Automi, Calcolabilità e Complessità

Alessio Marini, 2122855

Appunti presi durante il corso di **Automi, Calcolabilità e Complessità** nell'anno **2025/2026** del professore Daniele Venturi.
Gli appunti li scrivo principalmente per rendere il corso più comprensibile **a me** e anche per imparare il linguaggio Typst. Se li usate per studiare verificate sempre le informazioni 🙏🏼 .

Contatti:
alem1105
marini.2122855@studenti.uniroma1.it

September 27, 2025

Indice

1. Introduzione alla terminologia

Introduciamo delle definizioni e delle operazioni che utilizzeremo durante il corso.

Alfabeto

É un insieme finito di simboli, quindi ad esempio $\Sigma = \{0, 1, x, y, z\}$.

Stringa

Una stringa è una sequenza di simboli che appertongono ad un alfabeto. Quindi, ad esempio, dato l'alfabeto $\Sigma = \{0, 1, x, y, z\}$ una sua stringa è $w = 01z$.

1.1. Operazioni sulle Stringhe

Lunghezza di una Stringa

Data una stringa $w \in \Sigma^*$ indichiamo la lunghezza con $|w|$ ed è definita come il numero di simboli che contiene.

Concatenazione

Data la stringa $x = x_1, ..., x_n \in \Sigma^*$ e la stringa $y = y_1, ..., y_m \in \Sigma^*$ definiamo come **concatenazione di x con y** la stringa $x \cdot y = x_1 ..., x_n y_1 ... y_m$.

Stringa Vuota

Durante il corso indicheremo con ε la stringa vuota, ovvero una stringa tale che $|\varepsilon| = 0$ Se concateniamo una qualsiasi stringa non vuota con una stringa vuota otteniamo la prima stringa:

$$\forall w \in \Sigma^* \ w \cdot \varepsilon = w$$

Conteggio

Data una stringa $w \in \Sigma^*$ e un simbolo $a \in \Sigma$ indichiamo il contaggio di a in w con $|w|_a$ e lo definiamo come il numero di occorrenze del carattere a nella stringa w.

Stringa Rovesciata

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

Potenza

Data la stringa $w \in \Sigma^*$ e dato $n \in \mathbb{N}$ definiamo la potenza in modo ricorsivo:

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

Linguaggio

Dato un alfabeto Σ definiamo Σ^* come linguaggio di Σ, ovvero l'insieme di tutte le stringhe di quell'alfabeto.

2. DFA - Automa a Stati Finiti

Il modello di computazione che utilizzeremo per ora è un DFA, questo ha una memoria limitata e permette una gestione dell'input. La memoria gli permette di memorizzare i suoi stati e tramite gli input decide in quale stato futuro muoversi.

Esempio - Una porta automatica

Una porta automatica avrà due stati:
- Aperta
- Chiusa

E due input:
- Rileva qualcuno
- Non rileva nessuno

Quindi lo stato iniziale sarà la porta chiusa, se rileva qualcuno va nello stato di aperta mentre se non rileva nessuno rimane chiusa.

DFA

Definiamo un DFA come una tupla $(Q, \Sigma, \delta, q_0, F)$ dove:
- Q è l'insieme degli stati
- Σ è l'insieme finito dei simboli in input
- $\delta : Q \times \Sigma \to Q$ è la funzione di transizione degli stati, ovvero dato lo stato in cui si trova ed un input, restituisce lo stato in cui andremo
- $q_0 \in Q$ è lo stato iniziale dell'automa
- $F \subseteq Q$ è l'insieme degli stati di accettazione dell'automa, ovvero gli stati dove l'automa si trova dopo aver riconosciuto determinate stringhe e consente la terminazione.

Dato DFA M possiamo definire l'insieme delle stringhe riconosciute dall'automa, ovvero quelle che lo portano in uno stato di accettazione come $L(M)$. Da notare che può anche accadere che $L(M) = \emptyset$. Daremo una definizione più formale di quest'ultimo più avanti.

Dati dei DFA vogliamo iniziare a definire dei linguaggi dedicati a questi, per farlo abbiamo bisogno della **funzione di transizione estesa**.

Funzione di Transizione Estesa

La definiamo come:

$$\delta^* = Q \times \Sigma^* \to Q$$

Quindi questa a differenza di quella classica non usa degli input singoli ma delle intere stringhe appartenenti al **linguaggio** del DFA.
È definibile in modo ricorsivo:

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

Quindi data una stringa, partiamo dal primo carattere a sinistra e andiamo avanti utilizzando la funzione di transizione fino ad arrivare ad una stringa vuota.

Adesso diamo le definizioni di **Configurazione** e **Passo di Computazione** che ci serviranno a definire più formalmente un **Linguaggio Accettato** del DFA.

Configurazione

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

Passo di Configurazione

Indica il passaggio da una configurazione ad un'altra rispettando la funzione di transizione δ, il passaggio lo indichiamo con il simbolo $\vdash_M$ dove M indica il DFA. Possiamo dire quindi che esiste una relazione binaria fra un passo di configurazione e la funzione di transizione:

$$(p, ax) \vdash_M (q, x) \Leftrightarrow \delta(p, a) = q$$

Dove $p, q \in Q$ - $a \in \Sigma$ e $x \in \Sigma^*$. Un passaggio di configurazione può avvenire, quindi, soltanto se la funzione di transizione lo permette.

Possiamo estendere questa relazione con il simbolo $\vdash_M^*$ considerando anche la **chiusura riflessiva e transitiva**:

- **Riflessività:** $(q, x) \vdash_M^* (q, x)$
- **Transitività:** Se $(q, aby) \vdash_M (p, by) \wedge (p, by) \vdash_M (r, y) \Rightarrow (q, aby) \vdash_M^* (r, y)$
  - Dove $q, p, r \in Q$ - $a, b \in \Sigma$ ed $y \in \Sigma^*$

Definiamo quindi il linguaggio accettato dal DFA.

Linguaggio Accettato

Diciamo che $x \in \Sigma^*$ è accettato da un automa $M = (Q, \Sigma, \delta, q_0, F)$ se $\delta^*(q_0, x) \in F$ oppure usando la relazione del passaggio, se $(q_0, x) \vdash_M^* (q, \varepsilon)$ con $q \in F$.

3. Linguaggi Regolari

Definizione

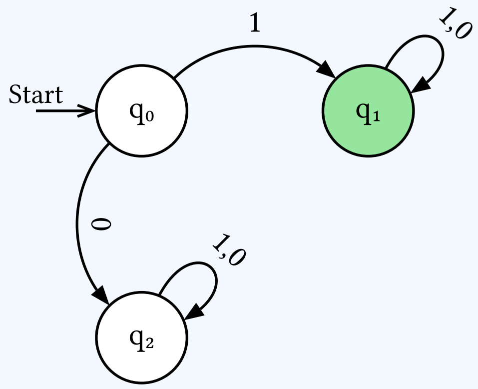
$$\text{REG} = \{L \subseteq \Sigma^* : \exists \;\; \text{DFA M t.c.} \;\; L(M) = L\}$$

Quindi i linguaggi regolari sono tutti quei linguaggi che sono accettati da almeno un DFA.

Uno dei nostri obiettivi nel corso è quello di, dato un linguaggio, progettare dei DFA adatti.

Esempio

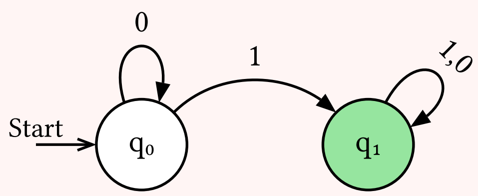
Dato il linguaggio $L = \{x \in \{0,1\}^* \;\; \text{t.c.} \;\; x = 1y, y \in \{0,1\}^*\}$, un possibile DFA potrebbe essere:



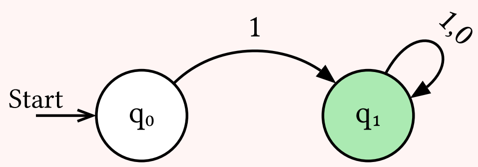
Questo DFA accetta quindi tutte le stringhe che iniziano con il simbolo 1 mentre rifiuta tutte quelle che iniziano con il simbolo 0.

