Progamma sintetico

- Nozioni preliminari
- **Automi Finiti**
- Macchine di Turing
- Limiti delle macchine di Turing
- La tesi di Church-Turing
- Le classi P e NP

Un problema classico

- Un uomo viaggia con un lupo, una pecora ed un cavolo
- Vuole attraversare un fiume
- Ha una barca che può contenere solo l'uomo più uno dei suoi possedimenti,
- Il lupo mangia la pecora se soli insieme
- La pecora mangia il cavolo se soli insieme
- Come può attraversare fiume senza perdite?

Soluzione come Stringa

Mosse possono essere rappresentate da simboli

- Uomo attraversa con lupo (*l*)
- Uomo attraversa con pecora (*p*)
- Uomo attraversa con cavolo (*c*)
- Uomo attraversa con niente (*n*)

➔Sequenza mosse ⇔ stringa,

Es. soluzione *pnlpcnp*:

- **Prima attraversa *con pecora*,**
- **poi ritorna con *niente*,**
- **poi attraversa con il lupo , …**

Mosse ⇔transizioni di stato

- Ogni mossa porta puzzle da uno stato ad un'altro
- Es. Mossa p provoca transizione

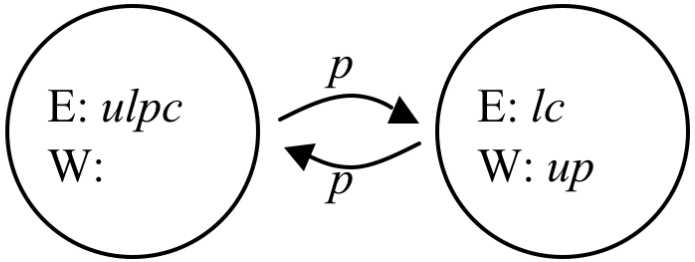
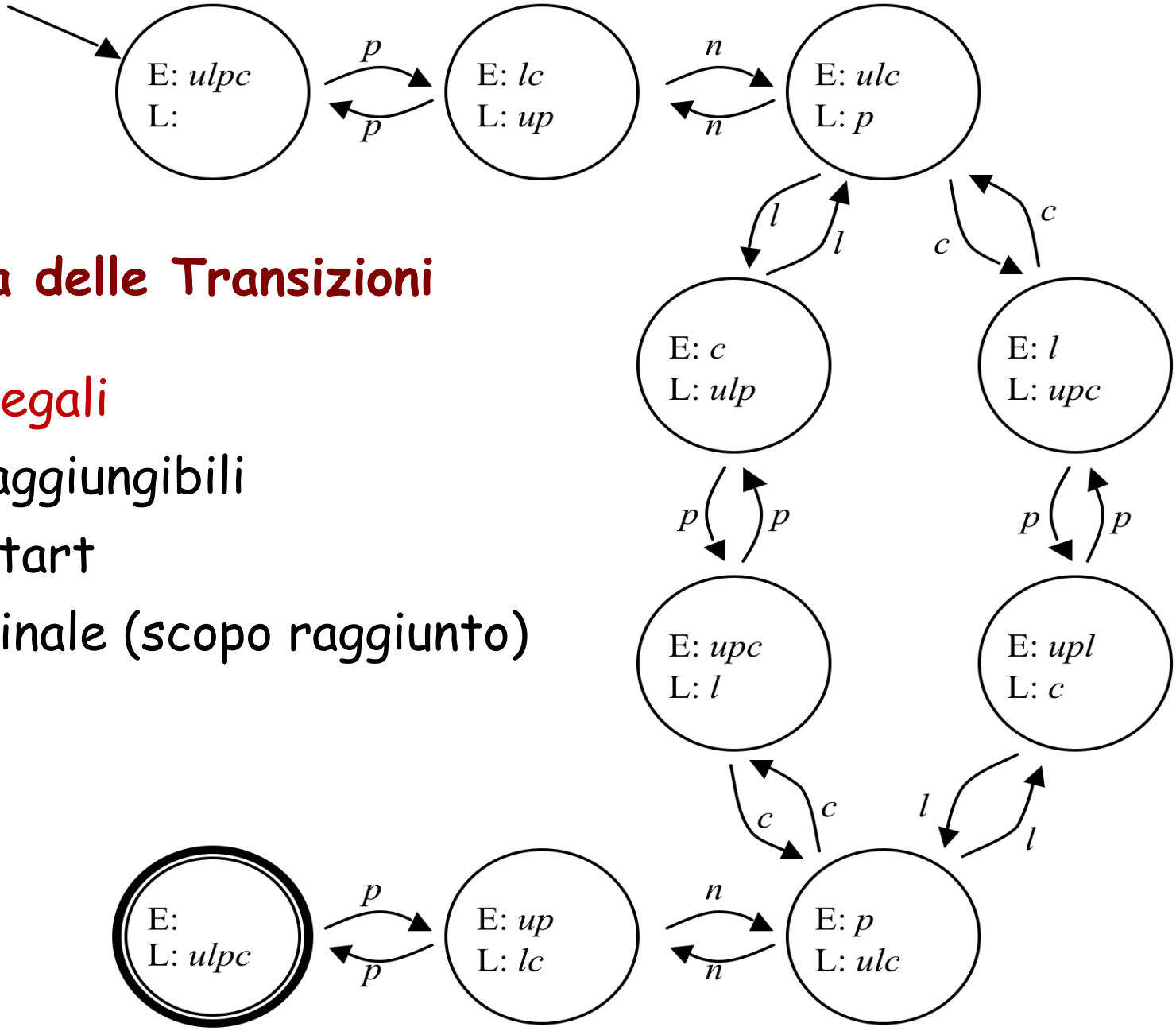


Diagramma delle Transizioni

- Mosse legali
- Stati raggiungibili
- Stato start
- Stato finale (scopo raggiunto)



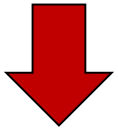
Linguaggio delle soluzioni

- Cammino $\Leftrightarrow$ qualche $x \in \{l,p,c,n\}^*$
- Sequenza di mosse è una soluzione $\Leftrightarrow$ stringa corrispondente è un cammino dallo stato iniziale allo stato finale del diagramma
- diagramma definisce il *linguaggio delle soluzioni*:

  $\{x \in \{l,p,c,n\}^*$ | iniziando in stato start e seguendo transizioni di x, terminano nello stato finale$\}$
- Nota: Linguaggio infinito

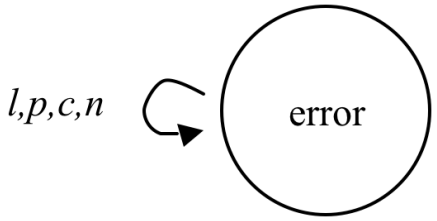
Nota:
Dal Problema al Linguaggio

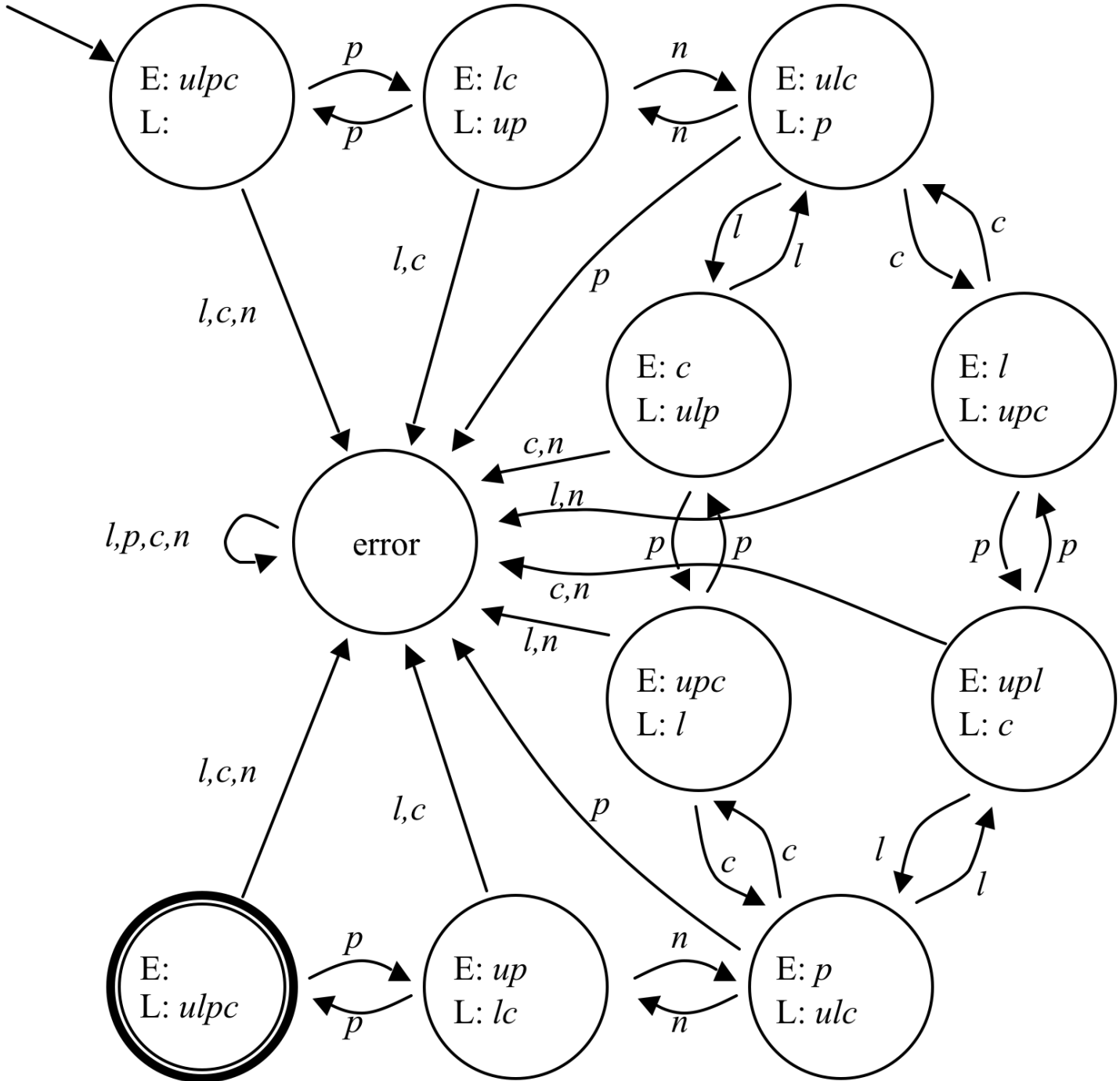
- Problema: trovare sequenza di mosse che permette di trasportare capra, cavolo e lupo



- Linguaggio: insieme delle stringhe sull'alfabeto {*l*,*p*,*c*,*n*}. Le stringhe corrispondono a sequenze di mosse. Tra esse cerchiamo una stringa che corrisponde ad una soluzione del problema.

- Per alcune stringhe automa non sa cosa fare
- Vogliamo automi che sanno sempre cosa fare
- Es. Usiamo stato addizionale




E: ulpc
L:
p
p
E: lc
L: up
n
n
E: ulc
L: p
l,c,n
l,c
p
l
l
c
c
E: c
L: ulp
E: l
L: upc
c,n
l,n
l,p,c,n
error
p
p
p
p
c,n
l,n
E: upc
L: l
E: upl
L: c
l,c,n
l,c
p
c
c
l
l
E:
L: ulpc
p
p
E: up
L: lc
n
n
E: p
L: ulc

Automa Completamente Specificato

- Nel diagramma esattamente una transizione per ogni stato e per ogni lettera nell'alfabeto Σ

- Fornisce **procedura computazionale** per decidere se una stringa è una soluzione o meno:
  - Parti nello stato start
  - Segui una transizione per ogni simbolo input
  - Se alla fine arrivi in stato obiettivo accetta altrimenti rifiuta l'input

AUTOMI FINITI

Modello semplice di calcolatore avente una quantità finita di memoria
E' noto come **macchina a stati finiti** o **automa finito**.

Idea di base del funzionamento:

- Input= stringa w su un alfabeto Σ
- Legge i simboli di w da sinistra a destra.
- Dopo aver letto l'ultimo simbolo, l'automa indica se accetta o rifiuta la stringa input w

Alla base di importanti tipi di software, es.:

✓ Compilatori: realizzazione del **parser**, analisi lessicale

Il Parsing o analisi sintattica è il processo di analizzare una stringa in accordo ad una grammatica.

Un parser è un programma, di solito parte di un compilatore, che

- *riceve un input sotto forma di istruzioni di un programma (o comandi online interattivi, tag di codice, …) e*
- *lo divide in parti*
  (per esempio, i sostantivi (oggetti), verbi (metodi) e loro attributi o opzioni)
  che possono poi essere gestiti da altri programmi
  (per esempio, altri componenti di un compilatore).

