Informatica Teorica

Ede Boanini

26 ottobre 2025

Indice

1 Introduzione

1.1 Definizioni essenziali

Definizione 1.1 (Grafo)**.** Sia $G = (V, E)$ un grafo non orientato, dove:

- V è l'insieme dei nodi
- E è l'insieme degli archi

Definizione 1.2 (Coppia di nodi)**.** Siano u e v due nodi di un grafo $G = (V, E)$. La coppia $\{u, v\}$ rappresenta un arco che connette i nodi u e v.

Definizione 1.3 (Grado di un nodo)**.** Numero di archi che collegano un nodo v ad altri nodi.

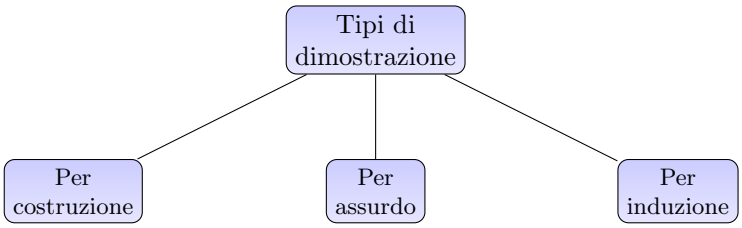
$$deg(v) = k$$

Definizione 1.4 (Grafo k-regolare)**.** Un grafo $G = (V, E)$ è k-regolare se ogni nodo ha grado k:

$$\forall v \in V, \quad deg(v) = k$$

1.2 Leggi di De Morgan

1.3 Tipi di dimostrazioni



Definizione 1.5 (Dimostrazione per Costruzione)**.** Il teorema afferma che esiste un particolare tipo di oggetto. Un modo per dimostrare un teorema di questo tipo è mostrare come costruire l'oggetto.

★ Idea: vuoi dimostrare che un oggetto esiste? Lo costruisci direttamente.

Esempio

Per ogni numero pari $n > 2$, $\exists$ un grafo 3-regolare con n nodi.

Dimostrazione. Sia n un numero pari maggiore di 2. Costruisco un grafo $G = (V, E)$ con n nodi come segue:
Dispongo i nodi in cerchio. Collego ogni nodo con il successivo $\{i, i + 1\}$ formando un ciclo. Dopodichè collego ogni nodo con il suo opposto $\{i, i + n/2\}$. In questo modo, ogni nodo ha 3 archi, quindi G è 3-regolare. □

Ho dimostrato il teorema costruendo un grafo che rispetta l'ipotesi (che il numero di nodi sia un numero pari maggiore di 2) arrivando poi alla tesi: ipotesi → costruzione → tesi.

Definizione 1.6 (Dimostrazione per Assurdo)**.** Assumo che il teorema sia falso e mostro che questa assunzione conduce a una proposizione che è logicamente impossibile, cioè che contraddice un fatto già dimostrato o una proprietà nota. Questa contraddizione implica che l'assunzione iniziale era falsa, quindi il teorema è vero.

★ Idea: supponi che il teorema sia falso (neghi la tesi). Se questa assunzione porta ad un'assurdità, allora il teorema deve essere vero.

Definizione 1.7 (Dimostrazione per Induzione)**.** Metodo usato per mostrare che tutti gli elementi di un insieme infinito possiedono una proprietà specifica. Questa dimostrazione consiste in due fasi:

- **Base:** dimostro che la proprietà $\mathcal{P}$ vale per il primo elemento dell'insieme. Verifico che $\mathcal{P}(1)$ (oppure $\mathcal{P}(0)$, dipende da dove parte l'insieme) è vera.
- **Passo induttivo:** suppongo che, per ogni $k \geq 1$, la proprietà $\mathcal{P}(k)$ sia vera (ipotesi induttiva). Ciò implica che anche $\mathcal{P}(k+1)$ è vera.

★ Idea: dimostri che una proprietà vale per infiniti casi.

Esempio

Per ogni $t \geq 0$, vale la seguente formula:

$$P_t = PM^t - Y\Big(\frac{M^t - 1}{M - 1}\Big)$$

Dimostrazione. **Base:** dimostra che la formula è vera per $t = 0$.

$$P_0 = PM^0 - Y\Big(\frac{M^0 - 1}{M - 1}\Big)$$

Sapendo che $P_0 = P$ e $M^0 = 1$, dunque ottengo:

$$P = P - Y\Big(\frac{0}{M - 1}\Big)$$
$$P = P - Y(0)$$
$$P = P - 0$$
$$P = P$$

Passo induttivo: $\forall k \geq 0$, assumo che la formula è vera per $t = k$ (ipotesi induttiva), ovvero assumo vera che:

$$P_k = PM^k - Y\Big(\frac{M^k - 1}{M - 1}\Big) \qquad \text{(ipotesi induttiva)}$$

questo per dimostrare che:

$$P_{k+1} = PM^{k+1} - Y\Big(\frac{M^{k+1} - 1}{M - 1}\Big) \qquad \text{(tesi)}$$

Se riesco a dimostrare che per $t = k + 1$ è vera, allora automaticamente è vera $\forall t \geq 0$. Iniziamo:
Per definizione so che,

$$P_{k+1} = P_k M - Y$$

usando l'ipotesi induttiva sostituisco

$$P_{k+1} = P_k M - Y$$

e ottengo

$$P_{k+1} = \Big[PM^k - Y\Big(\frac{M^k - 1}{M - 1}\Big)\Big]M - Y$$

sviluppando alla fine ottengo

$$= PM^{k+1} - Y\Big(\frac{M^{k+1} - 1}{M - 1}\Big)$$

Che è proprio quello che volevo dimostrare. □

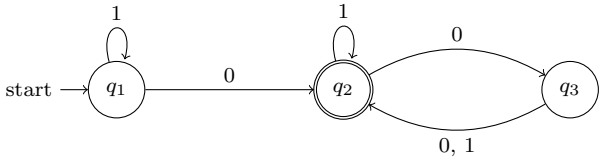
Ho dimostrato il teorema utilizzando l'ipotesi induttiva (che supponevo vera, per questo posso applicarla) nella definizione di P_{k+1}, poi ho sviluppato ed ottenuto la tesi.

IPOTESI	Condizioni che si assumono vere
TESI	Ciò che bisogna dimostrare

1.4 Macchina di Turing

Definizione 1.8 (Macchina deterministica)**.** Esiste una sola scelta possibile per ogni combinazione di stato e simbolo dell'alfabeto.

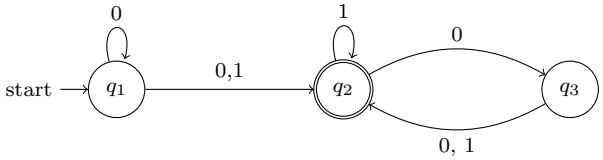
Esempio



Se sono nello stato q_1 e leggo il simbolo 0 ho solo una scelta: cambiare stato in q_2.

Definizione 1.9 (Macchina non deterministica)**.** Esistono più scelte possibili per ogni combinazione di stato e simbolo dell'alfabeto.

Esempio



Se sono nello stato q_1 e leggo il simbolo 0 ho più scelte:

- cambiare stato in q_2
- tornare nello stato q_1

Definizione 1.10 (Linguaggio)**.** Insieme di stringhe costruite a partire da un alfabeto e che rispettano certe regole.

Esempio

- Sia l'alfabeto $\Sigma = \{0, 1\}$
- Sia L il linguaggio

Linguaggio che contiene stringhe con un numero pari di 1:
$L = \{11, 011, 0011, 1100, 0110, 1111, 01111, \dots\}$

Definizione 1.11 (MdT modello standard)**.** Una Macchina di Turing standard è una 7-upla $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ dove Q, Σ, Γ sono insiemi finiti e:

- Q insieme degli stati
- Σ alfabeto di input (non contiene il simbolo * blank); $\Sigma \subseteq \Gamma$
- Γ alfabeto del nastro (tutti i simboli che può leggere e scrivere sul nastro, include anche il simbolo * blank)
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{S, D\}$ funzione di transizione
- $q_0 \in Q$ stato iniziale
- $q_{accept} \in Q$ è lo stato di accettazione
- $q_{reject} \in Q$ è lo stato di rifiuto dove $q_{accept} \neq q_{reject}$

Il modello standard è dunque una macchina deterministica.

Definizione 1.12 (Linguaggio decidibile/ricorsivo)**.** Un linguaggio L è **decidibile** se $\exists$ una MdT M tale che, per ogni stringa $w \in \Sigma^*$ in input:

- Se $w \in L \implies$ M si ferma (accettando w) nello stato q_{accept}; Quindi accetta la stringa.
- Se $w \notin L \implies$ M si ferma (rifiutando w) nello stato q_{reject}; Quindi rifiuta la stringa.

Notare che la macchina si ferma sempre. In questo caso si dice che la macchina "decide" L.

Definizione 1.13 (Linguaggio semidecidibile/ricorsivamente enumerabile)**.** Un linguaggio L è **semidecidibile** se $\exists$ una MdT M tale che, per ogni stringa $w \in \Sigma^*$ in input:

- Se $w \in L \implies$ M si ferma (accettando w) nello stato q_{accept}; Quindi accetta la stringa.
- Se $w \notin L \implies$ M va in loop, non fermandosi mai.

Notare che la macchina non si ferma sempre. In questo caso si dice che la macchina "riconosce" L.

Definizione 1.14 (Linguaggio indecidibile)**.** Un linguaggio L è **indecidibile** quando $\nexists$ una MdT M in grado di decidere L. Ma può esistere una MdT in grado di riconoscere L.
Nota: L è indecidibile e potrebbe essere semidecidibile, ma mai decidibile.

Definizione 1.15 (Linguaggio di una macchina)**.** Se A (linguaggio) è l'insieme di tutte le stringhe che la MdT M accetta oppure riconosce, dico che M accetta o riconosce A e lo indico come $L(M) = A$.
Se M non accetta nessuna stringa, lo indico come $L(M) = \emptyset$

2 Computabilità

Definizione 2.1 (Funzione totale)**.** Una funzione è totale se restituisce sempre un risultato per ogni input possibile. Quindi:

$$\forall x,\ f(x) \text{ è ben definita}$$

Definizione 2.2 (Funzione parziale)**.** Una funzione è parziale se non è definita per tutti gli input, cioè per alcuni valori di x non restituisce nessun risultato (va in loop o si blocca). Quindi:

$$\text{per alcuni } x,\ f(x) \text{ non è definita}$$

Definizione 2.3 (Funzione computabile)**.** Una funzione è computabile se $\exists$ una MdT M che calcola f.

Definizione 2.4 (Funzione computabile totale)**.** Una funzione $f : \Sigma^* \to \Sigma^*$ è computabile totale se esiste una MdT che calcola f e termina sempre per ogni input[1]. Ovvero:

$$\exists \text{ una MdT } M \text{ che termina sempre t.c. } \forall x \in \Sigma^* \text{ calcola } f(x)$$

Definizione 2.5 (Funzione computabile parziale)**.** Una funzione $f : \Sigma^* \to \Sigma^*$ è computabile parziale se $\exists$ una MdT M che calcola f ma può non terminare su alcuni input [2].

Funzione computabile $\neq$ MdT

Una funzione computabile è un concetto matematico; una MdT è uno strumento che può calcolarla. La funzione non è la MdT stessa.

Esempio: $f(n) = n^2$

1. f è una funzione matematica pura, definita su tutti i numeri naturali. In sé, non è un algoritmo, è solo una regola che dice: "dato un n come input , restituisci n^2" come output.
2. per calcolare f in modo concreto, posso costruire una MdT con il seguente comportamento.

   M su n:

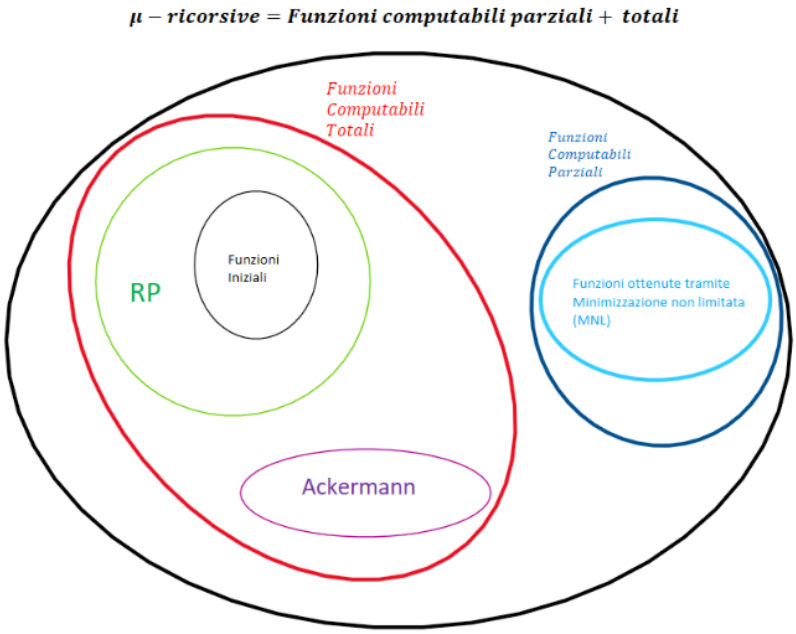
   (a) Calcola $n \cdot n$

   (b) Scrive risultato su nastro e termina

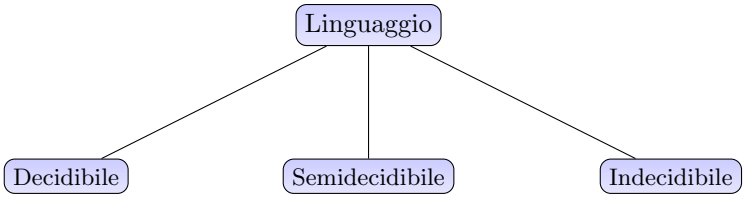
f è una funzione computabile totale perchè la MdT termina sempre fornendo una risposta per ogni input n.

[1] coincide con la definizione di decidibilità
[2] coincide con la definizione di semidecidibilità

2.1 Funzioni Computabili totali e parziali



3 Decidibilità



Come provare che un linguaggio è decidibile, semidecidibile o indecidibile?

- L è decidibile: dimostrazione per costruzione
- L è semidecidibile: dimostrazione per costruzione
- L è indecidibile: dimostrazione per assurdo + riduzione ad un problema che sappiamo essere indecidibile (es: Teorema dell'arresto) oppure Teorema di Rice

Definizione 3.1 (Codifica di una MdT)**.** La notazione $R(M)$ indica la codifica di una macchina. Spesso utilizzata nelle dimostrazioni.

3.1 Decidibilità

Un linguaggio L è **decidibile** se $\exists$ una MdT M tale che, per ogni stringa $w \in \Sigma^*$ in input:

- Se $w \in L \implies$ M si ferma (accettando w) nello stato q_{accept}; Quindi accetta la stringa.
- Se $w \notin L \implies$ M si ferma (rifiutando w) nello stato q_{reject}; Quindi rifiuta la stringa.

Notare che la macchina si ferma sempre. In questo caso si dice che la macchina "decide" L.

Definizione 3.2 (Enumeratore)**.** Un enumeratore è una MdT che genera tutte le stringhe del linguaggio (separandole con il simbolo "#"), una dopo l'altra, senza ricevere nessun input. Infatti, la macchina E inizia a lavorare su nastro vuoto (input vuoto).

Funzionamento:

1. L'enumeratore viene eseguito inizialmente su nastro vuoto (nessun input)
2. Genera la stringa scrivendola sul nastro
3. Quando ha terminato di scrivere la stringa, la invia al dispostivo di output (stampante)
4. Torna al passo 2

Definizione 3.3 (Funzione caratteristica)**.** Sia L un linguaggio su Σ^*. La funzione caratteristica di L, dato in input una stringa w, restituisce 1 se la stringa appartiene al linguaggio, 0 altrimenti:

$$\chi_L(w) = \begin{cases} 1 & \text{se } w \in L \\ 0 & \text{se } w \notin L \end{cases}$$

La funzione caratteristica è una MdT.

Teorema : 2.1.1

Se L è decidibile $\implies$ L è enumerabile.

Ragionamento. Per dimostrarlo, utilizzo la dimostrazione per costruzione.

Dimostrazione.

L è decidibile (ipotesi)

L è enumerabile (tesi)

Sia L il linguaggio su Σ^*. Per ipotesi, L è decidibile quindi $\exists$ una MdT M che decide L. Costruisco un MdT E che enumera L come segue.

E non ha nessun input ma $\forall w_i \in \Sigma^*$:

1. Esegue M su w_i:
   - Se M accetta ($w_i \in L$) allora E scrive w_i
   - Se M rifiuta ($w_i \notin L$) allora E non scrive w_i

Ho costruito un enumeratore per L. Pertanto L è enumerabile. □

Teorema : 2.1.2

L è decidibile $\iff$ L è enumerabile $\wedge$ $\overline{L}$ è enumerabile.

Ragionamento. $\overline{L}$ è enumerabile vuol dire che una MdT scrive tutte le stringhe $\notin L$

Teorema : 2.1.3

L è decidibile $\iff$ χ_L è una funzione computabile.

Ragionamento. χ_L funzione computabile vuol dire che $\exists$ una MdT, dato in input w, restituisce:

$$\chi_L(w) = \begin{cases} 1 & \text{se } w \in L, \\ 0 & \text{se } w \notin L. \end{cases}$$

Teorema : 2.1.4

Se L è decidibile $\implies$ L è semidecidibile.

Questa dimostrazione fa riferimento ad un automa a stati finiti deterministico (DFA). Lo stesso quesito per una MdT non è decidibile (vedi esercizio 2.2). Lo scopo è mostrare come funziona la dimostrazione per costruzione.

Esercizio 1.1

Sia il linguaggio $L_{DFA} = \{(R(M), w) \mid M \text{ accetta } w\}$. Dimostrare che L è decidibile.

Ragionamento. L_{DFA} è l'insieme delle codifiche di automi a stati finiti deterministici che accettano w. Ovvero, siano M_1,M_2,M_3 tre DFA:

- M_1 accetta w, allora $R(M_1) \in L_{DFA}$
- M_2 rifiuta w, allora $R(M_2) \notin L_{DFA}$
- M_3 accetta w, allora $R(M_3) \in L_{DFA}$

Quindi $L_{DFA} = \{R(M_1), R(M_3)\}$
**utilizzo dimostrazione per costruzione.

Dimostrazione.

$$L_{DFA} = \{(R(M), w) \mid M \text{ accetta } w\} \qquad \text{(ipotesi)}$$

$$L_{DFA} \text{è decidibile} \qquad \text{(tesi)}$$

Sapendo che per ipotesi $L_{DFA} = \{(R(M), w) \mid M \text{ accetta } w\}$, allora costruisco una MdT N che decide L_{DFA}. Dato in input $(R(M), w)$, dove $R(M)$ è la codifica di un DFA arbitrario e w una stringa:

1. Controlla che $R(M)$ sia una codifica valida e che w sia una stringa, altrimenti rifiuta.
2. Simula M su input w.
3. Se la simulazione termina:
   - in uno stato accettante $\implies$ N termina accettando $R(M)$;
     Quindi $R(M) \in L_{DFA}$
   - in uno stato di rifiuto $\implies$ N termina rifiutando $R(M)$;
     Quindi $R(M) \notin L_{DFA}$

Ho costruito una macchina N in grado di decidere L_{DFA}; Inoltre, poichè un DFA ha un numero di stati finiti e la stringa w è finita, la simulazione termina in uno stato finale (accettante o di rifiuto), garantendo che anche N si fermi sempre accettando o rifiutando l'input. Pertanto, si dimostra che L_{DFA} è decidibile. □

3.2 Semidecidibilità

Un linguaggio L è **semidecidibile** se $\exists$ una MdT M tale che, per ogni stringa $w \in \Sigma^*$ in input:

- Se $w \in L \implies$ M si ferma (accettando w) nello stato q_{accept}; Quindi accetta la stringa.
- Se $w \notin L \implies$ M va in loop, non fermandosi mai.

Notare che la macchina non si ferma sempre. In questo caso si dice che la macchina "riconosce" L.

Teorema : 2.2.1

L è enumerabile $\iff$ L è semidecidibile.

Ragionamento. Se io ho un linguaggio L e mi chiedono:
- Sia L enumerabile, è anche semidecidibile? (sempre vero)
- Sia L semidecidibile, è anche enumerabile? (sempre vero)

Dimostrazione. ($\implies$)

L è enumerabile (ipotesi)

L è semidecidibile (tesi)

Per ipotesi, $\exists$ una MdT E che enumera L. Costruisco un algoritmo di semidecisione M per L con il seguente funzionamento.

M su input w:
1. Esegue E e osserva le stringhe che esso stampa. Per ogni nuova stringa s_i stampata da E:
   M confronta w con s_i:
   - Se $w = s_i$, allora M accetta w.

