



SAPIENZA
UNIVERSITÀ DI ROMA

“SAPIENZA” UNIVERSITÀ DI ROMA
INGEGNERIA DELL'INFORMAZIONE,
INFORMATICA E STATISTICA
DIPARTIMENTO DI INFORMATICA

Automi, Calcolabilità e Complessità

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

Author
Simone Bianco

28 ottobre 2023

Indice

Informazioni e Contatti	1
1 Linguaggi e Automi	2
1.1 Linguaggi	2
1.2 Determinismo	3
1.3 Non determinismo	8
1.3.1 Equivalenza tra NFA e DFA	10
1.4 Linguaggi regolari	13
1.5 Espressioni regolari	20
1.5.1 NFA generalizzati	23
1.5.2 Equivalenza tra espressioni e linguaggi regolari	29
1.6 Linguaggi non regolari	29
1.6.1 Pumping lemma per i linguaggi regolari	30
1.7 Esercizi svolti	33
2 Grammatiche e Linguaggi acontestuali	37
2.1 Grammatiche acontestuali	37

Informazioni e Contatti

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

Ulteriori informazioni ed appunti possono essere trovati al seguente link:

<https://github.com/Exyss/university-notes>. Chiunque si senta libero di segnalare incorrettezze, migliorie o richieste tramite il sistema di Issues fornito da GitHub stesso o contattando in privato l'autore :

- Email: bianco.simone@outlook.it
- LinkedIn: [Simone Bianco](#)

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

Prerequisiti consigliati per lo studio:

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

Licence:

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

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

1

Linguaggi e Automi

1.1 Linguaggi

Definizione 1: Alfabeto

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

Esempio:

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

Definizione 2: Stringa

Dato un alfabeto Σ , definiamo come **stringa di Σ** una sequenza di simboli $x_1x_2 \dots x_n$ dove $x_1, \dots, x_n \in \Sigma$ e $n \in \mathbb{N}$.

In particolare, indichiamo come ε la **stringa vuota**

Esempio:

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

Definizione 3: Linguaggio

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

In particolare, notiamo che $\varepsilon \in \Sigma^*$ per qualsiasi linguaggio Σ^*

Definizione 4: 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** la seguente operazione:

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

Definizione 5: Potenza

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

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

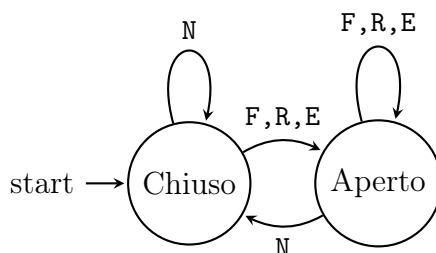
1.2 Determinismo

Definizione 6: 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)

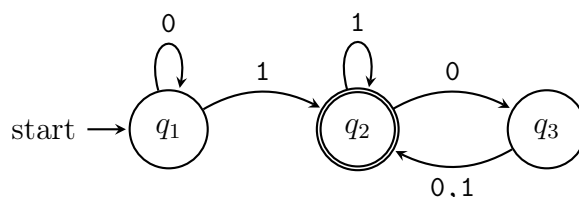
Definizione 7: Deterministic Finite Automaton (DFA)

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

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

Esempio:

- Consideriamo il seguente DFA



dove:

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

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

è la funzione di transizione degli stati dell'automa

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

Definizione 8: Funzione di transizione estesa

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

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

Proposizione 1: Stringa accettata in un DFA

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

Esempio:

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

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

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

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

Definizione 9: Linguaggio di un DFA

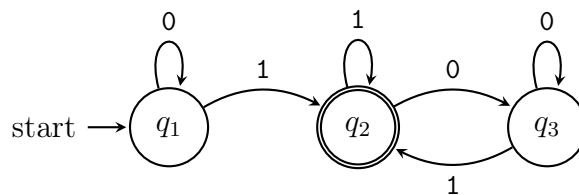
Sia $D := (Q, \Sigma, \delta, q_0, F)$ un DFA. Definiamo come **linguaggio di D** , indicato come $L(D)$, l'insieme di stringhe accettate da D

$$L(D) = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in F\}$$

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

Esempi:

- Consideriamo il seguente DFA D



- Il linguaggio riconosciuto da tale DFA corrisponde a

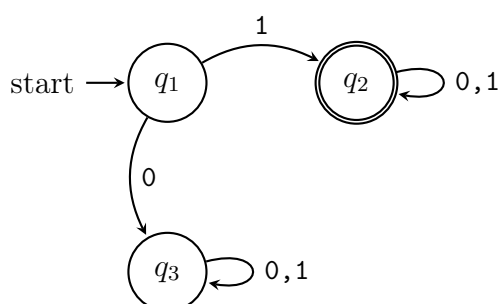
$$L(D) = \{x \in \{0, 1\}^* \mid x := y1, \exists y \in \{0, 1\}^*\}$$

ossia al linguaggio composto da tutte le stringhe terminanti con 1

- Consideriamo il seguente linguaggio

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

- Un DFA in grado di riconoscere tale linguaggio corrisponde a

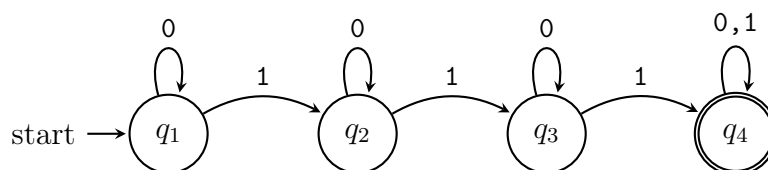


- Consideriamo il seguente linguaggio

$$L = \{x \in \{0, 1\}^* \mid w_H(x) \geq 3\}$$

dove w_H è il **peso di Hamming** (ossia $w_H(x) = \text{numero di "1" in } x$)

