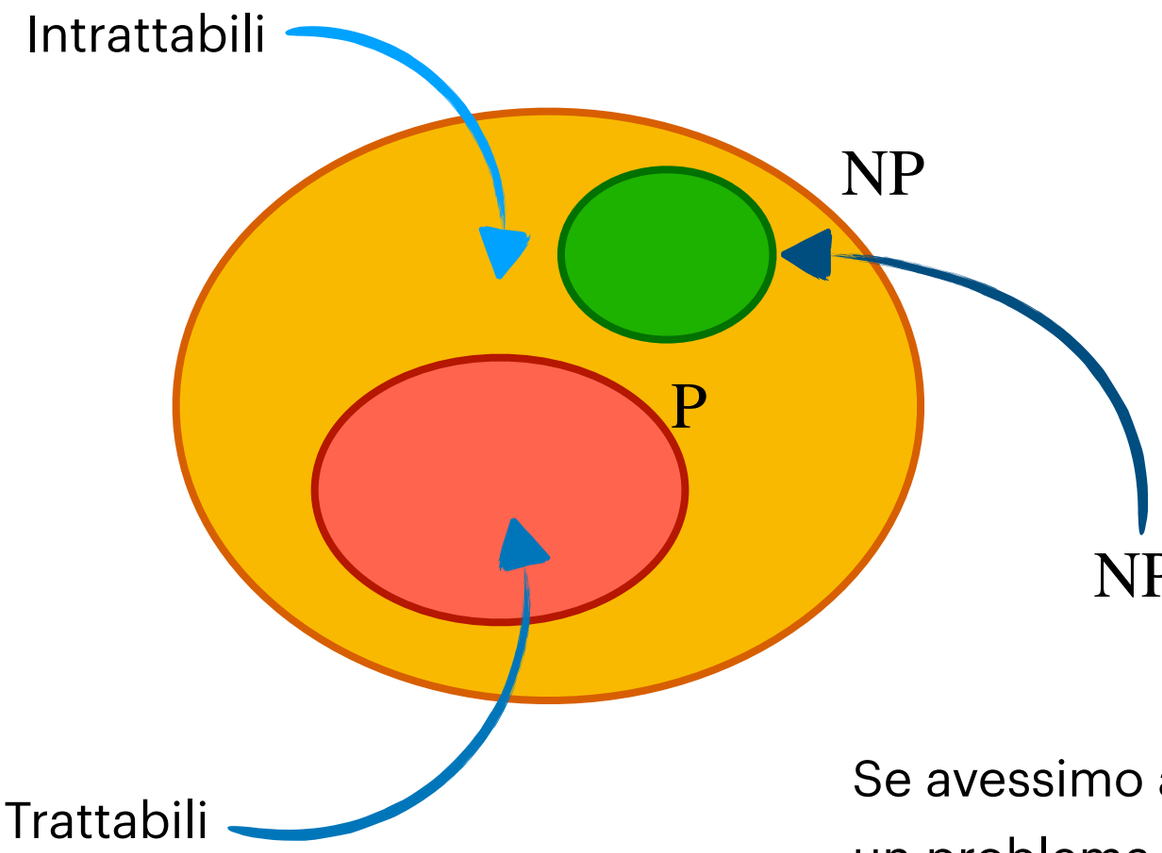
- Percorso hamiltoniano: dato un grafo e due vertici p e q, esiste un percorso da p a q che passa per tutti i vertici esattamente una volta?
- Subset-sum: dato un insieme di numeri S e un numero target t esiste un sottoinsieme di S' di S per cui la somma di tutti i numeri in S' è esattamente t?
- Quadrati latini: una "generalizzazione" del sudoku in cui ci controllano solo le condizioni di unicità su righe e colonne
- *"It is NP-complete to decide whether a given target location is reachable from a given start location in generalized Pokémon in which the only overworld game elements are enemy Trainers. "*
  Dimostrato nel 2015 da Greg Aloupis, Erik D. Demaine, Alan Guo, Giovanni Viglietta in
  "*Classic Nintendo Games are (Computationally) Hard*"

Problemi NP-completi

- Perché sono importanti i problemi NP-completi?
- Ogni problema NP-completo permette di risolvere ogni altro problema in NP applicando prima una riduzione polinomiale
- Se fossimo in grado di decidere un linguaggio NP-completo L in tempo polinomiale deterministico (ovvero $L \in \mathrm{P}$)...
- ...allora **ogni** linguaggio in NP avrebbe un algoritmo per risolverlo in tempo polinomiale deterministico (ovvero $\mathrm{P} = \mathrm{NP}$)

Classi di complessità

E problemi NP-completi



Se avessimo anche solo
un problema NP-completo in P
allora avremmo $\mathrm{P} = \mathrm{NP}$

Se $\mathrm{P} \neq \mathrm{NP}$ allora esistono nei problemi fuori P ma non NP-completi
(problemi NP-intermedi). Questo è il teorema di Ladner (1975)

Un primo problema NP-completo

È possibile definire un problema NP-completo "banale" simulando una macchina non deterministica che si arresta in tempo polinomiale

$L = \{\langle M, w, 1^k \rangle :$ la MdT non deterministica M su input w accetta in al più k passi$\}$

Scriviamo in unario (come sequenza di 1) il numero di passi che la macchina richiede per arrestarsi.

Dobbiamo dimostrare due cose:

1. $L \in \mathrm{NP}$

2. Per ogni $L' \in \mathrm{NP}$ abbiamo $L' \leq_P L$

Un primo problema NP-completo

Mostriamo che $L \in \mathrm{NP}$

$L = \{\langle M, w, 1^k \rangle :$ la MdT non deterministica M su input w accetta in al più k passi$\}$

Dato un input $\langle M, w, 1^k \rangle$ possiamo simulare M su input w per k passi (facendo scelte non deterministiche quando M le farebbe), accettando se M accetta dopo aver simulato al più k passi e rifiutando altrimenti.

Questa simulazione si può fare in tempo polinomiale rispetto a M, w e k usando una MdT non deterministica.

Quindi $L \in \mathrm{NP}$

Un primo problema NP-completo

Mostriamo che $L' \in \mathrm{NP}$ implica $L' \leq_P L$

Sia $L' \in \mathrm{NP}$. Quindi:

Esiste una MdT non deterministica M

Esiste una funzione $t(n)$ limitata da un polinomio $p(n)$ (i.e., per ogni n si ha $t(n) \leq p(n)$)

M su input w si arresta entro al più $t(n)$ passi

Un primo problema NP-completo

Data una parola w con $|w| = n$ costruiamo $\langle M, w, 1^{p(n)} \rangle$

M ha dimensione costante (è la descrizione della MdT, quindi fissata)

w può essere ottenuto copiando l'input

Si ha $1^{p(n)}$ che è scrivibile in tempo polinomiale (per assunzione che $L' \in \mathrm{NP}$

Quindi $f(w) = \langle M, w, 1^{p(n)} \rangle$ è computabile in tempo polinomiale

Un primo problema NP-completo

È quindi vero che $w \in L' \iff \langle M, w, 1^{p(n)} \rangle \in L$?

Se $w \in L'$ allora M su input w ha una computazione accettante e ogni computazione di M richiede al più $t(n) \leq p(n)$ passi. Quindi $f(w) \in L$

Se $w \notin L'$ allora ogni computazione di M su input w rifiuta e, in particolare, rifiuta in al più $t(n) \leq p(n)$ passi. Quindi $f(w) \notin L$

Quindi L è un linguaggio NP-completo