Attenzione - Stato Pozzo

Notiamo che è presente lo stato q_2 dal quale il DFA non esce più una volta entrato, questo è necessario perchè se omesso il DFA accetterebbe tutte le stringhe:



Infatti in questo modo se in q_0 riceve 0 rimane su stesso ma poi continua ad attendere input.
Il modo corretto per lasciare lo stesso significato del primo DFA ma omettere lo stato q_2 è quello di omettere anche il comportamento di q_0 in caso riceviamo 0, ovvero:



Adesso dobbiamo dimostrare formalmente che questo DFA accetta il linguaggio fornito, dobbiamo quindi dimostrare che:

$$\text{DFA accetta } \; x \Leftrightarrow x \in L$$

Innanzitutto facciamo due osservazioni, ovvero che se il DFA si trova in q_1 o q_2 allora non cambierà mai più stato:
- $\delta^*(q_1, u) = q_1 \quad \forall u \in \{0,1\}^*$
- $\delta^*(q_2, u) = q_2 \quad \forall u \in \{0,1\}^*$

Dimostriamo per induzione che il linguaggio è accettato, quindi presa una stringa dobbiamo far vedere che se inizia con 1 terminiamo in q_1 altrimenti in q_2.

Dimostrazione

Caso Base

Come caso base prendiamo una stringa vuota, quindi $|x| = 0$ ovvero $x = \varepsilon$, abbiamo che:

$$\delta^*(q_0, \varepsilon) = \delta(q_0, \varepsilon) = q_0 \notin F$$

Infatti se abbiamo una stringa vuota il DFA non fa nulla e rimane in q_0

Passo Induttivo

Adesso dobbiamo prendere una stringa w tale che $|w| \leq n$ con $n > 0$, la funzione di transizione avrà quindi 3 risultati possibili:

$$\delta^*(q_0, w) = \begin{cases} q_0 & \text{se} \quad w = \varepsilon \\ q_1 & \text{se} \quad w \quad \text{inizia con} \quad 1 \\ q_2 & \text{se} \quad w \quad \text{inizia con} \quad 0 \end{cases}$$

Prendiamo quindi una stringa x tale che $|x| = n + 1$ e la costruiamo come $x = au$ con $a \in \{0,1\}$ e $u \in \{0,1\}^*$, la funzione di transizione ci restituirà:

$$\delta^*(q_0, x) = \delta^*(q_0, au) = \delta^*\left(\underbrace{\delta(q_0, a)}_{\text{ha 2 soluzioni}}, u\right)$$

Le due soluzione del passaggio evidenziato sono:
- $\delta(q_0, a) = q_1$ se $a = 1$
- $\delta(q_0, a) = q_2$ se $a = 0$

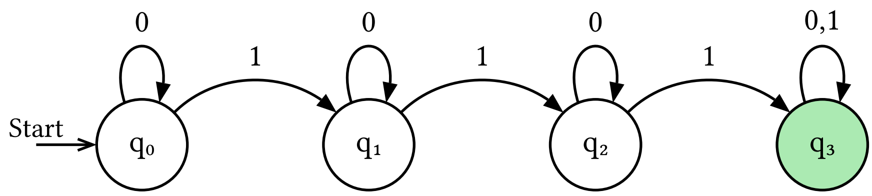
Quindi il DFA andrà sicuramante in uno dei due stati q_1 o q_2 e da lí non si muoverà più, per il ragionamento fatto all'inizio della dimostrazione.

Esercizi

DFA 1

Dato il linguaggio $L = \{x : x \in \{0,1\}^* \land W_H(x) \geq 3\}$ con $W_H(x) = \#1$ ovvero il numero di 1 presenti nella stringa. Progettare un automa che accetta il linguaggio e dimostrarlo.

Un possibile automa potrebbe essere:

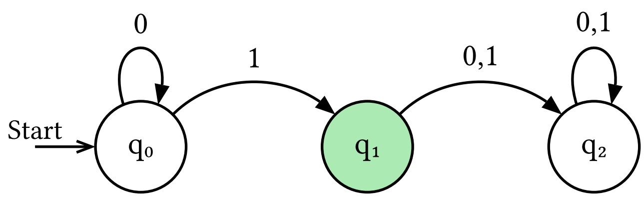


Dimostrazione: Copiare da iPad

DFA 2

Dato il linguaggio $L = \{x : x = 0^n 1 \ \text{ con } \ n \in \mathbb{N}\}$ progettare un automa che accetta il linguaggio e dimostrarlo.

Un possibile automa potrebbe essere:



Dimostrazione: Copiare da iPad

3.1. Operazioni sui Linguaggi

Definiamo adesso delle operazioni sui linguaggi che ci torneranno utili.

Unione

$$L_1 \cup L_2 = \{x \in \Sigma^* : x \in L_1 \lor x \in L_2\}$$

Intersezione

$$L_1 \cap L_2 = \{x \in \Sigma^* : x \in L_1 \land x \in L_2\}$$

Complemento

$$\overline{L} = \{x \in \Sigma^* : x \notin L\}$$

Concatenazione

$$L_1 \circ L_2 = \{xy : x \in L_1 \land y \in L_2\}$$

Da notare che questa operazione non è commutativa quindi $L_1 \circ L_2 \neq L_2 \circ L_1$

Potenza

Possiamo definirla ricorsivamente:

$$\begin{cases} L^0 = \{\varepsilon\} \\ L^{n+1} = L^N \circ L \end{cases}$$

Operatore * «star»

$$L^* = \bigcup_{n \geq 0} L^n = \{\varepsilon\} \cup L^1 \cup L^2 \cup \dots$$

3.2. Introduzione alla proprietà di chiusura dei Linguaggi Naturali

Vogliamo capire se dati due linguaggi naturali $L_1, L_2 \in \text{REG}$ il linguaggio risultante di operazioni effettuate con questi linguaggi è naturale o no, ad esempio se $L_1 \cup L_2 \in \text{REG}$ oppure se $L_1 \cap L_2 \in \text{REG}$.

Vedremo qualche dimostrazione ma in realtà sarà più semplice dimostrare tutte le chiusure utilizzando gli NFA, ovvero gli automi non deterministici.

3.2.1. Chiusura per Unione

Teorema - Chiusura per Unione

Come prima idea possiamo dire che:

$$L_1, L_2 \in \text{REG} \Rightarrow \exists M_1, M_2 \in \text{DFA t.c. } \; L(M_1) = L_1 \land L(M_2) = L_2$$

Quindi dati due linguaggi naturali esistono due automi che li hanno come linguaggi accettati. Noi dobbiamo definire un terzo automa M tale che $L(M) = L_1 \cup L_2$, ma data una stringa x candidata non possiamo provare a vedere prima cosa succede su M_1 e se non la accetta provare M_2 perchè perderemmo la sequenza corretta della stringa su M.

Quello che dobbiamo fare è testare ogni carattere di x in parallelo su M_1 e M_2 e in base al risultato aggiorniamo lo stato di M.

Input Dimostrazione

Vogliamo mostrare che dati

- $M_1 = (Q_1, \Sigma, \delta_1, q_o^1, F_1)$
- $M_2 = (Q_2, \Sigma, \delta_2, q_0^2, F_2)$

Assumiamo lo stesso Σ per semplicità

Tali che: $L(M_1) = L_1 \wedge L(M_2) = L_2$

Costruiamo un terzo DFA $M(Q, \Sigma, \delta, q_0, F)$ t.c. $L(M) = L_1 \cup L_2$

Avremo che:

- $Q = \{(r_1, r_2) : r_1 \in Q_1, r_2 \in Q_2\} = Q_1 \times Q_2$ (Tutte le coppie di stati possibili)
- $\delta : Q \times \Sigma \to Q$
  - $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $F = \{(r_1, r_2) : r_1 \in F_1 \vee r_2 \in F_2\} = \underbrace{(F_1 \times Q_2)}_{\text{il primo stato è accettato}} \cup \underbrace{(F_2 \times Q_1)}_{\text{il secondo stato è accettato}}$

Infatti basta che soltanto uno dei due stati della coppia venga accettato per accettare la coppia.