Se w non appare mai tra le stringhe prodotte da E, M non si ferma (continuerà a confrontare ogni stringa stampata da E).
Ho costruito un algoritmo di semidecidibile per L, pertanto L è semidecidibile. $\square$

Nota per non confondersi: durante la costruzione di un algoritmo semidecidibile, specifica solo il comportamento di M nel caso di $w \in L$, ma non nel caso in cui $w \notin L$. Perchè essendo M semidecidibile, non importa specificarlo e potresti confonderti. Nel caso puoi menzionare che M non si ferma ma non andare nello specifico.

Dimostrazione. ($\Longleftarrow$)

$$L \text{ è semidecidibile} \qquad \text{(ipotesi)}$$

$$L \text{ è enumerabile} \qquad \text{(tesi)}$$

Per ipotesi, L è semidecidibile quindi $\exists$ una MdT M che riconosce (semidecide) L.
Costruisco un algoritmo di enumerazione E per L con il seguente funzionamento.
a) Versione con MdT deterministica
In questo caso si utilizza la tecnica di Dovetailing per Macchine di Turing.
Ricordiamoci che M è la MdT deterministica che semidecide L dove:
M su input w, se $w \in L \implies M$ accetta. (non specifico nel caso di $w \notin L$, perchè M può non terminare essendo semidecidibile).
Costruisco un **enumeratore E deterministico** come segue.

E non ha nessun input ma per ogni passo i:

- passo $i = 1$:
  1. E simula M su w_1 per 1 passo (cioè M effettua 1 transizione su w_1)
  2. Se M accetta, E stampa w_1
- passo $i = 2$:
  1. E simula M su w_1 per 2 passi (cioè M effettua 2 transizioni su w_1, ripartendo dallo stato iniziale)
  2. Se M accetta, E stampa w_1
  3. E simula M su w_2 per 2 passi (cioè M effettua 2 transizioni su w_2)
  4. Se M accetta, E stampa w_2
- passo $i = 3$:
  1. E simula M su w_1 per 3 passi (cioè M effettua 3 transizioni su w_1, ripartendo dallo stato iniziale)
  2. Se M accetta, E stampa w_1
  3. E simula M su w_2 per 3 passi (cioè M effettua 3 transizioni su w_2, ripartendo dallo stato iniziale)
  4. Se M accetta, E stampa w_2
  5. E simula M su w_3 per 3 passi (cioè M effettua 3 transizioni su w_3)
  6. Se M accetta, E stampa w_3
- passo $i = 4$:
  1. E simula M su w_1 per 4 passi (cioè M effettua 4 transizioni su w_1, ripartendo dallo stato iniziale)

2. Se M accetta, E stampa w_1

   CONTA CHE w_2 È GIÀ STATA ACCETTATA NEI PASSI PRECEDENTI QUINDI QUI NON C'È BISOGNO DI SCRIVERLA

3. E simula M su w_3 per 4 passi (cioè M effettua 4 transizioni su w_3, ripartendo dallo stato iniziale)
4. Se M accetta, E stampa w_3
5. E simula M su w_4 per 4 passi (cioè M effettua 4 transizioni su w_4)
6. Se M accetta, E stampa w_4

- ... (iterazioni successive)

È ovvio che con questo metodo E non rimane mai bloccata perchè M non si blocca su nessuna stringa w_i perchè al i-esimo passo, M esegue sulla stringa i transizioni[a] (che sono finite[b]).

Dopo aver compreso il meccanismo di Dovetailing per costruire un enumeratore tramite MdT deterministiche semidecidibili, posso riassumerlo così:
E non ha nessun input:
Ripeti quanto segue per $i = 1, 2, 3, \dots$ passi

1. Simula M su ogni input $w_1, w_2, ..., w_i$ per i passi
2. Se una qualsiasi simulazione accetta, E stampa la corrispondente stringa accettata.

Conclusione: Ho costruito un enumeratore E deterministico che stampa tutte e solo le stringhe di L. Pertanto L è enumerabile.

b) Versione con MdT non deterministica
In questo caso si utilizza la seguente tecnica: Costruisco un enumeratore non deterministico che simula una mdt M det? Ricordiamoci che M è la MdT deterministica che semidecide L dove:
M su input w, se $w \in L \implies M$ accetta. (non specifico nel caso di $w \notin L$, perchè M può non terminare essendo semidecidibile).
Costruisco un **enumeratore E non deterministico** come segue.

E non ha nessun input:

1. E indovina[c] una stringa w.
2. E simula M su w:
   - se M accetta w, allora E stampa la stringa

Conclusione: Ho costruito un enumeratore E non deterministico che stampa tutte e sole le stringhe di L. Pertanto L è enumerabile. □

[a] a seconda del passo di E
[b] quindi M si ferma per certo su w_i dopo aver fatto i transizioni
[c] Con "indovinare" si intende che E eslora diversi rami contemporaneamente; Ogni ramo i è indipendente e sceglie una stringa w_i:

$$Ramo_1 : w_1 \to M(w_1)$$
$$Ramo_2 : w_2 \to M(w_2)$$
$$Ramo_3 : w_3 \to M(w_3)$$
$$\vdots$$
$$Ramo_i : w_i \to M(w_i)$$

Quindi, è come se ci fossero tante M parallele che si eseguono, ciasucuna su input diverso w_i. Inoltre, se M va in loop su una w (ramo), gli altri rami non rimangono bloccati quindi segue che E non rimane bloccata. Questo garantisce che, se almeno un ramo termina con accettazione, allora l'enumeratore E stampa la stringa accettata da M.

Teorema : 2.2.2

L è semidecidibile $\wedge$ $\overline{L}$ è semidecidibile $\implies$ L decidibile.

Teorema : 2.2.3

L è semidecidibile $\wedge$ $\overline{L}$ non è semidecidibile $\implies$ L indecidibile.

3.3 Indecidibilità

Un linguaggio L è indecidibile quando $\nexists$ una MdT in grado di fermarsi sempre accettando o rifiutando l'input. Ciò vuol dire che:

- L è indecidibile e anche semidecidibile
- L è indecidibile ma non semidecidibile

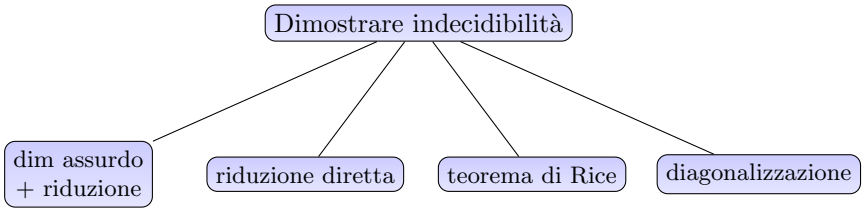
Pertanto, quando viene richiesto di dimostrare l'indecidibilità di un linguaggio:

- Se sospetto che L possa essere indecidibile + semidecidibile, allora:

  1. Applico la dimostrazione per costruzione;
     cioè, costruisco una MdT che riconosce[3] il linguaggio.
  2. Effettuo la riduzione ad un linguaggio noto indecidibile.

  Nel primo punto dimostro la semidecidibilità di L e nel secondo la indecidibilità.

- Se sospetto che il linguaggio è indecidibile ma non semidecidibile, allora posso considerare una di queste tecniche:



[3]Turing-reconizable: MdT si ferma accettando le stringhe che appartengono al linguaggio e va in loop per quelle che non appartengono.

- dimostrazione per assurdo + effettuo la riduzione ad un linguaggio noto indecidibile
- riduzione diretta
- uso il Teorema di Rice
- applico la diagonalizzazione

3.4 Proprietà di Chiusura dei linguaggi

PROPRIETÀ DI CHIUSURA	
Chiusura sotto unione	Siano L_1, L_2 due linguaggi decidibili $\Rightarrow L_1 \cup L_2$ è decidibile. Siano L_1, L_2 due linguaggi semidecidibili $\Rightarrow L_1 \cup L_2$ è semidecidibile. Siano L_1 decidibile e L_2 semidecidibile $\Rightarrow L_1 \cup L_2$ è semidecidibile. Siano L_1 decidibile e L_2 indecidibile $\Rightarrow L_1 \cup L_2$ è _____ (dipende). Siano L_1 semidecibile e L_2 indecidibile $\Rightarrow L_1 \cup L_2$ è _____ (dipende).
Chiusura sotto intersezione	Siano L_1, L_2 due linguaggi decidibili $\Rightarrow L_1 \cap L_2$ è decidibile. Siano L_1, L_2 due linguaggi semidecidibili $\Rightarrow L_1 \cap L_2$ è semidecidibile. Siano L_1 decidibile e L_2 semidecidibile $\Rightarrow L_1 \cap L_2$ è semidecidibile. Siano L_1 decidibile e L_2 indecidibile $\Rightarrow L_1 \cap L_2$ è _____ (dipende). Siano L_1 semidecibile e L_2 indecidibile $\Rightarrow L_1 \cap L_2$ è _____ (dipende).
Chiusura sotto concatenazione	Siano L_1, L_2 due linguaggi decidibili $\Rightarrow L_1 \circ L_2$ è decidibile. Siano L_1, L_2 due linguaggi semidecidibili $\Rightarrow L_1 \circ L_2$ è semidecidibile. Siano L_1 decidibile e L_2 semidecidibile $\Rightarrow L_1 \circ L_2$è semidecidibile. Siano L_1 decidibile e L_2 indecidibile $\Rightarrow L_1 \circ L_2$ è _____ (dipende). Siano L_1 semidecidibile e L_2 indecidibile $\Rightarrow L_1 \circ L_2$ è _____ (dipende).
Chiusura sotto stella di Kleene	Sia L_1 un linguaggio decidibile $\Rightarrow L_1^*$ è decidibile. Sia L_1 un linguaggio semidecidibile $\Rightarrow L_1^*$ è semidecidibile. Sia L_1 un linguaggio indecidibile $\Rightarrow L_1^*$ è _____ (dipende).
Chiusura sotto il complemento	Sia L un linguaggio decidibile $\Rightarrow L^C$ è decidibile. Sia L un linguaggio semidecidibile e L^C è semidecidibile $\Rightarrow L$ è decidibile. Sia L un linguaggio semidecidibile + indecidibile $\Rightarrow L^C$ è non-semidecidibile. Sia L un linguaggio indecidibile $\Rightarrow L^C$ è _____ (dipende).

3.5 Esercizi

Teorema

Sia il linguaggio $L = \{(R(M), w) \mid M \text{ accetta } w\}$. Dimostrare che L è semidecidibile ma anche indecidibile.

Esercizio 2.1

Sia il linguaggio $L_1 = \{(R(M), w) \mid M \text{ accetta } w\}$. Dimostrare che L è semidecidibile.

Ragionamento. L_1 è l'insieme delle codifiche di MdT che accettano w. Ovvero, siano M_1,M_2,M_3 tre MdT:

- M_1 accetta w, allora $R(M_1) \in L_1$
- M_2 rifiuta w, allora $R(M_2) \notin L_1$
- M_3 accetta w, allora $R(M_3) \in L_1$

Quindi $L_1 = \{R(M_1), R(M_3)\}$
Dato che devo dimostrare la semidecidibilità di L_1, costruisco una MdT che riconosce il linguaggio.

Dimostrazione.

$$L_1 = \{(R(M), w) \mid M \text{ accetta } w\} \qquad \text{(ipotesi)}$$

$$L_1 \text{ è semidecidibile} \qquad \text{(tesi)}$$

Costruisco una MdT N che riconosce L_1. Dato in input $(R(M), w)$, dove $R(M)$ è la codifica di una MdT arbitraria e w una stringa, la MdT N si comporta come segue:

1. Controlla che $R(M)$ sia una codifica valida e che w sia una stringa, altrimenti rifiuta.
2. Simula M su input w.
3. Se la simulazione termina:
   - in uno stato accettante $\implies$ N termina accettando $R(M)$;
     Quindi $R(M) \in L_1$
   - in uno stato di rifiuto $\implies$ N termina rifiutando $R(M)$;
     Quindi $R(M) \notin L_1$

Notare che la macchina N va in loop sull'input $(R(M), w)$ se M va in loop su w.
Un decisore deve sempre fermarsi (in ogni caso), ma qui, se M va in loop su w, N non si fermerà mai. Questo rende N un riconoscitore e non un decisore.
Ho dimostrato che L_1 è semidecidibile costruendo la MdT N che riconosce tale linguaggio.
Pertanto, si dimostra che L_1 è semidecidibile. □

Esercizio 2.2

Sia il linguaggio $L_1 = \{(R(M), w) \mid M \text{ accetta } w\}$. Dimostrare che L è indecidibile.

Ragionamento. Per dimostrarlo, applico:

- Dim per assurdo (suppongo per assurdo (nego la tesi) per poi ottenere una contraddizione, che rende falsa l'assunzione fatta)

Dimostrazione.

$$L_1 = \{(R(M), w) \mid M \text{ accetta } w\} \qquad \text{(ipotesi)}$$

$$L_1 \text{ è indecidibile} \qquad \text{(tesi)}$$

Per applicare la dimostrazione per assurdo nego la tesi, quindi suppongo per assurdo che L_1 sia decidibile. Sapendo che per ipotesi (la mia per assurdo, non quella del teorema) L_1 è decidibile, allora $\exists$ una MdT N che decide L_1 con il seguente funzionamento:

$$N(R(M), w) = \begin{cases} \text{accept} & \text{se } M \text{ accetta } w \\ \text{reject} & \text{se } M \text{ non accetta } w \end{cases}$$

Con "non accetta" si intende che M potrebbe rifiutare w oppure andare in loop.
Dato in input $(R(M), w)$ alla MdT N, dove $R(M)$ è la codifica di una MdT arbitraria e w una stringa:

1. N controlla che $R(M)$ sia una codifica valida e che w sia una stringa, altrimenti rifiuta.
2. Simula M su input w.
3. Se la simulazione termina:
   - in uno stato accettante $\implies$ N termina accettando $R(M)$;
     Quindi $R(M) \in L_1$
   - in uno stato di rifiuto/va in loop $\implies$ N termina rifiutando $R(M)$;
     Quindi $R(M) \notin L_1$

Adesso costruisco una nuova MdT D con N come subroutine. D chiama N per determinare cosa fa M quando l'input per M è la sua stessa codifica (e non la stringa w). Il comportamento di D è l'opposto di N, ovvero:

$$D(R(M)) = \begin{cases} \text{accept} & \text{se } M \text{ non accetta } R(M) \\ \text{reject} & \text{se } M \text{ accetta } R(M) \end{cases}$$

Funzionamento di D:

1. Esegue N su input $(R(M), R(M))$ dove $R(M)$ è la codifica della MdT M:
   - Se N si ferma accettando $\implies$ D rifiuta.
     (ricorda che se N accetta allora vuol dire che M ha accettato $R(M)$)
   - Se N si ferma rifiutando $\implies$ D accetta.
     (ricorda che se N rifiuta allora vuol dire che M non ha accettato $R(M)$)

Cosa succederebbe se fornissimo alla MdT D la propria codifica come input? Otterrei:

$$D(R(D)) = \begin{cases} \text{accept} & \text{se } D \text{ non accetta } R(D) \\ \text{reject} & \text{se } D \text{ accetta } R(D) \end{cases}$$

che è una contraddizione.
Conclusione: Poiché abbiamo ottenuto una contraddizione (D rifiuta $R(D)$ quando D accetta $R(D)$), l'ipotesi che L_1 sia decidibile è falsa. Pertanto, L_1 è indecidibile. □

Teorema

Sia il linguaggio $L = \{(R(M), w) \mid M \text{ accetta } w\}$. Dimostrare che $\overline{L}$ non è semidecidibile.

Esercizio 2.3

Sia il linguaggio $L_1 = \{(R(M), w) \mid M \text{ accetta } w\}$. Dimostrare che $\overline{L_1}$ non è semidecidibile.

Ragionamento. Devo dimostrare che $\overline{L_1} = \{(R(M), w) \mid M \text{ non accetta } w\}$ non è semidecidibile, ovvero che $\nexists$ una MdT in grado di riconoscere $\overline{L_1}$. Procedo per assurdo.

Dimostrazione.

$$L_1 = \{(R(M), w) \mid M \text{ accetta } w\} \qquad \text{(ipotesi)}$$

$$\overline{L_1} \text{ non è semidecidibile} \qquad \text{(tesi)}$$

Sappiamo che L_1 è un linguaggio semidecidibile (esercizio 2.1).
Suppongo per assurdo che anche $\overline{L_1}$ sia semidecidibile (nego la tesi, ipotesi per assurdo). Dato che $L_1, \overline{L_1}$ sono entrambi semidecidibili $\implies$ per definizione, L_1 è decidibile. Ma questo è assurdo perchè abbiamo dimostrato che L_1 è indecidibile (esercizio 2.2). Ciò porta ad una contraddizione e l'ipotesi che $\overline{L_1}$ sia semidecidibile è falsa. Pertanto, $\overline{L_1}$ è non semidecidibile.

□

3.6 Famosi problemi indecidibili

3.6.1 Halting Problem

Definizione 3.4. (Problema dell'arresto) Esiste una MdT H che preso in input (M, w), termina dicendo:

- Sì, se M termina su w
- No, se M va in loop su w

Tale macchina esiste? No.

Definizione 3.5. (Linguaggio Halting Problem) Sia $\mathcal{L}_{\text{Halt}}$ il linguaggio del problema dell'arresto definito come segue:

$$\mathcal{L}_{\text{Halt}} = \{(R(M), w) \mid M \text{ termina su } w\}$$

Notare che $\mathcal{L}_{\text{Halt}}$ è semidecidibile, ovvero $\exists$ una MdT N che dato in input $(R(M), w)$:

- Se M termina su w, allora N accetta
  (Se M termina su $w \implies (R(M), w) \in \mathcal{L}_{\text{Halt}}$)[4].
- Se M non termina su w, allora N va in loop

*Attenzione: "M termina su w" e non dice "M accetta w", che sono due cose differenti. Se M termina su w, allora N accetta; ovvero l'importante è che M termini su w, che sia in uno stato accettante o di rifiuto. (Quindi non ci interessa se M accetta o rifiuta w, quello che ci interessa è che M si fermi su w.)

Pertanto, $\exists$ una MdT che riconosce (ma non decide) $\mathcal{L}_{\text{Halt}}$.

Osservazione

Il problema dell'arresto (o, equivalentemente, il linguaggio $\mathcal{L}_{\text{Halt}}$) è **indecidibile**, cioè non esiste una MdT che termina sempre dando la risposta corretta. Tuttavia, $\mathcal{L}_{\text{Halt}}$ è **semidecidibile**: esiste infatti una MdT universale che, dato in input $(R(M), w)$, termina accettando se M termina su w, mentre può non terminare se M non termina su w.

Il problema dell'arresto è semidecidibile + indecidibile, oppure, equivalentemente scrivo:

$\mathcal{L}_{\text{Halt}}$ è semidecidibile ma non decidibile

[4]Ho scritto "M termina su w" e non "$w \in L(M)$" poichè si sta parlando di terminare su w e non accettare w. La dicitura classica $w \in L(M)$ si scrive solo quando M accetta w.

Teorema : Halting Problem (2.4.1)

Il problema dell'arresto è indecidibile.
$\mathcal{L}_{\text{Halt}} = \{(R(M), w) \mid M \text{ termina su } w\}$ è indecidibile.

