STRUMENTI FORMALI PER LA BIOINFORMATICA

Automi finiti - Parte 1

Ringraziamenti

Le figure sono prese dai libri (o dispense):

P. Degano, *Fondamenti di Informatica: Calcolabilità e Complessità*, Dispense, 2019.

J. Hopcroft, R. Motwani, J. Ullman , *Automi, Linguaggi e Calcolabilità*, Addison Wesley Pearson Education Italia s.r.l, Terza Edizione, 2009.

Michael Sipser, *Introduzione alla teoria della Computazione*, Apogeo Education, Maggioli Editore, 2016 (traduzione italiana di Introduction to the Theory of Computation, 3rd Edition).

La teoria degli automi finiti è nata in relazione a due problematiche completamente diverse.

La teoria degli automi finiti è nata in relazione a due problematiche completamente diverse.

La prima, in ambiente logico-matematico, riguarda la formalizzazione matematica della nozione di algoritmo. Contributi fondamentali in questa direzione sono stati dati verso la fine degli anni 30 da ricercatori quali Turing, Church e Kleene.

La teoria degli automi finiti è nata in relazione a due problematiche completamente diverse.

La prima, in ambiente logico-matematico, riguarda la formalizzazione matematica della nozione di algoritmo. Contributi fondamentali in questa direzione sono stati dati verso la fine degli anni 30 da ricercatori quali Turing, Church e Kleene.

La seconda ebbe origine verso la metà degli anni 40 ad opera del neurofisiologo McCulloch e del matematico Pitts. Essa riguardava la possibilità di fornire una descrizione matematica del funzionamento delle cellule nervose, o neuroni, e delle reti di neuroni.

Dai due filoni di ricerca nacquero e si svilupparono due teorie di grande interesse sia per la Matematica che per l'Informatica.

Dai due filoni di ricerca nacquero e si svilupparono due teorie di grande interesse sia per la Matematica che per l'Informatica.

Il primo lavoro che fornisce un'esposizione chiara e abbastanza esauriente della teoria degli automi finiti è un articolo di Rabin e Scott del 1959.

- Esempi di applicazioni di definizioni e risultati della teoria degli automi finiti:

- Esempi di applicazioni di definizioni e risultati della teoria degli automi finiti:
  - analizzatori lessicali, generatori di analizzatori lessicali

- Esempi di applicazioni di definizioni e risultati della teoria degli automi finiti:
  - analizzatori lessicali, generatori di analizzatori lessicali
  - uso delle espressioni regolari in unix/linux

- Esempi di applicazioni di definizioni e risultati della teoria degli automi finiti:
  - analizzatori lessicali, generatori di analizzatori lessicali
  - uso delle espressioni regolari in unix/linux
  - software per trovare occorrenze di parole o frasi in un testo

- Esempi di applicazioni di definizioni e risultati della teoria degli automi finiti:
  - analizzatori lessicali, generatori di analizzatori lessicali
  - uso delle espressioni regolari in unix/linux
  - software per trovare occorrenze di parole o frasi in un testo
- Automi finiti: modello per flip-flop, unità di controllo di un computer, gestione dei processi in un sistema operativo...

- Esempi di applicazioni di definizioni e risultati della teoria degli automi finiti:
  - analizzatori lessicali, generatori di analizzatori lessicali
  - uso delle espressioni regolari in unix/linux
  - software per trovare occorrenze di parole o frasi in un testo
- Automi finiti: modello per flip-flop, unità di controllo di un computer, gestione dei processi in un sistema operativo...
- Automi finiti: modello per controllo di ascensori, porte scorrevoli, calcolatrici, macchine distributrici di bibite e molti elettrodomestici di uso comune.

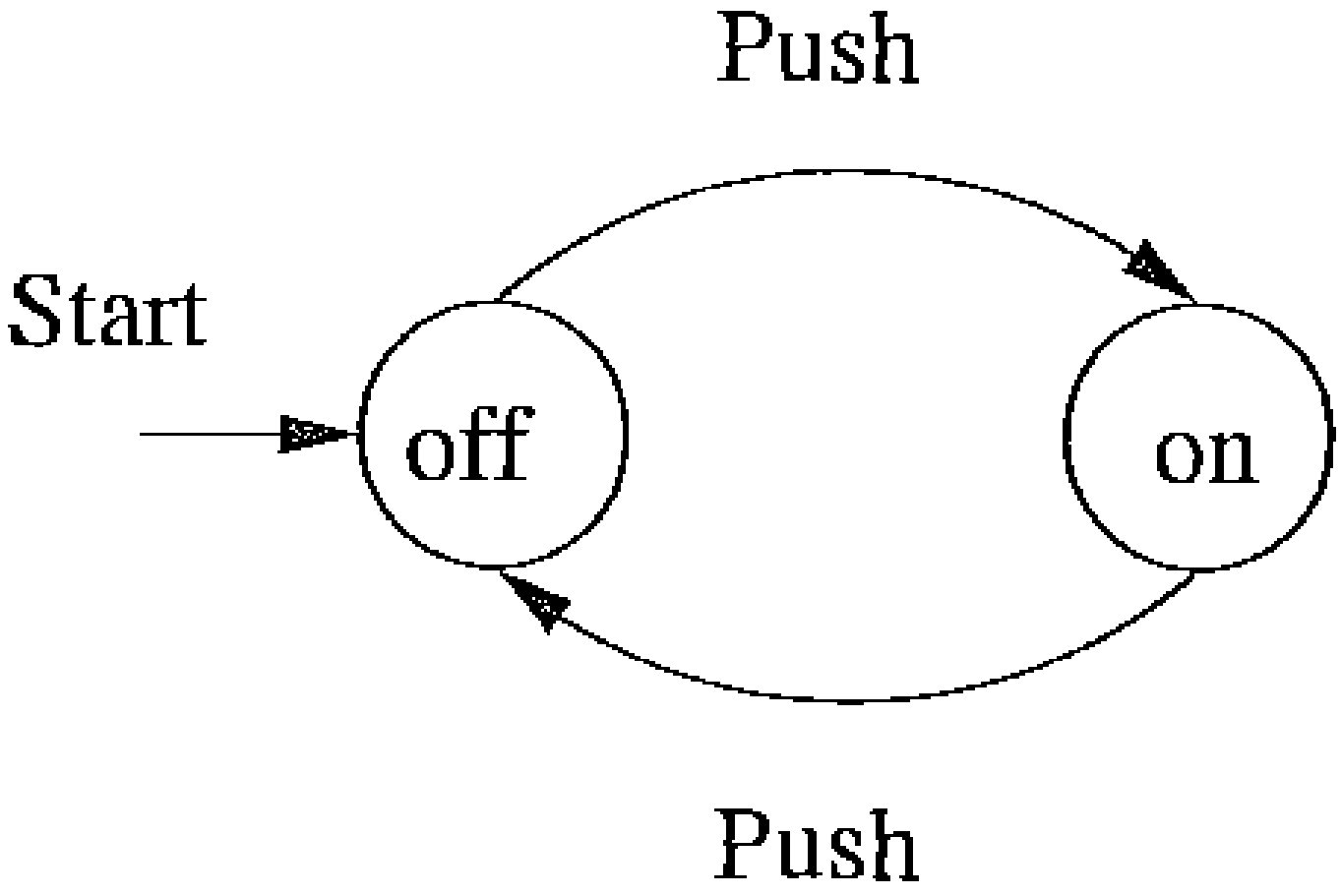


Figura: Un modello per il controllo di un interruttore.

I linguaggi associati agli automi finiti (che chiameremo **regolari**) sono particolarmente utili nella costruzione di compilatori e interpreti nella fase di individuazione dei *token*, cioè degli elementi basilari di un programma, per esempio identificatori, parole chiave, delimitatori, operatori ecc.

Per esempio, il comando seguente

```
PIPPO := 3.14 + A;
```

è composto da 18 caratteri e può venir rappresentato ad esempio come la successione di token

```
1 7 5 2 1 9
```

in cui si codificano con il token 1 gli identificatori, con il 7 l'operatore di assegnamento, con il 5 una costante, con 2 un operatore aritmetico e con 9 un delimitatore. Naturalmente bisognerà tener traccia del fatto che il primo 1 è associato a `PIPPO` e il secondo ad `A` ecc. e questo si fa utilizzando una tabella, detta dei simboli.

Per verificare la correttezza dal punto di vista strettamente sintattico del comando

```
PIPPO := 3.14 + A;
```

per esempio che l'espressione alla destra del := è generabile dalla grammatica delle espressioni aritmetiche, è sufficiente prendere in esame la successione di token

```
1 7 5 2 1 9
```

Il modello mentale

Un automa finito deterministico è un modello di computazione dotato di una quantità estremamente limitata di memoria.
Rappresenta un buon modello di computer.

Il modello mentale

Un automa finito deterministico è un modello di computazione dotato di una quantità estremamente limitata di memoria. Rappresenta un buon modello di computer.

Possiamo immaginarlo come un sistema che può trovarsi in un numero finito di stati e un "controllo" che si muove da stato a stato in risposta a input esterni.

Il modello mentale

Un automa finito deterministico è un modello di computazione dotato di una quantità estremamente limitata di memoria. Rappresenta un buon modello di computer.

Possiamo immaginarlo come un sistema che può trovarsi in un numero finito di stati e un "controllo" che si muove da stato a stato in risposta a input esterni.

È dotato di una testina di lettura che può muoversi solo verso destra.

Il modello mentale

Un automa finito deterministico è un modello di computazione dotato di una quantità estremamente limitata di memoria. Rappresenta un buon modello di computer.

Possiamo immaginarlo come un sistema che può trovarsi in un numero finito di stati e un "controllo" che si muove da stato a stato in risposta a input esterni.

È dotato di una testina di lettura che può muoversi solo verso destra.

In funzione dello stato in cui si trova e dell'input letto, la macchina cambia stato, poi sposta la testina di lettura sulla cella successiva.

Il modello mentale

Un automa finito deterministico è un modello di computazione dotato di una quantità estremamente limitata di memoria. Rappresenta un buon modello di computer.

Possiamo immaginarlo come un sistema che può trovarsi in un numero finito di stati e un "controllo" che si muove da stato a stato in risposta a input esterni.

È dotato di una testina di lettura che può muoversi solo verso destra.

In funzione dello stato in cui si trova e dell'input letto, la macchina cambia stato, poi sposta la testina di lettura sulla cella successiva.

Termina dopo aver letto l'ultimo simbolo sul nastro input; lo stato raggiunto determina se la parola è accettata o no.

Il modello mentale

La macchina automa finito incorpora un programma con istruzioni del tipo:

Il modello mentale

La macchina automa finito incorpora un programma con istruzioni del tipo:

- nello stato q se leggi i vai nello stato p

Il modello mentale

La macchina automa finito incorpora un programma con istruzioni del tipo:

- nello stato q se leggi i vai nello stato p
- nello stato p se leggi n vai nello stato t

Il modello mentale

La macchina automa finito incorpora un programma con istruzioni del tipo:

- nello stato q se leggi i vai nello stato p
- nello stato p se leggi n vai nello stato t
- ...

Il modello mentale

La macchina automa finito incorpora un programma con istruzioni del tipo:

- nello stato q se leggi i vai nello stato p
- nello stato p se leggi n vai nello stato t
- ...

A ogni passo la testina di lettura si sposta di una cella a destra.

Il modello mentale

La macchina automa finito incorpora un programma con istruzioni del tipo:

- nello stato q se leggi i vai nello stato p
- nello stato p se leggi n vai nello stato t
- ...

A ogni passo la testina di lettura si sposta di una cella a destra.

La macchina si ferma quando ha letto tutto l'input.

Definizione formale di un DFA

Definizione (Automa a stati finiti deterministico)

Un automa a stati finiti deterministico (in breve DFA) è una quintupla

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$$

dove:

- Q *è l'insieme* finito *degli stati;*
- Σ *è l'alfabeto (finito);*
- $q_0 \in Q$ *è lo stato iniziale*
- $\delta : Q \times \Sigma \to Q$ *è la funzione di transizione;*
- $F \subseteq Q$ *è l'insieme degli stati finali.*

Notazioni per DFA

Specificare un DFA attraverso una quintupla e una descrizione dettagliata della δ è noioso e di difficile lettura. Sistemi alternativi sono

Notazioni per DFA

Specificare un DFA attraverso una quintupla e una descrizione dettagliata della δ è noioso e di difficile lettura. Sistemi alternativi sono

- i diagrammi di stato (o di transizione)

Notazioni per DFA

Specificare un DFA attraverso una quintupla e una descrizione dettagliata della δ è noioso e di difficile lettura. Sistemi alternativi sono

- i diagrammi di stato (o di transizione)
- le tabelle di transizione, cioè specifiche tabellari della funzione δ da cui si deducono l'insieme degli stati e l'alfabeto input. Ovviamente vanno anche specificati stato iniziale e stati finali.

Un **diagramma di stato** per un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ è un grafo orientato definito come segue.

Un **diagramma di stato** per un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ è un grafo orientato definito come segue.

- Per ogni stato di Q esiste un nodo.

Notazioni per DFA

Un **diagramma di stato** per un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ è un grafo orientato definito come segue.

- Per ogni stato di Q esiste un nodo.
- Per ogni stato q in Q e carattere a in Σ se $\delta(q, a) = p$ c'è un arco dal nodo q al nodo p etichettato a.
  Se $\delta(q, a_1) = \delta(q, a_2) = \ldots = \delta(q, a_n) = p$, il diagramma può avere un solo arco da q a p etichettato $a_1, \ldots, a_n$.

Notazioni per DFA

Un **diagramma di stato** per un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ è un grafo orientato definito come segue.

- Per ogni stato di Q esiste un nodo.
- Per ogni stato q in Q e carattere a in Σ se $\delta(q, a) = p$ c'è un arco dal nodo q al nodo p etichettato a.
  Se $\delta(q, a_1) = \delta(q, a_2) = \ldots = \delta(q, a_n) = p$, il diagramma può avere un solo arco da q a p etichettato $a_1, \ldots, a_n$.
- C'è una freccia entrante nel nodo associato allo stato iniziale q_0 e tale freccia non proviene da nessun nodo.

Notazioni per DFA

Un **diagramma di stato** per un DFA $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ è un grafo orientato definito come segue.

- Per ogni stato di Q esiste un nodo.
- Per ogni stato q in Q e carattere a in Σ se $\delta(q, a) = p$ c'è un arco dal nodo q al nodo p etichettato a.
  Se $\delta(q, a_1) = \delta(q, a_2) = \ldots = \delta(q, a_n) = p$, il diagramma può avere un solo arco da q a p etichettato $a_1, \ldots, a_n$.
- C'è una freccia entrante nel nodo associato allo stato iniziale q_0 e tale freccia non proviene da nessun nodo.
- I nodi corrispondenti a stati finali sono indicati da un doppio circolo, gli altri da un solo circolo.

