RIDUZIONI
PROBLEMI NP-COMPLETI

INFORMATICA

CALCOLARE FUNZIONI

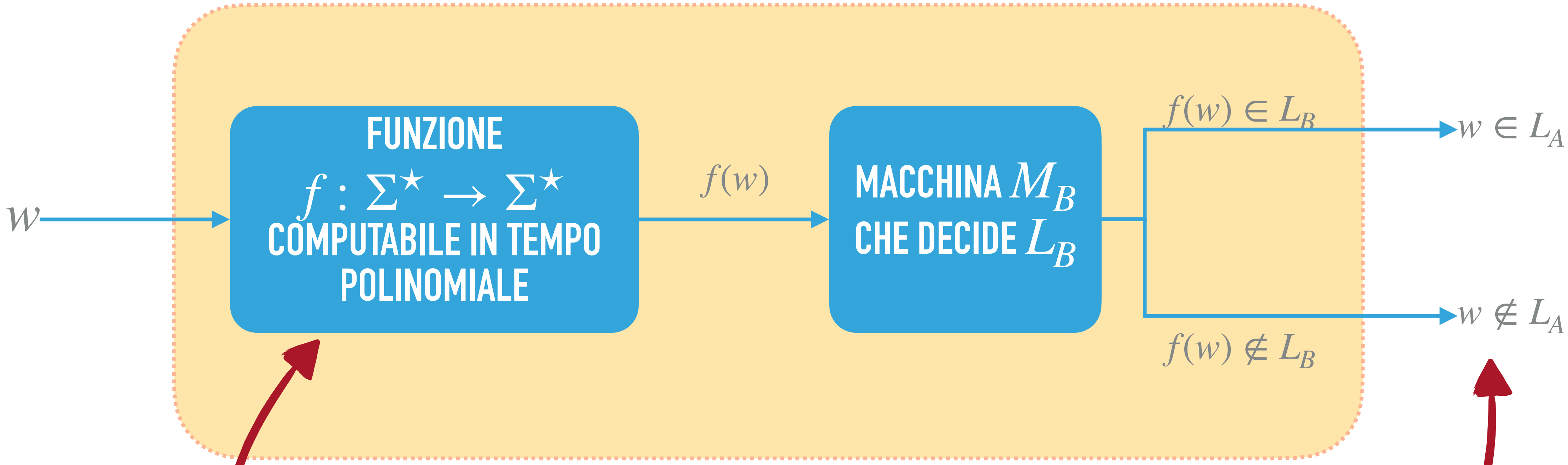
- Abbiamo visto macchine di Turing che decidono l'appartenenza di parole a linguaggi
- Possiamo usare macchine di Turing per calcolare funzioni fornendo una convenzione su come ottenere il risultato
- Per esempio una TM M può calcolare una funzione $f : \Sigma^\star \to \Sigma^\star$ scrivendo sul nastro $f(w)$ quando M accetta su input w
- La nozione di tempo di calcolo di una TM si è valida anche per le TM che calcolano funzioni

RIDUZIONI

- Possiamo sfruttare la possibilità di calcolare funzioni per trasformare un linguaggio in un altro
- 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 $L_A \leq_P L_B$

RIDUZIONE IN TEMPO POLINOMIALE

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



Macchina che decide L_A

Possiamo modificare l'input del problema applicando una funzione che richiede tempo polinomiale per essere calcolata

Ma non possiamo cambiare l'output, Se M_B accetta allora accettiamo e se M_B rifiuta allora rifiutiamo

PROBLEMI COMPLETI: INTUIZIONE

- Vogliamo individuare i problemi di decisione (linguaggi) che sono "i più difficili" per una certa classe di linguaggi $\mathscr{C}$
- 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 $\mathscr{C}$
- Ma abbiamo appena definito un metodo per usare un problema per risolverne altri: la 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

