

Teo 2.1: $\forall \text{NT } \exists T: \forall \text{ possibile input } x \text{ di NT } [O_{NT}(x) = O_T(x)]$.

dim. (Tecnica della simulazione)

Costruiamo una macchina deterministica T che simula il comportamento della macchina non deterministica NT con grado di non determinismo K . La macchina T su input x esegue una visita in ampiezza, basata sulla tecnica della coda di rondine con ripetizioni, dell'albero corrispondente alla computazione $NT(x)$.

Partendo dallo stato globale iniziale $s_0(T, x, 0)$ si simulano tutte le computazioni (lunghe un passo al più K); se non possiamo concludere nulla allora torniamo a $s_0(T, x, 0)$ e simuliamo tutte le computazioni lunghe due passi (al più K^2) e così via. □

Un linguaggio L è un sottoinsieme di Σ^* : $L \subseteq \Sigma^*$

Il linguaggio complemento L^c di un linguaggio $L \subseteq \Sigma^*$ è l'insieme delle parole non contenute in L : $L^c = \Sigma^* - L$

Un linguaggio $L \subseteq \Sigma^*$ è accettabile se esiste una macchina di Turing T tale che $\forall x \in \Sigma^* [O_T(x) = q_A \Leftrightarrow x \in L]$

La macchina T è detta accettare L . Nel caso in cui $x \notin L$ $O_T(x) = q_R$ oppure $T(x)$ non raggiunge mai uno stato finale, l'accettabilità di L non dà alcuna indicazione circa l'accettabilità di L^c .

Un linguaggio $L \subseteq \Sigma^*$ è decidibile se esiste una macchina di Turing T che termina $\forall x \in \Sigma^*$ e tale che:

La macchina T è detta decidere L . Se una macchina T decide $L \subseteq \Sigma^*$ allora

$$O_T(x) = \begin{cases} q_A & \text{se } x \in L \\ q_R & \text{se } x \notin L \end{cases}$$

Teo 3.1: Un linguaggio $L \subseteq \Sigma^*$ è decidibile sse L e L^c sono accettabili.

dim:

\Rightarrow Se L è decidibile $\Rightarrow \exists T: \forall x \in \Sigma^* O_T(x) = \begin{cases} q_A & \text{se } x \in L \\ q_R & \text{se } x \notin L \end{cases}$

dunque $O_T(x) = q_A$ sse $x \in L$, ossia T accetta L ,

deriviamo una nuova macchina T' con stati finali invertiti.

$\forall x \in \Sigma^*$, la computazione di $T'(x)$ coincide con $T(x)$ tranne per l'ultima istruzione dove se $T(x)$ termina in q_A , $T'(x)$ in q_R , analogo per q_R .

T accetta x sse $x \in L$ ossia T accetta L :

\Leftarrow) Se L e L' sono accettabili $\Rightarrow \exists T_1, T_2 : \forall x \in \Sigma^* T_1(x)$ accetta sse $x \in L$ e $T_2(x)$ accetta sse $x \in L'$. Definiamo una nuova macchina T che, simulando le computazioni eseguite da T_1 e da T_2 su input $x \in \Sigma^*$ decide L nella seguente maniera:

1) esegui una singola istruzione di T_1 sul nastro 1: se T_1 entra in q_A allora T accetta altrimenti esegui il passo 2).

2) esegui una singola istruzione di T_2 sul nastro 2: se T_2 entra in q_A allora T rigetta altrimenti esegui il passo 1).

In questo modo si ha garanzia della terminazione o di T_1 o di T_2 .

Quindi T decide L .

Siano Σ e Σ' , due alfabeti finiti; una funzione (parziale) $f : \Sigma^* \rightarrow \Sigma'^*$ è una funzione calcolabile se esiste una macchina di Turing T di tipo trasduttore che, dato in input $x \in \Sigma^*$, termina con la stringa $f(x)$ scritta sul nastro di output sse $f(x)$ è definita.

Sia Σ un alfabeto finito ed $L \subseteq \Sigma^*$ un linguaggio. La funzione caratteristica $\chi_L : \Sigma^* \rightarrow \{0, 1\}$ di L è una funzione totale tale che:

$$\forall x \in \Sigma^* \quad \chi_L(x) = \begin{cases} 1 & \text{se } x \in L \\ 0 & \text{se } x \notin L \end{cases}$$

Teo 3.2: Un linguaggio L è decidibile sse la funzione χ_L è calcolabile.

dim.

\Rightarrow) Supponiamo che $L \subseteq \Sigma^*$ sia decidibile, allora esiste una macchina T tipo riconoscitore

con stato di accettazione q_A e di rifiuto q_R tale che: $O_T(x) = \begin{cases} q_A & \text{se } x \in L \\ q_R & \text{se } x \notin L \end{cases}$

A partire da T , definiamo una macchina di tipo trasduttore T' con due nastri, che, con input $x \in \Sigma^*$ opera nella seguente maniera:

- 1) sul primo nastro, è scritto l'input x , esegue la computazione $T(x)$;
- 2) se $T(x)$ termina nello stato q_A allora sul nastro di output sarà scritto 1 altrimenti 0.
Successivamente termina;

Poiché L è decidibile il passo 1 termina $\forall x$. Se $T(x) = q_A$ $T'(x)$ scrive 1 altrimenti 0.

Questo dimostra che X_L è calcolabile.

\Leftrightarrow) Supponiamo che X_L sia calcolabile, allora esiste una macchina T tipo trasduttore, che, $\forall x \in \Sigma^*$, calcola $X_L(x)$.

A partire da T , definiamo una macchina di tipo riconoscitore T' con due nastri, che, con input $x \in \Sigma^*$ opera nella seguente maniera:

- 1) sul primo nastro è scritto l'input x , esegue la computazione $T(x)$, scrivendo il risultato sul secondo nastro.

- 2) se sul secondo nastro è scritto 1 allora la computazione $T(x)$ termina nello stato di accettazione, altrimenti di rigetto.

Poiché X_L è totale il passo 1 termina $\forall x$. Se $X_L(x) = 1$ $T'(x) = q_A$ altrimenti se $X_L(x) = 0$ $T'(x) = q_R$. Questo dimostra che L è decidibile. \square

Teo 3.3: Se la funzione $f: \Sigma^* \rightarrow \Sigma^*$ è totale e calcolabile allora il linguaggio $L_f = \Sigma \times \Sigma^*$ è decidibile.

dim.

Poiché f è calcolabile e totale, allora esiste una macchina di Turing T di tipo trasduttore, che, $\forall x \in \Sigma^*$, calcola $f(x)$.

A partire da T , definiamo una macchina di tipo riconoscitore T' con due nastri, che, con input $\langle x, y \rangle$, con $x \in \Sigma^*$ e $y \in \Sigma^*$. Opera nella seguente maniera:

- 1) sul primo nastro è scritto l'input $\langle x, y \rangle$.

- 2) sul secondo nastro esegue la computazione $T(x)$, scrivendo il risultato z .

- 3) esegue un confronto tra z e y : se $z = y$ allora la computazione $T'(x)$ termina nello stato di accettazione, altrimenti nello stato di rigetto.

Poiché f è totale il passo 2) termina $\forall x$. Questo dimostra che L_f è decidibile. \square

Teo 3.4: Sia $f: \Sigma^* \rightarrow \Sigma^*$ una funzione. Se il linguaggio $L_f \subseteq \Sigma^* \times \Sigma^*$, è decidibile allora f è calcolabile.

dim.

Poiché $L_f \subseteq \Sigma^* \times \Sigma^*$ è decidibile, esiste una macchina di Turing di tipo riconoscitore T , tale che $\forall x \in \Sigma^*$ e $\forall y \in \Sigma^*$

$$Q_f(x, y) = \begin{cases} q_s & \text{se } y = f(x) \\ q_R & \text{altrimenti} \end{cases}$$

A partire da T , definiamo una macchina di tipo trasduttore T' , che, con input $x \in \Sigma^*$ opera nella seguente maniera:

- 1) scrive il valore $i=0$ su N_1 ;
- 2) enumera tutte le stringhe $y \in \Sigma^*$ la cui lunghezza è pari al valore scritto sul primo nastro, simulando per ciascuna di esse la computazione $T(x, y)$, cioè opera come segue:
 - a) sia y la prima stringa di lunghezza i non ancora enumerata, allora scrive y su N_2 .
 - b) esegue la computazione $T(x, y)$ su N_3 .
 - c) se $T(x, y)$ termina nello stato q_s allora scrive sul nastro di output la stringa y e termina, altrimenti, eventualmente incrementando il valore i scritto su N_1 se y era l'ultima stringa di lunghezza i , torna al passo 2).

Poiché L_f è decidibile il passo (b) termina $\forall x, y$. Se x appartiene al dominio di f allora $\exists \bar{y} \in \Sigma^* : \bar{y} = f(x)$ e quindi $\langle x, \bar{y} \rangle \in L_f$. Prima o poi la stringa \bar{y} verrà scritta su N_2 e $T'(x, \bar{y})$ terminerà in q_s . Questo dimostra che f è calcolabile. \square

Cor: Un linguaggio L è decidibile sse L^c è decidibile.

Teo 5.7: Se L_1 e L_2 sono due linguaggi accettabili allora $L_1 \cup L_2$ è un linguaggio accettabile. Se L_1 e L_2 sono due linguaggi accettabili allora $L_1 \cap L_2$ è un linguaggio accettabile.

dim.

L_1 e L_2 sono accettabili $\Rightarrow \exists T_1: \forall x \in \Sigma^* [O_{T_1}(x) = q_s \Leftrightarrow x \in L_1]$

$\exists T_2: \forall x \in \Sigma^* [O_{T_2}(x) = q_s \Leftrightarrow x \in L_2]$

Costuiamo la macchina T' che con input $x \in \Sigma^*$ opera nella seguente maniera:

1) esegui una singola istruzione di T_1 sul nastro 1: se T_1 entra in q_A allora T' accetta
 altrimenti esegui il passo 2).

2) esegui una singola istruzione di T_2 sul nastro 2: se T_2 entra in q_A allora T' accetta
 altrimenti esegui il passo 1).

dim 2.

L_1 e L_2 sono accettabili $\Rightarrow \exists T_1: \forall x \in \Sigma^* [O_{T_1}(x) = q_A \Leftrightarrow x \in L_1]$

$\exists T_2: \forall x \in \Sigma^* [O_{T_2}(x) = q_A \Leftrightarrow x \in L_2]$

Creiamo la macchina T' che con input $x \in \Sigma^*$ opera nella seguente maniera:

1) simula $T_1(x)$ se T_1 entra in q_A va al passo 2).

2) simula $T_2(x)$ se T_2 entra in q_A allora T' accetta.

Teo 5.8: Se L_1 e L_2 sono due linguaggi decidibili allora $L_1 \cup L_2$ è un linguaggio decidibile. Se L_1 e L_2 sono due linguaggi decidibili allora $L_1 \cap L_2$ è un linguaggio decidibile.

dim.