Esempio. Sia $\mathcal{A}_1 = (Q, \Sigma, \delta, q_1, F)$ dove

$$
\begin{aligned}
Q &= \{q_1, q_2, q_3\} \\
\Sigma &= \{0, 1\} \\
F &= \{q_2\} \\
\delta(q_1, 0) &= q_1, \quad \delta(q_1, 1) = q_2, \\
\delta(q_2, 0) &= q_3, \quad \delta(q_2, 1) = q_2, \\
\delta(q_3, 0) &= q_2, \quad \delta(q_3, 1) = q_2
\end{aligned}
$$

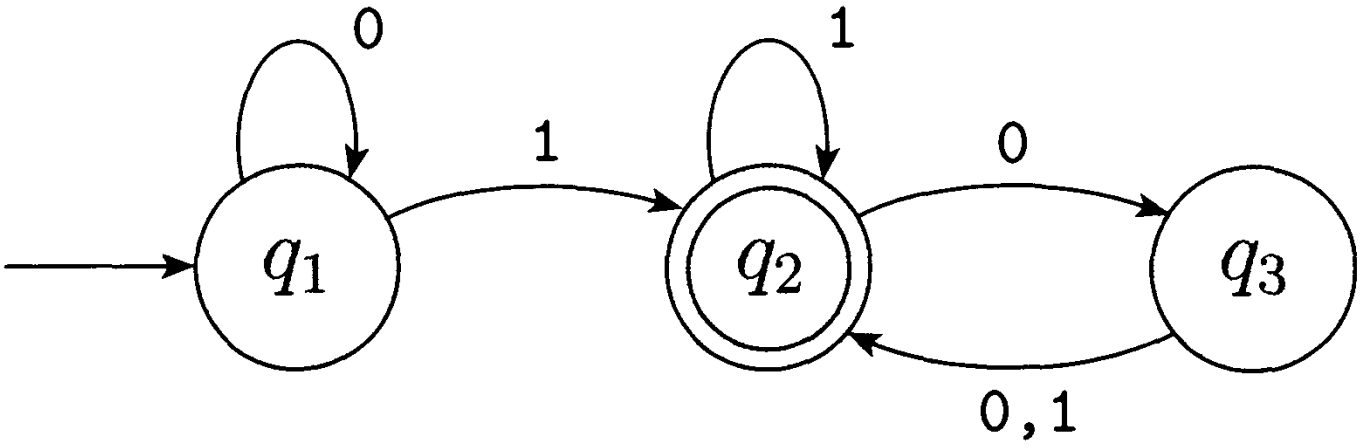


Figura: Il DFA $\mathcal{A}_1$

Funzione di transizione estesa di un DFA

Definizione

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un automa finito deterministico. La funzione di transizione estesa $\hat{\delta}$: $Q \times \Sigma^* \to Q$ è definita ricorsivamente come segue

$$
\begin{array}{rcl}
\textbf{Passo Base} & & \forall q \in Q, \\
\hat{\delta}(q, \epsilon) & = & q \\
\textbf{Passo Ricorsivo} & & \forall q \in Q, w \in \Sigma^*, a \in \Sigma, \\
\hat{\delta}(q, wa) & = & \delta(\hat{\delta}(q, w), a)
\end{array}
$$

Linguaggio accettato da un DFA

Definizione

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *un automa finito deterministico. Il linguaggio* **accettato o riconosciuto da** $\mathcal{A}$ **è**

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$$

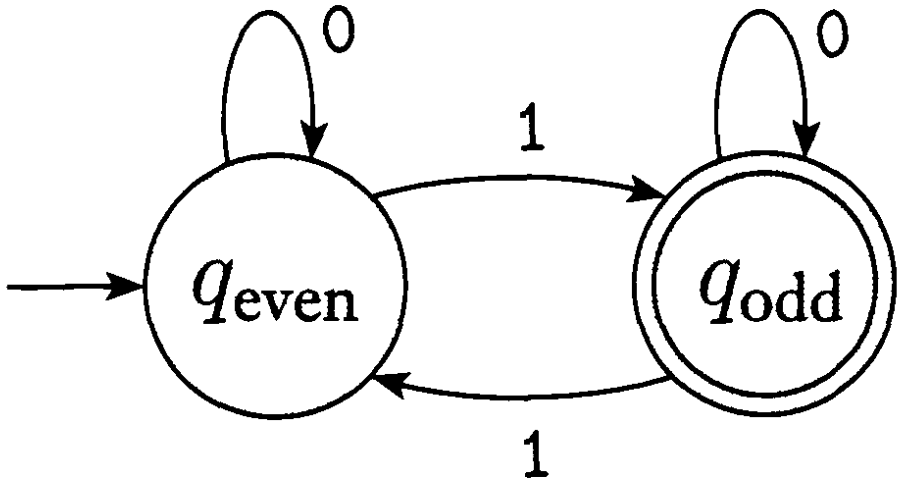


Figura: Un DFA che accetta $\{w \in \{0,1\}^* \mid |w|_1 = 2k+1,\ k \geq 0\}$

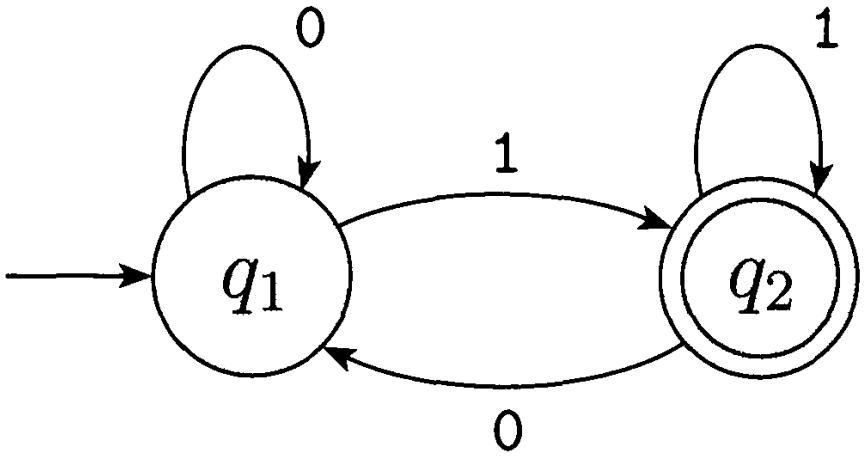


Figura: Un DFA che accetta $\{w1 \mid w \in \{0,1\}^*\}$

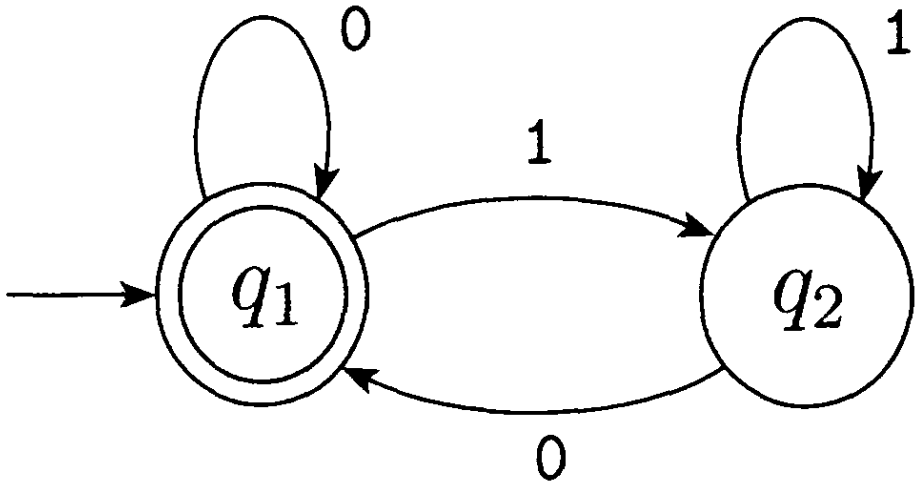


Figura: Un DFA che accetta $\{0,1\}^* \setminus \{w1 \mid w \in \{0,1\}^*\}$

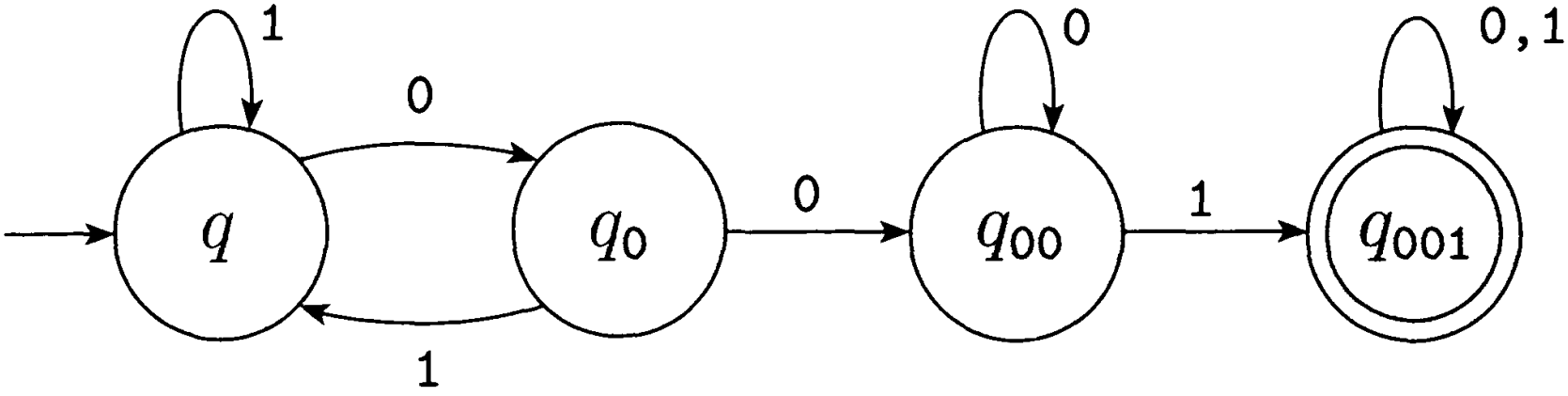


Figura: Un DFA che accetta $\{x001y \mid x, y \in \{0,1\}^*\}$

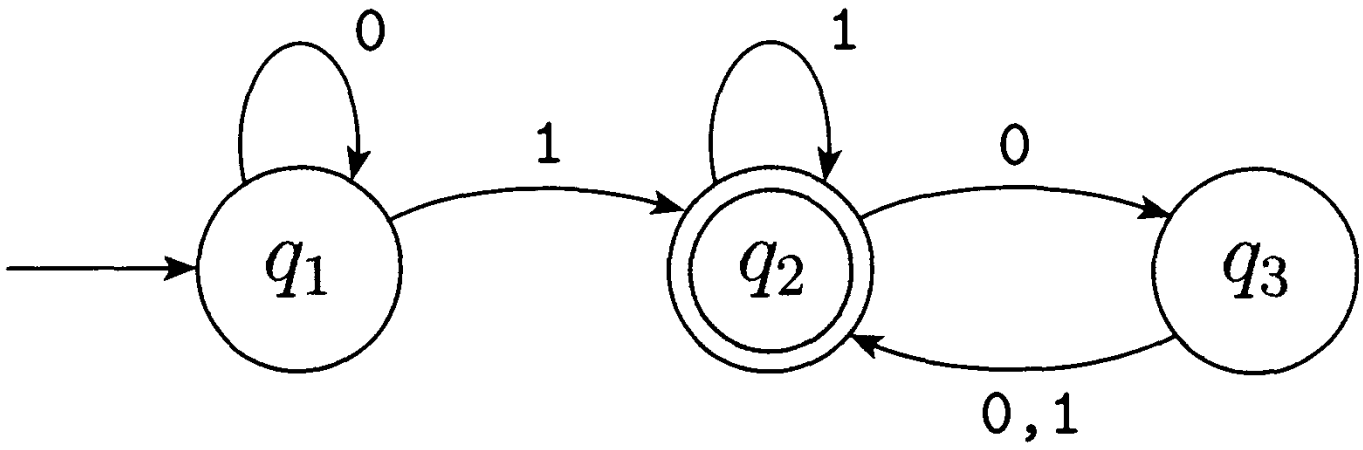


Figura: Un DFA che accetta $\{x1(00)^k \mid x \in \{0,1\}^*,\ k \geq 0\}$

Esempi di Automi

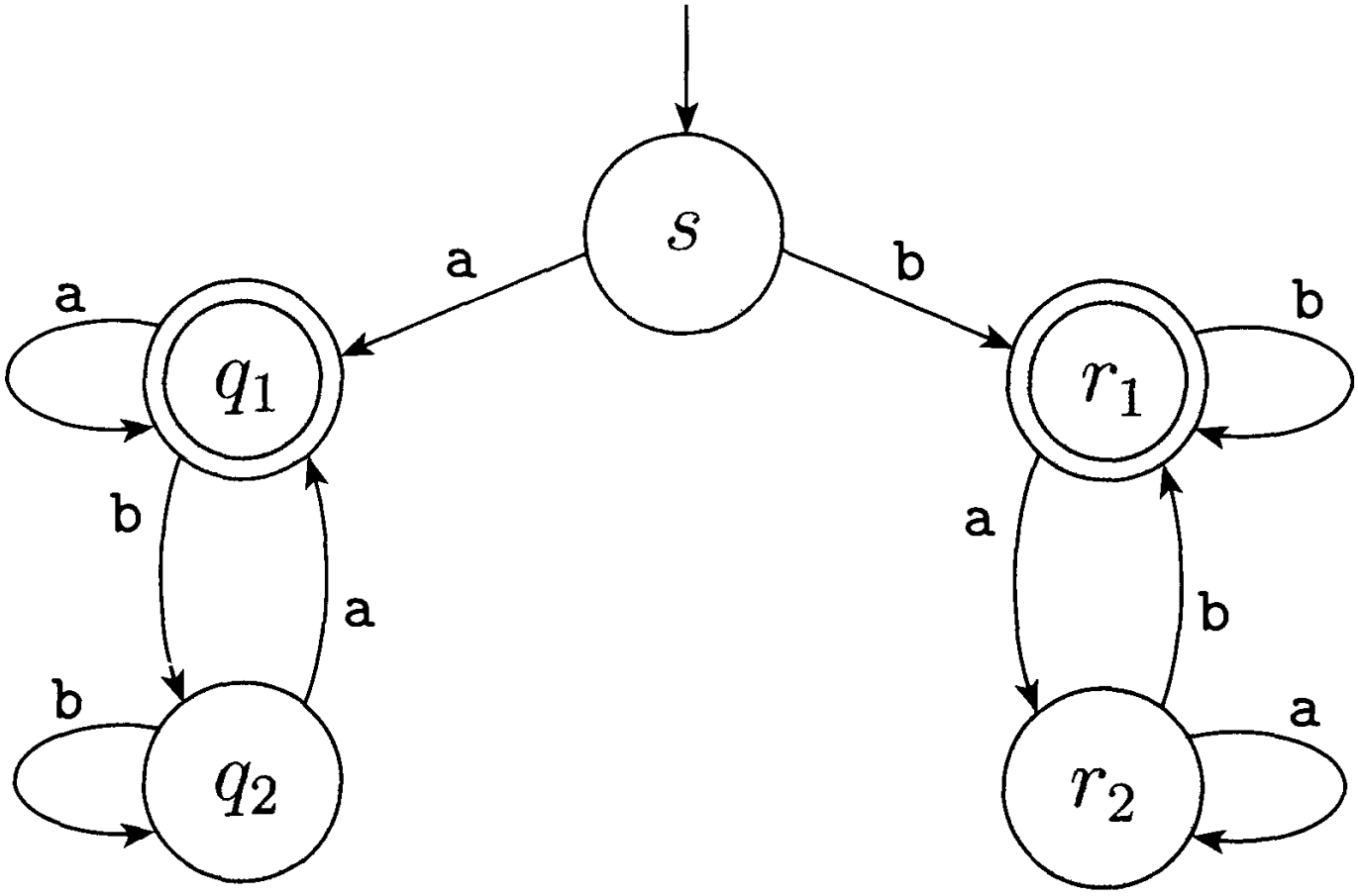


Figura: Un DFA che accetta
$\{w \in \{a, b\}^* \mid w$ inizia e termina con lo stesso simbolo$\}$

Definizione

Un linguaggio $L \subseteq \Sigma^*$ è regolare se esiste un DFA $\mathcal{A}$ tale che L è il linguaggio riconosciuto da $\mathcal{A}$, cioè $L = L(\mathcal{A})$.

