"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 novembre 2023

Indice

Informazioni e Contatti

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

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

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

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

Prerequisiti consigliati per lo studio:

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

1

Linguaggi regolari

1.1 Linguaggi

Definizione 1: Alfabeto

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

Esempio:

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

Definizione 2: Stringa

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

$$w := w_1 \ldots w_n$$

come **stringa (o parola) di** Σ

Esempio:

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

Definizione 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:

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

Proposizione 1: Stringa vuota

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

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

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**:

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

- **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*)

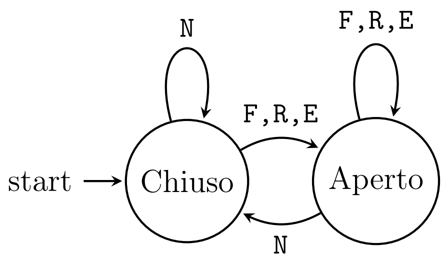
1.2 Determinismo

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`)

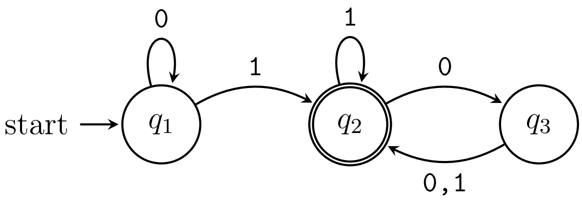
Definizione 10: Deterministic Finite Automaton (DFA)

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

Esempio:

- Consideriamo il seguente DFA



dove:

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

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

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

Definizione 11: Funzione di transizione estesa

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

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

Proposizione 3: Stringa accettata in un DFA

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

Esempio:

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

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

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

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

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

Definizione 12: Linguaggio di un automa

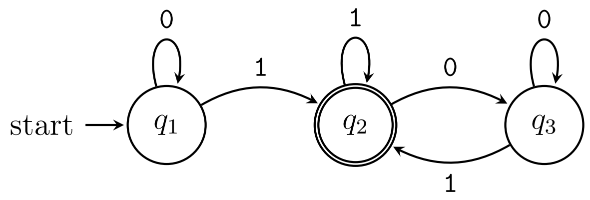
Sia A un automa. Definiamo come **linguaggio di** A, indicato come $L(A)$, l'insieme di stringhe accettate da A

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

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

Esempi:

1. - Consideriamo il seguente DFA D



   - Il linguaggio riconosciuto da tale DFA corrisponde a

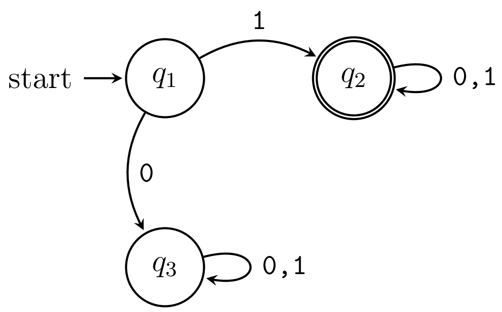
$$L(D) = \{x \in \{0,1\}^* \mid x := y1, \exists y \in \{0,1\}^*\}$$

   ossia al linguaggio composto da tutte le stringhe terminanti con 1

2. - Consideriamo il seguente linguaggio

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

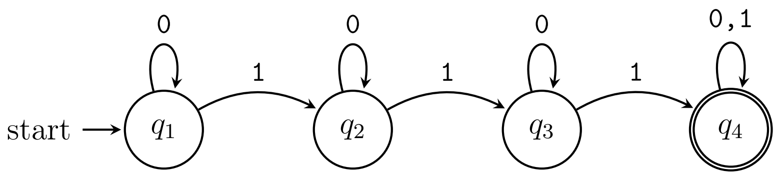
   - Un DFA in grado di riconoscere tale linguaggio corrisponde a



3. • Consideriamo il seguente linguaggio

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

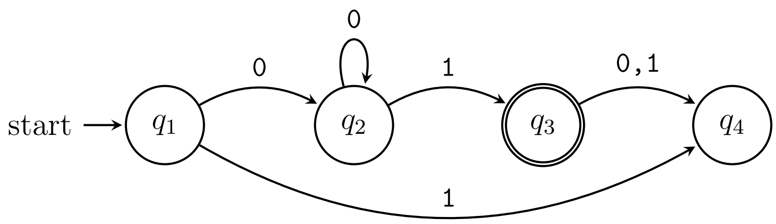
• Un DFA in grado di riconoscere tale linguaggio corrisponde a



4. • Consideriamo il seguente linguaggio

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

• Un DFA in grado di riconoscere tale linguaggio corrisponde a



Definizione 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:

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

Osservazione 1

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)

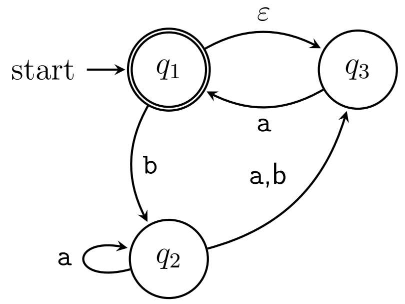
Un **Non-deterministic Finite Automaton (NFA)** (o *Automa Non-deterministico a Stati Finiti*) è una quintupla $(Q, \Sigma, \delta, q_0, F)$ dove:

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

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

Esempio:

- Consideriamo il seguente NFA



dove:

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

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

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

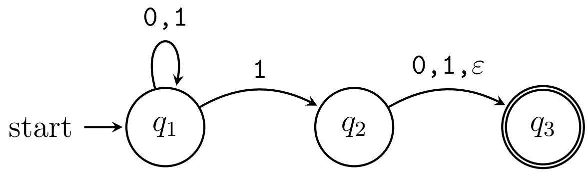
Osservazione 2: Computazione in un NFA

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:

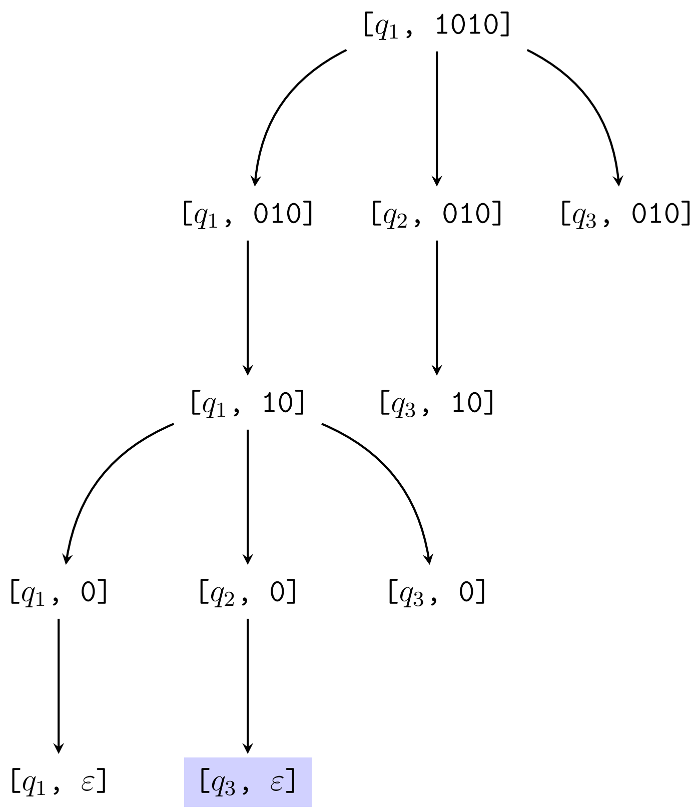
- Tutte le volte che uno stato potrebbe avere più transizioni per diversi simboli dell'alfabeto, l'automa N si duplica in **più copie**, ognuna delle quali segue il suo corso. Si vengono così a creare più **rami di computazione** indipendenti che sono eseguiti in **parallelo**.
- Se il prossimo simbolo 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:

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

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 di 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:
  $$E(R) = \{q \in Q_N \mid \text{q è raggiungibile in } N \text{ da } q' \in R \text{ tramite } k \geq 0 \text{ } \varepsilon\text{-archi}\}$$
  - $q_{0_D} = E(\{q_{0_N}\})$
  - $F_D = \{R \in Q_D \mid R \cap F_N \neq \varnothing\}$

