SAPIENZA
UNIVERSITÀ DI ROMA

"Sapienza" Università di Roma
Ingegneria dell'Informazione,
Informatica e Statistica
Dipartimento di Informatica

Automi, Calcolabilità e Complessità

Appunti integrati con il libro "Introduzione alla teoria della computazione", M. Sipser

Autore
Simone Bianco

13 luglio 2025

Indice

Informazioni e Contatti

Appunti e riassunti personali raccolti in ambito del corso di *Automi, Calcolabilità e Complessità* offerto dal corso di laurea in Informatica dell'Università degli Studi di Roma "La Sapienza".

Ulteriori informazioni ed appunti possono essere trovati al seguente link: **https://github.com/Exyss/university-notes**. Chiunque si senta libero di segnalare incorrettezze, migliorie o richieste tramite il sistema di Issues fornito da GitHub stesso o contattando in privato l'autore :

- Email: **bianco.simone@outlook.it**
- LinkedIn: **Simone Bianco**

Gli appunti sono in continuo aggiornamento, pertanto, previa segnalazione, si prega di controllare se le modifiche siano già state apportate nella versione più recente.

Prerequisiti consigliati per lo studio:

Apprendimento del materiale relativo al corso *Progettazione di Algoritmi*.

Licence:

1 Linguaggi regolari

1.1 Linguaggi

Definizione 1.1: Alfabeto

Definiamo come **alfabeto** un insieme finito di elementi detti **simboli**

Esempio:

- L'insieme $\Sigma = \{0, 1, \mathrm{x}, \mathrm{y}, \mathrm{z}\}$ è un alfabeto
- L'insieme $\Sigma = \{0, 1\}$ è un alfabeto. In particolare, tale alfabeto viene detto **alfabeto binario**

Definizione 1.2: Stringa

Data una sequenza di simboli $w_1, \ldots, w_n \in \Sigma$, definiamo:

$$w := w_1 \ldots w_n$$

come **stringa (o parola) di** Σ

Esempio:

- Dato l'alfabeto $\Sigma = \{0, 1, \mathrm{x}, \mathrm{y}, \mathrm{z}\}$, una stringa di Σ è 0x1yyy0

Definizione 1.3: Linguaggio

Dato un alfabeto Σ, definiamo come **linguaggio di** Σ, indicato come Σ^*, l'insieme delle stringhe di Σ

Definizione 1.4: Lunghezza di una stringa

Data una stringa $w \in \Sigma^*$, definiamo la **lunghezza di** w, indicata come $|w|$, come il numero di simboli presenti in w

Definizione 1.5: Concatenazione

Data la stringa $x := x_1 \dots x_n \in \Sigma^*$ e la stringa $y := y_1 \dots y_m \in \Sigma^*$, definiamo come **concatenazione di** x **con** y la seguente operazione:

$$xy = x_1 \dots x_n y_1 \dots y_n$$

Proposizione 1.1: Stringa vuota

Indichiamo con ε la **stringa vuota**, ossia l'unica stringa tale che:

- $|\varepsilon| = 0$
- $\forall w \in \Sigma^* \quad w \cdot \varepsilon = \varepsilon \cdot w = w$
- $\Sigma^* \neq \varnothing \implies \varepsilon \in \Sigma^*$

Definizione 1.6: Conteggio

Data una stringa $w \in \Sigma^*$ e un simbolo $a \in \Sigma$ definiamo il **conteggio di** a **in** w, indicato come $|w|_a$, il numero di simboli uguali ad a presenti in w

Esempio:

- Data la stringa $w := \texttt{010101000} \in \{0,1\}^*$, si ha che $|w|_0 = 6$ e $|w|_1 = 3$

Definizione 1.7: Stringa rovesciata

Data una stringa $w = a_1 \dots a_n \in \Sigma^*$, dove $a_1 \dots a_n \in \Sigma$, definiamo la sua **stringa rovesciata**, indicata con w^R, come $w^R = a_n \dots a_1$.

Esempio:

- Data la stringa $w := \texttt{abcdefg} \in \Sigma^*$, si ha che $w^R = \texttt{gfedcba}$

Definizione 1.8: Potenza

Data la stringa $w \in \Sigma^*$ e dato $n \in \mathbb{N}$, definiamo come **potenza** la seguente operazione:

$$w^n = \begin{cases} \varepsilon & \text{se } n = 0 \\ ww^{n-1} & \text{se } n > 0 \end{cases}$$

Proposizione 1.2: Operazioni sui linguaggi

Dati i linguaggi $L, L_1, L_2 \subseteq \Sigma^*$, definiamo le seguenti operazioni:

- **Operatore unione**:

$$L_1 \cup L_2 = \{w \in \Sigma^* \mid w \in L_1 \vee w \in L_2\}$$

- **Operatore intersezione**:

$$L_1 \cap L_2 = \{w \in \Sigma^* \mid w \in L_1 \wedge w \in L_2\}$$

- **Operatore complemento**:

$$\overline{L} = \{w \in \Sigma^* \mid w \notin L\}$$

- **Operatore concatenazione**:

$$L_1 \circ L_2 = \{xy \in \Sigma^* \mid x \in L_1, y \in L_2\}$$

- **Operatore potenza**:

$$L^n = \begin{cases} \{\varepsilon\} & \text{se } n = 0 \\ L \circ L^{n-1} & \text{se } n > 0 \end{cases}$$

- **Operatore star di Kleene**:

$$L^* = \{w_1 \dots w_k \in \Sigma^* \mid k \geq 0, \forall i \in [1,k] \;\; w_i \in L\} = \bigcup_{n \geq 0} L^n$$

- **Operatore plus di Kleene**:

$$L^+ = \{w_1 \dots w_k \in \Sigma^* \mid k \geq 1, \forall i \in [1,k] \;\; w_i \in L\} = \bigcup_{n \geq 1} L^n = L \circ L^*$$

Teorema 1.1: Leggi di DeMorgan

Dati due linguaggi L_1 e L_2, si ha che:

$$L_1 \cup L_2 = \overline{\overline{L_1} \cap \overline{L_2}}$$

$$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$$

(*dimostrazione omessa*)

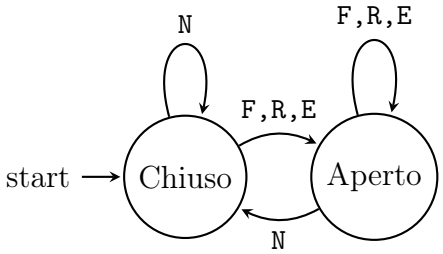
1.2 Determinismo

Definizione 1.9: Automa

Un **automa** è un meccanismo di controllo (o macchina) progettato per seguire automaticamente una sequenza di operazioni o rispondere a istruzioni predeterminate, mantenendo informazioni relative allo **stato** attuale dell'automa stesso ed agendo di conseguenza, **passando da uno stato all'altro**.

Esempio:

- Un sensore che apre e chiude una porta può essere descritto tramite il seguente automa, dove `Chiuso` e `Aperto` sono gli stati dell'automa e `N`, `F`, `R` e `E` sono le operazioni di transizione tra i due stati indicanti rispettivamente:
  - `N`: il sensore non rileva alcuna persona da entrambi i lati della porta
  - `F`: il sensore rileva qualcuno nel lato frontale della porta
  - `R`: il sensore rileva qualcuno nel lato retrostante della porta
  - `E`: il sensore rileva qualcuno da entrambi i lati della porta



- L'automa appena descritto è in grado di interpretare una **stringa in input** che ne descriva la sequenza di operazioni da svolgere (es: la stringa `NFNNNFRR` terminerà l'esecuzione dell'automa sullo stato `Aperto`)

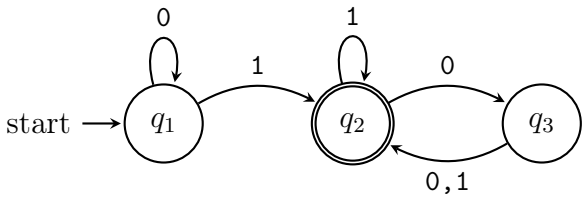
Definizione 1.10: Deterministic Finite Automaton (DFA)

Un **Deterministic Finite Automaton (DFA)** (o *Automa Deterministico a Stati Finiti*) è una quintupla $(Q, \Sigma, \delta, q_0, F)$ dove:

- Q è l'**insieme finito degli stati** dell'automa
- Σ è l'**alfabeto** dell'automa
- $\delta : Q \times \Sigma \to Q$ è la **funzione di transizione degli stati** dell'automa
- $q_0 \in Q$ è lo **stato iniziale** dell'automa
- $F \subseteq Q$ è l'**insieme degli stati accettanti** dell'automa, ossia l'insieme degli stati su cui, a seguito della lettura di una stringa in input, l'automa accetta la corretta terminazione

Esempio:

- Consideriamo il seguente DFA



  dove:

  - $Q = \{q_1, q_2, q_3\}$ è l'insieme degli stati dell'automa
  - $\Sigma = \{0, 1\}$ è l'alfabeto dell'automa
  - $\delta : Q \times \Sigma \to Q$ definita come

δ	q_1	q_2	q_3
0	q_1	q_3	q_2
1	q_2	q_2	q_2

    è la funzione di transizione degli stati dell'automa
  - q_1 è lo stato iniziale dell'automa
  - $F = \{q_2\}$ è l'insieme degli stati accettanti

Definizione 1.11: Funzione di transizione estesa

Sia $D := (Q, \Sigma, \delta, q_0, F)$ un DFA. Definiamo $\delta^* : Q \times \Sigma^* \to Q$ come **funzione di transizione estesa di** D la funzione definita ricorsivamente come:

$$\begin{cases} \delta^*(q, \varepsilon) = \delta(q, \varepsilon) = q \\ \delta^*(q, aw) = \delta^*(\delta(q, a), w), \text{ dove } a \in \Sigma, w \in \Sigma^* \end{cases}$$

Proposizione 1.3: Stringa accettata in un DFA

Sia $D := (Q, \Sigma, \delta, q_0, F)$ un DFA. Data una stringa $w \in \Sigma^*$, diciamo che w è **accettata da** D se $\delta^*(q_0, w) \in F$, ossia l'interpretazione di tale stringa **termina su uno stato accettante**

Esempio:

- Consideriamo ancora il DFA dell'esempio precedente.
- La stringa 0101 è accettata da tale DFA, poiché:

$$\delta^*(q_1, 0101) = \delta^*(\delta(q_1, 0), 101) = \delta^*(q_2, 101) = \delta^*(\delta(q_2, 1), 01) = \delta^*(q_2, 01) =$$

$$= \delta^*(\delta(q_2, 0), 1) = \delta^*(q_3, 1) = \delta^*(\delta(q_3, 1), \varepsilon) = \delta^*(q_2, \varepsilon) = q_2 \in F$$

- La stringa 1010, invece, non è accettata dal DFA, poiché:

$$\delta^*(q_1, 1010) = \delta^*(q_2, 010) = \delta^*(q_3, 10) = \delta^*(q_2, 0) = \delta^*(q_3, \varepsilon) = q_3 \notin F$$

Definizione 1.12: Linguaggio di un automa

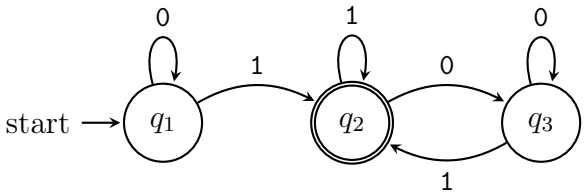
Sia A un automa. Definiamo come **linguaggio di** A, indicato come $L(A)$, l'insieme di stringhe accettate da A

$$L(A) = \{w \in \Sigma^* \mid A \text{ accetta } w\}$$

Inoltre, diciamo che A **riconosce** $L(A)$

Esempi:

1. - Consideriamo il seguente DFA D



   - Il linguaggio riconosciuto da tale DFA corrisponde a

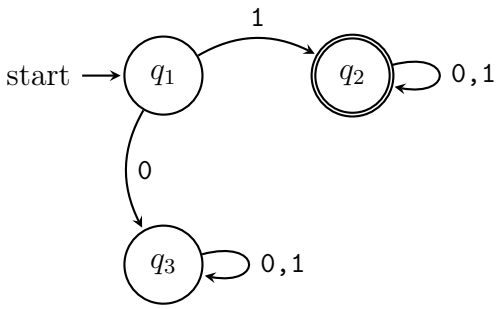
$$L(D) = \{x \in \{0,1\}^* \mid x := y1, \exists y \in \{0,1\}^*\}$$

   ossia al linguaggio composto da tutte le stringhe terminanti con 1

2. - Consideriamo il seguente linguaggio

$$L = \{x \in \{0,1\}^* \mid 1y, \exists y \in \{0,1\}^*\}$$

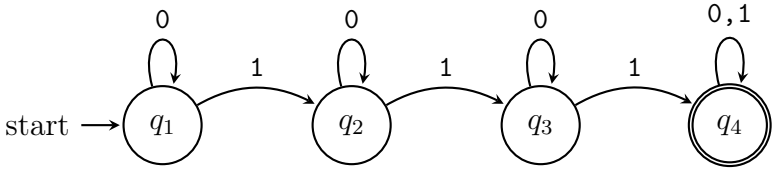
   - Un DFA in grado di riconoscere tale linguaggio corrisponde a



3. • Consideriamo il seguente linguaggio

$$L = \{w \in \{0,1\}^* \mid |w|_1 \geq 3\}$$

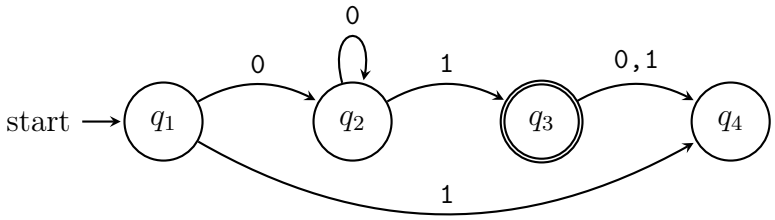
• Un DFA in grado di riconoscere tale linguaggio corrisponde a



4. • Consideriamo il seguente linguaggio

$$L = \{w \in \{0,1\}^* \mid w := 0^n 1, n \in \mathbb{N} - \{0\}\}$$

• Un DFA in grado di riconoscere tale linguaggio corrisponde a



Definizione 1.13: Configurazione di un DFA

Sia $D := (Q, \Sigma, \delta, q_0, F)$ un DFA. Definiamo la coppia $(q, w) \in Q \times \Sigma^*$ come **configurazione di** D

Definizione 1.14: Passo di computazione in un DFA

Definiamo come **passo di computazione** la relazione binaria definita come

$$(p, aw) \vdash_D (q, w) \iff \delta(p, a) = q$$

Definizione 1.15: Computazione deterministica

Definiamo una computazione come **deterministica** se ad ogni passo di computazione segue un'unica configurazione:

$$\forall (q, aw) \;\; \exists! (p, w) \mid (q, aw) \vdash_D (p, w)$$

Proposizione 1.4: Chiusura del passo di computazione

Sia $D := (Q, \Sigma, \delta, q_0, F)$ un DFA. La **chiusura riflessiva e transitiva** di $\vdash_D$, indicata come $\vdash_D^*$, gode delle seguenti proprietà:

- $(p, aw) \vdash_D (q, w) \implies (p, aw) \vdash_D^* (q, w)$
- $\forall q \in Q, w \in \Sigma^* \quad (q, w) \vdash_D^* (q, w)$
- $(p, abw) \vdash_D (q, bw) \land (q, bw) \vdash_D (r, w) \implies (p, abw) \vdash_D^* (r, w)$

Osservazione 1.1

Sia $D := (Q, \Sigma, \delta, q_0, F)$ un DFA. Dati $q_i, q_f \in Q, w \in \Sigma^*$, si ha che

$$\delta^*(q_i, w) = q_f \iff (q_i, w) \vdash_D^* (q_f, \varepsilon)$$

1.3 Non determinismo

Definizione 1.16: Alfabeto epsilon

Dato un alfabeto Σ, definiamo $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ come **alfabeto epsilon di** Σ

Definizione 1.17: Non-deterministic Finite Automaton (NFA)

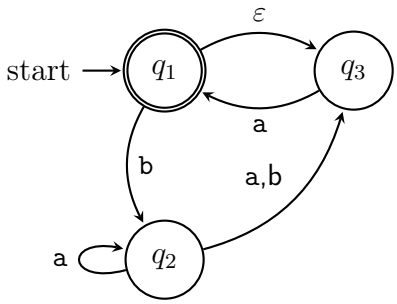
Un **Non-deterministic Finite Automaton (NFA)** (o *Automa Non-deterministico a Stati Finiti*) è una quintupla $(Q, \Sigma, \delta, q_0, F)$ dove:

- Q è l'**insieme finito degli stati** dell'automa
- Σ è l'**alfabeto** dell'automa
- $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ è la **funzione di transizione degli stati** dell'automa
- $q_0 \in Q$ è lo **stato iniziale** dell'automa
- $F \subseteq Q$ è l'**insieme degli stati accettanti** dell'automa

Nota: $\mathcal{P}(Q)$ è l'insieme delle parti di Q, ossia l'insieme contenente tutti i suoi sottoinsiemi possibili

Esempio:

- Consideriamo il seguente NFA



dove:

- $Q = \{q_1, q_2, q_3\}$ è l'insieme degli stati dell'automa
- $\Sigma = \{a, b\}$ è l'alfabeto dell'automa
- $\delta : Q \times \Sigma \to Q$ definita come

δ	q_1	q_2	q_3
ε	$\{q_3\}$	$\varnothing$	$\varnothing$
a	$\varnothing$	$\{q_2, q_3\}$	$\{q_1\}$
b	$\{q_2\}$	$\{q_3\}$	$\varnothing$

  è la funzione di transizione degli stati dell'automa
- q_1 è lo stato iniziale dell'automa
- $F = \{q_1\}$ è l'insieme degli stati accettanti

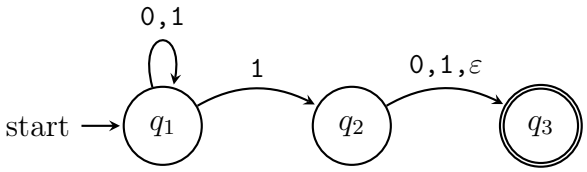
Osservazione 1.2: Computazione in un NFA

Sia $N := (Q, \Sigma, \delta, q_0, F)$ un NFA. Data una stringa $w \in \Sigma_\varepsilon$ in ingresso, la **computazione** viene eseguita nel seguente modo:

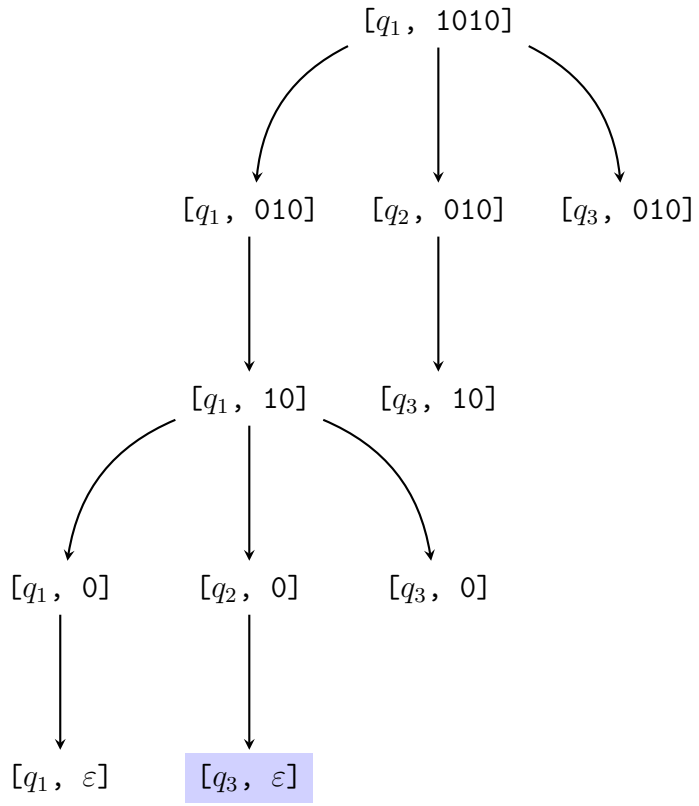
- Tutte le volte che uno stato potrebbe avere più transizioni per diversi simboli dell'alfabeto, l'automa N si duplica in **più copie**, ognuna delle quali segue il suo corso. Si vengono così a creare più **rami di computazione** indipendenti che sono eseguiti in **parallelo**.
- Se il prossimo simbolo della stringa da computare non si trova su nessuna delle transizioni uscenti dello stato attuale di un ramo di computazione, l'intero ramo **termina la sua computazione** (terminazione incorretta).
- Se almeno una delle copie di N termina correttamente su uno stato di accettazione, l'automa **accetta la stringa di partenza**.
- Quando a seguito di una computazione ci si ritrova in uno stato che possiede un **ε-arco** in uscita, la macchina si duplica in più copie: quelle che seguono gli ε-archi e quella che rimane nello stato raggiunto.

Esempio:

- Consideriamo il seguente NFA



- Supponiamo che venga computata la stringa `w = 1010`:



- Poiché esiste un ramo che termina correttamente, l'NFA descritto accetta la stringa `w = 1010`

Proposizione 1.5: Stringa accettata in un NFA

Sia $N := (Q, \Sigma, \delta, q_0, F)$ un NFA. Data una stringa $w := w_0 \dots w_k \in \Sigma^*$, dove $w_0, \dots, w_k \in \Sigma_\varepsilon$, diciamo che w è **accettata da** N se esiste una sequenza di stati $r_0, r_1, \dots, r_{k+1} \in Q$ tali che:

- $r_0 = q_0$
- $\forall i \in [0, k] \;\; r_{i+1} \in \delta(r_i, w_i)$
- $r_{k+1} \in F$

1.3.1 Equivalenza tra NFA e DFA

Definizione 1.18: Classe dei linguaggi riconosciuti da un DFA

Dato un alfabeto Σ, definiamo come **classe dei linguaggi di Σ riconosciuti da un DFA** il seguente insieme:

$$\mathcal{L}(\text{DFA}) = \{L \subseteq \Sigma^* \mid \exists \text{ DFA } D \text{ t.c } L = L(D)\}$$

Definizione 1.19: Classe dei linguaggi riconosciuti da un NFA

Dato un alfabeto Σ, definiamo come **classe dei linguaggi di Σ riconosciuti da un NFA** il seguente insieme:

$$\mathcal{L}(\text{NFA}) = \{L \subseteq \Sigma^* \mid \exists \text{ NFA } N \text{ t.c } L = L(N)\}$$

Teorema 1.2: Equivalenza tra NFA e DFA

Date le due classi dei linguaggi $\mathcal{L}(\text{DFA})$ e $\mathcal{L}(\text{NFA})$, si ha che:

$$\mathcal{L}(\text{DFA}) = \mathcal{L}(\text{NFA})$$

Dimostrazione.

Prima implicazione.

- Dato $L \in \mathcal{L}(\text{DFA})$, sia $D := (Q, \Sigma, \delta, q_0, F)$ il DFA tale che $L = L(D)$
- Poiché il concetto di NFA è una generalizzazione del concetto di DFA, ne segue automaticamente che D sia anche un NFA, implicando che $L \in \mathcal{L}(\text{NFA})$ e di conseguenza che:
$$\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{NFA})$$

Seconda implicazione.

- Dato $L \in \mathcal{L}(\text{NFA})$, sia $N := (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$ il NFA tale che $L = L(N)$
- Consideriamo quindi il DFA $D := (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$ costruito tramite N stesso:
  - $Q_D = \mathcal{P}(Q_N)$
  - Dato $R \in Q_D$, definiamo l'estensione di R come:
  $$E(R) = \{q \in Q_N \mid \exists p \in R \text{ che raggiunge } q \text{ in } N \text{ tramite solo } \varepsilon\text{-archi}\}$$
  - $q_{0_D} = E(\{q_{0_N}\})$
  - $F_D = \{R \in Q_D \mid R \cap F_N \neq \varnothing\}$

– Dati $R \in Q_D$ e $a \in \Sigma$, definiamo δ_D come:

$$\delta_D(R, a) = \bigcup_{r \in R} E(\delta_N(r, a))$$

- A questo punto, per costruzione stessa di D si ha che:

$$w \in L = L(N) \iff w \in L(D)$$

implicando dunque che $L \in \mathcal{L}(\mathsf{DFA})$ e di conseguenza che:

$$\mathcal{L}(\mathsf{NFA}) \subseteq \mathcal{L}(\mathsf{DFA})$$

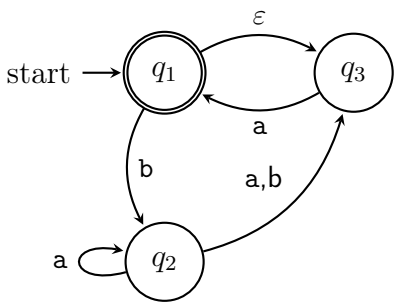
□

Osservazione 1.3

Dato un NFA N, seguendo i passaggi della dimostrazione precedente è possibile definire un DFA D equivalente ad N

Esempio:

- Consideriamo ancora il seguente NFA



- Definiamo quindi l'insieme degli stati del DFA equivalente a tale NFA:

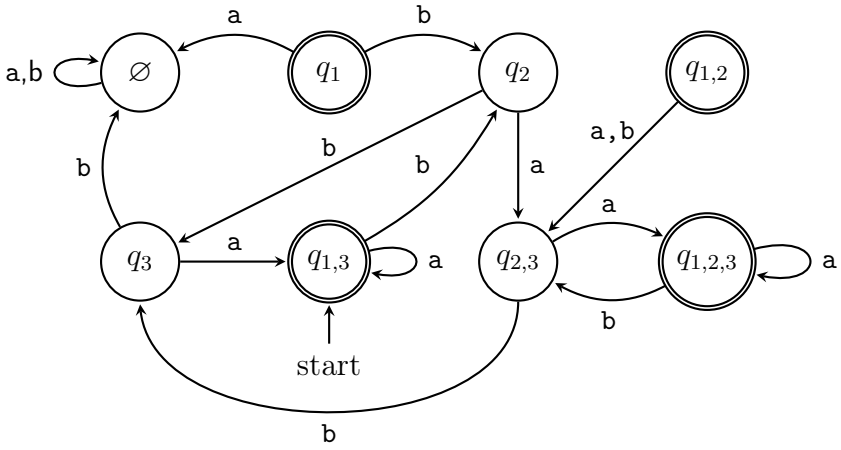
$$Q_D = \{\varnothing, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_2, q_3\}, \{q_1, q_3\}, \{q_1, q_2, q_3\}\} =$$

- Per facilitare la lettura, riscriviamo i vari stati con la seguente notazione

$$Q_D = \{\varnothing, q_1, q_2, q_3, q_{1,2}, q_{2,3}, q_{1,3}, q_{1,2,3}\}$$

- A questo punto, poniamo:
  - $q_{0_D} = E(\{q_{0_N}\}) = E(\{q_1\}) = \{q_1, q_3\} = q_{1,3}$
  - $F_D = \{q_1, q_{1,2}, q_{1,3}, q_{1,2,3}\}$

- Le transizioni del DFA corrisponderanno invece a:
  - $\delta_D(\{q_1\}, a) = E(\delta_N(q_1, a)) = \varnothing$
  - $\delta_D(\{q_1\}, b) = E(\delta_N(q_1, b)) = \{q_2\}$
  - $\delta_D(\{q_2\}, a) = E(\delta_N(q_2, a)) = \{q_2, q_3\}$
  - $\delta_D(\{q_2\}, b) = E(\delta_N(q_2, b)) = \{q_3\}$
  - $\delta_D(\{q_1, q_2\}, a) = E(\delta_N(q_1, a)) \cup E(\delta_N(q_2, a)) = \varnothing \cup \{q_2, q_3\} = \{q_2, q_3\}$
  - $\delta_D(\{q_1, q_2\}, b) = E(\delta_N(q_1, b)) \cup E(\delta_N(q_2, b)) = \{q_2\} \cup \{q_3\} = \{q_2, q_3\}$
  - ...
- Il DFA equivalente corrisponde dunque a:



Definizione 1.20: Linguaggi regolari

Dato un alfabeto Σ, definiamo come **insieme dei linguaggi regolari di** Σ, indicato con REG, l'insieme delle classi dei linguaggi riconosciuti da un DFA:

$$\mathsf{REG} := \mathcal{L}(\mathsf{DFA})$$

Osservazione 1.4

Tramite il teorema dell'Equivalenza tra NFA e DFA, si ha che:

$$\mathsf{REG} := \mathcal{L}(\mathsf{DFA}) = \mathcal{L}(\mathsf{NFA})$$

1.4 Chiusure dei linguaggi regolari

Teorema 1.3: Chiusura dell'unione in REG

L'operatore unione è **chiuso in REG**, ossia:

$$\forall L_1, L_2 \in \mathsf{REG} \;\; L_1 \cup L_2 \in \mathsf{REG}$$

Dimostrazione I.

- Dati $L_1, L_2 \in \mathsf{REG}$, siano $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ i due DFA tali che $L_1 = L(D_1)$ e $L_2 = L(D_2)$
- Definiamo quindi il DFA $D = (Q, \Sigma, \delta, q_0, F)$ tale che:
  - $q_0 = (q_1, q_2)$
  - $Q = Q_1 \times Q_2$
  - $F = (F_1 \times Q_2) \cup (Q_1 \times F_2) = \{(r_1, r_2) \mid r_1 \in F_1 \vee r_2 \in F_2\}$
  - $\forall (r_1, r_2) \in Q, a \in \Sigma$ si ha che:

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

- A questo punto, per costruzione stessa di D ne segue che:

$$w \in L_1 \cup L_2 \iff w \in L(D)$$

  dunque che $L_1 \cup L_2 = L(D) \in \mathsf{REG}$

□

Dimostrazione II.

- Dati $L_1, L_2 \in \mathsf{REG}$, siano $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ i due NFA tali che $L_1 = L(N_1)$ e $L_2 = L(N_2)$
- Definiamo quindi il NFA $N = (Q, \Sigma, \delta, q_0, F)$ tale che:
  - q_0 è un nuovo stato iniziale aggiunto
  - $Q = Q_1 \cup Q_2 \cup \{q_0\}$
  - $F = F_1 \cup F_2$
  - $\forall q \in Q, a \in \Sigma$ si ha che:

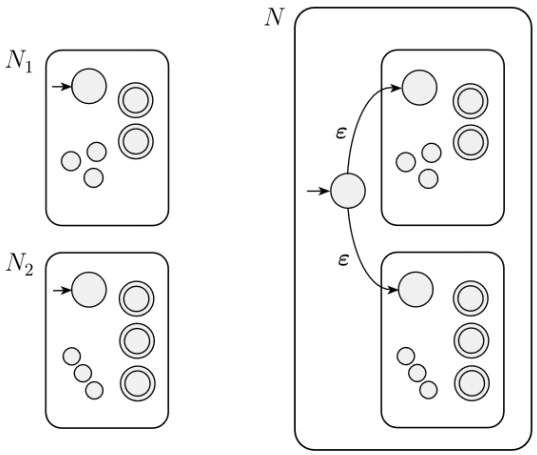
$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{se } q \in Q_1 \\ \delta_2(q, a) & \text{se } q \in Q_2 \\ \{q_1, q_2\} & \text{se } q = q_0 \wedge a = \varepsilon \\ \varnothing & \text{se } q = q_0 \wedge a \neq \varepsilon \end{cases}$$

- A questo punto, per costruzione stessa di N ne segue che:

$$w \in L_1 \cup L_2 \iff w \in L(N)$$

  dunque che $L_1 \cup L_2 = L(N) \in \mathsf{REG}$

□



Rappresentazione grafica della dimostrazione

Teorema 1.4: Chiusura dell'intersezione in REG

L'operatore intersezione è **chiuso in REG**, ossia:

$$\forall L_1, L_2 \in \mathsf{REG} \;\; L_1 \cap L_2 \in \mathsf{REG}$$

Dimostrazione.

- Dati $L_1, L_2 \in \mathsf{REG}$, siano $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ i due DFA tali che $L_1 = L(D_1)$ e $L_2 = L(D_2)$
- Definiamo quindi il DFA $D = (Q, \Sigma, \delta, q_0, F)$ tale che:
  - $q_0 = (q_1, q_2)$
  - $Q = Q_1 \times Q_2$
  - $F = F_1 \times F_2 = \{(r_1, r_2) \mid r_1 \in F_1 \wedge r_2 \in F_2\}$
  - $\forall (r_1, r_2) \in Q, a \in \Sigma$ si ha che:

$$\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$$

- A questo punto, per costruzione stessa di D ne segue che:

$$w \in L_1 \cap L_2 \iff w \in L(D)$$

  dunque che $L_1 \cap L_2 = L(D) \in \mathsf{REG}$

□

Teorema 1.5: Chiusura del complemento in REG

L'operatore complemento è **chiuso in REG**, ossia:

$$\forall L \in \mathsf{REG} \;\; \overline{L} \in \mathsf{REG}$$

Dimostrazione.

- Dato $L \in \mathsf{REG}$, sia $D = (Q, \Sigma, \delta, q_0, F)$ il DFA tale che $L = L(D)$
- Definiamo quindi il DFA $D' = (Q, \Sigma, \delta, q_0, Q - F)$, dunque il DFA uguale a D ma i cui stati accettanti sono invertiti. Per costruzione stessa di D' ne segue che:

  $$w \in L \iff w \notin L(D')$$

  dunque che $\overline{L} = L(D') \in \mathsf{REG}$

□

Teorema 1.6: Chiusura della concatenazione in REG

L'operatore concatenazione è **chiuso in REG**, ossia:

$$\forall L_1, L_2 \in \mathsf{REG} \;\; L_1 \circ L_2 \in \mathsf{REG}$$

Dimostrazione.

- Dati $L_1, L_2 \in \mathsf{REG}$, siano $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ i due NFA tali che $L_1 = L(N_1)$ e $L_2 = L(N_2)$
- Definiamo quindi il NFA $N = (Q, \Sigma, \delta, q_0, F)$ tale che:
  - $q_0 = q_1$
  - $Q = Q_1 \cup Q_2$
  - $F = F_2$
  - $\forall q \in Q, a \in \Sigma$ si ha che:

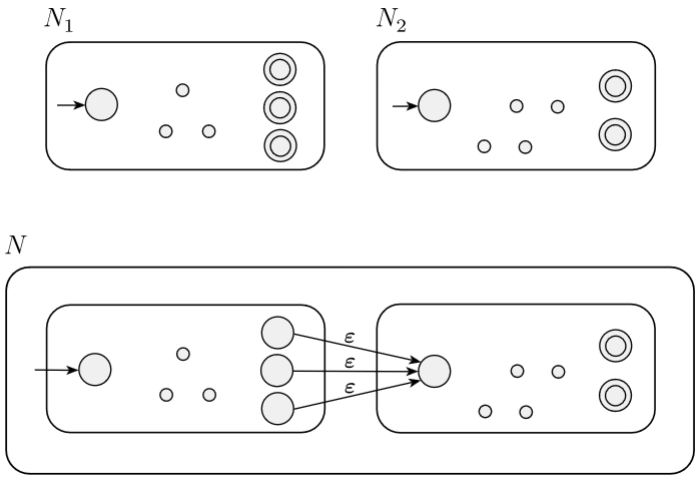
$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{se } q \in Q_1 - F_1 \\ \delta_1(q,a) & \text{se } q \in F_1 \wedge a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & \text{se } q \in F_1 \wedge a = \varepsilon \\ \delta_2(q,a) & \text{se } q \in Q_2 \end{cases}$$

- A questo punto, per costruzione stessa di N ne segue che:

  $$w \in L_1 \circ L_2 \iff w \in L(N)$$

  dunque che $L_1 \circ L_2 = L(N) \in \mathsf{REG}$

□



Rappresentazione grafica della dimostrazione

Corollario 1.1: Chiusura della potenza in REG

L'operatore potenza è **chiuso in REG**, ossia:

$$\forall L \in \mathsf{REG}, n \in \mathbb{N} \quad L^n \in \mathsf{REG}$$

Teorema 1.7: Chiusura di star in REG

L'operatore star è **chiuso in REG**, ossia:

$$\forall L \in \mathsf{REG} \quad L^* \in \mathsf{REG}$$

Dimostrazione.

- Dato $L \in \mathsf{REG}$, sia $N = (Q, \Sigma, \delta, q_0, F)$ il NFA tale che $L = L(N)$
- Definiamo quindi il DFA $N' = (Q', \Sigma, \delta', q_{0*}, F')$ tale che:
  - q_{0*} è un nuovo stato iniziale aggiunto
  - $Q' = Q \cup \{q_{0*}\}$
  - $F' = F \cup \{q_{0*}\}$
  - $\forall q \in Q', a \in \Sigma$ si ha che:

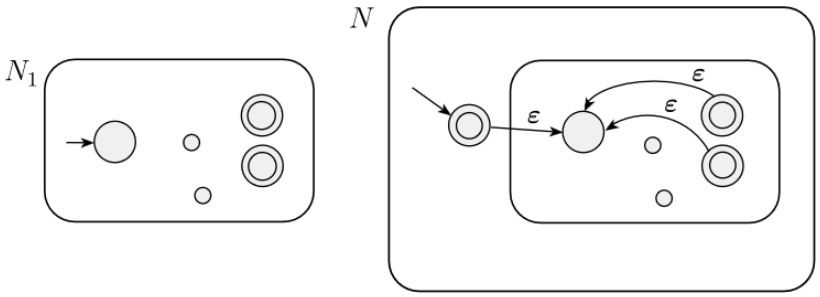
$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{se } q \in Q - F \\ \delta(q,a) & \text{se } q \in F \wedge a \neq \varepsilon \\ \delta(q,a) \cup \{q_0\} & \text{se } q \in F \wedge a = \varepsilon \\ \{q_0\} & \text{se } q = q_{0*} \wedge a = \varepsilon \\ \varnothing & \text{se } q = q_{0*} \wedge a \neq \varepsilon \end{cases}$$

- A questo punto, per costruzione stessa di N' ne segue che:

$$w \in L^* \iff w \in L(N')$$

  dunque che $L^* = L(N') \in \mathsf{REG}$

□



Rappresentazione grafica della dimostrazione

Corollario 1.2: Chiusura di plus in REG

L'operatore plus è **chiuso in REG**, ossia:

$$\forall L \in \mathsf{REG} \;\; L^+ \in \mathsf{REG}$$

Dimostrazione.

- Analoga a quella dell'operatore star, rimuovendo tuttavia lo stato iniziale dall'insieme degli stati accettanti

□

1.5 Espressioni regolari

Definizione 1.21: Espressione regolare

Dato un alfabeto Σ, definiamo come **espressione regolare di** Σ una stringa R rappresentante un linguaggio $L(R) \subseteq \Sigma^*$. In altre parole, ogni espressione regolare R rappresenta in realtà il linguaggio $L(R)$ ad essa associata.

In particolare, definiamo l'**insieme delle espressioni regolari di** Σ, indicato con $\text{re}(\Sigma)$, come:

- $\varnothing \in \text{re}(\Sigma)$
- $\varepsilon \in \text{re}(\Sigma)$
- $a \in \text{re}(\Sigma)$, dove $a \in \Sigma$
- $R_1, R_2 \in \text{re}(\Sigma) \implies R_1 \cup R_2 \in \text{re}(\Sigma)$
- $R_1, R_2 \in \text{re}(\Sigma) \implies R_1 \circ R_2 \in \text{re}(\Sigma)$
- $R \in \text{re}(\Sigma) \implies R^* \in \text{re}(\Sigma)$
- $R \in \text{re}(\Sigma) \implies R^+ \in \text{re}(\Sigma)$

Osservazione 1.5

Data un'espressione regolare $R \in \text{re}(R)$, si ha che:

- $R = \varnothing \in \text{re}(\Sigma) \implies L(R) = \varnothing$
- $R = \varepsilon \in \text{re}(\Sigma) \implies L(R) = \{\varepsilon\}$
- $R = a \in \text{re}(\Sigma), a \in \Sigma \implies L(R) = \{a\}$
- $R = R_1 \cup R_2 \in \text{re}(\Sigma) \implies L(R) = L(R_1) \cup L(R_2)$
- $R = R_1 \circ R_2 \in \text{re}(\Sigma) \implies L(R) = L(R_1) \circ L(R_2)$
- $R = R_1^* \in \text{re}(\Sigma) \implies L(R) = L(R_1)^*$
- $R = R_1^+ \in \text{re}(\Sigma) \implies L(R) = L(R_1)^+$

Esempi:

1. $0 \cup 1$ rappresenta il linguaggio $\{0\} \cup \{1\} = \{0, 1\}$
2. 0^*10^* rappresenta il linguaggio $\{0\}^* \circ \{1\} \circ \{0\}^* = \{x1y \mid x, y \in \{0\}^*\}$
3. $\Sigma^*1\Sigma^*$ rappresenta il linguaggio $\Sigma^* \circ \{1\} \circ \Sigma^* = \{x1y \mid x, y \in \Sigma^*\}$
4. $(0 \cup 1000)^*$ rappresenta il linguaggio $(\{0\} \cup \{1000\})^* = \{0, 1000\}^*$
5. $\varnothing^*$ rappresenta il linguaggio $\varnothing^* = \{\varepsilon\}$ (ricordiamo che per definizione stessa si ha che $\forall L \subseteq \Sigma^* \;\; L^0 = \{\varepsilon\}$)

6. $0^*\varnothing$ rappresenta il linguaggio $\{0\}^* \circ \varnothing = \varnothing$
7. $(0 \cup \varepsilon)(1 \cup \varepsilon)$ rappresenta il linguaggio $\{\varnothing, 0, 1, 01\}$
8. Σ^+ equivale all'espressione $\Sigma\Sigma^*$

Definizione 1.22: Classe dei linguaggi descritti da esp. reg.

Dato un alfabeto Σ, definiamo come **classe dei linguaggi di Σ descritti da un'espressione regolare** il seguente insieme:

$$\mathcal{L}(\text{re}) = \{L \subseteq \Sigma^* \mid \exists R \in \text{re}(\Sigma) \text{ t.c. } L = L(R)\}$$

Lemma 1.1: Conversione da espressione regolare a NFA

Date le due classi dei linguaggi $\mathcal{L}(\text{re})$ e $\mathcal{L}(\mathsf{NFA})$, si ha che:

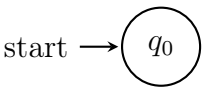
$$\mathcal{L}(\text{re}) \subseteq \mathcal{L}(\mathsf{NFA})$$

Dimostrazione.

Procediamo per induzione strutturale, ossia dimostrando che se per ogni sotto-componente vale una determinata proprietà allora essa varrà anche per ogni componente formato da tali sotto-componenti

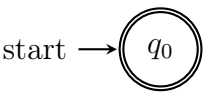
Caso base.

- Se $R = \varnothing \in \text{re}(\Sigma)$, definiamo il NFA $N_\varnothing = (\{q_0\}, \Sigma, \delta, q_0, \varnothing)$, ossia:



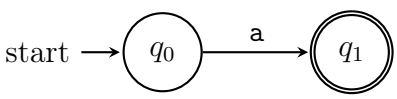
  per cui si ha che $w \in L(R) \iff w \in L(N_\varnothing)$ dunque $L(R) = L(N_\varnothing) \in \mathcal{L}(\mathsf{NFA})$

- Se $R = \varepsilon \in \text{re}(\Sigma)$, definiamo il NFA $N_\varepsilon = (\{q_0\}, \Sigma, \delta, q_0, \{q_0\})$, ossia:



  per cui si ha che $w \in L(R) \iff w \in L(N_\varepsilon)$ dunque $L(R) = L(N_\varepsilon) \in \mathcal{L}(\mathsf{NFA})$

- Se $R = a \in \text{re}(\Sigma)$ con $a \in \Sigma$, definiamo il NFA $N_a = (\{q_0, q_1\}, \Sigma, \delta, q_0, \{q_1\})$ dove per δ è definita solo la coppia $\delta(q_0, a) = q_1$, ossia:



  per cui si ha che $w \in L(R) \iff w \in L(N_a)$ dunque $L(R) = L(N_a) \in \mathcal{L}(\mathsf{NFA})$

Ipotesi induttiva.

- Date $R_1, R_2 \in \text{re}(\Sigma)$, assumiamo che $\exists\ \mathsf{NFA} N_1, N_2 \mid L(R_1) = L(N_1), L(R_2) = L(N_2)$, dunque che $L(R_1), L(R_2) \in \mathcal{L}(\mathsf{NFA})$

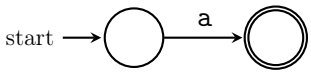
Passo induttivo.

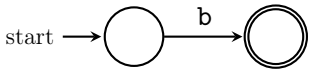
- Se $R = R_1 \cup R_2$, tramite la Chiusura dell'unione in REG, otteniamo che:
$$L(R) = L(R_1) \cup L(R_2) = L(N_1) \cup L(N_2) \in \mathsf{REG} = \mathcal{L}(\mathsf{NFA})$$
- Se $R = R_1 \circ R_2$, tramite la Chiusura della concatenazione in REG, otteniamo che:
$$L(R) = L(R_1) \circ L(R_2) = L(N_1) \circ L(N_2) \in \mathsf{REG} = \mathcal{L}(\mathsf{NFA})$$
- Se $R = R_1^*$, tramite la Chiusura di plus in REG, otteniamo che:
$$L(R) = L(R_1)^* = L(N_1)^* \in \mathsf{REG} = \mathcal{L}(\mathsf{NFA})$$

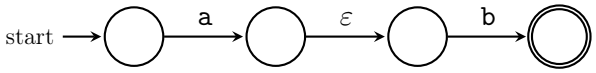
□

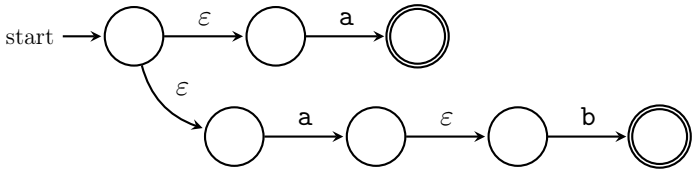
Esempio:

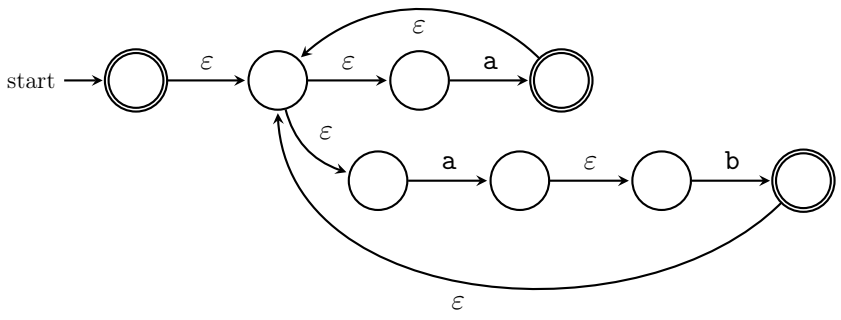
- Consideriamo l'espressione regolare $(a \cup ab)^*$
- Costruiamo il NFA corrispondente a tale espressione partendo dai suoi sotto-componenti

$a \implies$ 

$b \implies$ 

$ab \implies$ 

$(a \cup ab) \implies$ 

$(a \cup ab)^* \implies$ 

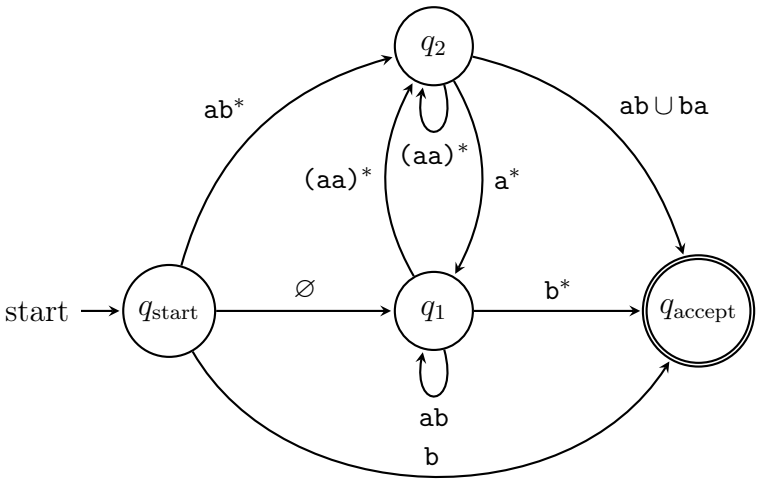
1.5.1 NFA generalizzati

Definizione 1.23: Generalized NFA (GNFA)

Un **Generalized NFA (GNFA)** è una quintupla $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ dove:

- Q è l'**insieme finito degli stati** dell'automa dove $|Q| \geq 2$
- Σ è l'**alfabeto** dell'automa
- $q_{\text{start}} \in Q$ è lo **stato iniziale** dell'automa
- $q_{\text{accept}} \in Q$ è l'**unico stato accettante** dell'automa
- $\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \to \text{re}(\Sigma)$ è la **funzione di transizione degli stati** dell'automa, implicando che:
  - Lo stato q_{start} abbia solo transizioni **uscenti**
  - Lo stato q_{accept} abbia solo transizioni **entranti**
  - Tra **tutte le possibili coppie di stati** $q, q' \in Q$ (incluso il caso in cui $q = q'$) vi sia una transizione $q \to q'$ ed una transizione $q' \to q$
  - Le "etichette" delle transizioni sono delle **espressioni regolari**

Esempio:



Osservazione 1.6

In un GNFA, il risultato $\delta(q, q') = R$ può essere interpretato come "l'espressione regolare che effettua la transizione da q a q' è R". Di conseguenza, possiamo immaginare un GNFA come un NFA che legga la stringa in input **blocco per blocco**

Proposizione 1.6: Stringa accettata in un GNFA

Sia $G := (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ un GNFA. Data una stringa $w := w_0 \dots w_k \in \Sigma^*$, dove $w_0, \dots, w_k \in \Sigma^*$ (ossia sono delle sottostringhe), diciamo che w è **accettata da** G se esiste una sequenza di stati $r_0, r_1, \dots, r_{k+1} \in Q$ tali che:

- $r_0 = q_{\text{start}}$
- $\forall i \in [0, k] \;\; w_i \in L(\delta(r_i, r_{i+1}))$
- $r_{k+1} = q_{\text{accept}}$

Esempio:

- Il GNFA dell'esempio precedente accetta la stringa `abababaaaba`, poiché:
  - $\delta(q_{\text{start}}, q_1) = \texttt{ab}^*$, dunque viene letta in blocco la sottostringa `abab`
  - $\delta(q_1, q_1) = \texttt{aa}^*$, dunque viene letta in blocco la sottostringa `aa`
  - $\delta(q_1, q_{\text{accept}}) = \texttt{ab} \cup \texttt{ba}$, dunque viene letta in blocco la sottostringa `ba`

Corollario 1.3

Una transizione con "etichetta" pari a $\varnothing$ è una **transizione inutilizzabile** in quanto $L(\varnothing) = \varnothing$

Definizione 1.24: Classe dei linguaggi riconosciuti da un GNFA

Dato un alfabeto Σ, definiamo come **classe dei linguaggi di Σ riconosciuti da un GNFA** il seguente insieme:

$$\mathcal{L}(\text{GNFA}) = \{L \subseteq \Sigma^* \mid \exists \text{ GNFA } G \text{ t.c } L = L(G)\}$$

Lemma 1.2: Conversione da DFA a GNFA

Date le due classi dei linguaggi $\mathcal{L}(\text{DFA})$ e $\mathcal{L}(\text{GNFA})$, si ha che:

$$\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{GNFA})$$

Dimostrazione.

- Dato $L \in \mathcal{L}(\text{DFA})$, sia $D := (Q, \Sigma, \delta, q_0, F)$ il DFA tale che $L(D) = L$
- Consideriamo quindi il GNFA $G := (Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$ costruito tramite D stesso:
  - $Q' = Q \cup \{q_{\text{start}}, q_{\text{accept}}\}$
  - $\delta'(q_{\text{start}}, q_0) = \varepsilon$
  - $\forall q \in F \;\; \delta'(q, q_{\text{accept}}) = \varepsilon$

  - Per ogni transizione con etichetta multipla in D, in G esiste una transizione equivalente con etichetta corrispondente all'unione di tali etichette multiple
  - Per ogni coppia di stati per cui non esiste una transizione entrante o uscente in D, viene aggiunta una transizione con etichetta $\varnothing$

- A questo punto, per costruzione stessa di G si ha che:

$$w \in L = L(D) \implies L(G)$$

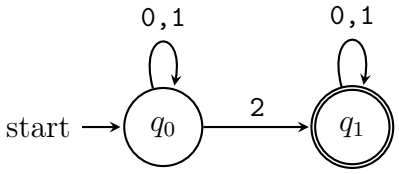
  implicando dunque che $L(D) \in \mathcal{L}(\mathsf{DFA})$ e di conseguenza che:

$$\mathcal{L}(\mathsf{DFA}) \subseteq \mathcal{L}(\mathsf{GNFA})$$

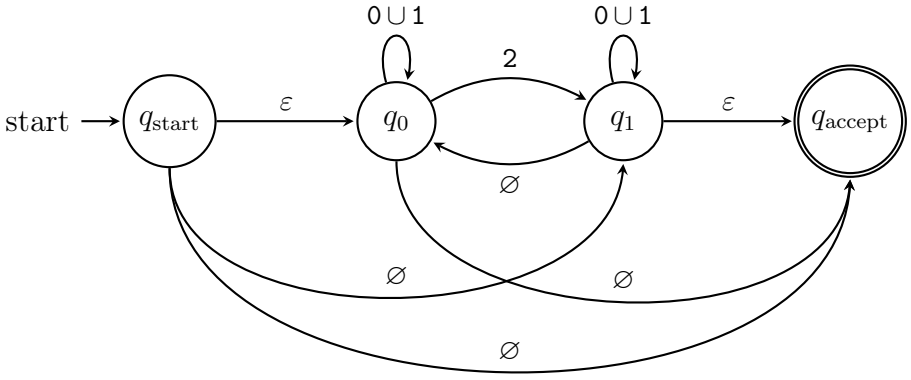
□

Esempio:

- Consideriamo il seguente DFA:



- Il suo GNFA equivalente corrisponde a:



Algoritmo 1.1: Riduzione minimale di un GNFA

Dato un GNFA $G = (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, il seguente algoritmo restituisce un GNFA G' avente solo due stati e tale che $L(G) = L(G')$:

function REDUCEGNFA(G)
  if $|Q| == 2$ **then**
    return G
  else if $|Q| > 2$ **then**
    $q := q \in Q - \{q_{\text{start}}, q_{\text{accept}}\}$
    $Q' := Q - \{q\}$
    for $q_i \in Q' - \{q_{\text{accept}}\}$ **do**
      for $q_j \in Q' - \{q_{\text{start}}\}$ **do**
        $\delta'(q_i, q_j) := \delta(q_i, q)\delta(q, q)^*\delta(q, q_j) \cup \delta(q_i, q_j)$
      end for
    end for
    $G' := (Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$
    return `reduceGNFA`(G')
  end if
end function

Dimostrazione.

Siano $G_0, \ldots, G_n$ i vari GNFA prodotti dalla ricorsione dell'algoritmo, implicando che $G_0 = G$ e che G_n sia l'output. Procediamo per induzione sul numero $k \in \mathbb{N}$ di riduzioni effettuate, mostrando che $L(G) = L(G_0) = \ldots = L(G_n)$

Caso base.

- Se $k = 0$, allora $G_0 = G$, dunque $L(G) = L(G_0)$

Ipotesi induttiva.

- Dato $k \in \mathbb{N}$, assumiamo che per il GNFA $G_k := (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ si abbia che $L(G) = L(G_k)$

Passo induttivo.

- Consideriamo quindi il GNFA $G_{k+1} := (Q', \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ ottenuto rimuovendo uno stato $q \in Q$ (dunque $Q' = Q - \{q\}$) e ponendo

$$\delta'(q_i, q_j) := \delta(q_i, q)\delta(q, q)^*\delta(q, q_j) \cup \delta(q_i, q_j)$$

  per ogni $q_i \in Q' - \{q_{\text{accept}}\}, q_j \in Q' - \{q_{\text{start}}\}$

- Data una stringa $w := w_0 \ldots w_m \in L(G_k)$, dove $w_0, \ldots, w_m \in \Sigma^*$, esiste una sequenza di stati $q_0, \ldots, q_m \in Q$ tali che:
  - $q_0 = q_{\text{start}}$ e $q_m = q_{\text{accept}}$
  - $\forall i \in [0, m-1] \;\; w_i \in L(\delta(q_i, q_{i+1}))$

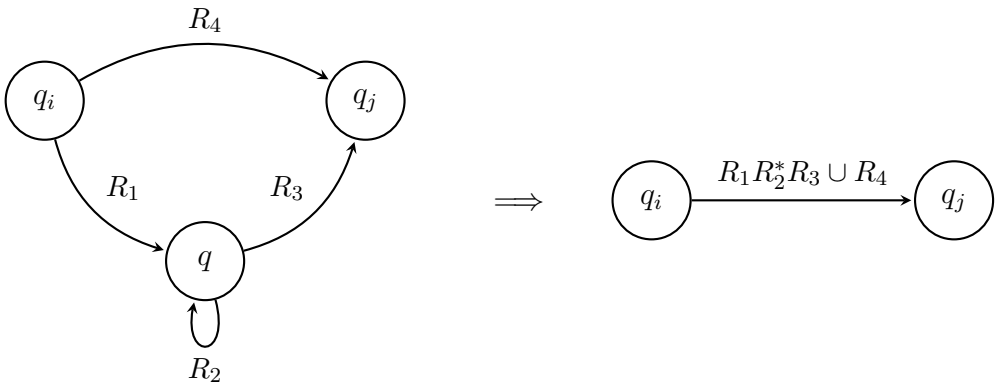
- A questo punto, consideriamo la costruzione della funzione δ':

$$\delta'(q_i, q_j) = \delta(q_i, q)\delta(q, q)^*\delta(q, q_j) \cup \delta(q_i, q_j)$$

  - Se $q \notin \{q_0, \ldots, q_m\}$, allora tramite l'unione si ha che $w_i \in L(\delta(q_i, q_j)) \implies w \in L(\delta'(q_i, q_j))$, dunque tutte le possibili sottostringhe passanti per le transizioni dirette da q_i a q_j vengono riconosciute
  - Se $q \in \{q_0, \ldots, q_m\}$, allora la concatenazione $\delta(q_i, q)\delta(q, q)^*\delta(q, q_j)$ permette il riconoscimento di tutti i cammini da q_i a q_j passanti per q, implicando che $w \in L(\delta'(q_i, q_j))$

- Viceversa, poiché ogni $\delta'(q_i, q_j)$ è definito come la combinazione di tutti i cammini possibili da q_i a q_j (dunque passando per q o non), ne segue automaticamente che $w \in L(G_{k+1}) \implies w \in L(G_k)$

- Esprimendo il tutto graficamente, risulta evidente che le seguenti transizioni siano del tutto equivalenti:

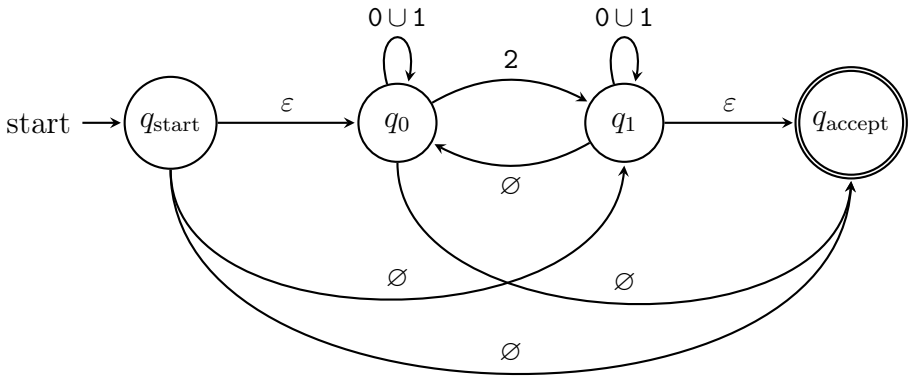


- Di conseguenza, otteniamo che $w \in L(G_k) \iff w \in L(G_{k+1})$, concludendo quindi, per ipotesi induttiva, che $L(G) = L(G_k) = L(G_{k+1})$

□

Esempio:

- Consideriamo nuovamente il seguente GNFA, applicando su esso l'algoritmo `reduceGNFA`:



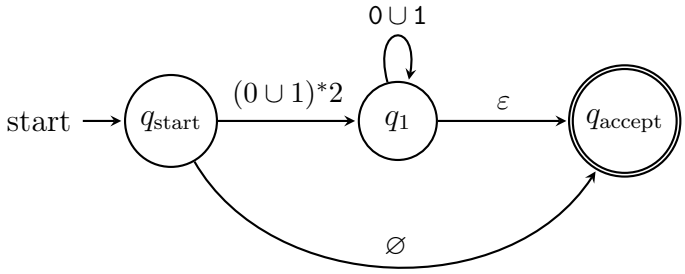
- Rimuoviamo quindi lo stato q_0 calcolando le nuove transizioni:

$$\delta'(q_{\text{start}}, q_1) = \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0, q_1) \cup \delta(q_{\text{start}}, q_1) = \varepsilon(0 \cup 1)^*2 \cup \varnothing = (0 \cup 1)^*2$$

$$\delta'(q_{\text{start}}, q_{\text{accept}}) = \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0, q_{\text{accept}}) \cup \delta(q_{\text{start}}, q_{\text{accept}}) = \varepsilon(0 \cup 1)^*\varnothing \cup \varnothing = \varnothing$$

$$\delta'(q_1, q_1) = \delta(q_1, q_0)\delta(q_0, q_0)^*\delta(q_0, q_1) \cup \delta(q_1, q_1) = \varnothing(0 \cup 1)^*2 \cup (0 \cup 1) = 0 \cup 1$$

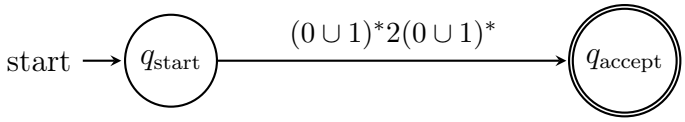
$$\delta'(q_1, q_{\text{accept}}) = \delta(q_1, q_0)\delta(q_0, q_0)^*\delta(q_0, q_{\text{accept}}) \cup \delta(q_1, q_{\text{accept}}) = \varnothing(0 \cup 1)^*\varnothing \cup \varepsilon = \varepsilon$$



- Infine, rimuoviamo lo stato q_1 calcolando le nuove transizioni:

$$\delta''(q_{\text{start}}, q_{\text{accept}}) = \delta'(q_{\text{start}}, q_1)\delta'(q_1, q_1)^*\delta'(q_1, q_{\text{accept}}) \cup \delta'(q_{\text{start}}, q_{\text{accept}}) =$$
$$= (0 \cup 1)^*2(0 \cup 1)^*\varepsilon \cup \varnothing = (0 \cup 1)^*2(0 \cup 1)^*$$

- Il GNFA minimale, dunque, corrisponde a:



Corollario 1.4: Conversione da GNFA ad espressione regolare

Date le due classi dei linguaggi $\mathcal{L}(\text{GNFA})$ e $\mathcal{L}(\text{re})$, si ha che:

$$\mathcal{L}(\text{GNFA}) \subseteq \mathcal{L}(\text{re})$$

Dimostrazione.

