

Concetti di base

Alfabeto: Insieme finito e non vuoto di simboli

Esempio: $\Sigma = \{0, 1\}$ alfabeto binario

Esempio: $\Sigma = \{a, b, c, \dots, z\}$ insieme di tutte le lettere minuscole

Esempio: Insieme di tutti i caratteri ASCII

Stringa: Sequenza finita di simboli da un alfabeto Σ , es. 0011001
e' una stringa su alfabeto $\{0, 1\}$

Stringa vuota: La stringa con zero occorrenze di simboli da Σ

- La stringa vuota e' denotata con ϵ

Lunghezza di una stringa: Numero di posizioni per i simboli nella stringa.

$|w|$ denota la lunghezza della stringa w

$$|0110| = 4, |\epsilon| = 0$$

Potenze di un alfabeto: Σ^k = insieme delle stringhe di lunghezza k con simboli da Σ

Example: $\Sigma = \{0, 1\}$

$$\Sigma^1 = \{0, 1\}$$

$$\Sigma^2 = \{00, 01, 10, 11\}$$

$$\Sigma^0 = \{\epsilon\}$$

Domanda: Quante stringhe ci sono in Σ^3

L'insieme di tutte le stringhe su Σ e' denotato da Σ^*

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$$

Anche:

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$$

$$\Sigma^* = \Sigma^+ \cup \{\epsilon\}$$

Concatenazione: Se x e y sono stringhe, allora xy e' la stringa ottenuta collocando una copia di y subito dopo una copia di x

$$x = a_1a_2 \dots a_i, y = b_1b_2 \dots b_j$$

$$xy = a_1a_2 \dots a_ib_1b_2 \dots b_j$$

Esempio: $x = 01101, y = 110, xy = 01101110$

Nota: Per ogni stringa x

$$x\epsilon = \epsilon x = x$$

Linguaggi:

Se Σ e' un alfabeto, e $L \subseteq \Sigma^*$ allora L e' un linguaggio

Esempi di linguaggi:

- L'insieme delle parole italiane legali
- L'insieme dei programmi C legali
- L'insieme delle stringhe che consistono di n zeri seguiti da n uni

$$\{\epsilon, 01, 0011, 000111, \dots\}$$

- L'insieme delle stringhe con un numero uguale di zeri e di uni

$$\{\epsilon, 01, 10, 0011, 0101, 1001, \dots\}$$

- $L_P =$ insieme dei numeri binari il cui valore e' primo

$$\{10, 11, 101, 111, 1011, \dots\}$$

- Il linguaggio vuoto \emptyset
- Il linguaggio $\{\epsilon\}$ consiste della stringa vuota

Nota: $\emptyset \neq \{\epsilon\}$

Nota: L'alfabeto Σ e' sempre finito

Automi a stati finiti deterministici

Un DFA e' una quintupla

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

- Q e' un insieme finito di *stati*
- Σ e' un *alfabeto finito* (= simboli in input)
- δ e' una *funzione di transizione* da $Q \times \Sigma$ a Q ,
cioè: $(q, a) \mapsto p$ stato q e input a -> stato p
- $q_0 \in Q$ e' lo *stato iniziale*
- $F \subseteq Q$ e' un insieme di *stati finali*

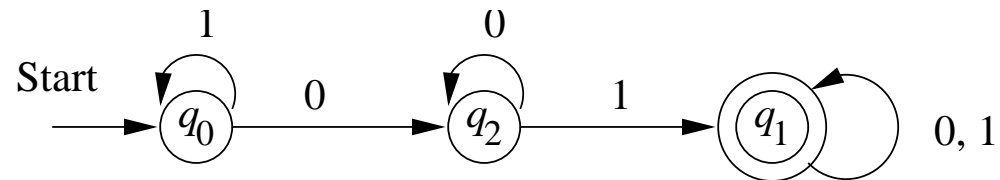
Esempio: Un automa A che accetta

$$L = \{x01y : x, y \in \{0, 1\}^*\}$$

L'automata $A = (\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_1\})$ definito tramite una *tabella di transizione*:

	0	1
$\rightarrow q_0$	q_2	q_0
$\star q_1$	q_1	q_1
q_2	q_2	q_1

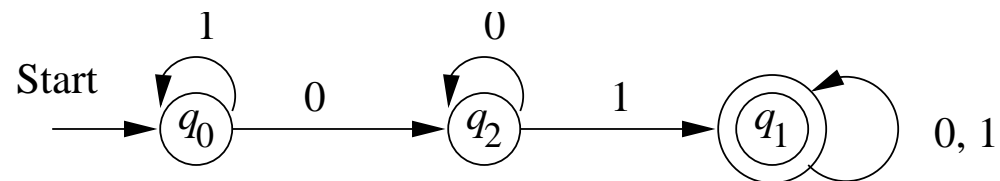
Lo stesso automa A definito, piu' semplicemente, tramite un *diagramma di transizione*:



Un automa a stati finiti (FA) *accetta* una stringa $w = a_1a_2 \cdots a_n$ se esiste un cammino nel diagramma di transizione che

1. Inizia nello stato iniziale
2. Finisce in uno stato finale (di accettazione)
3. Ha una sequenza di etichette $a_1a_2 \cdots a_n$

Esempio: L'automa a stati finiti



accetta ad esempio la stringa 01101

- La funzione di transizione δ puo' essere estesa a $\hat{\delta}$ che opera su stati e stringhe (invece che su stati e simboli)

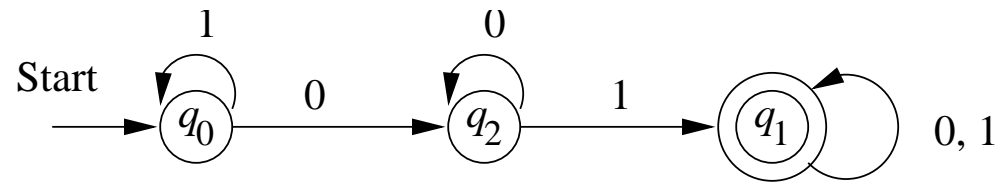
Base: $\hat{\delta}(q, \epsilon) = q$

Induzione: $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$

- Formalmente, il *linguaggio accettato da* un automa a stati finiti deterministico A e'

$$L(A) = \{w : \hat{\delta}(q_0, w) \in F\}$$

- I linguaggi accettati da automi a stati finiti sono detti *linguaggi regolari*

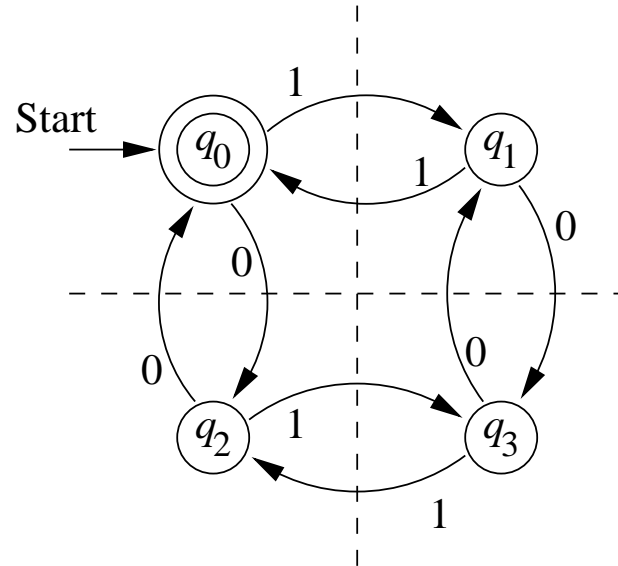


Base: $\hat{\delta}(q, \epsilon) = q \quad \forall q$

Induzione: $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a) \quad \forall q, x, a$

Esempio: Calcoliamo $\hat{\delta}(q_0, 0110)$

Esempio: DFA che accetta tutte e sole le stringhe con un numero pari di zeri e un numero pari di uni



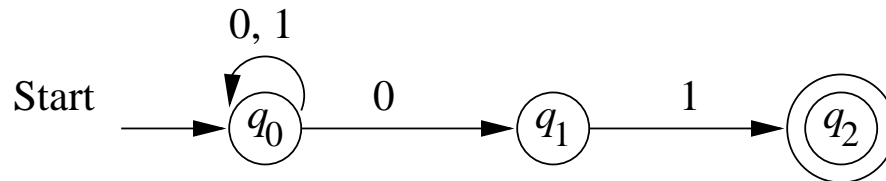
Rappresentazione tabulare dell'automa

	0	1
* \rightarrow q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

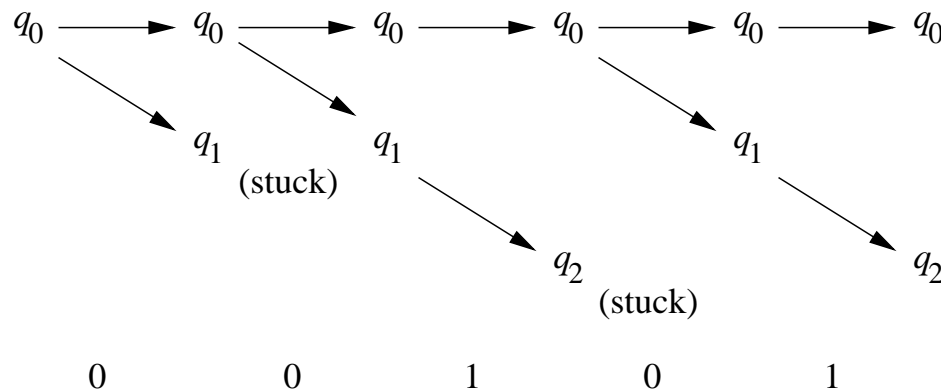
Automi a stati finiti nondeterministici (NFA)

Un NFA accetta una stringa se, tra i tanti possibili, esiste un cammino che conduce ad uno stato finale.

Esempio: un automa che accetta tutte e solo le stringhe che finiscono in 01.



Ecco cosa succede quando l'automa elabora l'input 00101



Formalmente, un NFA e' una quintupla

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

- Q e' un insieme finito di stati
- Σ e' un alfabeto finito
- δ e' una funzione di transizione da $Q \times \Sigma$ all'insieme dei sottoinsiemi di Q , cioè: $(q, a) \mapsto Q'$ con $Q' \subseteq Q$
- $q_0 \in Q$ e' lo *stato iniziale*
- $F \subseteq Q$ e' un insieme di *stati finali*

Esempio: L' NFA di due pagine fa e'

$$(\{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\})$$

dove δ e' la funzione di transizione

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	\emptyset	$\{q_2\}$
$\star q_2$	\emptyset	\emptyset

Funzione di transizione estesa $\hat{\delta}$.