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

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

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

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

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

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

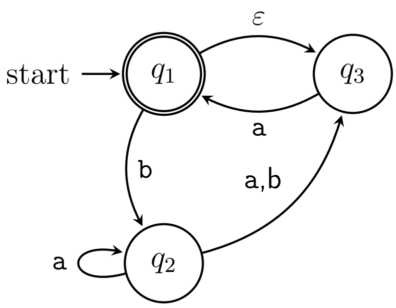
$\square$

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:

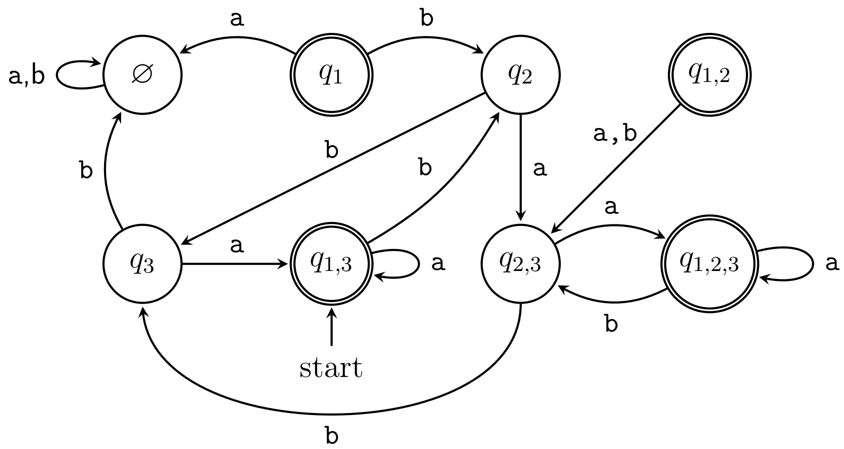
$$Q_D = \{\varnothing, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_2, q_3\}, \{q_1, q_3\}, \{q_1, q_2, q_3\}\} =$$

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

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

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

- Le transizioni del DFA corrisponderanno invece a:
  - $\delta_D(\{q_1\}, a) = E(\delta_N(q_1,), a) = \varnothing$
  - $\delta_D(\{q_1\}, b) = E(\delta_N(q_1,), b) = \{q_2\} = q_2$
  - $\delta_D(\{q_2\}, a) = E(\delta_N(q_2,), a) = \{q_2, q_3\} = q_{2,3}$
  - $\delta_D(\{q_2\}, b) = E(\delta_N(q_2,), b) = \{q_2\} = q_2$
  - $\delta_D(\{q_1, q_2\}, a) = E(\delta_N(q_1, a)) \cup E(\delta_N(q_2, a)) = \varnothing \cup \{q_2, q_3\} = \{q_2, q_3\} = q_{2,3}$
  - $\delta_D(\{q_1, q_2\}, b) = E(\delta_N(q_1, b)) \cup E(\delta_N(q_2, b)) = \{q_2\} \cup \{q_3\} = \{q_2, q_3\} = q_{2,3}$
  - ...
- 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:

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

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 $L_2 = L(M_2)$
- Definiamo quindi il NFA $N = (Q, \Sigma, \delta, q_0, F)$ tale che:
  - q_0 è un nuovo stato iniziale aggiunto
  - $Q = Q_1 \cup Q_2 \cup \{q_0\}$
  - $F = F_1 \cup F_2$
  - $\forall q \in Q, a \in \Sigma$ si ha che:

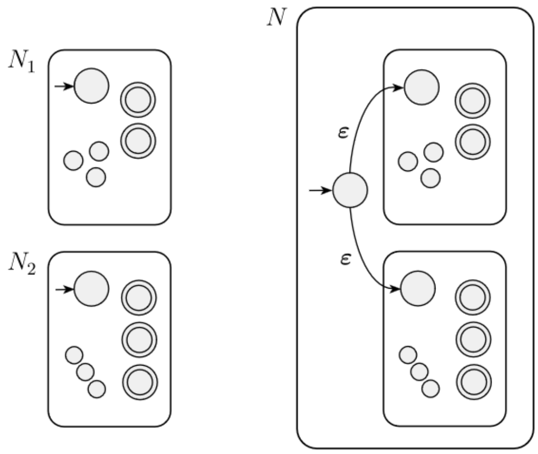
$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{se } q \in Q_1 \\ \delta_2(q, a) & \text{se } q \in Q_2 \\ \{q_1, q_2\} & \text{se } q = q_0 \wedge a = \varepsilon \\ \varnothing & \text{se } q = q_0 \wedge a \neq \varepsilon \end{cases}$$

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

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

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

$\square$



Rappresentazione grafica della dimostrazione

Teorema 4: Chiusura dell'intersezione in REG

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

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

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

$\square$

Teorema 5: Chiusura del complemento in REG

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

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

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 $\overline{L} = L(D') \in \text{REG}$

□

Teorema 6: Chiusura della concatenazione in REG

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

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

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:

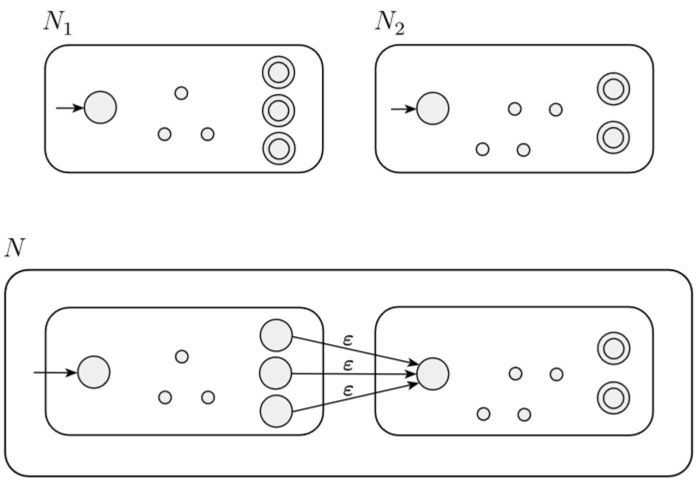
$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{se } q \in Q_1 - F_1 \\ \delta_1(q, a) & \text{se } q \in F_1 \wedge a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_2\} & \text{se } q \in F_1 \wedge a = \varepsilon \\ \delta_2(q, a) & \text{se } q \in Q_2 \end{cases}$$

- 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:

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

Teorema 7: Chiusura di star in REG

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

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

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:

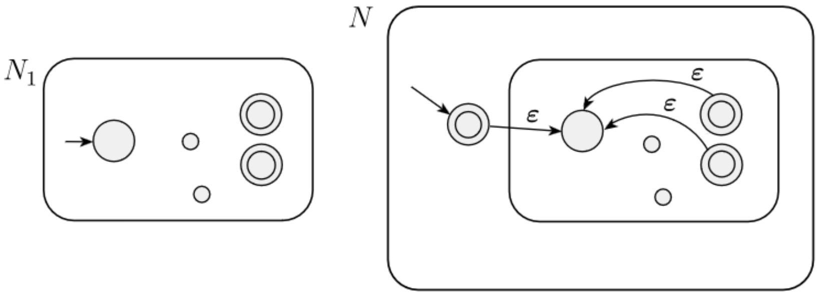
$$\delta'(q,a) = \begin{cases} \delta(q,a) & \text{se } q \in Q - F \\ \delta(q,a) & \text{se } q \in F \wedge a \neq \varepsilon \\ \delta(q,a) \cup \{q_0\} & \text{se } q \in F \wedge a = \varepsilon \\ \{q_0\} & \text{se } q = q_{0*} \wedge a = \varepsilon \\ \varnothing & \text{se } q = q_{0*} \wedge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

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

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

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

$\square$



Rappresentazione grafica della dimostrazione

Corollario 2: Chiusura di plus in REG

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

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

Dimostrazione.

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

$\square$

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 $\mathrm{re}(\Sigma)$, come:

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

Osservazione 5

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

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

Esempi:

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

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

Definizione 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 di linguaggi $\mathcal{L}(\text{re})$ e $\mathcal{L}(\text{NFA})$, si ha che:

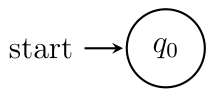
$$\mathcal{L}(\text{re}) \subseteq \mathcal{L}(\text{NFA})$$

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

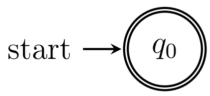
Caso base.

