

# Informatica Teorica



Anno Accademico 2022/2023  
Fabio Zanasi  
<https://www.unibo.it/sitoweb/fabio.zanasi>  
Tredicesima lezione

# Nelle puntate precedenti

- Complessità di tempo: le classi P e NP.
- NP: completezza: il teorema di Cook-Levin.
- Poly riduzioni tra problemi.

# In questa lezione

## Parte I

- Complessità di spazio: PSPACE e NPSPACE.
- Un problema PSPACE-completo: TQBF.
- PSPACE = NPSPACE: Il teorema di Savitch.

# In questa lezione

## Parte II

- Gerarchie di complessità di tempo/spazio.
- Esistenza di problemi intrattabili.
- I limiti della diagonalizzazione.

# Complessità di Spazio

# Complessità di spazio

Sia  $M$  una TM che si ferma su tutti gli input. La sua **complessità di spazio** è definita come la funzione  $t : \mathbb{N} \rightarrow \mathbb{N}$ , dove  $t(n)$  è il massimo numero di celle del nastro che la testina di  $M$  visita su arbitrari input di lunghezza  $n$ .

# Complessità di spazio

Sia  $M$  una TM che si ferma su tutti gli input. La sua **complessità di spazio** è definita come la funzione  $t : \mathbb{N} \rightarrow \mathbb{N}$ , dove  $t(n)$  è il massimo numero di celle del nastro che la testina di  $M$  visita su arbitrari input di lunghezza  $n$ .

Data una funzione  $t : \mathbb{N} \rightarrow \mathbb{N}$ , definiamo **la classe di complessità** (di spazio) **SPACE( $t(n)$ )** come la collezione di tutti i linguaggi decidibili da una TM (deterministica, a un nastro) in spazio  $O(t(n))$ .

# Complessità di spazio

Sia  $M$  una TM che si ferma su tutti gli input. La sua **complessità di spazio** è definita come la funzione  $t : \mathbb{N} \rightarrow \mathbb{N}$ , dove  $t(n)$  è il massimo numero di celle del nastro che la testina di  $M$  visita su arbitrari input di lunghezza  $n$ .

Data una funzione  $t : \mathbb{N} \rightarrow \mathbb{N}$ , definiamo **la classe di complessità** (di spazio) **SPACE( $t(n)$ )** come la collezione di tutti i linguaggi decidibili da una TM (deterministica, a un nastro) in spazio  $O(t(n))$ .

In modo analogo, considerare le TM non-deterministiche ci dà una classe **NSPACE( $t(n)$ )**.

# PSPACE e NPSPACE

PSPACE è la classe dei linguaggi decidibili da una TM (deterministica, a un nastro) in spazio **polinomiale**. Ovvero:

$$PSPACE = \bigcup_k SPACE(n^k)$$

Analogamente, abbiamo:

$$NPSPACE = \bigcup_k NSPACE(n^k)$$

# (N)P e PSPACE

Il fatto che P sia inclusa in PSPACE è ovvio per definizione.

È meno ovvio (ma vero) che NP sia inclusa in PSPACE.

Possiamo dimostrarlo come corollario del seguente lemma:

**Lemma** SAT è in PSPACE.

# (N)P e PSPACE

**Lemma** SAT é in PSPACE.

**Dimostrazione** Considera la TM M definita nel seguente modo.

Su input  $\langle F \rangle$ , dove F é una formula booleana:

[1] Per ogni assegnamento A di valore alle variabili di F:

[2] Valuta F rispetto a A. Se il valore di F é 1 (vero), accetta, se no rigetta.

Ogni iterazione di [1] può essere eseguita in spazio lineare rispetto a  $\langle F \rangle$ , dal momento che l'assegnamento avrà lunghezza  $O(m)$  dove m é il numero di variabili in F.

Nota che M nella sua interezza lavora in spazio  $O(m)$ : infatti possiamo eseguire ogni iterazione di [1] sulla stessa porzione di nastro, una volta cancellato il risultato dell'iterazione precedente.

# (N)P e PSPACE

**Corollario** NP é inclusa in PSPACE.

## Idea della dimostrazione

Sia  $L$  in NP. Per il teorema di Cook-Levin,  $L \leq_p SAT$ . Per il Lemma appena dimostrato, abbiamo anche che SAT é in PSPACE. Unendo questi due fatti, possiamo facilmente dimostrare che  $L$  é in PSPACE.

# Un problema PSPACE completo

## Definizione

Un linguaggio  $L$  è PSPACE-completo se è in PSPACE e ogni altro linguaggio  $L'$  in PSPACE è poly-riducibile ad esso ( $L' \leq_p L$ ).

# Un problema PSPACE completo

Ricorda la definizione di formula booleana.

Estendiamo la definizione permettendo l'uso di quantificatori, a vincolare le variabili della formula.

Esempio:  $\forall x \exists y (\bar{x} \vee y) \wedge (x \vee z)$

Una formula è un enunciato se tutte le sue variabili sono vincolate da un quantificatore.

Esempio:  $\forall x \exists y \forall z (\bar{x} \vee y) \wedge (x \vee z)$

Nota che la verità di un enunciato non dipende dalla scelta di assegnamento di valore di verità alle sue variabili. Se un enunciato è vero per *un* assegnamento, allora è vero per *qualsiasi* assegnamento.

# Un problema PSPACE completo

**TQBF = { <F> | F è un enunciato booleano vero. }**

# Un problema PSPACE completo

**Teorema** TQBF é PSPACE-completo.

## Idea della dimostrazione

É facile dimostrare che TQBF é in PSPACE. Dobbiamo poi dimostrare che qualsiasi L in PSPACE é poly riducibile a TQBF.

Imitare la dimostrazione data per SAT non funziona: il numero di righe del tableau non é più vincolato dal tempo polinomiale, perciò se lo rappresentassimo con una formula questa potrebbe avere lunghezza più che polinomiale.

Quello che faremo é dividere il tableau in quadranti e usare i quantificatori per rappresentare diversi quadranti con la stessa sottoformula: il risultato sarà una formula di lunghezza polinomiale.

# Un problema PSPACE completo

## Dimostrazione

Verifichiamo prima che TQBF sia in PSPACE. Ecco un algoritmo ALG che lo decide:

Su input  $\langle F \rangle$ , dove  $F$  è un enunciato booleano:

- [1] Se  $F$  non contiene quantificatori, allora contiene solo costanti (nessuna variabile). Valutiamola e accettiamo se è solo se è vera.
- [2] Se  $F = \exists x G$ , chiama ALG ricorsivamente su  $G$ , prima valutando  $x = 1$ , poi valutando  $x = 0$ . Se almeno una computazione è accettante, accetta. Altrimenti, rigetta.
- [3] Se  $F = \forall x G$ , chiama ALG ricorsivamente su  $G$ , prima valutando  $x = 1$ , poi valutando  $x = 0$ . Se entrambe le computazioni sono accettanti, accetta. Altrimenti rigetta.