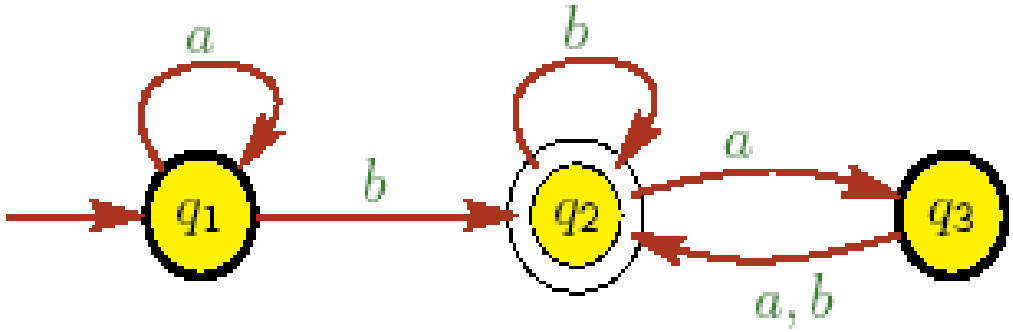
Un parser può anche verificare che è stato fornito tutto l'input necessario.

Alla base di importanti tipi di software, es.:

- ✓ Compilatori: realizzazione del parser, analisi lessicale
- ✓ Software per la progettazione di circuiti digitali
- ✓ **Strumenti per la specifica e la verifica di sistemi a stati finiti (sistemi biologici, protocolli di comunicazione)**
- ✓ **Realizzazione di strumenti di elaborazione testuale**
- ✓ **Ricerca di parole chiave in un file o sul web.**

Automa finito deterministico (DFA)

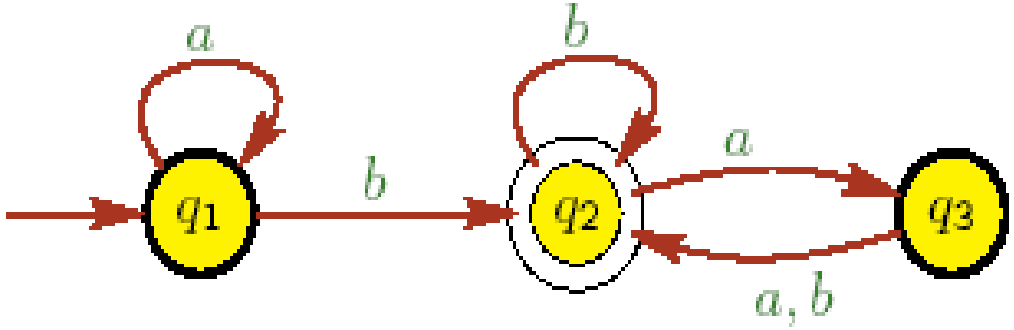
Es: DFA con alfabeto Σ={a, b}:



Stringa arriva in input,
DFA legge 1 simbolo alla volta dal primo all'ultimo
Quindi DFA accetta o rifiuta la stringa

Automa finito deterministico (DFA)

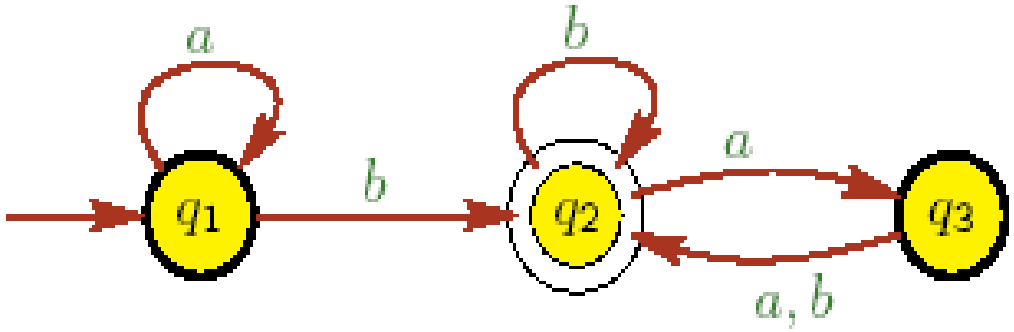
Es: DFA con alfabeto Σ={a, b}:



a	b	b	a	a
q1	q1	q2	q2	q3	q2

Automa finito deterministico (DFA)

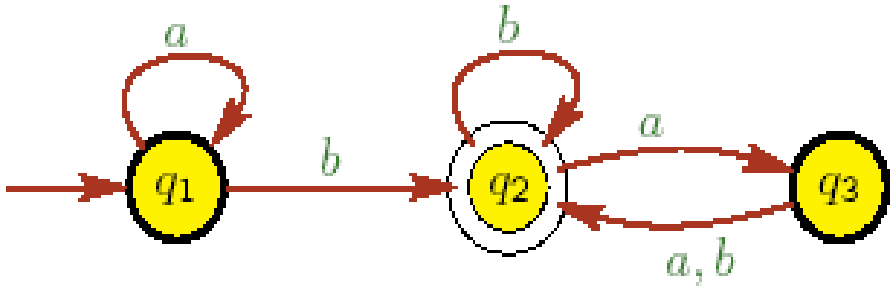
Es: DFA con alfabeto Σ={a, b}:



q1, q2, q3 soni gli **stati**.

q1 è lo **stato start** (ha freccia entrante da esterno)

q2 è uno **stato accetta** (doppio cerchio)

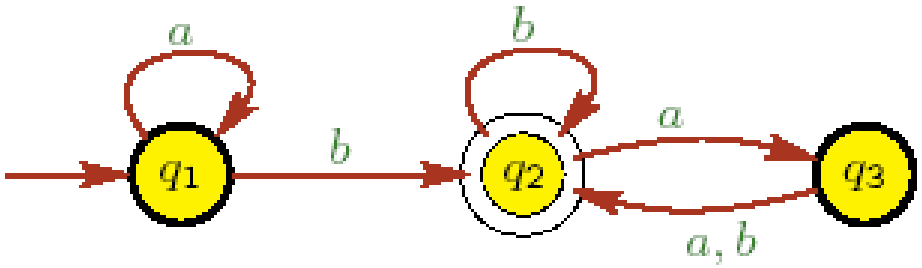


archi dicono come muoversi quando l'automa si trova in uno stato e simbolo di Σ è letto .

Dopo aver letto l'ultimo simbolo:

Se DFA è in uno stato accetta, allora stringa è **accettata**, altrimenti è **rifiutata** .

Nell'esempio:
abaa è accettata
aba è rifiutata
ε è rifiutata

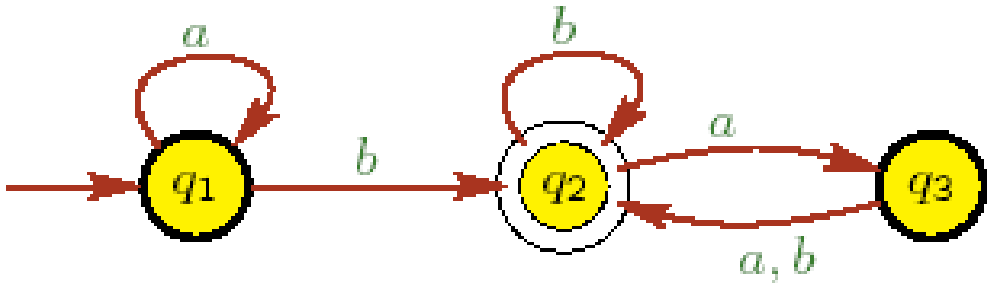


Def. formale di DFA

A **automa finito deterministico** (DFA) è 5-tupla
$M = (Q, \Sigma, f, q_1, F)$,
dove

1. Q è **insieme finito** di stati.
2. Σ è alfabeto, e il DFA processa stringhe su Σ.
3. $f : Q \times \Sigma \to Q$ è la funzione di transizione.
4. $q_1 \in Q$ è lo **stato start**.
5. F (sottoins. di Q) è l'insieme di **stati accetta** (o **finali**).

Nota: DFA chiamato anche semplic. **automa finito (FA)**.



Funzione di transizione di DFA

funzione di transizione f : Q × Σ-> Q :

per ogni stato e per ogni simbolo di input,

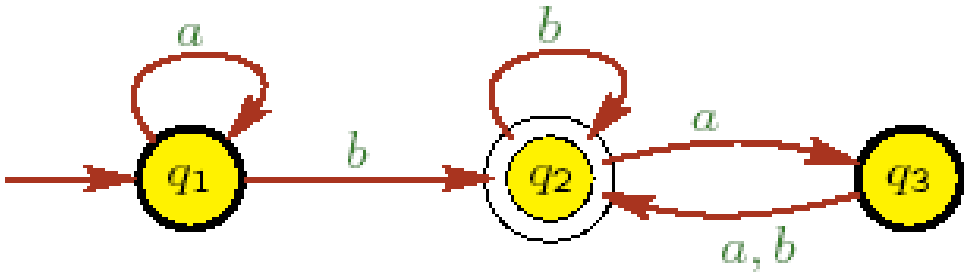
la funzione f dice in quale stato spostarsi.

Cioè, f(r, a) è lo stato che il DFA occupa quando trovandosi nello stato r legge a,

per stato r ϵ Q ed ogni simbolo a ϵ Σ

Esiste esattamente un arco uscente da r con label a

quindi la macchina è **deterministica**.



Es. di DFA

M = (Q,Σ, f, q_1, F) con

Q = {q_1, q_2, q_3}

Σ={a, b}

f è descritta da

	a	b
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

q_1 è lo stato start

F = {q_2}.

Come un DFA Computa

- **Riceve in input stringhe di simboli di alfabeto A**

  Inizia in stato start.

  Legge la stringa un simbolo alla volta, partendo da sinistra

  il simboli letto determina la sequenza di stati visitati.

- **Processo termina dopo lettura dell'ultimo simbolo**

  Dopo aver letto intera stringa input:

  Se DFA termina in stato accetta, allora stringa è accettata

  altrimenti , è rifiutata .

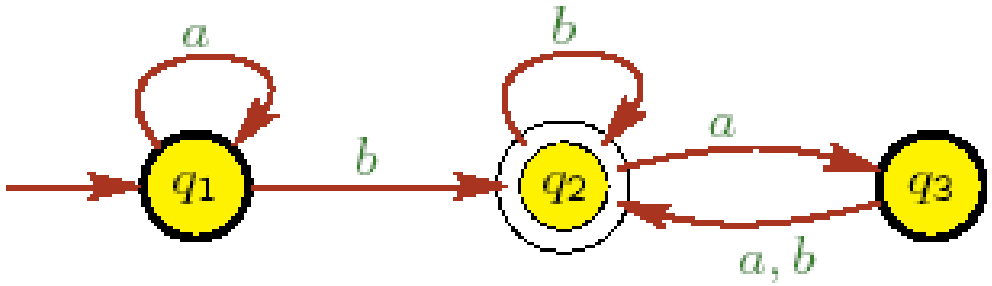
Definizione formale funzionamento

Sia $M = (Q, \Sigma, f, q_0, F)$ un DFA.

Considera stringa $w = w_1 w_2 \ldots w_n$ su Σ, dove ogni w_i in Σ.

M **accetta** w se esiste sequenza stati $r_0, r_1, \ldots, r_n$ in Q tali che:

1. $r_0 = q_0$ (primo stato della sequenza è quello iniziale)
2. r_n in F (ultimo stato in sequenza è uno stato accetta)
3. $f(r_{i-1}, w_i) = r_i$, per ogni $i = 1, 2, \ldots, n$ (sequenza di stati corrisponde a transizioni valide per la stringa w).



ES.

abaa accettata

⇔ Esiste sequenza stati q1,q1,q2,q3,q2

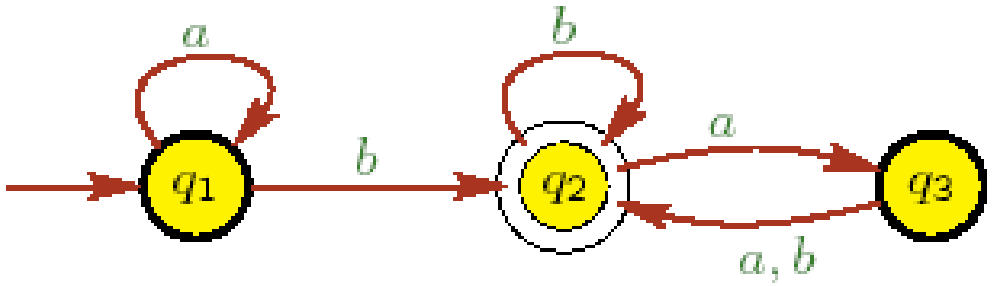
Linguaggio della Machina

Def: Se L è l'insieme di tutte le stringhe che la machina M accetta, allora si dice che

- L è il **Linguaggio della machina** M, e
- M **riconosce** (o **accetta**) L

Scriviamo anche L(M) = L.

