Computabilità, Complessità e Logica

Lezione 10

Complessità

Fino ad ora abbiamo visto i linguaggi che le MdT possono riconoscere o decidere

Ora passiamo ad analizzare l'efficienza con cui un linguaggio viene deciso.

Data una parola $w \in \Sigma^\star$ possiamo chiederci:

- **Complessità temporale**: quanti passi servono per decidere se $w \in L$?
- **Complessità spaziale**: quante celle di nastro ci servono per decidere se $w \in L$?

Complessità

Vi è una importante differenza con le questioni di individuare il tempo di calcolo di un algoritmo:

- Il tempo di calcolo di un algoritmo è specifico per quell'algoritmo, non è detto che lo stesso problema non sia risolvibile in modo più efficiente
- La complessità di un linguaggio dipende dal problema, non dall'algoritmo utilizzato.
- È come chiedere: *"tra tutti i possibili algoritmi che risolvono questo problema quale è il minimo tempo/ spazio richiesto?"*

Tempo di calcolo

Data una macchina di Turing M ed un input $w \in \Sigma^\star$ possiamo contare il numero di passi che M richiede prima di fermarsi, che indicheremo con $t(w)$

Per tutti gli $n \in \mathbb{N}$ definiamo $t(n)$ come il massimo numero di passi che che M richiede per arrestarsi tra tutte le parole $w \in \Sigma^\star$ con $|w| = n$ (ovvero il massimo valore di $t(w)$ tra tutte le parole di lunghezza n):

$$t(n) = \max_{w \in \Sigma^\star, |w| = n} t(w)$$

Tempo di calcolo per MdT non deterministiche

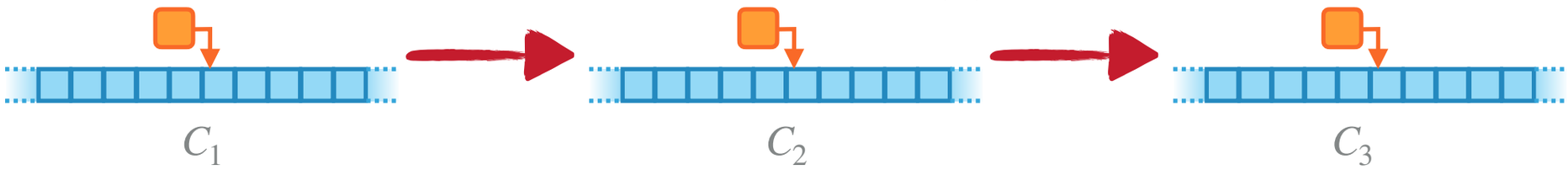
Il tempo di calcolo per MdT non deterministiche è lievemente più complesso perché abbiamo più di una computazione

Su una parola $w \in \Sigma^\star$ di input definiamo $t(w)$ il massimo numero di passi tra tutte le possibili computazioni

Come prima, $t(n)$ rappresenta il massimo $t(w)$ tra tutte le parole $w \in \Sigma^\star$ di lunghezza n