# Un problema PSPACE completo

## Dimostrazione

Su input  $\langle F \rangle$ , dove  $F$  é un enunciato booleano:

- [1] Se  $F$  non contiene quantificatori, allora contiene solo costanti (nessuna variabile). Valutiamola e accettiamo se e solo se é vera.
- [2] Se  $F = \exists xG$ , chiama ALG ricorsivamente su  $G$ , prima valutando  $x = 1$ , poi valutando  $x = 0$ . Se almeno una computazione é accettante, accetta. Altrimenti, rigetta.
- [3] Se  $F = \forall xG$ , chiama ALG ricorsivamente su  $G$ , prima valutando  $x = 1$ , poi valutando  $x = 0$ . Se entrambe le computazioni sono accettanti, accetta. Altrimenti rigetta.

Quest'algoritmo decide TQBF. Inoltre, il numero di chiamate ricorsive é al più il numero di variabili in  $F$ . Ogni chiamata ha bisogno di memorizzare solo il valore di una variabile, perciò lo spazio utilizzato é  $O(m)$ , dove  $m$  é il numero di variabili in  $F$ . In conclusione, TQBF é in PSPACE.

# Un problema PSPACE completo

## Dimostrazione

Veniamo alla seconda parte della dimostrazione. Sia  $L$  un linguaggio in PSPACE. Vogliamo dimostrare che  $L \leq_p \text{TQBF}$ .

Chiamiamo  $M$  la TM che decide  $L$  in spazio polinomiale, diciamo  $n^k$  per qualche costante  $k$ . Per dimostrare  $L \leq_p \text{TQBF}$ , è sufficiente costruire in tempo polinomiale una formula  $F_w$  tale che

$M$  accetta  $w$       se e solo se

$F_w$  è un enunciato booleano vero

# Un problema PSPACE completo

## Dimostrazione

M accetta w

se e solo se

$F_w$  é un enunciato booleano vero

Per costruire tale  $F_w$ , costruiamo prima un tipo di formula  $F_{c,c',t}$  più generale, con la seguente proprietà.

M può andare dalla configurazione  $c$  alla configurazione  $c'$  in al più  $t$  passi.

se e solo se

$F_{c,c',t}$  é un enunciato booleano vero

Basterà poi definire  $F_w$  come  $F_{c_{init}, c_{acc}, t}$ , dove  $c_{init}$  é la configurazione iniziale,  $c_{acc}$  codifica una qualsiasi configurazione accettante, e  $t = 2^{dn^k}$  per una costante  $d$  tale che M non ha più di  $t$  configurazioni possibili su input di lunghezza  $n$ .

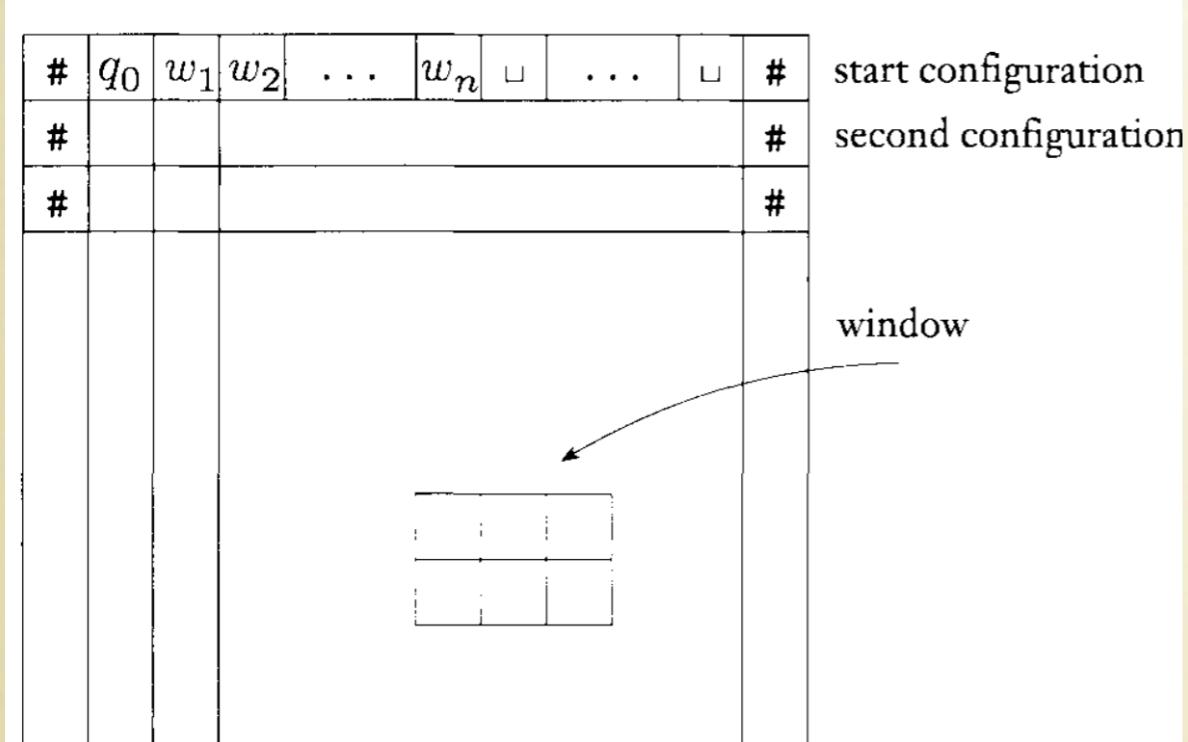
# Un problema PSPACE completo

## Dimostrazione

M può andare dalla configurazione  $c$  alla configurazione  $c'$  in al più  $t$  passi.

se e solo se

$F_{c,c',t}$  è un enunciato booleano vero



Ricorda l'insieme di variabili  $\{x_{i,j,s} \mid (i,j) \in n^k \times n^k \text{ e } s \in Q \cup \Sigma \cup \{\#\}\}$  introdotto nel teorema di Cook-Levin per descrivere i possibili valori di una cella  $cell[i,j]$  nel tableau di M su w.