Def:
Un Linguaggio è **regolare** se è riconosciuto da qualche DFA.

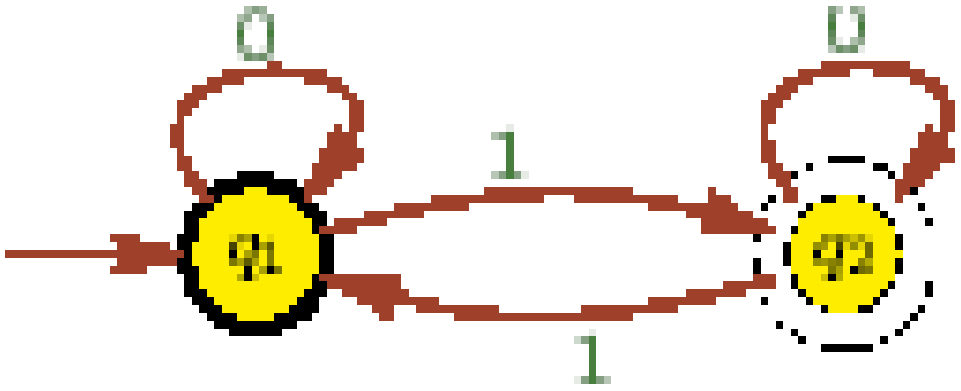


ES.

L(M) è l'insieme di tutte le stringhe sull'alfabeto {a,b} della forma

{a}*{b}{b,aa,ab}*

Es: Si consideri il seguente DFA M_1 con alfabeto $\Sigma=\{0, 1\}$:

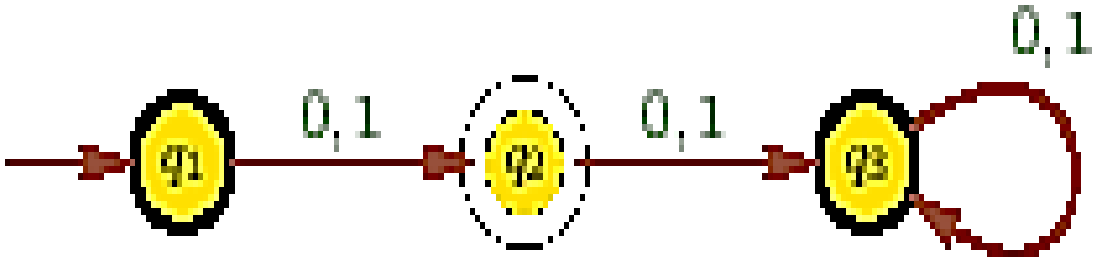


010110 è accettata , ma 0101 è rifiutata .

$L(M_1)$ è il Linguaggio di stringhe su Σ in cui il numero totale di 1 è dispari.

Esercizio: dare DFA che riconosce il Linguaggio di stringhe su Σ con un numero pari di 1

Es: DFA M_2 con alfabeto $\Sigma=\{0, 1\}$:



Nota:

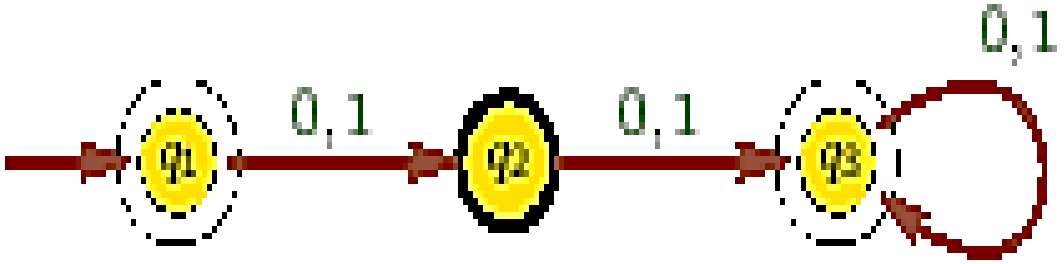
$L(M_2)$ è Linguaggio su A che ha lunghezza 1, cioè

$$L(M_2) = \{w \text{ in } A^* \mid |w| = 1\}$$

Si ricordi che $C(L(M_2))$, il complemento di $L(M_2)$, è l'insieme di stringhe su A che non sono in $L(M_2)$.

Domanda: DFA che riconosce $C(L(M_2))$?

Es: Si consideri il seguente DFA M_3 con alfabeto $A=\{0, 1\}$



$L(M_3)$ è il Linguaggio su Σ che **non** ha lunghezza 1,

- Più di uno stato accetta.
- Stato start anche stato accetta.

DFA accetta ε sse stato start è anche stato accetta.

La funzione di transizione f puo' essere estesa a f* che opera su stati e stringhe (invece che su stati e simboli)

$f^*(q,\varepsilon) = q$

$f^*(q,xa) = f(f^*(q, x), a)$

Il linguaggio accettato da M è quindi

$L(M) = \{ w : f^*(q_0, w) \in F \}$

ESERCIZI

Fornire DFA per i seguenti linguaggi sull'alfabeto {0, 1}:

1. Insieme di tutte le stringhe che terminano con 00

2. Insieme di tutte le stringhe con tre zeri consecutivi

3. Insieme delle stringhe con 011 come sottostringa

4. Insieme delle stringhe che cominciano o finiscono (o entrambe le cose) con 01

DFA per Complemento

In generale, dato DFA M per Linguaggio L,

possiamo costruire DFA M' per C(L) da M

- rendendo tutti gli stati accetta in M non-accetta in M',
- rendendo tutti stati non-accetta in M stati accetta in M'.

Più formalmente,

Se Linguaggio L su alfabeto Σ ha un DFA

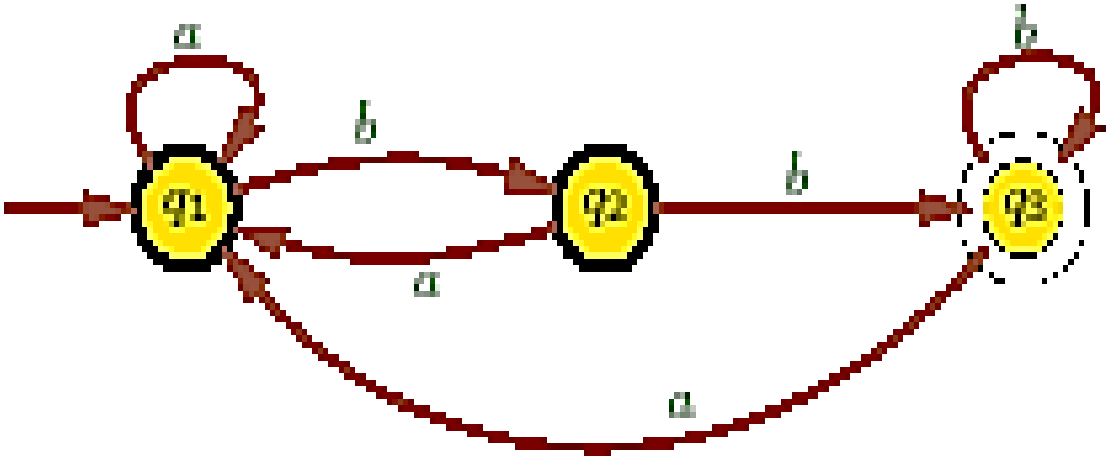
$M = (Q, \Sigma, f, q_1, F)$.

allora, DFA per il complemento di L è

$$M = (Q, \Sigma, f, q_1, Q-F).$$

Esercizio: Perchè funziona?

Es: Si consideri il seguente DFA M_4 con alfabeto Σ={a, b} :

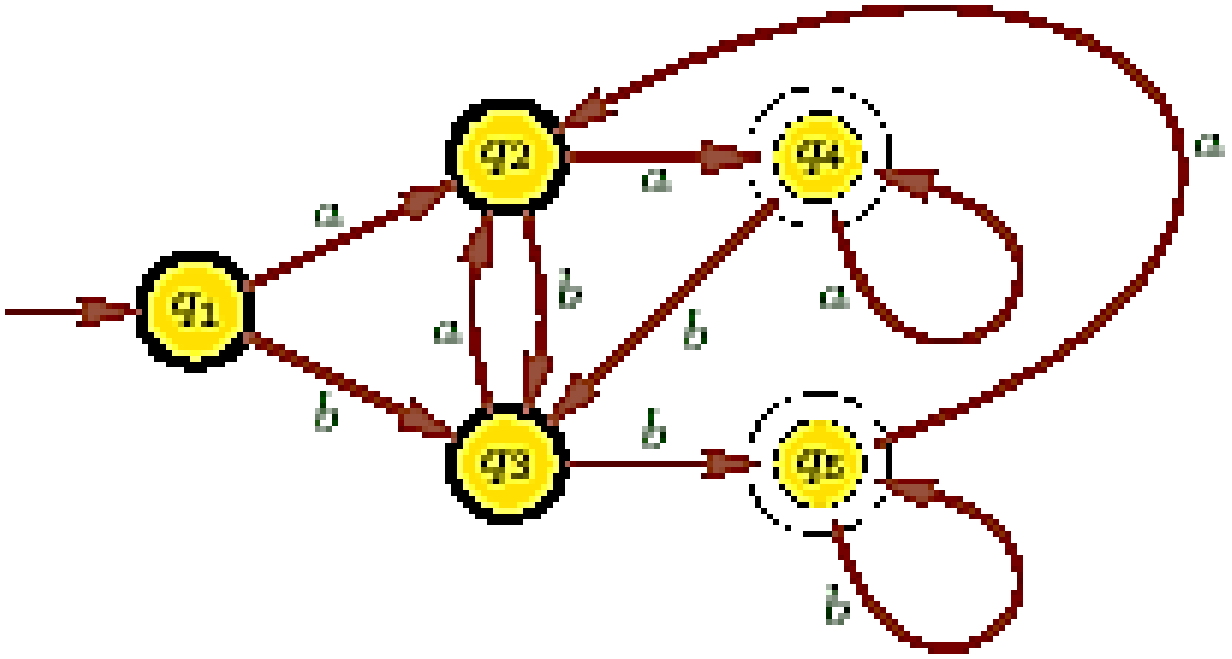


$L(M_4)$: Linguaggio di stringhe su Σ che terminano con bb,

Cioè

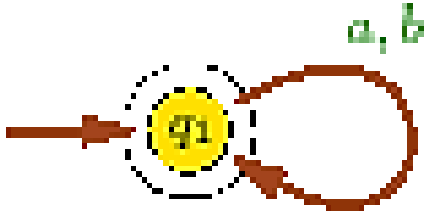
$L(M_4)$ = {w in Σ | w = sbb per qualche stringa s}

Es: Si consideri il seguente DFA M_5 con alfabeto $\Sigma=\{a, b\}$



$L(M_5)$ = {w | w = saa o w = sbb per qualche stringa s}.

Si consideri il seguente DFA M_6 con alfabeto $\Sigma=\{a,b\}$

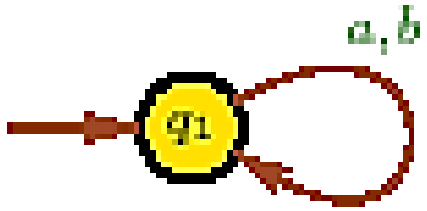


Accetta tutte le possibili stringhe su Σ, cioè

$L(M_6) = \Sigma^*$

In generale, ogni DFA in cui tutti stati sono stato accetta riconosce il linguaggio Σ^*

Si consideri il seguente DFA M6 con alfabeto Σ={a,b}

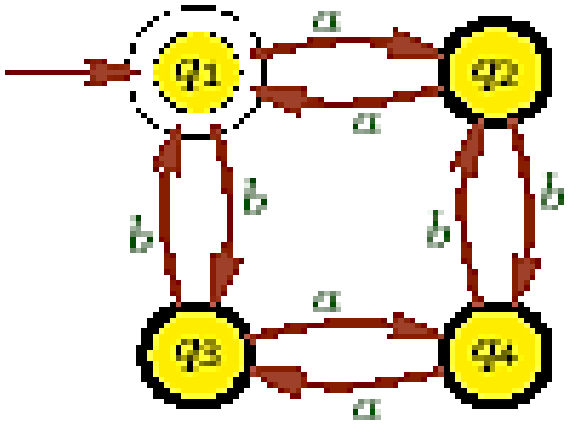


DFA non accetta stringhe su Σ, cioè

L(M7) = ϕ

In generale, un DFA può non avere stati accetta

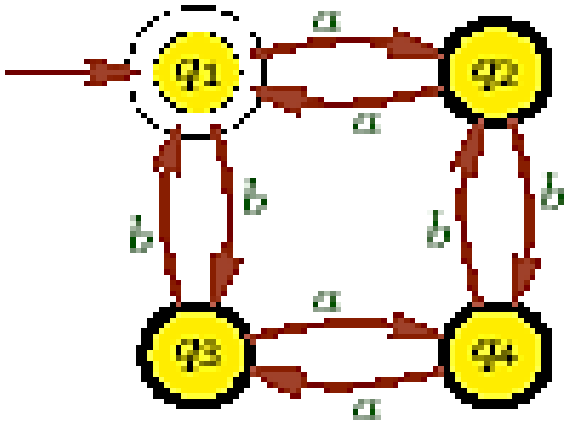
Es: Si consideri il seguente DFA M_8 con alfabeto $\Sigma=\{a, b\}$:



- Ogni a muove verso destra o sinistra.
- Ogni b muove verso l'alto o il basso
- DFA riconosce il Linguaggio di stringhe su Σ

...