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



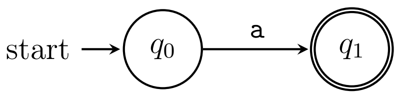
per cui si ha che $w \in L(R) \iff w \in L(N_\varnothing)$ dunque $L(R) = L(N_\varnothing) \in \mathcal{L}(\text{NFA})$

- 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 $\exists$ 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.

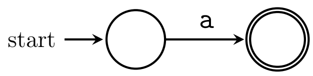
- Se $R = R_1 \cup R_2$, tramite la Chiusura dell'unione in REG, otteniamo che:
$$L(R) = L(R_1) \cup L(R_2) = L(N_1) \cup L(N_2) \in \text{REG} = \mathcal{L}(\text{NFA})$$
- Se $R = R_1 \circ R_2$, tramite la Chiusura della concatenazione in REG, otteniamo che:
$$L(R) = L(R_1) \circ L(R_2) = L(N_1) \circ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

$\square$

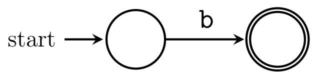
Esempio:

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

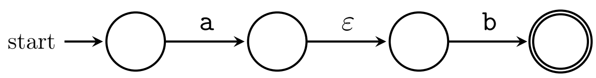
$a \implies$



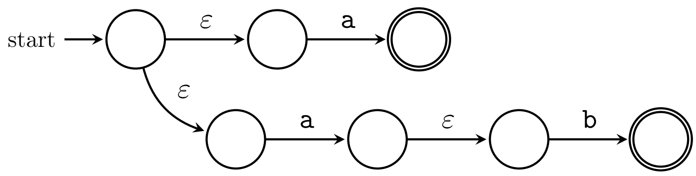
$b \implies$



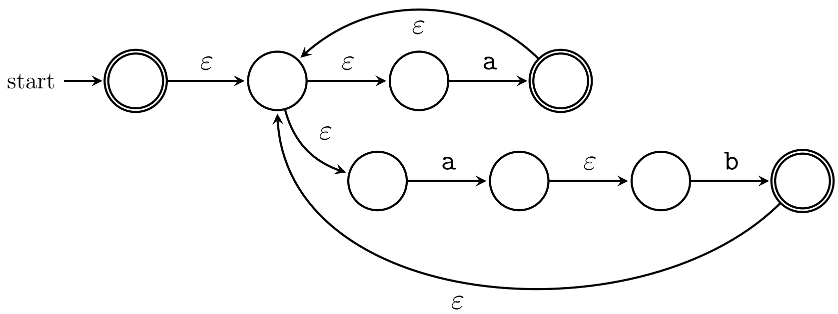
$ab \implies$



$(a \cup ab) \implies$



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



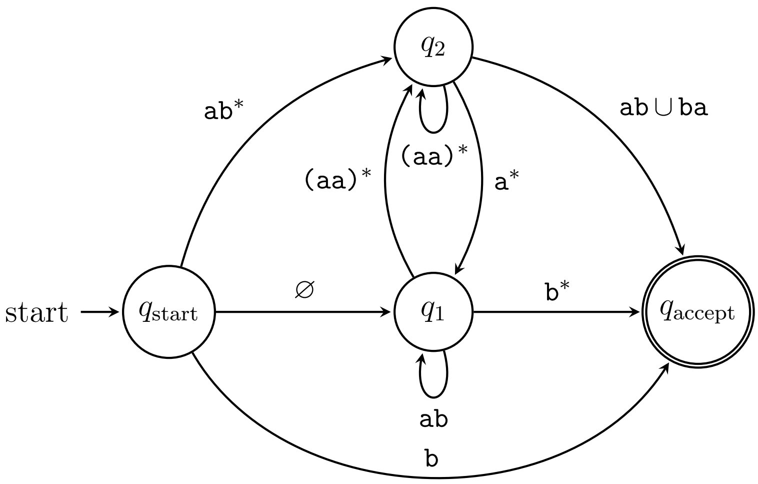
1.5.1 NFA generalizzati

Definizione 23: Generalized NFA (GNFA)

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

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

Esempio:



Osservazione 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:

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

Esempio:

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

Corollario 3

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

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 di linguaggi $\mathcal{L}(\text{DFA})$ e $\mathcal{L}(\text{GNFA})$, si ha che:

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

Dimostrazione.

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

  - $\forall q \in F \;\; \delta'(q, q_{\text{accept}}) = \varepsilon$
  - Per ogni transizione con etichetta multipla in D, in G esiste una transizione equivalente con etichetta corrispondente all'unione di tali etichette multiple
  - Per ogni coppia di stati per cui non esiste una transizione entrante o uscente in D, viene aggiunta una transizione con etichetta $\varnothing$

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

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

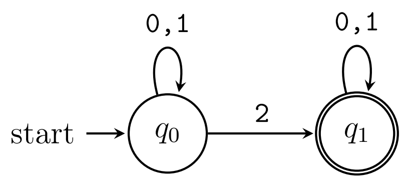
implicando dunque che $L(D) \in \mathcal{L}(\text{DFA})$ e di conseguenza che:

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

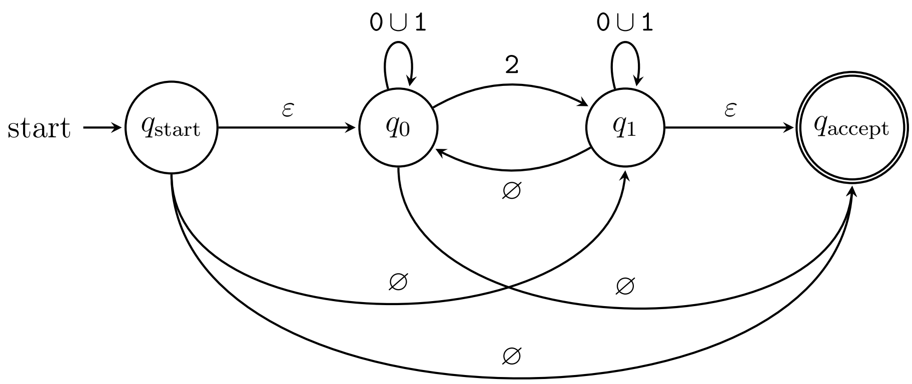
□

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

Caso base.

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

Ipotesi induttiva.

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

Passo induttivo.

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

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

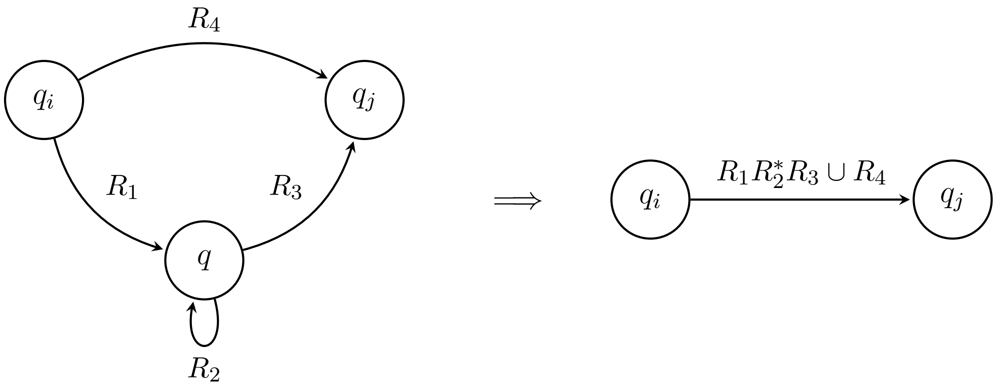
  per ogni $q_i \in Q' - \{q_{\text{accept}}\}, q_j \in Q' - \{q_{\text{start}}\}$

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

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

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

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

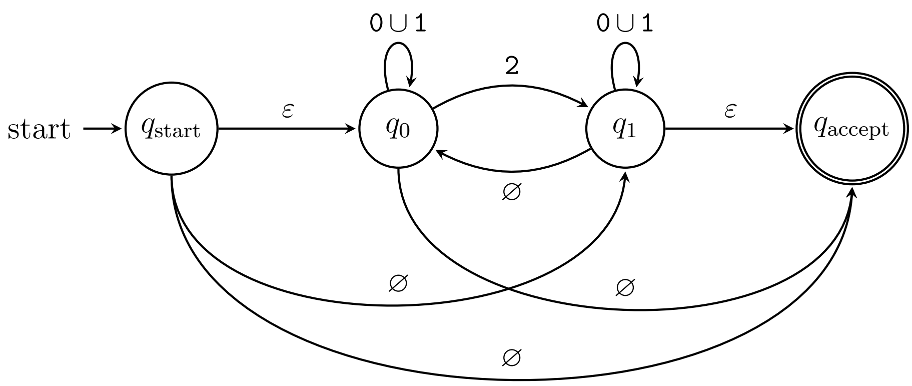


- Di conseguenza, otteniamo che $w \in L(G_k) \iff w \in L(G_{k+1})$, concludendo quindi, per ipotesi induttiva, che $L(G) = L(G_k) = L(G_{k+1})$