L_1 e L_2 sono accettabili $\Rightarrow \exists T_1: \forall x \in \Sigma^* [O_{T_1}(x) = q_A \Leftrightarrow x \in L_1]$

$\exists T_2: \forall x \in \Sigma^* [O_{T_2}(x) = q_A \Leftrightarrow x \in L_2]$

Creiamo la macchina T' che con input $x \in \Sigma^*$ opera nella seguente maniera:

1) simula $T_1(x)$ se T_1 entra in q_A T' accetta altrimenti va al passo 2).

2) simula $T_2(x)$ se T_2 entra in q_A allora T' accetta altrimenti T' rigetta.

dim 2.

L_1 e L_2 sono accettabili $\Rightarrow \exists T_1: \forall x \in \Sigma^* [O_{T_1}(x) = q_A \Leftrightarrow x \in L_1]$

$\exists T_2: \forall x \in \Sigma^* [O_{T_2}(x) = q_A \Leftrightarrow x \in L_2]$

Creiamo la macchina T' che con input $x \in \Sigma^*$ opera nella seguente maniera:

1) simula $T_1(x)$ se T_1 entra in q_A va al passo 2) altrimenti T' rigetta.

2) simula $T_2(x)$ se T_2 entra in q_A allora T' accetta altrimenti T' rigetta.

Tutti i modelli di calcolo definiti fino ad ora sono Turing-equivalenti, qualunque funzione calcolabile mediante uno di tali modelli è calcolabile anche mediante una macchina di Turing.

Tesi di Church-Turing: se la soluzione di un problema può essere descritta da una serie finita di passi elementari, allora esiste una macchina di Turing in grado di calcolarlo.
E' calcolabile tutto (e solo) ciò che può essere calcolato da una macchina di Turing.

Questa è universalmente accettata, ma tuttora non esiste per essa una dimostrazione formale.

Teo: Per ogni programma scritto in accordo con il linguaggio di programmazione PascalMinimo esiste una macchina di Turing T di tipo trasduttore che scrive sul nastro di output lo stesso valore fornito in output dal programma.

Teo: Per ogni macchina di Turing T di tipo riconoscitore ad un nastro esiste un programma P scritto in accordo alle regole di PascalMinimo tale che per ogni stringa x , se $T(x)$ termina nello stato finale $q_f \in \{q_p, q_r\}$ allora P con input x restituisce q_f in output.

Teo s.1: Sia Σ un insieme finito. Allora l'insieme Σ^* costituito dalle parole (di lunghezza finita) di caratteri di Σ è numerabile. □

Teo s.2: L'insieme \tilde{T} delle macchine di Turing definite sull'alfabeto $\{0,1\}$ e dotate di un singolo nastro (più eventualmente quello di output) è numerabile. □
dim.

Occorre dimostrare l'esistenza di una biezione fra tale insieme e l'insieme \mathbb{N} .
Sia T una macchina di Turing, rappresentiamo T mediante la parola $B_T \in \Sigma^*$ con $\Sigma = \{0,1,\oplus,\otimes,-,+,s,d\}$ come segue:

$$B_T = b^0(q_0) - b^0(q_1) \otimes b^0(q_{11}) - b_{11} - b_{12} - b^0(q_{12}) - m_1 \oplus \dots \oplus b^0(q_{h1}) - b_{h1} - b_{h2} - b^0(q_{h2}) - m_h \oplus$$

flagtail

Dove $b^0 : Q \rightarrow \{0,1\}^{n_q}$ una codifica binaria degli stati di T .

Fissati P , stato iniziale e finale, è fissato anche il comportamento della macchina di Turing su un qualunque $x \in \{0,1\}^*$. T'è univocamente rappresentata dalla parola $B_T \in \Sigma^*$:

$$\forall T, T' \in \tilde{T} : T \neq T' \Leftrightarrow B_T \neq B_{T'}$$

(Questo prova che abbiamo una biezione tra T e un sottoinsieme di Σ^* e, dunque, poiché Σ è un insieme finito, esso è numerabile.)

Possiamo trasformare la codifica β_T di una macchina di Turing T in un numero naturale $v(T)$ in modo tale che per ogni coppia di macchine di Turing distinte $T \neq T'$ $v(T) \neq v(T')$.

Sia dunque $T \in T$ una macchina di Turing ad un nastro (più, eventualmente, il nastro di output) e sia $\beta_T \in \Sigma^*$ la sua codifica. Trasformiamo $\beta_T \in \Sigma^*$ in una parola in $\{0, 1, \dots, 9\}^*$ nel seguente modo:

- sostituendo ogni carattere "s", "f" e "d" in β_T rispettivamente con i caratteri "5", "6" e "7";
- sostituendo ogni carattere "-" in β_T con il carattere "4";
- sostituendo ogni carattere " \oplus " e " \otimes " in β_T rispettivamente con i caratteri "3" e "2";
- premettendo il carattere "2" alla stringa ottenuta.

La parola in $\{0, 1, \dots, 9\}^*$ ottenuta può essere considerata come un numero espresso in notazione decimale: ecco il numero $v(T) \in \mathbb{N}$ associato a T . □

$$\begin{array}{ccccccccc} b^s(q_0) & - b^s(q_1) & \otimes b^s(q_{11}) & - b_{11} & - b_{12} & - b^s(q_{12}) & - m & \oplus \dots \\ \downarrow & \downarrow \\ 2 & 0 & 1 & 0 & 0 & 1 & 4 & 0 & 4 & 0 & 0 & 1 & 4 & 7 & 3 \dots \\ & & & & & & & & & & & & & & & \\ & & & & & & & & & & & & & & & \{5, 6, 7\} \end{array}$$

Consideriamo la funzione $v: T \rightarrow \mathbb{N}$ e definiamo il seguente linguaggio:

$$L_H = \{(i, x) : i \text{ è la codifica di una macchina di Turing } T; (x) \text{ termina}\} \subseteq \mathbb{N} \times \mathbb{N}$$

Se un numero naturale i non è la codifica di alcuna macchina di Turing in T , allora comunque si scelga $x \in \{0, 1\}^*$, $(i, x) \notin L_H$ (Halting problem)

Teo 5.3: L_H è un linguaggio accettabile.

dim.

Bisogna dimostrare che esiste una macchina di Turing T tale che:

$$\forall (i, x) \in \mathbb{N} \times \mathbb{N}, \Omega_T(i, x) = q_A \iff (i, x) \in L_H$$

La macchina T cercata è, una modifica della macchina di Turing universale U . La macchina T in questione presenta un paio di modifiche: T verifica che i sia la codifica di una macchina di Turing, se non lo è rigetta. Poi T simula $U(i, x)$, se termina in q_A o q_R allora $T(i, x) = q_A$.

Sia $(i, x) \in L_H$: allora, la computazione $T_i(x)$ termina e quindi, in virtù di quanto appena descritto circa la macchina T , la computazione $T(i, x)$ accetta.

Viceversa, sia $(i, x) \in \mathbb{N} \times \mathbb{N}$ tale che $T(i, x)$ accetta; poiché la computazione $T(i, x)$ simula la computazione $U(i, x)$, allora anche $U(i, x)$ termina e, dunque, i è la codifica di una macchina di Turing e $T_i(x)$ termina, quindi $(i, x) \in L_H$.

Teo 5.4: Il linguaggio L_H non è decidibile

dim.

Supponiamo per assurdo che L_H sia decidibile. Allora esiste una macchina di Turing T che:

$$T(i, x) = \begin{cases} q_A & \text{se } (i, x) \in L_H \\ q_R & \text{se } (i, x) \notin L_H \end{cases}$$

Da T possiamo, complementando gli stati di accettazione e di rigetto di T , derivare una nuova macchina T' che, terminando su ogni input, accetta tutte e solo le coppie $(i, x) \in \mathbb{N} \times \mathbb{N} - L_H$ ossia:

$$T'(i, x) = \begin{cases} q_A & \text{se } (i, x) \notin L_H \\ q_R & \text{se } (i, x) \in L_H \end{cases}$$

A partire da T deriviamo una terza macchina T^* che, opera su un singolo input $i \in \mathbb{N}$. Inoltre $T^*(i)$ accetta se $T'(i, i)$ accetta, mentre non termina se $T'(i, i)$ rigetta. È possibile farlo apportando a T' le seguenti modifiche:

- sostituendo lo stato q_R con un nuovo stato non finale q'_R in tutte le quintuple di T' che terminano nello stato q_R
- aggiungiamo alle quintuple di T' le quintuple $\langle q'_R, y, y, q'_R, f \rangle \forall y \in \{0, 1\}$

$$T^*(i) = \begin{cases} \text{non termina} & \text{se } T'(i, i) \text{ rigetta} \\ q_A & \text{se } T'(i, i) \text{ accetta} \end{cases}$$

Poiché T è un insieme numerabile e $T^* \in T$, allora deve esistere $K \in \mathbb{N}$ tale che $T^* = T_K$. Qual'è l'esito della computazione $T_K(K)$?
Se $T_K(K) = T^*(K)$ accettasse, allora $T'(K, K)$ dovrebbe accettare anch'essa, allora allora $(K, K) \in L_H$ ossia $T_K(K)$ non termina, ma per definizione di L_H , K è la codifica di una macchina di Turing $\hat{T}_K(K)$ termina.

(ma è assurdo dato che abbiamo supposto che $T_K(K)$ accetta e quindi termina)

Se $T^*(K)$ non termina, allora $T'(K, K)$ rigetta, quindi $(K, K) \in L_H^c$ dunque $T_K(K)$ termina.
Entrambe le ipotesi portano ad una contraddizione. Allora la macchina T non può esistere. Poiché T^* è ottenuta tramite semplici modifiche da T allora T non può esistere, quindi T non può esistere perciò L_H non è decidibile. □

Di conseguenza dato che: "un linguaggio L è decidibile sse L è accettabile e L^c è accettabile", poiché L_H è accettabile L_H^c non lo è.

Siano $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$ due linguaggi; diciamo che L_1 è (many to one) riducibile ad L_2 se esiste una funzione totale e calcolabile $f: \Sigma_1^* \rightarrow \Sigma_2^*$ tale che $\forall x \in \Sigma_1^* [x \in L_1 \Leftrightarrow f(x) \in L_2]$.

Se L_1 è riducibile ad L_2 scriviamo $L_1 \leq L_2$.

$(\exists f: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \text{ totale e calcolabile}: \forall (i, x) \in \mathbb{N} \times \mathbb{N} [i \in L_1 \Leftrightarrow f(i, x) \in L_2])$

P: Sia L_3 un linguaggio decidibile e sia L_4 un secondo linguaggio tale che $L_4 \neq L_3$: allora L_4 è decidibile.

dim.

Indichiamo con $f_{4,3}$ la funzione che riduce L_4 ad L_3 . L_3 è decidibile allora

$$\exists T_3: \forall x \in \Sigma_3^*, T_3(x) = \begin{cases} q_A & \text{se } x \in L_3 \\ q_B & \text{se } x \notin L_3 \end{cases}$$

$L_4 \leq L_3$ quindi $\exists f_{4,3}$ totale e calcolabile: $f(x) \in L_3 \Leftrightarrow x \in L_4$
 $\Rightarrow \exists T_f: \forall x \in \Sigma_4^* [x \in L_4 \Leftrightarrow T_f(x) \in L_3]$

Costruisco la macchina T : input $x \in \Sigma_4$

i) simulo $T_f(x)$, sia y l'output di $T_f(x)$.