Da notare che per l'intersezione abbiamo una situazione molto simile, infatti avremo che:

$$F = \{(r_1, r_2) : r_1 \in F_1 \wedge r_2 \in F_2\} = F_1 \times F_2$$

Dimostrazione

Vogliamo mostrare che dato

$$\delta^*(q_0, x) = \delta^*((q_0^1, q_0^2), x)$$

Si ha che $\forall x \in \Sigma^*$

$$= (\delta_1^*(q_0^1, x), \delta_2^*(q_0^2, x))$$

TODO: MANCA UNA PARTE DI DIMOSTRAZIONE (Mostrare che funziona per n e 0)

$$\forall x \in \Sigma^*, x \in L(M) \Leftrightarrow x \in L_1 \cup L_2$$
$$\Rightarrow x \in L(M) \Rightarrow \delta^*(q_0, x) \in F = F_1 \times Q_2 \cup F_2 \times Q_1 = (p, q)$$
$$\text{Dove} \quad p = \delta_1^*(q_0^1, x), q = \delta_2^*(q_0^2, x))$$

Questo significa che

- Se $x \in L_1$ allora $\delta^*(q_0^1, x) \in F_1$ e quindi:

$$\delta^*(q_0, x) = (\delta_1^*(q_0^1, x), \delta_2^*(q_0^2, x)) \in F_1 \times Q_2 \Rightarrow M \quad \text{accetta} \quad x$$

- Se $x \in L_2$ allora $\delta^*(q_0^2, x) \in F_2$ e quindi:

$$\delta^*(q_0, x) = (\delta_1^*(q_0^1, x), \delta_2^*(q_0^2, x)) \in F_2 \times Q_1 \Rightarrow M \quad \text{accetta} \quad x$$

Spiegato a parole, abbiamo che la funzione di transizione dell'automa M equivale ad eseguire lo stesso input sui due automi M_1, M_2. Presa una stringa del linguaggio questa è accettata dall'automa se e solo se appartiene all'unione dei due linguaggi di $M_1 \quad \text{e} \quad M_2$.

Partendo dalla sinistra dell'implicazione abbiamo che $x \in L(M)$ quindi la stringa è accettata e allora la funzione di transizione estesa ci porta in uno stato appartenente ad F. Ricordiamo che lo stato in cui ci troviamo è in realtà una coppia di stati uno dei quali deve essere accettato o da M_1 o da M_2 e questo appunto significa rispettivamente che o $x \in L(M_1)$ oppure $x \in L(M_2)$.

Resto delle dimostrazioni

Per dimostrare il resto delle proprietà introduciamo il concetto di non determinismo.

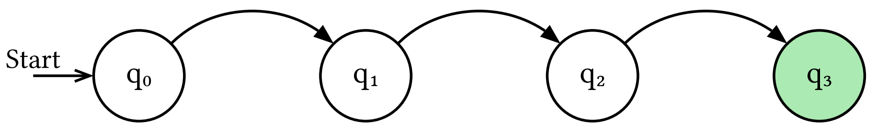
4. Non Determinismo

Per adesso abbiamo visto soltanto automi deterministici, questo significa che trovandoci in uno stato e ricevendo un input possiamo soltanto andare in un altro stato o rimanere fermi, ma in generale un solo movimento.

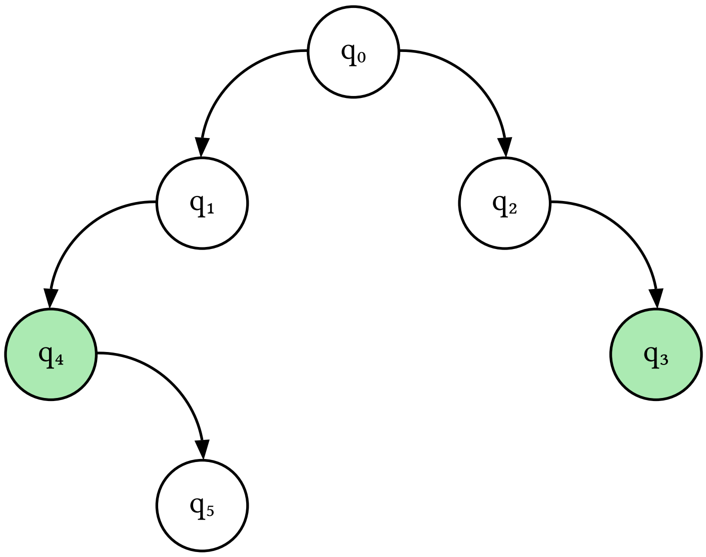
Nel **non determinismo** invece:
- Quando l'automa è in $q \in Q$ e legge $a \in \Sigma$ può andare in diversi stati
- Sono ammessi gli «ε-archi» ovvero l'automa può muoversi senza leggere input. Dallo stesso stato possono partire più «ε-archi».
- Accettazione: Se e solo se esiste un ramo che accetta, vedremo più avanti che quando studiamo un NFA avremo un albero con vari rami, se un ramo accetta allora consideriamo la stringa come accettata per il NFA.

Nel non determinismo quindi abbiamo un input che si dirama in vari stati invece che seguire un cammino di *uno stato alla volta.*

- **Determinismo**



- **Non Determinismo**



Definizione - NFA

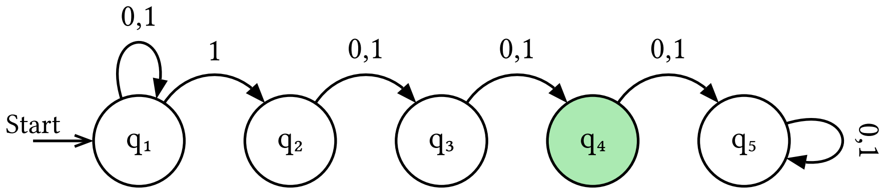
Un NFA è $(Q, \Sigma, \delta, q_0, F)$ dove Q, Σ, q_0, F sono come nei DFA ma:

$$\delta : Q \times \Sigma_\varepsilon \to \mathbb{P}(Q)$$
$$\text{Dove} \quad \Sigma_\varepsilon \cup \{\varepsilon\}$$

e $\mathbb{P}$ è l'insieme delle parti.

Vediamo un esempio e capiamo come ci si muove al loro interno.

Esempio

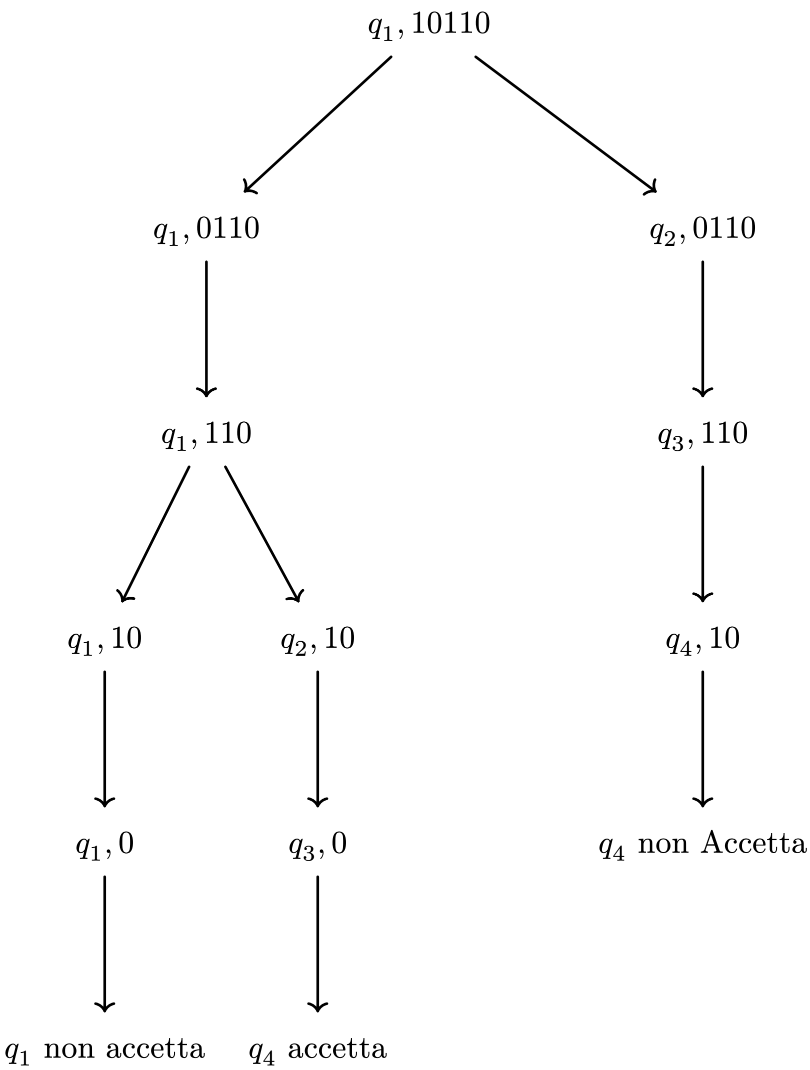


Che ha come linguaggio:

$$L = \{x : x \in \{0,1\}^* \text{ che hanno un '1' in terzultima posizione}\}$$

Da notare che lo stato q_5 possiamo anche ometterlo, ma non dobbiamo indicare in q_4 nessun arco.