□

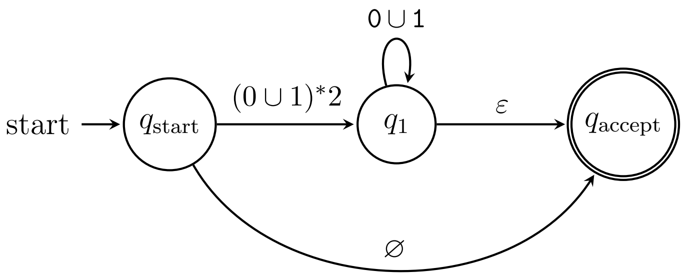
Esempio:

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



- Rimuoviamo quindi lo stato q_0 calcolando le nuove transizioni:

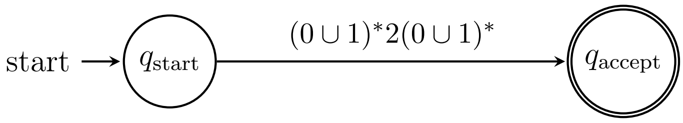
$$\delta'(q_{\text{start}}, q_1) = \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0, q_1) \cup \delta(q_{\text{start}}, q_1) = \varepsilon(0 \cup 1)^*2 \cup \varnothing = (0 \cup 1)^*2$$
$$\delta'(q_{\text{start}}, q_{\text{accept}}) = \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0,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



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

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

- Il GNFA minimale, dunque, corrisponde a:



Corollario 4: Conversione da GNFA ad espressione regolare

Date le due classi di 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 di 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:

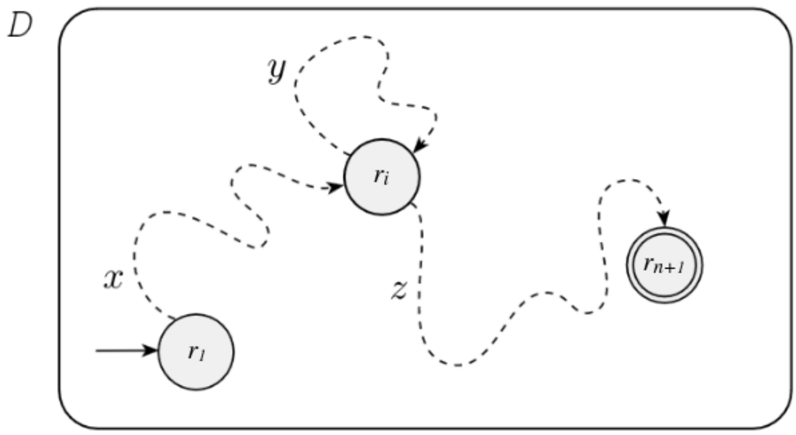
- $\forall i \in \mathbb{N} \;\; xy^i z \in L$, ossia è possibile concatenare y per i volte rimanendo in L
- $|y| > 0$, dunque $y \neq \varepsilon$
- $|xy| \leq p$, ossia y deve trovarsi nei primi p simboli di w

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:
$$\forall k \in [1, n] \;\; \delta(r_k, w_k) = r_{k+1}$$
- 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 $\delta^*(r_j, z) = r_n$
- Poiché $\delta^*(r_i, y) = r_i$, ossia y porta sempre r_i in se stesso, ne segue automaticamente che
$$\forall k \in \mathbb{N} \;\; \delta^*(r_i, y^k) = r_i \implies \delta(r_1, xy^k z) \in F \implies xy^k z \in L(D) = L$$

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

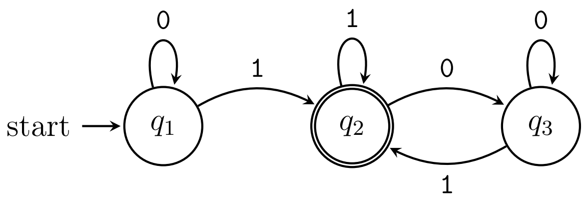
$\square$



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 $0000 \ldots 101010 \ldots 1111$
     - Di conseguenza, poiché in ogni caso viene contraddetto il pumping lemma, ne segue necessariamente che L non sia regolare
  2. **Approccio condizionale**:
     - Poiché la terza condizione del pumping lemma impone che $|xy| \leq p$ e poiché $w := 0^p 1^p$, ne segue che $xy = 0^m$ e $z = 0^{p-m} 1^p$, dove $m \in [1, p]$
     - Inoltre, per la seconda condizione, si ha che $|y| > 0$, dunque necessariamente si ha che $x = 0^{m-k}$ e $y = 0^k$, dove $k \in [1, m]$
     - A questo punto, consideriamo la stringa $xy^0 z$. 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 $\forall i \in \mathbb{N} \;\; xy^i z \in L$
     - 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
  - $Q' = Q \cup \{q_f\}$
  - $\forall q \in Q, a \in \Sigma \;\; \delta'(q, a) = \delta(q, a)$, ossia tutti gli archi rimangono invariati
  - $\forall q \in F \;\; \delta'(q, \varepsilon) = q_f$, ossia tutti gli stati finali precedenti hanno un ε-arco verso q_f
- A questo punto, per costruzione stessa di N ne segue che:

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

  dunque che $L = L(D) = L(N)$
- Definiamo quindi un secondo NFA $N^R = (Q', \Sigma, \delta'', q_f, \{q_0\})$ tale che:

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

  ossia avente tutti gli archi invertiti rispetto ad N
- A questo punto, per costruzione stessa di N' ne segue che:

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

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

$\square$

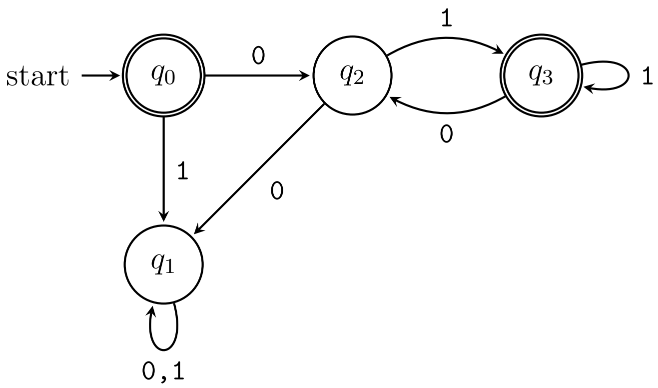
Problema 2: Complemento di un'espressione regolare

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

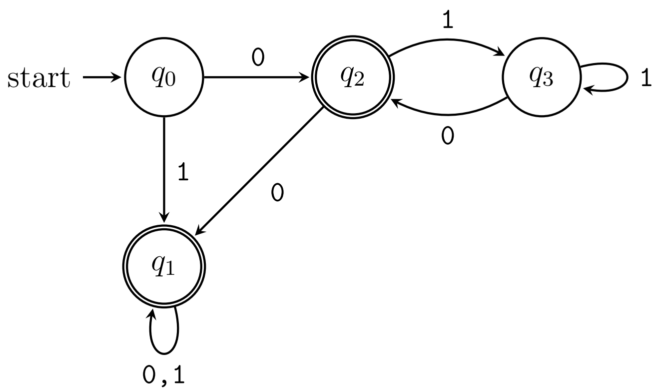
$$L(D) = \{w \in \{0,1\}^* \mid w \notin L(R)\}$$

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 ch 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^0z. Notiamo immediatamente che
$$xy^0z = 0^{m-k}(0^k)^0 0^{p-m}1^p = 0^{m-k}0^{p-m}1^p = 0^{p-k}1^p$$
$$\implies \left|xy^0z\right|_0 \neq \left|xy^0z\right|_1 \implies xy^0z \notin L$$
contraddicendo la prima condizione del lemma per cui si ha che $\forall i \in \mathbb{N} \;\; xy^iz \in L$
- Dunque, ne segue necessariamente che L non sia regolare

