



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", Michael Sipser

Author
Simone Bianco

29 dicembre 2023

Indice

Informazioni e Contatti	1
1 Linguaggi regolari	2
1.1 Linguaggi	2
1.2 Determinismo	5
1.3 Non determinismo	9
1.3.1 Equivalenza tra NFA e DFA	12
1.4 Chiusure dei linguaggi regolari	15
1.5 Espressioni regolari	20
1.5.1 NFA generalizzati	23
1.5.2 Equivalenza tra espressioni e linguaggi regolari	29
1.6 Pumping lemma per i linguaggi regolari	30
1.7 Esercizi svolti	33
2 Linguaggi acontestuali	41
2.1 Grammatiche acontestuali	41
2.2 Linguaggi acontestuali ad estensione dei regolari	45
2.3 Forma normale di Chomsky	47
2.4 Automi a pila	50
2.4.1 Equivalenza tra CFG e PDA	53
2.5 Pumping lemma per i linguaggi acontestuali	58
2.6 Chiusure dei linguaggi acontestuali	63
2.7 Esercizi svolti	69
3 Calcolabilità	71
3.1 Macchine di Turing	71
3.1.1 Varianti della macchina di Turing	76
3.1.2 Tesi di Church-Turing	81
3.2 Problemi decidibili	82
3.3 Argomento diagonale di Cantor	89
3.3.1 Esistenza di linguaggi non riconoscibili	92
3.4 Problemi indecidibili	93
3.5 Riducibilità	97
3.5.1 Riducibilità tramite mappatura	101
3.6 Esercizi svolti	105

4	Complessità	115
4.1	Complessità temporale	115
4.2	Classe P	118
4.3	Classe NP	121
4.4	Riducibilità in tempo polinomiale	126
4.5	Classe NP-Complete	129
4.5.1	Teorema di Cook-Levin	131
4.5.2	Classi coP, coNP e coEXP	135
4.6	Complessità spaziale	139
4.6.1	Rapporto tra spazio e tempo	142
4.6.2	Teorema di Savitch	143
4.7	Classe PSPACE	145
4.8	Classe L	148
4.9	Riduzione in spazio logaritmico	150
4.10	Classe NL-Complete	152
4.10.1	Teorema di Immerman-Szelepcsényi	153
4.11	Teoremi di gerarchia	156
4.11.1	Teorema di gerarchia di spazio	156
4.11.2	Teorema di gerarchia di tempo	160
4.12	Esercizi svolti	162

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:

These documents are distributed under the [GNU Free Documentation License](#), a form of copyleft intended for use on a manual, textbook or other documents. Material licensed under the current version of the license can be used for any purpose, as long as the use meets certain conditions:

- All previous authors of the work must be **attributed**.
- All changes to the work must be **logged**.
- All derivative works must be **licensed under the same license**.
- The full text of the license, unmodified invariant sections as defined by the author if any, and any other added warranty disclaimers (such as a general disclaimer alerting readers that the document may not be accurate for example) and copyright notices from previous versions must be maintained.
- Technical measures such as DRM may not be used to control or obstruct distribution or editing of the document.

1

Linguaggi regolari

1.1 Linguaggi

Definizione 1: Alfabeto

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

Esempio:

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

Definizione 2: Stringa

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

$$w := w_1 \dots w_n$$

come **stringa** (o **parola**) di Σ

Esempio:

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

Definizione 3: Linguaggio

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

Definizione 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 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:

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

Proposizione 1: Stringa vuota

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

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

Definizione 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 $w := 010101000 \in \{0, 1\}^*$, si ha che $|w|_0 = 6$ e $|w|_1 = 3$

Definizione 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 $w := abcdefg \in \Sigma^*$, si ha che $w^R = gfedcba$

Definizione 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 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: 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)

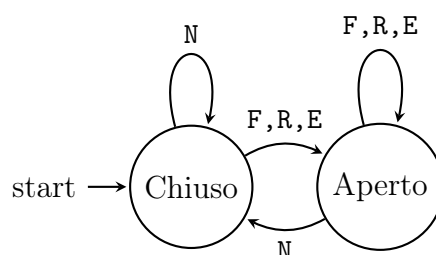
1.2 Determinismo

Definizione 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 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
- $\delta : Q \times \Sigma \rightarrow Q$ 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
- $\delta : Q \times \Sigma \rightarrow Q$ 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 11: Funzione di transizione estesa

Sia $D := (Q, \Sigma, \delta, q_0, F)$ un DFA. Definiamo $\delta^* : Q \times \Sigma^* \rightarrow Q$ 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 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:

$$\begin{aligned} \delta^*(q_1, 0101) &= \delta^*(\delta(q_1, 0), 101) = \delta^*(q_2, 101) = \delta^*(\delta(q_2, 1), 01) = \delta^*(q_2, 01) = \\ &= \delta^*(\delta(q_2, 0), 1) = \delta^*(q_3, 1) = \delta^*(\delta(q_3, 1), \varepsilon) = \delta^*(q_2, \varepsilon) = q_2 \in F \end{aligned}$$

- 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 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 D **riconosce** $L(A)$

Esempi:

- Consideriamo il seguente DFA D



- Il linguaggio riconosciuto da tale DFA corrisponde a

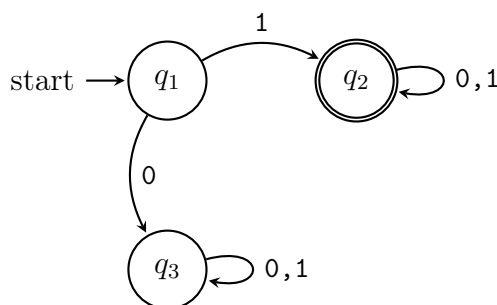
$$L(D) = \{x \in \{0, 1\}^* \mid x := y1, \exists y \in \{0, 1\}^*\}$$

ossia al linguaggio composto da tutte le stringhe terminanti con 1

- 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 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 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 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 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:

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

Osservazione 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 16: Alfabeto epsilon

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

Definizione 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
- $\delta : Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$ 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
- $\delta : Q \times \Sigma \rightarrow Q$ definita come

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

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 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 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:

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

1.3.1 Equivalenza tra NFA e DFA

Definizione 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 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 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:

$$E(R) = \{q \in Q_N \mid q \text{ è raggiungibile in } N \text{ da } q' \in R \text{ tramite } k \geq 0 \text{ } \varepsilon\text{-archi}\}$$

$$- q_{0_D} = E(\{q_{0_N}\})$$

$$- F_D = \{R \in Q_D \mid R \cap F_N \neq \emptyset\}$$

– 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 $L \in \mathcal{L}(\text{DFA})$ e di conseguenza che:

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

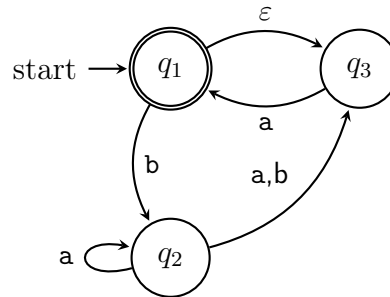
□

Osservazione 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:

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

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

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

• 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:

- $\delta_D(\{q_1\}, a) = E(\delta_N(q_1, a)) = \emptyset$
- $\delta_D(\{q_1\}, b) = E(\delta_N(q_1, b)) = \{q_2\} = q_2$
- $\delta_D(\{q_2\}, a) = E(\delta_N(q_2, a)) = \{q_2, q_3\} = q_{2,3}$
- $\delta_D(\{q_2\}, b) = E(\delta_N(q_2, b)) = \{q_2\} = q_2$
- $\delta_D(\{q_1, q_2\}, a) = E(\delta_N(q_1, a)) \cup E(\delta_N(q_2, a)) = \emptyset \cup \{q_2, q_3\} = \{q_2, q_3\} = q_{2,3}$
- $\delta_D(\{q_1, q_2\}, b) = E(\delta_N(q_1, b)) \cup E(\delta_N(q_2, b)) = \{q_2\} \cup \{q_3\} = \{q_2, q_3\} = q_{2,3}$
- ...

- Il DFA equivalente corrisponde dunque a:



Definizione 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:

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

Osservazione 4

Tramite il teorema dell'[Equivalenza tra NFA e DFA](#), si ha che:

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

1.4 Chiusure dei linguaggi regolari

Teorema 3: Chiusura dell'unione in REG

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

$$\forall L_1, L_2 \in \text{REG} \quad L_1 \cup L_2 \in \text{REG}$$

Dimostrazione I.

- Dati $L_1, L_2 \in \text{REG}$, 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 $L_1 \cup L_2 = L(D) \in \text{REG}$

□

Dimostrazione II.

- Dati $L_1, L_2 \in \text{REG}$, 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:

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

- 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 $L_1 \cup L_2 = L(N) \in \text{REG}$

□



Rappresentazione grafica della dimostrazione

Teorema 4: Chiusura dell'intersezione in REG

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

$$\forall L_1, L_2 \in \text{REG} \quad L_1 \cap L_2 \in \text{REG}$$

Dimostrazione.

- Dati $L_1, L_2 \in \text{REG}$, 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 $L_1 \cap L_2 = L(D) \in \text{REG}$

□

Teorema 5: Chiusura del complemento in REG

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

$$\forall L \in \text{REG} \quad \bar{L} \in \text{REG}$$

Dimostrazione.

- Dato $L \in \text{REG}$, 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 $\bar{L} = L(D') \in \text{REG}$

□

Teorema 6: Chiusura della concatenazione in REG

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

$$\forall L_1, L_2 \in \text{REG} \quad L_1 \circ L_2 \in \text{REG}$$

Dimostrazione.

- Dati $L_1, L_2 \in \text{REG}$, 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 $L_1 \circ L_2 = L(N) \in \text{REG}$

□



Rappresentazione grafica della dimostrazione

Corollario 1: Chiusura della potenza in REG

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

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

Teorema 7: Chiusura di star in REG

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

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

Dimostrazione.

- Dato $L \in \text{REG}$, 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:

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

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

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

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

□



Rappresentazione grafica della dimostrazione

Corollario 2: Chiusura di plus in REG

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

$$\forall L \in \text{REG} \quad L^+ \in \text{REG}$$

Dimostrazione.

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

□

1.5 Espressioni regolari

Definizione 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:

- $\emptyset \in \text{re}(\Sigma)$
- $\varepsilon \in \text{re}(\Sigma)$
- $a \in \text{re}(\Sigma)$, 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 5

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

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

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. \emptyset^* rappresenta il linguaggio $\emptyset^* = \{\varepsilon\}$ (ricordiamo che per definizione stessa si ha che $\forall L \subseteq \Sigma^* \quad L^0 = \{\varepsilon\}$)

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

Definizione 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: Conversione da espressione regolare a NFA

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

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

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



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

- 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 $L(R) = L(N_\varepsilon) \in \mathcal{L}(\text{NFA})$

- 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 $L(R) = L(N_a) \in \mathcal{L}(\text{NFA})$

Ipotesi induttiva.

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

Passo induttivo.

- Se $R = R_1 \cup R_2$, tramite la **Chiusura dell'unione in REG**, otteniamo che:

$$L(R) = L(R_1) \cup L(R_2) = L(N_1) \cup L(N_2) \in \text{REG} = \mathcal{L}(\text{NFA})$$

- Se $R = R_1 \circ R_2$, tramite la **Chiusura della concatenazione in REG**, otteniamo che:

$$L(R) = L(R_1) \circ L(R_2) = L(N_1) \circ L(N_2) \in \text{REG} = \mathcal{L}(\text{NFA})$$

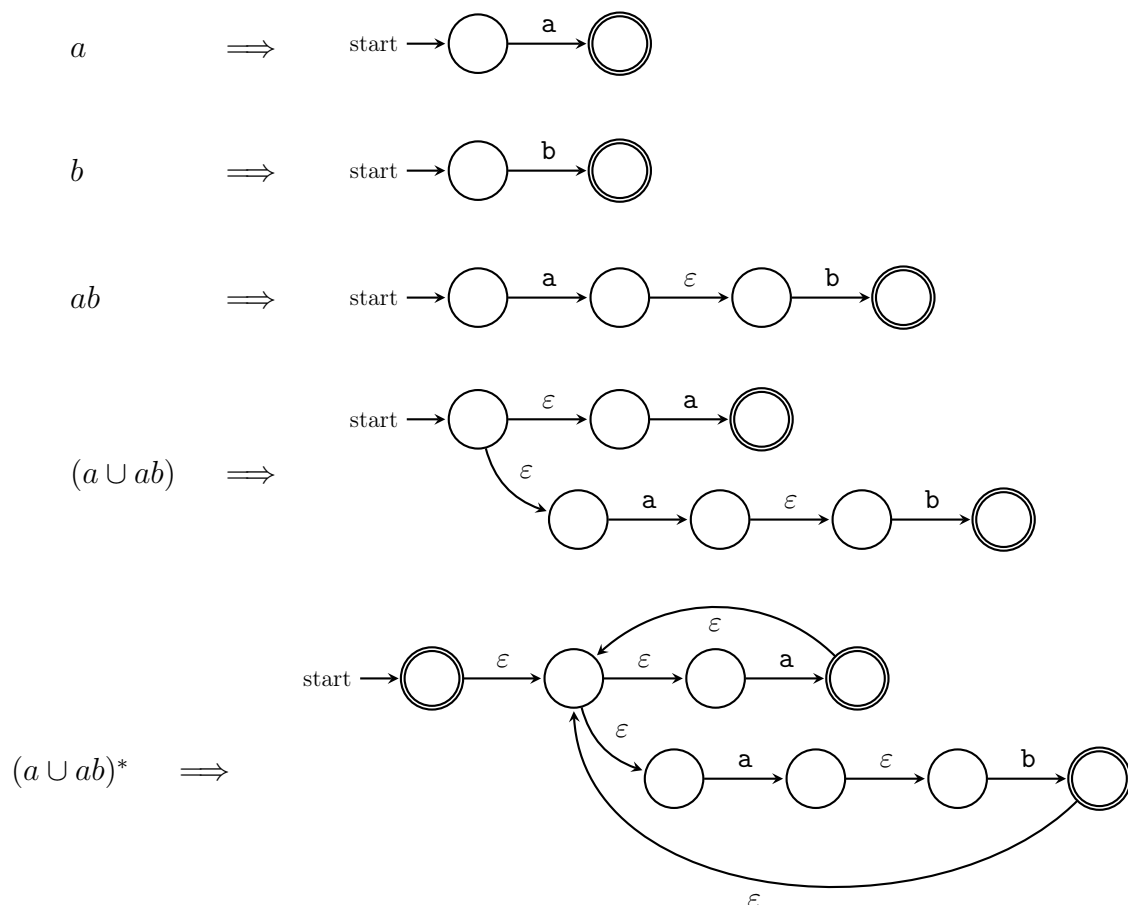
- Se $R = R_1^*$, tramite la **Chiusura di plus in REG**, otteniamo che:

$$L(R) = L(R_1)^* = L(N_1)^* \in \text{REG} = \mathcal{L}(\text{NFA})$$

1

Esempio:

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



1.5.1 NFA generalizzati

Definizione 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
- $\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow \text{re}(\Sigma)$ 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 $q \rightarrow q'$ ed una transizione $q' \rightarrow q$
 - Le "etichette" delle transizioni sono delle **espressioni regolari**

Esempio:

**Osservazione 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 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:

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

Esempio:

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

Corollario 3

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

Definizione 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 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:
 - $Q' = Q \cup \{q_{\text{start}}, q_{\text{accept}}\}$
 - $\delta'(q_{\text{start}}, q_0) = \varepsilon$
 - $\forall q \in F \quad \delta'(q, q_{\text{accept}}) = \varepsilon$

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

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

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

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

□

Esempio:

- Consideriamo il seguente DFA:



- Il suo GNFA equivalente corrisponde a:



Algoritmo 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, \dots, 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 $L(G) = L(G_0) = \dots = L(G_n)$

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 $w := w_0 \dots w_m \in L(G_k)$, dove $w_0, \dots, w_m \in \Sigma^*$, esiste una sequenza di stati $q_0, \dots, q_m \in Q$ tali che:

- $q_0 = q_{\text{start}}$ e $q_m = q_{\text{accept}}$
- $\forall i \in [0, m-1] \quad w_i \in L(\delta(q_i, q_{i+1}))$

- 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 $q \notin \{q_0, \dots, q_m\}$, 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 $q \in \{q_0, \dots, q_m\}$, 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:

$$\begin{aligned}\delta'(q_{\text{start}}, q_1) &= \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0, q_1) \cup \delta(q_{\text{start}}, q_1) = \varepsilon(0 \cup 1)^*2 \cup \emptyset = (0 \cup 1)^*2 \\ \delta'(q_{\text{start}}, q_{\text{accept}}) &= \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0, q_{\text{accept}}) \cup \delta(q_{\text{start}}, q_{\text{accept}}) = \varepsilon(0 \cup 1)^*\emptyset \cup \emptyset = \emptyset \\ \delta'(q_1, q_1) &= \delta(q_1, q_0)\delta(q_0, q_0)^*\delta(q_0, q_1) \cup \delta(q_1, q_1) = \emptyset(0 \cup 1)^*2 \cup (0 \cup 1) = 0 \cup 1 \\ \delta'(q_1, q_{\text{accept}}) &= \delta(q_1, q_0)\delta(q_0, q_0)^*\delta(q_0, q_{\text{accept}}) \cup \delta(q_1, q_{\text{accept}}) = \emptyset(0 \cup 1)^*\emptyset \cup \varepsilon = \varepsilon\end{aligned}$$



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

$$\begin{aligned}\delta''(q_{\text{start}}, q_{\text{accept}}) &= \delta'(q_{\text{start}}, q_1)\delta'(q_1, q_1)^*\delta'(q_1, q_{\text{accept}}) \cup \delta'(q_{\text{start}}, q_{\text{accept}}) = \\ &= (0 \cup 1)^*2(0 \cup 1)^*\varepsilon \cup \emptyset = (0 \cup 1)^*2(0 \cup 1)^*\end{aligned}$$

- Il GNFA minimale, dunque, corrisponde a:



Corollario 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 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 7: Classi 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 3: Pumping lemma per i linguaggi regolari

Dato un linguaggio L , se $L \in \text{REG}$ 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:

- $\forall i \in \mathbb{N} \quad xy^i z \in L$, 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 $L \in \text{REG}$, 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 :

$$\forall k \in [1, n] \quad \delta(r_k, w_k) = r_{k+1}$$

- 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

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

- 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 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 $0000 \dots 101010 \dots 1111$
- 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 $xy^0 z$. Notiamo immediatamente che

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

implicando dunque che $xy^0 z \notin L$, contraddicendo la prima condizione del lemma per cui si ha che $\forall i \in \mathbb{N} \quad xy^i z \in L$

- Dunque, ne segue necessariamente che L non sia regolare

1.7 Esercizi svolti

Problema 1: Linguaggio rovesciato

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

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

Dimostrazione.

- Dato $L \in \text{REG}$, 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
 - $Q' = Q \cup \{q_f\}$
 - $\forall q \in Q, a \in \Sigma \quad \delta'(q, a) = \delta(q, a)$, ossia tutti gli archi rimangono invariati
 - $\forall q \in F \quad \delta'(q, \varepsilon) = q_f$, 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 $N^R = (Q', \Sigma, \delta'', q_f, \{q_0\})$ tale che:

$$\forall p, q \in Q', a \in \Sigma \quad \delta'(p, a) = q \implies \delta''(q, a) = p$$

ossia avente tutti gli archi invertiti rispetto ad N

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

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

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

□

Problema 2: 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 3**

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

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 $w := 0^p 1^p \in L$. 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 $w := 0^p 1^p$, 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 $xy^0 z$. Notiamo immediatamente che

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

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

contraddicendo la prima condizione del lemma per cui si ha che $\forall i \in \mathbb{N} \quad xy^i z \in L$

- Dunque, ne segue necessariamente che L non sia regolare

□

Problema 4

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

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 $xy^0 z$. Notiamo immediatamente che

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

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

□

Problema 5

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

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 $\forall i \in \mathbb{N} \ xy^i z \in L$

- Dunque, ne segue necessariamente che L non sia regolare

□

Problema 6

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.