Per muoverci, ad esempio nel NFA dell'esempio sopra, ci torna utile disegnare un albero con tutte i cammini che stiamo intrapendendo. Se ad esempio riceviamo in input la stringa «10110»:



4.1. Configurazione negli NFA

Possiamo estendere il concetto di **configurazione** anche per gli NFA.

Dato un NFA N indichiamo come configurazione una coppia $(q, x) \in Q \times \Sigma_\varepsilon^*$ e avremo un passo di configurazione come:

$$(p, ax) \vdash_N (q, x) \Leftrightarrow q \in \delta(p, a)$$

Con:
- $x \in \Sigma_\varepsilon^*$
- $a \in \Sigma_\varepsilon$
- $p, q \in Q$

Quindi il risultato di una transizione deve far parte dell'insieme delle parti degli stati:

$$\delta(p, a) \in \mathbb{P}(Q)$$

Quando, l'automa N, accetta $w \in \Sigma_\varepsilon^*$?
- Se e solo se $\exists q \in F \quad \text{t.c.} \quad (q_0, w) \vdash_N^* (q, \varepsilon)$. Dove $\vdash_N^*$ è la relazione estesa.

4.2. Equivalenza tra NFA e DFA

Prendiamo le due classi:
- $\mathcal{L}(\text{DFA}) \subseteq \text{REG}$
- $\mathcal{L}(\text{NFA}) = \{L : \exists \text{ NFA N t.c. } \mathcal{L}(N) = L\}$

Teorema - Per ogni automa finito non deterministico esiste un automa finito deterministico equivalente.

Dimostrazione. Dobbiamo dimostrare la doppia implicazione $\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{NFA})$ e $\mathcal{L}(\text{NFA}) \subseteq \mathcal{L}(\text{DFA})$.

La **prima implicazione** è molto semplice infatti dato un linguaggio $L \in \mathcal{L}(\text{DFA})$ e un DFA D tale che $L = L(D)$ e siccome gli NFA sono una generalizzazione dei DFA avremo che D è anche un NFA e quindi $L \in \mathcal{L}(\text{NFA})$. Quindi $\mathcal{L}(\text{DFA}) \subseteq \mathcal{L}(\text{NFA})$.

Per la **seconda implicazione** prendiamo un NFA $= \left(Q_N, \Sigma, \delta_N, q_0^N, F_N\right)$ che riconosce un linguaggio A. Dobbiamo costruire un DFA $D = \left(Q_D, \Sigma, \delta_D, q_0^D, F_D\right)$ che riconosce A.

Consideriamo il caso in cui non abbiamo $\varepsilon -$ archi:
1. $Q_D = \mathbb{P}(Q_N)$ - Uno stato del DFA equivale quindi ad un insieme di stati del NFA.
2. Presi un $R \in Q_D$ e $a \in \Sigma$, sia

$$\delta_D(R, a) = \{q \in Q_N : q \in \delta_N(r, a) \quad \text{per qualche} \quad r \in R\}$$

Quindi la funzione di transizione del DFA equivale ad eseguire la transizione su tutti gli stati di R nel NFA.

Possiamo anche scriverla come:

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

3. $q_0^D = \{q_0^N\}$

4. $F_D = \{R \in Q : R \text{ contiene uno stato accettante di } N\}$ - Quindi il DFA accetta se e solo se nell'insieme risultante della transizione abbiamo almeno uno stato accettante dell'NFA. I due automi sono equivalenti.

Adesso consideriamo il caso con gli ε-archi, introduciamo delle notazioni. Per ogni R di D definiamo $E(R)$ come la collezione di stati che possono essere raggiunti dagli elementi di R proseguendo solo con ε-archi, includendo anche gli stessi elementi di R, in modo formale possiamo dire:

$$E(R) = \{q : q \text{ può essere raggiunto con } \geq 0 \quad \varepsilon - \text{archi}\}$$

Adesso modifichiamo la funzione di transizione di D in modo da far aggiungere gli stati che possono essere raggiunti da $\varepsilon -$ archi dopo ogni passo, sostituendo $\delta_N(r, a)$ con $E(\delta_N(r, a))$:

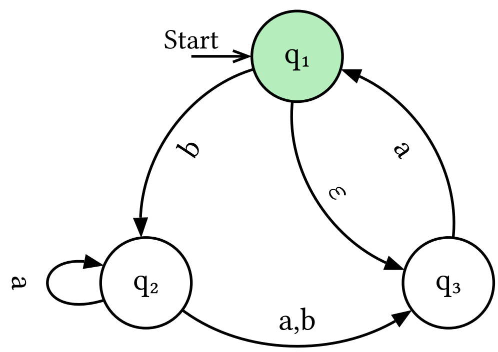
$$\delta_D(R, a) = \{q \in Q : q \in E(\delta_N(r, a)) \text{ per qualche } r \in R\}$$

Dobbiamo anche modificare lo stato iniziale di D in modo che anche nello stato iniziale raggiunga subito tutti gli stati possibili tramite $\varepsilon -$ archi e lo facciamo cambiando q_0^D in $E(\{q_0^N\})$.

Abbiamo completato la costruzione del DFA equivalente ad NFA, infatti ad ogni passo del NFA avremo che il DFA entra in uno stato equivalente all'insieme degli stati in cui si trova l'NFA.

4.3. Convertire un NFA in DFA

Prendiamo come esempio l'NFA:



Iniziamo a definire gli elementi del DFA D:

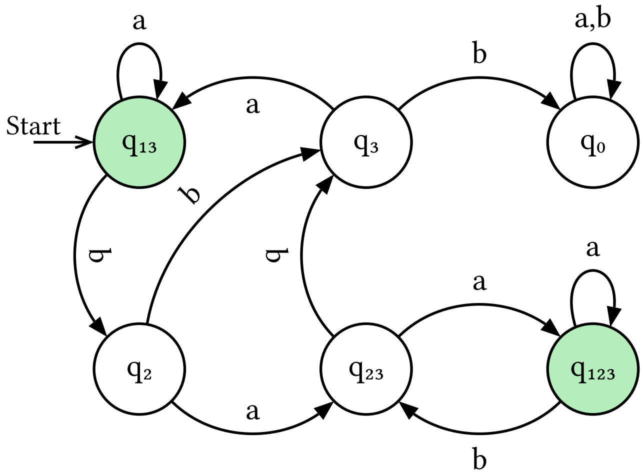
- $Q_D = \left\{q_0, q_{\{1\}}, q_{\{2\}}, q_{\{3\}}, q_{\{1,2\}}, q_{\{1,3\}}, q_{\{2,3\}}, q_{\{1,2,3\}}\right\}$
- $q_0^D = E(\{q_1\}) = q_{\{1,3\}}$ - Consideriamo quindi l'estensione dello stato iniziale con gli $\varepsilon -$ archi
- $F_D = \left\{q_{\{1\}}, q_{\{1,2\}}, q_{\{1,3\}}, q_{\{1,2,3\}}\right\}$ - Sono tutti gli stati che contengono almeno uno stato accettante, in questo caso soltanto q_1