$\square$

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 ch per esso debba valere il pumping lemma, dove p è la lunghezza del pumping
- Consideriamo quindi la stringa $w := 1^{p^2} \in L$. Poiché $|w| \geq p$, possiamo suddividerla in tre sottostringhe $x, y, z \in \Sigma^*$ tali che $w = xyz$
- Poiché la terza condizione del lemma impone che $|xy| \leq p$ e poiché $w := 1^{p^2}$, ne segue che $xy = 1^m$ e $z = 1^{p^2-m}$, dove $m \in [1, p]$
- Inoltre, per la seconda condizione del lemma, si ha che $|y| > 0$, dunque necessariamente si ha che $x = 1^{m-k}$ e $y = 1^k$, dove $k \in [1, m]$
- A questo punto, consideriamo la stringa xy^0z. Notiamo immediatamente che
$$xy^0z = 1^{m-k}(1^k)^0 1^{p^2-m} = 1^{p^2-k}$$
- Tuttavia, poiché $k \in [1, p]$, ne segue che $\nexists n \in \mathbb{N} \mid n^2 = p^2 - k$, implicando dunque che $xy^0z \notin L$, contraddicendo la prima condizione del lemma per cui si ha che $\forall i \in \mathbb{N} \;\; xy^iz \in L$
- Dunque, ne segue necessariamente che L non sia regolare

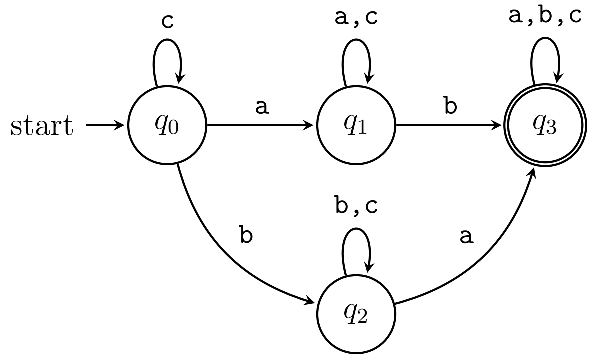
$\square$

Problema 5

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:

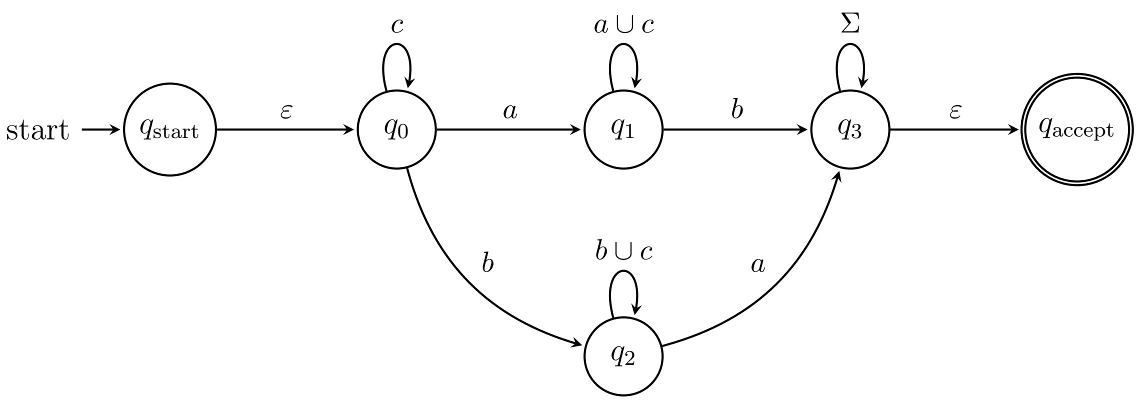
- Nonostante il problema inviti alla determinazione dell'espressione regolare e poi del DFA ad essa equivalente, trovare quest'ultimo risulta molto più rapido
- Difatti, il DFA D in grado di riconoscere il linguaggio richiesto corrisponde a:



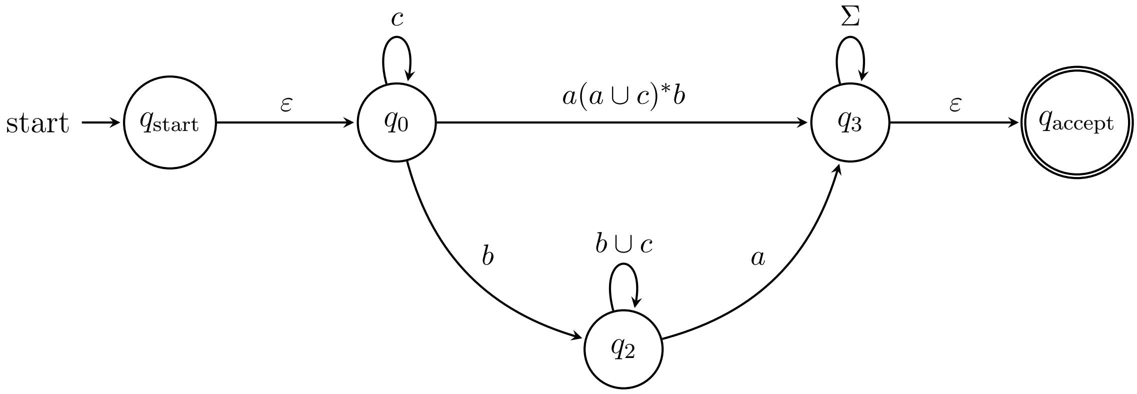
- A questo punto, osservando il DFA possiamo già notare che l'espressione regolare ad esso equivalente corrisponde a:

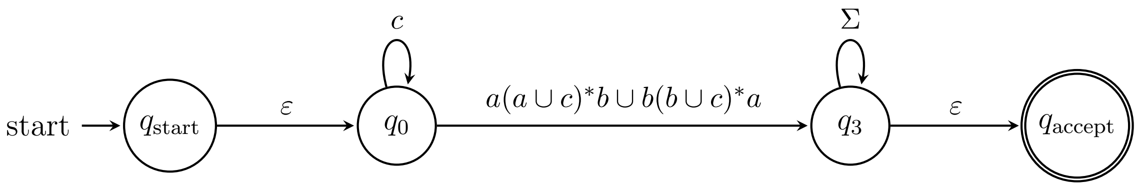
$$c^*(a(a \cup c)^* b \cup b(a \cup c)^* a)\Sigma^*$$

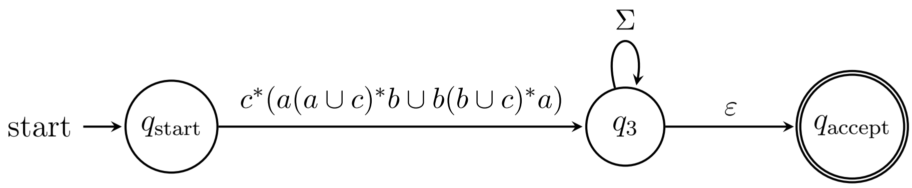
- Volendo procedere più rigorosamente, possiamo ricavare tale espressione regolare convertendo il DFA costruito nel suo GNFA equivalente, per poi ridurre al minimo tale GNFA, ottenendo l'espressione regolare
- Definiamo quindi il GNFA equivalente (del quale vengono omesse le sue transizioni etichettate con $\varnothing$):

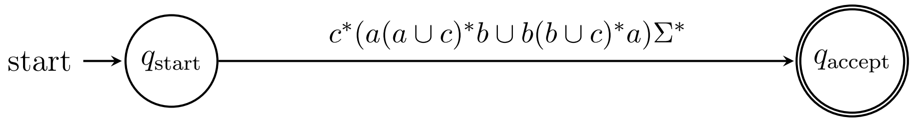


- Procediamo quindi con la riduzione:









- Come anticipato, l'espressione regolare ottenuta corrisponde a:

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

2

Linguaggi acontestuali

2.1 Grammatiche acontestuali

Definizione 25: Context-freee Grammar (CFG)

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

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

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

Esempio:

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

$$A \to 0A1$$

$$A \to B$$

$$B \to \#$$

Osservazione 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 $A \to B$ può appartenere ad una CFG
- La regola $AB \to B$ 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 $A \to X_1, X_2, \ldots, A \to X_n$ definite sulla stessa variabile A,è possibile indicare tali regole con la seguente notazione contratta:

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

Esempio:

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

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

$$B \to \#$$

Definizione 26: Produzione

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

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

Esempio:

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

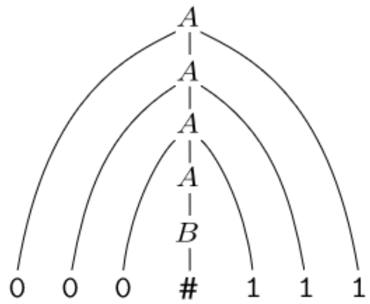
$$A \to 0A1 \mid B$$

$$B \to \#$$

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

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

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



Definizione 27: Derivazione

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

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

Definizione 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:

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

Esempi:

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

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

   si ha che:

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

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

$$S \to T1T1T1T$$

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

   si ha che:

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

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

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

si ha che:

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

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

$$S \to aSc \mid T$$

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

si ha che:

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

Osservazione 10

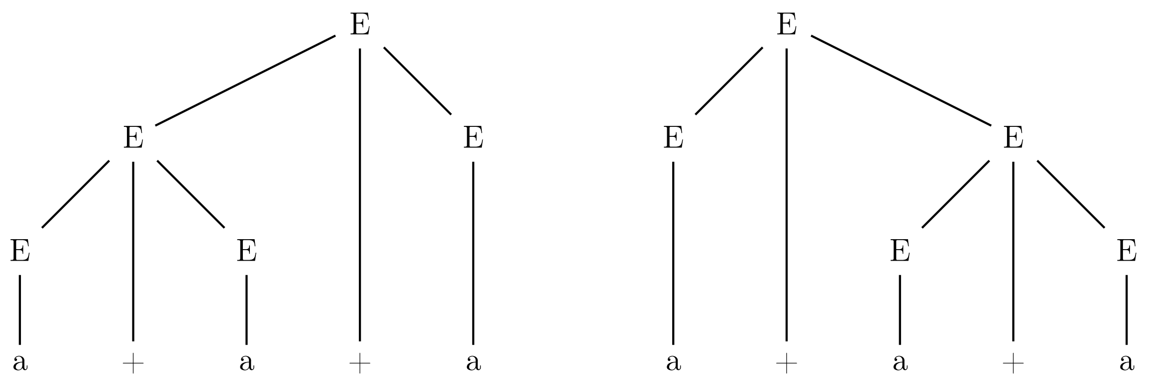
Sia G una CFG. Data la stringa $w \in L(G)$, possono esistere più derivazioni di w

Esempio:

- Data la CFG

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

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



Definizione 29: Derivazione a sinistra

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

Esempio:

- Riprendiamo la CFG dell'esempio precedente:

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

- Per maggior chiarezza, riscriviamo tali regole come:

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

$$F \to E$$

ottenendo una CFG del tutto equivalente alla precedente

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

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

Osservazione 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 di 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 $\varphi : Q \to V : q_i \mapsto V_i$

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

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

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

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

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

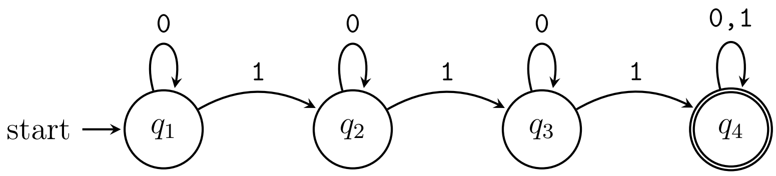
implicando dunque che $L(D) \in$ CFL e di conseguenza che:

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

□

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:

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

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

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

Teorema 9: Ling. acontestuali estensione dei ling. regolari

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

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

Dimostrazione.

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

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

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

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

$\square$

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:

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

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 $S_0 \to S$, dove S_0 è la **nuova variabile iniziale**
  2. Finché in R esiste una ε**-regola** $A \to \varepsilon$ dove $A \in V - \{S_0\}$, tale regola viene **eliminata** e per ogni regola in R contenente delle occorrenze di A vengono **aggiunte** delle regole in cui vengono eliminate tutte le possibili combinazioni di occorrenze di A

     (es: se viene rimossa $A \to \varepsilon$ e in R esiste $B \to uAvAw \mid u, v, w \in (V \cup \Sigma)^*$, vengono aggiunte le regole $B \to uvAw \mid uAvw \mid uvw$)
  3. Ogni regola nella forma $A \to B$ (dette **regole unitarie**) per cui esiste una regola nella forma $B \to u \mid u \in (V \cup \Sigma)^*$ viene **sostituita** con la regola $A \to u$
  4. Per ogni regola $A \to u_1 \dots u_k$ dove $k \geq 3$ e $u \in (V \cup \Sigma)$, vengono **aggiunte** le variabili $A_1, \dots, A_k$ e le seguenti regole:

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

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

$\square$

Esempio:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$ è la **funzione di transizione** dell'automa, dove se $(q, c) \in \delta(p, a, b)$ si ha che:
  - Viene letto il simbolo a dalla stringa in input e se il simbolo b è in cima allo stack allora l'automa passa dallo stato p allo stato q e il simbolo b viene sostituito dal simbolo c
  - L'etichetta della transizione da p a q viene indicata come $a;\ b \rightarrow c$

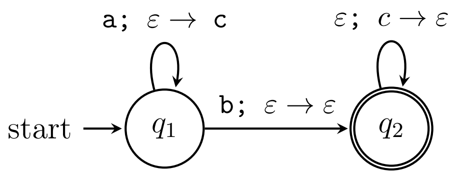
Osservazione 12

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

Esempio:

- Consideriamo il seguente PDA:



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

Proposizione 8: Stringa accettata in un PDA

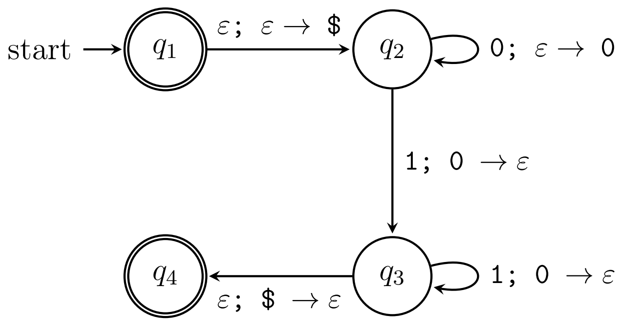
Sia $P := (Q, \Sigma, \Gamma, \delta, q_0, F)$ un PDA. Data una stringa $w := w_0 \dots w_k \in \Sigma^*$, dove $w_0, \dots, w_k \in \Sigma_\varepsilon$, diciamo che w è **accettata da** P se esiste una sequenza di stati $r_0, r_1, \dots, r_{k+1} \in Q$ ed una sequenza di stringhe $s_1, \dots, s_n \in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

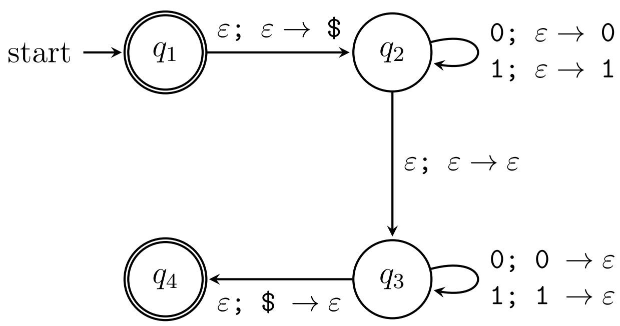
  dove $a, b \in \Gamma_\varepsilon$ e dove $t \in \Gamma^*$ è la stringa composta dai caratteri nello stack

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 $u := u_1 \dots u_k$.