Base: $\hat{\delta}(q, \epsilon) = \{q\}$

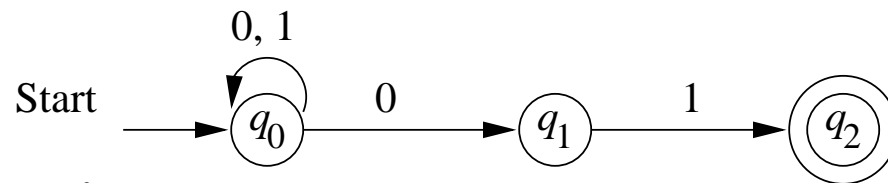
Induzione:

$$\hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q, x)} \delta(p, a)$$

Esempio: Calcoliamo $\hat{\delta}(q_0, 0010)$ sulla lavagna

- Formalmente, il *linguaggio accettato da* un automa a stati finiti nondeterministico A e'

$$L(A) = \{w : \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$$



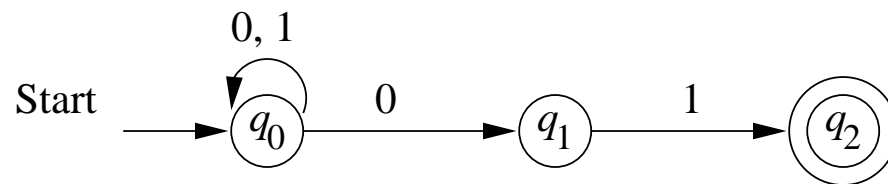
Base: $\hat{\delta}(q, \epsilon) = \{q\}$

Induzione: $\hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q, x)} \delta(p, a)$

Esempio: Calcoliamo $\hat{\delta}(q_0, 0010)$

- Gli NFA sono di solito piu' facili da "programmare" dei DFA

Esempio L'NFA



accetta il linguaggio $\{x01 : x \in \Sigma^*\}$.

Equivalenza di DFA e NFA

- Sorprendentemente, per ogni NFA N c'e' un DFA D , tale che $L(D) = L(N)$, e viceversa.
- Questo comporta una *costruzione a sottoinsiemi*, un esempio importante di come un automa B puo' essere costruito da un altro automa A .
- Dato un NFA

$$N = (Q_N, \Sigma, \delta_N, q_0, F_N)$$

costruiremo un DFA

$$D = (Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$$

tali che

$$L(D) = L(N)$$

I dettagli della costruzione a sottoinsiemi:

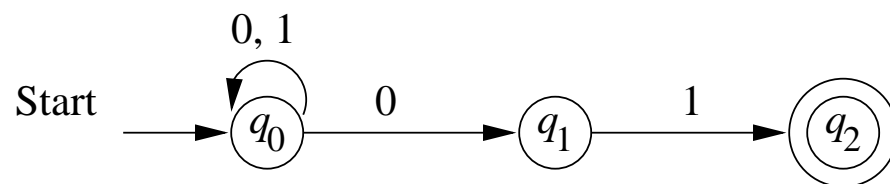
- $Q_D = \{S : S \subseteq Q_N\}$.

Nota: $|Q_D| = 2^{|Q_N|}$, anche se la maggior parte degli stati in Q_D sono "garbage", cioè non raggiungibili dallo stato iniziale.

- $F_D = \{S \subseteq Q_N : S \cap F_N \neq \emptyset\}$
- Per ogni $S \subseteq Q_N$ e $a \in \Sigma$,

$$\delta_D(S, a) = \bigcup_{p \in S} \delta_N(p, a)$$

Costruiamo δ_D (e F_D) dall' NFA



Costruiamo δ_D dall' NFA già visto:

	0	1
\emptyset	\emptyset	\emptyset
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	\emptyset	$\{q_2\}$
$\star\{q_2\}$	\emptyset	\emptyset
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\star\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\star\{q_1, q_2\}$	\emptyset	$\{q_2\}$
$\star\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

Nota: Gli stati di D corrispondono a sottoinsiemi di stati di N , ma potevamo denotare gli stati di D in un altro modo, per esempio $A - H$.

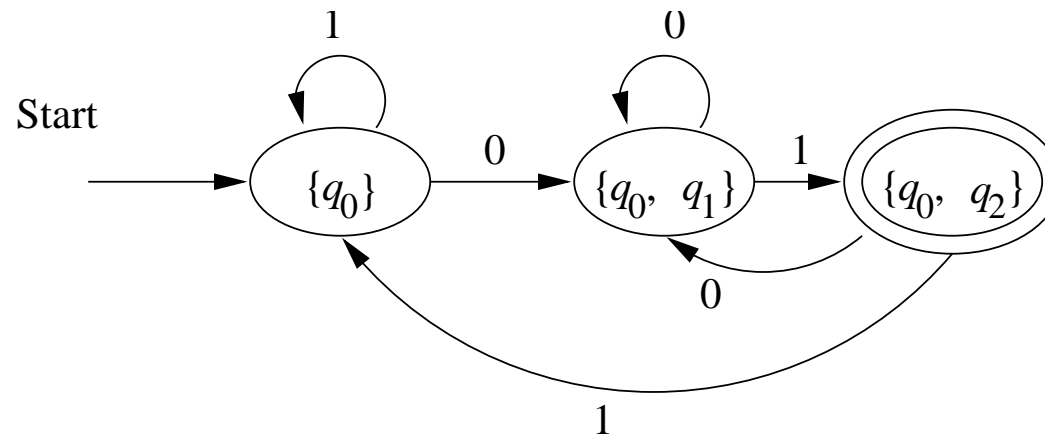
	0	1
A	A	A
$\rightarrow B$	E	B
C	A	D
$\star D$	A	A
E	E	F
$\star F$	E	B
$\star G$	A	D
$\star H$	E	F

Possiamo spesso evitare la crescita esponenziale degli stati costruendo la tabella di transizione per D solo per stati accessibili S come segue:

Base: $S = \{q_0\}$ e' accessibile in D

Induzione: Se lo stato S e' accessibile, lo sono anche gli stati $\delta_D(S, a)$ per ogni $a \in \Sigma$.

Esempio: Il "sottoinsieme" DFA solamente con stati accessibili.



Teorema 2.11: Sia D il DFA ottenuto da un NFA N con la costruzione a sottoinsiemi. Allora $L(D) = L(N)$.

Teorema 2.12: Un linguaggio L e' accettato da un DFA se e solo se L e' accettato da un NFA.

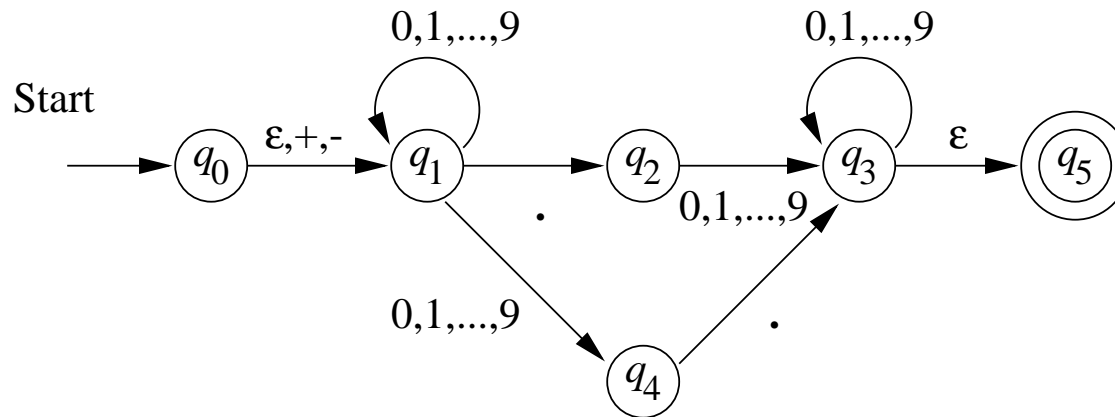
- Il numero di stati del DFA equivalente ad un NFA con n stati e', nel caso peggiore, pari a 2^n stati.

FA con transizioni epsilon

Un ϵ -NFA che accetta numeri decimali consiste di:

1. un segno $+$ o $-$, opzionale
2. una stringa di cifre decimali
3. un punto decimale
4. un'altra stringa di cifre decimali

Una delle stringhe (2) o (4) è opzionale



Un ϵ -NFA e' una quintupla $(Q, \Sigma, \delta, q_0, F)$ dove δ e' una funzione da $Q \times (\Sigma \cup \{\epsilon\})$ all'insieme dei sottoinsiemi di Q .

Esempio: L' ϵ -NFA della pagina precedente

$$E = (\{q_0, q_1, \dots, q_5\}, \{., +, -, 0, 1, \dots, 9\}, \delta, q_0, \{q_5\})$$

dove la tabella delle transizioni per δ e'

	ϵ	$+, -$	$.$	$0, \dots, 9$
$\rightarrow q_0$	$\{q_1\}$	$\{q_1\}$	\emptyset	\emptyset
q_1	\emptyset	\emptyset	$\{q_2\}$	$\{q_1, q_4\}$
q_2	\emptyset	\emptyset	\emptyset	$\{q_3\}$
q_3	$\{q_5\}$	\emptyset	\emptyset	$\{q_3\}$
q_4	\emptyset	\emptyset	$\{q_3\}$	\emptyset
$\star q_5$	\emptyset	\emptyset	\emptyset	\emptyset

Epsilon-chiusura

Chiudiamo uno stato aggiungendo tutti gli stati raggiungibili da lui tramite una sequenza $\epsilon\epsilon\cdots\epsilon$

Definizione induttiva di $ECLOSE(q)$

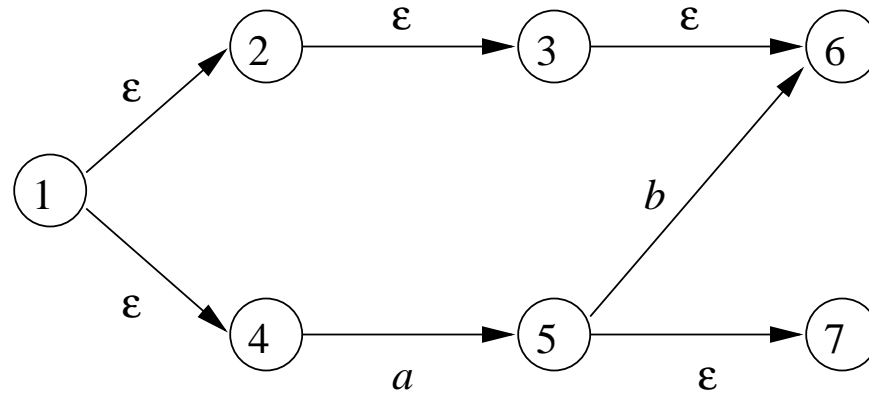
Base:

$$q \in ECLOSE(q)$$

Induzione:

$$p \in ECLOSE(q) \text{ and } r \in \delta(p, \epsilon) \Rightarrow \\ r \in ECLOSE(q)$$