Es: Si consideri il seguente DFA M_8 con alfabeto $\Sigma=\{a, b\}$:



- Ogni a muove verso destra o sinistra.
- Ogni b muove verso l'alto o il basso
- DFA riconosce il Linguaggio di stringhe su Σ con numero pari di a e numero pari di b.

Operazioni su linguaggi

Siano A e B Linguaggi.

Unione: $A \cup B = \{w \mid w \in A \text{ o } w \in B\}$.

Concatenazione: $AB = \{vw \mid v \in A, w \in B\}$.

Kleene star: $A^* = \{w_1 w_2 \cdots w_k \mid k \geq 0 \text{ e ogni } w_i \in A\}$.

Una collezione S di oggetti è chiusa per un operazione f se applicando f a membri di S, f restituisce oggetto in S.

Es. N = {0, 1, 2, . . .} chiuso per addizione, non per sottrazione

Abbiamo visto che dati DFA M_1 per Linguaggio L, possiamo costruire DFA M_2 per Linguaggio complemento L':

Rendi tutti stati accetta in M_1 in non-accetta in M_2.

Rendi tutti stati non-accetta in M_1 in accetta in M_2.

Quindi L regolare ➔ C(L) regolare.

Teorema. L' insieme dei linguaggi regolari è chiuso per l'operazione di complemento.

Insieme linguaggi regolari chiuso per l' unione

Teorema

La classe dei linguaggi regolari è chiusa per l'unione.
cioè, se L_1 e L_2 sono linguaggi regolari, allora lo è anche
$L_1 \cup L_2$.

Dim. Idea:

L_1 ha DFA M_1.

L_2 ha DFA M_2.

w in $L_1 \cup L_2$ sse w è accettata da M_1 oppure M_2.

Serve DFA M_3 che accetta w sse w accettata da M_1 o M_2.

Costruiamo M_3 tale

1.da tener traccia di dove l' input sarebbe se fosse contemporaneamente in input a M_1 e M_2.

2. accetta stringa sse M1 oppure M2 accetta.

Siano L1 e L2 definiti su stesso alfabeto Σ.
DFA M_1 riconosce L_1, dove $M_1 = (Q_1, \Sigma, f_1, q_1, F_1)$.
DFA M_2 riconosce L_2, dove $M_2 = (Q_2, \Sigma, f_2, q_2, F_2)$.

Costruiamo DFA $M3 = (Q_3, \Sigma, f_3, q_3, F_3)$:

- $Q_3 = Q_1 \times Q_2 = \{ (x, y) \mid x \text{ in } Q_1, y \text{ in } Q_2 \}$.
- Alfabeto di M_3 è Σ.
- M3 ha funzione di transizione $f_3 : Q_3 \times \Sigma \rightarrow Q_3$ t.c.
  per ogni x in Q_1, y in Q_2, a in Σ,
  $$f_3((x, y), a) = (f_1(x, a), f_2(y, a)) .$$
- Lo stato start di M_3 è $q_3 = (q1, q2)$ in Q_3.
- L' insieme di stati accetta di M_3 è
  $$F_3 = \{ (x, y) \text{ in } Q_3 \mid x \text{ in } F_1 \text{ o } y \text{ in } F_2 \}$$

- **M_3 è automa che riconosce UNIONE**

- Poichè $Q_3 = Q_1 \times Q_2$,

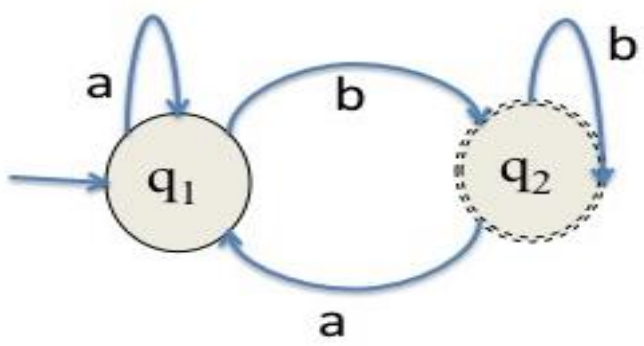
numero di stati in M_3 è $|Q_3| = |Q_1| \cdot |Q_2|$.

Quindi, $|Q_3|$ è **finito** poichè $|Q_1|$ e $|Q_2|$ sono finiti

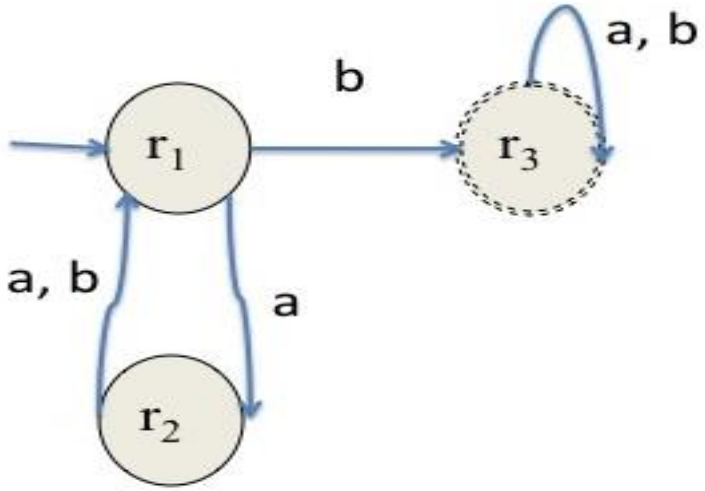
M_3 è DFA per UNIONE

Es: Si considerino i seguenti DFA e linguaggi su $\Sigma=\{a, b\}$:
DFA M1 riconosce linguaggio A1 = L(M1)
DFA M2 riconosce linguaggio A2 = L(M2)

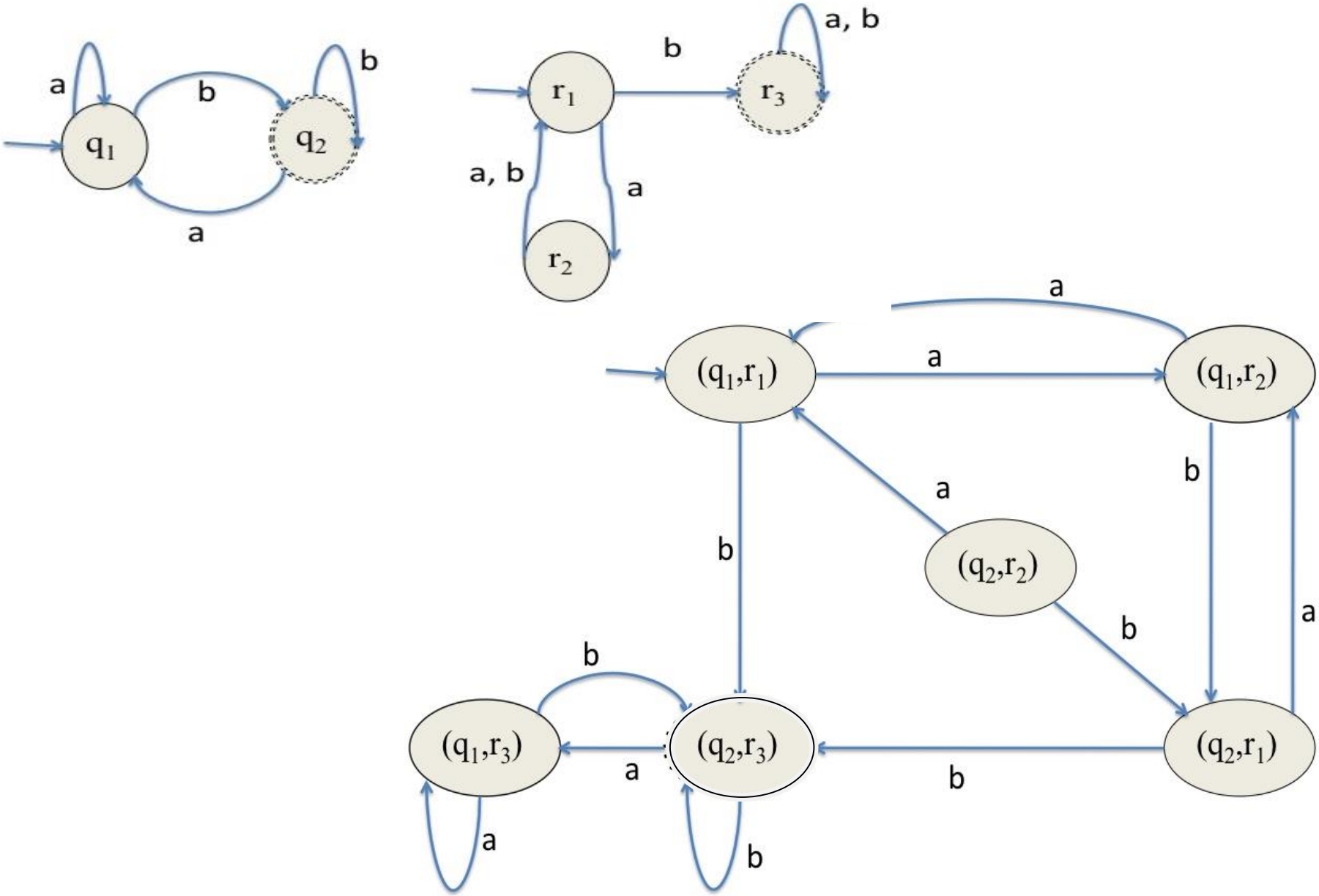
DFA M1 per A1



DFA M2 per A2



Vogliamo DFA per l'unione di A1 e A2.



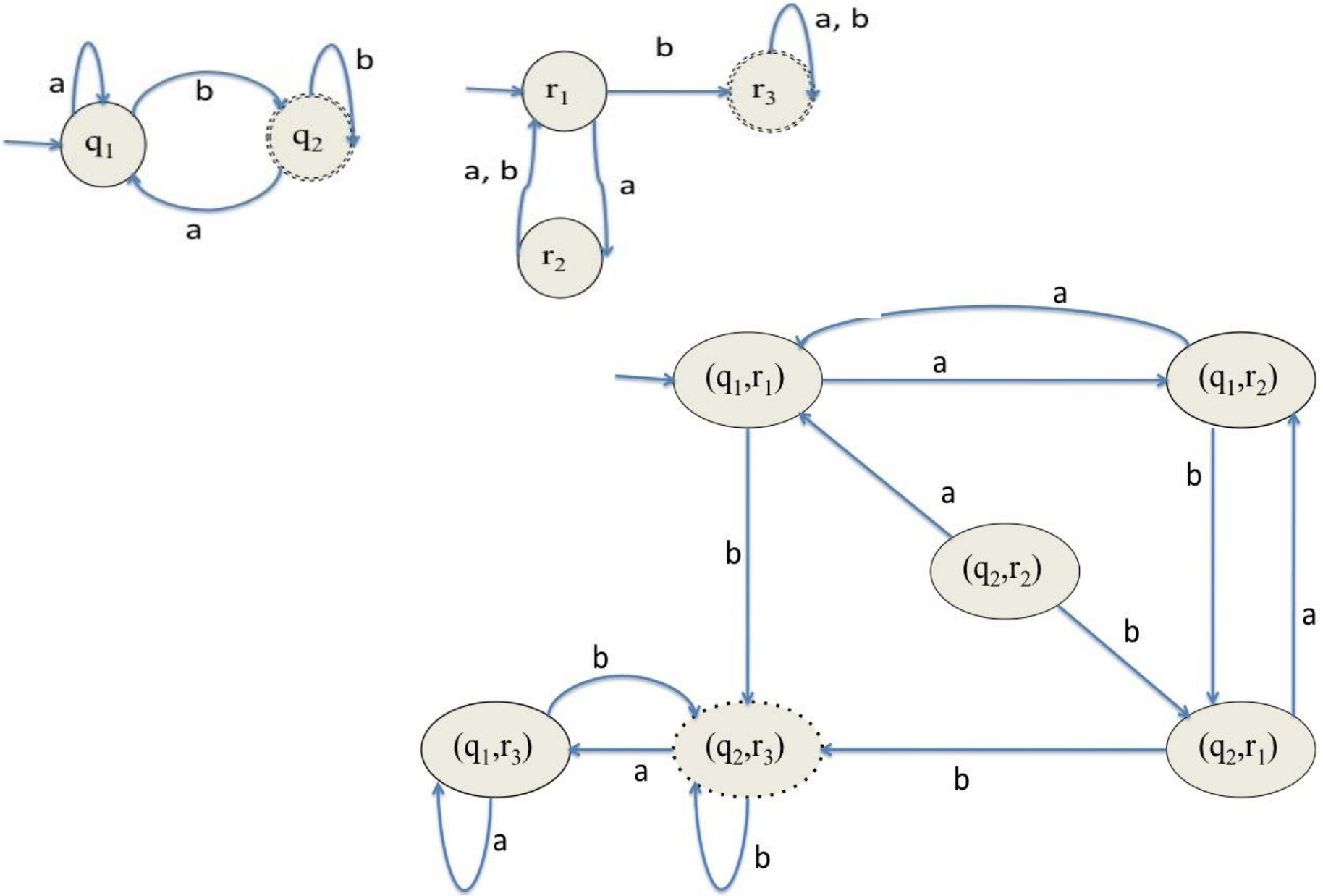
Stati finali: (q_2, r_1), (q_2, r_2), (q_2, r_3), (q_1, r_3)