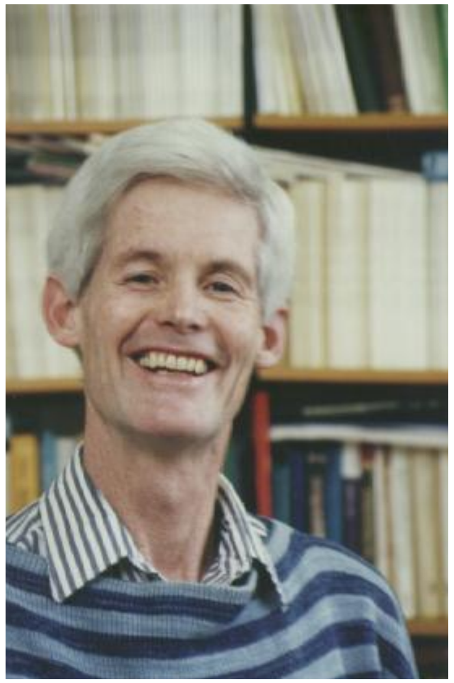
- La seconda condizione (ovvero che tutti i linguaggi di $\mathscr{C}$ sono riducibili a L) ci dice che L è **hard** o **difficile** per $\mathscr{C}$ (indicato anche con $\mathscr{C}$-hard o $\mathscr{C}$-difficile).
- Se L è $\mathscr{C}$-hard possiamo usarlo per risolvere tutti i problemi in $\mathscr{C}$, 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 $\mathscr{C} = \mathrm{NP}$ 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

TEOREMA DI COOK-LEVIN

Indipendentemente Cook (1971) e Levin (1973) dimostrarono che esiste problema è NP-completo



Stephen Cook

Leonid Levin

SAT (soddisfacibilità Booleana) è NP-completo

MA COSA È SAT?

- SAT chiede se data una formula in logica proposizionale esiste un assegnamento che la soddisfa.
- Per capire il problema ci serve sapere:
  - Cosa è una formula in logica proposizionale?
  - Cosa è un assegnamento?
  - Cosa significa che un assegnamento soddisfa una formula?

CONGIUNZIONE, DISGIUNZIONE, NEGAZIONE

- Un breve ripasso della notazione che utilizziamo
  - $a \wedge b$ è la **congiunzione** di a e b
  - $a \vee b$ è la **disgiunzione** di a e b
  - $\neg a$ è la **negazione** di a
- Useremo anche l'implicazione $a \to b$ come abbreviazione di $\neg a \vee b$ e l'abbreviazione $a \leftrightarrow b$ per $(a \to b) \wedge (b \to a)$

FORMULE

- Sia $V = \{x_1, \ldots, x_n\}$ un insieme di variabili proposizionali
- Le formule sono definite in modo induttivo come:
  - $\varphi \in V$ è una formula
  - Se φ è una formula, allora $\neg\varphi$ è una formula
  - Se φ e ψ sono formule, allora $(\varphi \land \psi)$ e $(\varphi \lor \psi)$ sono formule

FORMULE: ESEMPI

- Se $V = \{A, B, C\}$ allora $(A \wedge B)$ è una formula, A e $\neg A$ sono formule, $(A \vee (B \wedge C))$ è una formula, etc.
- Ma $A \wedge$, AB, $A \neg B$ non sono formule ben formate
- Generalmente assegnamo un significato alle variabili. Per esempio "oggi piove" è A "oggi c'è vento" è B, quindi $(A \wedge B)$ andrà a significare "oggi piove e oggi c'è vento"

ASSEGNAMENTO

- Data una formula, per esempio $((A \vee B) \wedge (\neg B \vee C))$ possiamo assegnare ad ogni variabile un valore vero (*true*) o falso (*false*)
- Possiamo quindi vedere un assegnamento come una funzione $V \to \{t, f\}$ o come un vettore $\overrightarrow{x}$ in cui in posizione x_i c'è il valore (in $\{t, f\}$) da assegnare all'i-esima variabile in V
- Per esempio $\overrightarrow{x} = (t, f, f)$ ci dice che assegnamo ad A il valore t e a B e C il valore f

