SAPIENZA
UNIVERSITÀ DI ROMA

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Informatica

Automi Calcolabilità e Complessità

Autore:
Simone Lidonnici

6 novembre 2024

Indice

1

Linguaggi regolari

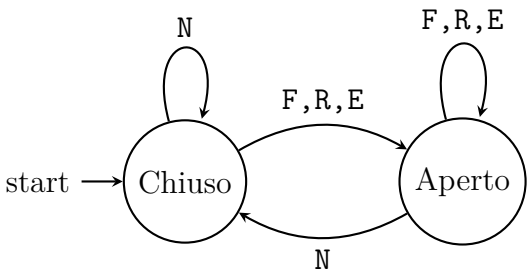
Definizione di linguaggio

Dato un **alfabeto** Σ, cioè un insieme di elementi, un **linguaggio** Σ^* è l'insieme di tutte le strighe ottenibili usando l'alfabeto Σ.

1.1 Automi Deterministici a Stati Finiti (DFA)

Il modello usato per definire i **linguaggi regolari** è l'**automa a stati finiti**, cioè una macchina che permette tramite l'input di passare da uno stato ad un altro, che ha memoria limitata e gestione dell'input limitata, ma è molto semplice.
Esempio:
Una porta che si apre tramite dei sensori può essere descritta tramite un automa con due stati (Aperta e Chiusa) e quattro input dati dai sensori (`N, F, R, E`):

- `N`: se non ci sono persone da nessun lato della porta
- `F`: se c'è una persona davanti alla porta
- `R`: se c'è una persona dietro la porta
- `E`: se ci sono persone da entrambi i lati della porta

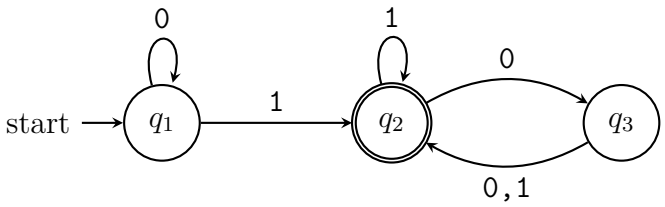


Automa Deterministico a Stati Finiti (DFA)

Un **DFA (Deterministic Finite Automaton)** è una tupla $(Q, \Sigma, \delta, q_o, F)$ in cui:

- Q è l'insieme degli stati dell'automa
- Σ è l'alfabeto dell'automa
- $\delta : Q \times \Sigma \to Q$ è la funzione di transizione degli stati
- $q_0 \in Q$ è lo stato iniziale dell'automa
- $F \subseteq Q$ è l'insieme di **stati accettanti** dell'automa

Esempio:
Preso il seguente DFA:



In questo caso:

- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- $\delta =$

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

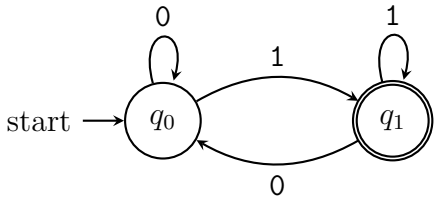
- q_1 è lo stato iniziale dell'automa
- $F = \{q_2\}$ è l'insieme degli stati accettanti

Funzione di transizione estesa

Dato un DFA D, definiamo una **funzione di transizione estesa** $\delta^* : Q \times \Sigma^* \to Q$ in modo ricorsivo:

$$\begin{cases} \delta^*(q, \varepsilon) = \delta(q, \varepsilon) = q \\ \delta^*(q, ax) = \delta^*(\delta(q, a), x) \quad a \in \Sigma, x \in \Sigma^* \end{cases}$$

Esempio:
Preso il seguente DFA:



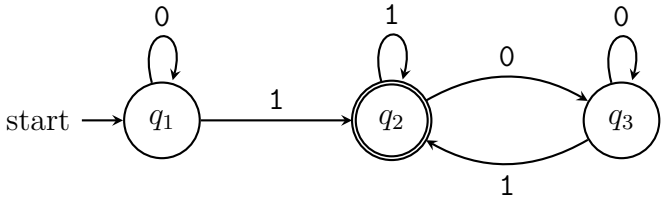
In questo DFA:

- $\delta^*(q_0, 011) = \delta^*(\delta(q_0, 0), 11) = \delta^*(q_0, 11) = \delta^*(\delta(q_0, 1), 1) = \delta^*(q_1, 1) = \delta^*(\delta(q_1, 1), \varepsilon) = \delta^*(q_1, \varepsilon) = q_1$

Linguaggio di un automa

Il **linguaggio di un DFA** D è l'insieme delle stringhe in input che l'automa accetta, cioè quelle per cui l'automa termina in uno stato accettante $q_0 \in F$.
Ogni automa ha un solo linguaggio che si scrive $L(D) = \{x \in \Sigma^* | D \text{ accetta } x\}$. Una stringa $x \in \Sigma^*$ sarà accettata da una DFA se $\delta^*(q_0, x) = q \in F$

Esempio:
Preso il seguente DFA:



Il linguaggio di questa DFA è l'insieme di tutte le stringhe che finiscono con 1:

$$L(D) = \{x \in \{0,1\}^* | x = y1 \wedge y \in \{0,1\}^*\}$$

1.1.1 Configurazione di un DFA

Configurazione di un DFA

Dato un DFA D, una **configurazione** di D è una coppia $Q \times \Sigma^*$ che indica:

- lo stato attuale dell'automa
- l'input ancora da leggere

La configurazione iniziale è sempre (q_0, x).

Passo di computazione di un DFA

Un **passo di computazione** è una relazione binaria con simbolo $\vdash_D$ per cui:

$$(q_1, ax) \vdash_D (q_2, x) \iff \delta(q_1, a) = q_2$$

La **chiusura per riflessione e transitività** di $\vdash_D$, scritta come $\vdash_D^*$, ha delle proprietà:

1. $(q_1, ax) \vdash_D (q_2, x) \implies (q_1, ax) \vdash_D^* (q_2, x)$
2. $\forall q, x \ (q, x) \vdash_D^* (q, x)$
3. $(q_1, abc) \vdash_D (q_2, bc) \vdash_D (q_3, c) \implies (q_1, abc) \vdash_D^* (q_3, c)$

Una stringa $x \in \Sigma^*$ sarà accettata da un DFA se:

$$\exists q \in F | (q_0, x) \vdash_D^* (q, \varepsilon)$$

1.2 Automi Non Deterministici a Stati Finiti (NFA)

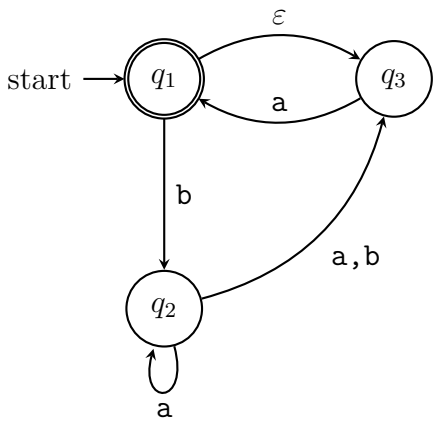
Automa Non Deterministico a Stati Finiti (NFA)

Un **NFA (Non-deterministic Finite Automaton)** è una tupla $(Q, \Sigma, \delta, q_o, F)$ in cui:

- Q è l'insieme 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
- $q_0 \in Q$ è lo stato iniziale dell'automa
- $F \subseteq Q$ è l'insieme di **stati accettanti** dell'automa

A differenza del DFA la funzione δ ha come uno dei parametri $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ e come ritorno un insieme di stati, contenuto nell'insieme delle parti di Q, cioè $\mathcal{P}(Q)$. Inoltre per considerare una stringa accettata da un NFA basta che in uno dei rami della computazione la stringa venga accettata.

Esempio:
Preso il seguente NFA:



In questo caso:

- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{\texttt{a},\texttt{b}\}$
- $\delta =$

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

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

1.2.1 Configurazione di un NFA

Configurazione di un NFA

Dato un NFA N, una **configurazione** di N è una coppia $Q \times \Sigma_\varepsilon^*$ che indica:

- lo stato attuale dell'automa
- l'input ancora da leggere

La configurazione iniziale è sempre (q_0, x).

Passo di computazione di un NFA

Un **passo di computazione** è una relazione binaria con simbolo $\vdash_N$ per cui:

$$(q_1, ax) \vdash_N (q_2, x) \iff q_2 \in \delta(q_1, a)$$

La **chiusura per simmetria e transitività** di $\vdash_N$, scritta come $\vdash_N^*$, ha delle proprietà:

1. $(q_1, ax) \vdash_N (q_2, x) \implies (q_1, ax) \vdash_N^* (q_2, x)$
2. $(q_1, abc) \vdash_N (q_2, bc) \vdash_N (q_3, c) \implies (q_1, abc) \vdash_N^* (q_3, c)$

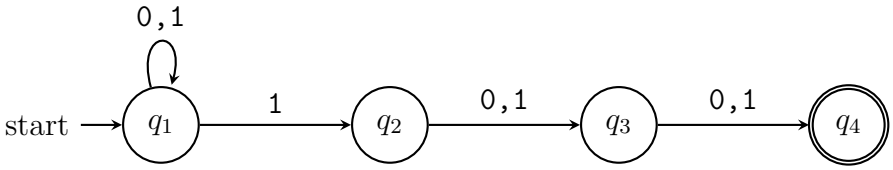
Una stringa $x \in \Sigma^*$ sarà accettata da un NFA se:

$$\exists q \in F | (q_0, x) \vdash_N^* (q, \varepsilon)$$

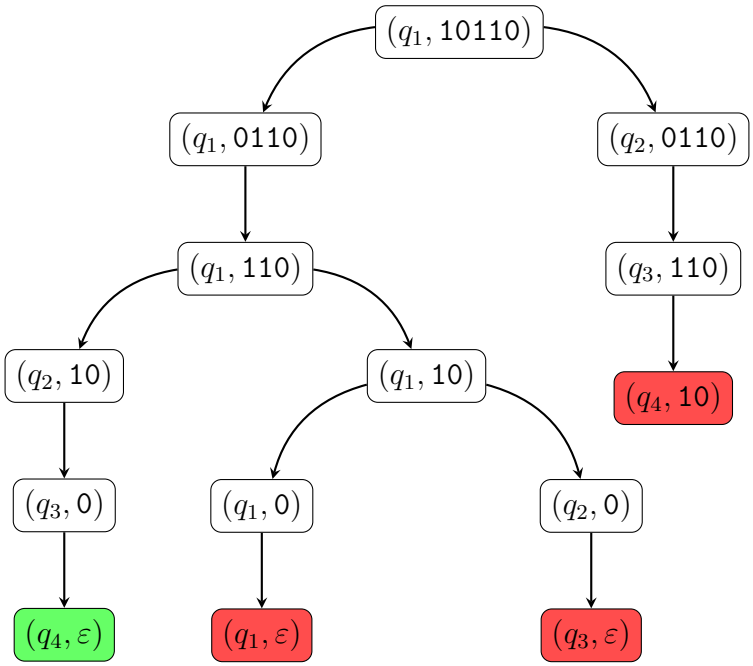
Oppure se:

$$x = y_1 \dots y_n \wedge \exists \underbrace{q_0 \dots q_n}_{\text{sequenza di stati}} | q_{i+1} = \delta(q_i, y_{i+1}) \wedge q_n \in F$$

Esempio:
Preso il seguente NFA:



Data la stringa `10110` la computazione sarà:



La stringa `10110` viene quindi accettata dal NFA.