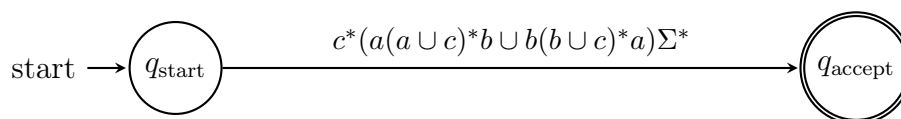
Soluzione:

- 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 $c^*(a(a \cup c)^*b \cup b(a \cup c)^*a)\Sigma^*$

- 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 \emptyset), per poi procedere con la sua riduzione:



Problema 7

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 8: Somma di cifre come multiplo di k

Sia L_k il linguaggio di tutte le stringhe su alfabeto $\Sigma = \{0, 1, \dots, 9\}$ 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:

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

- Sia $D_k = (Q, \Sigma, \delta, q_0, F)$ il DFA definito come:

- $Q = \{q_0, \dots, q_{k-1}\}$
- $F = \{q_0\}$
- $\forall q_i, q_j \in Q$ 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 $w = a_1 \dots a_n \in \Sigma^*$ si ha che:

$$\delta^*(q_0, w) = q_h \iff a_1 + \dots + a_n \equiv h \pmod{k}$$

Passo induttivo:

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

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

- Per ipotesi induttiva, si ha che:

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

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:

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

- A questo punto, si ha che:

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

$$\Longleftrightarrow a_1 + \dots + a_n \equiv 0 \pmod{k} \Longleftrightarrow w \in L$$

implicando quindi che $L_k = L(D_k)$

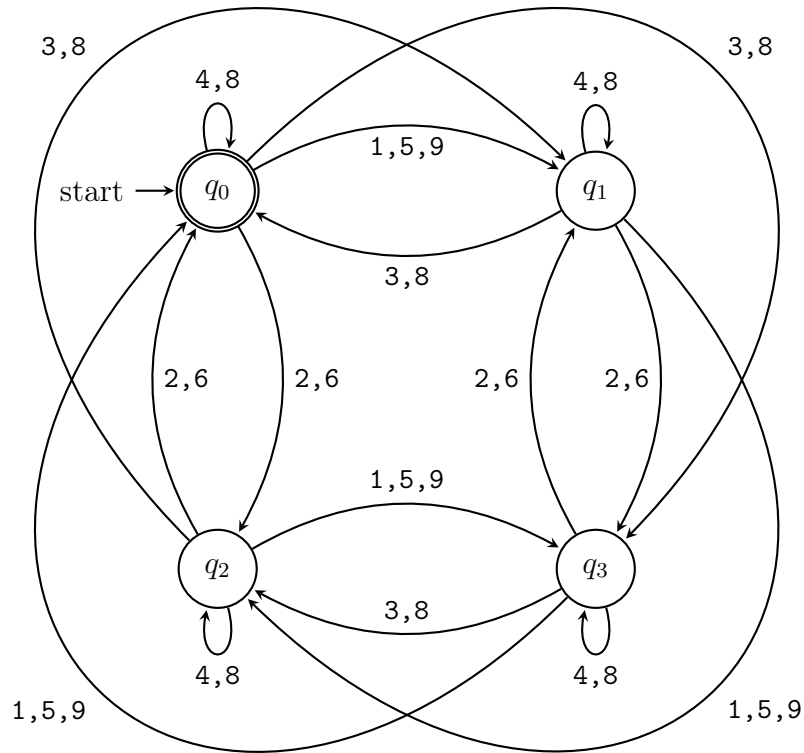


diagramma del DFA D_4 dettato dalla dimostrazione

Linguaggi acontestuali

2.1 Grammatiche acontestuali

Definizione 25: 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
- $V \cap \Sigma = \emptyset$, ossia variabili e terminali sono tutti distinti tra loro

Le **regole** in R assumono la forma $A \rightarrow X$, dove $A \in V$, ossia è una variabile, e $X \in (V \cup \Sigma_\epsilon)^*$, 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:

$$A \rightarrow 0A1$$

$$A \rightarrow B$$

$$B \rightarrow \#$$

Osservazione 8: 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 $A \rightarrow B$ può appartenere ad una CFG
- La regola $AB \rightarrow B$ non può appartenere ad una CFG

Osservazione 9: Notazione contratta per le regole

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

$$A \rightarrow X_1 \mid X_2 \mid \dots \mid X_n$$

Esempio:

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

$$A \rightarrow 0A1 \mid B$$

$$B \rightarrow \#$$

Definizione 26: Produzione

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

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

Esempio:

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

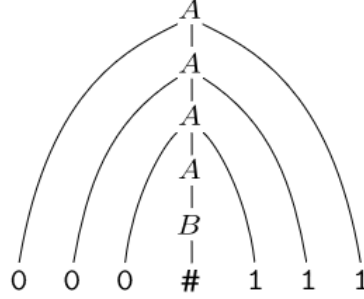
$$A \rightarrow 0A1 \mid B$$

$$B \rightarrow \#$$

- 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 27: Derivazione

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

$$u \Rightarrow u_1 \Rightarrow \dots \Rightarrow u_k \Rightarrow v$$

Definizione 28: 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 :

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

Esempi:

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

$$S \rightarrow \varepsilon \mid aSb \mid SS$$

si ha che:

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

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

$$S \rightarrow T1T1T1T$$

$$T \rightarrow \varepsilon \mid 0T \mid 1T$$

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:

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

si ha che:

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

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

$$S \rightarrow aSc \mid T$$

$$T \rightarrow bTc \mid \varepsilon$$

si ha che:

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

Osservazione 10

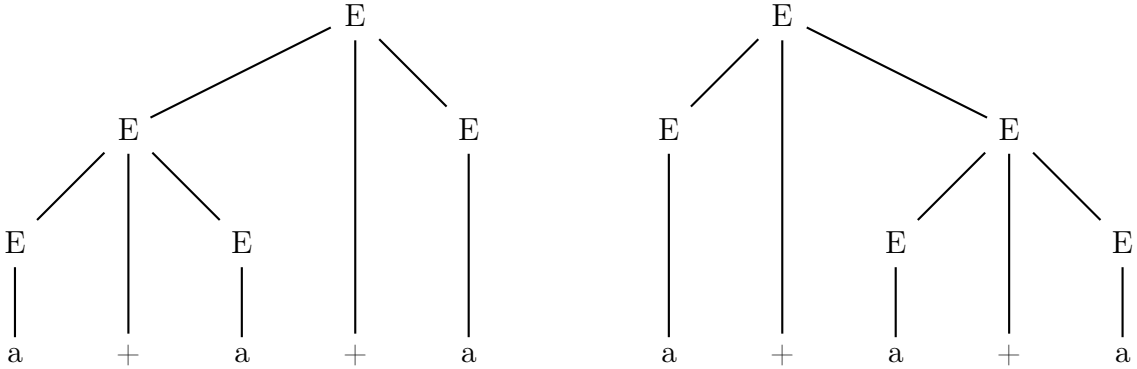
Sia G una CFG. Data la stringa $w \in L(G)$, possono esistere più derivazioni di w

Esempio:

- Data la CFG

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

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



Definizione 29: Derivazione a sinistra

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

Esempio:

- Riprendiamo la CFG dell'esempio precedente:

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

- Per maggior chiarezza, riscriviamo tali regole come:

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

$$F \rightarrow E$$

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 11

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 30: 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 31: Classe dei linguaggi acontestuali

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

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

Lemma 4: Conversione da DFA a CFG

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

$$\text{REG} \subseteq \text{CFL}$$

Dimostrazione.

- Dato $L \in \text{REG}$, 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 $\varphi : Q \rightarrow V : q_i \mapsto V_i$

$$- S = \varphi(q_0) = V_0$$

- Dati $q_i, q_j \in Q$ e $a \in \Sigma$, si ha che:

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

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

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

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

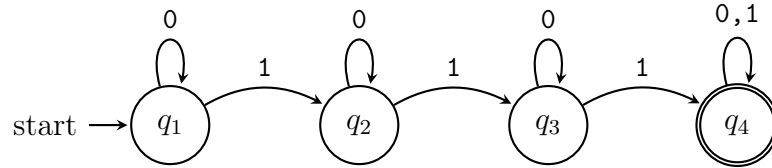
implicando dunque che $L(D) \in \text{CFL}$ e di conseguenza che:

$$\text{REG} \subseteq \text{CFL}$$

□

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:

$$V_1 \rightarrow 0V_1 \mid 1V_2$$

$$V_2 \rightarrow 0V_2 \mid 1V_3$$

$$V_3 \rightarrow 0V_3 \mid 1V_4$$

$$V_4 \rightarrow 0V_4 \mid 1V_4 \mid \varepsilon$$

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

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

Teorema 9: Ling. acontestuali estensione dei ling. regolari

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

$$\text{REG} \subsetneq \text{CFL}$$

Dimostrazione.

- Tramite la [Conversione da DFA a CFG](#), sappiamo che $\text{REG} \subseteq \text{CFL}$
- 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:

$$S \rightarrow 0S1 \mid \varepsilon$$

dunque abbiamo che $L = L(G) \in \mathcal{L}(\text{CFG})$

- Tuttavia, abbiamo già dimostrato nella sezione [1.6](#) che L non sia regolare, dunque abbiamo che $L \notin \text{REG}$
- Di conseguenza, concludiamo che:

$$\text{REG} \subsetneq \text{CFL}$$

□

2.3 Forma normale di Chomsky

Definizione 32: 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:

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

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

Teorema 10: 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 $S_0 \rightarrow S$, dove S_0 è la **nuova variabile iniziale**
 2. Finché in R esiste una ε -**regola** $A \rightarrow \varepsilon$ 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 $A \rightarrow \varepsilon$ e in R esiste $B \rightarrow uAvAw \mid u, v, w \in (V \cup \Sigma)^*$, vengono aggiunte le regole $B \rightarrow uvAw \mid uAvw \mid uvw$)
 3. Ogni regola nella forma $A \rightarrow B$ (dette **regole unitarie**) per cui esiste una regola nella forma $B \rightarrow u \mid u \in (V \cup \Sigma)^*$ viene **sostituita** con la regola $A \rightarrow u$
 4. Per ogni regola $A \rightarrow u_1 \dots u_k$ dove $k \geq 3$ e $u \in (V \cup \Sigma)$, vengono **aggiunte** le variabili A_1, \dots, A_k e le seguenti regole:

$$A \rightarrow u_1 A_1 \quad \dots \quad A_{k-3} \rightarrow u_{k-2} A_{k-2} \quad A_{k-2} \rightarrow u_{k-1} u_k$$

per poi eliminare la regola iniziale $A \rightarrow u_1 u_2 \dots u_k$

5. Per ogni regola rimanente nella forma $A \rightarrow u_1 u_2 \mid u_1, u_2 \in (V \cup \Sigma)$, se $u_1 \in \Sigma$ allora viene aggiunta una variabile U_1 ed una regola $U_1 \rightarrow u_1$, sostituendo la regola $A \rightarrow u_1 u_2$ con la regola $A \rightarrow U_1 u_2$. 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:

$$\begin{aligned} G: \quad S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

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

$$\begin{aligned} G: \quad S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \\ A &\rightarrow B \mid S \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

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

$$\begin{aligned} G : S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \mid \mathbf{a} \\ A &\rightarrow B \mid S \mid \varepsilon \\ B &\rightarrow b \mid \varepsilon \end{aligned}$$

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

$$\begin{aligned} G : S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \mid a \mid \mathbf{SA} \mid \mathbf{AS} \mid S \\ A &\rightarrow B \mid S \mid \varepsilon \\ B &\rightarrow b \end{aligned}$$

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

$$\begin{aligned} G : S_0 &\rightarrow S \\ S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid S \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

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

$$\begin{aligned} G : S_0 &\rightarrow S \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\ S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ A &\rightarrow B \mid S \\ B &\rightarrow b \end{aligned}$$

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

$$\begin{aligned} G : S_0 &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\ A &\rightarrow B \mid S \mid \mathbf{b} \mid \mathbf{ASA} \mid \mathbf{aB} \mid \mathbf{a} \mid \mathbf{SA} \mid \mathbf{AS} \\ B &\rightarrow b \end{aligned}$$

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

$$\begin{aligned} G : S_0 &\rightarrow ASA \mid \mathbf{AA_1} \mid aB \mid a \mid SA \mid AS \\ S &\rightarrow ASA \mid \mathbf{AA_1} \mid aB \mid a \mid SA \mid AS \\ A &\rightarrow b \mid ASA \mid \mathbf{AA_1} \mid aB \mid a \mid SA \mid AS \\ \mathbf{A_1} &\rightarrow \mathbf{SA} \\ B &\rightarrow b \end{aligned}$$

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

$$\begin{aligned}
G: S_0 &\rightarrow AA_1 \mid \mathbf{aB} \mid \mathbf{UB} \mid a \mid SA \mid AS \\
S &\rightarrow AA_1 \mid \mathbf{aB} \mid \mathbf{UB} \mid a \mid SA \mid AS \\
A &\rightarrow b \mid AA_1 \mid \mathbf{aB} \mid \mathbf{UB} \mid a \mid SA \mid AS \\
A_1 &\rightarrow SA \\
U &\rightarrow \mathbf{a} \\
B &\rightarrow b
\end{aligned}$$

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

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

2.4 Automi a pila

Definizione 33: 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
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \rightarrow \mathcal{P}(Q \times \Gamma_\epsilon)$ 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 $a; b \rightarrow c$

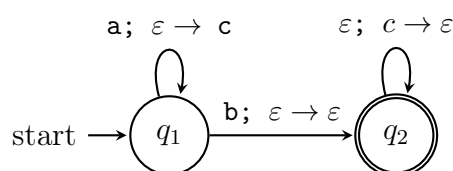
Osservazione 12

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 $a; \varepsilon \rightarrow \varepsilon$) 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 $a; \varepsilon \rightarrow c$) 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 $a; b \rightarrow \varepsilon$) 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 8: Stringa accettata in un PDA

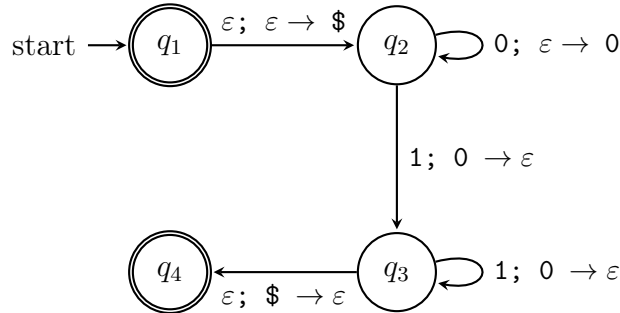
Sia $P := (Q, \Sigma, \Gamma, \delta, q_0, F)$ un PDA. Data una stringa $w := w_0 \dots w_k \in \Sigma^*$, dove $w_0, \dots, w_k \in \Sigma_\varepsilon$, diciamo che w è **accettata da P** se esiste una sequenza di stati $r_0, r_1, \dots, r_{k+1} \in Q$ ed una sequenza di stringhe $s_1, \dots, s_n \in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

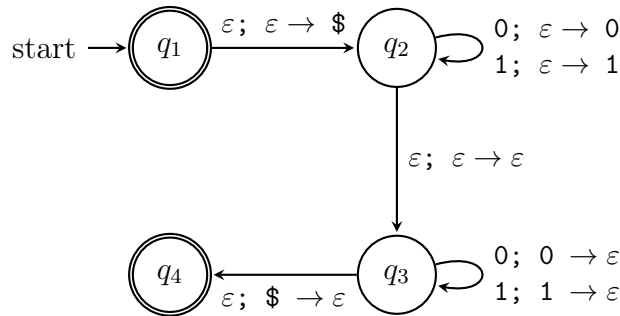
dove $a, b \in \Gamma_\varepsilon$ e dove $t \in \Gamma^*$ è la stringa composta dai caratteri nello stack

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 34: 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 9: 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 $u := u_1 \dots u_k$.

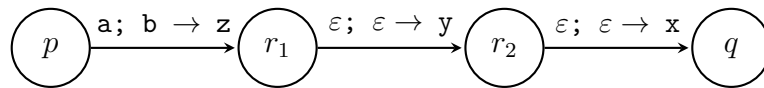
Formalmente, diciamo che:

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

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

Esempio:

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



Lemma 5: Conversione da CFG a PDA

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

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

Dimostrazione.

- Dato $L \in \text{CFL}$, 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

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

– $\forall a \in \Sigma$ 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 $L = L(P) \in \mathcal{L}(\text{PDA})$

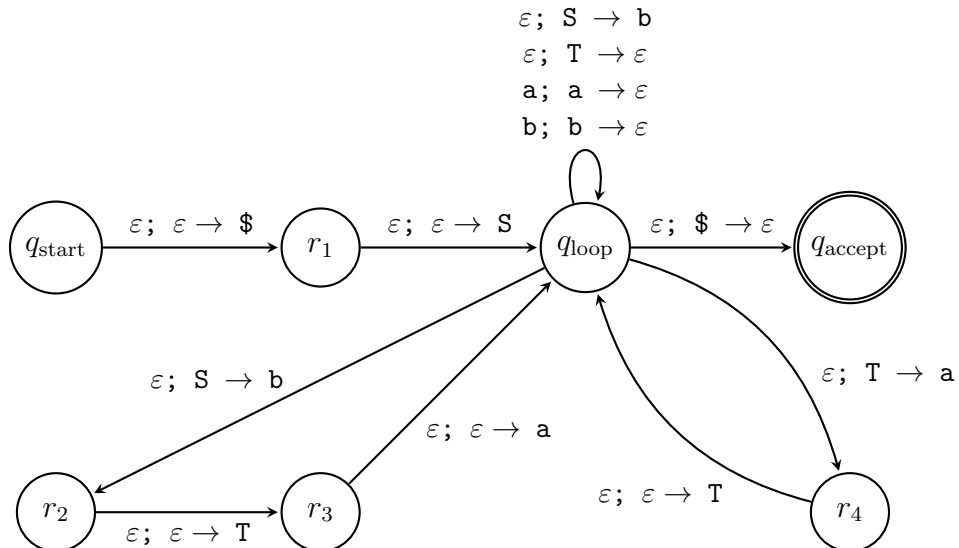
□

Esempio:

- Consideriamo la seguente grammatica:

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

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



Lemma 6: Conversione da PDA a CFG

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

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

Dimostrazione.

- Dato $L \in \mathcal{L}(\text{PDA})$, 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:

$$(q, c) \in \delta(p, a, b) \implies \exists r \in Q' \mid (r, \varepsilon) \in \delta'(p, a, b) \wedge \delta'(r, \varepsilon, \varepsilon) = \{(q, c)\}$$

- $Q' = Q \cup Q_{\delta'} \cup \{q_{\text{accept}}\}$, dove $Q_{\delta'}$ sono gli stati aggiunti per il punto precedente
- $q_{\text{accept}} \in Q'$ è il nuovo unico stato accettante:

$$\forall q \in F \quad (q_{\text{accept}}, \varepsilon) \in \delta'(q, \varepsilon, \varepsilon)$$

- Lo stack deve essere svuotato prima di poter accettare una stringa:

$$\forall q \in F, a \in \Sigma \quad (q, \varepsilon) \in \delta'(q, \varepsilon, a)$$

- 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:

$$(A_{p,p} \rightarrow \varepsilon) \in R$$

- * $\forall p, q, r, s \in Q', u \in \Gamma$ e $a, b \in \Sigma_\varepsilon$ si ha che:

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

- * $\forall p, q, r \in Q'$ si ha che:

$$(A_{p,q} \rightarrow A_{p,r}A_{r,q}) \in R$$

- **Affermazione:** dati $p, q \in Q'$ e $x \in \Sigma^*$, se $A_{p,q} \xRightarrow{*} x$ 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 $A_{p,q} \xRightarrow{*} x$

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

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

Sia $x = ayb$, dove $A_{r,s} \xRightarrow{*} y$. Poiché $A_{r,s} \xRightarrow{*} y$ 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:

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

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 $A_{p,q} \Rightarrow A_{p,r}A_{r,q} \xRightarrow{*} x$.

Sia $x = yz$, dove $A_{p,r} \xRightarrow{*} y$ e $A_{r,q} \xRightarrow{*} z$. Poiché $A_{p,r} \xRightarrow{*} y$ è composta da $m \leq n$ produzioni e $A_{r,q} \xRightarrow{*} z$ 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 $A_{p,q} \xRightarrow{*} x$

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 $A_{p,p} \rightarrow \varepsilon$ 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 $A_{p,q} \xRightarrow{*} x$

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 $A_{r,s} \xRightarrow{*} y$, dunque data la regola $A_{p,q} \rightarrow aA_{r,s}b$ concludiamo che:

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

- (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 $A_{p,r} \xRightarrow{*} y$ e $A_{r,q} \xRightarrow{*} z$, dunque data la regola $A_{p,q} \rightarrow A_{p,r}A_{r,q}$ concludiamo che:

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

- Tramite le due affermazioni, abbiamo che:

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

da cui concludiamo che:

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

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

□

Teorema 11: Equivalenza tra CFG e PDA

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

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

(segue dai due lemmi precedenti)

2.5 Pumping lemma per i linguaggi acontestuali

Proposizione 10: 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 $S \xRightarrow{*} x$

Caso base.

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

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 $S \rightarrow AB$.

- Sia quindi $x = yz$, dove $A \xRightarrow{*} y$ e $B \xRightarrow{*} z$. Poiché la derivazione $S \Rightarrow AB \xRightarrow{*} yz = x$ ha altezza $h + 1$, ne segue che l'altezza dei due sottoalberi delle derivazioni $A \xRightarrow{*} y$ e $B \xRightarrow{*} z$ 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 7: Pumping lemma per i linguaggi acontestuali

Dato un linguaggio L , se $L \in \text{CFL}$ 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:

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

Dimostrazione.