I linguaggi regolari sono chiusi per l'intersezione

Teorema La classe dei linguaggi regolari è chiusa per l'intersezione. Cioè, se L_1 e L_2 sono linguaggi regolari, allora lo è $L_1 \cap L2$.

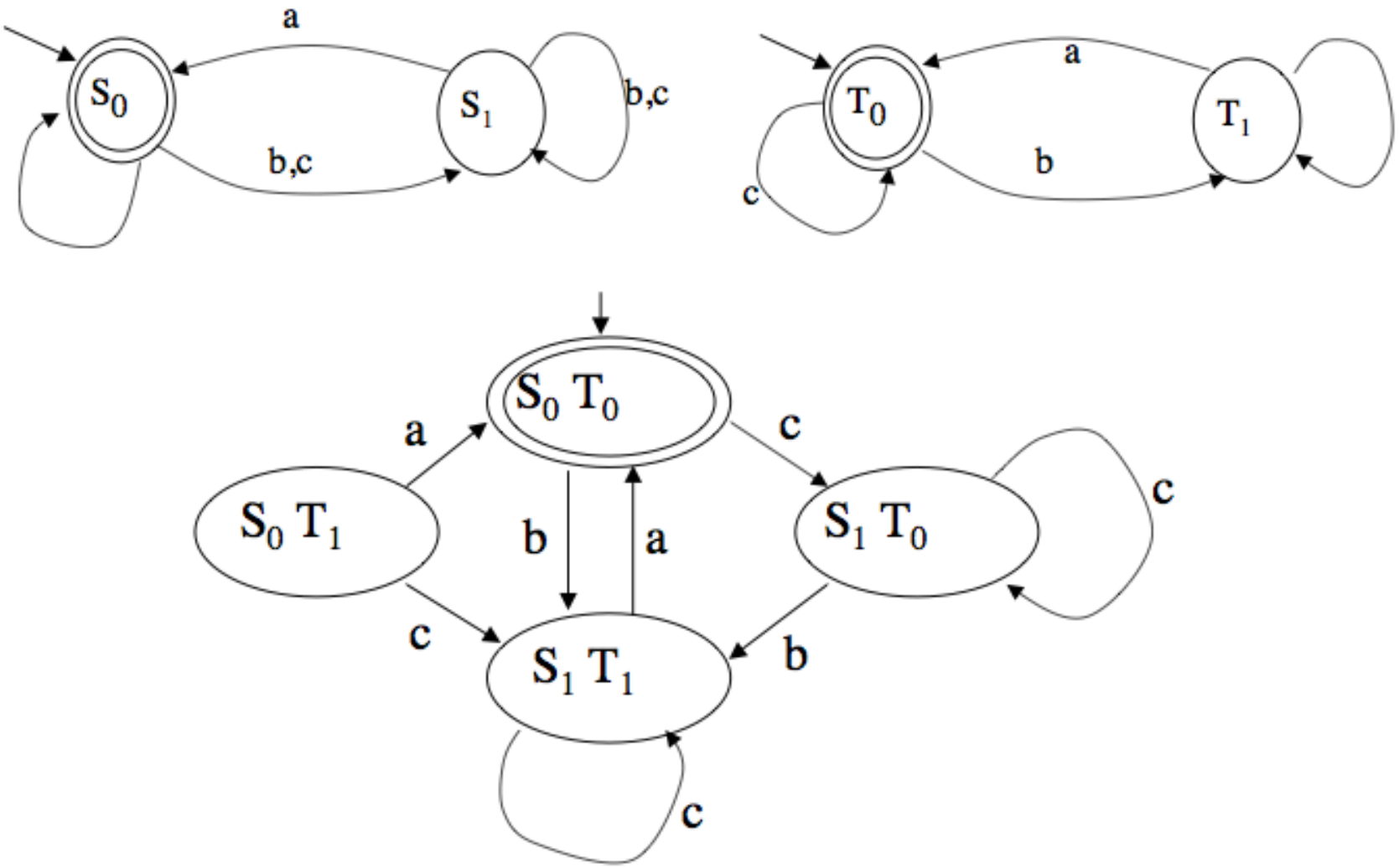
Dim. Idea:

- L_1 ha DFA M_1.
- L_2 ha DFA M_2.
- $w \in L_1 \cap L_2$ sse w è accettato sia da M_1 che da M_2.
- Si vuole DFA M_3 che accetta w sse w è accettata da M_1 e M_2.
- Costruiamo M_3 che contemporaneamente mantiene traccia dello stato in cui si troverebbero sia M_1 che M_2.
- Accetta stringa sse sia M_1 che M_2 accettano



Stato finale: (q_2, r_3)

Es.



I linguaggi regolari sono chiusi per l'intersezione

Teorema La classe dei linguaggi regolari è chiusa per l'intersezione. Cioè, se L_1 e L_2 sono linguaggi regolari, allora lo è $L_1 \cap L2$.

Dim.:

- Si vuole DFA M_3 che accetta w sse w è accettata da M_1 e M_2.

1. **Fornire la definizione formale di M_3**
2. **Mostrare che**

   M_3 accetta w sse w è accettata sia da M_1 che da M_2.

I linguaggi regolari sono chiusi per Concatenazione

Teorema

La classe dei linguaggi regolari è chiusa per la concatenazione.

Cioè, se A1 e A2 sono linguaggi regolari, allora lo è anche A1 A2.

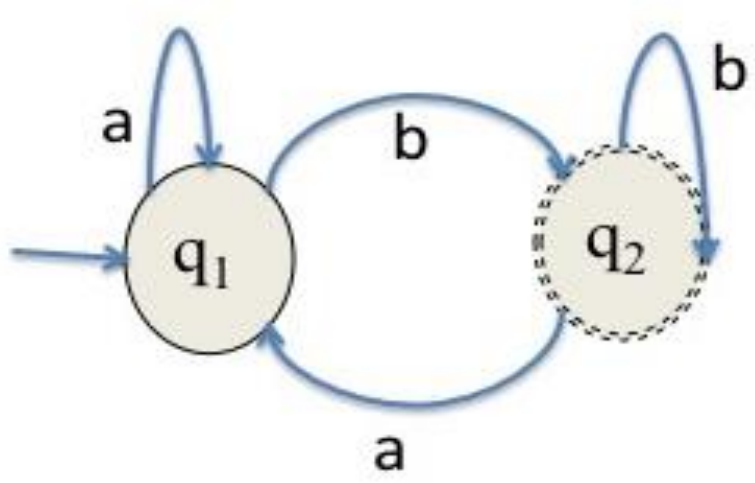
NOTA:

E' possibile (ma laborioso) costruire un DFA per A1 A2 dati i DFA per A1 e A2.

Introduciamo invece un nuovo tipo di macchina

Automi finiti non deterministici

In DFA, lo stato successivo occupato in corrispondenza di un dato input è unicamente determinato



Quindi le macchine sono **deterministiche.**

La funzione di transizione in un DFA è $f : Q \times \Sigma \rightarrow Q$.

Restituisce sempre un singolo stato

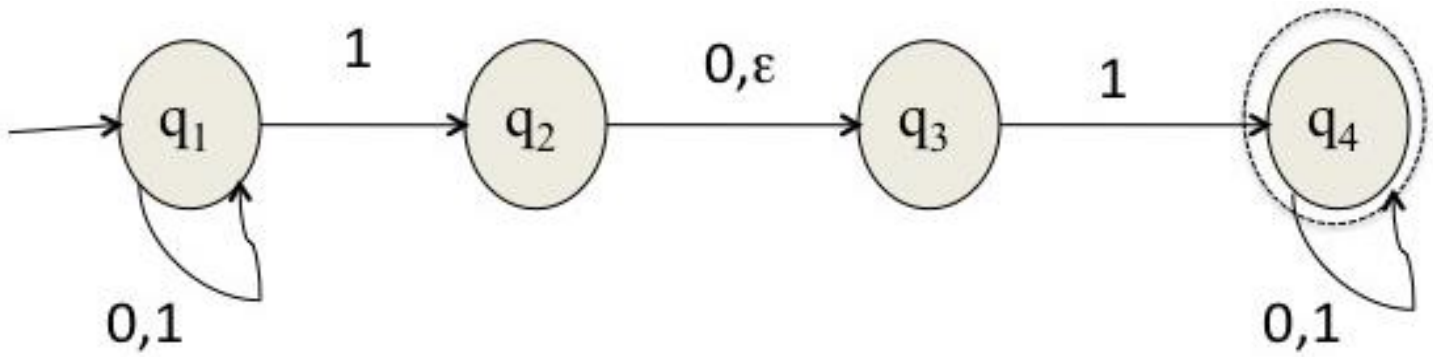
Nondeterminismo

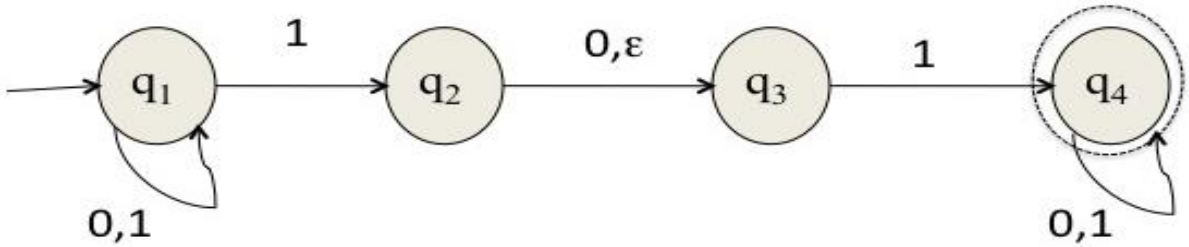
Automi finiti non deterministici (NFA) permettono più scelte per il prossimo stato per un dato input.

Per uno stato q NFA può

- avere più archi uscenti da q labellati con simbolo a, per i vari a $\in \Sigma$;
- prendere ε-edge senza leggere simboli da input.

Es.: NFA N_1 con alfabeto $A=\{0, 1\}$.





Se NFA è in stato con più scelte, (Es., in stato q_1 e input è 1)

- la macchina si divide in più copie di se stessa.
- ogni copia continua computazione indipendent. da altre

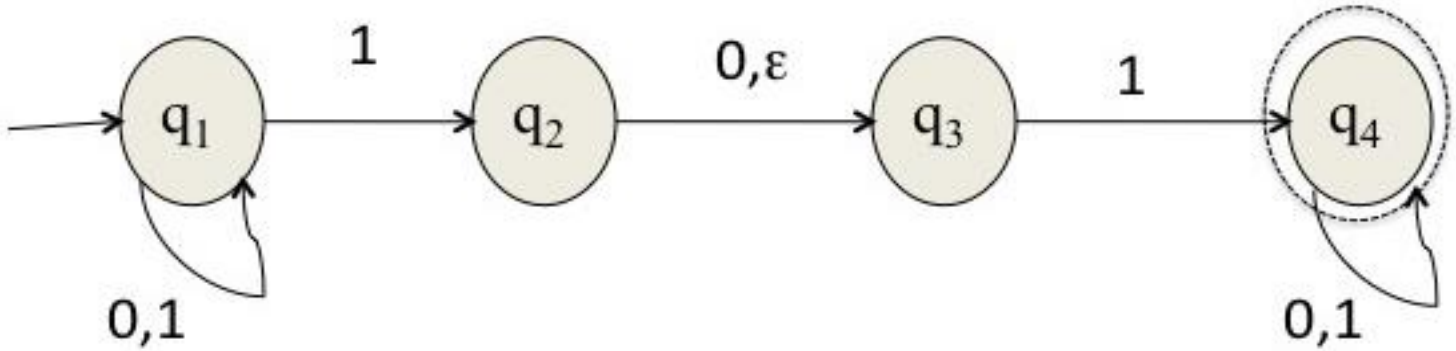
NFA può essere in insieme di stati, invece di singolo stato.