Chiusura rispetto a un'operazione

Definizione

Una classe di oggetti è **chiusa** *rispetto a un'operazione se l'applicazione di questa operazione a elementi della classe restituisce un oggetto ancora nella classe.*

Teorema

La classe dei linguaggi regolari è chiusa rispetto al complemento.

Prova (Cenni) Sia L un linguaggio regolare, sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un DFA che riconosce L, cioè $L = L(\mathcal{A})$. Consideriamo il DFA $\mathcal{B} = (Q, \Sigma, \delta, q_0, Q \setminus F)$. Il linguaggio riconosciuto da $\mathcal{B}$ è il complemento $\overline{L}$ di L. Quindi il complemento $\overline{L}$ di L è regolare.

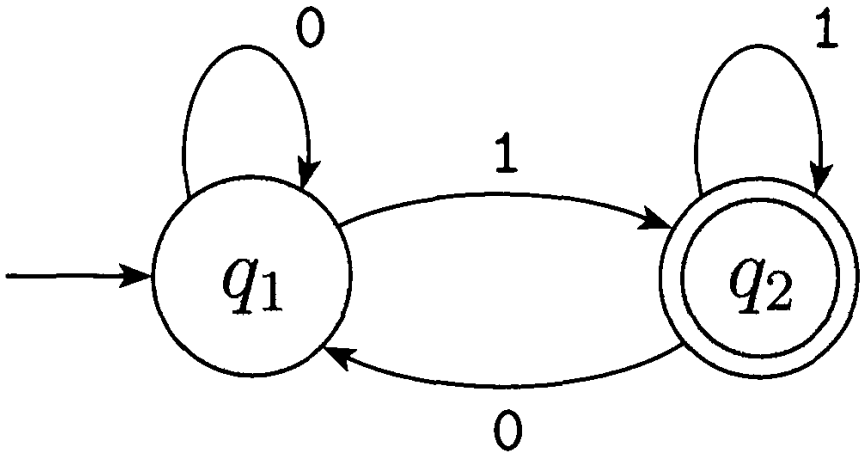


Figura: Un DFA che accetta $\{w1 \mid w \in \{0,1\}^*\}$

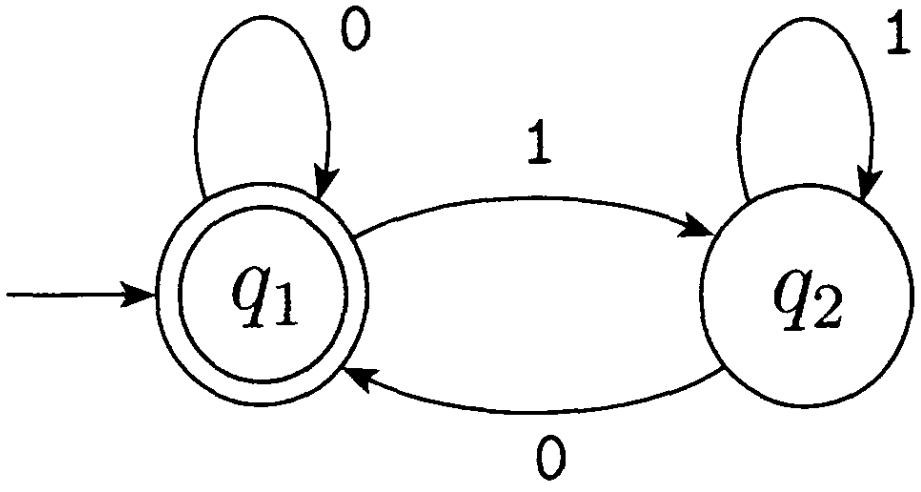


Figura: Un DFA che accetta $\{0,1\}^* \setminus \{w1 \mid w \in \{0,1\}^*\}$

Chiusura rispetto alle operazioni booleane

Teorema

La classe dei linguaggi regolari è chiusa rispetto a unione e intersezione.

Prova (Cenni) Sia X un linguaggio regolare, sia $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ un DFA che riconosce X, cioè $X = L(\mathcal{A})$. Sia Y un linguaggio regolare, sia $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ un DFA che riconosce Y, cioè $Y = L(\mathcal{A}_2)$.

Il DFA $\mathcal{A} = (Q_1 \times Q_2, \Sigma, \delta, (q_1, q_2), F)$, dove δ è definita come segue:

$$\forall r_1 \in Q_1, r_2 \in Q_2, a \in \Sigma$$

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

- riconosce $X \cap Y$ se $F = F_1 \times F_2$ (e quindi $X \cap Y$ è regolare)
- riconosce $X \cup Y$ se $F = (Q_1 \times F_2) \cup (F_1 \times Q_2)$ (e quindi $X \cup Y$ è regolare).

Chiusura rispetto alle operazioni booleane

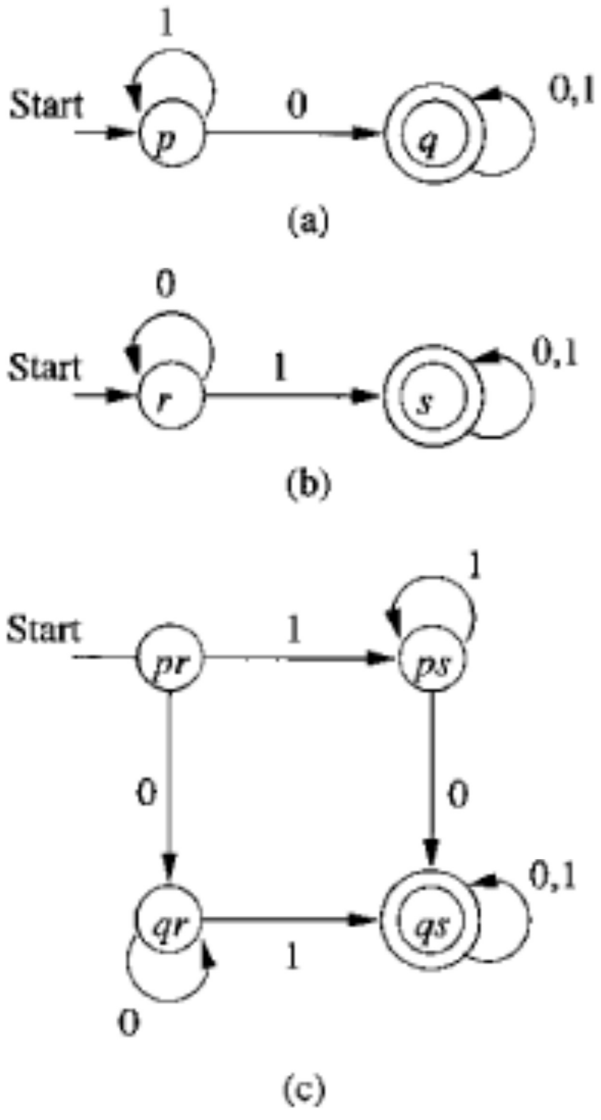


Figura: Un automa per l'intersezione di due linguaggi

Esempio. Sia $\mathcal{A}_2 = (Q, \Sigma, \delta, q_0, F)$ dove

$$\begin{aligned} Q &= \{q_0, q_1, q_2\} \\ \Sigma &= \{a, b\} \\ F &= \{q_2\} \\ \delta(q_0, a) &= q_1, \quad \delta(q_1, a) = q_1, \\ \delta(q_1, b) &= q_2, \quad \delta(q_2, b) = q_2 \end{aligned}$$

Nota. $\mathcal{A}_2$ non soddisfa la definizione di DFA perché non sono definiti né $\delta(q_0, b)$ né $\delta(q_2, a)$.

Per trasformare $\mathcal{A}_2$ in un DFA potremmo aggiungere uno stato q_3, un arco da q_0 a q_3 con etichetta b, un arco da q_2 a q_3 con etichetta a e due archi da q_3 a q_3 con etichetta a e b rispettivamente.

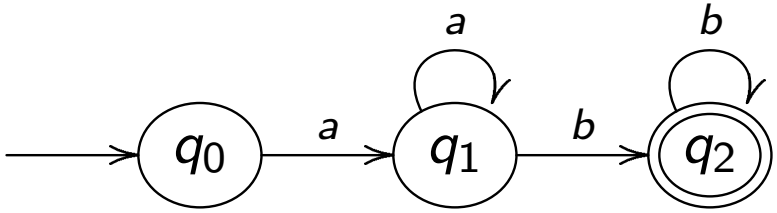


Figura: L'automa $\mathcal{A}_2$

Stato trappola

- Dato un DFA, uno stato non finale che conduce a sé stesso su ogni possibile simbolo input prende il nome di stato **trappola** o **pozzo**.

Stato trappola

- Dato un DFA, uno stato non finale che conduce a sé stesso su ogni possibile simbolo input prende il nome di stato **trappola** o **pozzo**.

- Quindi q è uno stato trappola in $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ se $q \in Q \setminus F$ e $\delta(q, \sigma) = q$, per ogni $\sigma \in \Sigma$.

Stato trappola

- Dato un DFA, uno stato non finale che conduce a sé stesso su ogni possibile simbolo input prende il nome di stato **trappola** o **pozzo**.

- Quindi q è uno stato trappola in $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ se $q \in Q \setminus F$ e $\delta(q, \sigma) = q$, per ogni $\sigma \in \Sigma$.

- Lo stato q_3 da aggiungere nell'automa $\mathcal{A}_2$ dell'esempio precedente è uno stato trappola.

Stato trappola

- Dato un automa $\mathcal{A}$ che abbia **non più** di una transizione per ogni stato e simbolo input, potremmo aggiungere uno stato trappola p e una transizione da uno stato q a p, su tutti i simboli di input per i quali q non ha transizioni. Otteniamo in questo modo un DFA $\mathcal{B}$.

Stato trappola

- Dato un automa $\mathcal{A}$ che abbia **non più** di una transizione per ogni stato e simbolo input, potremmo aggiungere uno stato trappola p e una transizione da uno stato q a p, su tutti i simboli di input per i quali q non ha transizioni. Otteniamo in questo modo un DFA $\mathcal{B}$.
- Vedremo che $\mathcal{A}$ è un automa finito non deterministico (NFA).

Stato trappola

- Dato un automa $\mathcal{A}$ che abbia **non più** di una transizione per ogni stato e simbolo input, potremmo aggiungere uno stato trappola p e una transizione da uno stato q a p, su tutti i simboli di input per i quali q non ha transizioni. Otteniamo in questo modo un DFA $\mathcal{B}$.
- Vedremo che $\mathcal{A}$ è un automa finito non deterministico (NFA).
- Possiamo rappresentare $\mathcal{A}$ mediante un diagramma. Ogni stringa è etichetta di **al più** un cammino in tale diagramma.

Stato trappola

- Dato un automa $\mathcal{A}$ che abbia **non più** di una transizione per ogni stato e simbolo input, potremmo aggiungere uno stato trappola p e una transizione da uno stato q a p, su tutti i simboli di input per i quali q non ha transizioni. Otteniamo in questo modo un DFA $\mathcal{B}$.
- Vedremo che $\mathcal{A}$ è un automa finito non deterministico (NFA).
- Possiamo rappresentare $\mathcal{A}$ mediante un diagramma. Ogni stringa è etichetta di **al più** un cammino in tale diagramma.
- Intuitivamente le stringhe che sono etichette di cammini dallo stato iniziale a uno stato finale nel diagramma di $\mathcal{B}$ lo sono anche in quello di $\mathcal{A}$.

Stato trappola

- Dato un automa $\mathcal{A}$ che abbia **non più** di una transizione per ogni stato e simbolo input, potremmo aggiungere uno stato trappola p e una transizione da uno stato q a p, su tutti i simboli di input per i quali q non ha transizioni. Otteniamo in questo modo un DFA $\mathcal{B}$.
- Vedremo che $\mathcal{A}$ è un automa finito non deterministico (NFA).
- Possiamo rappresentare $\mathcal{A}$ mediante un diagramma. Ogni stringa è etichetta di **al più** un cammino in tale diagramma.
- Intuitivamente le stringhe che sono etichette di cammini dallo stato iniziale a uno stato finale nel diagramma di $\mathcal{B}$ lo sono anche in quello di $\mathcal{A}$.
- Formalmente: l'NFA $\mathcal{A}$ è equivalente a $\mathcal{B}$, cioè $L(\mathcal{A}) = L(\mathcal{B})$.

Stato trappola

- Dato un automa $\mathcal{A}$ che abbia **non più** di una transizione per ogni stato e simbolo input, potremmo aggiungere uno stato trappola p e una transizione da uno stato q a p, su tutti i simboli di input per i quali q non ha transizioni. Otteniamo in questo modo un DFA $\mathcal{B}$.
- Vedremo che $\mathcal{A}$ è un automa finito non deterministico (NFA).
- Possiamo rappresentare $\mathcal{A}$ mediante un diagramma. Ogni stringa è etichetta di **al più** un cammino in tale diagramma.
- Intuitivamente le stringhe che sono etichette di cammini dallo stato iniziale a uno stato finale nel diagramma di $\mathcal{B}$ lo sono anche in quello di $\mathcal{A}$.
- Formalmente: l'NFA $\mathcal{A}$ è equivalente a $\mathcal{B}$, cioè $L(\mathcal{A}) = L(\mathcal{B})$.
- Per tale ragione, molti autori si riferiscono ad automi come $\mathcal{A}$ come a un DFA con stato trappola omesso.

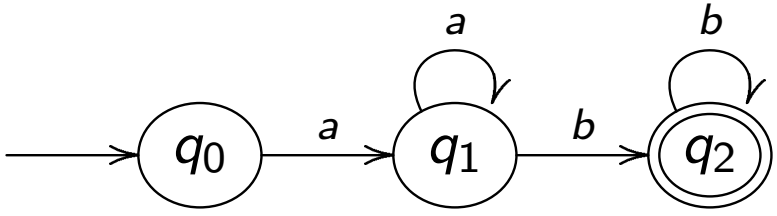


Figura: $\mathcal{A}_2$ accetta $\{a^n b^m \mid n, m \geq 1\}$

Stato trappola

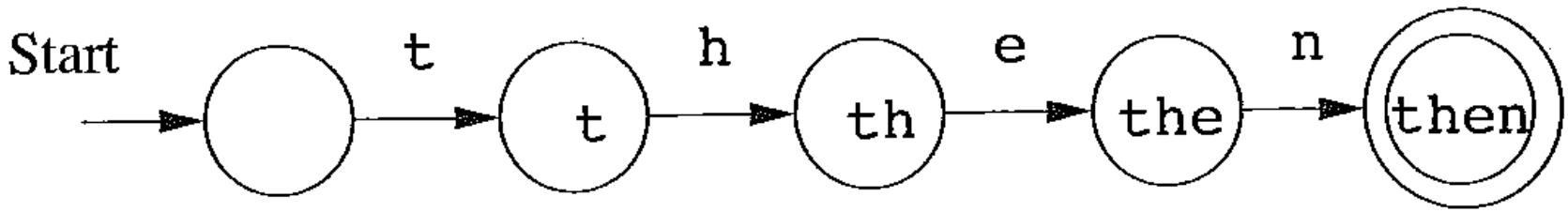


Figura: Un automa che accetta $\{then\}$

Automi finiti non deterministici

Alle volte può essere comodo considerare automi in cui ci siano alcune transizioni che da certi stati portano con lo *stesso* simbolo in più di uno stato.