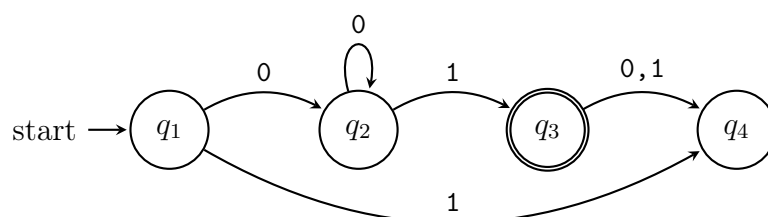
- Un DFA in grado di riconoscere tale linguaggio corrisponde a



- Consideriamo il seguente linguaggio

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

- Un DFA in grado di riconoscere tale linguaggio corrisponde a



Definizione 10: Configurazione di un DFA

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

Definizione 11: Passo di computazione

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

$$(p, ax) \vdash_D (q, x) \iff \delta(p, a) = q$$

Definizione 12: Computazione deterministica

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

$$\forall (q, ax) \exists! (p, x) \mid (q, ax) \vdash_D (p, x)$$

Proposizione 2: 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:

- $(p, ax) \vdash_D (q, x) \implies (p, ax) \vdash_D^* (q, x)$
- $\forall q \in Q, x \in \Sigma^* \quad (q, x) \vdash_D^* (q, x)$
- $(p, aby) \vdash_D (q, by) \wedge (q, by) \vdash_D (r, y) \implies (p, aby) \vdash_D^* (r, y)$

Osservazione 1

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

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

(*dimostrazione omessa*)

1.3 Non determinismo

Definizione 13: Alfabeto epsilon

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

Definizione 14: Non-deterministic Finite Automaton (NFA)

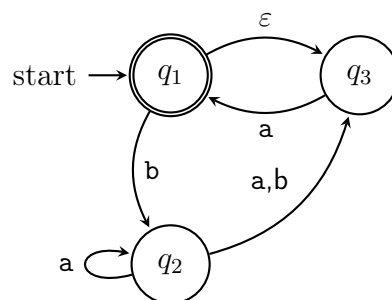
Un **Non-deterministic Finite Automaton (NFA)** (o *Automa Non-deterministico a Stati Finiti*) è una quintupla $(Q, \Sigma, \delta, q_0, F)$ dove:

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

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

Esempio:

- Consideriamo il seguente NFA



dove:

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

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

è la funzione di transizione degli stati dell'automa

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

Proposizione 3: Stringa accettata in un NFA

Sia $N := (Q, \Sigma, \delta, q_0, F)$ un NFA. Data una stringa $x := x_0 \dots x_k \in \Sigma_\varepsilon^*$, diciamo che x è **accettata da** N se esiste una sequenza di stati $r_0, r_1, \dots, r_{k+1} \in Q$ tali che:

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

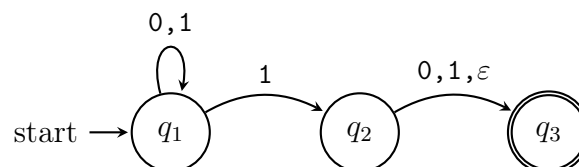
Osservazione 2: Computazione in un NFA

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

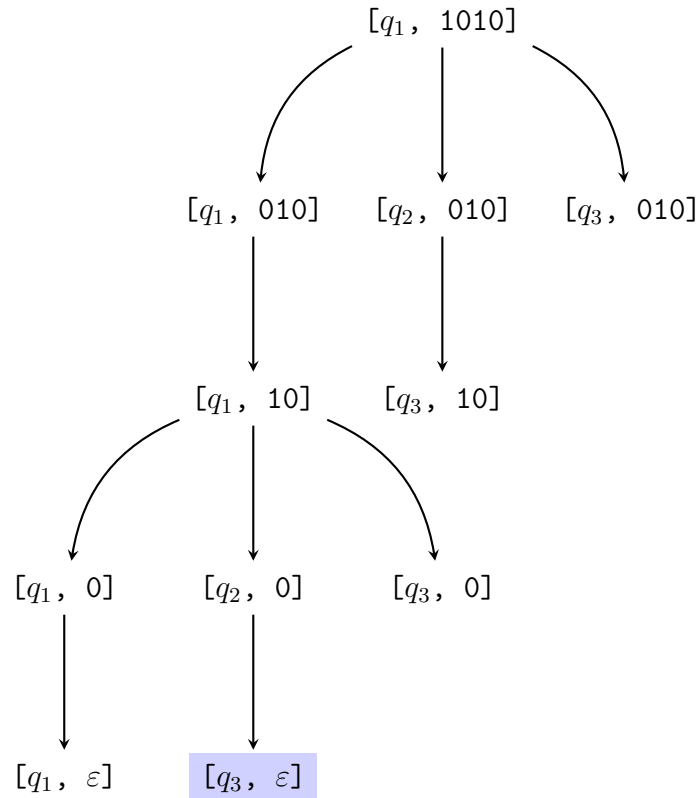
- Tutte le volte che uno stato potrebbe avere più transizioni per diversi simboli dell'alfabeto, l'automa N si duplica in **più copie**, ognuna delle quali segue il suo corso. Si vengono così a creare più **rami di computazione** indipendenti che sono eseguiti in **parallelo**.
- Se il prossimo simbolo 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 $x = 1010$:



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

1.3.1 Equivalenza tra NFA e DFA

Definizione 15: 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 16: 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:

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

Teorema 1: 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:

$$E(R) = \{q \in Q_N \mid q \text{ è raggiungibile in } N \text{ da } q' \in R \text{ tramite } k \geq 0 \text{ } \varepsilon\text{-archi}\}$$
 - $q_{0_D} = E(\{q_{0_N}\})$
 - $F_D = \{R \in Q_D \mid R \cap F_N \neq \emptyset\}$
 - Dati $R \in Q_D$ e $a \in \Sigma$, definiamo δ_D come:

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

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

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

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

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

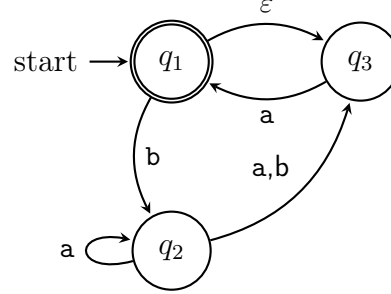
□

Osservazione 3

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

Esempio:

- Consideriamo ancora il seguente NFA



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

$$\begin{aligned}
 Q_D &= \{\emptyset, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_2, q_3\}, \{q_1, q_3\}, \{q_1, q_2, q_3\}\} = \\
 &= \{\emptyset, q_1, q_2, q_3, q_{1,2}, q_{2,3}, q_{1,3}, q_{1,2,3}\}
 \end{aligned}$$

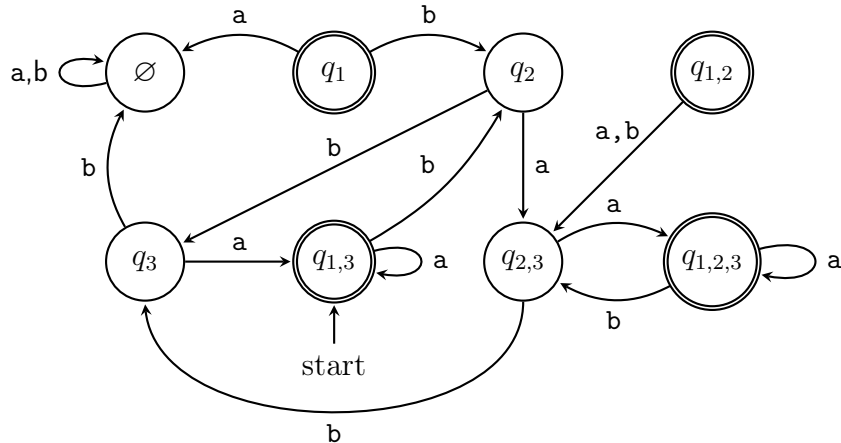
- A questo punto, lo stato iniziale sarà $q_{0_D} = E(\{q_{0_N}\}) = E(\{q_1\}) = \{q_1, q_3\} = q_{1,3}$, mentre gli stati accetanti saranno:

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

- Le transizioni del DFA corrisponderanno invece a:

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

- Il DFA equivalente corrisponde dunque a:



1.4 Linguaggi regolari

Definizione 17: 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:

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

Proposizione 4: Operazioni sui linguaggi

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

- Operatore unione:

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

- Operatore intersezione:

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

- Operatore complemento:

$$\neg L = \{x \in \Sigma^* \mid x \notin L\}$$

- Operatore concatenazione:

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

- Operatore potenza:

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

- Operatore star di Kleene:

$$L^* = \{x_1 \dots x_k \in \Sigma^* \mid k \geq 0, \forall i \in [1, k] \ x_i \in L\} = \bigcup_{n \geq 0} L^n$$

- Operatore plus di Kleene:

$$L^+ = \{x_1 \dots x_k \in \Sigma^* \mid k \geq 1, \forall i \in [1, k] \ x_i \in L\} = \bigcup_{n \geq 1} L^n = L \circ L^*$$

Corollario 1

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

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

Teorema 2: Chiusura dell'unione in REG

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

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

Dimostrazione I.

- Dati $L_1, L_2 \in \text{REG}$, siano $D_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ e $D_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ i due DFA tali che $L_1 = L(D_1)$ e $L_2 = L(D_2)$
- Definiamo quindi il DFA $D = (Q, \Sigma, \delta, q_0, F)$ tale che:

- $q_0 = (q_1, q_2)$
- $Q = Q_1 \times Q_2$
- $F = (F_1 \times Q_2) \cup (Q_1 \times F_2) = \{(r_1, r_2) \mid r_1 \in F_1 \vee r_2 \in F_2\}$
- $\forall (r_1, r_2) \in Q, a \in \Sigma$ si ha che:

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

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

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

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

□

Dimostrazione II.

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

- q_0 è un nuovo stato iniziale aggiunto
- $Q = Q_1 \cup Q_2 \cup \{q_0\}$
- $F = F_1 \cup F_2$
- $\forall q \in Q, a \in \Sigma$ si ha che:

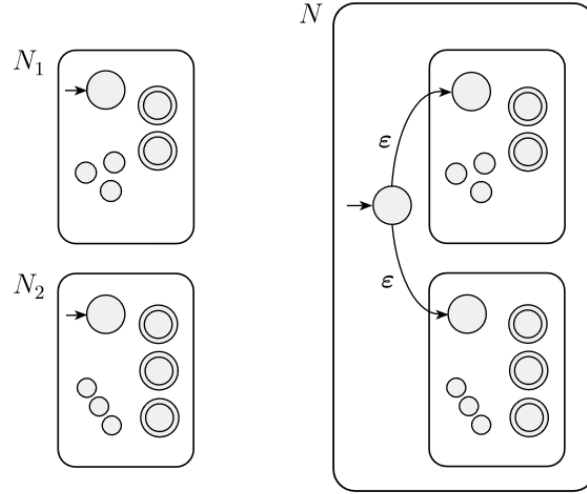
$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{se } q \in Q_1 \\ \delta_2(q, a) & \text{se } q \in Q_2 \\ \{q_1, q_2\} & \text{se } q = q_0 \wedge a = \varepsilon \\ \emptyset & \text{se } q = q_0 \wedge a \neq \varepsilon \end{cases}$$

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

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

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

□



Interpretazione grafica della dimostrazione precedente

Teorema 3: Chiusura dell'intersezione in REG

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

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

Dimostrazione.