NFA segue ogni possibile computazione in parallelo,

al temine dell'input:

se una copia giunge in stato accetta, NFA accetta la stringa;

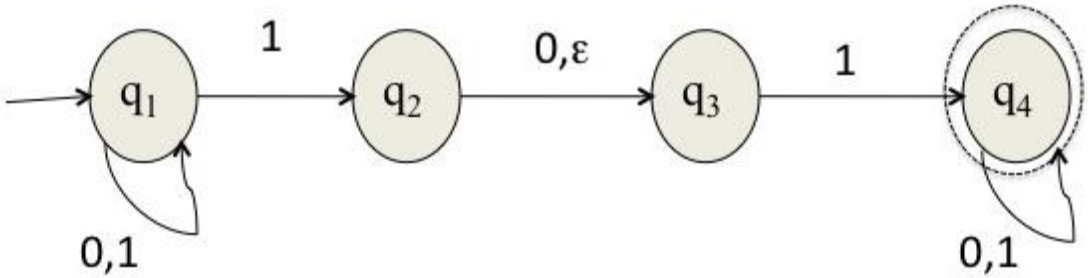
se nessun cammino giunge in stato accetta, allora NFA non accetta la stringa input.

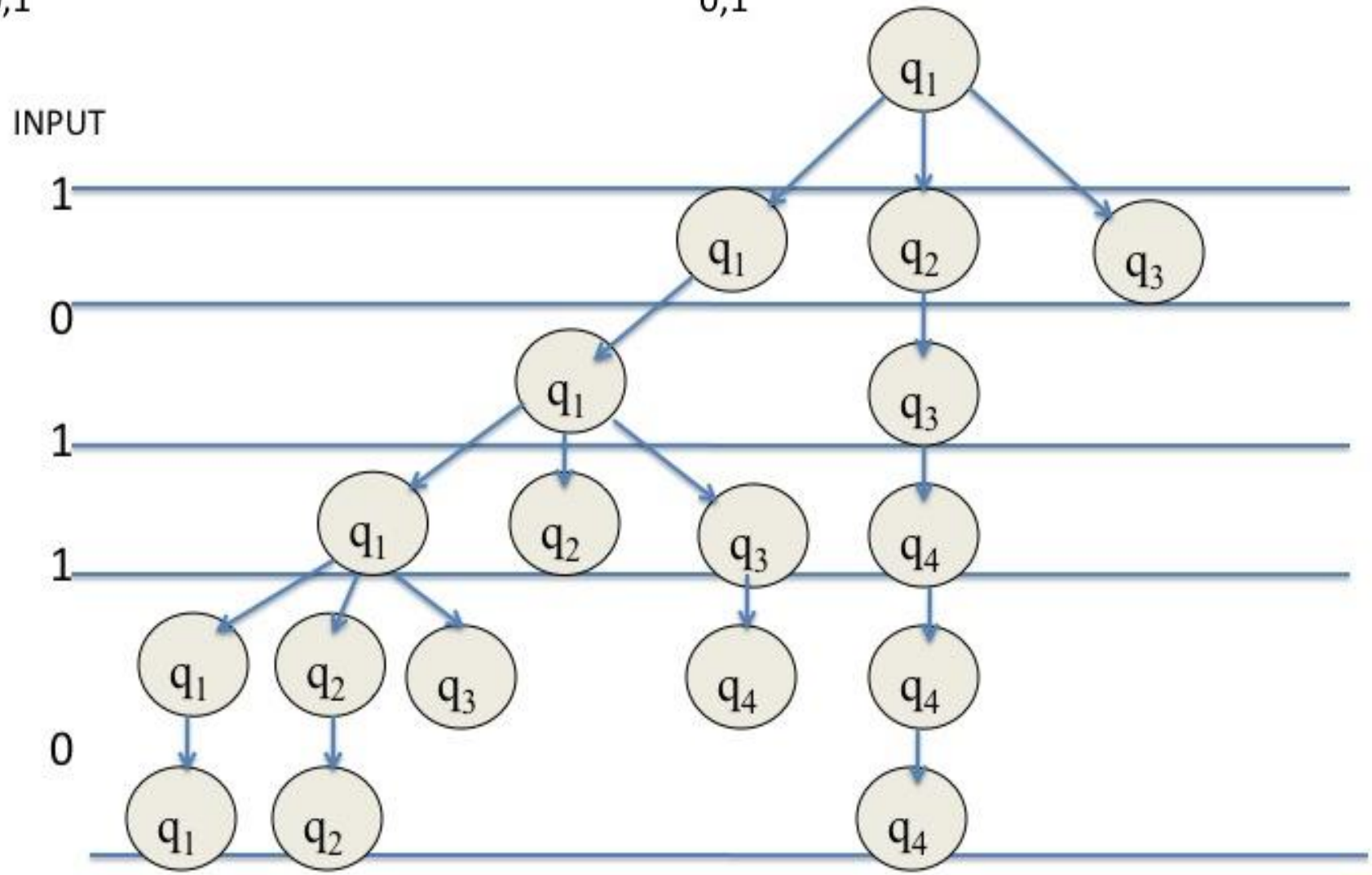


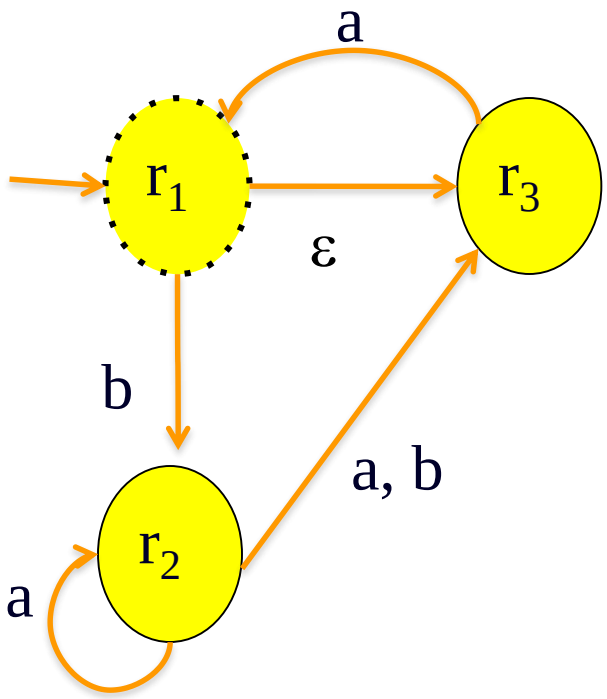
Se in stato con ε-transition, senza leggere input,

- NFA si divide in più copie, ognuna segue una possibile transizione,
- ogni copia continua indipendentemente da altre copie,
- NFA segue ogni possibile cammino in parallelo.
- NFA continua **non deterministicamente** come prima.

Su input 10110 ?


1
0,ε
1
q1
q2
q3
q4
0,1
0,1


INPUT
1
0
1
1
0
q1
q1
q2
q3
q1
q3
q1
q2
q3
q4
q1
q2
q3
q4
q4
q1
q2
q4



NFA N accetta stringhe ε, a, aa, baa, baba, ...

NFA N non accetta stringhe b, ba, bb, ...

Def.di NFA

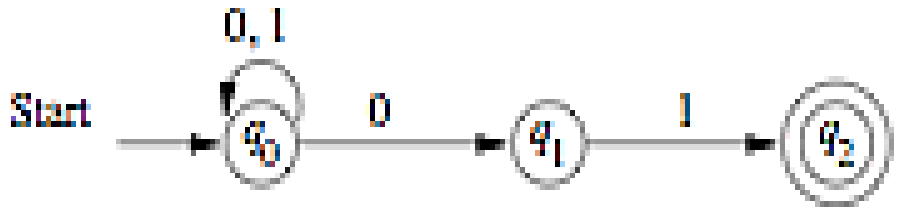
Def.: Per alfabeto Σ, sia Σ_ε ottenuto da Σ aggiungendo ε

Def.: A **NFA** è 5-tupla (Q, Σ, f, q_0, F), con

1. Q insieme di stati
2. Σ alfabeto
3. $f : Q \times \Sigma_\varepsilon \to P(Q)$ funzione di transizione
4. q_0 in Q è stato start
5. F insieme di stati accept.

Nota: La differenza tra DFA e NFA è nella funzione di transizione f:

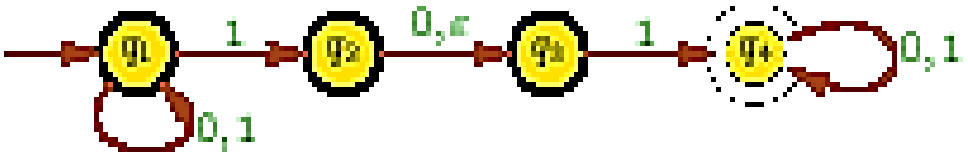
- ammette mosse tipo ε
- Restituisce <u>insieme</u> di stati invece di un solo stato.



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

dove δ e' la funzione di transizione

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



Formal description of above NFA $N = (Q, \Sigma, \delta, q_1, F)$

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{0, 1\}$
- Transition function $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

- q_1 is the start state
- $F = \{q_4\}$

Computazione di NFA

Sia $N = (Q, \Sigma, f, q_0, F)$ un NFA e w una stringa su A allora N **accetta** w se

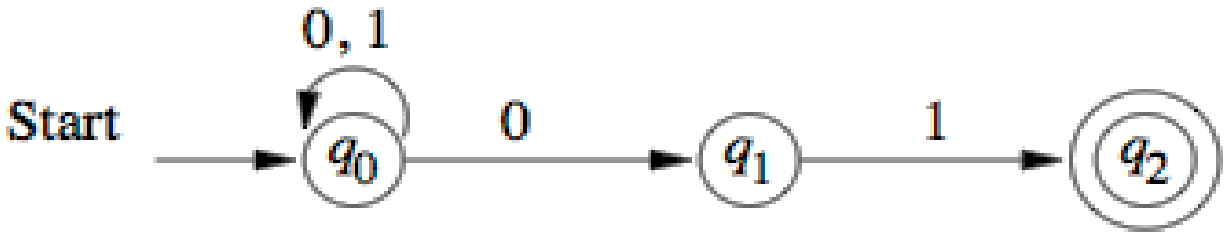
$w = y_1 y_2 \cdots y_m$, dove ogni y_i in $\Sigma\varepsilon$, e
esiste sequenza di stati $r_0, r_1, \ldots, r_m$ in Q t.c.

1. $r_0 = q_0$
2. r_{i+1} in $f(r_i, y_{i+1})$ per ogni $i = 0, 1, 2, \ldots, m - 1$
3. r_m in F

Def.: insieme di stringhe accettate da NFA N è il linguaggio **riconosciuto** da N ed è denotato con L(N).

Nota: ogni DFA è anche un NFA.

Es.



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

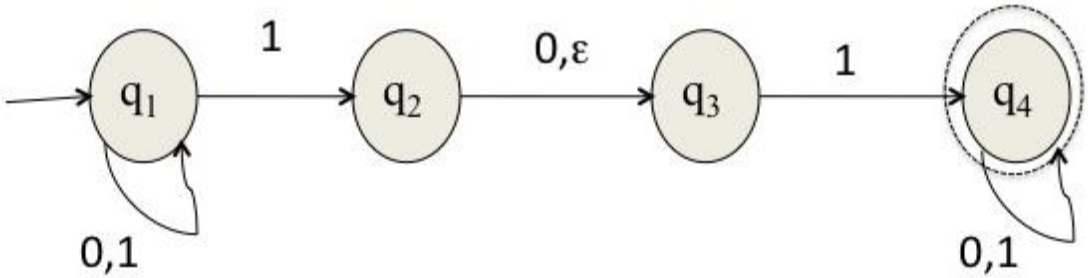
Input 01001?