2) simulo $T_3(y)$: se accetta $\Rightarrow q_A$.
se rigetta $\Rightarrow q_R$. \square

P: Sia L_1 un linguaggio non decidibile e sia L_2 un secondo linguaggio tale che $L_1 \leq L_2$: allora L_2 è non decidibile.

dim.

Supponiamo per assurdo che L_2 è decidibile \Rightarrow per il Teorema sopra dimostrato anche L_1 è decidibile, assurdo. \square

P: Sia L_3 un linguaggio accettabile e sia L_4 un secondo linguaggio tale che $L_4 \leq L_3$: allora L_4 è accettabile.

dim.

Indichiamo con $f_{4,3}$ la funzione che riduce L_4 ad L_3 . L_3 è accettabile allora

$$\exists T_3: \forall x \in \Sigma_3^*, T_3(x) = \begin{cases} q_A & \text{se } x \in L_3 \\ \text{acc} & \text{altrimenti} \end{cases}$$

$L_4 \leq L_3$, quindi $\exists f_{4,3}$ totale e calcolabile: $f(x) \in L_3 \Leftrightarrow x \in L_4$

$$\Rightarrow \exists T_f: \forall x \in \Sigma_4^* [x \in L_4 \Leftrightarrow T_f(x) \in L_3]$$

Costruisco la macchina T : input $x \in \Sigma_4$

1) simulo $T_f(x)$, sia y l'output di $T_f(x)$.

2) simulo $T_3(y)$: se accetta $\Rightarrow q_A$ \square

P: Sia L_1 un linguaggio non accettabile e sia L_2 un secondo linguaggio tale che $L_1 \leq L_2$: allora L_2 è non accettabile.

dim.

Supponiamo per assurdo che L_2 è accettabile \Rightarrow per il Teorema sopra dimostrato anche L_1 è accettabile, assurdo. \square

gode anche di proprietà riflessiva e transitiva.

Una misura di complessità è una funzione c che associa ad ogni suo problema possibile input un valore numerico che corrisponde al "costo" della computazione della macchina sull'input considerato.

Affinché una funzione f possa essere considerata una misura di compatibilità, essa deve soddisfare le due seguenti proprietà, note come assiomi di Blum:

1) la funzione f è definita solo per computazioni che terminano.

se una computazione $T(x)$ non termina, non ha senso considerare che tale computazione abbia come costo un valore finito;

2) f deve essere una funzione calcolabile.

ossia, deve esistere una macchina di Turing M che, ricevendo in input una macchina di Turing T ed un suo input x , calcola $f(T, x)$ ogniqualvolta $f(T, x)$ è definita (cioè, ogniqualvolta $T(x)$ termina) - questo significa che, il costo di una computazione $T(x)$, dobbiamo poterlo calcolarlo effettivamente.

Per macchina di Turing deterministica T (riconoscitore o trasduttore) definita su un alfabeto Σ , e $\forall x \in \Sigma^*$ tali che $T(x)$ termina, definiamo le due funzioni seguenti:

$dtime(T, x)$ = numero di istruzioni eseguite da $T(x)$.

$dspace(T, x)$ = numero di celle utilizzate da $T(x)$

Per macchina di Turing non deterministica NT (riconoscitore) definita su un alfabeto Σ ,

e $\forall x \in \Sigma^*$ tali che $NT(x)$ accetta definiamo le due funzioni seguenti:

$ntime(NT, x)$ = minimo numero di istruzioni eseguite da una computazione deterministica accettante di $NT(x)$.

$ntime$ e $nspac$ e sono funzioni parziali avendole definite solo per computazioni accettanti. Per poter estendere la definizione a computazioni che rigettano è necessario tenere in considerazione la assimmetria delle definizioni di accettazione e rigetto di una macchina non deterministica:

$NT(x)$ accetta se esiste una computazione deterministica che accetta.

$NT(x)$ rigetta se tutte le sue computazioni deterministiche rigettano.

Per macchina di Turing non deterministica NT (riconoscitore) definita su un alfabeto Σ ,

e $\forall x \in \Sigma^*$ tali che $NT(x)$ rigetta definiamo le due funzioni seguenti:

Anche con questa estensione, le funzioni $ntime$ e $nspac$ e restano funzioni parziali.

$\text{ntime}(NT, x)$ = massimo numero di istruzioni eseguite da una computazione deterministica
di $NT(x)$ se $NT(x)$ rigetta.

$\text{nspac}(NT, x)$ = massimo numero di celle utilizzate da una computazione deterministica
di $NT(x)$ se $NT(x)$ rigetta.

Teo 6.1: Sia T una macchina di Turing deterministica, definita su un alfabeto Σ (non contenente il simbolo \square) e un insieme di stati Q e sia $x \in \Sigma^*$ tale che $T(x)$ termina.

Allora:

$$\text{dspace}(T, x) \leq \text{dtime}(T, x) \leq \text{dspace}(T, x) |Q| (|\Sigma| + 1)^{\text{dspace}(T, x)}$$

dim.

$$1) \text{dspace}(T, x) \leq \text{dtime}(T, x)$$

Se $T(x)$ utilizza $\text{dspace}(T, x)$ celle di memoria, quelle celle deve almeno leggerle, per leggere ciascuna cella impiega un'istruzione.

$$2) \text{dtime}(T, x) \leq \text{dspace}(T, x) |Q| (|\Sigma| + 1)^{\text{dspace}(T, x)}$$

$\text{dspace}(T, x) |Q| (|\Sigma| + 1)^{\text{dspace}(T, x)}$ è il numero di stati globali possibili di T nel caso in cui non vengono utilizzate più di $\text{dspace}(T, x)$ celle della computazione $T(x)$: poiché ogni cella del nastro può contenere un simbolo di Σ oppure il blank, il numero di possibili configurazioni di $\text{dspace}(T, x)$ celle è: $(|\Sigma| + 1)^{\text{dspace}(T, x)}$; per ognuna di queste configurazioni la testina può trovarsi su una qualsiasi delle $\text{dspace}(T, x)$ celle e la macchina può essere in uno qualsiasi dei $|Q|$ stati interni.

$$(Chiamiamo K(T, x) = \text{dspace}(T, x) |Q| (|\Sigma| + 1)^{\text{dspace}(T, x)})$$

Una computazione (deterministica) è una successione di stati globali tali che si passa da uno stato globale al successivo eseguendo una quintupla.

Se $T(x)$ durasse più di $K(T, x)$ passi (senza uscire dalle $\text{dspace}(T, x)$ celle) allora sarebbe una successione di stati globali contenente almeno due volte uno stesso stato globale:

$$S_{G_1} \rightarrow S_{G_2} \rightarrow \dots \rightarrow S_{G_i} \xrightarrow{\text{quintupla}} \dots \rightarrow S_{G_{K(T,x)}}$$

Ma T è deterministica: a partire da s_{in} è possibile eseguire un'unica quintupla ed essa viene eseguita tutte le volte in cui $T(x)$ si trova in s_{in} . $T(x)$ sarebbe in loop (contro l'ipotesi che termina). □

Analogamente: Sia NT una macchina di Turing non deterministica, definita su un alfabeto Σ (non contenente il simbolo \square) e un insieme di stati Q e sia $x \in \Sigma^*$ tale che $T(x)$ termina. Allora:

$$n\text{space}(T, x) \leq n\text{time}(T, x) \leq n\text{space}(T, x) | Q | (| \Sigma | + 1)^{n\text{space}(T, x)}$$

Sia $f: \mathbb{N} \rightarrow \mathbb{N}$ una funzione totale e calcolabile, sia Σ un alfabeto finito e sia $x \in \Sigma^*$ indichiamo con $|x|$ il numero di caratteri di x (non cardinalità non è un insieme bensì lunghezza)

Un linguaggio $L \subseteq \Sigma^*$ è deciso in tempo (spazio) deterministico $f(n)$ se:

$\exists T$ che decide $L: \forall x \in \Sigma^* [n\text{time}(T, x) \leq f(|x|)] \quad (n\text{space}(T, x) \leq f(|x|))$.

Un linguaggio $L \subseteq \Sigma^*$ è accettato in tempo (spazio) non deterministico $f(n)$ se:

$\exists NT$ che accetta $L: \forall x \in L [n\text{time}(NT, x) \leq f(|x|)] \quad (n\text{space}(NT, x) \leq f(|x|))$.

Un linguaggio $L \subseteq \Sigma^*$ è deciso in tempo (spazio) non deterministico $f(n)$ se:

$\exists NT$ che decide $L: \forall x \in \Sigma^* [n\text{time}(NT, x) \leq f(|x|)] \quad (n\text{space}(NT, x) \leq f(|x|))$.

Teo 6.2: Sia $f: \mathbb{N} \rightarrow \mathbb{N}$ una funzione totale e calcolabile. Se $L \subseteq \Sigma^*$ è accettata da una macchina di Turing non deterministica NT tale che, $\forall x \in L \quad n\text{time}(NT, x) \leq f(|x|)$ allora L è decidibile.

dim.

f è totale e calcolabile $\Rightarrow \exists T_f: \forall n \in \mathbb{N} [O_T(n) = f(n)]$, con $f(n)$ scritto in unario sul nastro di output.

Costruiamo una nuova macchina non deterministica NT' , a tre nastri che decide L :

$\forall x \in \Sigma^* :$

1) $NT'(x)$ scrive $|x|$ in unario sul N_2 .

2) simula $T_f(|x|)$ utilizzando N_2 come nastro di input e N_3 come nastro di output.

3) simula $NT(x)$ utilizzando N_1 come nastro di input e N_3 come contatore del numero di istruzioni eseguite.

Le computazioni di NT' terminano sempre, inoltre:

- se $x \in L$ allora $NT(x)$ accetta in al più $f(|x|)$ passi e quindi $NT'(x)$ accetta
- se $x \notin L$ allora o $NT(x)$ rigetta in al più $f(|x|)$ passi e quindi $NT'(x)$ rigetta oppure $NT(x)$ non termina entro $f(|x|)$ passi e quindi $NT'(x)$ rigetta.

NT' decide L e quindi L è decidibile. □

Dimostrazione analoga per NSPACE .