1.3 Linguaggi regolari

Insieme dei Linguaggi regolari

Dato un alfabeto Σ, l'insieme dei **linguaggi regolari** di Σ, scritto come REG, è l'insieme dei linguaggi per cui esiste una DFA che li accetta:

$$\text{REG} = \{L \subseteq \Sigma^* | \exists D \; L(D) = L\}$$

1.3.1 Proprietà dei linguaggi regolari

I linguaggi sono insiemi di stringhe di un alfabeto Σ, quindi dati due linguaggi $L_1, L_2 \subseteq \Sigma^*$ possiamo definire le operazioni:

- **Unione:**
$$L_1 \cup L_2 = \{x \in \Sigma^* | x \in L_1 \vee x \in L_2\}$$
- **Intersezione:**
$$L_1 \cap L_2 = \{x \in \Sigma^* | x \in L_1 \wedge x \in L_2\}$$
- **Complemento:**
$$\neg L = \{x \in \Sigma^* | x \notin L\}$$
- **Concatenazione:**
$$L_1 \circ L_2 = \{xy \in \Sigma^* | x \in L_1 \wedge y \in L_2\}$$
- **Potenza:**
$$L^n = \begin{cases} \{\varepsilon\} & n = 0 \\ L \circ L^{n-1} & n > 0 \end{cases}$$
- **Star di Kleene:**
$$L^* = \{x_1 \dots x_k \in \Sigma^* | x_i \in L\} = \bigcup_{n \geq 0} L^n$$

1.3.2 Chiusura dei linguaggi regolari

Chiusura dell'unione in REG

L'unione è chiusa in REG, cioè:

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

Dimostrazioni:

- **Tramite DFA:**
  Dati $L_1, L_2 \in$ REG, allora esistono:
  - $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) | L(D_1) = L_1$
  - $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) | L(D_2) = L_2$

  Creo il DFA $D_0 = (Q_0, \Sigma, \delta_0, q_0, F_0)$ per cui:
  - $Q_0 = Q_1 \times Q_2 = \{(q_x, q_y) | q_x \in Q_1 \wedge q_y \in Q_2\}$
  - $q_0 = (q_1, q_2)$
  - $F_0 = (F_1 \times Q_2) \cup (Q_1 \times F_2) = \{(q_x, q_y) | q_x \in F_1 \vee q_y \in F_2\}$
  - $\forall (q_x, q_y) \in Q_0, a \in \Sigma$:

    $$\delta((q_x, q_y), a) = (\delta_1(q_x, a), \delta_2(q_y, a))$$

  Quindi $x \in L(D_0) \iff x \in L_1 \vee x \in L_2$, quindi $L_1 \cup L_2 \in$ REG.

- **Tramite NFA:**
  Dati $L_1, L_2 \in$ REG, allora esistono:
  - $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) | L(N_1) = L_1$
  - $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) | L(N_2) = L_2$

  Creo il NFA $N_0 = (Q_0, \Sigma, \delta_0, q_0, F_0)$per cui:
  - $Q_0 = Q_1 \cup Q_2 \cup \{q_0\}$
  - q_0 è un nuovo stato iniziale
  - $F_0 = F_1 \cup F_2$
  - $\forall q \in Q_0, a \in \Sigma$:

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

  Quindi $x \in L(N_0) \iff x \in L_1 \vee x \in L_2$, quindi $L_1 \cup L_2 \in$ REG.

Chiusura dell'intersezione in REG

L'intersezione è chiusa in REG, cioè:

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

Dimostrazione:
Dati $L_1, L_2 \in$ REG, allora esistono:

- $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) | L(D_1) = L_1$
- $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) | L(D_2) = L_2$

Creo il DFA $D_0 = (Q_0, \Sigma, \delta_0, q_0, F_0)$ per cui:

- $Q_0 = Q_1 \times Q_2 = \{(q_x, q_y) | q_x \in Q_1 \wedge q_y \in Q_2\}$
- $q_0 = (q_1, q_2)$
- $F_0 = F_1 \times F_2$
- $\forall (q_x, q_y) \in Q_0, a \in \Sigma$:
$$\delta((q_x, q_y), a) = (\delta_1(q_x, a), \delta_2(q_y, a))$$

Quindi $x \in L(D_0) \iff x \in L_1 \wedge x \in L_2$, quindi $L_1 \cap L_2 \in$ REG.

Chiusura del complemento in REG

Il complemento è chiuso in REG, cioè:

$$\forall L \in \text{REG} \implies \neg L \in \text{REG}$$

Dimostrazione:
Dato $L \in$ REG, esiste $D = (Q, \Sigma, \delta, q_0, F) | L(D) = L$.
Creo il DFA $D^* = (Q, \Sigma, \delta, q_0, Q - F)$, uguale a D ma con gli stati accettanti invertiti, quindi $x \in L(D^*) \iff x \notin L(D)$, quindi $\neg L \in$ REG.

Chiusura della concatenazione in REG

La concatenazione è chiusa in REG, cioè:

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

Dimostrazione:
Dati $L_1, L_2 \in \text{REG}$, allora esistono:

- $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1) | L(N_1) = L_1$
- $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2) | L(N_2) = L_2$

Creo il NFA $N_0 = (Q_0, \Sigma, \delta_0, q_0, F_0)$ per cui:

- $Q_0 = Q_1 \cup Q_2$
- $q_0 = q_1$
- $F_0 = F_2$
- $\forall q \in Q_0, a \in \Sigma$:

$$\delta_0(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 - F_1 \\ \delta_1(q,a) & q \in F_1 \wedge a \neq \varepsilon \\ \delta_1(q,a) \cup q_2 & q \in F_1 \wedge a = \varepsilon \\ \delta_2(q,a) & q \in Q_2 \end{cases}$$

Quindi $x \in L(D_0) \iff x \in L_1 \circ L_2$, quindi $L_1 \cap L_2 \in \text{REG}$.

Chiusura di star in REG

Star è chiusa in REG, cioè:

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

Dato $L \in \text{REG}$, esiste $N = (Q, \Sigma, \delta, q_0, F) | L(D) = L$.
Creo il NFA $N^* = (Q^*, \Sigma, \delta^*, q_0^*, F^*)$ in cui:

- q_0^* è un nuovo stato iniziale
- $Q^* = Q \cup q_0^*$
- $F^* = F \cup q_0^*$
- $\forall q \in Q^*, a \in \Sigma$:

$$\delta^*(q,a) = \begin{cases} \delta(q,a) & q \in Q - F \\ \delta(q,a) & q \in F \wedge a \neq \varepsilon \\ \delta(q,a) \cup q_0 & q \in F \wedge a = \varepsilon \\ q_0 & q = q_0^* \wedge a = \varepsilon \\ \emptyset & q = q_0^* \wedge a \neq \varepsilon \end{cases}$$

Quindi $x \in L(N^*) \iff x \in L^*$, quindi $L^* \in \text{REG}$.

1.4 Equivalenza tra DFA e NFA

Linguaggi accettati da DFA e NFA

Sia $\mathcal{L}(\text{NFA})$ l'insieme dei linguaggi per cui esiste un NFA che li accetta e $\mathcal{L}(\text{DFA})$ l'insieme dei linguaggi per cui esiste un DFA che li accetta, allora:

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

Dimostrazione per doppia inclusione:

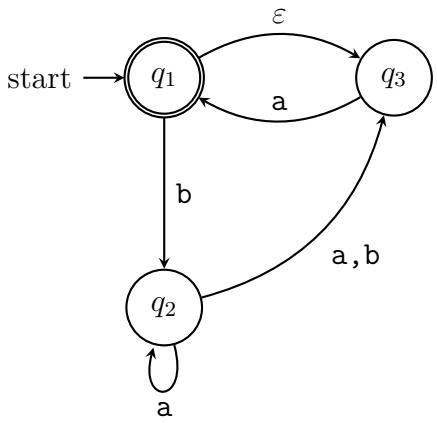
1. $\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{NFA})$:
   Dato un $L \in \mathcal{L}(\text{DFA})$ allora esiste $D = (Q, \Sigma, \delta, q_0, F)$ tale che $L(D) = L$.
   Visto che NFA è una generalizzazione di DFA, allora è ovvio che:
   $$\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{NFA})$$

2. $\mathcal{L}(\text{NFA}) \subseteq \mathcal{L}(\text{DFA})$:
   Dato un $L \in \mathcal{L}(\text{NFA})$ allora esiste $N = (Q_N, \Sigma, \delta_N, q_{0_N}, F_N)$ tale che $L(N) = L$.
   Considero un DFA $D = (Q_D, \Sigma, \delta_D, q_{0_D}, F_D)$, costrito partendo da N in cui:

   - $Q_D = \mathcal{P}(Q_N)$
   - Dato $R \in Q_D$, definiamo l'**estensione** di R:
     $$E(R) = \left\{ q \in Q_N \,\middle|\, \begin{array}{c} q \text{ può essere raggiunto da uno stato } r \in R \\ \text{tramite solamente } \varepsilon\text{-archi} \end{array} \right\}$$
   - $q_{0_D} = E(\{q_{0_N}\})$
   - $F_D = \{R \in Q_D | R \cap F_N \neq \emptyset\}$
   - Dati $R \in Q_D$ e $a \in \Sigma$, δ_D è definita:
     $$\delta_D(R, a) = \bigcup_{r \in R} E(\delta_N(r, a))$$

   Detto questo allora abbiamo che:
   $$w \in L(N) \iff w \in L(D)$$
   quindi:
   $$\mathcal{L}(\text{NFA}) \subseteq \mathcal{L}(\text{DFA})$$

Esempio:
Preso il seguente NFA:



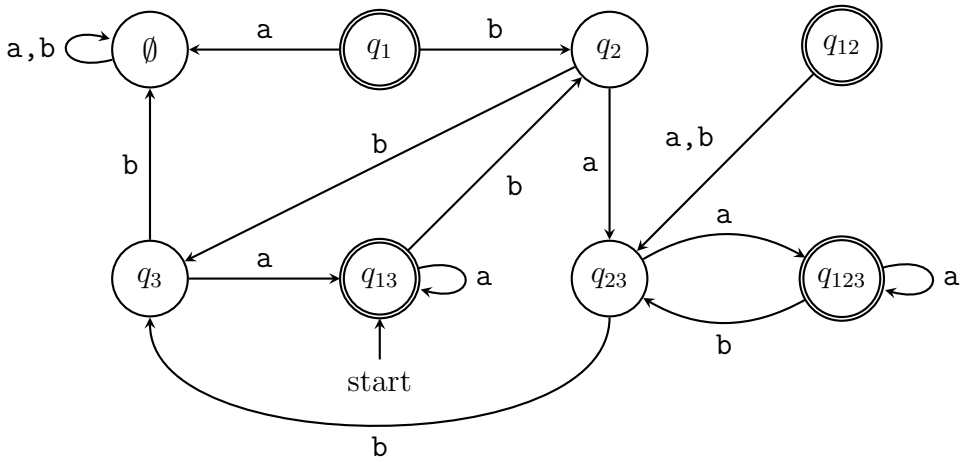
Per costruire il DFA equivalente, che sarà definito:

- $Q_D = \{\emptyset, q_1, q_2, q_3, \{q_1, q_2\}, \{q_1, q_3\}, \{q_2, q_3\}, \{q_1, q_2, q_3\}\}$
  che scriviamo in notazione semplificata:

$$Q_D = \{\emptyset, q_1, q_2, q_3, q_{12}, q_{13}, q_{23}, q_{123}\}$$

- $q_{0_D} = E(\{q_{0_N}\}) = E(q_1) = \{q_1, q_3\} = q_{13}$
- $F_D = \{q_1, q_{12}, q_{13}, q_{123}\}$
- δ_D viene definita come scritto prima, per esempio:
  - $\delta_D(q_1, a) = E(\delta_N(q_1, a)) = \emptyset$
  - $\delta_D(q_1, b) = E(\delta_N(q_1, b)) = q_2$
  - $\delta_D(q_2, a) = E(\delta_N(q_2, a)) = \{q_2, q_3\} = q_{23}$
  - $\delta_D(q_2, b) = E(\delta_N(q_2, b)) = q_3$
  - $\delta_D(q_{12}, b) = E(\delta_N(q_1, b)) \cup E(\delta_N(q_2, b)) = \{q_2, q_3\} = q_{23}$
  - $\delta_D(q_{13}, a) = E(\delta_N(q_1, a)) \cup E(\delta_N(q_3, a)) = \{\emptyset, q_1, q_3\} = q_{13}$

Il DFA equivalente quindi è:



1.5 Espressioni regolari

Definizione di espressione regolare

Dato un alfabeto Σ, un'**espressione regolare** di Σ è una stringa r che rappresenta un linguaggio $L(r) \subseteq \Sigma^*$. L'insieme delle espressioni regolari di un alfabeto Σ, scritte come $\text{re}(\Sigma)$, è definito:

- $\emptyset \in \text{re}(\Sigma)$
- $\varepsilon \in \text{re}(\Sigma)$
- $a \in \text{re}(\Sigma)\ \forall a \in \Sigma$
- $r_1, r_2 \in \text{re}(\Sigma) \implies r_1 \cup r_2 \in \text{re}(\Sigma)$
- $r_1, r_2 \in \text{re}(\Sigma) \implies r_1 \circ r_2 \in \text{re}(\Sigma)$
- $r \in \text{re}(\Sigma) \implies r^* \in \text{re}(\Sigma)$