Dimostrazione.

$$\mathcal{L}_{\text{Halt}} = \{(R(M), w) \mid M \text{ termina su } w\} \qquad \text{(ipotesi)}$$

$$\mathcal{L}_{\text{Halt}} \text{ è indecidibile} \qquad \text{(tesi)}$$

Suppongo per assurdo che il problema dell'arresto sia decidibile (ipotesi per assurdo, nego la tesi). Quindi, per ipotesi, $\exists$ una MdT H che risolve il problema dell'arresto. H su input $(R(M), w)$ dove $R(M)$ è la codifica di una MdT arbitraria e w una stringa:

- Se M termina su w, allora H accetta
- Se M non termina su w, allora H rifiuta

Modifico la MdT H per costruire H' con il seguente comportamento.

$$H'(R(M), w) = \begin{cases} \text{loop} & \text{se } M \text{ termina su } w \\ \text{reject} & \text{se } M \text{ non termina su } w \end{cases}$$

H' su input $(R(M), w)$:

- Se M termina su w, allora H' va in loop
- Se M non termina su w, allora H' rifiuta

Adesso costruisco un'altra MdT D (che prende in input solo la codifica di una macchina)[a] combinando H' con una procedura.

$$D(R(M)) = \begin{cases} \text{loop} & \text{se } M \text{ termina su } R(M) \\ \text{reject} & \text{se } M \text{ non termina su } R(M) \end{cases}$$

La macchina D su input $(R(M))$:

1. Controlla che $R(M)$ sia un codifica valida, altrimenti rifiuta
2. Esegue varie transizioni che, a partire dall'input, produce la coppia $(R(M), R(M))$
   (legge la codifica della macchina in input e la copia come secondo argomento da fornire a H')
3. Esegue H' sull'input $(R(M), R(M))$:
   - Se H' va in loop, allora anche D va in loop
     (ricorda che se H' va in loop vuol dire che M ha terminato su $R(M)$)
   - Se H' rifiuta, allora D rifiuta
     (ricorda che se H' rifiuta vuol dire che M non ha mai terminato su $R(M)$)

E adesso cosa succederebbe se fornissi alla MdT D la propria codifica come input? Otterrei:

$$D(R(D)) = \begin{cases} \text{loop} & \text{se } D \text{ termina su } R(D) \\ \text{reject} & \text{se } D \text{ non termina su } R(D) \end{cases}$$

una contraddizione.
Conclusione: Poichè ho ottenuto una contraddizione (D va in loop su $R(D) \iff D$ termina su $R(D)$), l'ipotesi che il problema dell'arresto sia decidibile è falsa. Pertanto, il problema dell'arresto è indecidibile (oppure scrivo: Pertanto, $\mathcal{L}_{\text{Halt}}$ è indecidibile). □

[a] invece dell'input $(R(M), w)$, prende in input $(R(M))$

Teorema

$\mathcal{L}_{\text{Halt}}$ è semidecidibile. $\overline{\mathcal{L}_{\text{Halt}}}$ non è semidecidibile.

Dimostrazione. Sappiamo per definizione, che $\mathcal{L}_{\text{Halt}}$ è semidecidibile.
Suppongo per assurdo che $\overline{\mathcal{L}_{\text{Halt}}}$ sia anch'esso semidecidibile. Dato che $\overline{\mathcal{L}_{\text{Halt}}}$,$\mathcal{L}_{\text{Halt}}$ sono entrambi semidecidibili $\implies$ per definizione, $\mathcal{L}_{\text{Halt}}$ è decidibile. Ma questo è assurdo perchè abbiamo dimostrato che $\mathcal{L}_{\text{Halt}}$ è indecidibile (teorema 2.4.1). Ciò porta ad una contraddizione e l'ipotesi che $\overline{\mathcal{L}_{\text{Halt}}}$ sia semidecidibile è falsa.
Pertanto, $\overline{\mathcal{L}_{\text{Halt}}}$ è non semidecidibile. □

3.6.2 Problema del nastro vuoto

Definizione 3.6 (Problema del nastro vuoto)**.** Esiste una MdT E che preso in input $R(M)$, termina dicendo:

- Sì, se M termina su nastro vuoto
- No, se M non termina su nastro vuoto

Tale macchina esiste? No.
*Attenzione: "termina su nastro vuoto" significa "termina quando parte su nastro vuoto".

Definizione 3.7 (Linguaggio Blank-Tape Halting Problem)**.** Sia $\mathcal{L}_{\text{Blank-Tape}}$ il linguaggio del problema del nastro vuoto definito come segue:

$$\mathcal{L}_{\text{Blank-Tape}} = \{R(M) \mid M \text{ termina su nastro vuoto}\}$$

Notare che $\mathcal{L}_{\text{Blank-Tape}}$ è semidecidibile.

Quindi, il problema del nastro vuoto è semidecidibile + indecidibile, oppure, equivalentemente scrivo:

$\mathcal{L}_{\text{Blank-Tape}}$ è semidecidibile ma non decidibile

Teorema : Blank-Tape Halting Problem (versione A)

Il problema del nastro vuoto è indecidibile.
$\mathcal{L}_{\text{Blank-Tape}} = \{R(M) \mid M \text{ termina su nastro vuoto}\}$ è indecidibile.

Ragionamento. Per dimostrarlo, uso la dim per assurdo e poi effettuo una riduzione da un problema noto indecidibile (es: problema dell'arresto) al mio (problema del nastro vuoto).
Dato che sto ragionando per assurdo, non posso usare questa: se $A \leq_m B \ \wedge \ A$ è indecidibile $\implies$ B è indecidibile (teorema 3.2); ma devo usare questa: se $A \leq_m B \ \wedge \ B$ è decidibile $\implies$ A è decidibile (teorema 3.1);
Quindi costruisco una riduzione del tipo: $\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_{\text{Blank-Tape}}$.

Dimostrazione.

$$\mathcal{L}_{\text{Blank-Tape}} = \{R(M) \mid M \text{ termina su nastro vuoto}\} \qquad \text{(ipotesi)}$$

$$\mathcal{L}_{\text{Blank-Tape}} \text{ è indecidibile} \qquad \text{(tesi)}$$

Siano $\mathcal{L}_{\text{Halt}}$, $\mathcal{L}_{\text{Blank-Tape}}$ due linguaggi su Σ_H^*, Σ_B^* rispettivamente, dove:

- $\mathcal{L}_{\text{Halt}}$ è il linguaggio del problema dell'arresto
- $\mathcal{L}_{\text{Blank-Tape}}$ è il linguaggio del problema del nastro vuoto

1. Ipotesi per assurdo

Suppongo per assurdo che $\mathcal{L}_{\text{Blank-Tape}}$ sia decidibile (ipotesi per assurdo). Quindi per ipotesi $\exists$ una MdT E che decide $\mathcal{L}_{\text{Blank-Tape}}$. Ovvero:

$$E(R(M)) = \begin{cases} \text{accept} & \text{se } M \text{ termina su nastro vuoto} \\ \text{reject} & \text{se } M \text{ non termina su nastro vuoto} \end{cases}$$

2. Costruisco funzione di riduzione

Costruisco una funzione di riduzione $f : \Sigma_H^* \to \Sigma_B^*$ t.c. $\forall w \in \Sigma_H^*$:

$$w \in \mathcal{L}_{\text{Halt}} \iff f(w) \in \mathcal{L}_{\text{Blank-Tape}}$$

Quindi, costrusco la funzione di riduzione (che sarebbe la MdT R):

- Se w non è della forma $(R(M), w)$, pongo $f(w) = 1$
- Se $w = (R(M), w)$, allora pongo $f(w) = R(M')$

Ovvero, il funzionamento della MdT R è costruire una nuova macchina M', praticamente il seguente:

1. R riceve come input $w = (R(M), w)$ da N
2. R costruisce la nuova macchina M' che ha questo comportamento:
   (a) M' viene avviata su nastro vuoto (condizione del Blank-Tape Halting Problem)
   (b) M' scrive w sul nastro
   (c) M' riporta la testina all'inizio del nastro
   (d) M' esegue M su w
3. R genera l'output $R(M')$ (che è la nuova macchina costruita nel passo 2)

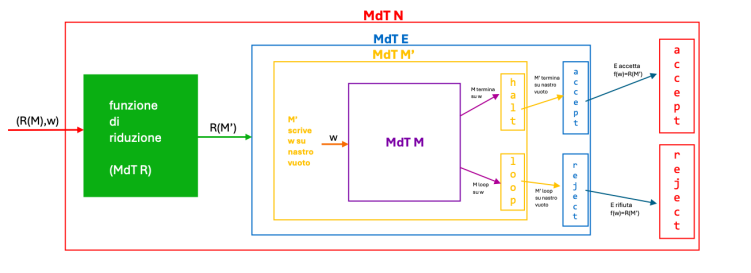
3. Costruisco MdT che decide il problema dell'arresto

Adesso, costrusco una MdT N che decide $\mathcal{L}_{\text{Halt}}$:

1. N su input $w \in \Sigma_H^*$ calcola $f(w) \in \Sigma_B^*$
2. N esegue E su $f(w)$:
   - Se E accetta, allora N accetta
     (ricorda che se E accetta, vuol dire che M' ha terminato su nastro vuoto (se M' termina su nastro vuoto, vuol dire che M ha terminato su w))
     ovvero: Se $R(M') \in \mathcal{L}_{\text{Blank-Tape}}$, allora $(R(M), w) \in \mathcal{L}_{\text{Halt}}$
     ma è più corretto scrivere:
     $w \in \mathcal{L}_{\text{Halt}} \iff f(w) \in \mathcal{L}_{\text{Blank-Tape}}$
   - Se E rifiuta, allora N rifiuta
     (ricorda che se E rifiuta, vuol dire che M' non ha terminato su nastro vuoto (se M' non termina su nastro vuoto (loop), vuol dire che M non ha terminato su w (loop)))
     ovvero: Se $R(M') \notin \mathcal{L}_{\text{Blank-Tape}}$, allora $(R(M), w) \notin \mathcal{L}_{\text{Halt}}$
     ma è più corretto scrivere:
     $w \notin \mathcal{L}_{\text{Halt}} \iff f(w) \notin \mathcal{L}_{\text{Blank-Tape}}$

Quindi N è un algoritmo di decisione per il problema dell'arresto che applica la funzione di riduzione per ogni input e sfrutta la MdT E che decide $\mathcal{L}_{\text{Blank-Tape}}$ per decidere $\mathcal{L}_{\text{Halt}}$. Perchè alla fine N si ferma accettando w se e solo se E si ferma accettando $f(w)$ oppure N si ferma rifiutando w se e solo se E si ferma rifiutando $f(w)$.
La costruzione e il funzionamento della MdT N rende il problema dell'arresto decidibile. Ma questo è assurdo perchè sappiamo che il problema dell'arresto è indecidibile, quindi questo porta ad una contraddizione. Poichè ho ottenuto una contraddizione, l'ipotesi che il problema del nastro vuoto sia decidibile è falsa. Pertanto, il problema del nastro vuoto è indecidibile. □

Disegno:



Teorema : Blank-Tape Halting Problem (versione B)

Il problema del nastro vuoto è indecidibile.
$\mathcal{L}_{\text{Blank-Tape}} = \{R(M) \mid M \text{ termina su nastro vuoto}\}$ è indecidibile.

Ragionamento. Per dimostrarlo, non uso la dim per assurdo ma effettuo **direttamente** una riduzione dal problema dell'arresto al mio. Ovvero, costruisco una riduzione del tipo: $\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_{\text{Blank-Tape}}$.
Uso solo la riduzione perchè "sfrutto":

1. La definizione di funzione di riduzione
   ($L_1 \leq_m L_2$ se $\exists$ una funzione di riduzione $f : \Sigma_1^* \to \Sigma_2^*$ t.c. $\forall w \in \Sigma_1^*$:
   $w \in L_1 \iff f(w) \in L_2$)
2. Il teorema $A \leq_m B \ \wedge \ A$ è indecidibile $\implies B$ è indecidibile (teorema 3.2).

Dimostrazione.

$$\mathcal{L}_{\text{Blank-Tape}} = \{R(M) \mid M \text{ termina su nastro vuoto}\} \qquad \text{(ipotesi)}$$

$$\mathcal{L}_{\text{Blank-Tape}} \text{ è indecidibile} \qquad \text{(tesi)}$$

Siano $\mathcal{L}_{\text{Halt}}$, $\mathcal{L}_{\text{Blank-Tape}}$ due linguaggi su Σ_H^*, Σ_B^* rispettivamente, dove:

- $\mathcal{L}_{\text{Halt}}$ è il linguaggio del problema dell'arresto
- $\mathcal{L}_{\text{Blank-Tape}}$ è il linguaggio del problema del nastro vuoto

Sia $\mathcal{L}_{\text{Halt}} = \{(R(M), w) \mid M \text{ termina su } w\}$. È noto che $\mathcal{L}_{\text{Halt}}$ è indecidibile.
1. Riduzione
Definisco una funzione di riduzione[a] $f : \Sigma_H^* \to \Sigma_B^*$ come segue.
Dato in input $(R(M), w)$[b] la funzione f genera come output $R(M')$[c], dove M' è la nuova MdT costruita dalla funzione di riduzione f che ha questo comportamento:

1. M' inizia la computazione su nastro vuoto
2. M' scrive w sul nastro
3. M' riporta la testina all'inizio del nastro
4. M' esegue M su w

La funzione di riduzione f trasforma ogni istanza del problema dell'arresto $(R(M), w)$ in un'istanza $R(M')$ del problema del nastro vuoto, dove M' inizia la computazione

su nastro vuoto, scrive w sul nastro ed esegue M su w. Per definizione di funzione di riduzione, vale:

$$(R(M), w) \in \mathcal{L}_{\text{Halt}} \iff f((R(M), w)) \in \mathcal{L}_{\text{Blank-Tape}}.^{d}$$

Poiché $\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_{\text{Blank-Tape}}$ e $\mathcal{L}_{\text{Halt}}$ è indecidibile, segue che $\mathcal{L}_{\text{Blank-Tape}}$ è indecidibile. □

---

[a] ricorda la funzione di riduzione trasforma un'istanza del problema dell'arresto H in un'istanza del problema del nastro vuoto B

[b] istanza del problema dell'arresto

[c] istanza del problema del nastro vuoto

[d] dove $f((R(M), w)) = R(M')$; precisando che $w = (R(M), w)$ e $f(w) = R(M')$ dove w è l'input dato alla funzione di riduzione e $f(w)$ è l'ouput generato dalla stessa funzione di riduzione, ovvero il valore ottenuto applicando f a w.

3.6.3 Problema del linguaggio vuoto

Teorema : Emptiness Problem (versione lunga)

Il problema del linguaggio vuoto è indecidibile.
$\mathcal{L}_{\text{Emptiness}} = \{R(M) \mid L(M) = \emptyset\}$ è indecidibile.

Ragionamento. Per dimostrarlo, uso la dim per assurdo + costruisco una riduzione dal problema dell'arresto al mio, ovvero $\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_{\text{Emptiness}}$.
Uso quindi la MdT E che decide $\mathcal{L}_{\text{Emptiness}}$ per decidere $\mathcal{L}_{\text{Halt}}$. Uso solo la riduzione perchè "sfrutto":

1. La definizione di funzione di riduzione
   ($L_1 \leq_m L_2$ se $\exists$ una funzione di riduzione $f : \Sigma_1^* \to \Sigma_2^*$ t.c. $\forall w \in \Sigma_1^*$:
   $w \in L_1 \iff f(w) \in L_2$)
2. Il teorema $A \leq_m B \ \wedge \ B$ è decidibile $\implies A$ è decidibile (teorema 3.1).

Dimostrazione.

$$\mathcal{L}_{\text{Emptiness}} = \{R(M) \mid L(M) = \emptyset\} \qquad \text{(ipotesi)}$$

$$\mathcal{L}_{\text{Emptiness}} \text{ è indecidibile} \qquad \text{(tesi)}$$

Siano $\mathcal{L}_{\text{Halt}}$, $\mathcal{L}_{\text{Emptiness}}$ due linguaggi su Σ_H^*, Σ_E^* rispettivamente, dove:

- $\mathcal{L}_{\text{Halt}}$ è il linguaggio del problema dell'arresto
- $\mathcal{L}_{\text{Emptiness}}$ è il linguaggio dell'emptiness problem

1. Ipotesi per assurdo

Suppongo per assurdo che $\mathcal{L}_{\text{Emptiness}}$ sia decidibile (ipotesi per assurdo). Allora $\exists$ una MdT E che decide $\mathcal{L}_{\text{Emptiness}}$. Ovvero:

$$E(R(M)) = \begin{cases} \text{accept} & \text{se } M \text{ non accetta nessuna stringa} \\ \text{reject} & \text{se } M \text{ accetta almeno una stringa} \end{cases}$$

2. Riduzione

Definisco una funzione di riduzione[a] $f : \Sigma_H^* \to \Sigma_E^*$ come segue.
Dato in input $(R(M), w)$[b] la funzione f genera come output $R(M')$[c], dove M' è la nuova MdT costruita dalla funzione di riduzione f che ha il seguente comportamento ($R(M)$ è la codifica di una MdT arbitraria e w una stringa).

M' su una stringa di input x generica:

- se $x \neq w$, rifiuta
- se $x = w$, esegue M su w dove:
  - se M termina su w, allora M' accetta w (perchè $x = w$)
  - se M va in loop su w, allora M' va in loop

In sostanza, osservandolo dal punto di vista del linguaggio accettato da M':

$$L(M') = \begin{cases} \{w\} & \text{se } M \text{ termina su } w \\ \emptyset & \text{se } M \text{ non termina (loop) su } w \end{cases}$$

3. Costruisco MdT che decide il problema dell'arresto

Adesso, costrusco una MdT N che decide $\mathcal{L}_{\text{Halt}}$:

1. N su input $w \in \Sigma_H^*$ calcola $f(w) \in \Sigma_E^*$ (riduzione)
2. N esegue E su $f(w)$:
   - Se E accetta, allora N rifiuta
     (se E accetta, vuol dire che M' ha linguaggio vuoto, cioè M' loop w, quindi $L(M') = \emptyset$ (se M' loop su w, vuol dire che M ha loopato w))
     ovvero: Se $R(M') \in \mathcal{L}_{\text{Emptiness}}$, allora $(R(M), w) \notin \mathcal{L}_{\text{Halt}}$
     ma è più corretto scrivere:
     $w \notin \mathcal{L}_{\text{Halt}} \iff f(w) \in \mathcal{L}_{\text{Emptiness}}$
   - Se E rifiuta, allora N accetta
     (se E rifiuta, vuol dire che M' non ha linguaggio vuoto, cioè M' ha accettato w, quindi $L(M') = \{w\}$ (se M' accetta w, vuol dire che M ha terminato su w))
     ovvero: Se $R(M') \notin \mathcal{L}_{\text{Emptiness}}$, allora $(R(M), w) \in \mathcal{L}_{\text{Halt}}$
     ma è più corretto scrivere:
     $w \in \mathcal{L}_{\text{Halt}} \iff f(w) \notin \mathcal{L}_{\text{Emptiness}}$

*Precisando che $w = (R(M), w)$ e $f(w) = R(M')$ dove w è l'input dato alla funzione di riduzione e $f(w)$ è l'ouput generato dalla stessa funzione di riduzione, ovvero il valore ottenuto applicando f a w.

Poiché $\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_{\text{Emptiness}}$ e $\mathcal{L}_{\text{Emptiness}}$ è decidibile (ipotesi per assurdo), segue che $\mathcal{L}_{\text{Halt}}$ è decidibile. Ma questo è assurdo perché è ben noto che $\mathcal{L}_{\text{Halt}}$ è indecidibile (contraddizione). Poichè ho ottenuto una contraddizione, l'ipotesi che $\mathcal{L}_{\text{Emptiness}}$ sia decidibile è falsa. Pertanto, l'emptiness problem è indecidibile. □

[a] ricorda la funzione di riduzione trasforma un'istanza del problema dell'arresto H in un'istanza dell'emptiness problem E

[b] istanza del problema dell'arresto

[c] istanza del problema dell'emptiness problem

Teorema : Emptiness Problem (versione corta)

Il problema del linguaggio vuoto è indecidibile.
$\mathcal{L}_{\text{Emptiness}} = \{R(M) \mid L(M) = \emptyset\}$ è indecidibile.

Ragionamento. Dim per assurdo + riduzione:

1. Nego la tesi (suppongo per assurdo che il nostro problema sia decidibile)
2. Costruisco una funzione di riduzione da un problema noto indecidibile al nostro: $\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_{\text{Emptiness}}$

Concludo che, se $\mathcal{L}_{\text{Emptiness}}$ fosse decidibile, allora potrei usare f e il decisore di $\mathcal{L}_{\text{Emptiness}}$ per costruire un decisore per $\mathcal{L}_{\text{Halt}}$. Ma $\mathcal{L}_{\text{Halt}}$ è noto per essere indecidibile: contraddizione. Pertanto, $\mathcal{L}_{\text{Emptiness}}$ è indecidibile.

Dimostrazione. Siano $\mathcal{L}_{\text{Halt}}$, $\mathcal{L}_{\text{Emptiness}}$ due linguaggi su Σ_H^*, Σ_E^* rispettivamente, dove:

- $\mathcal{L}_{\text{Halt}}$ è il linguaggio del problema dell'arresto
- $\mathcal{L}_{\text{Emptiness}}$ è il linguaggio dell'emptiness problem

1. Ipotesi per assurdo

Suppongo per assurdo che $\mathcal{L}_{\text{Emptiness}}$ sia decidibile. Quindi $\exists$ una MdT E che decide $\mathcal{L}_{\text{Emptiness}}$.

2. Riduzione

Costruisco una funzione di riduzione $f : \Sigma_H^* \to \Sigma_E^*$ tale che dato $(R(M), w)^a$ definisco

$$f((R(M), w)) = (R(M'))^b,$$

dove M' su una stringa di input x generica:

- se $x \neq w$, rifiuta
- se $x = w$, esegue M su w dove:
  - se M termina su w, allora M' accetta w $(x = w)$
  - se M non termina (loop) su w, allora M' non termina

Dalla costruzione di f segue che: $w \in \mathcal{L}_{\text{Halt}} \iff f(w) \notin \mathcal{L}_{\text{Emptiness}}$. Concludo che, se $\mathcal{L}_{\text{Emptiness}}$ fosse decidibile, allora potrei usare f e il decisore di $\mathcal{L}_{\text{Emptiness}}$ per costruire un decisore per $\mathcal{L}_{\text{Halt}}$. Ma $\mathcal{L}_{\text{Halt}}$ è noto per essere indecidibile: contraddizione. Pertanto, $\mathcal{L}_{\text{Emptiness}}$ è indecidibile.