Teo 6.6: Sia $L \subseteq \Sigma^*$ un linguaggio deciso da una macchina di Turing deterministica ad un nastro T : $\forall x \in \Sigma^*$ $\text{dSPACE}(T, x) = s(|x|)$ e sia $K > 0$ una costante. Allora:

-] \exists una macchina di Turing ad un nastro T' che decide L , e $\forall x \in \Sigma^*$, $\text{dSPACE}(T', x) \leq \lceil \frac{s(|x|)}{K} \rceil + O(|x|)$. (compressione lineare)

Teo 6.7: Sia $L \subseteq \Sigma^*$ un linguaggio deciso da una macchina di Turing deterministica ad un nastro T : $\forall x \in \Sigma^*$ $\text{dTime}(T, x) = t(|x|)$ e sia $K > 0$ una costante. Allora:

-] \exists una macchina di Turing ad un nastro T' che decide L , e $\forall x \in \Sigma^*$, $\text{dTime}(T', x) \leq \lceil \frac{t(|x|)}{K} \rceil + O(|x|)$.
-] \exists una macchina di Turing a due nastri T'' che decide L , e $\forall x \in \Sigma^*$, $\text{dTime}(T'', x) \leq \lceil \frac{t(|x|)}{K} \rceil + O(|x|)$. (accelerazione lineare).

$$\text{DTIME}[f(n)] = \{L \subseteq \{0,1\}^*: \exists T \text{ che decide } L \text{ e, } \forall x \in \{0,1\}^* [\text{dtime}(T, x) \in O(f(|x|))]\}$$

$$\text{NTIME}[f(n)] = \{L \subseteq \{0,1\}^*: \exists NT \text{ che accetta } L \text{ e, } \forall x \in L [\text{ntime}(T, x) \in O(f(|x|))]\}$$

$$\text{DSPACE}[f(n)] = \{L \subseteq \{0,1\}^*: \exists T \text{ che decide } L \text{ e, } \forall x \in \{0,1\}^* [\text{dspace}(T, x) \in O(f(|x|))]\}$$

$$\text{NSPACE}[f(n)] = \{L \subseteq \{0,1\}^*: \exists NT \text{ che accetta } L \text{ e, } \forall x \in L [\text{nspace}(T, x) \in O(f(|x|))]\}$$

$$\text{coDTIME}[f(n)] = \{L \subseteq \{0,1\}^*: L^c \in \text{DTIME}[f(n)]\}$$

$$\text{coNTIME}[f(n)] = \{L \subseteq \{0,1\}^*: L^c \in \text{NTIME}[f(n)]\}$$

$$\text{coDSPACE}[f(n)] = \{L \subseteq \{0,1\}^*: L^c \in \text{DSPACE}[f(n)]\}$$

$$\text{coNSPACE}[f(n)] = \{L \subseteq \{0,1\}^*: L^c \in \text{NSPACE}[f(n)]\}$$

Teo 6.8: $\forall f$ totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$

$$\text{DTIME}[f(n)] \subseteq \text{NTIME}[f(n)] \quad \text{e} \quad \text{DSPACE}[f(n)] \subseteq \text{NSPACE}[f(n)]$$

Una macchina di Turing deterministica è una particolare macchina di Turing non deterministica avendo grado di non determinismo pari a 1 e, ogni parola decisa in K passi è anche accettata in K passi. \square

Teo 6.9: $\forall f$ totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$

$$\text{DTIME}[f(n)] \subseteq \text{DSPACE}[f(n)] \quad \text{e} \quad \text{NTIME}[f(n)] \subseteq \text{NSPACE}[f(n)]$$

Sia $L \subseteq \{0,1\}^*$: $L \in \text{DTIME}[f(n)]$ allora $\exists T$ che decide L :

$$\forall x \in \{0,1\}^* \text{ dtime}(T,x) \in O(f(|x|)).$$

Poiché $\text{dspace}(T,x) \leq \text{dtime}(T,x)$ allora $\text{dspace}(T,x) \in O(f(|x|))$ e che dunque $L \in \text{DSPACE}[f(n)]$. \square

Teo 6.10: $\forall f$ totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$

$$\text{DSPACE}[f(n)] \subseteq \text{DTIME}[2^{O(1)f(n)}] \quad \text{e} \quad \text{NSPACE}[f(n)] \subseteq \text{NTIME}[2^{O(1)f(n)}]$$

Sia $L \subseteq \{0,1\}^*$: $L \in \text{DSPACE}[f(n)]$ allora $\exists T$ che decide L :

$$\forall x \in \{0,1\}^* \text{ dspace}(T,x) \in O(f(|x|)).$$

$$\begin{aligned} \text{Poiché } \text{dtime}(T,x) &\leq \text{dspace}(T,x) |Q| (|\Sigma| + 1)^{\text{dspace}(T,x)} \\ &= \text{dspace}(T,x) |Q| 3^{\text{dspace}(T,x)} \\ &= 2^{\log \text{dspace}(T,x)} |Q| [2^{\log 3}]^{\text{dspace}(T,x)} = |Q| 2^{\log \text{dspace}(T,x) + \text{dspace}(T,x) \log 3} \leq |Q| 2^{[1 + \log 3] \text{dspace}(T,x)} \end{aligned}$$

allora $\text{dtime}(T,x) \in O(2^{O(1)f(|x|)})$ e dunque $L \in \text{DTIME}[2^{O(1)f(n)}]$. \square

Teo 6.11: $\forall f$ totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$

$$\text{DTIME}[f(n)] = \text{coDTIME}[f(n)] \quad \text{e} \quad \text{DSPACE}[f(n)] = \text{coDSPACE}[f(n)]$$

Sia $L \subseteq \{0,1\}^*$: $L \in \text{DTIME}[f(n)]$ allora $\exists T$ che decide L :

$$\forall x \in \{0,1\}^* \text{ dtime}(T,x) \in O(f(|x|)).$$

Poiché T decide L allora $T(x) = q_A$ se $x \in L$, e $T(x) = q_R$ se $x \in \{0,1\}^* - L = L^c$.

Costruiamo una macchina T' identica a T ma con stati di accettazione e rifiuto invertiti. Dunque T' decide L^c e, $\forall x \in \{0,1\}^*$, $\text{dtime}(T',x) \in O(f(|x|))$.

Quindi $L^c \in \text{DTIME}[f(n)]$. Poiché L è un qualunque linguaggio in $\text{DTIME}[f(n)]$ e, quindi, L^c è un qualunque linguaggio in $\text{coDTIME}[f(n)]$, questo significa che:
 $\forall L^c \in \text{coDTIME}[f(n)], L^c \in \text{DTIME}[f(n)]$ - ossia $\text{coDTIME}[f(n)] \subseteq \text{DTIME}[f(n)]$,
 $\forall L \in \text{DTIME}[f(n)]$, poiché $L^c \in \text{DTIME}[f(n)]$, allora $L \in \text{coDTIME}[f(n)]$,
ossia $\text{DTIME}[f(n)] \subseteq \text{coDTIME}[f(n)]$. □

Teo 6.12: \forall coppie di funzioni $f: \mathbb{N} \rightarrow \mathbb{N}$ e $g: \mathbb{N} \rightarrow \mathbb{N}$ tali che $\exists n_0 \in \mathbb{N}: \forall n \geq n_0 [f(n) \leq g(n)]$ ossia $f(n) \leq g(n)$. $\text{DTIME}[f(n)] \subseteq \text{DTIME}[g(n)]$, $\text{NTIME}[f(n)] \subseteq \text{NTIME}[g(n)]$, $\text{DSPACE}[f(n)] \subseteq \text{DSPACE}[g(n)]$, $\text{NSPACE}[f(n)] \subseteq \text{NSPACE}[g(n)]$.

Sia $L \subseteq \{0,1\}^*$: $L \in \text{DTIME}[f(n)]$ allora $\exists T$ che decide L :
 $\text{dtime}(T, x) \in O(f(|x|)) \leq O(g(|x|))$. Questo significa che $L \in \text{DTIME}[g(n)]$. □

Teo 6.13: Esiste una funzione totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$:

$$\text{DTIME}[2^{f(n)}] \subseteq \text{DTIME}[f(n)]$$

Una funzione totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$ è **time-constructible** se esiste una macchina di Turing T di tipo trasduttore che, preso in input un intero n espresso in notazione unaria, scrive sul nastro output il valore $f(n)$ in unario e $\text{dtime}(T, n) \in O(f(n))$.

Una funzione totale e calcolabile $f: \mathbb{N} \rightarrow \mathbb{N}$ è **space-constructible** se esiste una macchina di Turing T di tipo trasduttore che, preso in input un intero n espresso in notazione unaria, scrive sul nastro output il valore $f(n)$ in unario e $\text{dspace}(T, n) \in O(f(n))$.

Teo 6.14: Siano $f: \mathbb{N} \rightarrow \mathbb{N}$ e $g: \mathbb{N} \rightarrow \mathbb{N}$ due funzioni tale che f è space-constructible e
 $\lim_{n \rightarrow \infty} \frac{g(n)}{f(n)} = 0$ allora, $\text{DSPACE}[g(n)] \subseteq \text{DSPACE}[f(n)]$, ossia $\exists L: L \in \text{DSPACE}[f(n)]$ e
 $L \notin \text{DSPACE}[g(n)]$.

Teo 6.15: Siano $f: \mathbb{N} \rightarrow \mathbb{N}$ e $g: \mathbb{N} \rightarrow \mathbb{N}$ due funzioni tale che f è time-constructible e
 $\lim_{n \rightarrow \infty} \frac{g(n) \log(g(n))}{f(n)} = 0$ allora, $\text{DTIME}[g(n)] \subseteq \text{DTIME}[f(n)]$, ossia $\exists L: L \in \text{DTIME}[f(n)]$ e
 $L \notin \text{DTIME}[g(n)]$

Teo 6.16: Sia $f: \mathbb{N} \rightarrow \mathbb{N}$ una funzione time-constructible. Allora $\forall L \in \text{NTIME}[f(n)]$, si ha che L è decidibile in tempo non deterministico in $O(f(n))$.

dim.
Sia $f: \mathbb{N} \rightarrow \mathbb{N}$ una funzione time-constructible, allora esiste una macchina di Turing di tipo trasduttore T_f che, avendo scritto sul suo nastro di input $n \in \mathbb{N}$ codificato in unario, in $O(f(n))$ passi scrive sul nastro di output il valore $f(n)$ codificato in unario.

Sia $L \subseteq \Sigma^*$ tale che $L \in \text{NTIME}[f(n)]$, allora esiste una macchina di Turing non deterministica NT che accetta L : $\forall x \in L, \text{ntime}(NT, x) \in O(f(|x|))$. Costruiamo, ora, a partire da T_f e NT, la macchina NT' descritta nella dimostrazione del Teo 6.2 che decide L :

- 1) Scrive su N_2 in unario; termina in $O(|x|)$ passi.
- 2) Simula $T_f(|x|)$ e scrive risultato in unario su N_3 ; termina in $O(f(|x|))$ passi.
- 3) finché legge 1 su N_3 esegue un'istruzione di NT(x), termina in $O(f(|x|))$ passi.

Se termina in q_1 o q_2 NT' termina nel medesimo stato, se viene letto 0 su N_3 NT' rigetta.

Dunque NT' decide L , e $\forall x \in \Sigma^*, \text{ntime}(NT, x) \in O(f(|x|))$. \square

Analogo per f space-constructible.

Teo 6.17: $\forall f$ time-constructible $f: \mathbb{N} \rightarrow \mathbb{N}$,

$$\text{NTIME}[f(n)] \subseteq \text{DTIME}[2^{O(f(n))}]$$

dim.

Sia $L \subseteq \Sigma^*$ tale che $L \in \text{NTIME}(f(n))$; allora esistono una macchina di Turing non deterministica NT che accetta L e una costante h tali che, $\forall x \in L, \text{ntime}(NT, x) \leq h f(|x|)$. Poiché f è time-constructible esiste una macchina di Turing T_f che, con input la rappresentazione in unario di un intero n , calcola la rappresentazione in unario di $f(n)$ in tempo $O(f(n))$.

