

Calcolabilità e complessità

Modelli di computazione

Informazioni generali sul corso

Docente: Giovanni Pani

Laboratorio: Graziella De Martino

giovanni.pani@uniba.it

Ricevimento: Giovedì 11.30
o per appuntamento

Libro 1 : Introduzione alla
teoria della computazione

Michael Sipser

Libro 2 : Linguaggi modelli Ausiello, d'Amore,
complessità Gambosi,

Lunedì pomeriggio
Laboratorio, portare
personal.

Esonero: 8 Aprile , su tutta la parte
fatta.

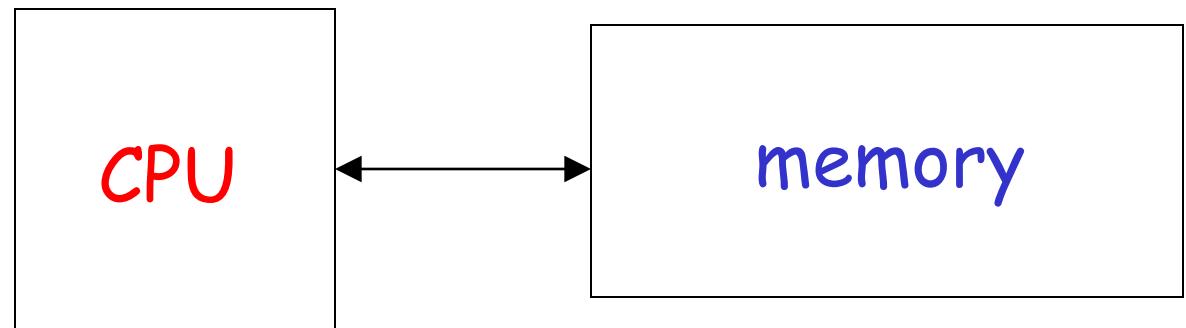
Ada di.uniba.it

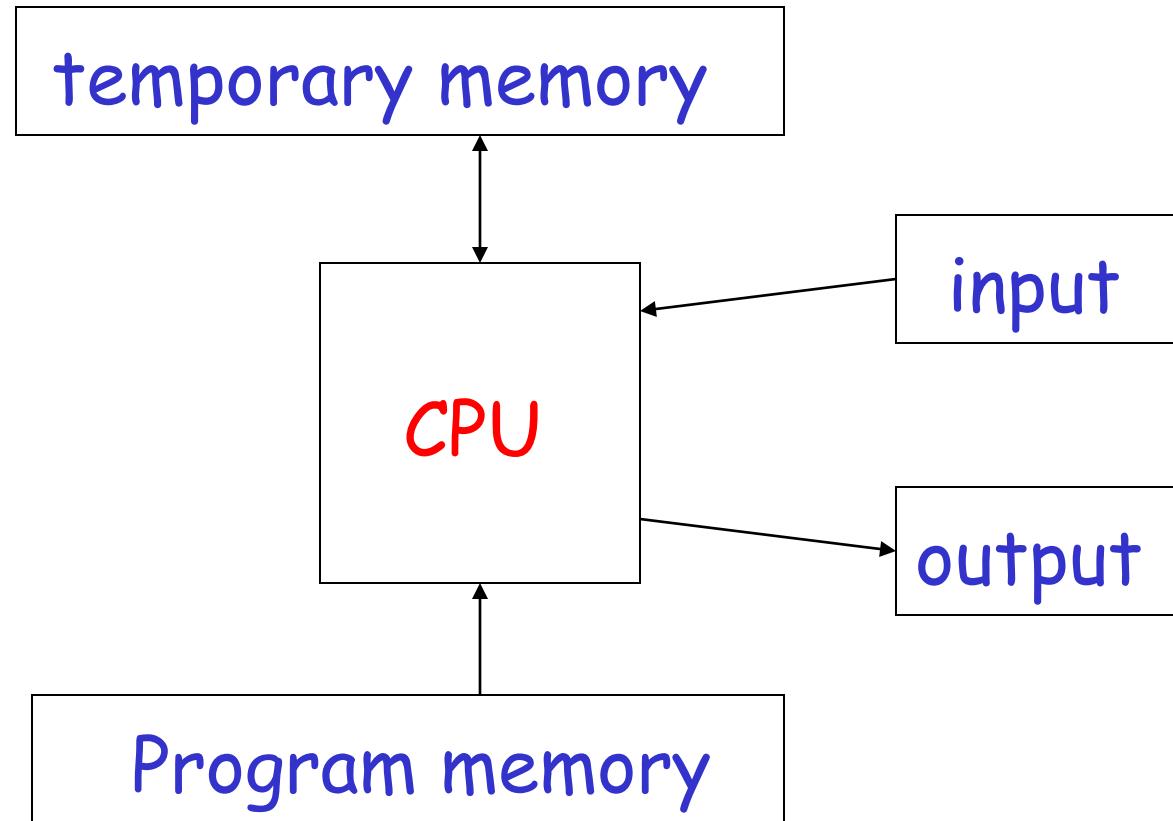
<http://informatica2.di.uniba.it/>

Psw CC-INF1920

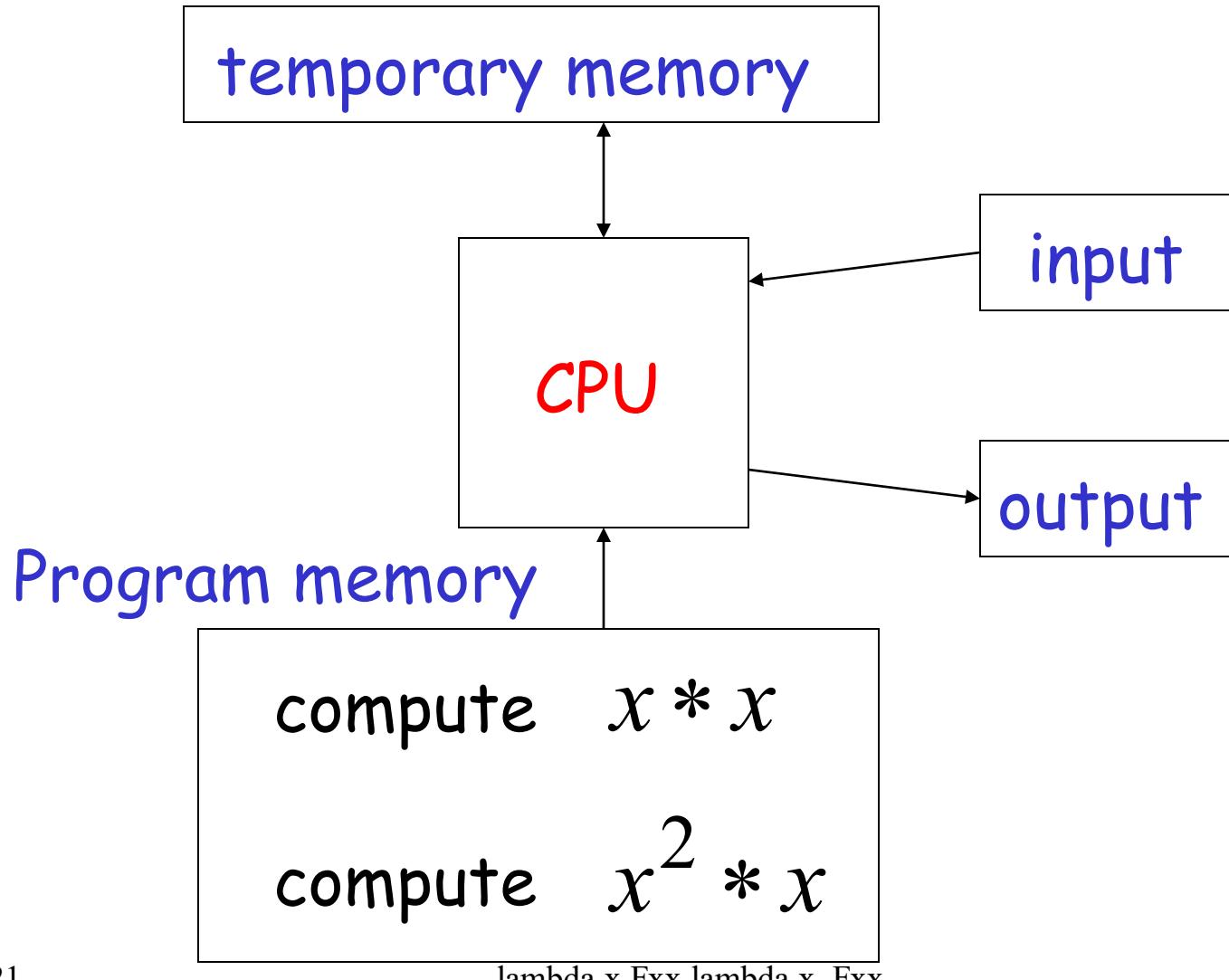
Contenuto del corso

Calcolo

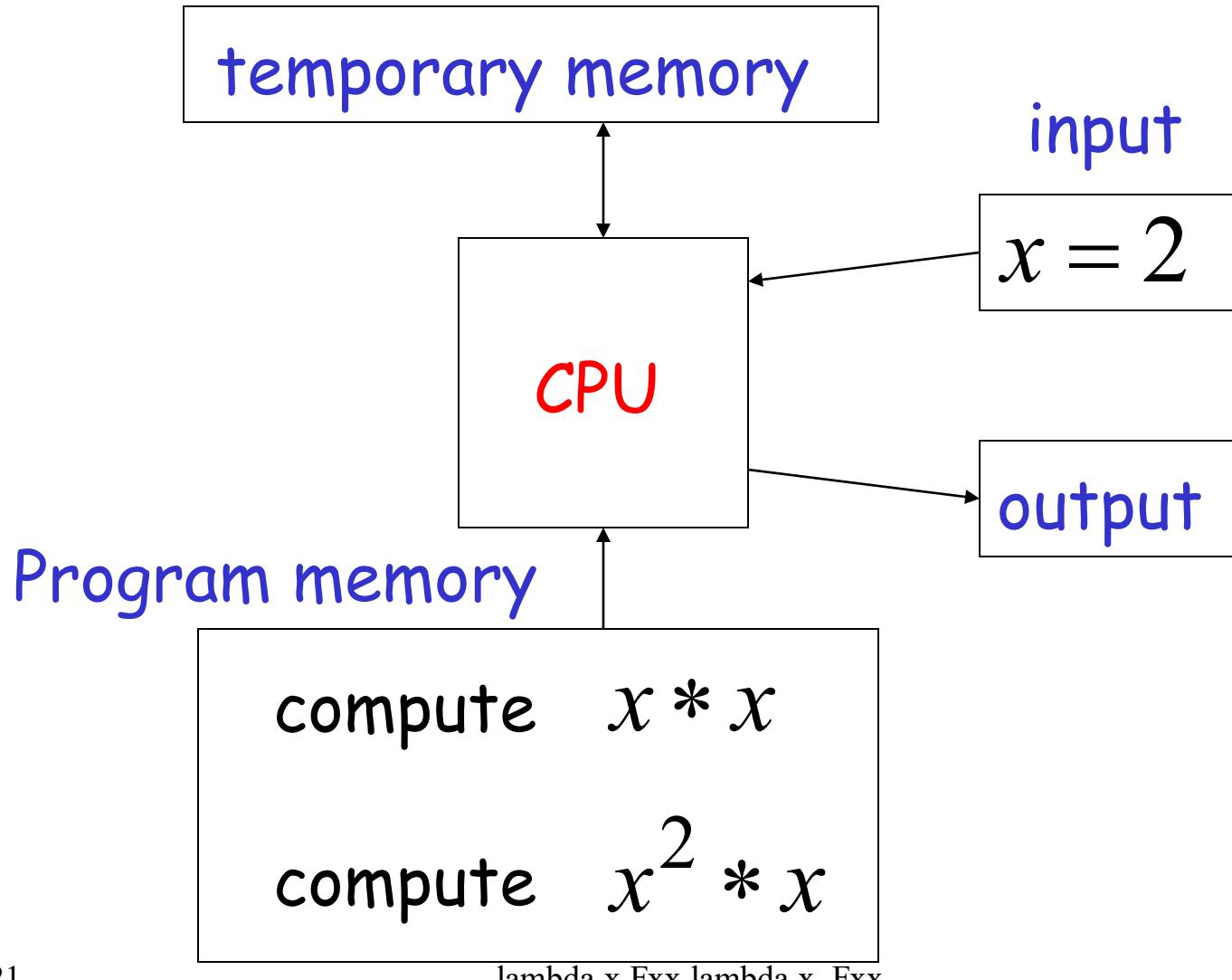




Esempio: $f(x) = x^3$



$$f(x) = x^3$$



temporary memory

$$f(x) = x^3$$

$$z = 2 * 2 = 4$$

$$f(x) = z * 2 = 8$$

input

$$x = 2$$

output

CPU

Program memory

$$\text{compute } x * x$$

$$\text{compute } x^2 * x$$

temporary memory

$$f(x) = x^3$$

$$z = 2 * 2 = 4$$

$$f(x) = z * 2 = 8$$

Program memory