Esempio di ϵ -chiusura



Per esempio,

$$\text{ECLOSE}(1) = \{1, 2, 3, 4, 6\}$$

- Definizione induttiva di $\hat{\delta}$ per automi ϵ -NFA

Base:

$$\hat{\delta}(q, \epsilon) = \text{ECLOSE}(q)$$

Induzione:

$$\hat{\delta}(q, xa) = \bigcup_{p \in \hat{\delta}(q, x)} \left(\bigcup_{t \in \delta(p, a)} \text{ECLOSE}(t) \right)$$

- *Linguaggio L accettato* è ancora definito $\{w : \hat{\delta}(q_0, w) \cap F \neq \emptyset\}$

Equivalenza di DFA e ϵ -NFA

Dato un ϵ -NFA

$$E = (Q_E, \Sigma, \delta_E, q_0, F_E)$$

costruiremo un DFA

$$D = (Q_D, \Sigma, \delta_D, q_D, F_D)$$

tale che

$$L(D) = L(E)$$

Dettagli della costruzione:

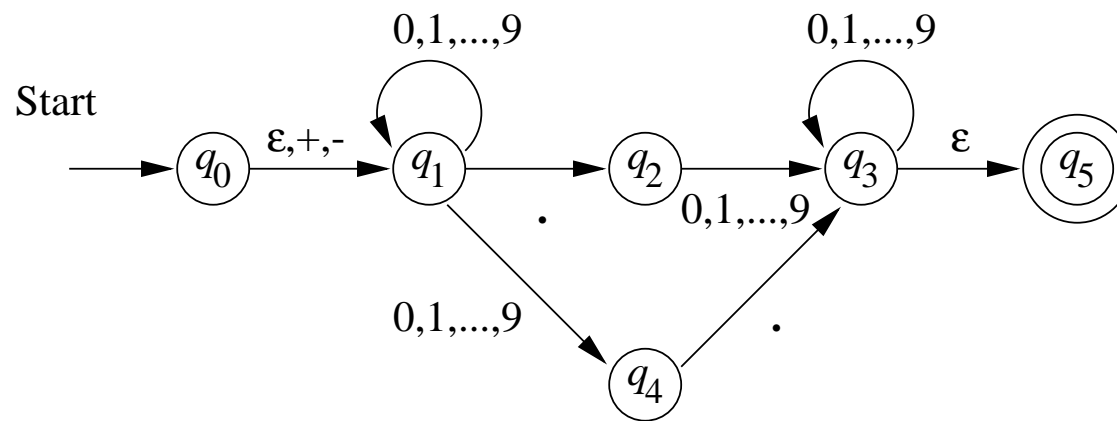
- $Q_D =$

$$\{S : S \subseteq Q_E \text{ e } S = \bigcup_{s \in S} \text{ECLOSE}(s)\}$$

- $q_D = \text{ECLOSE}(q_0)$
- $F_D = \{S : S \in Q_D \text{ e } S \cap F_E \neq \emptyset\}$
- $\delta_D(S, a) =$

$$\bigcup_{t \in S} \left(\bigcup_{p \in \delta_E(t, a)} \text{ECLOSE}(p) \right)$$

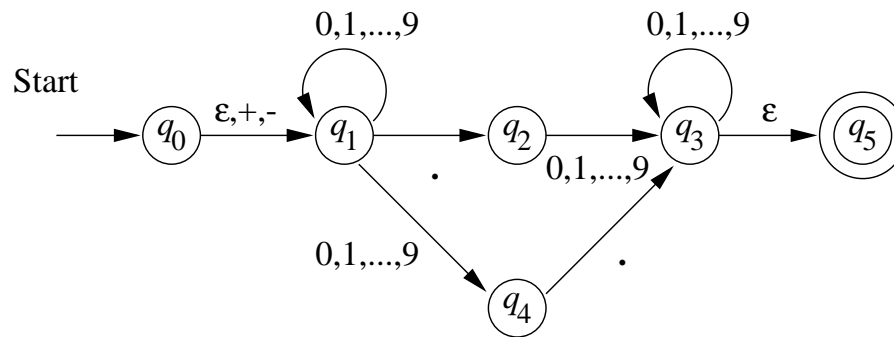
Esempio: ϵ -NFA E



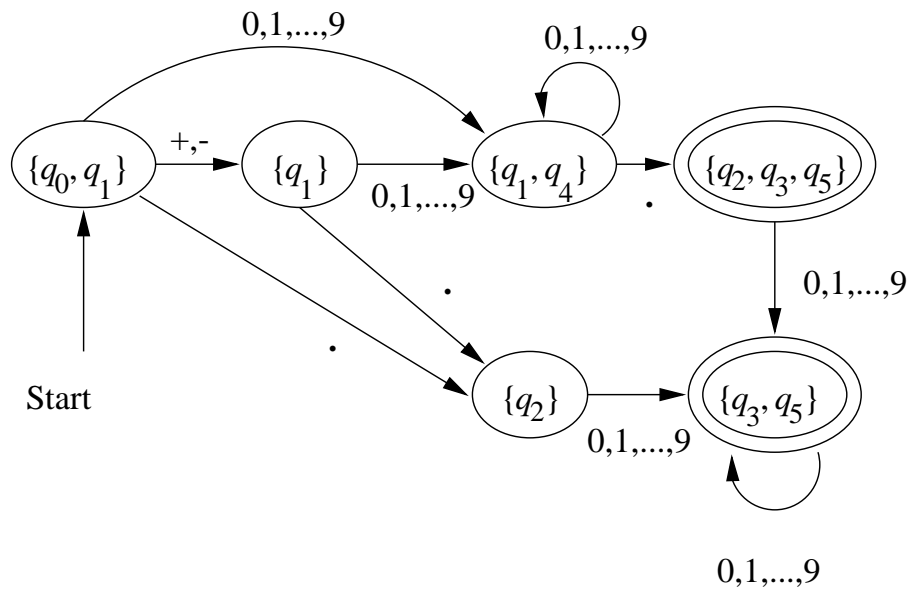
DFA D corrispondente ad E



Esempio: ϵ -NFA E



DFA D corrispondente ad E



Teorema 2.22: Un linguaggio L e' accettato da un ϵ -NFA E se e solo se L e' accettato da un DFA.

Espressioni regolari

Un FA (NFA o DFA) e' un metodo per costruire una macchina che riconosce linguaggi regolari.

Una *espressione regolare* e' un modo dichiarativo per descrivere un linguaggio regolare.

Esempio: $01^* + 10^*$

Le espressioni regolari sono usate, ad esempio, in

1. comandi UNIX (grep)
2. strumenti per l'analisi lessicale di linguaggi: es. Lex (Lexical analyzer generator), Flex (Fast Lex) e ANTLR (Another Tool For Language Recognition).

Operazioni sui linguaggi

Unione:

$$L \cup M = \{w : w \in L \text{ o } w \in M\}$$

Concatenazione:

$$L.M = \{w : w = xy, x \in L, y \in M\}$$

Potenze:

$$L^0 = \{\epsilon\}, L^1 = L, L^{k+1} = L.L^k$$

Chiusura di Kleene:

$$L^* = \bigcup_{i \geq 0} L^i$$

Domanda: Cosa sono \emptyset^0 , \emptyset^i , e \emptyset^* ?
epsilon vuoto epsilon

Leggi algebriche per i linguaggi

- $L \cup M = M \cup L$.

L'unione e' *commutativa*.

- $(L \cup M) \cup N = L \cup (M \cup N)$.

L'unione e' *associativa*.

- $(LM)N = L(MN)$.

La concatenazione e *associativa*

Nota: La concatenazione non e' commutativa, *cioe'*, esistono L e M tali che $LM \neq ML$.

- $\emptyset \cup L = L \cup \emptyset = L$.

\emptyset e' l'*identita'* per l'unione.

- $\{\epsilon\}L = L\{\epsilon\} = L$.

$\{\epsilon\}$ e' l'*identita' sinistra e destra* per la concatenazione.

- $\emptyset L = L\emptyset = \emptyset$.

\emptyset e' l'*annichilatore sinistro e destro* per la concatenazione.

- $L(M \cup N) = LM \cup LN$.

La concatenazione e' *distributiva a sinistra* sull'unione.

- $(M \cup N)L = ML \cup NL$.

La concatenazione e' *distributiva a destra* sull'unione.

- $L \cup L = L$.

L'unione e' *idempotente*.

- $\emptyset^* = \{\epsilon\}$, $\{\epsilon\}^* = \{\epsilon\}$.

- $L^+ = LL^* = L^*L$, $L^* = L^+ \cup \{\epsilon\}$

- $(L^*)^* = L^*$

Costruire le espressioni regolari

Definizione induttiva di espressioni regolari:

Base: ϵ e \emptyset sono espressioni regolari

$$L(\epsilon) = \{\epsilon\}, \text{ e } L(\emptyset) = \emptyset.$$

Se $a \in \Sigma$, allora a e' un'espressione regolare

$$L(a) = \{a\}.$$

Induzione:

Se E e' un'espr. regolare, allora (E) e' un'espr. regolare

$$L((E)) = L(E).$$

Se E e F sono espr. regolari, allora $E + F$ e' un'espr. regolare

$$L(E + F) = L(E) \cup L(F).$$

Se E e F sono espr. regolari, allora $E.F$ e' un'espr. regolare

$$L(E.F) = L(E).L(F).$$

Se E e' un'espr. regolare, allora E^* e' un'espr. regolare

$$L(E^*) = (L(E))^*.$$

$E = (0 + 1)(1^*)$ $L(E) = ?$
 $L(0) = \{0\}$ $L(1) = \{1\}$
 $L(0 + 1) = L(0) \cup L(1) = \{0, 1\}$
 $L(1^*) = L(1)^* = \{e, 1, 11, \dots, 1\dots 1\} = \{1^i \mid i \geq 0\}$
 $L(E) = L(0 + 1) \cdot L(1^*) = \{a1^i \mid a \in \{0,1\} \text{ \& } i \geq 0\}$

Esempio: Espressioni regolari per

$L = \{w \in \{0,1\}^* : 0 \text{ e } 1 \text{ alternati in } w\}$

$$(01)^* + (10)^* + 0(10)^* + 1(01)^*$$

o, equivalentemente,

$$(\epsilon + 1)(01)^*(\epsilon + 0)$$

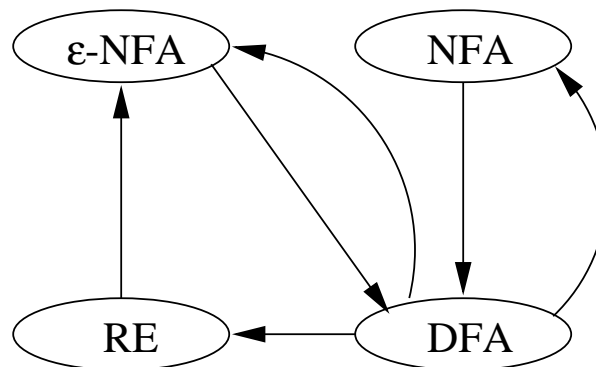
Ordine di precedenza per gli operatori:

1. Chiusura
2. Concatenazione (punto)
3. Più' (+)

Esempio: $01^* + 1$ e' raggruppato in $(0(1)^*) + 1$

Equivalenza di FA e espr. regolari

Abbiamo già mostrato che DFA, NFA, e ϵ -NFA sono tutti equivalenti.