Infatti è più facile definire in questo modo l'automa per il linguaggio di interesse e la definizione è spesso più concisa.

Definiamo quindi gli automi a stati finiti non deterministici.

Proveremo che tale estensione non modifica la classe dei linguaggi riconosciuti.
Vedremo un algoritmo per ottenere un automa deterministico da uno non deterministico.

Ricordiamo che, dato un insieme X, l'insieme $\mathcal{P}(X)$ è l'**insieme potenza** di X, ovvero la collezione di tutti i sottoinsiemi di X.

Definizione formale di un NFA

Definizione (Automa a stati finiti non deterministico)

Un automa a stati finiti non deterministico (in breve NFA) è una quintupla

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$$

dove:

- Q *è l'insieme* finito *degli stati;*
- Σ *è l'alfabeto (finito);*
- $q_0 \in Q$ *è lo stato iniziale*
- $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ *è la funzione di transizione, con* $\Sigma_\epsilon = \Sigma \cup \{\epsilon\}$
- $F \subseteq Q$ *è l'insieme degli stati finali.*

Notazioni per DFA

Useremo per rappresentare un NFA le stesse convenzioni adottate per i DFA.

Nel diagramma di stato vi può essere più di un arco con la stessa etichetta che porta da uno stato ad altri.

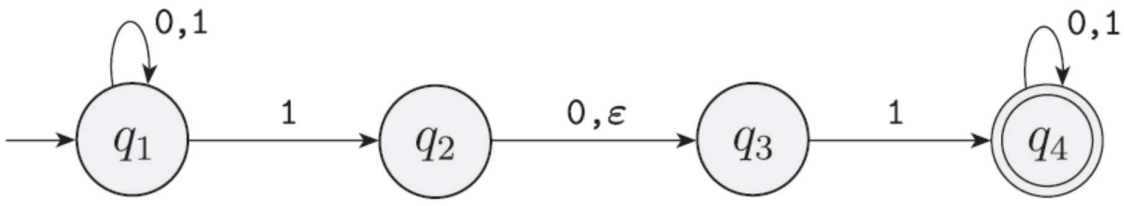


Figura: L'automa finito non deterministico N_1

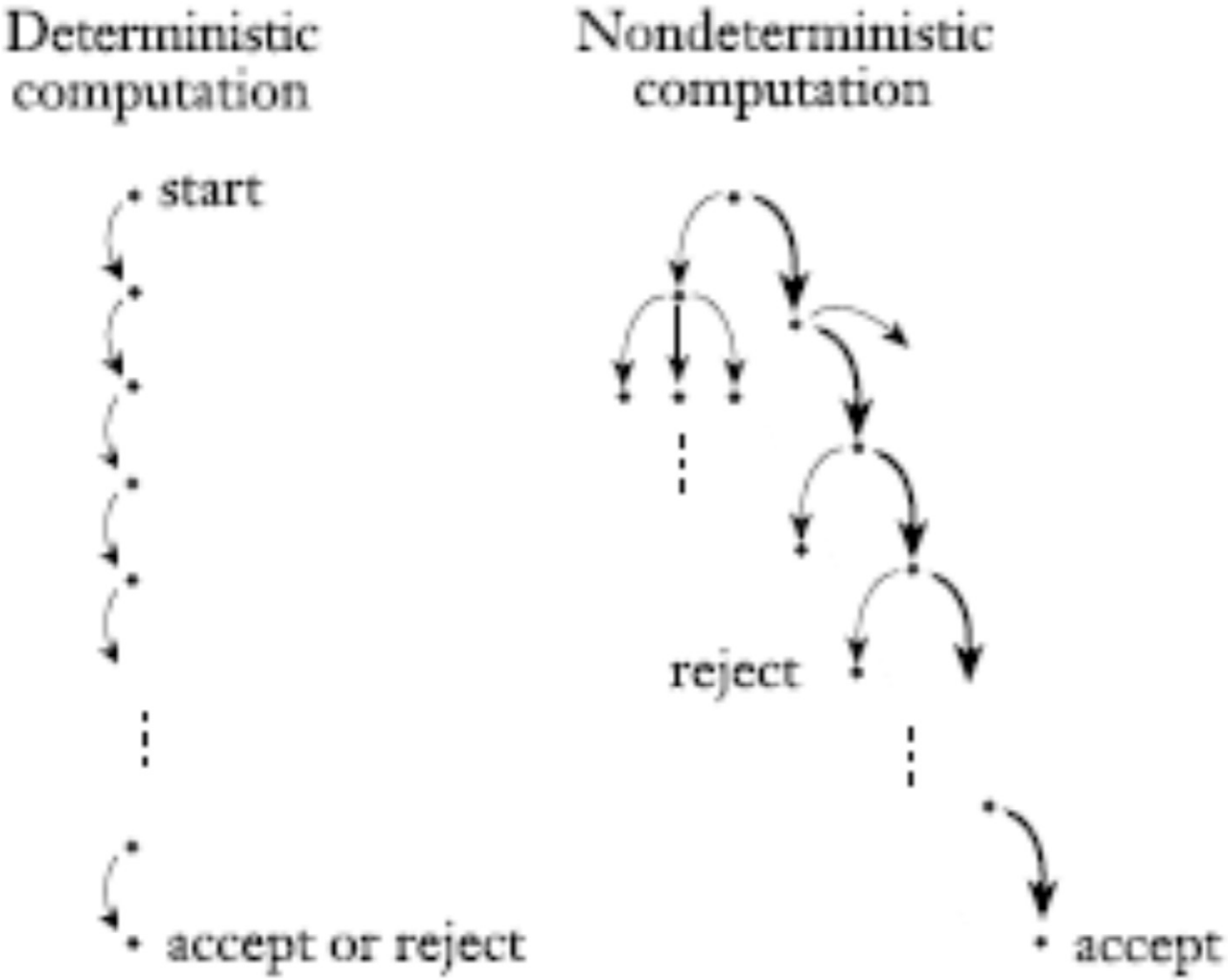


Figura: Computazione in un DFA e in un NFA

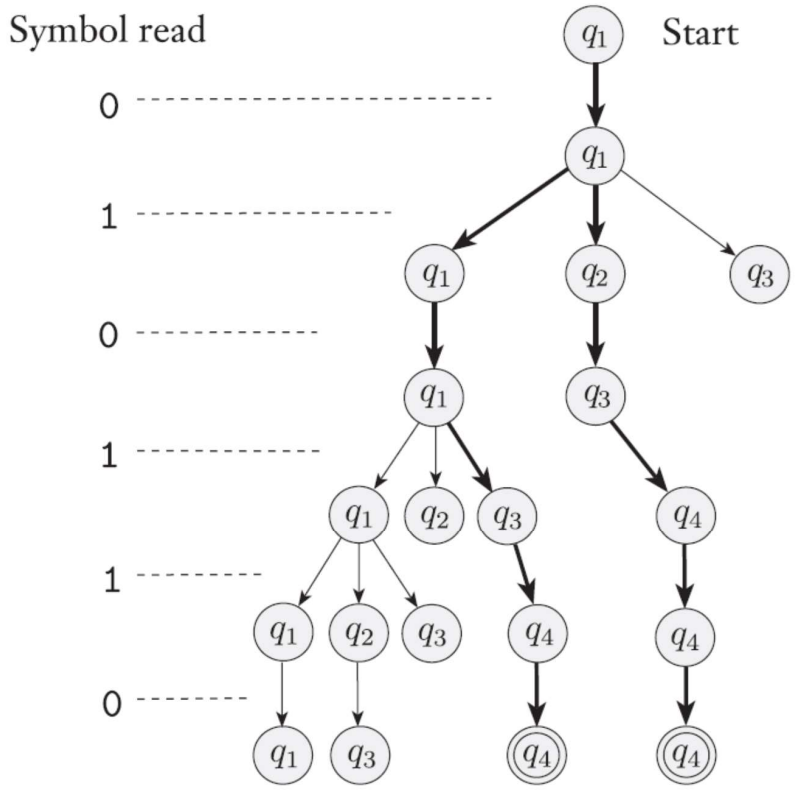


Figura: La computazione di M_1 sull'input 010110

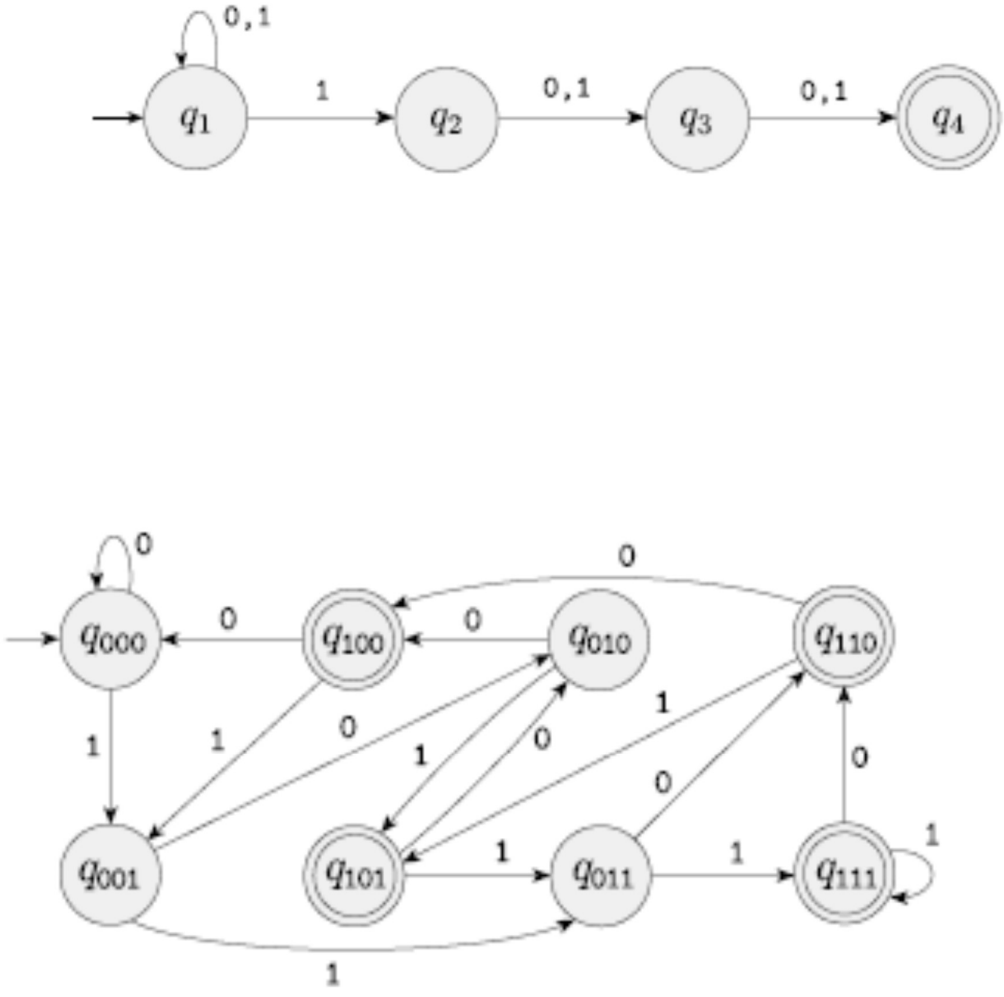


Figura: Un NFA e un DFA che riconoscono lo stesso linguaggio

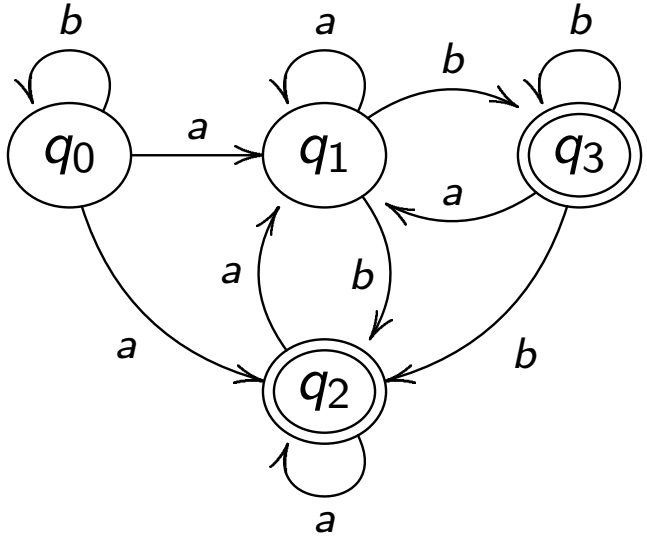


Figura: Un automa a stati finiti non deterministico con stato iniziale q_0

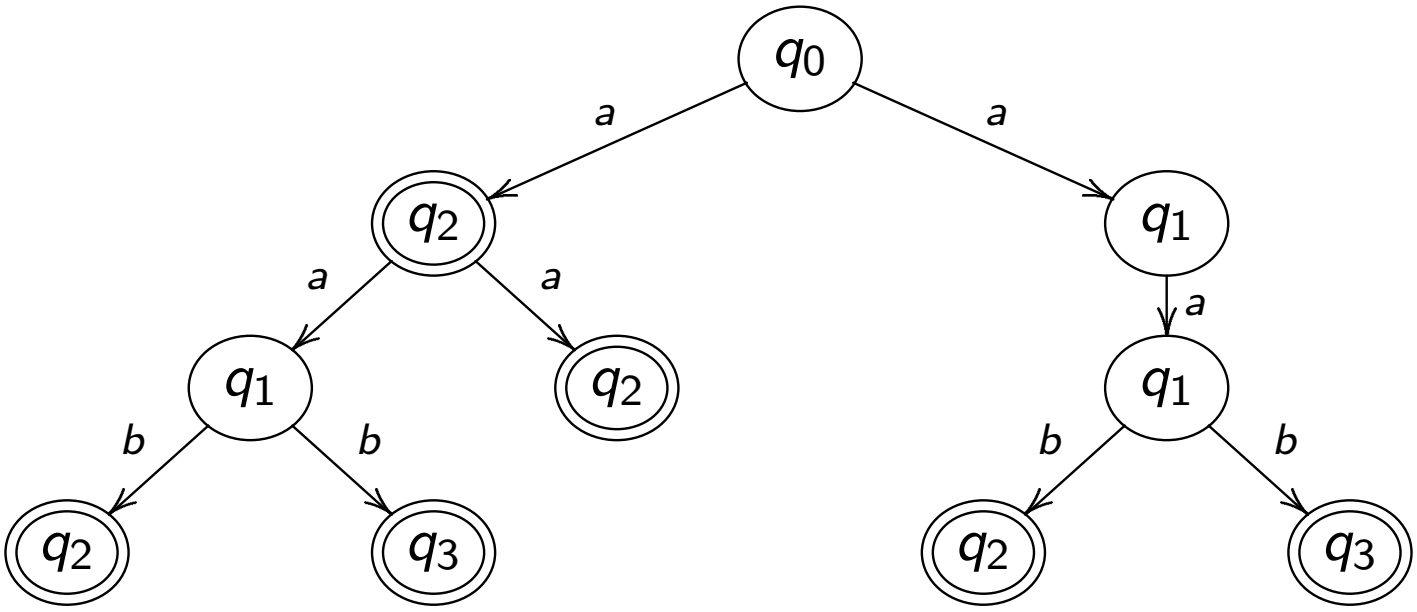


Figura: I cammini accettanti dell'NFA della precedente figura su *aab*, organizzati in un albero (che include i cammini che accettano *a* e *aa*).