Esempio:
Dato l'alfabeto $\Sigma = \{0, 1\}$

- $0 \cup 1 = \{0\} \cup \{1\} = \{0, 1\}$
- $0^*10^* = \{0\}^* \circ \{1\} \circ \{0\}^* = \{x1y | x, y \in \{0\}^*\}$
- $\Sigma^*1\Sigma^* = \Sigma^* \circ \{1\} \circ \Sigma^* = \{x1y | x, y \in \Sigma^*\}$
- $1^*\emptyset = \{1\}^* \circ \emptyset = \emptyset$
- $\emptyset^* = \varepsilon$

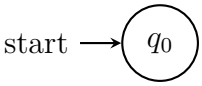
Conversione da espressione regolare a DFA

Date la classe dei linguaggi descritti da un'espressione regolare $\mathcal{L}(\text{re})$ e $\mathcal{L}(\text{DFA})$:

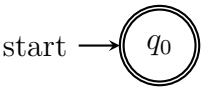
$$\mathcal{L}(\text{re}) \subseteq \mathcal{L}(\text{DFA})$$

Dimostrazione per induzione:

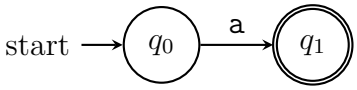
- Caso base:
  - $r = \emptyset \in \text{re}(\Sigma)$, definiamo il DFA $D_\emptyset$:

    in cui $x \in L(r) \iff x \in L(D_\emptyset)$ quindi $L(r) \in \mathcal{L}(\text{DFA})$
  - $r = \varepsilon \in \text{re}(\Sigma)$, definiamo il DFA D_ε:

    in cui $x \in L(r) \iff x \in L(D_\varepsilon)$ quindi $L(r) \in \mathcal{L}(\text{DFA})$
  - $r = a \in \text{re}(\Sigma)$, definiamo il DFA D_a:

    in cui $x \in L(r) \iff x \in L(D_a)$ quindi $L(r) \in \mathcal{L}(\text{DFA})$
- Passo induttivo:
  - $r = r_1 \cup r_2$, allora abbiamo che:

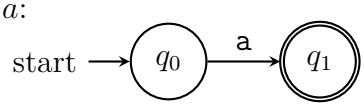
    $$L(r) = L(r_1) \cup L(r_2) = L(D_1) \cup L(D_2) \in \mathcal{L}(\text{DFA})$$
  - $r = r_1 \circ r_2$, allora abbiamo che:

    $$L(r) = L(r_1) \circ L(r_2) = L(D_1) \circ L(D_2) \in \mathcal{L}(\text{DFA})$$
  - $r = r_1^*$, allora abbiamo che:

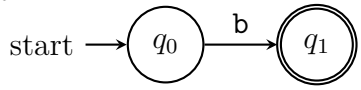
    $$L(r) = L(r_1)^* = L(D_1)^* \in \mathcal{L}(\text{DFA})$$

Esempio:
Data l'espressione regolare $(a \cup ab)^*$, il NFA corrispondente a tale espressione creata partendo dai sotto-componenti:

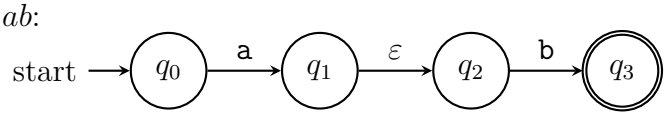
- a:



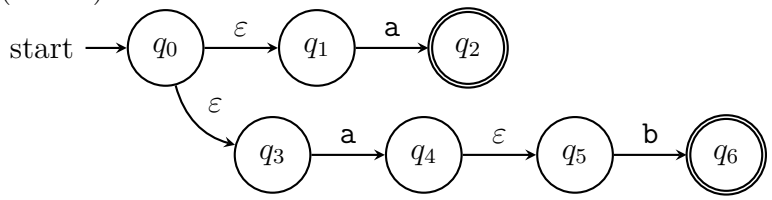
- b:



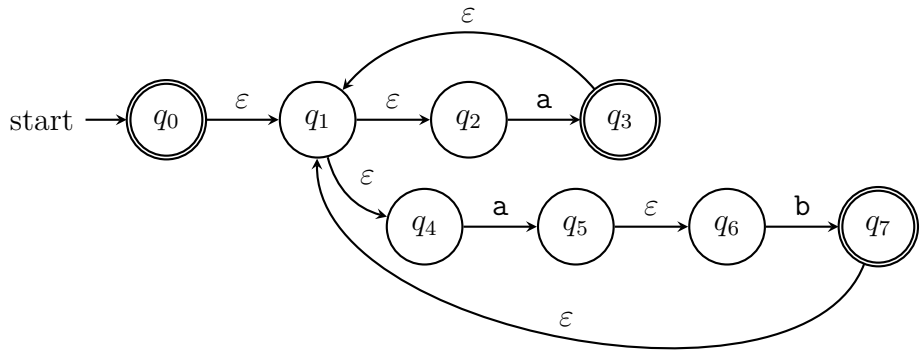
- ab:



- $(a \cup ab)$:



- $(a \cup ab)^*$:



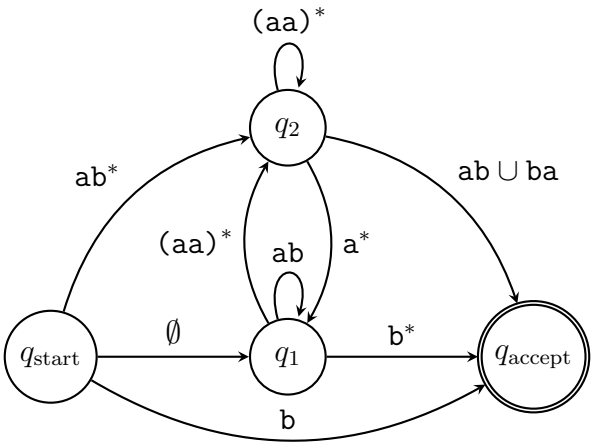
1.5.1 NFA generalizzati (GNFA)

NFA generalizzato (GNFA)

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

- $|Q| \geq 2$ è l'insieme degli stati dell'automa
- 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 in cui però:
  - q_{start} ha solo archi uscenti
  - q_{accept} ha solo archi entranti
  - Per ogni coppia di stati (q_1, q_2), c'è esattamente un arco $q_1 \to q_2$ e un arco $q_2 \to q_1$, incluse le coppie (q, q)
  - Le etichette degli archi sono espressioni regolari

Esempio:



Conversione da DFA a GNFA

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

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

Dimostrazione:
Dato $L \in \mathcal{L}(\text{DFA})$, allora esiste $D = (Q, \Sigma, \delta, q_0, F)$ tale che $L(D) = L$.
Costruisco un GNFA $G = (Q_G, \Sigma, \delta_G, q_{\text{start}}, q_{\text{accept}})$ costruito da D in cui:

- $Q_G = Q \cup \{q_{\text{start}}, q_{\text{accept}}\}$
- $\delta_G(q_{\text{start}}, q_0) = \varepsilon$
- $\forall q \in F \ \delta_G(q, q_{\text{accept}}) = \varepsilon$
- Ogni transizione con etichette multiple in D viene trasformata in un'etichetta con l'unione delle etichette multiple
- Per ogni coppia per cui non ci sia un arco in un verso o nell'altro in D viene aggiunto un arco in G con etichetta $\emptyset$

Quindi $x \in L(D) \implies x \in L(G)$, quindi $\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{GNFA})$.

1.5.2 Riduzione minimale di un GNFA

Dato un GNFA $G = (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, possiamo usare un'algoritmo per ottenere un GNFA G' con solo due stati (equivalente quindi ad un'espressione regolare) e per cui $L(G) = L(G')$:

Algoritmo: Riduzione minimale di un GNFA

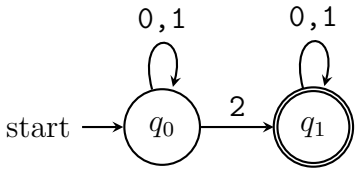
```
def ReduceGNFA(G):
    if |Q|==2 :
        return G
    q = q ∈ Q − {q_start, q_accept}// scelto in modo casuale
    Q' = Q − q
    for q_i in Q' − {q_accept} :
        for q_j in Q' − {q_start} :
            δ'(q_i, q_j) = δ(q_i, q)δ(q, q)*δ(q, q_j) ∪ δ(q_i, q_j)
    G' = (Q', Σ, δ', q_start, q_accept)
    return ReduceGNFA(G')
```

Dimostrazione per induzione:
Induzione in base al numero degli stati k

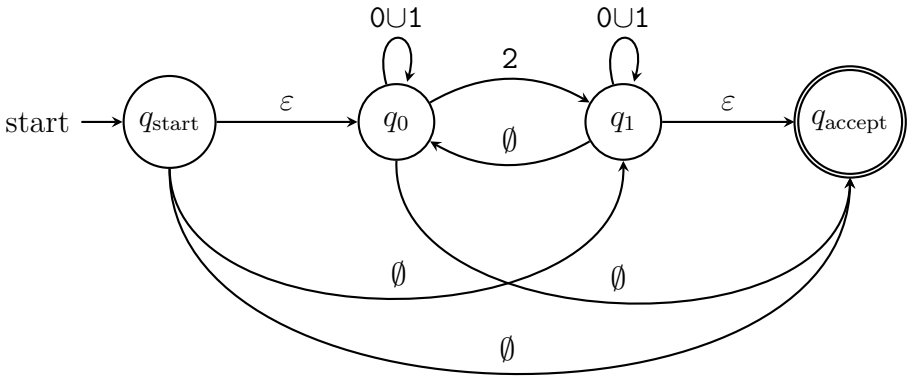
- Caso base:
  $k = 2 \implies G' = G \implies L(G') = L(G)$
- Caso induttivo:
  Assumo che per un G_k sia vero che $L(G_k) = L(G)$
- Passo induttivo:
  Dato $G_{k-1} = G_k - \{q_r\}$ si ha che se G_k accetta una stringa x, allora esiste una serie di stati $q_{\text{start}} q_1 \dots q_{\text{accept}}$ in cui abbiamo due casi:
  - $q_r \notin q_{\text{start}} q_1 \dots q_{\text{accept}}$ allora se $x \in L(G_k) \implies x \in L(G_{k-1})$
  - $q_r \in q_{\text{start}} q_1 \dots q_{\text{accept}}$ allora gli stati q_i e q_j vicini allo stato rimosso saranno collegati da una nuova espressione regolare che per cui $\delta'(q_i, q_j) = \delta(q_i, q_r)\delta(q_r, q_r)^*\delta(q_r, q_j)$ per cui $x \in L(G_k) \implies x \in L(G_{k-1})$

  Allora se $x \in L(G_{k+1})$ allora per ogni coppia (q_i, q_j) l'etichetta rappresenterà tutti i cammini tra q_i e q_j anche in G_k quindi $x \in L(G_{k-1}) \implies x \in L(G_k)$. Quindi alla fine si ha che $L(G) = L(G_k) = L(G_{k-1})$

Esempio:
Dato il seguente DFA:

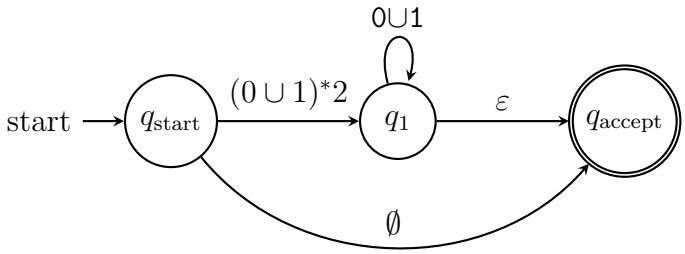


Il GNFA equivalente sarà:



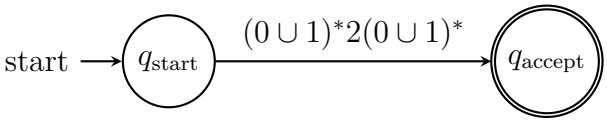
Rimuovendo lo stato q_0 e ricalcolando le transizioni:

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



Rimuovendo anche lo stato q_1 e ricalcolando l'ultima transizione:

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



Conversione da GNFA a espressione regolare

Dati $\mathcal{L}(\text{GNFA})$ e $\mathcal{L}(\text{re})$ si ha quindi che:

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

Equivalenza nella classe dei linguaggi regolari

Per un alfabeto Σ, dai vari lemmi precedenti di conversione di ha che:

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

1.6 Pumping Lemma per linguaggi regolari

Preso un linguaggio L, ad esempio:

$$L = \{0^n 1^n | n \geq 0\}$$

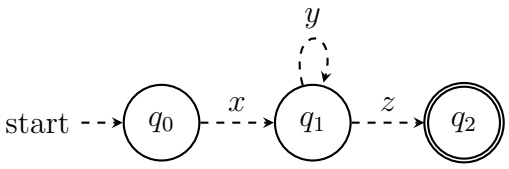
Questo linguaggio avrebbe bisogno di un automa ad infiniti stati per poter sapere il numero di 0 e 1, quindi impossibile con un DFA, NFA o GNFA.