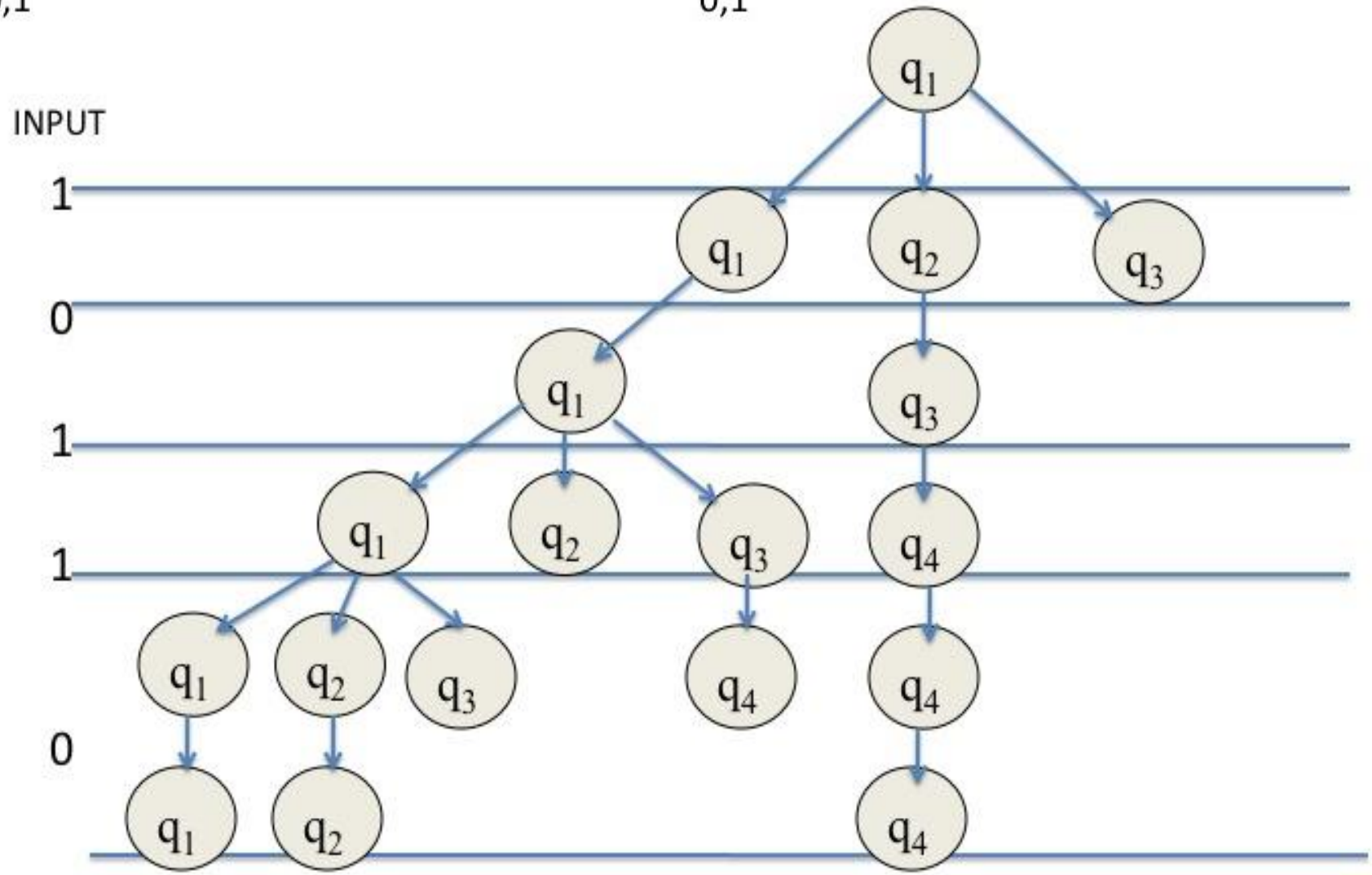
Equivalenza di DFAs e NFAs

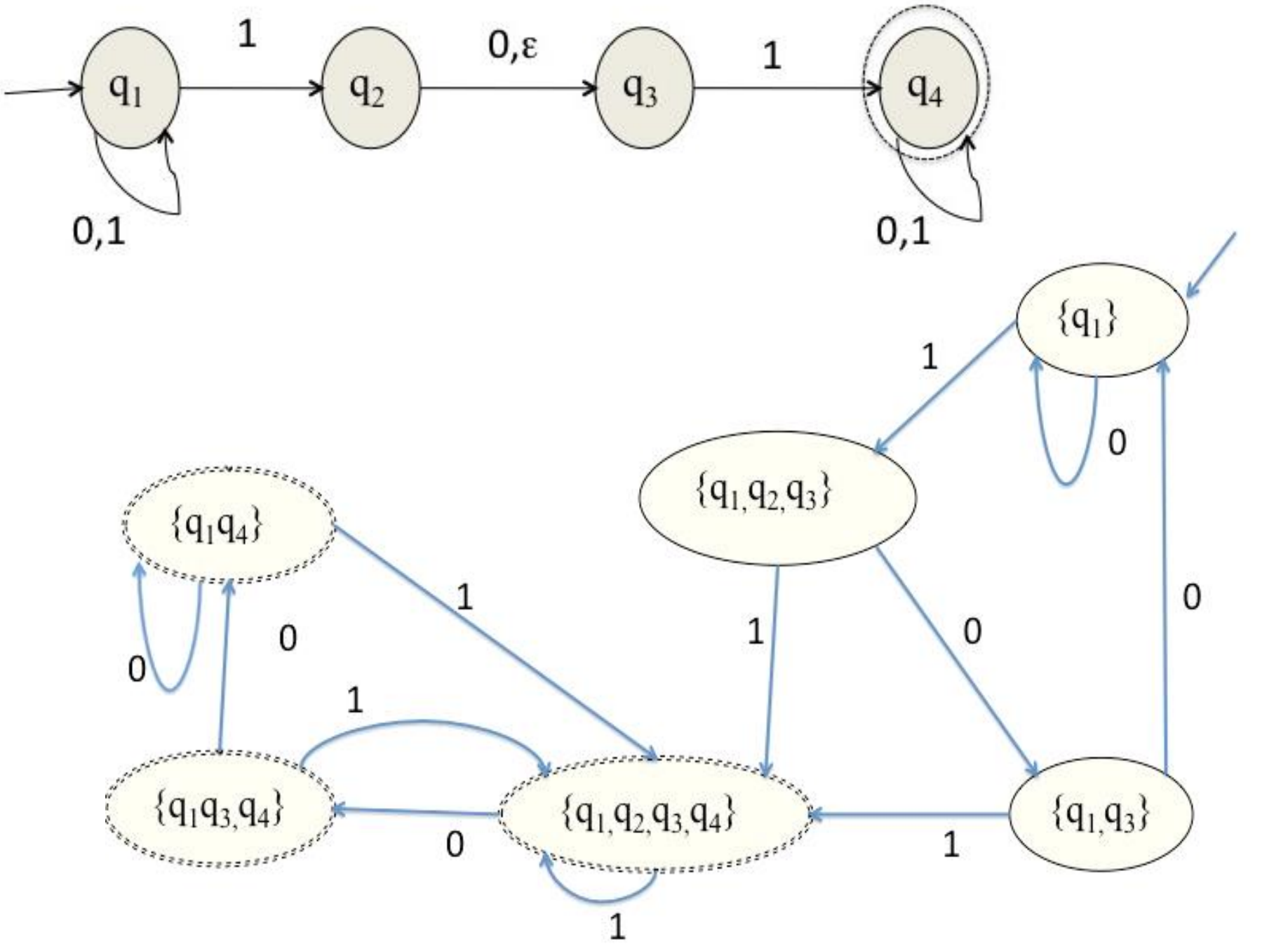
Def.: due macchine sono **equivalenti** se riconoscono lo stesso linguaggio.

Teorema
Ogni NFA N ha un equivalente DFA M;
cioè, se N è un NFA, allora esiste DFA M t.c. L(M) = L(N).

Dim. Costruiamo a partire da N, il DFA M equivalente


1
0,ε
1
q1
q2
q3
q4
0,1
0,1


INPUT
1
0
1
1
0
q1
q1
q2
q3
q1
q3
q1
q2
q3
q4
q1
q2
q3
q4
q4
q1
q2
q4


q1
q2
q3
q4
1
0,ε
1
0,1
0,1
{q1}
{q1,q2,q3}
{q1q4}
{q1q3,q4}
{q1,q2,q3,q4}
{q1,q3}
1
0
1
0
0
1
1
0
0
1
1
0
1

Sia $L = L(N)$ per NFA $N = (Q_N, \Sigma, f_N, q_N, F_N)$

Costruiamo DFA $D=(Q_D, \Sigma, f_D, q_D, F_D)$.

Partiamo da D' che non considera le ε-transition:

$$Q = P(Q_N)$$

$$f(R,a) = \bigcup_{r \in R} f_N(r,a),$$

per ogni $R \in Q$ e $a \in \Sigma$,

$$q = \{q_N\}$$

$$F = \{R \in Q \mid R \cap F_N \neq \emptyset\}$$

- Per ogni stato in $f_{N,}$ se vi è ε-transition allora dobbiamo considerarla
- Definiamo

E(R)=

R U {q| q raggiungibile da stato in R con 1 o più archi lab. ε}

$$Q_D = P(Q_N)$$

$$f_D(R,a) = \bigcup_{r \in R} E(f_N(r,a)),$$

$$\text{per ogni } R \in Q_D \text{ e } a \in \Sigma,$$

$$q_D = E(\{q_N\})$$

$$F_D = \{R \in Q_D \mid R \cap F_N \neq \varnothing\}$$

- Risulta, per ogni $x \in \Sigma^*$,

$$f_D^*(q_D, x) = f_N^*(q_N, x)$$

Si può provare per induzione

- *D* simula *N su ogni* input *x*
- *D* accetta sse *N* accetta
- *L* = *L*(*N*) = *L*(*D*)

Nota $f^*_N(q_N, x)$ = insieme stati raggiungibili da q_N su input x

$f^*_N(q_N, \varepsilon) = E(\{q_N\})$

$f^*_N(q_N, xa) = \bigcup_{r \in f^*_N(q_N, x)} E(f_N(r, a))$

Mostriamo per induzione su |w| che

$f^*_D(\{q_0\},w) = f^*_N(q_0,w)$ [poniamo $q_0=q_N$]

Base: $w = \varepsilon$. L'enunciato segue dalla definizione.

Passo:

$$f^*_D(\{q_0\},xa) = f_D(f^*_D(\{q_0\},x),a) \quad \text{(def.)}$$
$$= f_D(f^*_N(q_0,x),a) \quad \text{(i.i.)}$$
$$= \bigcup_{r \in f^*_N(q_0,x)} E(f_N(r,a)) \quad \text{(def.)}$$
$$= f^*_N(q_0,xa)$$

Linguaggio riconoscuto da NFA ⇔ Regolare

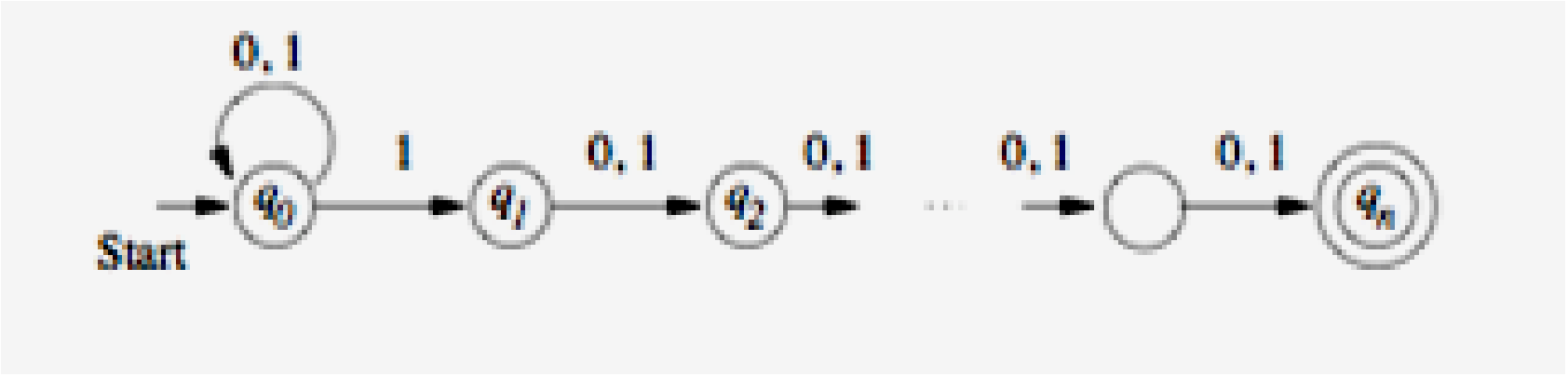
Corollario
Linguaggio L è regolare sse esiste NFA che riconosce A.