Definizione

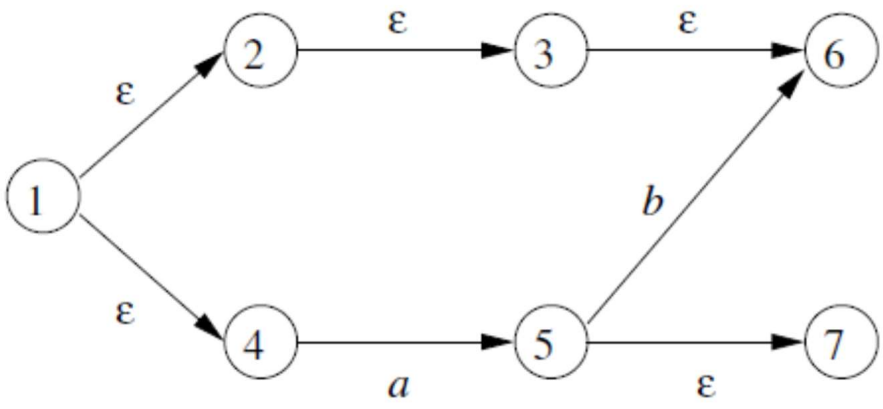
Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un automa finito non deterministico, sia $q \in Q$. La ϵ-chiusura $E(q)$ di q è un sottoinsieme di Q definito ricorsivamente come segue

Passo Base	$q \in E(q)$
Passo Ricorsivo	$\forall p \in E(q), \quad \delta(p, \epsilon) \subseteq E(q)$

Sia $R \subseteq Q$. La ϵ-chiusura $E(R)$ di R è

$$E(R) = \cup_{q \in R} E(q)$$

$E(1) = \{1, 2, 3, 4, 6\}$, $E(2) = \{2, 3, 6\}$, $E(3) = \{3, 6\}$, $E(4) = \{4\}$, $E(5) = \{5, 7\}$, $E(6) = \{6\}$, $E(7) = \{7\}$



Funzione di transizione estesa di un NFA

Definizione

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *un automa finito non deterministico. La funzione di transizione estesa* $\hat{\delta}$: $Q \times \Sigma^* \to \mathcal{P}(Q)$ *è definita ricorsivamente come segue*

$$
\begin{aligned}
\textbf{Passo Base} \qquad & \forall q \in Q, \\
\hat{\delta}(q, \epsilon) &= E(q) \\
\textbf{Passo Ricorsivo} \qquad & \forall q \in Q, w \in \Sigma^*, a \in \Sigma, \\
\hat{\delta}(q, w\,a) &= E\Big(\bigcup_{p \in \hat{\delta}(q,w)} \delta(p, a)\Big) \\
&= \bigcup_{p \in \hat{\delta}(q,w)} E(\delta(p, a))
\end{aligned}
$$

Linguaggio accettato da un NFA

Definizione

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *un automa finito non deterministico. Il linguaggio* **accettato o riconosciuto da** $\mathcal{A}$ **è**

$$L(\mathcal{A}) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$

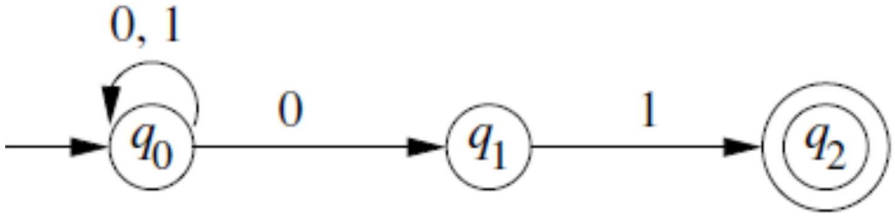


Figura: Un NFA che accetta $\{x01 \mid x \in \{0,1\}^*\}$

Teorema

Per ogni automa finito non deterministico $\mathcal{A}$ esiste un automa finito deterministico $\mathcal{B}$ tale che $L(\mathcal{A}) = L(\mathcal{B})$.

Prova (Cenni) Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un NFA che riconosce L, cioè $L = L(\mathcal{A})$. Costruiamo un DFA $\mathcal{B} = (Q', \Sigma, \delta', q_0', F')$ tale che $L(\mathcal{A}) = L(\mathcal{B})$.

1. $Q' = \mathcal{P}(Q)$.
   Ogni stato di $\mathcal{B}$ è un insieme di stati di $\mathcal{A}$.
2. $\forall R \in Q', a \in \Sigma$

$$\delta'(R, a) = \cup_{r \in R} E(\delta(r, a)) = E(\cup_{r \in R} \delta(r, a))$$

3. $q_0' = E(\{q_0\})$
4. $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$

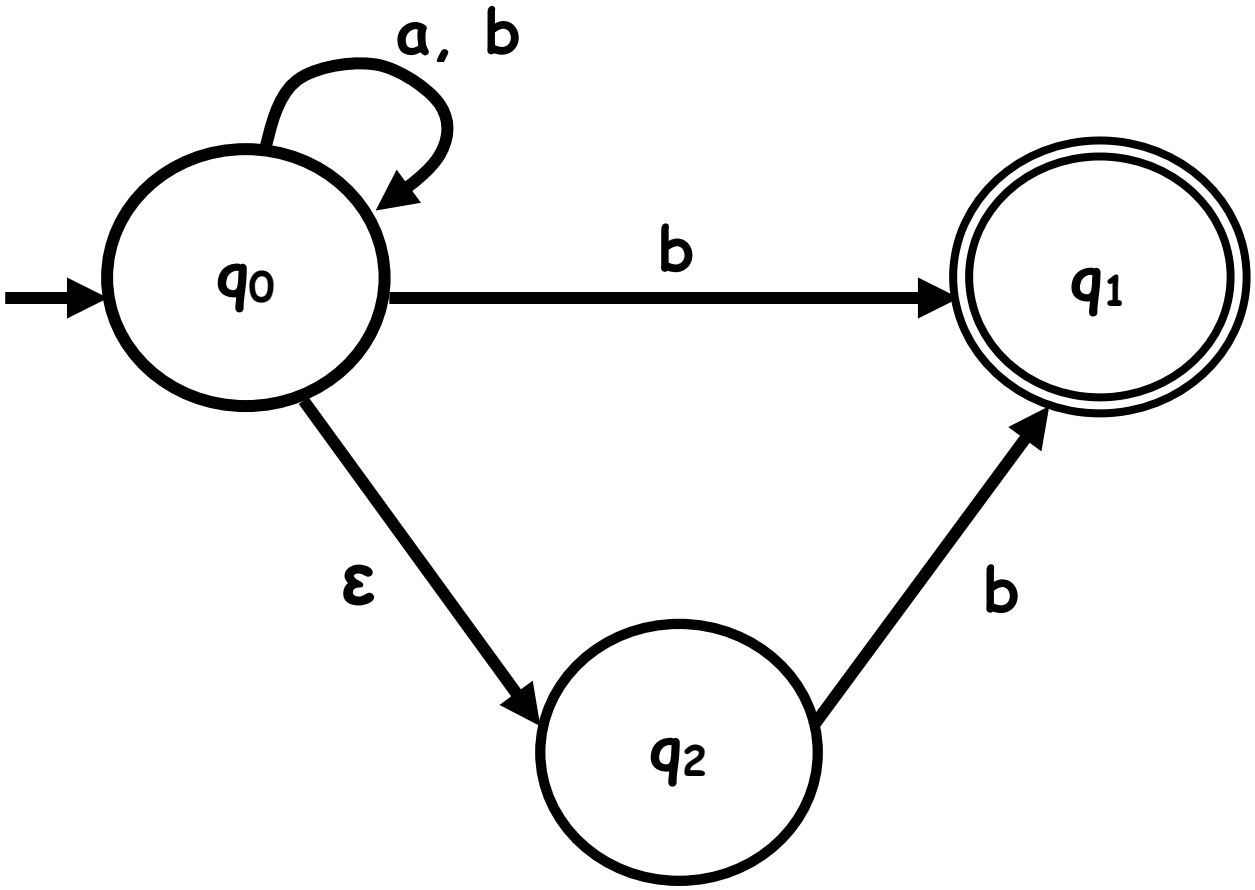


Figura: Automa $\mathcal{A}$.

DFA $\mathcal{B}$ equivalente ad $\mathcal{A}$

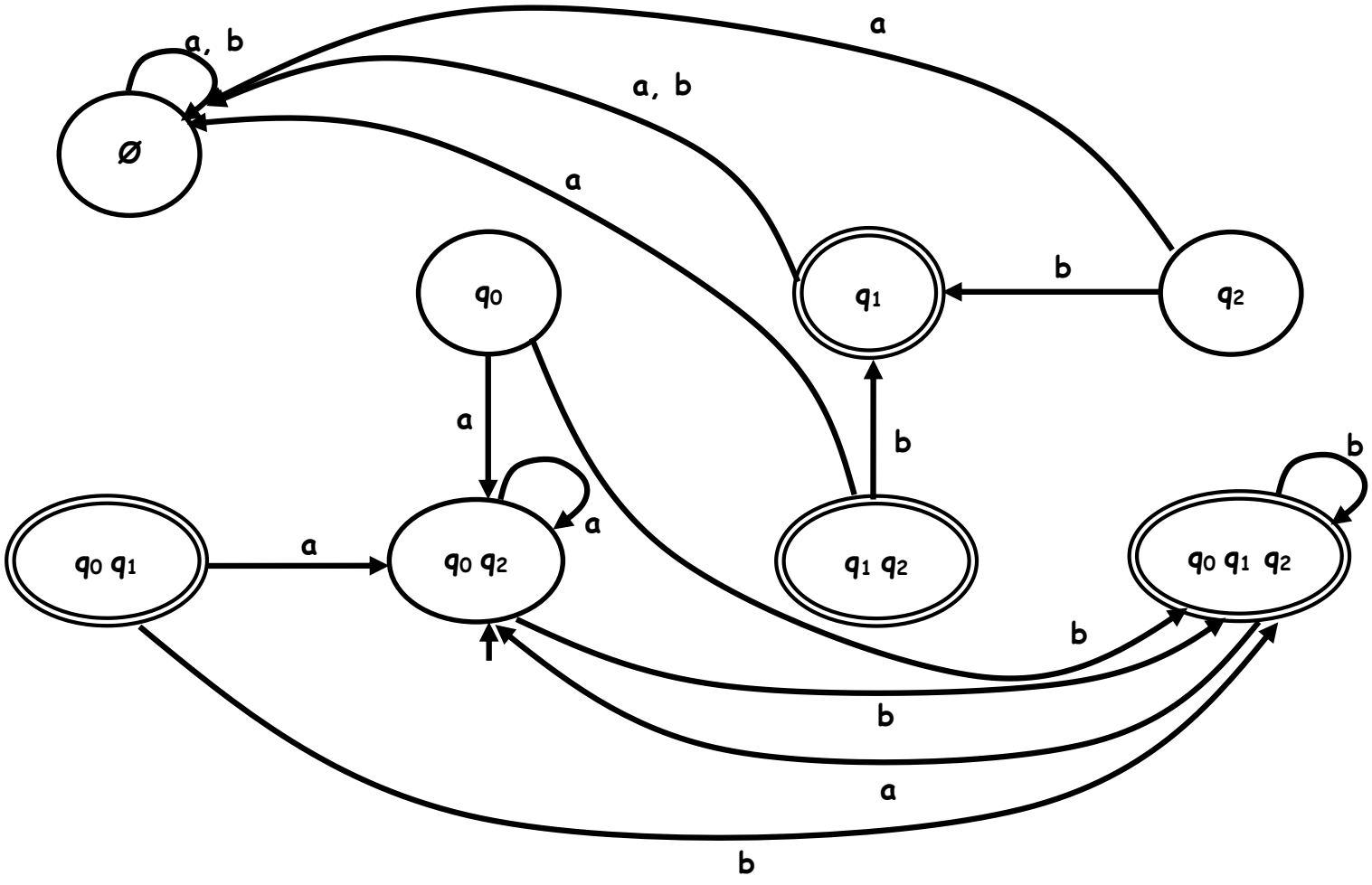


Figura: DFA $\mathcal{B}$ equivalente ad $\mathcal{A}$.

DFA equivalente ad $\mathcal{B}$

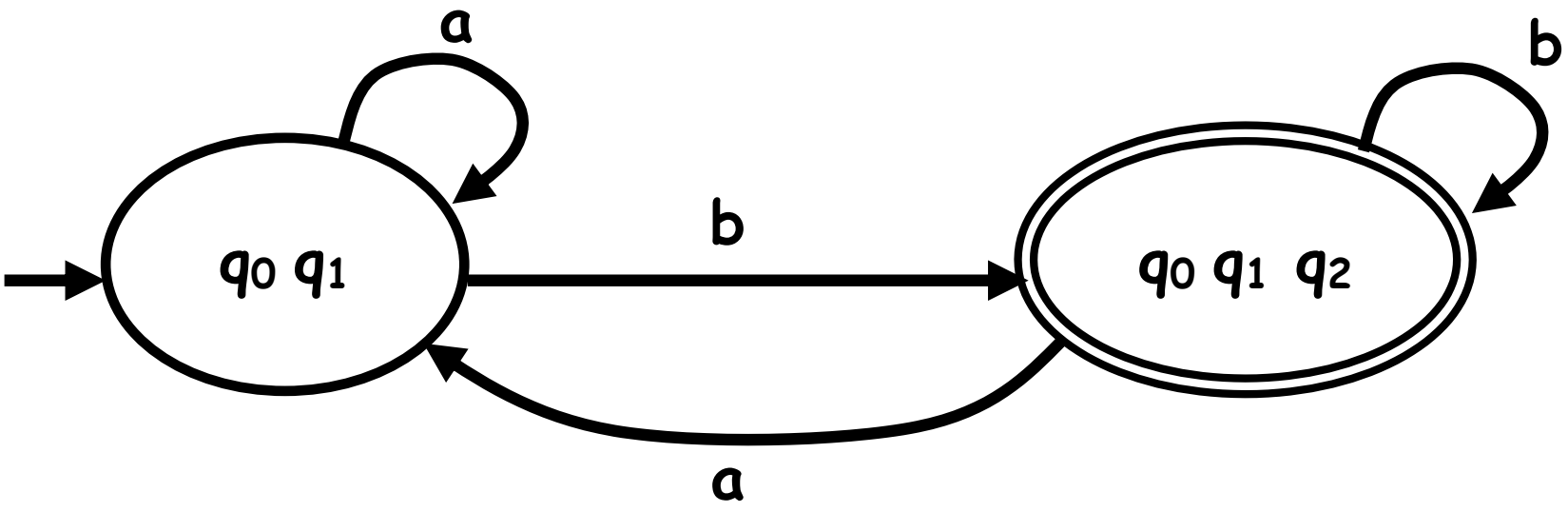


Figura: DFA equivalente a $\mathcal{B}$.

Equivalenza NFA e DFA

- Un algoritmo che usa la **costruzione per sottoinsiemi**, fornita dal teorema, per definire un DFA equivalente a un NFA assegnato $\mathcal{A}$ richiede tempo esponenziale (nella lunghezza di una codifica di $\mathcal{A}$).

Equivalenza NFA e DFA

- Un algoritmo che usa la **costruzione per sottoinsiemi**, fornita dal teorema, per definire un DFA equivalente a un NFA assegnato $\mathcal{A}$ richiede tempo esponenziale (nella lunghezza di una codifica di $\mathcal{A}$).
- La costruzione richiede un DFA con un numero di stati pari al numero dei sottoinsiemi dell'insieme degli stati di $\mathcal{A}$. Poi occorre costruire tutte le transizioni tra tali stati.