- Dato $L \in \mathcal{L}(\text{GNFA})$, sia $G := (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ il GNFA tale che $L(G) = L$
- Dato il GNFA G' ottenuto applicando `reduceGNFA`, sia $R \in \text{re}(\Sigma)$ l'espressione regolare tale che $R = \delta'(q_{\text{start}}, q_{\text{accept}})$. Essendo l'unica transizione di G' ed essendo G' equivalente a G, ne segue automaticamente che:

$$L = L(G) = L(G') = L(R) \in \text{re}(\Sigma)$$

da cui traiamo che:

$$\mathcal{L}(\text{GNFA}) \subseteq \mathcal{L}(\text{re})$$

□

1.5.2 Equivalenza tra espressioni e linguaggi regolari

Teorema 1.8: Equivalenza tra espressioni e linguaggi regolari

Date le due classi dei linguaggi $\mathcal{L}(\text{re})$ e REG, si ha che:

$$\mathcal{L}(\text{re}) = \text{REG}$$

Dimostrazione.

Prima implicazione.

- Tramite la Conversione da espressione regolare a NFA, otteniamo che:

$$\mathcal{L}(\text{re}) \subseteq \mathcal{L}(\text{NFA}) = \text{REG}$$

- Inoltre, in quando un NFA è anche un GNFA, ne segue automaticamente che:

$$\mathcal{L}(\text{NFA}) \subseteq \mathcal{L}(\text{GNFA})$$

Seconda implicazione.

- Tramite la Conversione da DFA a GNFA e Conversione da GNFA ad espressione regolare, otteniamo che:

$$\text{REG} = \mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{GNFA}) \subseteq \mathcal{L}(\text{re})$$

□

Proposizione 1.7: Classe dei linguaggi regolari

Dato un alfabeto Σ, si ha che:

$$\text{REG} := \mathcal{L}(\text{DFA}) = \mathcal{L}(\text{NFA}) = \mathcal{L}(\text{GNFA}) = \mathcal{L}(\text{re})$$

In altre parole, per ogni linguaggio regolare L esistono un DFA, un NFA e un GNFA che lo riconoscono e un'espressione regolare che lo descrive

1.6 Pumping lemma per i linguaggi regolari

Consideriamo il seguente linguaggio composto dalle stringhe aventi un numero uguale di simboli 0 ed 1:

$$L = \{0^n 1^n \mid n \in \mathbb{N}\}$$

Nel provare a costruire un automa che riconosca tale linguaggio, notiamo che sarebbe necessario che l'automa avesse **infiniti stati**, in quanto esso dovrebbe memorizzare la quantità di simboli 0 ed 1 letti. Di conseguenza, non è possibile costruire un **automa a stati finiti** (dunque un DFA, NFA o GNFA) che riconosca tale linguaggio.

Lemma 1.3: Pumping lemma per i linguaggi regolari

Dato un linguaggio L, se $L \in \mathsf{REG}$ allora $\exists p \in \mathbb{N}$, detto **lunghezza del pumping**, tale che $\forall w := xyz \in L$, con $|w| \geq p$ e $x, y, z \in \Sigma^*$ (ossia sono sue sottostringhe), si ha che:

- $\forall i \in \mathbb{N} \;\; xy^i z \in L$, ossia è possibile concatenare y per i volte rimanendo in L
- $|y| > 0$, dunque $y \neq \varepsilon$
- $|xy| \leq p$, ossia y deve trovarsi nei primi p simboli di w

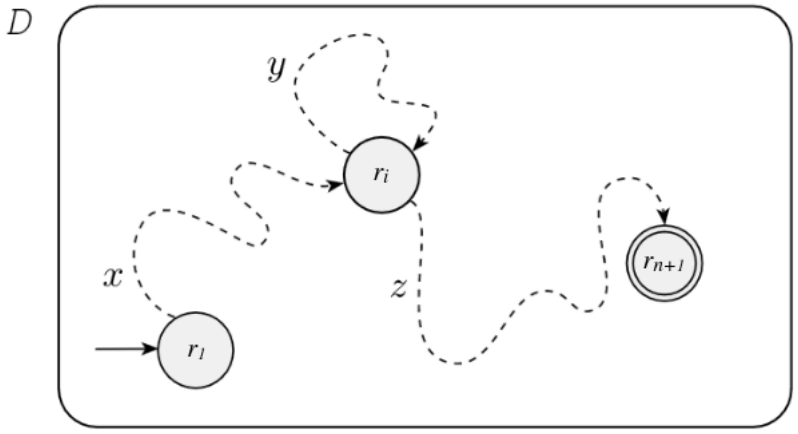
Dimostrazione.

- Dato $L \in \mathsf{REG}$, sia $D := (Q, \Sigma, \delta, q_0, F)$ il DFA tale che $L = L(D)$
- Consideriamo quindi $p := |Q|$. Data la stringa $w := w_1 \dots w_n \in L$ dove $w_1, \dots, w_n \in \Sigma$ e dove $n \geq p$, consideriamo la sequenza di stati $r_1, \dots, r_{n+1}$ tramite cui w viene accettata da D:
$$\forall k \in [1, n] \;\; \delta(r_k, w_k) = r_{k+1}$$
- Notiamo quindi che $|r_1, \dots, r_{n+1}| = n + 1$, ossia che il numero di stati attraversati sia $n + 1$. Inoltre, in quanto $n \geq p$, ne segue automaticamente che $n + 1 \geq p + 1$. Tuttavia, poiché $p := |Q|$ e $n + 1 \geq p + 1$, ne segue necessariamente che $\exists i, j \mid 1 \leq i < j \leq p + 1 \wedge r_i = r_j$, ossia che tra i primi $p + 1$ stati della sequenza vi sia almeno uno stato ripetuto
- A questo punto, consideriamo le seguenti sottostringhe di w:
  - $x = w_1 \dots w_{i-1}$, tramite cui si ha che $\delta^*(r_1, x) = r_i$
  - $y = w_i \dots w_{j-1}$, tramite cui si ha che $\delta^*(r_i, y) = r_j = r_i$
  - $z = w_j \dots w_n$, tramite cui si ha che $\delta^*(r_j, z) = r_{n+1}$
- Poiché $\delta^*(r_i, y) = r_i$, ossia y porta sempre r_i in se stesso, ne segue automaticamente che

$$\forall k \in \mathbb{N} \;\; \delta^*(r_i, y^k) = r_i \implies \delta(r_1, xy^k z) = r_{n+1} \in F \implies xy^k z \in L(D) = L$$

- Inoltre, ne segue direttamente che $|y| > 0$ in quanto $i < j$ e che $|xy| \leq p$ in quanto $j \leq p + 1$

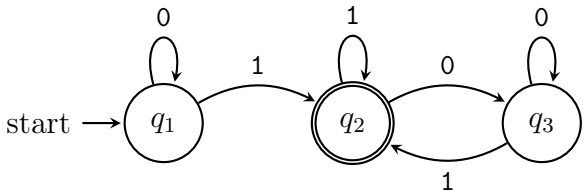
□



Rappresentazione grafica della dimostrazione

Esempio:

- Consideriamo il linguaggio $L = \{x \in \{0,1\}^* \mid x := y1, \exists y \in \{0,1\}^*\}$
- Tale linguaggio risulta essere regolare in quanto il seguente DFA è in grado di riconoscerlo:



- Essendo un linguaggio regolare, per esso vale il Pumping lemma per i linguaggi regolari. Ad esempio, preso $p = 5$ e la stringa $w := 0100010101 \in L$, è possibile separare w in tre sottostringhe $x := 010$, $y = 00$ e $z = 10101$ tali che:
  - $xy^0z = 01010101 \in L$
  - $xy^1z = 0100010101 \in L$
  - $xy^2z = 010000010101 \in L$
  - $xy^3z = 01000000010101 \in L$
  - ...

Osservazione 1.7: Dimostrazione di non regolarità

Il Pumping lemma per i linguaggi regolari può essere utilizzato per dimostrare che un linguaggio **non è regolare**

Esempi:

- Consideriamo il linguaggio $L = \{0^n 1^n \mid n \in \mathbb{N}\}$
- Supponiamo per assurdo che L sia regolare. In tal caso, ne segue che per esso debba valere il pumping lemma, dove p è la lunghezza del pumping
- Consideriamo quindi la stringa $w := 0^p 1^p \in L$. Poiché $|w| \geq p$, possiamo suddividerla in tre sottostringhe $x, y, z \in \Sigma^*$ tali che $w = xyz$, per poi procedere con uno dei due seguenti approcci:
  1. **Approccio enumerativo**:
     - Se y è composta da soli 0, allora ogni stringa generata dal pumping non sarà in L in quanto il numero di 0 sarà superiore al numero di 1
     - Se y è composta da soli 1, allora ogni stringa generata dal pumping non sarà in L in quanto il numero di 1 sarà superiore al numero di 0
     - Se y è composta sia da 0 che da 1, allora ogni stringa generata dal pumping non sarà in L in quanto esse assumeranno la forma $0000 \ldots 101010 \ldots 1111$
     - Di conseguenza, poiché in ogni caso viene contraddetto il pumping lemma, ne segue necessariamente che L non sia regolare
  2. **Approccio condizionale**:
     - Poiché la terza condizione del pumping lemma impone che $|xy| \leq p$ e poiché $w := 0^p 1^p$, ne segue che $xy = 0^m$ e $z = 0^{p-m} 1^p$, dove $m \in [1, p]$
     - Inoltre, per la seconda condizione, si ha che $|y| > 0$, dunque necessariamente si ha che $x = 0^{m-k}$ e $y = 0^k$, dove $k \in [1, m]$
     - A questo punto, consideriamo la stringa xy^0z. Notiamo immediatamente che
       $$xy^0z = 0^{m-k}(0^k)^0 0^{p-m} 1^p = 0^{m-k} 0^{p-m} 1^p = 0^{p-k} 1^p$$
       implicando dunque che $xy^0z \notin L$, contraddicendo la prima condizione del lemma per cui si ha che $\forall i \in \mathbb{N} \;\; xy^iz \in L$
     - Dunque, ne segue necessariamente che L non sia regolare

1.7 Esercizi svolti

Problema 1.1: Linguaggio rovesciato

Dato un linguaggio L e il suo linguaggio rovesciato $L^R = \{w^R \mid w \in L\}$, dimostrare che

$$L \in \mathsf{REG} \implies L^R \in \mathsf{REG}$$

Dimostrazione.

- Dato $L \in \mathsf{REG}$, sia $D = (Q, \Sigma, \delta, q_0, F)$ il DFA tale che $L = L(D)$
- Definiamo quindi un primo NFA $N = (Q', \Sigma, \delta', q_0, \{q_f\})$ tale che:
  - q_f è il nuovo unico stato accettante aggiunto
  - $Q' = Q \cup \{q_f\}$
  - $\forall q \in Q, a \in \Sigma \;\; \delta'(q, a) = \delta(q, a)$, ossia tutti gli archi rimangono invariati
  - $\forall q \in F \;\; \delta'(q, \varepsilon) = q_f$, ossia tutti gli stati finali precedenti hanno un ε-arco verso q_f
- A questo punto, per costruzione stessa di N ne segue che:

  $$w \in L = L(D) \iff w \in L(N)$$

  dunque che $L = L(D) = L(N)$
- Definiamo quindi un secondo NFA $N^R = (Q', \Sigma, \delta'', q_f, \{q_0\})$ avente tutti gli archi invertiti rispetto ad N, ossia tale che:

  $$\forall q \in Q, a \in \Sigma_\varepsilon \;\; \delta''(q, a) = R_q$$

  dove $R_q = \{p \in Q' \mid \delta'(p, a) = q\}$
- A questo punto, per costruzione stessa di N' ne segue che:

  $$w \in L = L(N) \iff w^R \in L(N^R)$$

  dunque che $L^R = L(N)^R = L(N^R) \in \mathsf{REG}$

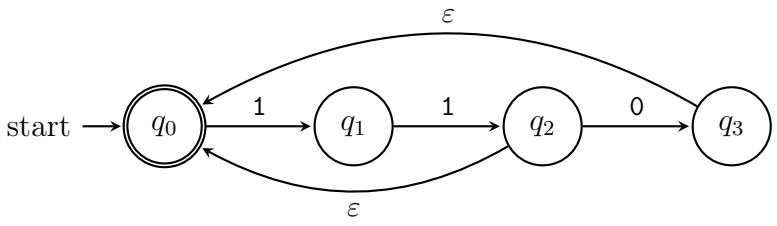
□

Problema 1.2

Dato il linguaggio $L = \{11, 110\}^*$, costruire un NFA N con 4 stati che riconosca L. Convertire il NFA in un DFA M equivalente.

Soluzione:

- Definiamo il seguente NFA $N = (Q_N, \Sigma, \delta_N, q_0^N, F_N)$ come:



  dove risulta evidente che $L(N) = L$

- Convertiamo quindi N nel DFA equivalente $D = (Q_D, \Sigma, \delta_D, q_0^D, F_D)$, dove:
  - $Q_D = \mathcal{P}(Q_N) = \mathcal{P}(\{q_1, q_2, q_3, q_4\})$
  - Dato $R \in Q_D$, sia:

$$E(R) = \{q \in Q_N \mid \exists p \in R \text{ che raggiunge } q \text{ in } N \text{ tramite solo } \varepsilon\text{-archi}\}$$

  - $q_0^D = E(\{q_0^N\}) = \{q_0^N\}$
  - $F_D = \{R \in Q_D \mid R \cap F_N \neq \varnothing\}$
  - Dati $a \in \Sigma$ e $R \in Q_D$ vale che:

$$\delta_D(R, a) = \bigcup_{r \in R} E(\delta_N(r, a))$$

- Per costruzione di D, si ha che:

$$w \in L(D) \iff \delta_D^*(q_0^D, w) \in F_D \iff$$

$$\exists r \in \delta_D^*(q_0^D, w),\ \delta_N^*(q_0^N) = r \in F_N \iff w \in L(N)$$

  concludendo che $L(N) = L(D)$

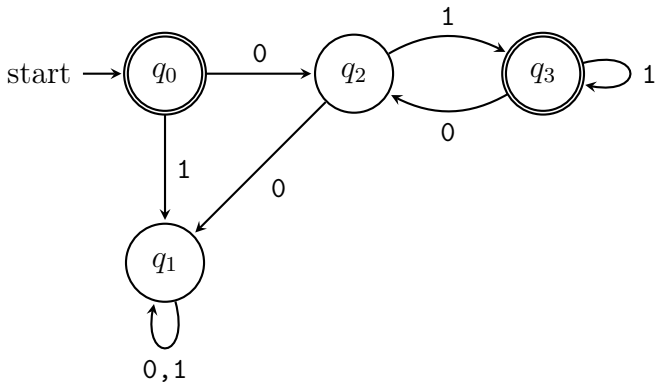
Problema 1.3: Complemento di un'espressione regolare

Data l'espressione regolare $R = (01^+)^*$, costruire il DFA D tale che:

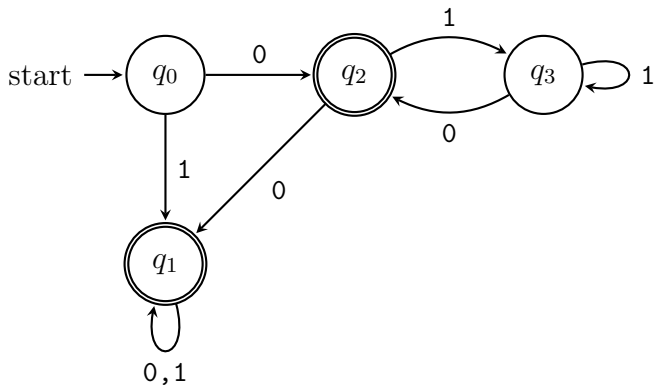
$$L(D) = \{w \in \{0,1\}^* \mid w \notin L(R)\}$$

Soluzione:

- Prima di tutto, costruiamo un DFA D_R tale che $L(D_R) = L(R)$:



- A questo punto, ci basta costruire il DFA D tale che $L(D) = \overline{L(D_R)}$ utilizzando la Chiusura del complemento in REG:



Problema 1.4

Dato il linguaggio $L = \{w \in \{0,1\}^* \mid |w|_0 = |w|_1\}$, dimostrare che $L \notin \mathsf{REG}$

Dimostrazione.

- Supponiamo per assurdo che L sia regolare, implicando che per esso debba valere il pumping lemma, dove p è la lunghezza del pumping
- Consideriamo quindi la stringa $w := 0^p1^p \in L$. Poiché $|w| \geq p$, possiamo suddividerla in tre sottostringhe $x, y, z \in \Sigma^*$ tali che $w = xyz$
- Poiché la terza condizione del pumping lemma impone che $|xy| \leq p$ e poiché $w := 0^p1^p$, ne segue che $xy = 0^m$ e $z = 0^{p-m}1^p$, dove $m \in [1, p]$
- Inoltre, per la seconda condizione, si ha che $|y| > 0$, dunque necessariamente si ha che $x = 0^{m-k}$ e $y = 0^k$, dove $k \in [1, m]$
- A questo punto, consideriamo la stringa xy^0z. Notiamo immediatamente che

$$xy^0z = 0^{m-k}(0^k)^00^{p-m}1^p = 0^{m-k}0^{p-m}1^p = 0^{p-k}1^p$$

$$\implies |xy^0z|_0 \neq |xy^0z|_1 \implies xy^0z \notin L$$

  contraddicendo la prima condizione del lemma per cui si ha che $\forall i \in \mathbb{N} \;\; xy^iz \in L$
- Dunque, ne segue necessariamente che L non sia regolare

□

Problema 1.5

Dato il linguaggio $L = \{1^{n^2} \mid n \in \mathbb{N}\}$, dimostrare che $L \notin \mathsf{REG}$

Dimostrazione.

- Supponiamo per assurdo che L sia regolare, implicando che per esso debba valere il pumping lemma, dove p è la lunghezza del pumping
- Consideriamo quindi la stringa $w := 1^{p^2} \in L$. Poiché $|w| \geq p$, possiamo suddividerla in tre sottostringhe $x, y, z \in \Sigma^*$ tali che $w = xyz$
- Poiché la terza condizione del lemma impone che $|xy| \leq p$ e poiché $w := 1^{p^2}$, ne segue che $xy = 1^m$ e $z = 1^{p^2-m}$, dove $m \in [1, p]$
- Inoltre, per la seconda condizione del lemma, si ha che $|y| > 0$, dunque necessariamente si ha che $x = 1^{m-k}$ e $y = 1^k$, dove $k \in [1, m]$
- A questo punto, consideriamo la stringa xy^0z. Notiamo immediatamente che

$$xy^0z = 1^{m-k}(1^k)^01^{p^2-m} = 1^{p^2-k}$$

- Tuttavia, poiché $k \in [1, p]$, ne segue che $\nexists n \in \mathbb{N} \mid n^2 = p^2 - k$, implicando dunque che $xy^0z \notin L$, contraddicendo la prima condizione del lemma per cui si ha che $\forall i \in \mathbb{N} \;\; xy^iz \in L$
- Dunque, ne segue necessariamente che L non sia regolare

□

Problema 1.6

Dato il linguaggio $L = \{1^i\#1^j\#1^{i+j} \mid i, j \in \mathbb{N}\}$, dimostrare che $L \notin \mathsf{REG}$

Dimostrazione.

- Supponiamo per assurdo che L sia regolare, implicando che per esso debba valere il pumping lemma, dove p è la lunghezza del pumping
- Consideriamo quindi la stringa $w := 1^p\#1^p\#1^{2p} \in L$. Poiché $|w| \geq p$, possiamo suddividerla in tre sottostringhe $x, y, z \in \Sigma^*$ tali che $w = xyz$
- Poiché la terza condizione del lemma impone che $|xy| \leq p$ e poiché $w := 1^p\#1^p\#1^{2p}$, ne segue che $xy = 1^m$ e $z = 1^{p-m}\#1^p\#1^{2p}$, dove $m \in [1, p]$
- Inoltre, per la seconda condizione del lemma, si ha che $|y| > 0$, dunque necessariamente si ha che $x = 1^{m-k}$ e $y = 1^k$, dove $k \in [1, m]$
- A questo punto, consideriamo la stringa xy^0z. Notiamo immediatamente che:

$$xy^0z = 1^{m-k}(1^k)^0 1^{p-m}\#1^p\#1^{2p} = 1^{p-k}\#1^p\#1^{2p} \implies xy^0z \notin L$$

  contraddicendo la prima condizione del lemma per cui si ha che $\forall i \in \mathbb{N} \;\; xy^iz \in L$
- Dunque, ne segue necessariamente che L non sia regolare

□

Problema 1.7

Dato il linguaggio $L = \{c^4a^nb^m \mid n, m \in \mathbb{N}, n \geq m\}$, dimostrare che $L \notin \mathsf{REG}$

Dimostrazione.

- Supponiamo per assurdo che L sia regolare, implicando che per esso debba valere il pumping lemma, dove p è la lunghezza del pumping
- Consideriamo quindi la stringa $w := c^4a^pb^p \in L$. Poiché $|w| \geq p$, possiamo suddividerla in tre sottostringhe $x, y, z \in \Sigma^*$ tali che $w = xyz$
- Poiché la terza condizione del lemma impone che $|xy| \leq p$ e poiché $w := c^4a^pb^p$, ne segue che $xy = c^4a^k$ e $z = a^{p-k}b^p$, dove $k \in [0, p-4]$

- Per la seconda condizione del lemma, si ha che $|y| > 0$, dunque y necessariamente $\exists u \in \Sigma^*$ tale che $y = ua$, ossia y contiene almeno un simbolo a. Abbiamo quindi quattro sotto-casi:
  1. Se $x = c$ e $y = c^3a^k$ allora notiamo che $xy^0z = ca^{p-k}b^p \notin L$
  2. Se $x = c^2$ e $y = c^2a^k$ allora notiamo che $xy^0z = c^2a^{p-k}b^p \notin L$
  3. Se $x = c^3$ e $y = ca^k$ allora notiamo che $xy^0z = c^3a^{p-k}b^p \notin L$
  4. Se $x = c^4a^h$ e $y = a^{k-h}$, dove $h \in [0, k-1]$ allora notiamo che $xy^0z = c^4a^ha^{p-k}b^p = c^4a^{p-k+h}b^p \notin L$

  In altre parole, in ogni caso possibile viene contraddetta la prima condizione del lemma per cui si ha che $\forall i \in \mathbb{N} \;\; xy^iz \in L$
- Dunque, ne segue necessariamente che L non sia regolare

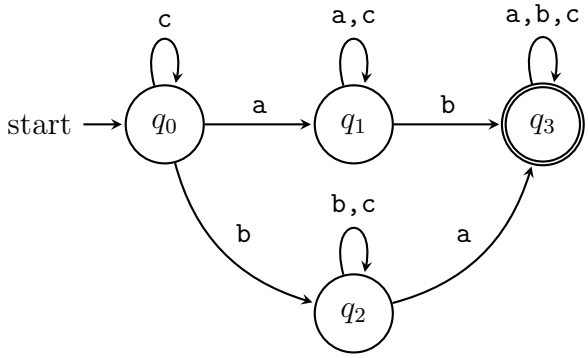
□

Problema 1.8

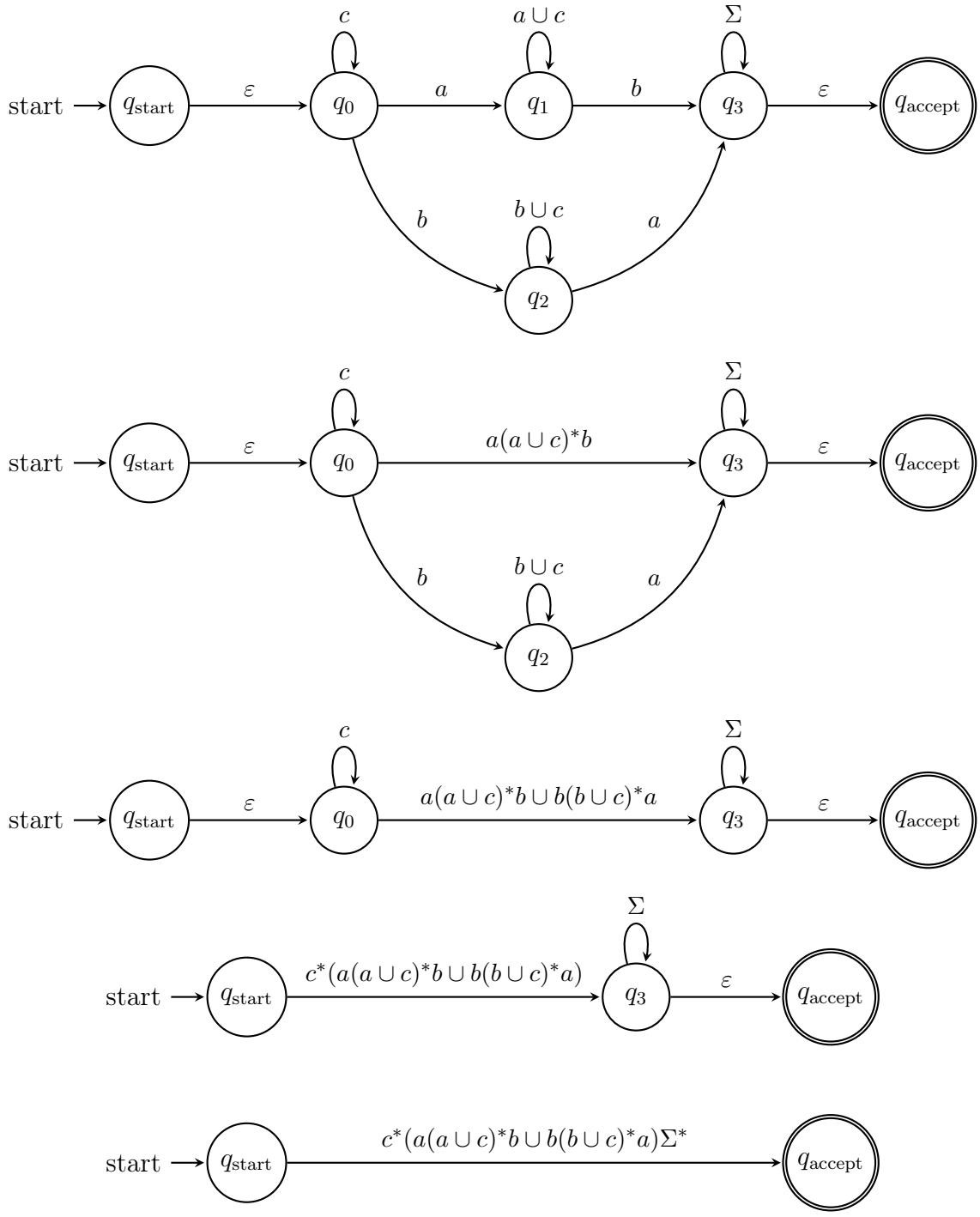
Sia $\Sigma = \{a, b, c\}$. Determinare un'espressione regolare $R \in \text{re}(\Sigma)$ descrivente il linguaggio di Σ composto dalle stringhe contenenti almeno una a ed almeno una b. Determinare inoltre un DFA D che riconosca lo stesso linguaggio.

Soluzione:

- Nonostante il problema inviti alla determinazione dell'espressione regolare e poi del DFA ad essa equivalente, trovare quest'ultimo risulta molto più rapido



- A questo punto, osservando il DFA possiamo già notare che l'espressione regolare ad esso equivalente corrisponda a $c^*(a(a \cup c)^*b \cup b(b \cup c)^*a)\Sigma^*$
- Volendo procedere più rigorosamente, possiamo ricavare tale espressione regolare convertendo il DFA costruito nel suo GNFA equivalente, per poi ridurre al minimo tale GNFA, ottenendo l'espressione regolare
- Definiamo quindi il GNFA equivalente (del quale vengono omesse le sue transizioni etichettate con $\varnothing$), per poi procedere con la sua riduzione:


start
q_{start}
ε
q_0
c
a
q_1
$a \cup c$
b
q_3
Σ
ε
q_{accept}
b
q_2
$b \cup c$
a
start
q_{start}
ε
q_0
c
$a(a \cup c)^*b$
q_3
Σ
ε
q_{accept}
b
q_2
$b \cup c$
a
start
q_{start}
ε
q_0
c
$a(a \cup c)^*b \cup b(b \cup c)^*a$
q_3
Σ
ε
q_{accept}
start
q_{start}
$c^*(a(a \cup c)^*b \cup b(b \cup c)^*a)$
q_3
Σ
ε
q_{accept}
start
q_{start}
$c^*(a(a \cup c)^*b \cup b(b \cup c)^*a)\Sigma^*$
q_{accept}

Problema 1.9

Sia $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ un DFA e si supponga che $\forall a \in \Sigma$ si abbia che $\delta(q_0, a) = \delta(q_f, a)$. Si dimostri che:

1. Per ogni $w \neq \varepsilon$, si ha che $\delta^*(q_0, w) = \delta^*(q_f, w)$, dove δ^* è la funzione di transizione estesa
2. Si mostri che se $x \neq \varepsilon \in L(A)$, allora $\forall k > 0 \in \mathbb{N}\ \ x^k \in L(A)$

Dimostrazione.

1. • Data $w := ay \in \Sigma^*$, dove $a \in \Sigma, y \in \Sigma^*$, si ha che:

$$\delta^*(q_0, w) = \delta^*(q_0, ay) = \delta^*(\delta(q_0, a), y) = \delta^*(\delta(q_f, a), y) = \delta^*(q_f, ay) = \delta^*(q_f, w)$$

2. • Data $x \neq \varepsilon \in L(A)$, per definizione stessa si ha che:

$$x \neq \varepsilon \in L(A) \iff \delta^*(q_0, x) \in \{q_f\} \iff \delta^*(q_0, x) = q_f$$

• Procediamo quindi per induzione su $k \in \mathbb{N}$

Caso base $(k = 1)$:

– $x \neq \varepsilon \in L(A) \implies x^1 \in L(A)$

Ipotesi induttiva:

– Per $k > 1 \in \mathbb{N}$, si ha che:

$$x \neq \varepsilon \in L(A) \implies x^k \in L(A)$$

Passo induttivo:

– Per $k + 1 \in \mathbb{N}$, notiamo che:

$$\delta^*(q_0, x^{k+1}) = \delta^*(q_0, xx^k) = \delta^*(\delta^*(q_0, x), x^k) = \delta^*(q_f, x^k)$$

a questo punto, tramite il punto 1. si ha che:

$$\delta^*(q_f, x^k) = \delta^*(q_0, x^k) = q_f \implies \delta^*(q_0, x^{k+1}) \in \{q_f\} \iff x^{k+1} \in L(A)$$

□

Problema 1.10: Somma di cifre come multiplo di k

Sia L_k il linguaggio di tutte le stringhe su alfabeto $\Sigma = \{0, 1, \ldots, 9\}$ la cui somma delle cifre è un multiplo di k, dove $k \in \mathbb{N}$. Descrivere formalmente un DFA che riconosca L_k per ogni $k \in \mathbb{N}$. Provare la correttezza del DFA proposto e disegnare il diagramma di transizione degli stati del DFA per il caso $k = 4$

Dimostrazione.

- Dato $k \in \mathbb{N}$, sia L_k il linguaggio richiesto, dove:

$$L_k = \{a_1 \ldots a_n \mid a_1 + \ldots + a_n \equiv 0 \pmod{k}\}$$

- Sia $D_k = (Q, \Sigma, \delta, q_0, F)$ il DFA definito come:
  - $Q = \{q_0, \ldots, q_{k-1}\}$
  - $F = \{q_0\}$
  - $\forall q_i, q_j \in Q$ e $\forall a \in \Sigma$ vale che:

$$\delta(q_i, a) = q_j \iff i + a \equiv j \pmod{k}$$

- Procediamo quindi per induzione su $n \in \mathbb{N}$:

  Caso base ($n = 0$):
  - Se $w = \varepsilon$, si ha che $\delta^*(q_0, \varepsilon) = q_0$ e che $0 \equiv 0 \pmod{k}$

  Ipotesi induttiva:
  - Per $n \in \mathbb{N}$, data $w = a_1 \ldots a_n \in \Sigma^*$ si ha che:

$$\delta^*(q_0, w) = q_m \iff a_1 + \ldots + a_n \equiv m \pmod{k}$$

  Passo induttivo:
  - Data $w = a_1 \ldots a_{n+1} \in \Sigma^*$, siano $q_m, q_h \in Q$ tali che:

$$\delta^*(q_0, a_1 \ldots a_n) = q_m \qquad \delta(q_m, a_{n+1}) = q_h$$

  - Per ipotesi induttiva, si ha che:

$$\delta^*(q_0, a_1 \ldots a_n) = q_m \iff a_1 + \ldots + a_n \equiv m \pmod{k}$$

    inoltre, per come è stata definita δ, abbiamo che:

$$\delta(q_m, a_{n+1}) = q_h \iff m + a_{n+1} \equiv h \pmod{k}$$

  - Di conseguenza, concludiamo che:

$$\delta^*(q_0, w) = q_h \iff \delta(\delta^*(q_0, a_1 \ldots a_n), a_{n+1}) = q_h \iff \delta(q_m, a_{n+1}) = q_h$$
$$\iff m + a_{k+1} \equiv h \pmod{k} \iff a_1 + \ldots + a_n + a_{k+1} \equiv h \pmod{k}$$

- A questo punto, si ha che:

$$w = a_1 \dots a_n \in L(D) \iff \delta^*(q_0, w) \in F \iff \delta^*(q_0, w) = q_0$$

$$\iff a_1 + \ldots + a_n \equiv 0 \pmod{k} \iff w \in L$$

implicando quindi che $L_k = L(D_k)$

□

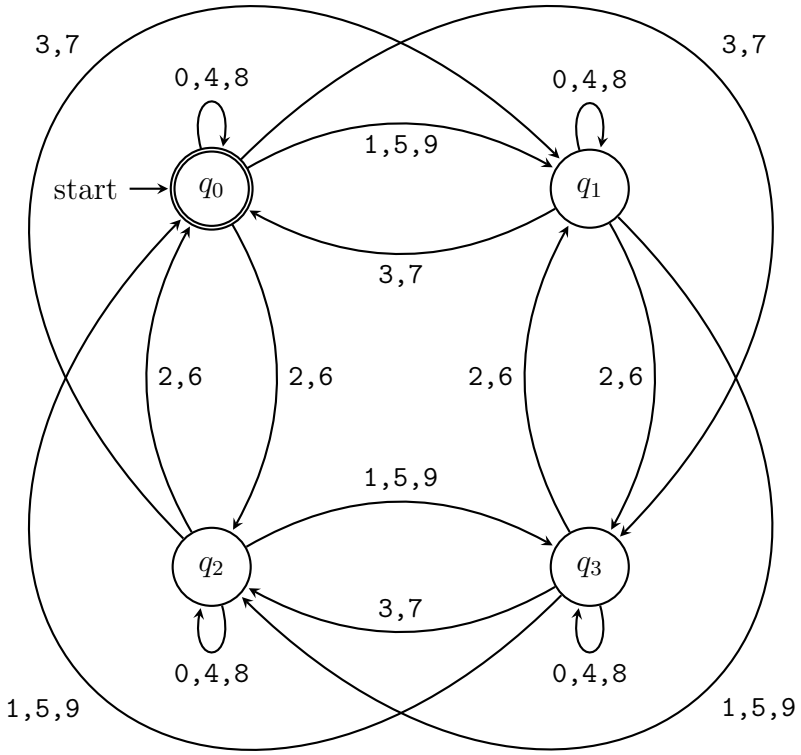


Diagramma del DFA D_4 dettato dalla dimostrazione

Problema 1.11: Linguaggio quozientato

Dati due linguaggi A e B, sia il quoziente su B del linguaggio A definito come:

$$A/B = \{w \in \Sigma^* \mid wx \in A \text{ per qualche } x \in B\}$$

Dimostrare che se $A \in \mathsf{REG}$ allora $A/B \in \mathsf{REG}$

Nota: B può anche non essere regolare

Dimostrazione.

- Dato $A \in \mathsf{REG}$, sia $D = (Q, \Sigma, \delta, q_0, F)$ il DFA tale che $A = L(D)$
- Consideriamo quindi il DFA $D' = \{Q, \Sigma, \delta, q_0, F'\}$, dove:

$$F' = \{q \in Q \mid \exists x \in B,\ \delta^*(q, x) \in F\}$$

- A questo punto, notiamo che:

$$w \in L(D') \iff \delta^*(q_0, w) \in F' \iff$$

$$\exists x \in B,\ \delta^*(\delta^*(q_0, w), x) = \delta^*(q_0, wx) \in F \iff w \in A/B$$

  concludendo che $A/B = L(D') \in \mathsf{REG}$

□

Problema 1.12

Dato il seguente linguaggio:

$$L = \{0^k w 0^k \mid k \in \mathbb{N}_{>0}, w \in \Sigma^*\}$$

dimostrare che $L \in \mathsf{REG}$

Dimostrazione.

- Consideriamo il linguaggio $L' = \{0w0 \mid w \in \Sigma^*\}$
- Data $x \in L$, si ha che:

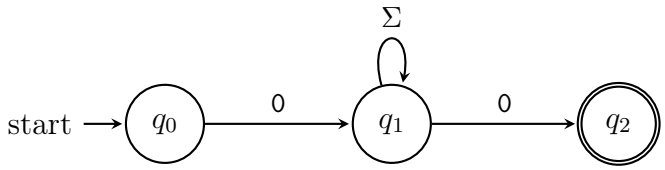
$$x \in L \iff \exists k \in \mathbb{N}_{>0}, w \in \Sigma^*, x = 0^k w 0^k$$

  di conseguenza, posto $y = 0^{k-1} w 0^{k-1} \in \Sigma^*$, si ha che $x = 0y0 \in L'$
- Viceversa, dato $x \in L'$, si ha che:

$$x \in L' \iff \exists w \in \Sigma^*, x = 0w0 = 0^1 w^1 \implies x \in L$$

- Di conseguenza, concludiamo che $L = L'$

- A questo punto, definiamo il seguente NFA N tale che $L = L' = L(N) \in$ REG:



□

Problema 1.13: Strict-NFA

Uno Strict-NFA è una quintupla $S = (Q, \Sigma, \delta, q_0 F)$ la cui funzione di transizione è non deterministica e dove una stringa $w \in \Sigma^*$ viene accettata da S se e solo se ogni ramo di computazione di $S(w)$ termina su uno stato in accettante.

Dimostrare che dato un linguaggio L si ha che:

$$L \in \text{REG} \iff \text{Esiste Strict-NFA che riconosce } L$$

Dimostrazione.

Prima implicazione.

- Dato $L \in$ REG, sia D il DFA tale che $L = L(D)$
- Notiamo che un DFA sia un particolare Strict-NFA dotato di un solo ramo di computazione. Di conseguenza, banalmente abbiamo che D stesso sia lo Strict-NFA che riconosce L

Seconda implicazione.

- Sia L un linguaggio riconosciuto dallo Strict-NFA $S = (Q_S, \Sigma, \delta_S, q_{0_S}, F_S)$
- Consideriamo quindi il DFA $D := (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$ costruito tramite S stesso:
  - $Q_D = \mathcal{P}(Q_S)$
  - Dato $R \in Q_D$, definiamo l'estensione di R come:

    $$E(R) = \{q \in Q_S \mid \exists p \in R \text{ che raggiunge } q \text{ in } S \text{ tramite solo } \varepsilon\text{-archi}\}$$

  - $q_{0_D} = E(\{q_{0_N}\})$
  - $F_D = \{R \in Q_D \mid R \subseteq F_S\}$
  - Dati $R \in Q_D$ e $a \in \Sigma$, definiamo δ_D come:

    $$\delta_D(R, a) = \bigcup_{r \in R} E(\delta_S(r, a))$$

- In particolare, notiamo che tale conversione sia identica alla conversione da NFA a DFA fatta eccezione del modo in cui l'insieme F_D è definito
- Per costruzione stessa di D si ha che:

$$w \in L = L(S) \iff w \in L(D)$$

implicando dunque che $L = L(S) = L(D) \in \mathsf{REG}$.

□

Problema 1.14

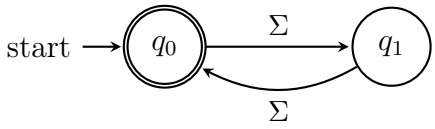
Dato il linguaggio $L = \{w \mid w \text{ termina con 0 oppure ha lunghezza pari}\}$, dimostrare che $L \in \mathsf{REG}$

Dimostrazione.

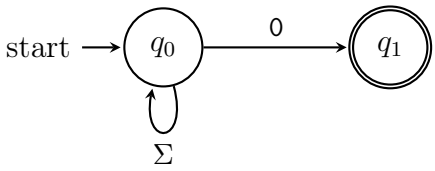
- Prima di tutto, notiamo che il linguaggio L proposto corrisponda all'unione dei seguenti due linguaggi:

$$L = \{w \mid \exists k \in \mathbb{N}, |w| = 2k\} \cup \{w \mid \exists y \in \Sigma^*, w = y0\}$$

- Siano quindi $L_1 = \{w \mid \exists k \in \mathbb{N}, |w| = 2k\}$ e $L_2 = \{w \mid \exists y \in \Sigma^*, w = y0\}$
- Definiamo il DFA D tale che $L_1 = L(D) \in \mathsf{REG}$



- Inoltre, definiamo il NFA N tale che $L_2 = L(N) \in \mathsf{REG}$



- Poiché $L_1, L_2 \in \mathsf{REG}$, per la Chiusura dell'unione in REG concludiamo che $L \in \mathsf{REG}$

□

2

Linguaggi acontestuali

2.1 Grammatiche acontestuali

Definizione 2.1: Context-free Grammar (CFG)

Una **Context-free Grammar (CFG)** (o *Grammatica acontestuale*) è una quadrupla (V, Σ, R, S) dove:

- V è l'**insieme delle variabili** della grammatica
- Σ è l'**insieme dei terminali** della grammatica e
- R è l'**insieme delle regole** o **produzioni** della grammatica
- $S \in V$ è la **variabile iniziale** della grammatica
- $V \cap \Sigma = \varnothing$, ossia variabili e terminali sono tutti distinti tra loro

Le **regole in** R assumono la forma $A \to X$, dove $A \in V$, ossia è una variabile, e $X \in (V \cup \Sigma_\varepsilon)^*$, ossia è una stringa composta da una o più variabili e/o terminali.

Esempio:

- La seguente quadrupla $G = (\{A, B\}, \{0, 1, \#\}, R, A)$ è una CFG dove in R sono definite le seguenti regole:

$$A \to 0A1$$

$$A \to B$$

$$B \to \#$$

Osservazione 2.1: Acontestualità

Con **acontestualità** intendiamo la condizione secondo cui il lato sinistro delle regole della grammatica è composto sempre e solo da **una singola variabile**.

Esempio:

- La regola $A \to B$ può appartenere ad una CFG
- La regola $AB \to B$ non può appartenere ad una CFG

Osservazione 2.2: Notazione contratta per le regole

Data una CFG $G = (V, \Sigma, R, S)$, se in R esistono più regole $A \to X_1, X_2, \ldots, A \to X_n$ definite sulla stessa variabile A,è possibile indicare tali regole con la seguente notazione contratta:

$$A \to X_1 \mid X_2 \mid \ldots \mid X_n$$

Esempio:

- Le regole della CFG dell'esempio precedente possono essere contratte in:

$$A \to 0A1 \mid B$$

$$B \to \#$$

Definizione 2.2: Produzione

Sia $G = (V, \Sigma, R, S)$ una CFG. Se u, v, w sono stringhe di variabili o terminali ed esiste la regola $A \to w$, allora la stringa uAv **produce** la stringa uwv, denotato come $uAv \Rightarrow uwv$.

$$u, v, w \in (V \cup \Sigma)^*, A \to w \in R \implies uAv \Rightarrow uwv$$

Esempio:

- Consideriamo la grammatica $G = (\{A, B\}, \{0, 1, \#\}, R, A)$ dove:

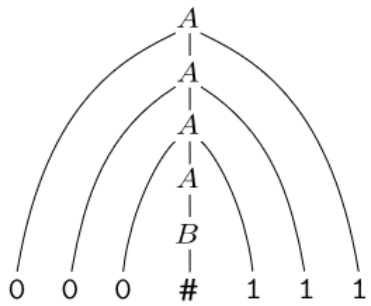
$$A \to 0A1 \mid B$$

$$B \to \#$$

- Tramite le regole di G è possibile ottenere la stringa 000#111 attraverso la seguente catena di produzioni:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000\#111$$

- Tale catena può anche essere descritta graficamente dal seguente **albero di produzione**:



Definizione 2.3: Derivazione

Sia $G = (V, \Sigma, R, S)$ un CFG. Date $u, v \in (V \cup \Sigma)^*$, diciamo che u **deriva** v, denotato come $u \overset{*}{\Rightarrow} v$, se $u = v$ oppure se $\exists u_1, \ldots, u_k \in (V \cup \Sigma)^*$ tali che:

$$u \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$$

Definizione 2.4: Context-free Language (CFL)

Sia $G = (V, \Sigma, R, S)$ una CFG. Definiamo come **Context-free Language (CFL)** (o *Linguaggio acontestuale*) **generato da** G, indicato come $L(G)$, l'insieme di stringhe derivate dalle regole di G tramite la variabile S:

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Rightarrow} w\}$$

Esempi:

1. Data la CFG $G = (\{S\}, \{\mathtt{a}, \mathtt{b}\}, R, S)$, dove:

$$S \to \varepsilon \mid aSb \mid SS$$

   si ha che:

   - $S \Rightarrow aSb \Rightarrow a\varepsilon b = ab$, dunque $ab \in L(G)$
   - $S \Rightarrow aSb \Rightarrow aaSbb \Longrightarrow aa\varepsilon bb = aabb$, dunque $aabb \in L(G)$
   - $S \Rightarrow SS \overset{*}{\Rightarrow} aSbaSb \overset{*}{\Rightarrow} a\varepsilon ba\varepsilon b = abab$, dunque $abab \in L(G)$

2. Data la CFG $G = (\{S, T\}, \{0, 1\}, R, S)$, dove:

$$S \to T1T1T1T$$

$$T \to \varepsilon \mid 0T \mid 1T$$

   si ha che:

$$L(G) = \{w \in \{0, 1\}^* \mid |w|_1 \geq 3\}$$

3. Data la CFG $G = (\{S\}, \{0,1\}, R, S)$, dove:

$$S \to \varepsilon \mid 0S0 \mid 1S1$$

si ha che:

$$L(G) = \{w \in \{0,1\}^* \mid w = w^R \wedge |w| \equiv 0 \pmod 2\}$$

4. Data la CFG $G = (\{S,T\}, \{\mathtt{a},\mathtt{b},\mathtt{c}\}, R, S)$, dove:

$$S \to aSc \mid T$$

$$T \to bTc \mid \varepsilon$$

si ha che:

$$L(G) = \{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^{i+j} \in \Sigma^* \mid i,j \in \mathbb{N}\}$$

Osservazione 2.3

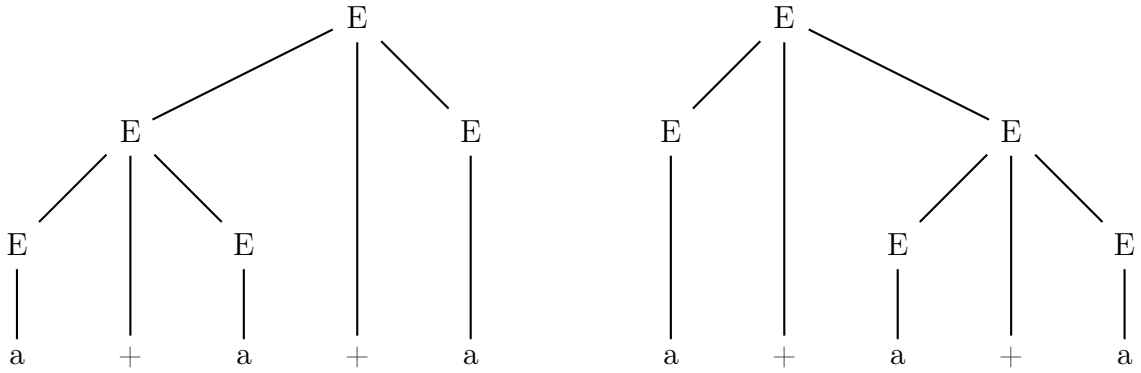
Sia G una CFG. Data la stringa $w \in L(G)$, possono esistere più derivazioni di w

Esempio:

- Data la CFG

$$E \to E + E \mid E \cdot E \mid (E) \mid a$$

la stringa $a + a + a$ può essere derivata in due modi:



Definizione 2.5: Derivazione a sinistra

Data una CFG $G = (V, \Sigma, R, S)$, definiamo la derivazione $S \overset{*}{\Rightarrow} w$ come **derivazione sinistra** se ad ogni produzione interna alla derivazione viene valutata la variabile più a sinistra

Esempio:

- Riprendiamo la CFG dell'esempio precedente:

$$E \to E + E \mid E \cdot E \mid (E) \mid a$$

- Per maggior chiarezza, riscriviamo tali regole come:

$$E \to E + F \mid E \cdot E \mid (E) \mid a$$

$$F \to E$$

  ottenendo una CFG del tutto equivalente alla precedente

- Una derivazione sinistra della stringa $a + a + a$ corrisponde a:

$$E \Rightarrow E{+}F \Rightarrow E{+}F{+}F \Rightarrow a{+}F{+}F \Rightarrow a{+}E{+}F \Rightarrow a{+}a{+}F \Rightarrow a{+}a{+}E \Rightarrow a{+}a{+}a$$

Osservazione 2.4

L'uso delle derivazioni a sinistra permette di fissare un "ordine", rimuovendo la maggior parte delle derivazioni multiple per una stessa stringa.

Tuttavia, in alcune grammatiche possono esistere più di una derivazione a sinistra per la stessa stringa.

Definizione 2.6: Grammatica ambigua

Definiamo una grammatica G come **ambigua** se $\exists w \in L(G)$ tale che esistono almeno due derivazioni a sinistra per w

2.2 Linguaggi acontestuali ad estensione dei regolari

Definizione 2.7: Classe dei linguaggi acontestuali

Dato un alfabeto Σ, definiamo come **classe dei linguaggi acontestuali di** Σ il seguente insieme:

$$\mathsf{CFL} = \{L \subseteq \Sigma^* \mid \exists\ \mathsf{CFG}\ G \text{ t.c } L = L(G)\}$$

Lemma 2.1: Conversione da DFA a CFG

Date le due classi dei linguaggi REG e CFL, si ha che:

$$\mathsf{REG} \subseteq \mathsf{CFL}$$

Dimostrazione.

- Dato $L \in \mathsf{REG}$, sia $D = (Q, \Sigma, \delta, q_0, F)$ il DFA tale che $L = L(D)$
- Consideriamo quindi la CFG $G = (V, \Sigma, R, S)$ tale che:
  - Esiste una funzione biettiva $\varphi : Q \to V : q_i \mapsto V_i$

  - $S = \varphi(q_0) = V_0$
  - Dati $q_i, q_j \in Q$ e $a \in \Sigma$, si ha che:

$$\delta(q_i, a) = q_j \implies \varphi(q_i) \to a\varphi(q_j) \implies V_i \to aV_j$$

  - $q_f \in F \implies \varphi(q_f) \to \varepsilon \implies V_f \to \varepsilon$

- A questo punto, per costruzione stessa di G si ha che:

$$w \in L(D) \iff w \in L(G)$$

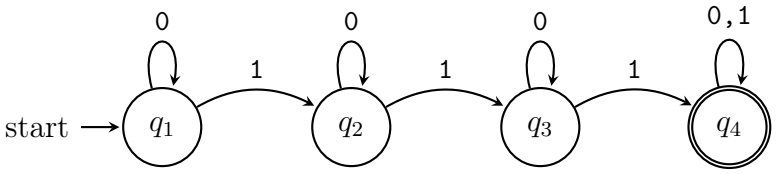
  implicando dunque che $L(D) \in \mathsf{CFL}$ e di conseguenza che:

$$\mathsf{REG} \subseteq \mathsf{CFL}$$

□

Esempio:

- Consideriamo il seguente DFA



- Una CFG $G = (V, \Sigma, R, S)$ equivalente è costituita da:
  - $V = \{V_1, V_2, V_3, V_4\}$
  - $S = V_1$
  - R definito come:

$$\begin{aligned} V_1 &\to 0V_1 \mid 1V_2 \\ V_2 &\to 0V_2 \mid 1V_3 \\ V_3 &\to 0V_3 \mid 1V_4 \\ V_4 &\to 0V_4 \mid 1V_4 \mid \varepsilon \end{aligned}$$

- Difatti, sia il DFA sia la CFG descrivono il seguente linguaggio:

$$L = \{w \in \Sigma^* \mid |w|_1 \geq 3\}$$

Teorema 2.1: Ling. acontestuali estensione dei ling. regolari

Date le due classi dei linguaggi REG e CFL, si ha che:

$$\mathsf{REG} \subsetneq \mathsf{CFL}$$

Dimostrazione.

- Tramite la Conversione da DFA a CFG, sappiamo che $\mathsf{REG} \subseteq \mathsf{CFL}$
- Consideriamo quindi il linguaggio $L = \{0^n 1^n \mid n \in \mathbb{N}\}$
- Tale linguaggio è generabile dalla grammatica $G = (\{S\}, \{0,1\}, R, S)$, dove:

  $$S \to 0S1 \mid \varepsilon$$

  dunque abbiamo che $L = L(G) \in \mathsf{CFL}$
- Tuttavia, abbiamo già dimostrato nella sezione 1.6 che L non sia regolare, dunque abbiamo che $L \notin \mathsf{REG}$
- Di conseguenza, concludiamo che:

  $$\mathsf{REG} \subsetneq \mathsf{CFL}$$

□

2.3 Forma normale di Chomsky

Definizione 2.8: Chomsky's Normal Form (CNF)

Una CFG $G = (V, \Sigma, R, S)$ viene detta in **Chomsky's Normal Form (CNF)** (o *Forma Normale di Chomsky*) se tutte le regole in R assumono una delle seguenti tre forme:

$$A \to BC \qquad A \to a \qquad S \to \varepsilon$$

dove $A \in V$, $a \in \Sigma$ e $B, C \in V - \{S\}$

Teorema 2.2: Conversione in Forma Normale di Chomsky

Per ogni CFG G, si ha che:

$$\exists \text{ CFG } G' \text{ in CNF } \mid L(G) = L(G')$$

Dimostrazione.