Adesso dobbiamo calcolare δ_D, vediamo alcuni casi ma non tutti:

- $\delta_D\left(q_{\{2\}}, a\right) = q_{\{2,3\}}$
- $\delta_D\left(q_{\{2\}}, b\right) = q_{\{3\}}$

- $\delta_D\left(q_{\{3\}}, a\right) = q_{\{3\}}$ - Perché dobbiamo considerare anche l'ε-archi
- $\delta_D\left(q_{\{3\}}, b\right) = q_{\{\emptyset\}}$ - Infatti finisce la stringa ma non siamo in uno stato accettante

Ci sarebbero altre funzioni, ma vediamo cosa otteniamo:

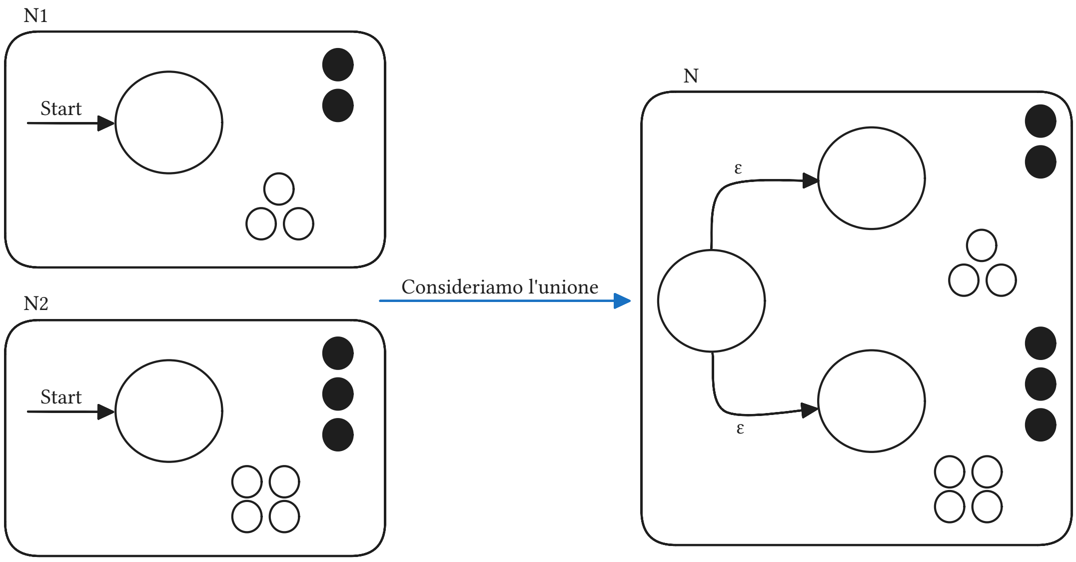


5. Proprietà di chiusura dei Linguaggi Regolari

5.1. Chiusura per Unione

Rivediamo l'unione utilizzando gli NFA, infatti adesso sappiamo che NFA e DFA sono equivalenti.

Uniamo due DFA in un NFA equivalente ai due:



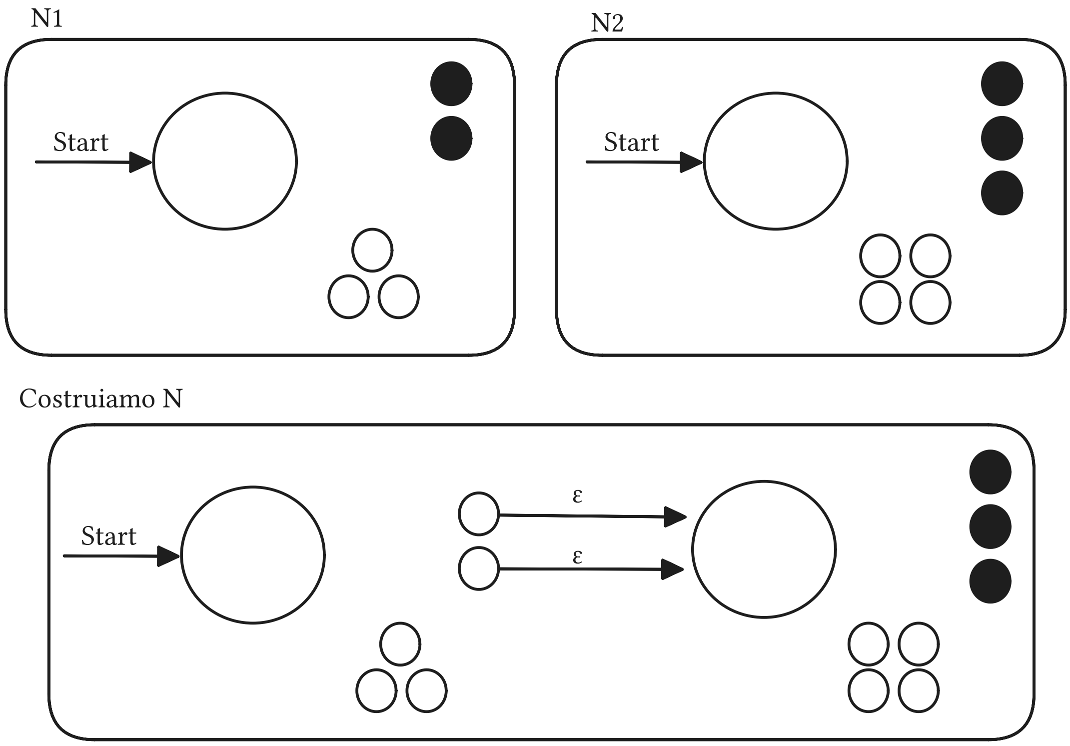
Infatti possiamo semplicemente considerare un NFA che come primo stato ci porta in modo parallelo su entrambi i DFA, avremo quindi che:

- $Q = Q_1 \cup Q_2 \cup \{q_0\}$
- $F = F_1 \cup F_2$
- $\forall q \in Q, a \in \Sigma_\varepsilon$:

$$\delta(q, a) = \begin{cases} \delta_1(q, a) \quad \text{se} \quad q \in Q_1 \\ \delta_2(q, a) \quad \text{se} \quad q \in Q_2 \\ \{q_0^1, q_0^2\} \quad \text{se} \quad q = q_0 \wedge a = \varepsilon \\ \emptyset \quad \text{se} \quad q = q_0 \wedge a \neq \varepsilon \end{cases}$$

5.2. Chiusura per Concatenazione

Dati due NFA N_1, N_2 per L_1, L_2 costruisco NFA per $L = L_1 \circ L_2$



Quindi li stati finali di N_1 li facciamo diventare dei normali stati che però hanno un $\varepsilon -$ arco verso lo stato iniziale di N_2

Formalmente:

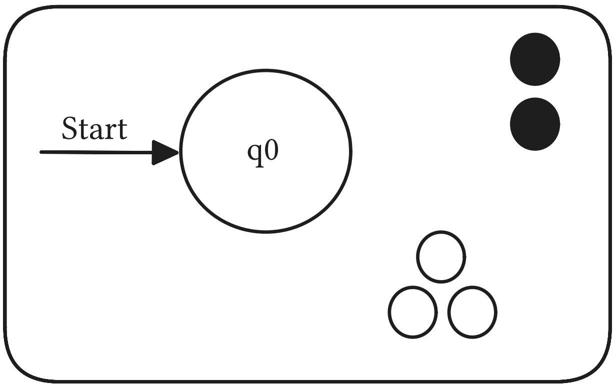
- $N = \{Q, \Sigma, \delta, q_0, F\}$
- $Q = Q_1 \cup Q_2$
- $q_0 = q_0^1$
- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$
- $F = F_2$
- $\forall q \in Q, \forall a \in \Sigma_\varepsilon$:

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{se} \quad q \in Q_1 \land q \notin F_1 \\ \delta_1(q, a) & \text{se} \quad q \in F_1 \land a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_0^2\} & \text{se} \quad q \in F_1 \land a = \varepsilon \\ \delta_2(q, a) & \text{se} \quad q \in Q_2 \end{cases}$$