Equivalenza NFA e DFA

- Un algoritmo che usa la **costruzione per sottoinsiemi**, fornita dal teorema, per definire un DFA equivalente a un NFA assegnato $\mathcal{A}$ richiede tempo esponenziale (nella lunghezza di una codifica di $\mathcal{A}$).
- La costruzione richiede un DFA con un numero di stati pari al numero dei sottoinsiemi dell'insieme degli stati di $\mathcal{A}$. Poi occorre costruire tutte le transizioni tra tali stati.
- In alcuni casi, come nell'esempio precedente, possiamo omettere alcuni di tali stati utilizzando una variante di tale algoritmo.

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un NFA.

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un NFA.

Definiamo particolari sottoinsiemi di Q che chiameremo accessibili.

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un NFA.

Definiamo particolari sottoinsiemi di Q che chiameremo accessibili.

- **Passo base.** $E(\{q_0\})$ è accessibile.

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un NFA.

Definiamo particolari sottoinsiemi di Q che chiameremo accessibili.

- **Passo base.** $E(\{q_0\})$ è accessibile.
- **Passo ricorsivo.** Se $R \subseteq Q$ è accessibile allora per ogni $a \in \Sigma$, $R' = \cup_{r \in R} E(\delta(r, a))$ è accessibile.

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un NFA.

Definiamo particolari sottoinsiemi di Q che chiameremo accessibili.

- **Passo base.** $E(\{q_0\})$ è accessibile.
- **Passo ricorsivo.** Se $R \subseteq Q$ è accessibile allora per ogni $a \in \Sigma$, $R' = \cup_{r \in R} E(\delta(r, a))$ è accessibile.

Sia $\mathcal{P}' = \{R \in \mathcal{P}(Q) \mid R \text{ è accessibile}\}$

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un NFA. Sia $\mathcal{B} = (Q', \Sigma, \delta', q_0', F')$, dove

1. $Q' = \mathcal{P}' = \{R \in \mathcal{P}(Q) \mid R \text{ è accessibile}\}$
2. $\forall R \in Q', a \in \Sigma$

$$\delta'(R, a) = \cup_{r \in R} E(\delta(r, a)) = E(\cup_{r \in R} \delta(r, a))$$

3. $q_0' = E(\{q_0\})$
4. $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$

Sia $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un NFA. Sia $\mathcal{B} = (Q', \Sigma, \delta', q_0', F')$, dove

1. $Q' = \mathcal{P}' = \{R \in \mathcal{P}(Q) \mid R \text{ è accessibile}\}$
2. $\forall R \in Q', a \in \Sigma$

$$\delta'(R, a) = \cup_{r \in R} E(\delta(r, a)) = E(\cup_{r \in R} \delta(r, a))$$

3. $q_0' = E(\{q_0\})$
4. $F' = \{R \in Q' \mid R \cap F \neq \emptyset\}$

L'automa $\mathcal{B}$ è un DFA equivalente ad $\mathcal{A}$.

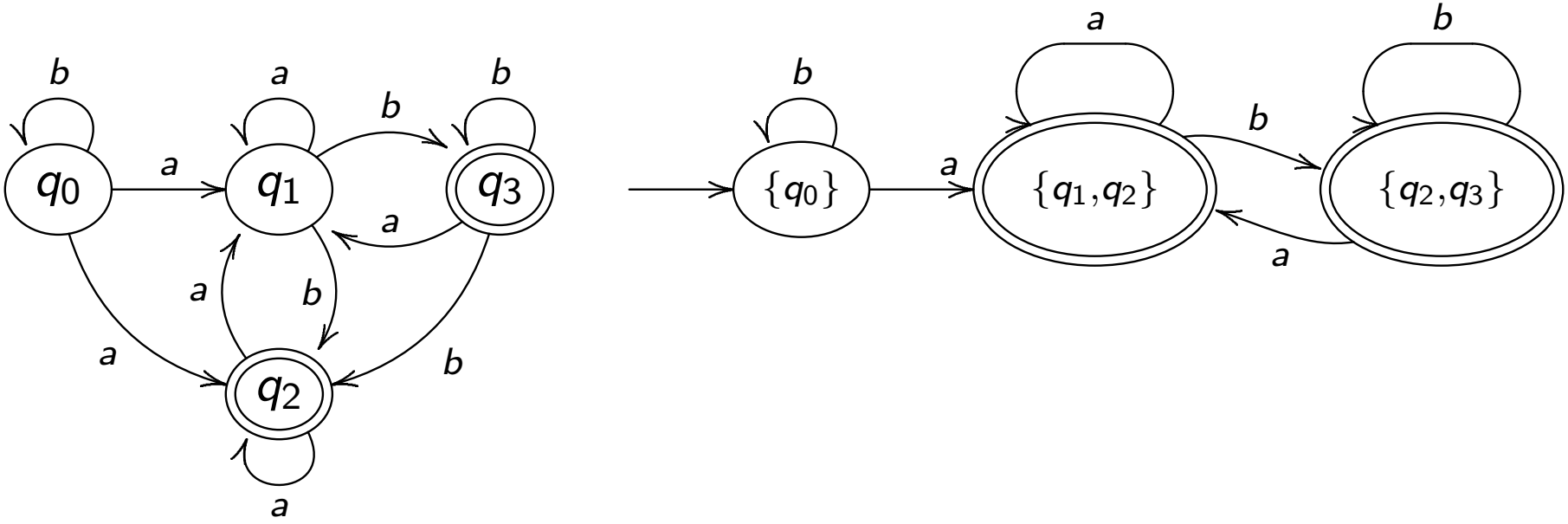


Figura: Un automa a stati finiti nondeterministico con stato iniziale q_0, nella parte sinistra, e il suo equivalente deterministico, in quella destra

Equivalenza NFA e DFA

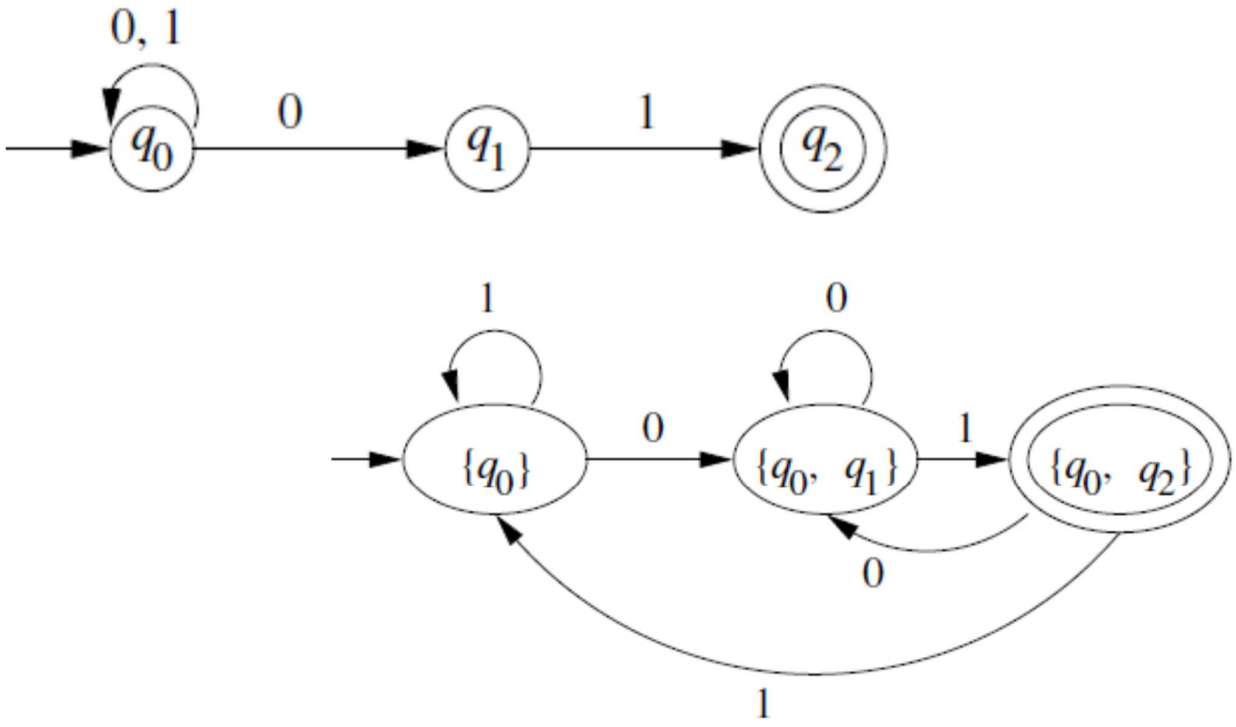


Figura: Un NFA che accetta $\{x01 \mid x \in \{0,1\}^*\}$ e il suo equivalente DFA

Caso pessimo

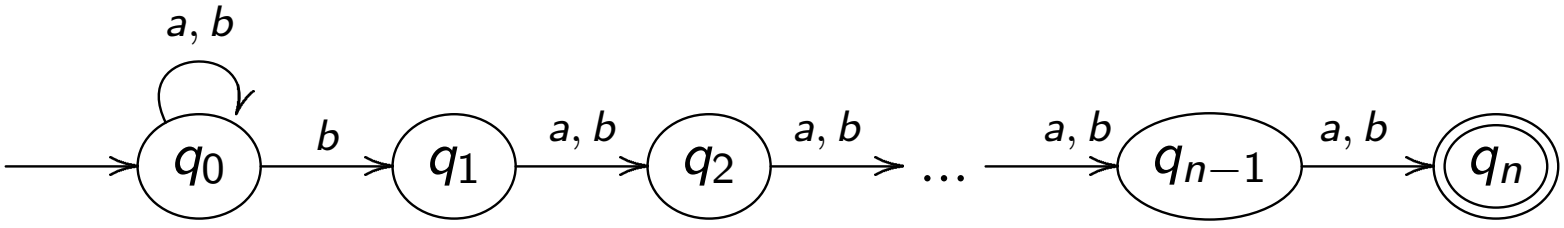


Figura: Un NFA che non ha DFA equivalente con meno di 2^n stati

Corollario

Un linguaggio è regolare se e solo se esiste un automa finito non deterministico che lo riconosce.

Espressioni regolari

C'è un terzo modo per esprimere i linguaggi regolari, che va sotto il nome di *espressioni regolari*. Le espressioni regolari offrono notevoli vantaggi di scrittura, e possono essere manipolate in base ad alcune loro proprietà algebriche senza modificare i linguaggi che rappresentano.
Le espressioni regolari sono un utile strumento nel progetto dei compilatori per i linguaggi di programmazione. Gli oggetti elementari in un linguaggio di programmazione, chiamati *token*, come i nomi delle variabili e le costanti, possono essere descritti con espressioni regolari.
Una volta che la sintassi di un linguaggio di programmazione è stata descritta usando espressioni regolari per i suoi token, dei sistemi automatici possono generare l'analizzatore lessicale, la parte di un compilatore che elabora inizialmente il programma input.

Espressioni regolari

Definizione ricorsiva delle espressioni regolari su un alfabeto Σ:

Definizione ricorsiva delle espressioni regolari su un alfabeto Σ:

PASSO BASE:

Per ogni $a \in \Sigma$, a è un'espressione regolare;

ϵ è un'espressione regolare;

$\emptyset$ è un'espressione regolare.

Definizione ricorsiva delle espressioni regolari su un alfabeto Σ:

PASSO BASE:
Per ogni $a \in \Sigma$, a è un'espressione regolare;
ϵ è un'espressione regolare;
$\emptyset$ è un'espressione regolare.

PASSO RICORSIVO: Se E_1, E_2 sono espressioni regolari, allora
(E_1) è un'espressione regolare;
$(E_1 + E_2)$ è un'espressione regolare;
$(E_1 E_2)$ è un'espressione regolare;
(E_1^*) è un'espressione regolare.

Definizione ricorsiva dei linguaggi rappresentati dalle espressioni regolari su un alfabeto Σ.

Data un'espressione regolare E, indicheremo con $L(E)$ il linguaggio che essa rappresenta, definito come segue

Definizione ricorsiva dei linguaggi rappresentati dalle espressioni regolari su un alfabeto Σ.

Data un'espressione regolare E, indicheremo con $L(E)$ il linguaggio che essa rappresenta, definito come segue

PASSO BASE:

Per ogni $a \in \Sigma$, $L(a) = \{a\}$;
$L(\epsilon) = \{\epsilon\}$;
$L(\emptyset) = \emptyset$.

Definizione ricorsiva dei linguaggi rappresentati dalle espressioni regolari su un alfabeto Σ.

Data un'espressione regolare E, indicheremo con $L(E)$ il linguaggio che essa rappresenta, definito come segue

PASSO BASE:

Per ogni $a \in \Sigma$, $L(a) = \{a\}$;
$L(\epsilon) = \{\epsilon\}$;
$L(\emptyset) = \emptyset$.

PASSO RICORSIVO: Se E_1, E_2 sono espressioni regolari, allora

$L((E_1)) = L(E_1)$;
$L(E_1 + E_2) = L(E_1) \cup L(E_2)$;
$L(E_1 E_2) = L(E_1)L(E_2)$;
$L(E_1^*) = L(E_1)^*$.

Regole di precedenza degli operatori

Operatore	precedenza
$*$	1
$\cdot$	2
$+$	3

Regole di precedenza degli operatori

Operatore	precedenza
$*$	1
$\cdot$	2
$+$	3

L'espressione regolare $ab^* + b$ corrisponde a $(a(b)^*) + b$ ed è diversa da $(ab)^* + b$.

Regole di precedenza degli operatori

Operatore	precedenza
$*$	1
$\cdot$	2
$+$	3

L'espressione regolare $ab^* + b$ corrisponde a $(a(b)^*) + b$ ed è diversa da $(ab)^* + b$.

L'espressione regolare $ab^* + b$ è anche diversa da $a(b^* + b)$.

Teorema (Kleene)

Un linguaggio è regolare se e solo se esiste un'espressione regolare che lo rappresenta.

Prova (Cenni): dall'espressione regolare all'automa
Supponiamo di avere un'espressione regolare E che descrive un linguaggio L. Mostriamo come trasformare E in un NFA che riconosce L.
È una dimostrazione per induzione strutturale.

PASSO BASE:

Prova (Cenni): dall'espressione regolare all'automa
Supponiamo di avere un'espressione regolare E che descrive un linguaggio L. Mostriamo come trasformare E in un NFA che riconosce L.
È una dimostrazione per induzione strutturale.

PASSO BASE:

Se $E = a$, con $a \in \Sigma$ allora $L(a) = \{a\}$. Risulta $L(E) = L(\mathcal{A})$, $\mathcal{A} = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, dove δ è definita da $\delta(q_1, a) = \{q_2\}$ e $\delta(r, b) = \emptyset$ per $r \neq q_1$ o $b \neq a$;

Prova (Cenni): dall'espressione regolare all'automa

Supponiamo di avere un'espressione regolare E che descrive un linguaggio L. Mostriamo come trasformare E in un NFA che riconosce L.

È una dimostrazione per induzione strutturale.

