

Dispense per il corso di

TEORIA DELLA COMPUTABILITÀ

- Modulo di Complessità -

A cura del Dr. Salvatore Cristofaro
cristofaro@dmi.unict.it

Libri di testo di riferimento:

- (1) *Computability, Complexity and Languages*, Martin D. Davis, Ron Sigal, Elaine J. Weyuker
- (2) *Introduction to the Theory of Computation*, Michael Sipser
- (3) *Computers and Intractability*, Michael R. Garey, David S. Johnson

1 Notazioni asintotiche

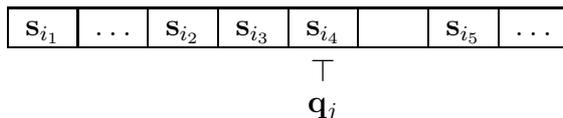
Definizione 1. Siano $f, g : \mathbb{N} \rightarrow \mathbb{R}$ funzioni.

- (1) $f(n) = O(g(n))$ se esistono $c \in]0, +\infty[$ e $\nu \in \mathbb{N}$ tali che $0 \leq f(n) \leq c \cdot g(n)$, per ogni $n \geq \nu$.
- (2) $f(n) = \Omega(g(n))$ se esistono $c \in]0, +\infty[$ e $\nu \in \mathbb{N}$ tali che $0 \leq c \cdot g(n) \leq f(n)$, per ogni $n \geq \nu$.
- (3) $f(n) = \Theta(g(n))$ se esistono $c_1, c_2 \in]0, +\infty[$ e $\nu \in \mathbb{N}$ tali che $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$, per ogni $n \geq \nu$.

2 Macchine di Turing

Una *Macchina di Turing* è un dispositivo computazionale che effettua operazioni su di un nastro lineare suddiviso in caselle (squares). Ogni casella può contenere al suo interno uno qualsiasi dei simboli di un preassegnato insieme finito $\{s_0, s_1, \dots, s_n\}$ di possibili *simboli di nastro*, oppure una casella può essere vuota. In ogni istante di tempo il nastro ha lunghezza finita, cioè è composto da un numero finito di caselle, però può essere espanso “incollando” nuove caselle vuote alle sue estremità (così il nastro è potenzialmente infinito verso sinistra e verso destra). La macchina interagisce con il nastro tramite una testina meccanica mobile che può leggere e modificare il contenuto delle singole caselle (una casella alla volta). La testina meccanica è collegata ad un *sistema di controllo (a stati finiti)* che determina il comportamento della macchina, cioè le operazioni che vengono eseguite sul nastro. Il sistema di controllo può assumere (o può trovarsi in) un numero finito di possibili *stati interni* q_0, q_1, \dots, q_m , prefissati, dove q_0 è lo *stato iniziale*.

In ogni dato istante di tempo, la testina meccanica si trova posizionata su una particolare casella del nastro (la *casella scandita*) e il sistema di controllo si trova in un particolare stato interno (lo *stato attivo*).



La testina meccanica comunica al sistema controllo il contenuto della casella scandita e il sistema di controllo, a seconda dello stato attivo in cui si trova, fa eseguire alla testina una ben determinata operazione e successivamente assume (transisce in) un nuovo stato interno (possibilmente lo stesso di prima), oppure il sistema di controllo determina che la macchina si deve fermare. L'esecuzione dell'operazione della testina meccanica e la successiva transizione di stato del sistema di controllo è un atto unico ed indivisibile e costituisce *un'azione (atomica)* della macchina di Turing.

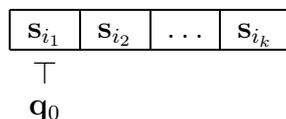
Le operazioni che può eseguire la testina meccanica sono di quattro tipi:

- (1) (*cancellazione*) cancellare il contenuto della casella scandita rendendola vuota;
- (2) (*stampa*) stampare all'interno della casella scandita un simbolo di nastro;
- (3) (*spostamento a sinistra*) spostarsi sulla casella che si trova immediatamente a sinistra della casella scandita. Se prima dell'esecuzione dell'operazione, la casella scandita si trova all'estremità sinistra del nastro, una nuova casella vuota viene aggiunta a questa estremità.

- (4) (*spostamento a destra*) spostarsi sulla casella che si trova immediatamente a destra della casella scandita. Se prima dell'esecuzione dell'operazione, la casella scandita si trova all'estremità destra del nastro, una nuova casella vuota verrà aggiunta a questa estremità.

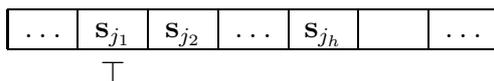
Il sistema di controllo determina la particolare azione che deve compiere la macchina di Turing, oppure che la macchina si deve fermare, consultando un insieme finito di *istruzioni* memorizzate al suo interno (un programma). Se indichiamo il "contenuto di una casella vuota" con il simbolo \square (*blank*) e con \uparrow e \uparrow le operazioni che consistono nel muovere la testina meccanica di una casella a sinistra e a destra, rispettivamente, ogni istruzione può essere descritta da una quadrupla (q, t, o, p) dove q e p sono stati interni, $t \in \{s_0, s_1, \dots, s_n\} \cup \{\square\}$ e $o \in \{s_0, s_1, \dots, s_n\} \cup \{\square\} \cup \{\uparrow, \uparrow\}$. L'interpretazione è la seguente: quando (e se) il sistema di controllo si trova nello stato attivo q e il contenuto della casella scandita è t , la testina meccanica esegue l'operazione rappresentata da o e successivamente il sistema di controllo transisce nello stato interno p . $o = \square$ significa che la testina meccanica deve cancellare il contenuto della casella scandita (operazione 1); $o = s_j$ significa che la testina meccanica deve stampare sulla casella scandita il simbolo s_j (operazione 2); $o = \uparrow$ significa che la testina meccanica si deve spostare a sinistra della casella scandita (operazione 3); $o = \uparrow$ significa che la testina meccanica si deve spostare a destra della casella scandita (operazione 4). Se, in corrispondenza di un certo stato attivo q e di un certo contenuto t della casella scandita non c'è nessuna istruzione nel sistema di controllo che inizia con la coppia (q, t) , la macchina si ferma.

Inizialmente il nastro è composto da un numero finito di caselle ciascuna delle quali contiene un simbolo di nastro; il sistema di controllo si trova nello stato iniziale q_0 e la testina meccanica è posizionata sulla casella più a sinistra; questa è la *configurazione iniziale* della macchina:



La macchina di Turing compirà una successione di azioni sotto la guida del sistema di controllo, passando attraverso varie configurazioni, fino a quando eventualmente si fermerà.

La stringa $s_{i_1}s_{i_2}\dots s_{i_k}$ ottenuta concatenando i simboli contenuti nelle caselle del nastro, da sinistra verso destra, nella configurazione iniziale è l'input fornito alla macchina. Quando (e se) la macchina si ferma, raggiungendo una *configurazione terminale*, l'output prodotto sarà costituito dalla stringa $s_{j_1}s_{j_2}\dots s_{j_h}$ ottenuta concatenando i simboli contenuti nelle caselle del nastro a partire dalla casella scandita fino alla prima casella verso destra seguita immediatamente da una casella vuota, ovvero fino all'ultima casella del nastro se alla destra della casella scandita non ci sono caselle vuote. (Se la casella scandita è vuota, l'output è la *stringa vuota*).



3 Formalizzazione delle Macchine di Turing

Siano fissati:

(1) Un insieme \mathbf{Q} di SIMBOLI DI STATO:

$$\mathbf{q}_0 \quad \mathbf{q}_1 \quad \mathbf{q}_2 \quad \mathbf{q}_3 \quad \dots$$

(2) Un insieme \mathbf{S} di SIMBOLI DI NASTRO:

$$\mathbf{s}_0 \quad \mathbf{s}_1 \quad \mathbf{s}_2 \quad \mathbf{s}_3 \quad \dots$$

(3) Il SIMBOLO DI BLANK:

$$\square$$

(4) I SIMBOLI DI SPOSTAMENTO A SINISTRA E DESTRA:

$$\uparrow \quad \uparrow'$$

Definizioni preliminari

- (1) Per ogni sottoinsieme S di $\mathbf{Q} \cup \mathbf{S} \cup \{\square\}$ indichiamo con S^* l'insieme delle STRINGHE su S , compresa la STRINGA VUOTA ε .
- (2) La LUNGHEZZA DI UNA STRINGA X è indicata con $|X|$.
- (3) Una n -STRINGA, dove $n \in \mathbb{N}$, è una stringa di lunghezza n .
- (4) Un ALFABETO è un sottoinsieme finito di \mathbf{S} .
- (5) La CARDINALITÀ DI UN ALFABETO Σ è indicata con $|\Sigma|$.
- (6) Un LINGUAGGIO è un sottoinsieme di Σ^* , per qualche alfabeto Σ .
- (7) L'ALFABETO DI UN LINGUAGGIO \mathcal{L} è il "più piccolo" alfabeto $\Sigma_{\mathcal{L}}$ tale che $\mathcal{L} \subseteq \Sigma_{\mathcal{L}}^*$.
- (8) La FUNZIONE CARATTERISTICA DI UN LINGUAGGIO \mathcal{L} è l'insieme $C_{\mathcal{L}}$ delle coppie ordinate (X, ε) dove $X \in \mathcal{L}$.
- (9) Il COMPLEMENTARE DI UN LINGUAGGIO \mathcal{L} è il linguaggio $\mathcal{C}(\mathcal{L}) = \Sigma_{\mathcal{L}}^* \setminus \mathcal{L}$.
- (10) Una RELAZIONE (DI STRINGHE) è un sottoinsieme del prodotto cartesiano $\Sigma^* \times \Sigma^*$, per qualche alfabeto Σ .
- (11) L'ALFABETO DI UNA RELAZIONE ρ è il più piccolo alfabeto Σ_{ρ} tale che $\rho \subseteq \Sigma_{\rho}^* \times \Sigma_{\rho}^*$.

Definizione 2. Una ISTRUZIONE (O QUADRUPLA) è una 4-stringa $qt\text{op}$ dove $q, p \in \mathbf{Q}$, $t \in \mathbf{S} \cup \{\square\}$ e $o \in \mathbf{S} \cup \{\square\} \cup \{\uparrow, \uparrow'\}$.

Ogni 1-stringa o , dove $o \in \mathbf{S} \cup \{\square\} \cup \{\uparrow, \uparrow'\}$, è chiamata UN'OPERAZIONE ed ogni 2-stringa op , dove $o \in \mathbf{S} \cup \{\square\} \cup \{\uparrow, \uparrow'\}$ e $p \in \mathbf{Q}$, è chiamata un'AZIONE.

Una CONFIGURAZIONE (o DESCRIZIONE ISTANTANEA) è una stringa γ della forma $XqtY$ dove $t \in \mathbf{S} \cup \{\square\}$, $q \in \mathbf{Q}$ e $X, Y \in (\mathbf{S} \cup \{\square\})^*$.

Data una stringa $Z \in \mathbf{S}^*$, la CONFIGURAZIONE INIZIALE (O DI INPUT) corrispondente a Z è la configurazione $\text{Start}(Z)$ definita come segue:

$$\text{Start}(Z) = \begin{cases} \mathbf{q}_0 \square, & \text{se } Z = \varepsilon \\ \mathbf{q}_0 Z, & \text{altrimenti.} \end{cases}$$

Una CONFIGURAZIONE γ È INIZIALE se $\gamma = \text{Start}(Z)$, per qualche stringa $Z \in \mathbf{S}^*$. ■

Esempio 4. La TM $\{\mathbf{q}_0 \square \mathbf{s}_0 \mathbf{q}_1, \mathbf{q}_0 \square \uparrow \mathbf{q}_2, \mathbf{q}_1 \mathbf{s}_0 \uparrow \mathbf{q}_0\}$ è nondeterministica dato che le istruzioni $\mathbf{q}_0 \square \mathbf{s}_0 \mathbf{q}_1$ e $\mathbf{q}_0 \square \uparrow \mathbf{q}_2$ cominciano con la stessa coppia stato-simbolo (\mathbf{q}_0, \square) .
La TM $\{\mathbf{q}_0 \square \mathbf{s}_0 \mathbf{q}_1, \mathbf{q}_0 \mathbf{s}_1 \uparrow \mathbf{q}_2, \mathbf{q}_1 \mathbf{s}_0 \uparrow \mathbf{q}_0\}$ è invece deterministica. ■

Definizione 5. Sia \mathfrak{M} una TM.

L'ALFABETO DI \mathfrak{M} è l'insieme $\mathbf{S}(\mathfrak{M})$ dei simboli di nastro che occorrono nelle istruzioni di \mathfrak{M} . L'insieme dei simboli di stato che occorrono nelle istruzioni di \mathfrak{M} (gli stati di \mathfrak{M}) è indicato con $\mathbf{Q}(\mathfrak{M})$. Indichiamo con $\mathbf{F}(\mathfrak{M})$ l'insieme delle coppie $(q, t) \in \mathbf{Q}(\mathfrak{M}) \times (\mathbf{S}(\mathfrak{M}) \cup \{\square\})$ tali che nessuna istruzione di \mathfrak{M} inizia con qt .

Una CONFIGURAZIONE $XqtY$ È TERMINALE (RISP. NON-TERMINALE) RISPETTO AD \mathfrak{M} , se nessuna (risp. qualche) istruzione di \mathfrak{M} inizia con qt .

Una CONFIGURAZIONE γ'' CONSEGUE (IMMEDIATAMENTE) DA UNA CONFIGURAZIONE $\gamma' = XqtY$ MEDIANTE L'ESECUZIONE DI UNA ISTRUZIONE DI \mathfrak{M} , e in tal caso si scrive $\gamma' \vdash_{\mathfrak{M}} \gamma''$, se esiste un'azione $A = op$ tale che $\gamma'' = A(\gamma')$ ed \mathfrak{M} contiene l'istruzione $qtop$.

Un SEGMENTO DI COMPUTAZIONE DI \mathfrak{M} è una sequenza finita $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ di configurazioni, dove $n \geq 0$, tali che $\gamma_i \vdash_{\mathfrak{M}} \gamma_{i+1}$, per ogni $i = 0, 1, \dots, n-1$. La LUNGHEZZA di Γ è $|\Gamma| = n$.

Per configurazioni γ' e γ'' ed un $n \in \mathbb{N}$ scriviamo $\gamma' \vdash^n_{\mathfrak{M}} \gamma''$ se esiste un segmento di computazione $(\gamma_0, \gamma_1, \dots, \gamma_n)$ di \mathfrak{M} (di lunghezza n) dove $\gamma_0 = \gamma'$ e $\gamma_n = \gamma''$. Scriviamo anche $\gamma' \vdash^*_{\mathfrak{M}} \gamma''$ se esiste un $n \in \mathbb{N}$ tale che $\gamma' \vdash^n_{\mathfrak{M}} \gamma''$.

Una COMPUTAZIONE di \mathfrak{M} è un segmento di computazione $(\gamma_0, \gamma_1, \dots, \gamma_n)$ di \mathfrak{M} dove γ_0 è una configurazione iniziale e γ_n è una configurazione terminale rispetto a \mathfrak{M} .

Sia $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ una computazione di \mathfrak{M} .

(1) L'INPUT DI Γ è la stringa $\mathbf{Input}(\Gamma) = \langle \gamma_0 \rangle$;

(2) L'OUTPUT DI Γ è la stringa $\mathbf{Output}(\Gamma) = \langle \gamma_n \rangle$.

Una COMPUTAZIONE PROPRIA DI \mathfrak{M} è una computazione Γ di \mathfrak{M} tale che $\mathbf{Input}(\Gamma) \in \mathbf{S}(\mathfrak{M})^*$.

Una CONFIGURAZIONE DI \mathfrak{M} (O \mathfrak{M} -CONFIGURAZIONE) è una configurazione $XqtY$ dove X e Y sono stringhe su $\mathbf{S}(\mathfrak{M}) \cup \{\square\}$, $t \in \mathbf{S}(\mathfrak{M}) \cup \{\square\}$ e $q \in \mathbf{Q}(\mathfrak{M})$. ■

Si osservi che se \mathfrak{M} è una DTM allora, per ogni stringa $X \in \mathbf{S}^*$ esiste al più una sola computazione di \mathfrak{M} con input X . Se invece \mathfrak{M} è una NDTM, possono esistere anche due o più computazioni distinte di \mathfrak{M} con lo stesso input X . Si osservi inoltre che ogni configurazione contenuta in una computazione propria di una TM \mathfrak{M} è una \mathfrak{M} -configurazione.

Esempio 5. Sia $\mathfrak{M} = \{\mathbf{q}_0 \mathbf{s}_0 \uparrow \mathbf{q}_0, \mathbf{q}_0 \mathbf{s}_1 \mathbf{s}_0 \mathbf{q}_0, \mathbf{q}_0 \square \uparrow \mathbf{q}_1, \mathbf{q}_1 \mathbf{s}_0 \uparrow \mathbf{q}_1, \mathbf{q}_1 \square \uparrow \mathbf{q}_2\}$.

Allora:

(1) $\mathbf{S}(\mathfrak{M}) = \{\mathbf{s}_0, \mathbf{s}_1\}$;

(2) $\mathbf{Q}(\mathfrak{M}) = \{\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2\}$;

(3) $\mathbf{F}(\mathfrak{M}) = \{(\mathbf{q}_1, \mathbf{s}_1), (\mathbf{q}_2, \square), (\mathbf{q}_2, \mathbf{s}_0), (\mathbf{q}_2, \mathbf{s}_1)\}$;

(4) La seguente sequenza di configurazioni è una computazione di \mathfrak{M} con input $\mathbf{s}_1 \mathbf{s}_0 \mathbf{s}_1 \mathbf{s}_1 \mathbf{s}_0$ e

output $s_0s_0s_0s_0s_0$ di lunghezza 15:

$q_0s_1s_0s_1s_1s_0$
 $q_0s_0s_0s_1s_1s_0$
 $s_0q_0s_0s_1s_1s_0$
 $s_0s_0q_0s_1s_1s_0$
 $s_0s_0q_0s_0s_1s_0$
 $s_0s_0s_0q_0s_1s_0$
 $s_0s_0s_0q_0s_0s_0$
 $s_0s_0s_0s_0q_0s_0$
 $s_0s_0s_0s_0s_0q_0$
 $s_0s_0s_0s_0s_0q_0$
 $s_0s_0s_0s_0q_1s_0$
 $s_0s_0s_0q_1s_0s_0$
 $s_0s_0q_1s_0s_0s_0$
 $s_0q_1s_0s_0s_0s_0$
 $q_1s_0s_0s_0s_0s_0$
 q_1 $s_0s_0s_0s_0s_0$
 $q_2s_0s_0s_0s_0s_0$

■

Definizione 6. Siano \mathfrak{M} una TM e $X, Y \in \mathbf{S}^*$ stringhe.

- (1) \mathfrak{M} CON INPUT X PRODUCE IN OUTPUT Y , e si scrive $\mathfrak{M}(X) \downarrow Y$, se esiste una computazione Γ di \mathfrak{M} tale che $\text{Input}(\Gamma) = X$ e $\text{Output}(\Gamma) = Y$;
- (2) \mathfrak{M} CON INPUT X PRODUCE IN OUTPUT Y IN n -PASSI, O IN TEMPO n , dove $n \in \mathbb{N}$, e si scrive $\mathfrak{M}(X) \downarrow^{(n)} Y$, se esiste una computazione Γ di \mathfrak{M} di lunghezza n tale che $\text{Input}(\Gamma) = X$ e $\text{Output}(\Gamma) = Y$. Scriviamo $\mathfrak{M}(X) \downarrow^{(\leq n)} Y$ se $\mathfrak{M}(X) \downarrow^{(m)} Y$, per qualche $m \leq n$.
- (3) \mathfrak{M} ACCETTA (O RICONOSCE) X , e si scrive $\mathfrak{M}(X) \downarrow$, se $\mathfrak{M}(X) \downarrow Z$, per qualche stringa $Z \in \mathbf{S}^*$. Scriviamo $\mathfrak{M}(X) \uparrow$ se $\mathfrak{M}(X) \downarrow$ non vale.
- (4) \mathfrak{M} ACCETTA (O RICONOSCE) X IN n -PASSI, O IN TEMPO n , dove $n \in \mathbb{N}$, e si scrive $\mathfrak{M}(X) \downarrow^{(n)}$, se $\mathfrak{M}(X) \downarrow^{(n)} Z$, per qualche stringa $Z \in \mathbf{S}^*$. Scriviamo $\mathfrak{M}(X) \downarrow^{(\leq n)}$ se $\mathfrak{M}(X) \downarrow^{(m)}$, per qualche $m \leq n$. ■

Esempio 6. Riferendoci all'esempio precedente:

- (1) $\mathfrak{M}(s_1s_0s_1s_1s_0) \downarrow s_0s_0s_0s_0s_0$;
- (2) $\mathfrak{M}(s_1s_0s_1s_1s_0) \downarrow^{(15)} s_0s_0s_0s_0s_0$;
- (3) $\mathfrak{M}(s_1s_0s_1s_1s_0) \downarrow^{(\leq n)} s_0s_0s_0s_0s_0$, per ogni $n \geq 15$;
- (4) $\mathfrak{M}(s_1s_0s_1s_1s_0) \downarrow$;
- (5) $\mathfrak{M}(s_1s_0s_1s_1s_0) \downarrow^{(15)}$;
- (6) $\mathfrak{M}(s_1s_0s_1s_1s_0) \downarrow^{(\leq n)}$, per ogni $n \geq 15$. ■

Definizione 7. Sia \mathfrak{M} una TM.

- (1) La RELAZIONE COMPUTATA DA \mathfrak{M} è l'insieme $\varrho(\mathfrak{M})$ delle coppie (X, Y) di stringhe su $\mathbf{S}(\mathfrak{M})$ tali che $\mathfrak{M}(X) \downarrow Y$. (Si osservi che se \mathfrak{M} è una DTM allora $\varrho(\mathfrak{M})$ è una funzione: LA FUNZIONE COMPUTATA DA \mathfrak{M} .)
- (2) Il LINGUAGGIO ACCETTATO (O RICONOSCIUTO) da \mathfrak{M} è l'insieme $\mathcal{L}(\mathfrak{M})$ delle stringhe su $\mathbf{S}(\mathfrak{M})$ che sono accettate da \mathfrak{M} .
- (3) Una RELAZIONE ρ DI STRINGHE È COMPUTATA DA \mathfrak{M} se e solo se $\rho = \varrho(\mathfrak{M})$.
- (4) Un LINGUAGGIO \mathcal{S} È ACCETTATO DA \mathfrak{M} se e solo se $\mathcal{S} = \mathcal{L}(\mathfrak{M})$. ■

Definizione 8. Siano \mathcal{S} un linguaggio ed f una funzione.

- (1) \mathcal{S} È (TURING) SEMIDECIDIBILE se esiste una TM \mathfrak{M} tale che \mathcal{S} è accettato da \mathfrak{M} .
- (2) f È (TURING) COMPUTABILE se esiste una DTM \mathfrak{M} tale che f è computata da \mathfrak{M} .
- (3) \mathcal{S} È DECISO DA UNA DTM \mathfrak{D} se:
 - (3.a) $\mathcal{L}(\mathfrak{D}) = \Sigma_{\mathcal{S}}^*$ e
 - (3.b) $\mathcal{S} = \{X \in \mathbf{S}(\mathfrak{D})^* : \mathfrak{D}(X) \downarrow \varepsilon\}$.
- (4) \mathcal{S} È (TURING) DECIDIBILE se esiste una DTM \mathfrak{D} tale che \mathcal{S} è deciso da \mathfrak{D} . ■

Definizione 9. Sia \mathfrak{M} una TM.

- (1) Per ogni stringa $X \in \mathcal{L}(\mathfrak{M})$, la COMPLESSITÀ TEMPORALE DI \mathfrak{M} SULL'INPUT X è la quantità $\widehat{T}_{\mathfrak{M}}(X)$ definita come segue:

$$\widehat{T}_{\mathfrak{M}}(X) = \min\{n \in \mathbb{N} : \mathfrak{M}(X) \downarrow^{(n)}\}.$$

(Cioè $\widehat{T}_{\mathfrak{M}}(X)$ è la lunghezza della più corta computazione di \mathfrak{M} con input X .)

- (2) Il RUNNING TIME DI \mathfrak{M} è la funzione $T_{\mathfrak{M}} : \mathbb{N} \rightarrow \mathbb{N} \cup \{-1\}$ definita come segue:

$$T_{\mathfrak{M}}(n) = \begin{cases} \max\{\widehat{T}_{\mathfrak{M}}(X) : X \in \mathcal{L}(\mathfrak{M}) \text{ AND } |X| = n\}, & \text{se } \{X \in \mathcal{L}(\mathfrak{M}) : |X| = n\} \neq \emptyset \\ -1, & \text{altrimenti.} \end{cases}$$

- (3) Diciamo che \mathfrak{M} È REGOLARE (RISPETTO ALLA COMPLESSITÀ TEMPORALE) se:

$$\text{per ogni } Y, Z \in \mathcal{L}(\mathfrak{M}), \text{ se } |Y| = |Z| \text{ allora } \widehat{T}_{\mathfrak{M}}(Y) = \widehat{T}_{\mathfrak{M}}(Z).$$

(Cioè, \mathfrak{M} è regolare se per ogni stringa $X \in \mathcal{L}(\mathfrak{M})$, la complessità temporale di \mathfrak{M} con input X dipende solo dalla lunghezza $|X|$ di X .) ■

Nel seguito useremo la seguente notazione: per ogni sottoinsieme non vuoto S di $\mathbf{S} \cup \{\square\}$, ogni stato q e ogni azione A , indicheremo con qSA l'insieme di istruzioni $\{qtA : t \in S\}$.