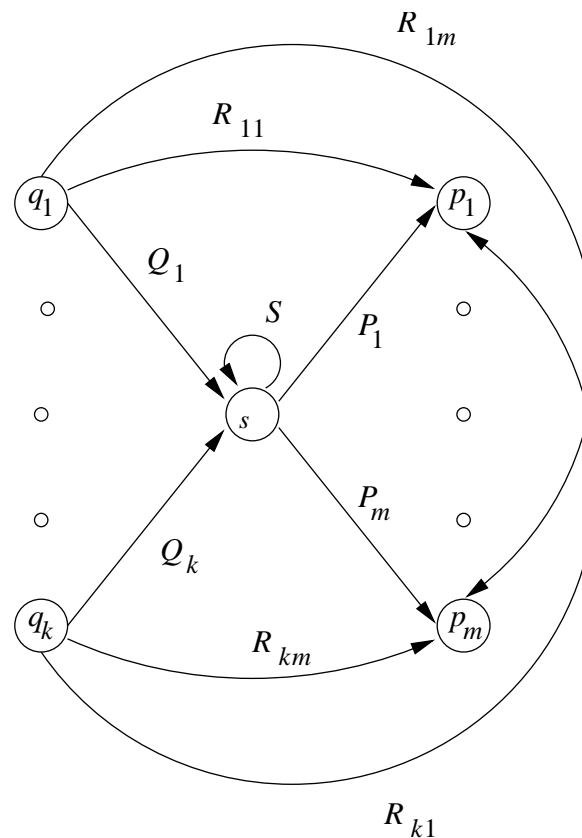
Per mostrare che gli FA sono equivalenti alle espressioni regolari dobbiamo stabilire che

1. Per ogni DFA A possiamo trovare (costruire, in questo caso) un'espressione regolare R , tale che $L(R) = L(A)$.
2. Per ogni espressione regolare R esiste un ϵ -NFA A , tale che $L(A) = L(R)$.

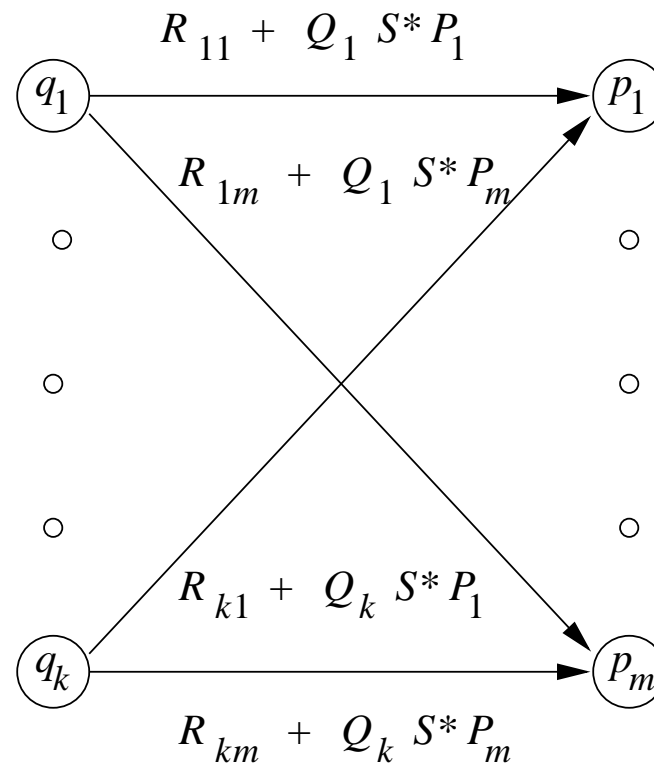
Da DFA a espressioni regolari

Teorema 3.4: Per ogni DFA $A = (Q, \Sigma, \delta, q_0, F)$ esiste una espressione regolare R , tale che $L(R) = L(A)$.

Trasformiamo l'automa etichettando gli archi con espressioni regolari di simboli

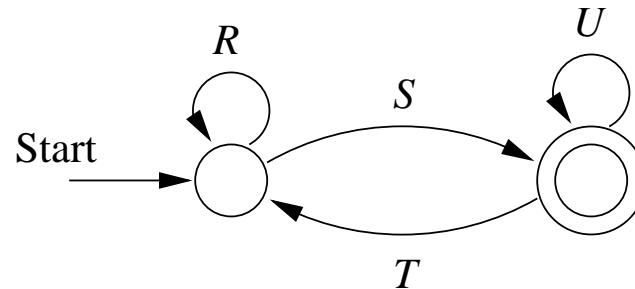


Ora eliminiamo lo stato s .

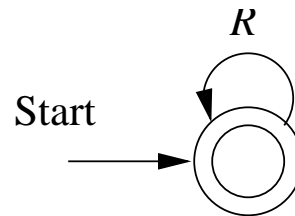


Per lo stato accettante q eliminiamo dall'automa originale tutti gli stati eccetto q_0 e q .

Per ogni $q \in F$ saremo rimasti con A_q della forma



e che corrisponde all'espressione regolare $E_q = (R + SU^*T)^*SU^*$
o con A_q della forma



che corrisponde all'espressione regolare $E_q = R^*$

- L'espressione finale e'

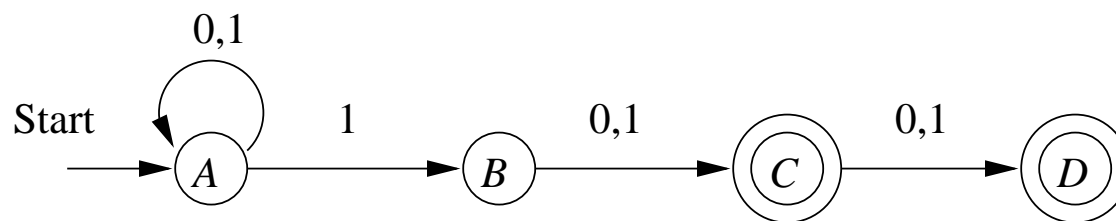
$$\bigoplus_{q \in F} E_q$$



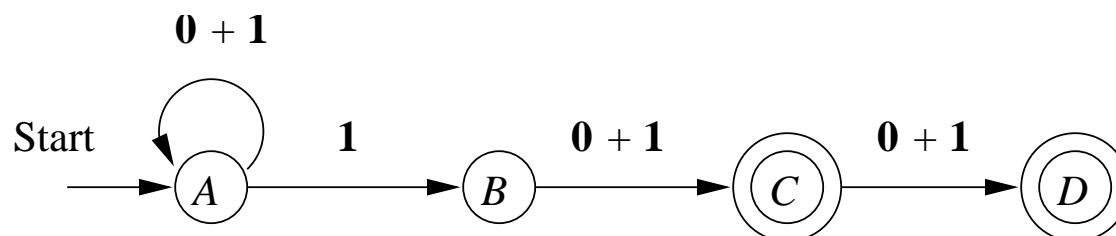
In realta' l'algoritmo e' applicabile anche a NFA!

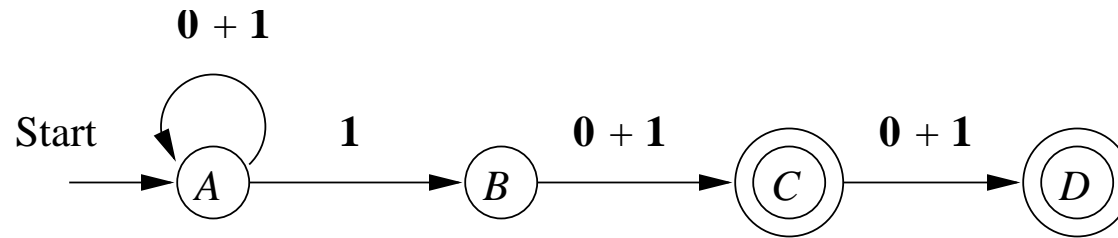
Esempio: \mathcal{A} , dove

$$L(\mathcal{A}) = \{w : w = x1b \text{ o } w = x1bc, \ x \in \{0,1\}^*, \{b,c\} \subseteq \{0,1\}\}$$

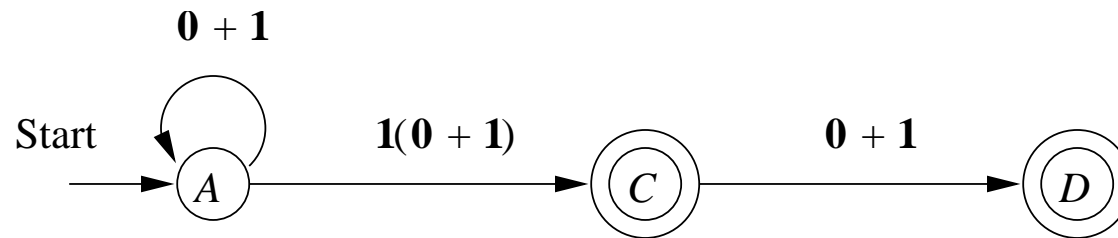


La trasformiamo in un automa con espressioni regolari come etichette

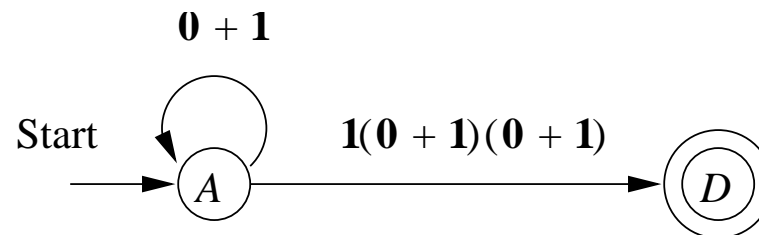




Eliminiamo lo stato B

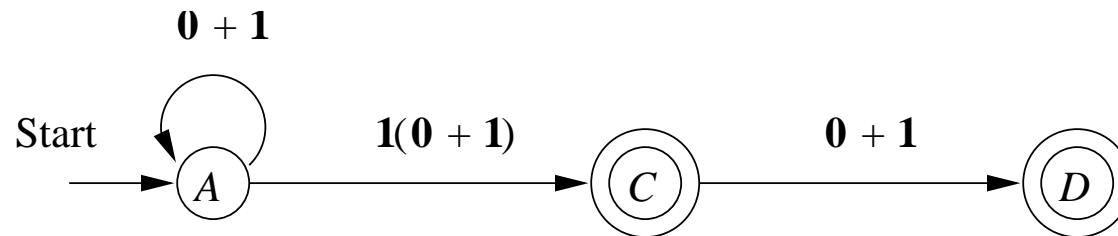


Poi eliminiamo lo stato C e otteniamo \mathcal{A}_D

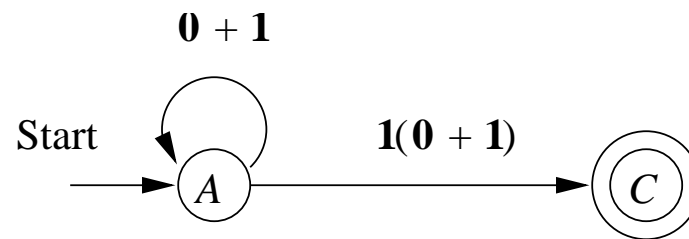


con espressione regolare $(0+1)^*1(0+1)(0+1)$

Da



possiamo eliminare D e ottenere \mathcal{A}_C



con espressione regolare $(0 + 1)^*1(0 + 1)$

- L'espressione finale e' la somma delle due espressioni regolari precedenti:

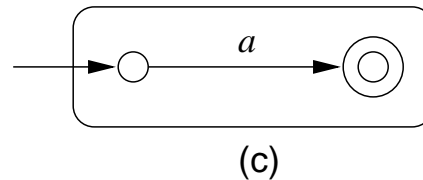
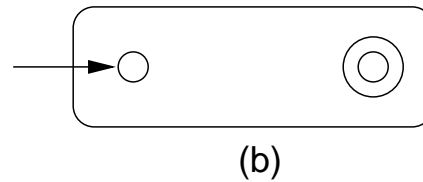
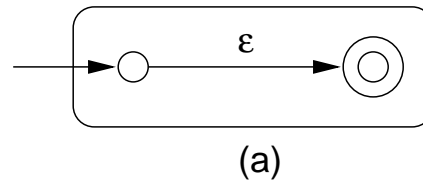
$$(0 + 1)^*1(0 + 1)(0 + 1) + (0 + 1)^*1(0 + 1)$$

Da espressioni regolari a ϵ -NFA

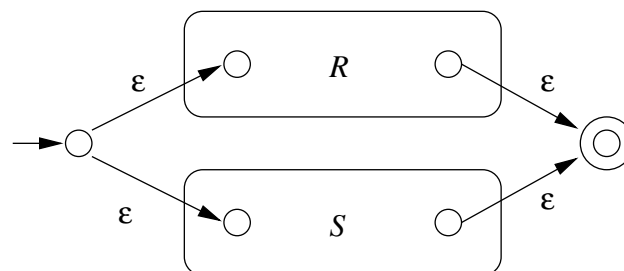
Teorema 3.7: Per ogni espressione regolare R possiamo costruire un ϵ -NFA A (con un unico stato finale, diverso da quello iniziale), tale che $L(A) = L(R)$.

Prova: Per induzione strutturale:

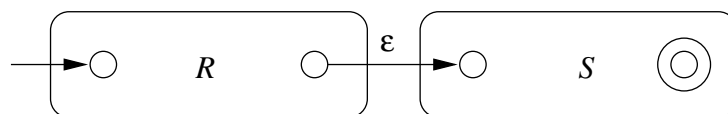
Base: Automa per ϵ , \emptyset , e a .



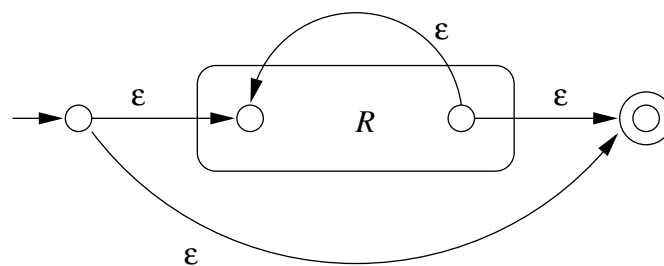
Induzione: Automa per $R + S$, RS , e R^*



(a)

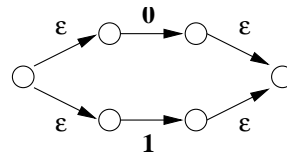


(b)

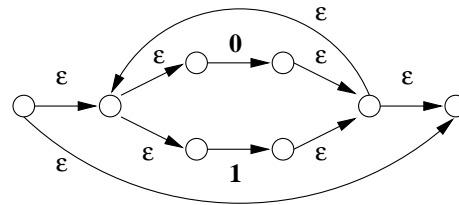


(c)

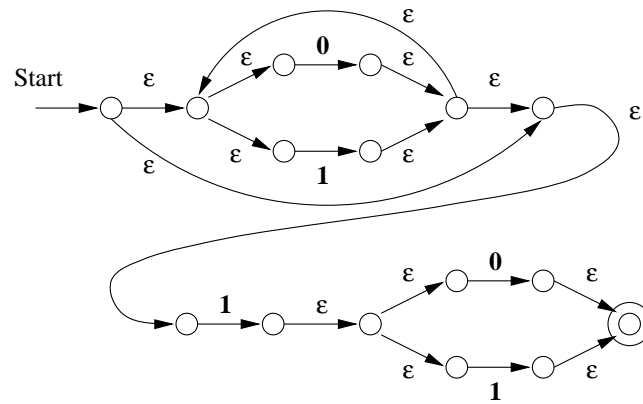
Esempio: Trasformiamo $(0 + 1)^*1(0 + 1)$



(a)



(b)



(c)

Proprieta' dei Linguaggi regolari

- *Pumping Lemma*. Ogni linguaggio regolare soddisfa il pumping lemma. Dato un linguaggio, se usando pumping lemma si ottiene una contraddizione allora esso non e' regolare.
- *Proprieta' di chiusura*. Come costruire automi da componenti usando delle operazioni e per quali operazioni è possibile farlo. Ad esempio dati L e M possiamo costruire un automa per $L \cap M$.
- *Proprieta' di decisione*. Analisi computazionale di automi, ad esempio quando due automi sono equivalenti.
- *Tecniche di minimizzazione*. Possiamo risparmiare costruendo automi piu' piccoli.

Il Pumping Lemma, idea di base

Supponiamo, per assurdo, che $L = \{0^h 1^h : h \geq 0\}$ sia regolare.

Allora deve essere accettato da un qualche DFA A . Chiamo n il **numero degli stati** di A .

Se A riceve in input **una stringa di lunghezza** $\geq n$ allora lungo il cammino di riconoscimento incontra **due volte uno stesso stato**.

Es. supponiamo che A **legga** 0^n . Considero gli *stati che incontra*:

ϵ	p_0 ($= q_0$ stato iniziale di A)
0	p_1
00	p_2
\dots	\dots
0^n	p_n

Sono $n + 1$

$\Rightarrow \exists i < j : p_i = p_j$, chiamiamo q questo stato

(dopo aver letto 0^i , legge altri 0 fino a 0^j , facendo **un ciclo** su q)

A ha dei **problemi di memoria!**

Va in q sia **ricevendo in input** 0^i che **ricevendo in input** 0^j !!

Siccome abbiamo supposto che A accetti $\{0^h 1^h : h \geq 0\}$ si avrà:

- l'automa, dovendo accettare $0^i 1^i$, sarà tale che $\hat{\delta}(q, 1^i) \in F$
- ma allora anche $0^j 1^i$ sarà accettata!! (**assurdo!**)

Quindi L non può essere regolare.

Proprietà di pompaggio dei DFA

- Il **pumping lemma** è basato su questa **idea del ciclo** e vale per **qualsiasi linguaggio regolare**.
- In particolare sulla seguente **proprietà di pompaggio** dei linguaggi riconosciuti da **DFA**

Dato un DFA A . Chiamo n il **numero degli stati** di A .

Osservazione: ogni stringa di lunghezza (almeno) n , data in input al DFA A , fa sì che l'automa **tocchi due volte uno stesso stato**.

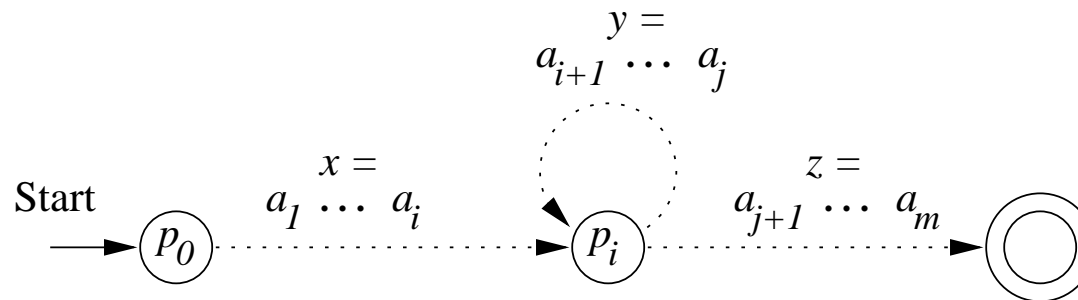
Dato un DFA A . Chiamo n il **numero degli stati** di A .

Osservazione: ogni stringa di lunghezza (almeno) n , data in input al DFA A , fa si' che l'automa **tocchi due volte uno stesso stato**.

Ogni $w = a_1a_2 \dots a_m \in L(A) : |w| \geq n$ e', quindi, **scomponibile in tre stringhe** $w = xyz$, cioe'

- $x = a_1a_2 \dots a_i$
- $y = a_{i+1}a_{i+2} \dots a_j$
- $z = a_{j+1}a_{j+2} \dots a_m$,

tali che il cammino di riconoscimento di w ha la seguente forma:



dove p_i e' lo stato raggiunto dopo l' i -esimo passo e $i < j \leq n$.

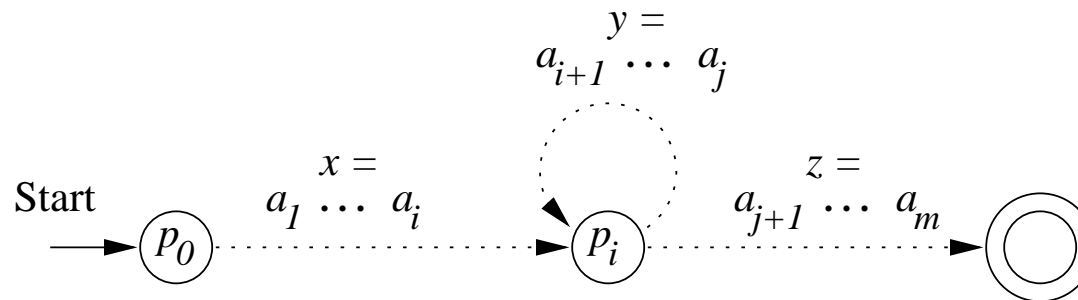
Dato un DFA A . Chiamo n il **numero degli stati** di A .

Osservazione: ogni stringa di lunghezza (almeno) n , data in input al DFA A , fa si' che l'automa **tocchi due volte uno stesso stato**.