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

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

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

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

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

□

Teorema 4: Chiusura del complemento in REG

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

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

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

$$x \in L \iff x \notin L(D)$$

dunque che $\neg L = L(D') \in \text{REG}$

□

Teorema 5: Chiusura della concatenazione in REG

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

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

Dimostrazione.

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

- $q_0 = q_1$
- $Q = Q_1 \cup Q_2$
- $F = F_2$
- $\forall q \in Q, a \in \Sigma$ si ha che:

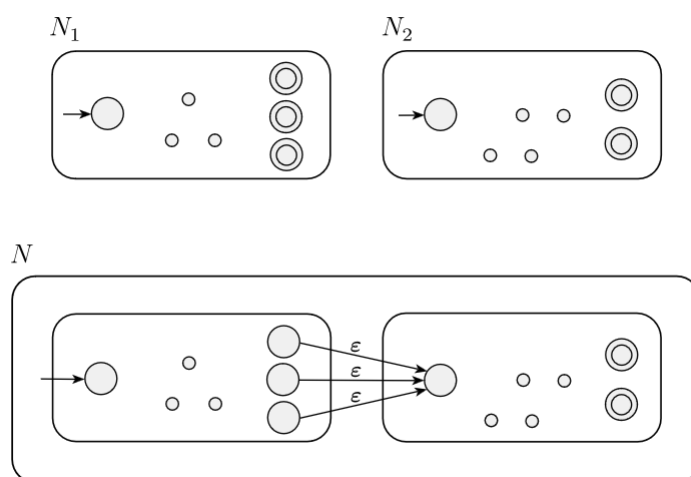
$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{se } q \in Q_1 - F_1 \\ \delta_1(q, a) & \text{se } q \in F_1 \wedge a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_2\} & \text{se } q \in F_1 \wedge a = \varepsilon \\ \delta_2(q, a) & \text{se } q \in Q_2 \end{cases}$$

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

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

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

□



Interpretazione grafica della dimostrazione precedente

Corollario 2: Chiusura della potenza in REG

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

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

Dimostrazione.

Caso base.

- Dato $n = 0$, si ha che $L^0 = \{\epsilon\} \in \text{REG}$

Ipotesi induttiva.

- Dato $n \in \mathbb{N}$, assumiamo che $L^n \in \text{REG}$

Passo induttivo.

- Tramite la [Chiusura della concatenazione in REG](#) otteniamo che

$$L^{n+1} = L \circ L^n \in \text{REG}$$

□

Teorema 6: Chiusura di star in REG

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

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

Dimostrazione I.

Caso base.

- Dato $n = 0$, si ha che $\bigcup_{n \geq 0} L^n = L^0 \in \text{REG}$

Ipotesi induttiva.

- Dato $n \in \mathbb{N}$, assumiamo che $\bigcup_{n \geq 0} L^n \in \text{REG}$

Passo induttivo.

- Tramite la [Chiusura dell'unione in REG](#) e la [Chiusura della potenza in REG](#) otteniamo che:

$$\bigcup_{n \geq 0} L^{n+1} = L^{n+1} \cup \left(\bigcup_{n \geq 0} L^n \right) \in \text{REG}$$

□

Dimostrazione II.

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

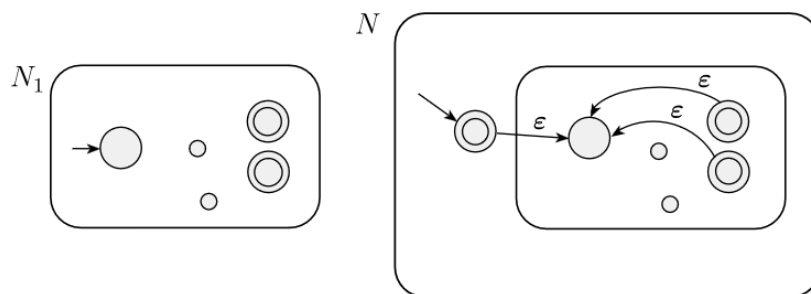
$$\delta'(q, a) = \begin{cases} \delta(q, a) & \text{se } q \in Q - F \\ \delta(q, a) & \text{se } q \in F \wedge a \neq \varepsilon \\ \delta(q, a) \cup \{q_{0*}\} & \text{se } q \in F \wedge a = \varepsilon \\ \{q_{0*}\} & \text{se } q = q_{0*} \wedge a = \varepsilon \\ \emptyset & \text{se } q = q_{0*}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

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

$$x \in L^* \iff N(x) \in F'$$

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

□



Interpretazione grafica della dimostrazione precedente

Corollario 3: Chiusura di plus in REG

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

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

Dimostrazione I.

- Analoga a quella dell'operatore star, utilizzando $n = 1$ come caso base

□

Dimostrazione II.