compute $x * x$

compute $x^2 * x$

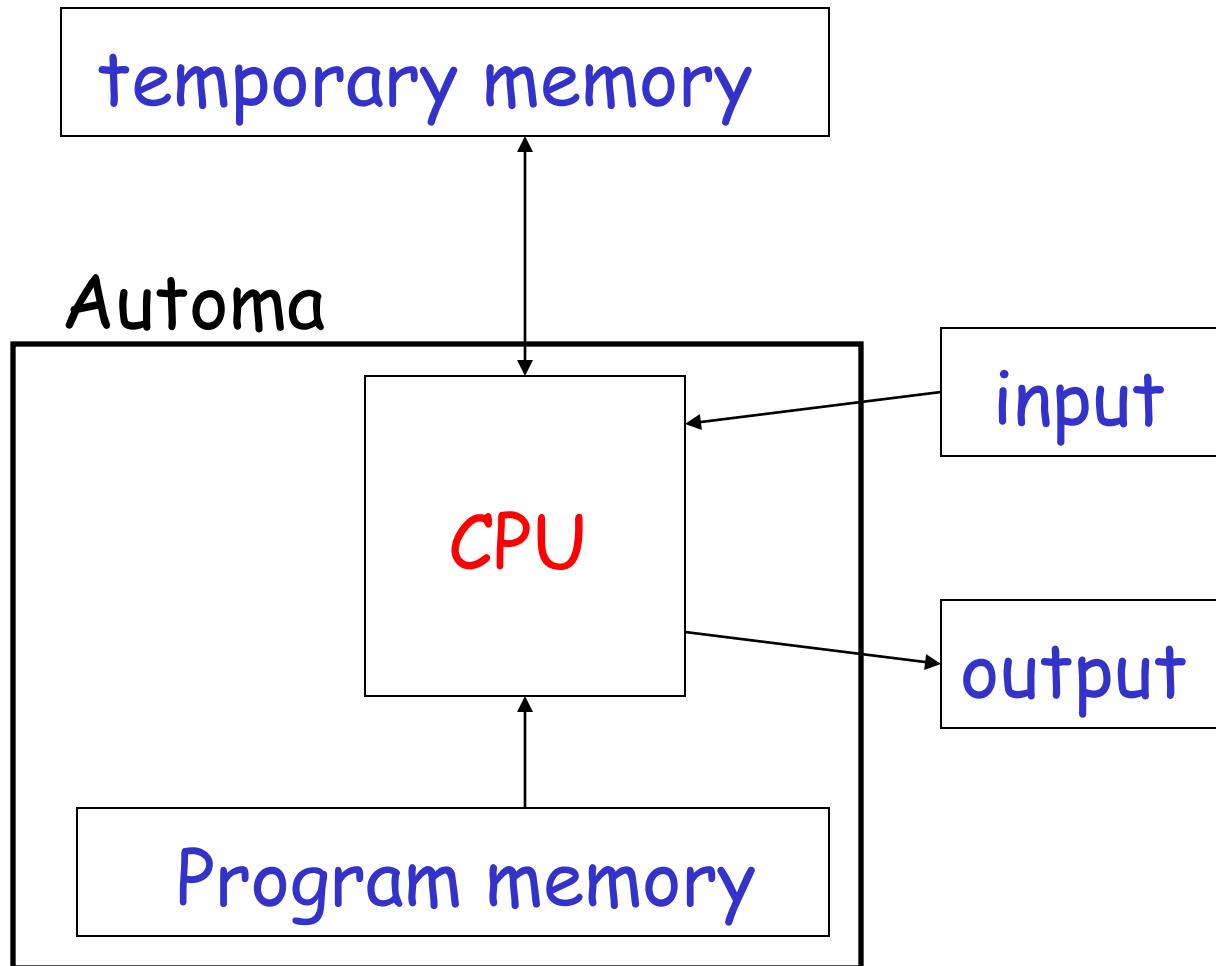
input

$$x = 2$$

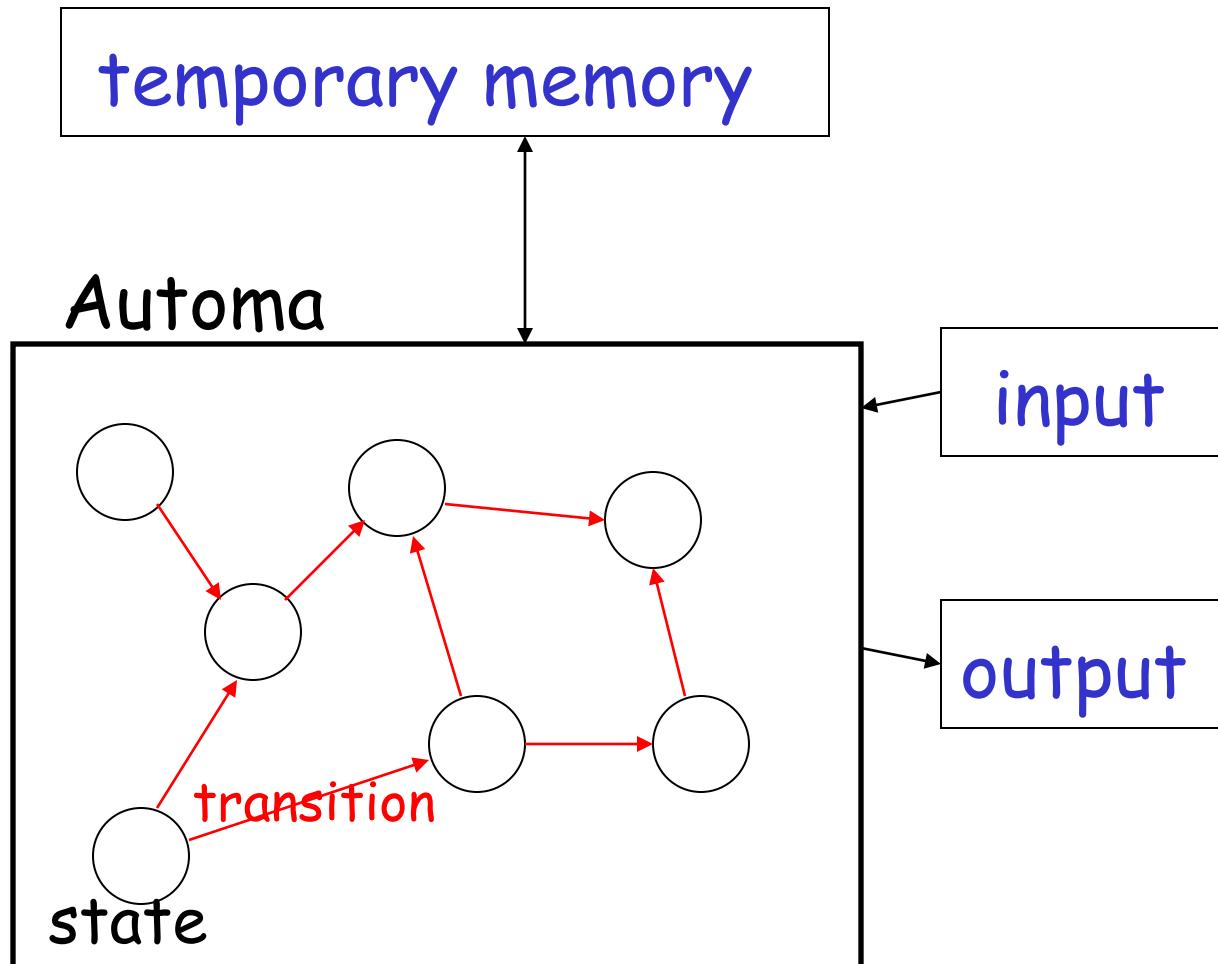
$$f(x) = 8$$

output

Automa



Automa

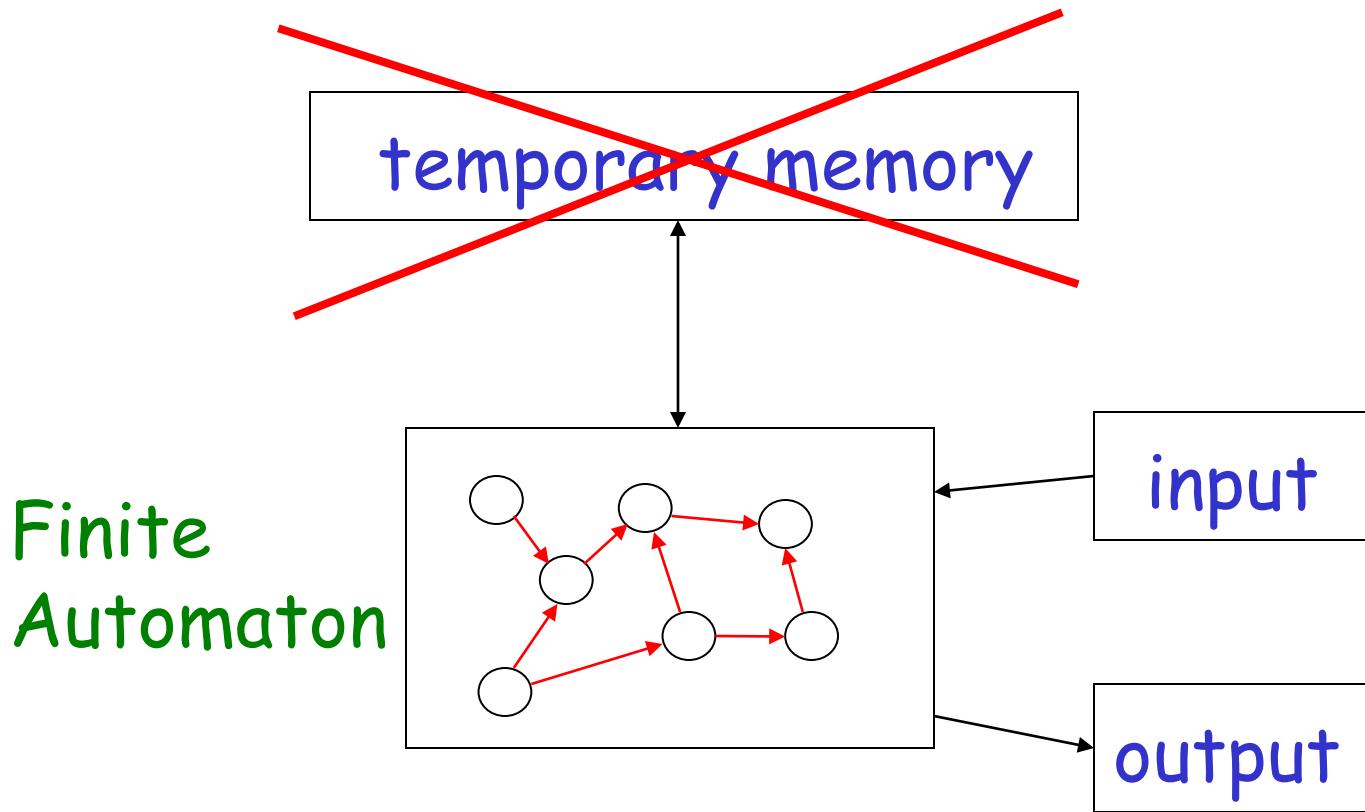


Automata

Automata si distinguono secondo il tipo di memoria

- **Finite Automata:** nessuna memoria
- **Pushdown Automata:** stack
- **Turing Machines:** random access memory

Finite Automata

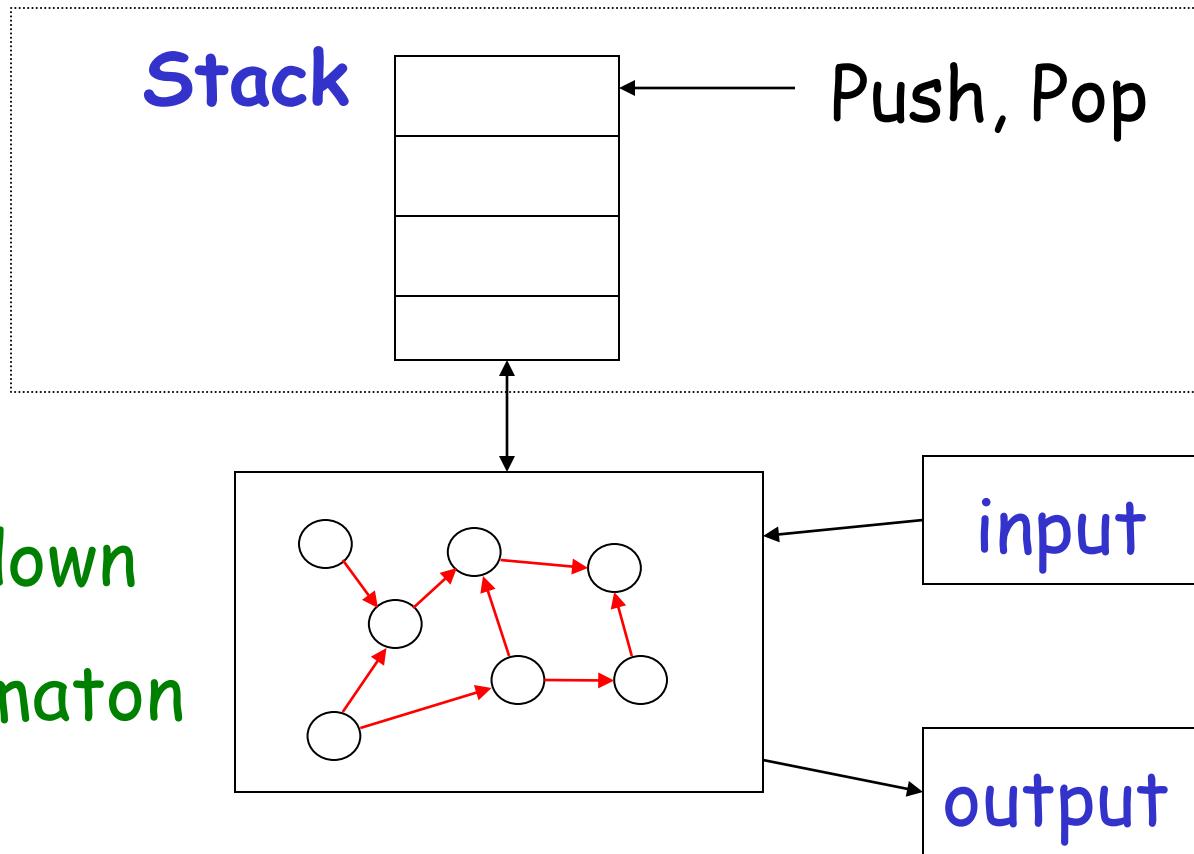


Esempio: ascensori, macchine per il caffè
(piccolo potere di computazione)

Pushdown Automata

Temp.
memory

Pushdown
Automaton



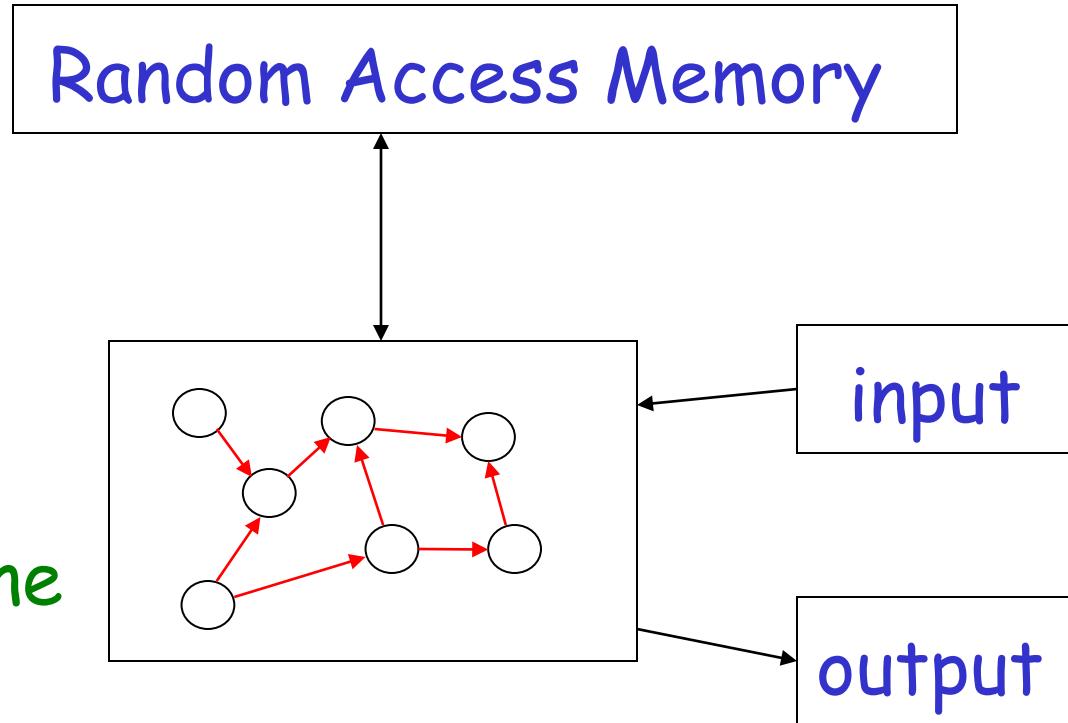
Esempio:

Compilatori per linguaggi di programmazione
(medio potere di calcolo)

Turing Machine

Temp.
memory

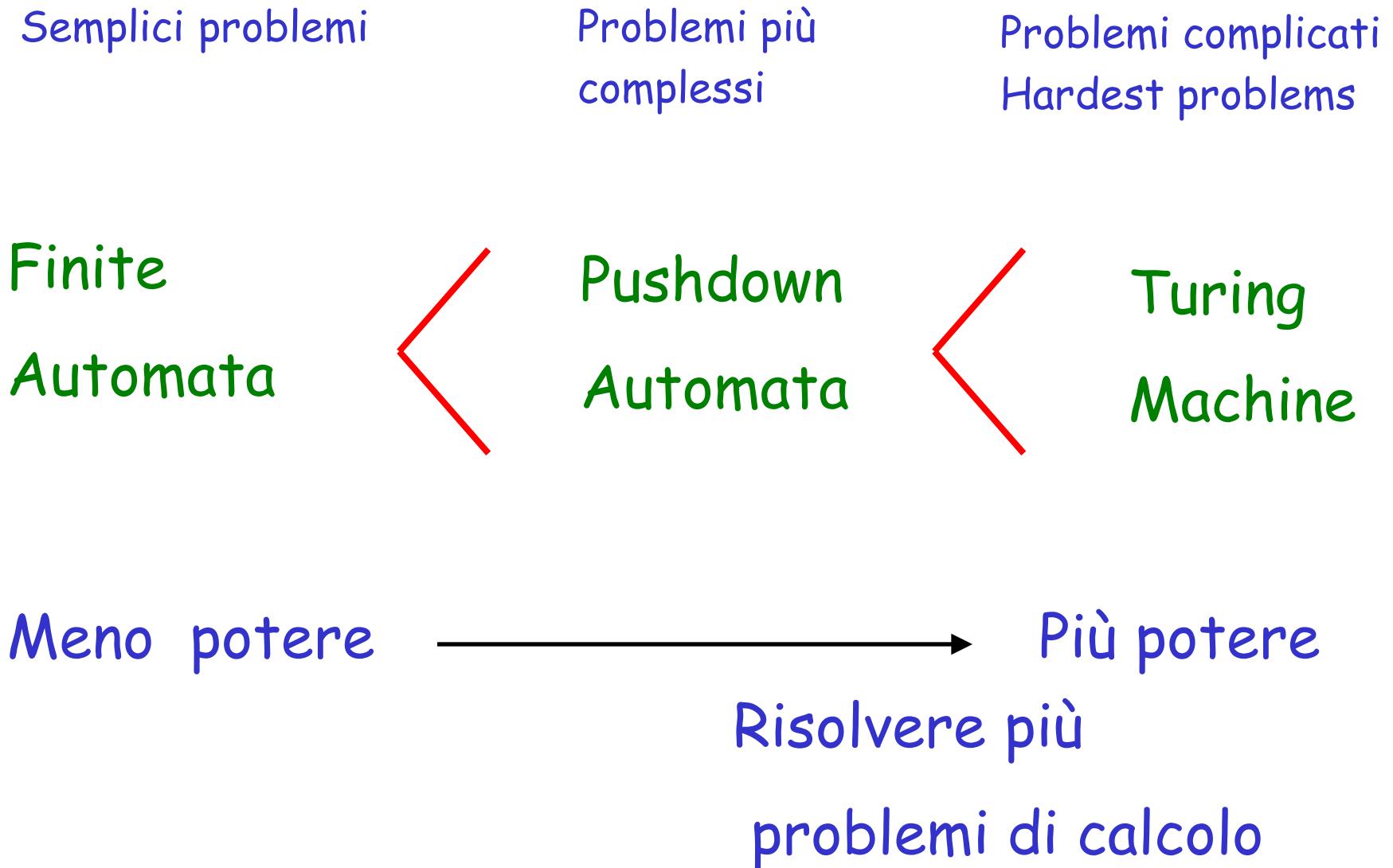
Turing
Machine



Esempio: qualsiasi algoritmo

(il più alto potere di calcolo)

Power of Automata



Turing Machine è il modello di calcolo
più potente che è stato definito

Domanda: Esistono problemi di calcolo che non possono essere risolti?