Esempio 7. La seguente TM \mathfrak{M} accetta il linguaggio \mathcal{L} costituito da tutte le stringhe palindrome sull'alfabeto $\{a, b\}$ ($a = \mathbf{s}_i, b = \mathbf{s}_j, i \neq j$):

$$\begin{aligned}
& \mathbf{q}_0 a \square \mathbf{q}_1 \\
& \mathbf{q}_0 b \square \mathbf{q}_2 \\
& \mathbf{q}_1 \square \dot{\rightarrow} \mathbf{q}_3 \\
& \mathbf{q}_2 \square \dot{\rightarrow} \mathbf{q}_4 \\
& \mathbf{q}_3 \{a, b\} \dot{\rightarrow} \mathbf{q}_3 \\
& \mathbf{q}_4 \{a, b\} \dot{\rightarrow} \mathbf{q}_4 \\
& \mathbf{q}_3 \square \dot{\leftarrow} \mathbf{q}_5 \\
& \mathbf{q}_4 \square \dot{\leftarrow} \mathbf{q}_6 \\
& \mathbf{q}_5 b b \mathbf{q}_5 \\
& \mathbf{q}_6 a a \mathbf{q}_6 \\
& \mathbf{q}_5 a \square \mathbf{q}_5 \\
& \mathbf{q}_6 b \square \mathbf{q}_6 \\
& \mathbf{q}_5 \square \dot{\leftarrow} \mathbf{q}_7 \\
& \mathbf{q}_6 \square \dot{\leftarrow} \mathbf{q}_7 \\
& \mathbf{q}_7 \{a, b\} \dot{\leftarrow} \mathbf{q}_7 \\
& \mathbf{q}_7 \square \dot{\rightarrow} \mathbf{q}_0
\end{aligned}$$

Osserviamo che \mathfrak{M} è regolare e che il suo running time $T_{\mathfrak{M}}$ soddisfa la seguente equazione di ricorrenza:

$$T_{\mathfrak{M}}(n) = \begin{cases} 0, & \text{se } n = 0 \\ 5, & \text{se } n = 1 \\ T_{\mathfrak{M}}(n-2) + 2n + 3, & \text{se } n \geq 2, \end{cases}$$

per ogni $n \in \mathbb{N}$.

Espandendo l'equazione si ottiene che

$$T_{\mathfrak{M}}(n) = T_{\mathfrak{M}}(n-2k) + 3k + 2 \left[\sum_{i=0}^{k-1} (n-2i) \right] = T_{\mathfrak{M}}(n-2k) + 2nk - 2k^2 + 5k,$$

per ogni $k \leq n/2$, e quindi:

$$T_{\mathfrak{M}}(n) = \begin{cases} \frac{1}{2}n^2 + \frac{5}{2}n, & \text{se } n \text{ è pari} \\ \frac{1}{2}n^2 + \frac{5}{2}n + 2, & \text{altrimenti.} \end{cases}$$

Pertanto $T_{\mathfrak{M}}(n) = \Theta(n^2)$; così il running time di \mathfrak{M} è quadratico. ■

Esempio 8. La seguente DTM computa la funzione $f : \{a, b\}^* \rightarrow \{a, b\}^*$ ($a = \mathbf{s}_i, b = \mathbf{s}_j$) tale che, per ogni stringa $X \in \{a, b\}^*$, $f(X)$ è la stringa ottenuta da X eliminando in essa

ogni eventuale occorrenza del simbolo a :

$$\begin{aligned}
& \mathbf{q}_0\{a, b\} \vdash \mathbf{q}_0 \\
& \mathbf{q}_0 * * \mathbf{q}_0 \\
& \mathbf{q}_0 \square * \mathbf{q}_1 \\
& \mathbf{q}_1\{a, b, *\} \frown \mathbf{q}_1 \\
& \mathbf{q}_1 \square \vdash \mathbf{q}_2 \\
& \mathbf{q}_2 a \square \mathbf{q}_3 \\
& \mathbf{q}_3 \square \vdash \mathbf{q}_2 \\
& \mathbf{q}_2 b \square \mathbf{q}_4 \\
& \mathbf{q}_4 \square \vdash \mathbf{q}_5 \\
& \mathbf{q}_5\{a, b, *\} \vdash \mathbf{q}_5 \\
& \mathbf{q}_5 \square b \mathbf{q}_6 \\
& \mathbf{q}_6\{a, b, *\} \frown \mathbf{q}_6 \\
& \mathbf{q}_6 \square \vdash \mathbf{q}_2 \\
& \mathbf{q}_2 * \square \mathbf{q}_2 \\
& \mathbf{q}_2 \square \vdash \mathbf{q}_7
\end{aligned}$$

(Si osservi che la macchina utilizza un simbolo ausiliario $* = \mathbf{s}_k$ diverso da a e da b). ■

Esempio 9. La seguente DTM computa la funzione $f : \{a, b\}^* \rightarrow \{a, b\}^*$ ($a = \mathbf{s}_i$, $b = \mathbf{s}_j$) tale che, per ogni stringa $X \in \{a, b\}^*$, $f(X)$ è il “reversal” di X , cioè X scritta al rovescio:

$$\begin{aligned}
& \mathbf{q}_0\{a, b\} \vdash \mathbf{q}_0 \\
& \mathbf{q}_0 * * \mathbf{q}_0 \\
& \mathbf{q}_0 \square * \mathbf{q}_1 \\
& \mathbf{q}_1 * \frown \mathbf{q}_1 \\
& \mathbf{q}_1 a * \mathbf{q}_2 \\
& \mathbf{q}_1 b * \mathbf{q}_3 \\
& \mathbf{q}_2\{a, b, *\} \vdash \mathbf{q}_2 \\
& \mathbf{q}_3\{a, b, *\} \vdash \mathbf{q}_3 \\
& \mathbf{q}_2 \square a \mathbf{q}_4 \\
& \mathbf{q}_3 \square b \mathbf{q}_4 \\
& \mathbf{q}_4\{a, b\} \frown \mathbf{q}_4 \\
& \mathbf{q}_4 * \frown \mathbf{q}_1 \\
& \mathbf{q}_1 \square \vdash \mathbf{q}_5 \\
& \mathbf{q}_5 * \square \mathbf{q}_6 \\
& \mathbf{q}_6 \square \vdash \mathbf{q}_5
\end{aligned}$$

(Si osservi che la macchina utilizza un simbolo ausiliario $* = \mathbf{s}_k$ diverso da a e da b). ■

4 Funzioni numeriche (Turing) computabili

Di seguito indichiamo i simboli \mathbf{s}_0 ed \mathbf{s}_1 con $\mathbf{0}$ e $\mathbf{1}$, rispettivamente.

Dato un $n \in \mathbb{N}$ denotiamo con $\lceil n \rceil$ la stringa di lunghezza n formata concatenando n copie del simbolo $\mathbf{1}$. (Es. $\lceil 0 \rceil = \varepsilon$, $\lceil 1 \rceil = \mathbf{1}$, $\lceil 2 \rceil = \mathbf{11}$, ...)

Sia f una funzione parziale su \mathbb{N} di arità k , dove $k \geq 1$, e sia D il dominio di f . Indichiamo con $\lceil f \rceil$ la funzione tale che:

(1) il dominio di $\ulcorner f \urcorner$ è l'insieme di stringhe

$$\{\ulcorner n_1 \urcorner \mathbf{0} \ulcorner n_2 \urcorner \mathbf{0} \cdots \mathbf{0} \ulcorner n_k \urcorner : (n_1, n_2, \dots, n_k) \in D\};$$

(2) per ogni $(n_1, n_2, \dots, n_k) \in D$,

$$\ulcorner f \urcorner(\ulcorner n_1 \urcorner \mathbf{0} \ulcorner n_2 \urcorner \mathbf{0} \cdots \mathbf{0} \ulcorner n_k \urcorner) = \ulcorner f(n_1, n_2, \dots, n_k) \urcorner.$$

Diciamo che f è (Turing) computabile se lo è la funzione $\ulcorner f \urcorner$.

Esempio 10. Siano $f : \mathbb{N} \rightarrow \mathbb{N}$ e $g : \mathbb{N}^2 \rightarrow \mathbb{N}$ definite da

$$f(n) = \begin{cases} \frac{n}{2}, & \text{se } n \text{ è pari} \\ \uparrow, & \text{altrimenti} \end{cases} \quad \text{e} \quad g(n, m) = \begin{cases} n \times m, & \text{se } n + m = 3 \\ \uparrow, & \text{altrimenti.} \end{cases}$$

Il dominio di $\ulcorner f \urcorner$ è composto dalle stringhe ε , **11**, **1111**, **111111**, ..., e si ha:

$$\begin{aligned} \ulcorner f \urcorner(\varepsilon) &= \varepsilon \\ \ulcorner f \urcorner(\mathbf{11}) &= \mathbf{1} \\ \ulcorner f \urcorner(\mathbf{1111}) &= \mathbf{11} \\ \ulcorner f \urcorner(\mathbf{111111}) &= \mathbf{111} \\ &\vdots \end{aligned}$$

Il dominio di $\ulcorner g \urcorner$ è composto dalle stringhe **0111**, **1110**, **1011** e **1101** e si ha:

$$\begin{aligned} \ulcorner g \urcorner(\mathbf{0111}) &= \varepsilon \\ \ulcorner g \urcorner(\mathbf{1110}) &= \varepsilon \\ \ulcorner g \urcorner(\mathbf{1011}) &= \mathbf{11} \\ \ulcorner g \urcorner(\mathbf{1101}) &= \mathbf{11}. \end{aligned}$$

Le funzioni f e g sono computabili. Infatti, ad esempio, $\ulcorner f \urcorner$ è computata dalla seguente DTM:

$q_0 \mathbf{00} q_0$
 $q_0 \mathbf{1} \rightarrow q_1$
 $q_0 \square \rightarrow q_5$
 $q_1 \mathbf{00} q_1$
 $q_1 \mathbf{10} \rightarrow q_2$
 $q_1 \square \square q_1$
 $q_2 \mathbf{00} q_2$
 $q_2 \mathbf{1} \rightarrow q_2$
 $q_2 \square \rightarrow q_3$
 $q_3 \mathbf{0} \square q_0$
 $q_3 \mathbf{1} \square \rightarrow q_4$
 $q_4 \mathbf{01} q_0$
 $q_4 \mathbf{1} \rightarrow q_4$
 $q_5 \mathbf{1} \rightarrow q_5$
 $q_5 \square \rightarrow q_6$



5 Macchine di Turing Multi-nastro

Un macchina di Turing multi-nastro possiede due o più nastri, ciascuno dotato della propria testina meccanica. Le testine possono operare sui rispettivi nastri in maniera indipendente l'una dall'altra però sono comandate da un unico sistema di controllo che determina il comportamento della macchina. Un'istruzione di una macchina di Turing multi-nastro può essere rappresentata come una quadrupla $(q, (t_1 \dots t_k), (o_1 \dots o_k), p)$ dove k è il numero dei nastri della macchina, q e p sono stati e, per $i = 1, \dots, k$, t_i rappresenta il contenuto della casella scandita dalla testina meccanica nell' i -esimo nastro, mentre o_i rappresenta l'operazione che la testina meccanica dell' i -esimo nastro deve eseguire. Quindi, un'azione di una macchina di Turing con k nastri consiste nell'esecuzione in parallelo di k -operazioni delle testine meccaniche, un'operazione su ogni singolo nastro, e nella successiva transizione di stato del sistema di controllo. Data una stringa input $s_{i_1} \dots s_{i_n}$, inizialmente il sistema di controllo assume lo stato iniziale q_0 ; il primo nastro è composto da n caselle, contenenti, da sinistra verso destra, i simboli s_{i_1}, \dots, s_{i_n} e la testina meccanica è posizionata sulla casella più a sinistra; i rimanenti nastri sono composti ciascuno da una singola casella vuota. Quando (e se) la macchina si ferma, l'output viene estratto dal primo nastro con la stessa modalità descritta in precedenza per le macchine di Turing con un solo nastro.

Le definizioni date nella Sezione 3 si generalizzano al caso delle macchine di Turing multi-nastro. Riportiamo di seguito quelle principali.

Definizione 10. Sia $k \in \mathbb{N}$, $k > 1$.

Una k -ISTRUZIONE è una k -upla ordinata $(q_1 t_1 o_1 p_1, \dots, q_k t_k o_k p_k)$ di istruzioni tali che $q_1 = \dots = q_k$ e $p_1 = \dots = p_k$.

Una MACCHINA DI TURING CON k -NASTRI (k -TM) è un insieme finito di k -istruzioni.

Una k -CONFIGURAZIONE è una k -upla ordinata $\gamma = (X_1 q_1 t_1 Y_1, \dots, X_k q_k t_k Y_k)$ di configurazioni tali che $q_1 = \dots = q_k$. Il contenuto estratto da γ è $\langle \gamma \rangle = \langle X_1 q_1 t_1 Y_1 \rangle$, cioè il contenuto estratto dalla prima configurazione.

La k -CONFIGURAZIONE INIZIALE corrispondente ad una stringa $Z \in \mathbf{S}^*$ è la k -configurazione $(\text{Start}(Z), q_0 \square, \dots, q_0 \square)$.

Sia \mathfrak{M} una k -TM.

Per k -configurazioni $\gamma' = (\gamma'_1, \dots, \gamma'_k)$ e $\gamma'' = (\gamma''_1, \dots, \gamma''_k)$, scriviamo $\gamma' \vdash_{\mathfrak{M}} \gamma''$ se esiste una k -istruzione $(I_1, \dots, I_k) \in \mathfrak{M}$ tale che $\gamma'_j \vdash_{\{I_j\}} \gamma''_j$, per ogni $j = 1, \dots, k$.

Una k -configurazione $\gamma = (X_1 q_1 t_1 Y_1, \dots, X_k q_k t_k Y_k)$ è terminale rispetto a \mathfrak{M} se non esiste alcuna k -istruzione $(I_1, \dots, I_k) \in \mathfrak{M}$ tale che I_j comincia con $q t_j$, per ogni $j = 1, \dots, k$.

Un segmento di computazione di \mathfrak{M} è una sequenza finita $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ di k -configurazioni, dove $n \geq 0$, tali che $\gamma_i \vdash_{\mathfrak{M}} \gamma_{i+1}$, per ogni $i = 0, 1, \dots, n-1$. ■

Nel seguito useremo le seguenti notazioni:

(1) ogni k -istruzione $(q_1 t_1 o_1 p_1, \dots, q_k t_k o_k p_k)$ verrà rappresentata come

$$q(t_1, \dots, t_k)(o_1, \dots, o_k)p,$$

dove $q_1 = \dots = q_k = q$ e $p_1 = \dots = p_k = p$;

(2) ogni k -configurazione $(X_1 q_1 t_1 Y_1, \dots, X_k q_k t_k Y_k)$ verrà rappresentata come

$$(q, X_1 \perp t_1 Y_1, \dots, X_k \perp t_k Y_k),$$

dove $q_1 = \dots = q_k = q$.

Esempio 11. La seguente 2-TM \mathfrak{A} accetta il linguaggio \mathcal{L} costituito da tutte le stringhe palindrome sull'alfabeto $\{a, b\}$ (si veda l'esempio 7):

$$\begin{aligned}
& \mathbf{q}_0(a, \square)(a, a)\mathbf{q}_1 \\
& \mathbf{q}_0(b, \square)(b, b)\mathbf{q}_1 \\
& \mathbf{q}_1(a, a)(\uparrow, \uparrow)\mathbf{q}_0 \\
& \mathbf{q}_1(b, b)(\uparrow, \uparrow)\mathbf{q}_0 \\
& \mathbf{q}_0(\square, \square)(\square, \uparrow)\mathbf{q}_2 \\
& \mathbf{q}_2(\square, a)(\square, \uparrow)\mathbf{q}_2 \\
& \mathbf{q}_2(\square, b)(\square, \uparrow)\mathbf{q}_2 \\
& \mathbf{q}_2(\square, \square)(\uparrow, \uparrow)\mathbf{q}_3 \\
& \mathbf{q}_3(a, a)(\uparrow, \uparrow)\mathbf{q}_3 \\
& \mathbf{q}_3(b, b)(\uparrow, \uparrow)\mathbf{q}_3 \\
& \mathbf{q}_3(a, b)(a, b)\mathbf{q}_3 \\
& \mathbf{q}_3(b, a)(b, a)\mathbf{q}_3
\end{aligned}$$

La macchina copia semplicemente la stringa input dal primo nastro sul secondo nastro e successivamente confronta i simboli delle due stringhe attraversandole in ordine inverso: la stringa sul primo nastro viene attraversata da destra verso sinistra mentre quella sul secondo nastro da sinistra verso destra.

Osserviamo che \mathfrak{A} è regolare che

$$T_{\mathfrak{A}}(n) = 4n + 2$$

per ogni $n \in \mathbb{N}$; pertanto $T_{\mathfrak{A}}(n) = \Theta(n)$ e quindi il running time di \mathfrak{A} è lineare. \blacksquare

5.1 Relazioni tra Macchine di Turing e Macchine di Turing multi-nastro

Vediamo adesso come ogni macchina di Turing multi-nastro \mathfrak{M} possa essere “simulata” da una macchina di Turing \mathfrak{T} con un singolo nastro. Siano

- (a) k il numero dei nastri di \mathfrak{M} ;
- (b) $\{\mathbf{s}_{i_0}, \dots, \mathbf{s}_{i_h}\}$ l'alfabeto di \mathfrak{M} ;
- (c) m il minimo indice tale che $\{\mathbf{s}_{i_0}, \dots, \mathbf{s}_{i_h}\} \subseteq \{\mathbf{s}_0, \dots, \mathbf{s}_m\}$.

L'alfabeto di \mathfrak{T} è l'insieme $\{\mathbf{s}_{i_0}, \dots, \mathbf{s}_{i_h}, \mathbf{s}_{i_0}^\perp, \dots, \mathbf{s}_{i_h}^\perp, \square^\perp, /, *\}$ dove

- (a) $\mathbf{s}_{i_j}^\perp = \mathbf{s}_{m+j+1}$, per $j = 0, 1, \dots, h$;
- (b) $\square^\perp = \mathbf{s}_{m+h+2}$;
- (c) $/ = \mathbf{s}_{m+h+3}$;
- (d) $*$ = \mathbf{s}_{m+h+4} .

Gli stati di \mathfrak{T} sono suddivisi in due gruppi: *stati semplici* e *stati composti*. Gli stati semplici sono $\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_{2k+9}$; gli stati composti corrispondono a coppie e triple ordinate del tipo (q, U) e (p, V, W) dove: q e p sono stati di \mathfrak{M} ; U è una stringa su $\{\mathbf{s}_{i_0}, \dots, \mathbf{s}_{i_h}, \square\}$ di lunghezza al più k che rappresenta i contenuti delle caselle scandite da \mathfrak{M} sui k nastri; V è una stringa

su $\{\mathbf{s}_{i_0}, \dots, \mathbf{s}_{i_h}, \square, \uparrow, \uparrow^\perp\}$ di lunghezza al più k che rappresenta le operazioni che \mathfrak{M} esegue sui nastri; W è una stringa su $\{\mathbf{s}_{i_0}, \dots, \mathbf{s}_{i_h}, \square, /, \mathbf{s}_{i_0}^\perp, \dots, \mathbf{s}_{i_h}^\perp, \square^\perp, \uparrow, \uparrow^\perp, \triangleleft, \triangle, \curvearrowright, \curvearrowleft, \leftarrow, \rightarrow\}$ di lunghezza al più 4 usata per determinare le operazioni che deve eseguire \mathfrak{T} per riprodurre ogni singola operazione di \mathfrak{M} .¹

La simulazione di \mathfrak{M} consiste essenzialmente nel rappresentare ogni k -configurazione $\gamma = (q, X_1 \perp t_1 Y_1, \dots, X_k \perp t_k Y_k)$ di \mathfrak{M} con la configurazione $\langle \gamma \rangle = *(q, \varepsilon) / X_1 t_1^\perp Y_1 / \dots / X_k t_k^\perp Y_k / *$. Si osservi che i simboli $*$ e $/$ vengono usati, rispettivamente, come marcatore di margine (sinistro e destro) e come separatore; i simboli $t_1^\perp, \dots, t_k^\perp$ (gli *head-marker*) rappresentano invece i contenuti delle caselle scandite da \mathfrak{M} sui k nastri.

Supponiamo che \mathfrak{M} contiene l'istruzione $\mathbf{I} = q(t_1 \dots t_k)(o_1 \dots o_k)p$. Allora, partendo dalla configurazione $*(q, \varepsilon) / X_1 t_1^\perp Y_1 / \dots / X_k t_k^\perp Y_k / *$, \mathfrak{T} simula l'effetto dell'esecuzione di \mathbf{I} in due fasi successive. Nella prima fase, \mathfrak{T} percorre il nastro da sinistra verso destra alla ricerca degli head-marker $t_1^\perp, \dots, t_k^\perp$. Quando incontra il primo head-marker, cioè t_1^\perp , si sposta di una casella a destra, "memorizza" il simbolo t_1 nel suo stato interno effettuando la transizione di stato da (q, ε) a (q, t_1) e successivamente prosegue alla ricerca del secondo head-marker mantenendosi nello stato (q, t_1) . Quando incontra il secondo head-marker, cioè t_2^\perp , si sposta di una casella a destra, memorizza il simbolo t_2 nel suo stato interno effettuando la transizione di stato da (q, t_1) a $(q, t_1 t_2)$ e successivamente prosegue alla ricerca del terzo head-marker mantenendosi nello stato $(q, t_1 t_2)$. \mathfrak{T} ripete questo processo fino a quando raggiunge la fine del nastro, cioè incontra il simbolo $*$. A questo punto \mathfrak{T} ha memorizzato l'intera sequenza di simboli t_1, \dots, t_k , cioè si trova nello stato $(q, t_1 \dots t_k)$, ed ha quindi sufficienti informazioni per determinare che l'azione compiuta da \mathfrak{M} sarà $(o_1, \dots, o_k)p$. Successivamente \mathfrak{T} si sposta a sinistra, effettua la transizione di stato da $(q, t_1 \dots t_k)$ a $(p, o_1 \dots o_k, \varepsilon)$, memorizzando la stringa di operazioni $o_1 \dots o_k$, ed inizia la seconda fase di simulazione. Nella seconda fase, \mathfrak{T} ripercorre il nastro all'indietro, da destra verso sinistra, e simula le operazioni che \mathfrak{M} esegue sui k nastri, cioè le operazioni o_1, \dots, o_k . Le operazioni vengono simulate in successione, una alla volta, a partire da o_k fino ad o_1 (quindi nell'ordine o_k, \dots, o_1). La simulazione dell'operazione o_i viene realizzata da uno specifico modulo $\mathfrak{A}_i = \mathfrak{A}_i(\mathbf{I})$ di istruzioni tramite le quali \mathfrak{T} effettua la particolare sequenza di operazioni che richiede la simulazione stessa. Questa sequenza di operazioni viene determinata usando una speciale *stringa (variabile) di controllo* $[o_i]$ che \mathfrak{T} memorizza in una componente del suo stato interno. La stringa di controllo funziona in maniera simile ad uno stack con il top all'estremità sinistra: quando \mathfrak{T} deve eseguire un gruppo di operazioni le inserisce al top dello stack una dopo l'altra, in un ordine specifico. \mathfrak{T} esegue sempre l'operazione che si trova al top; quando la termina, la "consuma" dallo stack facendo un pop e passa all'operazione successiva. La simulazione di o_i ha fine quando lo stack diventa vuoto, cioè la stringa di controllo diventa uguale alla stringa vuota ε . (Maggiori dettagli verranno forniti successivamente).