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

□

Teorema 7: Leggi di De Morgan

Dati $L_1, L_2 \in \text{REG}$, si ha che:

$$L_1 \cup L_2 = \neg(\neg L_1 \cap \neg L_2)$$

$$L_1 \cap L_2 = \neg(\neg L_1 \cup \neg L_2)$$

(dimostrazione omessa)

1.5 Espressioni regolari

Definizione 18: Espressione regolare

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

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

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

Osservazione 4

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

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

Esempi:

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

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

Definizione 19: 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:

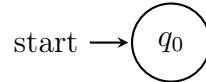
$$\mathcal{L}(\text{re}) \subseteq \mathcal{L}(\text{NFA})$$

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

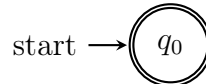
Caso base.

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



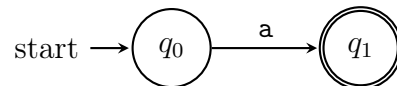
per cui si ha che $x \in L(R) \iff x \in L(N_\emptyset)$ dunque $L(R) = L(N_\emptyset) \in \mathcal{L}(\text{NFA})$

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



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

Ipotesi induttiva.

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

Passo induttivo.

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

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

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

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

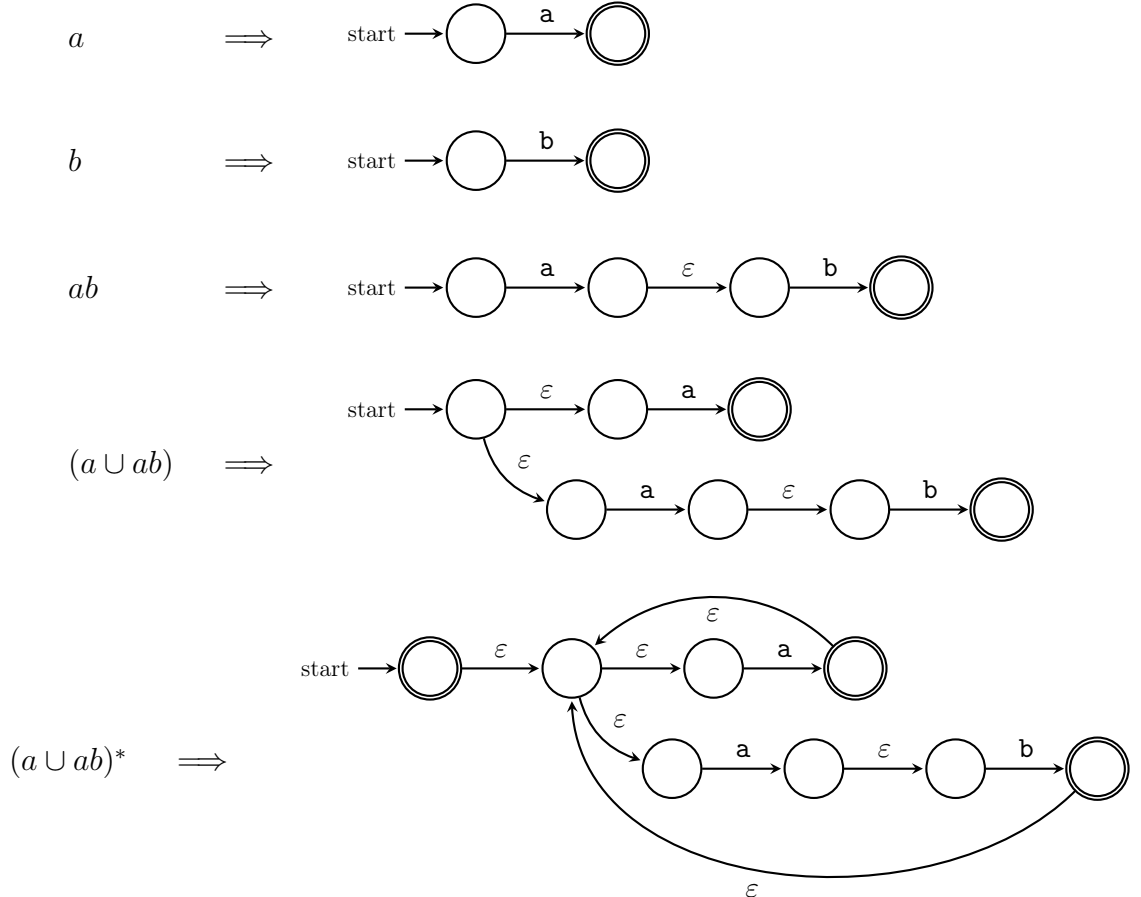
- Se $R = R_1^*$, tramite la **Chiusura di plus in REG**, otteniamo che:

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

□

Esempio:

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



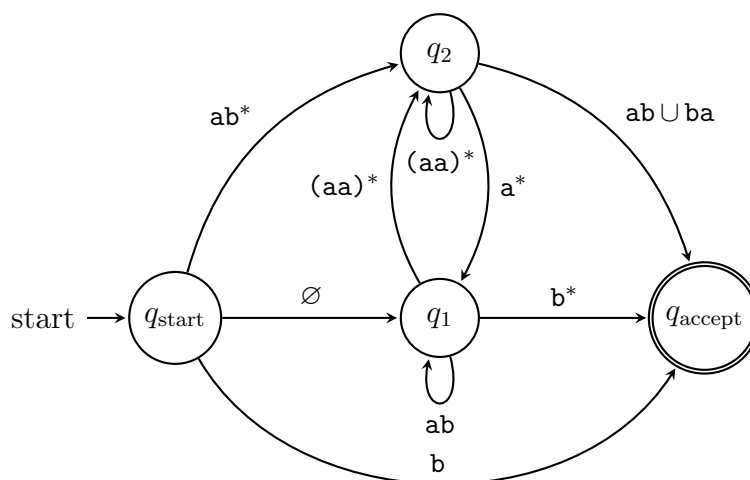
1.5.1 NFA generalizzati

Definizione 20: Generalized NFA (GNFA)