- Data una CFG $G = (V, \Sigma, R, S)$, costruiamo una CFG G' in CNF equivalente a G:
  1. Vengono aggiunte una variabile S_0 e una regola $S_0 \to S$, dove S_0 è la **nuova variabile iniziale**
  2. Finché in R esiste una **ε-regola** $A \to \varepsilon$ dove $A \in V - \{S_0\}$, tale regola viene **eliminata** e per ogni regola in R contenente delle occorrenze di A vengono **aggiunte** delle regole in cui vengono eliminate tutte le possibili combinazioni di occorrenze di A

     (es: se viene rimossa $A \to \varepsilon$ e in R esiste $B \to uAvAw \mid u, v, w \in (V \cup \Sigma)^*$, vengono aggiunte le regole $B \to uvAw \mid uAvw \mid uvw$)
  3. Ogni regola nella forma $A \to B$ (dette **regole unitarie**) viene **eliminata** e ogni regola nella forma $B \to u$, dove $u \in (V \cup \Sigma)^*$, viene **sostituita** da una regola $A \to u$
  4. Per ogni regola $A \to u_1 \dots u_k$ dove $k \geq 3$ e $\forall i \in [1, k] \;\; u_i \in (V \cup \Sigma)$, vengono **aggiunte** le variabili $A_1, \dots, A_{k-2}$ e le seguenti regole:

     $$A \to u_1 A_1 \qquad A_1 \to u_2 A_2 \qquad \dots \qquad A_{k-3} \to u_{k-2} A_{k-2} \qquad A_{k-2} \to u_{k-1} u_k$$

     per poi **eliminare** la regola iniziale $A \to u_1 u_2 \dots u_k$
  5. Per ogni regola rimanente nella forma $A \to u_1 u_2$ dove $u_1, u_2 \in (V \cup \Sigma)$, se $u_1 \in \Sigma$ allora viene **aggiunta** una variabile U_1 ed una regola $U_1 \to u_1$, **sostituendo** la regola $A \to u_1 u_2$ con la regola $A \to U_1 u_2$. Analogamente, lo stesso viene svolto se $u_2 \in \Sigma$.
- Poiché le operazioni svolte dall'algoritmo non modificano le stringhe generabili dalla CFG, ne segue automaticamente che $L(G) = L(G')$

□

Esempio:

- Consideriamo la seguente grammatica G non in CNF, dove S è la variabile iniziale:

$$\begin{array}{rlll} G: & S & \to & ASA \mid aB \\ & A & \to & B \mid S \\ & B & \to & b \mid \varepsilon \end{array}$$

- Aggiungiamo la nuova variabile iniziale S_0 e la regola $S_0 \to S$:

$$\begin{array}{rlll} G: & \boldsymbol{S_0} & \boldsymbol{\to} & \boldsymbol{S} \\ & S & \to & ASA \mid aB \\ & A & \to & B \mid S \\ & B & \to & b \mid \varepsilon \end{array}$$

- Eliminiamo la ε-regola $B \to \varepsilon$:

$$\begin{array}{rlll} G: & S_0 & \to & S \\ & S & \to & ASA \mid aB \mid \boldsymbol{a} \\ & A & \to & B \mid S \mid \boldsymbol{\varepsilon} \\ & B & \to & b \mid \varepsilon \end{array}$$

- Eliminiamo la ε-regola $A \to \varepsilon$:

$$\begin{array}{rlll} G: & S_0 & \to & S \\ & S & \to & ASA \mid aB \mid a \mid \boldsymbol{SA} \mid \boldsymbol{AS} \mid \boldsymbol{S} \\ & A & \to & B \mid S \mid \varepsilon \\ & B & \to & b \end{array}$$

- Eliminiamo la regola unitaria $S \to S$:

$$\begin{array}{rlll} G: & S_0 & \to & S \\ & S & \to & ASA \mid aB \mid a \mid SA \mid AS \mid \cancel{S} \\ & A & \to & B \mid S \\ & B & \to & b \end{array}$$

- Eliminiamo la regola unitaria $S_0 \to S$:

$$\begin{array}{rlll} G: & S_0 & \to & \cancel{S} \mid \boldsymbol{ASA} \mid \boldsymbol{aB} \mid \boldsymbol{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ & S & \to & ASA \mid aB \mid a \mid SA \mid AS \\ & A & \to & B \mid S \\ & B & \to & b \end{array}$$

- Eliminiamo le regole unitarie $A \to B$ e $A \to S$:

$$\begin{array}{rlll} G: & S_0 & \to & ASA \mid aB \mid a \mid SA \mid AS \\ & S & \to & ASA \mid aB \mid a \mid SA \mid AS \\ & A & \to & \cancel{B} \mid \cancel{S} \mid \boldsymbol{b} \mid \boldsymbol{ASA} \mid \boldsymbol{aB} \mid \boldsymbol{a} \mid \boldsymbol{SA} \mid \boldsymbol{AS} \\ & B & \to & b \end{array}$$

- Separiamo ogni regola con tre o più elementi a destra in regole con massimo due elementi a destra:

$$\begin{array}{rlll} G: & S_0 & \to & \cancel{ASA} \mid \boldsymbol{AA_1} \mid aB \mid a \mid SA \mid AS \\ & S & \to & \cancel{ASA} \mid \boldsymbol{AA_1} \mid aB \mid a \mid SA \mid AS \\ & A & \to & b \mid \cancel{ASA} \mid \boldsymbol{AA_1} \mid aB \mid a \mid SA \mid AS \\ & \boldsymbol{A_1} & \to & \boldsymbol{SA} \\ & B & \to & b \end{array}$$

- Infine, convertiamo tutte le regole aventi due elementi a destra di cui almeno uno è un terminale:

$$\begin{array}{rlll} G: & S_0 & \to & AA_1 \mid \cancel{aB} \mid \boldsymbol{UB} \mid a \mid SA \mid AS \\ & S & \to & AA_1 \mid \cancel{aB} \mid \boldsymbol{UB} \mid a \mid SA \mid AS \\ & A & \to & b \mid AA_1 \mid \cancel{aB} \mid \boldsymbol{UB} \mid a \mid SA \mid AS \\ & A_1 & \to & SA \\ & \boldsymbol{U} & \to & \boldsymbol{a} \\ & B & \to & b \end{array}$$

- La grammatica finale ottenuta risulta sia equivalente a quella iniziale sia in forma normale di Chomsky:

$$\begin{array}{rlll} G: & S_0 & \to & AA_1 \mid UB \mid a \mid SA \mid AS \\ & S & \to & AA_1 \mid UB \mid a \mid SA \mid AS \\ & A & \to & b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\ & A_1 & \to & SA \\ & U & \to & a \\ & B & \to & b \end{array}$$

2.4 Automi a pila

Definizione 2.9: Pushdown Automaton (PDA)

Un **Pushdown Automaton (PDA)** (o *Automa a pila*) è una sestupla $(Q, \Sigma, \Gamma, \delta, q_0, F)$ dove:

- Q è l'**insieme finito degli stati** dell'automa
- Σ è l'**alfabeto** dell'automa
- Γ è l'**alfabeto** dello stack (o *pila*) dell'automa
- $q_0 \in Q$ è lo **stato iniziale** dell'automa
- $F \subseteq Q$ è l'**insieme degli stati accettanti** dell'automa
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ è la **funzione di transizione** dell'automa, dove se $(q, c) \in \delta(p, a, b)$ si ha che:
  - Viene letto il simbolo a dalla stringa in input e se il simbolo b è in cima allo stack allora l'automa passa dallo stato p allo stato q e il simbolo b viene sostituito dal simbolo c
  - L'etichetta della transizione da p a q viene indicata come $a;\ b \to c$

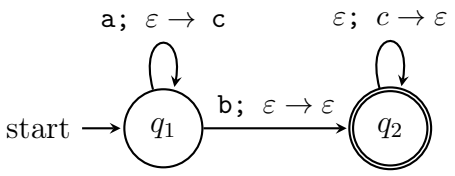
Osservazione 2.5

Dato $(q, c) \in \delta(p, a, b)$ dove δ è la funzione di transizione di un PDA, si ha che:

- Se $b, c = \varepsilon$ (dunque $a;\ \varepsilon \to \varepsilon$) allora l'automa leggerà a dalla stringa e passerà direttamente dallo stato p allo stato q, senza modificare lo stack
- Se $b = \varepsilon$ e $c \neq \varepsilon$ (dunque $a;\ \varepsilon \to c$) allora l'automa leggerà a dalla stringa, passerà direttamente dallo stato p allo stato q e in cima allo stack viene aggiunto il simbolo c (**push**)
- Se $b \neq \varepsilon$ e $c = \varepsilon$ (dunque $a;\ b \to \varepsilon$) allora l'automa leggerà a e se in cima allo stack vi è b, l'automa passerà dallo stato p allo stato q e rimuoverà b dalla cima dello stack (**pop**)

Esempio:

- Consideriamo il seguente PDA:



- Data la stringa `aab`, uno dei possibili rami di computazione del PDA procede nel seguente ordine:
  1. Viene letta la prima `a` e viene inserita la prima `c` in cima allo stack, rimanendo nello stato q_1.
  2. Viene letta la seconda `a` e viene inserita la seconda `c` in cima allo stack, rimanendo nello stato q_1.
  3. Viene letta la `b`, passando da q_1 a q_2 e lasciando lo stack inalterato
  4. Viene "letta" la prima ε, rimuovendo la seconda `c` dallo stack (poiché essa è in cima), rimanendo nello stato q_2.
  5. Viene "letta" la seconda ε, rimuovendo la prima `c` dallo stack (poiché essa è in cima), rimanendo nello stato q_2.
  6. Sia la stringa che lo stack sono vuoti, dunque la computazione termina necessariamente poiché non vi sono transizioni percorribili
- Notiamo in particolare che, in tal caso, la stringa verrebbe accettata anche se la computazione si fermasse al terzo passo
- Difatti, lo stack non deve necessariamente esser vuoto affinché la stringa possa essere accettata

Proposizione 2.1: Stringa accettata in un PDA

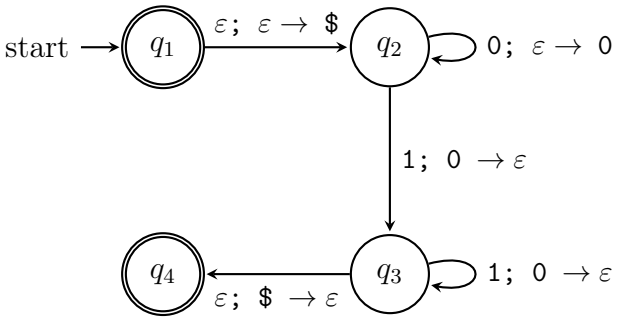
Sia $P := (Q, \Sigma, \Gamma, \delta, q_0, F)$ un PDA. Data una stringa $w := w_0 \dots w_k \in \Sigma^*$, dove $w_0, \dots, w_k \in \Sigma_\varepsilon$, diciamo che w è **accettata da** P se esiste una sequenza di stati $r_0, r_1, \dots, r_{k+1} \in Q$ ed una sequenza di stringhe $s_1, \dots, s_n \in \Gamma^*$ tali che:

- $r_0 = q_0$
- $r_{k+1} \in F$
- $s_0 = \varepsilon$, dunque lo stack è inizialmente vuoto
- $\forall i \in [0, k]$ si abbia che:
  - $(r_{i+1}, b) \in \delta(r_i, w_i, a)$
  - $s_i = at$
  - $s_{i+1} = bt$

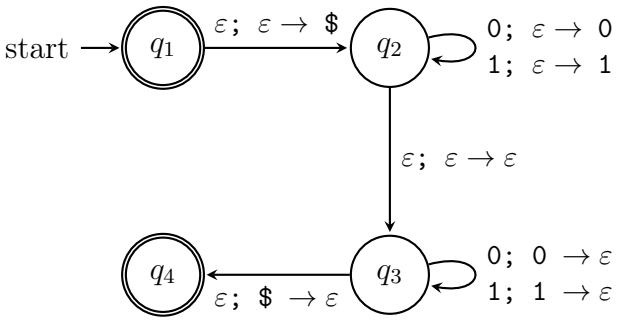
  dove $a, b \in \Gamma_\varepsilon$ e dove $t \in \Gamma^*$ è la stringa composta dai caratteri nello stack

Esempi:

- Il seguente automa riconosce il linguaggio $L = \{0^n 1^n \mid n \in \mathbb{N}\}$



- Il seguente automa riconosce il linguaggio $L = \{ww^R \mid w \in \{0,1\}^*\}$



2.4.1 Equivalenza tra CFG e PDA

Definizione 2.10: Classe dei linguaggi riconosciuti da un PDA

Dato un alfabeto Σ, definiamo come **classe dei linguaggi di Σ riconosciuti da un PDA** il seguente insieme:

$$\mathcal{L}(\text{PDA}) = \{L \subseteq \Sigma^* \mid \exists \text{ PDA } P \text{ t.c } L = L(P)\}$$

Proposizione 2.2: Scrittura di una stringa sullo stack

Sia $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ un PDA. Dati $u_1, \dots, u_k \in \Gamma$, introduciamo una notazione per cui δ possa ammettere la scrittura diretta sullo stack della stringa $u := u_1 \dots u_k$.

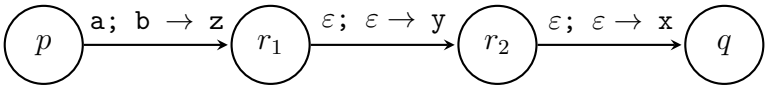
Formalmente, diciamo che:

$$(q, u_1 \dots u_k) \in \delta(p, a, b) \iff \exists r_1, \dots, r_{k-1} \in Q \text{ tali che:}$$

- $\delta(p, a, b) \ni (r_1, u_k)$
- $\delta(r_1, \varepsilon, \varepsilon) = \{(r_2, u_{k-1})\}$
- $\dots$
- $\delta(r_{k-1}, \varepsilon, \varepsilon) = \{(q, u_1)\}$

Esempio:

- Dato $(q, xyz) \in \delta(p, a, b)$ si ha che:



Lemma 2.2: Conversione da CFG a PDA

Date le due classi dei linguaggi CFL e $\mathcal{L}$(PDA), si ha che:

$$\text{CFL} \subseteq \mathcal{L}(\text{PDA})$$

Dimostrazione.

- Dato $L \in$ CFL, sia $G = (V, \Sigma, R, S)$ la CFG tale che $L = L(G)$
- Consideriamo quindi il PDA $P = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, F)$ tale che:
  - $Q = \{q_{\text{start}}, q_{\text{loop}}, q_{\text{accept}}\} \cup Q_\delta$, dove Q_δ sono i minimi stati aggiunti affinché la sua funzione δ sia ben definita (vedi i punti successivi)
  - $\Gamma = V \cup \Sigma$

- $F = \{q_{\text{accept}}\}$
- Dato $q_{\text{start}} \in Q$ si ha che

$$\delta(q_{\text{start}}, \varepsilon, \varepsilon) = \{(q_{\text{loop}}, S\$)\}$$

- $\forall A \in V$ si ha che

$$\delta(q_{\text{loop}}, \varepsilon, A) = \{(q_{\text{loop}}, u) \mid (A \to u) \in R,\ u \in \Gamma^*\}$$

- $\forall a \in \Sigma$ si ha che

$$\delta(q_{\text{loop}}, a, a) = \{(q_{\text{loop}}, \varepsilon)\}$$

- Dato $q_{\text{accept}} \in Q$ si ha che

$$\delta(q_{\text{loop}}, \varepsilon, \$) = \{(q_{\text{accept}}, \varepsilon)\}$$

- A questo punto, per costruzione stessa di P si ha che:

$$w \in L = L(G) \iff w \in L(P)$$

dunque che $L = L(P) \in \mathcal{L}(\mathsf{PDA})$

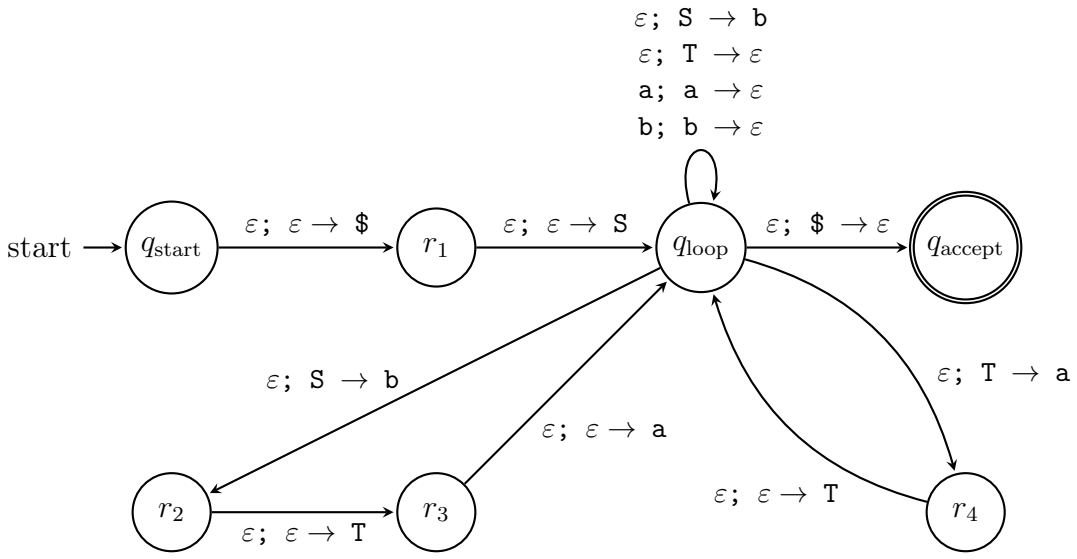
□

Esempio:

- Consideriamo la seguente grammatica:

$$\begin{aligned} G: \quad & S \to aTb \mid b \\ & T \to Ta \mid \varepsilon \end{aligned}$$

- Il PDA in grado di riconoscere $L(G)$ corrisponde a:



Lemma 2.3: Conversione da PDA a CFG

Date le due classi dei linguaggi $\mathcal{L}(\mathsf{PDA})$ e CFL, si ha che:

$$\mathcal{L}(\mathsf{PDA}) \subseteq \mathsf{CFL}$$

Dimostrazione.

- Dato $L \in \mathcal{L}(\mathsf{PDA})$, sia $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$ il PDA tale che $L = L(P)$
- Consideriamo il PDA $P' = (Q', \Sigma, \Gamma, \delta', q_0, \{q_{\text{accept}}\})$ tale che:
  - Ogni transizione effettua solo un'operazione di push o di pop, ma mai una sostituzione diretta:
    $$(q, c) \in \delta(p, a, b) \implies \exists r \in Q' \mid (r, \varepsilon) \in \delta'(p, a, b) \land \delta'(r, \varepsilon, \varepsilon) = \{(q, c)\}$$
  - $Q' = Q \cup Q_{\delta'} \cup \{q_{\text{accept}}\}$, dove $Q_{\delta'}$ sono gli stati aggiunti per il punto precedente
  - $q_{\text{accept}} \in Q'$ è il nuovo unico stato accettante:
    $$\forall q \in F \;\; (q_{\text{accept}}, \varepsilon) \in \delta'(q, \varepsilon, \varepsilon)$$
  - Lo stack deve essere svuotato prima di poter accettare una stringa:
    $$\forall q \in F, a \in \Sigma \;\; (q, \varepsilon) \in \delta'(q, \varepsilon, a)$$
- A questo punto, per costruzione stessa di P' si ha che:
  $$w \in L(P) \iff w \in L(P')$$
  dunque che $L = L(P) = L(P')$
- Consideriamo quindi la CFG $G = (V, \Sigma, R, S)$ tale che:
  - $V = \{A_{p,q} \mid p, q \in Q'\}$
  - $S = A_{q_0, q_{\text{accept}}}$
  - Ogni variabile $A_{p,q}$ è grado di derivare tutte le stringhe generabili passando dallo stato p allo stato q:
    - $\forall p \in Q'$ si ha che:
      $$(A_{p,p} \to \varepsilon) \in R$$
    - $\forall p, q, r, s \in Q', u \in \Gamma$ e $a, b \in \Sigma_\varepsilon$ si ha che:
      $$(r, u) \in \delta'(p, a, \varepsilon) \land (q, \varepsilon) \in \delta(s, b, u) \iff (A_{p,q} \to a A_{r,s} b) \in R$$
    - $\forall p, q, r \in Q'$ si ha che:
      $$(A_{p,q} \to A_{p,r} A_{r,q}) \in R$$

- **Affermazione**: dati $p, q \in Q'$ e $x \in \Sigma^*$, se $A_{p,q} \overset{*}{\Rightarrow} x$ allora x porta il PDA P' dallo stato p allo stato q con uno stack vuoto:

  Dimostrazione.

  Procediamo per induzione sul numero n di produzioni che compongono la derivazione $A_{p,q} \overset{*}{\Rightarrow} x$

  Caso base.

  – Per $n = 1$, la derivazione è composta da una sola produzione. Di conseguenza, l'unica regola possibile affinché $A_{p,q} \Rightarrow x$ è la regola $A_{p,q} \to \varepsilon$, implicando che $p = q$ e che $x = \varepsilon$, dunque la stringa x porta correttamente il PDA P' dallo stato p allo stato q con uno stack vuoto

  Ipotesi induttiva forte.

  – Assumiamo che per ogni stringa $x \in \Sigma^*$ derivabile da $A_{p,q}$ (dunque tale che $A_{p,q} \overset{*}{\Rightarrow} x$) tramite $k \leq n$ produzioni, tale stringa x porti il PDA P' da p a q con uno stack vuoto

  Passo induttivo.

  – Consideriamo la derivazione $A_{p,q} \overset{*}{\Rightarrow} x$ composta da $n + 1$ produzioni. Poiché tale derivazione è composta da almeno due produzioni, la prima produzione deve essere necessariamente data dalla regola $A_{p,q} \to aA_{r,s}b$ o dalla regola $A_{p,q} \to A_{p,r}A_{r,q}$

    (a) Consideriamo il caso in cui $A_{p,q} \Rightarrow aA_{r,s}b \overset{*}{\Rightarrow} x$.

    Sia $x = ayb$, dove $A_{r,s} \overset{*}{\Rightarrow} y$. Poiché $A_{r,s} \overset{*}{\Rightarrow} y$ è composta da n produzioni, per ipotesi induttiva la stringa y porta il PDA P' da r ad s con uno stack vuoto.

    Inoltre, per costruzione stessa di G, tale regola di derivazione si ha che:

    $$(r, u) \in \delta'(p, a, \varepsilon) \land (q, \varepsilon) \in \delta(s, b, u) \iff (A_{p,q} \to aA_{r,s}b) \in R$$

    dunque concludiamo che:

    $$\left.\begin{array}{l} a \text{ porta } P' \text{ da } p \text{ in } r \\ y \text{ porta } P' \text{ da } r \text{ in } s \\ b \text{ porta } P' \text{ da } s \text{ in } q \end{array}\right\} \implies x = ayb \text{ porta } P' \text{ da } p \text{ in } q$$

    (b) Consideriamo il caso in cui $A_{p,q} \Rightarrow A_{p,r}A_{r,q} \overset{*}{\Rightarrow} x$.

    Sia $x = yz$, dove $A_{p,r} \overset{*}{\Rightarrow} y$ e $A_{r,q} \overset{*}{\Rightarrow} z$. Poiché $A_{p,r} \overset{*}{\Rightarrow} y$ è composta da $m \leq n$ produzioni e $A_{r,q} \overset{*}{\Rightarrow} z$ da $n - m \leq n$ produzioni, per ipotesi induttiva le stringhe y e z portano il PDA P' rispettivamente da p ad r e da r a q con uno stack vuoto, dunque concludiamo che:

    $$\left.\begin{array}{l} y \text{ porta } P' \text{ da } p \text{ in } r \\ z \text{ porta } P' \text{ da } r \text{ in } q \end{array}\right\} \implies x = yz \text{ porta } P' \text{ da } p \text{ in } q$$

- **Affermazione**: dati $p, q \in Q'$ e $x \in \Sigma^*$, se la stringa x porta il PDA P' dallo stato p allo stato q con uno stack vuoto allora $A_{p,q} \overset{*}{\Rightarrow} x$

  Dimostrazione.

  Procediamo per induzione sul numero n di transizioni percorse da P' durante la lettura di x

  Caso base.

  – Per $n = 0$, il PDA percorre zero transizioni, dunque $x = \varepsilon$ e x porta il PDA da p a p. Pertanto, la regola $A_{p,p} \to \varepsilon$ soddisfa la derivazione $A_{p,p} \Rightarrow x$

  Ipotesi induttiva forte.

  – Assumiamo che per ogni stringa $x \in \Sigma^*$ che porta il PDA P' da p a q con uno stack vuoto percorrendo $k \leq n$ transizioni, si abbia che $A_{p,q} \overset{*}{\Rightarrow} x$

  Passo induttivo.

  – Consideriamo la stringa $x \in \Sigma^*$ che porta il PDA P' da p a q con uno stack vuoto percorrendo $n + 1$ transizioni. A seconda dell'evolvere dello stack durante la computazione, abbiamo due casi:

    (a) Se lo stack risulta vuoto solo all'inizio e alla fine della computazione, ciò implica che $\exists u \in \Gamma$ inserito nella prima transizione e rimosso solo nell'ultima.

    Sia quindi $a \in \Sigma_\varepsilon$ il simbolo letto durante tale prima transizione. In tal caso, $\exists r, s \in Q'$ tali che:

    $$(r, u) \in \delta(p, a, \varepsilon) \wedge (q, \varepsilon) \in \delta(s, b, u)$$

    Sia quindi $x = ayb$, dove y è una stringa che porta P' da r a s. Affinché la computazione di x termini con lo stack vuoto, è necessario che ciò valga anche per la computazione di y.

    Poiché la computazione di y percorre $n - 1$ transizioni, per ipotesi induttiva abbiamo che $A_{r,s} \overset{*}{\Rightarrow} y$, dunque data la regola $A_{p,q} \to aA_{r,s}b$ concludiamo che:

    $$A_{p,q} \Rightarrow aA_{r,s}b \overset{*}{\Rightarrow} ayb = x$$

    (b) Se lo stack si svuota durante la computazione, ciò implica che $\exists r \in Q'$ percorso durante la computazione di x in cui ciò accade.

    Sia quindi $x = yz$, dove y e z sono due stringhe che portano P' rispettivamente da p a r e da r a q.

    Poiché le computazioni di y e z percorrono rispettivamente $m \leq n$ e $n - m \leq n$ transizioni, per ipotesi induttiva abbiamo che $A_{p,r} \overset{*}{\Rightarrow} y$ e $A_{r,q} \overset{*}{\Rightarrow} z$, dunque data la regola $A_{p,q} \to A_{p,r}A_{r,q}$ concludiamo che:

    $$A_{p,q} \Rightarrow A_{p,r}A_{r,q} \overset{*}{\Rightarrow} yz = x$$

- Tramite le due affermazioni, abbiamo che:

$$A_{q_0,q_{\text{accept}}} \overset{*}{\Rightarrow} x \iff x \text{ porta } P' \text{ da } q_0 \text{ in } q_{\text{accept}} \text{ con uno stack vuoto}$$

da cui concludiamo che:

$$x \in L(G) \iff A_{q_0,q_{\text{accept}}} \overset{*}{\Rightarrow} x \iff x \in L(P')$$

dunque che $L = L(P) = L(P') = L(G) \in \mathsf{CFL}$

□

Teorema 2.3: Equivalenza tra CFG e PDA

Date le due classi dei linguaggi $\mathcal{L}(\mathsf{PDA})$ e CFL, si ha che:

$$\mathcal{L}(\mathsf{PDA}) = \mathsf{CFL}$$

(*segue dai due lemmi precedenti*)

2.5 Pumping lemma per i linguaggi acontestuali

Proposizione 2.3: Altezza delle derivazioni in una CFG in CNF

Sia $G = (V, \Sigma, R, S)$ una CFG in CNF. Data $x \in L(G)$ e data l'altezza h dell'albero di derivazione di x, si ha che $|x| \leq 2^{h-1}$

Dimostrazione. Procediamo per induzione sull'altezza h dell'albero di $S \overset{*}{\Rightarrow} x$

Caso base.

- Per $h = 1$, la derivazione è composta da una sola produzione. Essendo G in CNF, l'unica regola applicabile è nella forma $S \to a$, dove $x = a \in \Sigma$, implicando che $|x| = 1 \leq 2^{1-1} = 1$

Ipotesi induttiva forte.

- Assumiamo che data $x \in L(G)$ tale che il suo albero di derivazione abbia altezza $k \leq h$ si abbia che $|x| \leq 2^{h-1}$

Passo induttivo.

- Consideriamo la stringa x il cui albero di derivazione ha altezza $h+1$. Poiché G è in CNF, la prima produzione di tale derivazione deve essere ottenuta tramite una regola nella forma $S \to AB$.

- Sia quindi $x = yz$, dove $A \overset{*}{\Rightarrow} y$ e $B \overset{*}{\Rightarrow} z$. Poiché la derivazione $S \Rightarrow AB \overset{*}{\Rightarrow} yz = x$ ha altezza $h + 1$, ne segue che l'altezza dei due sottoalberi delle derivazioni $A \overset{*}{\Rightarrow} y$ e $B \overset{*}{\Rightarrow} z$ sia h
- Di conseguenza, per ipotesi induttiva si ha che $|y| \leq 2^{h-1}$ e $|z| \leq 2^{h-1}$, implicando che:

$$|x| = |y| + |z| \leq 2^{h-1} + 2^{h-1} = 2^h = 2^{(h+1)-1}$$

□

Lemma 2.4: Pumping lemma per i linguaggi acontestuali

Dato un linguaggio L, se $L \in \mathsf{CFL}$ allora $\exists p \in \mathbb{N}$, detto **lunghezza del pumping**, tale che $\forall w := uvxyz \in L$, con $|w| \geq p$ e $u, v, x, y, z \in \Sigma^*$ (ossia sono sue sottostringhe), si ha che:

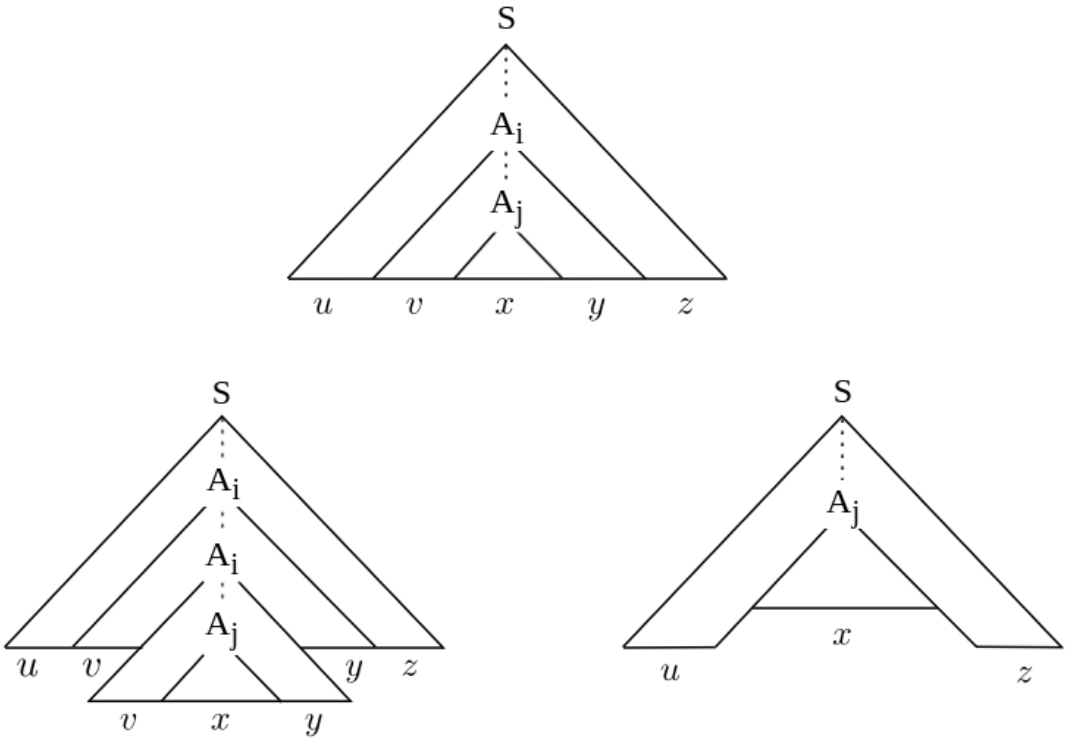
- $\forall i \in \mathbb{N} \;\; uv^i xy^i z \in L$
- $|vy| > 0$, dunque $v \neq \varepsilon$ o $y \neq \varepsilon$
- $|vxy| \leq p$

Dimostrazione.

- Dato $L \in \mathsf{CFL}$, sia $G = (V, \Sigma, R, S)$ la CFG in CNF tale che $L = L(G)$
- Sia $p = 2^{|V|}$. Data una stringa $w \in L$ tale che $|w| \geq p$, per la proposizione precedente l'albero di derivazione di w deve avere un'altezza $h \geq |V| + 1$, poiché altrimenti w non sarebbe generabile da esso
- Consideriamo quindi un cammino di lunghezza h di tale albero, dunque passante per almeno $k \geq |V| + 2$ nodi. Trattandosi di un cammino all'interno di un albero di derivazione, solo l'ultimo nodo del cammino corrisponderà ad un terminale, implicando che in tale cammino vi siano $k - 1 \geq |V| + 1$ variabili.
- Sia quindi $A_1, \ldots, A_{k-1}$ la sequenza di variabili del cammino (dove $S = A_1$). Poiché $k - 1 \geq |V| + 1 \geq |V|$, ne segue necessariamente che $\exists i, j \mid k - |V| - 2 \leq i < j \leq k - 1 \wedge A_i = A_j$, ossia che tra le ultime $|V| + 1$ variabili del cammino vi sia almeno una variabile ripetuta
- Consideriamo quindi le cinque sottostringhe $u, v, x, y, z \in \Sigma^*$ tali che:
  - $w = uvxyz$
  - $S \overset{*}{\Rightarrow} uA_i z$
  - $A_i \overset{*}{\Rightarrow} vA_j y$
  - $A_j \overset{*}{\Rightarrow} x$

- Poiché $A_i = A_j$, all'interno di ogni derivazione $A_i \overset{*}{\Rightarrow} vA_jy$ possiamo sostituire A_j con A_i stesso. Ripetendo tale procedimento $i \in \mathbb{N}$ volte ricorsivamente, otteniamo che:
$$A_i \overset{*}{\Rightarrow} vA_jy = vA_iy \overset{*}{\Rightarrow} v^iA_jy^i \Rightarrow v^ixy^i$$
implicando dunque che $\forall i \in \mathbb{N} \;\; S \overset{*}{\Rightarrow} uv^ixy^iz$ e quindi che $uv^ixy^iz \in L(G) = L$
- Poiché G è in CNF, dunque al suo interno non possono esserci ε-regole o regole unitarie, la derivazione $A_i \overset{*}{\Rightarrow} vA_jy$ deve necessariamente aver utilizzato una regola del tipo $A_i \to BC$ dove $B \overset{*}{\Rightarrow} vA_j$ e $C \overset{*}{\Rightarrow} y$ oppure $B \overset{*}{\Rightarrow} v$ e $C \overset{*}{\Rightarrow} A_jy$. Poiché non vi sono ε-regole, in entrambi i casi si ha che $v \neq \varepsilon$ o $y \neq \varepsilon$, implicando che $|vy| > 0$
- Poiché A_i si trova tra le ultime $|V| + 1$ variabili del cammino, ne segue che il suo sottoalbero abbia altezza $h' \leq |V| + 1$ (contando anche il terminale finale). Per la proposizione precedente, dunque, ne segue che:
$$|vxy| \leq 2^{h'-1} \leq 2^{|V|} = p$$

□



Rappresentazione grafica della dimostrazione

Esempio:

1.
   - Consideriamo il linguaggio $L = \{0^n 1^n 2^n \mid n \in \mathbb{N}\}$
   - Supponiamo per assurdo che $L \in \mathsf{CFL}$. In tal caso, ne segue che per esso debbia valere il pumping lemma, dove p è la lunghezza del pumping
   - Consideriamo quindi la stringa $w := 0^p 1^p 2^p$. Poiché $|w| \geq p$, possiamo suddividerla in cinque sottostringhe $u, v, x, y, z \in \Sigma^*$ tali che $w = uvxyz$.
   - Poiché la terza condizione del pumping lemma impone che $|vxy| \leq p$, le uniche possibilità sono:
     (a) Se $vxy = 0^m$ con $1 \leq m \leq p$, si ha che $u = 0^h$ e $z = 0^{p-m-h}1^p 2^p$, dove $1 \leq m + h \leq p$. Inoltre, poiché la seconda condizione impone che $|vy| > 0$, si ha che v e/o y contengono almeno uno 0
     (b) Se $vxy = 1^m$ con $1 \leq m \leq p$, si ha che $u = 0^p 1^h$ e $z = 1^{p-m-h}2^p$, dove $1 \leq m + h \leq p$. Inoltre, poiché la seconda condizione impone che $|vy| > 0$, si ha che v e/o y contengono almeno un 1
     (c) Se $vxy = 2^m$ con $1 \leq m \leq p$, si ha che $u = 0^p 1^p$ e $z = 2^{p-m-h}$, dove $leqm + h \leq p$. Inoltre, poiché la seconda condizione impone che $|vy| > 0$, si ha che v e/o y contengono almeno un 2
     (d) Se $vxy = 0^m 1^h$ con $1 \leq m + h \leq p$, si ha che $u = 0^{p-m}$ e $z = 1^{p-h}2^p$. Inoltre, poiché la seconda condizione impone che $|vy| > 0$, si ha che v contiene almeno uno 0 e/o y contiene almeno un 1
     (e) Se $vxy = 1^m 2^h$ con $1 \leq m + h \leq p$, si ha che $u = 0^p 1^{p-m}$ e $z = 2^{p-h}$. Inoltre, poiché la seconda condizione impone che $|vy| > 0$, si ha che v contiene almeno uno 1 e/o y contiene almeno un 2
   - In tutti i casi possibili descritti, risulta automatico che
     $$\nexists n \in \mathbb{N} \mid n = \left|uv^0xy^0z\right|_0 = \left|uv^0xy^0z\right|_1 = \left|uv^0xy^0z\right|_2 \implies uv^0xy^0z \notin L$$
     contraddicendo quindi la prima condizione del pumping lemma
   - Di conseguenza, ne segue necessariamente che $L \notin \mathsf{CFL}$

2.
   - Consideriamo il linguaggio $L = \{ww \mid w \in \{0,1\}^*\}$
   - Supponiamo per assurdo che $L \in \mathsf{CFL}$. In tal caso, ne segue che per esso debbia valere il pumping lemma, dove p è la lunghezza del pumping
   - Consideriamo quindi la stringa $w := 0^p 1^p 0^p 1^p$. Poiché $|w| \geq p$, possiamo suddividerla in cinque sottostringhe $u, v, x, y, z \in \Sigma^*$ tali che $w = uvxyz$.

- Poiché la terza condizione del pumping lemma impone che $|vxy| \leq p$, le uniche possibilità sono:
  (a) Se $u = 0^h$, $vxy = 0^m$ e $z = 0^{p-m-h}1^p0^p1^p$, dove $1 \leq m + h \leq p$, poiché la seconda condizione impone che $|vy| > 0$, si ha che v e/o y contengono almeno uno 0, dunque si ha che:

  $$\exists k < m \mid v^0xy^0 = 0^k \implies uv^0xy^0z = 0^h0^k0^{p-m-h}1^p0^p1^p = 0^{p-m+k}1^p0^p1^p$$

  dove $k < m \implies p - m - k < p$ e dunque che $uv^0xy^0z \notin L$
  (b) Se $u = 0^p1^p0^h$, $vxy = 0^m$ e $z = 0^{p-m-h}1^p$, dove $1 \leq m+h \leq p$, procedendo analogamente al caso (a) otteniamo che $uv^0xy^0z \notin L$
  (c) Se $u = 0^p1^h$, $vxy = 1^m$ e $z = 1^{p-m-h}0^p1^p$, dove $1 \leq m+h \leq p$, procedendo analogamente al caso (a) otteniamo che $uv^0xy^0z \notin L$
  (d) Se $u = 0^p1^p0^p1^h$, $vxy = 1^m$ e $z = 1^{p-m-h}$, dove $1 \leq m+h \leq p$, procedendo analogamente al caso (a) otteniamo che $uv^0xy^0z \notin L$
  (e) Se $u = 0^{p-h}$, $vxy = 0^h1^m$ e $z = 1^{p-m}0^p1^p$, dove $1 \leq m + h \leq p$, poiché la seconda condizione impone che $|vy| > 0$, si ha che v contiene almeno uno 0 e/o y contiene almeno un 1, dunque si ha che:

  $$\exists j < h, j < m \mid v^0xy^0 = 0^j1^k \implies$$

  $$uv^0xy^0z = 0^{p-h}0^j1^k1^{p-m}0^p1^p = 0^{p-h+j}1^{p-m+k}0^p1^p$$

  dove $j < h, k < m \implies p-h+j, p-m+k < p$ e dunque che $uv^0xy^0z \notin L$
  (f) Se $u = 0^p1^p0^{p-h}$, $vxy = 0^h1^m$ e $z = 1^{p-m}$, dove $1 \leq m+h \leq p$, procedendo analogamente al caso (e) otteniamo che $uv^0xy^0z \notin L$
  (g) Se $u = 0^p1^{p-h}$, $vxy = 1^h0^m$ e $z = 0^{p-m}1^p$, dove $1 \leq m + h \leq p$, poiché la seconda condizione impone che $|vy| > 0$, si ha che v contiene almeno uno 1 e/o y contiene almeno un 0, dunque si ha che:

  $$\exists j < h, j < m \mid v^0xy^0 = 1^j0^k \implies$$

  $$uv^0xy^0z = 0^p1^{p-h}1^j0^k0^{p-m}1^p = 0^p1^{p-h+j}0^{p-m+k}1^p$$

  dove $j < h, k < m \implies p-h+j, p-m+k < p$ e dunque che $uv^0xy^0z \notin L$
- Di conseguenza, poiché il pump down non può essere effettuato nè in un blocco di soli 0 o soli 1 (casi a, b, c, d), nè a cavallo tra degli 0 ed 1 (casi e, f), nè al centro della stringa (caso g), ne segue che la prima condizione del pumping lemma venga contraddetta
- Di conseguenza, ne segue necessariamente che $L \notin \mathsf{CFL}$

2.6 Chiusure dei linguaggi acontestuali

Teorema 2.4: Chiusura dell'unione in CFL

L'operatore unione è **chiuso in CFL**, ossia:

$$\forall L_1, \ldots, L_n \in \mathsf{CFL} \;\; L_1 \cup \ldots \cup L_n \in \mathsf{CFL}$$

Dimostrazione.

- Dati $L_1, \ldots, L_n \in \mathsf{CFL}$, siano $G_1, \ldots, G_n$ le grammatiche tali che $\forall i \in [1, n]$ $G_i = (V_i, \Sigma_i, R_i, S_i)$, dove $L_i = L(G_i)$ e $\forall i \neq j$ si ha che $V_i \cap V_j = \varepsilon$.
- Consideriamo quindi la CFG $G = (V, \Sigma, R, S)$ tale che:
  - S è una nuova variabile iniziale
  - $V = \left(\bigcup_{i=0}^{n} V_i \right) \cup \{S\}$
  - $\Sigma = \bigcup_{i=0}^{n} \Sigma_i$
  - $R = \left(\bigcup_{i=0}^{n} R_i \right) \cup \{S \to S_j \mid j \in [1, n]\}$
- Data $w \in \bigcup_{i=0}^{n} L(G_i)$, si ha che $\exists j \in [1, n] \mid w \in L(G_j)$

  Di conseguenza, poiché $(S \to S_j) \in R$, ne segue che

  $$w \in L(G_j) \iff S_j \overset{*}{\Rightarrow} w \implies S \Rightarrow S_j \overset{*}{\Rightarrow} w \implies w \in L(G)$$

- Data $w \in L(G)$, invece, dove $w \in L(G) \iff S \overset{*}{\Rightarrow} w$, poiché le uniche regole applicabili su S sono $\{S \to S_j \mid j \in [1, n]\}$, ne segue necessariamente che:

  $$w \in L(G) \implies \exists j \in [0, n] \mid S \Rightarrow S_j \overset{*}{\Rightarrow} w \implies w \in L(G_j) \subseteq \bigcup_{i=0}^{n} L(G_i)$$

- Di conseguenza, concludiamo che:

  $$L_1 \cup \ldots \cup L_n = L(G_1) \cup \ldots \cup L(G_n) = L(G) \in \mathsf{CFL}$$

□

Teorema 2.5: Chiusura della concatenazione in CFL

L'operatore concatenazione è **chiuso in CFL**, ossia:

$$\forall L_1, \ldots, L_n \in \mathsf{CFL} \;\; L_1 \circ \ldots \circ L_n \in \mathsf{CFL}$$

Dimostrazione.

- Dati $L_1, \ldots, L_n \in \mathsf{CFL}$, siano $G_1, \ldots, G_n$ le grammatiche tali che $\forall i \in [1, n]$ $G_i = (V_i, \Sigma_i, R_i, S_i)$, dove $L_i = L(G_i)$ e $\forall i \neq j$ si ha che $V_i \cap V_j = \varepsilon$.
- Consideriamo quindi la CFG $G = (V, \Sigma, R, S)$ tale che:
  - S è una nuova variabile iniziale
  - $V = \left(\bigcup_{i=0}^{n} V_i \right) \cup \{S\}$
  - $\Sigma = \bigcup_{i=0}^{n} \Sigma_i$
  - $R = \left(\bigcup_{i=0}^{n} R_i \right) \cup \{S \to S_1 \ldots S_n\}$
- Sia $w := w_1 \ldots w_n \in L(G_1) \circ \ldots \circ L(G_n)$, dove $\forall j \in [1, n] \;\; w_j \in L(G_j)$

  Poiché $(S \to S_1 \ldots S_n) \in R$, ne segue che

  $$\forall j \in [1, n] \;\; w_i \in L(G_j) \iff S_j \overset{*}{\Rightarrow} w_j$$

  dunque abbiamo che:

  $$S \Rightarrow S_1 \ldots S_n \overset{*}{\Rightarrow} w_1 \ldots w_n = w \implies w \in L(G)$$

- Data $w \in L(G)$, invece, dove $w \in L(G) \iff S \overset{*}{\Rightarrow} w$, poiché l'unica regola applicabile su S è $S \to S_1 \ldots S_n$, ne segue necessariamente che:

  $$w \in L(G) \implies S \Rightarrow S_1 \ldots S_n \overset{*}{\Rightarrow} w$$

  dunque $\exists w_1 \in L(G_1), \ldots, w_n \in L(G_n)$ tali che:

  $$S \Rightarrow S_1 \ldots S_n \overset{*}{\Rightarrow} w_1 S_2 \ldots S_n \overset{*}{\Rightarrow} w_1 w_2 \ldots w_n = w$$

  implicando che:

  $$w = w_1 w_2 \ldots w_n \in L(G_1) \circ \ldots \circ L(G_n)$$

- Di conseguenza, concludiamo che:

  $$L_1 \circ \ldots \circ L_n = L(G_1) \circ \ldots \circ L(G_n) = L(G) \in \mathsf{CFL}$$

□

Esempio:

- Consideriamo i seguenti linguaggi:

$$L_1 = \{0^n 1^n \mid n \in \mathbb{N}\} \qquad L_2 = \{1^m 0^m \mid m \in \mathbb{N}\}$$

- Consideriamo quindi le due grammatiche:

$$G_1 : A \to 0A1 \mid \varepsilon$$

$$G_2 : B \to 1A0 \mid \varepsilon$$

  tali che $L_1 = L(G_1)$ e $L_2 = L(G_2)$

- La grammatica G tale che $L(G) = L_1 \cup L_2$, corrisponderà a:

$$\begin{aligned} G : \quad & S \to A \mid B \\ & A \to 0A1 \mid \varepsilon \\ & B \to 0B1 \mid \varepsilon \end{aligned}$$

- La grammatica G' tale che $L(G') = L_1 \circ L_2$, corrisponderà a:

$$\begin{aligned} G : \quad & S \to AB \\ & A \to 0A1 \mid \varepsilon \\ & B \to 0B1 \mid \varepsilon \end{aligned}$$

Teorema 2.6: Chiusura di star in CFL

L'operatore star è **chiuso in CFL**, ossia:

$$\forall L \in \mathsf{CFL} \;\; L^* \in \mathsf{CFL}$$

Dimostrazione.

- Dato $L \in \mathsf{CFL}$, sia $G = (V, \Sigma, R, S)$ la CFG tale che $L = L(G)$.
- Consideriamo quindi la CFG $G' = (V, \Sigma, R', S_0)$ tale che:
  - S_0 è una nuova variabile iniziale
  - $R' = R \cup \{S_0 \to \varepsilon, S_0 \to S, S_0 \to S_0 S_0\}$
- Data $w := w_1 \dots w_n \in L^*$, abbiamo che:
  - Se $w = \varepsilon$, poiché $(S_0 \to \varepsilon) \in R$, ne segue che

$$S_0 \Rightarrow \varepsilon = w \implies w = \varepsilon \in L(G')$$

  - Se $w \neq \varepsilon$, invece, si ha che $\forall j \in [1,n]\ \ w_j \in L = L(G) \iff S \overset{*}{\Rightarrow} w_j$. Dunque si ha che:
    * Se $n = 1$, dunque $w = w_1$, tramite la regola $(S_0 \to S) \in R$ ne segue che:
    $$S_0 \Rightarrow S \overset{*}{\Rightarrow} w_1 = w \implies w \in L(G')$$
    * Se invece $n > 1$, tramite $(S_0 \Rightarrow S_0 S_0) \in R$ ne segue che:
    $$S_0 \Rightarrow S_0 S_0 \overset{*}{\Rightarrow} S_0^n \overset{*}{\Rightarrow} S^n \overset{*}{\Rightarrow} w_1 \dots w_n = w \implies w \in L(G')$$

- Data $w \in L(G')$, dove $w \in L(G') \iff S_0 \overset{*}{\Rightarrow} w$, poiché le uniche regole applicabili su S_0 sono $\{S_0 \to \varepsilon, S_0 \to S, S_0 \to SS\}$, ne segue necessariamente che:
  - Se $S_0 \Rightarrow \varepsilon = w$, ne segue direttamente che $w = \varepsilon \in L^0$
  - Se $S_0 \Rightarrow S \overset{*}{\Rightarrow} w$, ne segue direttamente che $w \in L(G) = L^1$
  - Se $S_0 \Rightarrow S_0 S_0 \overset{*}{\Rightarrow} w$, dato $n \geq 2$ si ha che:
  $$S_0 \Rightarrow S_0 S_0 \overset{*}{\Rightarrow} S_0^n \overset{*}{\Rightarrow} S^n$$
  Siano quindi $w_1, \dots, w_n \in L(G) = L$. Poiché $\forall j \in [1,n]\ \ w_j \in L(G) = L \iff S \overset{*}{\Rightarrow} w_j$, ne segue automaticamente che:
  $$S_0 \overset{*}{\Rightarrow} S^n \overset{*}{\Rightarrow} w_1 \dots w_n = w \implies w \in L^n$$

  Dunque, dato $n \geq 2$, abbiamo che:
  $$w \in L^0 \cup L^1 \cup L^n = L^*$$

- Di conseguenza, concludiamo che:
$$L^* = L(G') \in \mathsf{CFL}$$

□

Esempio:

- Consideriamo il seguente linguaggio e la sua grammatica generante:
$$L = \{0^n 1^n \mid n \in \mathbb{N}\} \qquad G : A \to 0A1 \mid \varepsilon$$

- La grammatica G' tale che $L(G) = L(G)^*$, corrisponderà a:
$$\begin{aligned} G' :\ & S \to \varepsilon \mid A \mid SS \\ & A \to 0A1 \mid \varepsilon \end{aligned}$$

Teorema 2.7: Non chiusura dell'intersezione in CFL

L'operatore intersezione **non è chiuso in CFL**, ossia:

$$\exists L_1, L_2 \in \mathsf{CFL} \mid L_1 \cap L_2 \notin \mathsf{CFL}$$

Dimostrazione.

- Consideriamo i seguenti due linguaggi:

$$L_1 = \{a^i b^i c^j \mid i, j \in \mathbb{N}\} \qquad L_2 = \{a^i b^j c^j \mid i, j \in \mathbb{N}\}$$

- Tali linguaggi sono descritti dalle seguenti due grammatiche:

$$G_1: \begin{array}{l} S \to TV \\ T \to aTb \mid \varepsilon \\ V \to cV \mid \varepsilon \end{array} \qquad G_2: \begin{array}{l} S \to VT \\ T \to bTc \mid \varepsilon \\ V \to aV \mid \varepsilon \end{array}$$

  dove $L_1 = L(G_1)$ e $L_2 = L_2(G_2)$

- L'intersezione di tali linguaggi risulta essere:

$$L_1 \cap L_2 = \{a^n b^n c^n \mid n \in \mathbb{N}\}$$

  il quale abbiamo già dimostrato non essere un linguaggio acontestuale (sezione 2.5)

- Di conseguenza, concludiamo che $L_1, L_2 \in \mathsf{CFL}$ ma $L_1 \cap L_2 \notin \mathsf{CFL}$

□

Teorema 2.8: Non chiusura del complemento in CFL

L'operatore complemento **non è chiuso in CFL**, ossia:

$$\exists L \in \mathsf{CFL} \mid \overline{L} \notin \mathsf{CFL}$$

Dimostrazione.

- Consideriamo il seguente linguaggio:

$$L = \{a, b\}^* - \{ww \mid w \in \{a, b\}^*\}$$

- Consideriamo quindi la seguente grammatica:

$$G: \begin{array}{l} S \to A \mid B \mid AB \mid BA \\ A \to a \mid aAa \mid aAb \mid bAa \mid bAb \\ B \to b \mid aBa \mid aBb \mid bBa \mid bBb \end{array}$$

- Data $x \in L$ tale che $|x|$ sia dispari, notiamo che:
  - Se il simbolo centrale di x è a, allora $S \Rightarrow A \overset{*}{\Rightarrow} x$
  - Se il simbolo centrale di x è b, allora $S \Rightarrow B \overset{*}{\Rightarrow} x$

  dunque ne segue che $x \in L(G)$
- Viceversa, data $x \in L(G)$ tale che $|x|$ sia dispari, ne segue immediatamente che $\nexists w \in \{a,b\}^* \mid x = ww \implies x \in L$
- Sia quindi $x \in L$ tale che $|x|$ sia pari.

  Dati $x_1, \ldots, x_n \in \{a,b\}$ tali che $x = x_1 \ldots x_n$, ne segue che:

$$x \in L \implies \exists i \in [1,n] \mid x_i \neq x_{\frac{n}{2}+i}$$

- Siano quindi $u := x_1 \ldots x_{2i-1}$ e $v := x_{2i} \ldots x_n$. Notiamo che il simbolo centrale di u corrisponde a $x_{\frac{1+2i-1}{2}} = x_i$, mentre quello di v corrisponde a $x_{\frac{2i+n}{2}} = x_{\frac{n}{2}+i}$, da cui traiamo che:

$$x_i \neq x_{\frac{n}{2}+i} \implies x_{\frac{1+2i-1}{2}} = x_i \neq x_{\frac{n}{2}+i} = x_{\frac{2i+n}{2}} \implies u \neq v$$

- Inoltre, notiamo che $|u|$ e $|v|$ siano dispari, dunque si ha che $u, v \in L(G)$. Di conseguenza, otteniamo che:

$$S \Rightarrow AB \overset{*}{\Rightarrow} uv = x \text{ oppure } S \Rightarrow BA \overset{*}{\Rightarrow} uv = x$$

  implicando quindi che $x \in L(G)$
- Sia quindi $x \in L(G)$ tale che $|x|$ sia pari.

  Poiché $|x|$ è pari, ne segue necessariamente che:

$$S \Rightarrow AB \overset{*}{\Rightarrow} x \text{ oppure } S \Rightarrow BA \overset{*}{\Rightarrow} x$$

  Poiché i due casi sono analoghi, senza perdita di generalità consideriamo il caso in cui $S \Rightarrow AB \overset{*}{\Rightarrow} x$
- Siano quindi $u := x_1 \ldots x_k$ e $v := x_{k+1} \ldots v_n$ tali che $x = uv$, $S \Rightarrow A \overset{*}{\Rightarrow} u$ e $S \Rightarrow B \overset{*}{\Rightarrow} v$.
- Poiché $S \Rightarrow A \overset{*}{\Rightarrow} u$ e $S \Rightarrow B \overset{*}{\Rightarrow} v$, otteniamo che:
  - $|u| = k$ e $|v| = n - k$ sono dispari
  - $S \Rightarrow A \overset{*}{\Rightarrow} u$ implica che il simbolo centrale di u sia a, ossia che $x_{\frac{1+k}{2}} = a$
  - $S \Rightarrow B \overset{*}{\Rightarrow} v$ implica che il simbolo centrale di v sia b, ossia che $x_{\frac{k+1+n}{2}} = b$

- Siano quindi che $w := w_1 \dots w_h$ e $w' := w'_1 \dots w'_h$ tali che $|w| = |w'| = h$ e che $u = ww'$, implicando dunque che $h = \frac{n}{2}$. Per il risultato precedente, ne segue automaticamente che:

$$w_{\frac{1+h}{2}} = x_{\frac{1+k}{2}} = a \neq b = x_{\frac{k+1+n}{2}} = w'_{\frac{1+h}{2}} \implies w \neq w' \implies x = ww' \in L$$

- Dunque, abbiamo ottenuto $L = L(G) \in \mathsf{CFL}$
- Il complemento di tale linguaggio risulta essere $\overline{L} = \{ww \mid w \in \{a, b\}^*\}$, il quale abbiamo già dimostrato non essere un linguaggio acontestuale (sezione 2.5). Di conseguenza, concludiamo che $L \in \mathsf{CFL}$, ma $\overline{L} \notin \mathsf{CFL}$

□

2.7 Esercizi svolti

Problema 2.1: Conversione da esp. reg a CFG

Si consideri l'espressione regolare $1\Sigma^*$, dove $\Sigma = \{0, 1\}$. Convertire tale espressione in una grammatica acontestuale e dimostrarne la correttezza

Dimostrazione I.

- Sia $R = 1\Sigma^* = 1(0 \cup 1)^*$
- Sia $G = (V, \Sigma, R, S)$ la CFG definita come:

$$\begin{aligned} G: \quad & S \to 1A \\ & A \to 0A \mid 1A \mid \varepsilon \end{aligned}$$

- Dalle regole di G, risulta evidente che se $A \overset{*}{\Rightarrow} w$ allora $w \in \Sigma^*$
- Procediamo quindi per induzione sulla lunghezza n di $w \in \Sigma^*$:

  Caso base $(n = 0)$:

  - Se $n = 0$, allora $w = \varepsilon$, implicando che $w \in \Sigma^*$ e che $A \Rightarrow w = \varepsilon$

  Ipotesi induttiva:

  - Per ogni stringa $w \in \Sigma^*$ tale che $|w| = n$, vale che:

$$w \in \Sigma^* \implies A \overset{*}{\Rightarrow} w$$

  Passo induttivo:

  - Data una stringa $w = a_1 \dots a_{n+1} \in \Sigma^*$, poiché $|a_2 \dots a_{n+1}| = n$, per ipotesi induttiva si ha che:

$$a_2 \dots a_{n+1} \in \Sigma^* \implies A \overset{*}{\Rightarrow} a_2 \dots a_{n+1}$$

  - A questo punto, notiamo che:
    * Se $w_1 = 0$, allora $A \Rightarrow 0A \overset{*}{\Rightarrow} 0a_2 \dots a_{n+1} = w$
    * Se $w_1 = 1$, allora $A \Rightarrow 1A \overset{*}{\Rightarrow} 1a_2 \dots a_{n+1} = w$

    dunque concludiamo che $A \overset{*}{\Rightarrow} w$

- Di conseguenza, otteniamo che $w \in \Sigma^* \iff A \overset{*}{\Rightarrow} w$
- Per le regole di G, otteniamo che:

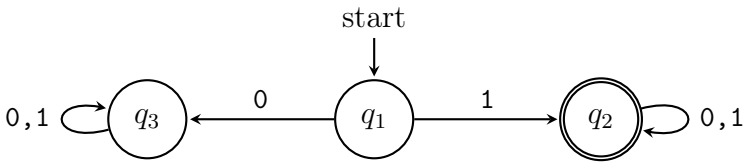
$$w \in L(G) \iff S \overset{*}{\Rightarrow} w \iff S \Rightarrow 1A \overset{*}{\Rightarrow} 1y = w \iff w \in L(R)$$

  dove $A \overset{*}{\Rightarrow} y$, implicando che $L(G) = L(R)$

□

Dimostrazione II.