Pumping Lemma per linguaggi regolari

Dato un linguaggio $L \in$ REG, allora esiste un DFA D con p stati tale che $L(D) = L$.
$\forall w \in L$, con $|w| \geq p$, è ovvio che alcuni stati devono ripetersi e dividendo $w = xyz$ con x, y, z sottostringhe di w si ha che:

1. $xy^k z \in L \; \forall k \geq 0$
2. $|y| > 0$
3. $|xy| < p$

Graficamente, con le freccie tratteggiate che indicano che in mezzo possono esserci altri stati:



Dimostrazione:
Dato $L \in$ REG e il DFA D per cui $L(D) = L$, consideriamo $p = |Q|$ e $w = w_1 \dots w_n$, con $n \geq p$, allora esiste una sequenza di stati $q_1 \dots q_{n+1}$ in cui:

$$\delta(q_k, w_k) = q_{k+1} \quad \forall k$$

La sequenza è lunga $n+1 \geq p+1$ perciò tra i primi $p+1$ stati ci deve essere almeno uno stato ripetuto. Siano q_i e q_j le due occorrenze di questo stato ripetuto, con $i < j \leq p+1$. Dividiamo la stringa in:

- $x = w_1 \dots w_{i-1} \implies \delta^*(q_1, x) = q_i$
- $y = w_i \dots w_{j-1} \implies \delta^*(q_i, y) = q_j = q_i$
- $z = w_j \dots w_n \implies \delta^*(q_j, z) = q_{n+1} \in F$

Quindi $\delta(q_1, xy^k z) = q_{n+1} \in F \implies xy^k z \in L$.
Inoltre sapendo che $i \neq j \implies |y| > 0$ e $j \leq p+1 \implies |xy| \leq p$.

1.6.1 Dimostrazione di non regolarità tramite pumping lemma

Il pumping lemma viene usato per dimostrare che una linguaggio non è regolare, per farlo basta trovare una stringa che non possa essere scomposta secondo il pumping lemma.
Esempio:
Dato il linguaggio:

$$L = \{0^n 1^n | n \geq 0\}$$

Preso un valore p del pumping lemma, scelgo la stringa $w = 0^p 1^p$ con quindi $|w| = 2p > p$, visto che $|xy| < p \implies y$ è composta da solo 0, questo vuol dire che con indice $i = 2$:

$$xy^2z = 0^q 1^p \; q > p \implies xy^2z \notin L$$

Quindi il linguaggio non è regolare.

2

Linguaggi acontestuali

2.1 Grammatiche acontestuali

Grammatica acontestuale (CFG)

Una **CFG (Context-free Grammar)** è una tupla (V, Σ, R, S) in cui:

- V è l'insieme delle **variabili**
- Σ è l'insieme dei **terminali** per cui $V \cap \Sigma = \emptyset$
- R è l'insieme delle **regole** nella forma $R : V \to (V \cup \Sigma)^*$
- S è la variabile iniziale della grammatica

Esempio:
Data la grammatica $G = (\{A, B\}, \{0, 1, \#\}, R, A)$ in cui R ha le seguenti regole:

- $A \to 0A1$
- $A \to B$
- $B \to \#$

Le regole possono essere scritte in modo contratto nel caso in cui una variabile A abbia diverse regole $A \to X_1, A \to X_2, \ldots, A \to X_n$, scrivendo $A \to X_1|X_2|\ldots|X_n$. Nel caso sopra per esempio avremmo potuto scrivere $A \to 0A1|B$.

Produzione e derivazione

Data una grammatica $G = (V, \Sigma, R, S)$, u, v, w stringhe di variabili o terminali ed esiste una regola $A \to w$, allora la stringa uAv **produce** la stringa uwv, scritto come:

$$uAv \Rightarrow uwv$$

Se invece esistono un insieme di stringhe $u_1, \ldots, u_k$ tale che:

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

allora la stringa u **deriva** v, scritto come:

$$u \overset{*}{\Rightarrow} v$$

Esempio:
Data la grammatica con le regole:

$$E \to E + E | E \cdot E | (E) | 0 | \dots | 9$$

Possiamo derivare la stringa:

$$E \Rightarrow E \cdot E \Rightarrow (E) \cdot E \Rightarrow (E + E) \cdot E \Rightarrow (3 + 4) \cdot 5$$

Linguaggio di una grammatica

Data una grammatica $G = (V, \Sigma, R, S)$, il linguaggio generato da G è l'insieme delle stringhe che si possono derivare da S:

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

Esempio:
Data la CFG $G = (\{S\}, \{a, b\}, R, S)$ con le regole:

$$S \to \varepsilon | aSb | SS$$

allora:

- $S \Rightarrow aSb \Rightarrow a\varepsilon b = ab \implies ab \in L(G)$
- $S \Rightarrow SS \Rightarrow aSbaSb \Rightarrow a\varepsilon ba\varepsilon b = abab \implies abab \in L(G)$

Classe dei linguaggi acontestuali

Dato un alfabeto Σ, la **classe dei linguaggi acontestuali** di Σ, scritta come CFL, è l'insieme dei linguaggi per cui esiste una grammatica che li accetta:

$$\text{CFL} = \{L \subseteq \Sigma^* | \exists CFGG\ L(G) = L\}$$

2.1.1 Grammatiche ambigue

Data una CFG G e una stringa w è possibile che esistano più derivazioni di w, cioè passaggi diversi che portano alla stringa w.
Esempio:
Data la CFG con regole:

$$E \to E + E | E \cdot E | a$$

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

1. $E \Rightarrow E + E \Rightarrow E + E \cdot E \Rightarrow a + a \cdot a$
2. $E \Rightarrow E \cdot E \Rightarrow E + E \cdot E \Rightarrow a + a \cdot a$

Derivazione a sinistra e grammatica ambigua

Data una CFG G e una stringa w, una **derivazione a sinistra** è una derivazione $S \overset{*}{\Rightarrow} w$ in cui ad ogni passo viene sostituita la variabile più a sinistra. Questo tipo di derivazione diminuisce le derivazioni multiple, ma ci sono comunque alcuni casi in cui può esistere più di una derivazione per una stringa.
Se una grammatica ha almeno una stringa con derivazioni a sinistra multiple allora la grammatica si dice **ambigua**.

2.2 Forma normale di una grammatica

Forma normale di una grammatica

Una CFG G è in **forma normale (CNF)** se tutte le regole sono della forma:

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

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

Trasformazione di una CFG in forma normale

Data una CFG G è sempre possibile trasformarla in una CFG G' in forma normale per cui:

$$L(G) = L(G')$$

Dimostrazione:
Data una CFG $G = (V, \Sigma, R, S)$, la CFG G' in forma normale si ottiene con i passaggi:

1. Aggiungo una variabile iniziale S_0 e una regola $S_0 \to S$
2. Elimino le ε-regole, cioè quelle nella forma $A \to \varepsilon$ con $A \neq S_0$ e per ogni regola contente A nella parte destra aggiungo una regola con quella occorrenza eliminata. Se più di una A è presente nella parte destra allora aggiungo una regola per ogni combinazione di occorrenze eliminate.
   Per esempio se elimino la regola $A \to \varepsilon$ e esiste la regola $B \to uAvAw$ allora aggiungerò le regole $B \to uvAw|uAvw|avw$
3. Elimino le regole unitarie, cioè nella forma $A \to B$ e per ogni regola della forma $B \to u$ in cui u è una stringa di variabili e terminali, aggiungo una regola $A \to u$
4. Per ogni regola $A \to v_1 v_2 \dots v_k$ in cui v è una variabile o terminale singolo e $k \geq 3$, creo una serie di variabili $A_1, \dots, A_{k-2}$ per cui:

   $$\begin{aligned} A &\to v_1 A_1 \\ A_1 &\to v_2 A_2 \\ &\dots \\ A_{k-1} &\to v_{k-1} v_k \end{aligned}$$

   ed elimino la regola $A \to v_1 v_2 \dots v_k$
5. Per ogni regola nella forma $A \to aB$ oppure $A \to Ba$, si crea una nuova variabile U e la regola $U \to a$ e si sostituisce la regola $A \to aB$ con $A \to UB$ oppure la regola $A \to Ba$ con $A \to BU$

Esempio:
Data la CFG con stato iniziale S e regole:

$$\begin{aligned} S &\to ASA|aB \\ A &\to B|S \\ B &\to b|\varepsilon \end{aligned}$$

Per ceare la CFG in forma normale corrispondente seguo i passaggi:

1. Aggiungo la variabile iniziale S_0 e la regola $S_0 \to S$:

   $$\begin{aligned} \boldsymbol{S_0} &\boldsymbol{\to S} \\ S &\to ASA|aB \\ A &\to B|S \\ B &\to b|\varepsilon \end{aligned}$$

2. Elimino le ε-regole:

   2.1 Elimino $B \to \varepsilon$:

$$
\begin{aligned}
&S_0 \to S\\
&S \to ASA|aB|\boldsymbol{a}\\
&A \to B|S|\boldsymbol{\varepsilon}\\
&B \to b|\not{\varepsilon}
\end{aligned}
$$

   2.2 Elimino $A \to \varepsilon$:

$$
\begin{aligned}
&S_0 \to S\\
&S \to ASA|aB|a|\boldsymbol{AS}|\boldsymbol{SA}|\boldsymbol{S}\\
&A \to B|S|\not{\varepsilon}\\
&B \to b
\end{aligned}
$$

3. Elimino le regole unitarie:

   3.1 Elimino $S \to S$ e $S_0 \to S$:

$$
\begin{aligned}
&S_0 \to \not{S}|\boldsymbol{ASA}|\boldsymbol{aB}|\boldsymbol{a}|\boldsymbol{AS}|\boldsymbol{SA}\\
&S \to ASA|aB|a|AS|SA|\not{S}\\
&A \to B|S\\
&B \to b
\end{aligned}
$$

   3.2 Elimino $A \to B$ e $A \to S$:

$$
\begin{aligned}
&S_0 \to ASA|aB|a|AS|SA\\
&S \to ASA|aB|a|AS|SA\\
&A \to \not{B}|\not{S}|\boldsymbol{b}|\boldsymbol{ASA}|\boldsymbol{aB}|\boldsymbol{a}|\boldsymbol{AS}|\boldsymbol{SA}\\
&B \to b
\end{aligned}
$$

4. Separo le regole con tre o più elementi a destra e le divido in regole con due elementi a destra:

$$
\begin{aligned}
&S_0 \to \cancel{ASA}|\boldsymbol{AC}|aB|a|AS|SA\\
&S \to \cancel{ASA}|\boldsymbol{AC}|aB|a|AS|SA\\
&A \to b|\cancel{ASA}|\boldsymbol{AC}|aB|a|AS|SA\\
&\boldsymbol{C \to SA}\\
&B \to b
\end{aligned}
$$

5. Converto le regole con due elementi a destra di cui almeno uno un terminale:

$$
\begin{aligned}
&S_0 \to AC|\cancel{aB}|\boldsymbol{UB}|a|AS|SA\\
&S \to AC|\cancel{aB}|\boldsymbol{UB}|a|AS|SA\\
&A \to b|AC|\cancel{aB}|\boldsymbol{UB}|a|AS|SA\\
&C \to SA\\
&\boldsymbol{U \to a}\\
&B \to b
\end{aligned}
$$

2.3 Automi a Pila

Automa a Pila (PDA)

Un **PDA (Push-Down Automata)** è una tupla $(Q, \Sigma, \Gamma, \delta, q_0, F)$ in cui:

- Q è l'insieme degli stati dell'automa
- Σ è l'alfabeto dell'automa
- Γ è l'alfabeto della pila (stack)
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ è la funzione di transizione per cui se $(q_2, c) \in \delta(q_1, a, b)$ allora:
  l'automa legge un simbolo $a \in \Sigma_\varepsilon$ e se in cima allo stack c'è $b \in \Gamma_\varepsilon$ passa dallo stato q_1 a q_2 e sostituisce b con c. Viene scritto come $a; b \to c$
- q_0 è lo stato iniziale
- F è l'insieme degli stati accettanti

Un PDA tramite la funzione δ, se $(q_2, c) \in \delta(q_1, a, b)$ ci sono diversi casi:

- $a, b \neq \varepsilon, c = \varepsilon$ $(a; b \to \varepsilon)$: l'automa legge a e se in cima allo stack c'è b passa dallo stato q_1 allo stato q_2 e rimuove b dallo stack (pop)
- $a, c \neq \varepsilon, b = \varepsilon$ $(a; \varepsilon \to c)$: l'automa legge a e passa dallo stato q_1 allo stato q_2 e aggiunge c allo stack (push)
- $a \neq \varepsilon, b, c \neq \varepsilon$ $(a; \varepsilon \to \varepsilon)$: l'automa legge a e passa dallo stato q_1 allo stato q_2 senza modificare lo stack

Una stringa $w = w_1 \dots w_k$ è accettata da un PDA se esiste una serie di stati $q_0 \dots q_{k+1}$ e una serie di stringhe $s_1 \dots s_n$ per cui:

- $s_0 = \varepsilon$ (pila vuota)
- $\forall i (q_{i+1}, a) \in \delta(q_i, w_i, b)$ con $s_i = bt$ e $s_{i+1} = at$
- $q_{k+1} \in F$

Passo di computazione di un PDA

Un **passo di computazione** è una relazione binaria con simbolo $\vdash_P$ per cui:

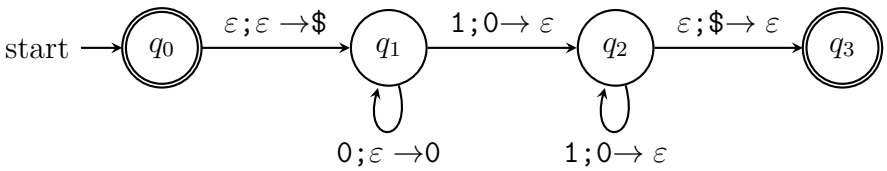
$$(q_1, ax, by) \vdash_P (q_2, x, cy) \iff (q_2, c) \in \delta(q_1, a, b)$$

La **chiusura transitiva** di $\vdash_P$, scritta come $\vdash_P^*$ definisce la transizione estesa per cui:

$$L(P) = \{w \in \Sigma^* | (q_0, w, \varepsilon) \vdash_P^* (q, \varepsilon, y) \land q \in F\}$$

Questa definizione sarebbe equivalente se sostituissimo (q, ε, y) con $(q, \varepsilon, \varepsilon)$, perchè è sempre possibile svuotare lo stack senza cambiare il linguaggio.

Esempio:
Dato il linguaggio $L = \{0^n 1^n | n \geq 0\}$, il PDA equivalente è:



Linguaggi accettati da PDA

Dato un alfabeto Σ, definiamo l'insieme dei linguaggi accettati da un PDA come:

$$\mathcal{L}(PDA) = \{L \subseteq \Sigma^* | \exists PDA P \; L(P) = L\}$$

2.4 Equivalenza tra CFG e PDA

Conversione da CFG a PDA

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

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

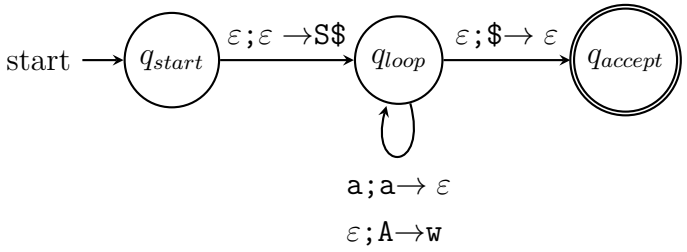
Dimostrazione:
Dato un linguaggio $L \in$ CFL, esiste una grammatica $G = (V, \Sigma, R, S) | L(G) = L$.
Costruiamo un PDA $P = (Q, \Sigma, \Gamma, \delta, q_{start}, F)$ tale che:

- $Q = (q_{start}, q_{loop}, q_{accept}) \cup E$, per cui E sono gli stati aggiunti per far si che δ esegua un push o un pop di un solo simbolo alla volta
- $\Gamma = V \cup \Sigma$
- $F = q_{accept}$
- δ è una funzione per cui:
  - $\delta(q_{start}, \varepsilon, \varepsilon) = (q_{loop}, S\$)$
  - $\forall A \in V \; \delta(q_{loop}, \varepsilon, A) = \{(q_{loop}, w) | (A \to w) \in R \wedge u \in \Gamma^*\}$
  - $\forall a \in \Sigma \; \delta(q_{loop}, a, a) = (q_{loop}, \varepsilon)$
  - $\delta(q_{lopp}, \varepsilon, \$) = (q_{accept}, \varepsilon)$

Quindi si ha che:

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

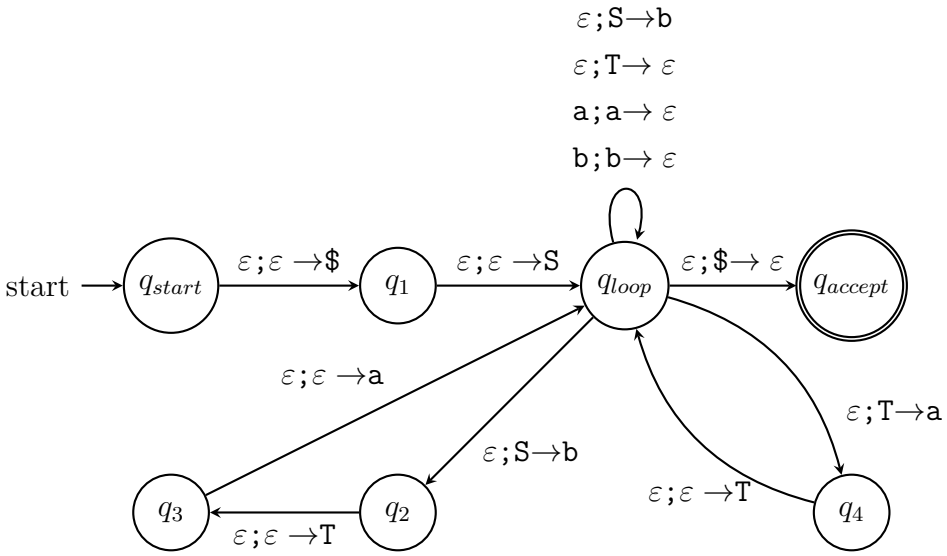
Graficamente un PDA schematizzato secondo queste regole:



Esempio:
Data la grammatica G con regole:

$$S \to aTb|b$$
$$T \to Ta|\varepsilon$$

Il PDA associato sarà:



Conversione da PDA a CFG

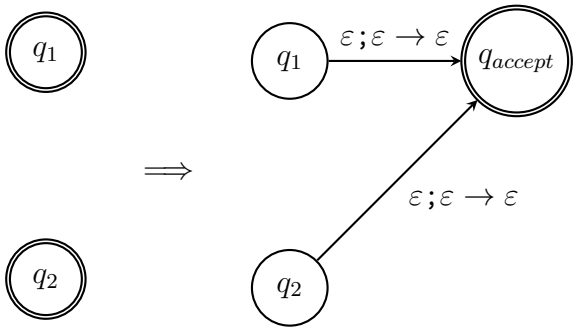
Date le due classi di linguaggi CFL e $\mathcal{L}$(PDA), si ha che:

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

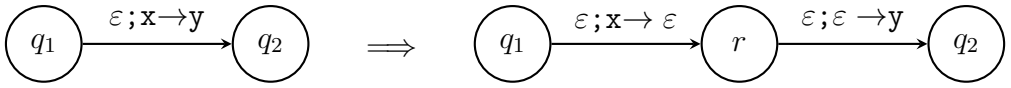
Dimostrazione:
Dato un linguaggio $L \in \mathcal{L}$(PDA), esiste un PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, F)|L(P) = L$.
Trasformiamo P scrivendolo in una forma "canonica" per cui:

- Il PDA inizia e finisce la computazione con la pila vuota, quindi con:

- Ha un solo stato accettante, quindi:



- Il PDA esegue solo un push o pop alla volta (non contemporaneamente), quindi:



Creiamo una CFG $G = (V, \Sigma, R, S)$ tale che:

- $V = \{A_{p,q} | p, q \in Q\}$
- $S = A_{q_0, q_{accept}}$

In cui la variabile $A_{p,q}$ è variabile che deriva tutte le stringe generabili passando dallo stato p allo stato q con stack vuoto, per cui:

- $\forall p \in Q$:
$$(A_{p,p} \to \varepsilon) \in R$$
- $\forall p, q, r \in Q$, nel caso in cui lo stack si svuoti in mezzo nello stato r:
$$(A_{p,q} \to A_{p,r} A_{r,q}) \in R$$
- $\forall p, q, r, s \in Q, u \in \Gamma, a, b \in \Sigma_\varepsilon$, nel caso in cui lo stack non si svuoti in mezzo:
$$(A_{p,q} \to a A_{r,s} b) \in R \iff (r, u) \in \delta'(p, a, \varepsilon) \land (q, \varepsilon) \in \delta(s, b, u)$$

Dati $p, q \in Q, x \in \Sigma^*$, allora:

$$A_{p,q} \overset{*}{\Rightarrow} x \iff x \text{ porta } P \text{ dallo stato } p \text{ allo stato } q \text{ con stack vuoto}$$

Dimostrazione per doppia implicazione:

- $\Longrightarrow$:
  Dimostriamo per induzione sul numero di passi della produzione di x in G:
  - Caso base:
    Con 1 passo l'unica regola possibile per cui $A_{p,q} \overset{*}{\Rightarrow} x$ è la regola $A_{p,q} \to \varepsilon$, quindi è vero che x porta P dallo stato p allo stato $q = p$ con stack vuoto

- Ipotesi induttiva:
  Supponiamo che per ogni stringa x con computazione di passi$\leq k$ per cui $A_{p,q} \overset{*}{\Rightarrow} x$, x porti P da p a q con stack vuoto
- Passo induttivo:
  Con una stringa x con computazione di $k+1$ passi ha due opzioni in base alle due regole possibili:
  1. $A_{p,q} \to aA_{r,s}b$:
     Sia $x = ayb$, per cui $A_{r,s} \overset{*}{\Rightarrow} y$ con un numero di passi $k-1$, per cui per l'ipotesi induttiva y porta P da r a s con stack vuoto. Per costruzione di G:

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

     Quindi $ayb = x$ porta P da p a q con stack vuota.
  2. $A_{p,q} \to A_{p,r}A_{r,q}$:
     Sia $x = yz$, per cui $A_{p,r} \overset{*}{\Rightarrow} y$ e $A_{r,q} \overset{*}{\Rightarrow} z$ con numero di passi $\leq k$, per ipotesi induttiva y porta P da p a r con stack vuoto e z porta P da r a q con stack vuoto. Quindi $yz = x$ porta P da p a q con stack vuoto.