□

3. Costruisco MdT che risolve problema dell'arresto (opzionale)

Adesso, costrusco una MdT N che decide $\mathcal{L}_{\text{Halt}}$:

1. N su input $(R(M), w) \in \Sigma_H^*$ calcola $f((R(M), w))^c \in \Sigma_E^*$ (riduzione)
2. N esegue E su $f((R(M), w))$:
   - se E accetta, N rifiuta
     (se E accetta vuol dire che $L(M') = \emptyset$, ovvero che M' loop perchè M loop su w; dato che M loop su w allora N rifiuta^d)
     $w \notin \mathcal{L}_{\text{Halt}} \iff f(w) \in \mathcal{L}_{\text{Emptiness}}$
   - se E rifiuta, N accetta
     (se E rifiuta vuol dire che $L(M') = \{w\}$, ovvero che M' ha accettato $x = w$ perchè M ha terminato su w; dato che M ha terminato su w allora N accetta^e)
     $w \in \mathcal{L}_{\text{Halt}} \iff f(w) \notin \mathcal{L}_{\text{Emptiness}}$

---

[a] istanza del problema dell'arresto

[b] istanza dell'emptiness problem

[c] $f((R(M), w)) = R(M')$

[d] (perchè il problema dell'arresto, per definizione, è definito in modo tale che se se M loop su w, allora rifiuta)

[e] (perché il problema dell'arresto, per definizione, è definito in modo tale che se M termina su w, allora accetta)

Indecidibilità e Semidecidibilità del Linguaggio Vuoto

$\mathcal{L}_\emptyset = \{R(M) \mid L(M) = \emptyset\}$ è indecidibile e **non** semidecidibile

$\overline{\mathcal{L}_\emptyset} = \{R(M) \mid L(M) \neq \emptyset\}$ è indecidibile e semidecidibile

3.6.4 Problema dell'equivalenza dei linguaggi

Teorema : Equivalence Problem

Il problema di determinare se i linguaggi di due MdT coincidono è indecidibile.
$\mathcal{L}_{\text{Equivalence}} = \{(R(M_1), R(M_2)) \mid L(M_1) = L(M_2)\}$ è indecidibile.

Ragionamento. Uso dim per assurdo + riduzione e come conclusione il teorema 3.1. Effettuo una riduzione da un problema noto indecidibile al mio: $\mathcal{L}_{\text{Emptiness}} \leq_m \mathcal{L}_{\text{Equivalence}}$

Dimostrazione.

$$\mathcal{L}_{\text{Equivalence}} = \{(R(M_1), R(M_2)) \mid L(M_1) = L(M_2)\} \qquad \text{(ipotesi)}$$

$$\mathcal{L}_{\text{Equivalence}} \text{ è indecidibile} \qquad \text{(tesi)}$$

Siano $\mathcal{L}_{\text{Emptiness}}$, $\mathcal{L}_{\text{Equivalence}}$ due linguaggi su Σ_E^*, Σ_Q^* rispettivamente, dove:

- $\mathcal{L}_{\text{Emptiness}}$ è il linguaggio dell'emptiness problem
- $\mathcal{L}_{\text{Equivalence}}$ è il linguaggio dell'equivalence problem

1. Ipotesi per assurdo

Suppongo per assurdo che $\mathcal{L}_{\text{Equivalence}}$ sia decidibile. Quindi $\exists$ una MdT Q che decide $\mathcal{L}_{\text{Equivalence}}$ con il seguente comportamento:

$$Q((R(M_1), R(M_2))) = \begin{cases} accept & \text{se } L(M_1) = L(M_2) \\ reject & \text{se } L(M_1) \neq L(M_2) \end{cases}$$

2. Riduzione

Definisco una funzione di riduzione[a] $f : \Sigma_E^* \to \Sigma_Q^*$ come segue.

Dato in input $R(M)$[b] la funzione f genera come output $(R(M), R(M_1))$[c], dove M_1 è la nuova MdT costruita dalla funzione di riduzione f che ha il seguente comportamento ($R(M)$ è la codifica di una MdT arbitraria):

M_1 per ogni stringa di input w: rifiuta; quindi $L(M_1) = \emptyset$

3. Costruisco una MdT che decide l'emptiness problem

Adesso, costruisco una MdT E che decide $\mathcal{L}_{\text{Emptiness}}$:

1. E su input $w \in \Sigma_E^*$ calcola $f(w) \in \Sigma_Q^*$ (riduzione)
2. E esegue Q su $f(w)$:
   - Se Q accetta, allora E accetta
     (se Q accetta vuol dire che i linguaggi delle MdT in input M e M_1 coincidono). $w \in \mathcal{L}_{\text{Emptiness}} \iff f(w) \in \mathcal{L}_{\text{Equivalence}}$
   - Se Q rifiuta, allora E rifiuta
     (se Q rifiuta vuol dire che i linguaggi delle MdT in input M e M_1 sono diversi). $w \notin \mathcal{L}_{\text{Emptiness}} \iff f(w) \notin \mathcal{L}_{\text{Equivalence}}$

*Precisando che $w = R(M)$ e $f(w) = (R(M), R(M_1))$ dove w è l'input dato alla funzione di riduzione e $f(w)$ è l'ouput generato dalla stessa funzione di riduzione, ovvero il valore ottenuto applicando f a w.

Poiché $\mathcal{L}_{\text{Emptiness}} \leq_m \mathcal{L}_{\text{Equivalence}}$ e $\mathcal{L}_{\text{Equivalence}}$ è decidibile (ipotesi per assurdo), segue che $\mathcal{L}_{\text{Emptiness}}$ è decidibile. Ma questo è assurdo perché è ben noto che $\mathcal{L}_{\text{Emptiness}}$ è indecidibile (contraddizione). Poichè ho ottenuto una contraddizione, l'ipotesi che $\mathcal{L}_{\text{Equivalence}}$ sia decidibile è falsa. Pertanto, l'equivalence problem è indecidibile. □

[a] ricorda la funzione di riduzione trasforma un'istanza del problema dell'emptiness problem E in un'istanza dell'equivalence problem Q
[b] istanza dell'emptiness problem
[c] istanza dell'equivalence problem

3.6.5 Problema della terminazione totale

Teorema : Total Halting Problem

Il problema della terminazione totale è indecidibile.
$\mathcal{L}_{\text{Total-Halt}} = \{R(M) \mid M \text{ termina su ogni input } w\}$ è indecidibile.

Ragionamento. Dim per assurdo + riduzione:

1. Nego la tesi (suppongo per assurdo che il nostro problema sia decidibile)
2. Costruisco una funzione di riduzione da un problema noto indecidibile al nostro: $\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_{\text{Total-Halt}}$

Concludo che, se $\mathcal{L}_{\text{Total-Halt}}$ fosse decidibile, allora potrei usare f e il decisore di $\mathcal{L}_{\text{Total-Halt}}$ per costruire un decisore per $\mathcal{L}_{\text{Halt}}$. Ma $\mathcal{L}_{\text{Halt}}$ è noto per essere indecidibile: contraddizione. Pertanto, $\mathcal{L}_{\text{Total-Halt}}$ è indecidibile.

Dimostrazione.

$$\mathcal{L}_{\text{Total-Halt}} = \{R(M) \mid M \text{ termina su ogni input } w\} \qquad \text{(ipotesi)}$$

$$\mathcal{L}_{\text{Total-Halt}} \text{ è indecidibile} \qquad \text{(tesi)}$$

Siano $\mathcal{L}_{\text{Total-Halt}}$, $\mathcal{L}_{\text{Halt}}$ due linguaggi su Σ_T^*, Σ_H^* rispettivamente, dove:

- $\mathcal{L}_{\text{Total-Halt}}$ è il linguaggio del total halting problem
- $\mathcal{L}_{\text{Halt}}$ è il linguaggio del problema dell'arresto

1. Ipotesi per assurdo
Suppongo per assurdo che $\mathcal{L}_{\text{Total-Halt}}$ sia decidibile. Quindi $\exists$ una MdT T che decide $\mathcal{L}_{\text{Total-Halt}}$.
2. Riduzione
Costruisco una funzione di riduzione $f : \Sigma_H^* \to \Sigma_T^*$ tale che dato $(R(M), w)^a$ definisco

$$f((R(M), w)) = (R(M'))^b,$$

dove M' su una stringa di input x generica:

- se $x \neq w$, rifiuta
- se $x = w$, esegue M su w dove:
  - se M termina su w, allora M' accetta o termina? w $(x = w)$
  - se M non termina (loop) su w, allora M' non termina

Dalla costruzione di f segue che: $w \in \mathcal{L}_{\text{Halt}} \iff f(w) \notin \mathcal{L}_{\text{Emptiness}}$. Concludo che, se $\mathcal{L}_{\text{Emptiness}}$ fosse decidibile, allora potrei usare f e il decisore di $\mathcal{L}_{\text{Emptiness}}$ per costruire un decisore per $\mathcal{L}_{\text{Halt}}$. Ma $\mathcal{L}_{\text{Halt}}$ è noto per essere indecidibile: contraddizione. Pertanto, $\mathcal{L}_{\text{Emptiness}}$ è indecidibile.

□

3. Costruisco MdT che risolve problema dell'arresto (opzionale)
Adesso, costrusco una MdT N che decide $\mathcal{L}_{\text{Halt}}$:

1. N su input $(R(M), w) \in \Sigma_H^*$ calcola $f((R(M), w))^c \in \Sigma_T^*$ (riduzione)
2. N esegue E su $f((R(M), w))$:
   - se T accetta, N accetta
     (se T accetta vuol dire che che M' ha accettato o terminato? su w perchè M ha terminato su w; dato che M ha terminato su w allora N accetta[d])
     $w \notin \mathcal{L}_{\text{Halt}} \iff f(w) \in \mathcal{L}_{\text{Emptiness}}$
   - se T rifiuta, N accetta
     (se E rifiuta vuol dire che M' loop su $x = w$ perchè M loop su w; dato che M ha terminato su w allora N accetta[e])
     $w \in \mathcal{L}_{\text{Halt}} \iff f(w) \notin \mathcal{L}_{\text{Emptiness}}$

[a] istanza del problema dell'arresto
[b] istanza dell'total halting problem
[c] $f((R(M), w)) = R(M')$
[d] (perchè il problema dell'arresto, per definizione, è definito in modo tale che se se M loop su w, allora rifiuta)
[e] (perché il problema dell'arresto, per definizione, è definito in modo tale che se M termina su w, allora accetta)

3.7 Proprietà banali e non banali dei linguaggi

Definizione 3.8 (Proprietà di un linguaggio)**.** La proprietà di un linguaggio è predicato applicabile ai linguaggi riconosciuti da MdT, che può essere vera o falsa.

Esempio

Sia $L(M_1)$ linguaggio accettato dalla MdT M:
$L(M_1) = \{a, aa, aaa, aaaa, aaaaa, \dots\}$ che accetta qualunque stringa w in input che contiene solo a.

- Mi chiedo se la proprietà "il linguaggio contiene almeno una $\{aaaaaaaa\}$" è vera o falsa per $L(M_1)$? Vera, perchè $aaaaaaaa \in L(M_1)$.
- Mi chiedo se la proprietà "il linguaggio è esattamente $\{aaaaaaaa\}$" è vera o falsa per $L(M_1)$? Falsa, perchè il linguaggio accettato dalla M_1 è $L(M_1) = \{a, aa, aaa, aaaa, aaaaa, \dots\}$ non è $L(M_1) = \{aaaaaaaa\}$.

Indico con $\mathscr{P}$ una qualunque proprietà di un linguaggio **semidecidibile**. Indico con $L_{\mathscr{P}} = \{L \text{ semidecidibile} \mid L \text{ soddisfa } \mathscr{P}\}$ oppure equivalentemente, $L_{\mathscr{P}} = \{R(M) \mid L(M) \text{ soddisfa } \mathscr{P}\}$

- Se L_1 semidecidibile soddisfa $\mathscr{P}$, ovvero quel linguaggio ha quella proprietà $\implies L_1 \in L_{\mathscr{P}}$

- Se L_1 semidecidibile non soddisfa $\mathscr{P}$, ovvero quel linguaggio non ha quella proprietà $\implies L_1 \notin L_{\mathscr{P}}$

Osservazioni

Una proprietà $\mathscr{P}$ è semplicemente un insieme di linguaggi (accettati da MdT) oppure insieme di codifiche di MdT i cui linguaggi accettati soddisfano quella proprietà.

- $L_\emptyset = \{R(M) \mid L(M) = \emptyset\}$ dove $\mathscr{P} = \emptyset$
  $L_\emptyset$ è l'insieme di tutte le codifiche di MdT che soddisfano la proprietà, ovvero in cui $L(M) = \emptyset$
- $L_{aa} = \{R(M) \mid aa \in L(M)\}$ dove $\mathscr{P} = aa$
  L_{aa} è l'insieme di tutte le codifiche di MdT che soddisfano la proprietà, ovvero in cui nel linguaggio accettato c'è almeno una stinga "aa".
- $L_{aa} = \{R(M) \mid L(M) = \{aa\}\}$ dove $\mathscr{P} = aa$
  L_{aa} è l'insieme di tutte le codifiche di MdT che soddisfano la proprietà, ovvero in cui il linguaggio accettato è esattamente "aa".

Definizione 3.9 (Proprietà banale)**.** Una proprietà $\mathscr{P}$ è banale se:

$$\underbrace{(\mathscr{P} \text{ vera } \forall L(M))}_{\text{Condizione 1}} \vee \underbrace{(\mathscr{P} \text{ falsa } \forall L(M))}_{\text{Condizione 2}}$$

Ma è meglio scrivere:

$$\mathscr{P} \text{ banale se } \underbrace{(\forall L(M) \in L_{\mathscr{P}})}_{\text{Condizione 1}} \vee \underbrace{(\forall L(M) \notin L_{\mathscr{P}})}_{\text{Condizione 2}}$$

Dove $L(M)$ indica il linguaggio accettato da una MdT M arbitraria. Quindi è banale se è vera solo una delle due condizioni.
Trucco: Scegli alcune MdT,osserva i linguaggio accettati. La proprietà è vera per ogni linguaggio accettato? La proprietà è falsa per ogni linguaggio accettato?

Definizione 3.10 (Proprietà non banale)**.** Una proprietà $\mathscr{P}$ è non banale se:

$$\underbrace{(\exists \text{ almeno un } L(M_1) \text{ per cui } \mathscr{P} \text{ è vera})}_{\text{Condizione 1}} \wedge \underbrace{(\exists \text{ almeno un } L(M_2) \text{ per cui } \mathscr{P} \text{ è falsa})}_{\text{Condizione 2}}$$

Ma è meglio scrivere:

$$\mathscr{P} \text{ non banale se } \underbrace{(\exists \text{ almeno un } L(M_1) \text{ per cui } L(M_1) \in L_{\mathscr{P}})}_{\text{Condizione 1}} \wedge \underbrace{(\exists \text{ almeno un } L(M_2) \text{ per cui } L(M_2) \notin L_{\mathscr{P}})}_{\text{Condizione 2}}$$

Quindi è non banale quando soddisfa (vera per) entrambe le condizioni. Ovvero, la proprietà è non banale quando riesco a trovare un $L(M_1)$ per il quale è vera, sia un altro $L(M_2)$ per il quale è falsa. Se, per esempio, ho una proprietà $\mathscr{P}$ e non riesco a trovare nemmeno un $L(M)$ per il quale risulti falsa (quindi per tutti i $L(M)$ è vera), allora $\mathscr{P}$ è banale (perchè soddisfa solo 1 delle 2 condizioni e non entrambe).

Esempio

Sia $L_\emptyset = \{R(M) \mid L(M) = \emptyset\}$ dove $\mathscr{P} = \emptyset$. $\mathscr{P}$ è banale o non banale?
Posso costruire una MdT E che non accetta nessuna stringa, ovvero dove $L(E) = \emptyset$.
Costruisco anche un'altra MdT M che accetta almeno una stringa, ovvero dove $L(M) \neq \emptyset$.
Quindi dato che:

$$(\exists E \text{ t.c. } R(E) \in L_\emptyset) \wedge (\exists M \text{ t.c. } R(M) \notin L_\emptyset) \implies \mathscr{P} \text{ è non banale}$$

Dato che la proprietà è vera per un linguaggio e allo stesso tempo falsa per un'altro, allora $\mathscr{P}$ è non banale.

3.7.1 Teorema di Rice

Teorema : Teorema di Rice

Se $\mathscr{P}$ è una proprietà non banale $\implies$ $L_{\mathscr{P}}$ è indecidibile.

Il teorema si applica solo ai linguaggi semidecidibili. Inoltre, il teorema di Rice indica che se $\mathscr{P}$ è una propietà non banale, $\nexists$ un algoritmo di decisione in grado di affermare se un linguaggio soddisfa quella proprietà oppure no.
Ovvero non esiste una MdT che termina sempre in grado di restituire:

- Sì, se il linguaggio accettato dalla $R(M)$ soddisfa quella propietà
- No, se il linguaggio accettato dalla $R(M)$ non soddisfa quella proprietà

In termini formali:
Se $\mathscr{P}$ è non banale allora $\nexists$ una MdT T che termina sempre su ogni input $R(M)$ con il seguente comportamento:

$$T(R(M)) = \begin{cases} \text{accept}, & \text{se } L(M) \text{ soddisfa } \mathscr{P} \\ \text{reject}, & \text{se } L(M) \text{ non soddisfa } \mathscr{P} \end{cases}$$

Se T accetta vuol dire che $R(M) \in L_{\mathscr{P}}$, se T rifiuta vuol dire che $R(M) \notin L_{\mathscr{P}}$.
Quindi $L_{\mathscr{P}}$ è indecidibile.

Osservazioni

Sappiamo che $L_{\mathscr{P}}$ è indecidibile, ma non dimentichiamoci che:

- $L_{\mathscr{P}}$ può essere semidecidibile
- $L_{\mathscr{P}}$ può essere non semidecidibile

Esercizio 1: applicazione del teorema di Rice

Dimostra che $L_\emptyset = \{R(M) \mid L(M) = \emptyset\}$ è indecidibile.

Ragionamento. Dimostro che $\mathscr{P}$ è una proprietà non banale per poi applicare il teorema di Rice.

Dimostrazione. Posso costruire una MdT E che non accetta nessuna stringa, ovvero dove $L(E) = \emptyset$. Costruisco anche un'altra MdT M che accetta almeno una stringa, ovvero

dove $L(M) \neq \emptyset$. Dato che la proprietà è vera per $L(E)$ ($R(E) \in L_\emptyset$) e falsa per $L(M)$ ($R(M) \notin L_\emptyset$), allora $\mathscr{P}$ è non banale.
Pertanto, per il teorema di Rice, $L_\emptyset$ è indecidibile. □

Esercizio 2: applicazione del teorema di Rice

Dimostra che $L_{\text{finito}} = \{R(M) \mid L(M) \text{ è finito}\}$ è indecidibile.

Dimostrazione. Posso costruire una MdT F che accetta una sola stringa. Costruisco anche un'altra MdT I che accetta tutte le stringhe su $\Sigma = \{0, 1\}$, ovvero dove $L(I) = \Sigma^*$[a]. Dato che la proprietà è vera per $L(F)$ e falsa per $L(I)$, allora $\mathscr{P}$ è non banale.
Pertanto, per il teorema di Rice, L_{finito} è indecidibile. □

[a] Σ^* è infinito poichè rappresenta tutte le possibili combinazioni di 0 e 1 dove $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$

Esercizio 3: non applicabilità del teorema di Rice

$L_{\text{Total-Halt}} = \{R(M) \mid M \text{ termina su ogni input } w\}$ è indecidibile.

Non è possibile applicare il teorema di Rice perchè la proprietà "termina su ogni input" riguarda il comportamento di M. Quindi non è una proprietà del linguaggio accettato $L(M)$ ma della MdT.