- Dato $L \in \text{CFL}$, 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, \dots, 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:
 - $w = uvxyz$
 - $S \xRightarrow{*} uA_iz$
 - $A_i \xRightarrow{*} vA_jy$
 - $A_j \xRightarrow{*} x$

- Poiché $A_i = A_j$, all'interno di ogni derivazione $A_i \xRightarrow{*} vA_jy$ possiamo sostituire A_j con A_i stesso. Ripetendo tale procedimento $i \in \mathbb{N}$ volte ricorsivamente, otteniamo che:

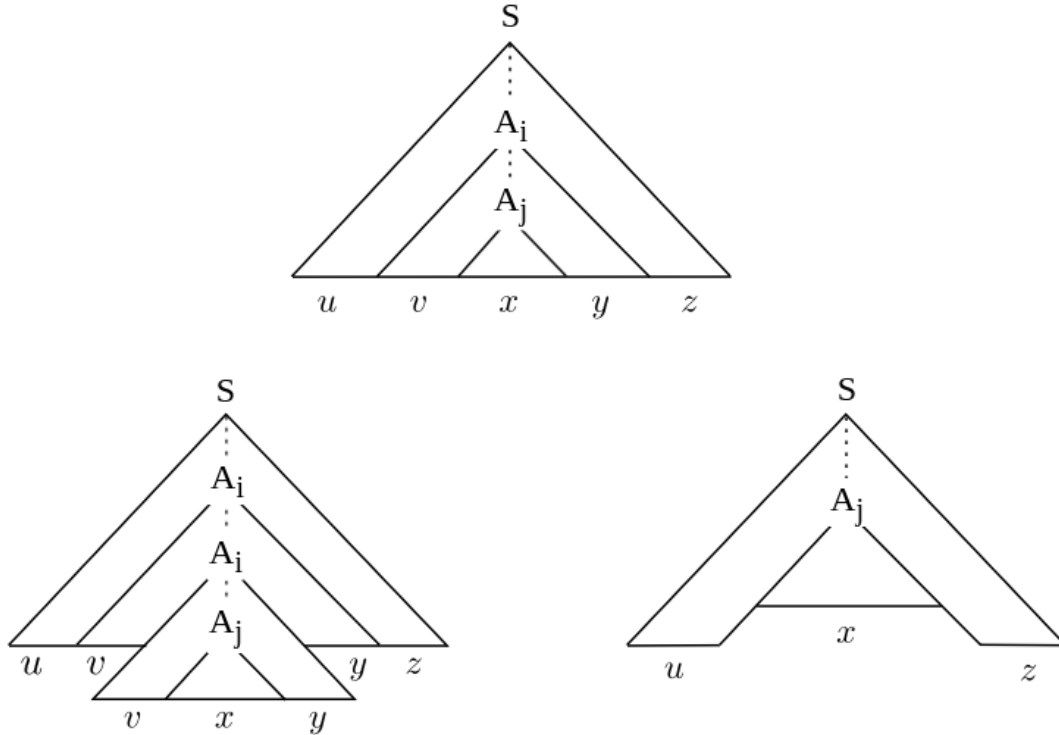
$$A_i \xRightarrow{*} vA_jy = vA_iy \xRightarrow{*} v^iA_jy^i \Rightarrow v^ixy^i$$

implicando dunque che $\forall i \in \mathbb{N} \ S \xRightarrow{*} uv^ixy^iz$ 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 $A_i \xRightarrow{*} vA_jy$ deve necessariamente aver utilizzato una regola del tipo $A_i \rightarrow BC$ dove $B \xRightarrow{*} vA_j$ e $C \xRightarrow{*} y$ oppure $B \xRightarrow{*} v$ e $C \xRightarrow{*} A_jy$. 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 $L \in \text{CFL}$. 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 $1 \leq m+h \leq p$. Inoltre, poiché la seconda condizione impone che $|vy| > 0$, si ha che v e/o y contengono almeno un 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

$$\nexists n \in \mathbb{N} \mid n = |uv^0xy^0z|_0 = |uv^0xy^0z|_1 = |uv^0xy^0z|_2 \implies uv^0xy^0z \notin L$$

contraddicendo quindi la prima condizione del pumping lemma

- Di conseguenza, ne segue necessariamente che $L \notin \text{CFL}$
2.
 - Consideriamo il linguaggio $L = \{ww \mid w \in \{0,1\}^*\}$
 - Supponiamo per assurdo che $L \in \text{CFL}$. 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 $L \notin \text{CFL}$

2.6 Chiusure dei linguaggi acontestuali

Teorema 12: Chiusura dell'unione in CFL

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

$$\forall L_1, \dots, L_n \in \text{CFL} \quad L_1 \cup \dots \cup L_n \in \text{CFL}$$

Dimostrazione.

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

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

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

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

$$w \in L(G) \implies \exists j \in [1, n] \mid S \Rightarrow S_j \xRightarrow{*} w \implies w \in L(G_j) \subseteq \bigcup_{i=1}^n L(G_i)$$

- Di conseguenza, concludiamo che:

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

□

Teorema 13: Chiusura della concatenazione in CFL

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

$$\forall L_1, \dots, L_n \in \text{CFL} \quad L_1 \circ \dots \circ L_n \in \text{CFL}$$

Dimostrazione.

- Dati $L_1, \dots, L_n \in \text{CFL}$, siano G_1, \dots, G_n le tali che $\forall i \in [1, n] \quad G_i = (V_i, \Sigma_i, R_i, S_i) \wedge L_i = L(G_i)$.
- Consideriamo quindi la CFG $G = (V, \Sigma, R, S)$ tale che:

– S è una nuova variabile iniziale

$$– V = \left(\bigcup_{i=1}^n V_i \right) \cup \{S\}$$

$$– \Sigma = \bigcup_{i=1}^n \Sigma_i$$

$$– R = \left(\bigcup_{i=1}^n R_i \right) \cup \{S \rightarrow S_1 \dots S_n\}$$

- Sia $w := w_1 \dots w_n \in L(G_1) \circ \dots \circ L(G_n)$, dove $\forall j \in [1, n] \quad w_j \in L(G_j)$

Poiché $(S \rightarrow S_1 \dots S_n) \in R$, ne segue che

$$\forall j \in [1, n] \quad w_j \in L(G_j) \iff S_j \xRightarrow{*} w_j$$

dunque abbiamo che:

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

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

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

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

$$S \Rightarrow S_1 \dots S_n \xRightarrow{*} w_1 S_2 \dots S_n \xRightarrow{*} w_1 w_2 \dots w_n = w$$

implicando che:

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

- Di conseguenza, concludiamo che:

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

□

Esempio:

- Consideriamo i seguenti linguaggi:

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

- Consideriamo quindi le due grammatiche:

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

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

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:

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

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

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

Teorema 14: Chiusura di star in CFL

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

$$\forall L \in \text{CFL} \quad L^* \in \text{CFL}$$

Dimostrazione.

- Dato $L \in \text{CFL}$, 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
 - $R' = R \cup \{S_0 \rightarrow \varepsilon, S_0 \rightarrow S, S_0 \rightarrow S_0 S_0\}$
- Data $w := w_1 \dots w_n \in L^*$, abbiamo che:
 - Se $w = \varepsilon$, poiché $(S_0 \rightarrow \varepsilon) \in R$, ne segue che

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

– Se $w \neq \varepsilon$, invece, si ha che $\forall j \in [1, n] \ w_j \in L = L(G) \iff S \xRightarrow{*} w_j$. Dunque si ha che:

* Se $n = 1$, dunque $w = w_1$, tramite la regola $(S_0 \rightarrow S) \in R$ ne segue che:

$$S_0 \Rightarrow S \xRightarrow{*} w_1 = w \implies w \in L(G')$$

* Se invece $n > 1$, tramite $(S_0 \Rightarrow S_0 S_0) \in R$ ne segue che:

$$S_0 \Rightarrow S_0 S_0 \xRightarrow{*} S_0^n \xRightarrow{*} S^n \xRightarrow{*} w_1 \dots w_n = w \implies w \in L(G')$$

• Data $w \in L(G')$, dove $w \in L(G') \iff S_0 \xRightarrow{*} w$, poiché le uniche regole applicabili su S_0 sono $\{S_0 \rightarrow \varepsilon, S_0 \rightarrow S, S_0 \rightarrow SS\}$, ne segue necessariamente che:

– Se $S_0 \Rightarrow \varepsilon = w$, ne segue direttamente che $w = \varepsilon \in L^0$

– Se $S_0 \Rightarrow S \xRightarrow{*} w$, ne segue direttamente che $w \in L(G) = L^1$

– Se $S_0 \Rightarrow S_0 S_0 \xRightarrow{*} w$, dato $n \geq 2$ si ha che:

$$S_0 \Rightarrow S_0 S_0 \xRightarrow{*} S_0^n \xRightarrow{*} S^n$$

Siano quindi $w_1, \dots, w_n \in L(G) = L$. Poiché $\forall j \in [1, n] \ w_j \in L(G) = L \iff S \xRightarrow{*} w_j$, ne segue automaticamente che:

$$S_0 \xRightarrow{*} S^n \xRightarrow{*} w_1 \dots w_n = w \implies w \in L^n$$

Dunque, dato $n \geq 2$, abbiamo che:

$$w \in L^0 \cup L^1 \cup L^n = L^*$$

• Di conseguenza, concludiamo che:

$$L^* = L(G') \in \text{CFL}$$

□

Esempio:

• Consideriamo il seguente linguaggio e la sua grammatica generante:

$$L = \{0^n 1^n \mid n \in \mathbb{N}\} \quad G : A \rightarrow 0A1 \mid \varepsilon$$

• La grammatica G' tale che $L(G) = L(G)^*$, corrisponderà a:

$$\begin{aligned} G' : \quad & S \rightarrow \varepsilon \mid A \mid SS \\ & A \rightarrow 0A1 \mid \varepsilon \end{aligned}$$

Teorema 15: Non chiusura dell'intersezione in CFL

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

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

Dimostrazione.

- Consideriamo i seguenti due linguaggi:

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

- Tali linguaggi sono descritti dalle seguenti due grammatiche:

$$\begin{array}{ll} G_1 : & S \rightarrow TV \\ & T \rightarrow aTb \mid \varepsilon \\ & V \rightarrow cV \mid \varepsilon \end{array} \quad \begin{array}{ll} G_2 : & S \rightarrow VT \\ & T \rightarrow bTc \mid \varepsilon \\ & V \rightarrow aV \mid \varepsilon \end{array}$$

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

- 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 $L_1, L_2 \in \text{CFL}$ ma $L_1 \cap L_2 \notin \text{CFL}$

□

Teorema 16: Non chiusura del complemento in CFL

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

$$\exists L \in \text{CFL} \mid \bar{L} \notin \text{CFL}$$

Dimostrazione.

- Consideriamo il seguente linguaggio:

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

- Consideriamo quindi la seguente grammatica:

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

- Data $x \in L$ tale che $|x|$ sia dispari, notiamo che:

- Se il simbolo centrale di x è a , allora $S \Rightarrow A \xRightarrow{*} x$
- Se il simbolo centrale di x è b , allora $S \Rightarrow B \xRightarrow{*} x$

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

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

- Siano quindi $u := x_1 \dots x_{2i-1}$ e $v := x_{2i} \dots x_n$. 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:

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

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:

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

Poiché i due casi sono analoghi, senza perdita di generalità consideriamo il caso in cui $S \Rightarrow AB \xRightarrow{*} x$

- Siano quindi $u := x_1 \dots x_k$ e $v := x_{k+1} \dots x_n$ tali che $x = uv$, $S \Rightarrow A \xRightarrow{*} u$ e $S \Rightarrow B \xRightarrow{*} v$.
- Poiché $S \Rightarrow A \xRightarrow{*} u$ e $S \Rightarrow B \xRightarrow{*} v$, otteniamo che:
 - $|u| = k$ e $|v| = n - k$ sono dispari
 - $S \Rightarrow A \xRightarrow{*} u$ implica che il simbolo centrale di u sia a , ossia che $x_{\frac{1+k}{2}} = a$
 - $S \Rightarrow B \xRightarrow{*} v$ 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 $L = L(G) \in \text{CFL}$
- Il complemento di tale linguaggio risulta essere $\bar{L} = \{ww \mid w \in \{a,b\}^*\}$, il quale abbiamo già dimostrato non essere un linguaggio acontestuale (sezione 2.5). Di conseguenza, concludiamo che $L \in \text{CFL}$, ma $\bar{L} \notin \text{CFL}$

□

2.7 Esercizi svolti

Problema 9: 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:

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

- Dalle regole di G , risulta evidente che se $A \xRightarrow{*} w$ 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:

$$w \in \Sigma^* \implies A \xRightarrow{*} w$$

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:

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

– A questo punto, notiamo che:

* Se $w_1 = 0$, allora $A \Rightarrow 0A \xRightarrow{*} 0a_2 \dots a_{n+1} = w$

* Se $w_1 = 1$, allora $A \Rightarrow 1A \xRightarrow{*} 1a_2 \dots a_{n+1} = w$

dunque concludiamo che $A \xRightarrow{*} w$

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

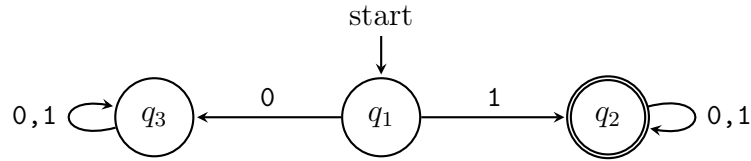
$$w \in L(G) \iff S \xRightarrow{*} w \iff S \Rightarrow 1A \xRightarrow{*} 1y = w \iff w \in L(R)$$

dove $A \xRightarrow{*} y$, 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 $L(G) = L(D)$:

$$G : \begin{array}{l} V_1 \rightarrow 1V_2 \mid 0V_3 \\ V_2 \rightarrow 1V_2 \mid 0V_2 \mid \varepsilon \\ V_3 \rightarrow 1V_3 \mid 0V_3 \mid \varepsilon \end{array}$$

□

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** 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:

$$\begin{array}{cccccccccccccccc}
 & \downarrow & & & & & & & & & & & & & & & & \\
 & 0 & 1 & 1 & 0 & 0 & 0 & \# & 0 & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & \\
 & & \downarrow & & & & & & & & & & & & & & & \\
 x & 1 & 1 & 0 & 0 & 0 & \# & 0 & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & & \\
 & & & & & & & \downarrow & & & & & & & & & & \\
 x & 1 & 1 & 0 & 0 & 0 & \# & x & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & & \\
 & \downarrow & & & & & & & & & & & & & & & & \\
 x & 1 & 1 & 0 & 0 & 0 & \# & x & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & & \\
 & & \downarrow & & & & & & & & & & & & & & & \\
 x & x & 1 & 0 & 0 & 0 & \# & x & 1 & 1 & 0 & 0 & 0 & \sqcup & \dots & & & \\
 & & & & & & & \dots & & & & & & & & & & \\
 & & & & & & & & & & \downarrow & & & & & & & \\
 x & x & x & x & x & x & \# & x & x & x & x & x & x & \sqcup & \dots & & &
 \end{array}$$

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

Definizione 35: 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 $q_{\text{reject}} \neq q_{\text{accept}}$
- $\delta : Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ 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, in 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 $a \rightarrow b; X$

Definizione 36: 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 37: 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:

$$uaq_i bv \text{ produce } uq_j acv \iff \delta(q_i, b) = (q_j, c, L)$$

$$uaq_i bv \text{ produce } uacq_j v \iff \delta(q_i, b) = (q_j, c, R)$$

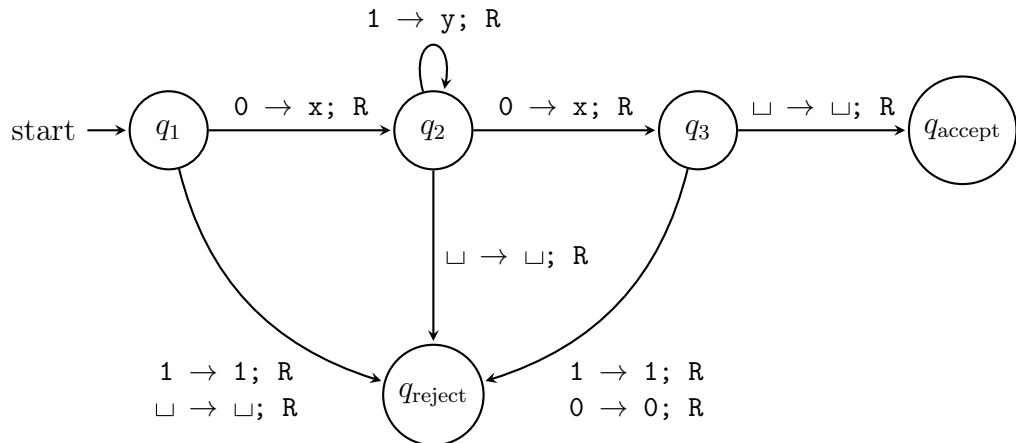
Proposizione 11: Stringa accettata in una TM

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

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

Esempio:

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

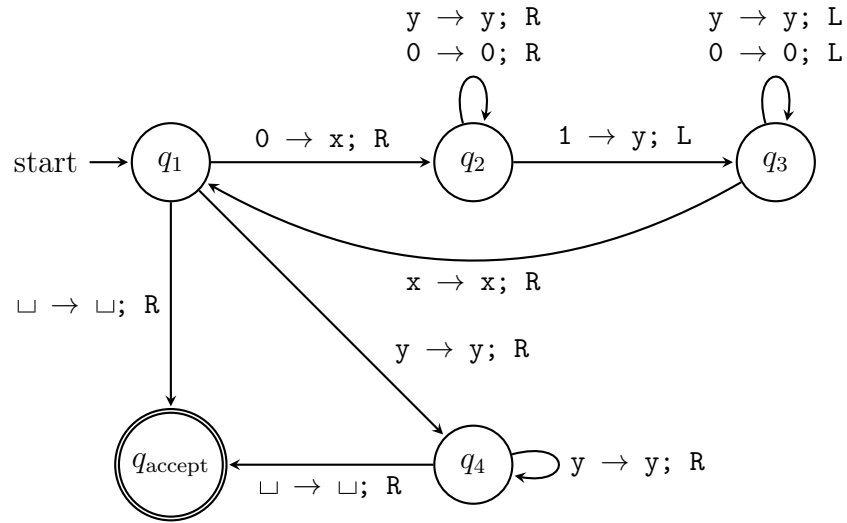


- Difatti, durante la lettura della stringa 01110, la macchina assume le seguenti configurazioni:

```

q1 0 1 1 1 0
x q2 1 1 1 0
x y q2 1 1 0
x y y q2 1 0
x y y y q2 0
x y y y x q3 □
x y y y x □ qaccept □
    
```

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:

q_1 0 0 0 1 1 1	$x q_1 x 0 y 1 1$	$x x q_1 0 y y 1$	
$x q_2 0 0 1 1 1$	$x x q_2 0 y 1 1$	$x x x q_2 y y 1$	$x x x q_3 y y y$
$x 0 q_2 0 1 1 1$	$x x 0 q_2 y 1 1$	$x x x y q_2 y 1$	$x x x y q_4 y y$
$x 0 0 q_2 1 1 1$	$x x 0 y q_2 1 1$	$x x x y y q_2 1$	$x x x y y q_4 y$
$x 0 q_3 0 y 1 1$	$x x 0 q_3 y y 1$	$x x x y q_3 y y$	$x x x y y y q_4 \square$
$x q_3 0 0 y 1 1$	$x x q_3 0 y y 1$	$x x x q_3 y y y$	$x x x y y y \square q_{\text{accept}} \square$
$q_3 x 0 0 y 1 1$	$x q_3 x 0 y y 1$	$x x q_3 x y y y$	

Definizione 38: 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 39: Classe dei linguaggi Turing-riconoscibili

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

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

Definizione 40: Classe dei linguaggi Turing-decidibili

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

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

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 13: 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 41: 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 $p = a_0 + a_1x_1 + \dots + a_nx_n$, possiamo immaginare la sua codifica come una stringa composta dai suoi coefficienti, ossia $\langle p \rangle = \#a_1, a_2, \dots, a_n\#$
- 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 Varianti della macchina di Turing

Definizione 42: 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:

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

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

Teorema 17: Equivalenza tra TM e Stay-put TM

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

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

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

Dimostrazione.

Prima implicazione.

- Dato $L \in \text{REC}$, 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:

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

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

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

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

□

Definizione 43: 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:

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

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

Teorema 18: Equivalenza tra TM e TM multinastro

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

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

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

Dimostrazione.

Prima implicazione.

- Dato $L \in \text{REC}$, 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

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

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 $L = L(M) = L(S) \in \text{REC}$

□

Definizione 44: 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:

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

Teorema 19: Equivalenza tra TM e NTM

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

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

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

Dimostrazione.

Prima implicazione.

- Dato $L \in \text{REC}$, 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 dallo 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:

$M = \text{"Data la stringa } w \text{ in input:}"$

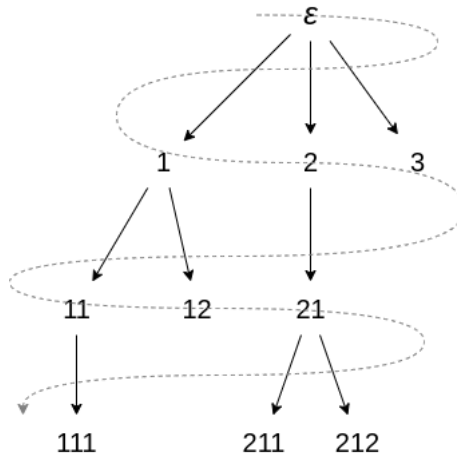
 1. Inizialmente, il nastro 1 di M contiene w , il nastro 3 contiene ε e il nastro 2 è vuoto
 2. Ripeti gli step successivi:
 3. M copia il nastro 1 sul nastro 2
 4. M simula N tramite il nastro 2 eseguendo un suo ramo di computazione. Prima di ogni passo simulato, M consulta il prossimo simbolo sul nastro 3 per poter scegliere su quale ramo proseguire.
 5. Se la simulazione accetta la stringa, anche M la *accetta*.
 6. Se invece non rimangono più simboli sul nastro 3 o se la simulazione rifiuta la stringa, sostituisci la stringa sul nastro 3 con l'indirizzo del nodo direttamente a destra del nodo precedente. Se non vi è un nodo a destra, viene scelto il nodo più a sinistra del livello successivo"
- 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 $L = L(N) = L(M) \in \text{REC}$

□



Rappresentazione grafica della dimostrazione

Definizione 45: 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 20: Equivalenza tra TM e Enumeratori

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

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

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

Dimostrazione.

Prima implicazione.

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

E = "Dato nulla in input:

1. Ripeti lo step seguente per $i = 1, 2, 3, \dots$:
2. Ripeti lo step seguente per $j = 1, \dots, i$:
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 $L = L(E) = L(M) \in \text{REC}$

□

3.1.2 Tesi di Church-Turing

Proposizione 12: 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 46: TM universale

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

Definizione 47: 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 48: Problema dell'accettazione per DFA

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

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

Teorema 21: A_{DFA} decidibile

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

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:

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

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

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

□

Definizione 49: Problema dell'accettazione per NFA

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

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

Teorema 22: A_{NFA} decidibile

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

Dimostrazione.