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

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

Esempio:



Osservazione 5

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 5: Stringa accettata in un GNFA

Sia $G := (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ un GNFA. Data una stringa $x := x_0 \dots x_k \in \Sigma_\varepsilon^*$, dove $x_0, \dots, x_k \in \Sigma_\varepsilon^*$ (ossia sono delle sottostringhe), diciamo che x è **accettata da G** se esiste una sequenza di stati $r_0, r_1, \dots, r_{k+1} \in Q$ tali che:

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

Esempio:

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

Corollario 4

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

Definizione 21: 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:

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

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

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

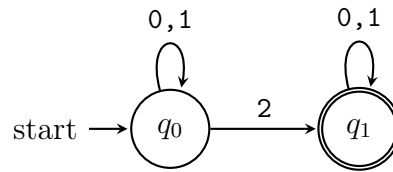
implicando dunque che $L(D) \in \mathcal{L}(\text{DFA})$ e di conseguenza che:

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

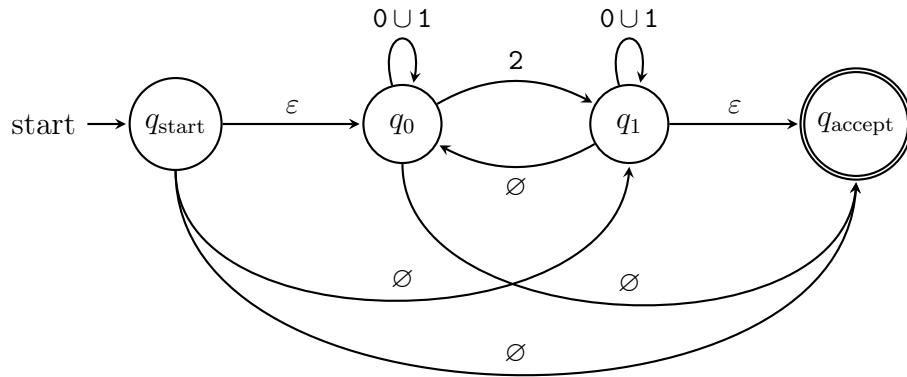
□

Esempio:

- Consideriamo il seguente DFA:



- Il suo GNFA equivalente corrisponde a:



Algoritmo 1: Riduzione minimale di un GNFA

Dato un GNFA $G = (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ in input, il seguente algoritmo restituisce un GNFA G' avente due soli stati ed una sola transizione:

```

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

```

Dimostrazione.

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

Caso base.

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

Ipotesi induttiva.

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

Passo induttivo.

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

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

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

- Data una stringa $x = x_0 \dots x_m \in L(G_k)$, dove $x_0, \dots, x_m \in \Sigma_\epsilon^*$, esiste una sequenza di stati $q_0, \dots, q_m \in Q$ tali che:

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

- 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 $x_i \in L(\delta(q_i, q_j)) \implies x \in L(\delta'(q_i, q_j))$, 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 $x \in L(\delta'(q_i, q_j))$
- 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 $x \in L(G_{k+1}) \implies x \in L(G_k)$
- Esprimendo il tutto graficamente, risulta evidente che le seguenti transizioni siano del tutto equivalenti:

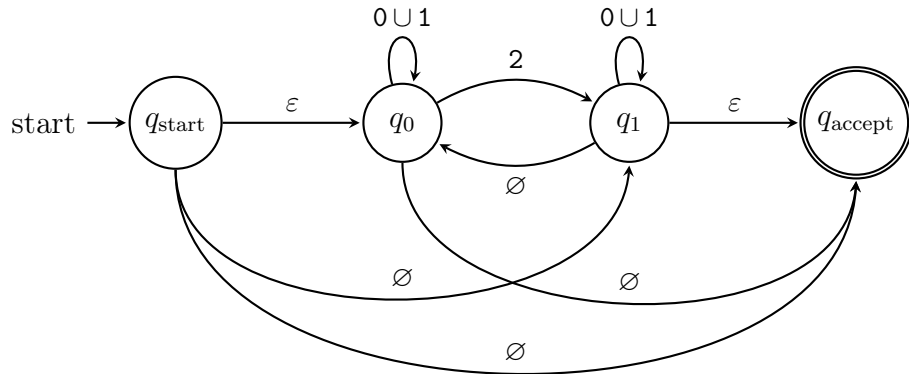


- Di conseguenza, otteniamo che $x \in L(G_k) \iff x \in L(G_{k+1})$, concludendo quindi, per ipotesi induttiva, che $L(G) = L(G_k) = L(G_{k+1})$

□

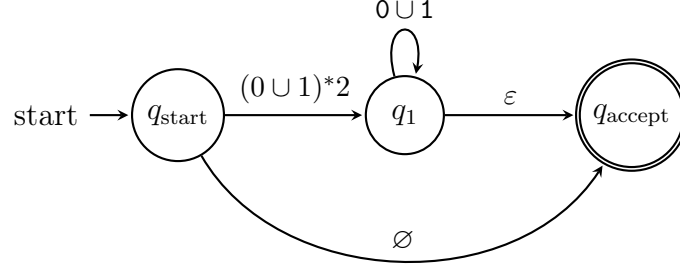
Esempio:

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



- Rimuoviamo quindi lo stato q_0 calcolando le nuove transizioni:

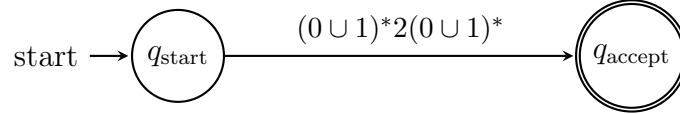
$$\begin{aligned}\delta'(q_{\text{start}}, q_1) &= \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0, q_1) \cup \delta(q_{\text{start}}, q_1) = \varepsilon(0 \cup 1)^*2 \cup \emptyset = (0 \cup 1)^*2 \\ \delta'(q_{\text{start}}, q_{\text{accept}}) &= \delta(q_{\text{start}}, q_0)\delta(q_0, q_0)^*\delta(q_0,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



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

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

- Il GNFA minimale, dunque, corrisponde a:



Corollario 5: 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

Data la classe $\mathcal{L}(\text{re})$, 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:

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

□

Corollario 6: Classi dei linguaggi regolari

Dato un alfabeto Σ , si ha che:

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

1.6 Linguaggi non 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.

Definizione 22: Linguaggio non regolare

Dato un alfabeto Σ , definiamo un linguaggio L di Σ come **non regolare** se $L \notin \text{REG}$, dunque se non è possibile definire un automa a stati finiti che lo riconosce o un'espressione regolare che lo descrive

1.6.1 Pumping lemma per i linguaggi regolari**Definizione 23: Lunghezza di una stringa**

Dato un linguaggio L e una stringa $s \in L$, indichiamo con $|s|$ la sua **lunghezza**, ossia la quantità di simboli al suo interno

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 $\forall s := xyz \in L$, con $|s| \geq p$ e $x, y, z \in L$ (ossia sono sue sottostringhe), si ha che:

- $\forall i \in \mathbb{N} \quad xy^iz \in L$, 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 s

Dimostrazione.

- Poiché $L \in \text{REG}$, sia $D := (Q, \Sigma, \delta, q_0, F)$ il DFA tale che $L = L(D)$
- Consideriamo quindi $p := |Q|$. Data la stringa $s := s_1 \dots s_n \in L$ dove $s_1, \dots, s_n \in \Sigma$ e dove $n \geq p$, consideriamo la sequenza di stati r_1, \dots, r_{n+1} tramite cui s viene accettata da D :

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

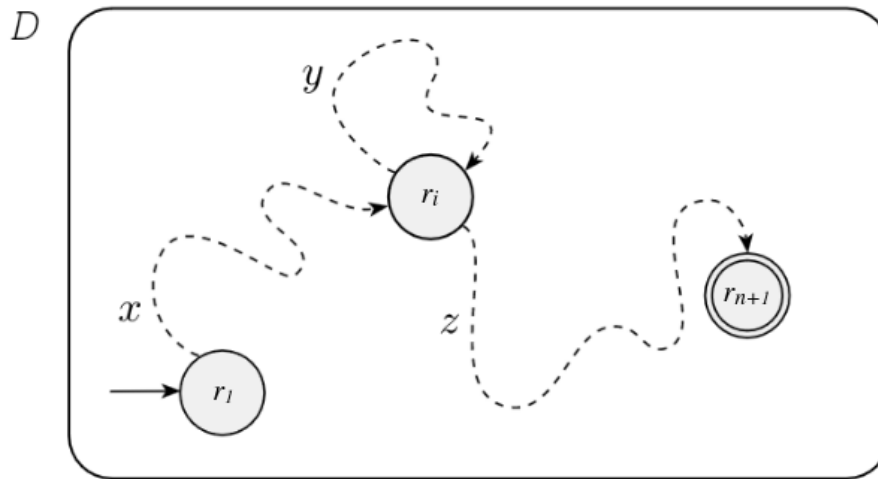
- Notiamo quindi che $|r_1, \dots, r_{n+1}| = n + 1$, ossia che il numero di stati attraversati sia $n + 1$. Inoltre, in quanto $n \geq p$, ne segue automaticamente che $n + 1 \geq p + 1$. Tuttavia, poiché $p := |Q|$ e $n + 1 \geq p + 1$, ne segue necessariamente che $\exists i, j \in [1, n + 1] \mid i < j \leq p + 1 \wedge r_i = r_j$, ossia che tra i primi $p + 1$ stati della sequenza vi sia almeno uno stato ripetuto
- A questo punto, consideriamo le seguenti sottostringhe di s :
 - $x = s_1 \dots s_{i-1}$, tramite cui si ha che $\delta^*(r_1, x) = r_i$
 - $y = s_i \dots s_{j-1}$, tramite cui si ha che $\delta^*(r_i, y) = r_j = r_i$
 - $z = s_j \dots s_n$, 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

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

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

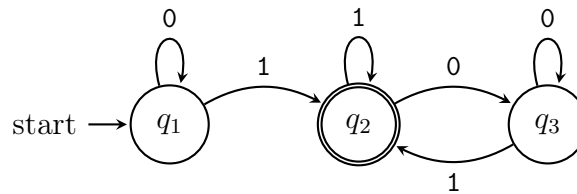
□



Interpretazione grafica della dimostrazione precedente

Esempio:

- Consideriamo il seguente 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 $s := 0100010101 \in L$, è possibile separare s 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 6: Dimostrazione di non regolarità

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

Esempio:

- Consideriamo il seguente 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 $s := 0^p 1^p \in L$. Poiché $|s| \geq p$, possiamo suddividerla in tre sottostringhe $x, y, z \in L$ tali che $s = xyz$, per poi procedere con uno dei due seguenti approcci:

1. Approccio enumerativo:

- Se y è composta da soli 0, allora ogni stringa generata dal pumping non sarà in L in quanto il numero di 0 sarà superiore al numero di 1
- Se y è composta da soli 1, allora ogni stringa generata dal pumping non sarà in L in quanto il numero di 1 sarà superiore al numero di 0
- Se y è composta sia da 0 che da 1, allora ogni stringa generata dal pumping non sarà in L in quanto esse assumeranno la forma $0000 \dots 101010 \dots 1111$
- Di conseguenza, poiché in ogni caso viene contraddetto il pumping lemma, ne segue necessariamente che L non sia regolare

2. Approccio condizionale:

- Poiché la terza condizione del pumping lemma impone che $|xy| \leq p$ e poiché $s := 0^p 1^p$, ne segue che $xy = 0^m$ e $z = 0^{p-m} 1^p$, dove $m \in [1, p]$
- Inoltre, per la seconda condizione, si ha che $|y| > 0$, dunque necessariamente si ha che $x = 0^{m-k}$ e $y = 0^k$, dove $k \in [1, m]$
- A questo punto, consideriamo la stringa $xy^0 z$. Notiamo immediatamente che

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

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

- Dunque, ne segue necessariamente che L non sia regolare

1.7 Esercizi svolti

Problema 1: Linguaggio rovesciato

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

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

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

Dimostrazione.

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

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

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

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

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

ossia avente tutti gli archi invertiti rispetto ad N

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

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

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

□

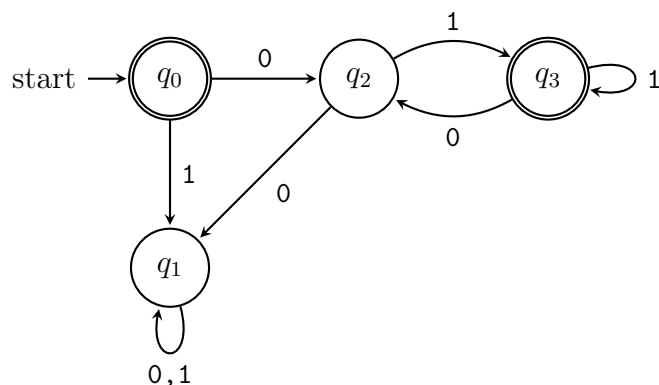
Problema 2: Complemento di un'espressione regolare

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

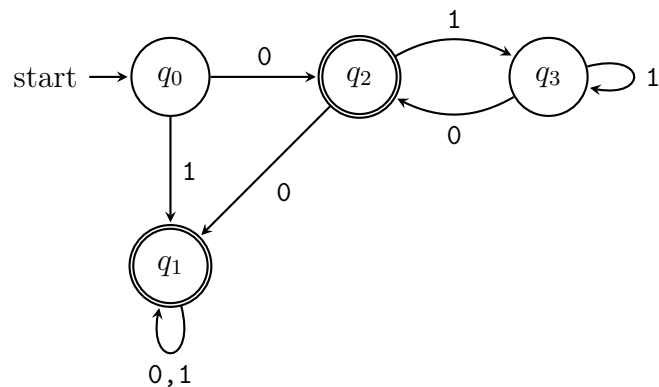
$$L(D) = \{w \in \{0, 1\}^* \mid w \notin L(R)\}$$

Soluzione:

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



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



Problema 3

Data una stringa $x \in \{0, 1\}^*$, definiamo $|x|_0$ come il numero di 0 presenti in x e $|x|_1$ come il numero di 1 presenti in x .

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

Dimostrazione.

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

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

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

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

- Dunque, ne segue necessariamente che L non sia regolare

□

Problema 4

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

Dimostrazione.

- Supponiamo per assurdo che L sia regolare, implicando che per esso debba valere il pumping lemma, dove p è la lunghezza del pumping
- Consideriamo quindi la stringa $s := 1^{p^2} \in L$. Poiché $|s| \geq p$, possiamo suddividerla in tre sottostringhe $x, y, z \in L$ tali che $s = xyz$
- Poiché la terza condizione del lemma impone che $|xy| \leq p$ e poiché $s := 1^{p^2}$, 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

□

Grammatiche e Linguaggi acontestuali

2.1 Grammatiche acontestuali

Definizione 24: Context-free Grammar (CFG)

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

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

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

Osservazione 7: Acontestualità

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

Esempio:

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

Definizione 25: Produzione

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

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

Esempio:

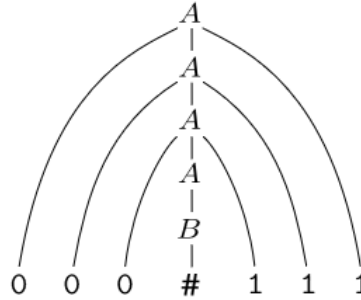
- Consideriamo la grammatica $G = (\{A, B\}, \{0, 1, \#\}, R, A)$ dove:

$$R = \{A \rightarrow 0A1, A \rightarrow B, B \rightarrow \#\}$$

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

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

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

**Definizione 26: Derivazione**

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

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

Definizione 27: Linguaggio di una CFG

Sia $G = (V, \Sigma, R, S)$ una CFG. Definiamo come **linguaggio di G** , indicato come $L(G)$, l'insieme di stringhe derivate dalle regole di G tramite la variabile S :

$$L(G) = \{x \in \Sigma^* \mid S \xRightarrow{*} x\}$$