VALUTAZIONE

- Data una formula ed un assegnamento possiamo valutare la formula
- Il valore di verità di una variabile proposizionale è dato direttamente dall'assegnamento
- $\neg\varphi$ è vera se la formula φ è falsa
- $\varphi \wedge \psi$ è vera se entrambe le formule che la compongono sono vere
- $\varphi \vee \psi$ è vera se almeno una delle due formule che la compongono è vera
- Diciamo che una formula φ è soddisfatta se esiste un assegnamento che la rende vera

NOTAZIONE

- Indicheremo con $\bigwedge_{i=1}^{m} \varphi_i$ la formula $\varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_m$
- Indicheremo con $\bigvee_{i=1}^{m} \varphi_i$ la formula $\varphi_1 \vee \varphi_2 \vee \ldots \vee \varphi_m$
- Per esempio
  $$\bigwedge_{i=1}^{2} \bigvee_{j=1}^{2} \varphi_{i,j} = (\varphi_{1,1} \vee \varphi_{1,2}) \wedge (\varphi_{2,1} \vee \varphi_{2,2})$$

SAT È CONTENUTO IN NP

- Data una formula φ e un assegnamento $\overrightarrow{x}$ di variabili possiamo verificare che φ è soddisfatta calcolando $\varphi(\overrightarrow{x})$, che è fattibile in tempo polinomiale

- Se vogliamo invece usare la definizione di NP con macchine non deterministiche, possiamo generare in modo non deterministico un assegnamento $\overrightarrow{x}$ di φ e verificare se $\overrightarrow{x}$ soddisfa φ e, in quel caso accettare. Questo richiede tempo polinomiale non-deterministico

SAT È NP-HARD

- Dobbiamo ora mostrare che ogni problema in NP si riduce (in tempo polinomiale) a SAT
- Dato che per ogni problema in NP esiste una NDTM che lavora in tempo polinomiale che lo decide…
- …possiamo vedere se possiamo usare una istanza di SAT per "simulare" una NDTM
- Data una NDTM M che lavora in tempo $t(n)$ e un input w dobbiamo scrivere una formula φ che è soddisfacibile se e solo se esiste una computazione accettante di M su input w

DIAGRAMMA SPAZIO-TEMPO DI UNA NDTM

Supponiamo di avere una NDTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ che lavora in tempo $t(n)$