- Sia $R = 1\Sigma^* = 1(0 \cup 1)^*$
- Sia $D = (Q, \Sigma, \delta, q_1, \{q_2\})$ il DFA definito come:



- Notiamo che:

$$w \in L(R) \implies \exists y \in \Sigma^* \mid w = 1y \implies \delta^*(q_0, 1y) = \delta^*(\delta(q_0, 1), y) =$$
$$\delta^*(q_2, y) = q_2 \implies w \in L(D)$$

  e inoltre che:

$$w \notin L(R) \implies \exists y \in \Sigma^* \mid w = 0y \implies \delta^*(q_0, 0y) = \delta^*(\delta(q_0, 0), y) =$$
$$\delta^*(q_3, y) = q_3 \implies w \notin L(D)$$

- Di conseguenza, si ha che:

$$w \in L(R) \iff w \in L(D)$$

  implicando che $L(R) = L(D)$

- A questo punto, tramite la Conversione da DFA a CFG, definiamo la seguente grammatica G tale che $L(G) = L(D)$:

$$\begin{aligned} G: \quad & V_1 \to 1V_2 \mid 0V_3 \\ & V_2 \to 1V_2 \mid 0V_2 \mid \varepsilon \\ & V_3 \to 1V_3 \mid 0V_3 \end{aligned}$$

□

Problema 2.2

Mostrare che la seguente grammatica è ambigua:

$$\begin{aligned} G: \quad & S \to TbT \\ & T \to aTbT \mid bTaT \mid \varepsilon \end{aligned}$$

Soluzione:

- Ricordiamo che per mostrare che una grammatica sia ambigua dobbiamo dimostrare che esistono due derivazioni sinistre per la stessa parola. Mostriamo quindi di poter derivare in due modi la parola *bab*.
- Primo modo: dopo aver applicato $S \to TbT$, applichiamo prima la regola $T \to bTaT$ sulla T più a sinistra per poi applicare la regola $T \to \varepsilon$ su tutte le T rimanenti:

$$S \Rightarrow TbT \Rightarrow bTaTbT \Rightarrow baTbT \Rightarrow babT \Rightarrow bab$$

- Secondo modo: dopo aver applicato $S \to TbT$, applichiamo prima la regola $T \to \varepsilon$ sulla T più a sinistra per poi applicare la regola $T \to \varepsilon$ sulla prima T rimanente a sinistra ed infine la regola $T \to \varepsilon$ su tutte le T rimanenti:

$$S \Rightarrow TbT \Rightarrow bT \Rightarrow baTbT \Rightarrow babT \Rightarrow bab$$

Problema 2.3

Data la seguente grammatica:

$$\begin{aligned} G: \quad & S \to WbT \\ & T \to aWbT \mid bVaT \mid \varepsilon \\ & W \to aWbW \mid \varepsilon \\ & V \to bVaV \mid \varepsilon \end{aligned}$$

Mostrare che $aabbbbaab \in L(G)$. Descrivere un PDA P tale che $L(G) = L(P)$

Soluzione:

- Analizzando la stringa *aabbbbaab* e la prima regola $S \to WbT$, notiamo che una delle *b* presenti nel centro della stringa debba necessariamente essere dovuta alla *b* introdotta da tale prima regola.
- A questo punto, notiamo che:

  $$W \Rightarrow aWbW \Rightarrow aaWbWbW \overset{*}{\Rightarrow} aabb$$

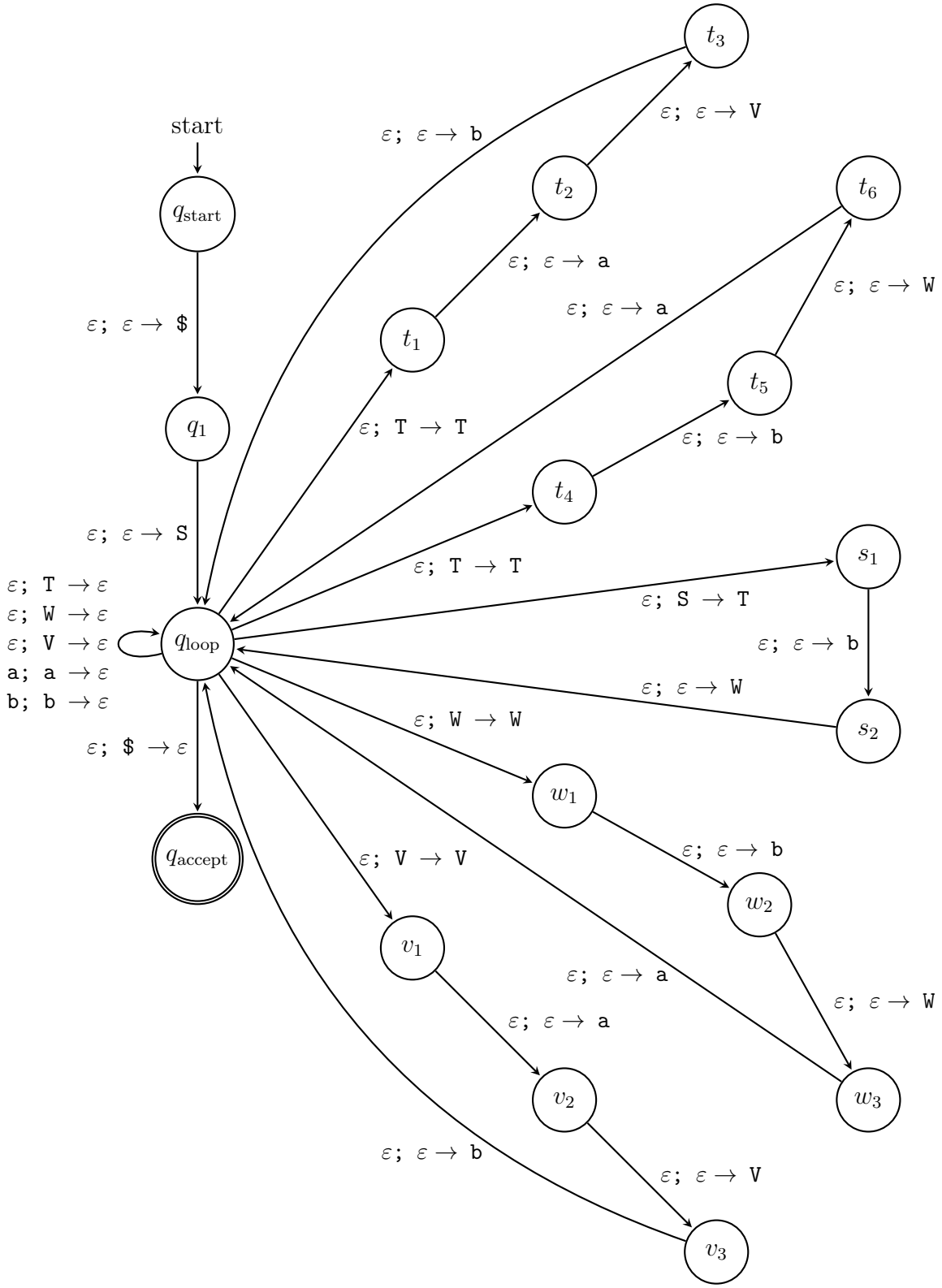
  e che:

  $$T \Rightarrow bVaT \Rightarrow baT \Rightarrow baaWbT \overset{*}{\Rightarrow} baab$$

  dunque otteniamo che:

  $$S \Rightarrow WbT \overset{*}{\Rightarrow} aabbbT \overset{*}{\Rightarrow} aabbbbaab$$

- Tramite la Conversione da CFG a PDA otteniamo che il PDA P tale che $L(G) = L(P)$ corrisponda a:



Problema 2.4

Classificare ognuno dei seguenti tre linguaggi come (a) regolare, (b) acontestuale ma non regolare, (c) non acontestuale:

1. $L_1 = \{w\#u \mid w, u \in \{0,1\}^* \text{ e } |w| = |u|\}$
2. $L_2 = \{w \in \{0,1\}^* \mid |w|_0 \text{ è dispari e } |w|_1 = 1\}$
3. $L_3 = \{w \in \{0,1,2\}^* \mid |w|_0 > |w|_1 \text{ e } |w|_0 > |w|_2\}$

Soluzione:

1.
   - Il linguaggio è facilmente riconoscibile da un PDA che memorizza nello stack ogni carattere della prima stringa, per poi rimuovere un carattere dallo stack ogni volta che ne viene letto un simbolo della seconda stringa. Questo conclude che $L_1 \in$ CFG.
   - Utilizziamo il pumping lemma per linguaggi regolari per dimostrare che $L_1 \notin$ REG. Consideriamo la stringa $0^p\#0^p \in L_1$, dove p è la lunghezza del pumping. Siano x, y, z le tre sottostringhe tali che $xyz = 0^p\#0^p$.
   - Per la terza condizione del lemma abbiamo che $| xy | \leq p$, dunque $x = 0^k$ per qualche $k \leq p$, mentre $yz = 0^{p-k}\#0^p$. Poiché per la seconda condizione del lemma abbiamo che $| y | > 0$, la sottostringa y deve o contenere solo simboli 0 oppure contenere anche il simbolo #.
   - In entrambi i casi, abbiamo che $xy^0z \notin L_1$, dunque L_1 non può essere regolare. Questo conclude che $L_1 \notin$ REG.
2.
   - Siano $L = \{w \in \{0,1\}^* \mid w \text{ contiene un numero dispari di } 0\}$ e $L' = \{w \in \{0,1\}^* \mid w \text{ contiene un singolo } 1\}$.
   - Notiamo che L e L' siano entrambi facilmente riconoscibili da un DFA, dunque $L, L' \in$ REG.
   - Dunque, poiché $L_2 = L \cap L'$ e i linguaggi regolari sono chiusi nell'intersezione, concludiamo che $L_2 \in$ REG.
3.
   - Utilizziamo il pumping lemma per linguaggi acontestuali per dimostrare che $L_3 \notin$ CFG. Consideriamo stringa $0^{p+1}1^p2^p \in L_3$, dove p è il primo numero dispari che sia maggiore della lunghezza del pumping q. Siano u, v, x, y, z le cinque sottostringhe tali che $uvxyz = 0^{p+1}1^p2^p$.
   - Abbiamo tre casi:
     (a) vxy contiene solo 0. In questo caso, poiché la seconda condizione del lemma impone che $| vy | > 0$, almeno una tra v e y dovrà contenere almeno uno 0, concludendo che il numero di 0 in uv^0xy^0z sia minore di p e dunque che $uv^0xy^0z \notin L_3$
     (b) vxy non contiene alcun 0. In questo caso, poiché la seconda condizione del lemma impone che $| vy | > 0$, almeno una tra v e y dovrà contenere almeno

un 1 o un 2, concludendo che il numero di 1 o 2 in uv^ixy^iz sia maggiore del numero di 0 (per qualche i sufficientemente grande), dunque $uv^ixy^iz \notin L_3$

(c) vxy contiene qualche 0 ma non solo. In questo caso, la terza condizione del lemma impone che $| \, vxy \, | \leq p$, dunque vxy non può contenere alcun 2. Inoltre, la seconda condizione impone che $| \, vy \, | > 0$. Per tanto, se vy contiene più 0 che 1 allora la stringa uv^0xy^0z conterrà più 1 che 0, mentre se vy contiene più 1 che 0 allora la stringa uv^ixy^iz per un i sufficientemente grade conterrà più 1 che 0 (ricordiamo che p è dispari per scelta, dunque non esiste in caso in cui abbiano lo stesso numero di 0 e 1). In entrambi i casi, concludiamo che la stringa ottenuta non appartenga a L_3.

- In tutti e tre i casi, otteniamo che L_3 non possa essere acontestuale. Questo conclude che $L_3 \notin$ CFG.

Problema 2.5

Definire una CFG per il seguente linguaggio:

$$L = \{0^n1^m2^p3^q \mid n, m, p, q \geq 0 \text{ con } n > m \text{ e } p < q\}$$

Soluzione:

- Consideriamo la seguente grammatica G:

$$\begin{aligned} G : S &\to AB \\ A &\to 0A1 \mid 0A \mid 0 \\ B &\to 2B3 \mid B3 \mid 3 \end{aligned}$$

- Osserviamo come la regola S divida la generazione della stringa in due parti: la creazione della sottostringa 0^n1^m con $n > m$ e la sottostringa 2^p3^q con $p < q$.
- In particolare, la regola $A \to 0A1$ garantisce che ogni qual volta venga generato un 1 venga anche generato uno 0 alla sua sinistra. Inoltre, la regola $A \to 0$ garantisce che la variabile A possa essere eliminata solo aggiungendo uno 0 senza un 1 associato, forzando $n > m$.
- Un'idea speculare viene fornita dalle regole della variabile B. Per tanto, abbiamo che $L(G) = L$.

3
Calcolabilità

3.1 Macchine di Turing

Nel 1936, il pioniere dell'informatica Alan Turing sviluppò un modello di calcolo simile ad un automa a stati finiti ma dotato di una memoria illimitata e senza alcuna restrizione. Sebbene essa richieda una grande mole di tempo, la **macchina di Turing** è in grado di elaborare tutto ciò che un reale computer è in grado di elaborare. Per tanto, essa costituisce un perfetto modello astratto di un reale computer, implicando che ogni problema per essa **irrisolvibile** lo sarà anche per un computer.

Il modello di Turing utilizza un **nastro infinito (solo a destra)** come memoria illimitata ed è dotata di una **testina di lettura-scrittura**. Il nastro è formato da celle, le quali, inizialmente, contengono solo una stringa data in input (tutte le altre celle sono vuote). Inoltre, il nastro viene continuamente **spostato** a sinistra e destra, in modo che la testina possa leggere o scrivere sulle varie celle. La macchina continua la sua computazione finché essa non raggiungerà lo stato di **accettazione** o lo stato di **rifiuto** della stringa in input. Se la macchina non è in grado di raggiungere nessuno dei due stati, essa rimarrà in un **loop infinito**, non terminando mai l'esecuzione.

Ad esempio, consideriamo il linguaggio $L = \{w\#w \mid w \in \{0,1\}^*\}$. Descriviamo in modo informale una macchina di Turing M in grado di accettare le stringhe di tale linguaggio:

M = "Data la stringa w in input:

1. Muoviti a zig-zag lungo il nastro tra tutte le posizioni corrispondenti su entrambi i lati del simbolo #. Se i due simboli combaciano, cancella entrambi sovrascrivendoli con una x. Se i due simboli non combaciano o se non viene mai trovato il simbolo #, rifiuta la stringa.
2. Quando tutti i simboli a sinistra del simbolo # sono stati cancellati, controlla se a destra del simbolo # vi sono simboli diversi da x. Se vi sono, rifiuta la stringa, altrimenti accettala."

Data la stringa in input `011000#011000`, l'esecuzione della macchina procede come:

$$
\begin{array}{cccccccccccccc}
 & \downarrow & & & & & & & & & & & & \\
0 & 1 & 1 & 0 & 0 & 0 & \# & 0 & 1 & 1 & 0 & 0 & 0 & \sqcup \ \ldots \\
 & & & & & & & & & & & & & \\
 & \downarrow & & & & & & & & & & & & \\
x & 1 & 1 & 0 & 0 & 0 & \# & 0 & 1 & 1 & 0 & 0 & 0 & \sqcup \ \ldots \\
 & & & & & & & & & & & & & \\
 & & & & & & & \downarrow & & & & & & \\
x & 1 & 1 & 0 & 0 & 0 & \# & x & 1 & 1 & 0 & 0 & 0 & \sqcup \ \ldots \\
 & & & & & & & & & & & & & \\
\downarrow & & & & & & & & & & & & & \\
x & 1 & 1 & 0 & 0 & 0 & \# & x & 1 & 1 & 0 & 0 & 0 & \sqcup \ \ldots \\
 & & & & & & & & & & & & & \\
 & \downarrow & & & & & & & & & & & & \\
x & x & 1 & 0 & 0 & 0 & \# & x & 1 & 1 & 0 & 0 & 0 & \sqcup \ \ldots \\
 & & & & & & & & & & & & & \\
 & & & & & & \ldots & & & & & & & \\
 & & & & & & & & & & & & \downarrow & \\
x & x & x & x & x & x & \# & x & x & x & x & x & x & \sqcup \ \ldots
\end{array}
$$

dove il simbolo $\sqcup$ indica una **cella vuota**

Definizione 3.1: Turing Machine (TM)

Una **Turing Machine (TM)** è una settupla $(Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ dove:

- Q è l'**insieme finito degli stati** della macchina
- Σ è l'**alfabeto della macchina**, dove $\sqcup \notin \Sigma$
- Γ è l'**alfabeto del nastro**, dove $\sqcup \in \Gamma$ e $\Sigma \subseteq \Gamma$
- $q_{\text{start}} \in Q$ è lo **stato iniziale** dell'automa
- $q_{\text{accept}} \in Q$ è lo **stato accettante** dell'automa
- $q_{\text{reject}} \in Q$ è lo **stato rifiutante** dell'automa, dove $q_{\text{reject}} \neq q_{\text{accept}}$
- $\delta : (Q - \{q_{\text{accept}}, q_{\text{reject}}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ è la **funzione di transizione** della macchina, dove se $\delta(p, a) = (q, b, X)$ si ha che:
  - Viene letto il simbolo a dal nastro, sostituendolo con b e la macchina passa dallo stato p allo stato q. Inoltre, il nastro viene spostato a sinistra se $X = L$ e a destra se $X = R$
  - L'etichetta della transizione da p a q viene indicata come $a \to b;\ X$
  - Se viene raggiunto lo stato q_{accept}, la TM accetta immediatamente

Definizione 3.2: Configurazione di una TM

Sia $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ una TM. Definiamo la stringa $uqav$ come **configurazione di** M, dove:

- $q \in Q$ è lo stato attuale della macchina
- $a \in \Gamma$ è il simbolo del nastro su cui si trova attualmente la testina della macchina
- $u \in \Gamma^*$ è composta dai simboli precedenti ad a sul nastro
- $v \in \Gamma^*$ è composta dai simboli successivi ad a sul nastro

Definizione 3.3: Passo di computazione in una TM

Data una TM $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$, si ha che:

$$uaq_ibv \text{ produce } uq_jacv \iff \delta(q_i, b) = (q_j, c, L)$$

$$uaq_ibv \text{ produce } uacq_jv \iff \delta(q_i, b) = (q_j, c, R)$$

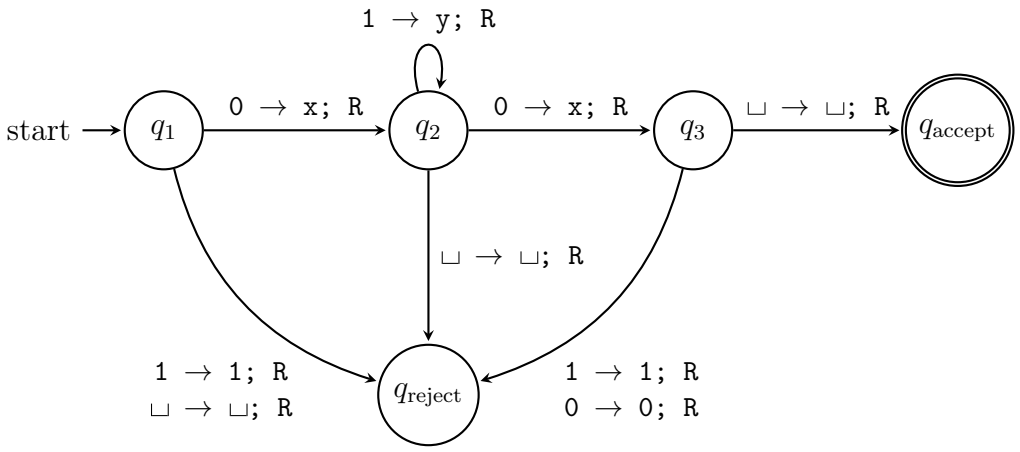
Proposizione 3.1: Stringa accettata in una TM

Sia $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ una TM. Data un stringa $w \in \Sigma^*$, diciamo che w è **accettata da** M se esiste una sequenza di configurazioni $c_1, \ldots, c_k$ tali che:

- $c_1 = q_{\text{start}}w$
- $\forall i \in [1, k-1]$ c_i produce c_{i+1}
- $q_{\text{accept}} \in c_k$

Esempio:

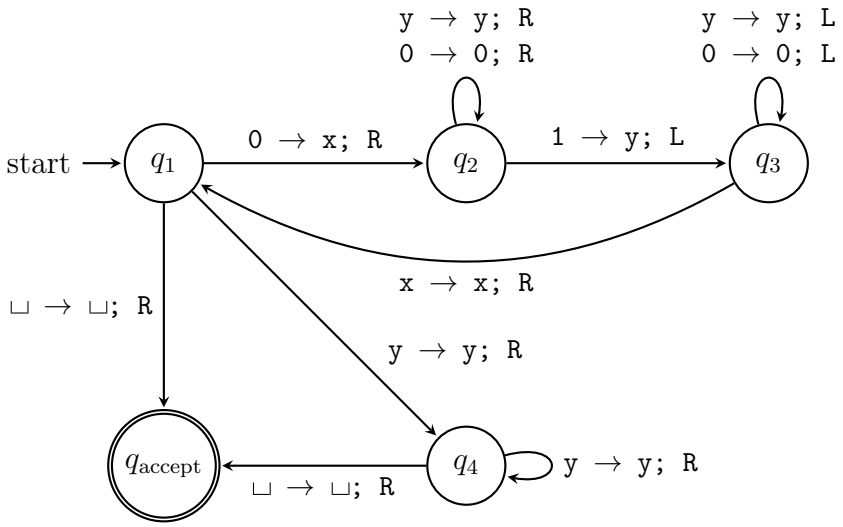
1. • La seguente TM riconosce il linguaggio $L = \{01^n0 \mid n \in \mathbb{N}\}$:



- Difatti, durante la lettura della stringa `01110`, la macchina assume le seguenti configurazioni:

$$
\begin{aligned}
&q_1\ 0\ 1\ 1\ 1\ 0\\
&x\ q_2\ 1\ 1\ 1\ 0\\
&x\ y\ q_2\ 1\ 1\ 0\\
&x\ y\ y\ q_2\ 1\ 0\\
&x\ y\ y\ y\ q_2\ 0\\
&x\ y\ y\ y\ x\ q_3\ \sqcup\\
&x\ y\ y\ y\ x\ \sqcup\ q_{\text{accept}}\ \sqcup
\end{aligned}
$$

2. - La seguente TM riconosce il linguaggio $L = \{0^n 1^n \mid n \in \mathbb{N}\}$:



(*tutte le transizioni omesse vanno allo stato* q_{reject})

- Difatti, durante la lettura della stringa `000111`, la macchina assume le seguenti configurazioni:

$$
\begin{aligned}
&q_1\ 0\ 0\ 0\ 1\ 1\ 1\\
&x\ q_2\ 0\ 0\ 1\ 1\ 1\\
&x\ 0\ q_2\ 0\ 1\ 1\ 1\\
&x\ 0\ 0\ q_2\ 1\ 1\ 1\\
&x\ 0\ q_3\ 0\ y\ 1\ 1\\
&x\ q_3\ 0\ 0\ y\ 1\ 1\\
&q_3\ x\ 0\ 0\ y\ 1\ 1
\end{aligned}
\qquad
\begin{aligned}
&x\ q_1\ x\ 0\ y\ 1\ 1\\
&x\ x\ q_2\ 0\ y\ 1\ 1\\
&x\ x\ 0\ q_2\ y\ 1\ 1\\
&x\ x\ 0\ y\ q_2\ 1\ 1\\
&x\ x\ 0\ q_3\ y\ y\ 1\\
&x\ x\ q_3\ 0\ y\ y\ 1\\
&x\ q_3\ x\ 0\ y\ y\ 1
\end{aligned}
\qquad
\begin{aligned}
&x\ x\ q_1\ 0\ y\ y\ 1\\
&x\ x\ x\ q_2\ y\ y\ 1\\
&x\ x\ x\ y\ q_2\ y\ 1\\
&x\ x\ x\ y\ y\ q_2\ 1\\
&x\ x\ x\ y\ q_3\ y\ y\\
&x\ x\ x\ q_3\ y\ y\ y\\
&x\ x\ q_3\ x\ y\ y\ y
\end{aligned}
\qquad
\begin{aligned}
&\\
&x\ x\ x\ q_3\ y\ y\ y\\
&x\ x\ x\ y\ q_4\ y\ y\\
&x\ x\ x\ y\ y\ q_4\ y\\
&x\ x\ x\ y\ y\ y\ q_4\ \sqcup\\
&x\ x\ x\ y\ y\ y\ \sqcup\ q_{\text{accept}}\ \sqcup
\end{aligned}
$$

Definizione 3.4: TM Decisore

Data una TM $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$, definiamo M come **decisore** se essa termina sempre la sua esecuzione (ossia non può entrare in un loop infinito).

Inoltre, se M è un decisore diciamo che M **decide** $L(M)$

Definizione 3.5: Classe dei linguaggi Turing-riconoscibili

Dato un alfabeto Σ, definiamo come **classe dei linguaggi Turing-riconoscibili di** Σ il seguente insieme:

$$\mathsf{REC} = \{L \subseteq \Sigma^* \mid \exists \text{ TM } M \text{ t.c } L = L(M)\}$$

Definizione 3.6: Classe dei linguaggi Turing-decidibili

Dato un alfabeto Σ, definiamo come **classe dei linguaggi Turing-decidibili di** Σ il seguente insieme:

$$\mathsf{DEC} = \{L \subseteq \Sigma^* \mid \exists \text{ decisore } M \text{ t.c } L = L(M)\}$$

Esempio:

- Entrambi i linguaggi dei due esempi precedenti sono Turing-decidibili in quanto nessuna delle due TM mostrate è in grado di entrare in un loop infinito

Osservazione 3.1: Descrizione informale delle TM

Negli esempi e dimostrazioni successive, le TM verranno descritte in modo informale, poiché la loro descrizione formale richiederebbe una grande quantità di stati e transizioni.

Ovviamente, tali descrizioni informali conterranno solo operazioni eseguibili dalle TM

Definizione 3.7: Codifica di un oggetto

Dato un oggetto O, indichiamo come $\langle O \rangle$ la sua **codifica**, ossia una stringa che ne descriva le caratteristiche

Esempi:

- Dato un polinomio $p = a_0 + a_1x_1 + \ldots + a_nx_n$, possiamo immaginare la sua codifica come una stringa composta dai suoi coefficienti, ossia $\langle p \rangle = \texttt{\#a}_1\texttt{,a}_2\texttt{,...,a}_n\texttt{\#}$
- Dato un grafo G, possiamo immaginare la sua codifica $\langle G \rangle$ come una stringa formata da una serie di coppie `(x,y)` rappresentanti gli archi del grafo

3.1.1 Equivalenze tra modelli di calcolo

Definizione 3.8: Stay-put TM

Una **Stay-put TM** è una TM $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ la cui **funzione di transizione** è definita come:

$$\delta : Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

dove il simbolo S indica che il nastro possa anche rimanere **immobile**

Teorema 3.1: Equivalenza tra TM e Stay-put TM

Dato un linguaggio $L \subseteq \Sigma^*$ si ha che:

$$L \in \mathsf{REC} \iff \exists \text{ Stay-put TM } M \text{ t.c } L = L(M)$$

In altre parole, le TM e le Stay-put TM sono equivalenti tra loro

Dimostrazione.

Prima implicazione.

- Dato $L \in \mathsf{REC}$, sia M la TM tale che $L = L(M)$
- Poiché una TM è una particolare Stay-put TM le cui transizioni con non rimangono mai immobili, ne segue automaticamente che essa stessa sia la Stay-put TM in grado di riconoscere $L = L(M)$

Seconda implicazione.

- Sia $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ la Stay-put TM tale che $L = L(M)$
- Consideriamo la TM $M' = (Q', \Sigma, \Gamma, \delta', q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ tale che:

  $$\delta(p, a) = (q, b, S) \iff \exists r \in Q \mid \forall c \in \Gamma \;\; \delta'(p, a) = (r, b, R) \wedge \delta'(r, c) = (q, c, L)$$

- Per costruzione stessa di M', si ha che:

  $$x \in L = L(M) \iff x \in L(M')$$

  implicando che $L = L(M) = L(M') \in \mathsf{REC}$

□

Definizione 3.9: TM multinastro

Una **TM multinastro a k nastri** è una TM $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ la cui **funzione di transizione** è definita come:

$$\delta : Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma^k \to Q \times \Gamma^k \times \{L, R, S\}$$

dove il simbolo S indica che il nastro possa anche rimanere **immobile**

Teorema 3.2: Equivalenza tra TM e TM multinastro

Dato un linguaggio $L \subseteq \Sigma^*$ si ha che:

$$L \in \mathsf{REC} \iff \exists \text{ Multitape TM } M \text{ t.c } L = L(M)$$

In altre parole, le TM e le TM multinastro sono equivalenti tra loro

Dimostrazione.

Prima implicazione.

- Dato $L \in \mathsf{REC}$, sia M la TM tale che $L = L(M)$
- Poiché una TM è una particolare TM multinastro ad 1 nastro le cui transizioni non rimangono mai immobili, ne segue automaticamente che essa stessa sia la TM multinastro in grado di riconoscere $L = L(M)$

Seconda implicazione.

- Sia M la TM multinastro a k nastri tale che $L = L(M)$
- Consideriamo la Stay-put TM S definita come:

  $S =$ "Date in input le stringhe $a_1 \dots a_n, b_1 \dots b_m, \dots, k_1 \dots k_h$ rappresentati gli input dei k nastri:

  1. S pone il nastro uguale a

     $$\# \dot{a}_1 \dots a_n \# \dot{b}_1 \dots b_m \# \dots \# \dot{k}_1 \dots k_h \#$$

     dove il simbolo # separa i vari k nastri simulati e il marcatore $\bullet$ indica le testine virtuali di ogni nastro

  2. Per simulare una mossa di M, S scansiona il nastro dal primo # fino al $(k+1)$-esimo #, ossia dall'estremità sinistra fino all'estremità destra, determinando i simboli puntati dalle testine virtuali. Successivamente, S esegue un secondo passaggio per aggiornare i nastri simulati in base alla funzione di transizione di M

3. Se in qualsiasi momento una delle testine virtuali finisce su un # durante uno spostamento a destra, S scrive un simbolo $\sqcup$ e sposta di una posizione a destra l'intero contenuto del nastro di S successivo al simbolo scritto, per poi riprendere la normale esecuzione"

- Per costruzione stessa di S, si ha che:

$$x \in L(M) \iff x \in L(S)$$

implicando che $L = L(M) = L(S)$

- Infine, per l'Equivalenza tra TM e Stay-put TM, ne segue automaticamente che $L = L(M) = L(S) \in \mathsf{REC}$

□

Definizione 3.10: Non deterministic TM

Una **Non deterministic TM (NTM)** è una TM $N = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ la cui **funzione di transizione** è definita come:

$$\delta : Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

Teorema 3.3: Equivalenza tra TM e NTM

Dato un linguaggio $L \subseteq \Sigma^*$ si ha che:

$$L \in \mathsf{REC} \iff \exists\, \mathsf{NTM}\ N \text{ t.c } L = L(N)$$

In altre parole, le TM e le NTM sono equivalenti tra loro

Dimostrazione.

Prima implicazione.

- Dato $L \in \mathsf{REC}$, sia M la TM tale che $L = L(M)$
- Poiché una TM è una particolare NTM le cui transizioni sono tutte deterministiche, ne segue automaticamente che essa stessa sia la NTM in grado di riconoscere $L = L(M)$

Seconda implicazione.

- Sia $N = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ la NTM tale che $L = L(N)$
- Consideriamo l'albero di computazione non deterministica di N. Ad ogni nodo di tale albero associamo un indirizzo:
  - Sia b il numero di transizioni uscenti dello stato di N avente il maggior numero di transizioni uscenti
  - Se il nodo è la radice dell'albero, il suo indirizzo è ε

  - Se il nodo non è la radice, il suo indirizzo è xa, dove x è l'indirizzo del padre di tale nodo ed $a \in \{1, \dots, b\}$ è l'identificatore associato a tale nodo tra i figli del suo padre

- Consideriamo quindi la seguente TM multinastro M a 3 nastri, dove:
  - Il nastro 1 contiene la stringa w in input ad N
  - Il nastro 2 è il nastro su cui viene simulata N con w in input
  - Il nastro 3 contiene l'indirizzo del nodo dell'albero di computazione fino a cui simulare N

- M è definita come:

  M = "Data la stringa w in input:

  1. Scrivi ε sul nastro 3
  2. Ripeti gli step successivi:
     3. Copia il nastro 1 sul nastro 2
     4. Simula N sul nastro 2 eseguendo un suo ramo di computazione in modo deterministico, dove ad ogni scelta la computazione viene eseguita in base al prossimo simbolo presente sul nastro 3
     5. Se la simulazione accetta la stringa, anche M *accetta*
     6. Se invece non rimangono più simboli sul nastro 3 o se la simulazione *rifiuta*, sostituisci la stringa sul nastro 3 con l'indirizzo del nodo direttamente a destra del nodo precedente all'interno dell'albero di computazione
     7. Se non vi è un nodo a destra, viene scelto il nodo più a sinistra del livello successivo. Se non rimangono più nodi, *rifiuta*"

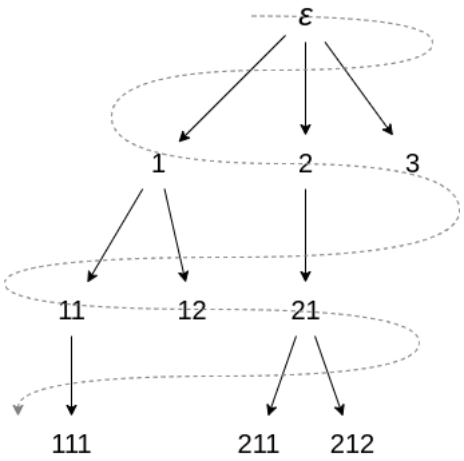
- Per costruzione stessa di M, si ha che:

$$x \in L(N) \iff x \in L(M)$$

  implicando che $L = L(N) = L(M)$

- Infine, per l'Equivalenza tra TM e TM multinastro, ne segue automaticamente che $L = L(N) = L(M) \in \mathsf{REC}$

□



Rappresentazione grafica della dimostrazione

Definizione 3.11: Enumeratore

Un **enumeratore** è una TM $E = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ connessa ad una "stampante" (ad esempio un nastro secondario), la quale stampa le stringhe di un linguaggio in ordine casuale e con eventuali ripetizioni.

Inoltre, il nastro di input dell'enumeratore è vuoto e diciamo che E **enumera** $L(E)$

Teorema 3.4: Equivalenza tra TM e Enumeratori

Dato un linguaggio $L \subseteq \Sigma^*$ si ha che:

$$L \in \mathsf{REC} \iff \exists \text{ enumeratore } E \text{ t.c } L = L(E)$$

In altre parole, le TM e gli enumeratori sono equivalenti tra loro

Dimostrazione.

Prima implicazione.

- Dato $L \in \mathsf{REC}$, sia M la TM tale che $L = L(M)$. Siano inoltre $w_1, w_2, \ldots \in \Sigma^*$ tutte le stringhe di Σ^*
- Consideriamo l'enumeratore E definito come:

  $E =$ "Dato nulla in input:

  1. Ripeti lo step seguente per $i = 1, 2, 3, \ldots$:

     2. Ripeti lo step seguente per $j = 1, \ldots, i$:

        3. Simula M per i passi con w_j in input. Se la simulazione accetta w_j, stampa w_j

- Per costruzione stessa di E, si ha che:

$$x \in L(M) \iff x \in L(E)$$

  implicando che $L = L(M) = L(E)$

Seconda implicazione.

- Sia E l'enumeratore tale che $L = L(E)$
- Consideriamo la TM M definita come:

  M = "Data la stringa w in input:

  1. Simula E. Ogni volta che E stampa una stringa, comparala con w.
  2. Se w appare almeno una volta nell'output di E, M *accetta*

- Per costruzione stessa di M, si ha che:

$$x \in L(E) \iff x \in L(M)$$

  implicando che $L = L(E) = L(M) \in$ REC

□

Proposizione 3.2: Tesi di Church-Turing

Data una funzione f, si ha che:

$$f \text{ computabile da un algoritmo} \iff f \text{ computabile da una TM}$$

In altre parole, le TM e gli algoritmi sono equivalenti tra loro, implicando che **qualsiasi tipo di computazione possa essere svolto tramite una TM**. Dunque, la tesi di Church-Turing può essere vista come una formalizzazione del concetto di algoritmo.

Definizione 3.12: TM universale

Una **TM universale** è una TM M in grado di simulare qualsiasi altra TM

Definizione 3.13: Turing-completezza

Definiamo un modello di calcolo come **Turing-completo** se esso è equivalente ad una TM universale

Esempi:

- Ogni computer moderno è un modello di calcolo Turing-completo
- Il *lambda calcolo non tipato* è un modello di calcolo Turing-completo
- Il gioco di carte *Magic: The Gathering* è un modello di calcolo Turing-completo (più info qui: https://arxiv.org/abs/1904.09828)

3.2 Problemi decidibili

Definizione 3.14: Problema dell'accettazione per DFA

Definiamo il linguaggio del **problema dell'accettazione per i DFA** come:

$$A_{\mathsf{DFA}} = \{\langle D, w\rangle \mid D \text{ DFA}, w \in L(D)\}$$

Teorema 3.5: A_{DFA} decidibile

Il linguaggio A_{DFA} è **decidibile**, ossia $A_{\mathsf{DFA}} \in \mathsf{DEC}$

Dimostrazione.

- Sia M la TM definita come:

  M = "Data in input la codifica $\langle D, w\rangle$, dove D è un DFA e w una stringa:

  1. Se la codifica in input è errata, M *rifiuta*
  2. M simula D con input w
  3. Se la simulazione termina su uno stato accettante di D, allora M *accetta*, altrimenti *rifiuta*."

- Per costruzione stessa di M, si ha che:

  $$\langle D, w\rangle \in L(M) \iff w \in L(D) \iff \langle D, w\rangle \in A_{\mathsf{DFA}}$$

  implicando che $L(M) = A_{\mathsf{DFA}}$

- Inoltre, poiché un DFA termina sempre, anche la simulazione terminerà sempre, implicando che M sia un decisore, concludendo che $A_{\mathsf{DFA}} = L(M) \in \mathsf{DEC}$.

□

Definizione 3.15: Problema dell'accettazione per NFA

Definiamo il linguaggio del **problema dell'accettazione per gli NFA** come:

$$A_{\mathsf{NFA}} = \{\langle N, w\rangle \mid N \text{ NFA}, w \in L(N)\}$$

Teorema 3.6: A_{NFA} decidibile

Il linguaggio A_{NFA} è **decidibile**, ossia $A_{\mathsf{NFA}} \in \mathsf{DEC}$

Dimostrazione.

- Sia M_{DFA} il decisore tale che $L(M_{\mathsf{DFA}}) = A_{\mathsf{DFA}}$ (teorema A_{DFA} decidibile)
- Sia M la TM definita come:

  M = "Data in input la codifica $\langle N, w \rangle$, dove N è un NFA e w una stringa:

  1. Se la codifica in input è errata, M *rifiuta*
  2. M converte N in un DFA D tale che $L(N) = L(D)$
  3. M esegue il programma di M_{DFA} con input $\langle D, w \rangle$
  4. Se l'esecuzione accetta, allora M *accetta*, altrimenti *rifiuta*"
- Per costruzione stessa di M, si ha che:

$$\langle N, w \rangle \in A_{\mathsf{NFA}} \iff \langle D, w \rangle \in A_{\mathsf{DFA}} = L(M_{\mathsf{DFA}}) \iff \langle N, w \rangle \in L(M)$$

  implicando che $L(M) = A_{\mathsf{NFA}}$
- Inoltre, poiché M_{DFA} è un decisore, dunque la sua esecuzione termina sempre, anche M terminerà sempre, concludendo che $A_{\mathsf{NFA}} = L(M) \in \mathsf{DEC}$.

□

Definizione 3.16: Problema dell'accettazione per esp. reg.

Definiamo il linguaggio del **problema dell'accettazione per le espressioni regolari** come:

$$A_{\mathsf{REX}} = \{\langle R, w \rangle \mid R \in \mathrm{re}(\Sigma), w \in L(R)\}$$

Teorema 3.7: A_{REX} decidibile

Il linguaggio A_{REX} è **decidibile**, ossia $A_{\mathsf{REX}} \in \mathsf{DEC}$

Dimostrazione.

- Sia M_{NFA} il decisore tale che $L(M_{\mathsf{NFA}}) = A_{\mathsf{NFA}}$ (teorema A_{NFA} decidibile)
- Sia M la TM definita come:

  M = "Data in input la codifica $\langle R, w \rangle$, dove $R \in \mathrm{re}(\Sigma)$ e w una stringa:

  1. Se la codifica in input è errata, M *rifiuta*
  2. M converte R in un NFA N tale che $L(R) = L(N)$

3. M esegue il programma di M_{NFA} con input $\langle N, w\rangle$
4. Se l'esecuzione accetta, allora M *accetta*, altrimenti *rifiuta*"

- Per costruzione stessa di M, si ha che:

$$\langle R, w\rangle \in A_{\mathsf{REX}} \iff \langle N, w\rangle \in A_{\mathsf{NFA}} = L(M_{\mathsf{NFA}}) \iff \langle R, w\rangle \in L(M)$$

  implicando che $L(M) = A_{\mathsf{REX}}$

- Inoltre, poiché M_{NFA} è un decisore, dunque la sua esecuzione termina sempre, anche M terminerà sempre, concludendo che $A_{\mathsf{REX}} = L(M) \in \mathsf{DEC}$.

□

Definizione 3.17: Problema del vuoto per DFA

Definiamo il linguaggio del **problema del vuoto per i DFA** come:

$$E_{\mathsf{DFA}} = \{\langle D\rangle \mid D \text{ DFA}, L(D) = \varnothing\}$$

Teorema 3.8: E_{DFA} decidibile

Il linguaggio E_{DFA} è **decidibile**, ossia $E_{\mathsf{DFA}} \in \mathsf{DEC}$

Dimostrazione.

- Sia M la TM definita come:

  M = "Data in input la codifica $\langle D\rangle$, dove $D = (Q, \Sigma, \delta, q_0, F)$ è un DFA:

  1. Se la codifica in input è errata, M *rifiuta*
  2. Marca lo stato iniziale di D
  3. Ripeti lo step seguente finché vengono marcati dei nuovi stati
     4. Marca ogni stato avente una transizione entrante da uno stato già marcato
  5. Se tra gli stati marcati vi è uno stato accettante di D, allora M *rifiuta*, altrimenti *accetta*"

- A questo punto, notiamo che:

$$\langle D\rangle \in E_{\mathsf{DFA}} \iff L(D) = \varnothing \iff \nexists w \in L(D) \iff$$

$$\forall w \in \Sigma^* \ \ \delta^*(q_0, w) \notin F \iff \langle D\rangle \in L(M)$$

  implicando che $L(M) = E_{\mathsf{DFA}}$

- Inoltre, poiché il numero di stati marcabili da M è finito, ne segue che M termini sempre, concludendo che $E_{\mathsf{DFA}} = L(M) \in \mathsf{DEC}$

□

Definizione 3.18: Problema dell'equivalenza tra DFA

Definiamo il linguaggio del **problema dell'equivalenza tra due DFA** come:

$$EQ_{\mathsf{DFA}} = \{\langle A, B\rangle \mid A, B \text{ DFA}, L(A) = L(B)\}$$

Teorema 3.9: EQ_{DFA} decidibile

Il linguaggio EQ_{DFA} è **decidibile**, ossia $EQ_{\mathsf{DFA}} \in \mathsf{DEC}$

Dimostrazione.

- Consideriamo la *differenza simmetrica* tra $L(A)$ e $L(B)$, definita come:

  $$L(A) \Delta L(B) := (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)})$$

  ossia tutti gli elementi presenti in $L(A)$ o $L(B)$, ma non in $L(A) \cap L(B)$

- Poiché le operazioni di unione, intersezione e complemento sono chiuse in REG (Teoremi 1.3, 1.4 e 1.5), ne segue automaticamente che:

  $$L(A), L(B) \in \mathsf{REG} \implies L(A) \Delta L(B) \in \mathsf{REG}$$

  dunque $\exists C$ DFA $\mid L(C) = L(A) \Delta L(B)$

- Inoltre, mostriamo che:

  $$L(A) \Delta L(B) = \varnothing \iff$$
  $$(L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) = \varnothing \iff$$
  $$\nexists x \in \Sigma^* \mid (x \in L(A) \land x \notin L(B)) \lor (x \in L(B) \land x \notin L(A)) \iff$$
  $$\forall x \in \Sigma^* \ (x \in L(A) \iff x \in L(B)) \iff$$
  $$L(A) = L(B)$$

- Sia M_E il decisore tale che $L(M_E) = E_{\mathsf{DFA}}$ (teorema E_{DFA} decidibile)
- Sia M la TM definita come:

  M = "Data in input la codifica $\langle A, B\rangle$, dove A e B sono due DFA:

  1. Se la codifica in input è errata, M rifiutante
  2. M costruisce il DFA C tale che $L(C) = L(A) \Delta L(B)$ tramite le procedure dei teoremi 1.2, 1.3, 1.4 e 1.5
  3. M esegue il programma di M_E con input $\langle C\rangle$
  4. Se l'esecuzione accetta, M *accetta*, altrimenti *rifiuta*."

- A questo punto, notiamo che:

$$\langle A, B\rangle \in EQ_{\mathsf{DFA}} \iff L(A) = L(B) \iff L(C) = L(A)\,\Delta\,L(B) = \varnothing \iff$$

$$\langle C\rangle \in L(M_E) \iff \langle A, B\rangle \in L(M)$$

  implicando che $L(M) = EQ_{\mathsf{DFA}}$

□

Definizione 3.19: Problema dell'accettazione per CFG

Definiamo il linguaggio del **problema dell'accettazione per le grammatiche acontestuali** come:

$$A_{\mathsf{CFG}} = \{\langle G, w\rangle \mid G\ \mathsf{CFG}, w \in L(G)\}$$

Teorema 3.10: A_{CFG} decidibile

Il linguaggio A_{CFG} è **decidibile**, ossia $A_{\mathsf{CFG}} \in \mathsf{DEC}$

Dimostrazione.

- **Affermazione**: Sia $G = (V, \Sigma, R, S)$ una CFG in CNF. Data $w \in L(G)$, se $|w| \geq 1$, la sua derivazione è composta da esattamente $2 \cdot |w| - 1$ produzioni

  Dimostrazione.

  Procediamo per induzione sulla lunghezza n di w

  Caso base.

  – Per $n = 1$, si ha che $w = a$, dove $a \in \Sigma$. Di conseguenza la sua derivazione è composta solo dalla regola $S \Rightarrow a = w$, ossia da $2 \cdot 1 - 1 = 1$ produzioni

  Ipotesi induttiva forte.

  – Assumiamo che per ogni stringa $w \in L(G)$ tale che $1 \leq |w| \leq n$ sia derivabile tramite tramite $2\,|w| - 1$ produzioni

  Passo induttivo.

  – Sia $w \in L(G)$ tale che $|w| = n + 1$. Essendo G in CNF, ne segue che la derivazione di w sia nella forma $S \Rightarrow AB \overset{*}{\Rightarrow} w$.

  – Siano quindi $x, y \in \Sigma^*$ tali che $w = xy$, dove $A \overset{*}{\Rightarrow} x$ e $B \overset{*}{\Rightarrow} y$.

  – Poiché G è in CNF, ne segue che $x, y \neq \varepsilon$, implicando che $1 \leq |x| \leq n$ e $1 \leq |y| \leq n$

  – Siano quindi $|x| = k$ e $|y| = n + 1 - k$. Per ipotesi induttiva, x e y sono derivabili tramite esattamente $2k - 1$ produzioni e $2(n + 1 - k) - 1$ produzioni

  - Di conseguenza, poiché $S \Rightarrow AB \overset{*}{\Rightarrow} xy = w$, ne segue che il numero di produzioni della derivazione di w sia esattamente:

$$1 + 2k - 1 + 2(n + 1 - k) - 1 = 2n + 2 - 1 = 2(n + 1) - 1 = 2\,|w| - 1$$

- Sia M la TM definita come:

  M = "Data in input la codifica $\langle G, w\rangle$, dove G è un CFG e w una stringa:

  1. Se la codifica in input è errata, M *rifiuta*
  2. M converte G in una CFG G' in CNF tale che $L(G) = L(G')$
  3. Se $|w| \neq 0$, M lista tutte le derivazioni di G composte da $2n - 1$ produzioni, dove $|w| = n$. Altrimenti, M lista tutte le derivazioni composte da 1 produzione
  4. Se almeno una delle derivazioni genera w, M *accetta*, altrimenti *rifiuta*"

- Per costruzione stessa di M, si ha che:

$$\langle G, w\rangle \in L(M) \iff w \in L(G) \iff \langle G, w\rangle \in A_{\mathsf{CFG}}$$

  implicando che $L(M) = A_{\mathsf{CFG}}$

- Inoltre, poiché la lista utilizzata da M sarà sempre composta da un numero finito di derivazioni, ne segue che M terminerà sempre, concludendo che $A_{\mathsf{CFG}} = L(M) \in$ DEC.

□

Definizione 3.20: Problema del vuoto per CFG

Definiamo il linguaggio del **problema del vuoto per le grammatiche acontestuali** come:

$$E_{\mathsf{CFG}} = \{\langle G\rangle \mid G \text{ CFG}, L(G) = \varnothing\}$$

Teorema 3.11: E_{CFG} decidibile

Il linguaggio E_{CFG} è **decidibile**, ossia $E_{\mathsf{CFG}} \in$ DEC

Dimostrazione.

- Sia M la TM definita come:

  M = "Data in input la codifica $\langle G\rangle$, dove $G = (V, \Sigma, R, S)$ è una CFG:

  1. Se la codifica in input è errata, M *rifiuta*
  2. Marca tutti i terminali in Σ
  3. Ripeti lo step seguente finché vengono marcate delle nuove variabili

4. Marca ogni variabile $A \in V$ per cui in R esiste una regola $A \to u_1 \dots u_k$ tale che $u_1, \dots, u_k$ sono variabili o terminali già marcati
5. Se la variabile S è marcata, M *rifiuta*, altrimenti *accetta*."

- A questo punto, notiamo che:

$$\langle G \rangle \in E_{\mathsf{CFG}} \iff L(G) = \varnothing \iff \nexists w \in L(G) \iff$$

$$\forall w \in \Sigma^* \ \ S \overset{*}{\not\Rightarrow} w \iff \langle G \rangle \in L(M)$$

  implicando che $L(M) = E_{\mathsf{CFG}}$

- Inoltre, poiché il numero di variabili marcabili da M è finito, ne segue che M termini sempre, concludendo che $E_{\mathsf{CFG}} = L(M) \in \mathsf{DEC}$

□

Corollario 3.1: Ling. decidibili estensione dei ling. acontestuali

Date le classi dei linguaggi CFL e DEC, si ha che:

$$\mathsf{CFL} \subsetneq \mathsf{DEC}$$

Dimostrazione.

- Sia M_{CFG} il decisore tale che $L(M_{\mathsf{CFG}}) = A_{\mathsf{CFG}}$ (teorema A_{CFG} decidibile)
- Dato $L \in \mathsf{CFL}$, sia G la CFG tale che $L = L(G)$
- Consideriamo quindi la TM M definita come:

  M = "Data la stringa w in input:

  1. M esegue il programma di M_{CFG} con input $\langle G, w \rangle$
  2. Se l'esecuzione accetta, M *accetta*, altrimenti *rifiuta*"

- Per costruzione stessa di M, si ha che:

$$w \in L(M) \iff \langle G, w \rangle \in A_{\mathsf{CFG}} \iff w \in L(G)$$

  implicando che $L(M) = L(G)$. Inoltre, poiché A_{CFG} è un decisore, anche M è un decisore, implicando che $\mathsf{CFG} \subseteq \mathsf{DEC}$

- Consideriamo quindi il linguaggio $L = \{ww \mid w \in \{a, b\}^*\}$. Per dimostrazione precedente (sezione 2.5), sappiamo che $L \notin \mathsf{CFL}$. Tuttavia, possiamo facilmente definire un decisore M (simile a quella vista nella sezione 3.1) per cui $L = L(M) \in \mathsf{DEC}$

- Di conseguenza, concludiamo che:

$$\mathsf{CFL} \subsetneq \mathsf{DEC}$$

□

3.3 Argomento diagonale di Cantor

Teorema 3.12: Insiemi con stessa cardinalità

Dati due insiemi A e B si ha che:

$$\exists f : A \to B \text{ biettiva} \implies |A| = |B|$$

(*dimostrazione omessa*)

Definizione 3.21: Insiemi infiniti numerabili

Un insieme A viene detto **numerabile** se $|A| < +\infty$ o se $|A| = |\mathbb{N}|$

Esempio:

- Dato l'insieme $2\mathbb{N} = \{2n \mid n \in \mathbb{N}\}$, consideriamo la seguente funzione:

  $$f : \mathbb{N} \to 2\mathbb{N} : n \mapsto 2n$$

- Tale funzione risulta essere sia iniettiva:

  $$f(n) = f(m) \implies 2n = 2m \implies n = m$$

  sia suriettiva:

  $$\forall 2n \in 2\mathbb{N} \;\; \exists n \in \mathbb{N} \mid f(n) = 2n$$

- Di conseguenza, poiché f è biettiva, concludiamo che $|\mathbb{N}| = |2\mathbb{N}|$ nonostante $2\mathbb{N} \subsetneq \mathbb{N}$

Metodo 3.1: Argomento diagonale di Cantor

L'**argomento diagonale di Cantor** è una tecnica dimostrativa atta a dimostrare l'**esistenza o inesistenza** di una funzione biettiva tra due insiemi A e B disponendo i loro elementi in forma tabellare, per poi concludere la tesi.

Teorema 3.13: Razionali positivi numerabili

L'insieme $\mathbb{Q}_{\geq 0}$ dei numeri razionali non negativi è **numerabile**

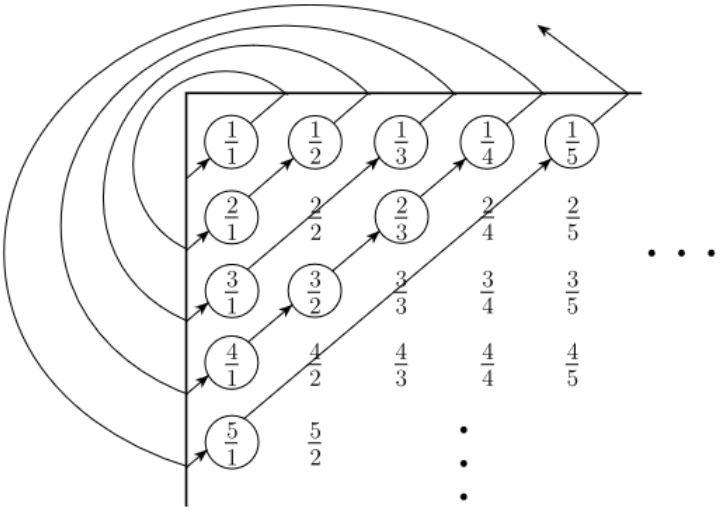
Dimostrazione.

- Siano $\mathbb{N}_{>0}$ e $\mathbb{Q}_{>0}$ gli insiemi dei numeri naturali e razionali positivi
- Consideriamo la matrice A avente righe e colonne infinite le cui entrate sono definite come:

  $$a_{i,j} = \frac{i}{j}$$

  dove $i, j \in \mathbb{N}$

- Costruiamo una lista di elementi di tale matrice procedendo diagonale per diagonale, partendo dalla diagonale composta dall'entrata $a_{1,1}$ e saltando tutti gli elementi che sono già stati inseriti nella lista (ad esempio, poiché $a_{1,1} = \frac{1}{1} = \frac{2}{2} = a_{2,2}$, l'entrata $a_{2,2}$ non verrà inserita nella lista):



Rappresentazione grafica del processo di creazione della lista

- Procedendo all'infinito, otterremo la lista $\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \frac{1}{3}, \ldots$ contenente tutti gli elementi di $Q_{>0}$, senza alcuna ripetizione. Inoltre, aggiungiamo all'inizio di tale lista il numero 0.
- A questo punto, consideriamo la funzione $f : \mathbb{N} \to \mathbb{Q}_{\geq 0}$ definita come:

$$f(n) = n\text{-esimo elemento della lista}$$

n	0	1	2	3	4	5	...
$f(n)$	0	$\frac{1}{1}$	$\frac{2}{1}$	$\frac{1}{2}$	$\frac{3}{1}$	$\frac{1}{3}$	...

- Poiché la lista contiene tutti gli elementi di $Q_{>0}$ senza alcuna ripetizione, ogni n-esimo elemento della lista sarà mappato esclusivamente dal numero $n \in \mathbb{N}$.
- Di conseguenza, otteniamo che f sia biettiva, concludendo che $|\mathbb{N}| = |\mathbb{Q}_{\geq 0}|$ e quindi che $\mathbb{Q}_{\geq 0}$ sia numerabile

□

Teorema 3.14: Reali non numerabili

L'insieme $\mathbb{R}$ dei numeri reali **non è numerabile**

Dimostrazione.

- Dato $[0,1] \subseteq \mathbb{R}$, supponiamo per assurdo che $\exists f : \mathbb{N} \to [0,1]$ biettiva
- Consideriamo il numero x definito come:

$$\forall i \geq 1 \quad i\text{-esima cifra decimale di } x \neq i\text{-esima cifra decimale di } f(i)$$

- Per definizione stessa di x, ne segue che $\nexists n \in \mathbb{N} \mid f(n) = x$, implicando che f non sia suriettiva, contraddicendo l'ipotesi per cui essa sia biettiva
- Di conseguenza, ne segue necessariamente che $\nexists f : \mathbb{N} \to [0,1]$ biettiva, implicando che $|\mathbb{N}| < |[0,1]| \leq |\mathbb{R}|$ e dunque che $\mathbb{R}$ non sia numerabile

□

n	$f(n)$
0	$0.\underline{1}18285101\ldots$
1	$0.2\underline{1}3812941\ldots$
2	$0.12\underline{3}124112\ldots$
3	$0.945\underline{8}53164\ldots$
4	$0.3924\underline{8}1412\ldots$
⋮	⋮

$\implies x = 0.67392\ldots$

Rappresentazione grafica della dimostrazione

Teorema 3.15: Sequenze binarie infinite non numerabili

L'insieme $\mathcal{B}$ di tutte le stringhe binarie infinite **non è numerabile**

Dimostrazione.

- Supponiamo per assurdo che $\exists f : \mathbb{N} \to \mathcal{B}$ biettiva
- Consideriamo la sequenza binaria x definita come:

$$\forall i \geq 1 \quad i\text{-esima cifra di } x \neq i\text{-esima cifra di } f(i)$$

- Per definizione stessa di x, ne segue che $\nexists n \in \mathbb{N} \mid f(n) = x$, implicando che f non sia suriettiva, contraddicendo l'ipotesi per cui essa sia biettiva
- Di conseguenza, ne segue necessariamente che $\nexists f : \mathbb{N} \to \mathbb{B}$ biettiva, implicando che $|\mathbb{N}| < |\mathcal{B}|$ e dunque che $\mathcal{B}$ non sia numerabile

□

n	$f(n)$
0	$\underline{1}0101010101\ldots$
1	$1\underline{1}101101011\ldots$
2	$01\underline{0}10100001\ldots$
3	$000\underline{0}1010100\ldots$
4	$1111\underline{1}111111\ldots$
$\vdots$	$\vdots$

$\implies x = 00110\ldots$

Rappresentazione grafica della dimostrazione

3.3.1 Esistenza di linguaggi non riconoscibili

Teorema 3.16: Esistenza di linguaggi non riconoscibili

Dato un alfabeto Σ, si ha che:

$$\exists L \subseteq \Sigma^* \mid L \notin \mathsf{REC}$$

Dimostrazione.

- Sia $<_\ell$ la relazione definita su Σ^* tale che:

  $$\forall x, y \in \Sigma^* \quad x <_\ell y \iff x \text{ precede } y \text{ lessico-graficamente}$$

- Sia inoltre $\prec$ la relazione definita su Σ^* tale che:

  $$\forall x, y \in \Sigma^* \quad x \prec y \iff (|x| < |y|) \vee (|x| = |y| \wedge x <_\ell y)$$

  ossia che ordina le stringhe di Σ^* in base alla loro lunghezza e, a parità di lunghezza, in base al loro ordine lessico-grafico

  Dalla definizione stessa di $\prec$, risulta evidente che tale relazione sia un ordine totale.

- Sia quindi $f : \mathbb{N} \to \Sigma^*$ la funzione definita come:

  $$f(i) = i\text{-esima stringa di } \Sigma^* \text{ secondo } \prec$$

  Tale funzione risulta intuitivamente essere biettiva, implicando che $|\mathbb{N}| = |\Sigma^*|$, dunque che Σ^* sia numerabile

- Consideriamo quindi il linguaggio $\mathcal{M} \subseteq \Sigma^*$ definito come:

  $$\mathcal{M} = \{\langle M \rangle \mid M \text{ è una } \mathsf{TM}\}$$

  Poiché $\mathcal{M} \subseteq \Sigma^*$ e Σ^* è numerabile, ne segue automaticamente che anche $\mathcal{M}$ sia numerabile

- Consideriamo inoltre l'insieme $\mathcal{L} = \mathcal{P}(\Sigma^*)$, corrispondente alla classe di tutti i linguaggi definiti su Σ
- Dato un linguaggio $L \in \mathcal{L}$, definiamo la sequenza binaria $\chi_L = b_1 b_2 \ldots$, detta *sequenza caratteristica di* L, come:

$$b_i = \begin{cases} 1 & \text{se } s_i \in L \\ 0 & \text{se } s_i \notin L \end{cases}$$

  dove $s_1, s_2, \ldots$ sono tutte le stringhe di Σ^* ordinate secondo $\prec$
- Consideriamo quindi la seguente funzione:

$$g : \mathcal{L} \to \mathcal{B} : L \mapsto \chi_L definita$$

  Tale funzione risulta intuitivamente essere biettiva, implicando che $|\mathcal{L}| = |\mathcal{B}|$. Di conseguenza, poiché $\mathcal{B}$ non è numerabile, ne segue che anche $\mathcal{L}$ non sia numerabile
- A questo punto, poiché $\mathcal{M}$ è numerabile e $\mathcal{L}$ no, concludiamo che la seguente funzione:

$$h : \mathcal{M} \to \mathcal{L} : M \mapsto L(M)$$

  non sia biettiva, implicando che $\exists L \in \mathcal{L} \mid \nexists M \in \mathcal{M}$ t.c $L = L(M)$

□

3.4 Problemi indecidibili

Definizione 3.22: Problema dell'accettazione per TM

Definiamo il linguaggio del **problema dell'accettazione per le TM** come:

$$A_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ TM}, w \in L(M)\}$$

Teorema 3.17: A_{TM} riconoscibile ma non decidibile

Il linguaggio A_{TM} è **riconoscibile** ma **non decidibile**, ossia $A_{\mathsf{TM}} \in \mathsf{REC} - \mathsf{DEC}$

Dimostrazione riconoscibilità.