In maniera più precisa, la seconda fase di simulazione viene dunque effettuata come segue. Procedendo da destra verso sinistra, quando \mathfrak{T} incontra il primo head-marker, cioè t_k^\perp , effettua la transizione di stato da $(p, o_1 \dots o_{k-1} o_k)$ a $(p, o_1 \dots o_{k-1}, [o_k])$ memorizzando la stringa di controllo $[o_k]$, e successivamente parte la simulazione di o_k mediante il modulo \mathfrak{A}_k . Quando \mathfrak{T} ha terminato la simulazione di o_k , ha consumato tutta la stringa di controllo (cioè si ha che $[o_k] = \varepsilon$) e assume quindi lo stato $(p, o_1 \dots o_{k-1}, \varepsilon)$. A questo punto riprende la scansione del nastro verso sinistra alla ricerca del prossimo head-marker t_{k-1}^\perp ; appena lo incontra memorizza la stringa di controllo $[o_{k-1}]$ assumendo lo stato $(p, o_1 \dots o_{k-2}, [o_{k-1}])$ e simula l'operazione o_{k-1} mediante il modulo \mathfrak{A}_{k-1} ; quindi procede verso sinistra alla ricerca del prossimo head-marker t_{k-2}^\perp , e così via. Quando \mathfrak{T} raggiunge l'estremità sinistra del nastro, cioè incontra il marcatore

¹Il significato dei simboli $\triangleleft, \triangle, \curvearrowright, \curvearrowleft, \leftarrow, \rightarrow$ sarà chiarito più avanti.

$*$, ha finito di simulare tutte le operazioni di \mathfrak{M} e si trova quindi nello stato $(p, \varepsilon, \varepsilon)$; si sposta a destra, assume lo stato (p, ε) e riprende il processo di simulazione a partire dalla nuova configurazione.

La prima fase di simulazione viene realizzata dal seguente insieme $\mathfrak{I}_1(\mathbf{I})$ di istruzioni, dove $S = \mathbf{S}(\mathfrak{M}) \cup \{\square, /\}$:

$$\begin{aligned}
& (q, \varepsilon)S\dot{\uparrow}(q, \varepsilon) \\
& (q, \varepsilon)t_1^\perp\dot{\uparrow}(q, t_1) \\
& (q, t_1)S\dot{\uparrow}(q, t_1) \\
& (q, t_1)t_2^\perp\dot{\uparrow}(q, t_1t_2) \\
& (q, t_1t_2)S\dot{\uparrow}(q, t_1t_2) \\
& (q, t_1t_2)t_3^\perp\dot{\uparrow}(q, t_1t_2t_3) \\
& \vdots \\
& (q, t_1t_2 \cdots t_{k-1})S\dot{\uparrow}(q, t_1t_2 \cdots t_{k-1}) \\
& (q, t_1t_2 \cdots t_{k-1})t_k^\perp\dot{\uparrow}(q, t_1t_2 \cdots t_{k-1}t_k) \\
& (q, t_1t_2 \cdots t_{k-1}t_k)S\dot{\uparrow}(q, t_1t_2 \cdots t_{k-1}t_k) \\
& (q, t_1t_2 \cdots t_{k-1}t_k)*\dot{\uparrow}(p, o_1o_2 \cdots o_{k-1}o_k, \varepsilon).
\end{aligned}$$

La seconda fase di simulazione viene realizzata dal seguente insieme $\mathfrak{I}_2(\mathbf{I})$ di istruzioni, dove $S = \mathbf{S}(\mathfrak{M}) \cup \{\square, /\}$:

$$\begin{aligned}
& (p, o_1 \cdots o_{k-1}o_k, \varepsilon)S\dot{\uparrow}(p, o_1 \cdots o_{k-1}o_k, \varepsilon) \\
& (p, o_1 \cdots o_{k-1}o_k, \varepsilon)t_k^\perp t_k^\perp(p, o_1 \cdots o_{k-1}, [o_k]) \\
& \mathfrak{A}_k \\
& (p, o_1 \cdots o_{k-1}, \varepsilon)S\dot{\uparrow}(p, o_1 \cdots o_{k-1}, \varepsilon) \\
& (p, o_1 \cdots o_{k-1}, \varepsilon)t_{k-1}^\perp t_{k-1}^\perp(p, o_1 \cdots o_{k-2}, [o_{k-1}]) \\
& \mathfrak{A}_{k-1} \\
& \vdots \\
& (p, o_1, \varepsilon)S\dot{\uparrow}(p, o_1, \varepsilon) \\
& (p, o_1, \varepsilon)t_1^\perp t_1^\perp(p, \varepsilon, [o_1]) \\
& \mathfrak{A}_1 \\
& (p, \varepsilon, \varepsilon)S\dot{\uparrow}(p, \varepsilon, \varepsilon) \\
& (p, \varepsilon, \varepsilon)*\dot{\uparrow}(p, \varepsilon).
\end{aligned}$$

Per ogni operazione o_i , il contenuto iniziale della stringa di controllo $[o_i]$ e il modulo di istruzioni \mathfrak{A}_i sono definiti come segue.

Se $o_i = t \in \mathbf{S}(\mathfrak{M}) \cup \{\square\}$ allora per simulare l'effetto di o_i , la macchina \mathfrak{T} deve semplicemente stampare il simbolo t^\perp e successivamente si deve muovere a sinistra. Per cui:

$$(1) [o_i] = t^\perp\dot{\uparrow} \text{ e}$$

$$(2) \mathfrak{A}_i = \{(p, o_1 \cdots o_{i-1}, t^\perp\dot{\uparrow})t_i^\perp t^\perp(p, o_1 \cdots o_{i-1}, \dot{\uparrow}), (p, o_1 \cdots o_{i-1}, \dot{\uparrow})t^\perp\dot{\uparrow}(p, o_1 \cdots o_{i-1}, \varepsilon)\}.$$

Se $o_i = \dot{\uparrow}$ allora nel simulare l'effetto di o_i , la macchina \mathfrak{T} potrebbe trovarsi di fronte ad un problema: se la casella scandita sull' i -esimo nastro di \mathfrak{M} si trova all'estremità destra del nastro stesso, \mathfrak{T} deve "shiftare" verso destra tutti i simboli del nastro che seguono t_i^\perp (oppure shiftare verso sinistra tutti i simboli del nastro che si trovano alla sinistra di t_i^\perp , compreso t_i^\perp) in modo da lasciare spazio al blank. Per cui, inizialmente \mathfrak{T} si sposta alla destra di t_i^\perp e controlla il simbolo t che incontra. Se questo simbolo è diverso dal separatore $/$, non è necessario effettuare alcuno shift e \mathfrak{T} semplicemente sostituisce t con il corrispondente head-marker t^\perp e poi si sposta a sinistra; in caso contrario \mathfrak{T} deve effettuare lo shift a destra dei

simboli. Per fare ciò \mathfrak{T} stampa inizialmente il simbolo $*$ per ricordarsi da dove deve iniziare lo shift; successivamente si sposta alla fine del nastro eseguendo un “loop di spostamento a destra” fino a quando incontra il marcatore $*$; quindi si sposta di una casella a destra e stampa $*$. A questo punto, ripercorre il nastro all’indietro shiftando di una casella verso destra ogni simbolo che incontra fino a $*$. Usiamo il simbolo \triangleright per indicare l’operazione di controllo del simbolo alla destra di t_i^\perp e usiamo i simboli \leftrightsquigarrow e \curvearrowright per indicare le operazioni di loop di spostamento a destra e l’operazione di shifting verso destra dei simboli, rispettivamente. Allora $[o_i] = t_i \triangleright$ e \mathfrak{A}_i consiste nelle seguenti istruzioni:

$$\begin{aligned}
& (p, o_1 \cdots o_{i-1}, t_i \triangleright) t_i^\perp t_i (p, o_1 \cdots o_{i-1}, \triangleright) \\
& (p, o_1 \cdots o_{i-1}, \triangleright) t_i \triangleright (p, o_1 \cdots o_{i-1}, \triangleright) \\
& \{ \\
& \quad (p, o_1 \cdots o_{i-1}, \triangleright) t t^\perp (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow) \\
& \quad (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow) t^\perp \leftrightsquigarrow (p, o_1 \cdots o_{i-1}, \varepsilon) : t \in \mathbf{S}(\mathfrak{M}) \cup \{\square\} \\
& \} \\
& (p, o_1 \cdots o_{i-1}, \triangleright) / * (p, o_1 \cdots o_{i-1}, \triangleright \leftrightsquigarrow) \\
& (p, o_1 \cdots o_{i-1}, \triangleright \leftrightsquigarrow) * \triangleright (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow) \\
& \{ \\
& \quad (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow) s \triangleright (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow) : s \in \mathbf{S}(\mathfrak{T}) \setminus \{*\} \\
& \} \\
& (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow) * \triangleright (p, o_1 \cdots o_{i-1}, * \leftrightsquigarrow \curvearrowright) \\
& (p, o_1 \cdots o_{i-1}, * \leftrightsquigarrow \curvearrowright) \square * (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow \curvearrowright) \\
& (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow \curvearrowright) * \leftrightsquigarrow (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow \curvearrowright) \\
& (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow \curvearrowright) * \leftrightsquigarrow (p, o_1 \cdots o_{i-1}, \curvearrowright) \\
& \{ \\
& \quad (p, o_1 \cdots o_{i-1}, \curvearrowright) s \triangleright (p, o_1 \cdots o_{i-1}, s \leftrightsquigarrow \curvearrowright) \\
& \quad (p, o_1 \cdots o_{i-1}, s \leftrightsquigarrow \curvearrowright) u s (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow \curvearrowright) \\
& \quad (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow \curvearrowright) s \leftrightsquigarrow (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow \curvearrowright) \\
& \quad (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow \curvearrowright) s \leftrightsquigarrow (p, o_1 \cdots o_{i-1}, \curvearrowright) : s \in \mathbf{S}(\mathfrak{T}) \setminus \{*\}, u \in \mathbf{S}(\mathfrak{T}) \\
& \} \\
& (p, o_1 \cdots o_{i-1}, \curvearrowright) * \square^\perp (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow) \\
& (p, o_1 \cdots o_{i-1}, \leftrightsquigarrow) \square^\perp \leftrightsquigarrow (p, o_1 \cdots o_{i-1}, \varepsilon).
\end{aligned}$$

Il caso $o_i = \leftrightsquigarrow$ è “simmetrico” di quello precedente ... (I simboli usati in questo caso per rappresentare le operazioni principali sono: \triangleleft per il controllo del simbolo a sinistra di t_i^\perp ; \curvearrowleft per lo shift verso sinistra; \curvearrowright per il loop di spostamento a sinistra).

Il seguente insieme \mathfrak{T}' di istruzioni permette a \mathfrak{T} di passare da ogni configurazione iniziale $\mathbf{Start}(Z)$, dove $Z \in \mathbf{S}(\mathfrak{M})^*$, alla configurazione $\langle \mathbf{Start}(Z) \rangle$ che rappresenta la k -

configurazione iniziale di \mathfrak{M} corrispondente a Z :

$$\begin{aligned}
& \{ \\
& \quad \mathbf{q}_0 t t^\perp \mathbf{q}_1 \\
& \quad \mathbf{q}_0 t^\perp t^\perp \mathbf{q}_0 : t \in \mathbf{S}(\mathfrak{M}) \cup \{\square\} \\
& \} \\
& \mathbf{q}_0 // \mathbf{q}_0 \\
& \mathbf{q}_0 ** \mathbf{q}_0 \\
& \{ \\
& \quad \mathbf{q}_1 t^\perp \curvearrowright \mathbf{q}_1 : t \in \mathbf{S}(\mathfrak{M}) \cup \{\square\} \\
& \} \\
& \mathbf{q}_1 \square / \mathbf{q}_1 \\
& \mathbf{q}_1 / \curvearrowright \mathbf{q}_2 \\
& \mathbf{q}_2 \square * \mathbf{q}_2 \\
& \mathbf{q}_2 * \curvearrowright \mathbf{q}_2 \\
& \mathbf{q}_2 / \curvearrowright \mathbf{q}_2 \\
& \{ \\
& \quad \mathbf{q}_2 t^\perp \curvearrowright \mathbf{q}_3 \\
& \quad \mathbf{q}_3 t \curvearrowright \mathbf{q}_3 \\
& \quad \mathbf{q}_3 t^\perp t^\perp \mathbf{q}_0 : t \in \mathbf{S}(\mathfrak{M}) \cup \{\square\} \\
& \} \\
& \mathbf{q}_3 // \mathbf{q}_0 \\
& \mathbf{q}_3 ** \mathbf{q}_0 \\
& \mathbf{q}_3 \square / \mathbf{q}_4 \\
& \mathbf{q}_4 / \curvearrowright \mathbf{q}_4 \\
& \mathbf{q}_4 \square \square^\perp \mathbf{q}_4 \\
& \mathbf{q}_4 \square^\perp \curvearrowright \mathbf{q}_5 \\
& \vdots \\
& \mathbf{q}_{2k-1} \square / \mathbf{q}_{2k} \\
& \mathbf{q}_{2k} / \curvearrowright \mathbf{q}_{2k} \\
& \mathbf{q}_{2k} \square \square^\perp \mathbf{q}_{2k} \\
& \mathbf{q}_{2k} \square^\perp \curvearrowright \mathbf{q}_{2k+1} \\
& \mathbf{q}_{2k+1} \square / \mathbf{q}_{2k+1} \\
& \mathbf{q}_{2k+1} / \curvearrowright \mathbf{q}_{2k+2} \\
& \mathbf{q}_{2k+2} \square * \mathbf{q}_{2k+2} \\
& \mathbf{q}_{2k+2} * \curvearrowright \mathbf{q}_{2k+3} \\
& \{ \\
& \quad \mathbf{q}_{2k+3} s \curvearrowright \mathbf{q}_{2k+3} : s \in \mathbf{S}(\mathfrak{T}) \setminus \{*\} \\
& \} \\
& \mathbf{q}_{2k+3} * \curvearrowright (\mathbf{q}_0, \varepsilon).
\end{aligned}$$

Infatti, l'insieme \mathfrak{J}' ha il seguente effetto:

per ogni stringa non vuota $Z = t_1 t_2 \cdots t_m \in (\mathbf{S}(\mathfrak{T}))^*$, se $Z \in (\mathbf{S}(\mathfrak{M}))^*$ allora

$$\mathbf{Start}(Z) \vdash_{\mathfrak{J}'}^* \mathbf{q}_0, \varepsilon / t_1^\perp t_2 \cdots t_m / \square^\perp / \square^\perp / \cdots \square^\perp / *,$$

dove il numero di occorrenze di $/$ è $k + 1$, altrimenti, cioè se $Z \notin (\mathbf{S}(\mathfrak{M}))^*$, allora $\mathfrak{J}'(Z) \uparrow$.

Similmente,

$$\mathbf{Start}(\varepsilon) \vdash_{\mathfrak{J}'}^* \mathbf{q}_0, \varepsilon / \square^\perp / \square^\perp / \square^\perp / \cdots \square^\perp / *.$$

Per completare la costruzione di \mathfrak{T} introduciamo infine l'insieme di istruzioni \mathfrak{J}'' che permette di “ripulire” il nastro di \mathfrak{T} lasciando soltanto la configurazione che rappresenta il primo nastro di \mathfrak{M} , quando \mathfrak{M} raggiunge una k -configurazione terminale. Poniamo

$$\mathfrak{J}'' = \bigcup_{(q, (t_1, \dots, t_k)) \in \mathbf{F}(\mathfrak{M})} \mathfrak{F}((q, (t_1, \dots, t_k))),$$

dove $\mathfrak{F}((q, (t_1, \dots, t_k)))$ è il seguente insieme di istruzioni, per ogni $(q, (t_1, \dots, t_k)) \in \mathbf{F}(\mathfrak{M})$:

$$\begin{aligned} & (q, t_1 \cdots t_k) * \ulcorner \mathbf{q}_{2k+4} \\ & \{ \\ & \quad \mathbf{q}_{2k+4} t \ulcorner \mathbf{q}_{2k+4} \\ & \quad \mathbf{q}_{2k+4} t^\perp \ulcorner \mathbf{q}_{2k+4} : t \in \mathbf{S}(\mathfrak{M}) \cup \{\square\} \\ & \} \\ & \mathbf{q}_{2k+4} / \ulcorner \mathbf{q}_{2k+4} \\ & \mathbf{q}_{2k+4} * \square \mathbf{q}_{2k+5} \\ & \{ \\ & \quad \mathbf{q}_{2k+5} t \urcorner \mathbf{q}_{2k+5} \\ & \quad \mathbf{q}_{2k+5} t^\perp \urcorner \mathbf{q}_{2k+5} : t \in \mathbf{S}(\mathfrak{M}) \cup \{\square\} \\ & \} \\ & \mathbf{q}_{2k+5} / \urcorner \mathbf{q}_{2k+6} \\ & \{ \\ & \quad \mathbf{q}_{2k+6} s \square \mathbf{q}_{2k+6} : s \in \mathbf{S}(\mathfrak{T}) \setminus \{\square, *\} \\ & \} \\ & \mathbf{q}_{2k+6} \square \urcorner \mathbf{q}_{2k+6} \\ & \mathbf{q}_{2k+6} * \square \mathbf{q}_{2k+7} \\ & \mathbf{q}_{2k+7} \square \ulcorner \mathbf{q}_{2k+7} \\ & \mathbf{q}_{2k+7} / \square \mathbf{q}_{2k+8} \\ & \{ \\ & \quad \mathbf{q}_{2k+8} t \ulcorner \mathbf{q}_{2k+8} \\ & \quad \mathbf{q}_{2k+8} t^\perp \ulcorner \mathbf{q}_{2k+9} : t \in \mathbf{S}(\mathfrak{M}) \cup \{\square\} \\ & \}. \end{aligned}$$

Stimiamo adesso il tempo impiegato da \mathfrak{T} per simulare un'intera computazione propria $\mathbf{\Gamma} = (\gamma_0, \gamma_1, \dots, \gamma_n)$ di \mathfrak{M} . Sia $\langle \mathbf{\Gamma} \rangle$ la computazione che effettua \mathfrak{T} per simulare $\mathbf{\Gamma}$.

Cominciamo con l'osservare che $\langle \mathbf{\Gamma} \rangle$ è uguale alla concatenazione dei segmenti di computazione $\Gamma', \Gamma(\gamma_0), \Gamma(\gamma_1), \dots, \Gamma(\gamma_{n-1}), \Gamma''$ dove:

- (a) Γ' è il segmento di computazione che esegue \mathfrak{T} (tramite \mathfrak{J}') per passare dalla configurazione iniziale $\mathbf{Start}(Z)$ alla configurazione $\langle \gamma_0 \rangle$;
- (b) $\Gamma(\gamma_i)$ è il segmento di computazione che esegue \mathfrak{T} per passare dalla configurazione $\langle \gamma_i \rangle$ alla configurazione $\langle \gamma_{i+1} \rangle$, per $i = 0, 1, \dots, n-1$;
- (c) Γ'' è il segmento di computazione che esegue \mathfrak{T} per passare (tramite \mathfrak{J}'') dalla configurazione $\langle \gamma_n \rangle$ ad una configurazione terminale.

Sia $Z = \mathbf{Input}(\mathbf{\Gamma})$ e sia $j \in \{0, 1, \dots, n-1\}$. Poiché $\gamma_j \vdash_{\mathfrak{M}} \gamma_{j+1}$, si ha che $|\langle \gamma_{j+1} \rangle| \leq |\langle \gamma_j \rangle| + k$ e quindi $|\langle \gamma_{j+1} \rangle| \leq |\langle \gamma_0 \rangle| + jk \leq |\langle \gamma_0 \rangle| + nk$. Dato che $|\langle \gamma_0 \rangle| = \mathcal{O}(|Z| + k)$, dalla precedente disuguaglianza segue che $|\langle \gamma_{j+1} \rangle| = \mathcal{O}(|Z| + nk)$; pertanto

$$|\langle \gamma_i \rangle| = \mathcal{O}(|Z| + nk), \tag{1}$$

per ogni $i = 0, 1, \dots, n$.

Per passare dalla configurazione $\langle \gamma_j \rangle$ alla configurazione $\langle \gamma_{j+1} \rangle$, \mathfrak{T} effettua al più k shift del nastro per simulare gli eventuali spostamenti a destra e/o a sinistra di \mathfrak{M} sui k nastri, e quindi dalla (1) segue che $\Gamma(\gamma_j)$ ha lunghezza $\mathcal{O}(k|Z| + nk^2)$, per ogni $j = 0, 1, \dots, n-1$. Pertanto la lunghezza della concatenazione di $\Gamma(\gamma_0), \Gamma(\gamma_1), \dots, \Gamma(\gamma_{n-1})$ è $\mathcal{O}(nk|Z| + n^2k^2)$. Si osservi infine che i segmenti Γ' e Γ'' hanno lunghezze proporzionali a $|\langle \gamma_0 \rangle|$ e $|\langle \gamma_n \rangle|$, rispettivamente, e quindi, per la (1), le loro lunghezze sono $\mathcal{O}(|Z| + nk)$. Riassumendo otteniamo che la lunghezza totale di $\langle \Gamma \rangle$ è $\mathcal{O}(|Z|k^2n^2)$.

Teorema 1. *Sia \mathfrak{M} una k -TM. Esiste una TM \mathfrak{T} con un solo nastro che simula \mathfrak{M} tale che per ogni computazione propria Γ di \mathfrak{M} si ha che*

$$|\langle \Gamma \rangle| = \mathcal{O}(|\text{Input}(\Gamma)|k^2|\Gamma|^2),$$

dove $\langle \Gamma \rangle$ è la computazione che effettua \mathfrak{T} per simulare Γ .

Osservazione 1. Se nel precedente teorema riguardiamo k come una costante e assumiamo (ragionevolmente) che $|\text{Input}(\Gamma)| \leq |\Gamma|$ (cioè \mathfrak{M} esamina tutta la stringa input durante la computazione) possiamo concludere che

$$|\langle \Gamma \rangle| = \mathcal{O}(|\Gamma|^2).$$

■

6 Classi di Complessità Temporale

Definizione 11. Sia $t : \mathbb{N} \rightarrow \mathbb{N}$ una funzione e sia \mathfrak{M} una TM.

Un LINGUAGGIO \mathcal{L} È RICONOSCIUTO (O ACCETTATO) DA \mathfrak{M} IN TEMPO (LIMITATO DA) t se:

- (1) \mathcal{L} è accettato da \mathfrak{M} ;
- (2) per ogni stringa $X \in \mathcal{L}$ si ha che $\mathfrak{M}(X) \downarrow^{(\leq t(|X|))}$.

Una RELAZIONE DI STRINGHE ρ È COMPUTATA DA \mathfrak{M} IN TEMPO (LIMITATO DA) t se:

- (1) ρ è computata da \mathfrak{M} ;
- (2) per ogni coppia di stringhe $(X, Y) \in \rho$ si ha che $\mathfrak{M}(X) \downarrow^{(\leq t(|X|))} Y$. ■

Definizione 12. Sia $t : \mathbb{N} \rightarrow \mathbb{N}$ una funzione.

- (1) Un LINGUAGGIO \mathcal{L} È RICONOSCIBILE IN TEMPO NONDETERMINISTICO t se esiste una TM \mathfrak{M} tale che \mathcal{L} è riconosciuto da \mathfrak{M} in tempo t . La CLASSE DEI LINGUAGGI RICONOSCIBILI IN TEMPO NONDETERMINISTICO t è indicata con **NTIME**(t).
- (2) Un LINGUAGGIO \mathcal{L} È RICONOSCIBILE IN TEMPO DETERMINISTICO t se esiste una DTM \mathfrak{D} tale che \mathcal{L} è riconosciuto da \mathfrak{D} in tempo t . La CLASSE DEI LINGUAGGI RICONOSCIBILI IN TEMPO DETERMINISTICO t è indicata con **TIME**(t).
- (3) Una FUNZIONE DI STRINGHE f È COMPUTABILE IN TEMPO t se esiste una DTM \mathfrak{D} tale che f è computata da \mathfrak{D} in tempo t . ■

Osservazione 2. Le precedenti definizioni si estendono al caso di funzioni $t : \mathbb{N} \rightarrow \mathbb{R}$ qualsiasi sostituendo $t(|X|)$ con la quantità $\max(0, \lceil t(|X|) \rceil)$, dove $\lceil r \rceil$ denota il “ceiling” di r (cioè il più piccolo numero intero maggiore o uguale a r), per ogni $r \in \mathbb{R}$. ■

Teorema 2. Siano \mathfrak{M} una TM, \mathcal{L} un linguaggio e $t : \mathbb{N} \rightarrow \mathbb{N}$ una funzione. Le seguenti condizioni sono equivalenti.

- (1) \mathcal{L} è riconosciuto da \mathfrak{M} in tempo t ;
- (2) \mathcal{L} è accettato da \mathfrak{M} e $T_{\mathfrak{M}}(n) \leq t(n)$, per ogni $n \in \mathbb{N}$.