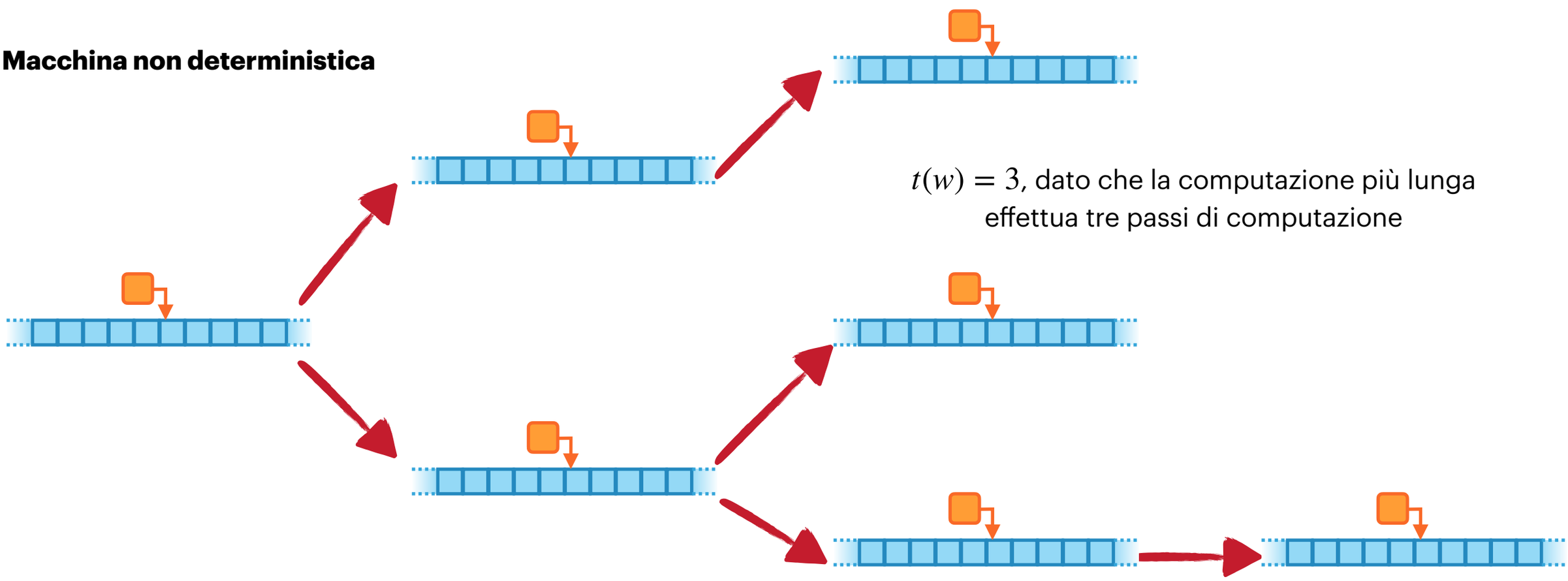
Tempo di calcolo per MdT non deterministiche

Macchina deterministica



$t(w) = 2$, dato che effettuiamo due passi di computazione

Macchina non deterministica



$t(w) = 3$, dato che la computazione più lunga effettua tre passi di computazione

Complessità temporale dei linguaggi

A noi però interessa non la singola MdT, ma un linguaggio

Dato un linguaggio L possiamo avere più MdT che lo decidono, ognuna con la sua complessità temporale

L'esistenza di una MdT che decide L in tempo $t(n)$ ci dice il tempo $t(n)$ è **sufficiente** per decidere L

Ma questo non significa che non sia possibile fare di meglio!

Classi di complessità temporali

- Definiamo come $\mathrm{TIME}(f(n))$ l'insieme di tutti i **linguaggi** per i quali esiste una macchina di Turing **deterministica** che li decide in tempo $O(f(n))$

- Similmente, definiamo come $\mathrm{NTIME}(f(n))$ l'insieme di tutti i **linguaggi** per i quali esiste una macchina di Turing **non deterministica** che li decide in tempo $O(f(n))$

- Chiaramente $\mathrm{TIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$ per qualsiasi funzione $f(n)$

Classi di complessità temporali

- Notate che per dimostrare che $L \notin \mathrm{TIME}(f(n))$ dobbiamo dimostrare che non esiste alcuna TM in grado di decidere L in tempo $O(f(n))$
- Questo è differente dalle analisi che fate nel corso di algoritmi!
- È anche il motivo per cui molti dei problemi aperti nell'ambito della complessità sono difficili da risolvere: non è sufficiente far vedere che esiste un algoritmi inefficiente, è necessario dimostrare che non ne esistono di efficienti!

Modelli di calcolo “ragionevoli”

- Noi usiamo MdT per studiare la complessità temporale
- Una possibile critica è che una MdT è dissimile da un moderno computer o dal modello di macchina RAM
- Ma, informalmente, tutti i sistemi di calcolo deterministici ragionevoli sono **polinomialmente equivalenti**
- Ovvero possono tutti simularsi a vicenda con un rallentamento al più polinomiale
- Quindi possiamo continuare a usare le MdT come modelli

Classi di complessità temporali

- La classe di complessità dei linguaggi riconoscibili da macchine di Turing deterministiche in tempo polinomiale è indicata con P ed è definita come
$$\mathrm{P} = \bigcup_{k \in \mathbb{N}} \mathrm{TIME}(n^k)$$

- La classe di complessità dei linguaggi riconoscibili da macchine di Turing non deterministiche in tempo polinomiale è indicata con NP ed è definita come
$$\mathrm{NP} = \bigcup_{k \in \mathbb{N}} \mathrm{NTIME}(n^k)$$