- Sia U una TM universale a 2 nastri definita come:

  U = "Data in input la codifica $\langle M, w\rangle$, dove $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ è una TM e w una stringa:

  1. Se la codifica in input è errata, U *rifiuta*
  2. M scrive $\langle M, w\rangle$ sul nastro 1
  3. M scrive $\langle q_{\text{start}}, w\rangle$ sul nastro 2

4. Ripeti lo step seguente:
    5. Sia $\langle (x, q, y) \rangle$ la stringa attuale sul nastro 2, dove $x, y \in \Sigma^*$.

       M scansiona il nastro 1 in cerca di $\langle \delta \rangle$. Una volta trovato, M cerca una stringa $\langle (q, a), (r, b, Z) \rangle$, dove $\delta(q, a) = (r, b, Z)$ e $Z \in \{L, R\}$
    6. Se $a \neq y[i]$, M cerca la prossima regola valida
    7. Se $a = y[i]$, M scrive sul nastro due la configurazione prodotta dalla configurazione xqy passando per la transizione $\delta(q, a) = (r, b, Z)$
    8. Se nel nastro 2 è scritto $\langle q_{\text{accept}} \rangle$, M *accetta*. Se è scritto $\langle q_{\text{reject}} \rangle$, M *rifiuta*"

- Per costruzione stessa di U, si ha che:

$$\langle M, w \rangle \in L(U) \iff w \in L(M) \iff \langle M, w \rangle \in A_{\mathsf{TM}}$$

  implicando che $A_{\mathsf{TM}} = L(U) \in \mathsf{REC}$.

  Nota: poiché M potrebbe andare in loop, anche U può andare in loop, implicando che essa non sia un decisore.

□

Dimostrazione indecidibilità.

- Supponiamo per assurdo che $A_{\mathsf{TM}} \in \mathsf{DEC}$. Sia quindi H il decisore tale che $L(H) = A_{\mathsf{TM}}$
- Sia D la TM definita come:

  D = "Data in input la codifica $\langle M, w \rangle$, dove M è una TM e w una stringa:
    1. Esegui il programma di H con input $\langle M, w \rangle$
    2. Se l'esecuzione accetta, D *rifiuta*, altrimenti *accetta*"
- Per costruzione stessa di D, si ha che:

$$\langle M, w \rangle \in L(D) \iff \langle M, w \rangle \notin L(H) = A_{\mathsf{TM}} \iff w \notin L(M)$$

  Inoltre, poiché H è un decisore, ne segue che anche D sia un decisore, implicando che essa possa solo accettare o rifiutare, senza altre opzioni
- Consideriamo quindi la codifica $\langle D, \langle D \rangle \rangle$. Notiamo che:

$$\langle D, \langle D \rangle \rangle \in L(D) \iff \langle D, \langle D \rangle \rangle \notin L(H) = A_{\mathsf{TM}}$$

$$\iff \langle D \rangle \notin L(D) \iff \langle D, \langle D \rangle \rangle \notin L(D)$$

  ottenendo quindi una contrazione in quanto D possa solo accettare o rifiutare
- Di conseguenza, ne segue necessariamente che $A_{\mathsf{TM}} \notin \mathsf{DEC}$

□

Corollario 3.2: Gerarchia dei linguaggi di Chomsky

Dato un alfabeto Σ, si ha che:

$$\mathsf{REG} \subsetneq \mathsf{CFL} \subsetneq \mathsf{DEC} \subsetneq \mathsf{REC} \subsetneq \mathcal{P}(\Sigma^*)$$

(*segue dai teoremi* 2.1, 3.1, 3.16 *e* 3.17)

Definizione 3.23: Classe dei linguaggi coTuring-riconoscibili

Dato un alfabeto Σ, definiamo come **classe dei linguaggi coTuring-riconoscibili di** Σ il seguente insieme:

$$\mathsf{coREC} = \{L \subseteq \Sigma^* \mid \overline{L} \in \mathsf{REC}\}$$

Nota: $\mathsf{coREC} \neq \mathcal{P}(\Sigma^*) - \mathsf{REC}$

Teorema 3.18: $\mathsf{DEC} = \mathsf{REC} \cap \mathsf{coREC}$

Un linguaggio L è **decidibile** se e solo se è **riconoscibile** e **co-riconoscibile**.

In altre parole, si ha che:

$$\mathsf{DEC} = \mathsf{REC} \cap \mathsf{coREC}$$

Dimostrazione.

Prima implicazione.

- Dato $L \in \mathsf{DEC}$, sia M il decisore tale che $L = L(M)$
- Sia $\overline{M}$ la TM definita come:

  $\overline{M}$ = "Data la stringa w in input:

  1. Esegui il programma di M con input w
  2. Se l'esecuzione accetta, $\overline{M}$ *rifiuta*, altrimenti *accetta*"
- Per costruzione stessa di $\overline{M}$, si ha che:

  $$w \in L(\overline{M}) \iff w \notin L(M)$$

  implicando che $\overline{L} = \overline{L(M)} = L(\overline{M}) \in \mathsf{REC}$
- Dunque, poiché $L \in \mathsf{DEC} \subseteq \mathsf{REC}$ e $\overline{L} \in \mathsf{REC}$, ne segue che $L \in \mathsf{REC} \cap \mathsf{coREC}$

Seconda implicazione.

- Dato $L \in \mathsf{REC} \cap \mathsf{coREC}$, siano M e $\overline{M}$ le TM tali che $L = L(M)$ e $\overline{L} = L(\overline{M})$
- Sia D la TM definita come:

  $D =$ "Data la stringa w in input:

  1. Esegui in parallelo, ossia alternando ad ogni istruzione le loro esecuzioni, i programmi di M e $\overline{M}$ con input w
  2. Se l'esecuzione di M accetta, D *accetta*. Se l'esecuzione di $\overline{M}$ accetta, D *rifiuta*"

- Per costruzione stessa di D, si ha che:

  $$w \in L(D) \iff w \in L(M)$$

  implicando che $L(D) = L(M) = L$

- Inoltre, per definizione stessa si ha che:

  $$w \in L = L(M) \iff w \notin \overline{L} = L(\overline{M})$$

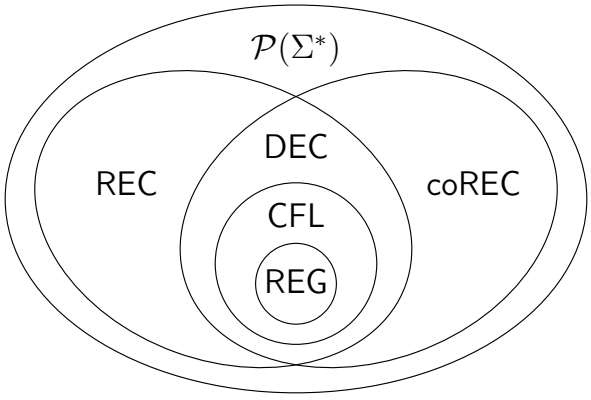
  Di conseguenza, una delle due esecuzioni parallele accetterà qualsiasi stringa in input, implicando che D non vada mai in loop e quindi che $L = L(D) \in \mathsf{DEC}$

□

Corollario 3.3: $\overline{A_{\mathsf{TM}}}$ non riconoscibile

Il linguaggio $\overline{A_{\mathsf{TM}}}$ è **irriconoscibile**

(*segue dai teoremi 3.17 e 3.18*)



Gerarchia delle classi dei linguaggi studiate fino ad ora

3.5 Riducibilità

Metodo 3.2: Riducibilità

Dati due problemi A e B, definiamo come **riduzione** il metodo dimostrativo tramite cui sapendo la soluzione di B è possibile risolvere A.

Definizione 3.24: Problema della terminazione per le TM

Definiamo il linguaggio del **problema della terminazione per le TM** come:

$$HALT_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ TM e } M(w) \text{ termina}\}$$

Teorema 3.19: $HALT_{\mathsf{TM}}$ non decidibile

Il linguaggio $HALT_{\mathsf{TM}}$ è **indecidibile**, ossia $HALT_{\mathsf{TM}} \notin \mathsf{DEC}$

Dimostrazione.

- Supponiamo per assurdo che $HALT_{\mathsf{TM}} \in \mathsf{DEC}$. Sia quindi H il decisore tale che $L(H) = HALT_{\mathsf{TM}}$
- Sia D la TM definita come:

  D = "Data la stringa $\langle M, w\rangle$ in input:

  1. Se la codifica in input è errata, D *rifiuta*
  2. Esegui il programma di H con input $\langle M, w\rangle$. Se l'esecuzione rifiuta, allora D *rifiuta*
  3. Altrimenti, D simula M con input w.
  4. Se la simulazione accetta, D *accetta*. Se rifiuta, D *rifiuta*"
- Per costruzione stessa di D, si ha che:

  $$\langle M, w\rangle \in L(D) \iff \langle M, w\rangle \in L(H), w \in L(M) \iff \langle M, w\rangle \in A_{\mathsf{TM}}$$

  implicando che $L(D) = A_{\mathsf{TM}}$.
- A questo punto, notiamo che se $\langle M, w\rangle \in L(H)$, allora la simulazione terminerà sempre. Di conseguenza, poiché H è un decisore e la simulazione termina sempre, ne segue che anche D sia un decisore, implicando che $A_{\mathsf{TM}} = L(D) \in \mathsf{DEC}$. Tuttavia, ciò risulta assurdo in quanto $A_{\mathsf{TM}} \notin \mathsf{DEC}$ (A_{TM} riconoscibile ma non decidibile) Di conseguenza, ne segue necessariamente che $HALT_{\mathsf{TM}} \notin \mathsf{DEC}$

□

Definizione 3.25: Problema del vuoto per le TM

Definiamo il linguaggio del **problema del vuoto per le TM** come:

$$E_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ TM}, L(M) = \varnothing\}$$

Teorema 3.20: E_{TM} non decidibile

Il linguaggio E_{TM} è **indecidibile**, ossia $E_{\mathsf{TM}} \notin \mathsf{DEC}$

Dimostrazione.

- Supponiamo per assurdo che $E_{\mathsf{TM}} \in \mathsf{DEC}$. Sia quindi E il decisore tale che $L(E) = E_{\mathsf{TM}}$
- Sia D la TM definita come:

  D = "Data la stringa $\langle M, w\rangle$ in input :

  1. Se la codifica in input è errata, D *rifiuta*
  2. Costruisci una TM M' definita come:

     M' = "Data la stringa x in input:

     i. Se $x \neq w$, allora *rifiuta*

     ii. Se $x = w$, esegui il programma di M con input x

     iii. Se l'esecuzione accetta, M' *accetta*"
  3. Esegui il programma di E con input $\langle M'\rangle$
  4. Se l'esecuzione accetta, D *rifiuta*. Altrimenti, D *accetta*"
- Per costruzione stessa di D, si ha che:

  $$\langle M, w\rangle \in L(D) \iff \langle M'\rangle \notin L(E) \iff L(M') = \{w\}$$

  $$\iff w \in L(M) \iff \langle M, w\rangle \in A_{\mathsf{TM}}$$

  implicando che $L(D) = A_{\mathsf{TM}}$.
- Tuttavia, poiché E è un decisore, anche D risulta esserlo, implicando che $A_{\mathsf{TM}} = L(D) \in \mathsf{DEC}$. Tuttavia, ciò risulta assurdo in quanto $A_{\mathsf{TM}} \notin \mathsf{DEC}$. Di conseguenza, ne segue necessariamente che $E_{\mathsf{TM}} \notin \mathsf{DEC}$

□

Definizione 3.26: Problema della regolarità per le TM

Definiamo il linguaggio del **problema della regolarità per le TM** come:

$$REG_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ TM}, L(M) \in \mathsf{REG}\}$$

Teorema 3.21: REG_{TM} non decidibile

Il linguaggio REG_{TM} è **indecidibile**, ossia $REG_{\mathsf{TM}} \notin \mathsf{DEC}$

Dimostrazione.

- Supponiamo per assurdo che $REG_{\mathsf{TM}} \in \mathsf{DEC}$. Sia quindi R il decisore tale che $L(R) = REG_{\mathsf{TM}}$
- Sia D la TM definita come:

  D = "Data la stringa $\langle M \rangle$ in input:

  1. Se la codifica in input è errata, D *rifiuta*
  2. Costruisci una TM M' definita come:

     M' = "Data la stringa x in input:

     i. Se $x \in \{0^n 1^n \mid n \in \mathbb{N}\}$, allora *accetta*.

     ii. Altrimenti, esegui il programma di M con input w.

     iii. Se l'esecuzione accetta, M' *accetta*, altrimenti *rifiuta*"
  3. Esegui il programma di R con input $\langle M' \rangle$.
  4. Se l'esecuzione accetta, D *accetta*. Altrimenti, D *rifiuta*"
- Supponiamo che $w \in L(M)$. In tal caso, M' accetterà qualsiasi stringa x, implicando che $L(M') = \Sigma^* \in \mathsf{REG}$
- Supponiamo ora che $w \notin L(M)$. In tal caso, abbiamo che:
  - Se $x \in \{0^n 1^n \mid n \in \mathbb{N}\}$, allora $x \in L(M')$
  - Se $x \notin \{0^n 1^n \mid n \in \mathbb{N}\}$, allora $x \notin L(M')$ poiché M' andrà in loop

  di conseguenza, otteniamo che $L(M') = \{0^n 1^n \mid n \in \mathbb{N}\} \notin \mathsf{REG}$ (sezione 1.6)
- Di conseguenza, concludiamo che $w \in L(M) \iff L(M') \in \mathsf{REG}$
- A questo punto, per costruzione stessa di D, si ha che:

$$\langle M, w \rangle \in L(D) \iff \langle M' \rangle \in L(R) \iff L(M') \in \mathsf{REG}$$

$$\iff w \in L(M) \iff \langle M, w \rangle \in A_{\mathsf{TM}}$$

  implicando che $L(D) = A_{\mathsf{TM}}$.

- Tuttavia, poiché R è un decisore, anche D risulta esserlo, implicando che $A_{\mathsf{TM}} = L(D) \in \mathsf{DEC}$. Tuttavia, ciò risulta assurdo in quanto $A_{\mathsf{TM}} \notin \mathsf{DEC}$. Di conseguenza, ne segue necessariamente che $REG_{\mathsf{TM}} \notin \mathsf{DEC}$

□

Definizione 3.27: Problema dell'equivalenza per le TM

Definiamo il linguaggio del **problema del'equivalenza per le TM** come:

$$EQ_{\mathsf{TM}} = \{\langle M, M' \rangle \mid M, M' \ \mathsf{TM}, L(M) = L(M')\}$$

Teorema 3.22: EQ_{TM} non decidibile

Il linguaggio EQ_{TM} è **indecidibile**, ossia $EQ_{\mathsf{TM}} \notin \mathsf{DEC}$

Dimostrazione.

- Supponiamo per assurdo che $EQ_{\mathsf{TM}} \in \mathsf{DEC}$. Sia quindi E il decisore tale che $L(E) = EQ_{\mathsf{TM}}$
- Sia D la TM definita come:

  D = "Data la stringa $\langle M, w \rangle$ in input:

  1. Se la codifica in input è errata, D *rifiuta*
  2. Costruisci una TM M' definita come:

     M' = "Data la stringa x in input:

     i. *Rifiuta*"
  3. Esegui il programma di E con input $\langle M, M' \rangle$.
  4. Se l'esecuzione accetta, D *accetta*. Altrimenti, D *rifiuta*"
- Per costruzione stessa di D, si ha che:

  $$\langle M, w \rangle \in L(D) \iff \langle M, M' \rangle \in L(R) \iff L(M) = L(M') = \varnothing \iff \langle M \rangle \in E_{\mathsf{TM}}$$

  implicando che $L(D) = E_{\mathsf{TM}}$.
- Tuttavia, poiché E è un decisore, anche D risulta esserlo, implicando che $E_{\mathsf{TM}} = L(D) \in \mathsf{DEC}$. Tuttavia, ciò risulta assurdo in quanto sappiamo che $E_{\mathsf{TM}} \notin \mathsf{DEC}$ (E_{TM} non decidibile)
- Di conseguenza, ne segue necessariamente che $EQ_{\mathsf{TM}} \notin \mathsf{DEC}$

□

3.5.1 Riducibilità tramite mappatura

Definizione 3.28: Funzione calcolabile

Data $f : \Sigma^* \to \Sigma^*$, definiamo f come **calcolabile** se esiste una TM F tale che:

$$\forall w \in \Sigma^* \quad F(w) \text{ termina con solo } f(w) \text{ sul nastro}$$

Nota: dalla definizione risulta implicito che F debba terminare sempre

Definizione 3.29: Riducibilità tramite mappatura

Dati due linguaggi A e B, con $A, B \neq \varnothing, \Sigma^*$, diciamo che A è **riducibile a B tramite mappatura**, indicato come $A \leq_m B$, se esiste una funzione calcolabile $f : \Sigma^* \to \Sigma^*$, detta **riduzione da A a B**, tale che:

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

Teorema 3.23: Decidibilità tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m B$, si ha che:

$$B \in \mathsf{DEC} \implies A \in \mathsf{DEC}$$

Dimostrazione.

- Dato $B \in \mathsf{DEC}$, sia D_B il decisore tale che $L(D_B) = B$
- Sia D_A la TM definita come:

  $D_A =$ "Data la stringa w in input:

  1. Calcola $f(w)$
  2. Esegui il programma di D_B con input $f(w)$.
  3. Se l'esecuzione accetta, D *accetta*. Altrimenti, D *rifiuta*"
- Per costruzione stessa di D_A, si ha che:

  $$w \in L(D_A) \iff f(w) \in L(D_B) = B \iff w \in A$$

  implicando che $L(D_A) = A$. Inoltre, poiché D_B è un decisore e poiché f è calcolabile, ne segue che anche D_A sia un decisore e quindi che $A = L(D_A) \in \mathsf{DEC}$

□

Corollario 3.4: Indecidibilità tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m B$, si ha che:

$$A \notin \mathsf{DEC} \implies B \notin \mathsf{DEC}$$

Esempi:

1.
   - Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

     $F =$ "Data la stringa $\langle M, w\rangle$ in input:

     1. Costruisci una TM M' definita come:

        $M' =$ "Data la stringa x in input:

        i. Esegui il programma di M con input x.

        ii. Se l'esecuzione accetta, M' *accetta*. Altrimenti, M' muove la testina a destra per sempre (va in loop)"

     2. Restituisci in output la stringa $\langle M', w\rangle$"
   - Notiamo che:

     $$\langle M, w\rangle \in A_{\mathsf{TM}} \iff w \in L(M) \implies w \in L(M')$$
     $$\implies f(\langle M, w\rangle) = \langle M', w\rangle \in HALT_{\mathsf{TM}}$$

     e inoltre che:

     $$\langle M, w\rangle \notin A_{\mathsf{TM}} \iff w \notin L(M) \implies M'(w) \text{ va in loop}$$
     $$\implies f(\langle M, w\rangle) = \langle M', w\rangle \notin HALT_{\mathsf{TM}}$$

   - Di conseguenza, poiché:

     $$\langle M, w\rangle \in A_{\mathsf{TM}} \iff f(\langle M, w\rangle) \in HALT_{\mathsf{TM}}$$

     ne segue che $A_{\mathsf{TM}} \leq_m HALT_{\mathsf{TM}}$
   - Infine, poiché $A_{\mathsf{TM}} \notin \mathsf{DEC}$, concludiamo che $HALT_{\mathsf{TM}} \notin \mathsf{DEC}$
2.
   - Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

     $F =$ "Data la stringa $\langle M\rangle$ in input:

     1. Costruisci una TM M' definita come:

        $M' =$ "Data la stringa x in input:

        i. *Rifiuta*"

     2. Restituisci in output la stringa $\langle M, M'\rangle$"
   - Notiamo che:

     $$\langle M\rangle \in E_{\mathsf{TM}} \iff L(M) = \varnothing = L(M') \iff f(\langle M\rangle) = \langle M, M'\rangle \in EQ_{\mathsf{TM}}$$

   - Di conseguenza, poiché:

     $$\langle M\rangle \in E_{\mathsf{TM}} \iff f(\langle M\rangle) \in EQ_{\mathsf{TM}}$$

     ne segue che $E_{\mathsf{TM}} \leq_m EQ_{\mathsf{TM}}$
   - Infine, poiché $E_{\mathsf{TM}} \notin \mathsf{DEC}$, concludiamo che $EQ_{\mathsf{TM}} \notin \mathsf{DEC}$

Teorema 3.24: Riconoscibilità tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m B$, si ha che:

$$B \in \mathsf{REC} \implies A \in \mathsf{REC}$$

(*dimostrazione analoga al teorema 3.23*)

Corollario 3.5: Irriconoscibilità tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m B$, si ha che:

$$A \notin \mathsf{REC} \implies B \notin \mathsf{REC}$$

Teorema 3.25: Riducibilità complementare

Dati due linguaggi A e B, si ha che:

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

Dimostrazione.

- Data la riduzione f tale che $A \leq_m B$, si ha che:

$$w \in \overline{A} \iff w \notin A \iff f(w) \notin B \iff f(w) \in \overline{B}$$

□

Teorema 3.26: EQ_{TM} non riconoscibile e non co-riconoscibile

Il linguaggio EQ_{TM} non è né riconoscibile né co-riconoscibile

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  F = "Data la stringa $\langle M, w \rangle$ in input:

  1. Costruisci una TM M_1 definita come:

     M_1 = "Data la stringa x in input:

     i. *Rifiuta*"

  2. Costruisci una TM M_2 definita come:

     M_2 = "Data la stringa x in input:

     i. Esegui il programma di M con input w. Se l'esecuzione accetta, M_2 *accetta*. Altrimenti, *rifiuta*"

  3. Restituisci in output la stringa $\langle M_1, M_2 \rangle$"

- Notiamo che:
$$\langle M, w\rangle \in A_{\mathsf{TM}} \implies L(M_1) = \varnothing, L(M_2) = \Sigma^* \implies L(M_1) \neq L(M_2)$$
$$\iff f(\langle M, w\rangle) = \langle M_1, M_2\rangle \notin EQ_{\mathsf{TM}} \iff f(\langle M, w\rangle) \in \overline{EQ}_{\mathsf{TM}}$$
e inoltre che:
$$\langle M, w\rangle \notin A_{\mathsf{TM}} \implies L(M_1) = \varnothing, L(M_2) = \varnothing \implies L(M_1) = L(M_2)$$
$$\iff f(\langle M, w\rangle) = \langle M_1, M_2\rangle \in EQ_{\mathsf{TM}} \iff f(\langle M, w\rangle) \notin \overline{EQ_{\mathsf{TM}}}$$

- Di conseguenza, poiché:
$$\langle M, w\rangle \in A_{\mathsf{TM}} \iff f(\langle M, w\rangle) \in \overline{EQ_{\mathsf{TM}}}$$
ne segue che $A_{\mathsf{TM}} \leq_m \overline{EQ_{\mathsf{TM}}}$

- Sia inoltre $g : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM G definita come:

  $G =$ "Data la stringa $\langle M, w\rangle$ in input:

  1. Costruisci una TM M_1 definita come:

     $M_1 =$ "Data la stringa x in input:

     i. *Accetta*"

  2. Costruisci una TM M_2 definita come:

     $M_2 =$ "Data la stringa x in input:

     i. Esegui il programma di M con input w. Se l'esecuzione accetta, M_2 *accetta*. Altrimenti, *rifiuta*"

  3. Restituisci in output la stringa $\langle M_1, M_2\rangle$"

- Notiamo che:
$$\langle M, w\rangle \in A_{\mathsf{TM}} \implies L(M_1) = \Sigma^*, L(M_2) = \Sigma^* \implies L(M_1) = L(M_2)$$
$$\iff g(\langle M, w\rangle) = \langle M_1, M_2\rangle \in EQ_{\mathsf{TM}}$$
e inoltre che:
$$\langle M, w\rangle \notin A_{\mathsf{TM}} \implies L(M_1) = \Sigma^*, L(M_2) = \varnothing \implies L(M_1) \neq L(M_2)$$
$$\iff g(\langle M, w\rangle) = \langle M_1, M_2\rangle \notin EQ_{\mathsf{TM}}$$

- Di conseguenza, poiché:
$$\langle M, w\rangle \in A_{\mathsf{TM}} \iff g(\langle M, w\rangle) \in EQ_{\mathsf{TM}}$$
ne segue che $A_{\mathsf{TM}} \leq_m EQ_{\mathsf{TM}}$

- A questo punto, per la Riducibilità complementare, si ha che:
$$A_{\mathsf{TM}} \leq_m \overline{EQ_{\mathsf{TM}}} \iff \overline{A_{\mathsf{TM}}} \leq_m EQ_{\mathsf{TM}}$$
$$A_{\mathsf{TM}} \leq_m EQ_{\mathsf{TM}} \iff \overline{A_{\mathsf{TM}}} \leq_m \overline{EQ_{\mathsf{TM}}}$$

- Infine, poiché $\overline{A_{\mathsf{TM}}} \notin \mathsf{REC}$, ne segue automaticamente che $\overline{EQ_{\mathsf{TM}}} \notin \mathsf{REC}$

□

3.6 Teoremi di incompletezza di Gödel

Agli inizi del XIX secolo, David Hilbert – uno dei matematici più influenti del suo tempo – tenne durante il Congresso Internazionale dei Matematici a Parigi una conferenza di importanza storica, denominata "Conferenza dei 23 problemi". All'interno di tale conferenza, Hilbert mostrò le sue idee riguardo il futuro della matematica, ponendo particolare attenzione su 23 problemi – tutti al tempo irrisolti – su cui sosteneva si sarebbe basata l'intera ricerca matematica del prossimo secolo.

Tra tali problemi, vi era anche l'obiettivo di **formalizzare** la matematica tramite un **logica**, riformulare tutte le fondamenta della matematica in termini di verità logiche e regole rigorose che eliminassero ogni forma di ambiguità e arbitrarietà al fine di sviluppare un **sistema logico**, un sistema di regole tramite cui generare nuove verità – una sorta di procedura algoritmica che partendo dalle basi possa generare ogni possibile teorema. Tra i suoi obiettivi principali c'era quello di *assiomatizzare* le fondamenta matematiche tramite un sistema logico adatto e di *dimostrare* che tale sistema logico fosse:

- **Consistente**, ossia che non possa generare contraddizioni
- **Completo**, ossia che sia in grado di dimostare ogni affermazione vera
- **Decidibile**, ossia che esista un algoritmo per determianare la verità o falsità di ogni proposizione

Questo obiettivo ciò che diventò noto come *Programma di Hilbert*: la speranza che la matematica potesse essere messa su fondamenta perfettamente solide e meccanicamente verificabili.

Nel 1931, dopo aver assistito ad una conferenza di Hilbert dove esponeva nuovamente il suo programma, il logico austriaco Kurt Gödel dimostrò in un solo giorno un risultato che distrusse immediatamente il sogno di Hilbert: non esiste un sistema logico abbastanza potente da contenere l'aritmetica (ossia ogni sistema che possa essere di nostro interesse) che sia contemporaneamente consistente e completo. Questo risultato è noto come **primo teorema di incompletezza**.

Negli anni successivi, come già sappiamo, Alan Turing dimostrerà come non esista alcun un sistema logico che sia decidibile, altrimenti potremmo utilizzare tale sistema per decidere il problema della terminazione per le TM. Per tanto, ogni sistema logico abbastanza potente è **indecidibile**, eliminando direttamente la terza caratteristica voluta da Hilbert. Tra le due caratteristiche rimanenti, ossia la *consistenza* e la *completezza*, siamo decisamente più interessati alla prima proprietà: possiamo accettare (a malincuore) che alcune affermazioni vere non siano dimostrabili, l'importante è che il nostro sistema non vada a dedurre che una affermazione contemporaneamente vera e falsa.

Tuttavia, nell'arco di poco tempo Gödel dimostrò un **secondo teorema di incompletezza**, il quale afferma che nessun sistema consistente possa dimostrare la propria consistenza. Tale risultato implica che non possiamo far altro che *sperare* che il sistema logico attualmente utilizzato per le fondamenta della matematica, ossia il sistema ZFC (Zermelo-Fraenkel + Assioma della Scelta), sia consistente, il che ovviamente lo rende incompleto per via del primo teorema.

Nonostante le dimostrazioni originali di Gödel siano molto lunghe e complesse, tramite il risultato di Turing è possibile dimostrare entrambi i teoremi in modo molto facile.

Definizione 3.30: Sistema di prova

Definiamo un sistema logico Π come **sistema di dimostrazione** se:

- Π possiede un insieme di regole logiche tramite cui effettuare deduzioni.
- Per ogni dimostrazione w producibile da Π tramite applicazioni successive delle regole di Π esiste una codifica $\langle w \rangle$ che descrive w
- Esiste un decisore V_Π tale che $\forall \langle x, w \rangle \in \Sigma^*$ si ha che $\langle x, w \rangle \in L(V_\Pi)$ se e solo se w è una dimostrazione in Π per l'affermazione x

Un'affermazione $\langle x \rangle \in \Sigma^*$ è detta:

- **Dimostrabile** in Π se e solo se $\exists \langle w \rangle \in \Sigma^*$ tale che $\langle x, w \rangle \in L(V_\Pi)$.
- **Indipendente** da Π se e solo se né x né $\neg x$ sono dimostrabili in Π, dove $\neg x$ è l'affermazione esprimente l'opposto di x.

Un sistema di prova Π è detto:

- **Consistente** se per ogni affermazione $\langle x \rangle \in \Sigma^*$ al massimo una tra x e $\neg x$ è dimostrabile
- **Valido** se ogni affermazione dimostrabile è vera
- **Completo** se ogni affermazione vera è dimostrabile
- **Incompleto** se esiste un'affermazione indipendente

Osservazione 3.2

Se un sistema è valido allora esso è anche consistente poichè se x è dimostrabile allora essa è anche vera, implicando che $\neg x$ sia falsa e quindi non dimostrabile. Viceversa, un sistema consistente non è garantito essere valido.

Per dare un'idea di come dimostrare i teoremi di Gödel tramite le TM, dimostriamo un caso speciale del primo teorema sostituendo la consistenza con la validità.

Proposizione 3.3

Sia Π un sistema di dimostrazione abbastanza potente da comprendere l'aritmentica. Allora, Π non può essere sia valido che completo.

Dimostrazione.

- Sia Π un sistema abbastanza potente da comprendere l'aritmentica. Allora, tale sistema è in grado di descrivere il funzionamento di una TM.

- Definiamo quindi la seguente TM P_Π.

  P_Π = "Data la stringa $\langle x \rangle$ in input:

  1. Verifica che x sia un'affermazione. Se falso, rifiuta.
  2. Ripeti per ogni $k = 1, 2, 3, \ldots$:
     3. Ripeti per ogni $\langle w \rangle \in \Sigma^*$ con $|w| = k$:
        4. Se $V_\Pi(x, w)$ accetta, P_Π accetta"
        5. Se $V_\Pi(\neg x, w)$ accetta, P_Π rifiuta"

- Supponiamo per assurdo che Π sia valido e completo. Osserviamo che le due proprietà assieme implicano che un'affermazione x sia vera se e solo se esiste una sua dimostrazione in Π.

- Per il principio del *tertium non datur*, almeno una tra x e $\neg x$ deve essere vera. Abbiamo quindi che:

  – Se x è vera ciò può accadere se e solo se esiste una dimostrazione w per cui $V_\Pi(x, w)$ accetta, dunque $\langle x \rangle \in L(P_\Pi)$

  – Se $\neg x$ è vera ciò può accadere se e solo se esiste una dimostrazione w per cui $V_\Pi(\neg x, w)$ accetta, dunque $\langle x \rangle \notin L(P_\Pi)$

  Per tanto, le due proprietà implicano che P_Π sia un decisore e che $L(P_\Pi) = \{\langle x \rangle \mid x \text{ è vera}\}$.

- Data una TM M e un input $y \in \Sigma^*$, consideriamo la seguente affermazione:

$$\phi_{M,y} = \text{“}M(y) \text{ termina”}$$

- Sia D la TM definita come $D(\langle M, w \rangle) = P_\Pi(\langle \phi_{M,y} \rangle)$. Poiché P_Π è un decisore, anche D risulta esserlo. Tuttavia, notiamo che $L(D) = \text{HALT}_{\text{TM}}$, contraddicendo $HALT_{\text{TM}}$ non decidibile. Per tanto, Π non può essere sia valido che completo.

□

Lemma 3.1

Sia Π un sistema di dimostraazione abbastanza potente da comprendere l'aritmetica. Sia M una TM e sia $x \in \Sigma^*$ un input. Se $M(x)$ ha una traccia di esecuzione il cui comportamento è noto, allora esiste una dimostrazione in Π che descrive ciò

Dimostrazione.

- Sia Π un sistema abbastanza potente da comprendere l'aritmentica. Allora, tale sistema è in grado di descrivere il funzionamento di una TM. Per tanto, se il comportamento di $M(x)$ è noto allora è possibile dimostrare tale fatto tramite una dimostrazione che emula gli spostamenti della testina tramite le regole di Π

□

Osservazione 3.3

Il comportamento della traccia di esecuzione di $M(x)$ può essere noto non solo se la TM M termina sull'input x, ma anche se $M(x)$ va in loop volontariamente

Teorema 3.27: Primo Teorema di Incompletezza

Sia Π un sistema di dimostrazione abbastanza potente da comprendere l'aritmentica. Se Π è consistente allora Π non è completo.

Dimostrazione.

- Sia Π un sistema abbastanza potente da comprendere l'aritmentica. Allora, tale sistema è in grado di descrivere il funzionamento di una TM.
- Data una TM P_Π, consideriamo la seguente affermazione:

$$\phi_M = \text{“}M(\langle M\rangle) \text{ termina”}$$

- Definiamo quindi la seguente TM P_Π.

  P_Π = "Data la stringa $\langle x\rangle$ in input:

  1. Verifica che x sia un'affermazione. Se falso, rifiuta.
  2. Ripeti per ogni $k = 1, 2, 3, \ldots$:
     3. Ripeti per ogni $\langle w\rangle \in \Sigma^*$ con $|w| = k$:
        4. Se $V_\Pi(\phi_{P_\Pi}, w)$ accetta, P_Π va in loop volontariamente"
        5. Se $V_\Pi(\neg\phi_{P_\Pi}, w)$ accetta, P_Π termina"

- **Affermazione**: se ϕ_{P_Π} o $\neg\phi_{P_\Pi}$ è dimostrabile allora Π è inconsistente

  Dimostrazione:

  – Supponiamo che $\neg\phi_{P_\Pi}$ sia dimostrabile. Dunque, esiste una dimostrazione w tale che $V_\Pi(\neg\phi_{P_\Pi}, w)$ accetti, implicando che $P_\Pi(\langle P_\Pi\rangle)$ termina.

  – Poichè l'esecuzione termina, il comportamento della traccia è noto. Dunque, per il lemma precedente, esiste una dimostrazione che descrive la traccia di esecuzione di P_Π, la quale è anche una dimostrazione per ϕ_{P_Π}.

  – Similmente, supponiamo che ϕ_{P_Π} sia dimostrabile. Dunque, esiste una dimostrazione w tale che $V_\Pi(\phi_{P_\Pi}, w)$ accetti, implicando che $P_\Pi(\langle P_\Pi\rangle)$ va in loop volontariamente

  – Poiché l'esecuzione va in loop volontariamente, il comportamento della traccia noto anche in questo caso. Dunque, per il lemma precedente, esiste una dimostrazione che descrive la traccia di esecuzione di P_Π, la quale è anche una dimostrazione per $\neg\phi_{P_\Pi}$.

- Assumiamo che Π sia consistente. Per contronominale dell'affermazione, otteniamo che né ϕ_{P_Π} né $\neg\phi_{P_\Pi}$ sia dimostrabile in Π.
- Per costruzione di P_Π, dunque, la computazione $P_\Pi(\langle P_\Pi \rangle)$ non troverà mai una dimostrazione valida, andando in loop. Ciò implica che $\neg\phi_{P_\Pi}$ sia un'affermazione vera ma non dimostrabile, concludendo che Π non sia completo.

□

Teorema 3.28: Secondo Teorema di Incompletezza

Sia Π un sistema di dimostrazione abbastanza potente da comprendere l'aritmentica. Se Π è consistente allora esso non può dimostrare l'affermazione "Π è consistente"

Dimostrazione.

- Sia Π un sistema abbastanza potente da comprendere l'aritmentica. Allora, tale sistema è in grado di descrivere il funzionamento di una TM.
- Sia P_Π la TM definita tramite ϕ_{P_Π} esattamente come nella dimostrazione del teorema precedente.
- **Affermazione**: se ϕ_{P_Π} o $\neg\phi_{P_\Pi}$ è dimostrabile allora Π è inconsistente

  Dimostrazione: Analoga al teorema precedente
- Assumiamo che Π sia consistente. Per contronominale dell'affermazione, otteniamo che né ϕ_{P_Π} né $\neg\phi_{P_\Pi}$ sia dimostrabile in Π.
- Supponiamo quindi per assurdo che l'affermazione "Π è consistente" sia dimostrabile tramite $w \in \Sigma^*$. Allora, unendo w alla dimostrazione dell'affermazione precedente, otteniamo una dimostrazione per l'affermazione "né ϕ_{P_Π} né $\neg\phi_{P_\Pi}$ sia dimostrabile in Π"
- Tuttavia, per costruzione di P_Π, ciò corrisponde anche ad una dimostrazione per $\neg\phi_{P_\Pi}$ in quanto è garantito che la macchina vada in loop, creando una contraddizione. Concludiamo quindi che "Π è consistente" non sia dimostrabile in Π.

□

Corollario 3.6

Sia Π un sistema di dimostrazione abbastanza potente da comprendere l'aritmentica. Se Π è consistente allora esso non è completo e non può dimostrare la sua consistenza.

3.7 Esercizi svolti

Problema 3.1

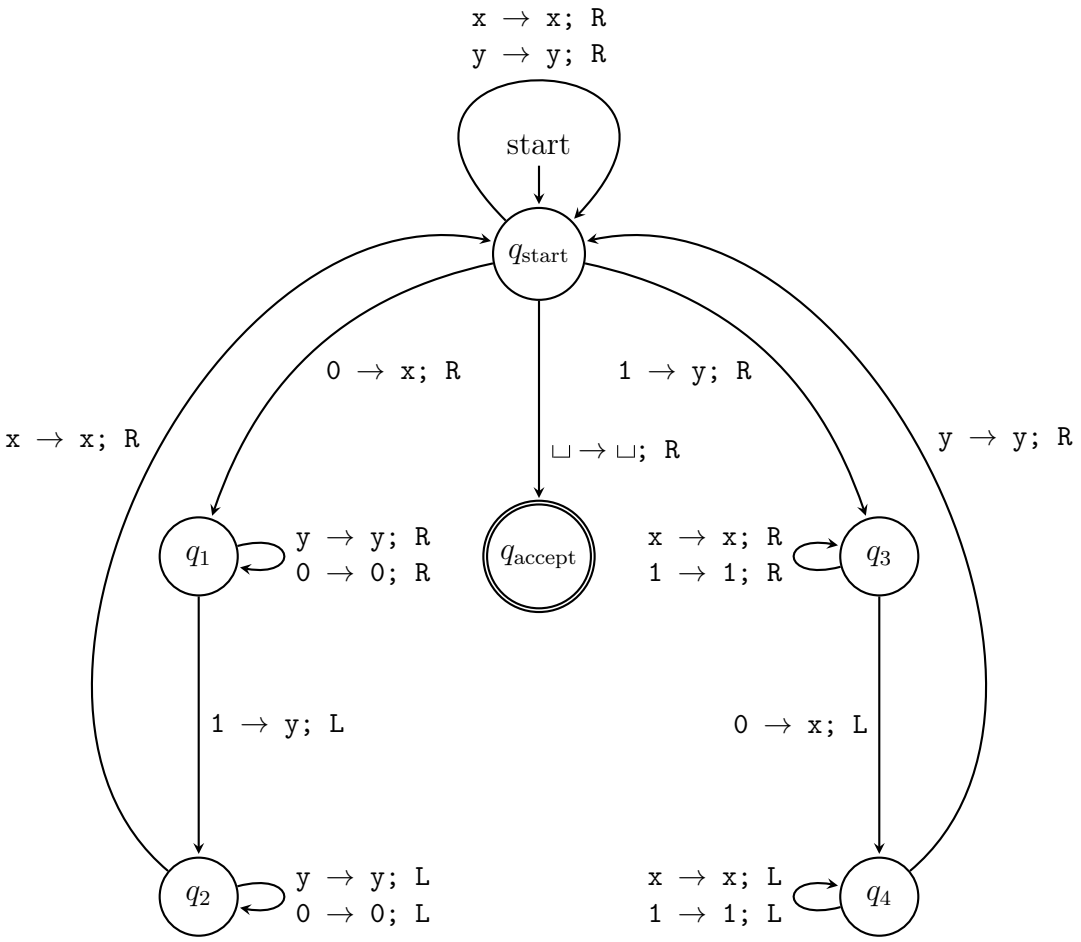
Descrivere formalmente una TM che riconosca il linguaggio delle stringhe binarie contenenti lo stesso numero di 0 ed 1. Dimostrare la correttezza della soluzione proposta

Dimostrazione.

- Sia L il linguaggio richiesto, dove:

$$L = \{w \in \{0,1\}^* \mid |w|_0 = |w|_1\}$$

- Sia $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ la TM definita come:
  - $\Sigma = \{0, 1\}$
  - $\Gamma = \{0, 1, x, y, \sqcup\}$
  - Gli stati di Q e le transizioni di δ sono definiti dal seguente diagramma:



(*tutte le transizioni omesse vanno a* q_{reject})

- Notiamo che, se lo stato attuale è q_{start}, si verifica che:
  - Se viene letto uno 0, il cammino $C := q_{\text{start}} \to q_1 \to q_2 \to q_{\text{start}}$ marca tale 0 con una x, per poi scorrere la testina a destra fino al primo 1 presente sul nastro, il quale verrà marcato con una y. Successivamente, la testina scorrerà a sinistra fino alla prima x presente sul nastro, la quale corrisponderà esattamente con lo 0 precedentemente marcato
  - Se viene letto un 1, il cammino $C' := q_{\text{start}} \to q_3 \to q_4 \to q_{\text{start}}$ marca tale 1 con una y, per poi scorrere la testina a destra fino al primo 0 presente sul nastro, il quale verrà marcato con una x. Successivamente, la testina scorrerà a sinistra fino alla prima y presente sul nastro, la quale corrisponderà esattamente con l'1 precedentemente marcato
  - Il cammino $q_{\text{start}} \to q_{\text{start}}$ assicura che, ad ogni lettura di uno 0, un 1 o un $\sqcup$ non vi siano x o y a destra del simbolo letto
- Supponiamo che $w \in L$. Poiché $|w|_0 = |w|_1$, ad ogni 0 in w corrisponderà un 1 in w. Di conseguenza, la computazione di w su M può percorrere correttamente i cammini C e C' rispettivamente per n ed m volte, tornando sempre allo stato q_{start}. Infine, verrà letto il simbolo $\sqcup$, portando la computazione nello stato q_{accept}, implicando che $w \in L(M)$
- Supponiamo invece che $w \in L(M)$. Essendo una stringa accettata da M, ne segue che la sua computazione percorra correttamente i cammini C e C' per rispettivamente i e j volte. Di conseguenza, poiché ad ogni passaggio su uno dei due cammini vengono marcati uno 0 ed un 1, ne segue che:

$$|w|_0 = i + j = |w|_1 \implies w \in L$$

- Dunque, concludiamo che:

$$w \in L \iff w \in L(M)$$

  implicando che $L = L(M)$

□

Problema 3.2: TM a nastro infinito in entrambe le direzioni

Una Bi-TM è una TM in cui il nastro di lavoro è infinito in entrambe le direzioni. Dato un linguaggio $L \subseteq \Sigma^*$ dimostrare che:

$$L \in \mathsf{REC} \iff \exists \text{ Bi-TM} M \text{ t.c. } L = L(M)$$

Dimostrazione.

Prima implicazione.

- Dato $L \in \mathsf{REC}$, sia M la TM tale che $L = L(M)$
- Poiché una TM è una particolare Bi-TM il cui lato infinito sinistro del nastro non viene mai utilizzato, ne segue automaticamente che essa stessa sia la Bi-TM in grado di riconoscere $L = L(M)$

Seconda implicazione.

- Sia M la Bi-TM tale che $L = L(M)$
- Consideriamo la TM M' definita come:

  M' = "Data in input la stringa w:

  1. Marca con # la cella più a sinistra (ossia la prima cella del nastro)
  2. Esegui il programma di M su input w
  3. Ogni volta che l'esecuzione raggiunge la cella contenente #, scorri di una cella a destra tutto il contenuto del nastro e riprendi l'esecuzione"

- Per costruzione stessa di M', si ha che:

  $$w \in L = L(M) \iff w \in L(M')$$

  concludendo che $L = L(M) = L(M') \in \mathsf{REC}$

□

Problema 3.3: Insieme di linguaggi distinti

Dato un alfabeto Σ, siano $L_1, \ldots, L_k \subseteq \Sigma^*$, dove $k \in \mathbb{N}$, dei linguaggi tali che:

1. $\forall i \neq j \;\; L_i \cap L_j = \varnothing$
2. $L_1 \cup \ldots \cup L_k = \Sigma^*$
3. $\forall i \in [1,k] \;\; L_i \in \mathsf{REC}$, ossia sono Turing-riconoscibili

Dimostrare che $L_1, \ldots, L_k \in \mathsf{DEC}$, ossia sono decidibili

Dimostrazione.

- Le prime due proprietà dei linguaggi $L_1, \ldots, L_k$ implicano che essi siano una partizione di Σ^*
- Difatti, tramite esse, $\forall i \in [1,k]$ si ha che:

$$w \in \overline{L_i} \iff w \in \Sigma^* - L_i \iff \exists h \neq i \in [1,k], w \in L_h \iff w \in \bigcup_{j \neq i} L_j$$

  implicando che:

$$\forall i \in [1,k] \;\; \overline{L_i} = \bigcup_{j \neq i} L_j$$

- Poiché $L_1, \ldots, L_k \in \mathsf{REC}$, siano $M_1, \ldots, M_k$ le TM tali che $\forall i \in [1,k] \;\; L_i = L(M_i)$
- Dato $i \in [1,k]$, consideriamo una TM $\overline{M_i}$ definita come:

  $\overline{M_i}$ = "Data la stringa w in input:

  1. Esegui in parallelo, ossia alternando ad ogni istruzione le loro esecuzioni, i programmi di $M_1, \ldots, M_{i-1}, M_{i+1}, \ldots, M_k$ con input w
  2. Se una delle esecuzioni accetta, $\overline{M_i}$ *accetta*"

- Per costruzione stessa di $\overline{M_i}$, si ha che:

$$w \in L(\overline{M_i}) \iff \overline{L_i} = \bigcup_{j \neq i} L_j \iff w \in \overline{L_i}$$

  implicando che $\overline{L_i} = L(\overline{M_i}) \in \mathsf{REC}$, ossia che L_i sia coTuring-riconoscibile

- Di conseguenza, poiché:

$$L_i \in \mathsf{DEC} \iff L_i \in \mathsf{REC}, L_i \in \mathsf{coREC} \iff L_i \in \mathsf{REC}, \overline{L_i} \in \mathsf{REC}$$

  ne segue automaticamente che $L_1, \ldots, L_k$ siano decidibili

□

Problema 3.4: Riducibilità al proprio complemento

Dato un linguaggio A tale che $A \leq_m \overline{A}$, dimostrare che:

$$A \in \mathsf{REC} \implies A \in \mathsf{DEC}$$

Dimostrazione.

- Poiché $A \leq_m \overline{A}$, per la Riducibilità complementare abbiamo che:

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

- Supponiamo quindi che $A \in \mathsf{REC}$. In tal caso, ne segue che:

$$\overline{A} \leq_m A, A \in \mathsf{REC} \implies \overline{A} \in \mathsf{REC} \implies A \in \mathsf{coREC}$$

- Di conseguenza, poiché $\mathsf{DEC} = \mathsf{REC} \cap \mathsf{coREC}$, concludiamo che:

$$A \in \mathsf{REC} \cap \mathsf{coREC} \iff A \in \mathsf{DEC}$$

□

Problema 3.5: $START\text{-}0_{\mathsf{TM}}$ indecidibile

Dato il seguente linguaggio:

$$START\text{-}0_{\mathsf{TM}} = \{\langle M \rangle \mid M \ \mathsf{TM}, L(M) = \{0y \mid y \in \Sigma^*\}\}$$

dimostrare che è **indecidibile**, ossia che $START\text{-}0_{\mathsf{TM}} \notin \mathsf{DEC}$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  $F =$ "Data la stringa $\langle M, w \rangle$ in input:

  1. Costruisci una TM M' definita come:

     $M' =$ "Data la stringa x in input:

     i. Se $x \notin \{0y \mid y \in \Sigma^*\}$, *rifiuta*. Altrimenti, esegui il programma di M su input w

     ii. Se l'esecuzione accetta, allora M' *accetta*, altrimenti *rifiuta*"

  2. Restituisci in output la stringa $\langle M' \rangle$"

- Supponiamo che $w \notin L(M)$. In tal caso, M' rifiuterà qualsiasi stringa, implicando che $L(M') = \varnothing$ e dunque che $\langle M' \rangle \notin START\text{-}0_{\mathsf{TM}}$

- Supponiamo ora che $w \in L(M)$. In tal caso, abbiamo che:
  - Se $x \notin \{0w \mid w \in \Sigma^*\}$, allora $x \notin L(M')$
  - Se $x \in \{0w \mid w \in \Sigma^*\}$, allora $x \in L(M')$ poiché $w \in L(M)$

  di conseguenza, otteniamo che:

$$x \in L(M') \iff x \in \{0w \mid w \in \Sigma^*\}$$

  implicando che $\langle M' \rangle \in START\text{-}0_{\mathsf{TM}}$
- Di conseguenza, concludiamo che $w \in L(M) \iff L(M') \in START\text{-}0_{\mathsf{TM}}$
- A questo punto, notiamo che:

$$\langle M, w \rangle \in A_{\mathsf{TM}} \iff w \in L(M) \iff f(\langle M, w \rangle) = \langle M' \rangle \in START\text{-}0_{\mathsf{TM}}$$

  implicando quindi che $A_{\mathsf{TM}} \leq_m START\text{-}0_{\mathsf{TM}}$
- Infine, poiché $A_{\mathsf{TM}} \notin \mathsf{DEC}$, concludiamo che $START\text{-}0_{\mathsf{TM}} \notin \mathsf{DEC}$

□

Problema 3.6: W_{TM} indecidibile

Dato il seguente linguaggio:

$$W_{\mathsf{TM}} = \left\{ \langle M, w, a \rangle \;\middle|\; \begin{array}{c} M = (Q, \Sigma_M, \Gamma_M, \delta, q_s, q_a, q_r) \text{ TM}, \\ w \in \Sigma_M^*,\ a \in \Gamma_M, \\ M \text{ scrive } a \text{ su input } w \end{array} \right\}$$

dimostrare che è **indecidibile**, ossia che $W_{\mathsf{TM}} \notin \mathsf{DEC}$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  $F =$ "Data la stringa $\langle M, w \rangle$ in input:

  1. Costruisci una TM M' definita come:

     $M' =$ "Data la stringa x in input:

     i. Esegui il programma di M su input x

     ii. Se l'esecuzione accetta, scrivi a sul nastro, dove a è un simbolo non appartenente all'alfabeto di nastro di M, e *accetta*

     iii. Altrimenti, *rifiuta*"

  2. Restituisci in output la stringa $\langle M', w, a \rangle$"
- Notiamo che:

$$\langle M, w \rangle \in A_{\mathsf{TM}} \iff w \in L(M) \iff M'(w) \text{ scrive } a \text{ sul nastro}$$

$$\iff f(\langle M, w\rangle) = \langle M', w, a\rangle \in W_{\mathsf{TM}}$$

implicando quindi che $A_{\mathsf{TM}} \leq_m W_{\mathsf{TM}}$

- Infine, poiché $A_{\mathsf{TM}} \notin \mathsf{DEC}$, concludiamo che $W_{\mathsf{TM}} \notin \mathsf{DEC}$

□

Problema 3.7: $UNION\text{-}ALL_{\mathsf{TM}}$ indecidibile

Dato il seguente linguaggio:

$$UNION\text{-}ALL_{\mathsf{TM}} = \{\langle T, T'\rangle \mid T, T' \ \mathsf{TM}, L(T) \cup L(T') = \Sigma^*\}$$

dimostrare che è **indecidibile**, ossia che $UNION\text{-}ALL_{\mathsf{TM}} \notin \mathsf{DEC}$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  $F =$ "Data la stringa $\langle M, w\rangle$ in input:

  1. Costruisci una TM T definita come:

     $T =$ "Data la stringa x in input:

     i. *Rifiuta*"

  2. Costruisci una TM T' definita come:

     $T' =$ "Data la stringa x in input:

     i. Esegui il programma di M con input w.

     ii. Se l'esecuzione accetta, T' *accetta*, altrimenti *rifiuta*"

  3. Restituisci in output la stringa $\langle T, T'\rangle$"

- Notiamo che:

$$\langle M, w\rangle \in A_{\mathsf{TM}} \iff w \in L(M) \implies L(T) = \varnothing, L(T') = \Sigma^* \implies L(T) \cup L(T') = \Sigma^*$$
$$\iff f(\langle M, w\rangle) = \langle T, T'\rangle \in UNION\text{-}ALL_{\mathsf{TM}}$$

  e inoltre che:

$$\langle M, w\rangle \notin A_{\mathsf{TM}} \iff w \notin L(M) \implies L(T) = \varnothing, L(T') = \varnothing \implies L(T) \cup L(T') = \varnothing$$
$$\implies f(\langle M, w\rangle) = \langle T, T'\rangle \notin UNION\text{-}ALL_{\mathsf{TM}}$$

  implicando quindi che $A_{\mathsf{TM}} \leq_m UNION\text{-}ALL_{\mathsf{TM}}$

- Infine, poiché $A_{\mathsf{TM}} \notin \mathsf{DEC}$, concludiamo che $UNION\text{-}ALL_{\mathsf{TM}}$

□

Problema 3.8: ODD_{TM} indecidibile

Dato il seguente linguaggio:

$$ODD_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ TM}, L(M) = \{w \in \Sigma^* \mid |w| \text{ dispari}\}\}$$

dimostrare che è **indecidibile**, ossia che $ODD_{\mathsf{TM}} \notin \mathsf{DEC}$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  $F =$ "Data la stringa $\langle M, w \rangle$ in input:

  1. Costruisci una TM M' definita come:

     $M' =$ "Data la stringa x in input:

     i. Se $|x|$ è pari, *rifiuta*. Altrimenti, esegui il programma di M su input w

     ii. Se l'esecuzione accetta, allora M' *accetta*, altrimenti *rifiuta*"

  2. Restituisci in output la stringa $\langle M' \rangle$"

- Supponiamo che $w \notin L(M)$. In tal caso, M' rifiuterà qualsiasi stringa, implicando che $L(M') = \varnothing$ e dunque che $\langle M' \rangle \notin ODD_{\mathsf{TM}}$

- Supponiamo ora che $w \in L(M)$. In tal caso, abbiamo che:

  – Se $|x|$ è pari, allora $x \notin L(M')$

  – Se $|x|$ è dispari, allora $x \in L(M')$ poiché $M(w)$ accetta

  di conseguenza, otteniamo che:

  $$x \in L(M') \iff |x| \text{ dispari}$$

  implicando che $\langle M' \rangle \in ODD_{\mathsf{TM}}$

- Di conseguenza, concludiamo che $w \in L(M) \iff L(M') \in ODD_{\mathsf{TM}}$

- A questo punto, notiamo che:

  $$\langle M, w \rangle \in A_{\mathsf{TM}} \iff w \in L(M) \iff f(\langle M, w \rangle) = \langle M' \rangle \in ODD_{\mathsf{TM}}$$

  implicando quindi che $A_{\mathsf{TM}} \leq_m ODD_{\mathsf{TM}}$

- Infine, poiché $A_{\mathsf{TM}} \notin \mathsf{DEC}$, concludiamo che $ODD_{\mathsf{TM}} \notin \mathsf{DEC}$

□

Problema 3.9: ODD_{TM} irriconoscibile

Dato il seguente linguaggio:

$$ODD_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ TM}, L(M) = \{w \in \Sigma^* \mid |w| \text{ dispari}\}\}$$

dimostrare che è **irriconoscibile**, ossia che $ODD_{\mathsf{TM}} \notin \mathsf{REC}$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  $F =$ "Data la stringa $\langle M, w \rangle$ in input:

  1. Costruisci una TM M' definita come:

     $M' =$ "Data la stringa x in input:

     i. Se $|x|$ è dispari, *accetta*. Altrimenti, esegui il programma di M su input w

     ii. Se l'esecuzione accetta, allora M' *accetta*, altrimenti *rifiuta*"

  2. Restituisci in output la stringa $\langle M' \rangle$"

- Supponiamo che $w \in L(M)$. In tal caso, M' accetterà qualsiasi stringa, implicando che $L(M') = \Sigma^*$ e dunque che $\langle M' \rangle \notin ODD_{\mathsf{TM}}$

- Supponiamo ora che $w \notin L(M)$. In tal caso, abbiamo che:

  – Se $|x|$ è dispari, allora $x \in L(M')$

  – Se $|x|$ è pari, allora $x \notin L(M')$ poiché $M(w)$ va in loop o rifiuta

  di conseguenza, otteniamo che:

  $$x \in L(M') \iff |x| \text{ dispari}$$

  implicando che $\langle M' \rangle \in ODD_{\mathsf{TM}}$

- Di conseguenza, concludiamo che $w \in L(M) \iff L(M') \notin ODD_{\mathsf{TM}}$

- A questo punto, notiamo che:

  $$\langle M, w \rangle \in \overline{A_{\mathsf{TM}}} \iff \langle M, w \rangle \notin A_{\mathsf{TM}} \iff w \notin L(M)$$

  $$\iff f(\langle M, w \rangle) = \langle M' \rangle \in ODD_{\mathsf{TM}}$$

  implicando quindi che $\overline{A_{\mathsf{TM}}} \leq_m ODD_{\mathsf{TM}}$

- Infine, poiché $\overline{A_{\mathsf{TM}}} \notin \mathsf{REC}$, concludiamo che $ODD_{\mathsf{TM}} \notin \mathsf{REC}$

□

Problema 3.10: $DECIDABLE_{\mathsf{TM}}$ irriconoscibile

Dato il seguente linguaggio:

$$DECIDABLE_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ TM}, L(M) \in \mathsf{DEC}\}$$

dimostrare che è **irriconoscibile**, ossia che $DECIDABLE_{\mathsf{TM}} \notin \mathsf{REC}$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  F = "Data la stringa $\langle M, w \rangle$ in input:

  1. Costruisci una TM M' definita come:

     M' = "Data la stringa x in input:

     i. Esegui il programma di M su input w.

     ii. Se l'esecuzione rifiuta, M' *rifiuta*

     iii. Altrimenti, interpreta $x = \langle M'', y \rangle$ ed esegui M'' su input y.

     iv. Se l'esecuzione accetta, M' *accetta*. Altrimenti, M' *rifiuta*.