Dim.
Se L è regolare, allora esiste DFA
ma ogni DFA è anche un NFA, quindi esiste NFA per L.

Da Teorema precedente, ogni NFA ha equivalente DFA.
Quindi se esiste NFA allora esiste DFA per L

NOTA:

Esiste un NFA N con n + 1 stati che non ha nessun DFA equivalente con meno di 2^n stati



$$L(N) = \{x1c_2c_3 \cdots c_n : x \in \{0,1\}^*, c_i \in \{0,1\}\}$$

Idea:

Supponiamo esista DFA D equivalente con meno di 2^n stati.

D deve ricordare gli ultimi n simboli che ha letto.

Ci sono 2^n sequenze di bit $a_1a_2 \cdots a_n$ da ***ricordare***

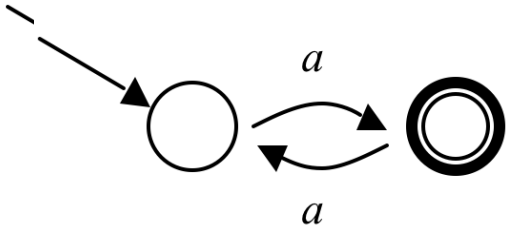
DIMOSTRAZIONE CHIUSURA CONCATENAZIONE

$AB = \{ vw \mid v \in A, w \in B \}$.

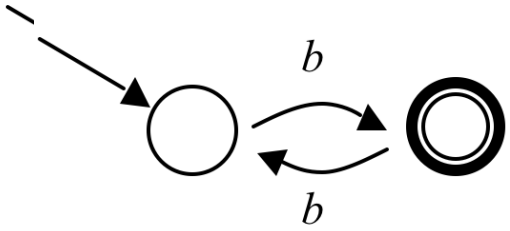
Teorema

La classe dei linguaggi regolari è chiusa per la concatenazione.

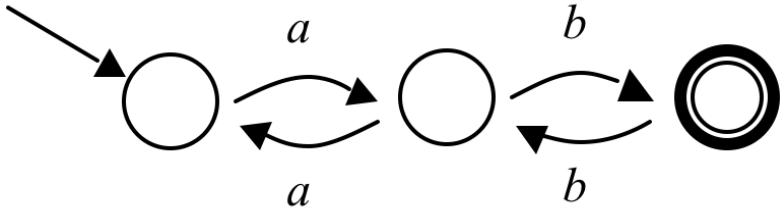
Concatenazione scorretta



$\{ a^n \mid n \text{ dispari} \}$.

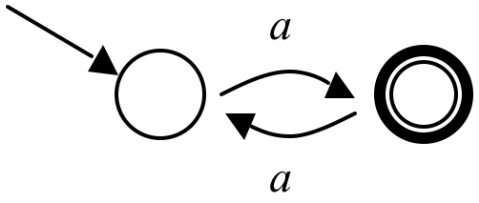


$\{ b^n \mid n \text{ dispari} \}$.

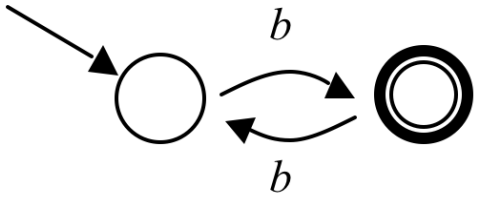


Riconosce abbaab!!

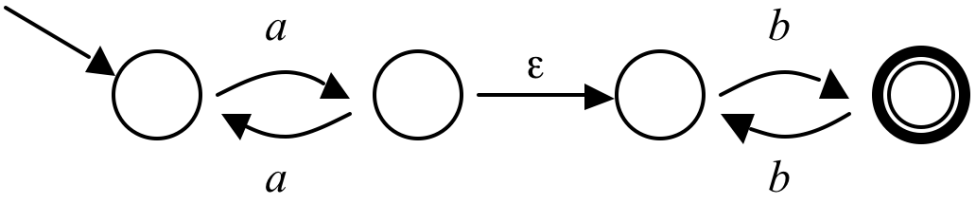
Concatenazione Corretta



{ a^n | n dispari }.

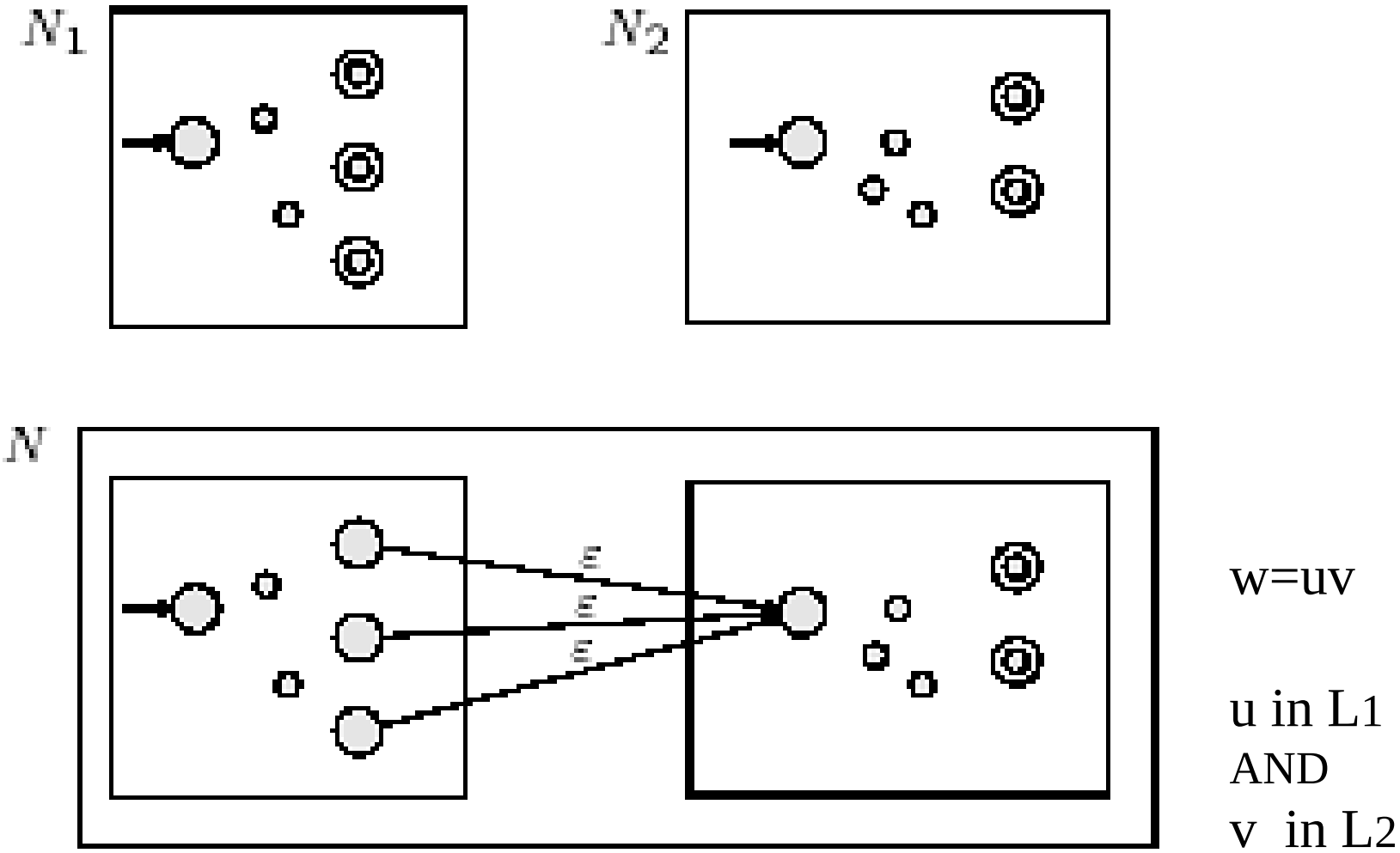


{ b^n | n dispari}.



{xy | $x \in A$ and $y \in B$}

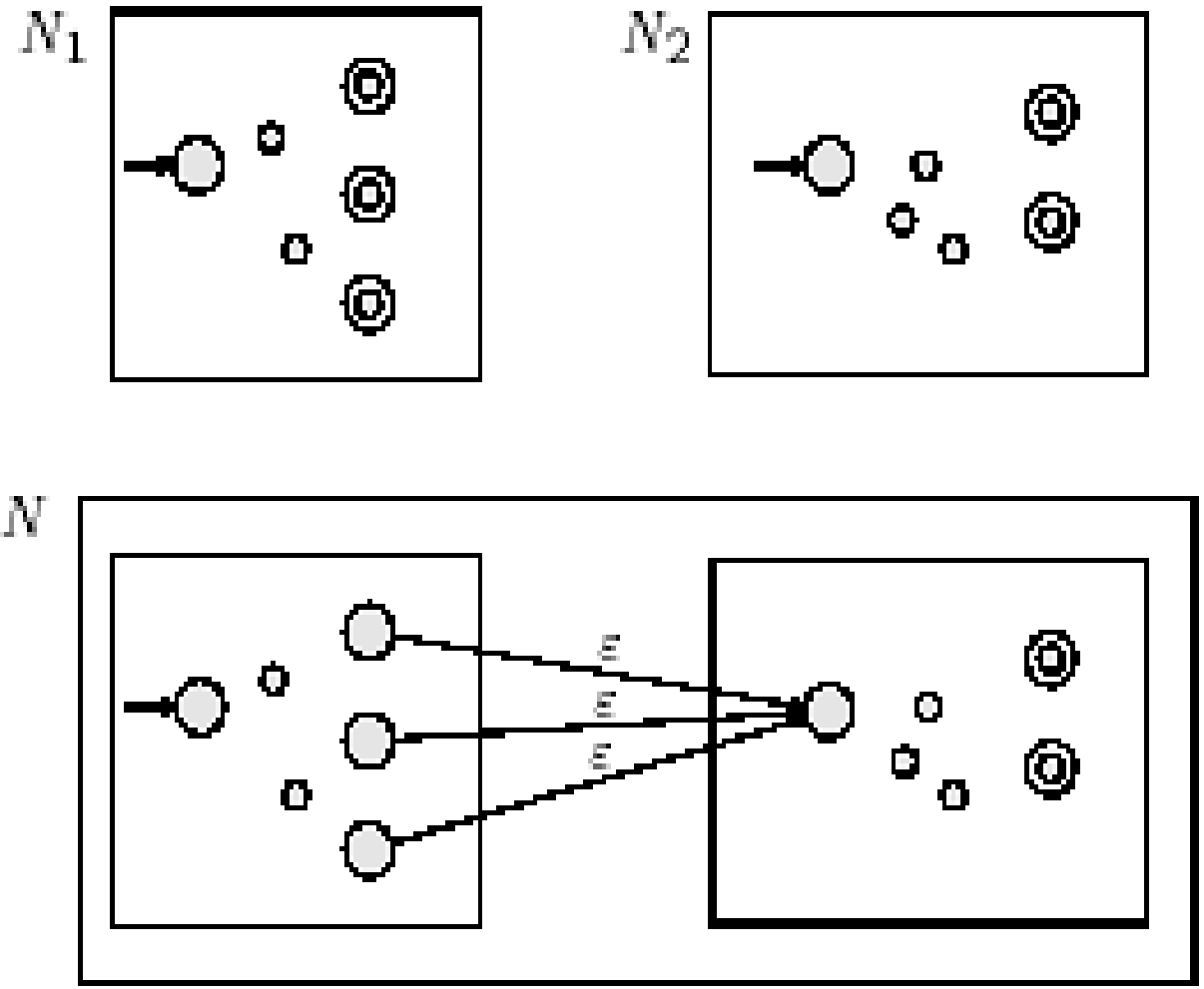
Dim Idea: NFA N per L1 L2 :



w=uv

u in L1
AND
v in L2

Dim Idea: NFA N per L_1 L_2 :



w non in L_1L_2

Comunque scriviamo w=uv:

u non in L_1
OR
v non in L_2

L_1 linguaggio riconosciuto da NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$.
L_2 linguaggio riconosciuto da NFA $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

NFA $N = (Q, \Sigma, \delta, q_1, F_2)$ per L_1L_2 :
$Q = Q_1 \cup Q_2$
Stato start q_1, stesso di N_1.
Stati finali F_2, stessi di N_2.
Funzione di transizione:

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

Si può estendere alla Kleene star:

$$L^* = \{ x_1 x_2 \cdots x_k \mid k \geq 0,\ x_i \text{ in } L\}.$$

Teorema

La classe dei linguaggi regolari è chiusa per l'operazione Kleene-star.

DIM IDEA. Se N_1 riconosce L. Costruiamo N da N_1 in cui ogni stato finale è collegato da ε-transizione allo stato iniziale