In  $F_{c,c',t}$ , c e c' saranno insiemi di tali variabili. Ciascuno codifica una configurazione (riga del tableau).

Per esempio,  $c_{init}$  è  $\{x_{1,1,\#}, x_{1,2,w_1}, \dots, x_{1,n+1,w_n}, x_{1,n+2,\vdash}, \dots, x_{1,n^k-1,\vdash}, x_{1,n^k,\#}\}$

# Un problema PSPACE completo

## Dimostrazione

M può andare dalla configurazione  $c$  alla configurazione  $c'$  in al più  $t$  passi.

se e solo se

$F_{c,c',t}$  è un enunciato booleano vero

Costruiamo  $F_{c,c',t}$ , per induzione su  $t$ . Se  $t=1$ , abbiamo solo due casi:

- la computazione ha 0 passi. Costruiamo un enunciato  $F_1$  che esprime (nel modo visto nel teorema di Cook-Levin) che  $cell[i,j]$  in  $c$  ha lo stesso valore di  $cell[i,j]$  in  $c'$ .
- la computazione ha 1 passo. Costruiamo un enunciato  $F_2$  che esprime che la configurazione  $c'$  è raggiungibile in un passo dalla configurazione  $c$  (nel modo visto nel teorema di Cook-Levin, tramite finestre).

$F_{c,c',1}$  sarà definita come  $F_1 \vee F_2$ .

# Un problema PSPACE completo

## Dimostrazione

M può andare dalla configurazione  $c$  alla configurazione  $c'$  in al più  $t$  passi.

se e solo se

$F_{c,c',t}$  é un enunciato booleano vero

Se  $t > 1$ , un'idea potrebbe essere di definire  $F_{c,c',t}$  come

$$\exists m_1 (F_{c,m_1,\frac{t}{2}} \wedge F_{m_1,c',\frac{t}{2}})$$

Qui  $\exists m_1$  abbrevia  $\exists x_1, \dots, x_l$ , dove  $x_1, \dots, x_l$  sono le variabili che codificano la configurazione  $m_1$ . Le sottoformule  $F_{c,m_1,\frac{t}{2}}$  e  $F_{m_1,c',\frac{t}{2}}$  sono definite per ipotesi induttiva.

Il significato é quello giusto, ma la formula risultante ha lunghezza esponenziale in  $n$ . Infatti, ogni passo della costruzione induttiva di  $F_{c,c',t}$  dimezza  $t$ , ma raddoppia la lunghezza della formula. Per cui nel caso che ci interessa  $t = 2^{dn^k}$ , la lunghezza di  $F_{c,c',t}$  sarà esponenziale in  $n$ .

# Un problema PSPACE completo

## Dimostrazione

M può andare dalla configurazione  $c$  alla configurazione  $c'$  in al più  $t$  passi.

se e solo se

$F_{c,c',t}$  é un enunciato booleano vero

Con uso sapiente dei quantificatori otteniamo una formula della lunghezza giusta:

$$F_{c,c',t} := \exists m_1 \forall (c_3, c_4) \in \{(c, m_1), (m_1, c')\} F_{c_3, c_4, \frac{t}{2}}$$

Qui  $\forall x \in \{y, z\}$  é notazione per  $\forall x ((x = y \vee x = z) \rightarrow \dots)$ .

Intuitivamente, la formula dice che le variabili in  $c_3$  e  $c_4$  possono avere lo stesso valore che le variabili in  $c$  e  $m_1$ , oppure lo stesso che le variabili in  $m_1$  e  $c'$ .

La nuova definizione ha lo stesso significato della precedente, ma ogni passo induttivo aggiunge una parte di lunghezza  $O(n^k)$  (la quantificazione su  $c_3, c_4$ ), e vi sono  $\log(2^{dn^k}) = O(n^k)$  passi induttivi. La sua lunghezza é dunque polinomiale in  $n$ .

# Un problema PSPACE completo

## Dimostrazione

M può andare dalla configurazione  $c$  alla configurazione  $c'$  in al più  $t$  passi.

se e solo se

$F_{c,c',t}$  é un enunciato booleano vero

Abbiamo così concluso la definizione di  $F_{c,c',t}$  per induzione su  $t$ .

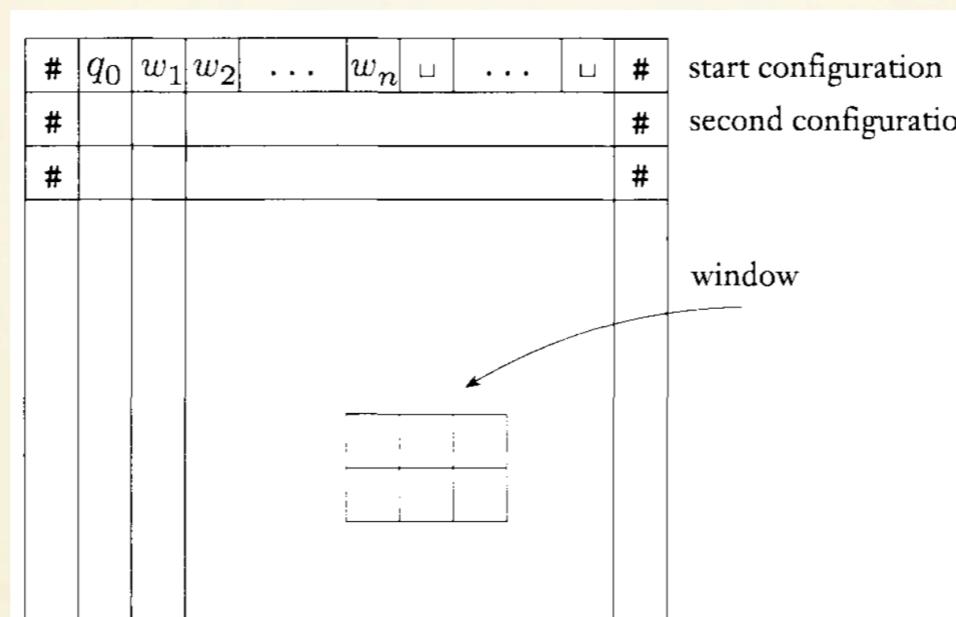
Come anticipato, per concludere la dimostrazione basta definire  $F_w$ , come  $F_{c_{init}, c_{acc}, t}$ , dove  $c_{init}$  é la configurazione iniziale,  $c_{acc}$  codifica una qualsiasi configurazione accettante, e  $t = 2^{dn^k}$  per una costante  $d$  tale che M non ha più di  $t$  configurazioni possibili su input di lunghezza  $n$ .