Risposta: Si (problemi irrisolvibili)

Complessità temporale dei problemi di calcolo:

NP-complete problems

Si crede che occorre un tempo esponenziale per calcolarli

P problems

Risolti in tempo polinomiale

Preliminari matematici

Preliminari matematici

- Insiemi
- Funzioni
- Relazioni
- Grafi
- Tecniche di dimostrazioni

SETS

A insieme è una collezione di elementi

$$A = \{1, 2, 3\}$$

$$B = \{train, bus, bicycle, airplane\}$$

Scriveremo:

$$1 \in A$$

$$ship \notin B$$

Rappresentazione degli insiemi

$$C = \{ a, b, c, d, e, f, g, h, i, j, k \}$$

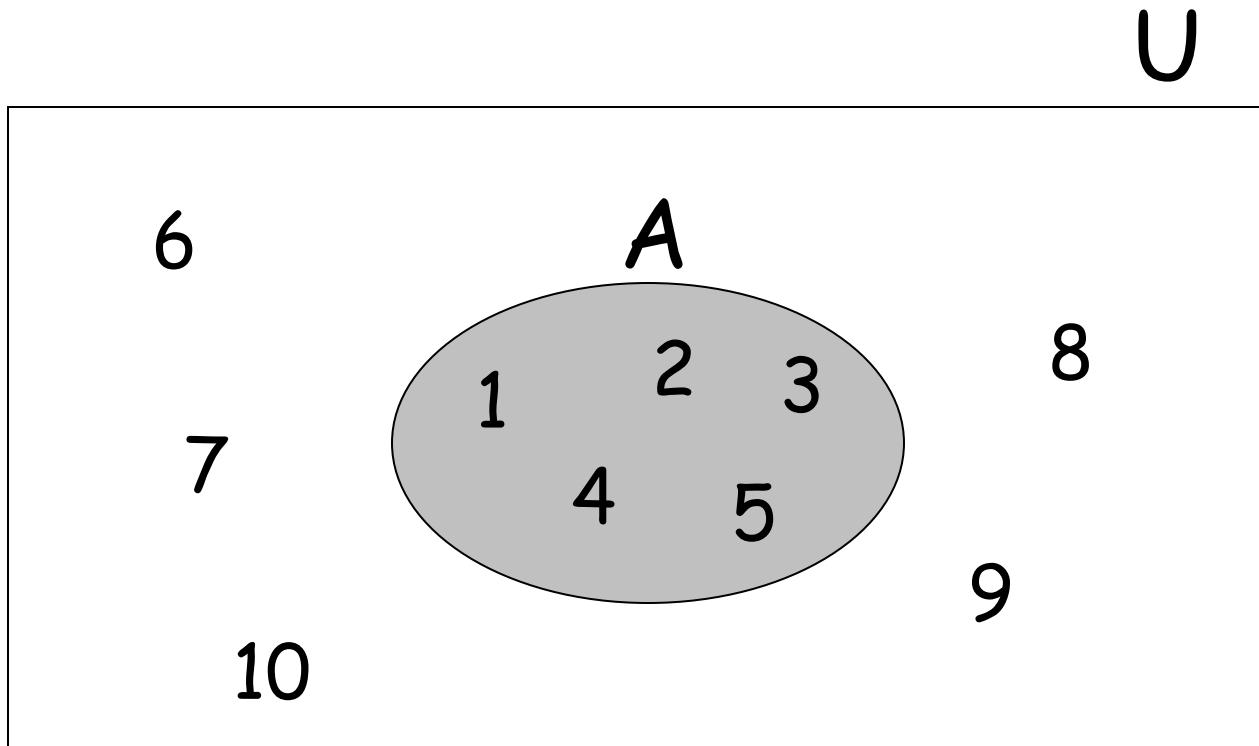
$$C = \{ a, b, \dots, k \} \longrightarrow \text{Insieme finito}$$

$$S = \{ 2, 4, 6, \dots \} \longrightarrow \text{Insieme infinito}$$

$$S = \{ j : j > 0, \text{ e } j = 2k \text{ per qualche } k > 0 \}$$

$$S = \{ j : j \text{ è non negativo e pari} \}$$

$$A = \{ 1, 2, 3, 4, 5 \}$$



Insieme universale: tutti gli elementi possibili

$$U = \{ 1, \dots, 10 \}$$

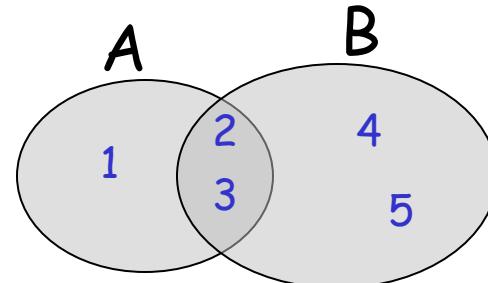
Operazione sugli insiemi

$$A = \{ 1, 2, 3 \}$$

$$B = \{ 2, 3, 4, 5 \}$$

- Unione

$$A \cup B = \{ 1, 2, 3, 4, 5 \}$$



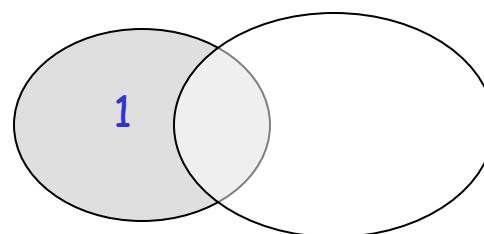
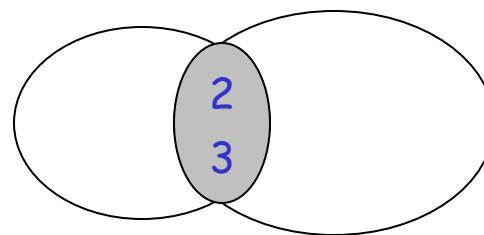
- Intersezione

$$A \cap B = \{ 2, 3 \}$$

- Differenza

$$A - B = \{ 1 \}$$

$$B - A = \{ 4, 5 \}$$

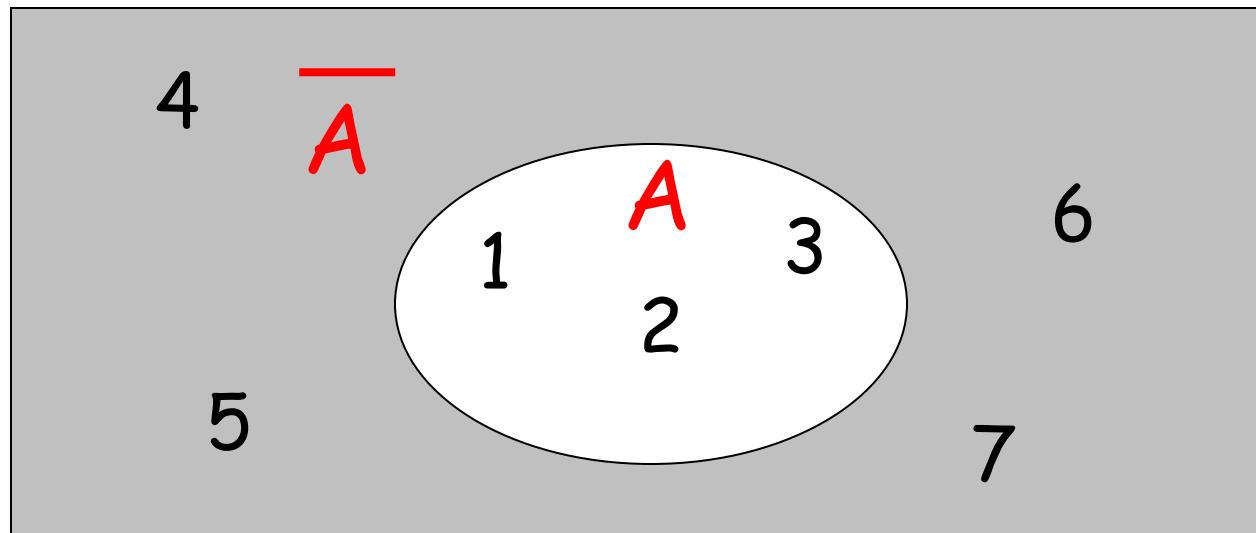


Venn diagrams

• Complemento

Insieme universale = $\{1, \dots, 7\}$

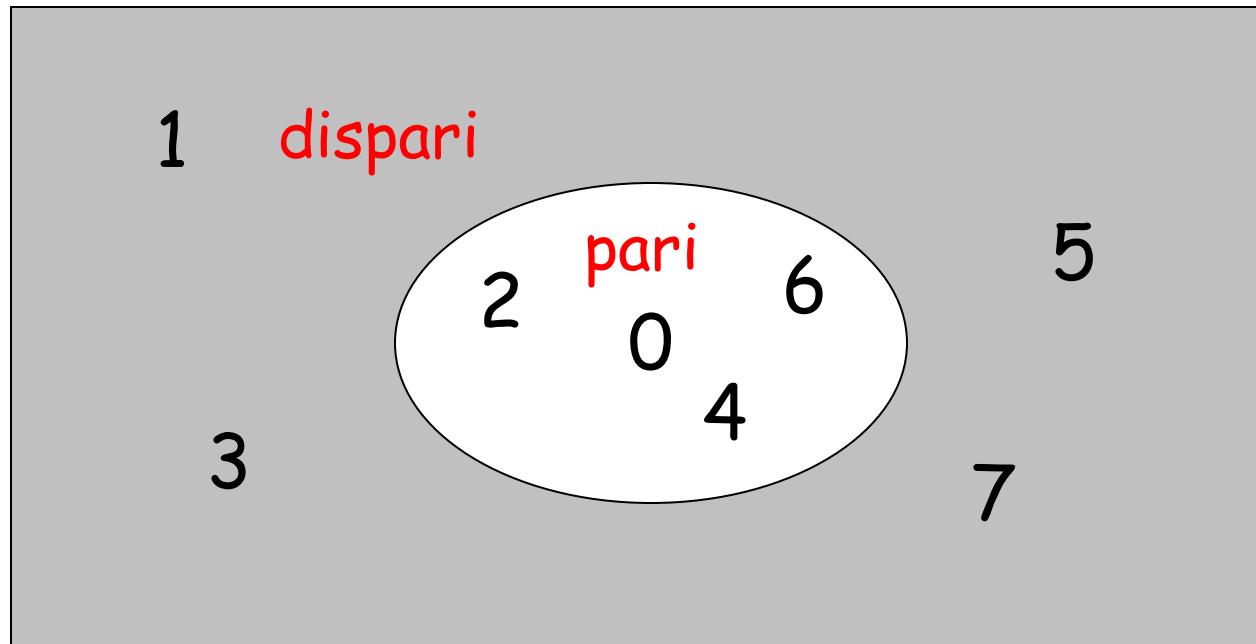
$$A = \{1, 2, 3\} \rightarrow \overline{A} = \{4, 5, 6, 7\}$$



$$\overline{\overline{A}} = A$$

$$\{ \text{ interi pari} \} = \{ \text{ interi dispari} \}$$

interi



Leggi di DeMorgan

$$\overline{A \cup B} = \overline{A} \cap \overline{B}$$

$$\overline{A \cap B} = \overline{A} \cup \overline{B}$$

Vuoto, insieme nullo: \emptyset

$$\emptyset = \{ \}$$

$$S \cup \emptyset = S$$

$$S \cap \emptyset = \emptyset$$

$\overline{\emptyset}$ = Universal Set

$$S - \emptyset = S$$

$$\emptyset - S = \emptyset$$

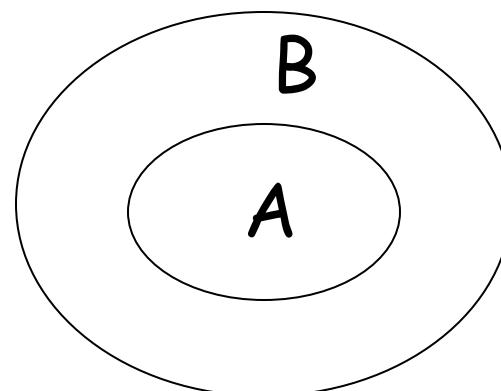
Sottoinsieme

$$A = \{ 1, 2, 3 \}$$

$$B = \{ 1, 2, 3, 4, 5 \}$$

$$A \subseteq B$$

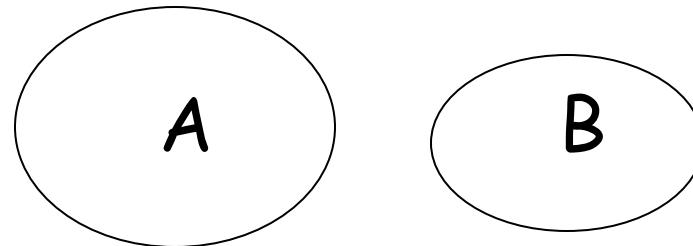
Sottoinsieme proprio: $A \subset B$



Insieme disgiunti

$$A = \{ 1, 2, 3 \} \quad B = \{ 5, 6 \}$$

$$A \cap B = \emptyset$$



Cardinalità

- per gli insiemi finiti

$$A = \{ 2, 5, 7 \}$$

$$|A| = 3$$

(dimensione dell'insieme)

Insieme potenza

Un insieme potenza è un insieme di insiemi

$$S = \{ a, b, c \}$$

Potenza di S = l'insieme di tutti I sottointersiemi di S

$$2^S = \{ \emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\} \}$$

Osservazione: $|2^S| = 2^{|S|}$ ($8 = 2^3$)

Prodotto Cartesiano

$$A = \{ 2, 4 \}$$

$$B = \{ 2, 3, 5 \}$$

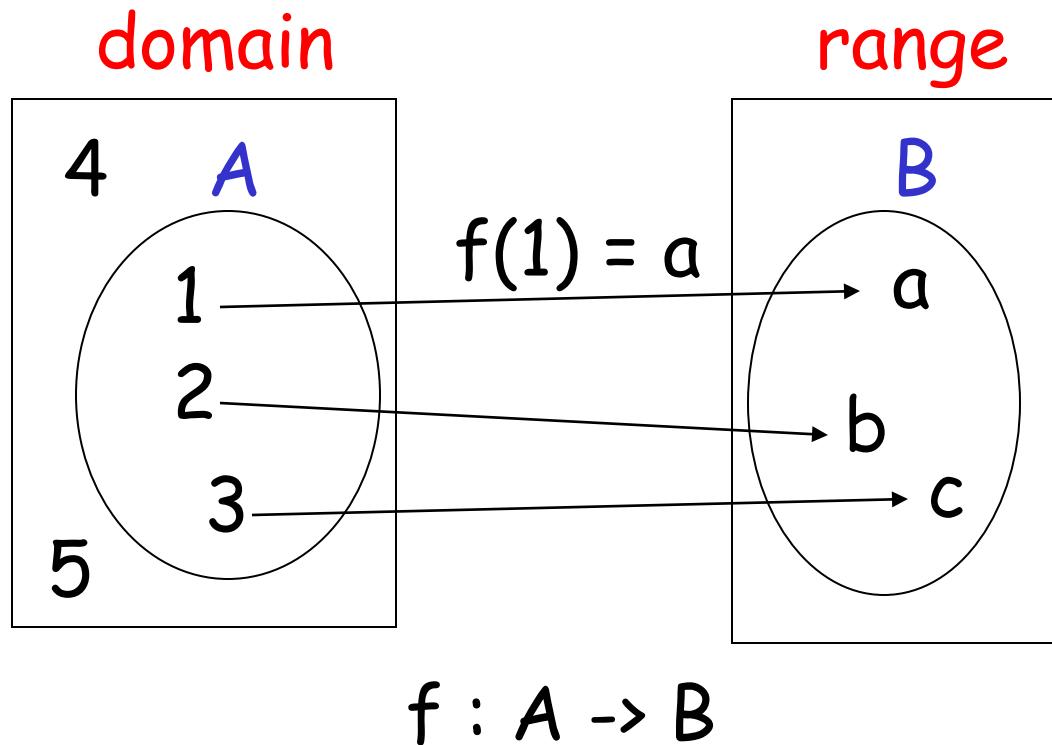
$$A \times B = \{ (2, 2), (2, 3), (2, 5), (4, 2), (4, 3), (4, 5) \}$$

$$|A \times B| = |A| |B|$$

Possiamo generalizzarlo a più insiemi

$$A \times B \times \dots \times Z$$

Funzioni



Se $A = \text{dominio}$

allora f è una funzione totale

altrimenti f è una funzione parziale

Relazioni

$$R = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots\}$$

$$x_i R y_i$$

per esempio . se $R = >$: $2 > 1, 3 > 2, 3 > 1$

Relazioni di equivalenza

- Riflessiva: $x R x$
- Simmetrica: $x R y \rightarrow y R x$
- Transitiva: $x R y \text{ and } y R z \rightarrow x R z$

Esempio: $R = '='$

- $x = x$
- $x = y \rightarrow y = x$
- $x = y \text{ e } y = z \rightarrow x = z$

Classi di equivalenza

Data la relazione di equivalenza R

la classe di equivalenza per $x = \{y : x R y\}$

Esempio:

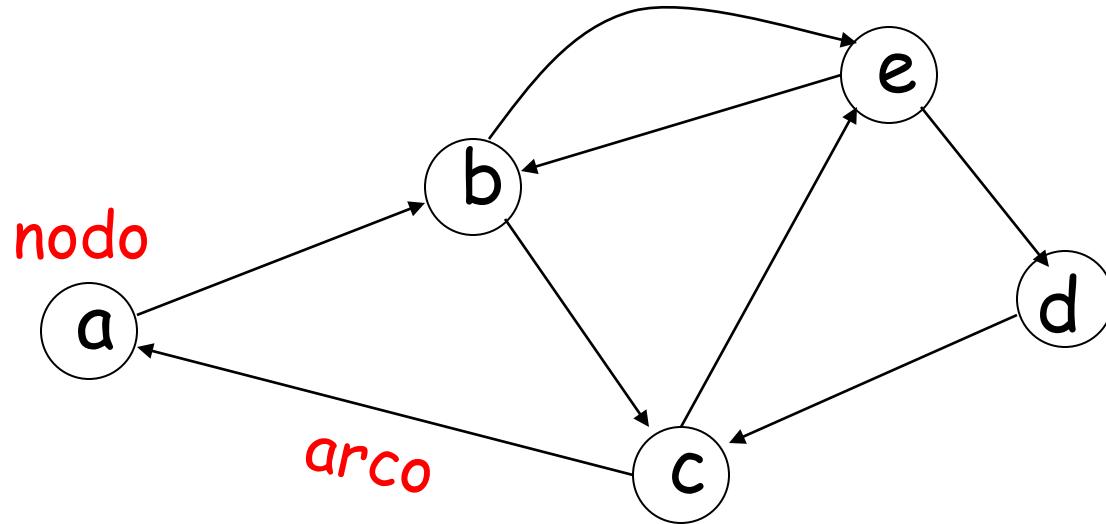
$$R = \{(1, 1), (2, 2), (1, 2), (2, 1), (3, 3), (4, 4), (3, 4), (4, 3)\}$$

classe di equivalenza per $1 = \{1, 2\}$

classe di equivalenza per $3 = \{3, 4\}$

Grafi

Grafo diretto



- Nodi (Vertici)