4 Riducibilità

Ricorda la notazione f: input $\to$ output

Definizione 4.1 (Funzione di riduzione)**.** Sia L_1,L_2 due linguaggi su Σ_1^*, Σ_2^*, rispettivamente. Si dice che L_1 è riducibile a L_2, e si scrive $L_1 \leq_m L_2$ se $\exists$ una funzione computabile totale $f : \Sigma_1^* \to \Sigma_2^*$ chiamata **funzione di riduzione** t.c. $\forall w \in \Sigma_1^*$

$$w \in L_1 \iff f(w) \in L_2$$

Spiegazione informale:
Per effetturare una riduzione da L_1 a L_2, deve esistere una MdT R (macchina di riduzione) che prende in input una qualunque stringa $w \in \Sigma_1^*$ e la trasforma in una stringa $f(w) \in \Sigma_2^*$. Ovvero, se la MdT R prende in input un'istanza di L_1 allora produce come output un'istanza di L_2.
La MdT R calcola la funzione di riduzione f.
In questo modo, se avessimo una MdT che decide L_2, potremmo decidere L_1 applicando f.

Teorema : 3.1

Se $A \leq_m B$ e B è decidibile $\implies$ A è decidibile.

Dimostrazione 3.1

Se $A \leq_m B$ e B è decidibile $\implies$ A è decidibile.

Ragionamento. A,B sono due linguaggi su Σ_A^*,Σ_B^* rispettivamente.
Per ipotesi:

- $A \leq_m B$, quindi $\exists$ una funzione di riduzione (totale e calcolabile da una MdT) da A a B.
- B è decidibile, quindi $\exists$ una MdT che decide B.

Dimostro che A è decidibile costruendo una MdT che decide A.

Dimostrazione.

$$A \leq_m B \ \wedge \ B \text{ è decidibile} \qquad \text{(ipotesi)}$$

$$A \text{ è decidibile} \qquad \text{(tesi)}$$

Per ipotesi, posso definire M la MdT di decisione per B e $f : \Sigma_A^* \to \Sigma_B^*$ la funzione di riduzione da A a B (calcolabile dalla MdT R).
Costruisco la MdT N che decide A:

1. N su input $w \in \Sigma_A^*$, calcola $f(w) \in \Sigma_B^*$
   (N esegue R su w che effettua la riduzione producendo come output $f(w)$)
2. N esegue M su $f(w)$:
   - Se M accetta $f(w)$, allora N accetta w.
     $f(w) \in B \iff w \in A$
   - Se M rifiuta $f(w)$, allora N rifiuta w.
     $f(w) \notin B \iff w \notin A$

Conclusione: Ho costruito un algoritmo di decisione per A che applica la riduzione per ogni input e sfrutta la MdT che decide B per A. Inoltre, la MdT N si arresta sempre per ogni input w, perchè f è una funzione totale computabile (la funzione di riduzione) e M è un algoritmo di decisione (MdT che si ferma per ogni input) per B.
Pertanto, A è decidibile.

□

*Notare che scrivo "accetta" o "rifiuta" e non "termina in uno stato accettante" o "termina in uno stato di rifiuto" perchè ho indicato "costruisco la MdT N che decide A" e una MdT che decide un linguaggio si ferma per ogni input, quindi sarebbe ridondante scriverlo.

Spiegazione extra della dimostrazione:
Ovvero, la macchina N ha due MdT al suo interno (prima esegue R e poi M):

- MdT R che effettua la riduzione da A a B:
  1. R riceve in input w (istanza di A)
  2. R effettua la riduzione; calcola la funzione di riduzione su w (applica f a w che scrivo come $f(w)$)
  3. R produce come output $f(w)$ (istanza di B)
- MdT M che decide B:
  1. M riceve in input $f(w)$
  2. Se la computazione di M termina:
     - in uno stato accettante, allora $f(w) \in B$
     - in uno stato di rifiuto, allora $f(w) \notin B$

Quindi, N termina accettando w se e solo se M termina accettando $f(w)$. Oppure, N termina rifiutando w se e solo se M termina rifiutando $f(w)$. Come ben sappiamo, M decide solo istanze di B, per questo è necessario trasformare un'istanza di A in una di B. La riduzione serve perchè M non può ricevere un'istanza di A ma solo di B (perchè per ipotesi B è decidibile e M è la macchina di decisione per B), quindi è necessario che "qualcuno" effettui la trasformazione, che è proprio quello che fa la MdT R.
Potendo trasformare un'istanza di A in B e sapendo che B è decidibile allora posso "decidere" A.

Teorema : 3.2

Se $A \leq_m B$ e A è indecidibile $\implies$ B è indecidibile.

Dimostrazione 3.2

Se $A \leq_m B$ e A è indecidibile $\implies$ B è indecidibile.

Ragionamento. Per dimostarlo, suppongo per assurdo che B sia decidibile (nego la tesi, ipotesi per assurdo) e poi applico la riduzione e poi le definizioni che mi porteranno ad una contraddizione che rende falsa la mia ipotesi per assurdo rendendo poi vero il teorema.

Dimostrazione.

$$A \leq_m B \ \wedge \ A \text{ è indecidibile} \qquad \text{(ipotesi)}$$

$$B \text{ è indecidibile} \qquad \text{(tesi)}$$

Suppongo per assurdo che B sia decidibile (ipotesi per assurdo).
Quindi per ipotesi $\exists$ una MdT M che decide B. Inoltre, per ipotesi (del teorema), $\exists$ una

funzione di riduzione $f : \Sigma_A^* \to \Sigma_B^*$ t.c. $\forall w \in \Sigma_A^*$:

$$w \in A \iff f(w) \in B$$

Costruisco una MdT N che decide A:

1. N su input $w \in \Sigma_A^*$ calcola $f(w) \in \Sigma_B^*$
2. N esegue M su $f(w)$:
   - Se M accetta, allora N accetta
     (Se $f(w) \in B$, allora $w \in A$)
   - Se M rifiuta, allora N rifiuta
     (Se $f(w) \notin B$, allora $w \notin A$)

Ho costruito un algoritmo di decisione per A. Ma quindi se $A \leq_m B$ e B è decidibile, allora per definizione (teorema 3.1) A è decidibile. Ma questo è assurdo (contraddizione) perchè A è indecidibile (secondo l'ipotesi del teorema). Poichè abbiamo ottenuto una contraddizione, l'ipotesi che B sia decidibile è falsa. Pertanto, B è indecidibile. □

Teorema : 3.3

Se $A \leq_m B$ e B è semidecidibile $\implies$ A è semidecidibile.

Teorema : 3.4

Se $A \leq_m B$ e A non è semidecidibile $\implies$ B non è semidecidibile.

4.0.1 Riduzione non sempre funziona

La riduzione non sempre funziona, perchè sappiamo che vale:

Teorema : 3.5

$A \leq_m B \iff \overline{A} \leq_m \overline{B}$

Esempio 3.5

Il linguaggio del problema dell'arresto è riducibile al linguaggio dell'emptiness problem?
$\mathcal{L}_{\text{Halt}} \overset{?}{\leq_m} \mathcal{L}_\emptyset$

Ragionamento. Sappiamo che:

- $\mathcal{L}_{\text{Halt}}$ è indecidibile e semidecidibile ($\overline{\mathcal{L}_{\text{Halt}}}$ non è semidecidibile)
- $\mathcal{L}_\emptyset$ è indecidibile e non semidecidibile ($\overline{\mathcal{L}_\emptyset}$ è semidecidibile)

Se voglio che la riduzione funzioni, deve valere:
$\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_\emptyset \iff \overline{\mathcal{L}_{\text{Halt}}} \leq_m \overline{\mathcal{L}_\emptyset}$

Dimostrazione. ($\implies$)
$\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_\emptyset \implies \overline{\mathcal{L}_{\text{Halt}}} \leq_m \overline{\mathcal{L}_\emptyset}$

$$\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_\emptyset \qquad \text{(ipotesi)}$$

$$\overline{\mathcal{L}_{\text{Halt}}} \leq_m \overline{\mathcal{L}_\emptyset} \qquad \text{(tesi)}$$

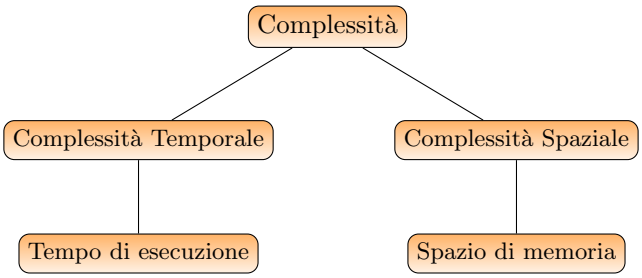
Suppongo per assurdo che $\overline{\mathcal{L}_{\text{Halt}}} \leq_m \overline{\mathcal{L}_\emptyset}$. Se fosse vero, allora per definizione:
Se $\overline{\mathcal{L}_{\text{Halt}}} \leq_m \overline{\mathcal{L}_\emptyset}$ e $\overline{\mathcal{L}_{\text{Halt}}}$ non è semidecidibile $\implies$ $\overline{\mathcal{L}_\emptyset}$ non è semidecidibile (teorema 3.4). Ma questo è assurdo perchè $\overline{\mathcal{L}_\emptyset}$ è noto per essere semidecidibile. Ciò porta ad una contraddizione che rende falsa la mia ipotesi per assurdo (che $\overline{\mathcal{L}_{\text{Halt}}} \leq_m \overline{\mathcal{L}_\emptyset}$).
Quindi, $\overline{\mathcal{L}_{\text{Halt}}} \not\leq_m \overline{\mathcal{L}_\emptyset}$. Dato che per ipotesi $\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_\emptyset$ è vera e abbiamo dimostrato che $\overline{\mathcal{L}_{\text{Halt}}} \leq_m \overline{\mathcal{L}_\emptyset}$ è falsa, allora per definizione, questa implicazione:
$\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_\emptyset \implies \overline{\mathcal{L}_{\text{Halt}}} \leq_m \overline{\mathcal{L}_\emptyset}$ è falsa. □

Dimostrazione. ($\Longleftarrow$)
//TODO: □

Conclusione: $\mathcal{L}_{\text{Halt}} \leq_m \mathcal{L}_\emptyset \iff \overline{\mathcal{L}_{\text{Halt}}} \leq_m \overline{\mathcal{L}_\emptyset}$ è falsa. Pertanto, $\mathcal{L}_{\text{Halt}} \not\leq_m \mathcal{L}_\emptyset$.

5 Complessità Temporale

Una volta trovato un algoritmo di risoluzione al problema (decidibile o semidecidibile), ne calcolo l'efficienza.



A cosa serve analizzare la complessità?

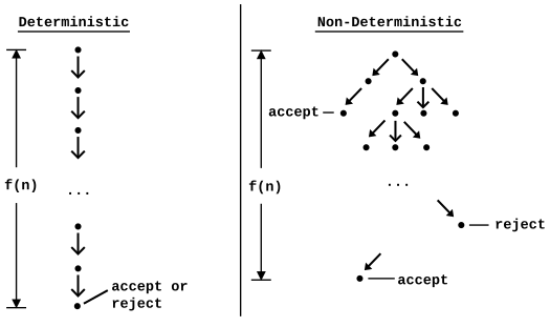
Anche se un problema è risolvibile in linea di principio, la sua soluzione potrebbe non essere fattibile dal punto di vista pratico. Ad esempio, l'algoritmo che risolve il problema potrebbe richiedere un tempo di esecuzione troppo elevato (ad esempio milioni di anni) oppure richiede un numero inaccettabile di risorse computazionali.

Problemi non trattabili

Quando la soluzione di un problema richiede una quantità di tempo o di spazio di memoria che cresce in modo **non polinomiale** rispetto alla dimensione dell'input, il problema viene detto non trattabile.

Analisi della complessità temporale

Per analizzare la complessità temporale di un algoritmo, si considera il **caso peggiore**, cioè guardiamo il massimo tempo possibile tra tutti gli input di lunghezza n.



Definizione 5.1 (Time Complexity - Deterministica)**.** La complessità temporale di una MdT *decider*[5] deterministica M è una funzione $tc_M : \mathbb{N} \to \mathbb{N}$ dove

$$tc_M(n) = \text{massimo numero di passi (transizioni) che } M \text{ esegue su qualunque input di lunghezza } n$$

[5]che termina per ogni input

Definizione 5.2 (Time Complexity - Non deterministica)**.** La complessità temporale di una MdT *decider* non deterministica M è una funzione $tc_M : \mathbb{N} \to \mathbb{N}$ dove

$tc_M(n)$ = massimo numero di passi (transizioni) che M esegue sul ramo più lungo, su qualunque input di lunghezza n

Se, per esempio, $tc_M(n) = 6n^3 + 2n^2 + 3$ allora posso dire che:

$tc_M(n) = O(n^3)$ dove n è la lunghezza dell'input

5.1 $\mathcal{P}$

La classe $\mathcal{P}$ è l'insieme dei linguaggi *decisi*[6] da una MdT (decider) deterministica in tempo polinomiale. Quindi, è l'insieme dei linguaggi decidibili in tempo polinomiale.

Come dimostro che un algoritmo appartiene alla classe $\mathcal{P}$?

1. Descrivo l'algoritmo ad alto livello dividendolo in passi (ignorando i dettagli di nastri e testine).
2. Mostro che il numero totale di passi dell'algoritmo, su input di lunghezza n, cresce al massimo come un polinomio. Uso notazione O-grande.
3. Verifico che ciascun passo possa essere eseguito in tempo polinomiale su una MdT deterministica.
4. Mi assicuro che la MdT costruita sia un decider, quindi che si arresti e fornisca una risposta per ogni input.

Codifica dei grafi: Sia $G = (V, E)$ un grafo.

- **Lista di adiacenza** (da usare quando il grafo è sparso[7]): lista dove per ogni nodo del grafo vengono elencati altri nodi a cui è connesso.
- **Matrice di adiacenza** (da usare quando il grafo è denso[8]): matrice quadrata dove l'elemento (i, j) vale 1 se esiste un arco dal nodo i al nodo j, 0 altrimenti.

Osservazione

Quando analizziamo algoritmi su grafi, possiamo esprimere il tempo di esecuzione in base al numero di nodi del grafo, invece che della dimensione della rappresentazione del grafo (cioè della lunghezza dell'input codificato). Esempio: in un problema su grafi, l'algoritmo impiega tempo $O(n^2)$ dove n è il numero di nodi del grafo.

[6] un linguaggio L è decidibile se $\exists$ una MdT che decide L.
[7] ha pochi archi rispetto ai nodi
[8] ha molti archi, quasi tutti i nodi collegati tra loro

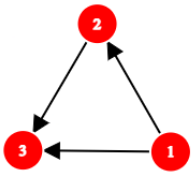
Esempio: codifica binaria della lista di adiacenza per MdT

Sia G un grafo orientato $G = (V, E)$:

- $V = \{1, 2, 3\}$
- $E = \{(1, 2), (1, 3), (2, 3)\}$

con la rappresentazione delle liste di adiacenza per ogni nodo:

Nodo	**Liste di adiacenza**
1	2, 3
2	3
3	-



Conversione in codifica binaria:

- Sia $\Sigma = \{0, 1, \#\}$ l'alfabeto
- Il simbolo # separa i nodi nella rispettiva lista di adiacenza
- Il simbolo ## separa le liste di adiacenza dei diversi nodi
- Il simbolo ### indica la fine della lista

Pertanto:
Nodo 1: archi verso 2 e 3, quindi la codifica è 1#10#11
Nodo 2: arco verso 3, quindi la codifica è 10#11
Nodo 3: -
Quindi l'intera codifica del grafo sarà 1#10#11##10#11###

Esempio: codifica binaria della matrice di adiacenza per MdT

Sia G un grafo orientato $G = (V, E)$:

- $V = \{1, 2, 3\}$
- $E = \{(1, 2), (1, 3), (2, 3)\}$

con la matrice di adiacenza associata:

- i sono le righe
- j sono le colonne

$A[i, j] =$

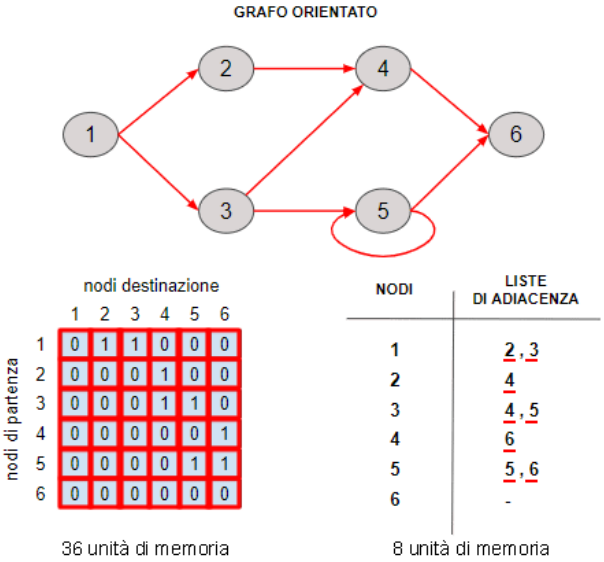
	1	2	3
1	0	1	1
2	0	0	1
3	0	0	0

Conversione in codifica binaria:

- Sia $\Sigma = \{0, 1, \#\}$ l'alfabeto
- 0/1 valori nella matrice (assenza/presenza di arco)
- Il simbolo # separa le righe
- Il simbolo ### indica della codifica

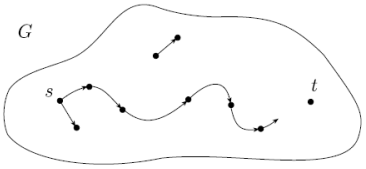
Scrivo ogni riga (sequenza di bit) separata dal simbolo "#", pertanto:
Riga 1, per $i = 1$: scrivo 011 [a]
Riga 2, per $i = 2$: scrivo 001
Riga 3, per $i = 3$: scrivo 010
Quindi l'intera codifica della matrice di adiacenza è 011#001#000###

[a] leggendo l'intera riga 1 vediamo la sequenza di bit 011



5.1.1 *PATH*

Sia un grafo orientato G che contiene i nodi s e t. Il problema $PATH$ consiste nel determinare se esiste un cammino da s a t.



Teorema : $PATH$ Problem

$PATH = \{\langle G, s, t\rangle \mid G \text{ è un grafo orientato con cammino da } s \text{ a } t\}$

$$PATH \in \mathcal{P}$$

Ragionamento. Costruisco un algoritmo di decisione deterministico in tempo polinomiale per $PATH$. Uso l'algoritmo di ricerca in ampiezza[a].

Dimostrazione. Costruisco una MdT M deterministica a 2 nastri che decide $PATH$ come segue. Sia $\langle G, s, t\rangle$ la codifica del grafo orientato G e dei nodi s, t.

1. Costruzione della MdT M

M su input $\langle G, s, t\rangle$:

1. Marca il nodo s come visitato sul secondo nastro
2. Ripeti quanto segue finchè si possono marcare nuovi nodi[b]:
   (a) Scorri tutti gli archi del grafo G sul nastro 1.
   (b) Se trovi un arco (u, v) che va da un nodo u già visitato a un nodo v non visitato, allora marca v sul nastro 2.
3. Se t è marcato come visitato[c], allora accetta. Altrimenti, rifiuta.

2. Analisi della complessità temporale della MdT M

1. Verifico che ciascun passo possa essere eseguito in tempo polinomiale su una MdT deterministica:
   - **Passo 1:** operazione che, indipendentemente dalla dimensione del grafo, ha sempre tempo costante O(1).
   - **Passo 2:** questo passo (ciclo, ripetuto massimo n volte dove n sono i nodi) fa il lavoro grosso:
     – Scorre tutti gli archi e del grafo, tempo O(e)
     – Controlla se, per ogni arco (u, v), u è stato visitato e v no (ricerca nel nastro 2, tempo O(n))
     – Se trovo un arco valido, allora marco v come visitato (marcatura tempo O(1))

     Pertanto, ho $\mathrm{O}(n) \cdot \big[O(e) \cdot O(n) \cdot O(1)\big] = O(n) \cdot \big[O(e \cdot n)\big] = O(e \cdot n^2)$
   - **Passo 3:** operazione che, indipendentemente dalla dimensione del grafo, ha sempre tempo costante O(1).
2. Mostro che il numero tot di passi dell'algoritmo, su input di lunghezza n, cresce al massimo come un polinomio:
   - **Passo 1:** tempo O(1)
   - **Passo 2:** tempo O($e \cdot n^2$)
   - **Passo 3:** tempo O(1)

   Pertanto, la complessità nel caso pessimo dell'algoritmo è:

   $$tC_M(x) = \mathrm{O}(1) + \mathrm{O}(e \cdot n^2) + \mathrm{O}(1) = \mathrm{O}(e \cdot n^2)$$

   dove "n" sono i nodi, "e" gli archi e "x" è la lunghezza dell'input $\langle G, s, t\rangle$

   La complessità è quindi polinomiale.

Conclusione: Ho costruito una MdT M deterministica che decide $PATH$ in tempo polinomiale. Quindi, $PATH \in \mathcal{P}$. □

[a] algoritmo BFS:

1. parto dal nodo sorgente s, lo aggiungo alla queue e lo segno come visitato
2. finché la queue non è vuota:
   (a) estraggo un nodo u dalla queue
   (b) per ogni nodo adiacente a u non ancora visitato, aggiungilo alla queue e segnalo come visitato

L'algoritmo termina quando tutti i nodi raggiungibili da s sono stati visitati. Se t è tra i nodi visitati, allora esiste un cammino da s a t.