Ogni $w = a_1a_2 \dots a_m \in L(A) : |w| \geq n$ e', quindi, **scomponibile in tre stringhe** $w = xyz$, cioe'

- $x = a_1a_2 \dots a_i$
- $y = a_{i+1}a_{i+2} \dots a_j$
- $z = a_{j+1}a_{j+2} \dots a_m$,

tali che il cammino di riconoscimento di w ha la seguente forma:



dove p_i e' lo stato raggiunto dopo l' i -esimo passo e $i < j \leq n$.

Quindi tutte le stringhe $xy^kz \in L(A)$, **per ogni** $k \geq 0$
(cioe', oltre a $w = xyz$, A accetta anche $xz, xyyz, xyyyz, \dots$)

Inoltre $y \neq \epsilon$ e $|xy| \leq n$ (poiche' $i < j \leq n$).

Teorema 4.1.

Il Pumping Lemma per Linguaggi Regolari.

Sia L un linguaggio regolare.

Allora $\exists n \geq 1$ che soddisfa:

ogni $w \in L : |w| \geq n$ e' scomponibile in tre stringhe $w = xyz$ tali che

1. $y \neq \epsilon$

2. $|xy| \leq n$

3. $\forall k \geq 0, xy^kz \in L$

Prova: Supponiamo che L sia regolare.

Allora L è riconosciuto da un DFA A . Chiamo n il **numero degli stati** di A .

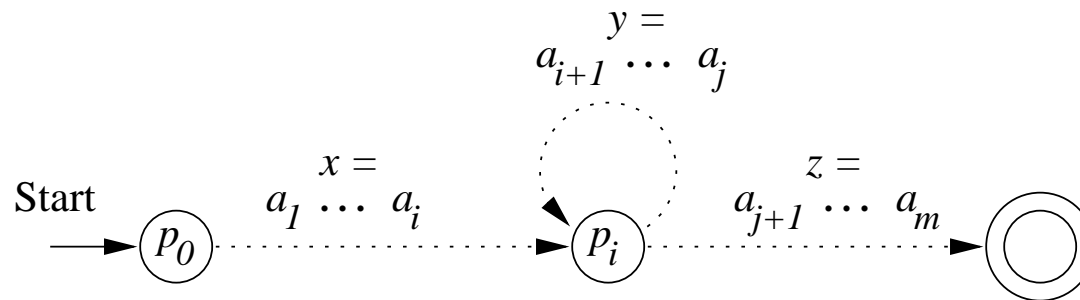
Sia $w = a_1a_2 \dots a_m \in L$, $m \geq n$.

Sia $p_i = \hat{\delta}(q_0, a_1a_2 \dots a_i)$, dove q_0 è lo stato iniziale di A .

$\Rightarrow \exists i < j \leq n: p_i = p_j$

Ora $w = xyz$, dove

- $x = a_1 a_2 \cdots a_i$
- $y = a_{i+1} a_{i+2} \cdots a_j$
- $z = a_{j+1} a_{j+2} \cdots a_m$



Quindi anche $xy^kz \in L$, per ogni $k \geq 0$.

Inoltre $y \neq \epsilon$ e $|xy| \leq n$ poiche' $i < j \leq n$.

Esempio: sia L_{eq} il linguaggio delle stringhe con **ugual numero di zeri e di uni**.

Supponiamo che L_{eq} sia regolare. Allora, sia n dato dal pumping lemma, consideriamo $w = 0^n 1^n \in L$.

Per il pumping lemma

$\exists x, y, z : w = xyz, |xy| \leq n, y \neq \epsilon$ e $xy^kz \in L_{eq}$, per ogni $k \geq 0$.

Essendo $|xy| \leq n$ si ha che y deve essere per forza fatta di soli 0, cioè:

$$w = \underbrace{0 \dots 0}_x \underbrace{0 \dots 0}_y \underbrace{0 \dots 0 111 \dots 111}_z$$

In particolare, quindi (per $k = 0$) dovrebbe valere $xz \in L_{eq}$, ma xz ha meno zeri di uni! (**assurdo**)

Proprieta' di chiusura dei linguaggi regolari

Siano L e M due linguaggi regolari. Allora i seguenti linguaggi sono regolari:

- *Unione:* $L \cup M$
- *Intersezione:* $L \cap M$
- *Complemento:* \overline{L}
- *Differenza:* $L \setminus M$
- *Inversione:* $L^R = \{w^R : w \in L\}$
- *Chiusura:* L^* .
- *Concatenazione:* $L.M$

Teorema 4.4. Per ogni coppia di linguaggi regolari L e M , $L \cup M$ e' regolare.

Prova. Sia $L = L(E)$ e $M = L(F)$. Allora $L(E + F) = L \cup M$ per definizione.

Teorema 4.5. Se L e' un linguaggio regolare su Σ , allora anche $\overline{L} = \Sigma^* \setminus L$ e' regolare.

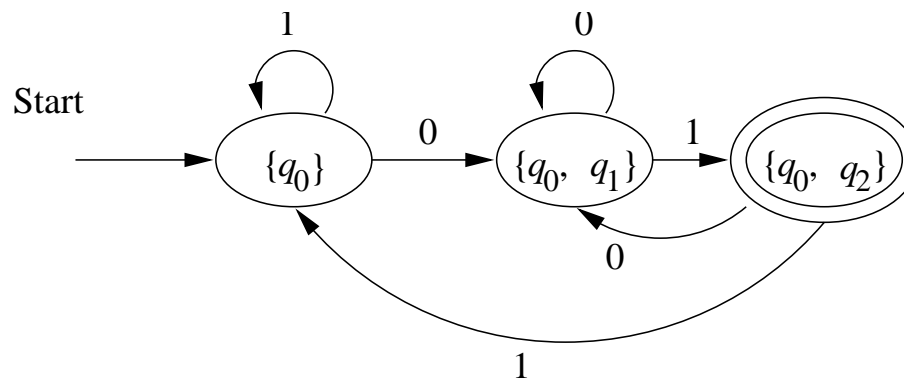
Prova. Sia L riconosciuto da un DFA

$$A = (Q, \Sigma, \delta, q_0, F).$$

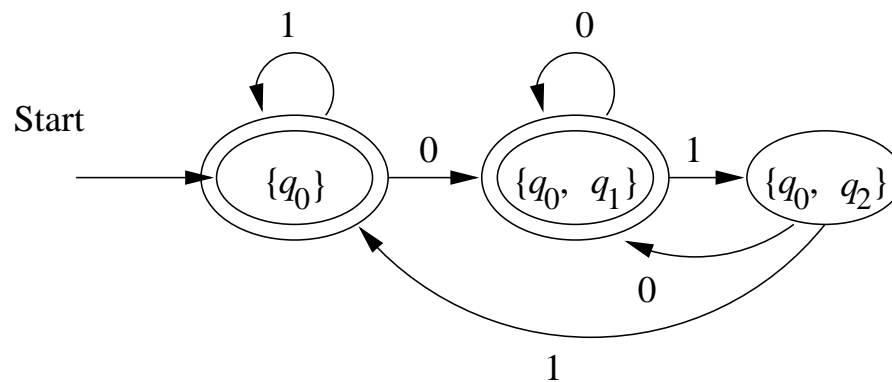
Sia $B = (Q, \Sigma, \delta, q_0, Q \setminus F)$. Allora $L(B) = \overline{L}$.

Esempio:

Sia L riconosciuto dal DFA qui sotto:



Allora \overline{L} e' riconosciuto da:



Domanda: Espressione regolare per \overline{L} a partire da una per L ?

Teorema 4.8. Se L e M sono regolari, allora anche $L \cap M$ e' regolare.

Prova. Per la legge di DeMorgan, $L \cap M = \overline{\overline{L} \cup \overline{M}}$. Sappiamo gia' che i linguaggi regolari sono chiusi rispetto al complemento e all'unione.

Teorema 4.8. Se L e M sono regolari, allora anche $L \cap M$ e' regolare.

Prova alternativa. Sia L il linguaggio di

$$A_L = (Q_L, \Sigma, \delta_L, q_L, F_L)$$

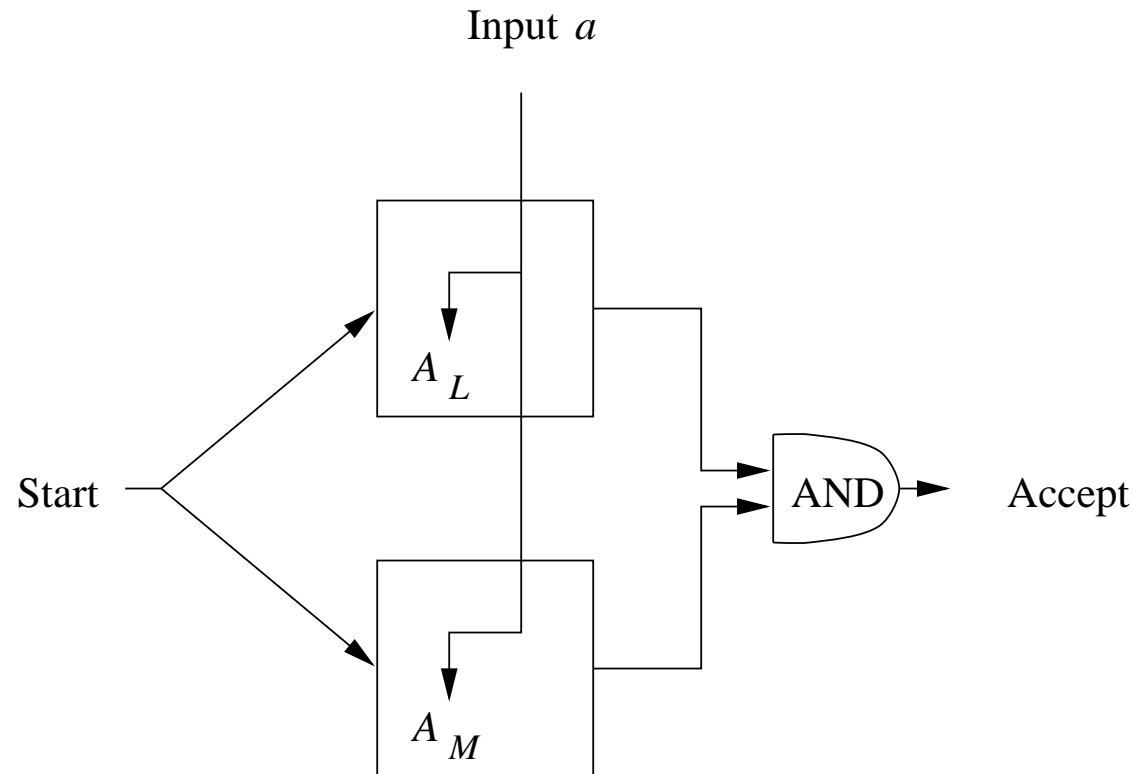
e M il linguaggio di

$$A_M = (Q_M, \Sigma, \delta_M, q_M, F_M)$$

Assumiamo senza perdita di generalita' che entrambi gli automi siano deterministici.