  2. Restituisci in output la stringa $\langle M' \rangle$"

- Supponiamo che $\langle M, w \rangle \in A_{\mathsf{TM}}$. In tal caso, la macchina M' salterà le prime due istruzioni, comportandosi come un normalissimo riconoscitore di A_{TM}, linguaggio che sappiamo essere indecidibile.

$$\langle M, w \rangle \in A_{\mathsf{TM}} \implies L(M) = A_{\mathsf{TM}} \implies f(\langle M, w \rangle) = \langle M' \rangle \notin DECIDABLE_{\mathsf{TM}}$$

- Viceversa, se $\langle M, w \rangle \notin A_{\mathsf{TM}}$ allora l'esecuzione $M(w)$ all'interno di M' andrà o in loop o rifiuterà. In entrambi i casi, M' non accetterà alcuna stringa, dunque $L(M) = \varnothing$, linguaggio decidibile da una macchina che rifiuta sempre.

$$\langle M, w \rangle \notin A_{\mathsf{TM}} \implies L(M) = \varnothing \implies f(\langle M, w \rangle) = \langle M' \rangle \in DECIDABLE_{\mathsf{TM}}$$

- Di conseguenza, abbiamo che:

$$\langle M, w \rangle \in \overline{A_{\mathsf{TM}}} \iff \langle M, w \rangle \in A_{\mathsf{TM}} \iff f(\langle M, w \rangle) = \langle M' \rangle \in DECIDABLE_{\mathsf{TM}}$$

  implicando quindi che $\overline{A_{\mathsf{TM}}} \leq_m DECIDABLE_{\mathsf{TM}}$. Poiché $\overline{A_{\mathsf{TM}}} \notin \mathsf{REC}$, concludiamo che $DECIDABLE_{\mathsf{TM}} \notin \mathsf{REC}$

□

Problema 3.11: $USELESS_{\mathsf{TM}}$ indecidibile

Dato il seguente linguaggio:

$$USELESS_{\mathsf{TM}} = \{\langle M \rangle \mid M \text{ TM con almeno uno stato inutile } \}$$

dimostrare che è **indecidibile**, ossia che $USELESS_{\mathsf{TM}} \notin \mathsf{DEC}$

Nota: uno stato di un TM è detto *inutile* se non esiste un input per cui tale stato venga raggiunto almeno una volta durante l'esecuzione

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  $F =$ "Data la stringa $\langle M, w \rangle$ in input:

  1. Costruisci una TM M' definita come:

     $M' =$ "Data la stringa x in input:

     i. In M' esistono solo tre stati: $q_{\text{start}}, q_{\text{accept}}$ e q_{reject}

     ii. Se $x = w$, *rifiuta*

     iii. Se $x \neq w$, esegui il programma di M su input w

     iv. Se l'esecuzione accetta, M' *rifiuta*, altrimenti *accetta*"

  2. Restituisci in output la stringa $\langle M' \rangle$"

- Essendo la TM M' dotata solo di tre stati, notiamo che:
  - Lo stato q_{start} è raggiungibile da ogni esecuzione di M', dunque non è mai considerabile uno stato inutile
  - Lo stato q_{reject} è raggiungibile dalle esecuzioni di M' su input w, dunque non è mai considerabile uno stato inutile
  - Lo stato q_{accept} è raggiungibile se e solo se $w \notin L(M)$

  di conseguenza, concludiamo che $w \in L(A) \iff$ Esiste uno stato inutile in M'

- A questo punto, si ha che:

$$\langle M, w \rangle \in A_{\mathsf{TM}} \iff w \in L(M) \iff \text{Esiste uno stato inutile in } M'$$

$$\iff f(\langle M, w \rangle) = \langle M' \rangle \in USELESS_{\mathsf{TM}}$$

  implicando quindi che $A_{\mathsf{TM}} \leq_m USELESS_{\mathsf{TM}}$

- Infine, poiché $A_{\mathsf{TM}} \notin \mathsf{DEC}$, concludiamo che $USELESS_{\mathsf{TM}} \notin \mathsf{DEC}$

□

Problema 3.12: REJ_{TM} riconoscibile ma indecidibile

Dato il seguente linguaggio:

$$REJ_{\mathsf{TM}} = \{\langle M, w\rangle \mid M \text{ TM che rifiuta } w\}$$

dimostrare che sia **riconoscibile** ma **indecidibile**, ossia che $REJ_{\mathsf{TM}} \in \mathsf{REC}-\mathsf{DEC}$

Dimostrazione riconoscibilità.

- Sia M' la TM definita come:

  M' = "Data la stringa $\langle M, w\rangle$ in input:

  1. Simula M su input w
  2. Se la simulazione rifiuta, M' *accetta*. Se la simulazione accetta, *rifiuta*"

- Per costruzione stessa di M', si ha che:

  $$\langle M, w\rangle \in L(M') \iff M(w) \text{ rifiuta} \iff \langle M, w\rangle \in REJ_{\mathsf{TM}}$$

  concludendo che $REJ_{\mathsf{TM}} = L(M) \in \mathsf{REC}$

  Nota: poiché M potrebbe andare in loop, anche M' può andare in loop, implicando che essa non sia un decisore.

□

Dimostrazione indecidibilità.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  F = "Data la stringa $\langle M, w\rangle$ in input:

  1. Costruisci una TM M' definita come:

     M' = "Data la stringa x in input:

     i. Esegui il programma di M su input w

     ii. Se l'esecuzione accetta, M' *rifiuta*, altrimenti *accetta*"

  2. Restituisci in output la stringa $\langle M', w\rangle$"

- Per costruzione di M', risulta evidente che:

  $$\langle M, w\rangle \in A_{\mathsf{TM}} \iff M(w) \text{ accetta} \implies f(\langle M, w\rangle) = \langle M', w\rangle \in REJ_{\mathsf{TM}}$$

  e inoltre che:

  $$\langle M, w\rangle \notin A_{\mathsf{TM}} \iff M(w) \text{ va in loop o rifiuta} \implies f(\langle M, w\rangle) = \langle M', w\rangle \notin REJ_{\mathsf{TM}}$$

  implicando quindi che $A_{\mathsf{TM}} \leq_m REJ_{\mathsf{TM}}$

- Infine, poiché $A_{\mathsf{TM}} \notin \mathsf{DEC}$, concludiamo che $REJ_{\mathsf{TM}} \notin \mathsf{DEC}$

□

Problema 3.13: ALL_{NFA} decidibile

Dato il seguente linguaggio:

$$ALL_{\mathsf{NFA}} = \{\langle N \rangle \mid N \text{ NFA tale che } L(N) = \Sigma^*\}$$

dimostrare che sia **decidibile**, ossia $ALL_{\mathsf{NFA}} \in \mathsf{DEC}$.

Dimostrazione:

- Prima di tutto ricordiamo che, il teorema Equivalenza tra NFA e DFA, sappiamo che ogni NFA abbia un DFA equivalente ad esso.
- Inoltre, dato il seguente linguaggio:

  $$EQ_{\mathsf{DFA}} = \{\langle A, B \rangle \mid A, B \text{ DFA}, L(A) = L(B)\}$$

  tramite il teorema EQ_{DFA} decidibile sappiamo che $EQ_{\mathsf{DFA}} \in \mathsf{DEC}$. Di conseguenza, esiste una TM E tale che tale che $L(E) = EQ_{\mathsf{DFA}}$.
- Sia quindi M la TM definita come:

  M = "Data la stringa $\langle N \rangle$ in input, dove N è un NFA:

  1. Converti N in un DFA D tale che $L(N) = L(D)$, salvando il risultato in una stringa $\langle D \rangle$.
  2. Sia Σ l'alfabeto descritto in $\langle N \rangle$. Costruisci il DFA D' dotato di un unico stato q_0, il quale è accettante, ed un'unica transizione $q_0 \to q_0$ avente per etichetta l'intero alfabeto Σ.
  3. Esegui la procedura E su input $\langle D, D' \rangle$. Se la procedura accetta, anche M *accetta*, altrimenti *rifiuta*."
- Per costruzione stessa di M abbiamo che:

  $$\langle N \rangle \in L(M) \implies \langle D, D' \rangle \in L(E) = EQ_{\mathsf{DFA}}$$

  $$\implies L(N) = L(D) = L(D') = \Sigma^* \implies \langle N \rangle \in ALL_{\mathsf{NFA}}$$

  e viceversa che:

  $$\langle N \rangle \notin L(M) \implies \langle D, D' \rangle \notin L(E) = EQ_{\mathsf{DFA}}$$

  $$\implies L(N) = L(D) \neq L(D') = \Sigma^* \implies \langle N \rangle \notin ALL_{\mathsf{NFA}}$$

  concludendo che $L(D) = ALL_{\mathsf{NFA}}$ e dunque che $ALL_{\mathsf{NFA}} \in \mathsf{DEC}$.

4 Complessità

Finora abbiamo considerato possibili soluzioni a problemi senza considerare le **risorse** necessarie a risolvere tali problemi. Difatti, dire che un problema (dunque un linguaggio) sia decidibile equivale a dire che vi sia sempre una soluzione a tale problema, ma non che tale soluzione sia effettivamente utilizzabile.

In particolare, quindi, vogliamo studiare:

- Le risorse necessarie ad una TM per risolvere un problema, in particolare il **tempo**, ossia il numero di passi, e lo **spazio**, ossia la memoria necessaria
- **Classificare** i problemi in base alle risorse necessarie

4.1 Complessità temporale

Definizione 4.1: Complessità temporale di una TM

Sia D un decisore. Definiamo come **complessità temporale** (o *tempo di esecuzione*) di D la funzione $f : \mathbb{N} \to \mathbb{N}$ tale che $f(n)$ sia il massimo numero di passi necessari a D per processare una stringa di lunghezza n.

Esempio:

- Sia M il decisore definito come:

  M = "Data la stringa w in input:

  1. Muovi la testina a destra finché non viene letto il simbolo $\sqcup$
  2. *Accetta*"

- Data in input una stringa w tale che $|w| = n$, la TM M impiega n passi
- Di conseguenza, il suo tempo computazionale è $f(n) = n$

Definizione 4.2: O grande

Siano $f, g : \mathbb{N} \to \mathbb{R}^+$. Diciamo che $f(n)$ **è in O grande di** $g(n)$, indicato come $f(n) = O(g(n))$, se:

$$\exists c, n_0 \in \mathbb{N}_{>0} \mid \forall n \geq n_0 \;\; f(n) \leq c \cdot g(n)$$

In altre parole, $g(n)$ corrisponde al **limite superiore asintotico** di $f(n)$

Esempio:

- Date le due funzioni $f(n) = 100n^2$ e $g(n) = n^3$, esistono $c = 10$ e $n_0 = 10$ tali che:

  $$\forall n \geq 100 \;\; f(n) = 100n^2 \leq 10 \cdot n^3$$

  di conseguenza, si ha che $100n^2 = O(n^3)$

Definizione 4.3: o piccolo

Siano $f, g : \mathbb{N} \to \mathbb{R}^+$. Diciamo che $f(n)$ **è in o piccolo di** $g(n)$, indicato come $f(n) = o(g(n))$, se:

$$\lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0$$

In altre parole, si ha che:

$$\forall c \in \mathbb{R}_{>0} \; \exists n_0 \in \mathbb{N}_{>0} \mid \forall n \geq n_0 \;\; f(n) < c \cdot g(n)$$

Esempio:

- Date le due funzioni $f(n) = 100n^2$ e $g(n) = n^3$, si ha che $100n^2 = o(n^3)$

Osservazione 4.1

Date due funzioni $f, g : \mathbb{N} \to \mathbb{R}^+$, si ha che:

$$f(g) = o(g(n)) \implies f(g) = O(g(n))$$

Proposizione 4.1: Algebra asintotica

Date le funzioni $f, g, h : \mathbb{N} \to \mathbb{R}^+$, si ha che:

- $\forall c \in \mathbb{R} \;\; f(n) = c \cdot o(g(n)) \implies f(n) = o(g(n))$
- $f(n) = o(g(n)) + o(h(n)) \implies f(n) = o(m(n))$, dove $m(n) = \max(g(n), h(n))$
- $f(n) = o(g(n)) \cdot o(h(n)) \implies f(n) = o(g(n) \cdot h(n))$

Nota: per l'osservazione precedente, ciò vale anche per O grande

Teorema 4.1: Rapporto di tempo tra TM multinastro e TM

Sia $t(n)$ una funzione tale che $t(n) \geq n$. Per ogni TM multinastro M con tempo $t(n)$ esiste una TM M' tale che $L(M) = L(M')$ avente tempo $O(t^2(n))$

Dimostrazione.

- Consideriamo la TM S in grado di simulare una TM multinastro vista nella dimostrazione dell'Equivalenza tra TM e TM multinastro
- Analizziamo quindi la complessità temporale di S:
  - Il numero di nastri k è indipendente dall'input
  - Per preparare il nastro sono necessari $k \cdot O(n) = O(n)$ passi
  - Per ogni passo simulato della TM multinastro, S richiede $k \cdot O(t(n)) = O(t(n))$ passi
  - Il numero di passi della TM multinastro è $t(n)$
- Di conseguenza, la complessità temporale di S risulta essere:

$$t'(n) = O(n) + t(n) \cdot O(t(n)) = O(n) + O(t^2(n)) = O(t^2(n))$$

□

Definizione 4.4: Complessità temporale di una NTM

Sia N un decisore non-deterministico. Definiamo come **complessità temporale** di N la funzione $f : \mathbb{N} \to \mathbb{N}$ tale che $f(n)$ sia il massimo numero di passi necessari ad ogni ramo dell'esecuzione di D per processare una stringa di lunghezza n

Teorema 4.2: Rapporto di tempo tra NTM e TM

Sia $t(n)$ una funzione tale che $t(n) \geq n$. Per ogni NTM N con tempo $t(n)$ esiste una TM M' tale che $L(M) = L(M')$ avente tempo $2^{O(t(n))}$

Dimostrazione.

- Consideriamo la TM M in grado di simulare una NTM vista nella dimostrazione dell'Equivalenza tra TM e NTM
- Analizziamo quindi la complessità temporale di M:
  - Il numero massimo b di figli di ogni nodo è indipendente dall'input
  - Di conseguenza, il numero di foglie dell'albero computazionale risulta essere $k \leq b^{t(n)}$, implicando che il numero massimo di nodi sia $m \leq 2 \cdot b^{t(n)}$, dunque $m = O(b^{t(n)})$

  - M simula ogni nodo dell'albero computazionale ripartendo sempre dalla radice, dunque esegue $O(t(n) \cdot b^{t(n)})$ passi

- A questo punto, notiamo che:

$$t(n) \cdot b^{t(n)} = 2^{\log_2(t(n) \cdot b^{t(n)})} = 2^{\log_2(t(n)) + t(n) \cdot \log_2 b} = 2^{O(t(n))}$$

- Di conseguenza, la complessità temporale di M risulta essere:

$$t'(n) = O(t(n) \cdot b^{t(n)}) = O(2^{O(t(n))}) = 2^{O(t(n))}$$

□

4.2 Classe P

Definizione 4.5: Classe DTIME

Data la funzione $t : \mathbb{N} \to \mathbb{R}^+$, definiamo come **classe dei linguaggi decidibili in tempo** $O(t(n))$ il seguente insieme:

$$\mathsf{DTIME}(t(n)) = \{L \in \mathsf{DEC} \mid L \text{ decidibile da una } \mathsf{TM} \text{ in tempo } O(t(n))\}$$

Nota: nella definizione ammettiamo solo le TM, dunque non le sue varianti

Esempio:

- Consideriamo il linguaggio $L = \{0^n 1^n \mid n \in \mathbb{N}\}$
- Consideriamo il seguente decisore M:

  M = "Data la stringa w in input:

  1. Scansiona l'input per vedere se sia nella forma 0^*1^*. Se falso, *rifiuta*
  2. Torna all'inizio del nastro. Esegui il seguente loop:
     3. Marca la cella attuale con x e cerca il prossimo 1 sul nastro.
     4. Se viene trovato, marcalo con x. Altrimenti, *rifiuta*
     5. Torna all'inizio e cerca il primo 0 sul nastro. Se non viene trovato, *rifiuta*
  6. Se ci sono solo x sul nastro, *accetta*. Altrimenti, *rifiuta*"

  implicando che $L = L(M)$

- Analizziamo quindi la complessità temporale di M:
  - La prima e l'ultima istruzione richiedono entrambe n passi, dunque hanno costo $O(2n) = O(n)$
  - Il ritorno all'inizio del nastro richiede massimo n passi, dunque ha costo $O(n)$

  - Ogni iterazione del loop richiede massimo n passi per cercare il simbolo 1, massimo n passi per tornare all'inizio del nastro e massimo n passi per arrivare al primo 0. Di conseguenza, ogni iterazione ha costo $O(3n) = O(n)$
  - Il loop viene eseguito per massimo $\frac{n}{2}$ iterazioni

- Di conseguenza, la complessità temporale di M risulta essere:

$$t(n) = O(n) + \frac{n}{2} \cdot O(n) = O(n^2)$$

concludendo che $L \in \mathsf{DTIME}(n^2)$

Definizione 4.6: Classe P

Definiamo la **classe dei linguaggi decidibili in tempo polinomiale** come:

$$\mathsf{P} = \bigcup_{k=0}^{+\infty} \mathsf{DTIME}(n^k)$$

Definizione 4.7: Problema dei cammini

Definiamo il linguaggio del **problema dei cammini** come:

$$PATH = \{\langle G, s, t\rangle \mid G = (V_G, E_G) \text{ grafo diretto con un cammino } s \to t\}$$

Teorema 4.3: $PATH$ polinomialmente decidibile

Il linguaggio $PATH$ è **polinomialmente decidibile**, ossia $PATH \in \mathsf{P}$

Dimostrazione.

- Sia M la TM definita come:

  M = "Data la stringa $\langle G, s, t\rangle$ in input:

  1. Se la codifica in input è errata, M *rifiuta*
  2. Marca il vertice s
  3. Ripeti lo step seguente finché vengono marcati dei nuovi vertici:
     4. Marca ogni vertice avente un arco entrante da un vertice già marcato
  5. Se tra i vertici marcati vi è t, allora M *accetta*, altrimenti *rifiuta*"

- Di conseguenza, la complessità temporale di M risulta essere:

$$t(n) = O(n^k) + |E_G| \cdot O(n^k) + O(n^k) = O(n^k)$$

implicando che $PATH \in \mathsf{DTIME}(n^k) \subseteq \mathsf{P}$

□

Definizione 4.8: Classe EXP

Definiamo la **classe dei linguaggi decidibili in tempo esponenziale** come:

$$\mathsf{EXP} = \bigcup_{k=1}^{\infty} \mathsf{DTIME}(2^{n^k})$$

Definizione 4.9: Problema dei cammini hamiltoniani

Definiamo il linguaggio del **problema dei cammini hamiltoniani** come:

$$HAMPATH = \left\{ \langle G, s, t \rangle \;\middle|\; \begin{array}{c} G = (V_G, E_G) \text{ grafo diretto con un} \\ \text{cammino hamiltoniano } s \to t \end{array} \right\}$$

dove un cammino è detto *hamiltoniano* se passa una sola volta per ogni nodo del grafo di appartenenza

Teorema 4.4: $HAMPATH$ esponenzialmente decidibile

Il linguaggio $HAMPATH$ è **esponenzialmente decidibile**, ossia $HAMPATH \in \mathsf{EXP}$

Dimostrazione.

- Sia M la TM definita come:

  M = "Data la stringa $\langle G, s, t \rangle$ in input:

  1. Ripeti per ogni cammino possibile in G:
     2. Verifica se il cammino è hamiltoniano. Se lo è, *accetta*
  3. *Rifiuta*"

- Poiché il numero di cammini possibili in un grafo al massimo 2^{n^k} per qualche $k \in \mathbb{N}$ e poiché verificare se un cammino è hamiltoniamo richiede tempo $O(n^h)$ per qualche $h \in \mathbb{N}$, la complessità temporale di M risulta essere:

  $$t(n) = O(n^h) \cdot O(2^{n^k}) = O(2^{n^k})$$

  implicando che $HAMPATH \in \mathsf{DTIME}(2^{n^k}) \subseteq \mathsf{EXP}$

□

4.3 Classe NP

Definizione 4.10: Classe NTIME

Data la funzione $t : \mathbb{N} \to \mathbb{R}^+$, definiamo come **classe dei linguaggi decidibili non-deterministicamente in tempo** $O(t(n))$ il seguente insieme:

$$\mathsf{NTIME}(t(n)) = \{L \in \mathsf{DEC} \mid L \text{ decidibile da una } \mathsf{NTM} \text{ in tempo } O(t(n))\}$$

Proposizione 4.2

Dato un linguaggio L, si ha che:

$$\mathsf{NTIME}(n^k) \subseteq \mathsf{DTIME}(2^{O(n^k)})$$

(*segue dal teorema* 4.2)

Definizione 4.11: Classi NP e NEXP

Definiamo la **classe dei linguaggi decidibili non-deterministicamente in tempo polinomiale** come:

$$\mathsf{NP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(n^k)$$

Analogamente, definiamo la **classe dei linguaggi decidibili non-deterministicamente in tempo esponenziale** come:

$$\mathsf{NEXP} = \bigcup_{k=1}^{\infty} \mathsf{NTIME}(2^{n^k})$$

Corollario 4.1: Gerarchia temporale dei linguaggi

Date le classi P, NP e EXP, si ha che:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP}$$

Dimostrazione.

- Per definizione stessa, una TM risulta essere anche una NTM. Di conseguenza, ne segue automaticamente che $\mathsf{P} \subseteq \mathsf{NP}$ e che $\mathsf{EXP} \subseteq \mathsf{NEXP}$
- Inoltre, per il corollario precedente, abbiamo che:

$$L \in \mathsf{NP} \implies \exists k \in \mathbb{N} \mid L \in \mathsf{NTIME}(n^k) \implies L \in \mathsf{DTIME}(2^{O(n^k)}) \implies L \in \mathsf{EXP}$$

□

Definizione 4.12: $CLIQUE$

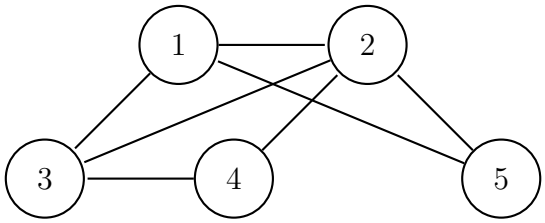
Definiamo come $CLIQUE$ il seguente linguaggio:

$$CLIQUE = \{\langle G, k\rangle \mid G \text{ grafo non diretto con una } k\text{-clique}\}$$

dove una k-clique è un sottoinsieme di nodi dove ognuno di quest'ultimi è adiacente a tutti gli altri nodi del sottoinsieme

Esempio:

- All'interno del seguente grafo, gli insiemi di nodi $C_1 = \{1, 2, 3\}$ e $C_2 = \{1, 2, 5\}$ formano due 3-clique



Teorema 4.5: $CLIQUE$ **non deter. polinomialmente decibile**

Il linguaggio $CLIQUE$ è **non deterministicamente polinomialmente decidibile**, ossia $CLIQUE \in$ NP

Dimostrazione.

- Sia N la NTM definita come:

  $N =$ "Data la stringa $\langle G, k\rangle$ in input, dove $G = (V_G, E_G)$ è un grafo:

  1. Scegli non deterministicamente un sottoinsieme C di k nodi appartenenti a G
  2. Ripeti lo step successivo per ogni coppia di nodi (v_i, v_j) in C
     4. Se $(v_i, v_j) \notin E_G$, allora *rifiuta*
  3. *Accetta*"

- Essendo N una NTM, l'esecuzione del primo passo genererà un albero computazionale in cui ogni ramo analizza uno solo tra tutti i possibili sottoinsiemi di k nodi. In particolare, se almeno uno di questi rami analizzerà una k-clique valida, allora la stringa verrà accettata

- Di conseguenza, si ha che:

  $$\langle G, k\rangle \in CLIQUE \iff \text{Esiste una } k\text{-clique in } G \iff$$

  $$\text{Esiste un ramo di } N \text{ che accetta } \langle G, k\rangle \iff \langle G, k\rangle \in L(N)$$

  implicando che $CLIQUE = L(N)$

- Inoltre, poiché per ogni sottoinsieme di k nodi le coppie possibili sono $\binom{k}{2} = \frac{k^2-k}{2}$, la verifica della k-clique richiede tempo polinomiale, implicando che $CLIQUE = L(N) \in \mathsf{NTIME}(n^h)$ per qualche $h \in \mathbb{N}$

□

Definizione 4.13: Verificatore

Dato un linguaggio $L \in \mathsf{DEC}$, definiamo un decisore V come **verificatore di** L se:

$$L = \{w \in \Sigma^* \mid \exists c \in \Sigma^* \quad \langle w, c\rangle \in L(V)\}$$

dove la stringa c viene detta **certificato di appartenenza**

Osservazione 4.2: Tempo di esecuzione di un verificatore

Sia V un verificatore. Data una stringa $\langle w, c\rangle \in L(V)$, il **tempo di esecuzione di** V viene misurato in base alla **lunghezza di** w, ignorando la lunghezza del certificato.

Se il tempo di esecuzione di un verificatore è $f(n)$, assumiamo che i suoi certificati siano di **lunghezza** $O(f(n))$, poiché altrimenti essi verrebbero solo parzialmente letti

Teorema 4.6: Classe NP (definizione alternativa)

Data la classe NP, si ha che:

$$\mathsf{NP} = \{L \in \mathsf{DEC} \mid \exists\ V \text{ verificatore polinomiale per } L\}$$

In altre parole, possiamo definire NP anche come la **classe dei linguaggi verificabili in tempo polinomiale**

Dimostrazione.

Prima implicazione.

- Dato $L \in \mathsf{NP}$, sia N la NTM tale che $L = L(N)$
- Durante la sua computazione, N effettuerà una serie di scelte, creando così il suo albero di computazione
- Sia V la TM definita come:

  V = "Data la stringa $\langle w, c\rangle$ in input:

  1. Interpreta c come una serie di scelte da effettuare
  2. Simula N su input w eseguendo le scelte dettate da c
  3. Se la simulazione accetta, V *accetta*. Altrimenti, *rifiuta*"

- Per costruzione stessa di V, si ha che:

$$w \in L = L(N) \iff \text{Esiste un ramo di } N \text{ che accetta } w$$

$$\iff \exists c \in \Sigma^*, \langle w, c \rangle \in L(V)$$

  dunque V risulta essere un verificatore di $L = L(N)$

- Inoltre, poiché $L = L(N) \in \mathsf{NP}$, la lunghezza massima dei rami dell'albero di computazione risulta essere $O(n^k)$. Di conseguenza, ogni certificato può avere al massimo $O(n^k)$ scelte, implicando che V sia un verificatore polinomiale

Seconda implicazione.

- Dato un linguaggio L, supponiamo che esista un verificatore V che verifica L in tempo polinomiale, implicando che:

$$\exists k \in \mathbb{N} \mid V \text{ verifica } L \text{ in } O(n^k)$$

- Sia N la NTM definita come:

  $N =$ "Data la stringa $\langle w \rangle$ in input:

  1. Scegli non deterministicamente una stringa c di lunghezza $O(n^k)$
  2. Simula V su input $\langle w, c \rangle$
  3. Se la simulazione accetta, B *accetta*. Altrimenti, *rifiuta*"

- Essendo N una NTM, l'esecuzione del primo passo genererà un albero computazionale in cui ogni ramo analizza una sola tra tutte le possibili stringhe di lunghezza $O(n^k)$. In particolare, almeno uno di questi rami analizzerà un certificato valido

- Di conseguenza, si ha che:

$$w \in L \iff \exists c \in \Sigma^*, \langle w, c \rangle \in V \iff$$

$$\text{Esiste un ramo di } N \text{ che accetta } w \iff x \in L$$

  implicando che $L \in L(N) \in \mathsf{NP}$

□

Definizione 4.14: Problema della tricolorazione dei grafi

Definiamo il linguaggio del **problema della tricolorazione dei grafi** come:

$$3COL = \{\langle G \rangle \mid G = (V_G, E_G) \text{ grafo 3-colorabile}\}$$

dove un grafo è detto *3-colorabile* se ad ogni suo nodo è possibile assegnare un colore diverso dai suoi nodi adiacenti utilizzando massimo tre colori

Teorema 4.7: $3COL$ **polinomialmente verificabile**

Il linguaggio $3COL$ è **polinomialmente verificabile**, ossia $3COL \in \mathsf{NP}$

Dimostrazione.

- Sia V la TM definita come:

  V = "Data la stringa $\langle \langle G \rangle, c \rangle$ in input, dove $G = (V_G, E_G)$ è un grafo:

  1. Interpreta c come $c = \langle c_1, \ldots, c_m \rangle$, dove $m = |V_G|$ e $\forall k \in [1, m] \;\; c_k \in \{R, G, B\}$
  2. Ripeti lo step successivo per ogni arco $(v_i, v_j) \in E_G$:
     4. Se $c_i = c_j$, allora *rifiuta*
  5. *Accetta*"

- Per costruzione stessa di V, si ha che:

  $$\langle G \rangle \in 3COL \iff \text{Esiste una colorazione valida } c_1, \ldots, c_n$$

  $$\iff \exists c = \langle c_1, \ldots, c_n \rangle \in \Sigma^*, \langle \langle G \rangle, c \rangle \in V$$

  dunque V risulta essere un verificatore di $3COL$

- Analizziamo quindi il costo di V:
  - La prima istruzione è eseguibile in $O(|w|) = O(n)$
  - La seconda istruzione è eseguibile in $O(n^k)$ per qualche $k \in \mathbb{N}$
  - Il ciclo viene eseguito per $O(|E_G|) = O(m^2) = O(n^2)$ iterazioni, dove ognuna di quest'ultime richiede $O(1)$

- Di conseguenza il costo computazione di V risulta essere:

  $$t(n) = O(n) + O(n^k) + O(n^2) \cdot O(1) = O(n^k)$$

  implicando che V sia un verificatore polinomiale e quindi che $3COL \in \mathsf{NP}$

□

4.4 Riducibilità in tempo polinomiale

Definizione 4.15: Riducibilità in tempo polinomiale

Dati due linguaggi A e B, diciamo che A è **riducibile in tempo polinomiale a B tramite mappatura**, indicato come $A \leq_m^P B$, se esiste una funzione calcolabile $f : \Sigma^* \to \Sigma^*$, detta **riduzione in tempo polinomiale da A a B**, tale che:

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

e f è **calcolabile in tempo polinomiale**

Teorema 4.8: Decidibilità polinomiale tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m^P B$, si ha che:

$$B \in \mathsf{P} \implies A \in \mathsf{P}$$

(*dimostrazione analoga al teorema* 3.23)

Teorema 4.9: Verificabilità polinomiale tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m^P B$, si ha che:

$$B \in \mathsf{NP} \implies A \in \mathsf{NP}$$

(*dimostrazione analoga al teorema* 3.23)

Definizione 4.16: Problema della soddisfacibilità delle 3CNF

Definiamo il linguaggio del **problema della soddisfacibilità delle 3CNF** come:

$$3SAT = \{\langle \phi \rangle \mid \phi \text{ è una 3CNF soddisfacibile}\}$$

dove una *3CNF* è una formula logica formata da un *and logico* tra n clausole, ciascuna formata da un *or logico* tra 3 *letterali*, ossia una variabile negata o non

- La seguente formula è una 3CNF:

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

- Inoltre, tale formula è soddisfacibile dal seguente assegnamento:

$$x_1 = 1, \quad x_2 = 0, \quad x_3 = 1, \quad x_4 = 1, \quad x_5 = 1, \quad x_6 = 0$$

Teorema 4.10: $3SAT$ poly-riducibile a $CLIQUE$

Dati i linguaggi $3SAT$ e $CLIQUE$, si ha che $3SAT \leq_m^P CLIQUE$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  F = "Data la stringa $\langle \phi \rangle$ in input:

  1. Costruisci un grafo G definito come:
     i. Conta il numero k di clausole or in ϕ
     ii. Per ogni clausola, crea un nodo x_i per ogni letterale della clausola, creando dei doppioni se il letterale compare più volte
     iii. Organizza i nodi di G in k triple (una per clausola *or*), dove tre nodi appartengono alla stessa tripla se e solo se i loro letterali compaiono all'interno di una stessa clausola or
     iv. Crea un arco tra ogni coppia di nodi fatta eccezione di nodi appartenenti alla stessa tripla e nodi rappresentanti letterali opposti tra loro (es: x_i e $\overline{x_i}$)
  2. Restituisci in output la stringa $\langle G, k \rangle$"

  (*si consiglia di guardare l'immagine riportata in seguito prima di proseguire con la dimostrazione*)

- Supponiamo che $\langle \phi \rangle \in 3SAT$. Poiché ϕ è una 3CNF, affinché essa sia soddisfacibile ne segue che ogni clausola sia soddisfacibile, dunque che almeno uno dei letterali di ognuna di tali clausole sia vero
- Sia quindi $C = \{x_1, \ldots, x_k\}$ l'insieme dei nodi tali che $\forall i \in [1, k] \quad x_i$ è il nodo corrispondente al letterale vero della i-esima tripla
- Poiché tali nodi appartengono tutti a triple diverse e poiché $\nexists x_i, x_j \in C$ tali che $x_i = \overline{x_j}$ in quanto non possono essere entrambi veri, ne segue che essi siano tutti due a due adiacenti, implicando che C sia una k-clique
- Di conseguenza, abbiamo che:

$$\langle \phi \rangle \in 3SAT \implies f(\langle \phi \rangle) = \langle G, k \rangle \in CLIQUE$$

- Supponiamo ora che $f(\phi) = \langle G, k \rangle \in CLIQUE$, implicando che esista una k-clique al suo interno. Dunque, per costruzione di G, non esistono nodi all'interno della clique i cui letterali rappresentati appartengono alla stessa tripla
- A questo punto, poiché tali letterali non interferiscono tra loro trovandosi in triple diverse, esiste un un assegnamento che soddisfi ogni clausola. In particolare, tale assegnamento risulta sempre possibile in quanto all'interno della clique non possano appartenere sia x_i che $\overline{x_i}$

- Di conseguenza, abbiamo che:

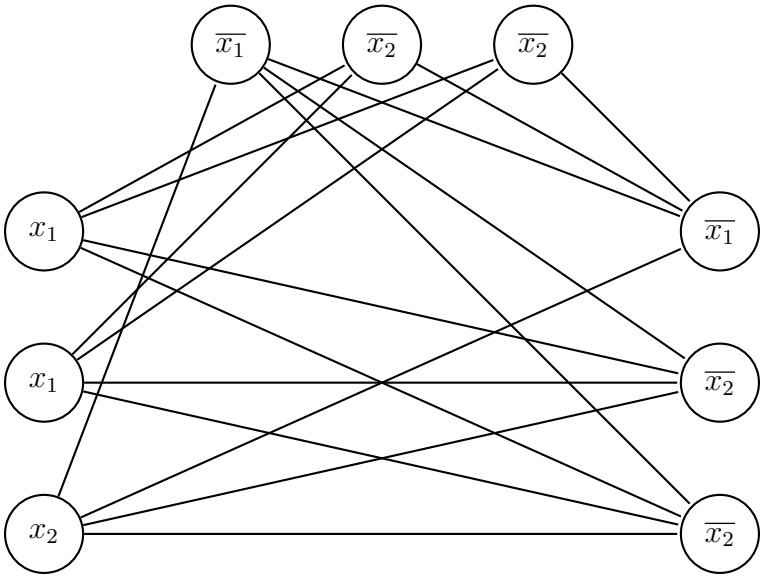
$$f(\langle\phi\rangle) = \langle G, k\rangle \in CLIQUE \implies \langle\phi\rangle \in 3SAT$$

  implicando quindi che:

$$\langle\phi\rangle \in 3SAT \iff f(\langle\phi\rangle) = \langle G, k\rangle \in CLIQUE$$

- Inoltre, poiché F svolge solo operazioni eseguibili in tempo polinomiale, concludiamo che $3SAT \leq_m^P CLIQUE$

□



Grafo generato dalla funzione f della dimostrazione a partire dalla formula $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

Corollario 4.2: $3SAT$ polinomialmente verificabile

Il linguaggio $3SAT$ è **polinomialmente verificabile**, ossia $3SAT \in \mathsf{NP}$

(*segue dai teoremi 4.5, 4.9 e 4.16*)

Teorema 4.11: Riducibilità polinomiale transitiva

Dati tre linguaggi $A, B, C \subseteq \Sigma^*$, si ha che:

$$A \leq_m^P B, B \leq_m^P C \implies A \leq_m^P C$$

In altre parole, la relazione $\leq_m^P$ è **transitiva**

Dimostrazione.

- Siano $f, g : \Sigma^* \to \Sigma^*$ le due funzioni calcolabili polinomialmente per cui rispettivamente si ha che $A \leq_m^P B$ e $B \leq_m^P C$

- Sia quindi H la TM definita come:

  H = "Data la stringa w in input:

  1. Calcola $f(w)$. Sia k l'output di tale calcolo.
  2. Calcola $g(k)$ e restituisci l'output"

- Risulta evidente che H sia la TM che calcola la funzione $g \circ f$ e che:

$$w \in A \iff f(w) \in B \iff g(f(w)) = (g \circ f)(w) \in C$$

- Inoltre, poiché f e g sono calcolabili polinomialmente, ne segue automaticamente che H svolga solo operazioni in tempo polinomiale, concludendo che $A \leq_m^P C$

□

Teorema 4.12: Riducibilità polinomiale complementare

Dati due linguaggi A e B, si ha che:

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

(*dimostrazione analoga al teorema* 3.25)

4.5 Classe NP-Complete

Definizione 4.17: Classe NP-Hard

Definiamo la **classe dei linguaggi NP-Difficili** come:

$$\text{NP-Hard} = \{B \subset \Sigma^* \mid \forall A \in \text{NP} \;\; A \leq_m^P B\}$$

dove $B \neq \varnothing, \Sigma^*$.

Nota: non è detto che un linguaggio NP-Hard sia decidibile o riconoscibile

Definizione 4.18: Classe NP-Complete

Definiamo la **classe dei linguaggi NP-Completi** come:

$$\text{NP-Complete} = \text{NP} \cap \text{NP-Hard}$$

Teorema 4.13: Completezza transitiva

Dati due linguaggi B e C tali che $B \leq_m^P C$ e $C \in \mathsf{NP}$, si ha che:

$$B \in \mathsf{NP\text{-}Complete} \implies C \in \mathsf{NP\text{-}Complete}$$

Dimostrazione.

- Poiché la relazione $\leq_m^P$ è transitiva e poiché $B \in \mathsf{NP\text{-}Complete}$, ne segue automaticamente che:
$$\forall A \in \mathsf{NP} \;\; A \leq_m^P B \leq_m^P C$$
implicando dunque che $C \in \mathsf{NP} \cap \mathsf{NP\text{-}Hard} = \mathsf{NP\text{-}Complete}$

□

Teorema 4.14: Linguaggio NP-completo in P

Dato un linguaggio $B \in \mathsf{NP\text{-}Complete}$, si ha che:

$$B \in \mathsf{P} \iff \mathsf{P} = \mathsf{NP}$$

Dimostrazione.

Prima implicazione.

- Per definizione stessa di NP-Complete, si ha che $\forall A \in \mathsf{NP} \;\; A \leq_m^P B$
- Di conseguenza, poiché $B \in \mathsf{P}$, ne segue automaticamente che:
$$\forall A \in \mathsf{NP} \;\; B \in \mathsf{P} \implies A \in P$$
implicando dunque che $\mathsf{NP} \subseteq \mathsf{P}$
- Inoltre, poiché sappiamo già che $\mathsf{P} \subseteq \mathsf{NP}$, concludiamo che $\mathsf{P} = \mathsf{NP}$

Seconda implicazione.

- Per definizione stessa di NP-Complete, si ha che $B \in \mathsf{NP}$
- Di conseguenza, se $\mathsf{P} = \mathsf{NP}$, ne segue automaticamente che $B \in \mathsf{P}$

□

4.5.1 Teorema di Cook-Levin

Definizione 4.19: Problema della soddisfacibilità

Definiamo il linguaggio del **problema della soddisfacibilità** come:

$$SAT = \{\langle\phi\rangle \mid \phi \text{ è una formula soddisfacibile}\}$$

Teorema 4.15: Teorema di Cook-Levin

Dato il linguaggio SAT, si ha che $SAT \in$ NP-Complete

Dimostrazione.

- Sia V la TM definita come:

  $V =$ "Data la stringa $\langle\langle\phi\rangle, c\rangle$ in input:

  1. Interpreta $c = \langle x_1, \ldots, x_n\rangle$ come un assegnamento di variabili
  2. Valuta l'assegnamento $\phi(x_1, \ldots, x_n)$
  3. Se vero, *accetta*. Altrimenti, *rifiuta*"

- Per costruzione stessa di V, si ha che:

  $$\langle\phi\rangle \in SAT \iff \text{Esiste un assegnamento che soddisfa } \phi \iff$$

  $$\exists c \in \Sigma^*, \langle\langle\phi\rangle, c\rangle \in V$$

  implicando che V sia un verificatore di SAT. Inoltre, il suo tempo di esecuzione risulta intuitivamente polinomiale, implicando che $SAT \in$ NP

- Dato un linguaggio $A \in$ NP, sia $N = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}})$ la NTM tale che $L(N) = A$ in tempo $O(n^k)$ per qualche $k \in \mathbb{N}$

- Definiamo un *tableau di computazione* dell'esecuzione di N su w come una tabella $n^k \times n^k$ dove la i-esima riga contiene la i-esima configurazione assunta da N durante la computazione, partendo dalla configurazione iniziale fino a quella finale.

  (*si consiglia di guardare l'immagine riportata in seguito prima di proseguire con la dimostrazione*)

  Nota: Essendo N non-deterministica, ne segue che essa sia dotata di un tableau per ogni ramificazione dell'albero di computazione.

- In particolare, notiamo che:

  $$w \in L(N) \iff \exists \text{ tableau accettante per } N(w)$$

  dove un tableau è detto accettante se al suo interno esiste una riga con una cella contenente il simbolo q_{accept}

- Sia $C = Q \cup \Gamma \cup \{\#\}$ e sia T un tableau della computazione $N(w)$
- Per ogni $i, j \in [1, n^k]$ e per ogni $s \in C$, definiamo la variabile logica $x_{i,j,s}$ tale che:

$$x_{i,j,s} = \text{True} \iff T[i,j] = s$$

- Consideriamo quindi la seguente formula:

$$\phi_{\text{cell}} = \bigwedge_{0 \leq i,j \leq n^k} \left[\left(\bigvee_{s \in C} x_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \overline{x_{i,j,s} \wedge x_{i,j,t}} \right) \right]$$

  Notiamo che tale formula descriva esattamente la struttura delle celle di un tableau valido per la computazione $N(w)$:

  - La prima parte della formula, ossia l'insieme di or, afferma in logica che per ogni cella $T[i,j]$ del tableau esista almeno un simbolo associato
  - La seconda parte della formula, ossia l'insieme di and, afferma in logica che per ogni cella $T[i,j]$ del tableau non vi possano essere due o più simboli associati

- Successivamente, consideriamo la seguente formula:

$$\begin{aligned} \phi_{\text{start}} = \quad & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \ldots \wedge x_{1,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \ldots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#} \end{aligned}$$

  la quale descrive la prima configurazione iniziale della computazione $N(w)$

- A seguire, consideriamo la successiva formula:

$$\phi_{\text{accept}} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_{\text{accept}}}$$

  la quale risulta vera se e solo se esiste almeno una cella contenente il simbolo q_{accept}

- A questo punto, definiamo come *finestra* un insieme 2×3 di celle di un tableau. Una finestra è detta *lecita* se le celle al suo interno rispettano la definizione della funzione di transizione δ, ossia se esse possono apparire all'interno del tableau solo se la configurazione $(i+1)$-esima segue correttamente dalla configurazione i-esima.

  (*si consiglia di guardare gli esempi riportati in seguito prima di proseguire con la dimostrazione*)

- Consideriamo quindi la seguente formula:

$$\phi_{\text{move}} = \bigwedge_{0 \leq i,j \leq n^k} \left(\bigvee_{\substack{(a_1,\ldots,a_6) \\ \text{è finestra lecita}}} x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6} \right)$$

- Tale formula descrive, tramite l'insieme di or, tutte le possibili combinazioni di finestre lecite imposte da δ partendo dalla configurazione iniziale, implicando che tale formula codifichi a tutti gli effetti ogni sequenza di configurazioni eseguibili dai rami di $N(w)$
- Infine, definiamo la formula:

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$$

- Per costruzione di ϕ, ogni suo assegnamento descriverà un tableau, il quale potrà essere invalido (dunque non descrivente alcun ramo di $N(w)$), valido ma non accettante o valido ma accettante
- In particolare, notiamo che le formule $\phi_{\text{cell}}, \phi_{\text{start}}$ e ϕ_{move} siano soddisfacibili assegnando le variabili al loro interno in modo che esse descrivano un qualsiasi tableau valido per $N(w)$, mentre l'intera formula ϕ risulti soddisfacibile solo se le variabili al suo interno sono assegnate in modo che esse descrivano un tableau accettante per $N(w)$
- Di conseguenza, otteniamo che:

$$\exists \text{ tableau accettante per } N(w) \iff \phi \text{ soddisfacibile}$$

- Sia quindi $f : \Sigma^* \to \Sigma^*$ la funzione calcolabile dalla seguente TM M definita come:

  M = "Data la stringa w in input:

  1. Costruisci la formula ϕ che descrive la computazione $N(w)$
  2. Restituisci in output la stringa $\langle \phi \rangle$"
- A questo punto, risulta evidente che:

$$w \in A = L(N) \iff \exists \text{ tableau accettante per } N(w)$$

$$\iff \phi \text{ soddisfacibile} \iff f(w) = \langle \phi \rangle \in SAT$$

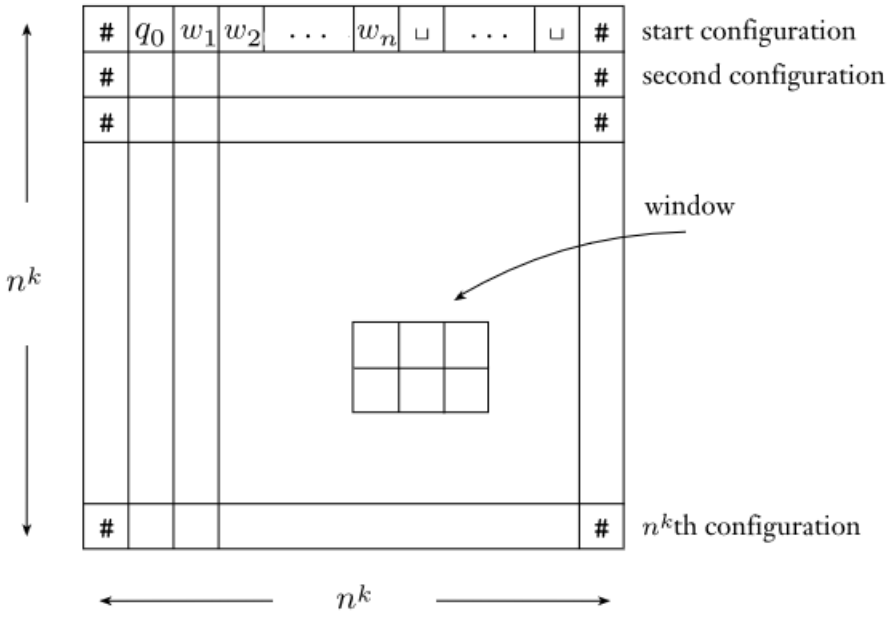
  implicando che:

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

- Inoltre, poiché:
  - La formula ϕ_{start} è costruibile in $O(n^k)$
  - Le formule ϕ_{accept}, ϕ_{cell} e ϕ_{move} sono costruibili in $O(n^{2k})$
- Di conseguenza, concludiamo che:

$$A \in \text{NP} \implies A \leq_m^P SAT$$

□



Rappresentazione grafica di un tableau di computazione

a	q_1	b
q_2	a	c

a	q_1	b
a	a	q_2

a	a	q_1
a	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

b	b	b
c	b	b

Esempi di finestre lecite nel caso in cui $\delta(q_1, a) = \{(q_1, b, R)\}$ *e* $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

a	b	a
a	a	a

a	q_1	b
q_2	a	a

b	q_1	b
q_2	b	q_2

Esempi di finestre illecite nel caso in cui $\delta(q_1, a) = \{(q_1, b, R)\}$ *e* $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

Teorema 4.16: *SAT* **riducibile in tempo polinomiale a** $3SAT$

Dati i linguaggi SAT e $3SAT$, si ha che $SAT \leq_m^P 3SAT$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla TM F definita come:

  $F =$ "Data la stringa $\langle\phi\rangle$ in input:

  1. Interpreta ϕ come una CNF
  2. Costruisci una formula ϕ' definita come:
     i. Ripeti le seguenti istruzioni per ogni clausola *or* C_i in ϕ:
     ii. Se $C_i = \alpha$, dove α è un letterale, allora crea in ϕ' le clausole $(\alpha \vee x \vee y), (\alpha \vee \overline{x} \vee y), (\alpha \vee x \vee \overline{y})$ e $(\alpha \vee \overline{x} \vee \overline{y})$, dove x e y sono due nuove variabili
     iii. Se $C_i = (\alpha \vee \beta)$, dove α e β sono dei letterali, allora crea in ϕ' le clausole $(\alpha \vee \beta \vee x)$ e $(\alpha \vee \beta \vee \overline{x})$, dove x è una nuova variabile
     iv. Se $C_i = (\alpha \vee \beta \vee \gamma)$, dove α, β e γ sono letterali, copia tale clausola in ϕ'
     v. Se $C_i = (\ell_1 \vee \ldots \vee \ell_k)$, dove $k > 3$ e $\ell_1, \ldots, \ell_k$ sono dei letterali, crea in ϕ' le clausole $(\ell_1 \vee \ell_2 \vee x_1), (\overline{x_1} \vee \ell_3 \vee x_2), (\overline{x_2} \vee \ell_4 \vee x_3), \ldots, (\overline{x_{k-3}} \vee \ell_{k-2} \vee x_{k-2}), (\overline{x_{k-2}} \vee \ell_{k-1} \vee \ell_k)$, dove $x_1, \ldots, x_{k-2}$ sono delle nuove variabili
  3. Restituisci in output la stringa $\langle\phi'\rangle$"

- Consideriamo quindi una formula ϕ espressa in CNF. Data una clausola *or* C_i in ϕ, si ha che:

  – Se $C_i = \alpha$, tramite le regole di semplificazione notiamo che:

  $$(\alpha \vee x \vee y) \wedge (\alpha \vee \overline{x} \vee y) \wedge (\alpha \vee x \vee \overline{y}) \wedge (\alpha \vee \overline{x} \vee \overline{y}) \equiv$$

  $$(\alpha \vee y) \wedge (\alpha \vee y) \wedge (\alpha \vee \overline{y}) \wedge (\alpha \vee \overline{y}) \equiv \alpha$$

  dunque si ha che:

  $$\alpha \text{ soddisfacibile in } \phi \iff$$

  $$(\alpha \vee x \vee y), (\alpha \vee \overline{x} \vee y), (\alpha \vee x \vee \overline{y}), (\alpha \vee \overline{x} \vee \overline{y}) \text{ soddisfacibili in } \phi'$$

  – Se $C_i = (\alpha \vee \beta)$, tramite le regole di semplificazione notiamo che:

  $$(\alpha \vee \beta \vee x) \wedge (\alpha \vee \beta \vee \overline{x}) \equiv (\alpha \vee \beta)$$

  dunque si ha che:

  $$(\alpha \vee \beta) \text{ soddisfacibile in } \phi \iff (\alpha \vee \beta \vee x), (\alpha \vee \beta \vee \overline{x}) \text{ soddisfacibili in } \phi'$$

  - Se $C_i = (\alpha \vee \beta \vee \gamma)$ ne segue automaticamente che:

$$(\alpha \vee \beta \vee \gamma) \text{ soddisfacibile in } \phi \iff (\alpha \vee \beta \vee \gamma) \text{ soddisfacibile in } \phi'$$

- Consideriamo quindi il caso meno ovvio in cui $C_i = (\ell_1 \vee \ldots \vee \ell_k)$ e la seguente sottoformula di ϕ' ad essa corrispondente:

$$(\ell_1 \vee \ell_2 \vee x_1) \wedge (\overline{x_1} \vee \ell_3 \vee x_2) \wedge (\overline{x_2} \vee \ell_4 \vee x_3) \wedge \ldots \wedge (\overline{x_{k-3}} \vee \ell_{k-2} \vee x_{k-2}) \wedge (\overline{x_{k-2}} \vee \ell_{k-1} \vee \ell_k)$$

- Supponiamo quindi che ϕ sia soddisfacibile. In tal caso, ogni C_i deve esserlo, implicando che almeno un letterale tra $\ell_1, \ldots, \ell_k$ sia vero
- Di conseguenza, si ha che:
  - Se $\ell_1 = \text{True}$ o $\ell_2 = \text{True}$, è sufficiente porre $x_1 = \ldots = x_{k-2} = \text{False}$ affinché l'intera sottoformula di ϕ' sia soddisfacibile
  - Se $\ell_{k-1} = \text{True}$ o $\ell_k = \text{True}$, è sufficiente porre $x_1 = \ldots = x_{k-2} = \text{True}$ affinché l'intera sottoformula di ϕ' sia soddisfacibile
  - Se $\ell_j = \text{True}$, dove $j \in [2, k-2]$, è sufficiente porre $x_1 = \ldots = x_{j-2} = \text{True}$ e $x_{j-1} = \ldots = x_{k-2} = \text{False}$ affinché l'intera sottoformula di ϕ' sia soddisfacibile
- Supponiamo invece che ϕ non sia soddisfacibile. In tal caso, almeno un C_i deve esserlo, implicando che nessun letterale tra $\ell_1, \ldots, \ell_k$ sia vero
- Di conseguenza, notiamo che anche ponendo $x_1 = \ldots = x_{i-1} = x_{i-1} = \ldots = x_{k-2} = \text{True}$, esisterà sempre una variabile x_i che renderà tale sottoformula soddisfacibile se e solo se $\overline{x_i} = x_i = \text{True}$, il che risulta impossibile
- Di conseguenza, otteniamo che:

$$\phi \text{ soddisfacibile} \iff \phi' \text{ soddisfacibile}$$

- Inoltre, poiché ϕ' è una 3CNF, otteniamo che:

$$\langle \phi \rangle \in SAT \iff f(\langle \phi \rangle) = \langle \phi' \rangle \in 3SAT$$

  e poiché F svolge solo operazioni in tempo polinomiale, concludiamo che $SAT \leq_m^P 3SAT$

□

Corollario 4.3: NP-Completezza di $3SAT$

Dato il linguaggio $3SAT$, si ha che $3SAT \in$ NP-Complete

Dimostrazione.

- Per il Teorema di Cook-Levin, sappiamo che $SAT \in$ NP-Complete
- Inoltre, per il teorema precedente, sappiamo che $SAT \leq_m^P 3SAT$
- Di conseguenza, per transitività si ha che:

$$\forall A \in \text{NP} \quad A \leq_m^P SAT \leq_m^P 3SAT$$

- Infine, poiché $3SAT \in$ NP, concludiamo che $3SAT \in$ NP-Complete

□

Corollario 4.4: NP-Completezza di $CLIQUE$

Dato il linguaggio $CLIQUE$, si ha che $CLIQUE \in$ NP-Complete

Dimostrazione.

- Tramite il corollario precedente, sappiamo che $3SAT \in$ NP-Complete
- Inoltre, abbiamo già mostrato che $3SAT \leq_m^P CLIQUE$
- Di conseguenza, per transitività si ha che:

$$\forall A \in \text{NP} \quad A \leq_m^P 3SAT \leq_m^P CLIQUE$$

- Infine, poiché $CLIQUE \in$ NP, concludiamo che $CLIQUE \in$ NP-Complete

□

4.6 Classi coP, coNP e coEXP

Definizione 4.20: Classi coP, coNP e coEXP

Definiamo le **classi dei linguaggi coP, coNP e coP** come:

$$\mathsf{coP} = \{A \in \mathsf{DEC} \mid \overline{A} \in \mathsf{P}\}$$

$$\mathsf{coNP} = \{A \in \mathsf{DEC} \mid \overline{A} \in \mathsf{NP}\}$$

$$\mathsf{coEXP} = \{A \in \mathsf{DEC} \mid \overline{A} \in \mathsf{EXP}\}$$

Teorema 4.17: Chiusura del complemento di P

Date le classi P e coP, si ha che:

$$\mathsf{P} = \mathsf{coP}$$

In altre parole, la classe P è **chiusa nel complemento**

Dimostrazione.

Prima implicazione.

- Dato $A \in \mathsf{P}$, sia D il decisore polinomiale tale che $L(D) = A$
- Sia $\overline{D}$ la TM definita come:

  $\overline{D}$ = "Data la stringa w in input:

  1. Esegui il programma di D su input w
  2. Se l'esecuzione accetta, *rifiuta*. Altrimenti, *accetta*"
- Per costruzione stessa di $\overline{D}$, abbiamo che:

  $$x \in L(\overline{D}) \iff x \notin L(D) = A$$

  implicando che $L(\overline{D}) = \overline{A}$
- Inoltre, poiché D è un decisore polinomiale, ne segue che anche $\overline{D}$ lo sia, concludendo che $\overline{A} = L(\overline{D}) \in \mathsf{P}$ e dunque che $A \in \mathsf{coP}$

Seconda implicazione.

- Analogo alla prima implicazione

□

Teorema 4.18: Chiusura del complemento di EXP

Date le classi EXP e coEXP, si ha che:

$$\mathsf{EXP} = \mathsf{coEXP}$$

In altre parole, la classe EXP è **chiusa nel complemento**

(*dimostrazione analoga al teorema precedente*)

Corollario 4.5: P ⊆ coNP ⊆ EXP

Date le classi P, coNP e EXP, si ha che:

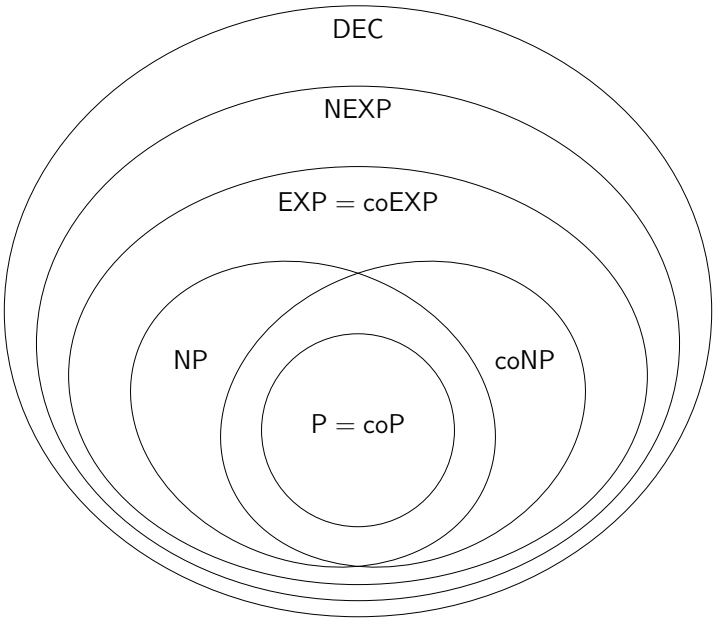
$$\mathsf{P} \subseteq \mathsf{coNP} \subseteq \mathsf{EXP}$$

Dimostrazione.

- Dato $A \in \mathsf{P}$, per i teoremi della Gerarchia temporale dei linguaggi, Chiusura del complemento di P e Chiusura del complemento di EXPSPACE, abbiamo che:
  - $A \in \mathsf{P} \implies \overline{A} \in \mathsf{P} \subseteq \mathsf{NP} \implies A \in \mathsf{coNP}$
  - $A \in \mathsf{coNP} \implies \overline{A} \in \mathsf{NP} \subseteq \mathsf{EXP} \implies A \in \mathsf{EXP}$

  implicando che $\mathsf{P} \subseteq \mathsf{coNP} \subseteq \mathsf{EXP}$

□



Gerarchia delle classi dei linguaggi decidibili studiate fino ad ora

Teorema 4.19

Date le classi P, NP e coNP, si ha che:

$$\mathsf{P} = \mathsf{NP} \implies \mathsf{P} = \mathsf{coNP}$$

Dimostrazione.

- Per il corollario precedente, sappiamo che $\mathsf{P} \subseteq \mathsf{coNP}$
- Dato $A \in \mathsf{coNP}$, per l'ipotesi e per il teorema Chiusura del complemento di P, si ha che:
$$A \in \mathsf{coNP} \implies \overline{A} \in \mathsf{NP} = \mathsf{P} \implies A \in \mathsf{P}$$
concludendo quindi che $\mathsf{P} = \mathsf{coNP}$

□

Corollario 4.6

Date le classi P, NP e coNP, si ha che:

$$\mathsf{NP} \neq \mathsf{coNP} \implies \mathsf{P} \neq \mathsf{NP}$$

Dimostrazione.