[b] Attenzione: non scrivo "finché tutti i nodi non vengono visitati" perchè se alcuni nodi non sono raggiungibili da s, non verranno mai marcati — quindi il ciclo "finché tutti" non terminerebbe mai.

[c] esite un cammino da s a t

5.1.2 *RELPRIME*

Due numeri x e y sono relativamente primi se 1 è l'unico numero intero che li divide entrambi. Per esempio, 4 e 8 non sono relativamente primi perchè 2 li divide entrambi.

Teorema : RELPRIME Problem

$RELPRIME = \{\langle x, y\rangle \mid x \text{ e } y \text{ sono relativamente primi}\}$

$$RELPRIME \in \mathcal{P}$$

Ragionamento. Costruisco un algoritmo di decisione determinisitco in tempo polinomiale per $RELPRIME$. Uso l'algoritmo di Euclide per calcolare il mcd.

$$x \text{ e } y \text{ sono relativamente primi} \iff mcd(x, y) = 1$$

Dimostrazione. 1. Algoritmo di Euclide:
Sia E la MdT che calcola il massimo comune divisore di x e y usando l'algoritmo di Euclide.

E su input $\langle x, y\rangle$ dove x e y sono numeri naturali in codifica binaria:

1. Ripeti finchè $y \neq 0$:
   (a) Calcola x mod y e assegna il valore a x ($x \leftarrow x$ mod y)
   (b) Scambia x e y ($swap(x, y)$)
2. Restituisci x (l'output che alla fine sarà il mcd)

2. Costruisco MdT che decide $RELPRIME$:
Costruisco la MdT R che simula la MdT E.

R su input $\langle x, y\rangle$ dove x e y sono numeri naturali in codifica binaria:

1. Esegue E su $\langle x, y\rangle$:
   - Se E restituisce 1, allora R accetta.
   - Se E restituisce un altro valore diverso da 1, allora R rifiuta.

Quindi R accetta solo se $mcd(x, y) = 1$, rifiuta altrimenti. Pertanto R è una MdT che decide correttamente $RELPRIME$. Adesso analizzo la complessità.
3. Analisi della complessità temporale

1. Analizzo ciascun passo della MdT E:
   - **Passo 1.a:** il calcolo del modulo richiede tempo $O(n^2)$, l'assegnazione $x = x$ mod y richiede tempo $O(n)$; quindi $O(n^2)+O(n)=O(n^2)$
   - **Passo 1.b:** scambiare x e y richiede tempo $O(n)$
   - **Passo 1:** Quante iterazioni servono affinchè $y = 0$? $O(\log_2 x)=O(n)$
   - **Passo 2:** restituire il valore di x richiede tempo $O(n)$

   Nel passo 1: $O(\log_2 x)\cdot[O(n^2)+O(n)] = O(\log_2 x)\cdot[O(n^2)] = O(n)\cdot[O(n^2)] = O(n^3)$
   Nel passo 2: $O(n)$
2. Mostro che il numero tot di passi dell'algoritmo, su input di lunghezza n, cresce al massimo come un polinomio:

- La MdT E: O(n^3)
- La MdT R che simula E ha O(n^3) e poi accetta o rifiuta in base a cosa restituisce E, quindi O(n)

Pertanto, la complessità è $tC_R(n) = O(n^3) + O(n) = O(n^3)$ dove n è la lunghezza dell'input, quindi polinomiale.

Conclusione: Ho costruito una MdT R determinisitca che decide $RELPRIME$ in tempo polinomiale. Quindi, $RELPRIME \in \mathcal{P}$ □

5.1.3 $2 - SAT$

5.2 $\mathcal{NP}$

La classe $\mathcal{NP}$ è l'insieme dei linguaggi *decisi*[9] da una MdT (decider) non deterministica in tempo polinomiale. Quindi, è l'insieme dei linguaggi decidibili in tempo polinomiale.

Osservazione

La classe $\mathcal{NP}$ è l'insieme dei linguaggi:

- risolvibili da una MdT non deterministica in tempo polinomiale (senza certificato), oppure
- verificabili da una MdT deterministica in tempo polinomiale, dato come input un certificato

Tipo di MdT	Input	Richiede certificato?
Non Deterministica	istanza	No
Deterministica (verificatore)	istanza + certificato	Sì

Tabella 1: Differenza tra MdT nondeterministica e MdT deterministica (verificatore) nei problemi NP

5.2.1 Riduzione polinomiale

Ricorda la notazione f: input $\rightarrow$ output

Definizione 5.3 (Funzione di riduzione in tempo polinomiale)**.** Sia L_1,L_2 due linguaggi su Σ_1^*, Σ_2^*, rispettivamente. Si dice che L_1 è polinomialmente riducibile a L_2, e si scrive $L_1 \leq_p L_2$ se $\exists$ una funzione computabile totale calcolabile in tempo polinomiale $f : \Sigma_1^* \rightarrow \Sigma_2^*$ chiamata **funzione di riduzione in tempo polinomiale** t.c. $\forall w \in \Sigma_1^*$

$$w \in L_1 \iff f(w) \in L_2$$

Spiegazione informale:
Per effetturare una riduzione da L_1 a L_2, deve esistere una MdT R (macchina

[9] un linguaggio L è decidibile se $\exists$ una MdT che decide L.

di riduzione) che prende in input una qualunque stringa $w \in \Sigma_1^*$ e la trasforma in una stringa $f(w) \in \Sigma_2^*$. Ovvero, se la MdT R prende in input un'istanza di L_1 allora produce come output un'istanza di L_2.
La MdT R calcola la funzione di riduzione f in tempo polinomiale.
In questo modo, se avessimo una MdT che decide L_2, potremmo decidere L_1 applicando f.

Teorema : 5.2.1.1

Se $L_1 \leq_p L_2$ e $L_2 \in \mathcal{P}$, allora $L_1 \in \mathcal{P}$

Teorema : 5.2.1.2

Se $L_1 \leq_p L_2$ e $L_2 \in \mathcal{NP}$, allora $L_1 \in \mathcal{NP}$

5.2.2 $\mathcal{NP}$-Hard

Un problema L è $\mathcal{NP}$-Hard (o $\mathcal{NP}$-Difficile) se tutti i problemi in $\mathcal{NP}$ si possono ridurre a L in tempo polinomiale.

Teorema : $\mathcal{NP}$-Hard

Un linguaggio L è $\mathcal{NP}$-Hard se:
$\forall Q$ linguaggio $\in \mathcal{NP}$, $\exists f$ riduzione polinomiale da Q a L ($Q \leq_p L$).

Teorema : $\mathcal{NP}$-Hard

Se $L_1 \leq_p L_2$ e L_1 è $\mathcal{NP}$-Hard, allora L_2 è $\mathcal{NP}$-Hard

Un problema L può:

- essere $\mathcal{NP}$-Hard $\wedge \in \mathcal{NP}$, oppure
- essere $\mathcal{NP}$-Hard $\wedge \notin \mathcal{NP}$

Se B è $\mathcal{NP}$-Hard $\wedge$ $B \in \mathcal{NP}$ $\implies$ B è $\mathcal{NP}$-Completo.

5.2.3 $\mathcal{NP}$-Completo

Teorema : $\mathcal{NP}$-Completo

Un linguaggio L è $\mathcal{NP}$-Completo se:

- $L \in \mathcal{NP}$, e
- $L \in \mathcal{NP}$-Hard

Teorema : 5.2.3

- B è $\mathcal{NP}$-Completo
- $C \in \mathcal{NP}$

Se $B \leq_p C$, allora C è $\mathcal{NP}$-Completo.

5.2.4 Polinomio Booleano

Definizione 5.4 (Variabile booleana)**.** Una variabile booleana è una variabile x che assume solo due valori:

- $x = 0$ (falso)
- $x = 1$ (vero)

Operatori logici:

- AND, indicato con il simbolo $\wedge$
- OR, indicato con il simbolo $\vee$
- NOT, indicato con il simbolo $\neg$

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \to Q$	$P \leftrightarrow Q$
V	V	F	V	V	V	V
V	F	F	F	V	F	F
F	V	V	F	V	V	F
F	F	V	F	F	V	V

Definizione 5.5 (Letterale)**.** Un letterale x è una variabile booleana o la sua negazione $\overline{x}$.

Definizione 5.6 (Clausola)**.** Una clausola è una disgiunzione di letterali:

$$(x_1 \vee \cdots \vee x_n)$$

La clausola è vera se almeno uno dei suoi letterali è vero.

Definizione 5.7 (CNF)**.** Una formula è in forma normale congiuntiva se è una congiunzione ($\wedge$) di clausole (dove le clausole sono una disgiunzione di letterali):

$$\underbrace{(x_{1,1} \vee \cdots \vee x_{1,n})}_{\text{clausola 1}} \wedge \underbrace{(x_{2,1} \vee \cdots \vee x_{2,n})}_{\text{clausola 2}} \wedge \cdots \wedge \underbrace{(x_{n,1} \vee \cdots \vee x_{n,n})}_{\text{clausola n}}$$

La formula in CNF è vera se tutte le clausole sono vere.

Definizione 5.8 (Formula booleana)**.** Un formula booleana è un'espressione costruita con:

- letterali (positivi o negativi)
- operatori logici $(\wedge, \vee, \neg)$

Definizione 5.9 (Polinomio booleano)**.** Un polinomio booleano è un'espressione costruita con:

- letterali (positivi o negativi)
- operatori aritmetici in cui:
  - $(\neg x)$ corrisponde a $1 - x$
  - $(x \wedge y)$ corrisponde a $x \cdot y$
  - $(x \vee y)$ corrisponde a $x + y - x \cdot y$

Quindi il polinomio booleano è la rappresentazione algebrica della formula booleana.

Esempio

Date le variabili $\{x_1, x_2, x_3\}$, una formula booleana in CNF può essere:

$$f = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$$

mentre il corrispondente polinomio booleano è:

$$\begin{gathered} p = (x_1 \vee (1 - x_2)) \wedge ((1 - x_1) \vee x_3) \\ = \big[(x_1 + (1 - x_2)) - (x_1 \cdot (1 - x_2))\big] \wedge \big[((1 - x_1) + x_3) - ((1 - x_1) \cdot x_3)\big] \\ = \big[(x_1 + (1 - x_2)) - (x_1 \cdot (1 - x_2))\big] \cdot \big[((1 - x_1) + x_3) - ((1 - x_1) \cdot x_3)\big] \\ \text{sviluppando e sapendo che per proprietà booleana } x_i^2 = x_i \text{ allora ottengo:} \\ = x_1 x_2 + x_1 x_3 - x_1 - x_2 + 1 \end{gathered}$$

5.2.5 *SAT*

Data una formula booleana f scritta in CNF, esiste un qualche assegnamento di valori (0/1) alle variabili che rende l'intera formula f vera[10]?

Osservazione

Il numero di letterali in ogni clausola può variare; non c'è un limite fisso.
Per esempio:

$$(x \vee y) \wedge (\overline{z} \vee w \vee u) \wedge (v)$$

Nella prima clausola ci sono 2 lettarli, nella seconda 3 e nell'ultima 1.

[10] la formula booleana restituisce come output 1

Teorema : SAT Problem

Data f, una formula booleana in CNF, determinare se $\exists$ un qualche assegnamento di valori alle variabili che renda f vera.

$$SAT = \{\langle f \rangle \mid f \text{ è una cnf-formula booleana soddisfacibile}\}$$
$$SAT \in \mathcal{NP}$$

Dimostrazione. Sia $\{x_1, x_2, ..., x_n\}$ l'insieme delle variabili booleane.

1. Codifica

Ogni variabile è codificata in binario come segue: numero del letterale in binario seguito da #1 se il letterale è positivo oppure #0 se il letterale è negativo.

Letterale	Codifica
x_i	$i\#1$
$\neg x_i$	$i\#0$

Esempio:
Sia f in CNF: $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$. Verrà codificato come:

$$1\#1 \vee 10\#0 \wedge 1\#0 \vee 11\#1$$

La codifica finale in input alla MdT sarà composta da:

- una lista di interi da 1 a n in binario separati dal simbolo # (che indicano le variabili presenti nella formula)
- codifica della formula booleana

la lista e la formula sono separati dal simbolo ##.

Esempio:
Sia f in CNF: $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$. La codifica in input alla MdT sarà:

$$\underbrace{1\#10\#11}_{\text{variabili}}\#\#\underbrace{1\#1 \vee 10\#0 \wedge 1\#0 \vee 11\#1}_{\text{formula booleana}}$$

2. Alfabeto e Linguaggio

- Sia $\Sigma_{SAT} = \{0, 1, \#, \wedge, \vee\}$ l'alfabeto del problema SAT.
- Sia L_{SAT} oppure semplicemente SAT il linguaggio del problema, ovvero l'insieme di tutte le formule booleane per cui esiste un assegnamento che le rende vere.
  $L_{SAT} = \{\langle f \rangle \mid f \text{ formula booleana soddisfacibile}\}$

3. Costruzione della MdT non deterministica

Costruisco una MdT S non deterministica che decide SAT in tempo polinomiale.
S ha due nastri:

- **Nastro 1:** codifica finale
- **Nastro 2:** nastro di lavoro che genera su di esso (non deterministicamente) un assegnamento alle variabili come segue:

  $$x_1\#t(x_1)\#\#x_2\#t(x_2)\#\#\cdots x_n\#t(x_n)$$

  dove:

  - x_i è la rappresentazione binaria dell'indice i della variabile corrispondente
  - $t(x_i)$ indica l'assegnamento del valore alla variabile x_i, che può essere 1 o 0

Esempio grafico dei nastri nel caso della formula $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$:

Nastro 2	*	1	#	$t(x_1)$	#	#	10	#	$t(x_2)$	#	#	11	#	$t(x_3)$	*	*	*	*	*	*	*	*	*	*	*	*
Nastro 1	*	1	#	10	#	11	#	#	1	#	1	V	10	#	0	∧	1	#	0	V	11	#	1	*	*	*

4. Funzionamento della MdT S non deterministica
S è la MdT non deterministica e $\langle f \rangle$ è codifica finale sopra descritta.

S su input $\langle f \rangle$:
Sul nastro 1 ho la codifica finale di f e il nastro 2 è vuoto.

1. S controlla che la stringa in input sul nastro 1 sia sintatticamente corretta; se non lo è, rifiuta.
2. S sul nastro 2 genera (non deterministicamente) un assegnamento alle variabili.
3. S esamina l'input sul nastro 1 da sinistra verso destra, fino ad incontrare un letterale:
   (a) Confronta il letterale della clausola sul nastro 1 $v\#t(v)$ con l'assegnamento corrispondente sul nastro 2 $x_i\#t(x_i)$:
   - Se corrispondono, la clausola è soddisfatta (basta un assegnamento vero per rendere vera l'intera clausola vera);
     - Se la clausola appena esaminata era l'ultima, allora S accetta e termina ($f \in SAT$).
     - Se la clausola appena esaminata non era l'ultima, allora passo alla clausola successiva;
   - Se non corrispondono, esamino il letterale successivo;
     - Se il letterale appena esaminato è l'ultimo della clausola, allora S rifiuta e termina ($f \notin SAT$).
     - Se il letterale appena esaminato non è l'ultimo della clausola, allora passa al letterale successivo;

Spiegazione dettagliata:
A seguito viene spiegato il comportamento di S con ciclo interno ed esterno perchè cosi è più facile calcolare la complessità. Ricordo che la formula è vera se tutte le sue clausole sono vere (dove ogni clausola è vera se almeno uno dei valori è vero).

(a) Ciclo esterno: scansiona tutte le clausole della formula
(b) Ciclo interno: scansione ogni letterale della clausola

- Per ogni clausola della formula booleana (ciclo esterno):
  - Per ogni letterale della clausola (ciclo interno):
    (a) Confronta il letterale sul nastro 1 $v\#t(v)$ con l'assegnamento corrispondente sul nastro 2 $x_1\#t(x_1)$:
    * Se coincidono, la clausola è soddisfatta, quindi esci dal ciclo interno (uscendo dal ciclo interno va allo step (a) sotto)
    * Se non coincidono e il letterale esaminato è l'ultimo della clausola, S rifiuta e termina; Altrimenti passa al prossimo letterale.

(a) Se la clausola è l'ultima dell'intera formula booleana, allora S accetta e termina.[a]
(b) Se la clausola non è l'ultima dell'intera formula booleana, allora passa alla prossima clausola.

5. Complessità della MdT S

1. Analizzo complessità ciclo interno: (ripetuto per n letterali in caso pessimo)
   In questo caso il caso pessimo è quando analizza tutti i letterali fino all'ultimo della clausola. Confronto costa $O(\log_2 n)$ sul nastro 1 e $O(\log_2 n)$ sul nastro 2 perchè deve leggere e muovere la testina in base a quanto è lungo il letterale in esame (in codifica binaria).
   Approssimativamente, ripeto il ciclo per n letterali, quindi la complessità è $n \cdot O(\log_2 n) = O(n \log_2 n)$.
2. Analizzo complessità ciclo esterno: (ripetuto per n clausole della formula booleana in caso pessimo)
   In questo caso il caso pessimo è quando analizza tutte le clausole dell'intera formula. Approssimativamente, costa $O(n)$.

Mettendo tutto insieme, la complessità di S è $O(n \log_2 n) \cdot O(n) = O(n^2 \log_2 n)$ e quindi polinomiale.

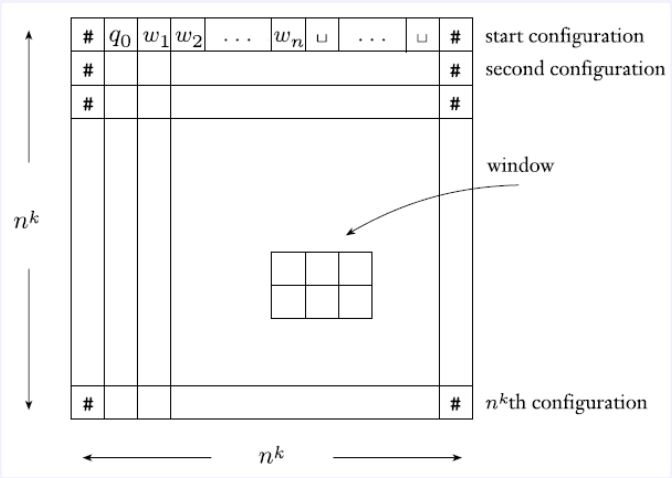
Conclusione: La MdT S non deterministica decide SAT in tempo polinomiale. Pertanto, $SAT \in \mathcal{NP}$ □

[a]non importa specificare che la clausola è soddisfatta perchè questo passo della computazione è eseguibile uscendo dal ciclo interno solo quando la clausola è soddisfatta

Teorema : Teorema di Cook-Levin

SAT è $\mathcal{NP}$-Hard.

Ragionamento. Devo costruire una funzione di riduzione in tempo polinomiale in modo che ogni problema in $\mathcal{NP}$ possa essere riducibile a SAT. Ma i problemi in $\mathcal{NP}$ sono infiniti, non posso provarli tutti, non finirei mai. Dato che ogni problema $\mathcal{NP}$ è per definizione decidibile da una MdT non deterministica in tempo polinomiale, per fare la riduzione uso il concetto di *tableau*.
Il tableau è una rappresentazione generica dell'esecuzione di una macchina non deterministica, quindi il tableau può simulare qualsiasi macchina non deterministica che lavora in tempo polinomiale. In questo modo posso rappresentare qualsiasi problema $\mathcal{NP}$.



Osservazioni sul tableau:
È una tabella $n^k \times n^k$ dove ogni riga rappresenta "la fotografia" della MdT in un instante di tempo.
Le righe rappresentano le configurazioni di un ramo della computazione di una MdT su un input w (l'intero tableau rappresenta solo un ramo di computazione, non è che ogni riga rappresenta un ramo diverso).

- **Tableau:** tabella che mostra tutte le configurazioni di un singolo ramo.
- **Riga del tableau:** Una configurazione (stato q_i + contenuto nastro + posizione testina).

Dimostrazione. Sia A un generale problema $\mathcal{NP}$ e sia N la MdT non deterministica che decide A in tempo polinomiale.
1. Tableau per MdT non deterministica N
Adesso descrivo il tableau per la MdT non deterministica N su input w:

- **La prima riga:** configurazione iniziale di N.
  (*esempio: la macchina parte nello stato iniziale q_0 con l'input scritto sul nastro e la testina sulla prima cella dell'input*)
- **La seconda riga:** rappresenta lo stato della macchina dopo il primo passo della computazione.