Dimostrazione. Supponiamo che \mathcal{L} è riconosciuto da \mathfrak{M} in tempo t e sia $n \in \mathbb{N}$. Se $T_{\mathfrak{M}}(n) = -1$ allora, ovviamente, $T_{\mathfrak{M}}(n) \leq t(n)$. Sia quindi $T_{\mathfrak{M}}(n) \neq -1$. Allora esiste una stringa $X \in \mathcal{L}$ tale che $|X| = n$ e $T_{\mathfrak{M}}(n) = \widehat{T}_{\mathfrak{M}}(X)$. Poiché \mathcal{L} è riconosciuto da \mathfrak{M} in tempo t esiste un $m \leq t(|X|) = t(n)$ tale che $\mathfrak{M}(X) \downarrow^{(m)}$. Dalla definizione di $\widehat{T}_{\mathfrak{M}}(X)$ segue che $\widehat{T}_{\mathfrak{M}}(X) \leq m$ e quindi $T_{\mathfrak{M}}(n) \leq t(n)$. Pertanto (1) implica (2). Il viceversa, cioè che (2) implica (1), segue osservando che se $X \in \mathcal{L}$ allora $\mathfrak{M}(X) \downarrow^{(T_{\mathfrak{M}}(X))}$ e $\widehat{T}_{\mathfrak{M}}(X) \leq T_{\mathfrak{M}}(|X|)$. □

Teorema 3. Siano \mathfrak{D} una DTM, f una funzione di stringhe e $t : \mathbb{N} \rightarrow \mathbb{N}$ una funzione. Le seguenti condizioni sono equivalenti.

- (1) f è computata da \mathfrak{D} in tempo t ;
- (2) f è computata da \mathfrak{D} e $T_{\mathfrak{D}}(n) \leq t(n)$, per ogni $n \in \mathbb{N}$;
- (3) f è computata da \mathfrak{D} e $|\Gamma| \leq t(|\text{Input}(\Gamma)|)$, per ogni computazione propria Γ di \mathfrak{D} .

Dimostrazione. Segue osservando che, poichè \mathfrak{D} è deterministica, per ogni stringa $X \in \mathcal{L}$ esiste una (ed una) sola computazione di \mathfrak{D} con input X . □

Teorema 4. Per ogni funzione $t : \mathbb{N} \rightarrow \mathbb{N}$, $\mathbf{TIME}(t) \subseteq \mathbf{NTIME}(t)$.

Teorema 5. Sia $\mathcal{L} \in \mathbf{NTIME}(t)$ dove $t(n) \geq n$ per ogni $n \in \mathbb{N}$. Esistono una costante $c \geq 2$ ed una funzione $u(n) = O(c^{t(n)})$ tali che $\mathcal{L} \in \mathbf{TIME}(u)$.

Dimostrazione. Si veda “M. Sipser, *Introduction to the Theory of Computation*”, teoremi 7.11 e 3.16. □

Definizione 13. Un LINGUAGGIO \mathcal{L} È RICONOSCIBILE IN TEMPO NONDETERMINISTICO POLINOMIALE se esiste una funzione polinomiale $p : \mathbb{N} \rightarrow \mathbb{N}$ tale che $\mathcal{L} \in \mathbf{NTIME}(p)$. La CLASSE DEI LINGUAGGI RICONOSCIBILI IN TEMPO NONDETERMINISTICO POLINOMIALE è indicata con **NP**.

Un LINGUAGGIO \mathcal{L} È RICONOSCIBILE IN TEMPO DETERMINISTICO POLINOMIALE se esiste una funzione polinomiale $p : \mathbb{N} \rightarrow \mathbb{N}$ tale che $\mathcal{L} \in \mathbf{TIME}(p)$. La CLASSE DEI LINGUAGGI RICONOSCIBILI IN TEMPO DETERMINISTICO POLINOMIALE è indicata con **P**.

Una funzione di stringhe f è COMPUTABILE IN TEMPO POLINOMIALE se esiste una funzione polinomiale $p : \mathbb{N} \rightarrow \mathbb{N}$ tale che f è computabile in tempo p . ■

Teorema 6. $\mathbf{P} \subseteq \mathbf{NP}$. Se $\mathcal{L} \in \mathbf{NP}$ allora \mathcal{L} è decidibile.

Dimostrazione. Che $\mathbf{P} \subseteq \mathbf{NP}$ segue immediatamente dal Teorema 4.

Sia $\mathcal{L} \in \mathbf{NP}$ e siano \mathfrak{M} una TM e $p : \mathbb{N} \rightarrow \mathbb{N}$ una funzione polinomiale tali che \mathfrak{M} riconosce \mathcal{L} in tempo p . Per decidere se una stringa $Z \in \Sigma_{\mathcal{L}}^*$ è membro di \mathcal{L} possiamo procedere come segue. Sia $S_0 = \{\text{Start}(Z)\}$. Per ogni $i > 0$ sia S_i l'insieme di tutte le \mathfrak{M} -configurazioni γ tali che esiste una configurazione $\delta \in S_{i-1}$ per la quale si ha che $\delta \vdash_{\mathfrak{M}} \gamma$.

Poiché \mathfrak{M} riconosce \mathcal{L} in tempo p , si ha che $Z \in \mathcal{L}$ se e solo se qualcuno degli insiemi $S_0, S_1, \dots, S_{p(|Z|)}$ contiene una configurazione terminale rispetto a \mathfrak{M} . Dato che ogni insieme S_i può essere effettivamente costruito a partire da S_{i-1} esaminando semplicemente le istruzioni di \mathfrak{M} , per ogni $i > 0$, abbiamo il seguente algoritmo per decidere se $Z \in \mathcal{L}$: partendo da S_0 , costruiamo la sequenza $S_0, S_1, \dots, S_{p(|Z|)}$ e successivamente controlliamo se qualcuno degli insiemi S_i contiene una configurazione terminale rispetto a \mathfrak{M} . Se è così possiamo concludere che $Z \in \mathcal{L}$; altrimenti si ha che $Z \notin \mathcal{L}$. \square

Teorema 7. *Sia \mathcal{L} un linguaggio. Allora $\mathcal{L} \in \mathbf{P}$ se e solo se la funzione caratteristica di \mathcal{L} è computabile in tempo polinomiale.*

Dimostrazione. Basta osservare che se \mathfrak{D} è una DTM tale che $C_{\mathcal{L}}$ è computata da \mathfrak{D} in tempo t , allora \mathcal{L} è accettato da \mathfrak{D} in tempo t . Viceversa, se \mathfrak{D} è una DTM tale che \mathcal{L} è accettato da \mathfrak{D} in tempo t , sia \mathfrak{A} la DTM definita da

$$\mathfrak{A} = \mathfrak{D} \cup \{qt \square \mathbf{q}_{k+1} : (q, t) \in \mathbf{F}(\mathfrak{D})\},$$

dove k è il minimo indice tale che $\mathbf{Q}(\mathfrak{D}) \subseteq \{\mathbf{q}_0, \dots, \mathbf{q}_k\}$. Si verifica che $C_{\mathcal{L}}$ è computata da \mathfrak{A} in tempo $t + 1$. \square

Definizione 14. Una FUNZIONE POLINOMIALE p È SEMPLICE se esistono costanti $a, b, c \in \mathbb{N}^+$ tali che $p(n) = a + b \cdot n^c$, per ogni $n \in \mathbb{N}$. \blacksquare

Teorema 8. *Sia $p : \mathbb{N} \rightarrow \mathbb{N}$ una funzione polinomiale. Esiste una funzione polinomiale semplice q tale che $p(n) < q(n)$, per ogni $n \in \mathbb{N}$.*

Dimostrazione. Sia δ il grado di p e sia k il coefficiente del termine di grado δ di p . Se $\delta = 0$ il risultato è banale. Supponiamo quindi che $\delta > 0$, e sia $c = \delta$. Poiché

$$\lim_{n \rightarrow +\infty} \frac{p(n)}{n^c} = k,$$

esistono $b \in \mathbb{N}^+$ e $\nu \in \mathbb{N}$ tali che $p(n) < b \cdot n^c$, per ogni $n \geq \nu$. Sia $a = \max\{p(0), \dots, p(\nu)\} + 1$. Allora $a \in \mathbb{N}^+$ e inoltre $p(n) < a + b \cdot n^c$, per ogni $n \in \mathbb{N}$. \square

Teorema 9. *Sia \mathfrak{M} una TM.*

Per ogni computazione Γ di \mathfrak{M} , si ha che $|\text{Output}(\Gamma)| \leq |\text{Input}(\Gamma)| + |\Gamma| + 1$.

Dimostrazione. Sia $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ e siano $X = \text{Input}(\Gamma)$ e $Y = \text{Output}(\Gamma)$. Si verifica facilmente che $|\gamma_{i+1}| \leq |\gamma_i| + 1$ per ogni $i = 0, 1, \dots, n-1$ (poiché $\gamma_i \vdash_{\mathfrak{M}} \gamma_{i+1}$), e pertanto $|\gamma_n| \leq |\gamma_0| + n = |X| + 1 + n$. Si osservi quindi che $|\gamma_0| \leq |X| + 2$ e $|\gamma_n| \geq |Y| + 1$. \square

Teorema 10. *Sia \mathfrak{M} una TM.*

Per ogni $(X, Y) \in \rho(\mathfrak{M})$ e ogni $n \in \mathbb{N}$, se $\mathfrak{M}(X) \downarrow^{(n)} Y$ allora $|Y| \leq |X| + n + 1$.

Dimostrazione. Segue dal teorema precedente. \square

Teorema 11. *Se f è una funzione computabile in tempo t , allora $|f(X)| \leq |X| + t(|X|) + 1$, per ogni stringa X nel dominio di f .*

Dimostrazione. Segue immediatamente dal Teorema 10. \square

Teorema 12. *Se f e g sono funzioni computabili in tempo polinomiale allora la funzione composta $f \circ g$ è computabile in tempo polinomiale.*

Dimostrazione. Siano \mathfrak{M}_f e \mathfrak{M}_g DTM e p_f e p_g funzioni polinomiali tali che f è computata da \mathfrak{M}_f in tempo p_f e g è computata da \mathfrak{M}_g in tempo p_g . Senza ledere la generalità possiamo assumere che p_f e p_g siano semplici (cf. Teorema 8).

Per computare la funzione composta $f \circ g$ usiamo una 2-DTM \mathfrak{D} . Inizialmente \mathfrak{D} copia l'input X dal primo nastro sul secondo nastro, lasciando il primo nastro vuoto (*fase 1*). Successivamente, \mathfrak{D} usa \mathfrak{M}_f per computare $g(X)$ sul secondo nastro (*fase 2*). Quando \mathfrak{D} ha finito la computazione di $g(X)$ copia il risultato (cioè $g(X)$) sul primo nastro (*fase 3*); quindi \mathfrak{D} usa \mathfrak{M}_f per computare $f(g(X))$ sul primo nastro (*fase 4*).

Sia n il minimo indice tale che $\mathbf{Q}(\mathfrak{M}_g) \subseteq \{\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_n\}$.

La fase 1 viene realizzata dal seguente insieme di 2-istruzioni:

$$\begin{aligned} \mathfrak{I}_1 = \{ & \mathbf{q}_0(t, \square)(\square, t)\mathbf{q}_0 \\ & \mathbf{q}_0(s, \square)(s, \square)\mathbf{q}_0 \\ & \mathbf{q}_0(\square, t)(\uparrow, \uparrow)\mathbf{q}_0 \\ & \mathbf{q}_0(\square, \square)(\square, \uparrow)\mathbf{q}_1 : t \in \mathbf{S}(\mathfrak{M}_g), s \in \mathbf{S}(\mathfrak{M}_f) \setminus \mathbf{S}(\mathfrak{M}_g) \}. \end{aligned}$$

La fase 2 viene realizzata dal seguente insieme di 2-istruzioni:

$$\mathfrak{I}_2 = \{ \mathbf{q}_{i+1}(\square, t)(\square, o)\mathbf{q}_{j+1} : \mathbf{q}_i t o \mathbf{q}_j \in \mathfrak{M}_g \}.$$

La fase 3 viene realizzata dal seguente insieme di 2-istruzioni:

$$\begin{aligned} \mathfrak{I}_3 = \{ & \mathbf{q}_{i+1}(\square, u)(\square, u)\mathbf{q}_{n+2} \\ & \mathbf{q}_{n+2}(\square, t)(t, \square)\mathbf{q}_{n+2} \\ & \mathbf{q}_{n+2}(\square, s)(\square, s)\mathbf{q}_{n+2} \\ & \mathbf{q}_{n+2}(t, \square)(\uparrow, \uparrow)\mathbf{q}_{n+2} \\ & \mathbf{q}_{n+2}(\square, \square)(\uparrow, \square)\mathbf{q}_{n+3} : (\mathbf{q}_i, u) \in \mathbf{F}(\mathfrak{M}_g), t \in \mathbf{S}(\mathfrak{M}_f), s \in \mathbf{S}(\mathfrak{M}_g) \setminus \mathbf{S}(\mathfrak{M}_f) \}. \end{aligned}$$

La fase 4 viene realizzata dal seguente insieme di 2-istruzioni:

$$\mathfrak{I}_4 = \{ \mathbf{q}_{i+n+3}(\square, t)(\square, o)\mathbf{q}_{j+n+3} : \mathbf{q}_i t o \mathbf{q}_j \in \mathfrak{M}_f \}.$$

Pertanto $\mathfrak{D} = \mathfrak{I}_1 \cup \mathfrak{I}_2 \cup \mathfrak{I}_3 \cup \mathfrak{I}_4$.

Sia X una stringa appartenente al dominio di $f \circ g$. Stimiamo la lunghezza della computazione Γ di \mathfrak{D} con input X . Cominciamo osservando che Γ è formata dalla concatenazione dei 4 segmenti di computazione $\Gamma_1, \Gamma_2, \Gamma_3$ e Γ_4 di \mathfrak{D} corrispondenti alle fasi 1, 2, 3 e 4, rispettivamente. Il segmento Γ_1 ha lunghezza $2|X| + 1$; il segmento Γ_2 ha lunghezza al più $p_g(|X|)$; il segmento Γ_3 ha lunghezza $2|g(X)| + 2$; il segmento Γ_4 ha lunghezza al più $p_f(|g(X)|)$. Quindi $|\Gamma| \leq 2|X| + 1 + p_g(|X|) + 2|g(X)| + 2 + p_f(|g(X)|)$. Per il Teorema 11 si ha che $|g(X)| \leq |X| + p_g(|X|) + 1$ che implica che $p_f(|g(X)|) \leq p_f(|X| + p_g(|X|) + 1)$ (poichè p_f è una funzione crescente in quanto polinomiale semplice) e quindi

$$|\Gamma| \leq 2|X| + 1 + p_g(|X|) + 2(|X| + p_g(|X|) + 1) + 2 + p_f(|X| + p_g(|X|) + 1).$$

Pertanto esiste una funzione polinomiale q tale che, per ogni computazione propria Γ di \mathfrak{D} si ha che

$$|\Gamma| \leq q(|\text{Input}(\Gamma)|).$$

Poichè $f \circ g$ è computata da \mathfrak{D} , dai teoremi 1 e 3 segue che $f \circ g$ è computabile in tempo polinomiale. \square

7 Una caratterizzazione della classe NP

Definizione 15. Sia ρ una relazione.

- (1) Un LINGUAGGIO \mathcal{L} RAPPRESENTA ρ se esiste un “simbolo separatore” $/ \in (\Sigma_{\mathcal{L}} \setminus \Sigma_{\rho})$ tale che

$$\mathcal{L} = \{X/Y : (X, Y) \in \rho\}.$$

- (2) Diciamo che ρ È VERIFICABILE IN TEMPO DETERMINISTICO POLINOMIALE se esiste un linguaggio $\mathcal{L} \in \mathbf{P}$ tale che \mathcal{L} rappresenta ρ . In tal caso, ogni DTM che riconosce \mathcal{L} in tempo p , dove p è una funzione polinomiale, è detta un VERIFICATORE (DETERMINISTICO) POLINOMIALE PER ρ .

- (3) Diciamo che ρ È POLINOMIALMENTE BILANCIATA se esiste una funzione polinomiale p tale che per ogni coppia di stringhe $(X, Y) \in \rho$ si ha che $|Y| \leq p(|X|)$. ■

Teorema 13. Sia \mathcal{L} un linguaggio. Allora, $\mathcal{L} \in \mathbf{NP}$ se e solo se esiste una relazione ρ tale che:

- (1) ρ è verificabile in tempo deterministico polinomiale;
(2) ρ è polinomialmente bilanciata;
(3) una stringa X appartiene a \mathcal{L} se e solo se esiste una stringa Y , chiamata un CERTIFICATO (POLINOMIALMENTE) SUCCINTO PER X , tale che $(X, Y) \in \rho$, cioè

$$\mathcal{L} = \{X \in \Sigma_{\rho}^* : (\exists Y \in \Sigma_{\rho}^*)((X, Y) \in \rho)\}.$$

Dimostrazione. Se $\mathcal{L} \in \mathbf{NP}$ esistono una TM \mathfrak{M} ed una funzione polinomiale p tali che \mathfrak{M} riconosce \mathcal{L} in tempo p . Questo significa che una stringa X appartiene a \mathcal{L} se e solo se esiste una computazione (propria) di \mathfrak{M} con input X la cui lunghezza è al più $p(|X|)$. È possibile rappresentare ogni computazione propria Γ di \mathfrak{M} mediante un stringa $\tilde{\Gamma}$, su un opportuno alfabeto Σ , in modo tale che la seguente relazione ρ sia polinomialmente bilanciata e verificabile in tempo deterministico polinomiale:

$$\rho = \{(\text{Input}(\Gamma), \tilde{\Gamma}) : \text{“} \Gamma \text{ è una computazione propria di } \mathfrak{M} \text{” AND } |\Gamma| \leq p(\text{Input}(\Gamma))\}.$$

Si osservi quindi che una stringa X appartiene a \mathcal{L} se e solo se esiste una stringa Y tale che $(X, Y) \in \rho$.

Viceversa, supponiamo che esiste una relazione ρ per cui valgono le condizioni (1), (2) e (3) e sia \mathfrak{P} un verificatore polinomiale per ρ . Allora una TM \mathfrak{M} può riconoscere \mathcal{L} in tempo nondeterministico polinomiale come segue. Partendo da una stringa input X , \mathfrak{M} si sposta alla destra di X , stampa il simbolo separatore $/$ e “genera nondeterministicamente” una stringa Y (si veda l’Osservazione 3 seguente). Successivamente \mathfrak{M} verifica se la coppia (X, Y) appartiene a ρ utilizzando il verificatore \mathfrak{P} con l’input X/Y . □

Osservazione 3. Si osservi che le macchine di Turing possono essere usate anche come *generatori di linguaggi*, oltre che come riconoscitori degli stessi.

Una TM \mathfrak{M} GENERA (NONDETERMINISTICAMENTE) UNA STRINGA X se $\varepsilon \vdash_{\mathfrak{M}} X$.

Ad esempio, sia Σ una alfabeto. La seguente NDTM genera tutte (e sole) le stringhe su Σ :

$$\mathfrak{M} = \{\mathbf{q}_0 \square \square \mathbf{q}_1, \mathbf{q}_0 \square x \mathbf{q}_1, \mathbf{q}_1 x \vdash \mathbf{q}_0, \mathbf{q}_1 \square \uparrow \mathbf{q}_2, \mathbf{q}_2 x \uparrow \mathbf{q}_2, \mathbf{q}_2 \square \vdash \mathbf{q}_3 : x \in \Sigma\}.$$

Si osservi che \mathfrak{M} genera ogni stringa $X \in \Sigma^*$ in tempo $O(|X|)$. ■

8 NP-Completezza

Definizione 16. Siano \mathcal{L}_1 e \mathcal{L}_2 linguaggi. Una RIDUZIONE DA \mathcal{L}_1 A \mathcal{L}_2 è una funzione $f : \Sigma_1^* \rightarrow \Sigma_2^*$, dove Σ_1 è l'alfabeto di \mathcal{L}_1 e Σ_2 è l'alfabeto di \mathcal{L}_2 , tale che $X \in \mathcal{L}_1 \Leftrightarrow f(X) \in \mathcal{L}_2$ per ogni stringa $X \in \Sigma_1^*$.

Una RIDUZIONE POLINOMIALE DA \mathcal{L}_1 A \mathcal{L}_2 è una riduzione f da \mathcal{L}_1 a \mathcal{L}_2 tale che f è computabile in tempo polinomiale.

Scriviamo $\mathcal{L}_1 \leq_p \mathcal{L}_2$ se esiste una riduzione polinomiale da \mathcal{L}_1 a \mathcal{L}_2 . ■

Definizione 17. Un linguaggio \mathcal{L} è NP-HARD se $\mathcal{S} \leq_p \mathcal{L}$ per ogni linguaggio $\mathcal{S} \in \text{NP}$. \mathcal{L} è NP-COMPLETO se \mathcal{L} è NP-hard e $\mathcal{L} \in \text{NP}$. ■

Teorema 14. Siano \mathcal{L}_1 e \mathcal{L}_2 linguaggi. Se $\mathcal{L}_1 \leq_p \mathcal{L}_2$ e $\mathcal{L}_2 \in \mathbf{P}$ allora $\mathcal{L}_1 \in \mathbf{P}$.

Dimostrazione. Sia f una riduzione polinomiale da \mathcal{L}_1 a \mathcal{L}_2 . Per il Teorema 7 la funzione caratteristica $C_{\mathcal{L}_2}$ di \mathcal{L}_2 è computabile in tempo polinomiale e quindi, per il Teorema 12, è computabile in tempo polinomiale la funzione composta $C_{\mathcal{L}_2} \circ f$. A questo punto si osservi che $C_{\mathcal{L}_2} \circ f$ è la funzione caratteristica di \mathcal{L}_1 . □

Teorema 15. Se esiste un linguaggio NP-completo \mathcal{L} tale che $\mathcal{L} \in \mathbf{P}$ allora $\mathbf{P} = \text{NP}$.

Dimostrazione. Sia $\mathcal{L} \in \mathbf{P}$ tale che \mathcal{L} è NP-completo e sia $\mathcal{S} \in \text{NP}$. Poiché \mathcal{L} è NP-completo si ha che $\mathcal{S} \leq_p \mathcal{L}$ e quindi $\mathcal{S} \in \mathbf{P}$ per il Teorema 14. Pertanto $\text{NP} \subseteq \mathbf{P}$. □

Teorema 16. Siano $\mathcal{L}_1, \mathcal{L}_2$ e \mathcal{L}_3 linguaggi. Se $\mathcal{L}_1 \leq_p \mathcal{L}_2$ e $\mathcal{L}_2 \leq_p \mathcal{L}_3$ allora $\mathcal{L}_1 \leq_p \mathcal{L}_3$.

Dimostrazione. Sia g una riduzione polinomiale da \mathcal{L}_1 a \mathcal{L}_2 e sia f una riduzione polinomiale da \mathcal{L}_2 a \mathcal{L}_3 . Poiché g ed f sono computabili in tempo polinomiale, per il Teorema 12, la funzione composta $h = f \circ g$ è computabile in tempo polinomiale. Si osservi quindi che h è una riduzione da \mathcal{L}_1 a \mathcal{L}_3 . □

Teorema 17 (Tecnica di dimostrazione di NP-completezza (1)). Siano \mathcal{L}_1 e \mathcal{L}_2 linguaggi. Se \mathcal{L}_1 è NP-completo e $\mathcal{L}_1 \leq_p \mathcal{L}_2$ allora \mathcal{L}_2 è NP-hard.

Dimostrazione. Segue dal Teorema 16. □

Teorema 18 (Tecnica di dimostrazione di NP-completezza (2)). Siano \mathcal{L}_1 e \mathcal{L}_2 linguaggi. Se \mathcal{L}_1 è NP-completo, $\mathcal{L}_1 \leq_p \mathcal{L}_2$ e $\mathcal{L}_2 \in \text{NP}$, allora \mathcal{L}_2 è NP-completo.

Dimostrazione. Segue dal Teorema 17. □

9 Problema della soddisfacibilità (SAT)

Siano fissati:

(1) Un insieme di LETTERE PROPOSIZIONALI (ATOMI)

$$P_0 \quad P_1 \quad P_2 \quad \dots$$

(2) Il SIMBOLO DI NEGAZIONE

¬

Definizione 18. Un LETTERALE è o un atomo \mathcal{P}_i oppure la NEGAZIONE DI UN ATOMO $\neg\mathcal{P}_j$.

Una CLAUSOLA è un insieme finito $C = \{\ell_0, \ell_1, \dots, \ell_m\}$ di letterali.

Un'ASSEGNAMENTO (O VALUTAZIONE) è una funzione $v : \{\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \dots\} \rightarrow \{\text{True}, \text{False}\}$ che associa ad ogni atomo \mathcal{P}_i il VALORE DI VERITÀ $v(\mathcal{P}_i)$ (“vero” o “falso”). ■

Definizione 19. Sia v un assegnamento.

(1) Per ogni atomo \mathcal{P}_i poniamo

$$(\mathcal{P}_i)^v = v(\mathcal{P}_i) \quad \text{e} \quad (\neg\mathcal{P}_i)^v = \begin{cases} \text{True,} & \text{se } v(\mathcal{P}_i) = \text{False} \\ \text{False,} & \text{altrimenti.} \end{cases}$$

(2) v SODDISFA UNA CLAUSOLA C se esiste un letterale $\ell \in C$ tale che $(\ell)^v = \text{True}$.