- $\Longleftarrow$:
  Dimostriamo per induzione sul numero di transizioni di P mentre legge x:
  - Caso base:
    Con 0 transizioni, l'unico input è $x = \varepsilon$ che porta P da p a p. Quindi per la regola $A_{p,p} \to \varepsilon$ si ha che $A_{p,p} \overset{*}{\Rightarrow} x$.
  - Ipotesi induttiva:
    Supponiamo che per ogni stringa che porta P da p a q con stack vuoto con numero di transizioni $\leq k$, sia vero che $A_{p,q} \overset{*}{\Rightarrow} x$
  - Passo induttivo:
    Con una stringa x che porta P da p a q con stack vuoto con $k+1$ transizioni ci sono due opzioni possibili in base a se lo stack si svuota o no in mezzo:
    1. Lo stack si suota solo in p e q:
       Allora il simbolo $u \in \Gamma$ inserito nella prima transizione è lo stesso che viene tolto dallo stack nell'ultima. Siano quindi $a, b \in \Sigma_\varepsilon$ i simboli letti nella prima e ultima transizione allora ci sono due stati $r, a \in Q$ per cui vale:

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

       Sia $x = ayb$ per cui x porta P da p a q con stack vuoto, per far si che ciò avvenga allora y deve portare P da r a s con stack vuoto. La computazione di y ha $k-1 < k$ transizioni quindi per ipotesi induttiva $A_{r,s} \overset{*}{\Rightarrow} y$ e allora $A_{p,q} \Rightarrow aA_{r,s}b \overset{*}{\Rightarrow} x$.
    2. Lo stack si svuota in uno stato intermedio r:
       Sia $x = yz$ per cui y porta P da p a r con stack vuoto e z porta P da r a q con stack vuoto, percorrendo entrambe un numero di transizioni minore di k, quindi per ipotesi induttiva $A_{p,r} \overset{*}{\Rightarrow} y$ e $A_{r,q} \overset{*}{\Rightarrow} z$ e allora $A_{p,q} \Rightarrow A_{p,r}A_{r,q} \overset{*}{\Rightarrow} x$.

Concludiamo quindi che:

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

Equivalenza tra CFG e PDA

Date i due lemmi precedenti abbiamo quindi che:

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

2.5 Pumping lemma per i linguaggi acontestuali

Albero di derivazione di una CFG

Data una CFG $G = (V, \Sigma, R, S)$ in forma normale, una stringa $x \in L(G)$ con l'albero di derivazione con altezza h (altezza del ramo di computazione più lungo), si ha che:

$$|x| \leq 2^{h+1}$$

Dimostrazione per induzione:
Dimostriamo per induzione sull'altezza dell'albero h:

- Caso base:
  Con $h = 1$, l'unica regola possibile in forma normale è $S \to a$, per cui $x = a$ e $|x| \leq 2^{1-1} = 1$
- Ipotesi induttiva:
  Suppongo per ogni stringa x con albero di derivazione di altezza k che $|x| \leq 2^{k-1}$
- Passo induttivo:
  Con una stringa x con albero di derivazione di altezza $k+1$, la prima produzione deve essere di tipo $S \to AB$. I due sottoalberi generati da B e da C hanno altezza k per cui per ipotesi induttiva le stinghe generate hanno lunghezza $le2^{k-1}$, quindi:

$$|x| \leq 2^{k-1} \cdot 2 = 2^h$$

Pumping lemma per linguaggi acontestuali

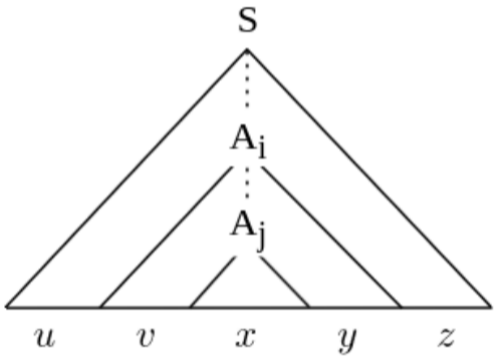
Dato un linguaggio $L \in \text{CFL}$, allora esiste p per cui per ogni w con $|w| \geq p$, scomponendo $w = uvxyz$ si ha che:

1. $uv^kxy^kz \in L \; \forall k \geq 0$
2. $|vy| > 0$
3. $|yxy| \leq p$

Dimostrazione:
Dato un linguaggio $L \in$ CFL, esiste una CFG in forma normale $G = (V, \Sigma, R, S) | L(G) = L$.
Sia $p = 2^m$ con $m = |V|$ e w una stringa con $|w| \geq p$, l'albero di derivazione di w avrà altezza $h \geq m+1$. Il numero di nodi nel ramo di computazione di altezza $m+1$ passa per $m+2$ nodi di cui $m+1$ variabili e 1 teminale. Essendo $|V| = m$, esiste almeno una variabile che si ripete, che denominiamo $A_i = A_j$.
La sottostringhe di w saranno quindi derivate:

- $S \overset{*}{\Rightarrow} uA_iz$
- $A_i \overset{*}{\Rightarrow} vA_jy$
- $A_j \overset{*}{\Rightarrow} x$

Graficamente:



Se nelle formule precendi sostituiamo A_j con A_i possiamo dire che:

$$S \overset{*}{\Rightarrow} uA_iz \overset{*}{\Rightarrow} uvA_iyz \overset{*}{\Rightarrow} uv^kA_iy^kz \overset{*}{\Rightarrow} uv^kxy^kz$$

quindi $uv^kxy^kz \in L(G)\ \forall k \geq 0$.
Visto che G è in forma normale il sottoalbero di A_i deve essere stato creato tramite una regola del tipo $A_i \to BC$ in cui ci sono due casi:

- $B \overset{*}{\Rightarrow} vA_i \wedge C \overset{*}{\Rightarrow} y$
- $B \overset{*}{\Rightarrow} v\wedge \overset{*}{\Rightarrow} A_iy$

Non esistendo ε-regole, si ha che $|vy| > 0$.
Prendendo il ramo di computazione con altezza $m+1$ che produce stringhe di lunghezza massima $2^{m+1-1} = 2^m = p$ allora:

$$|vxy| \leq 2^m = p$$

2.5.1 Dimostrazione di non acontestualità tramite pumping lemma

Il pumping lemma viene usato per dimostrare che un linguaggio non è acontestuale, per farlo basta trovare una stringa che non possa essere scomposta tramite il pumping lemma.
Esempio:

1. Dimostrare che il linguaggio:
$$L = \{0^n 1^n 2^n | n \geq 0\}$$
non è acontestuale.
Suppongo che il linguaggio sia acontestuale, quindi esiste p per cui $\forall w \in L$ con $|w| \geq p$, vale il pumping lemma.
Sia $w = 0^p 1^p 2^p$ con $|w| = 3p$, scompongo $w = uvxyz \in L$ in cui vxy possono essere in 2 modi:

   - vxy contiene un solo simbolo: allora uv^0xy^0z ha meno elementi di quel determinato simbolo rispetto agli altri, quindi $vxy \notin L$
   - vxy contiene due simboli: allora uv^0xy^0z ha meno elementi di quei due simboli rispetto agli altri, quindi $vxy \notin L$

   Non esistendo nessuna scomposizione di w per cui vale il pumping lemma allora $L \notin$ CFL.

2. Dimostrare che il linguaggio
$$L = \{ww | w \in \{0,1\}^*\}$$
non è acontestuale:
Suppongo che il linguaggio sia acontestuale, quindi esiste p per cui $\forall w \in L$ con $|w| \geq p$, vale il pumping lemma.
Sia $x = 0^p 1^p 0^p 1^p$ con $|w| = 4p$, scompongo $x = uvxyz \in L$ in cui vxy possono essere in 3 modi:

   - vxy contiene solo un simbolo: allora uv^0xy^0z ha una metà con meno elementi di quel simbolo rispetto all'altra metà, quindi $uv^0xy^0z \notin L$
   - vxy contiene sia 0 che 1 di una delle due metà: allora uv^0xy^0z ha il centro spostato e le due metà saranno diverese, quindi $uv^0xy^0z \notin L$
   - vxy contiene gli 1 della prima metà e gli 0 della seconda: allora uv^0xy^0z ha la prima metà con meno 1 della seconda e la seconda metà con meno 0 della prima, quindi $uv^0xy^0z \notin L$

   Non esistendo nessuna scomposizione di w per cui vale il pumping lemma allora $L \notin$ CFL.

2.6 Chiusura dei linguaggi acontestuali

Chiusura dell'unione in CFL

L'unione è chiusa in CFL, cioè:

$$\forall L_1, \dots, L_k \in \text{CFL} \implies \bigcup_{i=1}^{k} L_i \in \text{CFL}$$

Dimostrazione:
Dati $L_1, \dots, L_k \in \text{CFL}$, allora esistono le CFG $G_1, \dots, G_k$ per cui $G_i = (V_i, \Sigma_i, R_i, S_i) | L(G_i) = L_i$.
Creo la CFG $G = (V, \Sigma, R, S)$ in cui:

- S è una nuova variabile iniziale
- $V = \bigcup_{i=0}^{k} V_i \cup \{S\}$
- $\Sigma = \bigcup_{i=0}^{k} \Sigma_i$
- $R = \bigcup_{i=0}^{k} R_i \cup \{S \to S_j | j \in [1, k]\}$

Presa una strina $w \in \bigcup_{i=1}^{k} L_i$, allora $\exists j | w \in L_j$, ma visto che esiste $(S \to S_j) \in R$ allora:

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

Data invece $w \in L(G)$, le uniche regole applicabili su S sono $\{S \to S_j | j \in [1, k]\}$, allora:

$$w \in L(G) \implies \exists j | S \Rightarrow S_j \overset{*}{\Rightarrow} w \implies w \in L_j \subseteq \bigcup_{i=1}^{k} L_i$$

Quindi:

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

Chiusura della concatenazione in CFL

La concatenazione è chiusa in CFL, cioè:

$$\forall L_1, \dots, L_k \in \text{CFL} \implies L_1 \circ \dots \circ L_k \in \text{CFL}$$

Dimostrazione:
Dati $L_1, \dots, L_k \in$ CFL, allora esistono le CFG $G_1, \dots, G_k$ per cui $G_i = (V_i, \Sigma_i, R_i, S_i) | L(G_i) = L_i$.
Creo la CFG $G = (V, \Sigma, R, S)$ in cui:

- S è una nuova variabile iniziale
- $V = \bigcup_{i=0}^{k} V_i \cup \{S\}$
- $\Sigma = \bigcup_{i=0}^{k} \Sigma_i$
- $R = \bigcup_{i=0}^{k} R_i \cup \{S \to S_1 \dots S_k\}$

Presa una strina $w = w_1 \dots w_k \in L_1 \circ \dots \circ L_k$, allora $\forall j \; w_j \in L_j$, ma visto che esiste $(S \to S_1 \dots S_k) \in R$ allora:

$$\forall j \; w_j \in L_j \iff S_j \overset{*}{\Rightarrow} w_j \implies S \Rightarrow S_1 \dots S_k \overset{*}{\Rightarrow} w \implies w \in L(G)$$

Data invece $w \in L(G)$, l'unica regola applicabile su S è $(S \to S_1 \dots S_k)$, allora:

$$w \in L(G) \implies S \overset{*}{\Rightarrow} w = w_1 \dots w_k \implies w \in L_1 \circ \dots \circ L_k$$

Quindi:

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

Non chiusura dell'intersezione in CFL

L'intersezione **non** è chiusa in CFL, cioè:

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

Dimostrazione:
Dati i linguaggi:

$$L_1 = \{0^n 1^n 2^i | n, i \geq 0\}$$
$$L_2 = \{0^i 1^n 2^n | n, i \geq 0\}$$

Questi linguaggi sono rappresentati dalle grammatiche:

G_1 :
$S \to AB$
$A \to 0A1 | \varepsilon$
$B \to 2B | \varepsilon$

G_2 :
$S \to AB$
$A \to 0A | \varepsilon$
$B \to 1B2 | \varepsilon$

L'intersezione di questi linguaggi è:

$$L_1 \cap L_2 = \{0^n 1^n 2^n | n \geq 0\}$$

che abbiamo dimostrato precedentemente non essere acontestuale.