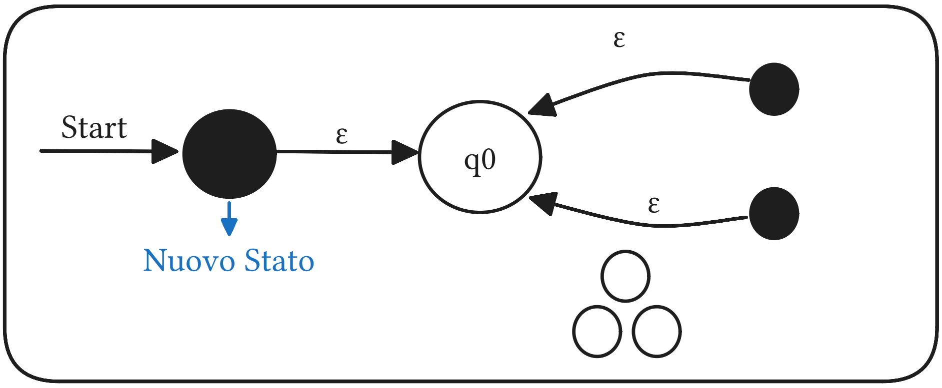
5.3. Chiusura per Operazione «*» star

Dato un NFA N t.c. $L(N) = L$ devo costruire NFA N^* t.c. $L(N^*) = L^*$

N



N*



Formalmente abbiamo $N^* = (Q', \Sigma, \delta', {q_0}', F')$ e $N = (Q, \Sigma, \delta, q_0, F)$:

- ${q_0}'$ nuovo stato iniziale
- $F' = F \cup \{{q_0}'\}$
- $Q' = Q \cup \{{q_0}'\}$
- $\forall q \in Q', \forall a \in \Sigma_\varepsilon$:

$$\delta(q, a) = \begin{cases} \delta(q, a) \quad \text{se} \quad q \in Q \wedge q \notin F \\ \delta(q, a) \quad \text{se} \quad q \in F \wedge a \notin \varepsilon \\ \delta(q, a) \cup \{q_0\} \quad \text{se} \quad q \in F \wedge a = \varepsilon \\ \{q_0\} \quad \text{se} \quad q = {q_0}' \wedge a = \varepsilon \\ \emptyset \quad \text{se} \quad q = {q_0}' \wedge a \neq \varepsilon \end{cases}$$

6. Espressioni Regolari

Possiamo vederle come delle espressioni algebriche, ma definiscono dei linguaggi su un certo alfabeto.

Esempio

$$(0 \cup 1) \bigcirc 0^* \quad \text{che equivale a} \quad \{0, 1\} \bigcirc 0^*$$

Definizione

Sia Σ un alfabeto possiamo definire un'espressione regolare ssu Σ (denotata con $\mathrm{re}(\Sigma)$) in modo ricorsivo:

$$\text{caso base} \begin{cases} \emptyset \in \mathrm{re}(\Sigma) \\ \varepsilon \in \mathrm{re}(\Sigma) \\ a \in \mathrm{re}(\Sigma) \quad \text{con} \quad a \in \Sigma \end{cases}$$

$$\text{induzione} \begin{cases} R_1 \cup R_2 \quad \text{se} \quad R_1, R_2 \in \mathrm{re}(\Sigma) \\ R_1 \bigcirc R_2 \quad \text{se} \quad R_1, R_2 \in \mathrm{re}(\Sigma) \\ (R_1)^* \quad \text{se} \quad R_1 \in \mathrm{re}(\Sigma) \end{cases}$$

Ogni espressione regolare ha un solo linguaggio associato:

$$L(r) \quad \text{t.c.} \quad r \in \mathrm{re}(\Sigma)$$

Vediamolo ricorsivamente:

$$\text{caso base} \begin{cases} L(r) = \emptyset \quad \text{se} \quad r = \emptyset \\ L(r) = \varepsilon \quad \text{se} \quad r = \varepsilon \\ L(r) = \{a\} \quad \text{se} \quad r = a \end{cases}$$

$$\text{induzione} \begin{cases} L(r) = L(R_1) \cup L(R_2) \quad \text{se} \quad r = R_1 \cup R_2 \\ L(r) = L(R_1) \bigcirc L(R_2) \quad \text{se} \quad r = R_1 \bigcirc R_2 \\ L(r) = (L(R_1))^* \quad \text{se} \quad r = R_1^* \end{cases}$$

Qualche esempio:

- $0^*10^* = \{w : w \ \text{ contiene esattamente un } 1\}$
- $\Sigma^*1\Sigma^* = \{w : w \ \text{ contiene almeno un } 1\}$

- $\Sigma^* 001 \Sigma^* = \{w : w \text{ contiene 001 come sottostringa}\}$
- $(0 \cup 1000)^* = \{w : \text{ogni occorrenza di 1 é seguita da 000}\}$

Teorema

Un linguaggio regolare è regolare se e solo se esiste un'espressione regolare che lo descrive:

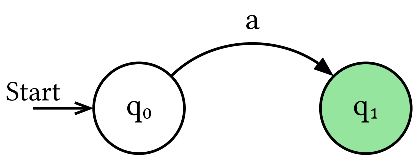
$$\text{REG} \equiv L(\text{re})$$

Per dimostrarlo dobbiamo dimostrare la doppia implicazione $L(\text{re}) \subseteq \text{REG}$ e $\text{REG} \subseteq L(\text{re})$.

Iniziamo dimostrando $L(\text{re}) \subseteq \text{REG}$, quindi data un'espressione regolare r costruiamo un NFA N_r tale che $L(N_r) = L(r)$.

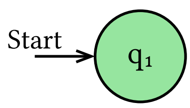
I casi base sono 3, quando l'espressione regolare è un solo carattere, quando l'espression regolare è la stringa vuota oppure quando è l'insieme vuoto.

Data l'espressione regolare $r = a$ con $a \in \Sigma$ costruiamo l'NFA che la riconosce:



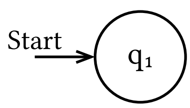
Dove:
- $N_r = (\{q_0, q_1\}, \Sigma, \delta, q_0, \{q_1\})$
- $\delta(q_0, a) = q_1$
- $\delta(q, b) = \emptyset$ con $q \neq q_1 \wedge b \neq a$

Se invece abbiamo $r = \varepsilon$, possiamo costrutire:



Definiamo:
- $N_r = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$
- $\delta(q_1, b) = \emptyset, \forall b \in \Sigma$

Se invece $r = \emptyset$ costruiamo:



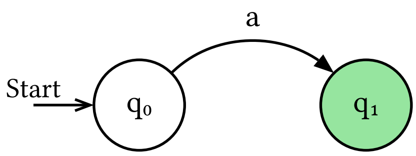
Definiamo:
- $N_r = (\{q_1\}, \Sigma, \delta, q_1, \emptyset)$
- $\delta(q_1, b) = \emptyset, \forall b \in \Sigma$

Adesso per il caso induttivo dobbiamo considerare un'espressione regolare $r = R_1 \cup R_2$ e per ipotesi induttiva sappiamo che $\exists M_1, M_2$ t.c. $L(M_1) = L(R_1) \wedge L(M_2) = L(R_2)$ e per chiusura di REG sappiamo che questo implica che $\exists M$ t.c. $L(M) = L(R_1) \cup L(R_2)$. Questo è analogo per la concatenazione e l'operatore star.