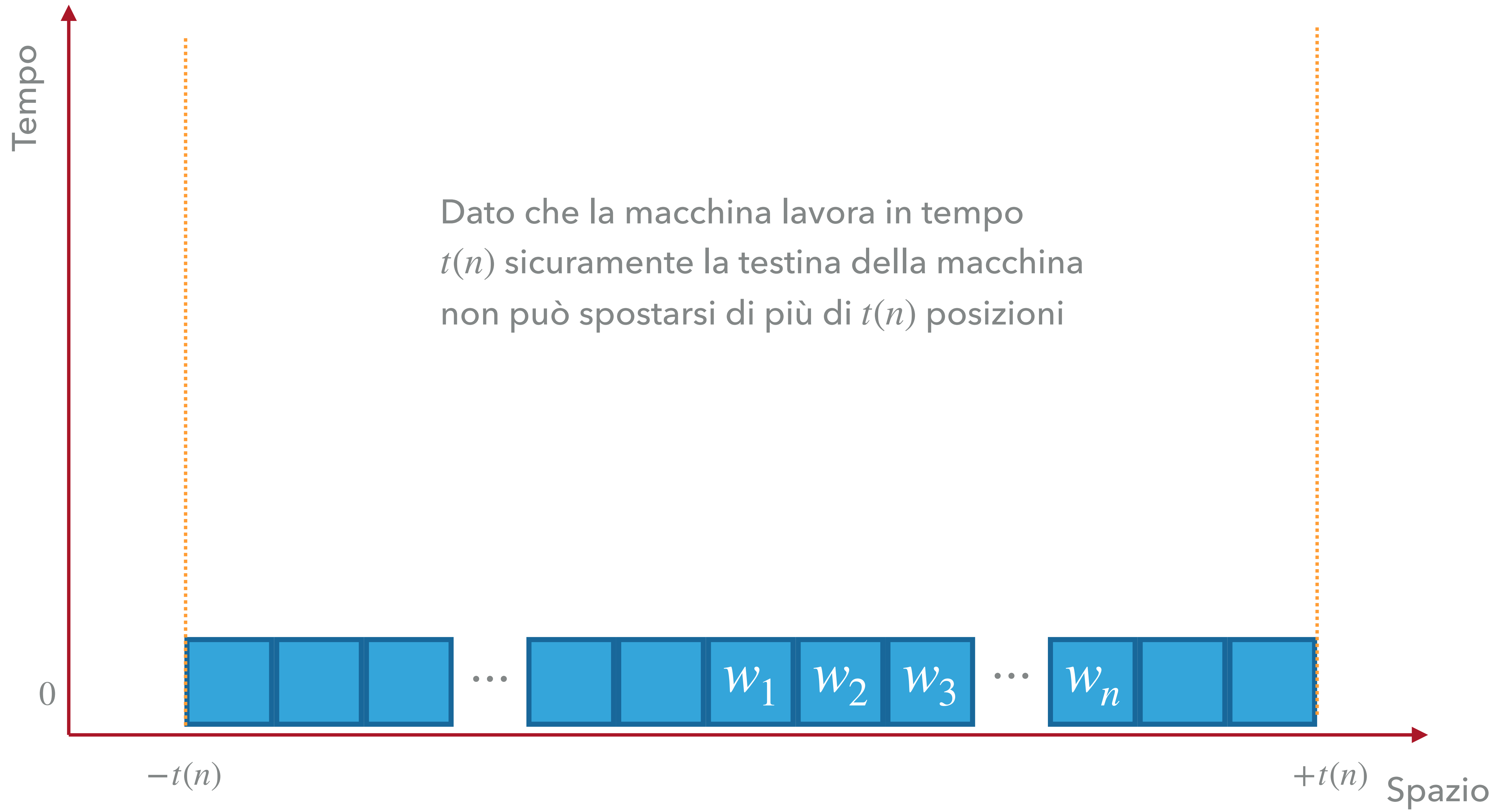