Non chiusura del complemento in CFL

Il complemento **non** è chiusa in CFL, cioè:

$$\exists L \in \text{CFL} \,|\neg L \notin \text{CFL}$$

Dimostrazione:
Dato il linguaggio:

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

e la grammatica associata:

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

Sia $x \in L$ una stringa per cui $|x|$ è dispari, allora $x \in L(G)$.
Sia x una stringa per cui $|x|$ è pari allora dobbiamo dimostrare che:

- $x \in L \implies S \overset{*}{\Rightarrow} x$:
  Scrivo $x = x_1 \dots x_n$, per cui:

  $$x \in L \implies \exists i | x_i \neq x_{i+\frac{n}{2}}$$

  Dividiamo $x = uv$ con $|u|, |v|$ dispari per cui $u = x_1 \dots x_{2i-1}$ e $v = x_{2i} \dots x_n$. Il simbolo centrale di u è x_i mentre quello di v è $x_{i+\frac{n}{2}}$, per cui:

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

  Visto che $|u|, |v|$ sono dispari allora $u, v \in L$, quindi:

  $$S \overset{*}{\Rightarrow} uv = x \implies x \in L(G)$$

- $S \overset{*}{\Rightarrow} x \implies x \in L$:
  Essendo $|x|$ pari, deve essere stata generata dalla regola $S \to AB$ oppure $S \to BA$. Consideriamo il caso in cui $S \to AB$.
  Scriviamo $x = uv$ in cui $u = x_i \dots x_k$ e $v = x_{k+1} \dots x_n$, allora $S \Rightarrow A \overset{*}{\Rightarrow} u$ e $S \Rightarrow B \overset{*}{\Rightarrow} v$ per cui abbiamo $|u|, |v|$ dispari. Il simbolo centrale di u è $x_{\frac{k+1}{2}} = a$ e quello di v è $x_{\frac{k+n+1}{2}} = b$.
  Dividiamo $x = w'w''$ tali che $|w'| = |w''| = \frac{n}{2}$, allora:

  $$w'_{\frac{k+1}{2}} = x_{\frac{k+1}{2}} \neq x_{\frac{k+l+1}{2}} = w''_{\frac{k+1}{2}} \implies w' \neq w'' \implies x = w'w'' \in L$$

Quindi $L = L(G) \in \text{CFL}$.
Il complemento di L è $\neg L = \{ww | w \in \{a,b\}^*\}$, che abbiamo dimostrato precedentemente non essere acontestuale.

3

Calcolabilità

3.1 Macchina di Turing (TM)

La **macchina di Turing** è un'estensione del modello di calcolo visto fin'ora, ed è un'astrazione di un computer.
Ha diverse caratteristiche:

- Utilizza un nastro di lavoro (memoria) di lunghezza infinita a destra, dentro cui viene copiato l'input all'inizio della computazione
- Ha una testina di lettura che si sposta a sinistra e destra
- Ha uno stato di acettazione e rifiuto, che però a differenza degli automi sono immediati

Macchina di Turing (TM)

Una **Turing Machine (TM)** è una tupla $(Q, \Sigma, \Gamma, \delta, q_{start}, q_{accept}, q_{reject})$ in cui:

- Q è l'insieme degli stati della macchina
- Σ è l'alfabeto in input, per cui $\sqcup \notin \Sigma$
- Γ è l'alfabeto del nastro, per cui $\sqcup \in \Gamma$ e $\Sigma \subseteq \Gamma$
- q_{start} è lo stato iniziale
- q_{accept} è lo stato accettante
- q_{reject} è lo stato rifiutante
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ la funzione di transizione per cui se $\delta(q_1, a) = (q_2, b, X)$:
  L'automa legge il simbolo a dal nastro, lo sostituisce con b passando dallo stato q_1 a q_2 e sposta il nastro a destra se $X = R$ e a sinistra se $X = L$. Viene scritto $a \to b; X$.
  Il simbolo $\sqcup$ indica uno spazio vuoto nel nastro.

Configurazione di una TM

Data una TM $M = (Q, \Sigma, \Gamma, \delta, q_{start}, q_{accept}, q_{reject})$ definiamo la stringa

$$uqav$$

come la **configurazione** di M, per cui:

- $u, v \in \Gamma^*$ sono i simboli prima e dopo a nel nastro
- $a \in \Gamma$ è il simbolo in cui sta la testina
- $q \in Q$ è lo stato attuale della macchina

La configurazione iniziale è sempre $q_{start}w$ in cui w è la stringa in input.

Produzione in una TM

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

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

Quindi data una stringa w, sarà accettata da M se esiste una serie di configurazioni $c_1, \ldots, c_k$, tali che:

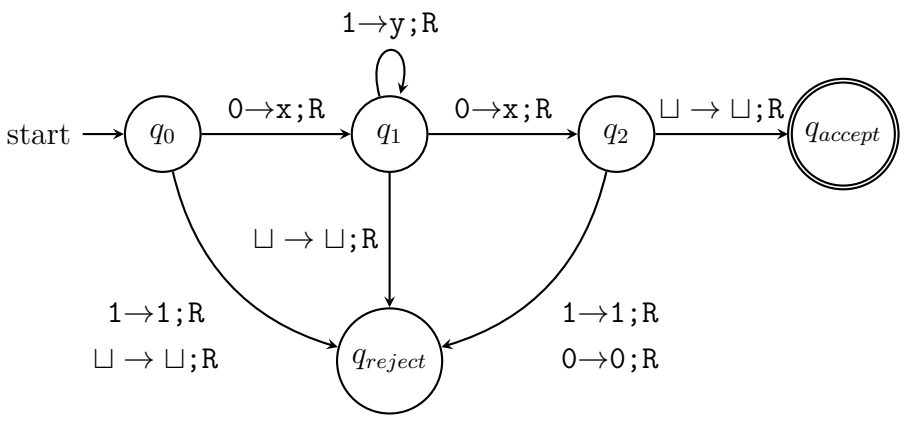
- $c_1 = q_0w$
- c_i produce c_k $\forall i$
- c_k contiene uno stato accettante (q_{accept})

Possiamo inoltre definire il linguaggio:

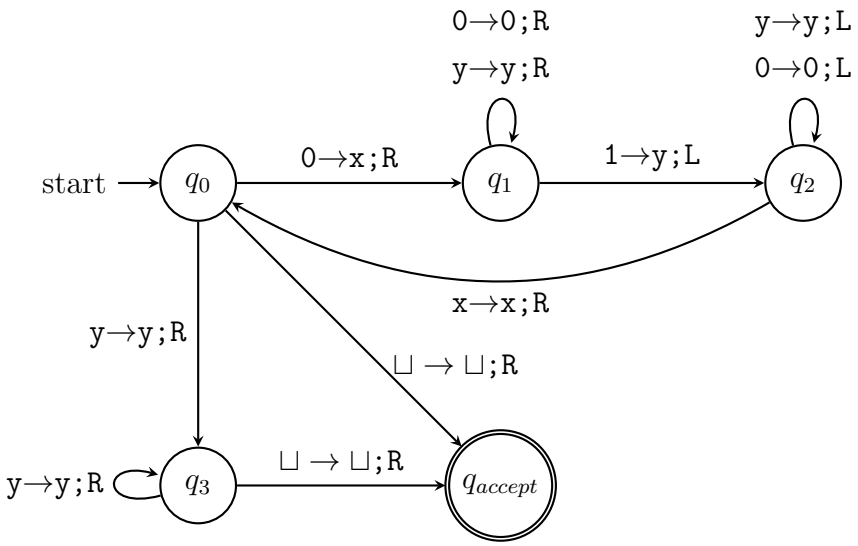
$$L(M) = \{w \in \Sigma^* | M \text{ accetta } w\}$$

Esempi:

1. Dato il linguaggio $L = \{01^*0\}$, la TM che lo riconosce è:



2. Dato il linguaggio $L = \{0^n1^n | n \geq 0\}$, la TM che lo riconosce è:



Tutte le transizioni omesse vengono considerate come se vanno allo stato q_{reject}.

Classe dei linguaggi Turing-riconoscibili

Dato un alfabeto Σ, la **classe dei linguaggi Turing-riconoscibili** di Σ, scritta come REC, è l'insieme dei linguaggi per cui esiste una macchina di Turing che li accetta:

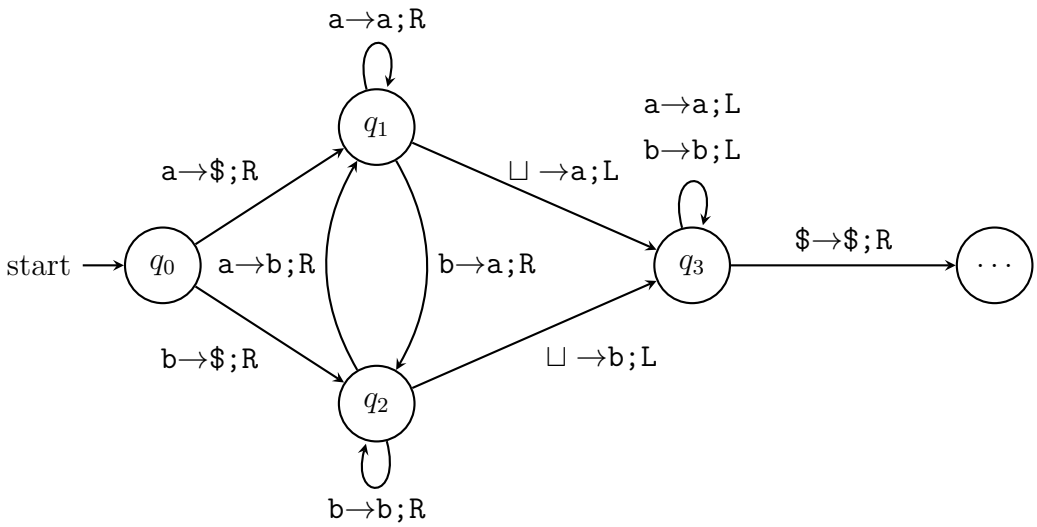
$$\text{REC} = \{L \subseteq \Sigma^* | \exists \, \text{TM} \, M \; L(M) = L\}$$

Classe dei linguaggi Turing-decidibili

Una TM M viene definita **decisore** se termina sempre la sua esecuzione, non andando mai in loop.
Dato un alfabeto Σ, la **classe dei linguaggi Turing-decidibili** di Σ, scritta come DEC, è l'insieme dei linguaggi per cui esiste una macchina di Turing che li decide:

$$\text{REC} = \{L \subseteq \Sigma^* | \exists \, \text{TM} \;\; \text{decisore} \; M \; L(M) = L\}$$

Le macchine di Turing hanno un nastro illimitato a destra, quindi è facile capire dove finisce la stringa in input (è semplicemente seguita da infiniti $\sqcup$), ma se si volesse capire la fine del nastro a sinistra bisognerebbe marcarlo in qualche modo. L'automa che permette di marcare l'inizio del nastro con un simbolo `$` spostando tutta la stringa in input di uno spazio più a destra cambia in base all'alfabeto Σ, nel caso l'alfabeto sia $\Sigma = \{a, b\}$:



Lo stato indicato dai puntini di sospensione (...), indica un possibile inizio di un automa che necessita di marcare il limite sinistro del nastro.
Se si volesse invece marcare un simbolo sul nastro, senza però modificarne il significato (potrebbe servire per calcoli successivi), basta aggiungere all'alfabeto Γ una versione marcata di ogni simbolo in Σ da sostituire per marcarli. Per esempio se l'alfabeto è $\Sigma = \{a, b, c\}$ aggiungiamo a Γ la versione marcata di ogni simbolo facendolo diventare $\Gamma = \{a, b, c, \sqcup, \dot{a}, \dot{b}, \dot{c}\}$.

3.1.1 Varianti di TM

TM stazionaria

Una **TM stazionaria** è una TM che permette alla testina di rimenere ferma dopo una transizione, la funzione di transizione δ sarà quindi definita come:

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

in sui S sta appunto per la possibilità della testina di restare ferma.
Per ogni TM stazionaria M' esiste una TM M equivalente.

Dimostrazione:
Qualsiasi transizione in M' con spostamento R o L è presente anche in M.
Le transizioni in M' con spostamento S, possono essere simulate creando una serie di regole che permette di muoversi in una direzione senza modificare il nastro. Una transizione in M' del tipo:

$$\delta(q, a) = (p, b, S)$$

è equivalente alle transizioni in M:

$$\delta(q, a) = (r, b, L)$$
$$\delta(r, x) = (p, x, R)\ \forall x \in \Gamma$$

TM multinastro

Una **TM multinastro** è una TM con k nastri, ognuno con la propria testina, e con la possibilità ad ogni transizione di aggiornare tutti i nastri, la funzione di transizione δ sarà quindi definita come:

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$$

Per ogni TM multinastro M' esiste una TM M equivalente.

Dimostrazione:
Definisco in M un simbolo $\#\notin \Gamma'$ che utilizzo per separare i nastri, in modo da poterli rappresentare su un solo nastro. Per segnare la posizione di tutte le testine invece aggiungo una versione marcata per ogni simbolo in Γ'. La configurazione iniziale di M, considerando che l'input in M' viene posizionato sul primo nastro, sarà:

$$\#\dot{w}_1 \dots w_n \# \dot{\sqcup} \# \dots \# \dot{\sqcup} \#$$

Un passo di computazione invece scorrerà tutto il nastro e controllerà tutti i simboli marcati e aggiornerà i nastri in base a δ'. Se una testina dovesse spostarsi su un $\#$, allora M sposterà prima tutti gli elementi successivi a destra e poi eseguirà lo spostamento.