PASSO BASE:

Se $E = a$, con $a \in \Sigma$ allora $L(a) = \{a\}$. Risulta $L(E) = L(\mathcal{A})$, $\mathcal{A} = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, dove δ è definita da $\delta(q_1, a) = \{q_2\}$ e $\delta(r, b) = \emptyset$ per $r \neq q_1$ o $b \neq a$;

Se $E = \epsilon$, allora $L(\epsilon) = \{\epsilon\}$. Risulta $L(E) = L(\mathcal{A})$, $\mathcal{A} = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$, dove $\delta(q_1, b) = \emptyset$ per ogni b;

Prova (Cenni): dall'espressione regolare all'automa

Supponiamo di avere un'espressione regolare E che descrive un linguaggio L. Mostriamo come trasformare E in un NFA che riconosce L.

È una dimostrazione per induzione strutturale.

PASSO BASE:

Se $E = a$, con $a \in \Sigma$ allora $L(a) = \{a\}$. Risulta $L(E) = L(\mathcal{A})$, $\mathcal{A} = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, dove δ è definita da $\delta(q_1, a) = \{q_2\}$ e $\delta(r, b) = \emptyset$ per $r \neq q_1$ o $b \neq a$;

Se $E = \epsilon$, allora $L(\epsilon) = \{\epsilon\}$. Risulta $L(E) = L(\mathcal{A})$, $\mathcal{A} = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$, dove $\delta(q_1, b) = \emptyset$ per ogni b;

Se $E = \emptyset$, allora $L(\emptyset) = \emptyset$. Risulta $L(E) = L(\mathcal{A})$, $\mathcal{A} = (\{q_1\}, \Sigma, \delta, q_1, \emptyset)$, dove $\delta(q_1, b) = \emptyset$ per ogni b.

Teorema di Kleene: passo base

Figura: Passo Base

PASSO INDUTTIVO:

PASSO INDUTTIVO:

Siano E_1, E_2 espressioni regolari. Allora
$L(E_1 + E_2) = L(E_1) \cup L(E_2)$;
$L(E_1 E_2) = L(E_1)L(E_2)$;
$L(E_1^*) = L(E_1)^*$
sono linguaggi regolari.

Teorema

La classe dei linguaggi regolari è chiusa rispetto a unione, concatenazione e star.

Prova (Cenni) Sia X un linguaggio regolare, sia $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ un NFA che riconosce X, cioè $X = L(\mathcal{A})$. Sia Y un linguaggio regolare, sia $\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ un NFA che riconosce Y, cioè $Y = L(\mathcal{A}_2)$.

Teorema di Kleene - passo induttivo

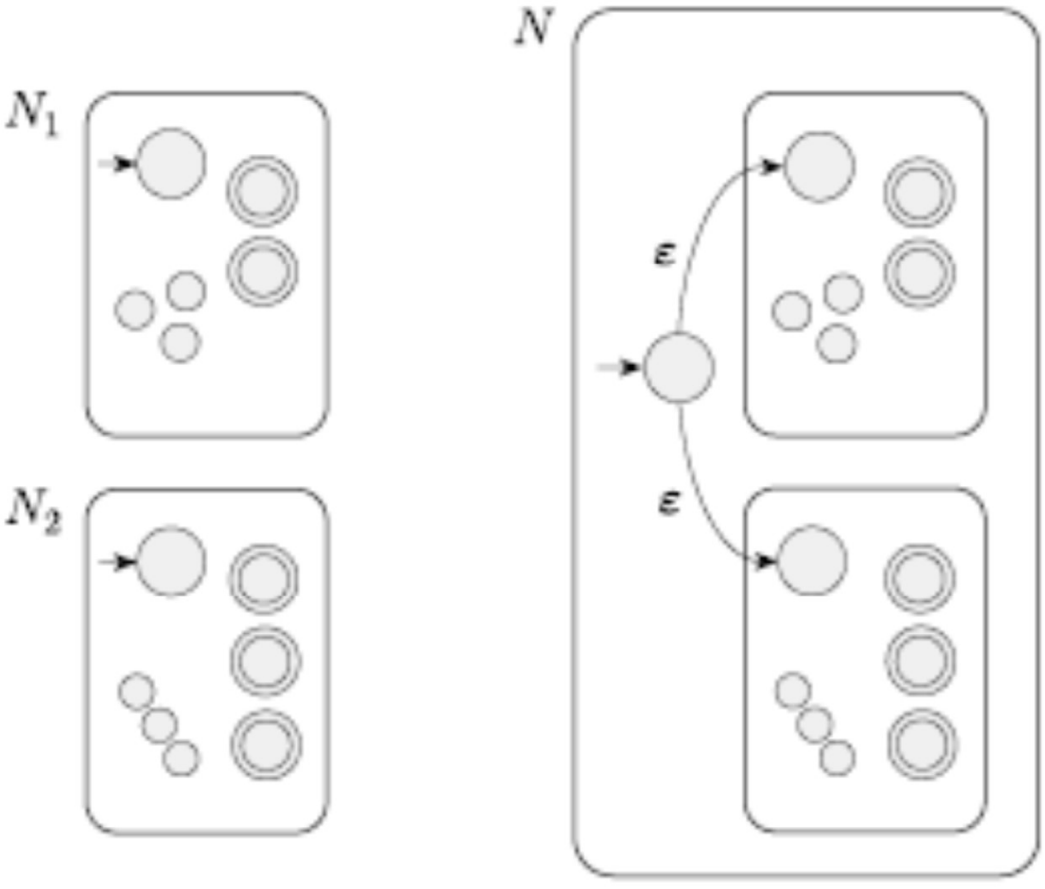


Figura: chiusura rispetto all'unione

Costruiamo $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ per riconoscere $X \cup Y$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
   Gli stati di $\mathcal{A}$ sono tutti gli stati di $\mathcal{A}_1$ e $\mathcal{A}_2$, con l'aggiunta di un nuovo stato iniziale q_0.
2. Lo stato q_0 è lo stato iniziale di $\mathcal{A}$.
3. L'insieme degli stati finali $F = F_1 \cup F_2$.
   Gli stati accettanti di $\mathcal{A}$ sono tutti gli stati accettanti di $\mathcal{A}_1$ e $\mathcal{A}_2$.
4. Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\epsilon$,

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

Teorema di Kleene - passo induttivo

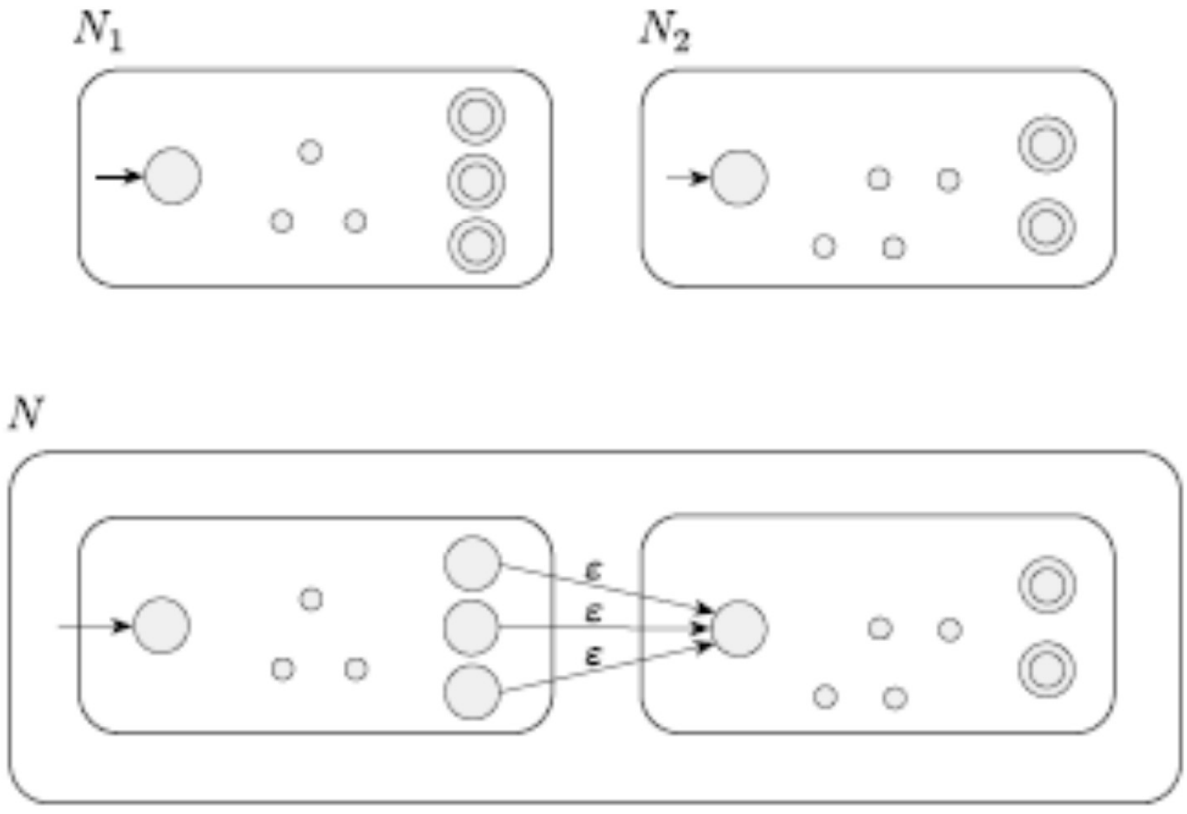


Figura: Chiusura rispetto alla concatenazione

Costruiamo $\mathcal{A} = (Q, \Sigma, \delta, q_1, F_2)$ per riconoscere $X \cdot Y$.

1. $Q = Q_1 \cup Q_2$.
   Gli stati di $\mathcal{A}$ sono tutti gli stati di $\mathcal{A}_1$ e $\mathcal{A}_2$.
2. Lo stato q_1 è uguale allo stato iniziale di $\mathcal{A}_1$.
3. L'insieme degli stati finali F_2 di $\mathcal{A}$ è uguale all'insieme degli stati finali di $\mathcal{A}_2$.
4. Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\epsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{se } q \in Q_1 \text{ e } q \notin F_1 \\ \delta_1(q,a) & \text{se } q \in F_1 \text{ e } a \neq \epsilon \\ \delta_1(q,a) \cup \{q_2\} & \text{se } q \in F_1 \text{ e } a = \epsilon \\ \delta_2(q,a) & \text{se } q \in Q_2. \end{cases}$$

Teorema di Kleene - passo induttivo

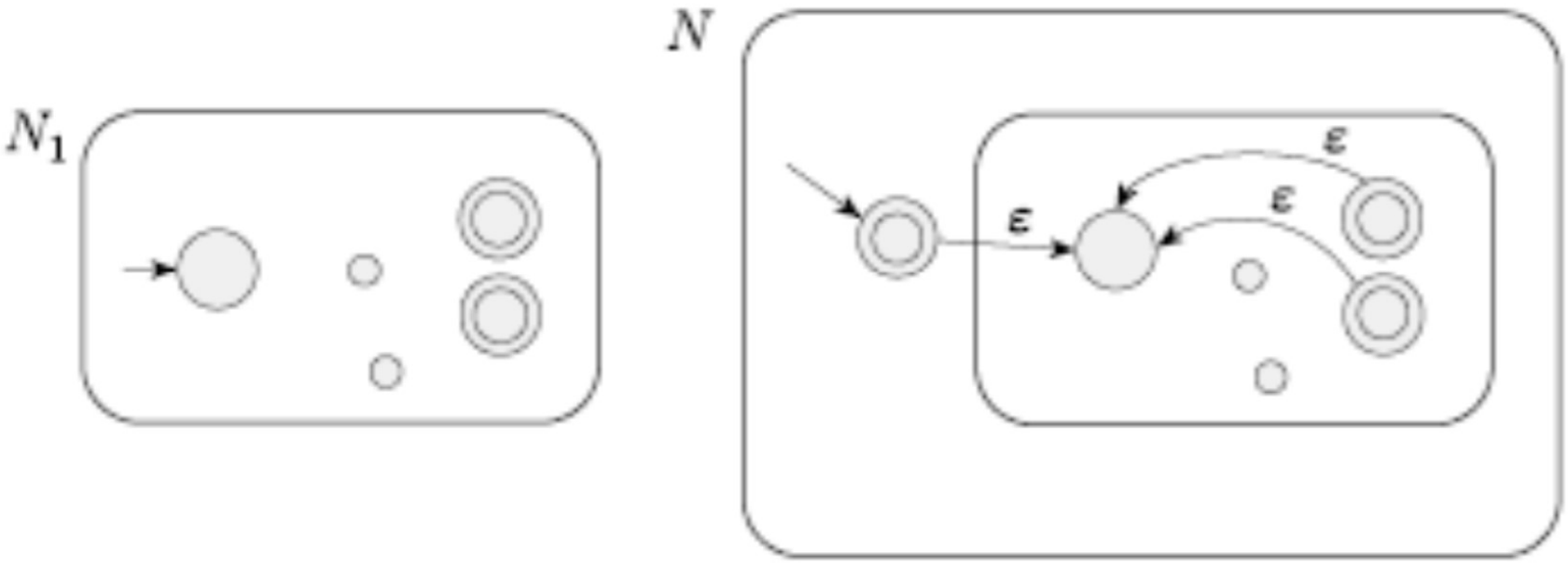


Figura: Chiusura rispetto allo star.

Costruiamo $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ per riconoscere X^*.

1. $Q = \{q_0\} \cup Q_1$.
   Gli stati di $\mathcal{A}$ sono gli stati di $\mathcal{A}_1$ più un nuovo stato q_0.
2. Lo stato q_0 è il nuovo stato iniziale.
3. $F = \{q_0\} \cup F_1$.
4. Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\epsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{se } q \in Q_1 \text{ e } q \notin F_1 \\ \delta_1(q,a) & \text{se } q \in F_1 \text{ e } a \neq \epsilon \\ \delta_1(q,a) \cup \{q_1\} & \text{se } q \in F_1 \text{ e } a = \epsilon \\ \{q_1\} & \text{se } q = q_0 \text{ e } a = \epsilon \\ \emptyset & \text{se } q = q_0 \text{ e } a \neq \epsilon \end{cases}$$