Indichiamo con K il grado di non determinismo di NT e utilizziamo la tecnica della simulazione per definire una macchina di Turing deterministica T che simuli il comportamento di NT: su input x , T simula tutte le computazioni deterministiche di NT(x) di lunghezza $hf(|x|)$:

- 1) Simula la computazione $T_f(|x|)$: $\forall x$, scrive sul nastro N_2 un carattere "1" e in

seguito, calcola $f(|x|)$ scrivendolo su N_3 . Infine concatena k volte il contenuto

2) Simula, una alla volta, tutte le computazioni deterministiche di $NT(x)$ di lunghezza $hf(|x|)$ utilizzando, per ciascuna computazione, la posizione della testina sul nastro N_3 come contatore.

Se $x \in L$, $ntime(NT, x) \leq hf(|x|)$ allora o in $hf(|x|)$ passi $NT(x)$ termina nello stato di accettazione oppure $x \notin L$. Quindi, se dopo aver simulato tutte le computazioni deterministiche di $NT(x)$ di lunghezza $hf(|x|)$, T non ha raggiunto lo stato di accettazione, allora può entrare nello stato di rifiuto. Questo prova che T decide L .

Detto K il grado di non determinismo di NT , il numero di computazioni deterministiche di $NT(x)$ di lunghezza $hf(|x|)$ è $K^{hf(|x|)}$ e ciascuna di esse viene simulata in $O(f(|x|))$ passi. Poiché il passo 1) descritto sopra richiede $O(f(|x|))$ passi, allora:

$$otime(T, x) \in O(f(|x|) K^{hf(|x|)}) \subseteq O(2^{O(f(|x|))}).$$

Quindi $L \in DTIME[2^{O(f(|x|))}]$. \square

$P = \bigcup_{k \in \mathbb{N}} DTIME[n^k]$: è la classe dei linguaggi decidibili in tempo deterministico polinomiale;

$NP = \bigcup_{k \in \mathbb{N}} NTIME[n^k]$: è la classe dei linguaggi accettabili in tempo non deterministico polinomiale;

$PSPACE = \bigcup_{k \in \mathbb{N}} DSPACE[n^k]$: è la classe dei linguaggi decidibili in spazio deterministico polinomiale;

$NPSPACE = \bigcup_{k \in \mathbb{N}} NSPACE[n^k]$: è la classe dei linguaggi accettabili in spazio non deterministico polinomiale;

Corollario 6.1: NP , $NPSPACE$ e $NEXPTIME$ sono le classi dei linguaggi decidibili, rispettivamente, in tempo non deterministico polinomiale, in spazio non deterministico polinomiale e in tempo non deterministico esponenziale.

Corollario 6.2: Valgono le relazioni di inclusione:

$$P \subseteq NP, PSPACE \subseteq NPSPACE \quad (\text{conseguenza del Teo 6.8})$$

$$P \subseteq PSPACE, NP \subseteq NPSPACE \quad (\text{conseguenza del Teo 6.9})$$

$$PSPACE \subseteq EXPTIME, NPSPACE \subseteq NEXPTIME \quad (\text{conseguenza del Teo 6.10})$$

$$NP \subseteq EXPTIME \quad (\text{conseguenza del Teo 6.17})$$

Corollario 6.3: $\text{coP} = P$.

Teo 6.18: $P \subseteq \text{EXPTIME}$. conseguenza del Teo 6.15 (gerarchia temporale)

Teo 6.19: $\text{PSPACE} = \text{NPSPACE}$.

Una possibile tecnica di dimostrazione che due classi di complessità C_1 e C_2 tali che $C_1 \subseteq C_2$ sono distinte consiste nell' individuare un linguaggio **separatore**, ossia, un linguaggio $L \in C_2 - C_1$.

È possibile individuare i linguaggi **candidati** ad essere separatori fra due classi utilizzando una nozione collegata al concetto di riducibilità funzionale, la nozione di linguaggio **completo** per una data classe.

Chiamiamo **π -riduzione** una riduzione funzionale che soddisfa il predicato π , e scriviamo $L_1 \leq_{\pi} L_2$.

- Una classe di complessità C è **chiusa** rispetto ad una generica π -riduzione se, per ogni coppia di linguaggi L_1 e L_2 tali che $L_1 \leq_{\pi} L_2$ e $L_2 \in C$, si ha che $L_1 \in C$.
- Sia C una classe di complessità di linguaggi e sia \leq_{π} una generica π -riduzione.
Un linguaggio $L \in \Sigma^*$ è **C -completo** rispetto alla π -riducibilità se:
 - a) $L \in C$ e
 - b) $\forall L' \in C$, vale che $L' \leq_{\pi} L$

Teo 6.20: Siano C e C' due classi di complessità tali che $C \subseteq C'$. Se C' è chiusa rispetto ad una π -riduzione allora, per ogni linguaggio L che sia C -completo rispetto a tale π -riduzione $L \in C'$ sse $C = C'$

dim.

\Rightarrow Se $C = C'$ allora $L \in C'$.

\Rightarrow Viceversa, supponiamo che $L \in C'$, poiché L è C -completo rispetto alla π -riducibilità, allora, $\forall L' \in C$, $L' \leq_{\pi} L$. Poiché C' è chiusa rispetto alla π -riduzione, questo implica che, $\forall L' \in C$, $L' \in C'$: quindi $C = C'$. □

Siano $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$ due linguaggi; diciamo che L_1 è **polinomialmente riducibile** ad L_2 , e scriviamo $L_1 \leq_p L_2$, se esiste una funzione totale e calcolabile $f: \Sigma_1^* \rightarrow \Sigma_2^*$: $f \in \text{FP}$ e $\forall x \in \Sigma_1^* [x \in L_1 \Leftrightarrow f(x) \in L_2]$.

Teo 6.21: La classe P è chiusa rispetto alla riducibilità polinomiale.

dim. (da qui in poi $\leq_p = \leq$)

Siano $L_1 \subseteq \Sigma_1^*$ e $L_2 \subseteq \Sigma_2^*$ due linguaggi: $L_1 \leq L_2$ e $L_2 \in P$. Indichiamo con $f: \Sigma_1 \rightarrow \Sigma_2$ la funzione in FP che riduce L_1 a L_2 , con T_f la macchina di Turing che calcola f in tempo polinomiale, e con T_2 la macchina di Turing deterministica che decide L_2 in tempo polinomiale.

Poiché T_f e T_2 operano in tempo polinomiale, esistono due costanti $h, K \in \mathbb{N}$ tali che $\forall x \in \Sigma_1^* \text{ e } \forall y \in \Sigma_2^*, \text{dtime}(T_f, x) \leq |x|^h \text{ e } \text{dtime}(T_2, x) \leq |y|^K$.

Combiniamo ora le due macchine per ottenere una macchina di Turing deterministica T_1 che decide L_1 . Opera nella seguente maniera:

- 1) simula $T_f(x)$ e scrive $f(x)$ sul secondo nastro.
- 2) simula $T_2(f(x))$ sul secondo nastro.

Se $T_2(f(x))$ accetta, $T_1(x)$ accetta, se $T_2(f(x))$ rigetta allora $T_1(x)$ rigetta, ($T_2(f(x))$ non può non terminare perché T_2 decide L_2): $T_1(x)$ accetta sse $T_2(f(x))$ accetta, ossia, $f(x) \in L_2$.

Ricordiamo che f è una riduzione da L_1 a L_2 e, quindi $f(x) \in L_2$ sse $x \in L_1$. T_1 termina sull'input x se $T_1(x)$ accetta sse $x \in L_1$, ossia T_1 decide L_1 .

La simulazione $T_f(x)$ richiede $\text{dtime}(T_f, x) \leq |x|^h$ passi e la simulazione di $T_2(f(x))$ richiede $\text{dtime}(T_2, f(x)) \leq |f(x)|^K$ passi: dunque, $\text{dtime}(T_1, x) \leq |x|^h + |f(x)|^K$. Poiché

$\text{dtime}(T_f, x) \leq |x|^h$ e T_f deve almeno scrivere il suo output $f(x)$, allora $|f(x)| \leq |x|^h$

quindi

$$\text{dtime}(T_1, x) \leq |x|^h + |f(x)|^K \leq |x|^h + (|x|^h)^K = |x|^h + |x|^{hk}$$

poiché h e K sono costanti $L_1 \in P$. □

Teo 6.22: Le classi NP, PSPACE, EXPTIME, NEXPTIME sono chiuse rispetto alla riducibilità polinomiale.

Corollario 6.4: Se $P \neq NP$ allora, per ogni linguaggio L NP-completo, $L \in P$.

dim.

Supponiamo L sia un linguaggio NP-completo e che $L \in P$. Poiché L è NP-completo, per ogni linguaggio $L' \in NP$, $L' \leq L$;

Se $L \in P$, poiché P è chiusa rispetto a \leq , questo implica che, per ogni $L' \in NP$, $L' \in P$. Ossia, $P = NP$, contraddicendo l'ipotesi.

Teo 6.23: Se $coNP \neq NP$, allora $P \neq NP$

dim.

Supponiamo che $P = NP$ allora $coP = coNP$ ma $coP = P$ allora $coNP = coP = P = NP$ □

Teo 6.24: La classe $coNP$ è chiusa rispetto alla riducibilità polinomiale.

dim.

poiché NP è chiusa rispetto alla riducibilità polinomiale, $\forall L_2 \in NP$ e $\forall L_1 : L_1 \leq L_2$
[$L_1 \in NP$] $\Rightarrow \forall L_2^c, L_1^c : L_1^c \leq L_2^c$ e $L_2^c \in coNP$ [$L_1^c \in coNP$] □

Teo 6.25: Un linguaggio L è NP-completo sse L^c è coNP-completo

dim.

\Rightarrow) Se L è NP-completo \Rightarrow

1) $L \in NP \Rightarrow L^c \in coNP$

2) $\forall L' \in NP [L' \leq L]$

$\forall L' \in NP$ esiste una funzione $f_{L'} : \Sigma^* \rightarrow \Sigma$ tale che $f_{L'} \in FP$ e

$$\forall x \in \Sigma^*, [x \in L' \text{ sse } f_{L'}(x) \in L]$$

$$\Rightarrow \forall x \in \Sigma^*, [x \notin L' \text{ sse } f_{L'}(x) \notin L]$$

$$\Rightarrow \forall x \in \Sigma^*, [x \in L'^c \text{ sse } f_{L'}(x) \in L^c]$$

quindi L^c è completo per $coNP$

\Leftarrow) Se L^c è coNP-completo \Rightarrow

1) $L^c \in coNP \Rightarrow L \in NP$

2) $\forall L' \in coNP [L'^c \leq L^c]$

$\forall L' \in coNP$ esiste una funzione $f_{L'} : \Sigma^* \rightarrow \Sigma$ tale che $f_{L'} \in FP$ e

$$\begin{aligned} & \forall x \in \sum^*, [\exists L' \text{ ssa } f_{L'}(x) \in L'] \\ \Rightarrow & \forall x \in \sum^*, [\exists L' \text{ ssa } f_{L'}(x) \notin L'] \\ \Rightarrow & \forall x \in \sum^*, [\exists L' \text{ ssa } f_{L'}(x) \in L] \end{aligned}$$

quindi L è completo per NP. \square

Teo 6.26: Se $\exists L \text{ NPC : } L \in \text{NP} \cap \text{coNP} \Rightarrow \text{NP} = \text{coNP}$

dim.