- Sia M_{DFA} il decisore tale che $L(M_{\text{DFA}}) = A_{\text{DFA}}$ (teorema A_{DFA} decidibile)
- Sia M la TM definita come:
 $M = \text{"Data in input la codifica } \langle N, w \rangle, \text{ dove } N \text{ è un NFA e } w \text{ una stringa:}$
 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:

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

implicando che $L(M) = A_{\text{NFA}}$

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

□

Definizione 50: Problema dell'accettazione per esp. reg.

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

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

Teorema 23: A_{REX} decidibile

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

Dimostrazione.

- Sia M_{NFA} il decisore tale che $L(M_{\text{NFA}}) = A_{\text{NFA}}$ (teorema A_{NFA} decidibile)
- Sia M la TM definita come:
 $M = \text{"Data in input la codifica } \langle R, w \rangle, \text{ dove } R \in \text{re}(\Sigma) \text{ e } w \text{ una stringa:}$
 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:

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

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

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

□

Definizione 51: Problema del vuoto per DFA

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

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

Teorema 24: E_{DFA} decidibile

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

Dimostrazione.

- Sia M la TM definita come:
 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:

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

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

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

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

□

Definizione 52: Problema dell'equivalenza tra DFA

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

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

Teorema 25: EQ_{DFA} decidibile

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

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 3, 4 e 5), ne segue automaticamente che:

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

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

- Inoltre, mostriamo che:

$$\begin{aligned} L(A) \Delta L(B) = \emptyset &\iff \\ (L(A) \cap \overline{L(B)}) \cup (L(B) \cap \overline{L(A)}) = \emptyset &\iff \\ \nexists x \in \Sigma^* \mid (x \in L(A) \wedge x \notin L(B)) \vee (x \in L(B) \wedge x \notin L(A)) &\iff \\ \forall x \in \Sigma^* (x \in L(A) \iff x \in L(B)) &\iff \\ L(A) = L(B) \end{aligned}$$

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

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

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 2, 3, 4 e 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:

$$\begin{aligned}\langle A, B \rangle \in EQ_{\text{DFA}} &\iff L(A) = L(B) \iff L(C) = L(A) \Delta L(B) = \emptyset \iff \\ &\langle C \rangle \in L(M_E) \iff \langle A, B \rangle \in L(M)\end{aligned}$$

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

□

Definizione 53: Problema dell'accettazione per CFG

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

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

Teorema 26: A_{CFG} decidibile

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

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 $2|w| - 1$ 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 $S \Rightarrow AB \xRightarrow{*} w$.
- Siano quindi $x, y \in \Sigma^*$ tali che $w = xy$, dove $A \xRightarrow{*} x$ e $B \xRightarrow{*} y$.
- 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 $S \Rightarrow AB \xrightarrow{*} xy = w$, 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:

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

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

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

□

Definizione 54: Problema del vuoto per CFG

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

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

Teorema 27: E_{CFG} decidibile

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

Dimostrazione.

- Sia M la TM definita come:

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

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 $A \rightarrow u_1 \dots u_k$ 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:

$$\begin{aligned} \langle G \rangle \in E_{\text{CFG}} &\iff L(G) = \emptyset \iff \nexists w \in L(D) \iff \\ &\forall w \in \Sigma^* \quad S \not\stackrel{*}{\Rightarrow} w \iff \langle G \rangle \in L(M) \end{aligned}$$

implicando che $L(M) = E_{\text{CFG}}$

- Inoltre, poiché il numero di variabili marcabili da M è finito, ne segue che M termina sempre, concludendo che $E_{\text{CFG}} = L(M) \in \text{DEC}$

□

Corollario 5: Ling. decidibili estensione dei ling. acontestuali

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

$$\text{CFL} \subsetneq \text{DEC}$$

Dimostrazione.

- Sia M_{CFG} il decisore tale che $L(M_{\text{CFG}}) = A_{\text{CFG}}$ (teorema [A_{CFG} decidibile](#))
- Dato $L \in \text{CFL}$, 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:

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

implicando che $L(M) = L(G)$. Inoltre, poiché A_{CFG} è un decisore, anche M è un decisore, implicando che $\text{CFG} \subseteq \text{DEC}$

- Consideriamo quindi il linguaggio $L = \{ww \mid w \in \{a,b\}^*\}$. Per dimostrazione precedente (sezione [2.5](#)), sappiamo che $L \notin \text{CFL}$. Tuttavia, possiamo facilmente definire un decisore M (simile a quella vista nella sezione [3.1](#)) per cui $L = L(M) \in \text{DEC}$
- Di conseguenza, concludiamo che:

$$\text{CFL} \subsetneq \text{DEC}$$

□

3.3 Argomento diagonale di Cantor

Teorema 28: Insiemi con stessa cardinalità

Dati due insiemi A e B si ha che:

$$\exists f : A \rightarrow B \text{ biettiva} \implies |A| = |B|$$

(*dimostrazione omessa*)

Definizione 55: 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:

$$f : \mathbb{N} \rightarrow 2\mathbb{N} : n \mapsto 2n$$

- 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 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 29: Razionali positivi numerabili

L'insieme $\mathbb{Q}_{\geq 0}$ dei numeri razionali non negativi è **numerabile**

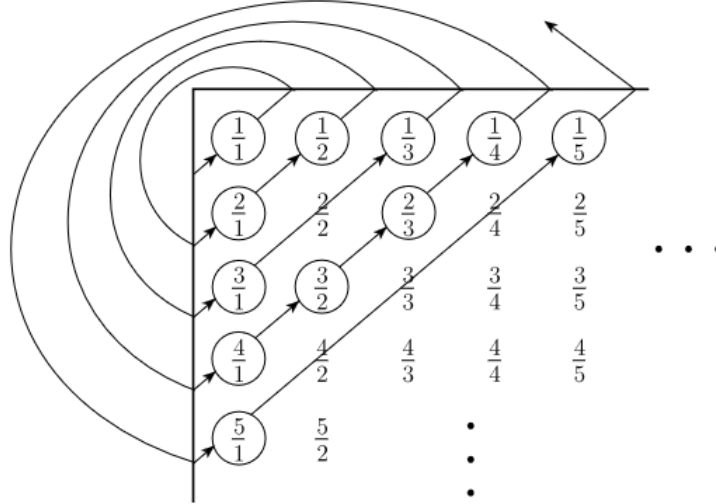
Dimostrazione.

- 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 $\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{3}{1}, \frac{1}{3}, \dots$ contenente tutti gli elementi di $\mathbb{Q}_{>0}$, senza alcuna ripetizione. Inoltre, aggiungiamo all'inizio di tale lista il numero 0.
- A questo punto, consideriamo la funzione $f : \mathbb{N} \rightarrow \mathbb{Q}_{\geq 0}$ definita come:

$f(n)$ = n -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 $\mathbb{Q}_{>0}$ 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 30: Reali non numerabili

L'insieme \mathbb{R} dei numeri reali **non è numerabile**

Dimostrazione.

- Dato $[0, 1] \subseteq \mathbb{R}$, supponiamo per assurdo che $\exists f : \mathbb{N} \rightarrow [0, 1]$ 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 $\nexists f : \mathbb{N} \rightarrow [0, 1]$ biettiva, implicando che $|\mathbb{N}| < |[0, 1]| \leq |\mathbb{R}|$ e dunque che \mathbb{R} non sia numerabile

□

n	$f(n)$
0	0. <u>1</u> 18285101...
1	0.2 <u>1</u> 3812941...
2	0.12 <u>3</u> 124112... $\implies x = 0.67392...$
3	0.945 <u>8</u> 53164...
4	0.3924 <u>8</u> 1412...
\vdots	\vdots

Rappresentazione grafica della dimostrazione

Teorema 31: Sequenze binarie infinite non numerabili

L'insieme \mathcal{B} di tutte le stringhe binarie infinite **non è numerabile**

Dimostrazione.

- Supponiamo per assurdo che $\exists f : \mathbb{N} \rightarrow \mathcal{B}$ 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 $\nexists f : \mathbb{N} \rightarrow \mathcal{B}$ biettiva, implicando che $|\mathbb{N}| < |\mathcal{B}|$ e dunque che \mathcal{B} non sia numerabile

□

n	$f(n)$
0	<u>1</u> 0101010101...
1	1 <u>1</u> 101101011...
2	010 <u>1</u> 0100001... $\implies x = 00110...$
3	0000 <u>1</u> 010100...
4	11111 <u>1</u> 111111...
\vdots	\vdots

Rappresentazione grafica della dimostrazione

3.3.1 Esistenza di linguaggi non riconoscibili

Teorema 32: Esistenza di linguaggi non riconoscibili

Dato un alfabeto Σ , si ha che:

$$\exists L \subseteq \Sigma^* \mid L \notin \text{REC}$$

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 $f : \mathbb{N} \rightarrow \Sigma^*$ 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:

$$\mathcal{M} = \{\langle M \rangle \mid M \text{ è una TM}\}$$

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 $\chi_L = b_1b_2\dots$ come *sequenza caratteristica di L* , definita 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, \dots sono tutte le stringhe di Σ^*

- Consideriamo quindi la seguente funzione:

$$g : \mathcal{L} \rightarrow \mathcal{B} : L \mapsto \chi_L$$

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:

$$h : \mathcal{M} \rightarrow \mathcal{L} : M \mapsto L(M)$$

non sia biettiva, implicando che $\exists L \in \mathcal{L} \mid \nexists M \in \mathcal{M} \text{ t.c. } L = L(M)$

□

3.4 Problemi indecidibili

Definizione 56: Problema dell'accettazione per TM

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

$$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ TM}, w \in L(M)\}$$

Teorema 33: A_{TM} riconoscibile ma non decidibile

Il linguaggio A_{TM} è **riconoscibile** ma **non decidibile**, ossia $A_{\text{TM}} \in \text{REC} - \text{DEC}$

Dimostrazione riconoscibilità.

- Sia U una TM universale a 2 nastri definita come:
 $U = \text{"Data in input la codifica } \langle M, w \rangle, \text{ dove } M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{accept}}, q_{\text{reject}}) \text{ è una TM e } w \text{ una stringa:}$
 1. Se la codifica in input è errata, U *rifuta*
 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:

$$\langle M, w \rangle \in L(U) \iff w \in L(M) \iff \langle M, w \rangle \in A_{\text{TM}}$$

implicando che $A_{\text{TM}} = L(U) \in \text{REC}$.

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 $A_{\text{TM}} \in \text{DEC}$. Sia quindi H il decisore tale che $L(H) = A_{\text{TM}}$

- 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:

$$\langle M, w \rangle \in L(D) \iff \langle M, w \rangle \notin L(H) = A_{\text{TM}} \iff w \notin L(M)$$

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:

$$\begin{aligned} \langle D, \langle D \rangle \rangle \in L(D) &\iff \langle D, \langle D \rangle \rangle \notin L(H) = A_{\text{TM}} \\ &\iff \langle D \rangle \notin L(D) \iff \langle D, \langle D \rangle \rangle \notin L(D) \end{aligned}$$

ottenendo quindi una contrazione in quanto D possa solo accettare o rifiutare

- Di conseguenza, ne segue necessariamente che $A_{\text{TM}} \notin \text{DEC}$

□

Corollario 6: Gerarchia dei linguaggi di Chomsky

Dato un alfabeto Σ , si ha che:

$$\text{REG} \subsetneq \text{CFL} \subsetneq \text{DEC} \subsetneq \text{REC} \subsetneq \mathcal{P}(\Sigma^*)$$

(segue dai teoremi 9, 5, 32 e 33)

Definizione 57: Classe dei linguaggi coTuring-riconoscibili

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

$$\text{coREC} = \{L \subseteq \Sigma^* \mid \bar{L} \in \text{REC}\}$$

Nota: $\text{coREC} \neq \mathcal{P}(\Sigma^*) - \text{REC}$

Teorema 34: Decidibilità, riconoscibilità e co-riconoscibilità

Un linguaggio L è **decidibile** se e solo se è **riconoscibile** e **co-riconoscibile**.

In altre parole, si ha che:

$$\text{DEC} = \text{REC} \cap \text{coREC}$$

Dimostrazione.

Prima implicazione.

- Dato $L \in \text{DEC}$, sia M il decisore tale che $L = L(M)$
- Sia \bar{M} la TM definita come:
 $\bar{M} = \text{"Data in input la stringa } w\text{:"}$
 1. Esegui il programma di M con input w
 2. Se l'esecuzione accetta, \bar{M} rifiuta, altrimenti accetta"
- Per costruzione stessa di \bar{M} , si ha che:

$$w \in L(\bar{M}) \iff w \notin L(M)$$

implicando che $\bar{L} = \overline{L(M)} = L(\bar{M}) \in \text{REC}$

- Dunque, poiché $L \in \text{DEC} \subseteq \text{REC}$ e $\bar{L} \in \text{REC}$, ne segue che $L \in \text{REC} \cap \text{coREC}$

Seconda implicazione.

- Dato $L \in \text{REC} \cap \text{coREC}$, siano M e \overline{M} le TM tali che $L = L(M)$ e $\overline{L} = L(\overline{M})$
- Sia D la TM definita come:

$D = \text{"Data in input la stringa } w\text{"}$

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:

$$w \in L = L(M) \iff w \notin \overline{L} = L(\overline{M})$$

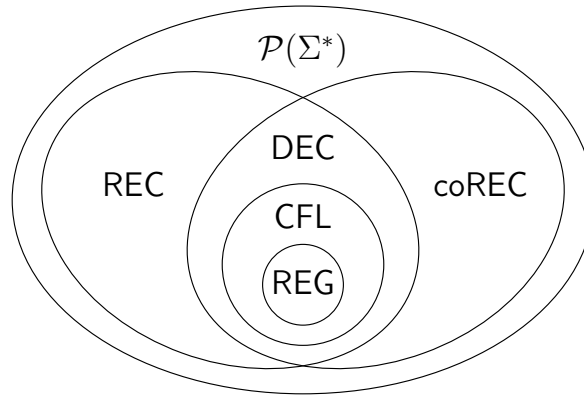
Di conseguenza, una delle due esecuzioni parallele accetterà qualsiasi stringa in input, implicando che D non vada mai in loop e quindi che $L = L(D) \in \text{DEC}$

□

Corollario 7: $\overline{A_{\text{TM}}}$ non riconoscibile

Il linguaggio $\overline{A_{\text{TM}}}$ è irriconoscibile

(segue dai teoremi 33 e 34)



Gerarchia delle classi dei linguaggi studiate fino ad ora

3.5 Riducibilità

Metodo 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 58: Problema della terminazione per le TM

Definiamo il linguaggio del **problema della terminazione per le TM** come:

$$HALT_{TM} = \{\langle M, w \rangle \mid M \text{ TM e } M(w) \text{ termina}\}$$

Teorema 35: $HALT_{TM}$ non decidibile

Il linguaggio $HALT_{TM}$ è **indecidibile**, ossia $HALT_{TM} \notin DEC$

Dimostrazione.

- Supponiamo per assurdo che $HALT_{TM} \in DEC$. Sia quindi H il decisore tale che $L(H) = HALT_{TM}$
- Sia D la TM definita come:
 $D = \text{"Data in input la stringa } \langle M, w \rangle \text{:}$
 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:

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

implicando che $L(D) = A_{TM}$.

- 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 $A_{TM} = L(D) \in DEC$. Tuttavia, ciò risulta assurdo in quanto $A_{TM} \notin DEC$ (A_{TM} riconoscibile ma non decidibile) Di conseguenza, ne segue necessariamente che $HALT_{TM} \notin DEC$

□

Definizione 59: Problema del vuoto per le TM

Definiamo il linguaggio del **problema del vuoto per le TM** come:

$$E_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ TM}, L(M) = \emptyset\}$$

Teorema 36: E_{TM} non decidibile

Il linguaggio E_{TM} è **indecidibile**, ossia $E_{\text{TM}} \notin \text{DEC}$

Dimostrazione.

- Supponiamo per assurdo che $E_{\text{TM}} \in \text{DEC}$. Sia quindi E il decisore tale che $L(E) = E_{\text{TM}}$

- Sia D la TM definita come:

$D =$ "Data in input la stringa $\langle M, w \rangle$:

1. Se la codifica in input è errata, D *rifiuta*

2. Costruisci una TM M' definita come:

$M' =$ "Data in input la stringa x :

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:

$$\langle M, w \rangle \in L(D) \iff \langle M' \rangle \notin L(E) \iff L(M') = \{w\}$$

$$\iff w \in L(M) \iff \langle M, w \rangle \in A_{\text{TM}}$$

implicando che $L(D) = A_{\text{TM}}$.

- Tuttavia, poiché E è un decisore, anche D risulta esserlo, implicando che $A_{\text{TM}} = L(D) \in \text{DEC}$. Tuttavia, ciò risulta assurdo in quanto $A_{\text{TM}} \notin \text{DEC}$. Di conseguenza, ne segue necessariamente che $E_{\text{TM}} \notin \text{DEC}$

□

Definizione 60: Problema della regolarità per le TM

Definiamo il linguaggio del **problema della regolarità per le TM** come:

$$REG_{TM} = \{\langle M \rangle \mid M \text{ TM}, L(M) \in REG\}$$

Teorema 37: REG_{TM} non decidibile

Il linguaggio REG_{TM} è **indecidibile**, ossia $REG_{TM} \notin DEC$

Dimostrazione.

- Supponiamo per assurdo che $REG_{TM} \in DEC$. Sia quindi R il decisore tale che $L(R) = REG_{TM}$
- Sia D la TM definita come:
 $D = \text{"Data in input la stringa } \langle M \rangle \text{"}$
 1. Se la codifica in input è errata, D *rifiuta*
 2. Costruisci una TM M' definita come:
 $M' = \text{"Data in input la stringa } x \text{"}$
 - 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*"
 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 $L(M') = \Sigma^* \in REG$
- 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 $L(M') = \{0^n 1^n \mid n \in \mathbb{N}\} \notin REG$ (sezione 1.6)
- Di conseguenza, concludiamo che $w \in L(M) \iff L(M') \in REG$
- A questo punto, per costruzione stessa di D , si ha che:

$$\langle M, w \rangle \in L(D) \iff \langle M' \rangle \in L(R) \iff L(M') \in REG$$

$$\iff w \in L(M) \iff \langle M, w \rangle \in A_{TM}$$

implicando che $L(D) = A_{TM}$.

- Tuttavia, poiché R è un decisore, anche D risulta esserlo, implicando che $A_{\text{TM}} = L(D) \in \text{DEC}$. Tuttavia, ciò risulta assurdo in quanto $A_{\text{TM}} \notin \text{DEC}$. Di conseguenza, ne segue necessariamente che $REG_{\text{TM}} \notin \text{DEC}$

□

Definizione 61: Problema dell'equivalenza per le TM

Definiamo il linguaggio del **problema dell'equivalenza per le TM** come:

$$EQ_{\text{TM}} = \{\langle M, M' \rangle \mid M, M' \text{ TM}, L(M) = L(M')\}$$

Teorema 38: EQ_{TM} non decidibile

Il linguaggio EQ_{TM} è **indecidibile**, ossia $EQ_{\text{TM}} \notin \text{DEC}$

Dimostrazione.

- Supponiamo per assurdo che $EQ_{\text{TM}} \in \text{DEC}$. Sia quindi E il decisore tale che $L(E) = EQ_{\text{TM}}$
- Sia D la TM definita come:

$D =$ "Data in input la stringa $\langle M, w \rangle$:

1. Se la codifica in input è errata, D *rifiuta*
2. Costruisci una TM M' definita come:

$M' =$ "Data in input la stringa x :

- 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:

$$\langle M, w \rangle \in L(D) \iff \langle M, M' \rangle \in L(E) \iff L(M) = L(M') = \emptyset \iff \langle M \rangle \in E_{\text{TM}}$$

implicando che $L(D) = E_{\text{TM}}$.

- Tuttavia, poiché E è un decisore, anche D risulta esserlo, implicando che $E_{\text{TM}} = L(D) \in \text{DEC}$. Tuttavia, ciò risulta assurdo in quanto sappiamo che $E_{\text{TM}} \notin \text{DEC}$ (E_{TM} non decidibile)
- Di conseguenza, ne segue necessariamente che $EQ_{\text{TM}} \notin \text{DEC}$

□

3.5.1 Riducibilità tramite mappatura

Definizione 62: Funzione calcolabile

Data $f : \Sigma^* \rightarrow \Sigma^*$, definiamo f come **calcolabile** se esiste una TM M tale che:

$$\forall w \in \Sigma^* \quad M(w) \text{ termina con solo } f(w) \text{ sul nastro}$$

Definizione 63: Riducibilità tramite mappatura

Dati due linguaggi A e B , diciamo che A è **riducibile a B tramite mappatura**, indicato come $A \leq_m B$, se esiste una funzione calcolabile $f : \Sigma^* \rightarrow \Sigma^*$, detta **riduzione da A a B** , tale che:

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

Teorema 39: Decidibilità tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m B$, si ha che:

$$B \in \text{DEC} \implies A \in \text{DEC}$$

Dimostrazione.