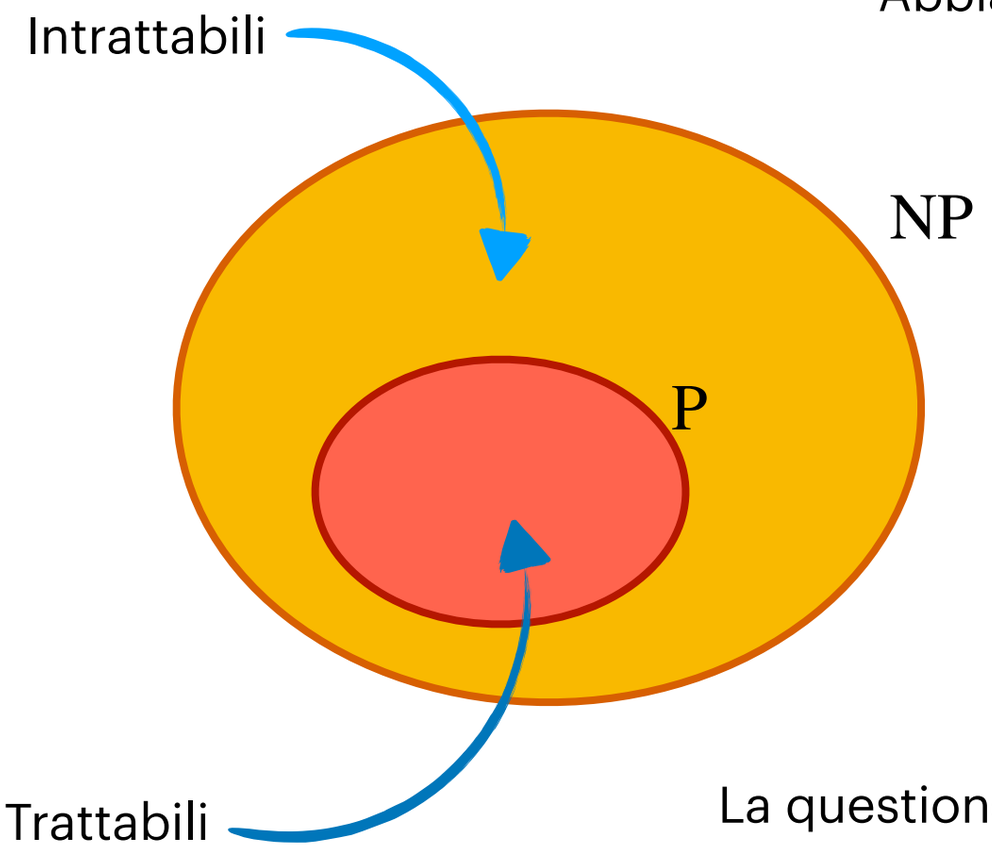
Classi di complessità temporali

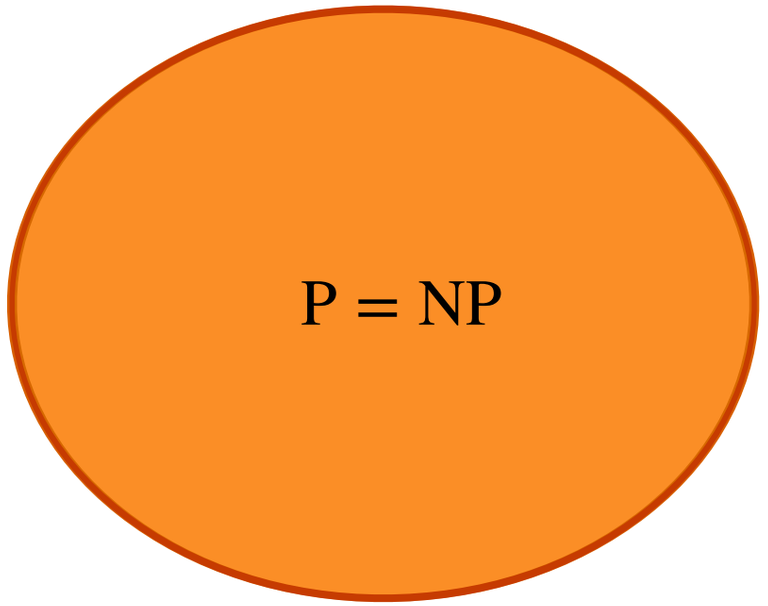
- P rappresenta l'insieme dei problemi che sono risolvibili in modo "efficiente" (anche se un tempo n^{100} non è particolarmente efficiente)
- P è invariante rispetto ad ogni modello di calcolo deterministico ragionevole
- Quindi P è di interesse pratico. Se un linguaggio L è in P allora esiste un algoritmo che lo risolve in tempo polinomiale
- P è una classe robusta, che non varia al variare del modello di calcolo