  ⋮
- **La i-esima riga:** rappresenta lo stato della macchina dopo l' $i-1$-esimo passo della computazione.

Assumo che ogni riga (configurazione) abbia come simbolo # nella prima e nell'ultima cella.
Inoltre, un tableau è detto accettante se almeno una delle sue righe è una configurazione accettante; In altre parole, se troviamo un tableau accettante, sappiamo che esiste un ramo della MdT non deterministica che accetta l'input w. Per sapere se la macchina N accetta w, basta verificare se esiste almeno un tableau accettante (perchè per ogni ramo esiste un tableau).
2. Descrizione della funzione di riduzione f
Sia w un'istanza del problema A e sia $f(w)$ un'istanza del problema SAT. Con input w, la funzione di riduzione produce come output la formula booleana $f(w) = \phi$.

1. **Descrivo le variabili booleane della formula ϕ** (delle parti di ϕ?)
   - Sia Q l'insieme degli stati di N
   - Sia Γ l'alfabeto del nastro di N
   - Sia $C = Q \cup \Gamma \cup \{\#\}$
   - Sia $s \in C$ un simbolo che può apparire in una cella del tableau (ogni cella può contenere uno solo dei simboli in C: o uno stato, o un simbolo del nastro, o il simbolo #)

   Per ogni cella $[i, j]$ e per ogni simbolo $s \in C$, ho la variabile booleana $x_{i,j,s}$ dove:
   $$x_{i,j,s} = \begin{cases} 1 & \text{se la cella } [i,j] \text{ contiene il simbolo } s \\ 0 & \text{altrimenti} \end{cases}$$
   Esempio nelle note[a]
2. **Descrivo la formula ϕ**
   Costruisco la formula booleana ϕ tale che un'assegnazione di valori che soddisfa ϕ rappresenta un tableau accettante della macchina N sull'input w, ovvero:
   $$\phi \text{ è soddisfacibile} \iff N \text{ accetta } w$$
   La formula ϕ è composta da 4 sottoformule, congiunte tra loro mediante l'operatore logico AND:
   $$\phi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{accept}$$
3. **Descrivo ogni sottoformula φ di ϕ**
   - φ_{cell}: ogni cella contiene esattamente un simbolo (solo uno).
   - φ_{start}: la prima riga del tableau rappresenta lo stato iniziale di N su w.
   - φ_{move}: ogni riga del tableau segue la precedente secondo le regole di N (assicura che ogni computazione segua legalmente la precedente secondo le regole di N)
   - φ_{accept}: almeno in una cella appare lo stato di accettazione q_{accept}.

Ogni sottoformula diventa vera se è vera anche nel tableau. Quindi la formula $\phi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{accept}$ è vera se e solo se ogni φ_i è vera.
In poche parole, se il tableau è un tableau accettante (N accetta w), allora ϕ è vera/soddisfacibile.

3. Calcolo complessità della funzione di riduzione f

Preso in input qualunque stringa w, voglio dimostrare che posso costruire un'istanza di SAT in tempo polinomiale. Ovvero dato in input w, la funzione di riduzione f restituisce $f(w) = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{accept}$ in tempo polinomiale rispetto alla lunghezza dell'input w. $f(w) = \phi$.

1. **Esamino la dimensione di ϕ, ovvero quante variabili ha (le variabili sono le $x_{i,j,s}$):** La tabella ha dimensione $n^k \times n^k$ e quindi ci sono n^{2k} celle. Per ogni cella ci sono tante variabili quante sono le possibilità di simbolo $s \in C$, quindi la complessità è $|C| \cdot O(n^{2k})$ e quindi $O(n^{2k})$.
2. **Analizzo ogni parte della formula ϕ:**
   (a) φ_{cell}: ogni cella contiene esattamente uno dei simboli di C, $O(n^{2k})$
   (b) φ_{start}: la prima riga è la configurazione iniziale di N su w; ogni riga contiene n^k celle, quindi $O(n^k)$
   (c) φ_{move}: ogni riga segue la precedente secondo le regole di N, $O(n^{2k})$
   (d) φ_{accept}: almeno una cella contiene lo stato di accettazione q_{accept}, controllo tutte le celle nel caso peggiore e quindi $O(n^{2k})$

Pertanto la complessità finale di ϕ è $O(n^{2k}) + O(n^k) + O(n^{2k}) + O(n^{2k}) = O(n^{2k})$ quindi polinomiale.
Quindi la riduzione è polinomiale.

4. La formula ϕ non è in CNF
La formula booleana $f(w)$ ovvero $\phi = \varphi_{cell} \wedge \varphi_{start} \wedge \varphi_{move} \wedge \varphi_{accept}$ non è in CNF. Tuttavia, il problema SAT richiede che l'input sia una formula in CNF. Come posso fare? Applico la trasformazione di Tseitin, che prende in input una formula booleana arbitraria e restituisce la formula in CNF equisoddisfacibile. Tutto ciò in tempo polinomiale.

$$w \xrightarrow[\text{polinomiale}]{\text{Cook–Levin}} \phi \xrightarrow[\text{polinomiale}]{\text{Tseitin}} \phi_{\text{CNF}}$$

Conclusione: Poiché la costruzione della formula ϕ richiede tempo polinomiale rispetto alla lunghezza di w, e ϕ è soddisfacibile se e solo se la macchina N accetta w, otteniamo una formula booleana ϕ che rappresenta l'accettazione di N su w.
Tuttavia, ϕ non è necessariamente in forma normale congiuntiva (CNF), come richiesto da SAT. Per questo, applichiamo la **trasformazione di Tseitin**, che in tempo polinomiale produce una formula ϕ' in CNF *equisoddisfacibile* a ϕ. Quindi esiste una riduzione polinomiale f tale che:

$$w \in A \iff \phi' \in SAT$$

Pertanto, SAT è $\mathcal{NP}$-Hard.
Poiché $SAT \in \mathcal{NP}$ e SAT è $\mathcal{NP}$-Hard, segue che SAT è $\mathcal{NP}$-Completo. □

[a] $x_{0,2,q_1} = 1$ la variabile vale 1 perchè la cella della riga 0 e della colonna 2 contiene il simbolo q_1.

Teorema : 5.2.8

SAT è $\mathcal{NP}$-Completo.

Dimostrazione. Dato che si è dimostrato che:

- $SAT \in \mathcal{NP}$
- SAT è $\mathcal{NP}$-Hard

Segue che SAT è $\mathcal{NP}$-Completo. □

5.2.6 $3 - SAT$

Data una formula booleana ϕ scritta in CNF, in cui ogni clausola contiene esattamente 3 letterali, esiste un qualche assegnamento di valori (0/1) alle variabili che rende l'intera formula vera[11]?

Teorema : $3 - SAT$ Problem

Data ϕ una formula booleana in CNF, dove ogni clausola contiene esattamente 3 letterali, determinare se $\exists$ un qualche assegnamento di valori alle variabili che renda ϕ vera.

[11] la formula booleana restituisce come output 1

$$3-SAT = \{\langle f \rangle \mid f \text{ è una 3cnf-formula booleana soddisfacibile}\}$$
$$3-SAT \text{ è } \mathcal{NP}\text{-Completo}$$

Ragionamento. Per dimostrare che $3-SAT$ è $\mathcal{NP}$-Completo devo:

1. Dimostrare che $3-SAT \in \mathcal{NP}$
2. Dimostrare che $3-SAT$ è $\mathcal{NP}$-Hard

Dimostrazione. **1. Dimostro che $3-SAT \in \mathcal{NP}$:**
La stessa MdT non deterministica che decide SAT in tempo polinomiale decide anche $3-SAT$. Pertanto, $3-SAT \in \mathcal{NP}$.

2. Dimostro che $3-SAT$ è $\mathcal{NP}$-Hard:
Per dimostrare che $3-SAT$ è $\mathcal{NP}$-Hard, effettuo una riduzione polinomiale da SAT (che è $\mathcal{NP}$-Hard) a $3-SAT$, costruendo per ogni formula ϕ CNF arbitraria una formula ϕ' 3-CNF equisoddisfacibile (ϕ soddisfacibile $\iff$ ϕ' soddisfacibile).

$$SAT \leq_p 3-SAT$$

2.1 Costruisco la funzione di riduzione f:
La funzione di riduzione f ha come:

- **input**: istanza di SAT, ovvero ϕ
- **output**: istanza di $3-SAT$, ovvero ϕ'

Ogni clausola u della formula ϕ che non ha lunghezza 3 viene trasformata indipendentemente in una 3-CNF formula. Vedo i diversi casi:

1. **Se la clausola ha lunghezza 1:** un solo letterale v_1. Per trasformarla in 3-CNF, introduco due nuove variabili, y e z e la clausola u diventa:
$$u' = (v_1 \vee y \vee z) \wedge (v_1 \vee y \vee \overline{z}) \wedge (v_1 \vee \overline{y} \vee z) \wedge (v_1 \vee \overline{y} \vee \overline{z})$$
2. **Se la clausola ha lunghezza 2:** due letterali ($v_\vee v_2$). Per trasformarla in 3-CNF, introduco una nuova variabile y e la clausola u diventa:
$$u' = (v_1 \vee v_2 \vee y) \wedge (v_1 \vee v_2 \vee \overline{y})$$
3. **Se la clausola ha lunghezza $k > 3$:** k letterali ($v_1 \vee \cdots \vee v_k$). Per trasformarla in 3-CNF: (Esempio nelle note [a])
   - introduco $k-3$ nuove variabili
   - ottengo $k-2$ clausole in u'

Poiché ogni clausola u è equisoddisfacibile con la sua trasformazione u', segue che la formula completa ϕ è soddisfacibile se e solo se la formula trasformata ϕ' è soddisfacibile:

$$\forall i (u_i \text{ soddisfacibile} \iff u'_i \text{ soddisfacibile})$$
$$\text{implica}$$
$$\underbrace{\phi}_{\text{SAT}} \text{ soddisfacibile} \iff \underbrace{\phi'}_{\text{3-SAT}} \text{ soddisfacibile}$$

2.2 Analizzo complessità della riduzione f:

- Ogni clausola viene trasformata indipendentemente in una piccola formula 3-CNF
- Una clausola di n letterali produce al massimo n clausole da 3 letterali usando nuove variabili

Se la formula originale ϕ ha m clausole, la formula finale ϕ' avrà al massimo $O(m \cdot n)$ clausole, cioè polinomiale rispetto al numero totale di letterali della formula originale. Pertanto la riduzione è polinomiale.

2.3 Dimostro la correttezza della riduzione polinomiale f:
Costruire la funzione di riduzione polinomiale non basta, dobbiamo anche dimostrare che valga:

$$u \text{ è soddisfacibile} \iff u' \text{ è soddisfacibile}$$

- ($\Longrightarrow$):

  Dimostrazione. u è soddisfacibile $\implies$ u' è soddisfacibile
  Per ipotesi $u = (v_1 \vee \cdots \vee v_k)$ è soddisfacibile, quindi $\exists$ un assegnamento t che soddisfa u. Quindi nella clausola u esiste almeno un letterale vero. Chiamo v_j il primo letterale vero in u.
  Adesso costruisco un assegnamento t' per le nuove variabili y_i introdotte in u' $(y_1, ..., y_{k-3})$ in modo che tutte le clausole u' siano vere:

$$y_i = \begin{cases} 1 & \text{se } i = 1, \ldots, j-2, \\ 0 & \text{se } i = j-1, \ldots, k-3. \end{cases}$$

  Pertanto, u' è soddisfacibile. □

  Esempio veloce: Sia una clausola di 6 letterali soddisfatta da t.
  Supponi che l'assegnamento t sia così:
  $v_1 = 0, v_2 = 0, v_3 = 1, v_4 = 0, v_5 = 0, v_6 = 0$
  il letterale vero v_j è v_3 e quindi $j = 3$.
  Adesso costruisco l'assegnamento t' per y_i in modo che tutte le clausole u' siano vere:

  1. *Imposto da y_1 a y_{j-2} uguali a 1.*
  2. *Imposto da y_{j-1} a y_{k-3} uguali a 0.*

  Sapendo che $j = 3$ allora:
  $y_1 = 1$, $y_2 = 0$, $y_3 = 0$
  In questo modo, tutte le clausole u' saranno vere.

- ($\Longleftarrow$):

  Dimostrazione. u è soddisfacibile $\impliedby$ u' è soddisfacibile
  Ragiono per assurdo (nego la tesi): suppongo che u non sia soddisfacibile, ovvero supponiamo per assurdo che t non soddisfi u, cioè che tutti i v_i siano falsi. Per far sì che u' sia soddisfacibile, le nuove variabili dovrebbero essere tutte vere, cioè $y_1, ..., y_{k-3} = 1$. In questo modo però, l'ultima clausola della formula 3-CNF $u' = (v_{k-1} \vee v_k \vee \overline{y_{k-3}})$ non è soddisfatta. Ciò porta ad una contraddizione che rende falsa la mia ipotesi (che u non fosse soddisfacibile); Pertanto, u è soddisfacibile. □

Abbiamo quindi dimostrato che in questa riduzione polinomiale, vale:

$$u \text{ è soddisfacibile} \iff u' \text{ è soddisfacibile.}$$

Poiché ogni clausola di ϕ è equisoddisfacibile con la sua trasformazione in 3-CNF, segue che: ϕ è soddisfacibile $\iff$ ϕ' è soddisfacibile.

Conclusione: Sapendo che SAT è $\mathcal{NP}$-Hard e avendo costruito correttamente una riduzione in tempo polinomiale da SAT a $3-SAT$, segue che $3-SAT$ è $\mathcal{NP}$-Hard. □

[a]Sia $u = (v_1 \vee v_2 \vee v_3 \vee v_4 \vee v_5 \vee v_6)$ una clausola di lunghezza $k = 6$. Per trasformarla in una formula 3-CNF:

(a) introduco $6 - 3 = 3$ nuove variabili y_1, y_2, y_3

(b) dovrò ottenere $6 - 2 = 4$ clausole nella nuova formula 3-CNF, quindi:

i. nella prima clausola metto due letterali e una variabile e ottengo:
$$(v_1 \vee v_2 \vee y_1)$$

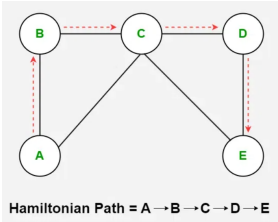
ii. nelle clausole successive alla prima eccetto l'ultima, metto un solo letterale e due variabili (la precedente negata e quella nuova) e ottengo:
- seconda clausola: $(\overline{y_1} \vee v_3 \vee y_2)$
- terza clausola: $(\overline{y_2} \vee v_4 \vee y_3)$

iii. nell'ultima clausola metto due letterali e la variabile precedente negata e ottengo:
$$(v_5 \vee v_6 \vee \overline{y_3})$$

Quindi alla fine u diventa:
$$u' = (v_1 \vee v_2 \vee y_1) \wedge (\overline{y_1} \vee v_3 \vee y_2) \wedge (\overline{y_2} \vee v_4 \vee y_3) \wedge (v_5 \vee v_6 \vee \overline{y_3})$$

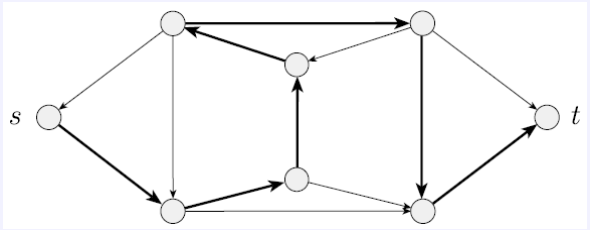
5.2.7 *HAMPATH*

Definizione 5.10 (Cammino Hamiltoniano)**.** Si definisce cammino hamiltoniano in un grafo (orientato o non orientato) un cammino che tocca tutti i nodi del grafo una e una sola volta.



Teorema : *HAMPATH* Problem

$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ grafo orientato con cammino hamiltoniano da } s \text{ a } t\}$
$HAMPATH \in \mathcal{NP}$



Ragionamento. per dimostrarlo, costruisco una MdT non deterministica che decide $HAMPATH$ in tempo polinomiale (senza certificato).

Dimostrazione. Sia H una MdT non deterministica che decide $HAMPATH$ in tempo polinomiale come segue. $\langle G, s, t\rangle$ è la codifica binaria del grafo orientato G e dei nodi s, t dove $s \neq t$ (codifica come lista di adiacenza o matrice di adiacenza).

H ha 3 nastri:

- **Nastro 1:** codifica del grafo in input
- **Nastro 2:** contiene sequenza di nodi generate $p_1, \cdots, p_n$ dove n rappresenza il numero di nodi di G
- **Nastro 3:** nastro di lavoro; usato per controllare se la sequenza è hamiltoniana (che non ci siano nodi ripetuti)

H su input $\langle G, s, t\rangle$:

1. H genera una sequenza di nodi e la scrive sul nastro 2.
2. H usa il nastro 3 per controllare se la sequenza (del nastro 2) è hamiltoniana; se non lo è, rifiuta.
3. H usa il nastro 2 per controllare se $p_1 = s \wedge p_n = t$; se il controllo fallisce, rifiuta.
4. H usa il nastro 2 per controllare, se, per la sequenza fornita, gli archi esistono davvero in G; Per ogni coppia di nodi consecutiva (p_i, p_{i+1}) della sequenza, verifica che esista l'arco corrispondente in G; se per qualche i tale arco non esiste, rifiuta. Altrimenti H accetta l'input $\langle G, s, t\rangle$.

Analizzo la complessità:

1. Per il passo 1, scrivere un numero binario n costa $O(\log_2 n)$. Quindi scrivere una sequenza di n nodi costa $O(n \cdot \log_2 n)$
2. Per il passo 2, viene copiata la sequenza dal nastro 2 al nastro 3 che costa $O(n \cdot \log_2 n)$, poi controlla che nessun nodo sia ripetuto, ovvero controllare che tutti i nodi siano distinti $O(n^2 \log_2 n)$, strategia spiegata nelle note[a].
3. Per il passo 3, controlla se $p_1 = s \wedge p_n = t$, costa $\mathrm{O}(n \log_2 n)$; strategia spiegata nelle note[b].
4. Per il passo 4, controlla se gli archi esistono, costa $O(n^2 \log_2 n)$; strategia spiegata nelle note[c].

Conclusione: La complessità finale della MdT H è $tC_H(n) = O(n \log_2 n) + O(n^2 \log_2 n) + O(n \log_2 n) + O(n^2 \log_2 n) = O(n^2 \log_2 n)$ dove n rappresenta la lunghezza dell'input. Quindi una complessità polinomiale. Ho costruito una MdT H non deterministica che decide $HAMPATH$ in tempo polinomiale, pertanto $HAMPATH \in \mathcal{NP}$. □

Importante: H è una MdT non deterministica e quindi questo vuol dire che H esegue i passi dal 1 al 4 su sequenze diverse in modo parallelo (rami). Ogni sequenza corrisponde a un ramo di computazione, e l'input è accettato se almeno un ramo accetta (termina in uno stato accettante).

Spiegazione dettagliata della MdT non deterministica H.

H su input $\langle G, s, t\rangle$:

1. H genera una sequenza di nodi e la scrive sul nastro 2
2. H usa il nastro 3 per controllare se la sequenza (del nastro 2) è hamiltoniana:
   - Se sì, vai al passo 3.
   - Se no, allora rifiuta la sequenza (questo ramo)
3. H usa il nastro 2 per controllare che $p_1 = s$ e che $p_n = t$

- Se sì, vai al passo 4.
- Se no, allora rifiuta la sequenza (questo ramo)

4. H usa il nastro 2 per controllare se per la sequenza fornita, gli archi esistono davvero in G; Per ogni coppia di nodi consecutiva (p_i, p_{i+1}) nella sequenza, verifica che esista l'arco corrispondente in G:
   - Se sì, allora accetta l'input $\langle G, s, t\rangle$
   - Se no, allora rifiuta (questo ramo)

---

[a] Ripeto il ciclo per gli n nodi della sequenza:

(a) Copiare la codifica del i-esimo nodo dal nastro 2 al nastro 3, costo $O(\log_2 n)$

(b) Riportare la testina del nastro 2 all'inizio della sequenza, costo $O(n \log_2 n)$

(c) Scorrere la sequenza degli n nodi confrontandoli con il nodo scritto sul nastro 3, costo $n \cdot O(\log_2 n)$ ovvero $O(n \log_2 n)$

Dato che i passi a,b,c sono in sequenza allora ho che la complessità è $O(\log_2 n) + O(\log_2 n) + O(n \log_2 n)$ ovvero $O(n \log_2 n)$. Ma il ciclo si deve ripetere per gli n nodi della sequenza, quindi la complessità finale sarà $n \cdot O(n \log_2 n) = O(n^2 \log_2 n)$.