- Dato $B \in \text{DEC}$, sia D_B il decisore tale che $L(D_B) = B$
- Sia D_A la TM definita come:
 $D_A =$ "Data in input la stringa w :
 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 $A = L(D_A) \in \text{DEC}$

□

Corollario 8: Indecidibilità tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m B$, si ha che:

$$A \notin \text{DEC} \implies B \notin \text{DEC}$$

Esempi:

1. • Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

$F = \text{"Data in input la stringa } \langle M, w \rangle \text{"}$

1. Costruisci una TM M' definita come:

$M' = \text{"Data in input la stringa } x \text{"}$

- 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:

$$\begin{aligned} \langle M, w \rangle \in A_{\text{TM}} &\iff w \in L(M) \implies w \in L(M') \\ &\implies f(\langle M, w \rangle) = \langle M', w \rangle \in \text{HALT}_{\text{TM}} \end{aligned}$$

e inoltre che:

$$\begin{aligned} \langle M, w \rangle \notin A_{\text{TM}} &\iff w \notin L(M) \implies M'(w) \text{ va in loop} \\ &\implies f(\langle M, w \rangle) = \langle M', w \rangle \notin \text{HALT}_{\text{TM}} \end{aligned}$$

- Di conseguenza, poiché:

$$\langle M, w \rangle \in A_{\text{TM}} \iff f(\langle M, w \rangle) \in \text{HALT}_{\text{TM}}$$

ne segue che $A_{\text{TM}} \leq_m \text{HALT}_{\text{TM}}$

- Infine, poiché $A_{\text{TM}} \notin \text{DEC}$, concludiamo che $\text{HALT}_{\text{TM}} \notin \text{DEC}$

2. • Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

$F = \text{"Data in input la stringa } \langle M \rangle \text{"}$

1. Costruisci una TM M' definita come:

$M' = \text{"Data in input la stringa } x \text{"}$

- i. *Rifuta*"

2. Restituisci in output la stringa $\langle M, M' \rangle$ "

- Notiamo che:

$$\langle M \rangle \in E_{\text{TM}} \iff L(M) = \emptyset = L(M') \iff f(\langle M \rangle) = \langle M, M' \rangle \in EQ_{\text{TM}}$$

- Di conseguenza, poiché:

$$\langle M \rangle \in E_{\text{TM}} \iff f(\langle M \rangle) \in EQ_{\text{TM}}$$

ne segue che $E_{\text{TM}} \leq_m EQ_{\text{TM}}$

- Infine, poiché $E_{\text{TM}} \notin \text{DEC}$, concludiamo che $EQ_{\text{TM}} \notin \text{DEC}$

Teorema 40: Riconoscibilità tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m B$, si ha che:

$$B \in \text{REC} \implies A \in \text{REC}$$

(*dimostrazione analoga al teorema 39*)

Corollario 9: Irriconoscibilità tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m B$, si ha che:

$$A \notin \text{REC} \implies B \notin \text{REC}$$

Teorema 41: 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 42: EQ_{TM} non riconoscibile e non co-riconoscibile

Il linguaggio EQ_{TM} non è né riconoscibile né co-riconoscibile

Dimostrazione.

- Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

$F =$ "Data in input la stringa $\langle M, w \rangle$:

1. Costruisci una TM M_1 definita come:

$M_1 =$ "Data in input la stringa x :

- i. *Rifiuta*"

2. Costruisci una TM M_2 definita come:

$M_2 =$ "Data in input la stringa x :

- 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:

$$\begin{aligned} \langle M, w \rangle \in A_{\text{TM}} &\implies L(M_1) = \emptyset, L(M_2) = \Sigma^* \implies L(M_1) \neq L(M_2) \\ &\iff f(\langle M, w \rangle) = \langle M_1, M_2 \rangle \notin EQ_{\text{TM}} \iff f(\langle M, w \rangle) \in \overline{EQ_{\text{TM}}} \end{aligned}$$

e inoltre che:

$$\begin{aligned} \langle M, w \rangle \notin A_{\text{TM}} &\implies L(M_1) = \emptyset, L(M_2) = \emptyset \implies L(M_1) = L(M_2) \\ &\iff f(\langle M, w \rangle) = \langle M_1, M_2 \rangle \in EQ_{\text{TM}} \iff f(\langle M, w \rangle) \notin \overline{EQ_{\text{TM}}} \end{aligned}$$

- Di conseguenza, poiché:

$$\langle M, w \rangle \in A_{\text{TM}} \iff f(\langle M, w \rangle) \in \overline{EQ_{\text{TM}}}$$

ne segue che $A_{\text{TM}} \leq_m \overline{EQ_{\text{TM}}}$

- Sia inoltre $g : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM G definita come:

$G =$ "Data in input la stringa $\langle M, w \rangle$:

1. Costruisci una TM M_1 definita come:

$M_1 =$ "Data in input la stringa x :

- i. *Accetta*"

2. Costruisci una TM M_2 definita come:

$M_2 =$ "Data in input la stringa x :

- 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:

$$\begin{aligned} \langle M, w \rangle \in A_{\text{TM}} &\implies L(M_1) = \Sigma^*, L(M_2) = \Sigma^* \implies L(M_1) = L(M_2) \\ &\iff g(\langle M, w \rangle) = \langle M_1, M_2 \rangle \in EQ_{\text{TM}} \end{aligned}$$

e inoltre che:

$$\begin{aligned} \langle M, w \rangle \notin A_{\text{TM}} &\implies L(M_1) = \Sigma^*, L(M_2) = \emptyset \implies L(M_1) \neq L(M_2) \\ &\iff g(\langle M, w \rangle) = \langle M_1, M_2 \rangle \in EQ_{\text{TM}} \end{aligned}$$

- Di conseguenza, poiché:

$$\langle M, w \rangle \in A_{\text{TM}} \iff g(\langle M, w \rangle) \in EQ_{\text{TM}}$$

ne segue che $A_{\text{TM}} \leq_m EQ_{\text{TM}}$

- A questo punto, per la [Riducibilità complementare](#), si ha che:

$$\begin{aligned} A_{\text{TM}} \leq_m \overline{EQ_{\text{TM}}} &\iff \overline{A_{\text{TM}}} \leq_m EQ_{\text{TM}} \\ A_{\text{TM}} \leq_m EQ_{\text{TM}} &\iff \overline{A_{\text{TM}}} \leq_m \overline{EQ_{\text{TM}}} \end{aligned}$$

- Infine, poiché $A_{\text{TM}} \notin \text{REC}$, ne segue automaticamente che $EQ_{\text{TM}}, \overline{EQ_{\text{TM}}} \notin \text{REC}$

□

3.6 Esercizi svolti

Problema 10

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

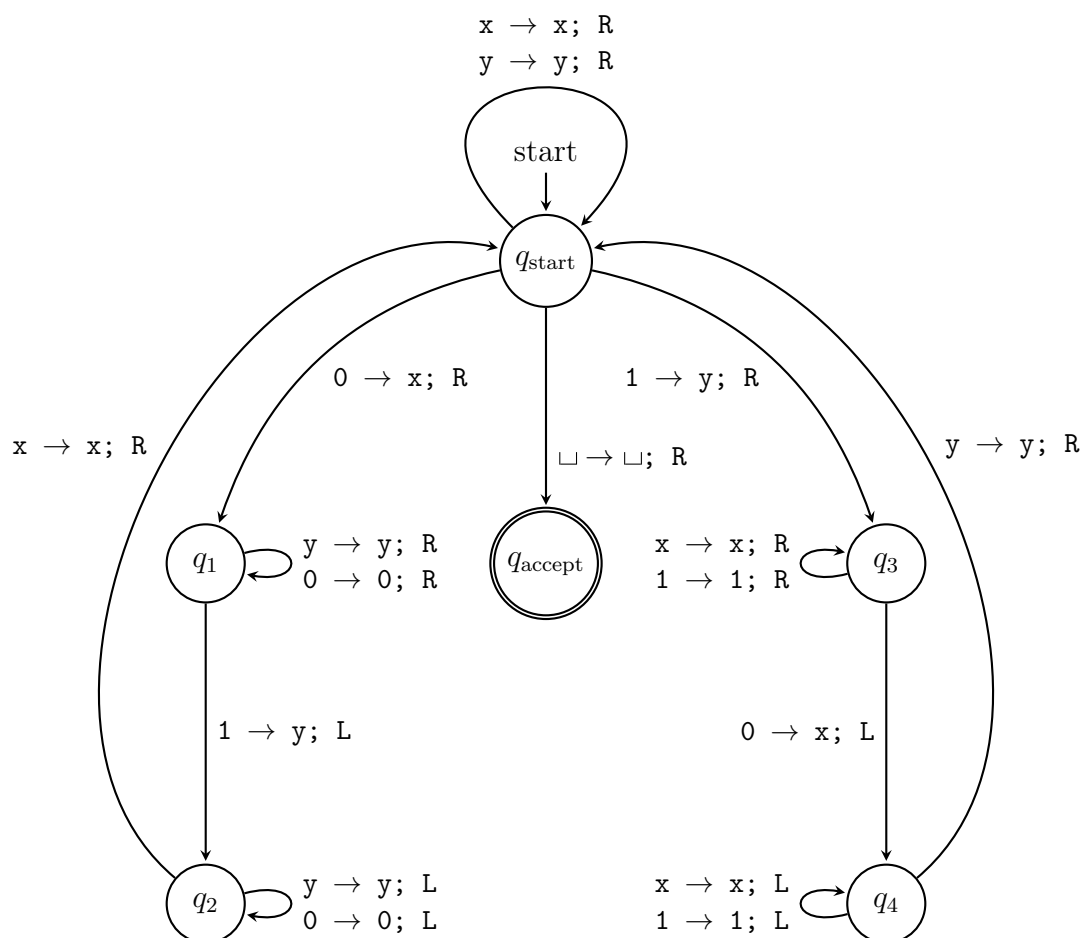
Dimostrazione.

- 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:



- Notiamo che, se lo stato attuale è q_{start} , si verifica che:
 - Se viene letto uno 0, il cammino $C := q_{\text{start}} \rightarrow q_1 \rightarrow q_2 \rightarrow q_{\text{start}}$ 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 $C' := q_{\text{start}} \rightarrow q_3 \rightarrow q_4 \rightarrow q_{\text{start}}$ 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 $q_{\text{start}} \rightarrow q_{\text{start}}$ 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 11: Insieme di linguaggi distinti

Dato un alfabeto Σ , siano $L_1, \dots, L_k \subseteq \Sigma^*$, dove $k \in \mathbb{N}$, dei linguaggi tali che:

1. $\forall i \neq j \quad L_i \cap L_j = \emptyset$
2. $L_1 \cup \dots \cup L_k = \Sigma^*$
3. $\forall i \in [1, k] \quad L_i \in \text{REC}$, ossia sono Turing-riconoscibili

Dimostrare che $L_1, \dots, L_k \in \text{DEC}$, ossia sono decidibili

Dimostrazione.

- Le prime due proprietà dei linguaggi L_1, \dots, 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:

$$\forall i \in [1, k] \quad \overline{L_i} = \bigcup_{j \neq i} L_j$$

- Poiché $L_1, \dots, L_k \in \text{REC}$, siano M_1, \dots, M_k le TM tali che $\forall i \in [1, k] \quad L_i = L(M_i)$
- 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 $M_1, \dots, M_{i-1}, M_{i+1}, \dots, M_k$ 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 $\overline{L_i} = L(\overline{M_i}) \in \text{REC}$, ossia che L_i sia coTuring-riconoscibile

- Di conseguenza, poiché:

$$L_i \in \text{DEC} \iff L_i \in \text{REC}, L_i \in \text{coREC} \iff L_i \in \text{REC}, \overline{L_i} \in \text{REC}$$

ne segue automaticamente che L_1, \dots, L_k siano decidibili

□

Problema 12: Riducibilità al proprio complemento

Dato un linguaggio A tale che $A \leq_m \bar{A}$, dimostrare che:

$$A \in \text{REC} \implies A \in \text{DEC}$$

Dimostrazione.

- Poiché $A \leq_m \bar{A}$, per la [Riducibilità complementare](#) abbiamo che:

$$A \leq_m \bar{A} \iff \bar{A} \leq_m \bar{\bar{A}} = A$$

- Supponiamo quindi che $A \in \text{REC}$. In tal caso, ne segue che:

$$\bar{A} \leq_m A, A \in \text{REC} \implies \bar{A} \in \text{REC} \implies A \in \text{coREC}$$

- Di conseguenza, per la [Decidibilità, riconoscibilità e co-riconoscibilità](#), concludiamo che:

$$A \in \text{REC} \cap \text{coREC} \iff A \in \text{DEC}$$

□

Problema 13: $START-0_{\text{TM}}$ indecidibile

Dato il seguente linguaggio:

$$START-0_{\text{TM}} = \{\langle M \rangle \mid M \text{ TM}, L(M) = \{0y \mid y \in \Sigma^*\}\}$$

dimostrare che è **indecidibile**, ossia che $START-0_{\text{TM}} \notin \text{DEC}$

Dimostrazione.

- Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

$F = \text{"Data in input la stringa } \langle M, w \rangle \text{"}$

1. Costruisci una TM M' definita come:

$M' = \text{"Data in input la stringa } x \text{"}$

- 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*."

2. Restituisci in output la stringa $\langle M' \rangle$ "

- Supponiamo che $w \notin L(M)$. In tal caso, M' rifiuterà qualsiasi stringa, implicando che $L(M') = \emptyset$ e dunque che $\langle M' \rangle \notin START-0_{\text{TM}}$

- 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 $\langle M' \rangle \in START-0_{TM}$

- Di conseguenza, concludiamo che $w \in L(M) \iff L(M') \in START-0_{TM}$
- A questo punto, notiamo che:

$$\langle M, w \rangle \in A_{TM} \iff w \in L(M) \iff f(\langle M, w \rangle) = \langle M' \rangle \in START-0_{TM}$$

implicando quindi che $A_{TM} \leq_m START-0_{TM}$

- Infine, poiché $A_{TM} \notin DEC$, concludiamo che $START-0_{TM} \notin DEC$

□

Problema 14: W_{TM} indecidibile

Dato il seguente linguaggio:

$$W_{TM} = \left\{ \langle M, w, a \rangle \mid \begin{array}{l} M = (Q, \Sigma_M, \Gamma_M, \delta, q_s, q_a, q_r) \text{ TM,} \\ w \in \Sigma_M^*, a \in \Gamma_M, \\ M \text{ scrive } a \text{ su input } w \end{array} \right\}$$

dimostrare che è **indecidibile**, ossia che $W_{TM} \notin DEC$

Dimostrazione.

- Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

$F =$ "Data in input la stringa $\langle M, w \rangle$:

1. Costruisci una TM M' definita come:

$M' =$ "Data in input la stringa x :

- i. Esegui il programma di M su input x
- ii. Se l'esecuzione accetta, scrivi a sul nastro e *accetta*

2. Restituisci in output la stringa $\langle M', w, a \rangle$ "

- Notiamo che:

$$\langle M, w \rangle \in A_{TM} \iff w \in L(M) \iff M'(w) \text{ scrive } a \text{ sul nastro}$$

$$\iff f(\langle M, w \rangle) = \langle M', w, a \rangle \in W_{TM}$$

implicando quindi che $A_{TM} \leq_m W_{TM}$

- Infine, poiché $A_{\text{TM}} \notin \text{DEC}$, concludiamo che $W_{\text{TM}} \notin \text{DEC}$

□

Problema 15: $UNION\text{-}ALL_{\text{TM}}$ indecidibile

Dato il seguente linguaggio:

$$UNION\text{-}ALL_{\text{TM}} = \{\langle T, T' \rangle \mid T, T' \text{ TM}, L(T) \cup L(T') = \Sigma^*\}$$

dimostrare che è **indecidibile**, ossia che $UNION\text{-}ALL_{\text{TM}} \notin \text{DEC}$

Dimostrazione.

- Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

$F = \text{"Data in input la stringa } \langle M, w \rangle \text{"}$

1. Costruisci una TM T' definita come:

$T = \text{"Data in input la stringa } x \text{"}$

- i. *Rifiuta*

2. Costruisci una TM T' definita come:

$T' = \text{"Data in input la stringa } x \text{"}$

- i. Esegui il programma di M con input w .
- ii. Se l'esecuzione accetta, T' *accetta*

3. Restituisci in output la stringa $\langle T, T' \rangle$

- Notiamo che:

$$\langle M, w \rangle \in A_{\text{TM}} \iff w \in L(M) \implies L(T) = \emptyset, L(T') = \Sigma^* \implies L(T) \cup L(T') = \Sigma^*$$

$$\iff f(\langle M, w \rangle) = \langle T, T' \rangle \in UNION\text{-}ALL_{\text{TM}}$$

e inoltre che:

$$\langle M, w \rangle \notin A_{\text{TM}} \iff w \notin L(M) \implies L(T) = \emptyset, L(T') = \emptyset \implies L(T) \cup L(T') = \emptyset$$

$$\implies f(\langle M, w \rangle) = \langle T, T' \rangle \notin UNION\text{-}ALL_{\text{TM}}$$

implicando quindi che $A_{\text{TM}} \leq_m UNION\text{-}ALL_{\text{TM}}$

- Infine, poiché $A_{\text{TM}} \notin \text{DEC}$, concludiamo che $UNION\text{-}ALL_{\text{TM}} \notin \text{DEC}$

□

Problema 16: ODD_{TM} indecidibile

Dato il seguente linguaggio:

$$ODD_{TM} = \{\langle M \rangle \mid M \text{ TM}, L(M) = \{w \in \Sigma^* \mid |w| \text{ dispari}\}\}$$

dimostrare che è **indecidibile**, ossia che $START-0_{TM} \notin DEC$

Dimostrazione.

- Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:
 $F = \text{"Data in input la stringa } \langle M, w \rangle \text{"}$
 1. Costruisci una TM M' definita come:
 $M' = \text{"Data in input la stringa } x \text{"}$
 - i. Se $|x|$ è pari, *rifiuta*. Altrimenti, esegui il programma di M su input w
 - ii. Se l'esecuzione accetta, allora M' *accetta*."
 2. Restituisci in output la stringa $\langle M' \rangle$ "
- Supponiamo che $w \notin L(M)$. In tal caso, M' rifiuterà qualsiasi stringa, implicando che $L(M') = \emptyset$ e dunque che $\langle M' \rangle \notin ODD_{TM}$
- 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 $\langle M' \rangle \in ODD_{TM}$

- Di conseguenza, concludiamo che $w \in L(M) \iff \langle M' \rangle \in ODD_{TM}$
- A questo punto, notiamo che:

$$\langle M, w \rangle \in A_{TM} \iff w \in L(M) \iff f(\langle M, w \rangle) = \langle M' \rangle \in ODD_{TM}$$

implicando quindi che $A_{TM} \leq_m ODD_{TM}$

- Infine, poiché $A_{TM} \notin DEC$, concludiamo che $ODD_{TM} \notin DEC$

□

Problema 17: ODD_{TM} irriconoscibile

Dato il seguente linguaggio:

$$ODD_{TM} = \{\langle M \rangle \mid M \text{ TM}, L(M) = \{w \in \Sigma^* \mid |w| \text{ dispari}\}\}$$

dimostrare che è **irriconoscibile**, ossia che $START-0_{TM} \notin REC$

Dimostrazione.

- Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:
 $F = \text{"Data in input la stringa } \langle M, w \rangle \text{"}$
 1. Costruisci una TM M' definita come:
 $M' = \text{"Data in input la stringa } x \text{"}$
 - i. Se $|x|$ è dispari, *accetta*. Altrimenti, esegui il programma di M su input w
 - ii. Se l'esecuzione accetta, allora M' *accetta*."
 2. Restituisci in output la stringa $\langle M' \rangle$ "
- Supponiamo che $w \in L(M)$. In tal caso, M' rifiuterà qualsiasi stringa, implicando che $L(M') = \Sigma^*$ e dunque che $\langle M' \rangle \notin ODD_{TM}$
- 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 $\langle M' \rangle \in ODD_{TM}$

- Di conseguenza, concludiamo che $w \in L(M) \iff \langle M' \rangle \notin ODD_{TM}$
- A questo punto, notiamo che:

$$\langle M, w \rangle \in \overline{A_{TM}} \iff \langle M, w \rangle \notin A_{TM} \iff w \notin L(M)$$

$$\iff f(\langle M, w \rangle) = \langle M' \rangle \in ODD_{TM}$$

implicando quindi che $\overline{A_{TM}} \leq_m ODD_{TM}$

- Infine, poiché $\overline{A_{TM}} \notin REC$, concludiamo che $ODD_{TM} \notin REC$

□

Problema 18: $DECIDABLE_{TM}$ indecidibile

Dato il seguente linguaggio:

$$DECIDABLE_{TM} = \{ \langle M \rangle \mid M \text{ TM}, L(M) \in DEC \}$$

dimostrare che è **indecidibile**, ossia che $DECIDABLE_{TM} \notin DEC$

Dimostrazione.

- Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

$F =$ "Data in input la stringa $\langle M, w \rangle$:

1. Costruisci una TM M' definita come:

$M' =$ "Data in input la stringa x :

- i. Esegui il programma di M su input w .
- ii. Se l'esecuzione accetta, M' accetta

2. Restituisci in output la stringa $\langle M' \rangle$ "

- Notiamo che:

$$\begin{aligned} \langle M, w \rangle \in A_{TM} &\iff w \in L(M) \implies M' \text{ termina sempre} \\ &\iff f(\langle M, w \rangle) = \langle M' \rangle \in DECIDABLE_{TM} \end{aligned}$$

e inoltre che:

$$\begin{aligned} \langle M, w \rangle \notin A_{TM} &\iff w \notin L(M) \implies M' \text{ va sempre in loop} \\ &\implies f(\langle M, w \rangle) = \langle M' \rangle \notin DECIDABLE_{TM} \end{aligned}$$

- Di conseguenza, abbiamo che:

$$\langle M, w \rangle \in A_{TM} \iff f(\langle M, w \rangle) \in DECIDABLE_{TM}$$

implicando quindi che $A_{TM} \leq_m DECIDABLE_{TM}$

- Infine, poiché $A_{TM} \notin DEC$, concludiamo che $DECIDABLE_{TM} \notin DEC$

□

Problema 19: $USELESS_{TM}$ indecidibile

Dato il seguente linguaggio:

$$USELESS_{TM} = \{ \langle M \rangle \mid M \text{ TM senza stati inutili} \}$$

dimostrare che è **indecidibile**, ossia che $USELESS_{TM} \notin DEC$

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 $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:
 $F = \text{"Data in input la stringa } \langle M, w \rangle \text{"}$
 1. Costruisci una TM M' definita come:
 $M' = \text{"Data in input la stringa } x \text{"}$
 - i. In M' esistono solo tre stati: q_{start} , q_{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' *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} viene raggiunto da ogni esecuzione di M'
 - Lo stato q_{reject} viene raggiunto solo dall'esecuzione di M' su input w
 - Se $w \in L(M)$, lo stato q_{accept} viene raggiunto da ogni esecuzione di M' su input $x \neq w$
 - Se $w \notin L(M)$, lo stato q_{accept} non viene mai raggiunto
 di conseguenza, concludiamo che $w \in L(A) \iff$ Non esiste uno stato inutile in M'
- A questo punto, si ha che:

$$\langle M, w \rangle \in A_{TM} \iff w \in L(M) \iff \text{Non esiste uno stato inutile in } M'$$

$$\iff f(\langle M, w \rangle) = \langle M' \rangle \in USELESS_{TM}$$
 implicando quindi che $A_{TM} \leq_m USELESS_{TM}$
- Infine, poiché $A_{TM} \notin DEC$, concludiamo che $USELESS_{TM} \notin DEC$

□

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 64: Complessità temporale di una TM

Sia D un decisore. Definiamo come **complessità temporale** (o *tempo di esecuzione*) di D la funzione $f : \mathbb{N} \rightarrow \mathbb{N}$ 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:
 $M = \text{"Data in input la stringa } w\text{:}$
 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 65: O grande

Siano $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$. Diciamo che $f(n)$ è **in O grande di** $g(n)$, indicato come $f(n) = O(g(n))$, se:

$$\exists c, n_0 \in \mathbb{N}_{>0} \mid \forall n \geq n_0 \quad f(n) \leq c \cdot g(n)$$

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:

$$\forall n \geq 100 \quad f(n) = 100n^2 \leq 10 \cdot n^3$$

di conseguenza, si ha che $100n^2 = O(n^3)$

Definizione 66: o piccolo

Siano $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$. Diciamo che $f(n)$ è **in o piccolo di** $g(n)$, indicato come $f(n) = o(g(n))$, se:

$$\lim_{n \rightarrow +\infty} \frac{f(n)}{g(n)} = 0$$

In altre parole, si ha che:

$$\forall c \in \mathbb{R}_{>0} \quad \exists n_0 \in \mathbb{N}_{>0} \mid \forall n \geq n_0 \quad f(n) < c \cdot g(n)$$

Esempio:

- Date le due funzioni $f(n) = 100n^2$ e $g(n) = n^3$, si ha che $100n^2 = o(n^3)$

Osservazione 14

Date due funzioni $f, g : \mathbb{N} \rightarrow \mathbb{R}^+$, si ha che:

$$f(n) = o(g(n)) \implies f(n) = O(g(n))$$

Proposizione 13: Algebra asintotica

Date le funzioni $f, g, h : \mathbb{N} \rightarrow \mathbb{R}^+$, si ha che:

- $\forall c \in \mathbb{R} \quad f(n) = c \cdot o(g(n)) \implies f(n) = o(g(n))$
- $f(n) = o(g(n)) + o(h(n)) \implies f(n) = o(m(n))$, 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 43: 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 67: Complessità temporale di una NTM

Sia N un decisore non-deterministico. Definiamo come **complessità temporale** di N la funzione $f : \mathbb{N} \rightarrow \mathbb{N}$ 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 44: 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:

$$t(n) \cdot b^{t(n)} = 2^{\log_2(t(n) \cdot b^{t(n)})} = 2^{t(n) \cdot \log_2(t(n) + \log_2(b))} = 2^{O(t(n))}$$

- 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 68: Classe DTIME

Data la funzione $t : \mathbb{N} \rightarrow \mathbb{R}^+$, definiamo come **classe dei linguaggi decidibili in tempo $O(t(n))$** il seguente insieme:

$$\text{DTIME}(t(n)) = \{L \in \text{DEC} \mid L \text{ decidibile da una TM in tempo } O(t(n))\}$$

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 $L \in \text{DTIME}(n^2)$

Definizione 69: Classe P

Definiamo la **classe dei linguaggi decibili in tempo polinomiale** come:

$$P = \bigcup_{k=0}^{+\infty} \text{DTIME}(n^k)$$

Definizione 70: Problema dei cammini

Definiamo il linguaggio del **problema dei cammini** come:

$$PATH = \{ \langle G, s, t \rangle \mid G = (V_G, E_G) \text{ grafo diretto con un cammino } s \rightarrow t \}$$

Teorema 45: PATH polinomialmente decibibile

Il linguaggio $PATH$ è **polinomialmente decibibile**, ossia $PATH \in P$

Dimostrazione.

- Sia M la TM definita come:
 $M = \text{"Data la stringa } \langle G, s, t \rangle \text{ in input:}$
 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 $PATH \in \text{DTIME}(n^k) \subseteq P$

□

Definizione 71: Classe EXP

Definiamo la **classe dei linguaggi decidable in tempo esponenziale** come:

$$\text{EXP} = \bigcup_{k=1}^{\infty} \text{DTIME}(2^{n^k})$$

Definizione 72: Problema dei cammini hamiltoniani

Definiamo il linguaggio del **problema dei cammini hamiltoniani** come:

$$\text{HAMPATH} = \left\{ \langle G, s, t \rangle \mid \begin{array}{l} G = (V_G, E_G) \text{ grafo diretto con un} \\ \text{cammino hamiltoniano } s \rightarrow t \end{array} \right\}$$

dove un cammino è detto *hamiltoniano* se passa una sola volta per ogni nodo del grafo di appartenenza

Teorema 46: HAMPATH esponenzialmente decidibile

Il linguaggio *HAMPATH* è **esponenzialmente decidibile**, ossia $\text{HAMPATH} \in \text{EXP}$

Dimostrazione.

- Sia M la TM definita come:

$M = \text{"Data la stringa } \langle G, s, t \rangle \text{ in input:}$

1. Ripeti per ogni cammino possibile in G :
2. Verifica se il cammino è hamiltoniano. Se lo è, *accetta*
3. *Rifuta*"

- 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 hamiltoniano 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 $\text{HAMPATH} \in \text{DTIME}(2^{n^k}) \subseteq \text{EXP}$

□

4.3 Classe NP

Definizione 73: Classe NTIME

Data la funzione $t : \mathbb{N} \rightarrow \mathbb{R}^+$, definiamo come **classe dei linguaggi decidibili non-deterministicamente in tempo** $O(t(n))$ il seguente insieme:

$$\text{NTIME}(t(n)) = \{L \in \text{DEC} \mid L \text{ decidibile da una NTM in tempo } O(t(n))\}$$

Proposizione 14

Dato un linguaggio L , si ha che:

$$\text{NTIME}(n^k) \subseteq \text{DTIME}(2^{O(n^k)})$$

(segue dal teorema 44)

Definizione 74: Classi NP e NEXP

Definiamo la **classe dei linguaggi decidibili non-deterministicamente in tempo polinomiale** come:

$$\text{NP} = \bigcup_{k=1}^{\infty} \text{NTIME}(n^k)$$

Analogamente, definiamo la **classe dei linguaggi decidibili non-deterministicamente in tempo esponenziale** come:

$$\text{NEXP} = \bigcup_{k=1}^{\infty} \text{NTIME}(2^{n^k})$$

Corollario 10: Gerarchia temporale dei linguaggi

Date le classi P , NP e EXP , si ha che:

$$P \subseteq NP \subseteq EXP \subseteq NEXP$$

Dimostrazione.

- Per definizione stessa, una TM risulta essere anche una NTM. Di conseguenza, ne segue automaticamente che $P \subseteq NP$ e che $EXP \subseteq NEXP$
- Inoltre, per il corollario precedente, abbiamo che:

$$L \in NP \implies \exists k \in \mathbb{N} \mid L \in \text{NTIME}(n^k) \implies L \in \text{DTIME}(2^{O(n^k)}) \implies L \in EXP$$

□

Definizione 75: CLIQUE

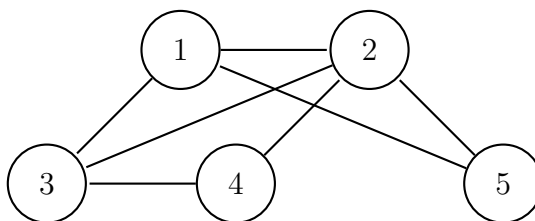
Definiamo come *CLIQUE* il seguente linguaggio:

$$CLIQUE = \{\langle G, k \rangle \mid G \text{ grafo non diretto con una } k\text{-clique}\}$$

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 47: CLIQUE non deter. polinomialmente verificabile**

Il linguaggio *CLIQUE* è **non deterministicamente polinomialmente verificabile**, ossia $CLIQUE \in NP$

Dimostrazione.

- Sia N la NTM definita come:
 $N = \text{"Data la stringa } \langle G, k \rangle \text{ in input, dove } G = (V_G, E_G) \text{ è un grafo:}$
 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 $CLIQUE = L(N) \in \text{NTIME}(n^h)$ per qualche $h \in \mathbb{N}$

□

Definizione 76: Verificatore

Dato un linguaggio $L \in \text{DEC}$, definiamo un decisore V come **verificatore di L** se:

$$L = \{w \in \Sigma^* \mid \exists c \in \Sigma^* \langle w, c \rangle \in L(V)\}$$

dove la stringa c viene detta **certificato di appartenenza**

Osservazione 15: 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 48: Classe NP (definizione alternativa)

Data la classe NP, si ha che:

$$\text{NP} = \{L \in \text{DEC} \mid \exists V \text{ verificatore polinomiale per } L\}$$

In altre parole, possiamo definire NP anche come la **classe dei linguaggi verificabili in tempo polinomiale**

Dimostrazione.

Prima implicazione.

- Dato $L \in \text{NP}$, 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:

$V = \text{"Data la stringa } \langle w, c \rangle \text{ in input:}$

 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 $L = L(N) \in \text{NP}$, 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:

$N = \text{"Data la stringa } \langle w \rangle \text{ in input:}$

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:

$$w \in L \iff \exists c \in \Sigma^*, \langle w, c \rangle \in V \iff$$

$$\text{Esiste un ramo di } N \text{ che accetta } w \iff w \in L$$

implicando che $L \in L(N) \in \text{NP}$

□

Definizione 77: Problema della tricolorazione dei grafi

Definiamo il linguaggio del **problema della tricolorazione dei grafi** come:

$$3COL = \{ \langle G \rangle \mid G = (V_G, E_G) \text{ grafo tricolorabile} \}$$

dove un grafo è detto *tricolorabile* se ad ogni suo nodo è possibile assegnare un colore diverso dai suoi nodi adiacenti utilizzando massimo tre colori

Teorema 49: 3COL polinomialmente verificabile

Il linguaggio 3COL è **polinomialmente verificabile**, ossia $3COL \in NP$

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 $c = \langle c_1, \dots, c_m \rangle$, 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:

$$\langle G \rangle \in 3COL \iff \text{Esiste una colorazione valida } c_1, \dots, c_n$$

$$\iff \exists c = \langle c_1, \dots, c_n \rangle \in \Sigma^*, \langle\langle G \rangle, c\rangle \in V$$

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 $3COL \in NP$

□

4.4 Riducibilità in tempo polinomiale

Definizione 78: 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 $f : \Sigma^* \rightarrow \Sigma^*$, 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 50: Decidibilità polinomiale tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m^P B$, si ha che:

$$B \in P \implies A \in P$$

(*dimostrazione analoga al teorema 39*)

Teorema 51: Verificabilità polinomiale tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m^P B$, si ha che:

$$B \in NP \implies A \in NP$$

(*dimostrazione analoga al teorema 39*)

Definizione 79: 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 52: 3SAT riducibile in tempo polinomiale a CLIQUE

Dati i linguaggi 3SAT e CLIQUE, si ha che $3SAT \leq_m^P CLIQUE$

Dimostrazione.

- Sia $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM F definita come:

$F = \text{"Data in input la stringa } \langle \phi \rangle \text{:}$

1. Costruisci un grafo G definito come:

- i. Crea un nodo x_i ed un nodo \bar{x}_i per ogni variabile in ϕ , creando dei dopplioni se la variabile compare più volte in ϕ
- ii. Organizza i nodi di G in k triple, dove tre nodi appartengono alla stessa tripla se e solo se i loro letterali compaiono all'interno di una stessa clausola *or* all'interno di ϕ
- iii. Crea un arco tra ogni coppia di nodi fatta eccezione di nodi appartenenti alla stessa tripla e nodi opposti tra loro (es: x_i e \bar{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 $\phi \in 3SAT$. 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 $C = \{x_1, \dots, x_k\}$ 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 $x_i = \bar{x}_j$ 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:

$$\phi \in 3SAT \implies f(\langle \phi \rangle) = \langle G, w \rangle \in CLIQUE$$

- Supponiamo ora che $f(\phi) = \langle G, w \rangle \in CLIQUE$, implicando che esista una k -clique al suo interno. Dunque, per costruzione di G , nessuno dei nodi corrispondenti ai letterali di tale clique può appartenere alla stessa tripla di un altro nodo della clique
- 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 \bar{x}_i

- Di conseguenza, abbiamo che:

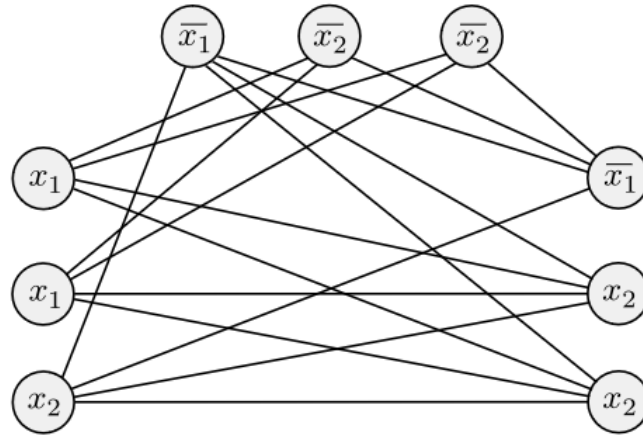
$$f(\langle \phi \rangle) = \langle G, w \rangle \in CLIQUE \implies \phi \in 3SAT$$

implicando quindi che:

$$\phi \in 3SAT \iff f(\langle \phi \rangle) = \langle G, w \rangle \in CLIQUE$$

- 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 11: 3SAT polinomialmente verificabile

Il linguaggio 3SAT è **polinomialmente verificabile**, ossia $3SAT \in NP$
 (segue dai teoremi 47, 51 e 52)

Teorema 53: 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 $f, g : \Sigma^* \rightarrow \Sigma^*$ 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 54: 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 41*)

4.5 Classe NP-Complete

Definizione 80: Classe NP-Hard

Definiamo la **classe dei linguaggi NP-Difficili** come:

$$\text{NP-Hard} = \{B \subseteq \Sigma^* \mid \forall A \in \text{NP} \ A \leq_m^P B\}$$

Nota: non è detto che un linguaggio NP-Hard sia decidibile o riconoscibile

Definizione 81: Classe NP-Complete

Definiamo la **classe dei linguaggi NP-Completi** come:

$$\text{NP-Complete} = \text{NP} \cap \text{NP-Hard}$$

Teorema 55: Completezza transitiva

Dati due linguaggi B e C tali che $B \leq_m^P C$ e $C \in \text{NP}$, si ha che:

$$B \in \text{NP-Complete} \implies C \in \text{NP-Complete}$$

Dimostrazione.

- Poiché la relazione \leq_m^P è transitiva e poiché $B \in \text{NP-Complete}$, ne segue automaticamente che:

$$\forall A \in \text{NP} \quad A \leq_m^P B \leq_m^P C$$

implicando dunque che $C \in \text{NP} \cap \text{NP-Hard} = \text{NP-Complete}$

□

Teorema 56: Linguaggio NP-completo in P

Dato un linguaggio $B \in \text{NP-Complete}$, si ha che:

$$B \in \text{P} \iff \text{P} = \text{NP}$$

Dimostrazione.

Prima implicazione.

- Per definizione stessa di NP-Complete, si ha che $\forall A \in \text{NP} \quad A \leq_m^P B$
- Di conseguenza, poiché $B \in \text{P}$, ne segue automaticamente che:

$$\forall A \in \text{NP} \quad B \in \text{P} \implies A \in \text{P}$$

implicando dunque che $\text{NP} \subseteq \text{P}$

- Inoltre, poiché sappiamo già che $\text{P} \subseteq \text{NP}$, concludiamo che $\text{P} = \text{NP}$

Seconda implicazione.

- Per definizione stessa di NP-Complete, si ha che $B \in \text{NP}$
- Di conseguenza, se $\text{P} = \text{NP}$, ne segue automaticamente che $B \in \text{P}$

□

4.5.1 Teorema di Cook-Levin

Definizione 82: Problema della soddisfacibilità

Definiamo il linguaggio del **problema della soddisfacibilità** come:

$$SAT = \{\langle \phi \rangle \mid \phi \text{ è una formula soddisfacibile}\}$$

Teorema 57: Teorema di Cook-Levin

Dato il linguaggio SAT , si ha che $SAT \in \text{NP-Complete}$

Dimostrazione.

- Sia V la TM definita come:

$V = \text{"Data la stringa } \langle \phi, c \rangle \text{ in input:}$

1. Interpreta $c = \langle x_1, \dots, x_n \rangle$ come un assegnamento di variabili
2. Verifica se $\phi(x_1, \dots, x_n)$ è soddisfacibile.
3. Se vero, *accetta*. Altrimenti, *rifiuta*"

- Per costruzione stessa di V , si ha che:

$$\langle \phi \rangle \in SAT \iff \text{Esiste un assegnamento che soddisfa } \phi \iff$$

$$\exists c \in \Sigma^*, \langle \phi, c \rangle \in V$$

implicando che V sia un verificatore di SAT . Inoltre, il suo tempo di esecuzione risulta intuitivamente polinomiale, implicando che $SAT \in \text{NP}$

- Dato un linguaggio $A \in \text{NP}$, 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 $C = Q \cup \Gamma \cup \{\#\}$
- Per ogni $i, j \in [1, n^k]$ e per ogni $s \in C$, definiamo la variabile logica $x_{i,j,s}$ tale che:

$$x_{i,j,s} = 1 \iff T[i, j] = s$$

- 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:

- 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:

$$\begin{aligned} \phi_{\text{start}} = & x_{1,1,\#} \wedge x_{1,2,q_0} \wedge \\ & x_{1,3,w_1} \wedge x_{1,4,w_2} \wedge \dots \wedge x_{1,2,n+2,w_n} \wedge \\ & x_{1,n+3,\sqcup} \wedge \dots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#} \end{aligned}$$

la quale descrive la prima configurazione iniziale di un tableau

(si consiglia di guardare gli esempi riportati in seguito prima di proseguire con la dimostrazione)

- 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 del tableau 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.
- Consideriamo quindi la seguente formula:

$$\phi_{\text{move}} = \bigwedge_{0 \leq i, j \leq n^k} \left(\bigvee_{\substack{(a_1, \dots, a_6) \\ \text{è finestra lecita}}} x_{i,j-1,a_1} \wedge x_{i,j,a_2} \wedge x_{i,j+1,a_3} \wedge x_{i+1,j-1,a_4} \wedge x_{i+1,j,a_5} \wedge x_{i+1,j+1,a_6} \right)$$

la quale risulta impone che tutte le configurazioni del tableau rispettino δ

- Infine, definiamo la formula:

$$\phi = \phi_{\text{cell}} \wedge \phi_{\text{start}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{move}}$$

- Notiamo quindi che le formule ϕ_{cell} , ϕ_{start} e ϕ_{move} siano tutte vere per qualsiasi tableau valido, mentre la formula ϕ_{accept} sia vera solo per i tableau accettanti
- Di conseguenza, otteniamo che:

$$T \text{ è un tableau accettante} \implies \phi_{\text{accept}} \text{ è vera} \implies \phi \text{ è vera}$$

e inoltre che:

$$T \text{ non è un tableau accettante} \implies \phi_{\text{accept}} \text{ è falsa} \implies \phi \text{ è falsa}$$

implicando che:

$$T \text{ è un tableau accettante} \iff \phi \text{ è vera}$$

- Sia quindi $f : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolabile dalla seguente TM M definita come:
 $M = \text{"Data la stringa } w \text{ in input:}$

1. Costruisci la formula ϕ tramite un'esecuzione accettante di $N(w)$
2. Restituisci in output la stringa $\langle \phi \rangle$ "

- A questo punto, risulta evidente che:

$$w \in A = L(N) \iff \exists \text{ tableau accettante per } N(w)$$

$$\implies \phi \text{ è vera} \iff f(w) = \langle \phi \rangle \in SAT$$

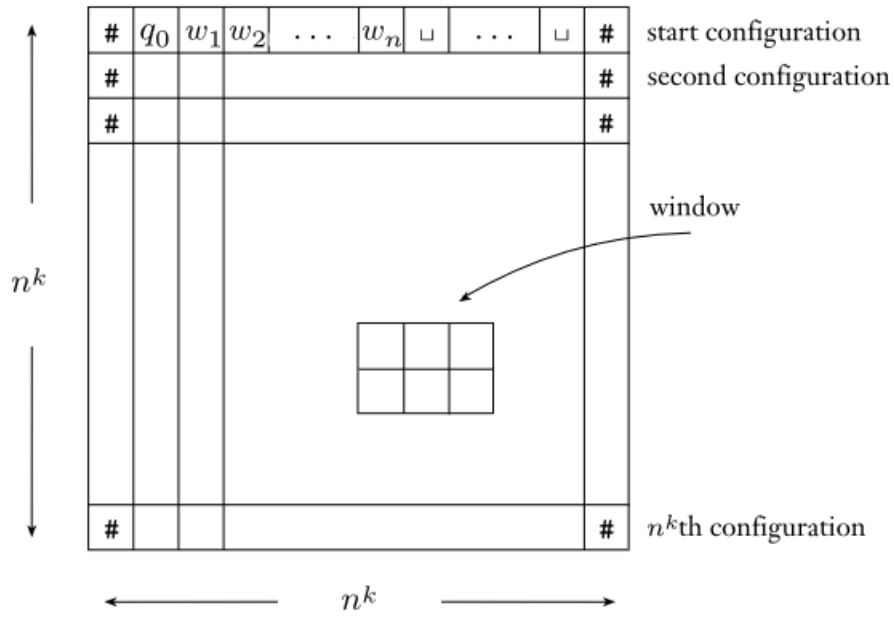
implicando che:

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

- Inoltre, poiché:
 - La formula ϕ_{start} è costruibile in $O(n^k)$
 - Le formule ϕ_{accept} e ϕ_{cell} sono costruibili in $O(n^{2k})$
 - La formula ϕ_{move} è costruibile in $O(n^{2k}) \cdot O(m)$, dove $m = \binom{|C|}{6}$. Tuttavia, poiché m non dipende dall'input, ne segue che $O(m) = O(1)$
- 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 $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$

4.5.2 Classi coP, coNP e coEXP

Definizione 83: Classi coP, coNP e coEXP

Definiamo le classi dei linguaggi coP, coNP e coEXP come:

$$\text{coP} = \{A \in \text{DEC} \mid \bar{A} \in \text{P}\}$$

$$\text{coNP} = \{A \in \text{DEC} \mid \bar{A} \in \text{NP}\}$$

$$\text{coEXP} = \{A \in \text{DEC} \mid \bar{A} \in \text{EXP}\}$$

Teorema 58: Chiusura del complemento di P

Date le classi P e coP, si ha che:

$$\text{P} = \text{coP}$$

In altre parole, la classe P è **chiusa nel complemento**

Dimostrazione.