TM non deterministica

Una **TM non deterministica (NTM)** è una TM in cui ogni transizione può finire su più di uno stato, la funzione di transizione δ sarà quindi definita come:

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

in cui una stringa w viene accettata se almeno un ramo della computazione accetta.
Per ogni TM non deterministica N esiste una TM M equivalente.

Dimostrazione:
Dato l'albero di computazione di N e considerando la radice dell'albero come ε, possiamo identificare ogni nodo dell'albero con un indirizzo che rappresenta le scelte non deterministiche che sono state eseguite per raggiungere quel nodo. M controllerà questo albero per livello (per evitare di entrare in un ramo di lunghezza infinita) e accetterà non appena troverà un nodo che accetta la stringa in input. M usa 3 nastri:

1. Nastro con l'input w, che non viene mai cambiato
2. Nastro di lavoro, che esplora un determinato cammino dell'albero
3. Nastro che contine l'indirizzo i del nodo corrente dell'albero di computazione

Ad ogni step M simulerà un ramo di computazione dalla radice fino all'indirizzo i, usando il secondo nastro come nastro di lavoro. Nel caso questo ramo accettasse, M accetterebbe, sennò va al prossimo indirizzo.

Enumeratore

Un **enumeratore** è una TM E che produce stringhe di un linguaggio in ordine casuale, con possibili ripetizioni. Può essere considerato come un generatore di linguaggi.
Dato un linguaggio L, allora:

$$L \in \text{REC} \iff \exists \text{enumeratore} E | L(E) = L$$

Dimostrazione per doppia implicazione:

1. $L \in \text{REC} \implies \exists E | L(E) = L$:
   Definisco la TM M per cui $L(M) = L$, l'enumeratore E di tutte le stringhe $s_1, s_2, \ldots, s_k \in \Sigma^*$ deve stampare solo quelle in L.
   Faccio eseguire ad E l'algoritmo:

Algoritmo: Enumeratore simula TM

```
for i in range(0,∞) :
    for j in range(0,i) :
        accept=M(s_j,i)
        if accept :
            return s_j
```

   In cui `M(`s_j`,i)` simula la TM M con input s_j per i passi di computazione e ritorna `True` se la stringa s_j viene accettata.

2. $L \in \text{REC} \impliedby \exists E | L(E) = L$:
   Definiamo una TM M che simula E e confronta l'output con w, accettandolo quando è uguale (in un tempo infinito succederà).

3.2 Linguaggi decidibili

Codifica binaria di un oggetto

Dato un **oggetto** O, definiamo con $< O >$ la sua **codifica binaria**, cioè una stringa binaria che ne descrive le caratteristiche.

Accettazione di DFA

Dato il linguaggio:

$$A_{\text{DFA}} = \{< D, w > | D \in \text{DFA} \land w \in L(D)\}$$

è decidibile.

Dimostrazione:
La TM M con input $< D, w >$:

1. Interpreta la codifica di D, se è sbagliata rifiuta
2. Simula D con input w

3. Se D termina con uno stato accettante, M accetta, sennò rifiuta.

Visto che i DFA terminano semple il linguaggio $A_{\text{DFA}} \in \text{DEC}$.

Analogamente anche i linguaggi:

$$A_{\text{NFA}} = \{< N, w > | N \in \text{DFA} \land w \in L(N)\}$$
$$A_{\text{REX}} = \{< R, w > | D \in \text{re} \land w \in L(R)\}$$

sono decidibili.

Linguaggi vuoti di DFA

Il linguaggio:

$$E_{\text{DFA}} = \{< D > | D \in \text{DFA} \land L(D) = \emptyset\}$$

è decidibile.

Dimostrazione:

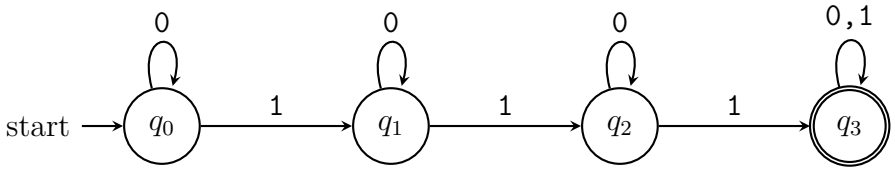
E

Esercizi

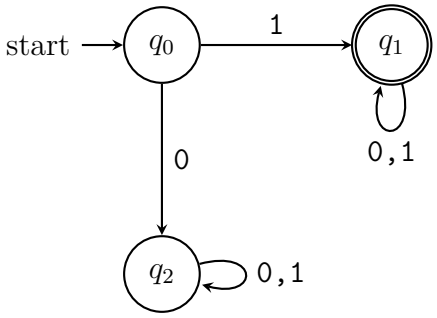
E.1 Esercizi sui linguaggi regolari

E.1.1 Costruire un automa da un linguaggio

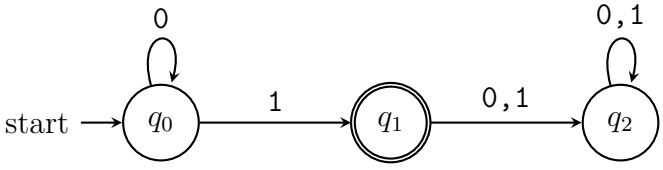
1. Dato un linguaggio $L(D) = \{x \in \{0,1\}^* | w_H(x) \geq 3\}$, per cui $w_H(x) = \{\text{n}^\circ \text{ di 1 in } x\}$, costruire un DFA che accetta questo linguaggio:



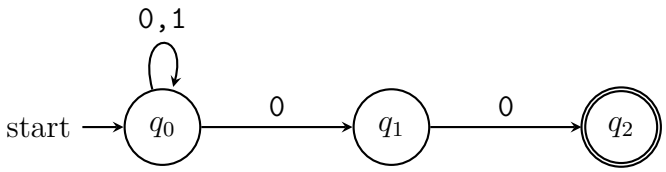
2. Dato un linguaggio $L(D) = \{x \in \{0,1\}^* | x = 1y \wedge y \in 0,1^*\}$, costruire un DFA che accetta questo linguaggio:



3. Dato un linguaggio $L(D) = \{x \in \{0,1\}^* | x = 0^n 1\}$, costruire un DFA che accetta questo linguaggio:



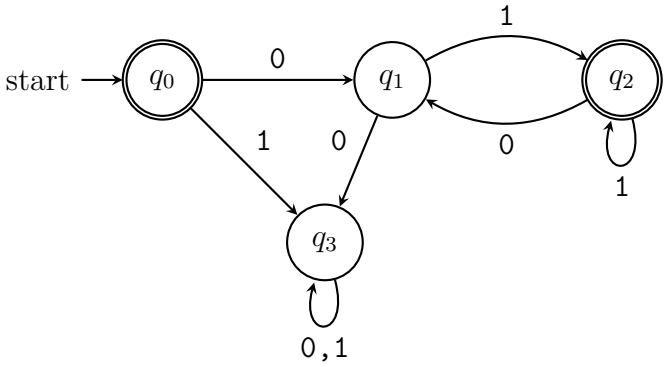
4. Dato un linguaggio $L = \{w \in \{0,1\}^* | w = x00\ x \in \{0,1\}^*\}$, costruire un NFA che accetta questo linguaggio:



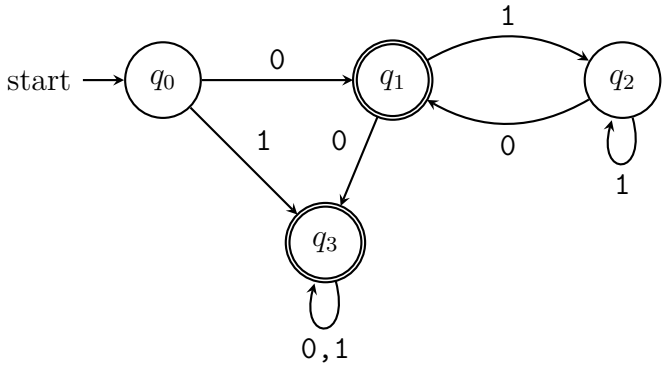
Dimostrazione per induzione:

- Caso base: $w = 00 \implies w \in L$
- Passo induttivo: $w = w'00$ con $w' = \{0,1\}^*$
  $w \in L$ perchè $\forall w'$ c'è sempre un ramo di computazione che si trova nello stato q_0. Quindi leggendo 00 alla fine arriva nello stato q_2, quindi $w \in L$.

5. Dato un linguaggio $L = \{w \in \{0,1\}^* | w \notin (01^+)^*\}$, costruire un DFA che accetta questo linguaggio:
   Posso creare un DFA che accetta il linguaggio complementare $\neg L = \{w \in (01^+)^*\}$:

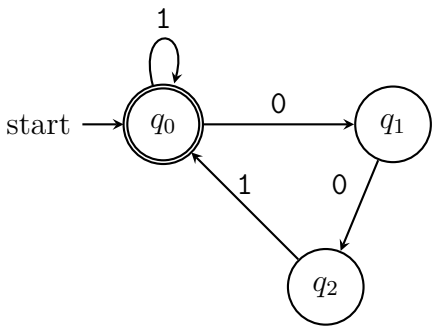


   Quindi il DFA che accetta il linguaggio iniziale è quello con gli stati accettanti invertiti rispetto a quello sopra:



E.1.2 Costruire un automa da un'espressione regolare

1. Data l'espressione regolare $r = 1^*(001^+)^*$ costruire il DFA equivalente a r:



E.1.3 Dimostrare che un linguaggio non è regolare

1. Dimostrare che il linguaggio

$$L = \{ww^R | w \in \{0,1\}^*\}$$

dove w^R è w rovesciata ($w = 100 \implies w^R = 001$) non è regolare:
Presa la stringa $w = 0^p 110^p \in L$ con $|w| \geq p$ e $|xy| < p$ allora y contiene solo 0.
Scrivo $w = 0^k 0^l 0^m 110^p$ con:

- $x = 0^k$ $k \geq 0$
- $y = 0^l$ $l > 0$
- $z = 0^m 110^p$
- $k + l + m = p$

Con $i = 2$ la stringa $xy^2z = 0^k 0^{2l} 0^m 110^p \implies k + 2l + m > p \implies xy^2z \notin L \implies L$ non è regolare.

2. Dimostrare che il linguaggio

$$L = \{1^{n^2} | n \geq 0\}$$

non è regolare:
Presa la stringa $w = 1^{p^2}$ con $|w| > p$ e $|xy| < p$.
Scrivo $w = 1^k 1^l 1^{p^2-k-l}$ con:

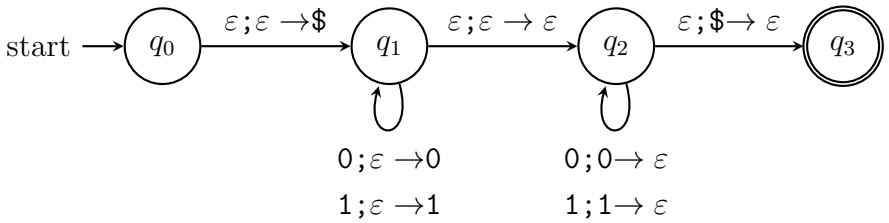
- $x = 1^k$ $k \geq 0$
- $y = 1^l$ $l > 0$
- $z = 1^{p^2-l-k}$
- $k + l \leq p$

Con $i = 2$ la stringa $xy^2z = 1^k 1^{2l} 1^{p^2-l-k} = 1^{p^2+l} \implies p^2 < |xy^2z| < (p+1)^2 \implies xy^2z \notin L \implies L$ non è regolare.

E.2 Esercizi sui linguaggi acontestuali

E.2.1 Costruire un automa da un linguaggio acontestuale

1. Dato il linguaggio $L = \{ww^R | w \in \{0,1\}^*\}$ dove w^R è w rovesciata ($w = 100 \implies w^R = 001$), costruire il PDA associato:


start
q_0
$\varepsilon;\varepsilon \to \$$
q_1
$\varepsilon;\varepsilon \to \varepsilon$
q_2
$\varepsilon;\$ \to \varepsilon$
q_3
$0;\varepsilon \to 0$
$1;\varepsilon \to 1$
$0;0 \to \varepsilon$
$1;1 \to \varepsilon$