[b] Prima di tutto, so che la testina del nastro 2 è alla fine, e quindi:

(a) Copio il valore di s dal nastro 1 al nastro 3. La testina del nastro 1 deve fare al massimo $O(n \log_2 n)$ spostamenti per arrivare al valore di s, poi copiarlo sul nastro 3 costa $O(\log_2 n)$. Quindi $O(n \log_2 n) + O(\log_2 n) = O(n \log_2 n)$

(b) Copio il valore di t dal nastro 1 al nastro 3, stesso costo $O(n \log_2 n)$

(c) Confronto p_n del nastro 2 con t sul nastro 3 muovendo entrambe le testine verso sinistra, costo $O(\log_2 n)$

(d) Muovo la testina del nastro 2 fino all'inizio della sequenza $O(n \log_2 n)$

(e) Confronto p_1 del nastro 2 con s sul nastro 3, costo $O(\log_2 n)$

Dato che i passi sono sequenziali, il costo complessivo è: $O(n \log_2 n) + O(n \log_2 n) + O(\log_2 n) + O(n \log_2 n) + O(\log_2 n) = O(n \log_2 n)$

[c] Ripeto il ciclo per ogni coppia (p_i, p_{i+1}) quindi per $i = 1, \cdots, n-1$:

(a) Posiziona la testina del nastro 2 sull'i-esimo nodo, ovvero su p_i, costo $O(n \log_2 n)$

(b) Posiziona la testina del nastro 1 sull'i-esimo nodo, costo $O(n \log_2 n)$

(c) Confronta l'i-esimo nodo nei due nastri per vedere se si tratta dello stesso, costo $O(\log_2 n)$

(d) Muovi la testina dal nodo p_i al nodo p_{i+1} sul 2 nastro, costo $O(\log_2 n)$

(e) Confronta i nodi della lista di adiacenza dell'i-esimo nodo sul nastro 1 con il nodo p_{i+1} del nastro 2; spostamento e lettura per n nodi $O(n \log_2 n)$ quindi sul nastro 1 costo $O(n \log_2 n)$ con confronto del nastro 2, $O(\log_2 n)$; quindi $O(n \log_2 n) + O(\log_2 n) = O(n \log_2 n)$.

Dato che i passi a-b-c-d-e sono in sequenza allora la complessità è $O(n \log_2 n) + O(n \log_2 n) + O(\log_2 n) + O(\log_2 n) + O(n \log_2 n) = O(n \log_2 n)$. Poi dato che il ciclo va ripetuto per $n-1$ volte allora la complessità finale sarà $(n-1) \cdot (O(n \log_2 n)) = O(n) \cdot (O(n \log_2 n)) = O(n^2 \log_2 n)$

Teorema : $HAMPATH$ Problem

$HAMPATH = \{\langle G, s, t\rangle \mid G \text{ grafo orientato con cammino hamiltoniano da } s \text{ a } t\}$
$HAMPATH$ è $\mathcal{NP}$-Completo

Dimostrazione. **1. Dimostrare che $HAMPATH \in \mathcal{NP}$**
Lo abbiamo già dimostrato sopra.

2. Dimostrare che $HAMPATH$ è $\mathcal{NP}$-Hard

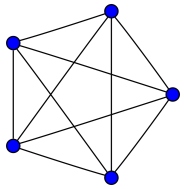
□

5.2.8 $HAMCYCLE$

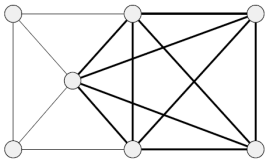
vedi pg. 476 di SudKamp

5.2.9 $CLIQUE$

Definizione 5.11 (Grafo completo)**.** Un grafo completo è un grafo non orientato nel quale ogni nodo è collegato a tutti i vertici rimanenti.



Un clique è un sottografo completo di un grafo non orientato.



Un k-clique è un clique che contiene k nodi. (La figura sopra è un 5-clique)

Teorema : $CLIQUE$ Problem

Dato G grafo non orientato, esiste un sottografo completo di G con k vertici?

$CLIQUE = \{\langle G, k\rangle \mid G \text{ grafo non orientato con } k\text{-clique}\}$
$CLIQUE$ è $\mathcal{NP}$-Completo

Dimostrazione. **1. Dimostrare che $CLIQUE \in \mathcal{NP}$**
Costruisco una MdT N non deterministica che decide $CLIQUE$ in tempo polinomiale come segue. Sia $G = (V, E)$ un grafo non orientato. L'input di N è la codifica $\langle G, k\rangle$, dove $G = (V, E)$ è un grafo non orientato e $k \in \mathbb{N}$.

N su input $\langle G, k\rangle$:

1. N indovina[a] (non deterministicamente) un sottoinsieme C dei vertici di G tale che $|C| = k$
2. N verifica se i k vertici scelti formano una clique:
   - Se sì, N termina in uno stato accettante $\langle G, k\rangle \in CLIQUE$
   - Se no, N termina in uno stato di rifiuto $\langle G, k\rangle \notin CLIQUE$

La complessità di N è polinomiale, pertanto $CLIQUE \in \mathcal{NP}$.

2. Dimostrare che $CLIQUE$ è $\mathcal{NP}$-Hard

Per dimostrare che $CLIQUE$ è $\mathcal{NP}$-Hard, effettuo una riduzione polinomiale da $3-SAT$ (che è $\mathcal{NP}$-Hard) a $CLIQUE$, costruendo per ogni formula ϕ 3-CNF arbitraria un grafo non orientato; faccio vedere che la riduzione funziona perchè ϕ è soddisfacibile $\iff$ G ha un $k-clique$.
Effettuo una riduzione da $3-SAT$ a $CLIQUE$:

$$3-SAT \leq_p CLIQUE$$

- Sia ϕ la 3-CNF formula con k clausole (3 letterali per clausola):

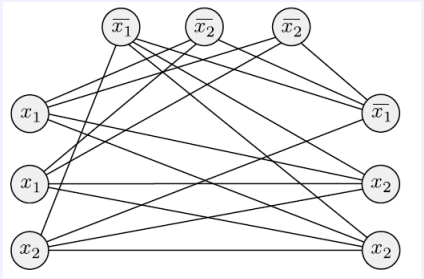
$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_k \vee b_k \vee c_k)$$

2.1 Costruisco la funzione di riduzione f

f riceve come input ϕ e genera come output $\langle G, k\rangle$, dove G è un grafo non orientato definito come segue:

1. **Costruzione dei nodi di G**
   - I nodi di G sono organizzati in k gruppi (k è il numero di clausole nella formula) di 3 nodi ciascuno
   - Ogni gruppo t_i corrisponde quindi ad una clausola della formila, $t_i = (a_i \vee b_i \vee c_i)$
   - Ogni nodo corrisponde ad un letterale della formula ϕ, quindi in totale G ha $3k$ nodi
2. **Costruzione degli archi di G**
   Mettiamo archi tra tutte le coppie di nodi, tranne per:
   - Nodi dello stesso gruppo (niente archi tra letterali della stessa clausola)
   - Nodi che rappresentano letterali opposti (niente arco tra a_i e $\overline{a_i}$)

Esempio illustrativo:
Sia $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$. La funzione di riduzione f, con input ϕ, produce il seguente grafo:



ϕ è soddisfacibile $\iff$ G ha un $k-clique$

2.2 Analizzo complessità della funzione di riduzione f
Voglio assicurarmi che f sia una riduzione polinomiale. Ricordiamo che: nota[b].
Nel caso peggiore ho complessità $O(n^2)$ dove n è la lunghezza dell'input, e quindi polinomiale.

2.3 Dimostrazione della correttezza della funzione di riduzione f
Costruire la funzione di riduzione polinomiale non basta, dobbiamo anche dimostrare che valga:

$$\phi \text{ è soddisfacibile} \iff \text{G ha un } k - \textit{clique}$$

- **($\Longrightarrow$):**

  Dimostrazione. ϕ è soddisfacibile $\implies$ G ha un $k - \textit{clique}$
  Per ipotesi, ϕ è soddisfacibile e quindi esiste un'assegnazione di valori alle variabili che rende vera ogni clausola. In ogni clausola di ϕ, almeno un letterale risulta vero.
  In ogni gruppo t_i di G seleziono il nodo corrispondente a un letterale vero (che rende vera la clausola). Se ci sono più letterali veri nella stessa clausola, scelgo arbitrariamente uno tra questi.
  I nodi appena selezionati formano una $k-\textit{clique}$: il numero di nodi selezionati è k perchè ne abbiamo scelto uno per ciascuno dei k gruppi. Inoltre, ogni coppia di nodi selezionati è collegata da un arco, perché nessuna coppia rientra nelle eccezioni descritte in precedenza (stesso gruppo o letterali opposti).
  Pertanto, G ha un $k - \textit{clique}$. □

- **($\Longleftarrow$):**

  Dimostrazione. ϕ è soddisfacibile $\impliedby$ G ha un $k - \textit{clique}$
  Per ipotesi, G ha una $k - \textit{clique}$ e nessuno dei k nodi della clique è nello stesso gruppo, perchè ricordiamoci che i nodi dello stesso gruppo t_i non sono collegati tra loro. Quindi, ognuno dei k gruppi contiene esattamente un nodo della clique.
  Adesso, assegno il valore 1 ($TRUE$) ai letterali corrispondenti ai nodi della clique. Questo è sempre possibile senza creare contraddizioni, perchè due letterali opposti (x_i e $\overline{x_i}$) non possono entrambi far parte della clique. Quindi la clique non può contenere contraddizioni, e quindi possiamo assegnare i valori di verità in modo coerente. Questa assegnazione soddisfa ϕ, perchè ogni gruppo t_i di G contiene un nodo della clique, e quindi ogni clausola contiene almeno un letterale vero (che rende vera la clausola e quindi l'intera formula).
  Pertanto, ϕ è soddisfacibile. □

Abbiamo quindi dimostrato che in questa riduzione polinomiale, vale: ϕ è soddisfacibile $\iff$ G ha un $k - \textit{clique}$.

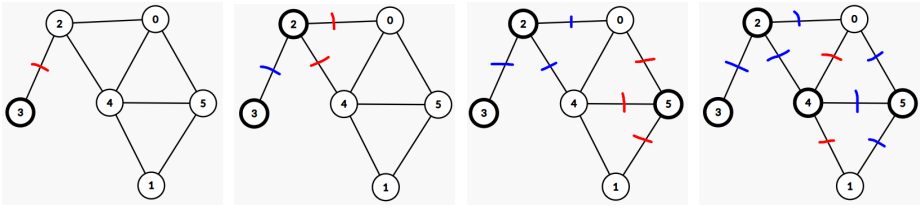
Conclusione: Ho costruito correttamente una riduzione polinomiale da $3 - SAT$ a $CLIQUE$, pertanto $CLIQUE$ è $\mathcal{NP}$-Hard. Essendo $CLIQUE \in \mathcal{NP}$ e $\mathcal{NP}$-Hard, segue che $CLIQUE$ è $\mathcal{NP}$-Completo. □

[a] "indovinare" una soluzione, cioè provare tutte le possibilità in parallelo, in senso teorico; Esistono tantissimi sottoinsiemi di k vertici, e dato che la MdT è non deterministica, è possibile provarli tutti in parallelo.

[b] *quando analizziamo algoritmi su grafi, possiamo esprimere il tempo di esecuzione in base al numero di nodi del grafo, invece che della dimensione della rappresentazione del grafo (cioè della lunghezza dell'input codificato). Esempio: in un problema su grafi, l'algoritmo impiega tempo $O(v^2)$ dove v è il numero di nodi del grafo.*

5.2.10 *VERTEX COVER*

Sia $G = (V, E)$ un grafo non orientato. Scelto un sottoinsieme di vertici C di V, si dice "vertex cover" se i nodi di C "coprono/toccano" tutti gli archi del grafo.



(a) Passo 1 (b) Passo 2 (c) Passo 3 (d) Passo 4

Figura 1: L'insieme $C = \{2, 3, 4, 5\}$ è vertex cover

Teorema : *VERTEX COVER* Problem

Dato G grafo non orientato e $k \in \mathbb{N}$, esiste un vertex cover C di G con k vertici?

$$VC = \{\langle G, k\rangle \mid G \text{ grafo non orientato con un vertex cover di } k \text{ nodi}\}$$
$$VC \text{ è } \mathcal{NP}\text{-Completo}$$

Dimostrazione. **1. Dimostrare che $VC \in \mathcal{NP}$**
Costruisco una MdT N non deterministica che decide VC in tempo polinomiale come segue. Sia $G = (V, E)$ un grafo non orientato. L'input di N è la codifica $\langle G, k\rangle$, dove $G = (V, E)$ è un grafo non orientato e $k \in \mathbb{N}$.

N su input $\langle G, k\rangle$:

1. Non deterministicamente sceglie un insieme $C \subseteq V$ di k vertici.
2. Per ogni arco $\{x, y\} \in E$, verifica che almeno uno tra x e y appartenga a C. (Per ogni singolo arco, devo verificare se uno dei due vertici x, y sono in C)
3. Se tutti gli archi sono coperti da C, accetta l'input; altrimenti rifiuta.

La scelta non deterministica del sottoinsieme C permette di "provare tutte le possibili soluzioni" contemporaneamente. La verifica che tutti gli archi siano coperti richiede al massimo $|E| \cdot k$ operazioni ($|E|$ numero totale archi del grafo), quindi è tempo polinomiale rispetto alla dimensione dell'input. Ho costruito una MdT non deterministica che decide VC in tempo polinomiale, pertanto $VC \in \mathcal{NP}$.

2. Dimostrare che VC è $\mathcal{NP}$-Hard
Effetto una riduzione polinomiale da $3 - SAT$(che è $\mathcal{NP}$-Hard) a VC, costruendo per ogni formula ϕ 3-CNF arbitraria un grafo non orientato; faccio vedere che la riduzione funziona perchè ϕ è soddisfacibile $\iff$ G ha un vertex cover di k nodi.

$$3 - SAT \leq_p VC$$

- Sia ϕ la 3-CNF formula con l clausole (3 letterali per clausola):

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \cdots \wedge (a_l \vee b_l \vee c_l)$$

2.1 Costruisco la funzione di riduzione f
f riceve come input ϕ e genera come output $\langle G, k \rangle$, dove G è un grafo non orientato definito come segue:

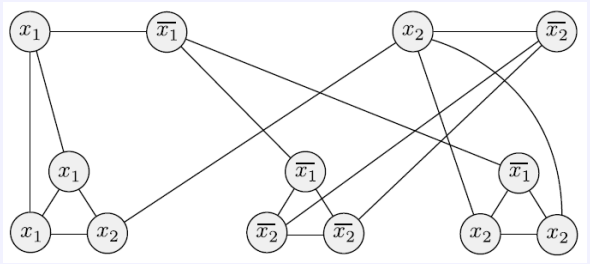
1. **Costruzione dei nodi di G**
   (a) Scrivo un nodo per ogni letterale di ogni clausola ($3l$ nodi)
   (b) Aggiungo altre m coppie di nodi (m numero delle variabili nella formula) che rappresentano le variabili positive e negate ($2m$ nodi)

   Il grafo avrà un totale di $3l + 2m$ nodi.
2. **Costruzione degli archi di G**
   Mettiamo archi tra:
   (a) Nodi che rappresentano letterali della stessa clausola (formando i triangoli)
   (b) Nodi che rappresentano letterali opposti (tra a_i e $\overline{a_i}$, le coppie di m nodi)
   (c) Nodi che rappresentano lo stesso letterale tra i triangoli e le coppie di m nodi

Un vertex cover di G possiede almeno $m + 2l$ vertici, perchè:

- Per ogni triangolo di clausola devo scegliere almeno 2 nodi nel vertex cover per coprire tutti gli archi del triangolo
- Per ogni coppia di nodi variabile/negazione devo scegliere 1 nodo per coprire l'arco che li collega

Esempio illustrativo:
Sia $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$. La funzione di riduzione f, con input ϕ, produce il seguente grafo:



ϕ è soddisfacibile $\iff$ G ha un vertex cover di k nodi

2.2. Analizzo complessità della funzione di riduzione f
Per costruire il grafo si richiede tempo $O(m + l)$, quindi la riduzione è polinomiale.
2.3. Dimostrazione della correttezza della funzione di riduzione f
Costruire la funzione di riduzione polinomiale non basta, dobbiamo anche dimostrare che valga:

$$\phi \text{ è soddisfacibile} \iff \text{G ha un vertex cover di } k \text{ nodi}$$

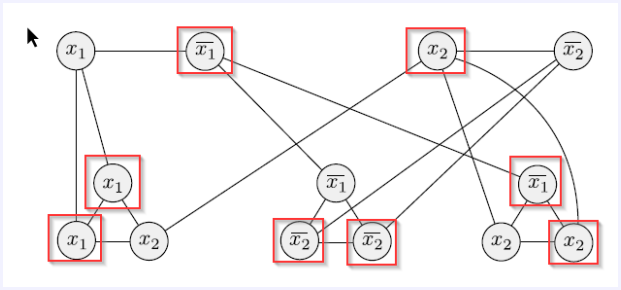
- **($\Longrightarrow$):**

  Dimostrazione. ϕ è soddisfacibile $\implies$ G ha un vertex cover di k nodi
  Per ipotesi, ϕ è soddisfacibile e quindi esiste un'assegnazione di valori alle variabili che rende vera ogni clausola. In ogni clausola di ϕ, almeno un letterale risulta vero.
  Adesso faccio vedere che, essendo ϕ soddifacibile, posso costruire il vertex cover di G come segue:

1. Per prima cosa, per ogni coppia di nodi variabile/negazione scelgo 1 nodo della coppia:
   - Se nella nostra assegnazione $x_1 = true$, allora metto nel vertex cover il nodo che rappresenta x_i
   - Se invece nella nostra assegnazione $x_i = false$, allora metto nel vc il nodo $\overline{x_i}$

   Quindi, alla fine scelgo m nodi (m è il numero di variabili di ϕ);
2. Per ogni clausola (i triangoli), scelgo uno dei letterali che è vero (almeno uno deve esserlo, visto che la formula è soddisfacibile) e metto nel vc gli altri due nodi.
   Quindi, alla fine scelgo $2l$ nodi (l è il numero di clausole di ϕ)

Esempio illustrativo:
Sia t l'assegnamento che rende ϕ soddisfacibile:
$x_1 = false, x_2 = true$



Quindi, l'insieme di vertici così definito è un vertex cover.
Pertanto, G ha un vertex cover di $k = m + 2l$ nodi. □

- **($\Longleftarrow$):**

  Dimostrazione. ϕ è soddisfacibile $\Longleftarrow$ G ha un vertex cover di k nodi
  Per ipotesi, G ha un vertex cover di k nodi, mostro che ϕ è soddisfacibile costruendo un'assegnazione di valori che la renda vera:
  Osservo i nodi che appartengono al vertex cover e assegno $true$ ai letterali corrispondenti (se nel vc c'è x_i, allora assegno $x_i = true$; se invece nel vc c'è $\overline{x_i}$, allora assegno $\overline{x_i} = false$).
  Con questo assegnamento, ogni clausola è soddisfatta dal letterale corrispondente al nodo che non è nel vertex cover di ogni triangolo.
  Pertanto, ϕ è soddisfacibile. □

Abbiamo quindi dimostrato che in questa riduzione polinomiale, vale: ϕ è soddisfacibile $\Longleftrightarrow$ G ha un vertex cover di k nodi.

Conclusione: Ho costruito correttamente una riduzione polinomiale da $3-SAT$ a VC, pertanto VC è $\mathcal{NP}$-Hard. Essendo $VC \in \mathcal{NP}$ e $\mathcal{NP}$-Hard, segue che VC è $\mathcal{NP}$-Completo. □

5.3 Come capire quali problemi sono riducibili ad altri (trucchi)

5.4 $\mathcal{P}$ vs $\mathcal{NP}$

La differenza sostanziale tra $\mathcal{P}$ e $\mathcal{NP}$ è:

- la classe $\mathcal{P}$ è l'insieme dei problemi *risolvibili* da una MdT deterministica in tempo polinomiale.
- la classe $\mathcal{NP}$ è l'insieme dei problemi *risolvibili* da una MdT non deterministica in tempo polinomiale (senza certificato); oppure, *verificabili* da una MdT deterministica in tempo polinomiale, dato un certificato.

Ovvero, se $C \in \mathcal{NP}$:

- $\nexists$ un algoritmo deterministico che decide C in tempo polinomiale
- $\exists$ un algoritmo deterministico che decide C in tempo esponenziale (o la complessità è piu grande di exp?)
- $\exists$ un algoritmo deterministico che verifica un certificato per C in tempo polinomiale
- $\exists$ un algoritmo non deterministico che decide C in tempo polinomiale (senza un certificato)

5.5 $\mathcal{P} \neq \mathcal{NP}$

5.5.1 $\mathcal{P} \stackrel{?}{=} \mathcal{NP}$

6 Complessità Spaziale