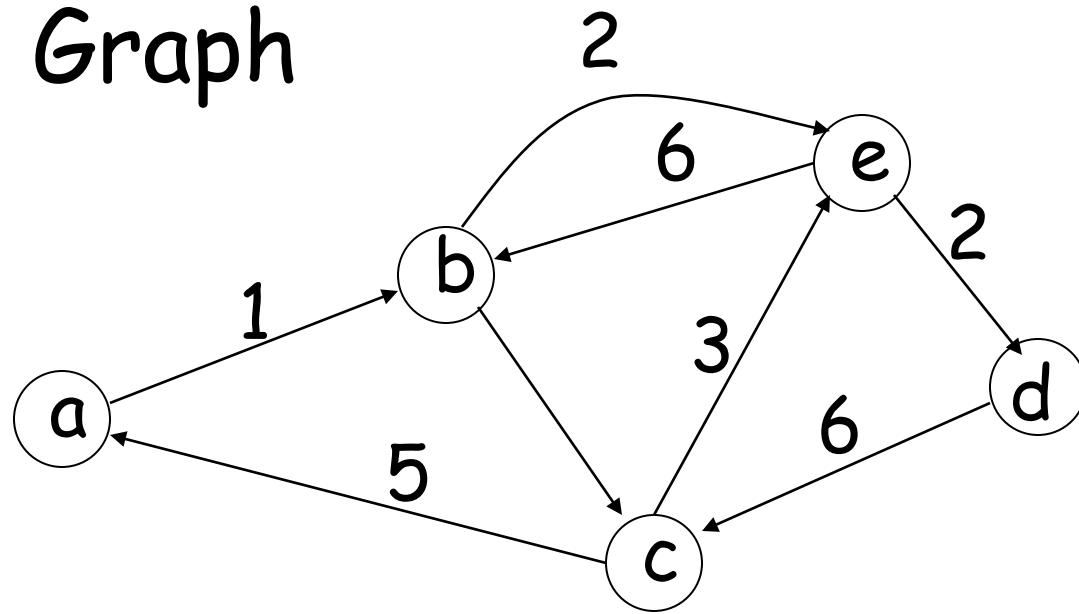
$$V = \{ a, b, c, d, e \}$$

- Archi

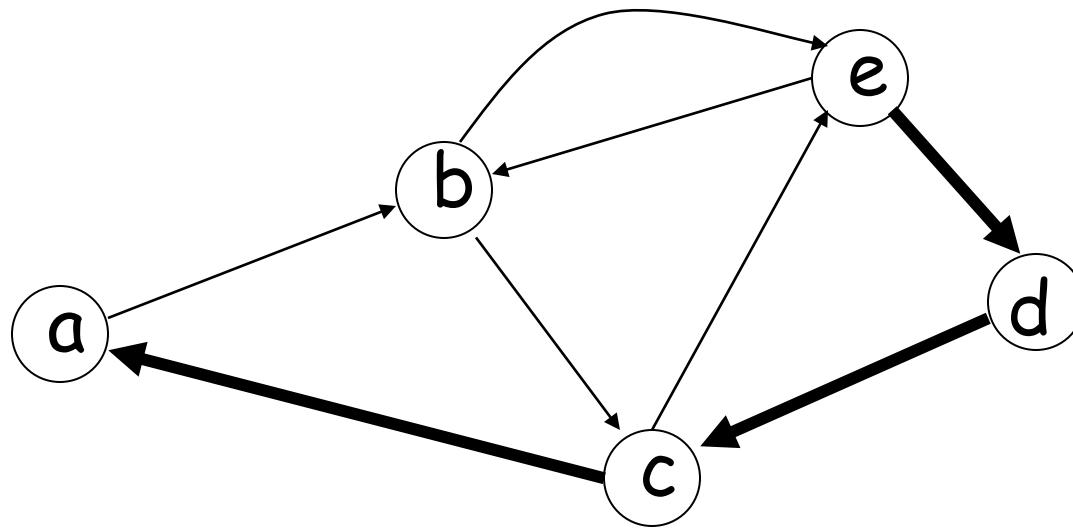
$$E = \{ (a,b), (b,c), (b,e), (c,a), (c,e), (d,c), (e,b), (e,d) \}$$

Grafo con etichette

Labeled Graph

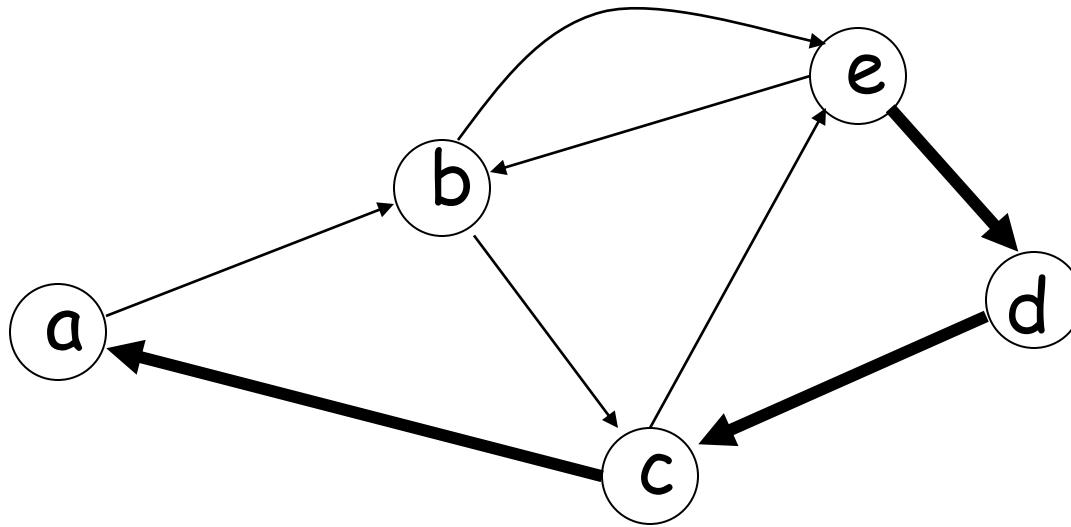


Cammino



Un cammino è una sequenza di archi adiacenti
 $(e, d), (d, c), (c, a)$

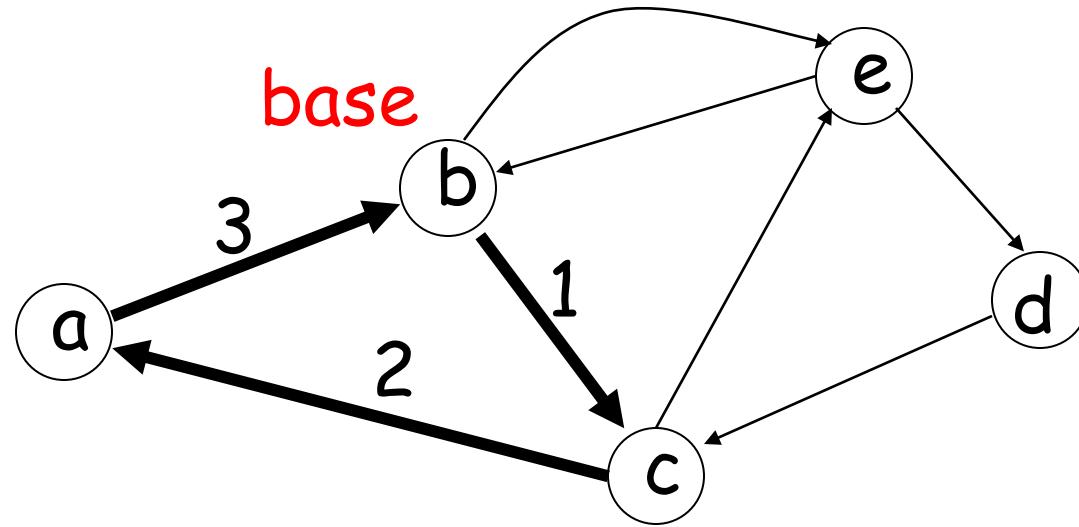
Path



Path è un cammino in cui nessun arco è ripetuto

Simple path : nessun nodo è ripetuto

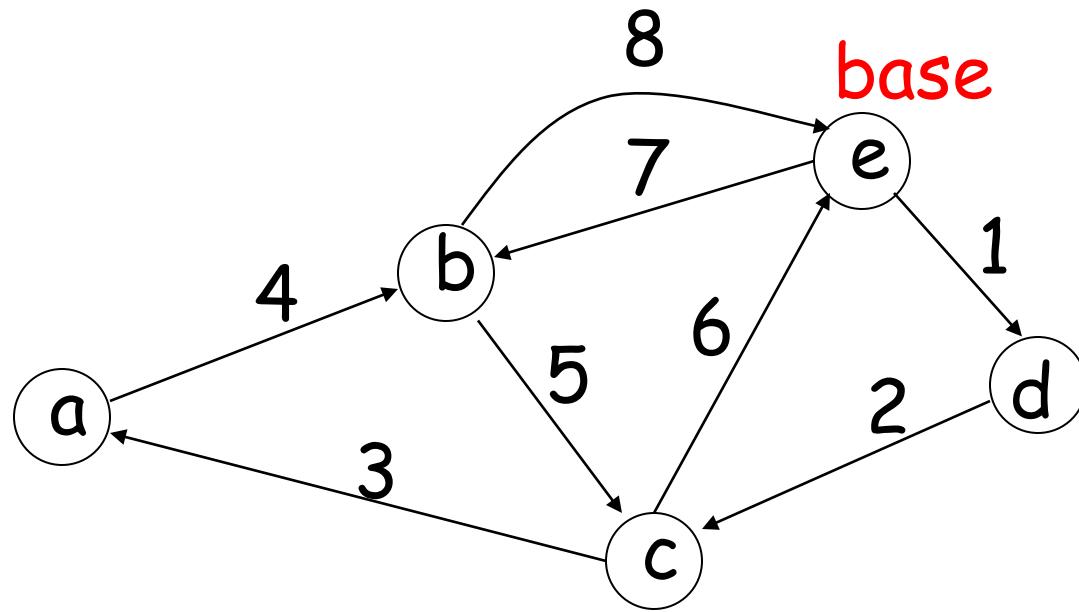
Ciclo



Ciclo: un cammino da un nodo(base) a se stesso

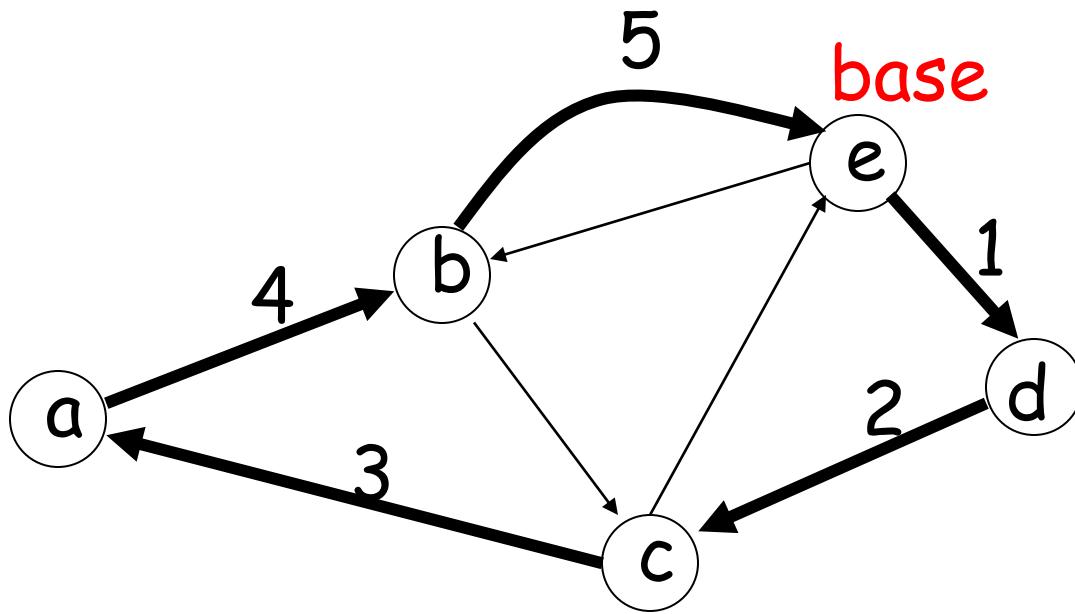
Ciclo semplice: solo la base è ripetuta

Euler Tour



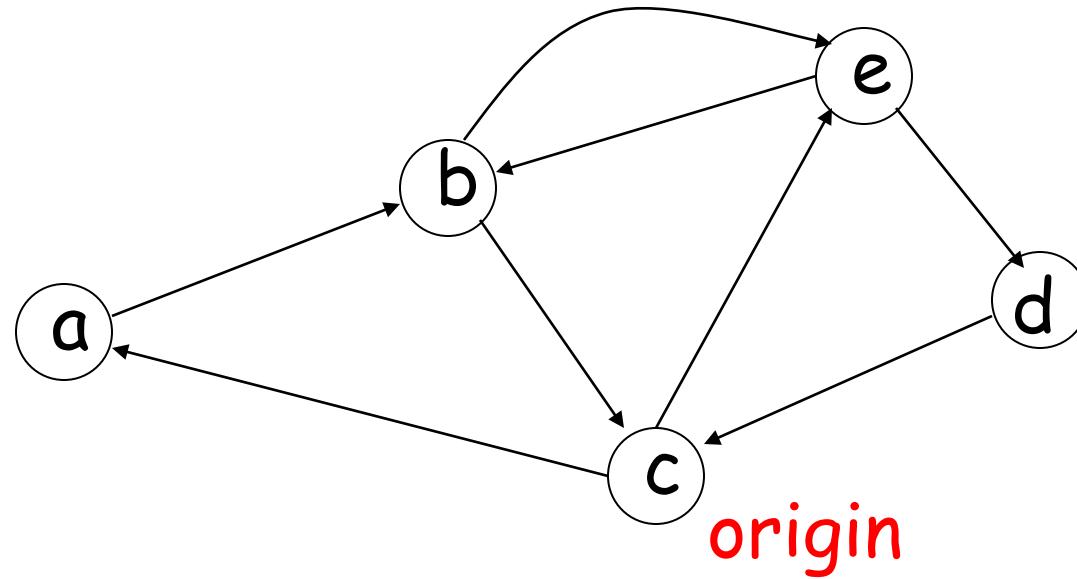
Un ciclo che contiene ogni arco una sola volta

Ciclo Hamiltonian

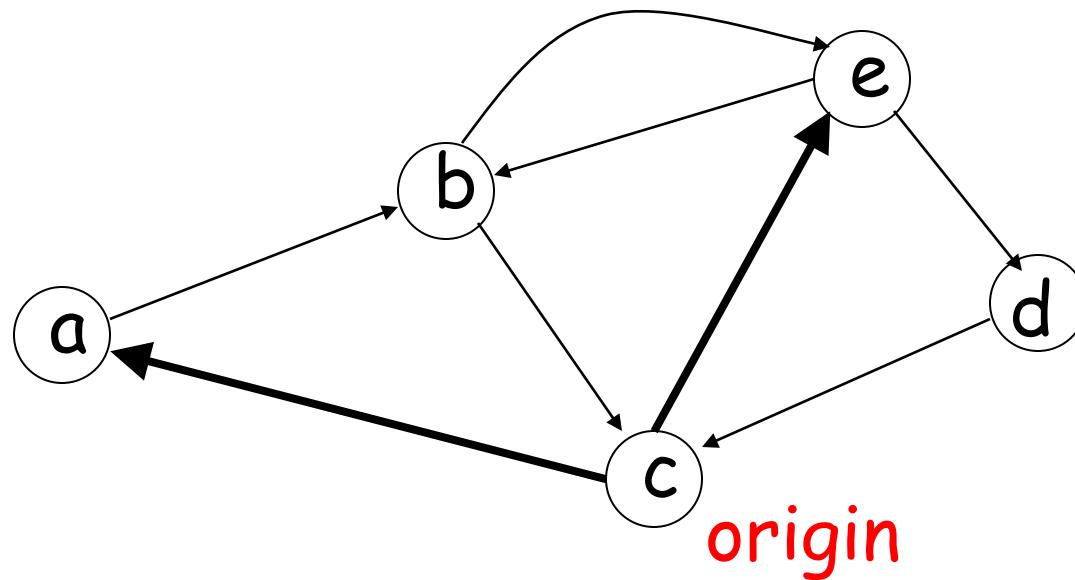


Un ciclo semplice che contiene tutti i nodi

Trovare tutti I path semplici



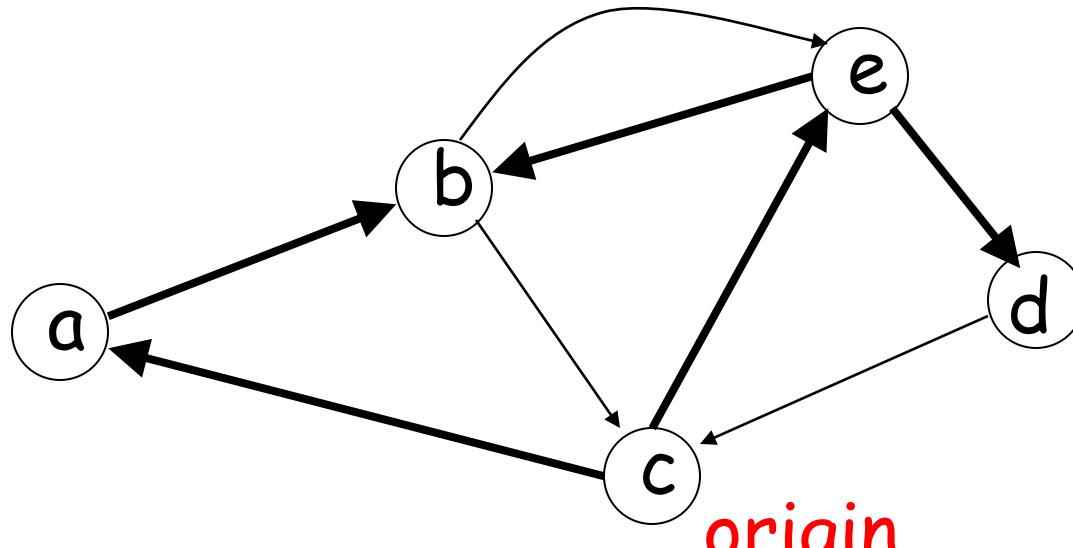
Step 1



(c, a)

(c, e)

Step 2



(c, a)