Classi di complessità

E il problema “P=NP”

Abbiamo due possibilità:





La questione se $\mathrm{P} = \mathrm{NP}$ o $\mathrm{P} \neq \mathrm{NP}$
è uno dei grandi problemi aperti
dell’informatica teorica

L’impressione corrente è che $\mathrm{P} \neq \mathrm{NP}$

Definizione alternativa di NP

- NP rappresenta una classe di linguaggi che sono verificabili in tempo polinomiale
- Vediamo una definizione alternativa di NP e mostriamo l'equivalenza con la definizione già data
- Verificare un linguaggio significa, in modo intuitivo, per ogni parola $w \in L$ avere una stringa aggiuntiva c detta certificato che possiamo usare per verificare che effettivamente $w \in L$

Verificatori in tempo polinomiale

- Un verificatore di un linguaggio L è un algoritmo V tale per cui
  $L = \{w \ : \ V \text{ accetta } \langle w, c \rangle \text{ per qualche string } c\}$

- Se V richiede tempo polinomiale rispetto a w per accettare/rifiutare, allora V è un verificatore in tempo polinomiale per w

- Se esiste un verificatore in tempo polinomiale per L, allora L è verificabile in tempo polinomiale

Verificatori in tempo polinomiale

- Dato che V deve eseguire in tempo polinomiale rispetto a w, ne segue che c, il certificato, deve essere di lunghezza polinomiale rispetto a w, altrimenti non avremmo il tempo di leggere tutto c
- Ma cosa è c?
- Come possiamo mostrare l'equivalenza delle due definizioni di $\mathbf{NP}$?

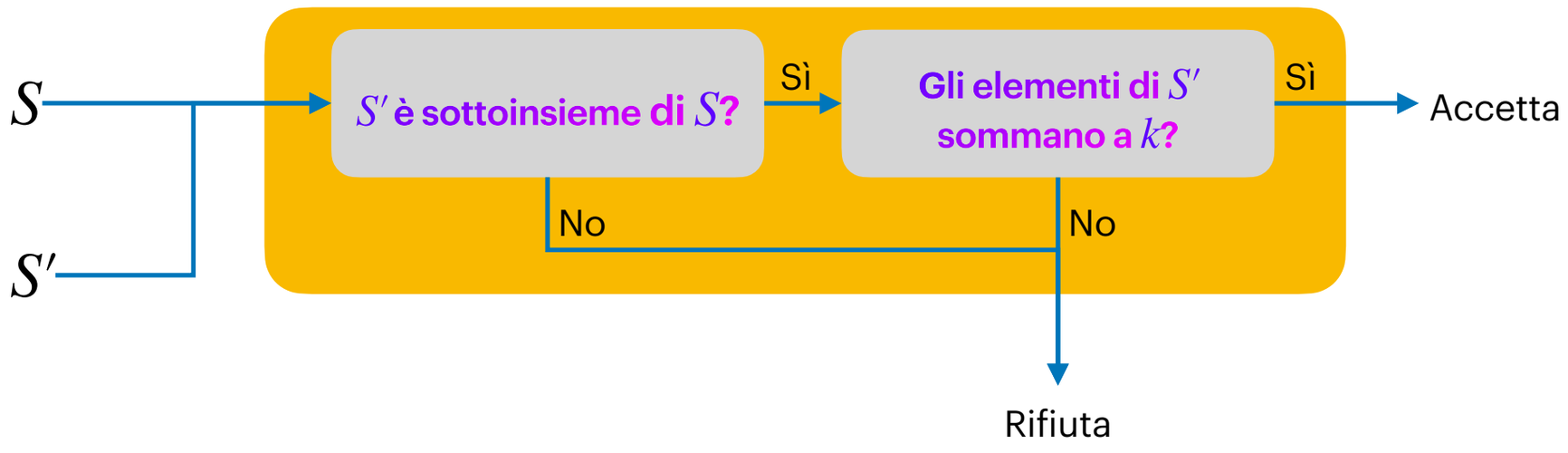
Verificatori in tempo polinomiale

Insieme di numeri interi: $S = \{s_1, s_2, \ldots, s_m\}$

Scegliamo il seguente linguaggio: $L = \{S \mid \text{esiste } S' \subseteq S \text{ con } \sum_{s \in S'} s = k\}$

Dato un insieme S come possiamo trovare un certificato che mostri che $S \in L$?