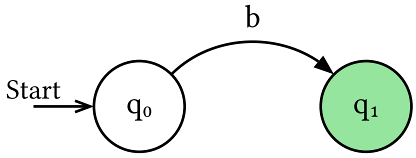
Esempio

Vogliamo costruire un NFA che riconosce il linguaggio $(\mathrm{ab} \cup \mathrm{a})^*$

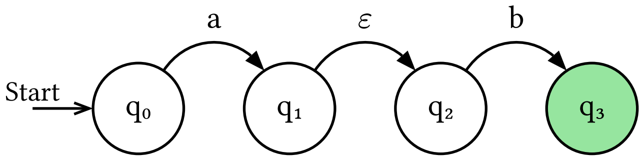
- Possiamo costruire prima l'automa che riconosce a:



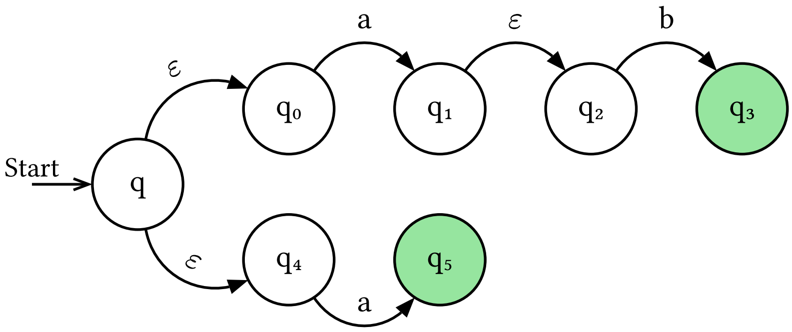
- Poi costruiamo l'automa che riconosce b:



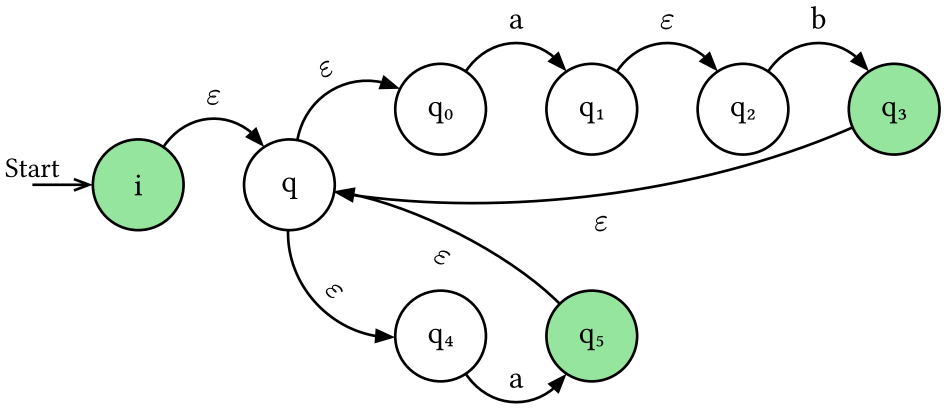
- Adesso possiamo ricavare quello che riconosce ab:



- Costruiamo l'automa che riconosce $(\mathrm{ab} \cup \mathrm{a})$:



- Infine costruiamo l'NFA che riconosce il linguaggio dell'espressione regolare:



Adesso dobbiamo dimostrare la seconda implicazione, quindi REG $\subseteq L(\text{re})$. Dobbiamo quindi prendere un NFA e vogliamo trovare l'espressione regolare corrispondente.

Prima di farlo vediamo come **convertire un NFA in un'espressione regolare**.

6.1. Convertire NFA in espressione regolare

Partiamo da un NFA N con $L \in \text{REG} : L(N) = L$ e introduciamo il concetto di **GNFA (NFA Generalizzato)**.

Per GNFA intendiamo un NFA dove le etichette degli archi sono delle espressioni regolari.

Forma Canonica del GNFA

Un GNFA si trova in forma canonica quando:
- Lo stato iniziale ha solo archi uscenti verso tutti gli altri stati
- Lo stato finale ha solo archi entranti
- Fatta eccezione per lo stato finale ed iniziale esiste un arco fra ogni coppia di stati

Più formalmente dato GNFA=$(Q, \Sigma, \delta, q_{\text{START}}, q_{\text{ACC}})$:

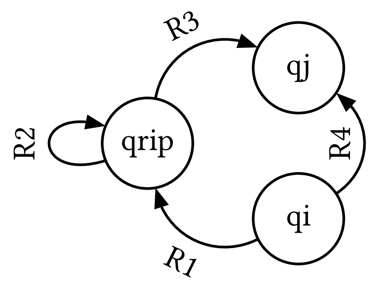
$$\delta : Q \setminus \{q_{\text{ACC}}\} \times Q \setminus \{q_{\text{START}}\} \to \mathcal{R} = \text{re}(\Sigma)$$

Dove $\mathcal{R}$ è l'insieme di tutte le espressioni regolari sul linguaggio Σ

Definiamo adesso la funzione `Convert(G)` che prende in input un grafo e restituisce l'espressione regolare associata.

Definiamo `Convert` :
- Definiamo $k =$ numero di stati in G.
- Se $k = 2$ significa che abbiamo soltanto $q_{\text{start}}, q_{\text{acc}}$ e un singolo arco con etichetta $R \in \mathcal{R}$. Avremo R come output.
- Se $k > 2$ scegliamo uno stato q_{rip} diverso da q_{start} e q_{acc} e definiamo $G' = \{Q', \Sigma, \delta', q_{\text{start}}, q_{\text{acc}}\}$ dove:
  - $Q' = Q \setminus \{q_{\text{rip}}\}$
  - $\delta' : Q' \setminus \{q_{\text{acc}}\} \times Q' \setminus \{q_{\text{start}}\} \to \mathcal{R}$

Adesso $\forall q_i \in Q' \setminus \{a_{\text{acc}}\}, q_j \in Q' \setminus \{q_{\text{start}}\}$ consideriamo l'automa:



Dove:
- $\text{R1} = \delta(q_i, q_{\text{rip}})$
- $\text{R2} = \delta(q_{\text{rip}}, q_{\text{rip}})$
- $\text{R3} = \delta(q_j, q_{\text{rip}})$

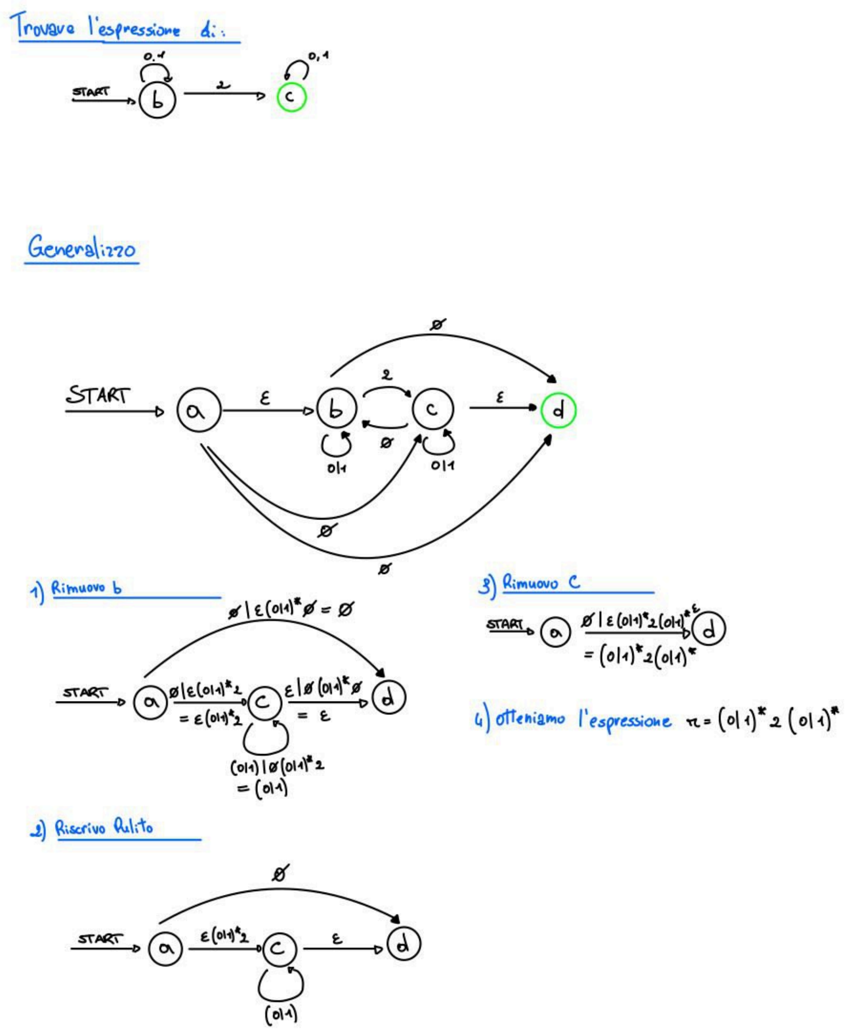
- $\mathrm{R4} = \delta(q_i, q_j)$

Consideriamo questo automa perché ci interessano soltanto gli archi che collegano q_i e q_j oppure che riguardano q_{rip}. Avremo quindi che

$$\delta'(q_i, q_j) = (\mathrm{R1})(\mathrm{R2})^*(\mathrm{R3}) \cup (\mathrm{R4})$$

Abbiamo quindi definito un automa che ci permette di muoverci fra q_i e q_j anche senza q_{rip}, dobbiamo continuare a ripetere questo procedimento finché non rimaniamo soltanto con lo stato iniziale e lo stato accettante.