Prima implicazione.

- Dato $A \in \text{P}$, sia D il decisore polinomiale tale che $L(D) = A$
- Sia \bar{D} la TM definita come:
 $\bar{D} = \text{"Data la stringa } w \text{ in input:}$
 1. Esegui il programma di D su input w
 2. Se l'esecuzione accetta, *rifiuta*. Altrimenti, *accetta*"
- Per costruzione stessa di \bar{D} , abbiamo che:

$$x \in L(\bar{D}) \iff x \notin L(D) = A$$

implicando che $L(\bar{D}) = \bar{A}$

- Inoltre, poiché D è un decisore polinomiale, ne segue che anche \bar{D} lo sia, concludendo che $\bar{A} = L(\bar{D}) \in \text{P}$ e dunque che $A \in \text{coP}$

Seconda implicazione.

- Analogamente alla prima implicazione

□

Teorema 59: Chiusura del complemento di EXP

Date le classi EXP e coEXP, si ha che:

$$\text{EXP} = \text{coEXP}$$

In altre parole, la classe EXP è **chiusa nel complemento**

(*dimostrazione analoga al teorema precedente*)

Corollario 12: $P \subseteq \text{coNP} \subseteq \text{EXP}$

Date le classi P, coNP e EXP, si ha che:

$$P \subseteq \text{coNP} \subseteq \text{EXP}$$

Dimostrazione.

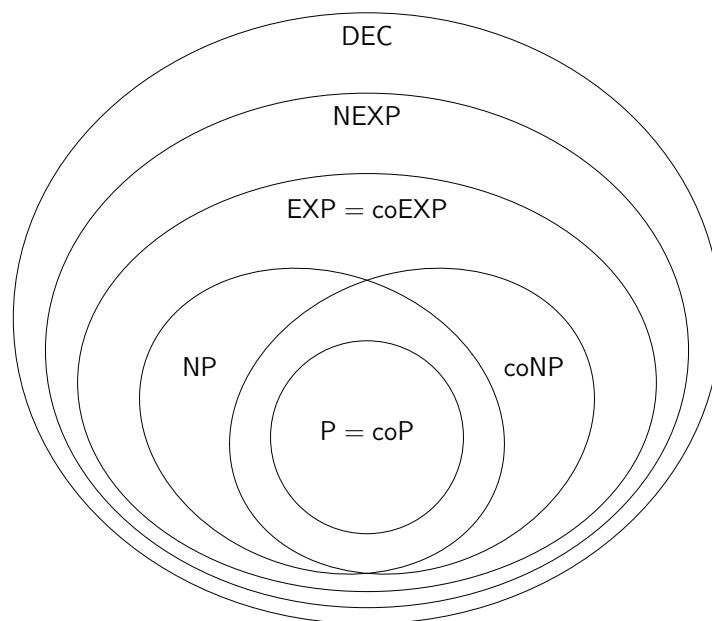
- Dato $A \in P$, per i teoremi della [Gerarchia temporale dei linguaggi](#), [Chiusura del complemento di P](#) e [Chiusura del complemento di EXPSPACE](#), abbiamo che:

$$- A \in P \implies \bar{A} \in P \subseteq \text{NP} \implies A \in \text{coNP}$$

$$- A \in \text{coNP} \implies \bar{A} \in \text{NP} \subseteq \text{EXP} \implies A \in \text{EXP}$$

implicando che $P \subseteq \text{coNP} \subseteq \text{EXP}$

□



Gerarchia delle classi dei linguaggi decidibili studiate fino ad ora

Teorema 60

Date le classi P , NP e $coNP$, si ha che:

$$P = NP \implies P = coNP$$

Dimostrazione.

- Per il corollario precedente, sappiamo che $P \subseteq coNP$
- Dato $A \in coNP$, per l'ipotesi e per il teorema [Chiusura del complemento di P](#), si ha che:

$$A \in coNP \implies \bar{A} \in NP = P \implies A \in P$$

concludendo quindi che $P = coNP$

□

Corollario 13

Date le classi P , NP e $coNP$, si ha che:

$$NP \neq coNP \implies P \neq NP$$

Dimostrazione.

- Tramite il teorema precedente, si ha che:

$$P = NP \implies P = coNP \implies coNP = NP$$

- Dunque, per contronominale, otteniamo che:

$$NP \neq coNP \implies P \neq NP$$

□

Definizione 84: Classe coNP-Complete

Definiamo la **classe dei linguaggi coNP-Completi** come:

$$coNP-Complete = \{L \in DEC \mid L \text{ è coNP-Completo}\}$$

dove un linguaggio B è detto *coNP-Completo* se:

- $B \in coNP$
- $\forall A \in coNP \quad A \leq_m^P B$

Teorema 61: Completezza del complemento in NP-Complete

Dato un linguaggio B si ha che:

$$B \in \text{NP-Complete} \iff \bar{B} \in \text{coNP-Complete}$$

Dimostrazione.

- Tramite la [Riducibilità logaritmica complementare](#), si ha che:

$$\begin{aligned} B \in \text{NP-Complete} &\iff B \in \text{NP}, \forall A \in \text{NP} \quad A \leq_m^P B \iff \\ B \in \text{NP}, \forall A \in \text{NP} \quad \bar{A} &\leq_m^P \bar{B} \iff \bar{B} \in \text{coNP}, \forall \bar{A} \in \text{coNP} \quad \bar{A} \leq_m^P \bar{B} \\ &\iff \bar{B} \in \text{coNP-Complete} \end{aligned}$$

□

Teorema 62: Ugualianza tra NP e coNP

Dato un linguaggio $B \in \text{NP-Complete}$, si ha che:

$$B \in \text{coNP} \iff \text{NP} = \text{coNP}$$

Dimostrazione.

Prima implicazione.

- Poiché $B \in \text{coNP} \iff \bar{B} \in \text{NP}$ e poiché $B \in \text{NP-Complete}$, si ha che:

$$A \in \text{NP} \implies A \leq_m^P B \iff \bar{A} \leq_m^P \bar{B} \implies \bar{A} \in \text{NP} \iff A \in \text{coNP}$$

implicando che $\text{coNP} \subseteq \text{NP}$

- Viceversa, si ha che:

$$A \in \text{coNP} \implies \bar{A} \in \text{NP} \implies \bar{A} \leq_m^P B \implies A \in \text{NP}$$

implicando che $\text{NP} \subseteq \text{coNP}$

Seconda implicazione.

- Se $\text{NP} = \text{coNP}$, allora $B \in \text{NP-Complete} \subseteq \text{NP} = \text{coNP}$

□

4.6 Complessità spaziale

Definizione 85: 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 $f : \mathbb{N} \rightarrow \mathbb{N}$ 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 63: 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 16: 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 in input la stringa w :

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 il 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 17: 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:
 $M = \text{"Data in input la stringa } w\text{:"}$
 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, *rifuta*
 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 86: Classi DSPACE e NSPACE

Data la funzione $s : \mathbb{N} \rightarrow \mathbb{R}^+$, definiamo come **classe dei linguaggi decidibili in spazio** $O(s(n))$ il seguente insieme:

$$\text{DSPACE}(s(n)) = \{L \in \text{DEC} \mid L \text{ decidibile da una TM in spazio } O(s(n))\}$$

Analogamente, definiamo come **classe dei linguaggi decidibili non-deterministicamente in spazio** $O(s(n))$ il seguente insieme:

$$\text{NSPACE}(s(n)) = \{L \in \text{DEC} \mid L \text{ decidibile da una NTM in spazio } O(s(n))\}$$

Teorema 64: PATH decidibile in $O(\log^2 n)$

Il linguaggio *PATH* è **decidibile in** $O(\log^2 n)$, ossia $\text{PATH} \in \text{DSPACE}(\log^2 n)$

Dimostrazione.

- Sia $G = (V, E)$ un grafo
- Consideriamo la seguente procedura ricorsiva definita su G :
 $\text{Path?}(x, y, k)$:
 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 $\text{Path?}(x, v, k-1)$ accetta e $\text{Path?}(v, y, k-1)$ accetta, allora la procedura *accetta*. Altrimenti, essa *rifiuta*
- Notiamo che le foglie dell'albero di ricorsione della chiamata iniziale della procedura corrispondano esattamente ai nodi del cammino da x a y (qualora esso esista).
- Inoltre, poiché ad ogni ricorsione vengono effettuate due chiamate, ne segue che le foglie dell'albero ricorsivo siano al massimo 2^k
- Sia quindi M la TM definita come:

$M =$ "Data la stringa $\langle G, s, t \rangle$ in input;

1. Esegui la procedura $\text{Path?}(s, t, \lceil \log n \rceil)$.
2. Se la procedura accetta, *accetta*. Altrimenti, *rifiuta*"

- Notiamo quindi che:

$$\langle G, s, t \rangle \in L(M) \iff \text{Path?}(s, t, \lceil \log n \rceil) \text{ accetta} \iff$$

$$\text{Esiste cammino } s \rightarrow t \text{ di } O(2^{\lceil \log n \rceil}) = O(n) \text{ nodi} \iff \langle G, s, t \rangle \in \text{PATH}$$

implicando che $L(M) = \text{PATH}$

- Analizziamo quindi il costo spaziale di M :
 - Ogni chiamata della procedura necessita di memorizzare i due nodi in input, richiedendo $2 \cdot O(\log n) = O(\log n)$ spazio
 - L'altezza dell'albero di ricorsione è $\lceil \log n \rceil = O(\log n)$
- 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 $PATH = L(M) \in DSPACE(\log^2 n)$

□

4.6.1 Rapporto tra spazio e tempo

Teorema 65: Tempo limitante lo spazio

Data una funzione $f(n)$ tale che $f(n) \geq n$, si ha che:

$$DTIME(f(n)) \subseteq NTIME(f(n)) \subseteq DSPACE(f(n))$$

Dimostrazione.

- Sappiamo già che $DTIME(f(n)) \subseteq NTIME(f(n))$
- Sia quindi N una NTM tale che $L(N) \in NTIME(f(n))$
- 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 NTM utilizzasse una nuova cella ad ogni passo, il massimo numero di celle utilizzabili sarebbe $O(f(n))$, implicando che $L(N) \in DSPACE(f(n))$

□

Teorema 66: Spazio limitante il tempo

Data una funzione $f(n)$ tale che $f(n) \geq \log n$, si ha che:

$$DSPACE(f(n)) \subseteq DTIME(2^{O(f(n))})$$

Dimostrazione.

- Sia M una TM tale che $L(M) \in DSPACE(f(n))$
- Per comodità, assumiamo che M sia dotata di un solo work tape
- Indichiamo le configurazioni di M con la seguente notazione:

$$WORK_{q_i}TAPE; j$$

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:

$$f(n) \geq n \implies 2^{f(n)} \geq n$$

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 $L(M) \in \text{DTIME}(2^{O(f(n))})$

□

4.6.2 Teorema di Savitch

Teorema 67: Teorema di Savitch

Data una funzione $f(n) \geq \log n$, si ha che:

$$\text{NSPACE}(f(n)) \subseteq \text{DSPACE}(f^2(n))$$

Dimostrazione.

- Sia N una NTM tale che $L(M) \in \text{NSPACE}(f(n))$
- 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 alterarne il suo linguaggio)
- Siano inoltre q_{start} e δ rispettivamente lo stato iniziale di N e la sua funzione di transizione

- Consideriamo la notazione WORK_qTape ; 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:

$$w \in L(N) \iff \text{Esiste cammino } c_{\text{start}} \rightarrow c_{\text{accept}} \text{ in } G_{N,w}$$

- Consideriamo quindi la procedura $\text{Path}_{G_{N,w}}?$, una versione modificata della procedura $\text{Path}?$ 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:

$M = \text{"Data la stringa } w \text{ in input:}"$

 1. Esegui la procedura $\text{Path}_{G_{N,w}}?(c_{\text{start}}, c_{\text{accept}}, \lceil \log m \rceil)$
 2. Se la procedura accetta, *accetta*. Altrimenti, *rifiuta*
- Per costruzione stessa di M , si ha che:

$$w \in L(M) \iff \text{Path}_{G_{N,w}}?(c_{\text{start}}, c_{\text{accept}}, \lceil \log m \rceil) \text{ accetta} \iff$$

$$\text{Esiste cammino } c_{\text{start}} \rightarrow c_{\text{accept}} \text{ in } G_{N,w} \iff w \in L(N)$$

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 $\text{Path}_{G_{N,w}}?(c_{\text{start}}, c_{\text{accept}}, \lceil \log m \rceil)$ 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 $L(N) = L(M) \in \text{DSpace}(f^2(n))$

□

4.7 Classe PSPACE

Definizione 87: Classi PSPACE e NPSPACE

Definiamo la **classe dei linguaggi decidibili in spazio polinomiale** come:

$$\text{PSPACE} = \bigcup_{k=1}^{\infty} \text{DSPACE}(n^k)$$

Analogamente, definiamo la **classe dei linguaggi decidibili non-deterministicamente in spazio polinomiale** come:

$$\text{NPSPACE} = \bigcup_{k=1}^{\infty} \text{NSPACE}(n^k)$$

Definizione 88: Classi EXPSPACE e NEXPSPACE

Definiamo la **classe dei linguaggi decidibili in spazio esponenziale** come:

$$\text{EXPSPACE} = \bigcup_{k=1}^{\infty} \text{DSPACE}(2^{n^k})$$

Analogamente, definiamo la **classe dei linguaggi decidibili non-deterministicamente in spazio esponenziale** come:

$$\text{NEXPSPACE} = \bigcup_{k=1}^{\infty} \text{NSPACE}(2^{n^k})$$

Teorema 68: Rapporto tra spazio e tempo

Dato un alfabeto Σ , si ha che:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \subseteq \text{EXP} \subseteq \text{NEXP} \subseteq \text{EXPSPACE}$$

Dimostrazione.

- Tramite il teorema [Tempo limitante lo spazio](#), si ha che:

$$\text{P} \subseteq \text{NP} \subseteq \text{PSPACE} \qquad \text{EXP} \subseteq \text{NEXP} \subseteq \text{EXPSPACE}$$

- Tramite il teorema [Spazio limitante il tempo](#), invece, si ha che:

$$\text{PSPACE} \subseteq \text{EXP}$$

□

Teorema 69: PSPACE = NPSPACE

Date le classi PSPACE e NPSPACE:

$$\text{PSPACE} = \text{NPSPACE}$$

Dimostrazione.

- Per definizione stessa, si ha che $\text{PSPACE} \subseteq \text{NPSPACE}$
- Inoltre, per il [Teorema di Savitch](#), si ha che:

$$L \in \text{NPSPACE} \implies \exists k \in \mathbb{N} \mid L \in \text{NSPACE}(n^k) \subseteq \text{DSPACE}(n^{2k}) \implies L \in \text{PSPACE}$$

implicando che $\text{NPSPACE} \subseteq \text{PSPACE}$

□

Teorema 70: EXPSPACE = NEXPSPACE

Date le classi EXPSPACE e NEXPSPACE:

$$\text{EXPSPACE} = \text{NEXPSPACE}$$

(dimostrazione analoga al teorema precedente)

Definizione 89: Classi coPSPACE e coEXPSPACE

Definiamo le classi dei linguaggi **coPSPACE** e **coEXPSPACE** come:

$$\text{coPSPACE} = \{A \in \text{DEC} \mid \bar{A} \in \text{PSPACE}\}$$

$$\text{coEXPSPACE} = \{A \in \text{DEC} \mid \bar{A} \in \text{EXPSPACE}\}$$

Teorema 71: Chiusura del complemento di PSPACE

Date le classi PSPACE e coPSPACE, si ha che:

$$\text{PSPACE} = \text{coPSPACE}$$

In altre parole, la classe PSPACE è **chiusa nel complemento**

(dimostrazione analoga al teorema 58)

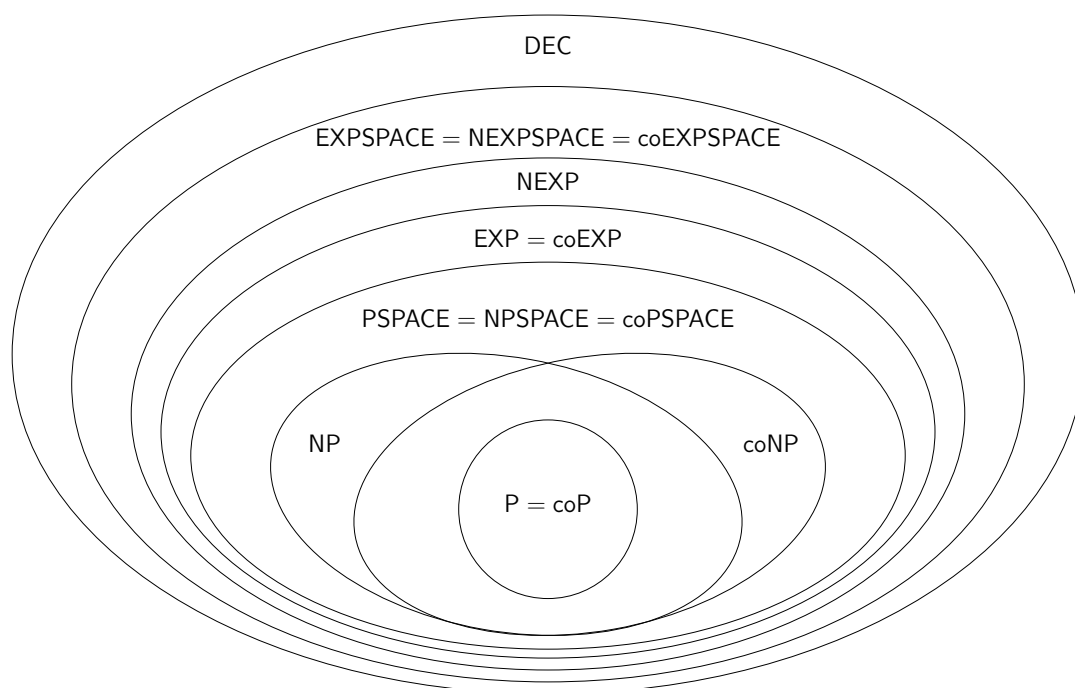
Teorema 72: Chiusura del complemento di EXPSPACE

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 58*)



Gerarchia delle classi dei linguaggi decidibili studiate fino ad ora

4.8 Classe L

Definizione 90: Classi L e NL

Definiamo la **classe dei linguaggi decidibili in spazio logaritmico** come:

$$L = \text{DSPACE}(\log n)$$

Analogamente, definiamo la **classe dei linguaggi decidibili non-deterministicamente in spazio logaritmico** come:

$$NL = \text{NSPACE}(\log n)$$

Corollario 14: Rapporto tra L e P

Date le classi L, NL e P, si ha che:

$$L \subseteq NL \subseteq P$$

(segue dal teorema 66)

Osservazione 18: $NL \stackrel{?}{=} L$

Tramite il [Teorema di Savitch](#), abbiamo che:

$$NL = \text{NSPACE}(\log n) \subseteq \text{DSPACE}(\log^2 n)$$

dunque non sappiamo se $NL = L$

Teorema 73: Classe NL (definizione alternativa)

Data la classe NL, si ha che:

$$NL = \{A \in \text{DEC} \mid \exists V \text{ verificatore in spazio logaritmico per } A\}$$

In altre parole, possiamo definire NL anche come la **classe dei linguaggi verificabili in spazio logaritmico**

Dimostrazione.