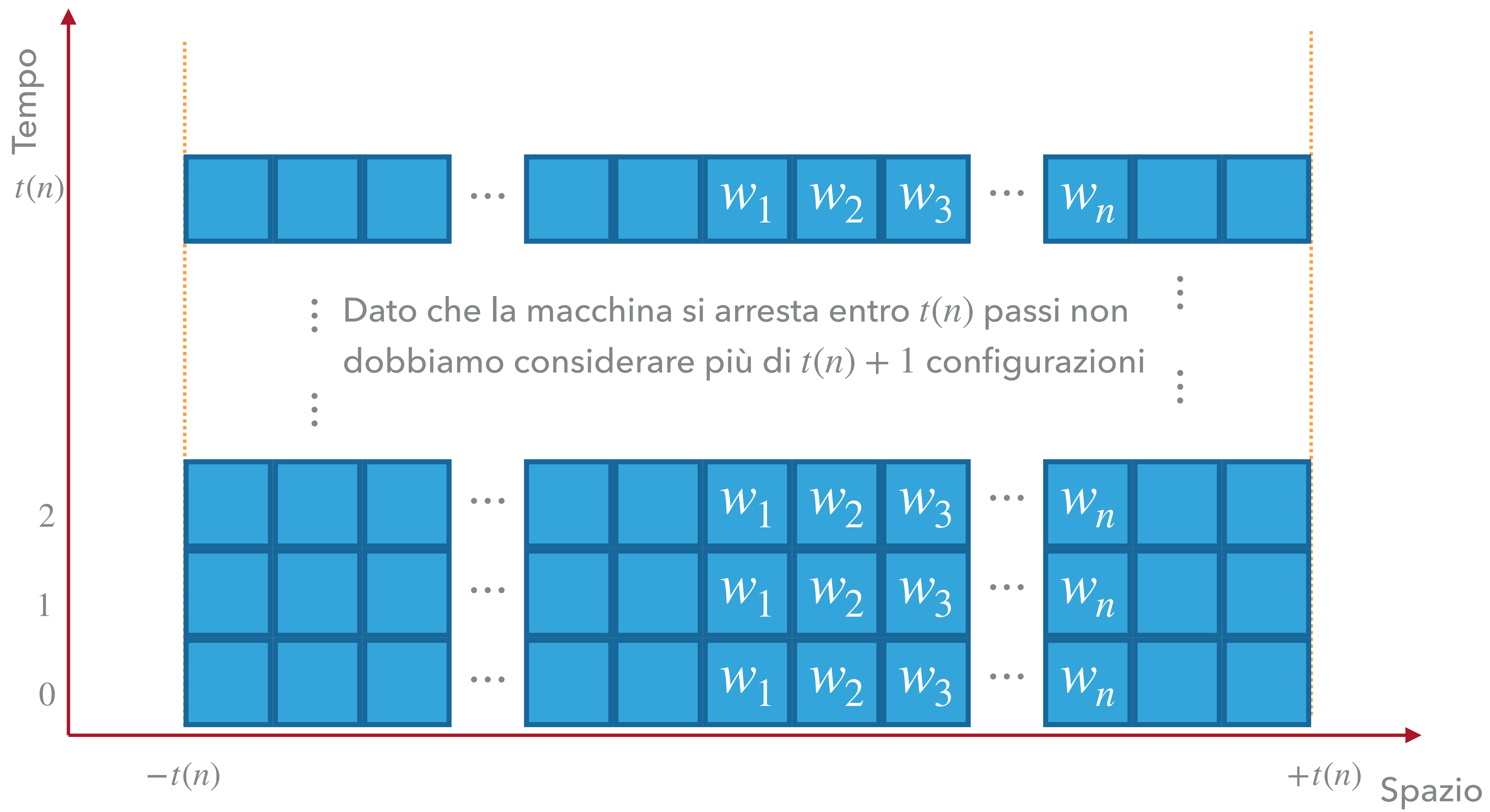