Se $L \in \text{NP} \cap \text{coNP}$ allora anche $L' \in \text{NP} \cap \text{coNP}$, poiché L è NP-completo $\Rightarrow L'$ è coNP-completo quindi $\forall L' \in \text{coNP} [L' \leq L]$. Ma NP è chiusa rispetto \subseteq e $L' \in \text{NP}$
 $\Rightarrow \forall L' \in \text{coNP} [L' \in \text{NP}]$ quindi $\text{coNP} \subseteq \text{NP}$.

Poiché L è NP-completo $\Rightarrow \forall L' \in \text{NP} [L' \leq L]$ ma $L \in \text{NP} \cap \text{coNP}$, quindi $L \in \text{coNP}$. Ma coNP è chiusa rispetto $\subseteq \Rightarrow \forall L' \in \text{NP} [L' \in \text{coNP}]$ quindi $\text{NP} \subseteq \text{coNP}$

Dunque $\text{NP} = \text{coNP}$ \square

Un problema è una quintupla $\langle I, R, S, \gamma, p \rangle$ in cui I è l'insieme delle **istanze**, R è l'insieme delle **risposte**, S è la funzione che specifica, per una data istanza, l'insieme delle **soluzioni possibili** per quell'istanza, $\gamma: S(I) \rightarrow 2^{S(I)}$ è la funzione che ricava dall'insieme delle soluzioni possibili per un'istanza del problema un insieme di **soluzioni effettive** di quella istanza e $p: I \times \gamma(S(I)) \rightarrow R$ è la richiesta del problema.

La risposta ad una istanza $x \in I$ di un problema è $p(x, \gamma(S(x)))$.

Possiamo definire diversi tipi di funzione p rispetto allo stesso insieme di istanze e rispetto alla stessa definizione di soluzioni effettive:

- 1) è un problema di **enumerazione**.
- 2) è un problema di **decisione**. (decisionali)
- 3) è un problema di **ricerca**.
- 4) è un problema di **ottimizzazione**.

Quindi un problema decisionale è descritto da $\langle I, S, \pi \rangle$.

3SAT

Dati un insieme X di variabili booleane ed un predicato f , definito sulle variabili X in forma 3CNF decidere se f è soddisfacibile.

$$X = \{x_1, \dots, x_n\}$$

$$f = c_1 \wedge \dots \wedge c_m \quad \text{clausola } c_j = \vee \text{ di 3 letterali } (x_n, \bar{x}_m)$$

$$I_{3SAT} = \{< f, X > : f \text{ è in 3CNF}\}$$

$$S_{3SAT}(f, X) = \{c_i : X \rightarrow \{0, 1\}\}$$

$$\Pi_{3SAT}(f, S_{3SAT}(f, X)) = \exists a \in S_{3SAT}(f, X) : f(a(x_1), \dots, a(x_n)) = \text{vero}$$

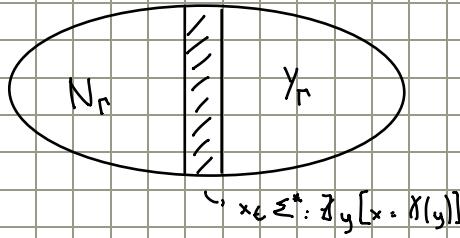
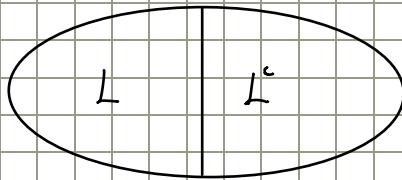
Sia $\Gamma = \langle I_\Gamma, S_\Gamma, \Pi_\Gamma \rangle$ un problema decisionale, e sia χ una codifica per I_Γ . La codifica χ è ragionevole per Γ se, per ogni altra codifica χ' per I_Γ esiste un intero $k \in \mathbb{N}$ tale che, $\forall x \in I_\Gamma$:

$$|\chi(x)| \in O(|\chi'(x)|^k).$$

Sia $\Gamma = \langle I_\Gamma, S_\Gamma, \Pi_\Gamma \rangle$ un problema decisionale, e sia $\chi : I_\Gamma \rightarrow \Sigma^*$ una codifica (ragionevole) per Γ . La codifica χ partiziona Σ^* in tre sottoinsiemi di parole:

- l'insieme Y_Γ delle parole che codificano istanze sì di Γ .
- l'insieme N_Γ delle parole che codificano istanze no di Γ .
- l'insieme delle parole che non codificano istanze di Γ .

È presente un'evidente asimmetria fra la partizione generata in Σ^* da un linguaggio e la partizione generata in Σ^* da un problema decisionale.



Dato un problema decisionale $\Gamma = \langle I_\Gamma, S_\Gamma, \Pi_\Gamma \rangle$ il suo problema complemento è:

$$\Gamma^c = \langle I_\Gamma, S_\Gamma, \neg \Pi_\Gamma \rangle$$

Dunque, fissata una codifica ragionevole $\chi: I_r \rightarrow \Sigma^*$ per il problema decisionale Γ , il linguaggio associato al problema Γ^c è:

$$L_{\Gamma^c}(\chi) = \{x \in \Sigma^*: x \in \chi(L_r)^c \wedge \pi_r(\chi^{-1}(x), S_r(\chi^{-1}(x)))\}.$$

L_{Γ^c} non coincide con $L_r^c(\chi)$ di $L_r(\chi)$.

$$L_r^c(\chi) = L_r^c \cup \{x \in \Sigma^*: x \notin \chi(I_r)\}.$$

Teo 2.1: Sia $\Gamma = \langle I_r, S_r, \pi_r \rangle$ un problema decisionale e sia $\chi: I_r \rightarrow \Sigma^*$ una codifica ragionevole. Se $\chi(I_r) \in P$ e $L_r(\chi) \in NP \Rightarrow L_{\Gamma^c}(\chi) \in coNP$

dim.

Poiché $\chi(I_r) \in P$, allora esistono una macchina di Turing deterministica T e un intero h tali che:
 $\forall x \in \Sigma^*, T$ decide se $x \in \chi(I_r)$ e $time(T, x) \in O(|x|^h)$.

Se $L_r(\chi) \in NP$, allora esistono una macchina di Turing non deterministica NT_r e un intero K t.c.:
 $\forall x \in L_r(\chi), NT_r$ accetta x e $ntime(NT_r, x) \in O(|x|^K)$. Combinando Te NT_r deriviamo una macchina NT' che con input x opera in due fasi:

- 1) simula $T(x)$: se $T(x)$ termina nello stato di rigetto allora NT' termina nello stato di accettazione, altrimenti va in fase 2.
- 2) simula $NT_r(x)$: se $NT_r(x)$ accetta allora NT' accetta.

Quindi NT' accetta sse $x \in \chi(I_r)$ oppure $x \in L_r(\chi)$, ossia, sse $x \in (L_r^c(\chi))^c$. Inoltre $ntime(NT', x) \in O(|x|^{max(h, K)})$. Quindi $(L_r^c(\chi))^c$ è in NP e $L_{\Gamma^c}(\chi) \in coNP$ □

Analogo EXPTIME, NEXPTIME. Poiché $coPSPACE = PSPACE = NPSPACE = coNPSPACE$ $L_{\Gamma^c}(\chi) \in PSPACE$ sse $L_{\Gamma^c}(\chi) \in PSPACE$.

Per dimostrare che un problema appartiene a P occorre trovare un algoritmo che lo risolve e dimostrare che richiede tempo polinomiale nella dimensione dell'input.

Quando si progetta un algoritmo non deterministico il numero delle opzioni fra le quali scegliere deve essere costante.

In tutti questi problemi π ha la forma seguente: esiste un elemento di S che soddisfa certe proprietà m

$$\pi(x, S(x)) = \exists y \in S(x) : \eta(x, y)$$

Inoltre tutti gli algoritmi seguono lo stesso schema: con input x ,
 fase 1 (non deterministica): sceglie una soluzione possibile $y \in S(x)$
 fase 2 (deterministica): verifica se y soddisfa il predicato $\eta(x, y)$

Entrambe le fasi richiedono tempo polinomiale in $|x|$.

Ogni problema con questa struttura è NP.

Problema 3SAT

Data una funzione booleana f , decidere se esiste una assegnazione di verità a tutte sue n variabili che soddisfa f .

$$I_{3SAT} = \{ \langle X, f \rangle : f = c_1 \wedge \dots \wedge c_n, \forall i [c_i = l_{i1} \vee l_{i2} \vee l_{i3}, l_{ij} \in X \vee \neg l_{ij} \in X] \}$$

$$S_{3SAT}(X, f) = \{ a : X \rightarrow \{0, 1\} \}$$

$$\pi_{3SAT}(X, f, S_{3SAT}(X, f)) = \{ a \in S_{3SAT}(X, f) : f(a(X)) = \text{vero} \}$$

Input: X, f

$i \leftarrow 1$

for ($i \leftarrow 1; i \leq |X|; i \leftarrow i + 1$) do

$\mathcal{O}(|X|)$

 scegli $a(x_i) \in \{\text{vero, falso}\}$

 for ($i \leftarrow 1; i \leq |X|; i \leftarrow i + 1$) do

$\mathcal{O}(|X|f)$

 sostituisce $a(x_i)$ con x_i in f e $\neg a(x_i)$ con $\neg x_i$

 if ($f(a(X)) = \text{vero}$) then $q \leftarrow q_a$

else $q \leftarrow q_b$

Output q

Teo 9.1: Un linguaggio $L \subseteq \Sigma^*$ è NP sse esistono una macchina di Turing deterministica T e una costante $K \in \mathbb{N}$ tali che $\forall x \in \Sigma^*$:

$$[x \in L \Leftrightarrow \exists y_x \in \{0,1\}^*: |y_x| \leq |x|^k \wedge T(x, y_x) \text{ accetta} \wedge \text{dtime}(T, x, y_x) \in O(|x|^k)]$$

dim.

\Rightarrow Sia $L \subseteq \Sigma^*$ un linguaggio in NP allora esistono una macchina di Turing non deterministica NT e un intero $h \in \mathbb{N}$ t.c. NT accetta L e, $\forall x \in L$ $\text{dtime}(NT, x) \leq |x|^h$.
 $\forall x \in L$ esiste una computazione deterministica di $NT(x)$ che termina nello stato di accettazione.

Poniamo, allora,

$$y(x) = q_1, s_1, s_{12}, q_{12}, m_1 - \dots - q_{(n^k)}, s_{(n^k)}, s_{(n^k)2}, q_{(n^k)2}, m_{(n^k)}$$

ossia $y(x)$ è la parola ottenuta concatenando le parole che corrispondono alla sequenza accettante di quintuple.