M accetta  $w$

se e solo se

$F_w$  é un enunciato booleano vero

# Riflessioni sul teorema



A differenza della dimostrazione del teorema di Cook-Levin, non potevamo assumere che il tableau avesse al più  $n^k$  righe, ma solo che avesse al più  $n^k$  colonne, in quanto il linguaggio considerato era in PSPACE, non in P.

Abbiamo compensato con un linguaggio logico più espressivo: tramite i quantificatori, siamo ugualmente riusciti a tradurre la computazione in una formula booleana.

Inoltre, la codifica coi quantificatori può essere resa efficiente (richiedere spazio polinomiale).

# Riflessioni sul teorema



- Il fatto che TQBF sia PSPACE-completo ci da un'indicazione di che tipo di problemi siano in PSPACE.
- Il problema di trovare una strategia vincente nei giochi posizionali (scacchi, dama, Go, ...) può essere espresso mediante un'alternanza di quantificatori: per ogni mossa dell'avversario, esiste una mia mossa tale che etc.
- Non è dunque sorprendente che questa tipologia di problemi è solitamente dimostrabile essere in PSPACE attraverso la costruzione di una Poly riduzione a TQBF.
- Molti problemi in robotica possono essere formulati come giochi, dove l'avversario è l'ambiente in cui si muove il robot.

# Il teorema di Savitch

**Teorema (Savitch, 1970)**

Per qualsiasi funzione  $t : \mathbb{N} \rightarrow \mathbb{N}$ , abbiamo:

$$NSPACE(t(n)) \subseteq SPACE(t^2(n))$$

# Il teorema di Savitch

**Teorema (Savitch, 1970)**

Per qualsiasi funzione  $t : \mathbb{N} \rightarrow \mathbb{N}$ , abbiamo:

$$NSPACE(t(n)) \subseteq SPACE(t^2(n))$$

**Corollario**  $NSPACE = PSPACE$



# Il teorema di Savitch

$$NSPACE(t(n)) \subseteq SPACE(t^2(n))$$

## Dimostrazione

Supponi che  $M$  sia una TM non-deterministica che decida un linguaggio  $L$  in spazio  $t(n)$  rispetto alla lunghezza  $n$  dell'input.

Vogliamo costruire una TM deterministica  $M'$  che decide  $L$  in spazio  $t^2(n)$ .

# Il teorema di Savitch

$$NSPACE(t(n)) \subseteq SPACE(t^2(n))$$

## Dimostrazione

La costruzione di  $M'$  usa la stessa idea utilizzata per dimostrare che TQBF è PSPACE-completo.

Definiremo una procedura REACH che lavora in spazio  $O(t^2(n))$ , tale che

$M$  può andare dalla configurazione  $c$  alla configurazione  $c'$  in al più  $t$  passi.

se e solo se

REACH( $c, c', t$ ) dà output ACCETTA.

Definiremo poi  $M'$  come la TM che su input di lunghezza  $n$  esegue REACH( $c_{init}, c_{acc}, 2^{dt(n)}$ ) e accetta se l'output è ACCETTA.

# Il teorema di Savitch

$NSPACE(t(n))$

scegli  $d$  tale che  $M$  non ha più di  $2^{dt(n)}$  configurazioni possibili su input di lunghezza  $n$ .

## Dimostrazione

La costruzione di  $M'$  usa la stessa idea utilizzata per dimostrare che TQBF è PSPACE-completo.

Definiremo una procedura REACH che lavora in spazio  $O(t^2(n))$ , tale che

$M$  può andare dalla configurazione  $c$  alla configurazione  $c'$  in al più  $t$  passi.

se e solo se

REACH( $c, c', t$ ) dà output ACCETTA.

Definiremo poi  $M'$  come la TM che su input di lunghezza  $n$  esegue REACH( $c_{init}, c_{acc}, 2^{dt(n)}$ ) e accetta se l'output è ACCETTA.

# Il teorema di Savitch

## Dimostrazione

REACH è definita nel modo seguente. Su input  $c, c', T$ :

- [1] Se  $T=1$ , verifica se  $c=c'$  o se la configurazione  $c'$  è raggiungibile da  $c$  in un passo di computazione di  $M$ . ACCETTA se almeno uno dei due test ha successo, se no RIGETTA.
- [2] Se  $T>1$ , per ogni configurazione  $c_m$  di  $M$  che usa spazio  $t(n)$ :
  - [3] Esegui  $\text{REACH}(c, c_m, T/2)$
  - [4] Esegui  $\text{REACH}(c_m, c', T/2)$
  - [5] Se entrambi [3] e [4] hanno output ACCETTA, ACCETTA.
- [6] Se non ha accettato in nessun passo precedente, RIGETTA.

# Il teorema di Savitch

## Dimostrazione

Rimane da dimostrare che REACH lavora in spazio  $O(t^2(n))$ .

- [1] Se  $T=1$ , verifica se  $c=c'$  o se la configurazione  $c'$  è raggiungibile da  $c$  in un passo di computazione di  $M$ . ACCETTA se almeno uno dei due test ha successo, se no RIGETTA.
- [2] Se  $T>1$ , per ogni configurazione  $c_m$  di  $M$  che usa spazio  $t(n)$ :
  - [3] Esegui  $\text{REACH}(c, c_m, T/2)$
  - [4] Esegui  $\text{REACH}(c_m, c', T/2)$
  - [5] Se entrambi [3] e [4] hanno output ACCETTA, ACCETTA.
- [6] Se non ha accettato in nessun passo precedente, RIGETTA.

# Il teorema di Cook-Levin

## Dimostrazione

La ricorsione in [2] divide ogni volta  $T$  per due.  
Dunque per  $T = 2^{dt(n)}$  ci sono  $O(\log 2^{dt(n)}) = O(t(n))$  chiamate ricorsive.

Rimane da dimostrare che REACH lavora in spazio  $O(t^2(n))$ .

- [1] Se  $T=1$ , verifica se  $c=c'$  o se la configurazione  $c'$  è raggiungibile da  $c$  in un passo di computazione di  $M$ . ACCETTA se almeno uno dei due test ha successo, se no RIGETTA.
- [2] Se  $T>1$ , per ogni configurazione  $c_m$  di  $M$  che usa spazio  $t(n)$ :
  - [3] Esegui  $\text{REACH}(c, c_m, T/2)$
  - [4] Esegui  $\text{REACH}(c_m, c', T/2)$
  - [5] Se entrambi [3] e [4] hanno output ACCETTA, ACCETTA.
- [6] Se non ha accettato in nessun passo precedente, RIGETTA.