- Tramite il teorema precedente, si ha che:
$$\mathsf{P} = \mathsf{NP} \implies \mathsf{P} = \mathsf{coNP} \implies \mathsf{coNP} = \mathsf{NP}$$
- Dunque, per contronominale, otteniamo che:
$$\mathsf{NP} \neq \mathsf{coNP} \implies \mathsf{P} \neq \mathsf{NP}$$

□

Definizione 4.21: Classe coNP-Complete

Definiamo la **classe dei linguaggi coNP-Completi** come:

$$\mathsf{coNP\text{-}Complete} = \{L \in \mathsf{DEC} \mid L \text{ è coNP-Completo}\}$$

dove un linguaggio B è detto *coNP-Completo* se:

- $B \in \mathsf{coNP}$
- $\forall A \in \mathsf{coNP} \;\; A \leq_m^P B$

Teorema 4.20: Completezza del complemento in NP-Complete

Dato un linguaggio B si ha che:

$$B \in \text{NP-Complete} \iff \overline{B} \in \text{coNP-Complete}$$

Dimostrazione.

- Tramite la Riducibilità logaritmica complementare, si ha che:

$$B \in \text{NP-Complete} \iff B \in \text{NP}, \forall A \in \text{NP} \;\; A \leq_m^P B \iff$$
$$B \in \text{NP}, \forall A \in \text{NP} \;\; \overline{A} \leq_m^P \overline{B} \iff \overline{B} \in \text{coNP}, \forall \overline{A} \in \text{coNP} \;\; \overline{A} \leq_m^P \overline{B}$$
$$\iff \overline{B} \in \text{coNP-Complete}$$

□

Teorema 4.21: Ugualianza tra NP e coNP

Dato un linguaggio $B \in$ NP-Complete, si ha che:

$$B \in \text{coNP} \iff \text{NP} = \text{coNP}$$

Dimostrazione.

Prima implicazione.

- Poiché $B \in \text{coNP} \iff \overline{B} \in \text{NP}$ e poiché $B \in$ NP-Complete, si ha che:

$$A \in \text{NP} \implies A \leq_m^P B \iff \overline{A} \leq_m^P \overline{B} \implies \overline{A} \in \text{NP} \iff A \in \text{coNP}$$

  implicando che coNP $\subseteq$ NP

- Viceversa, si ha che:

$$A \in \text{coNP} \implies \overline{A} \in \text{NP} \implies \overline{A} \leq_m^P B \implies A \in \text{NP}$$

  implicando che NP $\subseteq$ coNP

Seconda implicazione.

- Se NP = coNP, allora $B \in$ NP-Complete $\subseteq$ NP = coNP

□

Definizione 4.22: Problema dell'insoddisfacibilità

Definiamo il linguaggio del **problema dell'insoddisfacibilità** come:

$$UNSAT = \{\langle \phi \rangle \mid \phi \text{ è una formula insoddisfacibile}\}$$

Attenzione: precisiamo che $UNSAT \neq \overline{SAT}$ poiché $\overline{SAT}$ contiene anche stringhe che non sono formule valide

Teorema 4.22: coNP-Completezza di $UNSAT$

Dato il linguaggio $UNSAT$, si ha che $UNSAT \in$ coNP-Complete.

Dimostrazione.

- Per il Teorema di Cook-Levin sappiamo che SAT sia NP-Completo. Di conseguenza, per la Completezza del complemento in NP-Complete sappiamo che $\overline{SAT}$ sia coNP-Completo.
- Sia quindi $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F:

  $F =$ "Data la stringa x in input:

  1. Verifica se $x = \langle \phi \rangle$ per qualche formula ϕ.
  2. Se vero, restituisci x. Se falso, restituisci la stringa $\langle a \wedge \neg a \rangle$ "
- Notiamo che se $x \notin \overline{SAT}$ allora $x \in SAT$ e dunque sicuramente $x = \langle \phi \rangle$ per qualche formula soddisfacibile ϕ, implicando che $f(x) = \langle \phi \rangle \notin UNSAT$. Invece, se $x \in \overline{SAT}$ allora abbiamo due opzioni:
  - Se $x \in \overline{SAT}$ in quanto x è non è una formula valida allora $f(x) = \langle a \wedge \neg a \rangle \in UNSAT$ in quanto la formula $a \wedge \neg a$ è insoddisfacibile
  - Se $x \in \overline{SAT}$ in quanto x è una formula valida allora $f(x) = \langle \phi \rangle \in UNSAT$ in quanto la formula ϕ è insoddisfacibile
- Ciò conclude che $x \in \overline{SAT} \iff f(x) \in UNSAT$. Inoltre, la macchina F richiede tempo polinomiale, concludendo che $\overline{SAT} \leq_m^P UNSAT$. Infine, poiché $\overline{SAT}$ è coNP-Completo ed è riducibile a $UNSAT$, concludiamo che anche $UNSAT$ sia coNP-Completo.

□

4.7 Complessità spaziale

Definizione 4.23: Complessità spaziale di una TM e di una NTM

Sia D un decisore. Definiamo come **complessità spaziale** (o *spazio di esecuzione*) di D la funzione $f : \mathbb{N} \to \mathbb{N}$ tale che $f(n)$ sia il massimo numero di celle utilizzate da D per processare una stringa di lunghezza n.

Analogamente, nel caso delle NTM, $f(n)$ corrisponde al massimo numero di celle utilizzate da ogni ramo dell'esecuzione per processare una stringa di lunghezza n

Nota: le celle della stringa in input non vengono considerate all'interno del costo spaziale

Teorema 4.23: Rapporto di spazio tra TM multinastro e TM

Sia $s(n)$ una funzione tale che $s(n) \geq n$. Per ogni TM multinastro M con tempo $s(n)$ esiste una TM M' tale che $L(M) = L(M')$ avente tempo $O(s(n))$

(dimostrazione omessa)

Osservazione 4.3: TM con input tape e work tape

Poiché le celle della stringa in input non vengono considerate nel costo spaziale, nell'ambito della complessità spaziale consideriamo l'**uso esclusivo** di TM dotate di **due nastri di base**:

- **Input tape**: un nastro **read-only** contenente la stringa in input
- **Work tape**: un nastro **read-write** su cui la TM lavora

Di conseguenza, all'interno del costo spaziale consideriamo **solo le celle utilizzate sul work tape**

Esempio:

- Consideriamo il seguente linguaggio $L = \{0^n 1^n \mid n \in \mathbb{N}\}$
- Sia M il decisore definito come:

  M = "Data la stringa w in input:

  1. Controlla se w è nella forma 0^*1^*
  2. Sul nastro di lavoro, inizializza un contatore a 0 e torna all'inizio dell'input
  3. Leggendo l'input, incrementa in contatore finché viene letto il simbolo 0, per poi decrementarlo finché viene letto il simbolo 1
  4. Se a fine lettura il contatore è uguale a 0, *accetta*. Altrimenti, *rifiuta*"

- Innanzitutto, risulta evidente che $L = L(M)$ e che M sia un decisore
- Per quanto riguarda il costo spaziale di M, le uniche celle utilizzate dalla TM sono relative al contatore presente sul nastro di lavoro, il quale può facilmente essere realizzato in spazio $O(\log n)$ (si pensi ad un contatore binario, dunque costo $\log_2 n$, o un contatore decimale, dunque costo $\log_{10} n$)
- Di conseguenza, il suo spazio computazionale è $O(\log n)$

Osservazione 4.4: Differenza tra spazio e tempo

Poiché le celle del nastro possono essere riutilizzate, il costo spaziale tende ad essere "più flessibile" rispetto al costo temporale

Esempio:

- Consideriamo il seguente linguaggio $L = \{ww^R \mid w \in \Sigma^*\}$
- Sia M il decisore definito come:

  M = "Data la stringa w in input:

  1. Calcola $|w| = n$
  2. Ripeti la seguente istruzione per $i = 1, \dots, n$:

     3 Verifica se $w_i = w_{n+1-i}$. Se falso, *rifiuta*
  4. *Accetta.*"
- Innanzitutto, risulta evidente che $L = L(M)$ e che M sia un decisore
- Analizziamo quindi il costo spaziale di M:
  - Il calcolo di $|w| = n$ può essere realizzato tramite un contatore, dunque ha costo $O(\log n)$
  - Essendo i un contatore, esso ha costo $O(\log n)$
  - Per calcolare $n + 1 - i$ ad ogni iterazione, utilizziamo la seguente procedura:
    1. Copio n su un nastro secondario e i su un nastro terziario, richiedente $2 \cdot O(\log n)$ spazio
    2. Incremento di 1 il nastro contenente n, riutilizzando le celle precedenti
    3. Decremento il nastro contenente $n + 1$ e il nastro contenente i finché non si ha che $i = 0$, riutilizzando le celle precedenti
- Di conseguenza, il costo spaziale di M risulta essere:

$$s(n) = O(\log n) + 2 \cdot O(\log n) = O(\log n)$$

Definizione 4.24: Classi DSPACE e NSPACE

Data la funzione $s : \mathbb{N} \to \mathbb{R}^+$, definiamo come **classe dei linguaggi decidibili in spazio** $O(s(n))$ il seguente insieme:

$$\mathsf{DSPACE}(s(n)) = \{L \in \mathsf{DEC} \mid L \text{ decidibile da una } \mathsf{TM} \text{ in spazio } O(s(n))\}$$

Analogamente, definiamo come **classe dei linguaggi decidibili non-deterministicamente in spazio** $O(s(n))$ il seguente insieme:

$$\mathsf{NSPACE}(s(n)) = \{L \in \mathsf{DEC} \mid L \text{ decidibile da una } \mathsf{NTM} \text{ in spazio } O(s(n))\}$$

Teorema 4.24: $PATH$ decidibile in $O(\log^2 n)$

Il linguaggio $PATH$ è **decidibile in** $O(\log^2 n)$, ossia $PATH \in \mathsf{DSPACE}(\log^2 n)$

Dimostrazione.

- Sia $G = (V, E)$ un grafo
- Consideriamo la seguente procedura ricorsiva definita su G:

  `Path`?(x, y, k):

  1. Se $k = 0$, verifica se $(x, y) \in E$ o se $x = y$. Se vero, *accetta*. Altrimenti, *rifiuta*
  2. Se $k > 0$, ripeti la seguente istruzione per ogni nodo v in V:
     3. Se `Path`?$(x, v, k-1)$ accetta e `Path`?$(v, y, k-1)$ accetta, allora la procedura *accetta*. Altrimenti, essa *rifiuta*
- Per costruzione stessa di `Path`?, si ha che:

  $$\texttt{Path?}(x, y, k) \text{ accetta} \iff \text{Esiste cammino } x \to y \text{ di massimo } 2^k \text{ nodi}$$

- Sia quindi M la TM definita come:

  M = "Data la stringa $\langle G, s, t\rangle$ in input;

  1. Esegui la procedura `Path`?$(s, t, \lceil \log n \rceil)$.
  2. Se la procedura accetta, *accetta*. Altrimenti, *rifiuta*"
- Notiamo quindi che:

  $$\langle G, s, t\rangle \in L(M) \iff \texttt{Path?}(s, t, \lceil \log n \rceil) \text{ accetta} \iff$$

  $$\text{Esiste cammino } s \to t \text{ con massimo } 2^{\lceil \log n \rceil} \text{ nodi} \iff$$

  $$\text{Esiste cammino } s \to t \text{ con massimo } n \text{ nodi} \iff \langle G, s, t\rangle \in PATH$$

  implicando che $L(M) = PATH$

- Le due chiamate della procedura necessitano di memorizzare i due nodi in input, richiedendo $2 \cdot O(\log n) = O(\log n)$ spazio. Tale spazio è riutilizzabile ad ogni iterazione del ciclo e ad ogni ricorsione. Inoltre, l'altezza dell'albero di ricorsione corrisponde a $\lceil \log n \rceil = O(\log n)$
- Di conseguenza, concludiamo che il costo spaziale di M sia:

$$s(n) = O(\log n) \cdot O(\log n) = O(\log^2 n)$$

concludendo che $PATH = L(M) \in \mathsf{DSPACE}(\log^2 n)$

□

4.7.1 Rapporto tra spazio e tempo

Teorema 4.25: Tempo limitante lo spazio

Data una funzione $f(n)$ tale che $f(n) \geq n$, si ha che:

$$\mathsf{DTIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n))$$

$$\mathsf{NTIME}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$$

Dimostrazione.

- Sia D una TM tale che $L(D) \in \mathsf{DTIME}(f(n))$
- Poiché il tempo è $O(f(n))$, il numero massimo di passi che la TM può effettuare sono $O(f(n))$
- Di conseguenza, anche se la TM utilizzasse una nuova cella ad ogni passo, il massimo numero di celle utilizzabili sarebbe $O(f(n))$, implicando che $L(D) \in \mathsf{DSPACE}(f(n))$
- Per argomento analogo, concludiamo anche che $\mathsf{NTIME}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$

□

Teorema 4.26: Spazio limitante il tempo

Data una funzione $f(n)$ tale che $f(n) \geq \log n$, si ha che:

$$\mathsf{DSPACE}(f(n)) \subseteq \mathsf{DTIME}(2^{O(f(n))})$$

Dimostrazione.

- Sia M una TM tale che $L(M) \in \mathsf{DSPACE}(f(n))$
- Per comodità, assumiamo che M sia dotata di un solo work tape

- Indichiamo le configurazioni di M con la seguente notazione:

  $$\text{WORK}q_i\text{TAPE};\ j$$

  dove:

  - WORKTAPE è il contenuto attuale del work tape
  - q_i è lo stato attuale e il simbolo alla sua destra è il simbolo su cui si trova la testina del work tape
  - j indica che la testina dell'input tape si trova attualmente sulla j-esima cella
- Sia $t(n)$ il tempo massimo di computazione di M
- Poiché M è un decisore, le configurazioni di ogni computazione non possono ripetersi, poiché altrimenti essa andrebbe in loop infinito. Di conseguenza, $t(n)$ coinciderà con il numero massimo di configurazioni possibili
- Per via della notazione utilizzata per descrivere le configurazioni di M, tale numero massimo di configurazioni corrisponde al numero di disposizioni possibili per una configurazione, ossia:

  $$t(n) = |\Gamma|^{f(n)} \cdot |Q| \cdot n$$

- Inoltre, poiché per ipotesi si ha che:

  $$f(n) \geq \log n \implies 2^{f(n)} \geq n$$

  ne segue che:

  $$t(n) = |\Gamma|^{f(n)} \cdot |Q| \cdot n \leq |\Gamma|^{f(n)} \cdot |Q| \cdot 2^{f(n)} = 2^{O(f(n))}$$

  concludendo che $L(M) \in \mathsf{DTIME}(2^{O(f(n))})$

□

4.7.2 Teorema di Savitch

Teorema 4.27: Teorema di Savitch

Data una funzione $f(n) \geq \log n$, si ha che:

$$\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DSPACE}(f^2(n))$$

Dimostrazione.

- Sia N una NTM tale che $L(N) \in \mathsf{NSPACE}(f(n))$
- Per comodità, assumiamo che in N vi sia un solo stato accettante q_{accept} (abbiamo visto in capitoli precedenti come ogni NTM possa essere modificata senza alterare il suo linguaggio)

- Siano inoltre q_{start} e δ rispettivamente lo stato iniziale di N e la sua funzione di transizione
- Consideriamo la notazione WORKq_iTAPE; j per indicare le configurazioni di N (come nella dimostrazione dello Spazio limitante il tempo). Di conseguenza, il numero massimo di configurazioni durante una qualsiasi computazione di N corrisponde a $2^{O(f(n))}$
- Consideriamo quindi il grafo $G_{N,w} = (V, E)$ definito come:
  - Ad ogni nodo di V corrisponde una configurazione possibile di N durante la computazione di w
  - Per ogni nodo $v_i, v_j \in V$, esiste un arco se e solo se la computazione può passare dalla configurazione c_i alla configurazione c_j tramite δ
- Per costruzione di $G_{N,w}$, si ha che:

$$w \in L(N) \iff \text{Esiste cammino } c_{\text{start}} \to c_{\text{accept}} \text{ in } G_{N,w}$$

- Consideriamo quindi la procedura $PathG_{N,w}?$, una versione modificata della procedura `Path`? mostrata nella dimostrazione di $PATH$ decidibile in $O(\log^2 n)$, dove il grafo $G_{N,w}$ viene costruito durante la ricorsione stessa.
- Sia quindi M la TM definita come:

  M = "Data la stringa w in input:

  1. Esegui la procedura $PathG_{N,w}?(c_{\text{start}}, c_{\text{accept}}, \lceil \log m \rceil)$
  2. Se la procedura accetta, *accetta*. Altrimenti, *rifiuta*"
- Per costruzione stessa di M, si ha che:

$$w \in L(M) \iff PathG_{N,w}?(c_{\text{start}}, c_{\text{accept}}, \lceil \log m \rceil) \text{ accetta} \iff$$

$$\text{Esiste cammino } c_{\text{start}} \to c_{\text{accept}} \text{ in } G_{N,w} \iff w \in L(N)$$

  implicando che $L(M) = L(N)$
- Notiamo quindi che la costruzione parziale del grafo richieda solo qualche "variabile" di appoggio, mantenendo il costo della chiamata $PathG_{N,w}?(c_{\text{start}}, c_{\text{accept}}, \lceil \log m \rceil)$ inalterato, ossia pari a $O(\log^2 m)$, dove m è la dimensione del grafo in input
- Di conseguenza, poiché la dimensione di $G_{N,w}$ risulta essere $m = 2^{O(f(n))}$, il costo spaziale di M risulta essere:

$$s(n) = O(\log^2 m) = O(\log^2(2^{O(f(n))})) = O(f^2(n))$$

  concludendo che $L(N) = L(M) \in \mathsf{DSPACE}(f^2(n))$

□

4.8 Classe PSPACE

Definizione 4.25: Classi PSPACE e NPSPACE

Definiamo la **classe dei linguaggi decidibili in spazio polinomiale** come:

$$\mathsf{PSPACE} = \bigcup_{k=1}^{\infty} \mathsf{DSPACE}(n^k)$$

Analogamente, definiamo la **classe dei linguaggi decidibili non-deterministicamente in spazio polinomiale** come:

$$\mathsf{NPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(n^k)$$

Definizione 4.26: Classi EXPSPACE e NEXPSPACE

Definiamo la **classe dei linguaggi decidibili in spazio esponenziale** come:

$$\mathsf{EXPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{DSPACE}(2^{n^k})$$

Analogamente, definiamo la **classe dei linguaggi decidibili non-deterministicamente in spazio esponenziale** come:

$$\mathsf{NEXPSPACE} = \bigcup_{k=1}^{\infty} \mathsf{NSPACE}(2^{n^k})$$

Teorema 4.28: Rapporto tra spazio e tempo

Dato un alfabeto Σ, si ha che:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE}$$

Dimostrazione.

- Tramite il teorema Tempo limitante lo spazio, si ha che:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \qquad \mathsf{EXP} \subseteq \mathsf{NEXP} \subseteq \mathsf{EXPSPACE}$$

- Tramite il teorema Spazio limitante il tempo, invece, si ha che:

$$\mathsf{PSPACE} \subseteq \mathsf{EXP}$$

□

Teorema 4.29: PSPACE = NPSPACE

Date le classi PSPACE e NPSPACE:

$$\mathsf{PSPACE} = \mathsf{NPSPACE}$$

Dimostrazione.

- Per definizione stessa, si ha che $\mathsf{PSPACE} \subseteq \mathsf{NPSPACE}$
- Inoltre, per il Teorema di Savitch, si ha che:

  $$L \in \mathsf{NPSPACE} \implies \exists k \in \mathbb{N} \mid L \in \mathsf{NSPACE}(n^k) \subseteq \mathsf{DSPACE}(n^{2k}) \implies L \in \mathsf{PSPACE}$$

  implicando che $\mathsf{NPSPACE} \subseteq \mathsf{PSPACE}$

□

Teorema 4.30: EXPSPACE = NEXPSPACE

Date le classi EXPSPACE e NEXPSPACE:

$$\mathsf{EXPSPACE} = \mathsf{NEXPSPACE}$$

(dimostrazione analoga al teorema precedente)

Definizione 4.27: Classi coPSPACE e coEXPSPACE

Definiamo le **classi dei linguaggi coPSPACE e coEXPSPACE** come:

$$\mathsf{coPSPACE} = \{A \in \mathsf{DEC} \mid \overline{A} \in \mathsf{PSPACE}\}$$

$$\mathsf{coEXPSPACE} = \{A \in \mathsf{DEC} \mid \overline{A} \in \mathsf{EXPSPACE}\}$$

Teorema 4.31: Chiusura del complemento di PSPACE

Date le classi PSPACE e coPSPACE, si ha che:

$$\mathsf{PSPACE} = \mathsf{coPSPACE}$$

In altre parole, la classe PSPACE è **chiusa nel complemento**

(dimostrazione analoga al teorema 4.17)

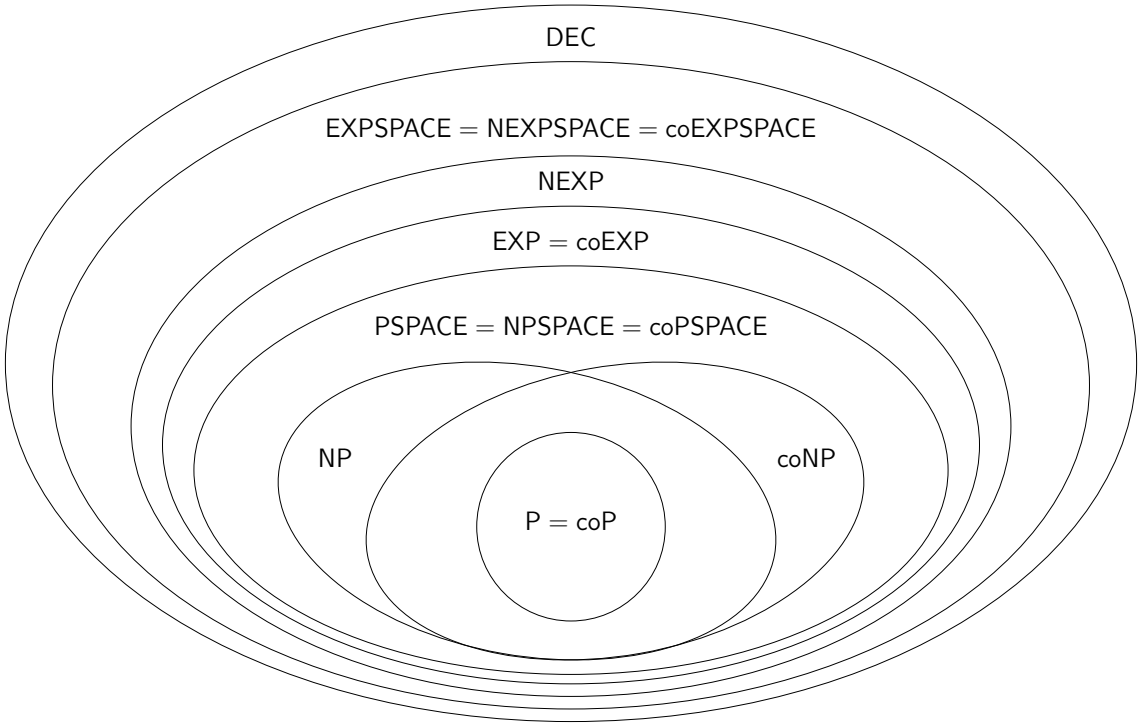
Teorema 4.32: Chiusura del complemento di EXPSPACE

Date le classi EXPSPACE e coEXPSPACE, si ha che:

$$\text{EXPSPACE} = \text{coEXPSPACE}$$

In altre parole, la classe EXPSPACE è **chiusa nel complemento**

(dimostrazione analoga al teorema 4.17)



Gerarchia delle classi dei linguaggi decidibili studiate fino ad ora

4.9 Classe L

Definizione 4.28: Classi L e NL

Definiamo la **classe dei linguaggi decidibili in spazio logaritmico** come:

$$\mathsf{L} = \mathsf{DSPACE}(\log n)$$

Analogamente, definiamo la **classe dei linguaggi decidibili non-deterministicamente in spazio logaritmico** come:

$$\mathsf{NL} = \mathsf{NSPACE}(\log n)$$

Corollario 4.7: Rapporto tra L e P

Date le classi L, NL e P, si ha che:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P}$$

(*segue dal teorema 4.26*)

Osservazione 4.5: NL $\stackrel{?}{=}$ L

Tramite il Teorema di Savitch, abbiamo che:

$$\mathsf{NL} = \mathsf{NSPACE}(\log n) \subseteq \mathsf{DSPACE}(\log^2 n)$$

ma non sappiamo se $\mathsf{NL} = \mathsf{L}$. Spesso si dice che $\mathsf{NL} \subseteq \mathsf{L}^2$.

A differenza delle classi NP e NEXP, nel caso della classe NL la definizione alternativa tramite verificatore ha bisogno di un piccolo cambiamento al fine di poter rispettare i vincoli di spazio:

- Vincolando il certificato ad una lunghezza logaritmica otteniamo un sottoinsieme troppo ristretto di problemi in quanto il certificato risulta troppo piccolo, dunque l'insieme dei problemi verificabili in spazio logaritmi con un certificato logaritmico non coinciderebbe con NL
- Mantenendo il certificato di lunghezza polinomiale, esso può essere utilizzato come *memoria aggiuntiva*: potremmo leggere più volte la stessa informazione evitando di doverla memorizzare nel work tape, risparmiando una notevole quantità di spazio

In altre parole, nel primo caso otteniamo dei verificatori troppo deboli, mentre nel secondo sono troppo potenti. Per ottenere una via di mezzo tra le due cose, possiamo tuttavia imporre che il certificato sia **read-once**: il verificatore possiede un nastro aggiuntivo contenente il certificato le cui celle possono essere lette una sola volta (dunque la testina

non può spostarsi a sinistra). In tal modo, preserviamo la possibilità che il certificato sia di lunghezza polinomiale ed impediamo che esso possa essere utilizzato come memoria aggiuntiva.

Teorema 4.33: Classe NL (definizione alternativa)

Data la classe NL, si ha che:

$$\mathsf{NL} = \{A \in \mathsf{DEC} \mid \exists \text{ V verif. con cert. read-once in spazio logaritmico per } A\}$$

In altre parole, possiamo definire NL anche come la **classe dei linguaggi verificabili in spazio logaritmico con un certificato read-once**

Dimostrazione.

Prima implicazione.

- Dato $A \in \mathsf{NL}$, sia N la NTM tale che $A = L(N)$
- Durante la sua computazione, N effettuerà una serie di scelte, creando così il suo albero di computazione
- Sia V la TM definita come:

  V = "Data la stringa $\langle w, c\rangle$ in input:

  1. Interpreta c come una serie di scelte da effettuare
  2. Simula N su input w eseguendo le scelte dettate da c leggendole in sequenza
  3. Se la simulazione accetta, V *accetta*. Altrimenti, *rifiuta*"
- Per costruzione stessa di V, si ha che:

  $$w \in L = L(N) \iff \text{Esiste un ramo di } N \text{ che accetta } w$$
  $$\iff \exists c \in \Sigma^*, \langle w, c\rangle \in L(V)$$

  dunque V risulta essere un verificatore con certificato read-once di $L = L(N)$
- Inoltre, poiché $A = L(N) \in \mathsf{NL}$, lo spazio massimo utilizzato da ogni ramo dell'albero di computazione risulta essere $O(\log n)$ e poiché la lettura del certificato non influenza lo spazio, concludiamo che V sia un verificatore in spazio logaritmico

Seconda implicazione.

- Dato un linguaggio A, supponiamo che esista un verificatore con certificato read-once V che verifica A in spazio logaritmico, implicando che:

  $$\exists k \in \mathbb{N} \mid V \text{ verifica } L \text{ in spazio } O(\log n)$$

- Sia N la NTM definita come:

  N = "Data la stringa $\langle w\rangle$ in input:

1. Scegli non deterministicamente una stringa c
2. Simula V su input $\langle w, c\rangle$
3. Se la simulazione accetta, B *accetta*. Altrimenti, *rifiuta*"

- Per costruzione stessa di N, si ha che

$$w \in A \iff \exists c \in \Sigma^*, \langle w, c\rangle \in V \iff$$

$$\text{Esiste un ramo di } N \text{ che accetta } w \iff x \in A$$

  implicando che $A = L(N)$

- Notiamo che le prime due operazioni di N possano essere svolte in contemporanea, generando il prossimo simbolo della stringa ad ogni passo di computazione. Inoltre, poiché V è un verificatore read-once in spazio logaritmico, non è sufficiente mantenere memorizzate le celle precedentemente generate. Dunque, è sufficiente memorizzare un solo simbolo per volta, richiedendo spazio $O(\log n)$. Dunque, concludiamo che $A = L(N) \in \mathsf{NL}$

□

Definizione 4.29: Classe coL

Definiamo la **classe dei linguaggi coL** come:

$$\mathsf{coL} = \{A \in \mathsf{DEC} \mid \overline{A} \in \mathsf{L}\}$$

Teorema 4.34: Chiusura del complemento di L

Date le classi L e coL, si ha che:

$$\mathsf{L} = \mathsf{coL}$$

In altre parole, la classe L è **chiusa nel complemento**

(dimostrazione analoga al teorema 4.17)

4.10 Riduzione in spazio logaritmico

Definizione 4.30: Riducibilità in spazio logaritmico

Dati due linguaggi A e B, diciamo che A è **riducibile in spazio logaritmico a B tramite mappatura**, indicato come $A \leq_m^L B$, se esiste una funzione calcolabile $f : \Sigma^* \to \Sigma^*$, detta **riduzione in spazio logaritmico da A a B**, tale che:

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

e f è **calcolabile in spazio logaritmico**

Teorema 4.35: Decidibilità logaritmica tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m^L B$, si ha che:

$$B \in \mathsf{L} \implies A \in \mathsf{L}$$

Dimostrazione.

- Dato $B \in \mathsf{DEC}$, sia D_B il decisore tale che $L(D_B) = B$
- Sia D_A la TM definita come:

  $D_A =$ "Data la stringa w in input:

  1. Calcola $f(w)$
  2. Esegui il programma di D_B con input $f(w)$.
  3. Se l'esecuzione accetta, D *accetta*. Altrimenti, D *rifiuta*"
- Per costruzione stessa di D_A, si ha che:

  $$w \in L(D_A) \iff f(w) \in L(D_B) = B \iff w \in A$$

  implicando che $L(D_A) = A$.
- Tuttavia, notiamo che, poiché $f(w)$ potrebbe richiedere spazio $O(\log n)$, è necessario eseguire le prime due operazioni in contemporanea, calcolando il prossimo simbolo di $f(w)$ ad ogni passo di computazione di D_B. Inoltre, poiché D_B computa in spazio logaritmico, concludiamo che $A = L(D_A) \in \mathsf{L}$

□

Teorema 4.36: Verificabilità logaritmica tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m^L B$, si ha che:

$$B \in \mathsf{NL} \implies A \in \mathsf{NL}$$

(*dimostrazione analoga al teorema precedente*)

Teorema 4.37: Riducibilità logaritmica transitiva

Dati tre linguaggi $A, B, C \subseteq \Sigma^*$, si ha che:

$$A \leq_m^L B, B \leq_m^L C \implies A \leq_m^L C$$

In altre parole, la relazione $\leq_m^L$ è **transitiva**

Dimostrazione.

- Siano $f, g : \Sigma^* \to \Sigma^*$ le due funzioni calcolabili polinomialmente per cui rispettivamente si ha che $A \leq_m^L B$ e $B \leq_m^L C$
- Sia quindi H la TM definita come:

  H = "Data la stringa w in input:

  1. Calcola $f(w)$. Sia k l'output di tale calcolo.
  2. Calcola $g(k)$ e restituisci l'output"
- Risulta evidente che H sia la TM che calcola la funzione $g \circ f$ e che:

$$w \in A \iff f(w) \in B \iff g(f(w)) = (g \circ f)(w) \in C$$

- Eseguendo contemporaneamente le due operazioni, ossia il prossimo simbolo di $f(w)$ assieme al prossimo simbolo di $g(k)$, è sufficiente utilizzare $2 \cdot O(\log n)$ spazio, concludendo che $A \leq_m^L C$

□

Teorema 4.38: Riducibilità logaritmica complementare

Dati due linguaggi A e B, si ha che:

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

(*dimostrazione analoga al teorema 3.25*)

4.11 Classe NL-Complete

Definizione 4.31: Classe NL-Complete

Definiamo la **classe dei linguaggi NL-Completi** come:

$$\text{NL-Complete} = \{L \in \text{DEC} \mid L \text{ è NL-Completo}\}$$

dove un linguaggio B è detto *NL-Completo* se:

- $B \in \text{NL}$
- $\forall A \in \text{NL} \;\; A \leq_m^L B$

Teorema 4.39: NL-Completezza di $PATH$

Dato il linguaggio $PATH$, si ha che $PATH \in \text{NL-Complete}$

Dimostrazione.

- Sia N la NTM definita come:

  N = "Data la stringa $\langle G, s, t\rangle$ in input, dove $G = (V, E)$ è un grafo:

  1. Se $s = t$, *accetta*
  2. Calcola $|V| = m$
  3. Inizializza $currNode = s$ su un nastro secondario
  4. Ripeti le seguenti istruzioni per $i = 1, \ldots, m$
     5. Seleziona non deterministicamente un nodo $u \in V$
     6. Se $(currNode, u) \notin E$, *rifiuta*
     7. Se $u = t$ *accetta*, altrimenti poni $currNode = u$
  8. *Rifiuta*"

- Per costruzione stessa di N, si ha che:

  $$\langle G, s, t\rangle \in L(N) \iff \text{Esiste cammino } s \to t \text{ in } G \iff \langle G, s, t\rangle \in PATH$$

  implicando che $L(N) = PATH$

- Inoltre, poiché i cammini vengono letti nodo per nodo senza tenere traccia dei precedenti, lo spazio richiesto risulta essere $O(\log n)$, concludendo che $PATH = L(N) \in \text{NL}$

- Sia quindi $A \in \text{NL}$ e sia N' una NTM tale che $A = L(N)$ in spazio $O(\log n)$

- Sia inoltre $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  F = "Data la stringa w in input:

  1. Costruisci il grafo $G_{N,w}$ delle configurazioni della computazione di N su input w come nella dimostrazione del teorema 4.24
  2. Restituisci in output la stringa $\langle G_{N,w}, c_{\text{start}}, c_{\text{accept}} \rangle$

- Notiamo che:

$$w \in L(N') \iff \text{Esiste cammino } c_{\text{start}} \to c_{\text{accept}} \text{ in } G_{N,w}$$
$$\iff f(w) = \langle G_{N,w}, c_{\text{start}}, c_{\text{accept}} \rangle \in PATH$$

- Inoltre, poiché la computazione contemporanea di $f(w)$ assieme a $\langle G_{N,w}, c_{\text{start}}, c_{\text{accept}} \rangle \in PATH$ richiede spazio $O(\log n)$, concludiamo che $A \leq_m^L PATH$

□

4.11.1 Teorema di Immerman-Szelepcsényi

Lemma 4.1: Ugualianza tra NL e coNL

Dato un linguaggio $B \in$ NL-Complete, si ha che:

$$B \in \text{coNL} \iff \text{NL} = \text{coNL}$$

(*dimostrazione analoga al teorema* 4.21)

Teorema 4.40: Teorema di Immerman-Szelepcsényi

Date le classi NL e coNL, si ha che:

$$\text{NL} = \text{coNL}$$

In altre parole, la classe NL è **chiusa nel complemento**

Dimostrazione.

- Poiché $PATH \in$ NL-Complete, tramite il lemma precedente abbiamo che:

$$\overline{PATH} \in \text{NL} \iff PATH \in \text{coNL} \iff \text{NL} = \text{coNL}$$

- Vogliamo quindi dimostrare che $\overline{PATH}$ sia in NL. L'idea dietro la seguente dimostrazione risiede nella possibilità di abusare la lunghezza polinomiale del certificato: non possiamo leggere più volte le stesse celle, ma possiamo inserire nel certificato un numero adeguato di copie delle celle precedenti.

- Sia $m = |V(G)|$ e siano $R_0, \ldots, R_{m-1}$ gli insiemi di vertici raggiungibili da s con il massimo di $i \in [0, m-1]$ passi:

$$R_i = \{v \in V(G) \mid s \to v \text{ con il massimo di } i \text{ archi}\}$$

- Supponiamo anche che i vertici di G siano identificati da un numero in $[0, m-1]$. Questa sarà un'assunzione chiave: la procedura deve essere valida per *qualche* certificato, quindi possiamo sempre supporre che il nostro certificato segua la nostra idea. Tuttavia, ciò implica anche che dobbiamo preservare la nostra assunzione rifiutando i certificati che non si basano su questa idea.

- L'idea alla base della seguente dimostrazione è che la lunghezza polinomiale del certificato può essere abusata: non possiamo leggere le stesse celle più volte, ma possiamo inserire un numero adeguato di copie delle stesse informazioni. In particolare, il nostro certificato di input sarà una sequenza di sotto-certificati definiti ricorsivamente. Il nostro verificatore sarà basato su quattro sotto-procedure. Ogni chiamata di queste procedure avrà un certificato di input ricorsivo. Per dare un'idea di ciò che stiamo cercando di ottenere, inizieremo definendo il verificatore finale:

  V = "Data la stringa $\langle \langle G, s, t \rangle, c \rangle$ come input:

  1. Controlla se $s \neq t$. Se falso, *rifiuta*.
  2. Interpreta $c = \langle (\ell_1, c_1), \ldots, (\ell_k, c_k), c_t \rangle$
  3. Imposta prev $= 1$ ▷ ℓ_0 è uguale a 1
  4. Imposta $i = 1$ e ripeti mentre $i \leq k$:
     5. Imposta curr $= \ell_i$
     6. Esegui la sottoprocedura `certify_` $|\mathtt{R_i}|$ `_with_` $|\mathtt{R_{i-1}}|$ $(i, \text{curr}, c_i, \text{prev},)$. Se la sottoprocedura rifiuta, anche V *rifiuta*.
     7. Imposta prev = curr.
  8. Controlla se $k = m - 1$. Se falso, *rifiuta*.
  9. Esegui la sottoprocedura `certify_v_not_in_R`$_\mathtt{i}$`_with_` $|\mathtt{R_i}|$ (m, t, c_t, curr). Se la sottoprocedura accetta, anche V *accetta*. Altrimenti V *rifiuta*.

- Qui, ogni input ℓ_i è un valore intero verificato dal sottocertificato c_i: la procedura `certify_` $|\mathtt{R_i}|$ `_with_` $|\mathtt{R_{i-1}}|$ verifica che $|R_i| = \ell_i$ utilizzando il valore verificato in precedenza ℓ_{i-1}. In altre parole, i valori $\ell_0, \ldots, \ell_m$ vengono utilizzati per verificarsi induttivamente a vicenda. In particolare, notiamo che $|R_0| = \ell_0 = 1$ è sempre vero poiché $R_0 = \{s\}$. Dopo aver verificato che $|R_m| = \ell_m$, la procedura `certify_v_not_in_R`$_\mathtt{i}$`_with_` $|\mathtt{R_i}|$ verificherà se $t \notin R_n$ è vero. Quando ciò è vero, possiamo concludere che non esiste alcun percorso da s a t. Notiamo inoltre che, grazie ai valori memorizzati, ogni cella del certificato di input viene letta una volta.

- Per ora, ci concentreremo sull'ultima procedura, la quale utilizza un sotto-certificato per testimoniare l'appartenenza di $|R_i|$ nodi diversi nell'insieme R_i. Se tutti questi nodi sono diversi dal nodo di input v allora possiamo concludere che $v \notin R_i$.

  `certify_v_not_in_R`$_i$`_with_` $|\texttt{R}_i|$ $(i, v, \langle c_{u_1}, \ldots, c_{u_k} \rangle, \ell_i)$:

  1. Imposta $i' = i$
  2. Imposta $v' = v$
  3. Imposta prev $= -\infty$.
  4. Imposta $h = 1$ e ripeti mentre $h \leq k$:
     3. Imposta curr $= u_h$
     4. Controlla se curr $\neq v'$. Se falso, *rifiuta*.
     5. Controlla se prev $<$ curr. Se falso, *rifiuta*.
     6. Controlla se next $\in R_i$ usando la sottoprocedura `certify_v_in_R`$_i$ $(c_{\text{curr}}, i', \text{curr})$. Se la sottoprocedura rifiuta, anche questa procedura *rifiuta*.
     7. Imposta prev = curr.
     8. Incrementa h di 1.
  6. Controlla se $h = \ell_i$. Se è vero, *accetta*, altrimenti *rifiuta*.

- Ogni sotto-certificato c_{u_k} viene utilizzato per testimoniare che $u_k \in R_i$. Il controllo prev $<$ curr assicura che il certificato con cui stiamo lavorando rispetti la nostra ipotesi. In particolare, questo ci consente di utilizzare la transitività per garantire che tutti i nodi testimoniati siano diversi tra loro.

- Per ogni vertice $v \in V(G)$, verificare che $v \in R_i$ sia vero è molto semplice: basta un singolo certificato $\langle u_1, \ldots u_k \rangle$ che descrive un percorso da s a v con al massimo i archi. Notiamo che, poiché $u_1, \ldots, u_k$ non sono sotto-certificati, questa procedura è una foglia della procedura di verifica ricorsiva.

  `certify_v_in_R`$_i$ $(v, \langle u_1, \ldots, u_k \rangle, i)$:

  1. Imposta $v' = v$.
  2. Imposta prev $= u_1$.
  3. Controlla se prev $\neq v'$. Se falso, *rifiuta*.
  4. Imposta $h = 2$ e ripeti mentre $h \leq k$:
     5. Imposta curr $= u_h$
     6. Controlla se curr $\neq v'$. Se falso, *rifiuta*.
     7. Controlla se prev $<$ curr. Se falso, *rifiuta*.
     8. Controlla se $(\text{prev}, \text{curr}) \in E(G)$. Se falso, *rifiuta*
     9. Imposta prev = curr.

10. Incrementa h di 1.

11. Controlla se $h \leq i$. Se è vero, *accetta*, altrimenti *rifiuta*.

- Passiamo ora alla procedura $\texttt{certify\_}|\texttt{R}_\texttt{i}|\texttt{\_with\_}|\texttt{R}_{\texttt{i}-\texttt{1}}|$. Questa procedura controllerà che ogni nodo testimoniato dal suo sotto-certificato sia all'interno di R_i, dove il numero previsto di nodi in ℓ_i. In particolare, per ogni nodo u_h dovremo eseguire due sotto-procedure. Per prima cosa, controlliamo se $u_h \in R_i$. Se falso, controlliamo anche se $u_h \notin R_i$ usando $\mathrm{R_{i-1}}$. Questo secondo controllo è necessario: la prima procedura potrebbe anche rifiutare quando il certificato ha il formato previsto sbagliato.

  $\texttt{certifica\_}|\texttt{R}_\texttt{i}|\texttt{\_con\_}|\texttt{R}_{\texttt{i}-\texttt{1}}|\left(i, \ell_{i-1}, \left\langle c_{u_1}, \ldots, c_{u_{\ell_k}} \right\rangle, \ell_i, \right)$:

  1. Imposta $i' = i$
  2. Imposta $\ell'_{i-1} = \ell_{i-1}$
  3. Imposta $v' = v$
  4. Imposta prev $= -\infty$.
  5. Imposta $h = 1$ e ripeti mentre $h \leq k$:
     3. Imposta curr $= u_h$
     4. Controlla se prev $<$ curr. Se falso, *rifiuta*.
     5. Controlla se next $\in R_i$ usando la sottoprocedura $\texttt{certify\_v\_in\_R}_\texttt{i}\,(c_{\mathrm{curr}}, i', \mathrm{curr})$.
     6. Se la prima sottoprocedura rifiuta, controlla se next $\notin R_i$ usando la procedura $\texttt{certify\_v\_not\_in\_R}_\texttt{i}\texttt{\_with\_}|\texttt{R}_{\texttt{i}-\texttt{1}}|\,\left(i', u_h, c_{\mathrm{curr}}, \ell'_{i-1}\right)$. Se la seconda sottoprocedura rifiuta, anche questa procedura *rifiuta*.
     7. Imposta prev = curr.
     8. Incrementa h di 1.
  6. Controlla se $h = \ell_i$. Se vero, *accetta*, altrimenti *rifiuta*.

- La sottoprocedura $\texttt{certify\_v\_not\_in\_R}_\texttt{i}\texttt{\_with\_}|\texttt{R}_{\texttt{i}-\texttt{1}}|$ qui utilizzata è molto simile alla sottoprocedura $\texttt{certify\_v\_not\_in\_R}_\texttt{i}\texttt{\_with\_}|\texttt{R}_\texttt{i}|$: dopo aver verificato che il sottocertificato di input testimonia i nodi di R_{i-1}, questa nuova sottoprocedura controllerà se i nodi adiacenti a R_{i-1} contengono v.

  $\texttt{certifica\_v\_non\_in\_R}_\texttt{i}\texttt{\_con\_}|\texttt{R}_{\texttt{i}-\texttt{1}}|\,(i, v, \langle c_{u_1}, \ldots, c_{u_k}\rangle, \ell_{i-1})$:

  1. Imposta $i' = i$
  2. Imposta $v' = v$
  3. Imposta prev $= -\infty$.
  4. Imposta $h = 1$ e ripeti mentre $h \leq k$:
     3. Imposta curr $= u_h$

4. Controlla se curr $\neq v'$. Se falso, *rifiuta*.
5. Controlla se prev $<$ curr. Se falso, *rifiuta*.
6. Controlla se curr $\in R_i$ usando la sottoprocedura `certify_v_in_R_i` $(c_{\text{curr}}, i', \text{curr})$. Se la sottoprocedura rifiuta, anche questa procedura *rifiuta*.
7. Controlla se v' è un vicino di curr. Se vero, *rifiuta*.
8. Imposta prev = curr.
9. Incrementa h di 1.

6. Controlla se $h = \ell_{i-1}$. Se è vero, *accetta*, altrimenti *rifiuta*.

- Dopo aver definito ogni procedura in modalità read-once, possiamo calcolare la complessità spaziale di tali procedure. Il numero di celle richieste dalla sottoprocedura `certify_v_in_R_i` è chiaramente $O(\log n)$ since abbiamo 5 variabili che memorizzano valori interi (ricorda che lo spazio può essere riutilizzato). Analogamente, per le altre procedure è sufficiente uno spazio di $O(\log n)$ poiché possiamo riutilizzare le celle precedenti a ogni chiamata ricorsiva. Quindi, la macchina V verifica correttamente $\overline{\text{PATH}}$ con uno spazio di $O(\log n)$.
- Infine, poiché stiamo lavorando con molti certificati ricorsivi, dobbiamo assicurarci che il certificato di input abbia una dimensione polinomiale. Il certificato di input utilizzato dalla procedura `certify_v_in_R_i` è chiaramente di dimensione $O(m \log n)$. Pertanto, otteniamo anche che i certificati di input delle procedure `certify_v_not_in_R_i_with_` $|\texttt{R}_\texttt{i}|$ e `certify_v_not_ in_R_i_with_` $|\texttt{R}_{\texttt{i}-1}|$ hanno dimensione $O(m^2 \log n)$ poiché abbiamo al massimo m sotto-certificati utilizzati da `certify_v_in_R_i` . Allo stesso modo, `certify_` $|\texttt{R}_\texttt{i}|$ `_with_` $|\texttt{R}_{\texttt{i}-1}|$ ha un certificato di dimensione $O(m^3 \log n)$, mentre il certificato finale utilizzato da V ha dimensione $O(m^4 \log n)$. Poiché $m \leq n$, otteniamo che il certificato ricorsivo totale ha dimensione $O(n^4 \log n)$.

□

4.12 Teoremi di gerarchia

4.12.1 Teorema di gerarchia di spazio

Definizione 4.32: Funzione spazio-costruibile

Definiamo una funzione $f : \mathbb{N} \to \mathbb{N}$, dove $f(n) \geq \log n$, come **spazio-costruibile** se la seguente funzione:

$$g : \Sigma^* \to \Sigma^* : 1^n \mapsto f(n)_2$$

è calcolabile in spazio $O(f(n))$, dove $f(n)_2$ è la codifica binaria di $f(n)$

Nota: se f è una funzione a valori non interi (es: $n \log n$ o $\sqrt{n}$), il risultato viene arrotondato all'intero precedente

Esempi:

(a)
- Consideriamo la funzione $f(n) = n^2$
- Sia $g : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM G definita come:

  $G =$ "Data la stringa 1^n in input:

  1. Calcola $|1^n| = n$ tramite un contatore binario
  2. Calcola $k = n \cdot n$ tramite una moltiplicazione binaria
  3. Restituisci in output la stringa k"
- Notiamo facilmente che $g(1^n) = f(n)_2$ e che lo spazio richiesto da G sia $O(\log(n^2)) = O(\log n)$, implicando che esso sia anche $O(n^2)$, concludendo che f sia spazio-costruibile

(b)
- Consideriamo la funzione $f(n) = n \log_2 n$
- Sia $g : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM G definita come:

  $G =$ "Data la stringa 1^n in input:

  1. Calcola $|1^n| = n$ tramite un contatore binario
  2. Inizializza $i = 1$ e $j = 0$
  3. Ripeti la seguente istruzione finché $i < n$:
     4. Incrementa j
     5. Moltiplica i per 2
  4. Moltiplica j per n
  5. Restituisci in output la stringa j"
- Notiamo facilmente che $g(1^n) = f(n)_2$ e che lo spazio richiesto da G sia $O(\log(n) + \log(n2^j))$, dove $j < n$, implicando che esso sia anche $O(n \log n) = O(f(n))$, concludendo che f sia spazio-costruibile

Teorema 4.41: Teorema di gerarchia di spazio

Data una funzione spazio-costruibile $f : \mathbb{N} \to \mathbb{N}$, esiste un linguaggio L decidibile da una TM in spazio $O(f(n))$ ma non in spazio $o(f(n))$

Dimostrazione.

- Sia D la TM definita come:

  $D =$ "Data la stringa w in input:

  1. Calcola $\langle w \rangle = n$
  2. Calcola $f(n)$ in modo spazio-costruibile
  3. Marca la cella numero $f(n)$ e ritorna all'inizio del nastro. Se tale cella viene superata durante le istruzioni successive, D *rifiuta*
  4. Interpreta $w = \langle M \rangle 10^*$, dove M è una TM. Se l'interpretazione fallisce, D *rifiuta*
  5. Simula M su input w contando il numero di passi effettuati nella simulazione. Se tale numero di passi supera $2^{f(n)}$, D *rifiuta*
  6. Se M accetta, D *rifiuta*. Altrimenti, D *accetta*"

- In particolare, notiamo che:
  - M può avere un alfabeto di nastro diverso da D. In tal caso, codifichiamo i simboli aggiuntivi di M tramite una codifica scelta a priori, introducendo un fattore costante d sullo spazio utilizzato da D per rappresentare ognuno dei simboli di M.

    In altre parole, se M computa in spazio $g(n)$, allora D simula M in spazio $d \cdot g(n)$
  - Anche nel caso in cui M vada in loop infinito, l'esecuzione di D verrà terminata una volta effettuati $2^{f(n)}$ passi, implicando che D sia un decisore
  - Utilizzando un contatore binario per il numero di passi della simulazione, lo spazio massimo necessario sarà $O(\log(2^{f(n)})) = O(f(n))$

- Sia quindi A il linguaggio decidibile da D. Poiché l'esecuzione di D richiede $d \cdot g(n)$ spazio, ne segue che D decida A in spazio $d \cdot O(f(n)) = O(f(n))$

- Supponiamo quindi per assurdo che esista una TM M che decida A in spazio $g(n) = o(f(n))$. Per definizione stessa di o-piccolo, si ha che:

$$\forall c \in \mathbb{R}_{>0}\ \exists n_0 \in \mathbb{N}_{>0} \mid \forall n \geq n_0 \quad g(n) < c \cdot f(n)$$

  Di conseguenza, dato $\frac{1}{d} \in R_{>0}$, si ha che:

$$\exists n_0 \in \mathbb{N} \mid \forall n \geq n_0 \quad g(n) < \frac{1}{d} f(n) \implies d \cdot g(n) < f(n)$$

- Tuttavia, essendo $o(f(n))$ un comportamento asintotico, ciò risulta vero solo se $n \geq n_0$. Assumiamo quindi che $w = \langle M \rangle 10^{n_0}$, ossia che la stringa contenga un sufficiente numero di simboli 0.
- Poiché la simulazione effettuata da D richiede $d \cdot g(n) < f(n)$ spazio, il limite imposto dall'istruzione 3 non verrà mai superato. Di conseguenza, la simulazione verrà terminata, eseguendo l'istruzione finale e implicando dunque che $w \in L(D) = A \iff w \notin L(M) = A$, il che risulta assurdo
- Dunque, concludiamo necessariamente che tale TM non possa esistere e dunque che A sia decidibile in spazio $O(f(n))$ ma non in spazio $o\left(\frac{f(n)}{\log(f(n))}\right)$

□

Corollario 4.8: Distinzione spaziale tra le classi

Date due funzioni $f, g : \mathbb{N} \to \mathbb{N}$, se $f(n) = o(g(n))$ e g è spazio-costruibile, si ha che:

$$\mathsf{DSPACE}(f(n)) \subsetneq \mathsf{DSPACE}(g(n))$$

Dimostrazione.

- Poiché $f(n) = o(g(n))$ implica anche che $f(n) = O(g(n))$, si ha che:

$$\mathsf{DSPACE}(f(n)) \subseteq \mathsf{DSPACE}(g(n))$$

- Inoltre, per il Teorema di gerarchia di spazio, poiché g è spazio-costruibile ne segue che esista un linguaggio L decidibile in $O(g(n))$ ma non in $f(n) = o(g(n))$, concludendo che:

$$\mathsf{DSPACE}(f(n)) \subsetneq \mathsf{DSPACE}(g(n))$$

□

Teorema 4.42: Rapporto tra NL e PSPACE

Date le classi NL e PSPACE, si ha che:

$$\mathsf{NL} \subsetneq \mathsf{PSPACE}$$

Dimostrazione.

- Consideriamo la funzione $f(n) = n$
- Sia $g : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM G definita come:

  $G =$ "Data la stringa 1^n in input:

  1. Calcola $|1^n| = n$ tramite un contatore binario
  2. Restituisci in output la stringa n"

- Notiamo facilmente che $g(1^n) = f(n)_2$ e che lo spazio richiesto da G sia $O(\log n)$, implicando che esso sia anche $O(n) = O(f(n))$, concludendo che f sia spazio-costruibile
- Per il Teorema di Savitch, abbiamo che $\mathsf{NL} \subseteq \mathsf{DSPACE}(\log^2 n)$. Notiamo quindi che:

$$\lim_{n \to +\infty} \frac{\log n}{n} = 0$$

  implicando che $\log n = o(n)$
- Di conseguenza, per la Distinzione spaziale tra le classi, concludiamo che:

$$\mathsf{NL} \subseteq \mathsf{DSPACE}(\log^2 n) \subsetneq \mathsf{DSPACE}(n) \subseteq \mathsf{PSPACE}$$

□

Teorema 4.43: Rapporto tra PSPACE e EXPSPACE

Date le classi PSPACE e EXPSPACE, si ha che:

$$\mathsf{PSPACE} \subsetneq \mathsf{EXPSPACE}$$

Dimostrazione.

- Consideriamo la funzione $f(n) = 2^n$
- Sia $g : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM G definita come:

  $G =$ "Data la stringa 1^n in input:

  1. Calcola $|1^n| = n$ tramite un contatore binario
  2. Calcola $k = 2^n$ tramite un contatore binario
  3. Restituisci in output la stringa k"
- Notiamo facilmente che $g(1^n) = f(n)_2$ e che lo spazio richiesto da G sia $O(\log(2^n)) = O(n)$, implicando che esso sia anche $O(2^n)$, concludendo che f sia spazio-costruibile
- Notiamo inoltre che:

$$\forall k \in \mathbb{N} \quad \lim_{n \to +\infty} \frac{n^k}{2^n} = \lim_{n \to +\infty} \frac{n^k}{n^{\log n}} \cdot \frac{n^{\log n}}{2^n} = 0 \cdot 0 = 0$$

  implicando che $\forall k \in \mathbb{N}$ si abbia che $n^k = o(2^n)$
- Di conseguenza, per la Distinzione spaziale tra le classi, otteniamo che:

$$\forall k, j \in \mathbb{N} \quad \mathsf{DSPACE}(n^k) \subsetneq \mathsf{DSPACE}(2^n) \subseteq \mathsf{DSPACE}(2^{n^j})$$

  concludendo che $\mathsf{PSPACE} \subsetneq \mathsf{EXPSPACE}$

□

4.12.2 Teorema di gerarchia di tempo

Definizione 4.33: Funzione tempo-costruibile

Definiamo una funzione $f : \mathbb{N} \to \mathbb{N}$, dove $f(n) \geq \log n$, come **tempo-costruibile** se la seguente funzione:

$$g : \Sigma^* \to \Sigma^* : 1^n \mapsto f(n)_2$$

è calcolabile in tempo $O(f(n))$, dove $f(n)_2$ è la codifica binaria di $f(n)$

Nota: se f è una funzione razionale (es: $n \log n$ o $\sqrt{n}$), il risultato viene arrotondato all'intero precedente

Teorema 4.44: Teorema di gerarchia di tempo

Data una funzione tempo-costruibile $f : \mathbb{N} \to \mathbb{N}$, esiste un linguaggio L decidibile da una TM in tempo $O(f(n))$ ma non in tempo $o\left(\frac{f(n)}{\log(f(n))}\right)$

Dimostrazione.

- Sia D la TM definita come:

  D = "Data la stringa w in input:

  1. Calcola $\langle w \rangle = n$
  2. Calcola $f(n)$ in modo tempo-costruibile
  3. Memorizza il valore $\left\lceil \frac{f(n)}{\log(f(n))} \right\rceil$ in un contatore. Se tale contatore raggiunge il valore 0 durante le istruzioni successive, D *rifiuta*
  4. Interpreta $w = \langle M \rangle 10^*$, dove M è una TM. Se l'interpretazione fallisce, D *rifiuta*
  5. Simula M su input w. Ad ogni passo della simulazione, decrementa il contatore.
  6. Se M accetta, D *rifiuta*. Altrimenti, D *accetta*"

- In particolare, notiamo che:
  - M può avere un alfabeto di nastro diverso da D. In tal caso, codifichiamo i simboli aggiuntivi di M tramite una codifica scelta a priori, introducendo un fattore costante d sul tempo impiegato da D per rappresentare ognuno dei simboli di M.
  - Inoltre, poiché ad ogni passo della simulazione D deve decrementare il contatore avente dimensione pari a $\log\left(\left\lceil \frac{f(n)}{\log(f(n))} \right\rceil\right)$, operazione che può essere svolta in tempo $\log(f(n))$.

    In altre parole, se M computa in tempo $g(n)$, allora D simula M in tempo $d \cdot g(n) \cdot \log(f(n))$

  - Anche nel caso in cui M vada in loop infinito, l'esecuzione di D verrà terminata una volta che il contatore raggiungerà il valore 0, implicando che D sia un decisore

- Sia quindi A il linguaggio decidibile da D. Poiché l'esecuzione di D impiega $d \cdot g(n) \cdot \log(f(n))$ tempo, ne segue che D decida A in tempo $d \cdot O\left(\frac{f(n)}{\log(f(n))}\right) \cdot \log(f(n)) = O(f(n))$
- Supponiamo quindi per assurdo che esista una TM M che decida A in tempo $g(n) = o\left(\frac{f(n)}{\log(f(n))}\right)$. Per definizione stessa di o-piccolo, si ha che:

$$\forall c \in \mathbb{R}_{>0}\ \exists n_0 \in \mathbb{N}_{>0} \mid \forall n \geq n_0 \ \ g(n) < c \cdot \frac{f(n)}{\log(f(n))}$$

  Di conseguenza, dato $\frac{1}{d} \in R_{>0}$, si ha che:

$$\exists n_0 \in \mathbb{N} \mid \forall n \geq n_0 \ \ g(n) < \frac{1}{d} \cdot \frac{f(n)}{\log(f(n))} \implies d \cdot g(n) < \frac{f(n)}{\log(f(n))}$$

- Tuttavia, essendo $o\left(\frac{f(n)}{\log(f(n))}\right)$ un comportamento asintotico, ciò risulta vero solo se $n \geq n_0$. Assumiamo quindi che $w = \langle M \rangle\, 10^{n_0}$, ossia che la stringa contenga un sufficiente numero di simboli 0.
- Poiché la simulazione effettuata da D impiega $d \cdot g(n) < \frac{f(n)}{\log(f(n))}$ passi simulati (dunque escludendo il decremento del contatore poiché irrilevante in tal caso), il limite imposto dall'istruzione 3 non verrà mai superato. Di conseguenza, la simulazione verrà terminata, eseguendo l'istruzione finale e implicando dunque che $w \in L(D) = A \iff w \notin L(M) = A$, il che risulta assurdo
- Dunque, concludiamo necessariamente che tale TM non possa esistere e dunque che A sia decidibile in tempo $O(f(n))$ ma non in tempo $o\left(\frac{f(n)}{\log(f(n))}\right)$

□

Corollario 4.9: Distinzione temporale tra le classi

Date due funzioni $f, g : \mathbb{N} \to \mathbb{N}$, se $f(n) = o\left(\frac{g(n)}{\log(g(n))}\right)$ e g è tempo-costruibile, si ha che:

$$\mathsf{DTIME}(f(n)) \subsetneq \mathsf{DTIME}(g(n))$$

Dimostrazione.