Definiamo ora una macchina di Turing deterministica a due nastri \bar{T} che corrisponde alla macchina NT : possiede codificata nelle sue quintuple, la descrizione dell'insieme P delle quintuple di NT . La computazione $\bar{T}(x, y)$ con input $x \in \Sigma^*$ scritto sul primo nastro e y scritto sul secondo nastro procede come segue:

- 1) \bar{T} verifica che y sia nella forma giusta altrimenti rigetta.
- 2) \bar{T} verifica che $\forall i: 1 \leq i \leq n^k, \langle q_i, s_i, s_{i2}, q_{i2}, m_i \rangle \in P$ altrimenti rigetta.
- 3) \bar{T} verifica che $q_{i1} = q_0$ e $q_{(n^k)2} = q_A$ altrimenti rigetta.
- 4) \bar{T} verifica che $\forall i: 2 \leq i \leq n^k, q_{i1} = q_{(i-1)2}$ altrimenti rigetta.
- 5) \bar{T} simula la computazione di $NT(x)$ descritta da y .
- 6) \bar{T} accetta.

Sia $x \in L$ allora $\exists y(x): \bar{T}(x, y(x))$ accetta, se invece $x \notin L$ allora qualunque sia $y(x)$ $\bar{T}(x, y(x))$ non può accettare: o $y(x)$ non codifica una possibile computazione accettante di NT . (rigetta al passo 1 o 2 o 3 o 4) o $y(x)$ codifica una computazione accettante di NT che però non può essere eseguita sull'input x e rigetta al passo 5).

Dunque $x \in L$ sse $\exists y(x): \bar{T}(x, y(x))$ accetta. \bar{T} opera in tempo polinomiale in $|x|$ e $|y(x)|$ e $y(x)$ è costituita di al più $|x|^k$ passi, quindi se $x \in L$, $|y(x)| \in O(|x|^k)$ e \bar{T} opera in tempo polinomiale in $|x|$. Infine basta trasformare \bar{T} in una macchina il cui alfabeto è $\{0,1\}$.

\Leftrightarrow Sia $L \subseteq \Sigma^*$ un linguaggio per il quale esistono una macchina di Turing T che opera in tempo polinomiale e una costante $K \in \mathbb{N}$ tali che $\forall x \in \Sigma^* \quad x \in L \Leftrightarrow \exists y \in \{0,1\}^{|x|} \text{ f.c.}$
 $|y_x| \leq |x|^K \wedge T(x, y_x) \text{ accetta.}$

Assumiamo T disponga di un solo nastro, inizialmente sono scritte x e y separate da "□".

Definiamo la macchina di Turing non deterministica NT che opera in due fasi:

- fase 1: genera una parola $y \in \{0,1\}^*$ di lunghezza al più $|x|^K$ scrivendolo alla destra dell'input x e separato da "□".

- fase 2: simula $T(x, y)$.

La prima fase richiede al più $O(|x|^K)$ passi e la seconda richiede tempo proporzionale a ottime $|T(x, y)|$, polinomiale in $|x|$.

Se $x \in L$, per ipotesi, esiste una parola y_x tale che $T(x, y_x)$ accetta, allora durante la fase 1 esiste una sequenza di scelte che genera y_x che nella fase 2 porta NT ad accettare. Di contro, se NT accetta x allora esiste una parola y_x generata durante la prima fase che induce la seconda fase ad accettare.

$x \in L$ sse $NT(x)$ accetta e dato che NT opera in tempo polinomiale: $L \in NP$. □

La parola y_x può essere considerata prova che x sia contenuto in L , una prova che deve essere successivamente verificata. Ad essa ci riferisce come ad un certificato in quanto certifica l'appartenenza di una parola ad un linguaggio. Ogni problema P in NP può essere formalizzato nel seguente modo:

- un insieme di istanze I_P .
- $\forall x \in I_P$, un insieme di soluzioni possibili $S_P(x)$ generabili non deterministicamente in tempo polinomiale.
- $\forall x \in I_P$, un predicato $\pi_P(x, S(x))$ nella forma $\pi_P(x, S(x)) = \exists y \in S_P(x) : \gamma_P(x, y)$ tale che $\gamma_P(x, y)$ è decidibile in tempo polinomiale di $|x|$.

Un problema è in NP sse le parole che esso contiene ammettono certificati verificabili in tempo polinomiale nella loro dimensione.

Teo di Cook-Levin: SAT è NP -completo.

Dati due problemi decisionali Γ_1 e Γ_2 allora $\Gamma_1 \leq \Gamma_2$ quando $L_{\Gamma_1} \leq L_{\Gamma_2}$ dove L sono i linguaggi associati alle codifiche ragionevoli delle istanze si dei problemi.

$\Gamma_1 \leq \Gamma_2$ se $\exists f: I_{\Gamma_1} \rightarrow I_{\Gamma_2}$ t.c.:

- $f \in FP$.

- x è una istanza si di Γ_1 sse $f(x)$ è un istanza si di Γ_2 .

non bisogna quindi utilizzare ogni volta Cook-Levin

Teo 9.3: Sia Γ_0 un problema in NP, se esiste un problema NP-completo riducibile a Γ_0 allora Γ_0 è NP-completo.

dim.

Sia Γ_1 un problema NP-completo tale che $\Gamma_1 \leq \Gamma_0 \Rightarrow \exists f_{10}: I_{\Gamma_1} \rightarrow I_{\Gamma_0}$ t.c. $f_{10} \in FP$ e $\forall x \in I_{\Gamma_1} [x \in \Gamma_1 \Leftrightarrow f_{10}(x) \in \Gamma_0]$. Poiché Γ_1 è NPC \forall problema $\Gamma_2 \in NP$ si ha che $\Gamma_2 \leq \Gamma_1$ e dunque $\exists f_{21}: I_{\Gamma_2} \rightarrow I_{\Gamma_1}$ t.c. $f_{21} \in FP$ e $\forall x \in I_{\Gamma_2} [x \in \Gamma_2 \Leftrightarrow f_{21}(x) \in \Gamma_1]$.

Poiché $f_{21} \in FP \Rightarrow \exists T_{21}, K \in \mathbb{N}: \forall x \in I_{\Gamma_2} [\text{dtime}(T_{21}, x) \leq |x|^k]$, analogamente

Poiché $f_{10} \in FP \Rightarrow \exists T_{10}, h \in \mathbb{N}: \forall x \in I_{\Gamma_1} [\text{dtime}(T_{10}, x) \leq |x|^h]$

$x \in \Gamma_2 \Leftrightarrow f_{21}(x) \in \Gamma_1 \Leftrightarrow f_{10}(f_{21}(x)) \in \Gamma_0$. Indichiamo con f_{20} la composizione delle funzioni.

Possiamo quindi definire T_{20} che calcola f_{20} , la computazione ha inizio con l'input $x \in \Gamma_2$ in seguito:

1) T_{20} simula $T_{21}(x)$ scrivendo $f_{21}(x)$ sul nastro di lavoro.

2) T_{20} simula $T_{10}(f_{21}(x))$ scrivendo $f_{10}(f_{21}(x))$ sul nastro output.

$\forall x \in \Gamma_2 \text{ dtime}(T_{20}, x) \leq |x|^k + |f_{21}(x)|^h \leq |x|^k + |x|^{kh} \leq 2|x|^{hk} \leq |x|^{hk+1}$, essendo h, k costanti

$f_{20} \in FP$. Perciò $\Gamma_2 \leq \Gamma_0$ e ogni problema in NP è $\leq \Gamma_0$. Dato che $\Gamma_0 \in NP$ segue che Γ_0 è NPC \square

Per dimostrare che un problema Γ è NPC è sufficiente:

- mostrare che $\Gamma \in NP$

- scegliere un problema NPC noto Γ_2 e dimostrare che $\Gamma_2 \leq \Gamma$

SAT

$$I_{SAT} = \{ \langle f, X \rangle : f \text{ è in CNF} \}$$

$$S_{SAT} (f, X) = \{ c: X \rightarrow \{0,1\} \}$$

$$\Pi_{SAT} (f, S_{SAT} (f, X)) = \exists c \in S_{SAT} (f, X) : f(c(x_1), \dots, c(x_n)) = \text{vero}$$

3SAT

$$I_{3SAT} = \{ \langle f, X \rangle : f \text{ è in 3CNF} \}$$

$$S_{3SAT}(f, X) = \{ c_i : X \rightarrow \{0, 1\} \}$$

$$\Pi_{3SAT}(f, X, S_{3SAT}(f, X)) = \exists c \in S_{3SAT}(f, X) : f(c(x_1), \dots, c(x_n)) = \text{vero}$$

VERTEX COVER VC

Un vertex cover è un insieme di nodi che copre tutti gli archi del grafo.

$$I_{VC} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$$

$$S_{VC}(G, K) = \{ V \subseteq V \}$$

$$\Pi_{VC}(G, K, S_{VC}(G, K)) = \exists V \in S_{VC}(G, K) : |V| \leq K \wedge \forall (u, v) \in E [u \in V \vee v \in V]$$

INDEPENDENT SET IS \rightarrow COL

Un independent set è un insieme di nodi tali che nessuna coppia di nodi condivide un arco

$$I_{IS} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$$

$$S_{IS}(G, K) = \{ I \subseteq V \}$$

$$\Pi_{IS}(G, K, S_{IS}(G, K)) = \exists I \in S_{IS}(G, K) : |I| \geq K \wedge \forall u, v \in I [(u, v) \notin E]$$

CLIQUE CL \rightarrow COL con G^c

Una clique è un insieme di nodi tali che ogni coppia di nodi condivide un arco

$$I_{CL} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$$

$$S_{CL}(G, K) = \{ C \subseteq V \}$$

$$\Pi_{CL}(G, K, S_{CL}(G, K)) = \exists C \in S_{CL}(G, K) : |C| \geq K \wedge \forall u, v \in C [(u, v) \in E]$$

DOMINATING SET DS

Una dominating set è un insieme D di nodi tali che ogni nodo che non è in D ha almeno un vicino in D .

$$I_{DS} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$$

$$S_{DS}(G, K) = \{ D \subseteq V \}$$

$$\Pi_{DS}(G, K, S_{DS}(G, K)) = \exists D \in S_{DS}(G, K) : |D| \leq K \wedge \forall v \in V - D [\exists v \in D : (u, v) \in E]$$

HAMILTONIAN CYCLE HC

Un ciclo hamiltoniano è un ciclo in G che passa una ad una sola volta attraverso ogni nodo di G

$$I_{HC} = \{ \langle G = (V, E) \rangle : G \text{ è un grafo non orientato} \}$$

$$S_{HC}(G) = \{ \langle v_1, \dots, v_n \rangle : \text{per } i=1, \dots, n \quad v_i \in V \wedge n = |V| \}$$

$$\Pi_{HC}(G, S_{HC}(G)) = \{ \langle v_1, \dots, v_n \rangle \in S_{HC}(G) : (v_n, v_1) \in E \wedge \forall i=1, \dots, n-1 \quad [(v_i, v_{i+1}) \in E] \wedge \\ \forall i, j = 1, \dots, n \text{ con } i \neq j \quad [v_i \neq v_j] \}$$