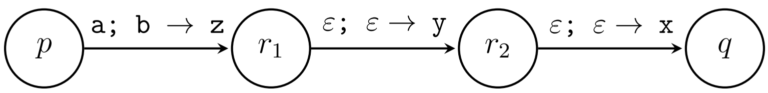
Formalmente, diciamo che:

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

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

Esempio:

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



Lemma 5: Conversione da CFG a PDA

Date le due classi di 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

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

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

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

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

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

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

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

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

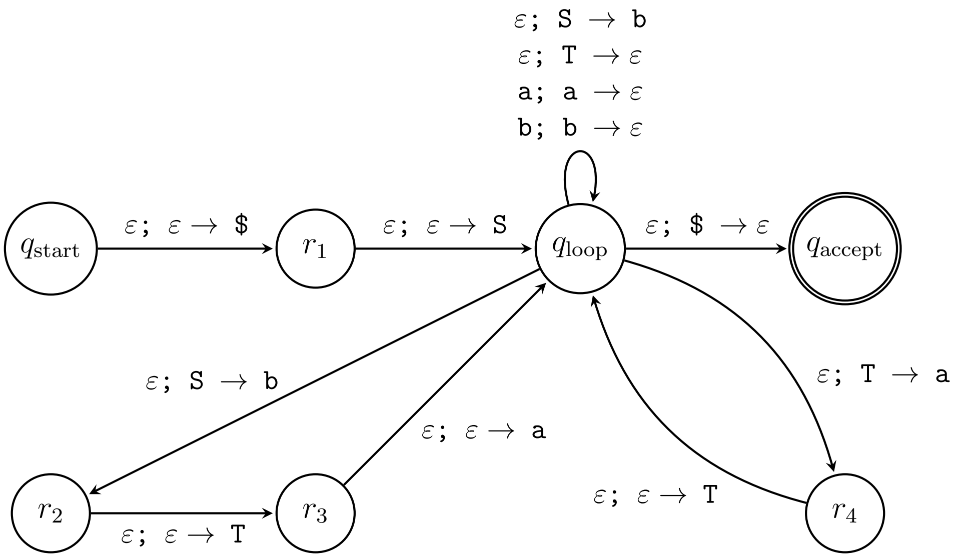
$\square$

Esempio:

- Consideriamo la seguente grammatica:

$$G: \begin{array}{l} S \to aTb \mid b \\ T \to Ta \mid \varepsilon \end{array}$$

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



Lemma 6: Conversione da PDA a CFG

Date le due classi di 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:
  $$\forall q \in F \;\; (q_{\text{accept}}, \varepsilon) \in \delta'(q, \varepsilon, \varepsilon)$$
  - Lo stack deve essere svuotato prima di poter accettare una stringa:
  $$\forall q \in F, a \in \Sigma \;\; (q, \varepsilon) \in \delta'(q, \varepsilon, a)$$
- A questo punto, per costruzione stessa di P' si ha che:
  $$w \in L(P) \iff w \in L(P')$$
  dunque che $L = L(P) = L(P')$
- Consideriamo quindi la CFG $G = (V, \Sigma, R, S)$ tale che:
  - $V = \{A_{p,q} \mid p, q \in Q'\}$
  - $S = A_{q_0, q_{\text{accept}}}$
  - Ogni variabile $A_{p,q}$ è grado di derivare tutte le stringhe generabili passando dallo stato p allo stato q:
    * $\forall p \in Q'$ si ha che:
    $$(A_{p,p} \to \varepsilon) \in R$$
    * $\forall p, q, r, s \in Q', u \in \Gamma$ e $a, b \in \Sigma_\varepsilon$ si ha che:
    $$(r, u) \in \delta'(p, a, \varepsilon) \wedge (q, \varepsilon) \in \delta(s, b, u) \iff (A_{p,q} \to aA_{r,s}b) \in R$$
    * $\forall p, q, r \in Q'$ si ha che:
    $$(A_{p,q} \to A_{p,r}A_{r,q}) \in R$$

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

  Dimostrazione.

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

  Caso base.

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

  Ipotesi induttiva forte.

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

  Passo induttivo.

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

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

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

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

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

    dunque concludiamo che:

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

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

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

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

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

  Dimostrazione.

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

  Caso base.

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

  Ipotesi induttiva forte.

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

  Passo induttivo.

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

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

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

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

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

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

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

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

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

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

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

- Tramite le due affermazioni, abbiamo che:

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

  da cui concludiamo che:

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

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

□

Teorema 11: Equivalenza tra CFG e PDA

Date le due classi di 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 della derivazione $S \overset{*}{\Rightarrow} x$

Caso base.

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

Ipotesi induttiva forte.

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

Passo induttivo.

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

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

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

$\square$

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:

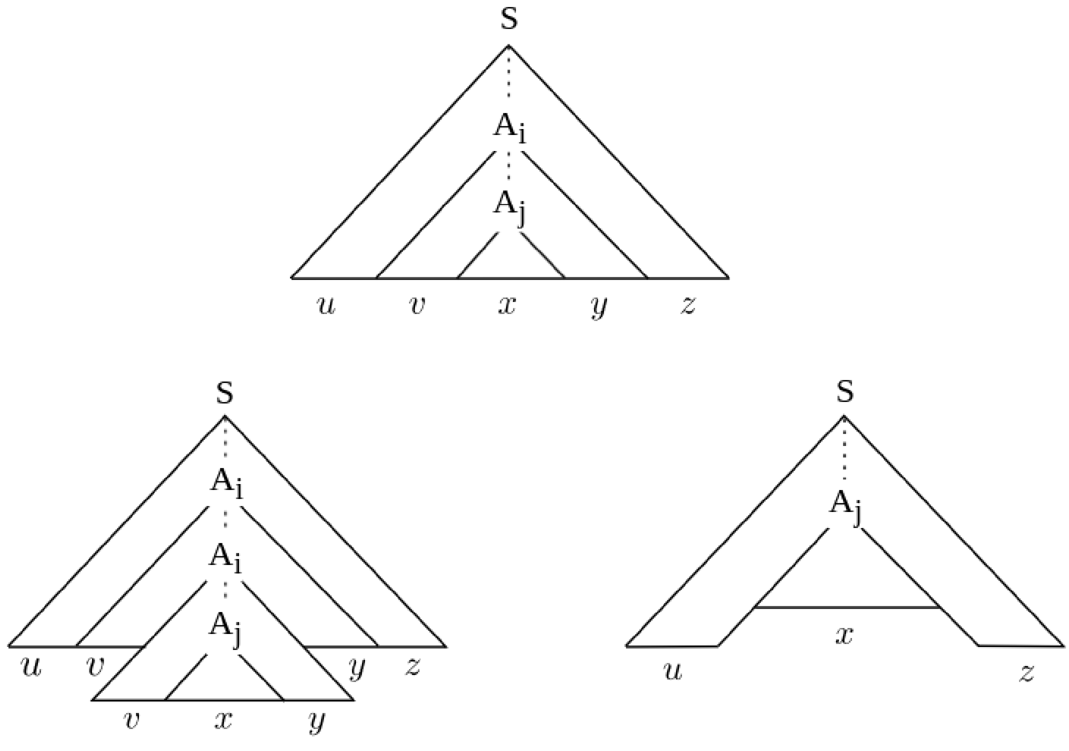
- $\forall i \in \mathbb{N} \;\; uv^ixy^iz \in L$
- $|vy| > 0$, dunque $v \neq \varepsilon$ o $y \neq \varepsilon$
- $|vxy| \leq p$

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 $A_1, \ldots, A_{k-1}$ la sequenza di variabili del cammino (dove $S = A_1$). Poiché $k - 1 \geq |V| + 1 \geq |V|$, ne segue necessariamente che $\exists i, j \mid k - |V| - 2 \leq i < j \leq k - 1 \wedge A_i = A_j$, ossia che tra le ultime $|V| + 1$ variabili del cammino vi sia almeno una variabile ripetuta
- Consideriamo quindi le cinque sottostringhe $u, v, x, y, z \in \Sigma^*$ tali che:
  - $w = uvxyz$
  - $S \overset{*}{\Rightarrow} uA_iz$
  - $A_i \overset{*}{\Rightarrow} vA_jy$
  - $A_j \overset{*}{\Rightarrow} x$

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

□



Rappresentazione grafica della dimostrazione

Esempio:

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

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

     contraddicendo quindi la prima condizione del pumping lemma
   - Di conseguenza, ne segue necessariamente che $L \notin$ CFL

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

2.6 Chiusure dei linguaggi acontestuali

Teorema 12: Chiusura dell'unione in CFL

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

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

Dimostrazione.

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

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

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

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

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

- Di conseguenza, concludiamo che:

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

□

Teorema 13: Chiusura della concatenazione in CFL

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

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

Dimostrazione.

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

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

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

  dunque abbiamo che:

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

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

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

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

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

  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:

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

- Consideriamo quindi le due grammatiche:

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

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

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

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

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

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

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

Teorema 14: Chiusura di star in CFL

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

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

Dimostrazione.