Un buon candidato è S' i cui elementi hanno somma k



Questa verifica può essere compiuta in tempo polinomiale, quindi L è verificabile in tempo polinomiale

Equivalenza tra le definizioni di NP

- Supponiamo di avere una MdT non deterministica M che lavora in tempo polinomiale, costruiamo un verificatore V che lavora in tempo polinomiale per lo stesso linguaggio deciso da M

- Se su input w la macchina M accetta, significa che esiste una computazione accettante $C_1, \ldots, C_k$ di lunghezza polinomiale rispetto a w

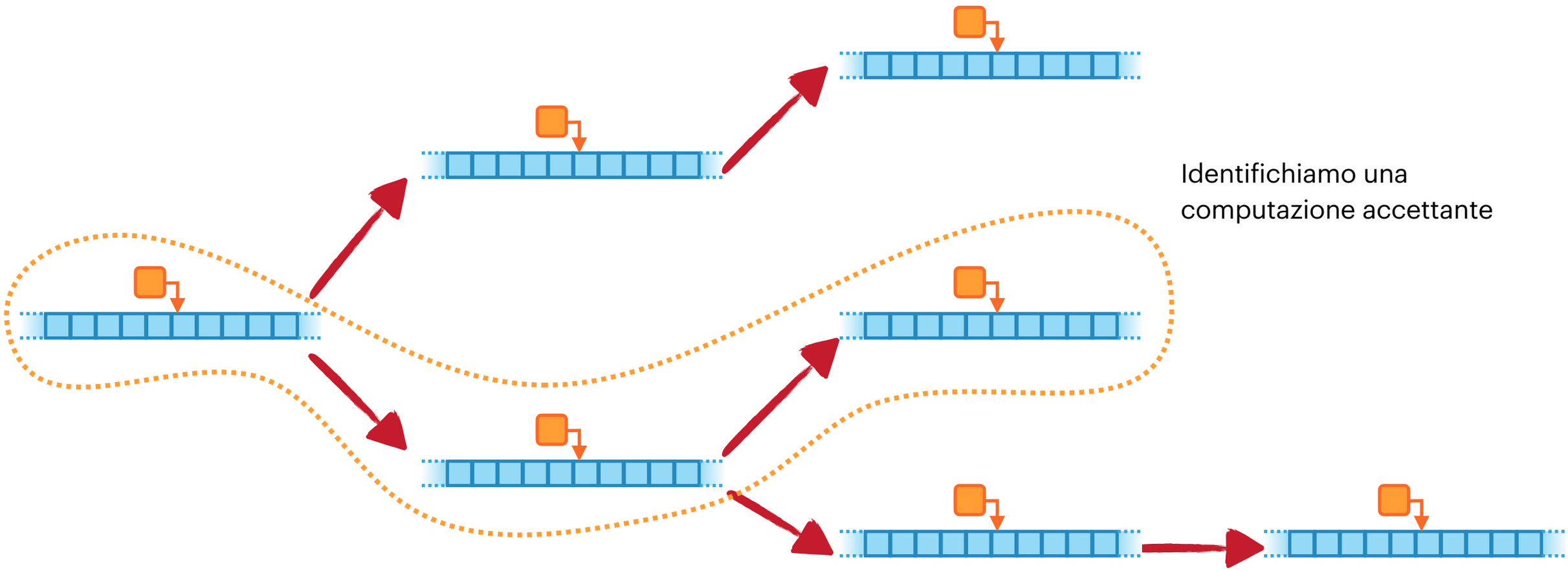
- Per ogni passaggio C_i a C_{i+1} è stata applicata una transizione nella forma (q, σ, M) con $q \in Q$, $\sigma \in \Sigma$ e $M \in \{ \leftarrow , \rightarrow \}$

Equivalenza tra le definizioni di NP

- Usiamo come certificato c la sequenza di transizioni applicate lungo tutta la computazione (sono in numero polinomiale)
- Queste transizioni ci identificano una specifica computazione della MdT non deterministica M
- Il verificatore deve solo simulare quella computazione, senza fare scelte non deterministiche, dato che le transizioni da fare sono tutte in c e verificare che la computazione accetti