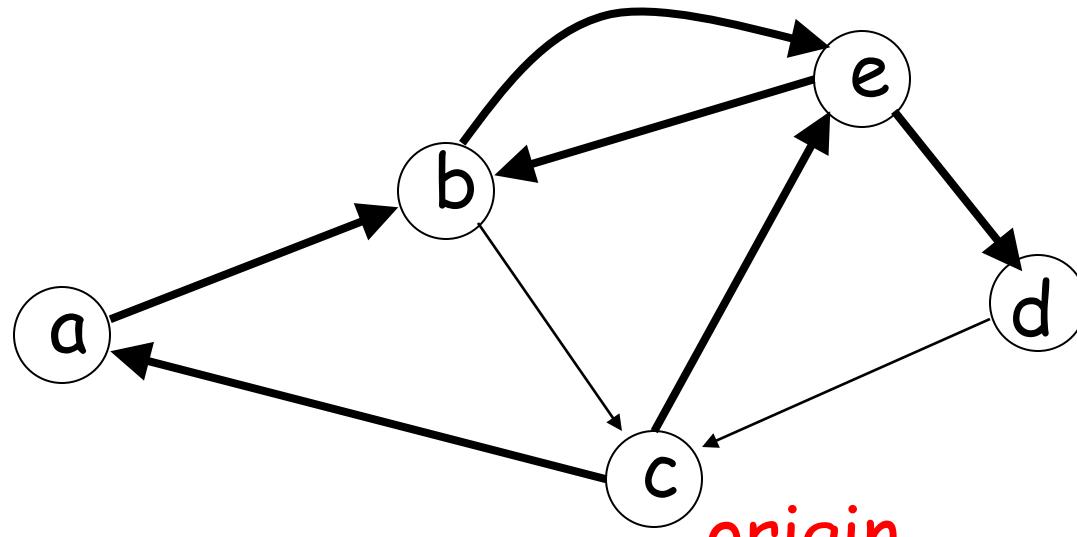
(c, a), (a, b)

(c, e)

(c, e), (e, b)

(c, e), (e, d)

Step 3



(c, a)

(c, a), (a, b)

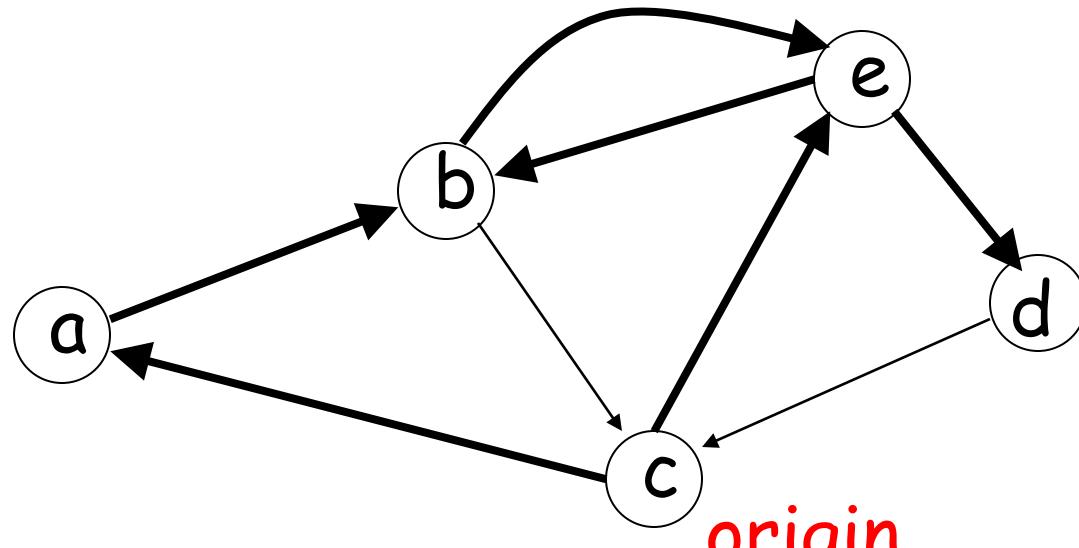
(c, a), (a, b), (b, e)

(c, e)

(c, e), (e, b)

(c, e), (e, d)

Step 4



(c, a)

(c, a), (a, b)

(c, a), (a, b), (b, e)

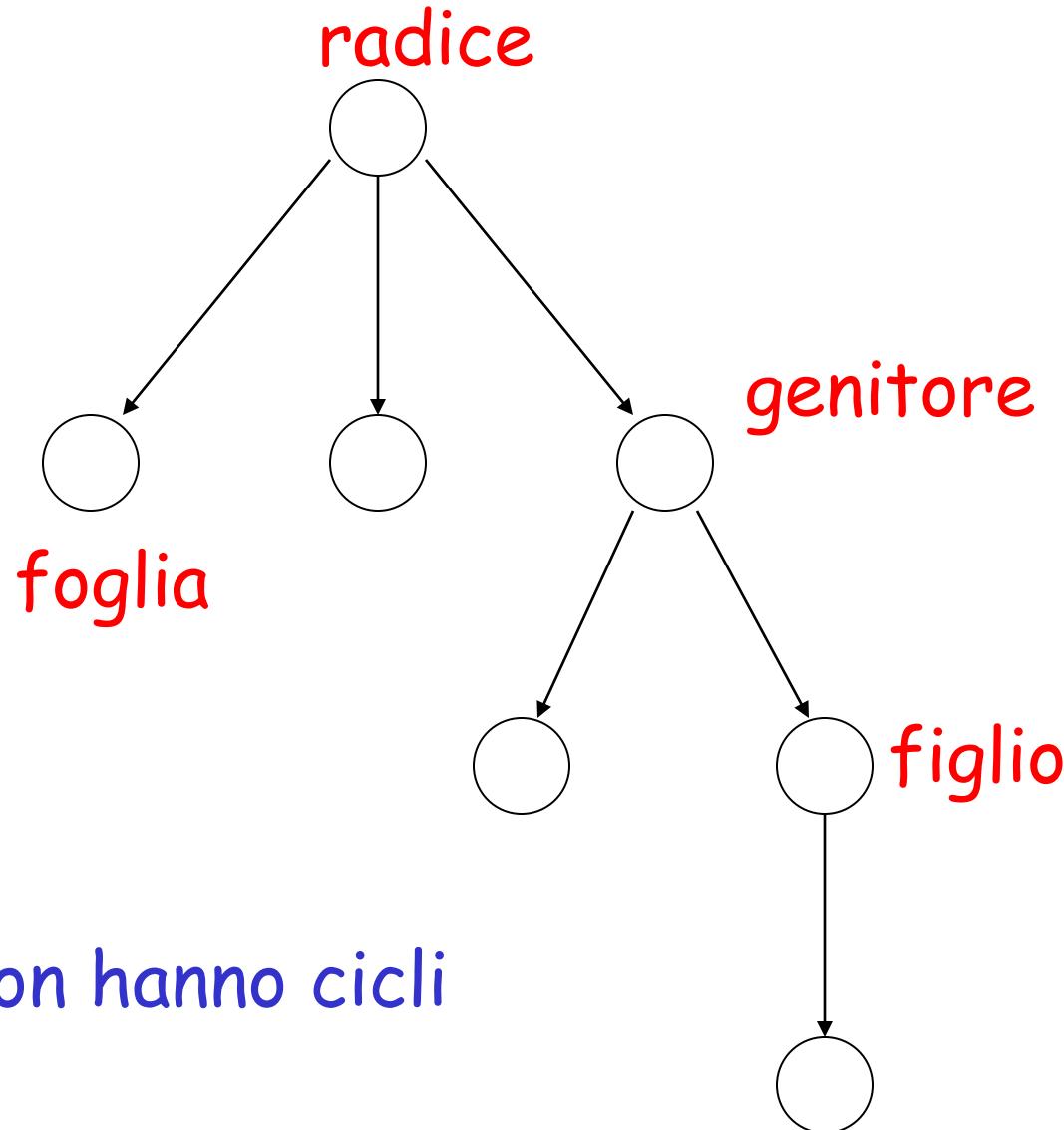
(c, a), (a, b), (b, e), (e, d)

(c, e)

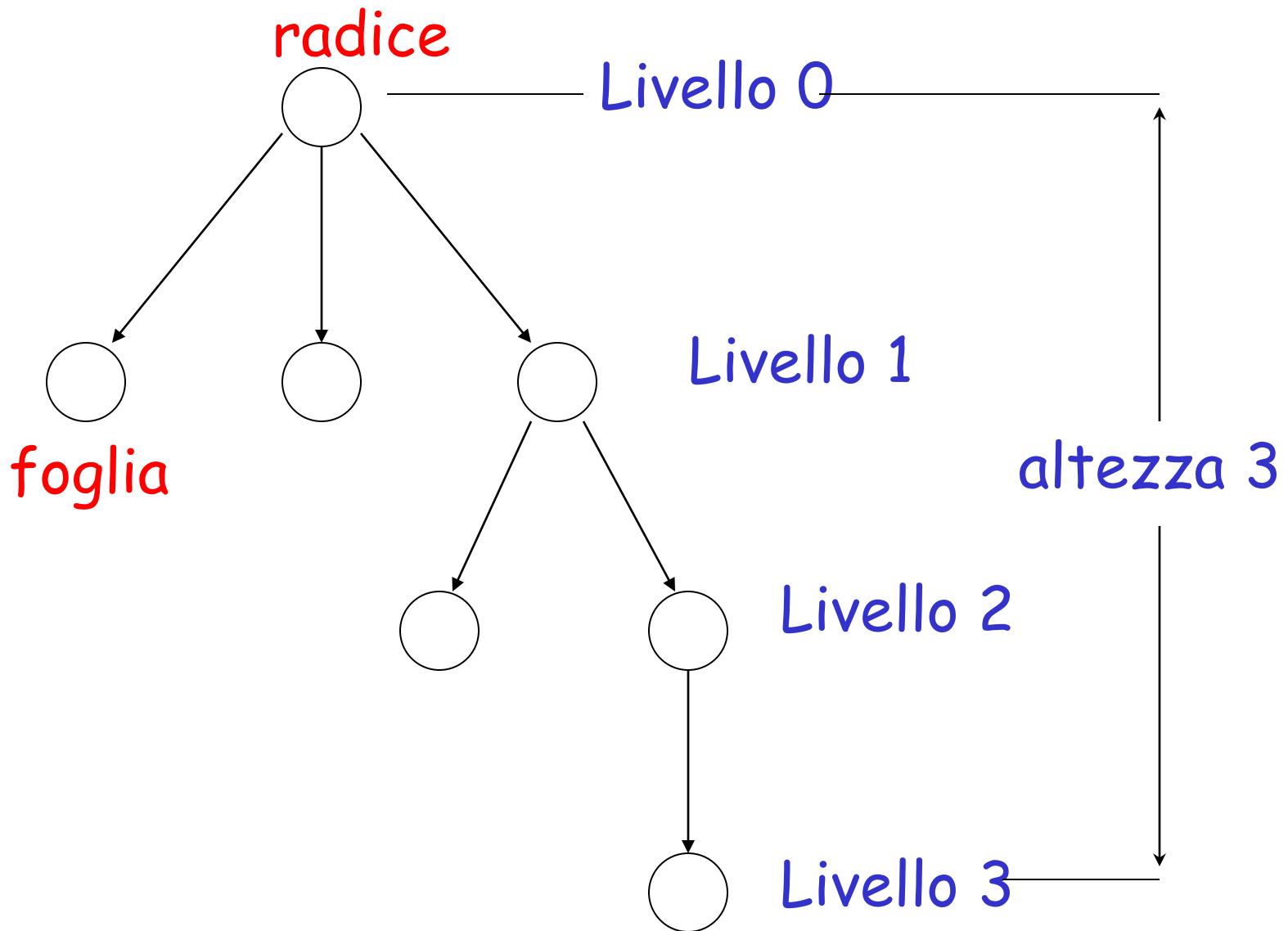
(c, e), (e, b)

(c, e), (e, d)

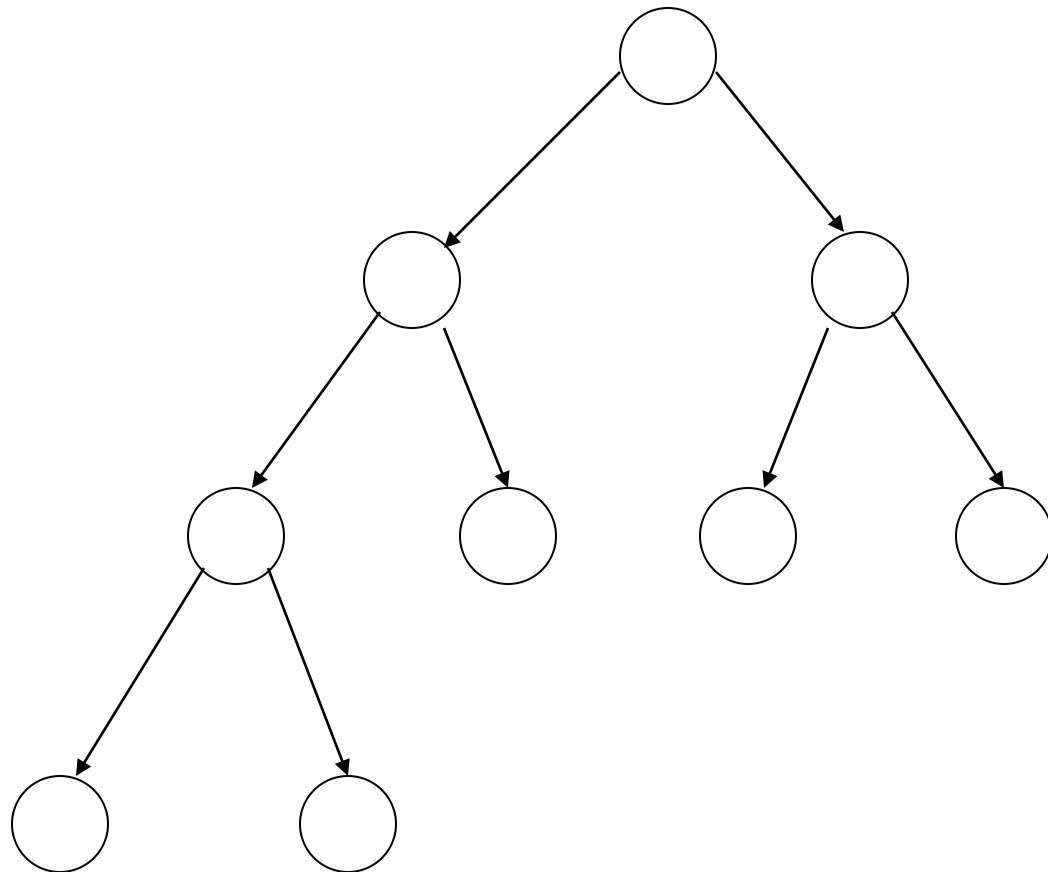
Alberi



Alberi non hanno cicli



Alberi binari



Tecniche di dimostrazione

- dimostrazione per induzione
- dimostrazione per assurdo

Induzione

Abbiamo una serie di affermazioni ordinate

$$P_1, P_2, P_3, \dots$$

Se sappiamo

- per qualche b che P_1, P_2, \dots, P_b sono vere
- per ogni $k \geq b$ che

$$P_1, P_2, \dots, P_k \text{ implica } P_{k+1}$$

Then

allora P_i è vera

Dimostrazione per induzione

- Base induttiva

trovare P_1, P_2, \dots, P_b che sono vere

- Ipotesi induttiva

Assumiamo che P_1, P_2, \dots, P_k sono vere,

Per ogni $k \geq b$

- Passo induttivo

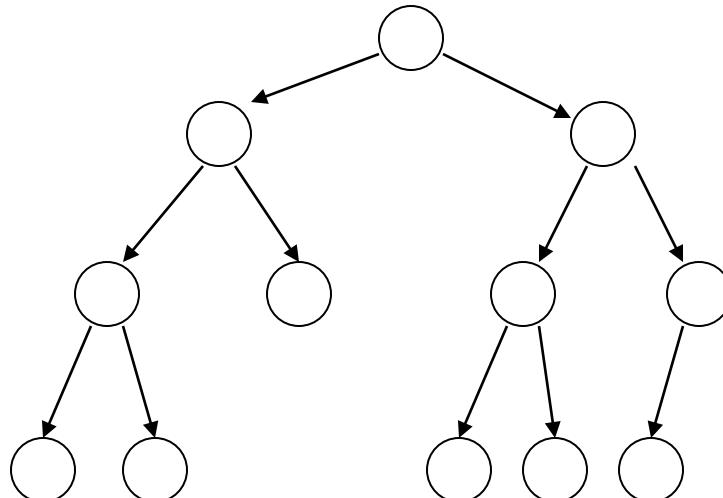
Dimostrare che P_{k+1} è vera

Esempio

Theorem: Un albero binario di altezza n ha al massimo 2^n foglie.