DIAGRAMMA SPAZIO-TEMPO DI UNA NDTM

Supponiamo di avere una NDTM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ che lavora in tempo $t(n)$



IDEA DELLA DIMOSTRAZIONE

- Definiamo delle variabili che ci definiscono tutte le $t(n) + 1$ configurazioni (ognuna consistente di $2t(n) + 1$ celle)
- Definiamo una formula che è vera se e solo se esiste un assegnamento delle variabili che rappresenta una computazione accettante
- Questa formula deve essere costruibile in tempo polinomiale a partire dall'input $w \in \Gamma^\star$ di lunghezza n

VARIABILI UTILIZZATE

- $c_{\sigma,i,j}$ per $\sigma \in \Gamma$, $i \in \{-t(n), \ldots, t(n)\}$, e $j \in \{0, \ldots, t(n)\}$ indica che la cella i contiene il simbolo σ al tempo j
- $p_{i,j}$ per $i \in \{-t(n), \ldots, t(n)\}$ e $j \in \{0, \ldots, t(n)\}$ indica che la testina della macchine è sulla cella i al tempo j
- $e_{q,j}$ per $q \in Q$ e $j \in \{0, \ldots, t(n)\}$ indica che la macchina si trova nello stato q al tempo j

VARIABILI UTILIZZATE

- Le variabili della forma $c_{\sigma,i,j}$ sono $|\Gamma|(t(n)+1)(2t(n)+1)$
- Le variabili della forma $p_{i,j}$ sono $(t(n)+1)(2t(n)+1)$
- Le variabili della forma $e_{q,j}$ sono $|Q|(t(n)+1)$
- Dato che $t(n)$ è polinomiale rispetto a $|w|=n$, il numero di variabili rimane polinomiale

COSA DEVE VERIFICARE LA FORMULA

- Le $t(n) + 1$ configurazioni sono valide:
  - C'è sempre esattamente un simbolo per cella
  - C'è esattamente uno stato della macchina
  - La testina è in esattamente una posizione sul nastro
- La configurazione iniziale è corretta
- L'ultima configurazione è accettante
- Le transizioni sono valide

COSA DEVE VERIFICARE LA FORMULA

- Per ognuna di queste condizioni daremo una formula
- Consideriamo tutte le formule messe in congiunzione
- Ovvero tutte queste sotto-formule devono essere vere tutte affinché la formula risultate sia vera
- Avremo quindi che la formula è soddisfacibile se:
  - Ognuna della $t(n) + 1$ configurazioni è valida, le transizioni sono valide, la computazione è accettante
  - Ovvero se la NDTM M che andiamo a modellare accettare su input w entro $t(n)$ passi

C'È ESATTAMENTE UN SIMBOLO PER CELLA

- La formula è $\bigwedge_{i=-t(n)}^{t(n)} \bigwedge_{j=0}^{t(n)} \bigvee_{\sigma\in\Gamma} \left(c_{\sigma,i,j} \wedge \bigwedge_{\sigma'\neq\sigma} \neg c_{\sigma',i,j} \right)$
- Che significa che per ogni posizione i del nastro e per ogni istante temporale j deve valere che c'è un simbolo σ nella casella j e nessun altro simbolo nella stessa casella
- Per esempio, per $i = 1$, $j = 0$ e alfabeto $\Gamma = \{a, b, \#\}$ avremo la formula:
  $(c_{a,1,0} \wedge \neg c_{b,1,0} \wedge \neg c_{\#,1,0}) \vee (c_{b,1,0} \wedge \neg c_{a,1,0} \wedge \neg c_{\#,1,0}) \vee (c_{\#,1,0} \wedge \neg c_{a,1,0} \wedge \neg c_{b,1,0})$

C'È ESATTAMENTE UNO STATO DELLA MACCHINA

- La formula è $\bigwedge_{j=0}^{t(n)} \bigvee_{q \in Q} \left(e_{q,j} \wedge \bigwedge_{q' \neq q} \neg e_{q',j} \right)$
- Che significa che in ogni istante temporale j quando lo stato della macchina è q allora non può essere in nessuno stato $q' \neq q$
- Per esempio all'istante temporale $j = 0$ per $Q = \{q, r\}$ avremmo la formula: $(e_{q,0} \wedge \neg e_{r,0}) \vee (e_{r,0} \wedge \neg e_{q,0})$

LA TESTINA È IN ESATTAMENTE UNA POSIZIONE SUL NASTRO

- La formula è $\bigwedge_{j=0}^{t(n)} \bigvee_{i=-t(n)}^{t(n)} \left(p_{i,j} \wedge \bigwedge_{i' \neq i} \neg p_{i',j} \right)$
- Che significa che in ogni istante temporale j quando la testina è in posizione i sul nastro allora non è in nessuna altra posizione $i' \neq i$
- Per esempio all'istante temporale $j = 0$ per $t(n) = 1$ avremmo la formula:
$(p_{-1,0} \wedge \neg p_{0,0} \wedge \neg p_{1,0}) \vee (p_{0,0} \wedge \neg p_{-1,0} \wedge \neg p_{1,0}) \vee (p_{1,0} \wedge \neg p_{-1,0} \wedge \neg p_{0,0})$