# Il teorema di Cook-Levin

## Dimostrazione

La ricorsione in [2] divide ogni volta  $T$  per due. Dunque per  $T = 2^{dt(n)}$  ci sono  $O(\log 2^{dt(n)}) = O(t(n))$  chiamate ricorsive.

Rimane da dimostrare che REACH lavora in spazio  $O(t^2(n))$ .

[1] Se  $T=1$ , verifica se  $c=c'$  o se c' è stato raggiungibile da c in un passo. Se almeno uno dei due test ha successo,

Ciascuno dei passi [3] e [4] usa spazio  $O(t(n))$ . Quando abbiamo eseguito [3], possiamo cancellare ed eseguire [4].

[2] Se  $T>1$ , per ogni configurazione  $c_m$  di  $M$  che usa spazio  $t(n)$ :

[3] Esegui REACH( $c, c_m, T/2$ )

[4] Esegui REACH( $c_m, c', T/2$ )

[5] Se entrambi [3] e [4] hanno output ACCETTA, ACCETTA.

[6] Se non ha accettato in nessun passo precedente, RIGETTA.

# Il teorema di Cook-Levin

## Dimostrazione

La ricorsione in [2] divide ogni volta  $T$  per due. Dunque per  $T = 2^{dt(n)}$  ci sono  $O(\log 2^{dt(n)}) = O(t(n))$  chiamate ricorsive.

Rimane da dimostrare che REACH lavora in spazio  $O(t^2(n))$ .

[1] Se  $T=1$ , verifica se  $c=c'$  o se c' è un stato raggiungibile da c in un passo. Se almeno uno dei due test ha successo,

Ciascuno dei passi [3] e [4] usa spazio  $O(t(n))$ . Quando abbiamo eseguito [3], possiamo cancellare ed eseguire [4].

[2] Se  $T>1$ , per ogni configurazione  $c_m$  di  $M$  che usa spazio  $t(n)$ :

[3] Esegui REACH( $c, c_m, T/2$ )

[4] Esegui REACH( $c_m, c', T/2$ )

[5] Se entrambi [3] e [4] hanno output ACCETTA, ACCETTA.

[6] Se non ha accettato in

Pertanto REACH nel suo complesso richiede spazio  $O(t(n)) \times O(t(n)) = O(t^2(n))$ .

# Il teorema di Savitch

## Dimostrazione

Come anticipato, definiamo  $M'$  come la TM che su input di lunghezza  $n$  esegue  $\text{REACH}(c_{init}, c_{acc}, 2^{dt(n)})$  e accetta se l'output è ACCETTA.

Dal momento che  $\text{REACH}(c_{init}, c_{acc}, 2^{dt(n)})$  lavora in tempo  $O(t^2(n))$ , abbiamo dimostrato che  $L$  è in  $\text{SPACE}(t^2(n))$ .

# Il teorema di Savitch: considerazioni finali

É sorprendente che  $\text{PSPACE} = \text{NSPACE}$ , oppure no?

# Il teorema di Savitch: considerazioni finali

É sorprendente che PSPACE = NSPACE, oppure no?

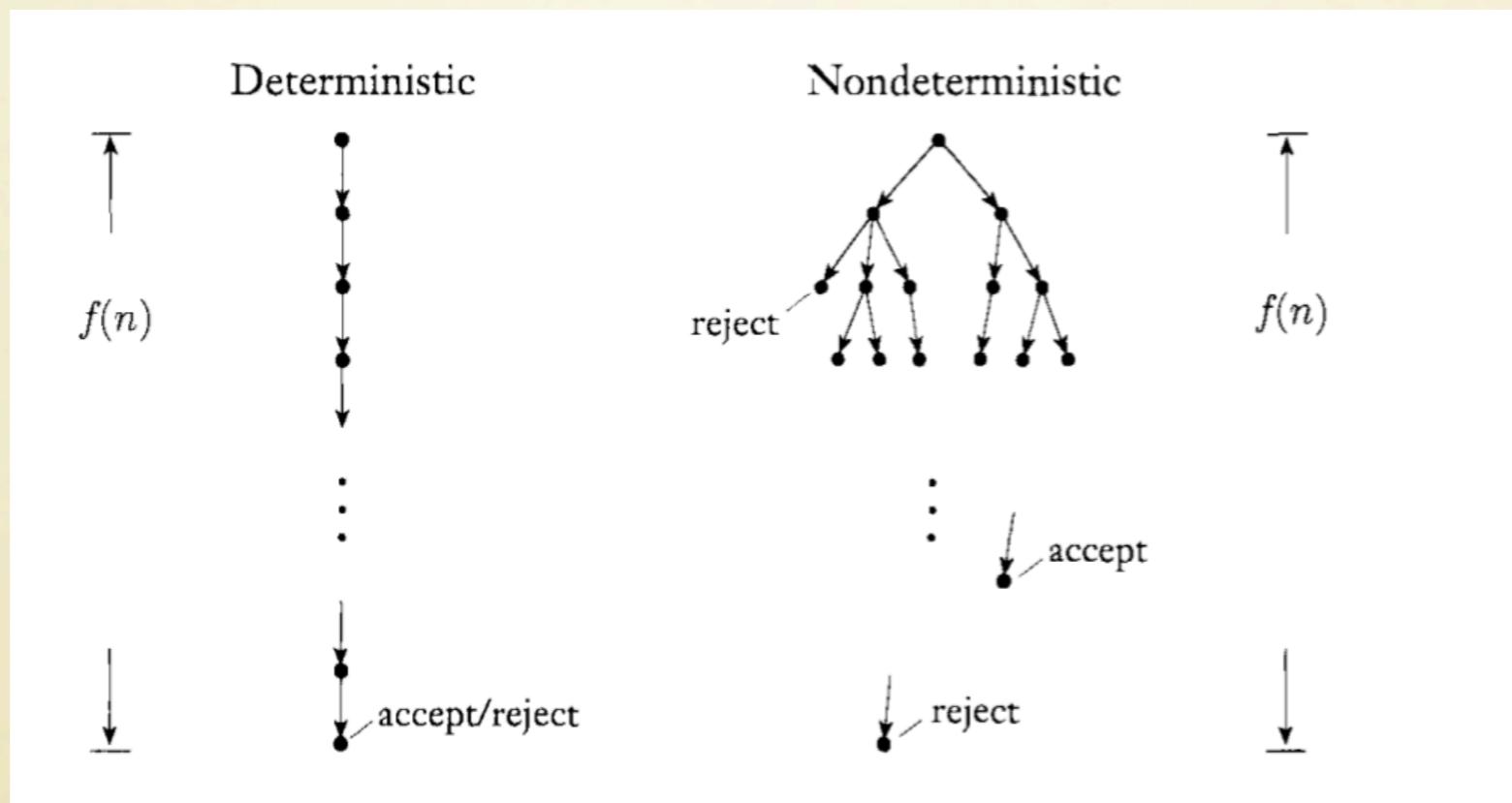


Immagine da Sisper

# Gerarchie e problemi intrattabili

# Gerarchizzare le classi di complessità

La nostra intuizione é che non tutte le classi di complessità di tempo e di spazio corrispondono allo stesso insieme di problemi. Più risorse computazionali sono allocate, più dovrebbe essere grande il numero di problemi che é possibile calcolare.