- Poiché $f(n) = o\left(\frac{g(n)}{\log(g(n))}\right)$ implica anche che $f(n) = O\left(g(n)\right)$, si ha che:

$$\mathsf{DTIME}(f(n)) \subseteq \mathsf{DTIME}(g(n))$$

- Inoltre, per il Teorema di gerarchia di tempo, poiché g è tempo-costruibile ne segue che esista un linguaggio L decidibile in $O(g(n))$ ma non in $f(n) = o\left(\frac{g(n)}{\log(g(n))}\right)$, concludendo che:

$$\mathsf{DTIME}(f(n)) \subsetneq \mathsf{DTIME}\left(\frac{g(n)}{\log(g(n))}\right) \subseteq \mathsf{DTIME}(g(n))$$

□

Teorema 4.45: Rapporto tra P e EXP

Date le classi P e EXP, si ha che:

$$\mathsf{P} \subsetneq \mathsf{EXP}$$

Dimostrazione.

- Consideriamo la funzione $f(n) = 2^n$
- Sia $g : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM G definita come:

  $G =$ "Data la stringa 1^n in input:

  1. Calcola $|1^n| = n$ tramite un contatore binario
  2. Calcola $k = 2^n$ tramite un contatore binario
  3. Restituisci in output la stringa k"
- Notiamo facilmente che $g(1^n) = f(n)_2$ e che il tempo richiesto da G sia $O(2^n)$, concludendo che f sia spazio-costruibile
- Notiamo inoltre che:

$$\forall k \in \mathbb{N} \quad \lim_{n\to+\infty} \frac{n^{k+1}}{n^{\log n}} = n^{k+1-\log n} = 0$$

  poiché $k + 1 - \log n$ tende a $-\infty$ per $n \to +\infty$, e analogamente che:

$$\lim_{n\to+\infty} \frac{n^{\log n}}{2^n} = \lim_{n\to+\infty} \frac{2^{\log(n^{\log n})}}{2^n} = \lim_{n\to+\infty} \frac{2^{\log^2 n}}{2^n} = \lim_{n\to+\infty} 2^{\log^2 n - n} = 0$$

  poiché $\log^2 n - n$ tende a $-\infty$ per $n \to +\infty$
- Di conseguenza, si ha che:

$$\forall k \in \mathbb{N} \quad \lim_{n\to+\infty} \frac{n^k}{\frac{2^n}{\log 2^n}} = \lim_{n\to+\infty} \frac{n^k}{\frac{2^n}{n}} = \lim_{n\to+\infty} \frac{n^{k+1}}{2^n} = \lim_{n\to+\infty} \frac{n^{k+1}}{n^{\log n}} \cdot \frac{n^{\log n}}{2^n} = 0 \cdot 0 = 0$$

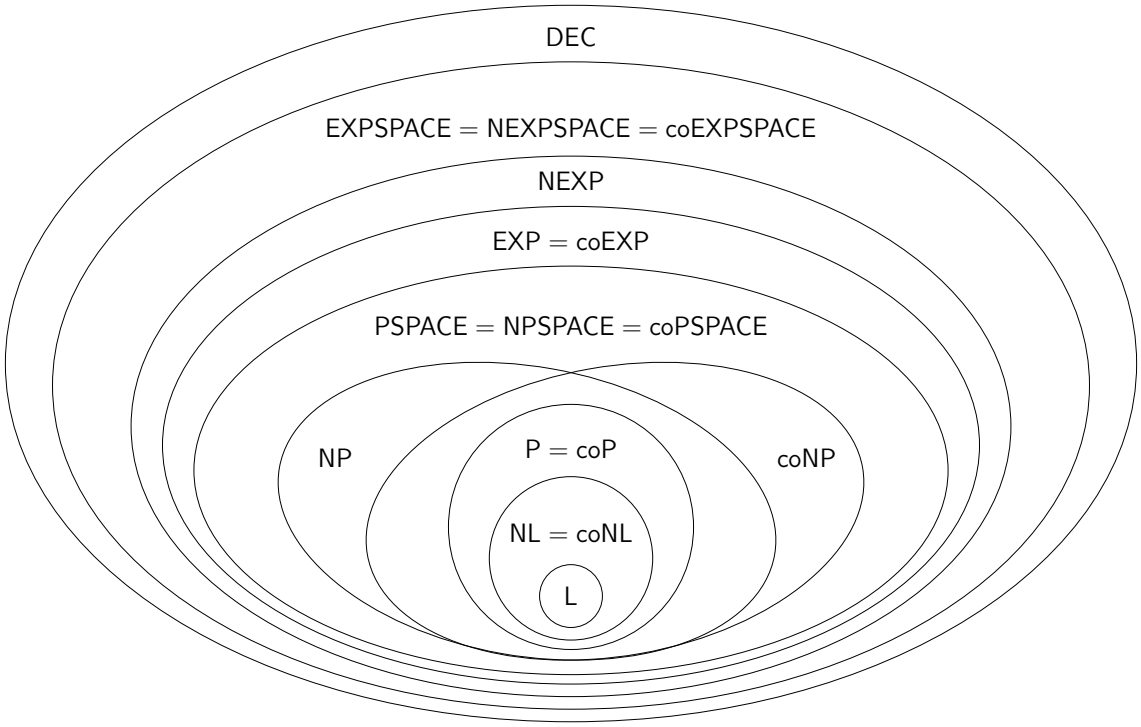
  implicando che $\forall k \in \mathbb{N}$ si abbia che $n^k = o\left(\frac{2^n}{\log 2^n}\right)$
- Di conseguenza, per la Distinzione temporale tra le classi, otteniamo che:

$$\forall k, j \in \mathbb{N} \quad \mathsf{DTIME}(n^k) \subsetneq \mathsf{DTIME}(2^n) \subseteq \mathsf{DTIME}(2^{n^j})$$

  concludendo che $\mathsf{P} \subsetneq \mathsf{EXP}$

□

Concludiamo quindi il nostro viaggio dando la seguente gerarchia attualmente congetturata dagli studiosi della teoria della complessità:



Gerarchia delle classi dei linguaggi decidibili attualmente congetturate

4.13 Esercizi svolti

Problema 4.1: Riducibilità quasi-polinomiale

Sia QP la classe dei linguaggi decidibili in tempo quasi-polinomiale, definita come:

$$\mathsf{QP} = \bigcup_{k=1}^{\infty} \mathsf{DTIME}(n^{\log^k n})$$

Dati due linguaggi A e B tali che $A \leq_m^P B$, stabilire se la seguente affermazione sia vera o falsa:

$$B \in \mathsf{QP} \implies A \in \mathsf{QP}$$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione tramite cui si ha che $A \leq_m^P B$, implicando che $\exists k \in \mathbb{N}$ tale che f sia computabile in $O(n^k)$
- Dato $B \in \mathsf{QP}$, sia D_B il decisore che decide B in tempo $O(n^{\log^j n})$, dove $j \in \mathbb{N}$
- Notiamo quindi che:

  $$\lim_{n \to +\infty} \frac{n^k}{n^{\log^j n}} = \lim_{n \to +\infty} n^{k - \log^j n} = 0$$

  poiché $k - \log^j n$ tende a $-\infty$ per $n \to +\infty$, implicando che $n^k = o(n^{\log^j n})$ e dunque che $n^k = O(n^{\log^j n})$
- Sia quindi D_A la TM definita come:

  D_A = "Data la stringa w in input:

  1. Calcola $f(w)$
  2. Esegui il programma di D_B su input $f(w)$.
  3. Se l'esecuzione accetta, *accetta*. Altrimenti, *rifiuta*"
- Risulta evidente che:

  $$w \in L(D_A) \iff f(w) \in L(D_B) = B$$

  implicando che $A = L(D_A) \in \mathsf{DEC}$.
- Inoltre, poiché ogni istruzione di D_A è eseguibile in tempo $O(n^{\log^j n})$, concludiamo che $A = L(D_A) \in \mathsf{DTIME}(n^{\log^j n}) \subseteq \mathsf{QP}$
- Dunque, stabiliamo che l'affermazione sia vera

□

Problema 4.2: P = NP = NP-Complete

Escludendo i linguaggi $\varnothing$ e Σ^* da qualsiasi classe, dimostrare che se P = NP allora NP = NP-Complete

Nota: l'esclusione di $\varnothing$ e Σ^* è necessaria in quanto per definizione stessa di riduzione tramite mappatura sia impossibile che un linguaggio sia riducibile a $\varnothing$ o Σ^* o che quest'ultimi siano riducibili ad un altro linguaggio

Dimostrazione.

- Osserviamo che NP-Complete $\subseteq$ NP-Complete = NP $\cap$ NP-Hard, dunque si ha che NP-Complete $\subseteq$ NP
- Consideriamo quindi $B \in \mathsf{NP} = \mathsf{P}$ tale che $B \neq \varnothing, \Sigma^*$. Poiché $B \neq \varnothing, \Sigma^*$, esistono necessariamente due stringhe $x, y \in \Sigma^*$ tali che $x \in B$ e $y \notin B$ (se non assumessimo ciò, una delle due stringhe non esisterebbe).
- Dato un qualsiasi linguaggio $A \in \mathsf{NP} = \mathsf{P}$, sia D_A il decisore in tempo polinomiale tale che $A = L(D_A)$
- Sia quindi $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F:

  F = "Data la stringa w in input:

  1. Esegui il programma di D_A su input w
  2. Se l'esecuzione accetta, restituisci in output la stringa x
  3. Altrimenti, restituisci in output la stringa y"
- In particolare, notiamo che poiché D_A sia un decisore, la TM F termini sempre l'esecuzione. Di conseguenza, non nascono problemi riguardo la calcolabilità di f
- Notiamo quindi che:

  $$w \in A = L(D_A) \implies f(w) = x \in B$$

  e inoltre che:

  $$w \notin A = L(D_A) \implies f(w) = y \notin B$$
- Poiché l'esecuzione di D_A viene svolta in tempo polinomiale, ne segue che f sia calcolabile in tempo polinomiale, implicando che

  $$\forall A \in \mathsf{NP} = \mathsf{P} \;\; A \leq_m^P B$$

  concludendo che $B \in$ NP-Complete

□

Problema 4.3: Padding argument

Dimostrare che se $\mathsf{P} = \mathsf{NP}$ allora $\mathsf{EXP} = \mathsf{NEXP}$

Dimostrazione.

- Sappiamo già che $\mathsf{EXP} \subseteq \mathsf{NEXP}$, quindi è sufficiente dimostrare che in tal caso si ha che $\mathsf{NEXP} \subseteq \mathsf{EXP}$. Dato $L \in \mathsf{NEXP}$, sappiamo che esiste una NTM N che decide L in tempo esponenziale.
- Consideriamo quindi il seguente linguaggio:

  $$L' = \{x1^{2^{|x|^c}} \mid x \in L\}$$

  dove c è una costante arbitraria.
- Sia N' la NTM definita come:

  $N' =$ "Data la stringa y in input:

  1. Genera una stringa casuale x.
  2. Verifica che $y = x1^{2^{|x|^c}}$. Se falso, *rifiuta*.
  3. Simula N su input x. Se la simulazione accetta, *accetta*. Altrimenti, *rifiuta*."

  Dove per definizione stessa di N' risulta evidente che $L(N') = L'$.
- A questo punto, è opportuno notare che la dimensione dell'input di N' sia di per se esponenziale. In particolare, per ogni stringa $x \in L$ tale che $|x| = n$, l'input di N' avrà lunghezza $m = n + 2^{n^c}$. Per tanto, poiché la simulazione di N su x richiede tempo esponenziale 2^{n^d}, dove d è un'altra costante, tale simulazione richiede $O(m^k)$ passi, dove $k = \max(c, d)$, dunque è polinomiale rispetto alla dimensione dell'input. Ciò conclude che N' sia un decisore non-deterministico per L'.
- Poiché in ipotesi si ha che $\mathsf{P} = \mathsf{NP}$, deve esistere un decisore di tempo polinomiale M' tale che $L(M') = L'$. Definiamo quindi la seguente TM:

  $M =$ "Data la stringa x in input:

  1. Simula M' su input $x1^{2^{|x|^c}}$. Se la simulazione accetta, *accetta*. Altrimenti, *rifiuta*.

  Dove per definizione stessa di N' risulta evidente che $L(M) = L$. Inoltre, dato $|x| = n$, la simulazione di M' richiede tempo esponenziale rispetto alla dimensione dell'input, dunque concludiamo che $L \in \mathsf{EXP}$.

□

Problema 4.4: Chiusure di NL

Dimostrare che NL è chiusa rispetto alle operazioni di unione, intersezione e star di Kleene

Dimostrazione.

- Dati due linguaggi $A, B \in \mathsf{NL}$, siano V_A e V_B i rispettivi verificatori in spazio $O(\log n)$ per A e B
- Sia quindi V_U la TM definita come:

  V_U = "Data la stringa $\langle w, c\rangle$ in input:

  1. Esegui il programma di V_A su input $\langle w, c\rangle$. Se l'esecuzione accetta, V_U *accetta*
  2. Esegui il programma di V_B su input $\langle w, c\rangle$. Se l'esecuzione accetta, V_U *accetta*
  3. Altrimenti, *rifiuta*"
- Per costruzione di V_U, risulta evidente che:

$$w \in A \cup B \iff w \in A \vee w \in B \iff$$

$$\exists c \in \Sigma^*, \langle w, c\rangle \in L(V_A) \vee \langle w, c\rangle \in L(V_B) \iff \langle w, c\rangle \in L(V_U)$$

  implicando che V_U sia un verificatore per $A \cup B$. Inoltre, poiché sia l'esecuzione di V_A che l'esecuzione di V_B richiedono spazio $O(\log n)$, concludiamo che $A \cup B \in \mathsf{NL}$
- Analogamente, sia V_I la TM definita come:

  V_I = "Data la stringa $\langle w, c\rangle$ in input:

  1. Esegui il programma di V_A su input $\langle w, c\rangle$. Se l'esecuzione rifiuta, V_U *rifiuta*
  2. Esegui il programma di V_B su input $\langle w, c\rangle$. Se l'esecuzione rifiuta, V_U *rifiuta*
  3. Altrimenti, *accetta*"
- Per costruzione di V_I, risulta evidente che:

$$w \in A \cap B \iff w \in A \wedge w \in B \iff$$

$$\exists c \in \Sigma^*, \langle w, c\rangle \in L(V_A) \wedge \langle w, c\rangle \in L(V_B) \iff \langle w, c\rangle \in L(V_I)$$

  implicando che V_I sia un verificatore per $A \cup B$. Inoltre, poiché sia l'esecuzione di V_A che l'esecuzione di V_B richiedono spazio $O(\log n)$, concludiamo che $A \cup B \in \mathsf{NL}$
- Sia invece N la NTM che decide A in tempo $O(\log n)$
- Sia quindi S la TM definita come:

  S = "Data la stringa $\langle w, c\rangle$ in input:

  1. Calcola $|w| = n$
  2. Scegli non deterministicamente un valore $k \in [1, n]$

3. Scegli non deterministicamente una partizione di w, ossia delle stringhe $w_1, \dots, w_k$ tali che $w = w_1 \dots w_k$
4. Ripeti le seguenti istruzioni per $i = 1, \dots, k$:
   5. Esegui N su input w_i. Se N rifiuta allora S *rifiuta*
   6. Altrimenti, pulisci lo spazio utilizzato ed esegui la prossima iterazione sullo stesso spazio
7. *Accetta*"

- Per costruzione stessa di S, si ha che:

$$w \in L(S) \iff \text{Esiste un ramo che accetta } w \iff$$

$$\text{Esiste una partizione } w_1, \dots, w_k \text{ tale che } \forall i \in [1, k] \;\; w_i \in L(N) = A$$

$$\iff w = w_1 \dots w_k \in A^*$$

implicando che $L(S) = A^*$. Inoltre, poiché l'esecuzione di N richiede spazio $O(\log n)$ ed S utilizza solo dei contatori, lo spazio richiesto da S risulta essere $O(\log n)$, concludendo $A^* = L(S) \in \mathsf{NL}$

□

Problema 4.5

Dimostrare che per ogni $k \in \mathbb{N}$ si abbia che $\mathsf{NTIME}(n^k) \subsetneq \mathsf{PSPACE}$

Stabilire se tale risultato sia sufficiente a dire che $\mathsf{NP} \subsetneq \mathsf{PSPACE}$

Dimostrazione.

- Tramite il teorema del Tempo limitante lo spazio e il Teorema di Savitch, si ha che:

$$\forall k \in \mathbb{N} \;\; \mathsf{NTIME}(n^k) \subseteq \mathsf{NSPACE}(n^k) \subseteq \mathsf{DSPACE}(n^{2k})$$

- Inoltre, poiché $n^{2k} = o(n^{2k+1})$, per la Distinzione spaziale tra le classi, si ha che:

$$\forall k \in \mathbb{N} \;\; \mathsf{NTIME}(n^k) \subseteq \mathsf{DSPACE}(n^{2k}) \subsetneq \mathsf{DSPACE}(n^{2k+1}) \subseteq \mathsf{PSPACE}$$

- Controintuitivamente, il risultato non è sufficiente a stabilire che $\mathsf{NP} \subsetneq \mathsf{PSPACE}$. Difatti, per stabilire ciò è necessario dimostrare che:

$$\forall k, j \in \mathbb{N} \;\; \mathsf{NTIME}(n^k) \subsetneq \mathsf{DSPACE}(n^j)$$

□

Problema 4.6: Problema della soddisfacibilità delle 2CNF

Dato il seguente linguaggio:

$$2SAT = \{\langle\phi\rangle \mid \phi \text{ è una 2CNF soddisfacibile}\}$$

dimostrare che $2SAT \in \mathsf{P}$

Dimostrazione.

- Prima di procedere con la dimostrazione, forniamo l'intuizione alla base della dimostrazione stessa:
  - Dati due letterali α e β si ha che:

$$(\alpha \vee \beta) \equiv (\overline{\alpha} \implies \beta) \equiv (\overline{\beta} \implies \alpha)$$

- A questo punto, sia P il decisore polinomiale tale che $L(P) = PATH \in \mathsf{P}$
- Sia inoltre M la TM definita come:

  M = "Data la stringa $\langle\phi\rangle$ in input:

  1. Costruisci un grafo $G = (V, E)$ definito come:
     - Per ogni variabile x in ϕ, crea i nodi x e $\overline{x}$
     - Per ogni clausola $(\alpha \vee \beta)$ in ϕ, dove α e β sono dei letterali (ossia una variabile o una sua negazione), crea gli archi $(\overline{\alpha}, \beta)$ e $(\overline{\beta}, \alpha)$
  2. Ripeti le seguenti istruzioni per ogni variabile x in ϕ:
     3. Esegui P su input $\langle G, x, \overline{x}\rangle$
     4. Esegui P su input $\langle G, \overline{x}, x\rangle$
     5. Se entrambe le esecuzioni accettano, *rifiuta*
  6. *Accetta*"
- Consideriamo quindi una 2CNF ϕ e il grafo G ad essa associato.
- Notiamo che dati due letterali α e β, si abbia che:

$$\begin{aligned}
&\text{Esiste cammino } \alpha \to \beta \text{ in } G \iff \exists(\alpha, \gamma_1), (\gamma_1, \gamma_2), \ldots, (\gamma_k, \beta) \in E \\
&\iff (\overline{\alpha} \vee \gamma_1) \wedge (\overline{\gamma_1} \vee \gamma_2) \wedge \ldots \wedge (\overline{\gamma_k} \vee \beta) \text{ sottoformula di } \phi \\
&\iff (\alpha \implies \gamma_1) \wedge (\gamma_1 \implies \gamma_2) \wedge \ldots \wedge (\gamma_k \implies \beta) \text{ sottoformula di } \phi \\
&\iff \text{Per } \phi \text{ vale l'implicazione } (\alpha \implies \beta)
\end{aligned}$$

- Supponiamo quindi per assurdo che ϕ sia soddisfacibile e che esista una variabile x_i in ϕ per cui esistano i cammini $x \to \overline{x}$ e $\overline{x} \to x$ in G

- Sia quindi $\phi(x_1, \ldots, x_i, \ldots, x_m)$ un assegnamento che soddisfa ϕ:
  - Se $x_i = \text{True}$, tramite il cammino $x_i \to \overline{x_i}$ ne segue che affinché ϕ rimanga soddisfacibile sia necessario che l'implicazione $x_i \implies \overline{x_i}$ sia vera, il che è possibile solo se $\overline{x_i} = \text{True}$, implicando che $x_i = \overline{x_i}$, portando dunque ad un assurdo
  - Se $\overline{x_i} = \text{True}$, tramite il cammino $\overline{x_i} \to x_i$ ne segue che affinché ϕ rimanga soddisfacibile sia necessario che l'implicazione $\overline{x_i} \implies x_i$ sia vera, cosa possibile solo se $x_i = \text{True}$, implicando che $x_i = \overline{x_i}$, portando dunque ad un assurdo
- Di conseguenza, ne segue necessariamente che:

$$\phi \text{ soddisfacibile} \implies \nexists \text{ variabile } x \text{ in } \phi \text{ per cui } x \to \overline{x} \text{ e } \overline{x} \to x$$

- Viceversa, supponiamo che tale variabile x_i non esista. Di conseguenza, per ogni coppia di letterali α e β ne segue che se $\alpha \to \beta$ in G, l'implicazione $(\alpha \implies \beta)$ valida in ϕ è sempre soddisfacibile a vuoto.
- Di conseguenza, esisterà sempre un assegnamento $\phi(x_1, \ldots, x_m)$ in grado di soddisfare ϕ, implicando che:

$$\nexists \text{ variabile } x \text{ in } \phi \text{ per cui } x \to \overline{x} \text{ e } \overline{x} \to x \implies \phi \text{ soddisfacibile}$$

- A questo punto, per costruzione di M, ne segue che:

$$\langle \phi \rangle \in 2SAT \iff \nexists \text{ variabile } x \text{ in } \phi \text{ per cui } x \to \overline{x} \text{ e } \overline{x} \to x \iff \langle \phi \rangle \in L(M)$$

  implicando che $2SAT = L(M)$
- Inoltre, poiché $L(P) = PATH \in \mathsf{P}$ e poiché il grafo è costruibile in tempo polinomiale, concludiamo che M sia un decisore polinomiale e dunque che $2SAT = L(M) \in \mathsf{P}$

□

Problema 4.7: Strongly Non deterministic TM

Una Strongly Non deterministic TM (SNTM) è una TM S dotata di tre possibili stati finali: *accept*, *reject* e *not sure*.

In particolare, una SNTM S decide il linguaggio $L(S)$ nel seguente modo:

- Se $w \in L(S)$ allora tutti i rami dell'albero di computazione di S su w terminano con *accept* o *not sure*, dove almeno uno di tali rami termina su *accept*
- Se $w \notin L(S)$ allora tutti i rami dell'albero di computazione di S su w terminano con *reject* o *not sure*, dove almeno uno di tali rami termina su *reject*

Dimostrare che dato un linguaggio L si ha che:

$$L \in \mathsf{NP} \cap \mathsf{coNP} \iff \text{Esiste SNTM che decide } L \text{ in tempo polinomiale}$$

Dimostrazione.

Prima implicazione.

- Dato $L \in \mathsf{NP} \cap \mathsf{coNP}$, si ha che $L, \overline{L} \in \mathsf{NP}$
- Siano quindi $N, \overline{N}$ le due NTM che decidono rispettivamente L e $\overline{L}$ in tempo polinomiale
- Sia inoltre S la SNTM definita come:

  $S =$ "Data la stringa w in input:

  1. Esegui non deterministicamente un ramo di N su input w
  2. Esegui non deterministicamente un ramo di $\overline{N}$ su input w
  3. Se l'esecuzione del ramo di N accetta e l'esecuzione del ramo di $\overline{N}$ rifiuta, allora S termina su *accept*
  4. Se l'esecuzione del ramo di N rifiuta e l'esecuzione del ramo di $\overline{N}$ accetta, allora S termina su *reject*
  5. Altrimenti, S termina su *not sure*
- In particolare, notiamo che per ogni ramo di esecuzione di S si abbia che:
  - Se il ramo di N accetta e il ramo di $\overline{N}$ rifiuta, allora $w \in L(N) = L$ e S termina su *accept*
  - Se il ramo di N rifiuta e il ramo di $\overline{N}$ accetta, allora $w \in \overline{L} \iff w \notin L$ e S termina su *reject*
  - Se il ramo di N rifiuta e il ramo di $\overline{N}$ rifiuta, non possiamo dire nulla sull'appartenenza di w ad uno dei due linguaggi e S termina su *not sure*
  - È impossibile che sia il ramo di N che il ramo di $\overline{N}$ accettino, poiché ciò implicherebbe che $w \in L \cap \overline{L} = \varnothing$

- Supponiamo quindi che $w \in L(S)$. Per definizione, ne segue che esista un ramo di S che accetta e nessun ramo di S che rifiuta, implicando che esista un ramo di N che accetta w e dunque che $w \in L$
- Viceversa, se $w \in L(S)$ allora per definizione, ne segue che esista un ramo di S che rifiuta e nessun ramo di S che accetta, implicando che esista un ramo di $\overline{N}$ che accetta w e dunque che $w \notin L$
- Di conseguenza, concludiamo che:

  $$w \in L(S) \iff w \in L$$

  implicando che $L(S) = L$
- Inoltre, poiché N e $\overline{N}$ sono decidono in tempo polinomiale, ogni ramo di S avrà tempo di esecuzione polinomiale, concludendo che anche S decida L in tempo polinomiale

Seconda implicazione.

- Supponiamo che esista una SNTM S che decide un linguaggio L in tempo polinomiale
- Sia N la NTM definita come:

  N = "Data la stringa w in input:

  1. Esegui non deterministicamente un ramo di S su input w
  2. Se l'esecuzione del ramo di S accetta, allora N *accetta*. Altrimenti, N *rifiuta*"
- Per costruzione stessa di N, ne segue che:

  $$w \in L(S) \iff \text{Esiste un ramo di } S \text{ che accetta e nessun ramo che rifiuta}$$
  $$\implies \text{Esiste almeno un ramo di } N \text{ che accetta } w \iff w \in L(N)$$

  e inoltre che:

  $$w \notin L(S) \iff \text{Esiste un ramo di } S \text{ che rifiuta e nessun ramo che accetta}$$
  $$\implies \text{Non esiste un ramo di } N \text{ che accetta } w \iff w \notin L(N)$$

  dunque concludiamo che $L = L(S) = L(N)$. Inoltre, poiché l'esecuzione di S è polinomiale, ne segue che anche N decida L in tempo polinomiale, implicando che $L \in \mathsf{NP}$
- Analogamente, sa $\overline{N}$ la NTM definita come:

  $\overline{N}$ = "Data la stringa w in input:

  1. Esegui non deterministicamente un ramo di S su input w
  2. Se l'esecuzione del ramo di S rifiuta, allora N *accetta*. Altrimenti, N *rifiuta*"

- Per costruzione stessa di N, ne segue che:

  $$w \in L(S) \iff \text{Esiste un ramo di } S \text{ che accetta e nessun ramo che rifiuta}$$

  $$\implies \text{Non esiste un ramo di } N \text{ che accetta } w \iff w \notin L(N)$$

  e inoltre che:

  $$w \notin L(S) \iff \text{Esiste un ramo di } S \text{ che rifiuta e nessun ramo che accetta}$$

  $$\implies \text{Esiste almeno un ramo di } N \text{ che accetta } w \iff w \in L(N)$$

  dunque concludiamo che $\overline{L} = \overline{L(S)} = L(N)$. Inoltre, poiché l'esecuzione di S è polinomiale, ne segue che anche N decida $\overline{L}$ in tempo polinomiale, implicando che $\overline{L} \in \mathsf{NP}$
- Di conseguenza, poiché $L, \overline{L} \in \mathsf{NP}$, concludiamo che $L \in \mathsf{NP} \cap \mathsf{coNP}$

□

Problema 4.8

Dato il seguente linguaggio:

$$\textit{min-k-PATH} = \left\{ \langle G, s, t, k \rangle \;\middle|\; \begin{array}{c} G = (V_G, E_G) \text{ grafo non diretto con} \\ \text{cammino } s \to t \text{ con minimo } k \text{ archi} \end{array} \right\}$$

dimostrare che $\textit{min-k-PATH} \in \mathsf{NP}$

Dimostrazione.

- Sia V la TM definita come:

  V = "Data in input la stringa $\langle \langle G, s, t, k \rangle, P \rangle$:

  1. Interpreta $P = v_1, \ldots, v_h$, dove $v_1, \ldots, v_h \in V_G$. Se falso, *rifiuta*.
  2. Verifica che $v_1 = s$ e che $v_h = t$. Se falso, *rifiuta*.
  3. Verifica che $h \geq k$. Se falso, *rifiuta*
  4. Verifica che $v_i \neq v_j$ per ogni $i \neq j$. Se falso, *rifiuta*.
  5. Verifica che per ogni $i = 1, \ldots, h-1$ si abbia che $(v_i, v_{i+1}) \in E_G$. Se falso, *rifiuta*.
  6. *Accetta*.
- Chiaramente, dalla definizione stessa di V risulta evidente che esso sia un verificatore per *min-k-PATH*. Inoltre, ogni operazione svolta da V è eseguibile in tempo polinomiale, dunque concludiamo $L \in \mathsf{NP}$.

□

Problema 4.9: $3COL \leq_m^P 4COL$

Dato un grafo G, diciamo che G è *k-colorabile* se dati k colori è possibile marcare ogni nodo di G con uno dei k colori in modo che non esistano due nodi adiacenti con lo stesso colore assegnato.

Dati i seguenti due linguaggi:

$$3COL = \{\langle G \rangle \mid G \text{ grafo 3-colorabile}\}$$

$$4COL = \{\langle G \rangle \mid G \text{ grafo 4-colorabile}\}$$

dimostrare che $3COL \leq_m^P 4COL$

Dimostrazione.

- Sia $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  $F =$ "Data la stringa $\langle G \rangle$ in input, dove $G = (V, E)$:

  1. Aggiungi un nodo v a G
  2. Per ogni nodo u in G diverso da v, aggiungi un arco (u, v)
  3. Restituisci in output la stringa $\langle G' \rangle$, dove G' è il grafo modificato"

- Supponiamo quindi che $\langle G \rangle \in 3COL$, implicando che $\forall v_1, v_2, v_3 \in V$ si abbia che (v_1, v_2, v_3) è 3-colorabile.
- Poiché il nodo v aggiunto è adiacente ad ogni altro nodo in V, assegnando ad esso un quarto colore risulta evidente che (v_1, v_2, v_3, v) è 4-colorabile, implicando che $\langle G' \rangle \in 4COL$
- Supponiamo invece che $\langle G \rangle \notin 3COL$, implicando che $\exists v_1, v_2, v_3 \in V$ per cui (v_1, v_2, v_3) non è 3-colorabile
- Consideriamo quindi la quadrupla (v_1, v_2, v_3, v). Assegnando un quarto colore a v, la quadrupla non sarà 4-colorabile poiché (v_1, v_2, v_3) non è 3-colorabile, implicando che (v_1, v_2, v_3, v) non è 4-colorabile, dunque che $\langle G' \rangle \notin 4COL$
- Dunque, ne segue che $\langle G \rangle \in 3COL \iff f(\langle G \rangle) = \langle G' \rangle \in 4COL$. Inoltre, poiché l'aggiunta del nodo v e degli m archi, dove $m = |V|$, richiede tempo polinomiale, concludiamo che $3COL \leq_m^P 4COL$

□

Problema 4.10

Dato un grafo G, diciamo che G è *k-colorabile* se dati k colori è possibile marcare ogni nodo di G con uno dei k colori in modo che non esistano due nodi adiacenti con lo stesso colore assegnato.

Dato il seguente linguaggio:

$$4COL = \{\langle G \rangle \mid G \text{ grafo 4-colorabile}\}$$

dimostrare che $4COL \leq_m^P SAT$

Dimostrazione.

- Assumiamo che i quattro colori utilizzati siano rosso, verde, blu e giallo. Per ogni nodo $v \in V(G)$, definiamo le variabili v_r, v_g, v_b, v_y, dove ad esempio $v_r = \texttt{True}$ se e solo se v viene colorato di rosso.
- Per ogni nodo $v \in V(G)$, definiamo la seguente formula:

  $$\phi_v = (v_r \vee v_g \vee v_b \vee v_y)$$

  Intuitivamente, tale formula esprime che il nodo v debba essere di almeno uno dei 4 colori disponibili. Per ogni arco $(v, u) \in E(G)$, invece, definiamo la seguente formula:

  $$\phi_{(v,u)} = (\overline{v_r} \vee \overline{u_r}) \wedge (\overline{v_g} \vee \overline{u_g}) \wedge (\overline{v_b} \vee \overline{u_b}) \wedge (\overline{v_y} \vee \overline{u_y})$$

  Tale formula esprime che i nodi v, u appartenenti all'arco (v, u) devono avere colore diverso. Infine, definiamo quindi la seguente formula:

  $$\phi_G = \left(\bigwedge_{v \in V(G)} \phi_v \right) \wedge \left(\bigwedge_{(v,u) \in E(G)} \phi_{(v,u)} \right)$$

  Per costruzione stessa, risulta evidente che G sia 4-colorabile se e solo se ϕ_G è soddisfacibile.
- Sia quindi $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F:

  $F =$ "Data la stringa $\langle G \rangle$ in input:

  1. Costruisci $\langle \phi \rangle_G$ e restituiscila"

  Chiaramente, abbiamo che $\langle G \rangle \in 4COL \iff f(\langle G \rangle) = \langle \phi_G \rangle \in SAT$.
- Siano quindi $|V(G)| = k$ e $|E(G)| = m$. Per costruire tutte le formule ϕ_v sono necessarie $O(k)$ operazioni, mentre per costruire tutte le formule $\phi_{(v,u)}$ sono necessarie $8O(m)$ operazioni, dunque F impiega $O(k + m)$ operazioni. Tuttavia, notiamo facilmente che $k, m = O(n)$, dunque concludiamo che f sia una riduzione polinomiale per cui $4COL \leq_m^P SAT$.

□

Problema 4.11: NP-Completezza di $HALF\text{-}CLIQUE$

Dato il seguente linguaggio:

$$HALF\text{-}CLIQUE = \left\{ \langle G \rangle \;\middle|\; \begin{array}{c} G = (V_G, E_G) \text{ grafo non diretto} \\ \text{con una clique di } \frac{|V_G|}{2} \text{ nodi} \end{array} \right\}$$

dimostrare che $HALF\text{-}CLIQUE \in$ NP-Complete

Suggerimento: ricordare che abbiamo dimostrato che $SAT, 3SAT$ e $CLIQUE$ siano NP-Completi

Dimostrazione.

- Sia V la TM definita come:

  $V =$ "Data la stringa $\langle \langle G \rangle, c \rangle$ in input:

  1. Interpreta $c = \langle v_1, \ldots, v_k \rangle$, dove $v_1, \ldots, v_k \in V_G$
  2. Calcola $\frac{|V_G|}{2} = m$
  3. Se $m \neq k$, *rifiuta*
  4. Altrimenti, ripeti lo step successivo per ogni coppia di nodi v_i, v_j:
     5. Se $(v_i, v_j) \notin E_G$, *rifiuta*
  6. *Accetta*"

- Posto $\frac{|V_G|}{2} = m$, per costruzione di V risulta evidente che:

  $$\langle G \rangle \in HALF\text{-}CLIQUE \iff \text{Esiste una } m\text{-clique in } G \iff$$

  $$\exists c \in \Sigma^*, \langle \langle G \rangle, c \rangle \in L(V)$$

  implicando che V sia un verificatore per $HALF\text{-}CLIQUE$. Inoltre, poiché V svolge solo operazioni polinomiali, concludiamo che $HALF\text{-}CLIQUE \in$ NP

- Successivamente, mostriamo che $3SAT \leq_m^P HALF\text{-}CLIQUE$

- Sia quindi $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  $F =$ "Data la stringa $\langle \phi \rangle$ in input:

  1. Costruisci un grafo G definito come:
     i. Conta il numero k di clausole or in ϕ
     ii. Per ogni clausola, crea un nodo x_i per ogni letterale della clausola, creando dei doppioni se la variabile compare più volte
     iii. Organizza i nodi di G in k triple (una per clausola *or*), dove tre nodi appartengono alla stessa tripla se e solo se i loro letterali compaiono all'interno di una stessa clausola or

iv. Crea un arco tra ogni coppia di nodi fatta eccezione di nodi appartenenti alla stessa tripla e nodi rappresentanti letterali opposti tra loro (es: x_i e $\overline{x_i}$)

v. Crea i nodi $v_1, \ldots, v_k$

vi. Per ogni nodo $v_1, \ldots, v_k$, crea un arco verso ogni nodo di G

2. Restituisci in output la stringa $\langle G, k \rangle$"

- Prima di tutto, notiamo che nel grafo G costruito da F vi siano $3k$ nodi derivati dalle k triple e k-nodi aggiunti alla fine, per un totale di $4k$ nodi
- Supponiamo che $\langle \phi \rangle \in 3SAT$. Poiché ϕ è una 3CNF, affinché essa sia soddisfacibile ne segue che ogni clausola sia soddisfacibile, dunque che almeno uno dei letterali di ognuna di tali clausole sia vero
- Sia quindi $C = \{x_1, \ldots, x_k\}$ l'insieme dei nodi tali che $\forall i \in [1, k] \quad x_i$ è il nodo corrispondente al letterale vero della i-esima tripla
- Poiché tali nodi appartengono tutti a triple diverse e poiché $\nexists x_i, x_j \in C$ tali che $x_i = \overline{x_j}$ in quanto non possono essere entrambi veri, ne segue che essi siano tutti due a due adiacenti
- Inoltre, poiché i nodi $v_1, \ldots, v_k$ sono adiacenti ad ogni nodo del grafo, concludiamo che $C \cup \{v_1, \ldots, v_k\}$ siano una $2k$-clique (ricordiamo che i nodi totali siano $4k$)
- Di conseguenza, abbiamo che:

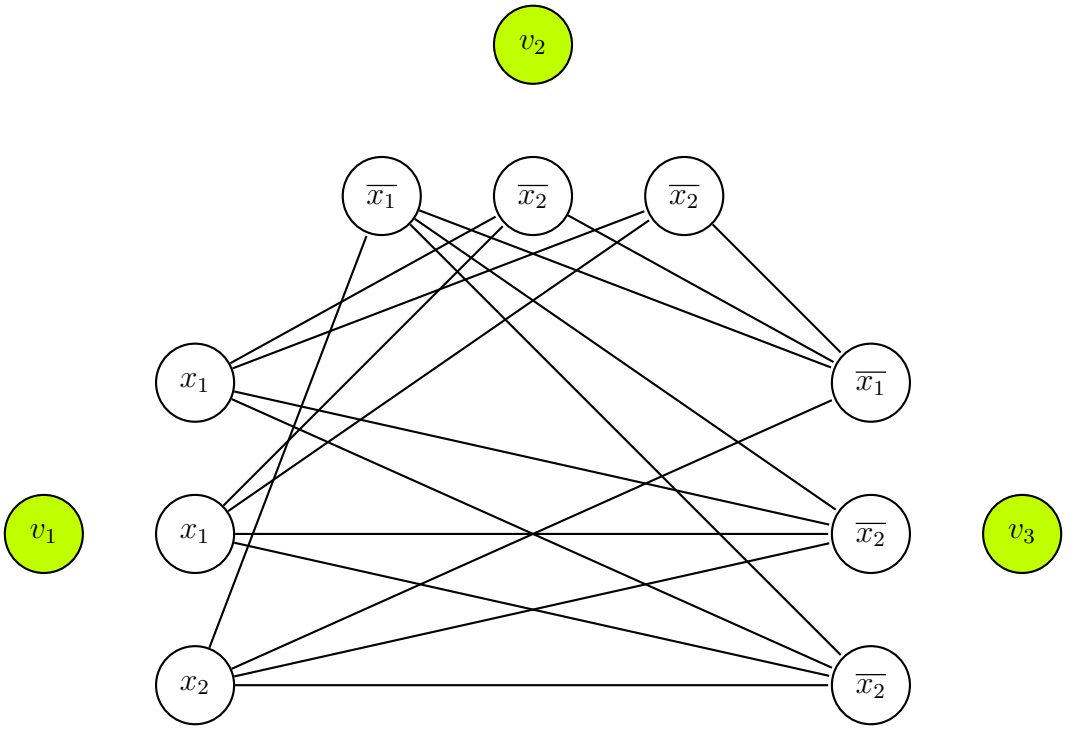
$$\langle \phi \rangle \in 3SAT \implies f(\langle \phi \rangle) = \langle G \rangle \in HALF\text{-}CLIQUE$$

- Supponiamo ora che $f(\phi) = \langle G \rangle \in HALF\text{-}CLIQUE$, implicando che esista una $2k$-clique al suo interno. In particolare, k di tali $2k$ nodi corrisponderanno ai nodi $v_1, \ldots, v_k$ aggiunti alla fine. Per quanto riguarda gli altri k nodi, per costruzione di G, non esistono nodi all'interno della clique i cui letterali rappresentati appartengono alla stessa tripla
- A questo punto, poiché tali letterali non interferiscono tra loro trovandosi in triple diverse, esiste un un assegnamento che soddisfi ogni clausola. In particolare, tale assegnamento risulta sempre possibile in quanto all'interno della clique non possano appartenere sia x_i che $\overline{x_i}$
- Di conseguenza, abbiamo che:

$$f(\langle \phi \rangle) = \langle G, w \rangle \in HALF\text{-}CLIQUE \iff \langle \phi \rangle \in 3SAT$$

Inoltre, poiché F svolge solo operazioni eseguibili in tempo polinomiale, concludiamo che $3SAT \leq_m^P HALF\text{-}CLIQUE$ e dunque che $HALF\text{-}CLIQUE \in$ NP-Complete

□



Grafo generato dalla funzione f della dimostrazione a partire dalla formula $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$
I nodi verdi sono adiacenti ad ogni nodo del grafo (archi omessi per chiarezza)

Problema 4.12: NP-Completezza di *DOM-SET*

Dato un grafo G, un *dominant set* di G è un sottoinsieme di vertici $D \subseteq V(G)$ per cui ogni vertice $v \in V(G)$ appartiene a D oppure è adiacente ad vertice in D.

Dato il seguente linguaggio:

$$DOM\text{-}SET = \left\{ \langle G, k \rangle \;\middle|\; \begin{array}{l} G = (V_G, E_G) \text{ grafo non diretto con} \\ \text{un dominant set di massimo } k \text{ nodi} \end{array} \right\}$$

dimostrare che $DOM\text{-}SET \in$ NP-Complete

Suggerimento: ricordare che abbiamo dimostrato che $SAT, 3SAT$ e $CLIQUE$ siano NP-Completi

Dimostrazione.

- Dato un vertice $v \in V_G$, definiamo $\delta(v)$ come l'insieme composto da v e dai nodi adiacenti a v, ossia $\delta(v) = \{u \in V_G \mid u = v \vee \exists (u, v) \in E(G)\}$
- Sia V la TM definita come:

  $V =$ "Data la stringa $\langle \langle G, k \rangle, c \rangle$ in input:

  1. Interpreta $c = \langle v_1, \ldots, v_h \rangle$ dove $v_1, \ldots, v_k \in V_G$
  2. Verifica che $h \leq k$. Se falso, *rifiuta*.
  3. Per ogni vertice $v \in V_G$, verifica se $\delta(v) \cap \{v_1, \ldots, v_h\} \neq \varnothing$. Se falso, *rifiuta*.
  4. *Accetta*."

  Per definizione stessa di V risulta evidente che $\exists c \in \Sigma^*$ tale che $\langle \langle G, k \rangle, c \rangle \in L(V)$ se e solo se $\langle G, k \rangle \in DOM\text{-}SET$. Inoltre, le prime due operazioni di V possono essere svolte in tempo $O(n)$, mentre la terza operazione può essere svolta in tempo $O(n^3)$, concludendo che V sia un verificatore polinomiale per $DOM\text{-}SET$ e dunque che $DOM\text{-}SET \in$ NP.
- Successivamente, mostriamo che $3SAT \leq_m^P DOM\text{-}SET$.
- Sia quindi $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

  $F =$ "Data la stringa $\langle \phi \rangle$ in input:

  1. Conta il numero di variabili m definite in ϕ
  2. Costruisci il grafo G definito come:
     i. Per ogni clausola C_k crea un nodo v_{C_k}
     ii. Per ogni variabile x definita su ϕ crea i nodi $v_x, v_{\overline{x}}, v_{t_x}$, dove v_{t_x} è un terzo nodo ausiliario, e crea gli archi $(v_x, v_{\overline{x}}), (v_x, v_{t_x})$ e $(v_{\overline{x}}, v_{t_x})$.
     iii. Per ogni letterale $\ell_{i,k}$ presente nella clausola C_k crea l'arco $(v_{C_k}, v_{\ell_{i,k}})$
  3. Restituisci in output la stringa $\langle G, m \rangle$"

- Supponiamo che ϕ sia soddisfacibile da un assegnamento α. Consideriamo quindi il sottoinsieme $D \subseteq V(G)$ dove per ogni letterale $\ell_{i,k}$ si ha che:

$$v_{\ell_{i,k}} \in D \iff \alpha(\ell_{i,k}) = \text{True}$$

- Poiché in ogni assegnamento, α incluso dunque, un letterale è vero se e solo se il suo letterale negato è falso, ne segue che almeno uno solo tra essi sia presente in D. Dunque, quello non presente in D sarà adiacente a quello presente. Inoltre, poiché α soddisfa ϕ, ogni clausola C_k in ϕ avrà almeno un letterale al suo interno impostato su vero, implicando che C_k sia sempre adiacente ad un vertice in D. Dunque, D è un dominant set di massimo , nodi.

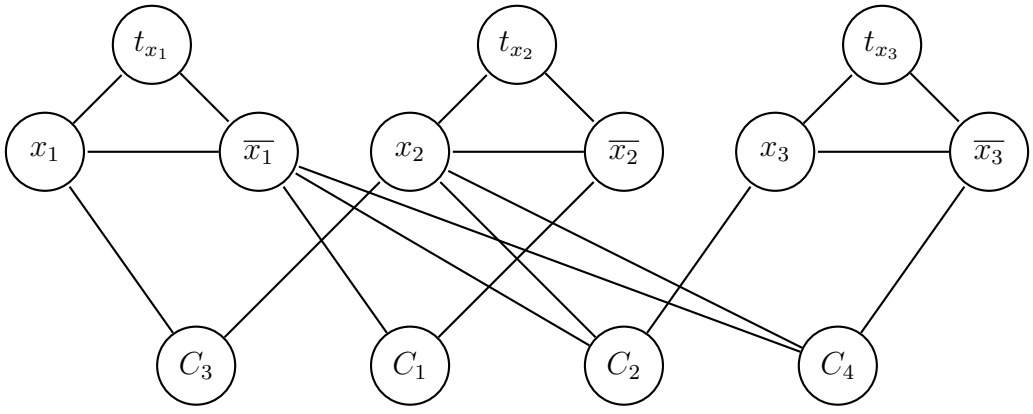
- Viceversa, supponiamo che D' sia un dominant set di massimo m nodi in G. Allora, ogni nodo in D dovrà necessariamente essere un letterale, poiché altrimenti uno dei nodi ausiliari o il nodo di una clausola rimarrebbero scoperti. Inoltre, poiché D ha massimo m nodi, per ogni variabile x solo uno tra i nodi $v_x, v_{\overline{x}}$ sarà presente in D in quanto il numero di variabili definite su ϕ sia m. Consideriamo quindi l'assegnamento α' dove:

$$v_{\ell_{i,k}} \in D' \iff \alpha'(\ell_{i,k}) = \text{True}$$

  Dalla costruzione stessa di G risulta evidente che α' soddisfi ϕ.

- Di conseguenza, abbiamo che $\langle\phi\rangle \in 3SAT \iff f(\langle\phi\rangle) = \langle G, m\rangle \in DOM\text{-}SET$. Inoltre, poiché F svolge solo operazioni eseguibili in tempo polinomiale, concludiamo che $3SAT \leq_m^P DOM\text{-}SET$ e dunque che $DOM\text{-}SET \in$ NP-Complete

□



Grafo generato dalla funzione f della dimostrazione a partire dalla formula $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_3})$

Problema 4.13: NP-Completezza di $BARRILETE$

Dato un grafo G, un *barrilete cosmico* (o *aquilone cosmico*) di dimensione k è un sottografo $B \subseteq G$ composto da una k-clique con un cammino di k nodi connesso ad un solo nodo della clique.

Dato il seguente linguaggio:

$$BARRILETE = \left\{ \langle G, k \rangle \;\middle|\; \begin{array}{l} G = (V_G, E_G) \text{ grafo non diretto con} \\ \text{un barrilete cosmico di dimensione } k \end{array} \right\}$$

dimostrare che $BARRILETE \in$ NP-Complete

Suggerimento: ricordare che abbiamo dimostrato che $SAT, 3SAT$ e $CLIQUE$ siano NP-Completi

Dimostrazione.

- Sia V la NTM definita come:

  V = "Data la stringa $\langle \langle G, k \rangle, c \rangle$ in input, dove $G = (V_G, E_G)$ è un grafo:

  1. Interpreta $c = \langle c_1, \ldots, c_h, p_1, \ldots, p_h \rangle$, dove $c_1, \ldots, c_h, p_1, \ldots, p_h \in V_G$.
  2. Verifica che $h = k$. Se falso, *rifiuta*.
  3. Verifica che $c_k = p_1$. Se falso, *rifiuta*.
  4. Per ogni coppia c_i, c_j con $i \neq j$, verifica se $(c_i, c_j) \in E(G)$. Se falso, *rifiuta*.
  5. Per ogni indice $i = 1, \ldots, h-1$, verifica se $(p_i, p_{i+1}) \in E(G)$. Se falso, *rifiuta*.
  6. *Accetta*."

  Intuitivamente, V è un verificatore polinomiale per $BARRILETE$, dunque otteniamo che $BARRILETE \in$ NP.

- Successivamente, mostriamo che $CLIQUE \leq_m^P BARRILETE$.

- Sia quindi $f : \Sigma^* \to \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

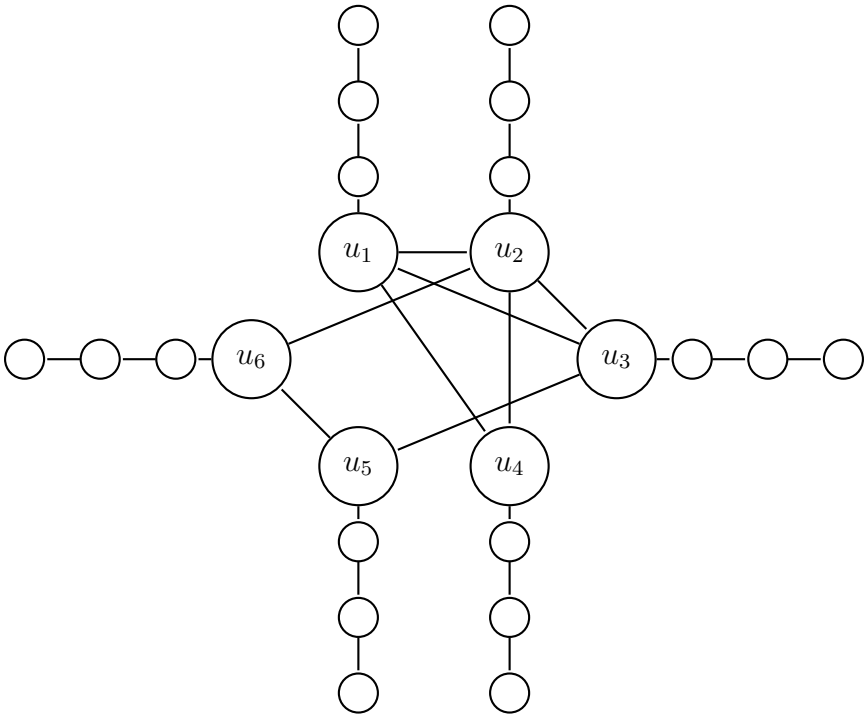
  F = "Data la stringa $\langle G, k \rangle$ in input:

  1. Costruisci il grafo G' definito come:
     i. Copia G.
     ii. Per ogni nodo $u \in V_G$, crea i nodi $v_2, \ldots, v_k$ e crea gli archi $(u, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$.
  2. Restituisci in output la stringa $\langle G', k \rangle$"

- Supponiamo quindi che $\langle G, k \rangle \in CLIQUE$. In tal caso, esiste una k-clique C in G, dunque tramite un qualsiasi cammino $P = u, v_2, \ldots, v_k$ aggiunto in G' otteniamo che $C \cup P$ sia un barrilete cosmico di dimensione k in G', dunque $\langle G', k \rangle \in BARRILETE$.

- Viceversa, se $\langle G', k\rangle \in BARRILETE$ contiene un barrilete cosmico B di dimensione k allora per costruzione stessa di G' la k-clique in B deve per forza essere anche in G, dunque $\langle G, k\rangle \in CLIQUE$.
- Di conseguenza, abbiamo che $\langle G, k\rangle \in CLIQUE \iff f(\langle G, k\rangle) = \langle G', k\rangle \in BARRILETE$. Inoltre, poiché F svolge solo operazioni eseguibili in tempo polinomiale, concludiamo che $CLIQUE \leq_m^P BARRILETE$ e dunque che $BARRILETE \in$ NP-Complete

□



Grafo generato dalla funzione f della dimostrazione.
I nodi $u_1, u_2, u_3, u_4, u_5, u_6$ sono i nodi originali del grafo.

Problema 4.14

Si considerino i seguenti due linguaggi:

$$\text{FACTOR} = \{\langle m, k\rangle \mid \exists p \in \mathbb{P} \text{ tale che } p \mid m \text{ e } p \leq k\}$$

$$\text{PRIMES} = \{\langle n\rangle \mid n \in \mathbb{P}\}$$

Assumendo che PRIMES $\in P$, dimostrare che se FACTOR è NP-Completo allora NP = coNP

Soluzione:

- Per dimostrare il risultato richiesto, è sufficiente dimostrare che FACTOR $\in$ coNP. Per ottenere ciò, è sufficiente definire un verificatore di tempo polinomiale per il linguaggio $\overline{FACTOR}$.

- Sia M la TM che decide il linguaggio PRIMES $= \{\langle n \rangle \mid n \text{ is prime}\}$. Per assunzione del problema, sappiamo che PRIMES $\in$ P (fun fact: questo risultato è in realtà vero, ma non viene coperto nel corso).
- Definiamo prima il nostro verificatore V:

  $V =$ "Su input $\langle \langle m, k \rangle, c \rangle$:

  1. Verifica che $\langle c \rangle = \langle p_1, \ldots, p_t \rangle$ dove $p_1, \ldots, p_t$ sono numero interi non-negativi. Se falso, rifiuta.
  2. Per ogni $i = 1, \ldots, t$:
     3. Verifica che $M(p_i)$, ossia che p_i sia primo (in tempo polinomiale). Se falso, V rifiuta.
     4. Verifica che p_i divida m. Se falso, V rifiuta.
     5. Verifica che $p_i \leq k$ per ogni $i = 1, \ldots, t$. Se falso, V rifiuta.
  6. Verifica che $m = p_1 \cdot \ldots \cdot p_t$. Se falso, V rifiuta
  7. Se tutti i controlli vengono superati, V accetta"
- Intuitivamente, questo verificatore controlla che il certificato in input sia la fattorizzazione di m, assicurandosi che nessun fattore sia minore di k.
- Tuttavia, è importante osservare un fatto cruciale: dobbiamo garantire che il numero t di fattori sia polinomiale rispetto alla lunghezza dell'input $\langle m, k \rangle$, altrimenti non sarà possibile leggere l'intero certificato in tempo polinomiale.
- Osserviamo che il caso peggiore, ossia che m abbia il massimo numero di fattori possibili, si verifica quando $m = 2^t$. Per tanto, abbiamo che $t = \log m$, confermando che t sia polinomiale rispetto alla lunghezza dell'input, il quale è lungo $O(\log m + \log k)$ bit.
- Per tanto, concludiamo che $\overline{\text{FACTOR}} \in \mathsf{NP}$ in quanto tutte le operazioni svolte da V richiedono tempo polinomiale.

Problema 4.15

Sia $L = \{w \in \{a, b\}^* \mid |w|_a = 2\,|w|_b\}$. Si descriva una TM a singolo nastro M_1 che decide L in tempo $O(n^2)$. Inoltre, si descriva una TM multinastro M_2 che decide L in tempo $O(n)$.

Soluzione:

- Definiamo la seguente TM M_1 per la prima richiesta del primo punto:

  $M_1 =$ "Su input w:

  1. Muovi la testina sulla cella più a sinistra
  2. Ripeti se la testina non si trova sul carattere speciale $\sqcup$:

3. Se il carattere sotto la testina è $'a'$, sostituiscilo con il carattere $'x'$ e continua a scorrere a destra finché non vengono trovati due caratteri $'b'$, sostituendoli con il carattere $'y'$. Se i due caratteri vengono trovati, sposta la testina a sinistra fino al primo carattere $'x'$ letto. Altrimenti, rifiuta se viene trovato il carattere $\sqcup$.
4. Se il carattere sotto la testina è $'b'$, sostituiscilo con il carattere $'x'$ e continua a scorrere a destra finché non vengono trovati un carattere $'b'$ e un carattere $'a'$, con il carattere $'y'$. Se i due caratteri vengono trovati, sposta la testina a sinistra fino al primo carattere $'x'$ letto. Altrimenti, rifiuta se viene trovato il carattere $\sqcup$.
5. Accetta"

- È facile vedere che M_1 decida L in tempo $O(n^2)$ in quanto nel caso peggiore ogni simbolo dell'inout richiederà di scorrere tutto l'input
- Per il secondo punto, invece, possiamo sfruttare la presenza di più nastri per contare il numero di $'a'$ e il numero di $'b'$ per poi confrontarli.
- Tuttavia, osserviamo come l'aggiornamento di due contatori binari abbia costo $O(\log n)$, dunque ripetere tale operazione per potenzialmente n caratteri avrebbe costo totale $O(n \log n)$, sfodando il bound richiesto.
- Per risolvere tale problema, utilizziamo due contatori unari. Ad esempio, ogni volta che leggiamo un $'a'$ aggiungiamo un simbolino 1 al contatore unario. A differenza del contatore binario, tale aggiornamento ha costo $O(1)$, rendendo il costo totale $O(n)$.
- Consideriamo quindi la seguente macchina a 3 nastri M_2.

  M_2 = "Su input w:

  1. Muovi la testina sulla cella più a sinistra
  2. Ripeti se la testina del nastro 1 (quello di input) non si trova sul carattere speciale $\sqcup$:
     3. Se il carattere sotto al testina del nastro 1 è $'a'$, scrivi $'1'$ sotto la testina del nastro 2 e sposta entrambe le testine a destra di una cella
     4. Se il carattere sotto al testina del nastro 1 è $'b'$, scrivi $'1'$ sotto la testina del nastro 3 e sposta entrambe le testine a destra di una cella
  5. Sposta le testine del nastro 2 e 3 sulle loro prime celle
  6. Ripeti se la testina del nastro 2 non si trova sul carattere speciale $\sqcup$:
     7. Sposta la testina del nastro 2 di una cella a destra
     8. Sposta la testina del nastro 3 di una cella a destra
     9. Se entrambe le testine si trovano sul carattere speciale $\sqcup$, accetta
  10. Rifiuta"