LA CONFIGURAZIONE INIZIALE È CORRETTA

- Mettiamo una congiunzione delle seguenti formule:
- $e_{q_0,0}$ ovvero al tempo 0 lo stato è q_0
- $p_{0,0}$ ovvero al tempo 0 la testina è sulla cella 0
- $c_{w_i,i,0}$ per $i \in \{0,\ldots,n-1\}$ con $w = w_0 w_1 \cdots w_{n-1}$. Ovvero in posizione i del nastro c'è l'i-esimo simbolo dell'input
- $c_{\#,i,0}$ per $i \in \{-p(n),\ldots,-1,n,\ldots,p(n)\}$. Ovvero, dove non c'è l'input c'è il simbolo di blank

L'ULTIMA CONFIGURAZIONE È ACCETTANTE

- Per semplificare la notazione consideriamo che, se anche la macchina si arresta prima del tempo $t(n)$ tutte le configurazioni successive al tempo di arresto non cambiano
- Quindi ci basta controllare che lo stato al tempo $t(n)$ sia accettante
- Si esprime con: $e_{q_{\text{final}},t(n)}$

LE TRANSIZIONI SONO VALIDE

- Questa è la parte più complessa
- Dobbiamo esprimere due cose:
  - La posizione sotto la testina cambia in modo compatibile con la funzione di transizione
  - Lo stato cambia in modo compatibile con la funzione di transizione
  - Nessuna delle altre celle cambia

NESSUNA DELLE ALTRE CELLE CAMBIA

- Per ogni transizione dal tempo $j-1$ al tempo j definiamo la formula

$$\bigwedge_{i=-p(n)}^{p(n)} \neg p_{i,j-1} \to \left(\bigwedge_{\sigma \in \Gamma} c_{\sigma,i,j-1} \leftrightarrow c_{\sigma,i,j} \right)$$

- Che dice "se al tempo $j-1$ la testina non era sulla cella i allora il contenuto della cella i coincide al tempo $j-1$ e j"
- Questo significa che il contenuto del nastro non cambia per le posizioni dove non c'è la testina

APPLICAZIONE DELLA FUNZIONE DI TRANSIZIONE

- Per ogni transizione dal tempo $j-1$ al tempo j definiamo la formula
$$\bigwedge_{i=-p(n)}^{p(n)} \bigwedge_{q\in Q} \bigwedge_{\sigma\in\Gamma} \Big((p_{i,j-1} \wedge e_{q,j-1} \wedge c_{\sigma,i,j-1} \rightarrow \psi_{q,\sigma,i} \Big)$$
- Dove $\psi_{q,\sigma,i}$ è definito come
$$\bigvee_{(q',\sigma',d)\in\delta(q,\sigma)} \Big(c_{\sigma',i,j} \wedge e_{q'j} \wedge p_{i+d,j} \Big)$$
dove usiamo $d = \{-1,1\}$ per i movimenti $\leftarrow$ e $\rightarrow$ della testina (ci serve indicare come varia la posizione)

VERSO LA FINE DELLA DIMOSTRAZIONE

- Se prendiamo tutte le formule definite fino ad ora e le mettiamo in congiunzione allora abbiamo che la formula è soddisfacibile solo se rispetta tutte le condizioni elencate
- Ognuna delle sotto-formule ha dimensione polinomiale rispetto a n e anche la formula finale ha dimensione polinomiale rispetto ad n e le operazioni per costruirla sono eseguibili in tempo polinomiale
- Abbiamo quindi mostrato che SAT è NP-completo

CONSEGUENZE

- Se esiste una soluzione efficiente (in tempo polinomiale deterministico) per SAT allora esiste per **ogni** problema in NP. E questo mostrerebbe $\mathrm{P} = \mathrm{NP}$
- Chiaramente questo non è stato mostrato
- SAT non è l'unico problema NP-completo noto, ma il primo trovato

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*"