Esempio



Quando avrò tempo lo farò più carino :P

Adesso possiamo concludere la dimostrazione iniziata nel capitolo precedente.

Dobbiamo dimostrare che quello che otteniamo da `Convert(G)` è equivalente a G.

Se $k = 2$ è sicuramente vero.

L'espressione regolare R descrive tutte le stringhe che portano, in G, da q_{start} a q_{acc}.

Supponiamo che sia vero per $k-1$ stati e dimostriamo che è vero per k stati mostrando che $L(G) = L(G')$ dove G' è l'automa con uno stato rimosso.

Se l'automa G accetta una stringa w significa che esiste un ramo di computazione che permette a G di percorrere gli stati $q_{\text{start}} \ldots q_{\text{acc}}$, se questa sequenza non contiene q_{rip} allora abbiamo che $L(G) = L(G')$ perché le nuove espressioni regolari conterranno le vecchie per unione.

Se invece q_{rip} è presente nella sequenza avremo comunque che gli stati a lui adiacenti (q_1, q_2) in G' hanno degli archi che tengono conto di tutti i modi per percorrere un cammino da q_1 a q_2 direttamente o passando per q_{rip} e quindi otteniamo di nuovo $L(G) = L(G')$.

7. Pumping Lemma

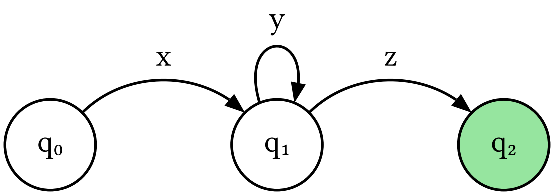
Serve a dimostare che un linguaggio non è naturale.

Teorema - Pumping Lemma

Se L è regolare, allora esiste p t.c. presa $w \in L$ con $|w| \geq p$, allora w può essere scomposta in $w = xyz$ in modo che:
1. $\forall i \geq 0$ si ha che $xy^i z \in L$
2. $|y| > 0$
3. $|xy| \leq p$

Vedremo che questo p è il numero di stati dell'automa, infatti prima di dimostrare ragionamo su questo caso:

Fissiamo $p =$#stati automa ed M t.c. $L(M) = L$ e siccome $|w| \geq p$, scomponiamo in questo modo:



Una ripetizione, in questo caso y, deve esistere sempre dato che la stringa è più grande del numero di stati. Significa appunto che uno stato deve sicuramente ripetersi.

7.1. Dimostrazione

Sia $M = (Q, \Sigma, \delta, q_1, F)$ t.c. $L(M) = L$ e sia $p = |Q|$. Consideriamo inoltre $w = w_1 w_2 ... w_n$ con $n \geq p$.

Consideriamo anche la sequenza di stati $r_1, ..., r_{n+1}$ attraversati da M su input w, avremo che $r_1 = q_1$ e $r_{n+1} \in F$. Ovviamente avremo che $n + 1 \geq p + 1$.

Per il **pigeonhole principle**, nella sequenza considerata ci sarà sicuramente uno stato che si ripete, sia questo stato r_j nella prima apparizione e r_l nella seconda ($j \neq l$, lo stato è lo stesso ma consideriamo due iterazioni diverse ovvero j quando lo incontriamo e l quando si ripete per la prima volta), avremo ovviamente che $l \leq p + 1$ perché r_l si presenta tra le prime $p + 1$ posizioni nella sequenza che inizia con r_1.

Scomponiamo la stringa in $w = xyz$ e poniamo:
- $x = w_1, ... w_{j-1}$. Ovvero la stringa prima del primo stato che si ripete.
- $y = w_j ... w_{l-1}$. Prima della prima ripetizioni.
- $z = w_l ... w_n$. Tutto il resto della stringa.

Abbiamo che:
- x porta M da $r_1 = q_1$ ad r_j
- y porta M da r_j ad $r_l = r_j$
- z porta M da $r_j = r_l$ a $r_{n+1} \in F$

Quindi notiamo che possiamo ripetere y quante volte vogliamo e la stringa ottenuta xy^iz apparterrà sempre al linguaggio. **Dimostrata la prima condizione**.

Siccome $j \neq l$ per costruzione allora $|y| > 0$. **Seconda condizione**.

Infine $l \leq p + 1$ e allora $|xy| = l - 1 \leq p$. **Terza condizione**

7.2. Esempi

Utilizziamo il pumping lemma.

1) Mostrare che $L = \{o^n 1^n : n \geq 0\}$ non è regolare.

Scegliamo una stringa $0^p 1^p$ con p che sarà il nostro **valore di pumping** che scegliamo per contraddire la prova. Vogliamo comunque $|w| \geq q$ per rientrare nelle condizioni.

Se il linguaggio fosse regolare allora presa $w = 0^p 1^p$, per qualsiasi scomposizione $w = xyz$ t.c. $|xy| \leq p$ avremo che y è composta da soli "0":

$$w = \underbrace{0...}_{\text{x}}\underbrace{...}_{\text{y}}01.....1$$

Per falsificare la condizione quindi ci basta prendere una $i \geq 2$ e avremo una stringa $xy^iz = o^q 1^p$ con $q > p$ che non rientra nel linguaggio dato che il numero di 0 ed 1 non è lo stesso.

2) Mostrare che il linguaggio $L = \{w \in \{0, 1\}^* : \#_0 w = \#_1 w\}$ ovvero le stringhe hanno lo stesso numero di 0 ed 1 ma in qualsiasi ordine.

Proviamo a scomporre con $w \in L$ t.c. $w = (01)^p$ e con $|w| = 2p \geq p$

Otteniamo una stringa:

$$\underbrace{01}_{\text{y}}\underbrace{0101010...01}_{\text{z}} \quad x = \varepsilon$$

Notiamo però che questa scomposizione non va bene per falsificare le condizioni, infatti qualsiasi i prendiamo aumentiamo sia il numero di 0 che di 1 quindi la stringa appartiene al linguaggio.

Proviamo con la stringa $w = 0^p 1^p$ con $|w| = 2p$ e rispettiamo $|xy| \leq p$ e $|y| > 0$.

Siccome $|xy| \leq p$ allora y è fatta solo da 0:

$$\underbrace{0....0}_{\text{x}}\underbrace{0.....0}_{\text{y}}\underbrace{1....1}_{\text{z}}$$

In questo caso aumentando y aumentiamo soltanto gli 0 e quindi la stringa non appartiene a L.

Più precisamente abbiamo che:

- $|y| = k > 0$ ma $k \leq p$ e inoltre $|x| = p - k$, $|z| = p$, tuttavia $|xy^2z| = (p - k) + 2k + p = 2p + k$ ma il numero di 0 è $(p - k) + 2k = p + k$ mentre quello degli 1 è sempre p che è $< p + k$, quindi non rientra nel linguaggio.

Con la stessa stringa possiamo provare anche questa scomposizione:

$$\underbrace{0.....}_{x}\underbrace{...00..}_{y}\underbrace{...01...1}_{z}$$

Anche in questo caso aumentiamo solo gli 0 e quindi non rientriamo nel linguaggio. Più precisamente:

- $|y| = k > 0$
- $|z| = p + l$
- $|x| = p - k - l$
- Assumendo $l > 0$

Tuttavia $|xy^2z| = (p - k - l) + 2k + p + l = 2p + k$ ma il numero di 0 è $(p - k - l) + 2k + l = p + k$ mentre quello degli 1 è $p < p + k$.