Costruiremo un automa che simula A_L e A_M in parallelo, e accetta se e solo se sia A_L che A_M accettano.

Se A_L va dallo stato p allo stato s leggendo a , e A_M va dallo stato q allo stato t leggendo a , allora $A_{L \cap M}$ andra' dallo stato (p, q) allo stato (s, t) leggendo a .



Formalmente

$$A_{L \cap M} = (Q_L \times Q_M, \Sigma, \delta_{L \cap M}, (q_L, q_M), F_L \times F_M),$$

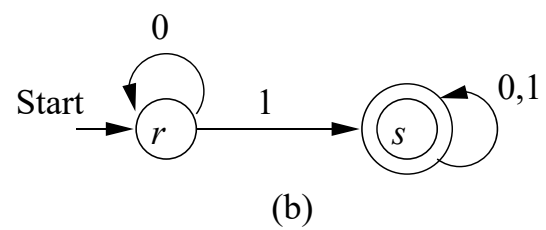
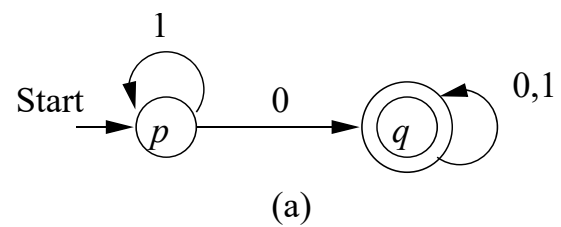
dove

$$\delta_{L \cap M}((p, q), a) = (\delta_L(p, a), \delta_M(q, a))$$

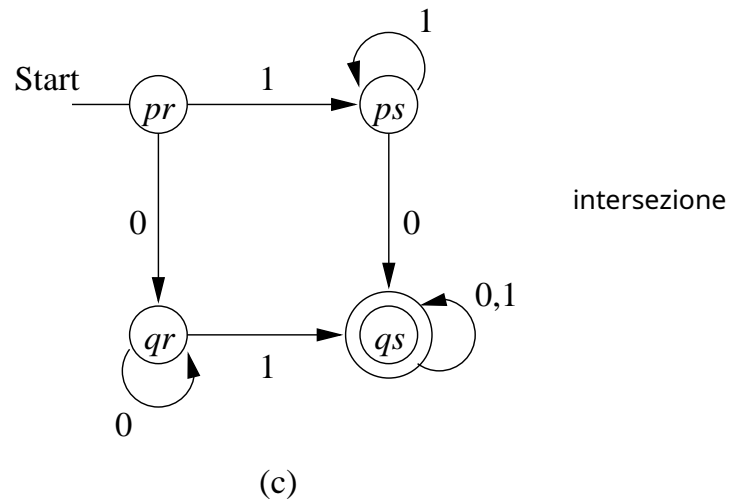
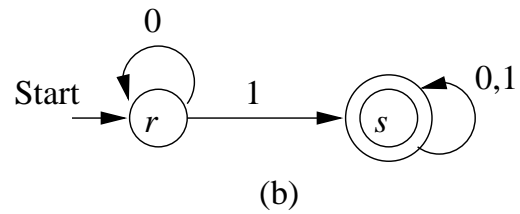
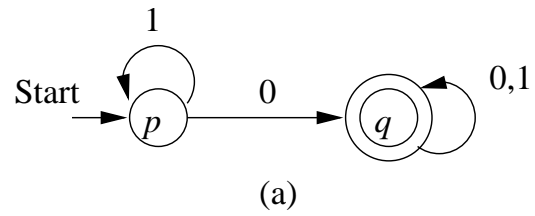
Si puo' mostrare per induzione su $|w|$ che

$$\hat{\delta}_{L \cap M}((q_L, q_M), w) = (\hat{\delta}_L(q_L, w), \hat{\delta}_M(q_M, w))$$

Esempio: $(c) = (a) \times (b)$ (= intersezione)



Esempio: $(c) = (a) \times (b)$



DIFFERENZA

Teorema 4.10. Se L e M sono linguaggi regolari, allora anche $L \setminus M$ e' regolare.

Prova. Osserviamo che $L \setminus M = L \cap \overline{M}$. Sappiamo gia' che i linguaggi regolari sono chiusi rispetto al complemento e all'intersezione.

Teorema 4.11. Se L e' un linguaggio regolare, allora anche L^R (le stringhe w^R , cioè w lette al “reverse”, con $w \in L$) è regolare.

Prova 1: Sia L riconosciuto da un FA A . Modifichiamo A per renderlo un FA per L^R :

1. Giriamo tutti gli archi.
2. Rendiamo il vecchio stato iniziale l'unico stato finale.
3. Creiamo un nuovo stato iniziale p_0 , con $\delta(p_0, \epsilon) = F$ (i vecchi stati finali).

Proprieta' di decisione

1. $E' \ L = \emptyset?$
2. $E' \ w \in L?$
3. Due descrizioni definiscono lo stesso linguaggio?

Testare se un linguaggio e' vuoto

$L(A) \neq \emptyset$ per FA A se e solo se uno stato finale e' raggiungibile dallo stato iniziale in A . Totale: $O(n^2)$ passi.

Oppure, possiamo guardare un'espressione regolare E di lunghezza s e vedere se $L(E) = \emptyset$ in $O(s)$ passi. Usiamo il seguente metodo:

$E = F + G$. Allora $L(E)$ e' vuoto se e solo se sia $L(F)$ che $L(G)$ sono vuoti.

$E = F.G$. Allora $L(E)$ e' vuoto se e solo se $L(F)$ o $L(G)$ sono vuoti.

$E = F^*$. Allora $L(E)$ non e' mai vuoto, perche' $\epsilon \in L(E)$.

$E = \epsilon$. Allora $L(E)$ non e' vuoto.

$E = a$. Allora $L(E)$ non e' vuoto.

$E = \emptyset$. Allora $L(E)$ e' vuoto.

Controllare l'appartenenza

Per controllare se $w \in L(A)$ per DFA A , simuliamo A su w . Se $|w| = n$, questo prende $O(n)$ passi.

Se A e' un NFA e ha s stati, simulare A su w prende $O(ns^2)$ passi. Lo stesso per ϵ -NFA calcolando preventivamente le ECLOSE di tutti gli stati.

Se $L = L(E)$, per l'espressione regolare E di lunghezza s , prima convertiamo E in un ϵ -NFA con $2s$ stati. Poi simuliamo w su questo automa, in $O(ns^2)$ passi.

Equivalenza e Minimizzazione di Automi

Sia $A = (Q, \Sigma, \delta, q_0, F)$ un DFA, e $\{p, q\} \subseteq Q$. Definiamo

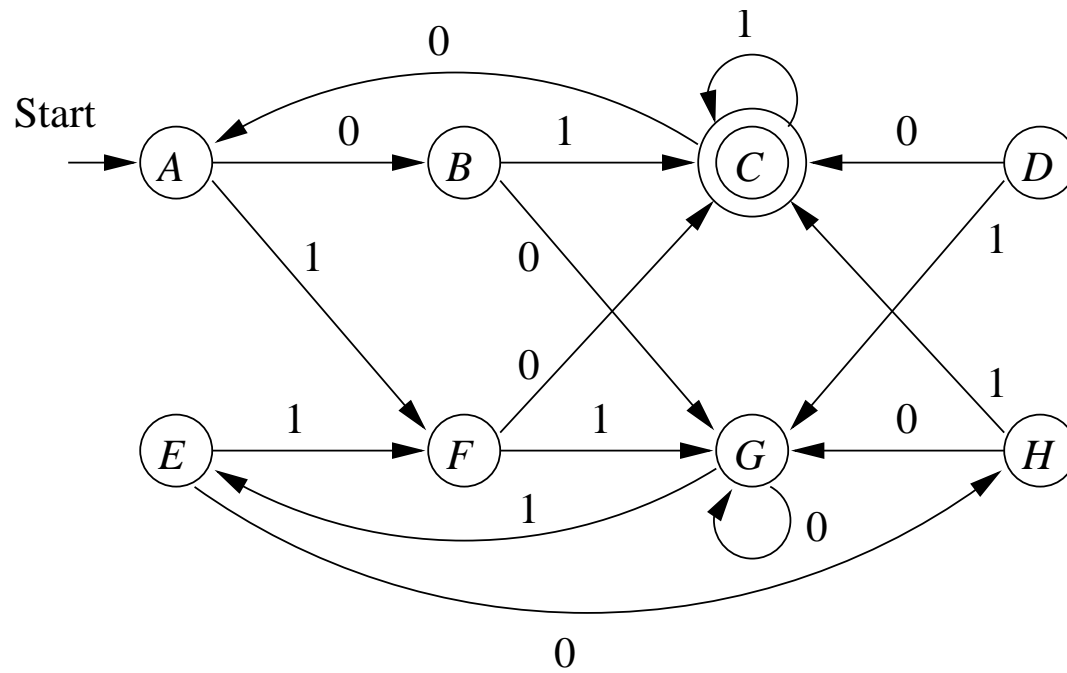
$$p \equiv q \Leftrightarrow \forall w \in \Sigma^* : \hat{\delta}(p, w) \in F \text{ se e solo se } \hat{\delta}(q, w) \in F$$

- Se $p \equiv q$ diciamo che p e q sono *equivalenti*
- Se $p \not\equiv q$ diciamo che p e q sono *distinguibili*

In altre parole: p e q sono distinguibili se e solo se

$$\exists w : \hat{\delta}(p, w) \in F \text{ e } \hat{\delta}(q, w) \notin F, \text{ o viceversa}$$

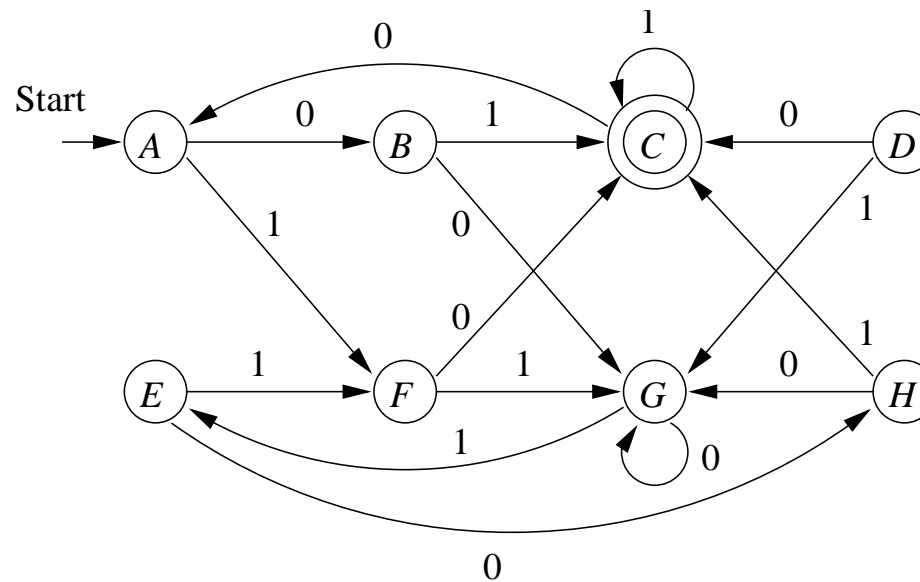
Esempio:



$$\hat{\delta}(C, \epsilon) \in F, \hat{\delta}(G, \epsilon) \notin F \Rightarrow C \neq G$$

$$\hat{\delta}(A, 01) = C \in F, \hat{\delta}(G, 01) = E \notin F \Rightarrow A \neq G$$

Cosa si puo' dire su A e E ?



$$\hat{\delta}(A, \epsilon) = A \notin F, \hat{\delta}(E, \epsilon) = E \notin F$$

$$\hat{\delta}(A, 1) = F = \hat{\delta}(E, 1)$$

$$\text{Quindi } \hat{\delta}(A, 1x) = \hat{\delta}(E, 1x) = \hat{\delta}(F, x)$$

$$\hat{\delta}(A, 0) = B \notin F \quad \hat{\delta}(E, 0) = H \notin F$$

$$\hat{\delta}(A, 00) = G = \hat{\delta}(E, 00) \quad \hat{\delta}(A, 01) = C = \hat{\delta}(E, 01)$$

Conclusione: $A \equiv E$.