Per esempio, una TM dovrebbe poter risolvere strettamente più problemi in tempo  $n^3$  che in tempo  $n^2$ .

I risultati che dimostrano tali inclusioni strette sono detti **teoremi di gerarchia**.

Tali risultati hanno come conseguenza che alcuni problemi non appartengono a classi come P o PSPACE. Si tratta dunque di problemi **intrattabili**: nonostante siano decidibili, non esistono algoritmi efficienti.

# Preliminare 1: notazione small-O

Date funzioni  $f, g : \mathbb{N} \rightarrow \mathbb{N}$ , scriviamo  $f(n) = o(g(n))$  e diciamo che  $g(n)$  è un **bound superiore asintotico stretto** se esistono  $c \in \mathbb{N}$ , e  $m \in \mathbb{N}$  tali che per ogni  $n \geq m$

$$f(n) < c \times g(n)$$

In altre parole,  $g(n)$  è strettamente più grande di  $f(n)$  per  $n$  sufficientemente grandi e modulo un fattore costante  $c$ .

Esempi:  $n^2 = o(n^3)$

$n^2 \neq o(n^2)$

$n \times \log n = o(n^2)$

Perché ci serve ora? Perché vogliamo **separare** classi di complessità

## Preliminare 2: Funzioni costruibili

$f : \mathbb{N} \rightarrow \mathbb{N}$ , dove  $f(n) \geq O(\log n)$ , é SPACE-costruibile se possiamo calcolare in spazio  $O(f(n))$  la funzione

$$1^n \mapsto \langle f(n) \rangle$$

dove  $\langle f(n) \rangle$  é la codifica binaria di  $f(n)$ .

Tutte le misure più comuni per misurare la complessità sono SPACE-costruibili.

Esempio:  $n^2$  é SPACE-costruibile. La TM che lo testimonia riceve in input  $1^n$ , traduce in binario,  $\langle n \rangle$ , contando il numero degli 1, e ritorna come output  $\langle n^2 \rangle$  usando qualsiasi metodo standard per moltiplicare n con sé stesso. Lo spazio utilizzato é  $O(n)$ .

## Preliminare 2: Funzioni costruibili

$f : \mathbb{N} \rightarrow \mathbb{N}$ , dove  $f(n) \geq O(\log n)$ , é SPACE-costruibile se possiamo calcolare in spazio  $O(f(n))$  la funzione

$$1^n \mapsto \langle f(n) \rangle$$

dove  $\langle f(n) \rangle$  é la codifica binaria di  $f(n)$ .

Perché ci serve ora? Immagina di avere  $o(f(n)) = g(n)$ , ma  $f(n)$  eccede  $g(n)$  solo di una quantità molto piccola e molto difficile da calcolare. Allora una TM che lavora in  $g(n)$  potrebbe non avere alcun vantaggio a lavorare in  $f(n)$ : anche solo calcolare la quantità di spazio extra potrebbe richiedere più dello spazio disponibile. Assumere  $f$  SPACE-costruibile evita questa situazione.

Esempio:  $n^2$  é SPACE-costruibile. La TM che lo testimonia riceve in input  $1^n$ , traduce in binario,  $\langle n \rangle$ , contando il numero degli 1, e ritorna come output  $\langle n^2 \rangle$  usando qualsiasi metodo standard per moltiplicare  $n$  con sé stesso. Lo spazio utilizzato é  $O(n)$ .

# Gerarchia di spazio

## **Teorema (teorema della gerarchia di spazio)**

Per ogni  $f : \mathbb{N} \rightarrow \mathbb{N}$  SPACE-costruibile, esiste un linguaggio  $L$  decidibile in spazio  $O(f(n))$  ma non in spazio  $o(f(n))$ .

### **Idea della dimostrazione**

Il linguaggio  $L$  verrà definito non da una proprietà ma da un algoritmo ALG, che usa spazio  $O(f(n))$  ed è costruito in modo tale da assicurare che  $L$  è diverso da ogni linguaggio decidibile in spazio  $o(f(n))$ .

Come ci riesce? Per diagonalizzazione. Su input  $\text{code}(M)$ , ALG simula  $M$  su  $\text{code}(M)$  in una porzione di nastro  $f(n)$ , e accetta se e solo se  $M$  ferma e rigetta.

# Gerarchia di spazio

## Teorema (teorema della gerarchia di spazio)

Per ogni  $f : \mathbb{N} \rightarrow \mathbb{N}$  SPACE-computabile esiste un linguaggio  $L$  decidibile in spazio  $O(f(n))$ .

Alcuni dettagli da sistemare: (1)  $M$  potrebbe usare più di  $f(n)$  spazio per  $n$  sufficientemente piccolo; (2) La simulazione di  $M$  su  $\text{code}(M)$  potrebbe richiedere più di  $f(n)$  spazio, e quindi essere rigettata a prescindere.

### Idea della dimostrazione

Il linguaggio  $L$  verrà definito non da una proprietà ma da un algoritmo ALG, che usa spazio  $O(f(n))$  ed è costruito in modo tale da assicurare che  $L$  è diverso da ogni linguaggio decidibile in spazio  $o(f(n))$ .

Come ci riesce? Per diagonalizzazione. Su input  $\text{code}(M)$ , ALG simula  $M$  su  $\text{code}(M)$  in una porzione di nastro  $f(n)$ , e accetta se e solo se  $M$  ferma e rigetta.

# Gerarchia di spazio

**Dimostrazione** Definiamo ALG.

Su input  $w$  di lunghezza  $n$

- [1] Calcola  $f(n)$ , e contrassegna tale quantità di nastro. Se i passi successivi provano ad usare più di  $f(n)$  celle, rigetta.
- [2] Se  $w \neq \text{code}(M)10^*$  per qualche TM  $M$ , rigetta.
- [3] Simula  $M$  su  $w$ . Se non ha concluso dopo  $2^{f(n)}$  passi, rigetta.
- [4] Se  $M$  accetta, rigetta. Se  $M$  rigetta, accetta.