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

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

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

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

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

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

- Data $w \in L(G')$, dove $w \in L(G') \iff S_0 \overset{*}{\Rightarrow} w$, poiché le uniche regole applicabili su S_0 sono $\{S_0 \to \varepsilon, S_0 \to S, S_0 \to SS\}$, ne segue necessariamente che:
  - Se $S_0 \Rightarrow \varepsilon = w$, ne segue direttamente che $w = \varepsilon \in L^0$
  - Se $S_0 \Rightarrow S \overset{*}{\Rightarrow} w$, ne segue direttamente che $w \in L(G) = L^1$
  - Se $S_0 \Rightarrow S_0 S_0 \overset{*}{\Rightarrow} w$, dato $n \geq 2$ si ha che:

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

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

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

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

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

- Di conseguenza, concludiamo che:

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

□

Esempio:

- Consideriamo il seguente linguaggio e la sua grammatica generante:

$$L = \{0^n 1^n \mid n \in \mathbb{N}\} \qquad\qquad G : A \to 0A1 \;|\; \varepsilon$$

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

$$\begin{aligned} G' :\;\; & S \to \varepsilon \;|\; A \;|\; SS \\ & A \to 0A1 \;|\; \varepsilon \end{aligned}$$

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:

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

- Tali linguaggi sono descritti dalle seguenti due grammatiche:

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

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

- L'intersezione di tali linguaggi risulta essere:

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

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

- Di conseguenza, concludiamo che $L_1, L_2 \in \text{CFL}$ 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:

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

Dimostrazione.

- Consideriamo il seguente linguaggio:

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

- Consideriamo quindi la seguente grammatica:

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

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

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

  Dati $x_1, \dots, x_n \in \{a,b\}$ 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:

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

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

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

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

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

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

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

- Dunque, abbiamo ottenuto $L = L(G) \in \mathrm{CFL}$
- Il complemento di tale linguaggio risulta essere:

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

  il quale abbiamo già dimostrato non essere un linguaggio acontestuale (sezione 2.5)
- Di conseguenza, concludiamo che $L \in \mathrm{CFL}$, ma $\overline{L} \notin \mathrm{CFL}$

$\square$

3 Decidibilità

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:

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

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

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

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

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

$$\dots$$

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

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 $q_{\text{reject}} \neq q_{\text{accept}}$
- $\delta : Q - \{q_{\text{accept}}, q_{\text{reject}}\} \times \Gamma \to Q \times \Gamma \times \{L, R\}$ è la **funzione di transizione** della macchina, dove se $\delta(p, a) = (q, b, X)$ si ha che:
  - Viene letto il simbolo a dal nastro, sostituendolo con b e la macchina passa dallo stato p allo stato q. Inoltre, 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 $a \to b;\ X$

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:

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

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

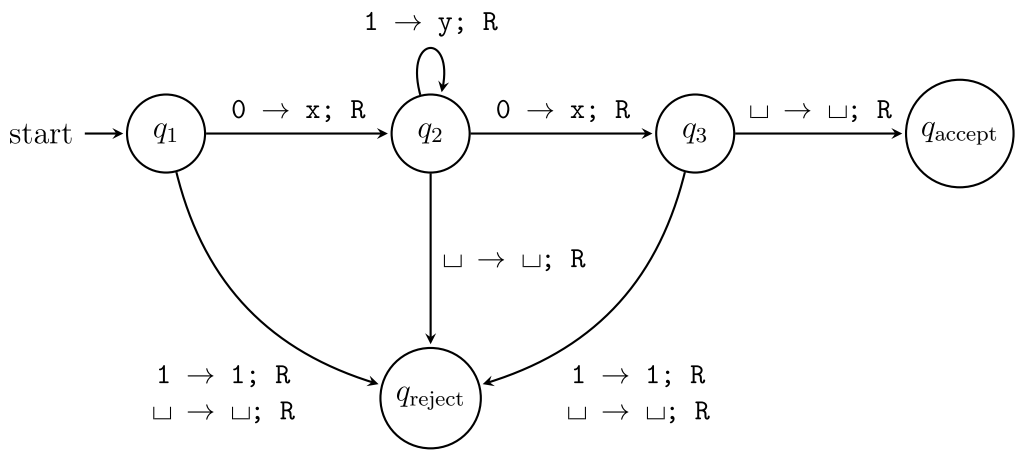
Proposizione 11: Stringa accettata in una TM

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

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

Esempio:

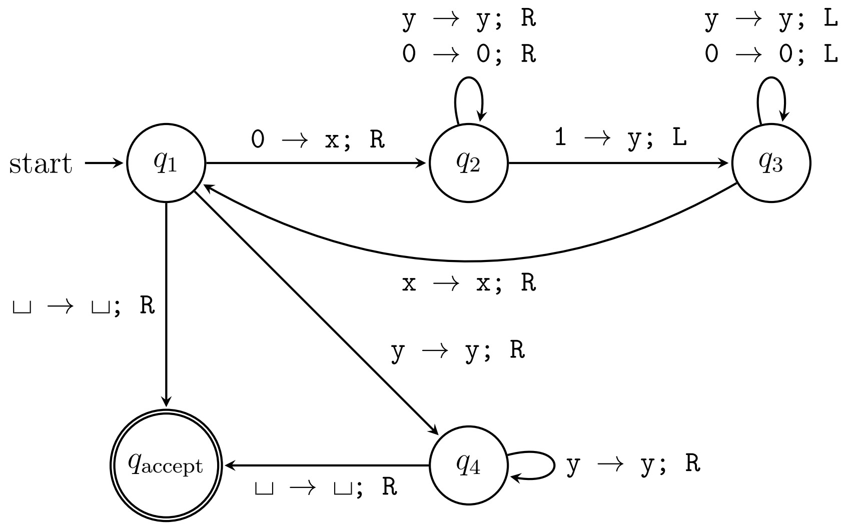
1. • La seguente TM riconosce il linguaggio $L = \{01^n0 \mid n \in \mathbb{N}\}$:



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

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

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:

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

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 (ossa 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:

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

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

3.1.1 Varianti della macchina di Turing

Definizione 41: Stay-put TM

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

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

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

Teorema 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:

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

- 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 42: Multitape TM

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

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

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

Teorema 18: Equivalenza tra TM e Multitape TM

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 Multitape 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 Multitape TM ad 1 nastro le cui transizioni non rimangono mai immobili, ne segue automaticamente che essa stessa sia la Multitape TM in grado di riconoscere $L = L(M)$

Seconda implicazione.

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

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

  1. S pone il nastro uguale a

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

     dove il simbolo # separa i vari k nastri simulati e il marcatore • 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.destra. 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, se segue automaticamente che $L = L(M) = L(S) \in \text{REC}$

□

Definizione 43: Non deterministic TM

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

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

Teorema 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 dello stato di N avente il maggior numero di transizioni uscenti
  - Se il nodo è la radice dell'albero, il suo indirizzo è ε

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

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

- M è definita come:

  M = "Data la stringa w in input:

  1. Inizialmente, il nastro 1 di M contiene w, il nastro 3 contiene ε e il nastro 2 è vuoto
  2. M copia il nastro 1 sul nastro 2
  3. 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. Se non rimangono più simboli sul nastro 3 o se la simulazione rifiuta la stringa, vai allo step 4). Se la simulazione accetta la stringa, anche M la accetta
  4. 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. Successivamente, ripeti gli step 2) e 3)"

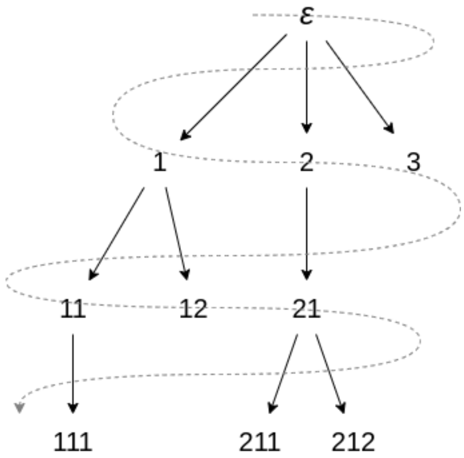
- Per costruzione stessa di M, si ha che:

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

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

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

□



Rappresentazione grafica della dimostrazione

Definizione 44: 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 $w_1, w_2, \ldots \in \Sigma^*$ tutte le stringhe di Σ^*
- Consideriamo l'enumeratore E definito come:

  $E =$ "Dato nulla in input:

  1. Ripeti gli step seguenti per $i = 1, 2, 3, \ldots$
  2. Ripeti gli step seguenti per $j = 1, \ldots, i$
  3. Simula M per i passi con w_j in input. Se essa 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, accetta w

- Per costruzione stessa di M, si ha che:

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

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

□