Proof by induction:

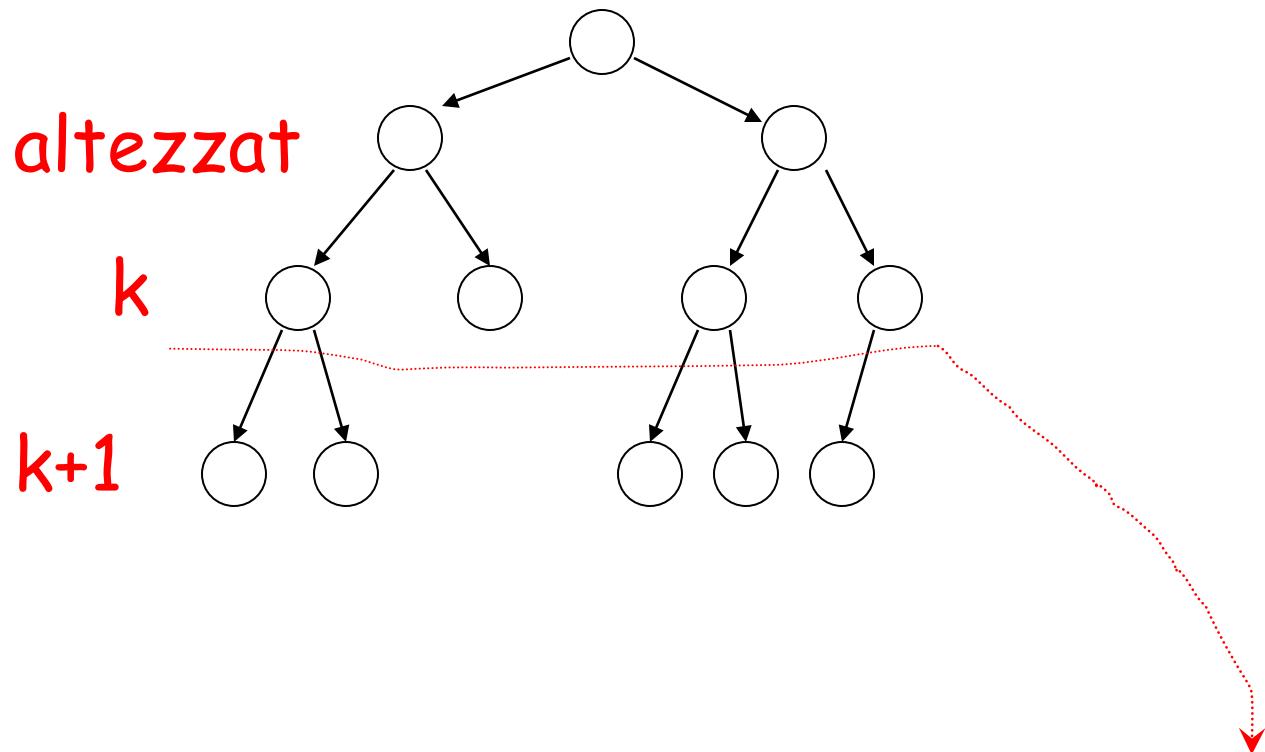
Sia $L(i)$ il massimo numero di foglie
di ogni sottoalbero di altezza i



Vogliamo dimostrare che : $L(i) \leq 2^i$

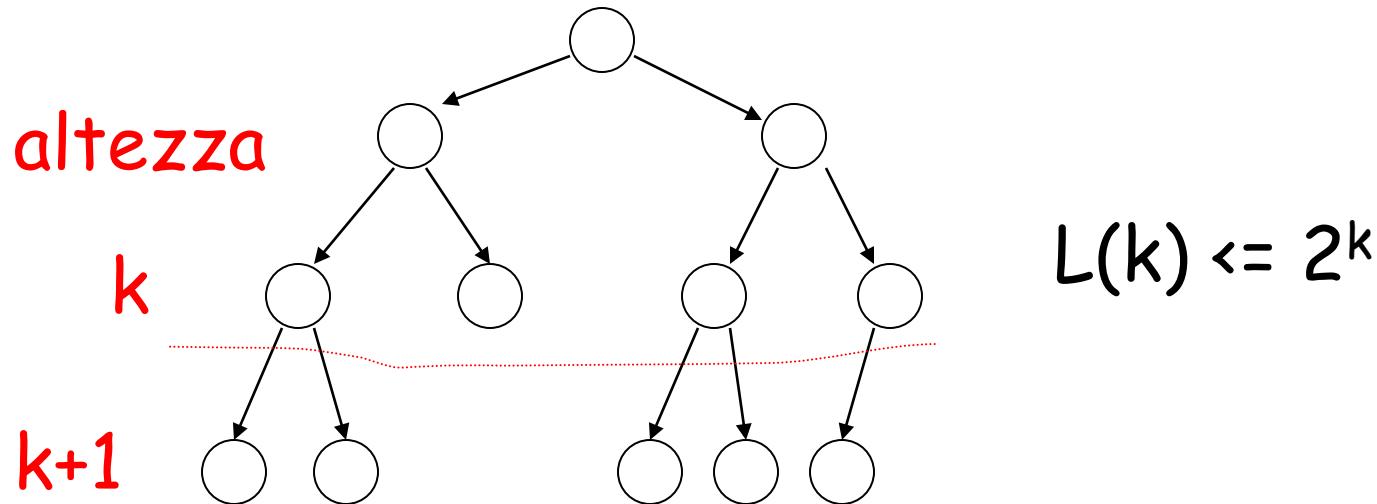
- Base induttiva
- $L(0) = 1$ (nodo radice)
- Ipotesi induttiva
- Assumiamo che $L(i) \leq 2^i$ for all $i = 0, 1, \dots, k$
- Step induttivo
- Dobbiamo dimostrare che $L(k + 1) \leq 2^{k+1}$

Step induttivo



Per ipotesi induttiva: $L(k) \leq 2^k$

Step induttivo



$$L(k+1) \leq 2 * L(k) \leq 2 * 2^k = 2^{k+1}$$

(possiamo addizionare al massimo due nodi per ogni
Foglia di livello k)

Remark

La ricorsione è un'altra cosa

Esempio di funzione ricorsiva:

$$f(n) = f(n-1) + f(n-2)$$

$$f(0) = 1, \quad f(1) = 1$$

Dimostrazione per assurdo

Vogliamo provare che P è vero

- Assumiamo che P è falso
- arriviamo ad una conclusione sbagliata
- quindi, P deve essere vero.

Esempio

Teorema: $\sqrt{2}$ non è razionale

Dimostrazione:

Assumiamo per assurdo che sia razionale

$$\sqrt{2} = n/m$$

n e m non devono avere fattori comuni

Proviamo che questa affermazione è impossibile

$$\sqrt{2} = n/m \longrightarrow 2m^2 = n^2$$

quindi, n^2 è pari
quindi n è pari
(quadrato di dispari
è dispari)

\longrightarrow

n è pari
 $n = 2k$

$$2m^2 = 4k^2 \longrightarrow m^2 = 2k^2 \longrightarrow$$

m è pari
 $m = 2p$

Allora, m e n hanno come fattore comune 2

Contradizione!

Linguaggi

Linguaggio: un insieme di stringhe

Stringa: una sequenza di simboli da un alfabeto

Esempio:

Stringhe: gatto, cane, casa

Linguaggio: {gatto, cane, casa}

Alfabeto: $\Sigma = \{a, b, c, \dots, z\}$

Linguaggi sono usati per descrivere
problemi di calcolo:

$$PRIMI = \{2, 3, 5, 7, 11, 13, 17, \dots\}$$

$$Pari = \{0, 2, 4, 6, \dots\}$$

Alfabeto: $\Sigma = \{0, 1, 2, \dots, 9\}$

Alfabeti e Stringe

Un alfabeto è un insieme di simboli

Esempio Alfabeto: $\Sigma = \{a, b\}$

Una stringa è una sequenza di simboli da un alfabeto

Esempio Stringhe

a	$u = ab$
ab	$v = bbbaaa$
$abba$	$w = abba$
$aaabbbaabab$	

Alfabeto dei numeri decimali

$$\Sigma = \{0,1,2,\dots,9\}$$

102345

567463386

Alfabeto dei numeri binari $\Sigma = \{0,1\}$

100010001

101101111

Alfabeto dei numeri unari $\Sigma = \{1\}$

Numeri unari : 11 111 1111 11111

Numeri decimali: 1 2 3 4

Zero?

Operazioni su stringhe

 $w = a_1 a_2 \cdots a_n$ $abba$ $v = b_1 b_2 \cdots b_m$ $bbbaaa$

Concatenazione

 $wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m$ $abbabbbbaaa$

$w = a_1 a_2 \cdots a_n$ $ababa aabb b$

Inverso

 $w^R = a_n \cdots a_2 a_1$ $b b b a a a b a b a$

Lunghezza di una stringa

$$w = a_1 a_2 \cdots a_n$$

Lunghezza: $|w| = n$

Esempi: $|abba| = 4$

$|aa| = 2$

$|a| = 1$

Lunghezza della concatenazione

$$|uv| = |u| + |v|$$

Esempio: $u = aab, |u| = 3$

$v = abaab, |v| = 5$

$$|uv| = |aababaab| = 8$$

$$|uv| = |u| + |v| = 3 + 5 = 8$$

Stringa vuota

Una stringa con nessuna lettera è denotata:

$$\lambda \text{ o } \varepsilon$$

Osservazioni:

$$|\lambda| = 0$$

$$\lambda w = w\lambda = w$$

$$\lambda abba = abba\lambda = ab\lambda ba = abba$$

Sottostringa

Sottostringa di una stringa:

Una sequenza consecutiva di caratteri:

Stringa

abbab

abbab

abbab

abbab

Sottostringa

ab

abba

b

bbab

Prefisso e Suffisso

abbab

Prefisso

Suffisso

λ

abbab

a

bbab

ab

bab

abb

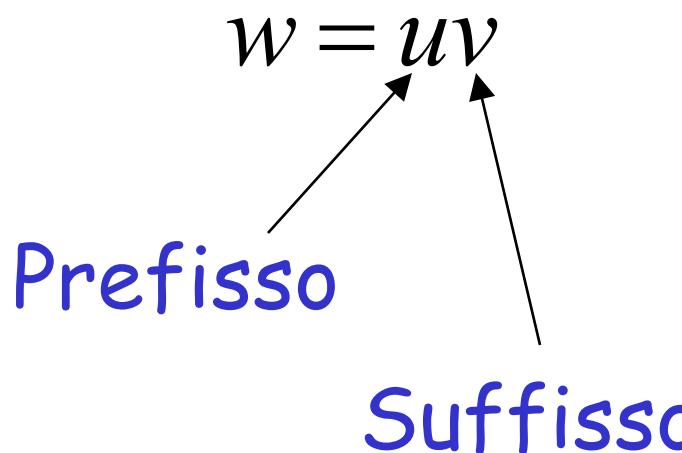
ab

abba

b

abbab

λ



Altre operazioni

$$w^n = \underbrace{ww\cdots w}_n$$

Esempio: $(abba)^2 = abbaabba$

Definizione: $w^0 = \lambda$

$$(abba)^0 = \lambda$$

L'operazione *

Σ^* : L'insieme di tutte le possibili stringe che è possibile generare a partire dall'alfabeto

$$\Sigma$$

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

L'operazione +

Σ^+ : L'insieme di tutte le possibili stringhe che è possibile generare a partire dall'alfabeto Σ eccetto λ

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

$$\Sigma^+ = \Sigma^* - \lambda$$

$$\Sigma^+ = \{a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

Linguaggi. estensionali

Un linguaggio su un alfabeto Σ

È un qualsiasi sottoinsieme di Σ^*

Esempio:

$$\Sigma = \{a, b\}$$

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, \dots\}$$

linguaggio : $\{\lambda\}$

linguaggio : $\{a, aa, aab\}$

linguaggio : $\{\lambda, abba, baba, aa, ab, aaaaaaa\}$

Esempi di linguaggi

Alfabeto $\Sigma = \{a, b\}$

Un linguaggio infinito $L = \{a^n b^n : n \geq 0\}$

λ

ab

$aabb$

$aaaaabbbbb$

$\} \in L$

$abb \notin L$

Numeri primi

alfabeto $\Sigma = \{0,1,2,\dots,9\}$

Linguaggio:

$PRIMES = \{x : x \in \Sigma^* \text{ and } x \text{ is prime}\}$

$PRIMES = \{2,3,5,7,11,13,17,\dots\}$

Numeri pari e dispari

alfabeto $\Sigma = \{0,1,2,\dots,9\}$

$EVEN = \{x : x \in \Sigma^* \text{ e } x \text{ è pari}\}$

$EVEN = \{0,2,4,6,\dots\}$

$ODD = \{x : x \in \Sigma^* \text{ e } x \text{ è dispari}\}$

$ODD = \{1,3,5,7,\dots\}$

Somma unaria

alfabeto: $\Sigma = \{1, +, =\}$

Linguaggio:

$ADDITION = \{x + y = z : x = 1^n, y = 1^m, z = 1^k, n + m = k\}$

$11 + 111 = 11111 \in ADDITION$

$111 + 111 = 111 \notin ADDITION$

Radici

Alfabeto: $\Sigma = \{1, \#\}$

Linguaggio:

$SQUARES = \{x \# y : x = 1^n, y = 1^m, m = n^2\}$

$11 \# 1111 \in SQUARES$

$111 \# 1111 \notin SQUARES$

Nota che :

Insieme vuoto $\emptyset = \{\} \neq \{\lambda\}$

Dimensione insiemi

$$|\{\}| = |\emptyset| = 0 \quad |\{\lambda\}| = 1$$

Lunghezza di una stringa

$$|\lambda| = 0$$

Operazioni sui linguaggi

Le stesse degli insiemi

$$\{a, ab, aaaa\} \cup \{bb, ab\} = \{a, ab, bb, aaaa\}$$

$$\{a, ab, aaaa\} \cap \{bb, ab\} = \{ab\}$$

$$\{a, ab, aaaa\} - \{bb, ab\} = \{a, aaaa\}$$

Complemento: $\overline{L} = \Sigma^* - L$

$$\overline{\{a, ba\}} = \{\lambda, b, aa, ab, bb, aaa, \dots\}$$

Inverso

Definizione: $L^R = \{w^R : w \in L\}$

Esempio: $\{ab, aab, baba\}^R = \{ba, baa, abab\}$

$$L = \{a^n b^n : n \geq 0\}$$

$$L^R = \{b^n a^n : n \geq 0\}$$

Concatenazione

Definizione: $L_1 L_2 = \{xy : x \in L_1, y \in L_2\}$

Esempio: $\{a, ab, ba\} \{b, aa\}$

$$= \{ab, aaa, abb, abaa, bab, baaa\}$$

Altre operazioni

Definizione: $L^n = \underbrace{LL\cdots L}_n$

$$\begin{aligned}\{a,b\}^3 &= \{a,b\}\{a,b\}\{a,b\} = \\ \{aaa, aab, aba, abb, baa, bab, bba, bbb\} &\end{aligned}$$

Casi speciale: $L^0 = \{\lambda\}$

$$\{a, bba, aaa\}^0 = \{\lambda\}$$

$$L = \{a^n b^n : n \geq 0\}$$

$$L^2 = \{a^n b^n a^m b^m : n, m \geq 0\}$$

$$aabbaaaabbb \in L^2$$

Star-Closure-intensione (Kleene *)

Tutte le stringhe che possono essere costruite da L

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

Definizione:

Esempio:

$$\{a, bb\}^* = \left\{ \lambda, a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbbb, \dots \right\}$$

Chiusure

Definizione: $L^+ = L^1 \cup L^2 \cup \dots$

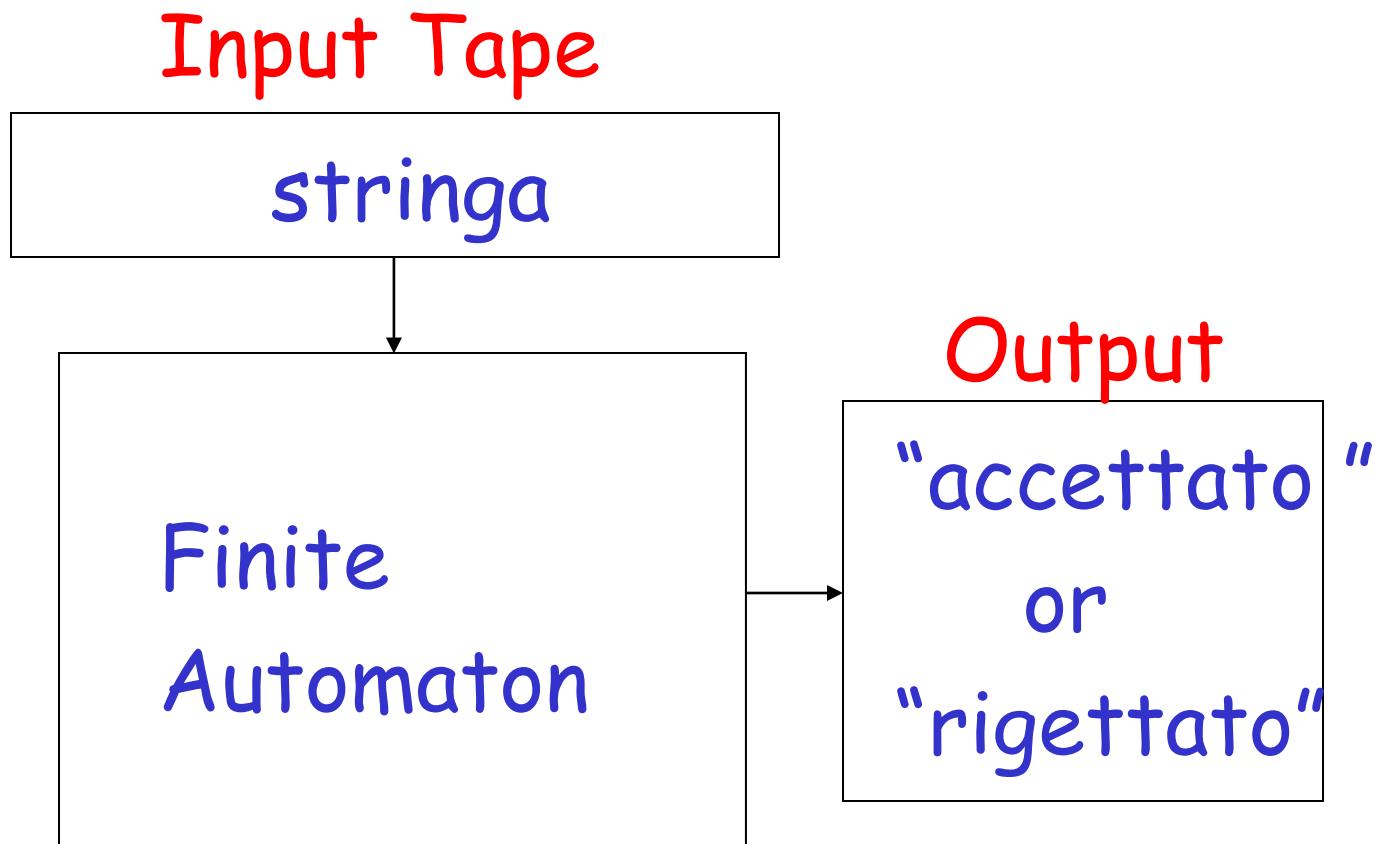
Lo stesso come L^* without the λ

$$\{a,bb\}^+ = \left\{ \begin{array}{l} a, bb, \\ aa, abb, bba, bbbb, \\ aaa, aabb, abba, abbbb, \dots \end{array} \right\}$$