# Gerarchia di S

## Dimostrazione

Definiamo ALG.

Su input  $w$  di lunghezza  $n$

- [1] Calcola  $f(n)$ , e contrassegna tale quantità di nastro. Se i passi successivi provano ad usare più di  $f(n)$  celle, rigetta.
- [2] Se  $w \neq \text{code}(M)10^*$  per qualche TM  $M$ , rigetta.
- [3] Simula  $M$  su  $w$ . Se non ha concluso dopo  $2^{f(n)}$  passi, rigetta.
- [4] Se  $M$  accetta, rigetta. Se  $M$  rigetta, accetta.

Aggiungiamo una stringa  $s \in 10^*$  a  $\text{code}(M)$  affinché la lunghezza dell'input di  $M$  diventi grande abbastanza perché valga il bound asintotico  $o(f(n))$  sullo spazio di computazione di  $M$ .

# Gerarchia di S

## Dimostrazione

Definiamo ALG.

Su input  $w$  di lunghezza  $n$

- [1] Calcola  $f(n)$ , e contrassegna tale quantità di nastro. Se i passi successivi provano ad usare più di  $f(n)$  celle, rigetta.
- [2] Se  $w \neq \text{code}(M)10^*$  per qualche TM  $M$ , rigetta.
- [3] Simula  $M$  su  $w$ . Se non ha concluso dopo  $2^{f(n)}$  passi, rigetta.
- [4] Se  $M$  accetta, rigetta. Se  $M$  rigetta, accetta.

$M$  ha un alfabeto arbitrario. Possiamo codificarlo rappresentando ogni cella del nastro di  $M$  con più celle del nastro di ALG. L'overhead di spazio è solo costante.

Aggiungiamo una stringa  $s \in 10^*$  a  $\text{code}(M)$  affinché la lunghezza dell'input di  $M$  diventi grande abbastanza perché valga il bound asintotico  $o(f(n))$  sullo spazio di computazione di  $M$ .

# Gerarchia di S

## Dimostrazione

Definiamo ALG.

Su input  $w$  di lunghezza  $n$

- [1] Calcola  $f(n)$ , e contrassegna tale quantità di nastro. Se i passi successivi provano ad usare più di  $f(n)$  celle, rigetta.
- [2] Se  $w \neq \text{code}(M)10^*$  per qualche TM  $M$ , rigetta.
- [3] Simula  $M$  su  $w$ . Se non ha concluso dopo  $2^{f(n)}$  passi, rigetta.
- [4] Se  $M$  accetta, rigetta. Se  $M$  rigetta, accetta.

$M$  ha un alfabeto arbitrario. Possiamo codificarlo rappresentando ogni cella del nastro di  $M$  con più celle del nastro di ALG. L'overhead di spazio è solo costante.

Aggiungiamo una stringa  $s \in 10^*$  a  $\text{code}(M)$  affinché la lunghezza dell'input di  $M$  diventi grande abbastanza perché valga il bound asintotico  $o(f(n))$  sullo spazio di computazione di  $M$ .

In linea di principio,  $M$  potrebbe entrare in un ciclo durante la computazione in spazio  $f(n)$ . Se termina, lo fa sempre entro  $2^{f(n)}$  passi. Vogliamo assicurarcici che ALG termini sempre.

# Gerarchia di spazio

## Dimostrazione

ALG richiede spazio  $O(f(n))$ . Definiamo  $L$  come il linguaggio

$$L = \{w \mid \text{ALG accetta } w\}$$

Pertanto  $L$  è decidibile in spazio  $O(f(n))$ . Dobbiamo dimostrare che non esiste TM che lo decida in spazio  $o(f(n))$ .

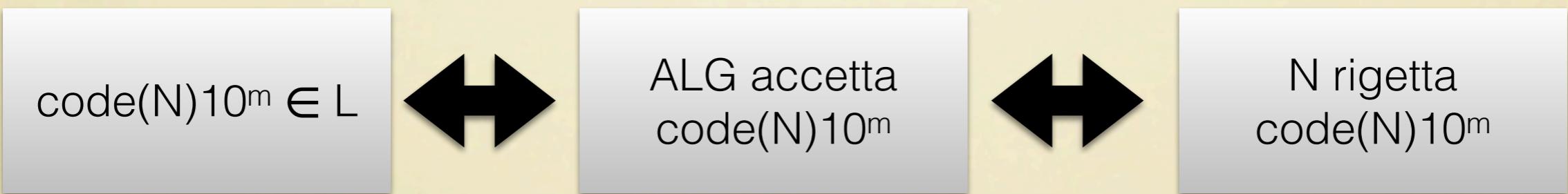
# Gerarchia di spazio

## Dimostrazione

Per contraddizione, supponiamo che esista  $N$  tale che  $N$  decide  $L$  in spazio  $g(n) = o(f(n))$ .

Per costruzione,  $ALG$  simula  $N$  in spazio  $d \times g(n)$  per qualche costante  $d$ . Poiché  $g(n) = o(f(n))$ , esiste una costante  $m$  tale che, per tutti gli  $n \geq m$ ,  $(d \times g(n)) < f(n)$ .

Perciò la simulazione da parte di  $ALG$  di  $N$  su input  $code(N)10^m$  andrà a buon fine nello spazio allocato  $f(n)$ . Abbiamo:



Perciò  $N$  non può decidere  $L$ , contraddizione.

# Conseguenze

**Corollario** Per ogni  $k, j$ , tali che  $k < j$ :

$$SPACE(n^k) \subsetneq SPACE(n^j)$$

# Conseguenze

**Corollario** Per ogni  $k, j$ , tali che  $k < j$ :

$$SPACE(n^k) \subsetneq SPACE(n^j)$$

**Corollario**  $SPACE(\log n) \subsetneq SPACE(n)$

# Conseguenze

**Corollario** Per ogni  $k, j$ , tali che  $k < j$ :

$$SPACE(n^k) \subsetneq SPACE(n^j)$$

**Corollario**  $SPACE(\log n) \subsetneq SPACE(n)$

Definiamo  $EXPSPACE = \bigcup_k SPACE(2^{n^k})$

**Corollario**  $PSPACE \subsetneq EXPSPACE$

# Conseguenze

**Corollario** Per ogni  $k, j$ , tali che  $k < j$ :

$$SPACE(n^k) \subsetneq SPACE(n^j)$$

**Corollario**  $SPACE(\log n) \subsetneq SPACE(n)$

Definiamo  $EXPSPACE = \bigcup_k SPACE(2^{n^k})$

**Corollario**  $PSPACE \subsetneq EXPSPACE$

Quindi i problemi EXPSPACE-completi sono **intrattabili** (non hanno algoritmi PSPACE).