(3) v SODDISFA UN INSIEME DI CLAUSOLE S se v soddisfa ogni clausola di S . ■

Definizione 20. Un INSIEME DI CLAUSOLE È SODDISFACIBILE se esiste un assegnamento che lo soddisfa. ■

Definizione 21. Il PROBLEMA SAT consiste nel determinare se un dato insieme finito di clausole è soddisfacibile. ■

Rappresenteremo formalmente il problema SAT con il linguaggio \mathcal{L}_{SAT} descritto di seguito.

L'alfabeto Σ_{SAT} di \mathcal{L}_{SAT} è costituito dai seguenti simboli:

$$\mathbf{a} \quad \mathbf{b} \quad \mathbf{1} \quad /$$

Per $i = 0, 1, 2, \dots$, rappresentiamo i letterali \mathcal{P}_i e $\neg\mathcal{P}_i$ con le stringhe $\alpha_i = \mathbf{a}^{\lceil i \rceil}$ e $\beta_i = \mathbf{b}^{\lceil i \rceil}$, rispettivamente, dove $\lceil i \rceil$ è la stringa formata dalla concatenazione di i copie del simbolo $\mathbf{1}$. (Si veda la Sezione 4).

Ogni stringa X della forma $/Y_0Y_1 \dots Y_m$, dove le stringhe Y_0, Y_1, \dots, Y_m sono le rappresentazioni dei letterali $\ell_0, \ell_1, \dots, \ell_m$, rispettivamente, rappresenta la clausola $\{\ell_0, \ell_1, \dots, \ell_m\}$. (Si osservi che ogni clausola C è rappresentata da molteplici stringhe.) La stringa $/$ rappresenta la CLAUSOLA VUOTA.

Ogni stringa W della forma $/X_0/X_1/\dots/X_p$, dove le stringhe $/X_0, /X_1, \dots, /X_p$ sono rappresentazioni delle clausole C_0, C_1, \dots, C_p , rispettivamente, rappresenta l'insieme di clausole $\{C_0, C_1, \dots, C_p\}$. (Quindi ogni insieme di clausole è rappresentato da molteplici stringhe.)

Definizione 22. Definiamo formalmente \mathcal{L}_{SAT} come il linguaggio costituito da tutte le stringhe $W \in \Sigma_{\text{SAT}}^*$ tali che W rappresenta un insieme di clausole soddisfacibile. ■

Esempio 12. Le stringhe $/\mathbf{ab11a1}$ e $/\mathbf{alab11}$ rappresentano entrambe la stessa clausola $\{\mathcal{P}_0, \mathcal{P}_1, \neg\mathcal{P}_2\}$. Similmente, la stringa $/\mathbf{bb1a}$ rappresenta la clausola $\{\mathcal{P}_0, \neg\mathcal{P}_0, \neg\mathcal{P}_1\}$. La stringa $/\mathbf{1ab11}$ non rappresenta alcuna clausola. La stringa $W = /ab11a1/a111/bb1a$ rappresenta l'insieme di clausole $\{\{\mathcal{P}_0, \mathcal{P}_1, \neg\mathcal{P}_2\}, \{\mathcal{P}_3\}, \{\mathcal{P}_0, \neg\mathcal{P}_0, \neg\mathcal{P}_1\}\}$. Si osservi che $W \in \mathcal{L}_{\text{SAT}}$ dato che ogni assegnamento v per cui $v(\mathcal{P}_0) = v(\mathcal{P}_3) = \text{True}$ soddisfa l'insieme di clausole che rappresenta W . ■

Teorema 19. $\mathcal{L}_{\text{SAT}} \in \text{NP}$.

Per dimostrare che $\mathcal{L}_{\text{SAT}} \in \text{NP}$ usiamo il Teorema 13. Preliminarmente, introduciamo le seguenti definizioni.

Definizione 23. Sia S un insieme di clausole.

(1) Il **SUPPORTO** DI S è l'insieme $\text{atoms}(S)$ degli atomi che “figurano” nelle clausole di S , cioè

$$\text{atoms}(S) = \{A \in \{\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \dots\} : (\exists C \in S)(\{A, \neg A\} \cap C \neq \emptyset)\}.$$

(2) Un **INSIEME** H DI **ATOMI** È UN **MODELLO** DI S se:

(2.a) $H \subseteq \text{atoms}(S)$;

(2.b) l'assegnamento v_H definito come segue soddisfa S :

$$v_H(\mathcal{P}_i) = \begin{cases} \text{True}, & \text{se } \mathcal{P}_i \in H \\ \text{False}, & \text{altrimenti,} \end{cases}$$

per ogni atomo \mathcal{P}_i . ■

Definizione 24. Una stringa $X \in \Sigma_{\text{SAT}}^*$ **RAPPRESENTA STRETTAMENTE** un insieme H di atomi se X ha la forma $Y_0Y_1 \cdots Y_m$ dove ogni stringa Y_i rappresenta un atomo (cioè $Y_i \in \{\alpha_0, \alpha_1, \dots\}$) e $Y_j \neq Y_k$, per ogni $j, k = 0, 1, \dots, m$ con $j \neq k$.

Sia ρ_{SAT} l'insieme delle coppie ordinate (W, X) di stringe su Σ_{SAT} tali che W rappresenta un insieme S di clausole e X rappresenta strettamente un insieme H di atomi tali che H è un modello di S . ■

Dalle precedenti definizioni segue che $W \in \mathcal{L}_{\text{SAT}}$ se e solo se esiste una stringa $X \in \Sigma_{\text{SAT}}^*$ tale che $(W, X) \in \rho_{\text{SAT}}$, per ogni stringa $W \in \Sigma_{\text{SAT}}^*$. Inoltre ρ_{SAT} è polinomialmente bilanciata. Si può ulteriormente dimostrare che ρ_{SAT} è verificabile in tempo deterministico polinomiale. Per il Teorema 13 si ha quindi che $\mathcal{L}_{\text{SAT}} \in \text{NP}$.

Teorema 20. \mathcal{L}_{SAT} è **NP-hard**.

Dimostrazione. Vedi la Sezione 10. □

Teorema 21 (Teorema di Cook). \mathcal{L}_{SAT} è **NP-completo**.

Dimostrazione. Segue immediatamente dai teoremi 19 e 20. □

10 Dimostrazione del Teorema di Cook

In questa sezione dimostriamo il Teorema 20.

Preliminari. Nel seguito useremo le seguenti notazioni:

- (1) Per ogni insieme finito I indichiamo con $|I|$ la **CARDINALITÀ** di I .
- (2) Un **ORDINAMENTO** di un insieme finito I è una funzione biiettiva $\theta : \{0, 1, \dots, |I|-1\} \rightarrow I$.
- (3) Per ogni insieme finito I , ogni ordinamento θ di I e ogni $x \in I$, l'**INDICE** DI x NELL'**ORDINAMENTO** θ è il numero $i \in \{0, 1, \dots, |I|-1\}$ tale che $\theta(i) = x$.
- (4) Per ogni $a, b, c \in \mathbb{N}$, l'**ORDINAMENTO LESSICOGRAFICO** dell'insieme di terne

$$I = \{0, \dots, a\} \times \{0, \dots, b\} \times \{0, \dots, c\}$$

è l'ordinamento θ di I definito come segue:

(a) $\theta(0) = (0, 0, 0)$;

(b)

$$\theta(n+1) = \begin{cases} (i, j, k+1), & \text{se } k < c \\ (i, j+1, 0), & \text{se } k = c \text{ e } j < b \\ (i+1, 0, 0), & \text{altrimenti,} \end{cases}$$

dove $(i, j, k) = \theta(n)$.

- (5) Per ogni stringa X e ogni indice i tale che $0 \leq i < |X|$, indichiamo con $(X)_i$ l' $(i+1)$ -esimo simbolo di X da sinistra verso destra.
- (6) Per ogni configurazione $\gamma = XqtY$ poniamo $\widehat{\gamma} = XtY$ (cioè $\widehat{\gamma}$ è la stringa ottenuta da γ eliminando da essa il suo simbolo di stato q).
- (7) Per ogni insieme finito $S = \{C_0, C_1, \dots, C_p\}$ di clausole, la DIMENSIONE DI S è la quantità $\text{size}(S) = \sum_{0 \leq i \leq p} |C_i|$.
- (8) Per ogni insieme finito A di atomi, poniamo

$$\nabla A = \{\{-a, -b\} : a, b \in A \text{ AND } a \neq b\} \cup \{A\}.$$

(Si osservi che:

(8.a) $\text{size}(\nabla A) = |A|(|A| - 1) + |A| = |A|^2$;

(8.b) per ogni assegnamento v , v soddisfa l'insieme di clausole ∇A se e solo se esiste un unico atomo $a \in A$ tale che $v(a) = \text{True}$.)

- (9) Per ogni coppia S_1, S_2 di insiemi di clausole poniamo

$$S_1 \wedge S_2 = S_1 \cup S_2 \quad (\text{CONGIUNZIONE di } S_1 \text{ e } S_2)$$

e

$$S_1 \vee S_2 = \{C \cup K : C \in S_1, K \in S_2\} \quad (\text{DISGIUNZIONE di } S_1 \text{ e } S_2).$$

(Si osservi che:

(9.a) le operazioni \wedge e \vee sono commutative e associative;

(9.b) $\text{size}(S_1 \wedge S_2) \leq \text{size}(S_1) + \text{size}(S_2)$;

(9.c) $\text{size}(S_1 \vee S_2) \leq |S_2| \text{size}(S_1) + |S_1| \text{size}(S_2)$;

(9.d) per ogni assegnamento v , v soddisfa $S_1 \wedge S_2$ se e solo se v soddisfa S_1 e v soddisfa S_2 ;

(9.e) per ogni assegnamento v , v soddisfa $S_1 \vee S_2$ se e solo se v soddisfa S_1 o v soddisfa S_2 ;

(9.f) se F_0, F_1, \dots, F_n sono insiemi di "clausole unitarie", cioè ogni clausola di ogni insieme contiene un solo letterale, allora $\text{size}(\bigvee_{0 \leq i \leq n} F_i) \leq (n+1) \cdot \prod_{0 \leq i \leq n} |F_i|$. ■

Definizione 25. Una TM \mathfrak{M} è in FORMA CANONICA se:

- (1) gli stati di \mathfrak{M} sono $\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_m$, per qualche $m > 0$;
- (2) i simboli dell'alfabeto di \mathfrak{M} sono $\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_r$, per qualche $r \geq 0$;

- (3) per ogni $(q, t) \in \{\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_{m-1}\} \times (\{\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_r\} \cup \{\square\})$ esiste almeno una istruzione di \mathfrak{M} che comincia con qt ;
- (4) per ogni $t \in \{\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_r\} \cup \{\square\}$ nessuna istruzione di \mathfrak{M} comincia con $\mathbf{q}_m t$. ■

Teorema 22. *Sia \mathcal{L} un linguaggio. $\mathcal{L} \in \mathbf{NP}$ se e solo se esistono una TM \mathfrak{M} in forma canonica ed una funzione polinomiale semplice p tali che:*

- (1) \mathcal{L} è accettato da \mathfrak{M} ;
- (2) per ogni stringa $X \in \mathcal{L}$ si ha che $\mathfrak{M}(X) \downarrow^{(\leq p(|X|))}$.

Definizione 26. Sia \mathfrak{M} una TM.

- (1) Un n -ARRAY DI \mathfrak{M} , dove $n \in \mathbb{N}$, è una sequenza finita $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ di $n + 1$ configurazioni di \mathfrak{M} tali che $|\gamma_0| = |\gamma_1| = \dots = |\gamma_n| = 2n + 2$.
- (2) Una n -COMPUTAZIONE DI \mathfrak{M} , dove $n \in \mathbb{N}$, è un n -array $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ di \mathfrak{M} soddisfacente le seguenti proprietà:
- (2.a) esistono stringhe $U, V \in \{\square\}^*$ tali che $|U| \geq n$ e $|V| \geq n+1 - |\langle \gamma_0 \rangle|$ e $\gamma_0 = U \mathbf{q}_0 \langle \gamma_0 \rangle V$;
- (2.b) per ogni $i = 0, 1, \dots, n-1$ almeno una delle condizioni “ $\gamma_i \vdash_{\mathfrak{M}} \gamma_{i+1}$ ” o “ $\gamma_{i+1} = \gamma_i$ ” è vera (cioè ogni configurazione nella sequenza è uguale a quella precedente oppure consegue da quest’ultima mediante l’esecuzione di una istruzione di \mathfrak{M});
- (2.c) γ_n è una configurazione terminale rispetto a \mathfrak{M} .

L’INPUT di Γ è $\mathbf{Input}(\Gamma) = \langle \gamma_0 \rangle$; la LUNGHEZZA di Γ è $|\Gamma| = n$. (Si osservi che se Γ è una n -COMPUTAZIONE di \mathfrak{M} allora $|\Gamma| \geq |\mathbf{Input}(\Gamma)| - 1$). ■

Teorema 23. *Sia \mathfrak{M} una TM e siano $X \in \mathbf{S}^*$ e $n \in \mathbb{N}$ tali che $n \geq |X| - 1$. Allora,*

$\mathfrak{M}(X) \downarrow^{(\leq n)}$ se e solo se esiste una n -computazione Γ di \mathfrak{M} tale che $\mathbf{Input}(\Gamma) = X$.

Teorema 24. *Sia \mathcal{L} un linguaggio. $\mathcal{L} \in \mathbf{NP}$ se e solo se esistono una TM \mathfrak{M} in forma canonica ed una funzione polinomiale semplice p tali che*

$$\mathcal{L} = \{X \in \mathbf{S}(\mathfrak{M})^* : (\exists \Gamma) (\text{“} \Gamma \text{ è una } p(|X|)\text{-computazione di } \mathfrak{M} \text{ con input } X \text{”})\}.$$

Sia adesso \mathfrak{M} una TM in forma canonica e siano $\{\mathbf{q}_0, \mathbf{q}_1, \dots, \mathbf{q}_m\}$ l’insieme degli stati di \mathfrak{M} e $\{\mathbf{s}_0, \mathbf{s}_1, \dots, \mathbf{s}_r\}$ l’alfabeto di \mathfrak{M} .

“Costruiamo” la funzione $\delta_{\mathfrak{M}}$ che associa ad ogni coppia (X, n) , dove X è una stringa sull’alfabeto di \mathfrak{M} ed n è un numero intero maggiore o uguale a $|X| - 1$, l’insieme di clausole $\delta_{\mathfrak{M}}(X, n)$ tale che $\delta_{\mathfrak{M}}(X, n)$ è soddisfacibile se e solo se $\mathfrak{M}(X) \downarrow^{(\leq n)}$.² (Difatti, $\delta_{\mathfrak{M}}(X, n)$ è soddisfacibile se e solo se esistono n -computazioni di \mathfrak{M} con input X (cf. Teorema 23).)

Definizione 27. Per ogni $n \in \mathbb{N}$ siano:

$$\mathbb{A}(n) = \{(i, j, k) : 0 \leq i \leq r + 1, 0 \leq j \leq 2n, 0 \leq k \leq n\}$$

e

$$\mathbb{B}(n) = \{(h, j, k) : 0 \leq h \leq m, 0 \leq j \leq 2n, 0 \leq k \leq n\}.$$

■

²Si veda “Martin D. Davis, Ron Sigal, Elaine J. Weyuker, *Computability, Complexity and Languages*”.

Definizione 28. Per ogni $n \in \mathbb{N}$, $(i, j, k) \in \mathbb{A}(n)$ e $(h, j, k) \in \mathbb{B}(n)$ siano:

- (1) $\alpha_{[i,j,k]}^n$ il $(p+1)$ -esimo atomo della lista $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \dots$, dove p è l'indice della terna (i, j, k) nell'ordinamento lessicografico di $\mathbb{A}(n)$;
- (2) $\beta_{[h,j,k]}^n$ il $(p+1)$ -esimo atomo della lista $\mathcal{P}_\ell, \mathcal{P}_{\ell+1}, \mathcal{P}_{\ell+2}, \dots$ dove p è l'indice della terna (h, j, k) nell'ordinamento lessicografico di $\mathbb{B}(n)$ e $\ell = (r+2)(2n+1)(n+1)$;
- (3) $\mathcal{A}_n = \{\alpha_{[i,j,k]}^n : (i, j, k) \in \mathbb{A}(n)\}$ e $\mathcal{B}_n = \{\beta_{[h,j,k]}^n : (h, j, k) \in \mathbb{B}(n)\}$.
(Così \mathcal{A}_n è l'insieme dei primi $(r+2)(2n+1)(n+1)$ atomi della lista $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \dots$ e \mathcal{B}_n è l'insieme dei successivi $(m+1)(2n+1)(n+1)$ atomi della stessa lista.) ■

$\delta_{\mathfrak{M}}(X, n)$ avrà come supporto³ l'insieme di atomi $\mathcal{U}_n = \mathcal{A}_n \cup \mathcal{B}_n$.

Definizione 29. Per ogni $n \in \mathbb{N}$ e ogni n -array $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ di \mathfrak{M} sia v^Γ l'assegnamento definito come segue:

- (1) per ogni atomo $a \notin \mathcal{U}_n$: $v^\Gamma(a) = \text{False}$;
- (2) per ogni $(i, j, k) \in \mathbb{A}(n)$:

$$v^\Gamma(\alpha_{[i,j,k]}^n) = \begin{cases} \text{True}, & \text{se } (\widehat{\gamma}_k)_j = \mathbf{s}_i \\ \text{False}, & \text{altrimenti} \end{cases} \quad \text{e} \quad v^\Gamma(\alpha_{[r+1,j,k]}^n) = \begin{cases} \text{True}, & \text{se } (\widehat{\gamma}_k)_j = \square \\ \text{False}, & \text{altrimenti;} \end{cases}$$

- (3) per ogni $(h, j, k) \in \mathbb{B}(n)$:

$$v^\Gamma(\beta_{[h,j,k]}^n) = \begin{cases} \text{True}, & \text{se } (\gamma_k)_j = \mathbf{q}_h \\ \text{False}, & \text{altrimenti.} \end{cases}$$

■

$\delta_{\mathfrak{M}}(X, n)$ soddisferà le seguenti condizioni:

- (cond.1) per ogni n -array Γ di \mathfrak{M} ,
“ Γ è una n -computazione di \mathfrak{M} con input X ” \Leftrightarrow “ v^Γ soddisfa $\delta_{\mathfrak{M}}(X, n)$ ”;
 - (cond.2) se v è un assegnamento che soddisfa $\delta_{\mathfrak{M}}(X, n)$, esiste un unico n -array Γ di \mathfrak{M} tale che $v(a) = v^\Gamma(a)$, per ogni atomo $a \in \mathcal{U}_n$.
- (E quindi $\delta_{\mathfrak{M}}(X, n)$ è soddisfacibile se e solo se $\mathfrak{M}(X) \downarrow^{(\leq n)}$.)

Si osservino i seguenti fatti.

- (1) Le condizioni “ $v^\Gamma(\alpha_{[i,j,k]}^n) = \text{True}$ ” e “ $v^\Gamma(\beta_{[h,j,k]}^n) = \text{True}$ ” possono essere espresse, rispettivamente, come segue:
 - (1.a) “ nella $(k+1)$ -esima configurazione di Γ , la macchina \mathfrak{M} esamina la $(j+1)$ -esima casella del nastro da destra verso sinistra e questa casella contiene il simbolo \mathbf{s}_i , ovvero la casella è vuota se $i = r+1$ ” e

³cf. Definizione 23.

(1.b) “ nella $(k + 1)$ -esima configurazione di Γ , la macchina \mathfrak{M} si trova nello stato interno \mathbf{q}_h ed esamina la $(j + 1)$ -esima casella del nastro da destra verso sinistra ”.

(2) Per ogni n -array Γ di \mathfrak{M} :

(2.a) per ogni $(j, k) \in \{0, \dots, 2n\} \times \{0, \dots, n\}$ esiste un unico $i \in \{0, \dots, r + 1\}$ tale che $v^\Gamma(\boldsymbol{\alpha}_{[i,j,k]}^n) = \text{True}$ (difatti $(\widehat{\gamma}_k)_j = \mathbf{s}_i$, ovvero $(\widehat{\gamma}_k)_j = \square$ nel caso $i = r + 1$);

(2.b) per ogni $k \in \{0, \dots, n\}$ esiste un'unica coppia $(h, j) \in \{0, \dots, m\} \times \{0, \dots, 2n\}$ tale che $v^\Gamma(\boldsymbol{\beta}_{[h,j,k]}^n) = \text{True}$ (difatti $(\gamma_k)_j = \mathbf{q}_h$).

(3) Per ogni Γ_1 e Γ_2 n -array di \mathfrak{M} , se $v^{\Gamma_1} = v^{\Gamma_2}$ allora $\Gamma_1 = \Gamma_2$ (come segue dalle proprietà precedenti).

Introduciamo adesso quattro insiemi di clausole la cui unione costituisce $\delta_{\mathfrak{M}}(X, n)$. I primi tre insiemi “forzano” la condizione (cond.1); l'altro la condizione (cond.2).

Insieme di clausole 1. Sia

$$\text{Terminal}_n = \{\{\boldsymbol{\beta}_{[m,j,n]}^n : j \in \{0, \dots, 2n\}\}\}.$$

Si osservi che:

(1) Per ogni n -array Γ di \mathfrak{M} , v^Γ soddisfa Terminal_n se e solo se l'ultima configurazione di Γ è terminale rispetto a \mathfrak{M} .

(2) $\text{size}(\text{Terminal}_n) = 2n + 1 = \mathcal{O}(n)$.

Insieme di clausole 2. Sia Initial_n^X l'insieme di clausole definito come segue

(1) se $X = \varepsilon$ allora

$$\text{Initial}_n^X = \{\{\boldsymbol{\alpha}_{[r+1,j,0]}^n : 0 \leq j \leq 2n\} \cup \{\{\boldsymbol{\beta}_{[0,n,0]}^n\}\}\};$$

(2) se $X = \mathbf{s}_{i_0}\mathbf{s}_{i_1} \cdots \mathbf{s}_{i_z}$, dove $z \geq 0$, allora

$$\begin{aligned} \text{Initial}_n^X = & \{\{\boldsymbol{\alpha}_{[r+1,j,0]}^n : 0 \leq j < n\} \cup \\ & \{\{\boldsymbol{\alpha}_{[i_j,n+j,0]}^n : 0 \leq j \leq z\} \cup \\ & \{\{\boldsymbol{\alpha}_{[r+1,n+z+1+j,0]}^n : 0 \leq j \leq n - z - 1\} \cup \\ & \{\{\boldsymbol{\beta}_{[0,n,0]}^n\}\}\}. \end{aligned}$$

Si osservi che:

(1) Per ogni n -array Γ di \mathfrak{M} , v^Γ soddisfa Initial_n^X se e solo se la prima configurazione di Γ è $U\mathbf{q}_0XV$ dove U e V sono le stringhe su $\{\square\}$ di lunghezze n e $n + 1 - |X|$, rispettivamente.

(2) $\text{size}(\text{Initial}_n^X) = 2n + 2 = \mathcal{O}(n)$.

Insieme di clausole 3. Sia

$$\begin{aligned} \mathfrak{M} = & \{\mathbf{q}_{i_a} t_{j_a} t_{k_a} \mathbf{q}_{l_a} : a \in \{0, 1, \dots, \bar{a}\}\} \cup \\ & \{\mathbf{q}_{i_b} t_{j_b} \ulcorner \mathbf{q}_{l_b} : b \in \{0, 1, \dots, \bar{b}\}\} \cup \\ & \{\mathbf{q}_{i_c} t_{j_c} \lrcorner \mathbf{q}_{l_c} : c \in \{0, 1, \dots, \bar{c}\}\} \end{aligned}$$

dove $t_0 = \mathbf{s}_0, t_1 = \mathbf{s}_1, \dots, t_r = \mathbf{s}_r, t_{r+1} = \square$.