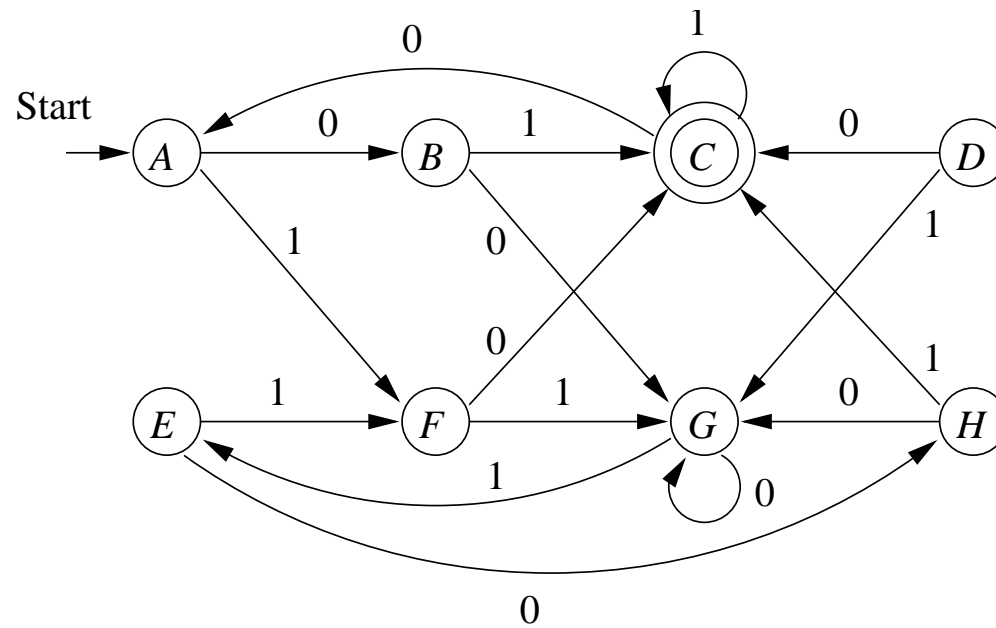
Possiamo calcolare coppie di stati distinguibili con il seguente metodo induttivo (*algoritmo riempi-tabella*):

Base: Se $p \in F$ e $q \notin F$, allora $p \not\equiv q$.

Induzione: Se $\exists a \in \Sigma : \delta(p, a) \not\equiv \delta(q, a)$,
allora $p \not\equiv q$.

Esempio: Applichiamo l'algoritmo ad A:

<i>B</i>	<i>x</i>						
<i>C</i>	<i>x</i>	<i>x</i>					
<i>D</i>	<i>x</i>	<i>x</i>	<i>x</i>				
<i>E</i>		<i>x</i>	<i>x</i>	<i>x</i>			
<i>F</i>	<i>x</i>	<i>x</i>	<i>x</i>		<i>x</i>		
<i>G</i>	<i>x</i>	<i>x</i>	<i>x</i>	<i>x</i>	<i>x</i>	<i>x</i>	
<i>H</i>	<i>x</i>		<i>x</i>	<i>x</i>	<i>x</i>	<i>x</i>	<i>x</i>
	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>	<i>G</i>



<i>B</i>	1						
<i>C</i>	1	1					
<i>D</i>	2	1	1				
<i>E</i>		2	1	1			
<i>F</i>	4	3	1		1		
<i>G</i>	1	4	1	2		1	
<i>H</i>	5		1	3	2	2	1
	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>	<i>G</i>

A = E B = H D = F

1
1
1
1
1
1
1
1

Teorema 4.20: Se p e q non sono distinti dall'algoritmo, allora $p \equiv q$.

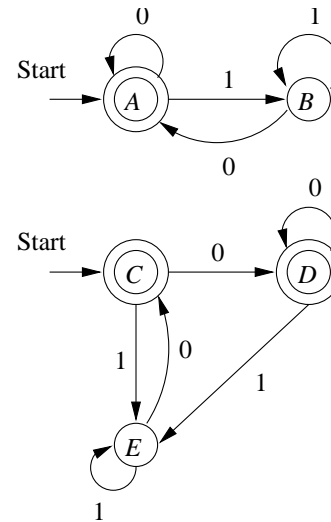
Testare l'equivalenza di linguaggi regolari

Siano L e M linguaggi regolari (descritti in qualche forma).

Per testare se $L = M$

1. convertiamo sia L che M in DFA.
2. Immaginiamo il DFA che e' l'unione dei due DFA (non importa se ha due stati iniziali)
3. Se l'algoritmo dice che i due stati iniziali sono distinguibili, allora $L \neq M$, altrimenti $L = M$.

Esempio:



Il risultato dell'algoritmo e'

<i>B</i>	<i>x</i>			
<i>C</i>		<i>x</i>		
<i>D</i>		<i>x</i>		
<i>E</i>	<i>x</i>		<i>x</i>	<i>x</i>
	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>

Se gli stati iniziali sono equivalenti allora i 2 automi sono equivalenti

Quindi i due automi sono equivalenti.

Minimizzazione di DFA

Algoritmo per minimizzare un DFA:
mettiamo insieme tutti gli stati equivalenti (considerando come stati le classi di equivalenza p/\equiv al posto di p)
ed eliminiamo eventuali stati non raggiungibili dallo stato iniziale.

Esempio: Il DFA di prima ha le seguenti classi di equivalenza:
 $\{\{A, E\}, \{B, H\}, \{C\}, \{D, F\}, \{G\}\}$.

Il DFA unione di prima ha le seguenti classi di equivalenza:
 $\{\{A, C, D\}, \{B, E\}\}$.

Nota: p/\equiv è *classe di equivalenza* della *relazione di equivalenza* \equiv (cioè una relazione riflessiva, simmetrica, e transitiva).

Quindi, **per minimizzare un DFA** $A = (Q, \Sigma, \delta, q_0, F)$, dove assumiamo di aver già eliminato tutti gli stati non raggiungibili dallo stato iniziale, costruiamo un DFA $B = (Q/\equiv, \Sigma, \gamma, q_0/\equiv, F/\equiv)$, dove

$$\gamma(p/\equiv, a) = \delta(p, a)/\equiv$$

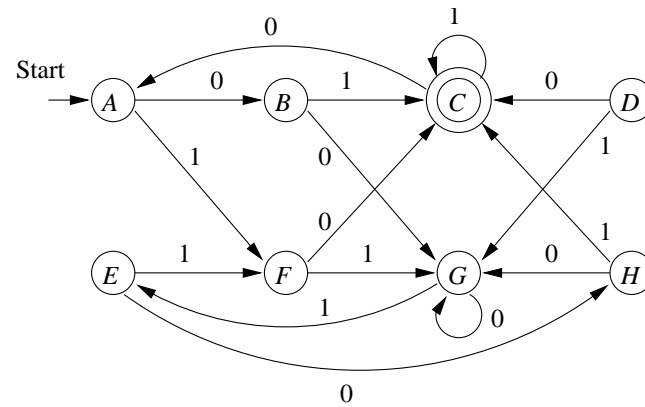
e Q/\equiv è l'insieme delle classi di equivalenza degli stati di Q (F/\equiv l'insieme di quelle degli stati accettanti).

Affinché B sia ben definito, dobbiamo mostrare che

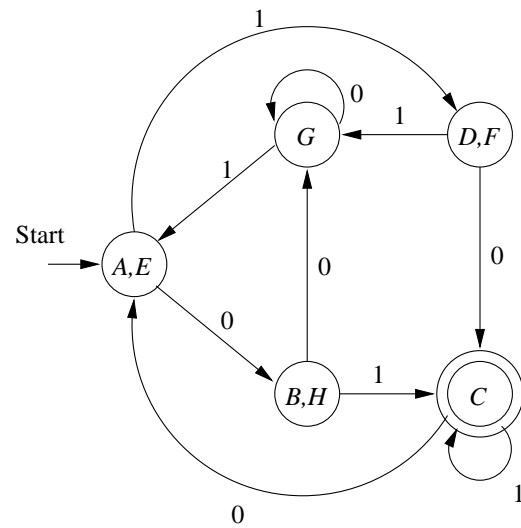
$$\text{Se } p \equiv q \text{ allora } \delta(p, a) \equiv \delta(q, a)$$

Se $\delta(p, a) \not\equiv \delta(q, a)$, allora l'algoritmo concluderebbe $p \not\equiv q$, quindi B è ben definito.

Esempio: Possiamo minimizzare

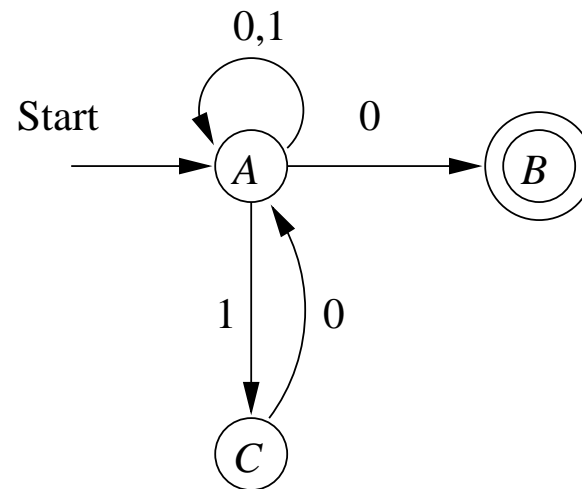


per ottenere



Notare: Non possiamo applicare l'algoritmo a NFA.

Per esempio, per minimizzare



dovrei poter rimuovere lo stato C .

Ma $A \neq C$.