Automa che riconosce X

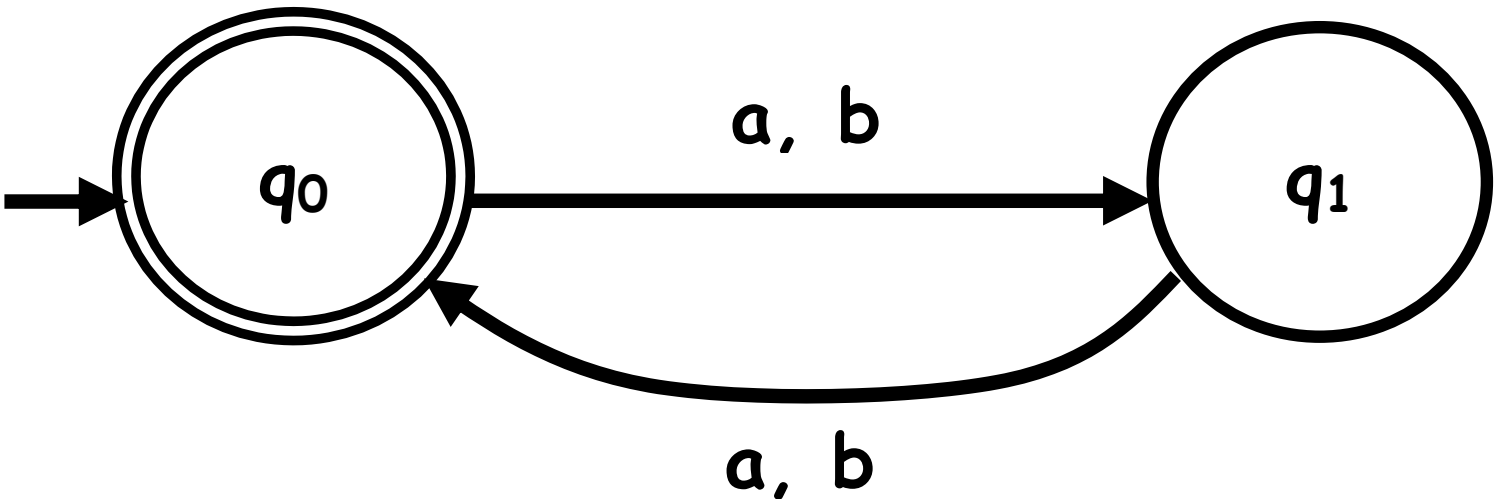


Figura: Automa che riconosce X

Automa che riconosce X^*

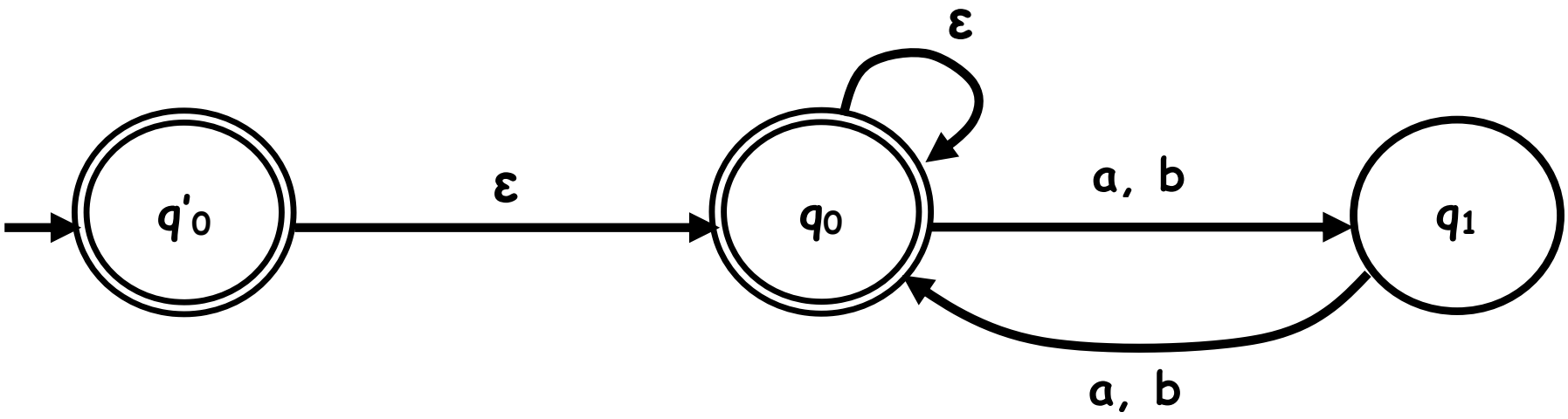


Figura: Automa che riconosce X^*

Esempio

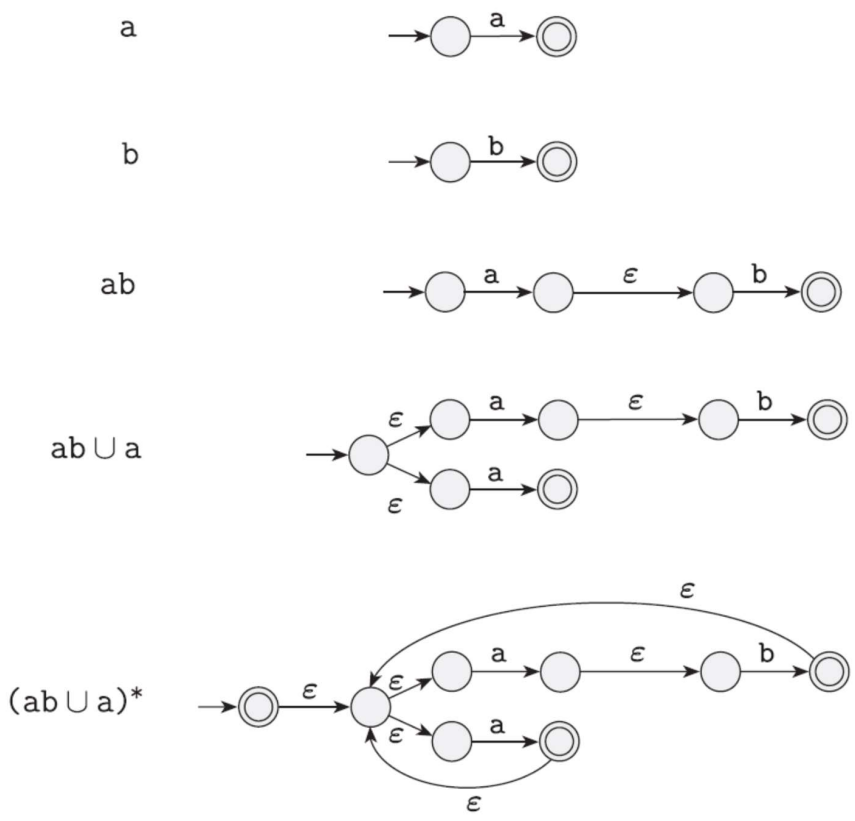


Figura: Un NFA costruito da $(ab + a)^*$

Prova (Cenni): dall'automa all'espressione regolare

Conversione di un DFA in un'espressione regolare

Esistono almeno due algoritmi per trasformare un DFA in un'espressione regolare equivalente

Conversione di un DFA in un'espressione regolare

Esistono almeno due algoritmi per trasformare un DFA in un'espressione regolare equivalente

- Metodo di Kleene

Conversione di un DFA in un'espressione regolare

Esistono almeno due algoritmi per trasformare un DFA in un'espressione regolare equivalente

- Metodo di Kleene
- Metodo di Brzozowski e McCluskey (BMC)

Supponiamo che $\mathcal{A} = (Q, \Sigma, \delta, 1, F)$ sia un DFA in cui gli stati sono indicati con numeri interi positivi.

Supponiamo che $\mathcal{A} = (Q, \Sigma, \delta, 1, F)$ sia un DFA in cui gli stati sono indicati con numeri interi positivi.

Sia $Q = \{1, 2, \ldots, n\}$, $n \geq 1$.

Supponiamo che $\mathcal{A} = (Q, \Sigma, \delta, 1, F)$ sia un DFA in cui gli stati sono indicati con numeri interi positivi.

Sia $Q = \{1, 2, \ldots, n\}$, $n \geq 1$.

Indichiamo con $R_{i,j}^{(k)}$ il linguaggio delle etichette dei cammini da i a j i cui nodi interni sono minori o uguali a k.

Supponiamo che $\mathcal{A} = (Q, \Sigma, \delta, 1, F)$ sia un DFA in cui gli stati sono indicati con numeri interi positivi.

Sia $Q = \{1, 2, \ldots, n\}$, $n \geq 1$.

Indichiamo con $R_{i,j}^{(k)}$ il linguaggio delle etichette dei cammini da i a j i cui nodi interni sono minori o uguali a k.

$R_{i,j}^{(0)}$ sarà il linguaggio delle etichette dei cammini da i a j che non hanno nodi interni.

Supponiamo che $\mathcal{A} = (Q, \Sigma, \delta, 1, F)$ sia un DFA in cui gli stati sono indicati con numeri interi positivi.

Sia $Q = \{1, 2, \ldots, n\}$, $n \geq 1$.

Indichiamo con $R_{i,j}^{(k)}$ il linguaggio delle etichette dei cammini da i a j i cui nodi interni sono minori o uguali a k.

$R_{i,j}^{(0)}$ sarà il linguaggio delle etichette dei cammini da i a j che non hanno nodi interni.

Quindi $R_{i,j}^{(0)}$ sarà il linguaggio delle etichette dei cammini da i a j di lunghezza 0 oppure 1.

Inoltre $R_{i,j}^{(n)}$ sarà il linguaggio delle etichette dei cammini da i a j.

Inoltre $R_{i,j}^{(n)}$ sarà il linguaggio delle etichette dei cammini da i a j.

Infatti ogni stato in $\mathcal{A}$ è rappresentato da un numero minore o uguale a n.

Inoltre $R_{i,j}^{(n)}$ sarà il linguaggio delle etichette dei cammini da i a j.

Infatti ogni stato in $\mathcal{A}$ è rappresentato da un numero minore o uguale a n.

Quindi

$$L(\mathcal{A}) = \bigcup_{j \in F} R_{1,j}^{(n)}$$

È possibile provare che esiste un'espressione regolare, denotata con abuso di notazione ancora con $R_{i,j}^{(k)}$, che rappresenta il linguaggio $R_{i,j}^{(k)}$ delle etichette dei cammini da i a j i cui nodi interni sono minori o uguali a k.

È possibile provare che esiste un'espressione regolare, denotata con abuso di notazione ancora con $R_{i,j}^{(k)}$, che rappresenta il linguaggio $R_{i,j}^{(k)}$ delle etichette dei cammini da i a j i cui nodi interni sono minori o uguali a k.

Quindi esiste un'espressione regolare che denota $L(\mathcal{A})$ ed è

$$\sum_{j \in F} R_{1,j}^{(n)}$$

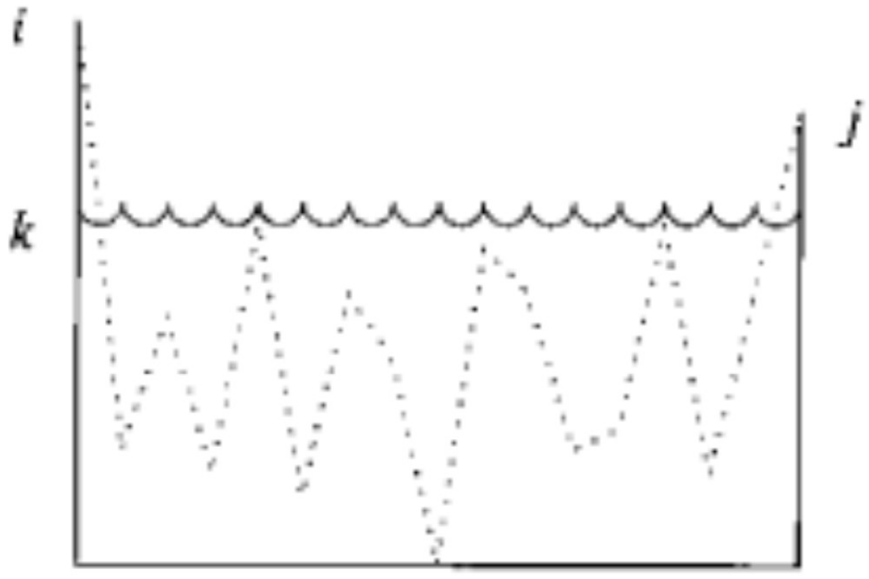


Figure 3.2: A path whose label is in the language of regular expression $R_{ij}^{(k)}$

Usando il principio di induzione è possibile provare che esiste un'espressione regolare che denota $R_{i,j}^{(k)}$, per ogni i, j, k.

Usando il principio di induzione è possibile provare che esiste un'espressione regolare che denota $R_{i,j}^{(k)}$, per ogni i, j, k.

La prova fornisce un algoritmo ricorsivo per calcolare le espressioni $R_{i,j}^{(k)}$ e quindi anche l'espressione regolare che denota $L(\mathcal{A})$.

Proviamo che esiste un'espressione regolare che denota $R_{i,j}^{(k)}$ per induzione su k.

Proviamo che esiste un'espressione regolare che denota $R_{i,j}^{(k)}$ per induzione su k.

PASSO BASE: $k = 0$.

$$R_{i,j}^{(0)} = \begin{cases} \sum_{\delta(i,a)=j} a & \text{se } i \neq j \\ \left(\sum_{\delta(i,a)=j} a\right) + \epsilon & \text{se } i = j \end{cases}$$

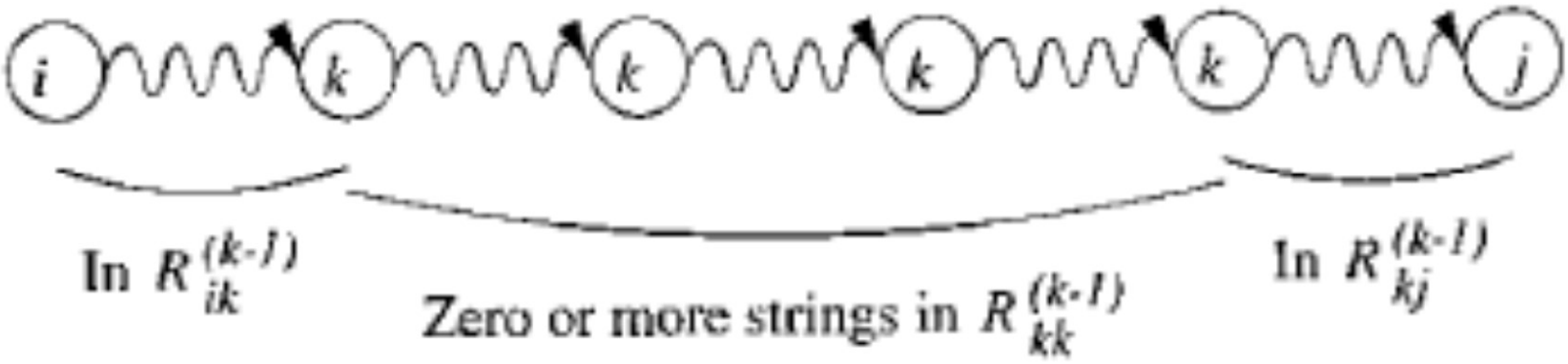
Proviamo che esiste un'espressione regolare che denota $R_{i,j}^{(k)}$ per induzione su k.

PASSO BASE: $k = 0$.

$$R_{i,j}^{(0)} = \begin{cases} \sum_{\delta(i,a)=j} a & \text{se } i \neq j \\ (\sum_{\delta(i,a)=j} a) + \epsilon & \text{se } i = j \end{cases}$$

PASSO INDUTTIVO: Se esiste un'espressione regolare che denota $R_{t,s}^{(k-1)}$, per ogni t, s, allora esiste un'espressione regolare che denota $R_{i,j}^{(k)}$, per ogni i, j. Infatti:

$$R_{i,j}^{(k)} = R_{i,j}^{(k-1)} + R_{i,k}^{(k-1)} (R_{k,k}^{(k-1)})^* R_{k,j}^{(k-1)}$$


i
k
k
k
k
j
In $R_{ik}^{(k-1)}$
Zero or more strings in $R_{kk}^{(k-1)}$
In $R_{kj}^{(k-1)}$

Metodo BMC

Il metodo BMC consiste nel calcolare, per ogni stato finale q, l'espressione regolare corrispondente ai cammini nell'automa dallo stato iniziale a q.

Metodo BMC

Il metodo BMC consiste nel calcolare, per ogni stato finale q, l'espressione regolare corrispondente ai cammini nell'automa dallo stato iniziale a q.

L'espressione si ottiene eliminando gli stati intermedi mediante una particolare procedura.

Metodo BMC

Il metodo BMC consiste nel calcolare, per ogni stato finale q, l'espressione regolare corrispondente ai cammini nell'automa dallo stato iniziale a q.

L'espressione si ottiene eliminando gli stati intermedi mediante una particolare procedura.

L'espressione regolare equivalente all'automa sarà l'unione delle espressioni precedentemente ottenute.