Equivalenza tra le definizioni di NP

Macchina non deterministica



Identifichiamo una computazione accettante

Il verificatore deve solo fare un “replay” della computazione e verificare che accetti

Equivalenza tra le definizioni di NP

- Abbiamo mostrato che per ogni linguaggio accettato da una MdT non deterministica che lavora in tempo polinomiale possiamo costruire un verificatore polinomiale per lo stesso linguaggio

- Dobbiamo mostrare anche l'inclusione in senso inverso

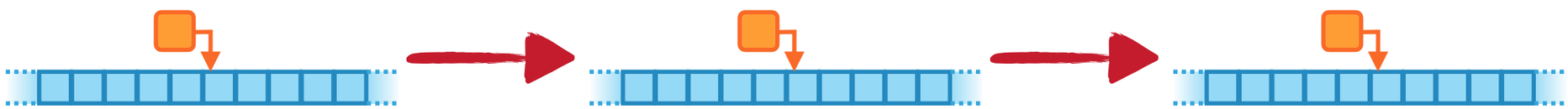
- Sfruttiamo il fatto che possiamo usare una MdT non deterministica per scrivere un "certificato" sul nastro e, se esiste un certificato che ci permette di accettare, questo sarà presente in una computazione

Equivalenza tra le definizioni di NP

- Sia V un verificatore in tempo polinomiale per L
- Assumiamo V esegua in tempo n^k
- Costruiamo una MdT non deterministica che su input w fa due cose:
  - Genera in modo non deterministico un certificato c di lunghezza al più n^k
  - Simula V su input $\langle w, c \rangle$ e accetta se V accetta e rifiuta se V rifiuta

Equivalenza tra le definizioni di NP

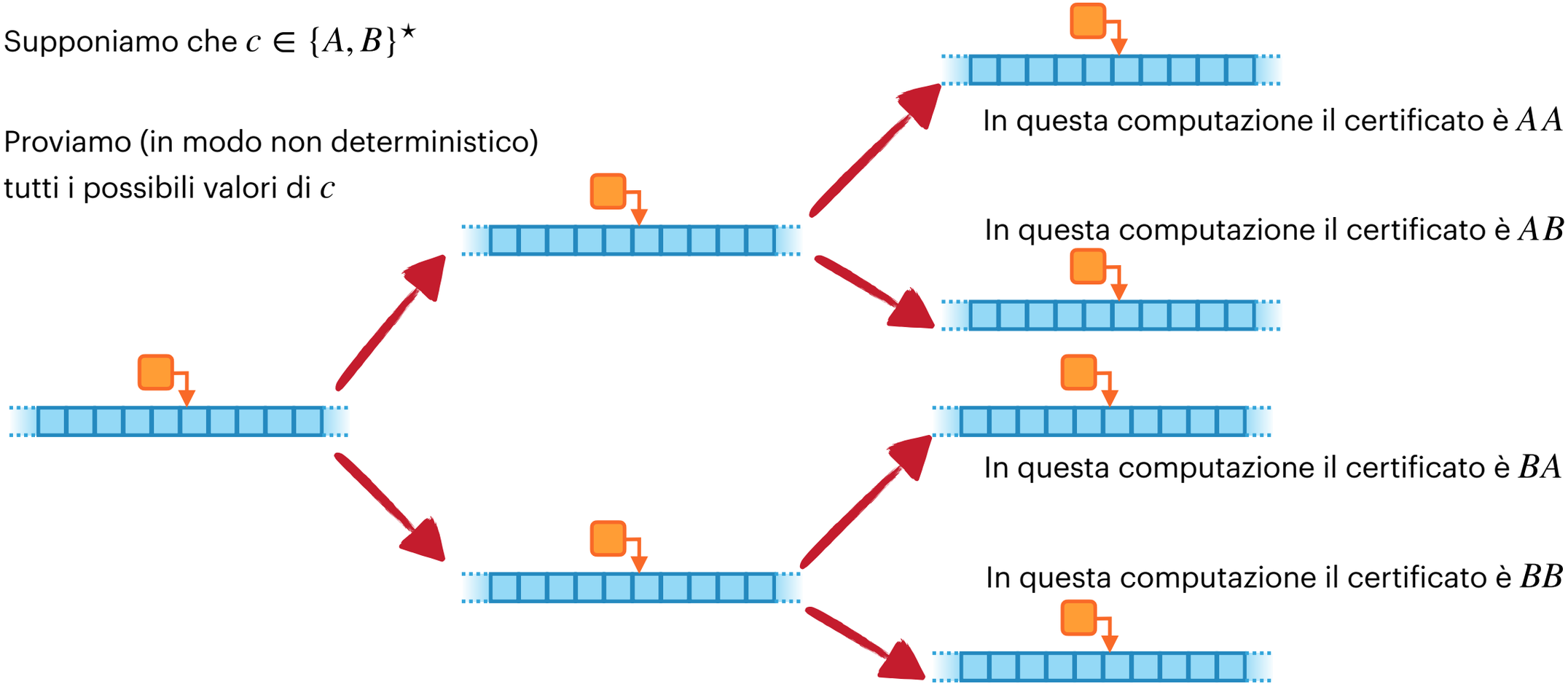
Sappiamo che V accetta su input $\langle w, c \rangle$ se $w \in L$