Prima implicazione.

- Dato $A \in NL$, 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
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 $A = L(N) \in \text{NL}$, 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 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, N accetta. Altrimenti, rifiuta"

- Per costruzione stessa di N , si ha che

$$w \in A \iff \exists c \in \Sigma^*, \langle w, c \rangle \in V \iff$$

$$\text{Esiste un ramo di } N \text{ che accetta } w \iff w \in A$$

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. Dunque, è sufficiente memorizzare un solo simbolo per volta, richiedendo spazio $O(\log n)$. Inoltre, poiché anche V è un verificatore in spazio logaritmico, concludiamo che $A = L(N) \in \text{NL}$

□

Definizione 91: Classe coL

Definiamo la **classe dei linguaggi coL** come:

$$\text{coL} = \{A \in \text{DEC} \mid \bar{A} \in \text{L}\}$$

Teorema 74: Chiusura del complemento di L

Date le classi L e coL, si ha che:

$$\text{L} = \text{coL}$$

In altre parole, la classe L è **chiusa nel complemento**

(*dimostrazione analoga al teorema 58*)

4.9 Riduzione in spazio logaritmico

Definizione 92: 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 $f : \Sigma^* \rightarrow \Sigma^*$, 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 75: Decidibilità logaritmica tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m^L B$, si ha che:

$$B \in \text{L} \implies A \in \text{L}$$

Dimostrazione.

- Dato $B \in \text{DEC}$, sia D_B il decisore tale che $L(D_B) = B$
- Sia D_A la TM definita come:

$D_A =$ "Data in input la stringa w :

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 $A = L(D_A) \in \mathbf{L}$

□

Teorema 76: Verificabilità logaritmica tramite riduzione

Dati due linguaggi A e B tali che $A \leq_m^L B$, si ha che:

$$B \in \mathbf{NL} \implies A \in \mathbf{NL}$$

(*dimostrazione analoga al teorema precedente*)

Teorema 77: 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 $f, g : \Sigma^* \rightarrow \Sigma^*$ 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:
 $H = \text{"Data la stringa } w \text{ in input:}$
 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 78: 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 41*)

4.10 Classe NL-Complete

Definizione 93: 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:

- $B \in \text{NL}$
- $\forall A \in \text{NL} \quad A \leq_m^L B$

Teorema 79: NL-Completezza di $PATH$

Dato il linguaggio $PATH$, si ha che $PATH \in \text{NL-Complete}$

Dimostrazione.

- Sia N la NTM definita come:
 $N = \text{"Data la stringa } \langle G, s, t \rangle \text{ in input, dove } G = (V, E) \text{ è un grafo:}$
 1. Se $s = t$, *accetta*
 2. Calcola deterministicamente $\langle V \rangle = m$
 3. Inizializza $currNode = s$ su un nastro secondario
 4. Ripeti le seguenti istruzioni per $i = 1, \dots, m$
 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:

$$\langle G, s, t \rangle \in L(N) \iff \text{Esiste cammino } s \rightarrow t \text{ in } G \iff \langle G, s, t \rangle \in PATH$$

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 $PATH = L(N) \in NL$
- Sia quindi $A \in NL$ e sia N' una NTM tale che $A = L(N')$ in spazio $O(\log n)$
- Sia inoltre $f : \Sigma^* \rightarrow \Sigma^*$ 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 64
2. Restituisci in output la stringa $\langle G_{N,w}, c_{\text{start}}, c_{\text{accept}} \rangle$

- Notiamo che:

$$w \in L(N') \iff \text{Esiste cammino } c_{\text{start}} \rightarrow c_{\text{accept}} \text{ in } G_{N,w}$$

$$\iff f(w) = \langle G_{N,w}, c_{\text{start}}, c_{\text{accept}} \rangle \in PATH$$

- 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 $A \leq_m^L PATH$

□

4.10.1 Teorema di Immerman-Szelepcsényi

Lemma 8: Ugualianza tra NL e coNL

Dato un linguaggio $B \in NL\text{-Complete}$, si ha che:

$$B \in \text{coNL} \iff NL = \text{coNL}$$

(dimostrazione analoga al teorema 62)

Teorema 80: Teorema di Immerman-Szelepcsényi

Date le classi NL e coNL, si ha che:

$$NL = \text{coNL}$$

In altre parole, la classe NL è **chiusa nel complemento**

Dimostrazione.

- Sia V la TM definita come:

$V =$ "Data la stringa $\langle w, c \rangle$ in input:

1. Interpreta $w = \langle G, s, t \rangle$, dove $G = (V_G, E_G)$ è un grafo contenente s e t . Se l'interpretazione fallisce, V accetta
2. Calcola $|V_G| = m$
3. Interpreta $c = \langle c_1, \dots, c_m \rangle$, ossia come una serie di sottocertificati, dove per ogni $i \in [0, m]$ si ha che $c_i = \langle s \rightarrow v_1, s \rightarrow v_2, \dots, s \rightarrow v_{k_i} \rangle$, dove $k_i \in [1, m]$
4. Ripeti le seguenti istruzioni per $i = 0, \dots, m$:
 5. Ripeti le seguenti istruzioni per ogni codifica $\langle s \rightarrow v_j \rangle$ all'interno di c_i :
 6. Verifica che $\langle s \rightarrow v_j \rangle$ venga prima lessicograficamente di $\langle s \rightarrow v_{j+1} \rangle$ (ricordiamo che le codifiche sono stringhe) e che siano diverse tra loro. Se falso, *rifiuta*
 7. Verifica che $\langle s \rightarrow v_j \rangle$ si trovi in c_{i+1}, \dots, c_m . Se falso, *rifiuta*
 8. Verifica che $c_0 = \langle s \rightarrow s \rangle$. Se falso, *rifiuta*
9. Ripeti le seguenti istruzioni per $i = 0, \dots, m - 1$:
 10. Ripeti le seguenti istruzioni per ogni arco $(a, b) \in E_G$:
 11. Se $\langle s \rightarrow a \rangle$ si trova in c_i , verifica che $\langle s \rightarrow b \rangle$ si trovi in c_{i+1} . Se falso, *rifiuta*.
 12. Verifica che $\langle t \rangle$ non sia in c_m . Se vero, *accetta*. Altrimenti, *rifiuta*"
- Il loop eseguito dall'istruzione 4 fino all'istruzione 8 verifica che ogni sottocertificato non contenga codifiche ripetute al suo interno e che esso sia un'estensione coerente del sottocertificato precedente, ossia che per ogni $i \in \mathbb{N}$ si abbia che:

$$c_i = \langle s \rightarrow v_1, s \rightarrow v_2, \dots, s \rightarrow v_{k_i} \rangle$$

$$c_{i+1} = \langle s \rightarrow v_1, s \rightarrow v_2, \dots, s \rightarrow v_{k_i}, s \rightarrow v_{k_i+1}, \dots, s \rightarrow v_{k_{i+1}} \rangle$$

dove $\forall j \neq h \in [1, k_{i+1}] \ v_j \neq v_h$

- Definiamo come R_i l'insieme di vertici raggiungibili da s con massimo i archi.
- Assumendo che la verifica del loop 4-8 vada a buon fine, mostriamo per induzione che dato $c_i = \langle v_1, \dots, v_{k_i} \rangle$, dove $i \in \mathbb{N}$, se V non rifiuta mai durante l'esecuzione delle istruzioni 8-11 allora $R_i = \{v_1, \dots, v_{k_i}\}$

Caso base ($i = 0$):

- Se V non rifiuta, allora $c_0 = \langle s \rightarrow s \rangle$. Di conseguenza, poiché $R_0 = \{s\}$, il sottocertificato c_0 indica i vertici di R_0

Ipotesi induttiva:

- Dato $i \in \mathbb{N}$, se V non rifiuta allora c_i indica tutti i vertici di R_i

Passo induttivo:

- Dato un arco $(a, b) \in E_G$, se $\langle s \rightarrow a \rangle$ si trova in c_i , per ipotesi induttiva abbiamo che $a \in R_i$, dunque esiste un cammino da s ad a con massimo i archi.
- Di conseguenza, tramite l'arco (a, b) , possiamo costruire un cammino da s a b con massimo $i + 1$ archi, implicando che $b \in R_{i+1}$
- Dunque, se V non rifiuta, la codifica $\langle s \rightarrow b \rangle$ è presente in c_{i+1} , implicando che quest'ultimo indichi tutti i vertici di R_{i+1}
- Di conseguenza, se V non rifiuta nell'esecuzione delle istruzioni 4-11, ne segue che c_m indichi tutti i vertici di R_m . Inoltre, poiché un cammino in G può avere massimo m archi, R_m contenga tutti e soli i vertici raggiungibili da s nell'intero grafo
- Supponiamo quindi che $w \in PATH$. In tal caso, si ha che:

$$w \in PATH \iff w = \langle G, s, t \rangle \in PATH \iff$$

$$\text{Esiste cammino } s \rightarrow t \text{ in } G \iff t \in R_m$$

Di conseguenza, si ha che:

- Se l'esecuzione dall'istruzione 4 all'istruzione 11 rifiuta, allora c non è un certificato valido, implicando che $\langle w, c \rangle = \langle \langle G, s, t \rangle, c \rangle \notin L(V)$
- Se invece non fallisce, la codifica $\langle s \rightarrow t \rangle$ è presente in c_m per qualsiasi certificato c . implicando che V rifiuti e quindi che $\langle w, c \rangle = \langle \langle G, s, t \rangle, c \rangle \notin L(V)$

Dunque, in ogni caso abbiamo che:

$$w \in PATH \implies \nexists c \in \Sigma^*, \langle w, c \rangle \in L(V)$$

- Supponiamo invece che $w \notin PATH$. In tal caso, si ha che:
 - Se w non è interpretabile come $\langle G, s, t \rangle$, allora $\forall c \in \Sigma^* \quad \langle w, c \rangle \in L(V)$
 - Se w è interpretabile come $\langle G, s, t \rangle$, allora poiché $w = \langle G, s, t \rangle \notin PATH$, esiste un certificato c valido per cui si ha che $\langle w, c \rangle \in L(V)$
- Di conseguenza, si ha che:

$$w \in \overline{PATH} \iff w \notin PATH \iff \exists c \in \Sigma^*, \langle w, c \rangle \in L(V)$$

implicando che V sia un verificatore per \overline{PATH}

- A questo punto, notiamo che all'interno di V vengano utilizzati solo dei contatori per tener traccia delle iterazioni svolte, richiedendo $O(\log n)$ spazio. Dunque, concludiamo che $\overline{PATH} \in \text{NL}$

- Infine, poiché $PATH \in \text{NL-Complete}$, tramite il lemma precedente abbiamo che:

$$\overline{PATH} \in \text{NL} \iff PATH \in \text{coNL} \iff \text{NL} = \text{coNL}$$

□

4.11 Teoremi di gerarchia

4.11.1 Teorema di gerarchia di spazio

Definizione 94: Funzione spazio-costruibile

Definiamo una funzione $f : \mathbb{N} \rightarrow \mathbb{N}$, dove $f(n) \geq \log n$, come **spazio-costruibile** se la seguente funzione:

$$g : \Sigma^* \rightarrow \Sigma^* : 1^n \mapsto f(n)_2$$

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 $g : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM G definita come:
 $G = \text{"Data la stringa } 1^n \text{ in input:}$
 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 $g : \Sigma^* \rightarrow \Sigma^*$ la funzione calcolata dalla seguente TM G definita come:
 $G = \text{"Data la stringa } 1^n \text{ in input:}$
 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^n
- 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 81: Teorema di gerarchia di spazio

Data una funzione spazio-costruibile $f : \mathbb{N} \rightarrow \mathbb{N}$, 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:

$D = \text{"Data la stringa } w \text{ in input:}"$

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
- Per via del limite imposto nell'istruzione 3, lo spazio utilizzato da D può essere massimo $f(n)$
- 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 in spazio $O(f(n))$
- Supponiamo per assurdo che esista un TM M che decida A in spazio $g(n) = o(f(n))$. Per definizione stessa di $g(n) = o(f(n))$, 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 $\frac{1}{d} \in \mathbb{R}_{>0}$, si ha che:

$$\forall n \geq n_0 \quad g(n) < \frac{1}{d} f(n) \implies d \cdot g(n) < f(n)$$

- 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 di D richiede $d \cdot g(n) < f(n)$ spazio, il limite imposto dall'istruzione 3 non verrà mai superato. Di conseguenza, si ha che:

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

contraddicendo l'ipotesi per cui M decida A

- Di conseguenza, ne segue necessariamente che tale TM non possa esistere e dunque che A sia decidibile in $O(f(n))$ ma non in $o(f(n))$

□

Corollario 15: Distinzione spaziale tra le classi

Date due funzioni $f, g : \mathbb{N} \rightarrow \mathbb{N}$, se $f(n) = o(g(n))$ e g è spazio-costruibile, si ha che:

$$\text{DSPACE}(f(n)) \subsetneq \text{DSPACE}(g(n))$$

Dimostrazione.

- Poiché $f(n) = o(g(n))$ implica anche che $f(n) = O(g(n))$, si ha che:

$$\text{DSPACE}(f(n)) \subseteq \text{DSPACE}(g(n))$$

- 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:

$$\text{DSPACE}(f(n)) \subsetneq \text{DSPACE}(g(n))$$

□

Teorema 82: Rapporto tra NL e PSPACE

Date le classi NL e PSPACE, si ha che:

$$\text{NL} \subsetneq \text{PSPACE}$$

Dimostrazione.

- Consideriamo la funzione $f(n) = n$
- Sia $g : \Sigma^* \rightarrow \Sigma^*$ 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 $\text{NL} \subseteq \text{DSPACE}(\log^2 n)$. Notiamo quindi che:

$$\lim_{n \rightarrow +\infty} \frac{\log n}{n} = 0$$

implicando che $\log n = o(n)$

- Di conseguenza, per la [Distinzione spaziale tra le classi](#), concludiamo che:

$$\text{NL} \subseteq \text{DSPACE}(\log^2 n) \subsetneq \text{DSPACE}(n) \subseteq \text{PSPACE}$$

□

Teorema 83: Rapporto tra PSPACE e EXPSPACE

Date le classi PSPACE e EXPSPACE, si ha che:

$$\text{PSPACE} \subsetneq \text{EXPSPACE}$$

Dimostrazione.

- Consideriamo la funzione $f(n) = 2^n$
- Sia $g : \Sigma^* \rightarrow \Sigma^*$ 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:

$$\forall k \in \mathbb{N} \quad \lim_{n \rightarrow +\infty} \frac{n^k}{n^{\log n}} = n^{k - \log n} = 0$$

poiché $k - \log n$ tende a $-\infty$ per $n \rightarrow +\infty$, implicando che $n^k = o(n^{\log n})$

- Analogamente, si ha che:

$$\lim_{n \rightarrow +\infty} \frac{n^{\log n}}{2^n} = \lim_{n \rightarrow +\infty} \frac{2^{\log(n^{\log n})}}{2^n} = \lim_{n \rightarrow +\infty} \frac{2^{\log^2 n}}{2^n} = \lim_{n \rightarrow +\infty} 2^{\log^2 n - n} = 0$$

poiché $\log^2 n - n$ tende a $-\infty$ per $n \rightarrow +\infty$, implicando che $n^{\log n} = o(2^n)$

- Di conseguenza, per la [Distinzione spaziale tra le classi](#), otteniamo che:

$$\forall k \in \mathbb{N} \quad \text{DSPACE}(n^k) \subsetneq \text{DSPACE}(n^{\log n}) \subsetneq \text{DSPACE}(2^n)$$

implicando che:

$$\exists L \in \text{DSPACE}(2^n) \subseteq \text{EXPSPACE} \mid \nexists k \in \mathbb{N} \text{ t.c. } L \in \text{DSPACE}(n^k) \implies L \notin \text{PSPACE}$$

concludendo che $\text{PSPACE} \subsetneq \text{EXPSPACE}$

□

4.11.2 Teorema di gerarchia di tempo

Definizione 95: Funzione tempo-costruibile

Definiamo una funzione $f : \mathbb{N} \rightarrow \mathbb{N}$, dove $f(n) \geq \log n$, come **tempo-costruibile** se la seguente funzione:

$$g : \Sigma^* \rightarrow \Sigma^* : 1^n \mapsto f(n)_2$$

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 84: Teorema di gerarchia di tempo

Data una funzione spazio-costruibile $f : \mathbb{N} \rightarrow \mathbb{N}$, 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 omessa, simile alla dimostrazione del teorema 81)

Corollario 16: Distinzione temporale tra le classi

Date due funzioni $f, g : \mathbb{N} \rightarrow \mathbb{N}$, se $f(n) = o\left(\frac{g(n)}{\log(g(n))}\right)$ e g è tempo-costruibile, si ha che:

$$\text{DSPACE}(f(n)) \subsetneq \text{DSPACE}(g(n))$$

Dimostrazione.

- Poiché $f(n) = o\left(\frac{g(n)}{\log(g(n))}\right)$ implica anche che $f(n) = O(g(n))$, si ha che:

$$\text{DSPACE}(f(n)) \subseteq \text{DSPACE}(g(n))$$

- 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:

$$\text{DSPACE}(f(n)) \subsetneq \text{DSPACE}(g(n))$$

□

Teorema 85: Rapporto tra P e EXP

Date le classi P e EXP, si ha che:

$$P \subsetneq \text{EXP}$$

Dimostrazione.

- TODO

□

4.12 Esercizi svolti

Problema 20: Riducibilità quasi-polinomiale

Sia QP la classe dei linguaggi decidibili in tempo quasi-polinomiale, definita come:

$$\text{QP} = \bigcup_{k=1}^{\infty} \text{DTIME}(n^{\log^k n})$$

Dati due linguaggi A e B tali che $A \leq_m^P B$, stabilire se la seguente affermazione è vera o falsa:

$$B \in \text{QP} \implies A \in \text{QP}$$

Dimostrazione.

- Sia $f : \Sigma^* \rightarrow \Sigma^*$ 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 $B \in \text{QP}$, sia D_B il decisore che decide B in tempo $O(n^{\log^j n})$, dove $j \in \mathbb{N}$
- Notiamo quindi che:

$$\lim_{n \rightarrow +\infty} \frac{n^k}{n^{\log^j n}} = \lim_{n \rightarrow +\infty} n^{k - \log^j n} = 0$$

poiché $k - \log^j n$ tende a $-\infty$ per $n \rightarrow +\infty$, 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 $A = L(D_A) \in \text{DEC}$.

- Inoltre, poiché ogni istruzione di D_A è eseguibile in tempo $O(n^{\log^j n})$, concludiamo che $A = L(D_A) \in \text{DTIME}(n^{\log^j n}) \subseteq \text{QP}$
- Dunque, stabiliamo che l'affermazione sia vera

□

Problema 21: Problema dei k -cammini

Dato il seguente linguaggio:

$$k\text{-}PATH = \{\langle G, s, t, k \rangle \mid G = (V_G, E_G) \text{ grafo con un cammino } s \rightarrow t \text{ di } k \text{ archi}\}$$

dimostrare che $k\text{-}PATH \in \text{NL}$

Dimostrazione.

- Sia V la TM definita come:
 - $V = \text{"Data la stringa } \langle \langle G, s, t, k \rangle, c \rangle \text{ in input:}$
 1. Interpreta $c = \langle v_1, \dots, v_m \rangle$.
 2. Calcola m tramite c . Se $m \neq k$, *rifiuta*
 3. Verifica che $s = v_1$ e che $t = v_m$. Se falso, *rifiuta*
 4. Ripeti le seguenti istruzioni per $i = 1, \dots, m - 1$:
 1. Verifica se $(v_i, v_{i+1}) \in E_G$. Se falso, *rifiuta*
 5. *Accetta*"

- Per costruzione di V , risulta evidente che:

$$\begin{aligned} \langle G, s, t, v \rangle \in k\text{-}PATH &\iff \text{Esiste un cammino } s \rightarrow t \text{ di } k \text{ archi} \\ &\iff \exists c \in \Sigma^*, \langle \langle G, s, t, k \rangle, c \rangle \in L(V) \end{aligned}$$

implicando che V sia un verificatore per $k\text{-}PATH$

- A questo punto, notiamo che per l'esecuzione di V siano sufficienti tre contatori, implicando che lo spazio necessario sia $O(\log n)$
- Di conseguenza, poiché V è un verificatore in spazio logaritmico per $k\text{-}PATH$, concludiamo che $k\text{-}PATH \in \text{NL}$

□

Problema 22: $P = NP = NP\text{-Complete}$

Dimostrare che se $P = NP$ allora $NP = NP\text{-Complete}$

Dimostrazione.

- Poiché $NP\text{-Complete} = NP \cap NP\text{-Hard}$, si ha che $NP\text{-Complete} \subseteq NP$
- Consideriamo quindi $B \in NP = P$
- Siano quindi $yes, no \in \Sigma^*$ due stringhe tali che $yes \in B$ e $no \notin B$
- Dato un qualsiasi linguaggio $A \in NP = P$, sia D_A il decisore in tempo polinomiale tale che $A = L(D_A)$
- Sia quindi $f : \Sigma^* \rightarrow \Sigma^*$ 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 yes
3. Altrimenti, restituisci in output la stringa no "

- Notiamo quindi che:

$$w \in A = L(D_A) \implies f(w) = yes \in B$$

e inoltre che:

$$w \notin A = L(D_A) \implies f(w) = no \notin B$$

- Poiché l'esecuzione di D_A viene svolta in tempo polinomiale, ne segue che f sia calcolabile in tempo polinomiale, implicando che

$$\forall A \in NP = P \quad A \leq_m^P B$$

concludendo che $B \in NP\text{-Complete}$

□