Deterministic Finite Automata

E linguaggi regolari
Simulatore <http://www.jflap.org/>

Deterministic Finite Automaton (DFA)



testa

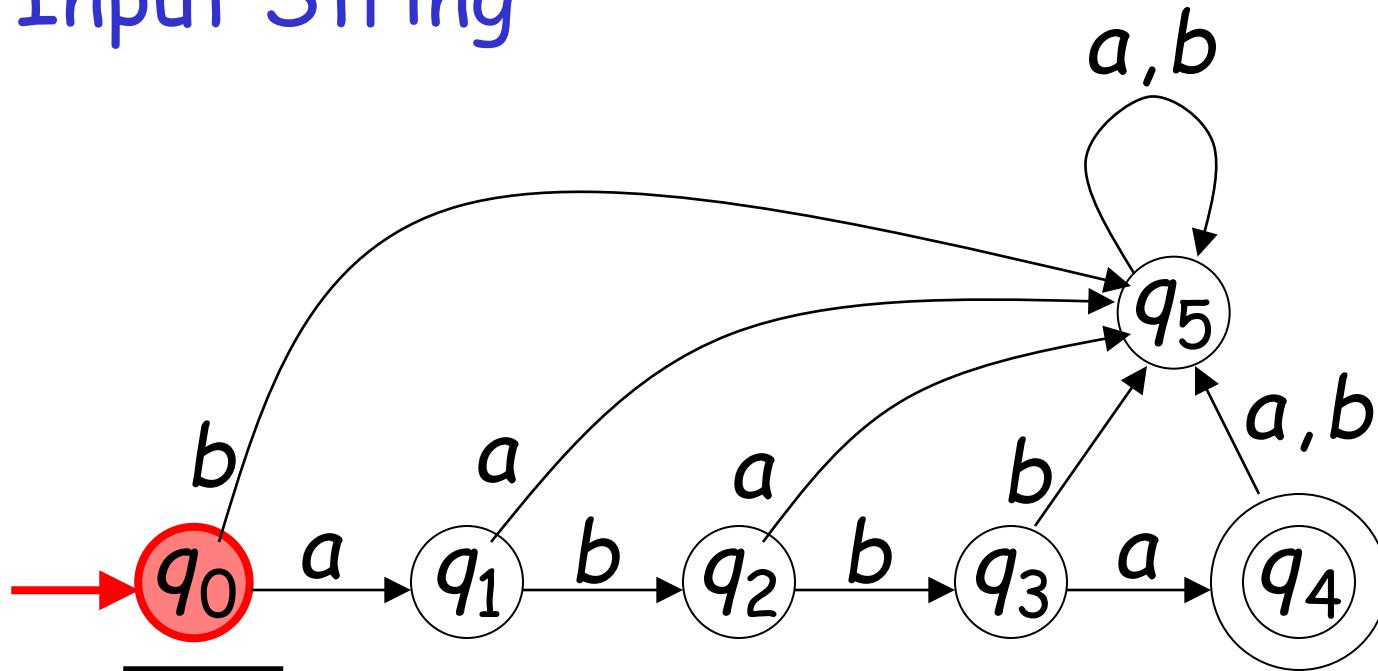
Configurazione iniziale



Input Tape

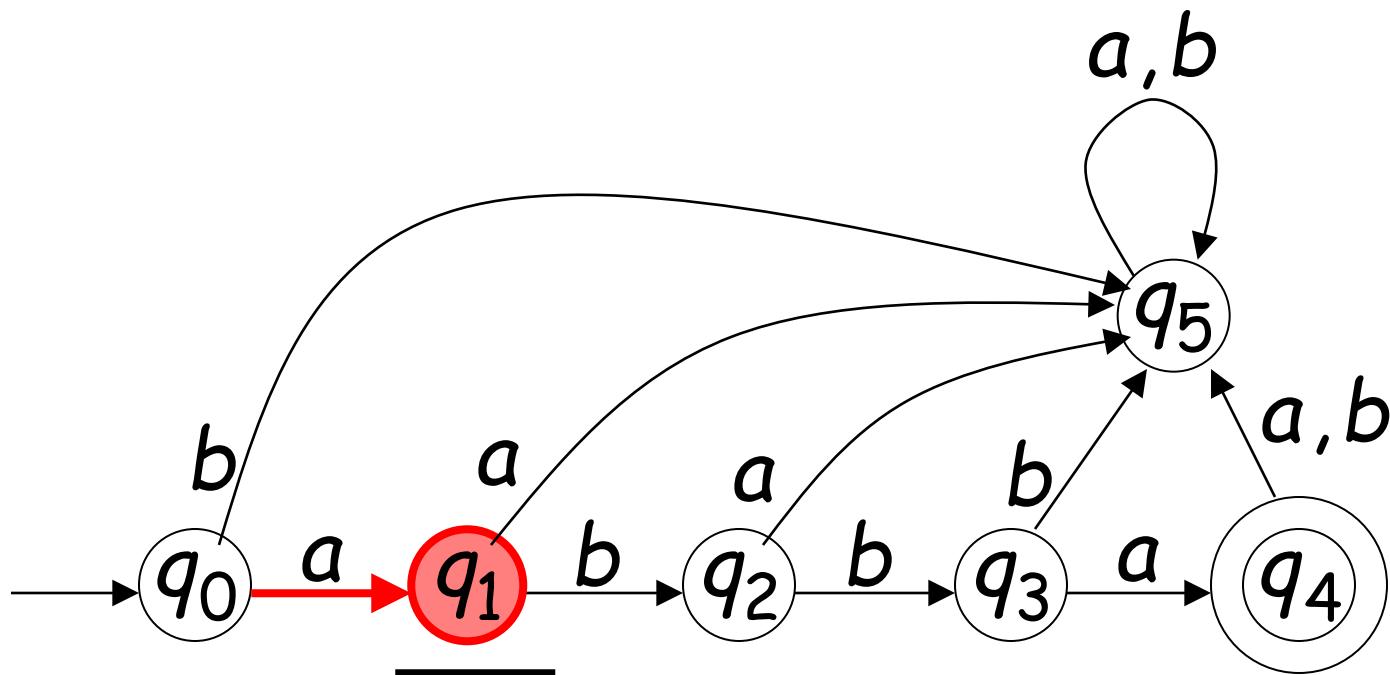
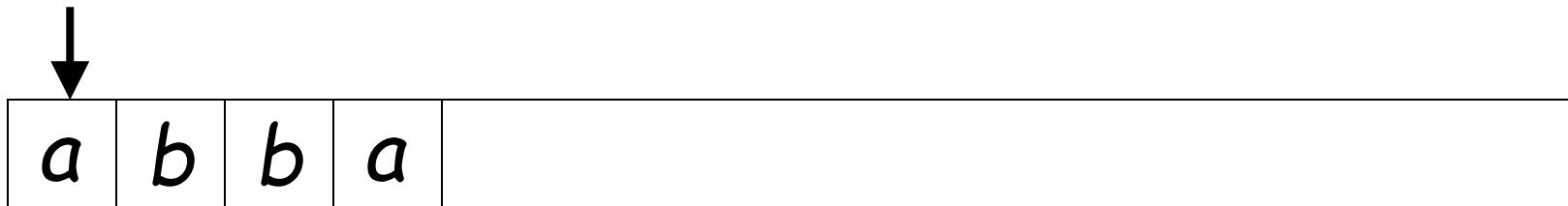
a	b	b	a	
---	---	---	---	--

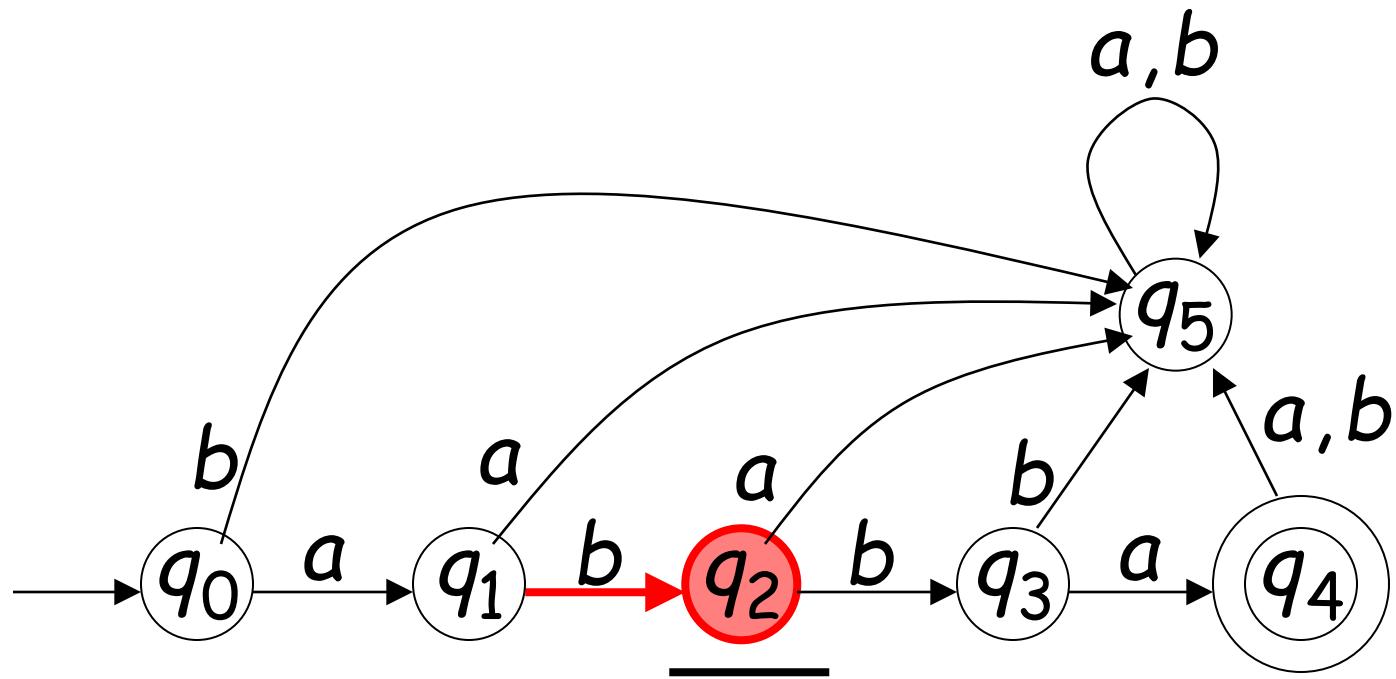
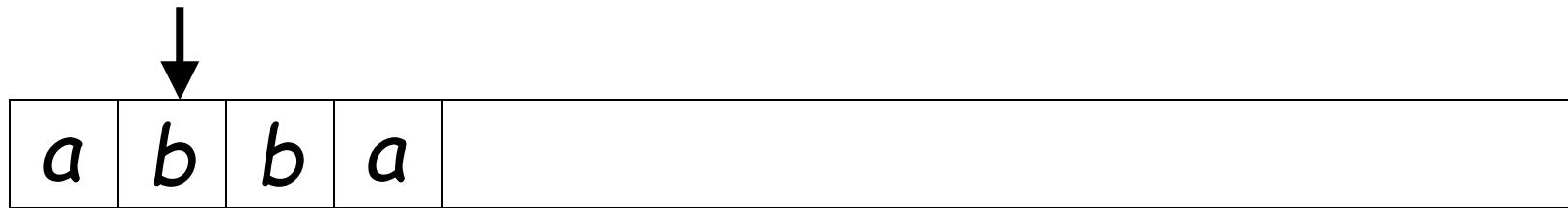
Input String

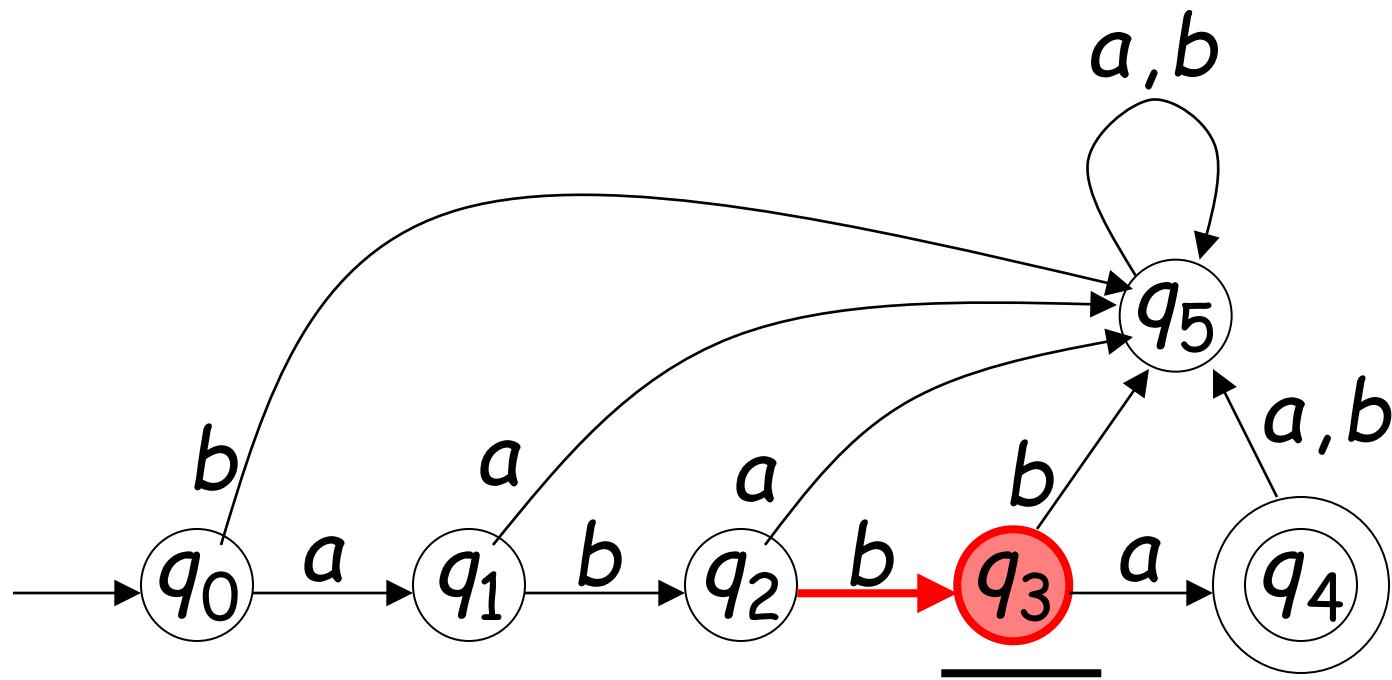
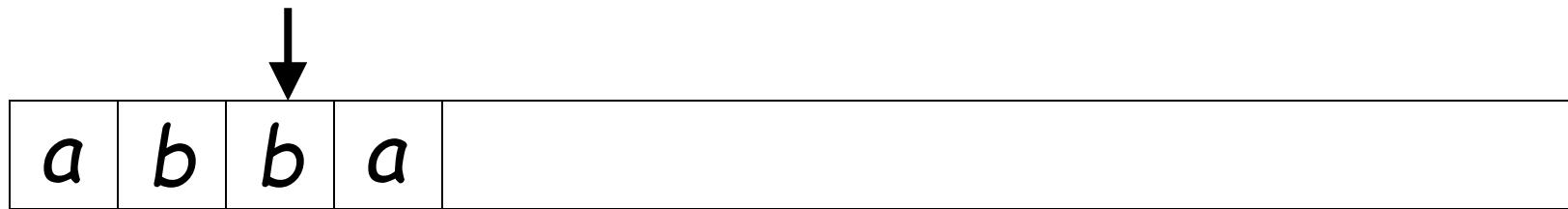