Poniamo

$$\text{Cons}_n = \bigwedge_{0 \leq k < n} \bigwedge_{0 \leq j \leq 2n} (\text{NotHead}(j, k) \vee \text{Ident}(j, k) \vee \text{Print}(j, k) \vee \text{Right}(j, k) \vee \text{Left}(j, k)),^4$$

dove

$$(1) \text{NotHead}(j, k) = \left(\bigvee_{0 \leq i \leq r+1} \{ \{\alpha_{[i,j,k]}^n\}, \{\alpha_{[i,j,k+1]}^n\} \} \right) \wedge \{ \{ \neg \beta_{[h,j,k]}^n \} : h \in \{0, \dots, m\} \};$$

$$(2) \text{Ident}(j, k) = \bigvee_{0 \leq h \leq m} \bigvee_{0 \leq i \leq r+1} \{ \{\alpha_{[i,j,k]}^n\}, \{\alpha_{[i,j,k+1]}^n\}, \{\beta_{[h,j,k]}^n\}, \{\beta_{[h,j,k+1]}^n\} \};$$

$$(3) \text{Print}(j, k) = \bigvee_{0 \leq a \leq \bar{a}} \{ \{\alpha_{[j_a,j,k]}^n\}, \{\alpha_{[k_a,j,k+1]}^n\}, \{\beta_{[i_a,j,k]}^n\}, \{\beta_{[l_a,j,k+1]}^n\} \};$$

$$(4) \text{Right}(j, k) = \begin{cases} \bigvee_{0 \leq b \leq \bar{b}} \{ \{\alpha_{[j_b,j,k]}^n\}, \{\alpha_{[j_b,j,k+1]}^n\}, \{\beta_{[i_b,j,k]}^n\}, \{\beta_{[l_b,j+1,k+1]}^n\} \}, & \text{se } j < 2n \\ \{ \{ \} \}, & \text{altrimenti;} \end{cases}$$

$$(5) \text{Left}(j, k) = \begin{cases} \bigvee_{0 \leq c \leq \bar{c}} \{ \{\alpha_{[j_c,j,k]}^n\}, \{\alpha_{[j_c,j,k+1]}^n\}, \{\beta_{[i_c,j,k]}^n\}, \{\beta_{[l_c,j-1,k+1]}^n\} \}, & \text{se } j > 0 \\ \{ \{ \} \}, & \text{altrimenti;} \end{cases}$$

Si osservi che:

(1) Per ogni n -array $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ di \mathfrak{M} , v^Γ soddisfa Cons_n se e solo se “ $\gamma_i \vdash_{\mathfrak{M}} \gamma_{i+1}$ ” o “ $\gamma_{i+1} = \gamma_i$ ”, per ogni $i = 0, 1, \dots, n-1$.

(2) $\text{size}(\text{NotHead}(j, k)) \leq (r+2)2^{r+2} + (m+1)$.

(3) $\text{size}(\text{Ident}(j, k)) \leq (m+1)(r+2)4^{(m+1)(r+2)}$.

(4) $\text{size}(\text{Print}(j, k)) \leq (\bar{a}+1)4^{\bar{a}+1}$.

(5) $\text{size}(\text{Right}(j, k)) \leq (\bar{b}+1)4^{\bar{b}+1}$.

(6) $\text{size}(\text{Left}(j, k)) \leq (\bar{c}+1)4^{\bar{c}+1}$.

(7) $\text{size}(\text{Cons}_n) = O(n^2)$ (poiché r ed m sono costanti).

⁴Se $n = 0$ allora $\text{Cons}_n = \emptyset$.

Insieme di clausole 4. Sia

$$\text{Unique}_n = \text{USS}_n \wedge \text{String}_n$$

dove

$$\text{USS}_n = \bigwedge_{0 \leq k \leq n} \nabla \{ \beta_{[h,j,k]}^n : 0 \leq h \leq m, 0 \leq j \leq 2n \}$$

e

$$\text{String}_n = \bigwedge_{0 \leq k \leq n} \bigwedge_{0 \leq j \leq 2n} \nabla \{ \alpha_{[i,j,k]}^n : 0 \leq i \leq r+1 \}.$$

Si osservi che:

(1) Per ogni assegnamento v ,

(1.1) v soddisfa Unique_n se e solo se valgono le seguenti condizioni:

(1.1.1) per ogni $(j, k) \in \{0, \dots, 2n\} \times \{0, \dots, n\}$ esiste un unico $i = \mathbf{e}(j, k) \in \{0, \dots, r+1\}$ tale che $v(\alpha_{[i,j,k]}^n) = \text{True}$;

(1.1.2) per ogni $k \in \{0, \dots, n\}$ esiste un'unica coppia $(h, j) = (\mathbf{f}(k), \mathbf{g}(k)) \in \{0, \dots, m\} \times \{0, \dots, 2n\}$ tale che $v(\beta_{[h,j,k]}^n) = \text{True}$;

e quindi

(1.2) v soddisfa Unique_n se e solo se esiste un unico n -array Γ di \mathfrak{M} tale che $v(a) = v^\Gamma(a)$ vale per ogni atomo $a \in \mathcal{U}_n$, e precisamente $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ dove

$$(\widehat{\gamma}_k)_j = \begin{cases} \mathbf{s}_{\mathbf{e}(j,k)}, & \text{se } \mathbf{e}(j,k) \neq r+1 \\ \square, & \text{altrimenti} \end{cases} \quad \text{e} \quad (\gamma_k)_{\mathbf{g}(k)} = \mathbf{q}_{\mathbf{f}(k)},$$

per ogni $(j, k) \in \{0, \dots, 2n\} \times \{0, \dots, n\}$.

(2) $\text{size}(\text{USS}_n) \leq (n+1)[(m+1)(2n+1)]^2 = \mathcal{O}(n^3)$.

(3) $\text{size}(\text{String}_n) \leq (n+1)(2n+1)(r+2)^2 = \mathcal{O}(n^2)$.

(4) $\text{size}(\text{Unique}_n) \leq \text{size}(\text{USS}_n) + \text{size}(\text{String}_n) = \mathcal{O}(n^3)$.

Definizione 30. Poniamo

$$\delta_{\mathfrak{M}}(X, n) = \text{Initial}_n^X \wedge \text{Cons}_n \wedge \text{Terminal}_n \wedge \text{Unique}_n.$$

■

Tenendo conto delle osservazioni precedenti, l'insieme di clausole $\delta_{\mathfrak{M}}(X, n)$ appena definito soddisfa le condizioni (cond.1) e (cond.2) e quindi $\delta_{\mathfrak{M}}(X, n)$ è soddisfacibile se e solo se $\mathfrak{M}(X) \downarrow^{(\leq n)}$; inoltre si ha che $\text{size}(\delta_{\mathfrak{M}}(X, n)) = \mathcal{O}(n^3)$. Ulteriormente, poiché

(a) il supporto di $\delta_{\mathfrak{M}}(X, n)$ è \mathcal{U}_n ;

(b) \mathcal{U}_n è formato dai primi $(r+m+3)(2n+1)(n+1)$ atomi della lista $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \dots$;

(c) ogni atomo \mathcal{P}_i è rappresentato da una stringa su Σ_{SAT} di lunghezza $i+1$;

l'insieme $\delta_{\mathfrak{M}}(X, n)$ può essere rappresentato da stringhe su Σ_{SAT} di lunghezza $\mathcal{O}(n^5)$ e data la sua “struttura”, almeno una di queste stringhe può essere computata effettivamente a partire da X e n in tempo proporzionale alla sua stessa lunghezza, e quindi in tempo $\mathcal{O}(n^5)$. Pertanto, se p è una funzione polinomiale semplice, esiste una funzione $\kappa_{\mathfrak{M}}^p : \mathbf{S}(\mathfrak{M})^* \rightarrow \Sigma_{\text{SAT}}^*$ tale che

- (1) $\kappa_{\mathfrak{M}}^p$ è computabile in tempo polinomiale;
- (2) per ogni $X \in \mathbf{S}(\mathfrak{M})^*$, la stringa $\kappa_{\mathfrak{M}}^p(X)$ rappresenta l'insieme di clausole $\delta_{\mathfrak{M}}(X, p(|X|))$ e quindi $\mathfrak{M}(X) \downarrow^{(\leq p(|X|))}$ se e solo se $\kappa_{\mathfrak{M}}^p(X) \in \mathcal{L}_{\text{SAT}}$.

Adesso, sia $\mathcal{L} \in \mathbf{NP}$ e siano \mathfrak{M} una TM in forma canonica e p una funzione polinomiale semplice tali che \mathcal{L} è accettato da \mathfrak{M} e $\mathfrak{M}(X) \downarrow^{(\leq p(|X|))}$, per ogni stringa $X \in \mathcal{L}$. Allora la funzione $\tilde{\kappa} : \Sigma_{\mathcal{L}}^* \rightarrow \Sigma_{\text{SAT}}^*$ definita da

$$\tilde{\kappa}(X) = \begin{cases} \kappa_{\mathfrak{M}}^p(X), & \text{se } X \in \Sigma_{\mathcal{L}}^* \\ \varepsilon, & \text{altrimenti} \end{cases}$$

è una riduzione polinomiale da \mathcal{L} in \mathcal{L}_{SAT} e quindi $\mathcal{L} \leq_p \mathcal{L}_{\text{SAT}}$. Così ogni linguaggio in \mathbf{NP} è polinomialmente riducibile a \mathcal{L}_{SAT} e pertanto \mathcal{L}_{SAT} è \mathbf{NP} -hard.

11 Catalogo di problemi NP-completi

(Si veda “Michael R. Garey, David S. Johnson, *Computers and Intractability*”.)

11.1 Problema 3–SAT

Definizione 31. Una 3-CLAUSOLA è una clausola contenente tre letterali. ■

Definizione 32. Il PROBLEMA 3–SAT consiste nel determinare se un dato insieme finito di 3-clausole è soddisfacibile. ■

Teorema 25. Il problema 3–SAT è NP-completo.

Dimostrazione. Dobbiamo dimostrare che:

- (1) 3–SAT è in NP;
- (2) 3–SAT è NP-hard.

La dimostrazione che 3–SAT è in NP segue da argomentazioni simili a quelle usate per dimostrare che SAT è in NP.

Per dimostrare che 3–SAT è NP-hard usiamo il Teorema 17 e dimostriamo che $\text{SAT} \leq_p 3\text{–SAT}$, cioè che esiste una funzione f che associa ad ogni insieme S di clausole l'insieme $f(S) = \tilde{S}$ di 3-clausole tale che S è soddisfacibile se e solo se \tilde{S} è soddisfacibile; inoltre f deve essere computabile in tempo polinomiale. Procediamo come segue. Sia $S = \{C_1, C_2, \dots, C_n\}$, dove $C_i = \{\ell_i^1, \dots, \ell_i^{|C_i|}\}$ per $i = 1, \dots, n$, e sia $U = \text{atoms}(S)$ il supporto di S . La costruzione di \tilde{S} coinvolge l'introduzione degli insiemi di atomi U_1, U_2, \dots, U_n e degli insiemi di 3-clausole $\tilde{C}_1, \tilde{C}_2, \dots, \tilde{C}_n$ (la coppia (U_i, \tilde{C}_i) corrispondente alla clausola C_i) soddisfacenti alle seguenti condizioni:

- (1) gli insiemi U_1, U_2, \dots, U_n sono a due a due disgiunti;

(2) l'unione di U_1, U_2, \dots, U_n è disgiunta da U ;

(3) per ogni $i = 1, 2, \dots, n$:

$$|U_i| = \begin{cases} 2, & \text{se } |C_i| = 1 \\ 1, & \text{se } |C_i| = 2 \\ 0, & \text{se } |C_i| = 3 \\ |C_i| - 3, & \text{altrimenti;} \end{cases}$$

(4) per ogni $i = 1, 2, \dots, n$:

(4.a) se $|C_i| = 1$, allora $\tilde{C}_i = \{\{\ell_i^1, u_i^1, u_i^2\}, \{\ell_i^1, u_i^1, \neg u_i^2\}, \{\ell_i^1, \neg u_i^1, u_i^2\}, \{\ell_i^1, \neg u_i^1, \neg u_i^2\}\}$;

(4.b) se $|C_i| = 2$ allora $\tilde{C}_i = \{\{\ell_i^1, \ell_i^2, u_i^1\}, \{\ell_i^1, \ell_i^2, \neg u_i^1\}\}$;

(4.c) se $|C_i| = 3$ allora $\tilde{C}_i = \{C_i\}$;

(4.d) se $|C_i| > 3$ allora

$$\tilde{C}_i = \{\{\ell_i^1, \ell_i^2, u_i^1\}\} \cup \{\{\neg u_i^j, \ell_i^{j+2}, u_i^{j+1}\} : 1 \leq j \leq |C_i| - 4\} \cup \{\{\ell_i^{|C_i|-1}, \ell_i^{|C_i|}, \neg u_i^{|C_i|-3}\}\},$$

dove $U_i = \{u_i^1, u_i^2, \dots, u_i^{|U_i|}\}$.⁵

Si definisce quindi $\tilde{S} = \bigcup_{i \in \{1, \dots, n\}} \tilde{C}_i$.

Dimostriamo che S è soddisfacibile se e solo se \tilde{S} è soddisfacibile.

Sia v un assegnamento che soddisfa S . Costruiamo induttivamente la sequenza v_0, \dots, v_n di assegnamenti dove $v_0 = v$ e v_i è definito come segue, per ogni $i = 1, \dots, n$. Se $|C_i| \leq 3$ allora $v_i = v_{i-1}$. Se $|C_i| > 3$ poniamo

$$v_i(a) = \begin{cases} \text{False}, & \text{se } (m = 1 \text{ OR } m = 2) \text{ AND } (a \in \{u_i^1, \dots, u_i^{|C_i|-3}\}) \\ \text{True}, & \text{se } (m = |C_i| - 1 \text{ OR } m = |C_i|) \text{ AND } (a \in \{u_i^1, \dots, u_i^{|C_i|-3}\}) \\ \text{True}, & \text{se } (2 < m < |C_i| - 1) \text{ AND } (a \in \{u_i^1, \dots, u_i^{m-2}\}) \\ \text{False}, & \text{se } (2 < m < |C_i| - 1) \text{ AND } (a \in \{u_i^{m-1}, \dots, u_i^{|C_i|-3}\}) \\ v_{i-1}(a), & \text{altrimenti,} \end{cases}$$

per ogni atomo a , dove $m = \min\{j \in \{1, \dots, |C_i|\} : v(\ell_i^j) = \text{True}\}$. Procedendo per induzione su i , si verifica che l'assegnamento v_i soddisfa l'insieme di clausole $\bigcup_{1 \leq j \leq i} \tilde{C}_j$, per ogni $i = 1, \dots, n$. Pertanto v_n soddisfa \tilde{S} . Quindi, se S è soddisfacibile allora anche \tilde{S} è soddisfacibile.

Viceversa, si verifica che ogni assegnamento che soddisfa \tilde{S} soddisfa anche S (si veda la nota a piè pagina N. 5). \square

11.2 Problema n-SAT ($n > 0$)

Definizione 33. Una n -CLAUSOLA, dove $n > 0$, è una clausola contenente n letterali. \blacksquare

Definizione 34. Per ogni $n > 0$, il PROBLEMA n -SAT consiste nel determinare se un dato insieme finito di n -clausole è soddisfacibile. \blacksquare

Teorema 26. Per ogni $n > 2$, il problema n -SAT è NP-hard.

⁵Si osservi che se $|C_i| \leq 3$ allora v soddisfa C_i se e solo se v soddisfa l'insieme di clausole \tilde{C}_i , per ogni assegnamento v . Se $|C_i| > 3$ allora ogni assegnamento che soddisfa \tilde{C}_i , soddisfa anche C_i .

Dimostrazione. Procediamo per induzione su n . Per $n = 3$ abbiamo già dimostrato che 3-SAT è NP-hard. Sia $n \geq 3$ e supponiamo, per ipotesi induttiva, che n -SAT è NP-hard. Dimostriamo che n -SAT \leq_p $(n + 1)$ -SAT. Sia S un insieme finito di n -clausole. Dobbiamo costruire un insieme \tilde{S} di $(n + 1)$ -clausole tale che S è soddisfacibile se e solo se \tilde{S} è soddisfacibile. Basta usare un nuovo atomo a non appartenente al supporto di S e porre $\tilde{S} = \{C \cup \{a\}, C \cup \{\neg a\} : C \in S\}$. \square

11.3 Problemi VertexCover, CLIQUE, IndependentSet

Definizione 35. Sia $G = (V, E)$ un grafo nondirezionato.

- (1) Un VERTEX COVER per G è un sottoinsieme W di V tale che $\{x, y\} \cap W \neq \emptyset$ per ogni arco $\{x, y\} \in E$.
- (2) Un CLIQUE per G è un sottoinsieme W di V tale che $\{x, y\} \in E$ per ogni $x, y \in W$ con $x \neq y$.
- (3) Un INDEPENDENT SET per G è un sottoinsieme W di V tale che $\{x, y\} \notin E$ per ogni $x, y \in W$. \blacksquare

Definizione 36. Il PROBLEMA VertexCover consiste nel determinare per ogni coppia (G, k) , dove $G = (V, E)$ è un grafo e $0 < k \leq |V|$, se esiste un vertex cover W per G tale che $|W| \leq k$. \blacksquare

Definizione 37. Il PROBLEMA CLIQUE consiste nel determinare per ogni coppia (G, k) , dove $G = (V, E)$ è un grafo e $0 < k \leq |V|$, se esiste un clique W per G tale che $|W| \geq k$. \blacksquare

Definizione 38. Il PROBLEMA IndependentSet consiste nel determinare per ogni coppia (G, k) , dove $G = (V, E)$ è un grafo e $0 < k \leq |V|$, se esiste un independent set W per G tale che $|W| \geq k$. \blacksquare

Teorema 27. Il problema VertexCover è NP-completo.

Dimostrazione. Dimostriamo che 3-SAT \leq_p VertexCover cosicché VertexCover è NP-hard.

Sia $S = \{C_1, C_2, \dots, C_n\}$ un insieme di 3-clausole, dove $C_i = \{\ell_i^1, \ell_i^2, \ell_i^3\}$ per $i = 1, \dots, n$, e sia U il supporto di S . Costruiamo il grafo $G_S = (V_S, E_S)$ e l'intero k_S , con $0 < k_S \leq |V_S|$, tali che S è soddisfacibile se e solo se esiste un vertex cover per G_S di cardinalità al più k_S . La costruzione di G_S coinvolge l'introduzione degli insiemi di vertici V_1, \dots, V_n e dei grafi $G_1 = (V_1 \cup C_1, E_1), \dots, G_n = (V_n \cup C_n, E_n)$ (il grafo G_i corrispondente alla clausola C_i) soddisfacenti alle seguenti condizioni:

- (1) gli insiemi di vertici V_1, \dots, V_n sono a due a due disgiunti;
- (2) l'unione di V_1, \dots, V_n è disgiunta da $U \cup \{\neg a : a \in U\}$;
- (3) per ogni $i = 1, \dots, n$:
 - (3.a) $|V_i| = 3$;
 - (3.b) $E_i = \{\{v_i^1, v_i^2\}, \{v_i^1, v_i^3\}, \{v_i^2, v_i^3\}, \{v_i^1, \ell_i^1\}, \{v_i^2, \ell_i^2\}, \{v_i^3, \ell_i^3\}\}$,
dove $V_i = \{v_i^1, v_i^2, v_i^3\}$.

Si definiscono quindi

$$V_S = U \cup \{\neg a : a \in U\} \cup \left(\bigcup_{i \in \{1, \dots, n\}} V_i \right)$$

$$E_S = \{\{a, \neg a\} : a \in U\} \cup \left(\bigcup_{i \in \{1, \dots, n\}} E_i \right)$$

e

$$k_S = |U| + 2|S|.$$

Mostriamo che S è soddisfacibile se e solo se esiste un vertex cover per G_S di cardinalità al più k_S .

Sia v un assegnamento che soddisfa S . Poniamo

$$W = \{a : a \in U \text{ AND } v(a) = \text{True}\} \cup \{\neg a : a \in U \text{ AND } v(a) = \text{False}\} \cup \left(\bigcup_{i \in \{1, \dots, n\}} H_i \right),$$

dove

$$H_i = \begin{cases} \{v_i^2, v_i^3\}, & \text{se } (\ell_i^1)^v = \text{True} \\ \{v_i^1, v_i^3\}, & \text{se } (\ell_i^1)^v = \text{False AND } (\ell_i^2)^v = \text{True} \\ \{v_i^1, v_i^2\}, & \text{altrimenti,} \end{cases}$$

per ogni $i = 1, \dots, n$. Allora W è un vertex cover per G_S tale che $|W| = k_S$. Infatti, si osservi innanzitutto che, dalla definizione di W , segue che $|W| = k_S$ e $|W \cap \{a, \neg a\}| = 1$, per ogni atomo $a \in U$; quindi è sufficiente dimostrare che $W \cap e \neq \emptyset$, per ogni arco $e \in \left(\bigcup_{i \in \{1, \dots, n\}} E_i \right)$. Procediamo come segue. Sia $j \in \{1, \dots, n\}$. Poiché v soddisfa S , si ha che v soddisfa la clausola C_j e quindi esiste il minimo indice $h \in \{1, 2, 3\}$ tale che $(\ell_j^h)^v = \text{True}$. Dalla definizione di H_j segue che

$$H_j \cap e \neq \emptyset, \text{ per ogni } e \in E_j \setminus \{\{v_j^h, \ell_j^h\}\}$$

e quindi

$$(W \cup \{\ell_j^h\}) \cap e \neq \emptyset, \text{ per ogni } e \in E_j.$$

Poiché $(\ell_j^h)^v = \text{True}$ si ha (usando la definizione di W) che $\ell_j^h \in W$ e pertanto

$$W \cap e \neq \emptyset, \text{ per ogni } e \in E_j.$$

Data l'arbitrarietà di j concludiamo che, difatti, $W \cap e \neq \emptyset$, per ogni $e \in \left(\bigcup_{i \in \{1, \dots, n\}} E_i \right)$.

Supponiamo adesso che W è un vertex cover per G_S tale che $|W| \leq k_S$ e mostriamo che esiste un assegnamento che soddisfa S . Sia $i \in \{1, \dots, n\}$. Poiché W contiene almeno un estremo di ogni arco di E_i , esistono almeno due vertici distinti $x, y \in V_i$ tali che $x, y \in W$. Pertanto l'insieme $W \cap \left(\bigcup_{i \in \{1, \dots, n\}} V_i \right)$ contiene almeno $2n = 2|S|$ vertici distinti. D'altra parte, poiché $W \cap \{a, \neg a\} \neq \emptyset$ vale per ogni $a \in U$, si ha anche che $|W \cap \{a, \neg a : a \in U\}| \geq |U|$. Dato che gli insiemi $\bigcup_{i \in \{1, \dots, n\}} V_i$ e $\{a, \neg a : a \in U\}$ sono disgiunti otteniamo quindi che $|W| \geq k_S$. Ma allora $|W| = k_S$ e inoltre:

- (i) uno solo tra i letterali a e $\neg a$ è un vertice di W , per ogni atomo $a \in U$;

(ii) W contiene esattamente due vertici di V_i , per ogni $i = 1, 2, \dots, n$.

Sia dunque v l'assegnamento tale che

$$v(a) = \begin{cases} \text{True}, & \text{se } a \in W \\ \text{False}, & \text{altrimenti,} \end{cases}$$

per ogni atomo a . Verifichiamo che v soddisfa ogni clausola di S . Sia $i \in \{1, \dots, n\}$. Poiché $|W \cap V_i| = 2$ (cf. condizione (ii)) e W contiene almeno un estremo di ogni arco di E_i , si ha che $W \cap C_i \neq \emptyset$, cioè W contiene almeno un letterale ℓ_i^h della clausola C_i . Se ℓ_i^h è un atomo, allora si ha che $(\ell_i^h)^v = \text{True}$ (perché $\ell_i^h \in W$) e quindi v soddisfa C_i . Altrimenti, sia $\ell_i^h = \neg a$ dove a è un atomo. Poiché $\neg a \in W$ si ha che $a \notin W$ ((cf. condizione (i))) e quindi $v(a) = \text{False}$; questo implica che $(\ell_i^h)^v = (\neg a)^v = \text{True}$ e così v soddisfa C_i . Data l'arbitrarietà di i concludiamo che v soddisfa ogni clausola di S . \square

Teorema 28. *Sia $G = (V, E)$ un grafo nondirezionato e sia $W \subseteq V$. Le seguenti condizioni sono equivalenti:*

- (1) W è un vertex cover per G ;
- (2) $V \setminus W$ è un independent set per G ;
- (3) $V \setminus W$ è un clique per il GRAFO COMPLEMENTARE $G^c = (V, E^c)$ di G dove

$$E^c = \{\{x, y\} : x, y \in V, x \neq y, \{x, y\} \notin E\}.$$

Teorema 29. *I problemi CLIQUE e IndependentSet sono NP-completi.*

Dimostrazione. Per dimostrare che i problemi CLIQUE e IndependentSet sono NP-hard si dimostri che $\text{VertexCover} \leq_p \text{IndependentSet}$ e $\text{VertexCover} \leq_p \text{CLIQUE}$ usando il Teorema 28. (Si osservi che per ogni grafo $G = (V, E)$, ogni $W \subseteq V$ e ogni $0 < k \leq |V|$, si ha che $|W| \leq k$ se e solo se $|V \setminus W| \geq |V| - k$.) \square

Osservazione 4. Se nella Definizione 36 del problema VertexCover si sostituisce il vincolo che “ $|W| \leq k$ ” con “ $|W| = k$ ” si ottiene un problema equivalente. Infatti, esiste un vertex cover per G di cardinalità minore o uguale a k se e solo se esiste un vertex cover per G avente cardinalità uguale a k . (Se W è un vertex cover per G tale che $|W| < k$, basta aggiungere $k - |W|$ vertici arbitrari a W in modo da ottenere un vertex cover di cardinalità k .) Si osservi inoltre che:

- (a) se $k > 1$ allora esiste un un vertex cover per G di cardinalità minore di k se e solo se esiste un un vertex cover per G di cardinalità minore o uguale a $k - 1$;
- (b) esiste sempre un vertex cover W per G di cardinalità maggiore o uguale a k (basta porre $W = V$);
- (c) se $k < |V|$ allora esiste un vertex cover W per G di cardinalità maggiore di k (basta porre $W = V$).