HAMILTONIAN PATH HP

Un percorso hamiltoniano da s a t è un percorso che passa una e una sola volta attraverso ciascun nodo di G

$$I_{HP} = \{ \langle G = (V, E), s, t \rangle : G \text{ è un grafo non orientato} \wedge s, t \in V \}$$

$$S_{HP}(G, s, t) = \{ \langle v_1, \dots, v_n \rangle : \text{per } i=1, \dots, n \quad v_i \in V \wedge n = |V| \}$$

$$\Pi_{HP}(G, s, t, S_{HP}(G, s, t)) = \{ \langle v_1, \dots, v_n \rangle \in S_{HP}(G, s, t) : s = v_1 \wedge t = v_n \wedge \forall i=1, \dots, n-1 \quad [(v_i, v_{i+1}) \in E] \wedge \\ \forall i, j = 1, \dots, n \text{ con } i \neq j \quad [v_i \neq v_j] \}$$

LONG PATH LP

Un long path è un percorso in G di almeno K archi.

$$I_{LP} = \{ \langle G = (V, E), s, t, K \rangle : G \text{ è un grafo non orientato} \wedge s, t \in V \wedge K \in \mathbb{N} \}$$

$$S_{LP}(G, s, t, K) = \{ \langle v_1, \dots, v_n \rangle : \text{per } i=1, \dots, n \quad v_i \in V \}$$

$$\Pi_{LP}(G, s, t, K, S_{LP}(G, s, t, K)) = \{ \langle v_1, \dots, v_n \rangle \in S_{LP}(G, s, t, K) : s = v_1 \wedge t = v_n \wedge \forall i=1, \dots, n-1 \quad [(v_i, v_{i+1}) \in E] \wedge \\ \forall i, j = 1, \dots, n \text{ con } i \neq j \quad [v_i \neq v_j] \wedge n \geq K \}$$

TRAVELLING SALESMAN PROBLEM TSP

Dato un grafo G non orientato, completo e pesato vedere se esiste un ciclo hamiltoniano tale che la somma dei pesi degli archi è $\leq K$.

$$I_{TSP} = \{ \langle G = (V, E, w), K \rangle : G \text{ è un grafo non orientato} \wedge w: E \rightarrow \mathbb{N} \wedge \forall u, v \in V \quad [(u, v) \in E] \wedge K \in \mathbb{N} \}$$

$$S_{TSP}(G, K) = \{ \langle v_1, \dots, v_n \rangle : \text{per } i=1, \dots, n \quad v_i \in V \wedge n = |V| \}$$

$$\Pi_{TSP}(G, K, S_{TSP}(G, K)) = \{ \langle v_1, \dots, v_n \rangle \in S_{TSP}(G, K) : \forall i, j = 1, \dots, n \text{ con } i \neq j \quad [v_i \neq v_j] \wedge \\ \sum_{(u,v)} w(u, v) \leq K \}$$

COLORABILITÀ COL

Dato un grafo G colorare ciascun nodo in G in modo che nodi adiacenti devono essere colorati con colori diversi.

$$I_{\text{col}} = \{<G=(V,E), K> : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N}\}$$

$$S_{\text{col}}(G, K) = \{c : V \rightarrow \{1, 2, \dots, K\}\}$$

$$\Pi_{\text{col}}(G, K, S_{\text{col}}(G, K)) = \exists c \in S_{\text{col}}(G, K) : \forall (u, v) \in E [c(u) \neq c(v)]$$

K-COLORABILITÀ K-COL

Dato un grafo G esiste un assegnazione di K colori in modo che nodi adiacenti devono essere colorati con colori diversi. K è costante.

$$I_{\text{kcol}} = \{<G=(V,E)> : G \text{ è un grafo non orientato}\}$$

$$S_{\text{kcol}}(G) = \{c : V \rightarrow \{1, \dots, K\}\}$$

$$\Pi_{\text{kcol}}(G, S_{\text{kcol}}(G)) = \exists c \in S_{\text{kcol}}(G) : \forall (u, v) \in E [c(u) \neq c(v)]$$

MIN2SAT

Se h non è una costante l'algoritmo di ricerca esaustiva ha complessità non polinomiale.

Dimostriamo che nonostante 2SAT ∈ P MIN2SAT è NPC.

$$I_{\text{MIN2SAT}} = \{<X, f, h> : f \text{ è in 2CNF} \wedge h \in \mathbb{N}\}$$

$$S_{\text{MIN2SAT}}(X, f, h) = \{a : X \rightarrow \{0, 1\}\}$$

$$\Pi_{\text{MIN2SAT}}(X, f, h, S_{\text{MIN2SAT}}(X, f, h)) = \exists a \in S_{\text{MIN2SAT}}(X, f, h) : f(a(x_1), \dots, a(x_n)) = \text{vero} \wedge \\ |\{x \in X : a(x) = \text{vero}\}| \leq h$$

Consideriamo una serie di problemi che combinano VC e 3COL:

$$1) \text{VC} \vee 3\text{COL}$$

$$2) \text{VC} \wedge 3\text{COL}$$

$$3) \text{VC} \vee \neg 3\text{COL}$$

$$4) \text{VC} \wedge \neg 3\text{COL}$$

$$1) I_{\text{VC} \vee 3\text{COL}} = \{<G=(V,E), K> : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N}\}$$

$$S_{\text{VC} \vee 3\text{COL}}(G, K) = \{(V, c) : V \subseteq V \wedge c : V \rightarrow \{1, 2, 3\}\}$$

$$\Pi_{\text{VC} \vee 3\text{COL}}(G, K, S_{\text{VC} \vee 3\text{COL}}(G, K)) = \exists (V, c) \in S_{\text{VC} \vee 3\text{COL}}(G, K) : [|V'| \leq K \wedge \forall (u, v) \in E [c(u) \neq c(v)] \wedge \\ \forall v \in V' \exists u \in V' \text{ tale che } c(u) = c(v)] \vee [\forall (u, v) \in E [c(u) \neq c(v)]]$$

1: $VC \wedge 3COL \in NP$

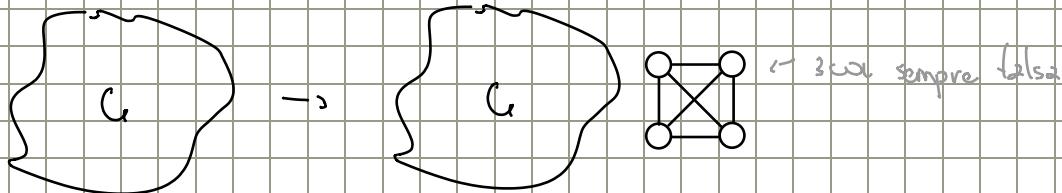
Un certificato è una coppia (V, c) e verificare se soddisfa il predicato richiede tempo polinomiale nell'istanza.

2: dimostrare $VC \wedge 3COL \in NPC$

$VC \leq VC \wedge 3COL$

$\langle G = (V, E), K \rangle \in VC \rightarrow \langle G' = (V', E'), K' \rangle \in VC \wedge 3COL$

G' è ottenuto aggiungendo a G una clique di 4 nodi e ponendo $K' = K + 3$



Se G ha un vc di K nodi $\Rightarrow G'$ ha un vc di $K+3$ nodi.

Se G' è $VC \wedge 3COL \Rightarrow$ poiché G' non è 3COL sicuramente avrà un istanza si di vc di K' .

2) $I_{VC \wedge 3COL} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato} \wedge K \in \mathbb{N} \}$

$S_{VC \wedge 3COL}(G, K) = \{ (V', c) : V \subseteq V \wedge c: V \rightarrow \{1, 2, 3\} \}$

$\Pi_{VC \wedge 3COL}(G, K, S_{VC \wedge 3COL}(G, K)) = \exists (V', c) \in S_{VC \wedge 3COL}(G, K) : [|V'| \leq K \wedge \forall (u, v) \in E[V \cup V'] \exists c(u) \neq c(v)] \wedge [\forall (u, v) \in E[c(u) \neq c(v)]]$

1: $VC \wedge 3COL \in NP$

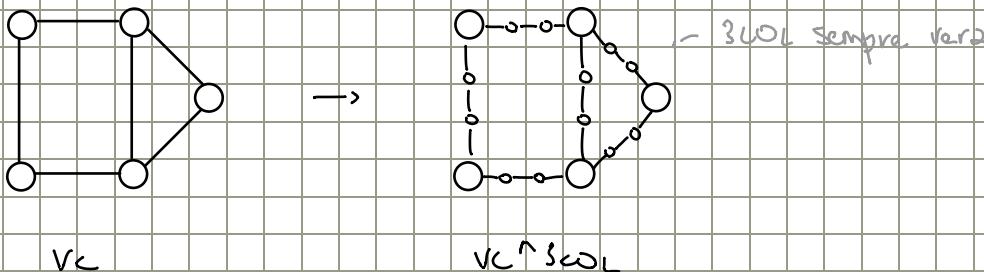
Un certificato è una coppia (V, c) e verificare se soddisfa il predicato richiede tempo polinomiale nell'istanza.

2: dimostrare $VC \wedge 3COL \in NPC$

$VC \leq VC \wedge 3COL$

$\langle G = (V, E), K \rangle \in VC \rightarrow \langle G' = (V', E'), K' \rangle \in VC \wedge 3COL$

G' è ottenuto sostituendo ad ogni arco in G una catena di 4 nodi e ponendo $K' = K + |E|$



G è 3COL perciò va dimostrato che G è istanza sì di $VC \cap 3COL$ se ha un vc di $K + |E|$ nodi. In qualunque VC per G i $|E|$ nodi devono essere quelli aggiuntivi per coprire gli archi nelle catene costruite.

Se G ha un vc di K nodi $\Rightarrow G$ ha un vc di $K + |E|$ nodi.

Se G non ha un vc di K nodi $\Rightarrow G$ non ha un vc di $K + |E|$ nodi.

4) $I_{VC \cap 3COL} = \{ \langle G = (V, E), K \rangle : G \text{ è un grafo non orientato } \wedge K \in \mathbb{N} \}$

$$S_{VC \cap 3COL}(G, K) = \{ (V', c) : V' \subseteq V \wedge c: V' \rightarrow \{1, 2, 3\} \}$$

$$\Pi_{VC \cap 3COL}(G, K, S_{VC \cap 3COL}(G, K)) = \exists (V', c) \in S_{VC \cap 3COL}(G, K) : [|V'| \leq K \wedge \forall (u, v) \in E[V \setminus V'] \exists u, v \in V' \text{ such that } c(u) \neq c(v)] \wedge$$

$$\forall (V', c) \in S_{VC \cap 3COL}(G, K) : [\exists (u, v) \in E[V \setminus V'] \text{ such that } c(u) = c(v)]$$

1: $VC \cap 3COL \in NP$

Un certificato è l'insieme di tutte le coppie (V', c) e verificare se soddisfa il predicato richiede tempo più che polinomiale nell'istanza in quanto bisogna verificare che in tutte le coppie V' non c'è un vc di dimensione al più K e che, fra tutte queste coppie ce ne è almeno una che è una 3-colorazione per G . Perciò $VC \cap 3COL \notin NP$

3) Si trova nella stessa situazione di 4).