Ma conosciamo solo w e non c

Supponiamo che $c \in \{A, B\}^\star$

Proviamo (in modo non deterministico) tutti i possibili valori di c



Equivalenza tra le definizioni di NP

- Abbiamo mostrato che per ogni linguaggi per il quale esiste un verificatore polinomiale possiamo costruire una MdT non deterministica che decide lo stesso linguaggio in tempo polinomiale
- Quindi le due definizioni di NP sono equivalenti
- Chiedersi se $\mathrm{P} = \mathrm{NP}$ può essere visto come chiedersi se il certificato sia necessario

Spazio di calcolo per MdT

Data una macchina di Turing M ed un input $w \in \Sigma^\star$, indichiamo con $s(w)$ il massimo numero di celle non blank durante la computazione di M su input w. Questo rappresenta la spazio richiesto da M su input w

Per tutti gli $n \in \mathbb{N}$ definiamo $s(n)$ come il massimo dello spazio richiesto da M per riconoscere una parola $w \in \Sigma^\star$ di lunghezza n (i.e., $|w| = n$):

$$s(n) = \max_{w \in \Sigma^\star,\ |w|=n} s(w)$$

Spazio di calcolo per MdT non deterministiche

- Come per il tempo di calcolo, anche per lo spazio nelle TM non deterministiche c'è da gestire la presenza di più computazioni
- Su una parola w di input definiamo $s(w)$ il massimo spazio utilizzato tra tutte le possibili computazioni
- Come prima, $s(n)$ rappresenta il massimo $s(w)$ tra tutti le parole w di lunghezza n

Classi di complessità spaziali

Definiamo come $\mathrm{SPACE}(f(n))$ l'insieme di tutti i linguaggi per i quali esiste una macchina di Turing deterministica che li decide in spazio $O(f(n))$

Similmente, definiamo come $\mathrm{NSPACE}(f(n))$ l'insieme di tutti i linguaggi per i quali esiste una macchina di Turing non-deterministica che li decide in spazio $O(f(n))$

Chiaramente $\mathrm{SPACE}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$ per qualsiasi funzione $f(n)$

Classi di complessità spaziali

- La classe di complessità dei linguaggi riconoscibili da macchine di Turing deterministiche in spazio polinomiale è indicata con PSPACE ed è definita come
$$\mathrm{PSPACE} = \bigcup_{k \in \mathbb{N}} \mathrm{SPACE}(n^k)$$
- La classe di complessità dei linguaggi riconoscibili da macchine di Turing non deterministiche in spazio polinomiale è indicata con $\mathrm{NPSPACE}$ ed è definita come
$$\mathrm{NPSPACE} = \bigcup_{k \in \mathbb{N}} \mathrm{NPSPACE}(n^k)$$

Relazione tra classi di complessità spaziali e temporali

- Se $f(n)$ è almeno lineare possiamo trovare una relazione tra classi di complessità spaziali e temporali:
  - Per utilizzare $s(n)$ celle la TM deve scrivere su $s(n) - n$ di esse almeno una volta (n celle sono già occupate dall'input)
  - Dato che una TM può scrivere al più una cella per ogni passo, abbiamo che $\mathrm{TIME}(f(n)) \subseteq \mathrm{SPACE}(f(n))$ e che $\mathrm{NTIME}(f(n)) \subseteq \mathrm{NSPACE}(f(n))$