Considerazioni analoghe valgono per i problemi CLIQUE e IndependentSet. \blacksquare

Teorema 30. $\text{SAT} \leq_p \text{CLIQUE}$.

Dimostrazione. Abbiamo dimostrato precedentemente che $\text{SAT} \leq_p 3\text{-SAT}$ e che $3\text{-SAT} \leq_p \text{VertexCover}$ e inoltre abbiamo osservato che $\text{VertexCover} \leq_p \text{CLIQUE}$; dalla transitività della relazione “ \leq_p ” segue quindi che $\text{SAT} \leq_p \text{CLIQUE}$. Dimostriamo tuttavia in maniera diretta che $\text{SAT} \leq_p \text{CLIQUE}$. Premettiamo la seguente definizione. Due LETTERALI ℓ' ED ℓ'' SONO COMPLEMENTARI, e si scrive $\ell' \bowtie \ell''$, se esiste un atomo a tale che “ $\ell' = a \text{ AND } \ell'' = \neg a$ ” oppure “ $\ell' = \neg a \text{ AND } \ell'' = a$ ”. Scriviamo $\ell' \not\bowtie \ell''$ per indicare che ℓ' ed ℓ'' non sono complementari. Si osservi che se $\ell_1, \ell_2, \dots, \ell_n$ sono letterali “mutuamente non complementari”, cioè $\ell_i \not\bowtie \ell_j$ per $0 \leq i < j \leq n$, allora esiste un assegnamento v tale che $(\ell_1)^v = (\ell_2)^v = \dots = (\ell_n)^v = \text{True}$. (Ad esempio,

$$v(a) = \begin{cases} \text{True}, & \text{se esiste } i \in \{1, 2, \dots, n\} \text{ tale che } a = \ell_i \\ \text{False}, & \text{altrimenti,} \end{cases}$$

per ogni atomo a .)

Dato un insieme $S = \{C_1, C_2, \dots, C_n\}$ di clausole sia $G_S = (V_S, E_S)$ il grafo nondirezionato dove

$$V_S = \bigcup_{1 \leq i \leq n} \{(\ell, i) : \ell \in C_i\},$$

e

$$E_S = \{(\ell', i), (\ell'', j) : i, j \in \{1, \dots, n\}, i \neq j, \ell' \in C_i, \ell'' \in C_j, \ell' \not\bowtie \ell''\}.$$

L'insieme S è soddisfacibile se e solo se esiste un clique W per G_S tale che $|W| \geq n$. Infatti sia v un assegnamento che soddisfa S . Per ogni $i = 1, 2, \dots, n$ esiste un letterale $\ell_i \in C_i$ tale che $(\ell_i)^v = \text{True}$. Sia $W = \{(\ell_i, i) : 1 \leq i \leq n\}$. Ovviamente $W \subseteq V_S$ e inoltre $|W| = n$. Mostriamo che W è un clique per G_S . Siano $(\ell_i, i), (\ell_j, j) \in W$ dove $i \neq j$. Poichè $(\ell_i)^v = (\ell_j)^v = \text{True}$ si ha che $\ell_i \not\bowtie \ell_j$ e quindi $\{(\ell_i, i), (\ell_j, j)\} \in E_S$. Così $\{x, y\} \in E_S$ per ogni $x, y \in W$ con $x \neq y$. Dunque W è un clique per G_S .

Viceversa, supponiamo che esiste un clique $W = \{w_1, w_2, \dots, w_m\}$ per G_S dove $m \geq n$. Per $k = 1, 2, \dots, m$, sia $w_k = (\ell_{i_k}, i_k)$. Per dimostrare che S è soddisfacibile è sufficiente dimostrare che valgono le seguente proprietà:

- $i_j \neq i_h$ per ogni $1 \leq j < h \leq m$ (così $\{i_1, i_2, \dots, i_n\} = \{1, 2, \dots, n\}$ perchè $\{i_1, i_2, \dots, i_n\} \subseteq \{1, 2, \dots, n\}$);
- $\ell_{i_k} \in C_{i_k}$ per ogni $1 \leq k \leq n$;
- i letterali $\ell_{i_1}, \ell_{i_2}, \dots, \ell_{i_n}$ sono mutuamente non complementari.

La (b) è ovvia; (a) e (c) seguono dal fatto che se $1 \leq j < h \leq m$ allora $\{(\ell_{i_j}, i_j), (\ell_{i_h}, i_h)\} \in E_S$ (poichè W è un clique per G_S). \square

11.4 Problemi HamiltonianPath e HamiltonianCircuit

Definizione 39. Sia $G = (V, E)$ una grafo nondirezionato.

- Un CAMMINO in G è una sequenza (v_0, v_1, \dots, v_n) di vertici di G , dove $n > 0$, tale che $\{v_i, v_{i+1}\} \in E$, per $i = 0, 1, \dots, n-1$.
- La LUNGHEZZA DEL CAMMINO (v_0, v_1, \dots, v_n) è n .
- Un CAMMINO SEMPLICE in G è un cammino (v_0, v_1, \dots, v_n) in G tale che $v_i \neq v_j$ per ogni $i, j = 0, 1, \dots, n$ con $i \neq j$.

- (4) Un CIRCUITO (O CICLO) SEMPLICE in G è un cammino semplice (v_0, v_1, \dots, v_n) dove $n > 1$ e $\{v_0, v_n\} \in E$.
- (5) Un CAMMINO HAMILTONIANO in G è un cammino semplice (v_0, v_1, \dots, v_n) dove $V = \{v_0, v_1, \dots, v_n\}$ (cioè un cammino semplice che include tutti i vertici di G).
- (6) Un CIRCUITO HAMILTONIANO in G è un circuito semplice (v_0, v_1, \dots, v_n) dove $V = \{v_0, v_1, \dots, v_n\}$ (cioè un circuito semplice che include tutti i vertici di G). ■

Definizione 40. Il PROBLEMA HamiltonianPath consiste nel determinare se una dato grafo G ha un cammino Hamiltoniano. ■

Definizione 41. Il PROBLEMA HamiltonianCircuit consiste nel determinare se una dato grafo G ha un circuito Hamiltoniano. ■

Teorema 31. $\text{VertexCover} \leq_p \text{HamiltonianCircuit}$.

Dimostrazione. Sia $G = (V, E)$ una grafo nondirezionato e sia $0 < k \leq |V|$. Costruiamo il grafo $\widehat{G} = (\widehat{V}, \widehat{E})$ tale che G ha un vertex cover di cardinalità al più k se e solo se \widehat{G} ha un circuito Hamiltoniano.

Per ogni vertice $v \in V$, sia $\text{Inc}(v)$ l'insieme degli archi di G che incidono su v , cioè

$$\text{Inc}(v) = \{e \in E : v \in e\},$$

e siano $\text{deg}(v) = |\text{Inc}(v)|$ e $e_v^1, e_v^2, \dots, e_v^{\text{deg}(v)}$ un ordinamento di $\text{Inc}(v)$.

La costruzione di \widehat{G} coinvolge l'introduzione dei seguenti insiemi disgiunti di vertici:

$$A = \{a_1, a_2, \dots, a_k\}$$

e

$$V' = \bigcup_{e \in E} V'_e$$

dove

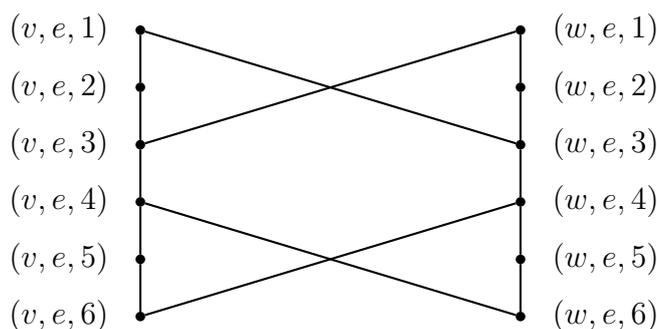
$$V'_e = \{(v, e, i), (w, e, i) : 1 \leq i \leq 6\} \quad \text{per ogni arco } e = \{v, w\} \in E$$

e dei seguenti insiemi di archi:

$$E' = \bigcup_{e \in E} E'_e, \quad E'' = \bigcup_{v \in V} E''_v, \quad E''' ,$$

dove

$$E'_e = \{ \{(v, e, i), (v, e, i+1)\}, \{(w, e, i), (w, e, i+1)\} : 1 \leq i \leq 5\} \cup \\ \{ \{(v, e, 3), (w, e, 1)\}, \{(w, e, 3), (v, e, 1)\}\} \cup \\ \{ \{(v, e, 6), (w, e, 4)\}, \{(w, e, 6), (v, e, 4)\}\} \quad \text{per ogni arco } e = \{v, w\} \in E,$$



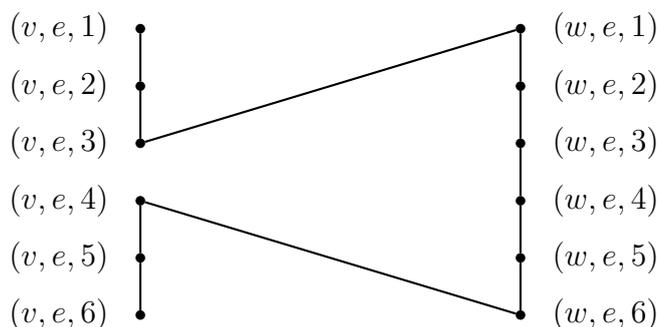
$$E'' = \{(v, e_v^i, 6), (v, e_v^{i+1}, 1)\} : 1 \leq i < \deg(v)\} \quad \text{per ogni vertice } v \in V,$$

$$E''' = \{a_i, (v, e_v^1, 1)\}, \{a_i, (v, e_v^{\deg(v)}, 6)\} : 1 \leq i \leq k, v \in V\}.$$

Si osservi che per ogni arco $e = \{v, w\} \in E$, se π è un cammino semplice nel grafo (V_e', E_e') dal vertice $(v, e, 1)$ al vertice $(v, e, 6)$ allora o

$$\pi = ((v, e, 1), (v, e, 2), (v, e, 3), (v, e, 4), (v, e, 5), (v, e, 6))$$

oppure π è il seguente cammino:



Poniamo

$$\widehat{V} = A \cup V' \quad \text{e} \quad \widehat{E} = E' \cup E'' \cup E'''.$$

Supponiamo che esiste un circuito Hamiltoniano $\pi = (z_0, z_1, \dots, z_n)$ in \widehat{G} . Siano:

- (i) $m = \min(\{i \in \{0, \dots, n\} : z_i \in A\})$;
- (ii) $\pi^* = (x_0, x_1, \dots, x_n, x_{n+1}) = (z_m, z_{m+1}, \dots, z_n, z_0, z_1, \dots, z_m)$;
- (iii) $A_i = \{v \in V : \{a_i, (v, e_v^1, 1)\} \text{ è un arco di } \pi^* \}$, per $i = 1, 2, \dots, k$.

Allora l'insieme

$$W = \bigcup_{1 \leq i \leq k} A_i$$

è un vertex cover per G tale che $|W| \leq k$.

Infatti, si osservi innanzitutto che per ogni arco $e \in E$ esiste un indice $i \in \{1, 2, \dots, k\}$ tale che $A_i \cap e \neq \emptyset$. Per cui W è un vertex cover per G . Il fatto che $|W| \leq k$ è conseguenza

della seguente osservazione.

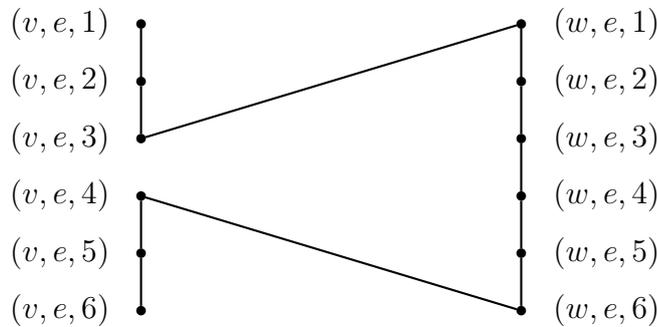
Se $i, j \in \{0, \dots, n+1\}$ sono indici tali che (a) $i < j$, (b) $x_i, x_j \in A$ e (c) $x_h \notin A$ per ogni $i < h < j$, allora esiste un unico vertice $v \in V$ tale che

$$(x_{i+1} = (v, e_v^1, 1) \text{ AND } x_{j-1} = (v, e_v^{\deg(v)}, 6)) \text{ OR } (x_{i+1} = (v, e_v^{\deg(v)}, 6) \text{ AND } x_{j-1} = (v, e_v^1, 1)).$$

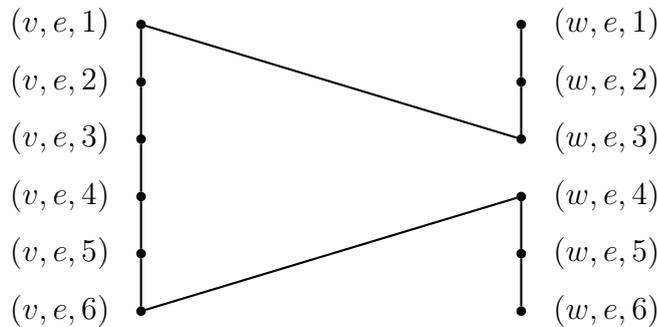
Pertanto W non può contenere più di k vertici perchè, altrimenti, esisterebbero indici $i, j \in \{0, \dots, n+1\}$ soddisfacenti alle condizioni (a), (b) e (c) precedenti e tali che $x_{i+1} = (v, e_v^1, 1)$ e $x_{j-1} = (w, e_w^1, 1)$ per alcuni vertici $v, w \in W$, contraddicendo l'osservazione appena fatta.

Viceversa, sia W un vertex cover per G tale che $|W| = k$ e siano v_1, v_2, \dots, v_k i vertici di W . Per ogni arco $e = \{v, w\} \in E$, sia $\text{Edges}(e)$ l'insieme di archi definito come segue:

(1) se $W \cap e = \{v\}$ allora $\text{Edges}(e)$ consiste nei seguenti archi:



(2) se $W \cap e = \{w\}$ allora $\text{Edges}(e)$ consiste nei seguenti archi:



(3) se $W \cap e = \{v, w\}$ allora $\text{Edges}(e)$ consiste nei seguenti archi:



(Si osservi che si verifica esattamente una sola delle precedenti possibilità in quanto W è un vertex cover per G .) Si verifica che un cammino Hamiltoniano in \widehat{G} può essere costruito usando gli archi del seguente insieme

$$\begin{aligned} & \{\text{Edges}(e) : e \in E\} \cup \\ & \left(\bigcup_{1 \leq i \leq k} E''_{v_i} \right) \cup \\ & \{ \{a_i, (v_i, e_{v_i}^1, 1)\} : 1 \leq i \leq k \} \cup \\ & \{ \{a_{i+1}, (v_i, e_{v_i}^{\deg(v_i)}, 6)\} : 1 \leq i < k \} \cup \\ & \{ \{a_k, (v_k, e_{v_k}^{\deg(v_k)}, 6)\} \}. \end{aligned}$$

□

Teorema 32. $\text{HamiltonianCircuit} \leq_p \text{HamiltonianPath}$.

Dimostrazione. Sia $G = (V, E)$ un grafo. Costruiamo il grafo G' tale che G ha un circuito Hamiltoniano se e solo se G' ha un cammino Hamiltoniano.

Si osservi innanzitutto che se G contiene meno di tre vertici allora G non può avere alcun circuito Hamiltoniano. In questo caso è sufficiente prendere come G' un qualunque grafo che non ha cammini Hamiltoniani (ad esempio un grafo contenente un solo vertice). Supponiamo quindi che G contiene almeno tre vertici.

Sia u un arbitrario vertice di G e siano u^* , v e v^* tre nuovi vertici distinti non appartenenti a V . Poniamo $G' = (V', E')$ dove

$$V' = V \cup \{u^*, v, v^*\}$$

ed

$$E' = E \cup \{ \{u^*, x\} : \{u, x\} \in E \} \cup \{ \{v, u\}, \{v^*, u^*\} \}.$$

Supponiamo che (v_0, v_1, \dots, v_n) è un circuito Hamiltoniano in G . Sia i tale che $v_i = u$. Allora la sequenza $(v, u, v_{i+1}, \dots, v_n, v_1, \dots, v_{i-1}, u^*, v^*)$ è un cammino Hamiltoniano in G' . Viceversa, sia (v_0, v_1, \dots, v_n) un cammino Hamiltoniano in G' . Data la struttura di G' , deve aversi necessariamente che $v_0 = v$ e $v_n = v^*$ oppure $v_0 = v^*$ e $v_n = v$ (cioè $v_i \notin \{v, v^*\}$ per ogni $0 < i < n$). Il primo caso, cioè $v_0 = v$ e $v_n = v^*$, implica che $v_1 = u$, $v_{n-1} = u^*$ e $\{v_{n-2}, u\} \in E$, così la sequenza (v_1, \dots, v_{n-2}) è un circuito Hamiltoniano in G . (Si osservi che $n \geq 5$ dato che G contiene almeno tre vertici). Il caso $v_0 = v^*$ e $v_n = v$ si tratta in maniera simmetrica. □

Teorema 33. $\text{HamiltonianPath} \leq_p \text{HamiltonianCircuit}$.

Dimostrazione. Poiché $\text{HamiltonianCircuit}$ è **NP**-completo e HamiltonianPath è in **NP**, si ha chiaramente che $\text{HamiltonianPath} \leq_p \text{HamiltonianCircuit}$. Mostriamo tuttavia un riduzione polinomiale diretta da HamiltonianPath a $\text{HamiltonianCircuit}$.

Sia $G = (V, E)$ un grafo nondirezionato. Costruiamo il grafo $G' = (V', E')$ tale che G ha un cammino Hamiltoniano se e solo se G' ha un circuito Hamiltoniano. La costruzione coinvolge l'introduzione del un nuovo vertice z non appartenente a V e degli archi $\{\{z, v\} : v \in V\}$; quindi $V' = V \cup \{z\}$ ed $E' = E \cup \{\{z, v\} : v \in V\}$.

Se (v_0, v_1, \dots, v_n) un cammino Hamiltoniano in G , si verifica immediatamente che la sequenza di vertici $(v_0, v_1, \dots, v_n, z)$ è un circuito Hamiltoniano in G' .

Viceversa, sia (w_0, w_1, \dots, w_n) un circuito Hamiltoniano in G' e sia $i \in \{0, 1, \dots, n\}$ tale che $w_i = z$. Se $i = 0$ allora la sequenza di vertici (w_1, \dots, w_n) è un cammino Hamiltoniano in G . Similmente, se $i = n$ allora la sequenza di vertici $(w_0, w_1, \dots, w_{n-1})$ è un cammino Hamiltoniano in G . Infine, se $0 < i < n$ allora la sequenza di vertici $(w_{i+1}, \dots, w_n, w_0, \dots, w_{i-1})$ è un cammino Hamiltoniano in G . □

Osservazione 5. I problemi sui grafi nondirezionati `VertexCover`, `CLIQUE`, `IndependentSet`, `HamiltonianCircuit` e `HamiltonianPath` introdotti precedentemente possono essere riformulati anche nel caso di grafi direzionati. Ciascuno di questi problemi è polinomialmente riducibile alla corrispondente “versione direzionata”. Ad esempio, consideriamo il problema `VertexCover`. Sia $G = (V, E)$ un grafo nondirezionato. Se $\hat{G} = (V, \hat{E})$ è il grafo direzionato dove $\hat{E} = \{(v, w) : \{v, w\} \in E\}$, si verifica che W è un vertex cover per G se e solo se W è un vertex cover per \hat{G} . ■

11.5 Problema LongestSimplePath

Definizione 42. Il `PROBLEMA LongestSimplePath` consiste nel determinare per ogni coppia (G, k) , dove G è un grafo e $k \in \mathbb{N} \setminus \{0\}$, se G ha un cammino semplice di lunghezza maggiore o uguale a k . ■

Teorema 34. `HamiltonianPath` \leq_p `LongestSimplePath`.

Dimostrazione. Sia $G = (V, E)$ un grafo. Poniamo $k = |V| - 1$. Allora esiste un cammino Hamiltoniano in G se e solo se esiste un cammino semplice in G di lunghezza maggiore o uguale a k . □

11.6 Problemi SetCover e HittingSet

Definizione 43. Sia $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$ una famiglia di insiemi finiti.

Un `RICOPRIMENTO` DI \mathcal{F} è una sottofamiglia $\mathcal{R} = \{S_{i_1}, S_{i_2}, \dots, S_{i_m}\}$ di \mathcal{F} tale che

$$\bigcup_{1 \leq j \leq n} S_j = \bigcup_{1 \leq h \leq m} S_{i_h}.$$

Un `HITTING SET` di \mathcal{F} è un sottoinsieme I di $\bigcup_{1 \leq j \leq n} S_j$ tale che

$$I \cap S_j \neq \emptyset,$$

per ogni $j = 1, 2, \dots, n$. ■

Definizione 44. Il `PROBLEMA SetCover` consiste nel determinare per ogni coppia (\mathcal{F}, k) , dove \mathcal{F} è una famiglia (finita) di insiemi finiti e $k \in \mathbb{N} \setminus \{0\}$, se esiste un ricoprimento \mathcal{R} di \mathcal{F} contenente al più k insiemi. ■

Definizione 45. Il `PROBLEMA HittingSet` consiste nel determinare per ogni coppia (\mathcal{F}, k) , dove \mathcal{F} è una famiglia (finita) di insiemi finiti e $k \in \mathbb{N} \setminus \{0\}$, se esiste un hitting set I di \mathcal{F} di cardinalità al più k . ■

Teorema 35. `VertexCover` \leq_p `SetCover`.

Dimostrazione. Sia $G = (V, E)$ un grafo nondirezionato dove $V = \{v_1, v_2, \dots, v_n\}$. Per ogni $i = 1, 2, \dots, n$, sia

$$S_i = \{e \in E : v_i \in e\},$$

e sia $\mathcal{F} = \{S_1, S_2, \dots, S_n\}$.

Per ogni sottoinsieme $\{i_1, i_2, \dots, i_m\}$ di $\{1, 2, \dots, n\}$, l'insieme $W = \{v_{i_1}, v_{i_2}, \dots, v_{i_m}\}$ è un vertex cover per G se e solo se $\mathcal{R} = \{S_{i_1}, S_{i_2}, \dots, S_{i_m}\}$ è un ricoprimento di \mathcal{F} .

Infatti, supponiamo che $W = \{v_{i_1}, v_{i_2}, \dots, v_{i_m}\}$ è un vertex cover per G . Sia $e \in \bigcup_{1 \leq j \leq n} S_j$. Allora e è un arco del grafo G e quindi $W \cap e \neq \emptyset$, cioè esiste $h \in \{1, 2, \dots, m\}$ tale che $v_{i_h} \in e$.

Questo implica che $e \in S_{i_h}$ e pertanto $e \in \bigcup_{1 \leq h \leq m} S_{i_h}$. Data l'arbitrarietà di e concludiamo che \mathcal{R} è un ricoprimento di \mathcal{F} .

Viceversa, supponiamo che $\mathcal{R} = \{S_{i_1}, S_{i_2}, \dots, S_{i_m}\}$ è un ricoprimento di \mathcal{F} . Sia $e = \{v_j, v_k\} \in E$. Allora $e \in S_j$ e quindi esiste $h \in \{1, 2, \dots, m\}$ tale che $e \in S_{i_h}$ (perchè \mathcal{R} è un ricoprimento di \mathcal{F}). Da ciò segue che $v_{i_h} \in e$ e quindi $v_{i_h} \in e \cap W$ che implica che $W \cap e \neq \emptyset$. Dunque W interseca ogni arco del grafo G e pertanto è un vertex cover per G . \square

Teorema 36. $\text{VertexCover} \leq_p \text{HittingSet}$.

Dimostrazione. Sia $G = (V, E)$ un grafo nondirezionato dove $E = \{e_1, e_2, \dots, e_n\}$.

Si osservi che se W è un vertex cover per G allora l'insieme $I = W \cap (\bigcup_{1 \leq i \leq n} e_i)$ è un hitting set per la famiglia $\mathcal{F} = \{e_1, e_2, \dots, e_n\}$. Inoltre $|I| \leq |W|$.

Viceversa, se I è un hitting set per la famiglia $\mathcal{F} = \{e_1, e_2, \dots, e_n\}$, allora I è un vertex cover per G . \square

11.7 Problema SubgraphIsomorphism

Definizione 46. Un SOTTOGRAFO di un grafo $G = (V, E)$ è un grafo $G' = (V', E')$ tale che $V' \subseteq V$ ed $E' \subseteq E$.