# Problemi EXPSPACE-completi

In generale, si tratta di problemi riguardanti strutture molto ‘succinte’ (nel modo in cui rappresentano l’informazione), ma che hanno una soluzione che richiede l’esplorazione di uno spazio di soluzioni ampio. Ad esempio:

- Se due espressioni regolari (basate su unione, concatenazione, duplicazione, e Kleene star) rappresentano differenti linguaggi.
- Word problems per varie strutture in algebra e geometria algebrica.

# Gerarchia di tempo

$f : \mathbb{N} \rightarrow \mathbb{N}$ , dove  $f(n) \geq O(n \times \log n)$ , é TIME-costruibile se possiamo calcolare in tempo  $O(f(n))$  la funzione

$$1^n \mapsto \langle f(n) \rangle$$

dove  $\langle f(n) \rangle$  é la codifica binaria di  $f(n)$ .

# Gerarchia di tempo

## **Teorema (teorema della gerarchia di tempo)**

Per ogni  $f : \mathbb{N} \rightarrow \mathbb{N}$  TIME-costruibile, esiste un linguaggio  $L$  decidibile in tempo  $O(f(n))$  ma non in tempo  $o(f(n)) / \log f(n)$ .

## **Idea della dimostrazione**

La dimostrazione è simile a quella data per la gerarchia di spazio. Il fattore  $\log f(n)$  è dovuto al fatto che la simulazione di una TM richiede un overhead di tempo logaritmico (mentre nel caso dello spazio, l'overhead era costante).

Omettiamo i dettagli della dimostrazione (che si possono trovare in Sipser, capitolo 8).

# Conseguenze

**Corollario** Per ogni  $k, j$ , tali che  $1 < k < j$ :

$$TIME(n^k) \subsetneq TIME(n^j)$$

**Corollario**  $P \not\subseteq EXPTIME$

Quindi i problemi EXPTIME-completi sono **intrattabili** (non hanno algoritmi polinomiali).

# Problemi EXPTIME-completi

Come nel caso di EXPTIME, si tratta di problemi riguardanti strutture molto ‘succinte’ (nel modo in cui rappresentano l’informazione), ma che hanno una soluzione che richiede l’esplorazione di uno spazio di soluzioni ampio e complesso. Ad esempio:

- Il problema della fermata ‘entro  $k$  passi’.
- Elaborare una strategia a scacchi, dama, GO, ed altri giochi.

# Considerazioni finali

I teoremi di gerarchia sono entrambi basati sull'uso della *diagonalizzazione* per separare classi di complessità.

In astratta, qualsiasi tecnica detta di diagonalizzazione si basa su due principi:

- la possibilità di rappresentare TM come stringhe.
- L'abilità di una TM di simulare un'altra TM di cui riceve il codice come input *senza 'troppo' overhead in termini di spazio e tempo*.

**Quanto lontano ci può portare la diagonalizzazione?  
Perché non la utilizziamo per dimostrare che  $P \neq NP$ ?**

# Considerazioni finali

Consideriamo una variante della TM:

**una TM con oracolo**  $O \subseteq \{0,1\}^*$  può, in qualunque momento della computazione, chiedere  $w \in O?$  e ricevere risposta in tempo/spazio costante.

L'osservazione che ci interessa è che, indipendentemente dalla scelta di  $O$ , i due presupposti indicati per la diagonalizzazione valgono anche per le TM con oracolo  $O$ .

Perciò i risultati dimostrati usando la diagonalizzazione (ad esempio, i teoremi di gerarchia) valgono anche per le TM con oracolo.

# Considerazioni finali

**Teorema** (Baker, Gill, Solovay 1975)

Esistono oracoli  $O$  e  $O'$  tali che  $P^O = NP^O$  e  $P^{O'} \neq NP^{O'}$ .

Classe dei problemi decidibili in tempo polinomiale da una TM con oracolo (deterministica se P, non-deterministica se NP)

# Considerazioni finali

**Teorema** (Baker, Gill, Solovay 1975)

Esistono oracoli  $O$  e  $O'$  tali che  $P^O = NP^O$  e  $P^{O'} \neq NP^{O'}$ .

Classe dei problemi decidibili in tempo polinomiale da una TM con oracolo (deterministica se P, non-deterministica se NP)

Perciò, se mai trovassimo una risposta al quesito P=NP?, o non usa la diagonalizzazione, o la utilizza insieme ad altre assunzioni che non valgono per TM con oracolo.

# Considerazioni finali

**Teorema** (Baker, Gill, Solovay 1975)

Esistono oracoli  $O$  e  $O'$  tali che  $P^O = NP^O$  e  $P^{O'} \neq NP^{O'}$ .

Classe dei problemi decidibili in tempo polinomiale da una TM con oracolo (deterministica se P, non-deterministica se NP)

Perciò, se mai trovassimo una risposta al quesito P=NP?, o non usa la diagonalizzazione, o la utilizza insieme ad altre assunzioni che non valgono per TM con oracolo.

Tali risultati si chiamano **non-relativizzanti** (nel senso che non relativizzano a TM con oracolo). Per esempio i teoremi di gerarchia relativizzano, il teorema di Cook-Levin no.

# Considerazioni finali

# Considerazioni finali

In conclusione, la diagonalizzazione é un metodo potente, ma con limiti.

# Considerazioni finali

In conclusione, la diagonalizzazione é un metodo potente, ma con limiti.

Per attaccare domande come  $P=NP?$ , non é sufficiente simulare computazioni, trattandole come ‘scatole nere’. Dobbiamo fare un’analisi più concreta, che riguardi l’effettivo contenuto della computazione.

# Considerazioni finali

In conclusione, la diagonalizzazione é un metodo potente, ma con limiti.

Per attaccare domande come  $P=NP?$ , non é sufficiente simulare computazioni, trattandole come ‘scatole nere’. Dobbiamo fare un’analisi più concreta, che riguardi l’effettivo contenuto della computazione.

Queste osservazioni hanno portato allo studio di modelli di calcolo differenti, dove l’analisi degli algoritmi é più flessibile. Un esempio importante sono i *circuiti booleani*.