Un GRAFO $G_1 = (V_1, E_1)$ è ISOMORFO ad un grafo $G_2 = (V_2, E_2)$, e si scrive $G_1 \approx G_2$, se esiste una funzione biettiva $f : V_1 \rightarrow V_2$, chiamata ISOMORFISMO DA G_1 IN G_2 , tale che $\{x, y\} \in E_1 \Leftrightarrow \{f(x), f(y)\} \in E_2$ per ogni $x, y \in V_1$.

Definizione 47. Il PROBLEMA SubgraphIsomorphism consiste nel determinare per ogni coppia (G_1, G_2) , dove G_1 e G_2 sono grafi, se e esiste un sottografo H di G_1 tale che $H \approx G_2$. \blacksquare

Teorema 37. $\text{CLIQUE} \leq_p \text{SubgraphIsomorphism}$.

Dimostrazione. Sia $G = (V, E)$ un grafo dove $V = \{v_1, \dots, v_n\}$ e sia $0 < k \leq |V|$. Indichiamo con \mathcal{G}_k il GRAFO COMPLETO DI DIMENSIONE k CON VERTICI v_1, \dots, v_k , cioè $\mathcal{G}_k = (\{v_1, \dots, v_k\}, \{\{v_i, v_j\} : i, j \in \{1, \dots, k\}, i \neq j\})$. Allora esiste un clique W per G tale che $|W| \geq k$ se e solo se esiste un sottografo H di G tale che $H \approx \mathcal{G}_k$. Infatti, sia $W = \{v_{i_1}, \dots, v_{i_h}\}$ un clique per G tale che $h \geq k$. Poniamo

$$H = (\{v_{i_1}, \dots, v_{i_k}\}, \{\{v_{i_a}, v_{i_b}\} : a, b \in \{1, \dots, k\}, a \neq b\}).$$

Ovviamente $H \approx \mathcal{G}_k$. Inoltre H è un sottografo di G dato che W è un clique per G .

Viceversa, sia $H = (V', E')$ un sottografo di G tale che $H \approx \mathcal{G}_k$ e sia f un'isomorfismo da H in \mathcal{G}_k . Ovviamente $|V'| = k$ (perchè f è un biiezione tra V' e l'insieme dei k vertici di \mathcal{G}_k). Verifichiamo che V' è un clique per G . Siano $x, y \in V'$ tali che $x \neq y$. Allora $f(x)$ e $f(y)$ sono vertici distinti di \mathcal{G}_k e quindi $\{f(x), f(y)\}$ è un'arco di \mathcal{G}_k . Questo implica che $\{x, y\}$ è un arco di H , cioè $\{x, y\} \in E'$. Poichè $E' \subseteq E$ (in quanto H è un sottografo di G) si ha che $\{x, y\} \in E$. Pertanto V' è un clique per G . \square

11.8 Problema TravelingSalesman

Definizione 48. Un TSG è una coppia ordinata $\mathcal{T} = (G, d)$ dove

- (1) $G = (V, E)$ è un GRAFO NONDIREZIONATO COMPLETO, cioè $\{v, w\} \in E$ per ogni $v, w \in V$ con $v \neq w$;
- (2) $d : V \times V \rightarrow \mathbb{N} \setminus \{0\}$ è la "funzione costo" tale che $d(v, w) = d(w, v)$, per ogni $v, w \in V$. \blacksquare

Definizione 49. Sia $\mathcal{T} = (G, d)$ un TSG.

Un TOUR in \mathcal{T} è un cammino Hamiltoniano π in G . Il COSTO DEL TOUR $\pi = (v_0, v_1, \dots, v_n)$ è la quantità $\tilde{d}(\pi)$ definita come segue:

$$\tilde{d}(\pi) = d(v_n, v_0) + \sum_{i=0}^{n-1} d(v_i, v_{i+1}).$$

■

Definizione 50. Il PROBLEMA TravelingSalesman consiste nel determinare per ogni coppia (\mathcal{T}, k) , dove \mathcal{T} è un TSG e $k \in \mathbb{N} \setminus \{0\}$, se esiste un tour π in \mathcal{T} tale che $\tilde{d}(\pi) \leq k$. ■

Teorema 38. HamiltonianCircuit \leq_p TravelingSalesman.

Dimostrazione. Sia $G = (V, E)$ un grafo nondirezionato dove $|V| = k$. Costruiamo il TSG \mathcal{T}_G tale che G ha un circuito Hamiltoniano se e solo se \mathcal{T}_G ha un tour di costo al più k .

Sia $G' = (V, E')$ il grafo nondirezionato dove $E' = \{\{v, w\} : v, w \in V \text{ AND } v \neq w\}$ e sia $d : V \times V \rightarrow \mathbb{N} \setminus \{0\}$ definita come segue:

$$d(v, w) = \begin{cases} 1, & \text{se } \{v, w\} \in E \\ 2, & \text{altrimenti,} \end{cases}$$

per ogni $v, w \in V$. Poniamo $\mathcal{T}_G = (G', d)$.

Sia $\pi = (v_0, v_1, \dots, v_{k-1})$ un circuito Hamiltoniano in G . Allora π è un tour in \mathcal{T}_G . Inoltre, poichè $\{v_0, v_1\}, \dots, \{v_{k-2}, v_{k-1}\}, \{v_{k-1}, v_0\} \in E$, si ha che $d(v_0, v_1) = \dots = d(v_{k-2}, v_{k-1}) = d(v_{k-1}, v_0) = 1$ e quindi il costo del tour π è uguale a k .

Viceversa, sia $\pi = (v_0, v_1, \dots, v_{k-1})$ un tour in \mathcal{T}_G tale che $\tilde{d}(\pi) \leq k$. Poichè la funzione d assume solo valori dell'insieme $\{1, 2\}$, deve aversi necessariamente che $d(v_0, v_1) = \dots = d(v_{k-2}, v_{k-1}) = d(v_{k-1}, v_0) = 1$ e quindi $\{v_0, v_1\}, \dots, \{v_{k-2}, v_{k-1}\}, \{v_{k-1}, v_0\} \in E$. Pertanto π è un circuito Hamiltoniano in G . □

Osservazione 6. Se nella dimostrazione del teorema precedente si sostituisce la definizione della funzione d con la seguente:

$$d(v, w) = \begin{cases} 2, & \text{se } \{v, w\} \in E \\ 1, & \text{altrimenti,} \end{cases}$$

per ogni $v, w \in V$, allora ogni circuito Hamiltoniano in G è un tour in \mathcal{T}_G di costo maggiore o uguale a $2k$ e, viceversa, se π è un tour in \mathcal{T}_G di costo maggiore o uguale a $2k$ allora π è un circuito Hamiltoniano in G . ■

11.9 Problema 3–DimensionalMatching

Definizione 51. Il PROBLEMA DEL 3–DimensionalMatching è definito come segue.

Dati tre insiemi finiti e disgiunti X, Y e Z tali che $|X| = |Y| = |Z| = q$ e assegnato un sottoinsieme M di $X \times Y \times Z$, esiste un sottoinsieme M' di M tale che $|M'| = q$ e $(x_1, y_1, z_1), (x_2, y_2, z_2) \in M'$ implica che $x_1 \neq x_2, y_1 \neq y_2$ e $z_1 \neq z_2$? In tal caso l'insieme M' costituisce un MATCHING di M . ■

Teorema 39. 3–SAT \leq_p 3–DimensionalMatching.

11.10 Problema Partition

Definizione 52. Il PROBLEMA Partition consiste nel determinare per ogni coppia (A, s) , dove A è un insieme finito e $s : A \rightarrow \mathbb{N} \setminus \{0\}$ è la “funzione dimensione”, se esiste un sottoinsieme B di A tale che

$$\sum_{x \in B} s(x) = \sum_{x \in A \setminus B} s(x).$$

■

Teorema 40. 3–Dimensional Matching \leq_p Partition.

Dimostrazione. Siano $X = \{x_1, \dots, x_q\}$, $Y = \{y_1, \dots, y_q\}$ e $Z = \{z_1, \dots, z_q\}$ insiemi finiti e disgiunti e sia $M = \{m_1, \dots, m_k\} \subseteq X \times Y \times Z$. Introduciamo l'insieme $A = \{a_1, \dots, a_k, b_1, b_2\}$ contenente $k + 2$ elementi. Per $i = 1, \dots, k$, sia

$$s(a_i) = 2^{\rho(3q-r)} + 2^{\rho(2q-s)} + 2^{\rho(q-t)},$$

dove $\rho = \lceil \log_2(k+1) \rceil$ e $(x_r, y_s, z_t) = m_i$ e sia

$$s(b_1) = 2 \left(\sum_{i=1}^k s(a_i) \right) - \omega \quad \text{e} \quad s(b_2) = \left(\sum_{i=1}^k s(a_i) \right) + \omega,$$

dove

$$\omega = \sum_{j=0}^{3q-1} 2^{j\rho}.$$

Si osservi che un sottoinsieme $M' = \{m_{i_1}, m_{i_2}, \dots, m_{i_n}\}$ di M costituisce un matching di M se e solo se

$$\sum_{j=1}^n s(a_{i_j}) = \omega.$$

Sia B un sottoinsieme di A tale che

$$\sum_{x \in B} s(x) = \sum_{x \in A \setminus B} s(x).$$

Allora,

$$\begin{aligned} 2 \sum_{x \in B} s(x) &= \sum_{x \in B} s(x) + \sum_{x \in A \setminus B} s(x) \\ &= \sum_{x \in A} s(x) \\ &= \sum_{i=1}^k s(a_i) + s(b_1) + s(b_2) \\ &= 4 \sum_{i=1}^k s(a_i), \end{aligned}$$

e quindi

$$\sum_{x \in B} s(x) = \sum_{x \in A \setminus B} s(x) = 2 \sum_{i=1}^k s(a_i).$$

Pertanto esiste $C \in \{B, A \setminus B\}$ tale che $b_1 \in C$ e $b_2 \notin C$. Sia $C \setminus \{b_1\} = \{a_{i_1}, a_{i_2}, \dots, a_{i_n}\}$. Allora

$$\sum_{j=1}^n s(a_{i_j}) = \omega$$

e quindi per l'osservazione precedente si ha che $M' = \{m_{i_1}, m_{i_2}, \dots, m_{i_n}\}$ costituisce un matching di M . Viceversa, se $M' = \{m_{i_1}, m_{i_2}, \dots, m_{i_q}\}$ costituisce un matching di M si ha che

$$\sum_{x \in B} s(x) = \sum_{x \in A \setminus B} s(x),$$

dove $B = \{b_1\} \cup \{a_{i_1}, a_{i_2}, \dots, a_{i_q}\}$. □

11.11 Problemi SubsetSum e Knapsack

Definizione 53. Il PROBLEMA `SubsetSum` consiste nel determinare per ogni tripla (A, s, k) , dove A è un insieme finito, $s : A \rightarrow \mathbb{N} \setminus \{0\}$ è la “funzione dimensione” e $k \in \mathbb{N} \setminus \{0\}$ se esiste un sottoinsieme B di A tale che

$$\sum_{x \in B} s(x) = k.$$

■

Teorema 41. `Partition` \leq_p `SubsetSum`.

Dimostrazione. Siano A un insieme finito e $s : A \rightarrow \mathbb{N} \setminus \{0\}$. Poniamo

$$k = \begin{cases} \frac{s(A)}{2}, & \text{se } s(A) \text{ è pari} \\ s(A) + 1, & \text{altrimenti,} \end{cases}$$

dove

$$s(A) = \sum_{x \in A} s(x).$$

Si verifica che se B è un sottoinsieme di A allora $\sum_{x \in B} s(x) = \sum_{x \in A \setminus B} s(x)$ se e solo se $\sum_{x \in B} s(x) = k$. □

Definizione 54. Il PROBLEMA `Knapsack` consiste nel determinare per ogni data quintupla (A, s, v, k, h) , dove A è un insieme finito, $s, v : A \rightarrow \mathbb{N} \setminus \{0\}$ e $k, h \in \mathbb{N} \setminus \{0\}$ se esiste un sottoinsieme B di A tale che

$$\sum_{x \in B} s(x) \leq k \quad \text{e} \quad \sum_{x \in B} v(x) \geq h.$$

■

Teorema 42. `Partition` \leq_p `Knapsack`.

Dimostrazione. Siano A un insieme finito e $s : A \rightarrow \mathbb{N} \setminus \{0\}$. Poniamo

$$k = h = \begin{cases} \frac{s(A)}{2}, & \text{se } s(A) \text{ è pari} \\ s(A) + 1, & \text{altrimenti} \end{cases} \quad \text{e} \quad v = s,$$

dove

$$s(A) = \sum_{x \in A} s(x).$$

Si verifica che se B è un sottoinsieme di A allora $\sum_{x \in B} s(x) = \sum_{x \in A \setminus B} s(x)$ se e solo se $\sum_{x \in B} s(x) \leq k$ e $\sum_{x \in B} v(x) \geq h$. \square

12 Complessità Spaziale delle Macchine di Turing

Definizione 55. Sia \mathfrak{M} una TM e sia $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_n)$ una computazione di \mathfrak{M} .

Lo SPAZIO USATO DA (\mathfrak{M} IN) Γ è il numero $\text{space}(\Gamma)$ delle caselle del nastro che esamina \mathfrak{M} durante l'effettuazione di Γ , dove ogni casella viene contata una sola volta. In maniera più precisa, $\text{space}(\Gamma)$ è definito come segue. Per $i = 0, \dots, n-1$ sia $\mathcal{I}_i = q_i t_i o_i p_i$ l'istruzione di \mathfrak{M} tale che $\gamma_i \vdash_{\{\mathcal{I}_i\}} \gamma_{i+1}$ (si osservi che \mathcal{I}_i è unica) e sia

$$s_{i+1} = s_i + \begin{cases} 1, & \text{se } o_i = \uparrow \\ -1, & \text{se } o_i = \downarrow \\ 0, & \text{altrimenti,} \end{cases}$$

dove $s_0 = 0$. Poniamo

$$\text{space}(\Gamma) = \max(\{s_0, s_1, \dots, s_n\}) - \min(\{s_0, s_1, \dots, s_n\}) + 1.$$

■

Teorema 43. Riferendoci alle notazioni precedenti, si ha che:

- (1) $\max(\mathbf{S}_i) - \min(\mathbf{S}_i) \leq i$;
- (2) $\text{space}(\Gamma) \leq |\Gamma| + 1$;
- (3) $\gamma_i = X_i q_i t_i Y_i$ dove q_i è il simbolo di stato di γ_i e t_i è il simbolo scandito in γ_i e le stringhe X_i e Y_i hanno le seguenti lunghezze:
 - (3.a) $|X_i| = s_i - \min(\mathbf{S}_i)$;
 - (3.b) $|Y_i| = \max\{\max(\mathbf{S}_i), \ell\} - s_i$;
- (4) $|\gamma_i| = \max(\mathbf{S}_i) - \min(\mathbf{S}_i) + 2 + \max\{0, \ell - \max(\mathbf{S}_i)\}$;
- (5) $|\gamma_i| \leq \text{space}(\Gamma) + 1 + \ell$;

per ogni $i = 0, 1, \dots, n$, dove $\mathbf{S}_i = \{s_0, s_1, \dots, s_i\}$ e $\ell = \max\{0, |\text{Input}(\Gamma)| - 1\}$.

Dimostrazione. (1): Per induzione su i .

Per $i = 0$ si ha che $\max(\mathbf{S}_i) - \min(\mathbf{S}_i) = s_0 - s_0 = 0 \leq i$. Supponiamo, per ipotesi induttiva, che $\max(\mathbf{S}_j) - \min(\mathbf{S}_j) \leq j$, dove $0 \leq j < n$. Se $o_j \notin \{\uparrow, \downarrow\}$ allora $\mathbf{S}_{j+1} = \mathbf{S}_j$ e quindi $\max(\mathbf{S}_{j+1}) - \min(\mathbf{S}_{j+1}) = \max(\mathbf{S}_j) - \min(\mathbf{S}_j) \leq j < j+1$. Se $o_j = \uparrow$ allora $\mathbf{S}_{j+1} = \mathbf{S}_j \cup \{s_j + 1\}$ che implica che $\min(\mathbf{S}_{j+1}) = \min(\mathbf{S}_j)$ e $\max(\mathbf{S}_{j+1}) \leq \max(\mathbf{S}_j) + 1$. Da queste ultime relazioni segue che $\max(\mathbf{S}_{j+1}) - \min(\mathbf{S}_{j+1}) \leq \max(\mathbf{S}_j) + 1 - \min(\mathbf{S}_j)$ e quindi, per l'ipotesi induttiva, si ha che $\max(\mathbf{S}_{j+1}) - \min(\mathbf{S}_{j+1}) \leq j+1$. Infine, se $o_j = \downarrow$ si ha che $\mathbf{S}_{j+1} = \mathbf{S}_j \cup \{s_j - 1\}$ e usando argomentazioni simili a quelle precedenti si conclude ancora che $\max(\mathbf{S}_{j+1}) - \min(\mathbf{S}_{j+1}) \leq j+1$.

(2): Segue da (1).

(3): Per induzione su i .

(4): Segue da (3).

(5): Si osservi che $|\gamma_0| \leq |\gamma_1| \leq \dots \leq |\gamma_n|$ (perché $\gamma_0 \vdash_{\mathfrak{M}} \gamma_1 \vdash_{\mathfrak{M}} \dots \vdash_{\mathfrak{M}} \gamma_n$) e quindi (per la (4)) si ha che

$$\begin{aligned} |\gamma_i| &\leq \max(\mathbf{S}_n) - \min(\mathbf{S}_n) + 2 + \max\{0, \ell - \max(\mathbf{S}_n)\} \\ &= \mathbf{space}(\Gamma) + 1 + \max\{0, \ell - \max(\mathbf{S}_n)\}. \end{aligned}$$

D'altra parte, poiché $\max(\mathbf{S}_n) \geq 0$ si ha che $\max\{0, \ell - \max(\mathbf{S}_n)\} \leq \ell$ e quindi

$$|\gamma_i| \leq \mathbf{space}(\Gamma) + 1 + \ell.$$

□

Definizione 56. Siano \mathfrak{M} una TM, \mathcal{L} un linguaggio e $s : \mathbb{N} \rightarrow \mathbb{N}$ una funzione. Diciamo che \mathfrak{M} ACCETTA (O RICONOSCE) \mathcal{L} USANDO SPAZIO (LIMITATO DA) s se

(1) \mathcal{L} è accettato da \mathfrak{M} e

(2) per ogni stringa $X \in \mathcal{L}$ esiste una computazione Γ di \mathfrak{M} tale che $\mathbf{Input}(\Gamma) = X$ e $\mathbf{space}(\Gamma) \leq s(|X|)$. ■

Teorema 44. Sia \mathfrak{M} una TM e sia $c_{\mathfrak{M}} = m(r+2)^2$, dove m è il numero degli stati di \mathfrak{M} ed r è il numero dei simboli dell'alfabeto di \mathfrak{M} . Per ogni computazione propria Ω di \mathfrak{M} esiste una computazione Γ di \mathfrak{M} tale che:

(1) $\mathbf{Input}(\Gamma) = \mathbf{Input}(\Omega)$;

(2) $|\Gamma| \leq (\mathbf{space}(\Omega) + |\mathbf{Input}(\Omega)|)c_{\mathfrak{M}}^{\mathbf{space}(\Omega) + |\mathbf{Input}(\Omega)|}$

Dimostrazione. Data la computazione propria Ω di \mathfrak{M} , sia $\Gamma = (\gamma_0, \gamma_1, \dots, \gamma_k)$ una computazione di \mathfrak{M} di lunghezza minimale tale che $\mathbf{Input}(\Gamma) = X$ e $\mathbf{space}(\Gamma) \leq S$, dove $X = \mathbf{Input}(\Omega)$ e $S = \mathbf{space}(\Omega)$.

Poiché Γ ha lunghezza minimale, si ha che $\gamma_i \neq \gamma_j$ per ogni $0 \leq i < j \leq k$,⁶ cioè le configurazioni di Γ sono tutte distinte tra loro. Inoltre, dato che $|\gamma_0| \leq |\gamma_1| \leq \dots \leq |\gamma_k|$ e $|\gamma_k| \leq \mathbf{space}(\Gamma) + 1 + \max\{0, |X| - 1\}$ (cf. (5) del Teorema 43), si ha che $|\gamma_i| \leq S + 1 + |X|$, per ogni $i = 0, 1, \dots, k$. Pertanto $k + 1 \leq N$ dove N è il numero delle configurazioni di \mathfrak{M} di lunghezza minore o uguale a $S + 1 + |X|$. Determiniamo N . Per ogni $\ell \geq 1$ il numero delle

⁶Infatti, se si avesse $\gamma_i = \gamma_j$, dove $i < j$, allora la sequenza $(\gamma_0, \dots, \gamma_i, \gamma_{j+1}, \dots, \gamma_k)$ sarebbe una computazione di \mathfrak{M} con input X , di lunghezza minore della lunghezza di Γ e che usa spazio al più S , contraddicendo la "minimalità" di Γ .

configurazioni di \mathfrak{M} di lunghezza ℓ è uguale a $(r+1)^{\ell-1}m(\ell-1)$, e quindi

$$\begin{aligned}
N &= \sum_{\ell=1}^{S+1+|X|} (r+1)^{\ell-1}m(\ell-1) = \sum_{\ell=0}^{S+|X|} (r+1)^{\ell}m\ell \\
&= m \sum_{\ell=0}^{S+|X|} (r+1)^{\ell}\ell \\
&\leq m(S+|X|) \sum_{\ell=0}^{S+|X|} (r+1)^{\ell} \\
&\leq m(S+|X|) \sum_{\ell=0}^{S+|X|} (r+2)^{\ell} \\
&= m(S+|X|) \frac{(r+2)^{S+1+|X|} - 1}{r+1} \\
&\leq m(S+|X|)(r+2)^{S+1+|X|} \\
&\leq m(S+|X|)[(r+2)^2]^{S+|X|} \\
&\leq (S+|X|)[m(r+2)^2]^{S+|X|} = (S+|X|)c_{\mathfrak{M}}^{S+|X|}.
\end{aligned}$$

Pertanto $N \leq (S+|X|)c_{\mathfrak{M}}^{S+|X|}$ e quindi $|\Gamma| = k \leq (S+|X|)c_{\mathfrak{M}}^{S+|X|}$. \square

Definizione 57. La CLASSE DEI LINGUAGGI ACCETTATI IN SPAZIO NONDETERMINISTICO POLINOMIALE è la classe **NPSPACE** dei linguaggi \mathcal{L} per cui esistono una TM \mathfrak{M} ed una funzione polinomiale p tali che \mathfrak{M} accetta \mathcal{L} usando spazio p .

La CLASSE DEI LINGUAGGI ACCETTATI IN SPAZIO DETERMINISTICO POLINOMIALE è la classe **PSPACE** dei linguaggi \mathcal{L} per cui esistono una DTM \mathfrak{D} ed una funzione polinomiale p tali che \mathfrak{D} accetta \mathcal{L} usando spazio p . \blacksquare

Teorema 45. Se $\mathcal{L} \in \mathbf{NPSPACE}$ allora esistono $c \in \mathbb{N}$ ed una funzione polinomiale p tali che $\mathcal{L} \in \mathbf{NTIME}(t)$, dove $t(n) = p(n)(c+2)^{p(n)}$ per ogni $n \in \mathbb{N}$.

Dimostrazione. Sia $\mathcal{L} \in \mathbf{NPSPACE}$ e siano \mathfrak{M} una TM e q una funzione polinomiale tali che \mathfrak{M} riconosce \mathcal{L} usando spazio q . Ponendo $c = m(r+2)^2 - 2$ e $p(n) = n + q(n)$, dove m è il numero degli stati di \mathfrak{M} ed r è il numero dei simboli dell'alfabeto di \mathfrak{M} , dal Teorema 44 segue che $\mathcal{L} \in \mathbf{NTIME}(p(n)(c+2)^{p(n)})$. \square

Teorema 46. **PSPACE** \subseteq **NPSPACE**. Se $\mathcal{L} \in \mathbf{NPSPACE}$ allora \mathcal{L} è decidibile.

Dimostrazione. Se $\mathcal{L} \in \mathbf{NPSPACE}$, la decidibilità di \mathcal{L} segue dal Teorema 45 usando argomentazioni simili a quelle del Teorema 6. \square

Teorema 47. **NP** \subseteq **NPSPACE**.

Dimostrazione. Segue da (2) del Teorema 43. \square

Teorema 48 (Teorema di Savitch). **PSPACE** = **NPSPACE**.

Definizione 58. Indichiamo con **EXPTIME** la classe dei linguaggi \mathcal{L} per i quali esiste una funzione polinomiale $p(n)$ tale che $\mathcal{L} \in \mathbf{TIME}(2^{p(n)})$. ■

Teorema 49. **PSPACE** \subseteq **EXPTIME**.

Dimostrazione. Segue dal Teorema 44 (Si veda la dimostrazione del Teorema 45). □

Teorema 50. **P** \subseteq **NP** \subseteq **NPSpace** = **PSPACE** \subseteq **EXPTIME**.

Dimostrazione. Segue dai teoremi precedenti ... □

Teorema 51. **P** \neq **EXPTIME**.