Stato iniziale

Analizzare l'Input



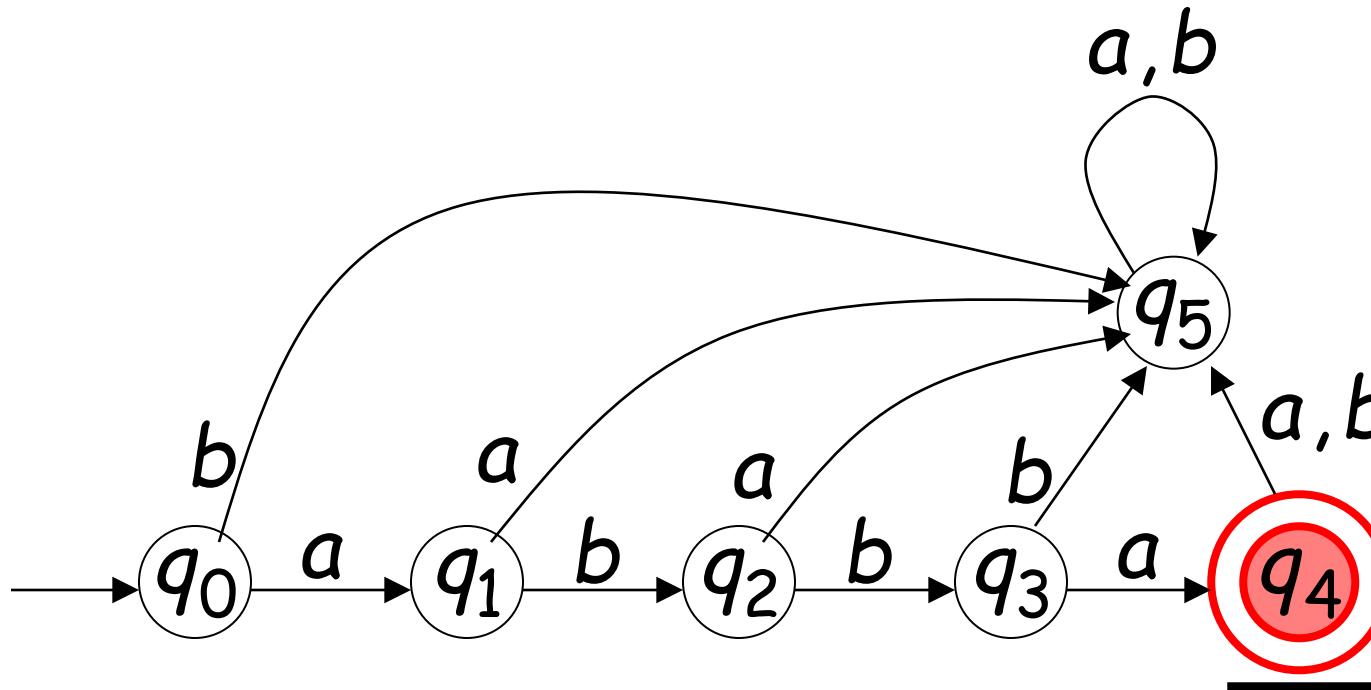




Input finito



a	b	b	a	
---	---	---	---	--



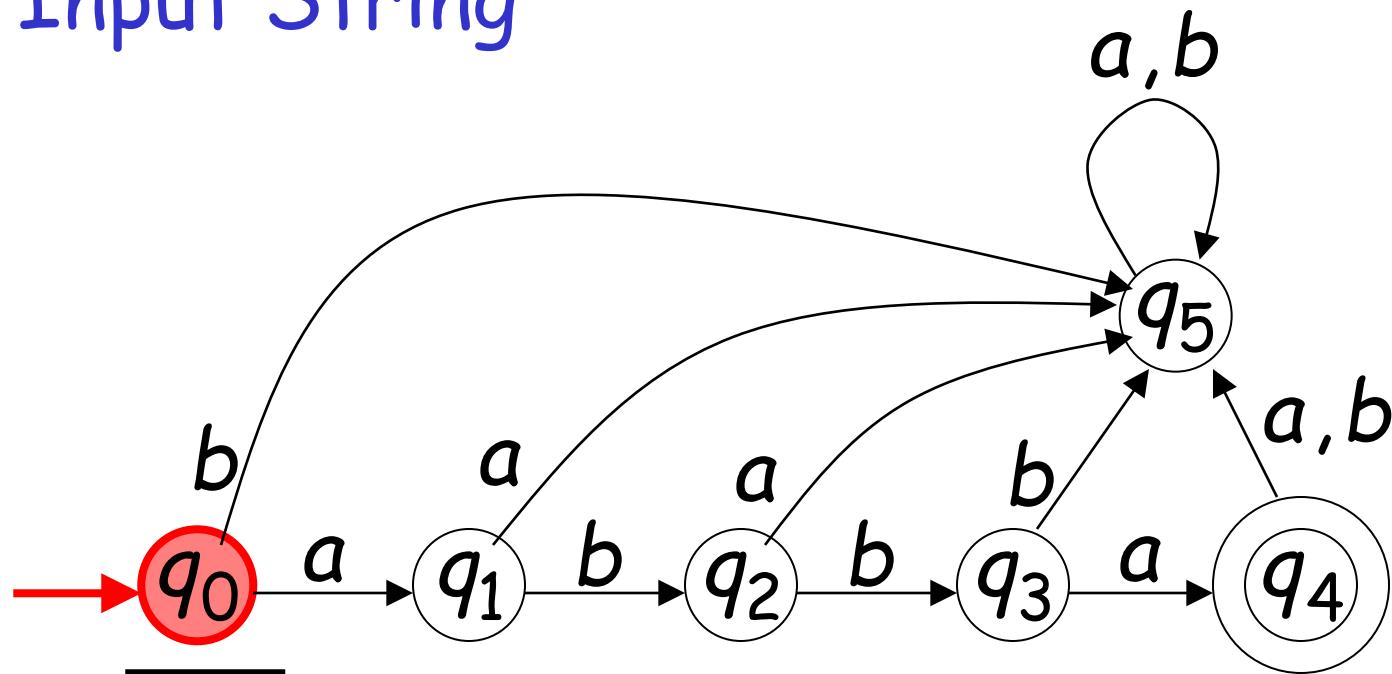
accettato

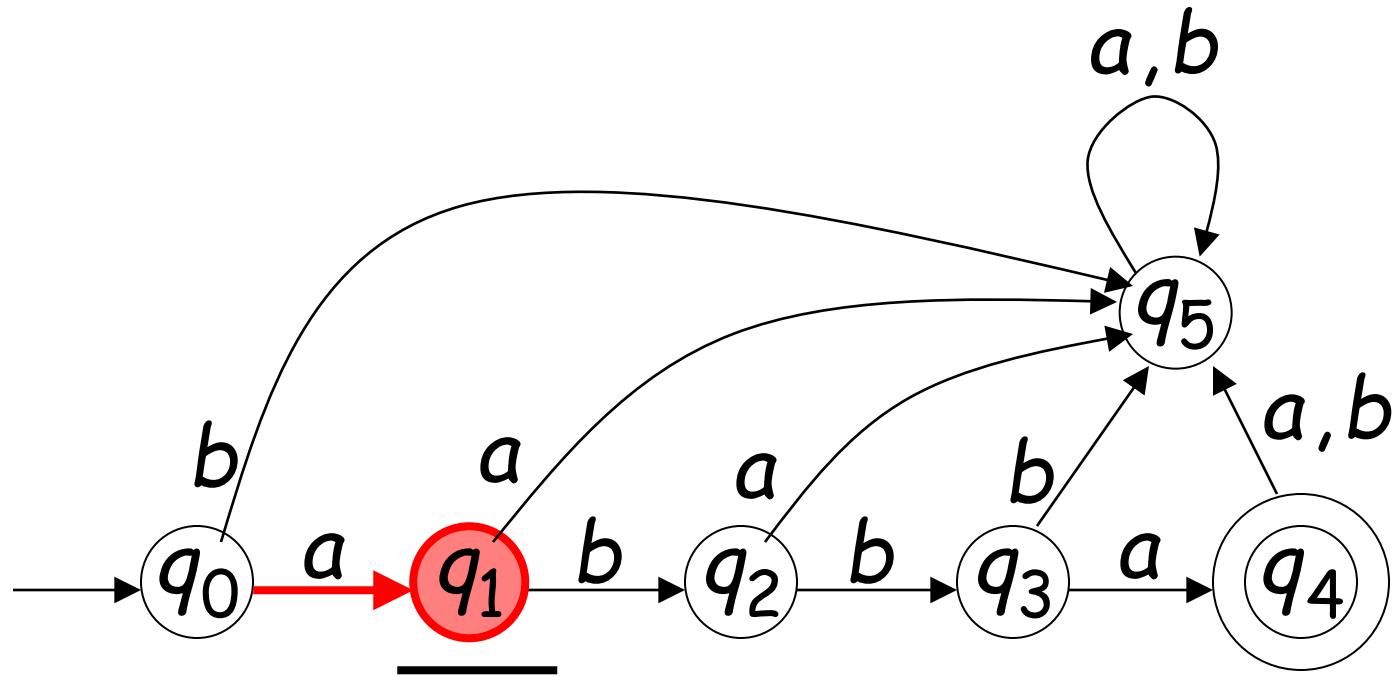
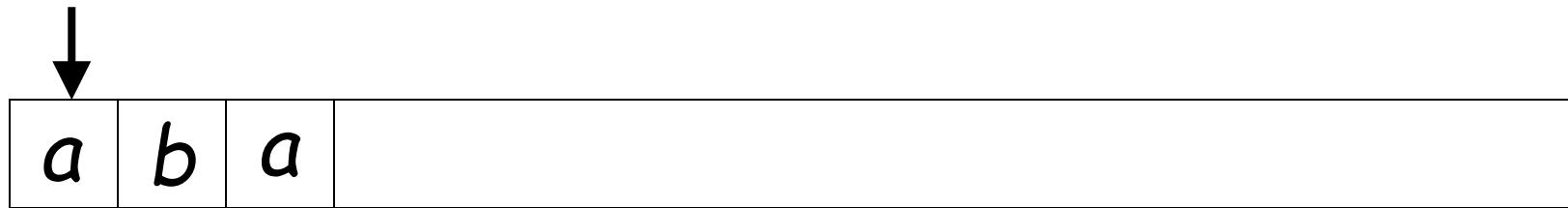
Un caso rigettato

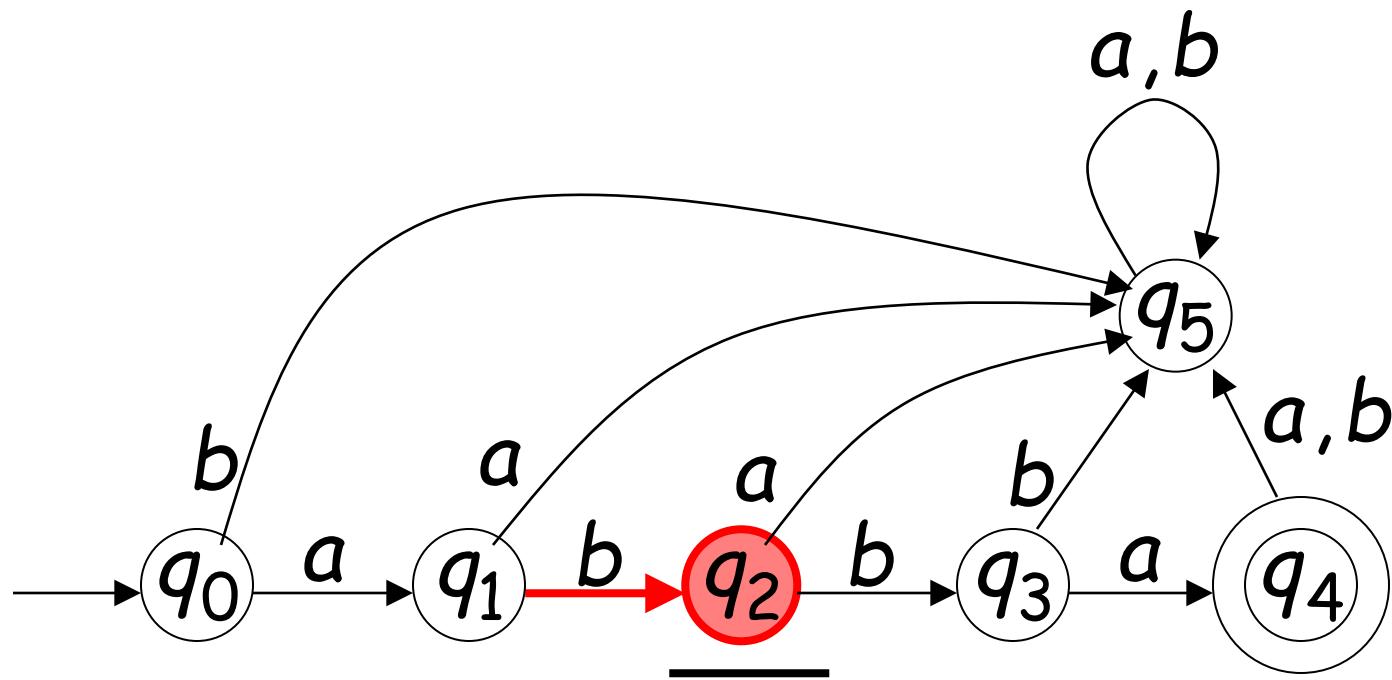
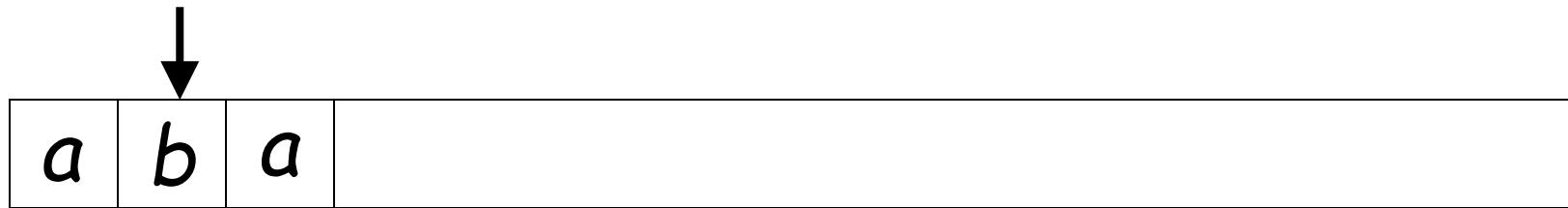


a	b	a	
---	---	---	--

Input String



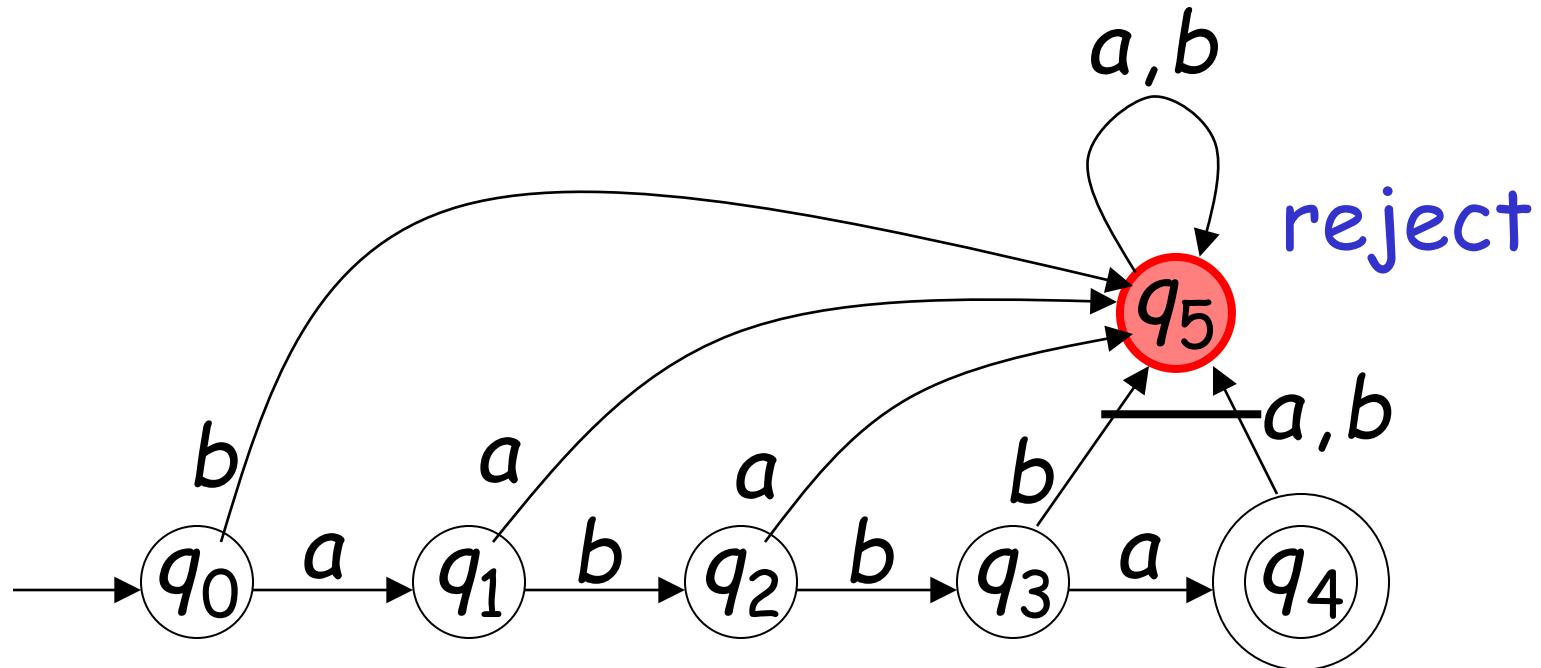




Input finito



a	b	a	
---	---	---	--

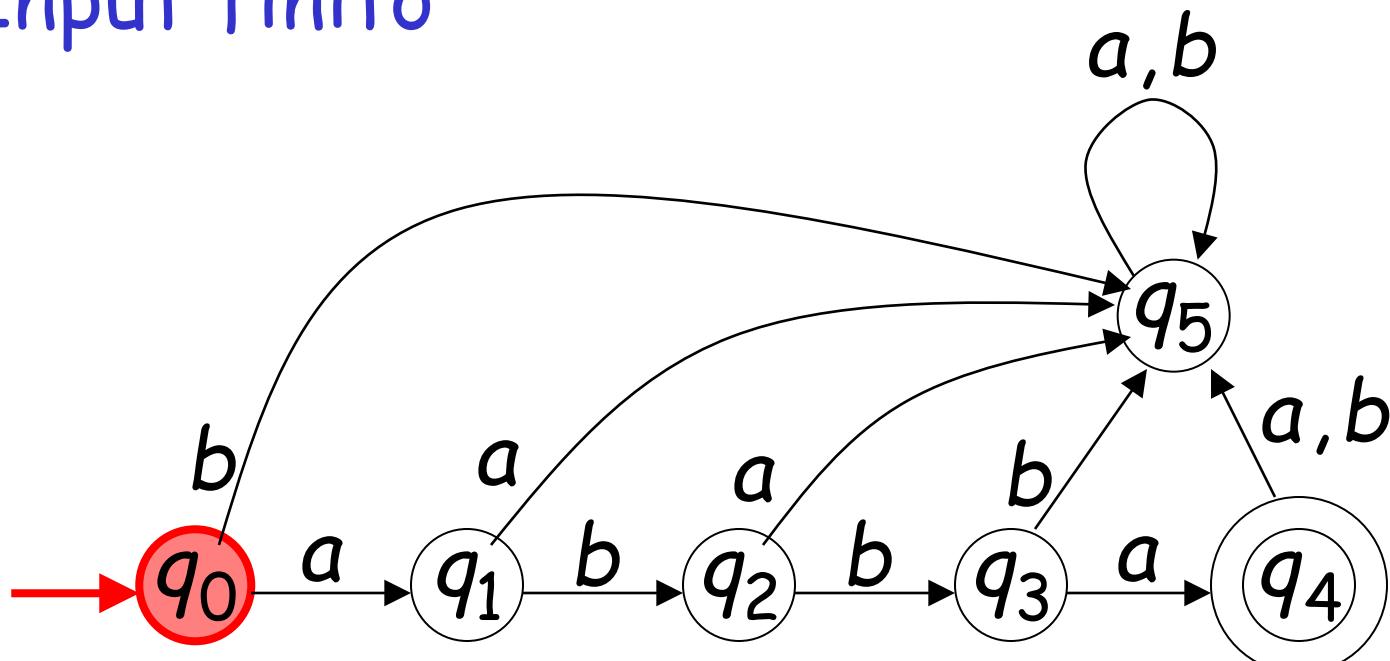


Un altro caso rigettato

Nastro vuoto

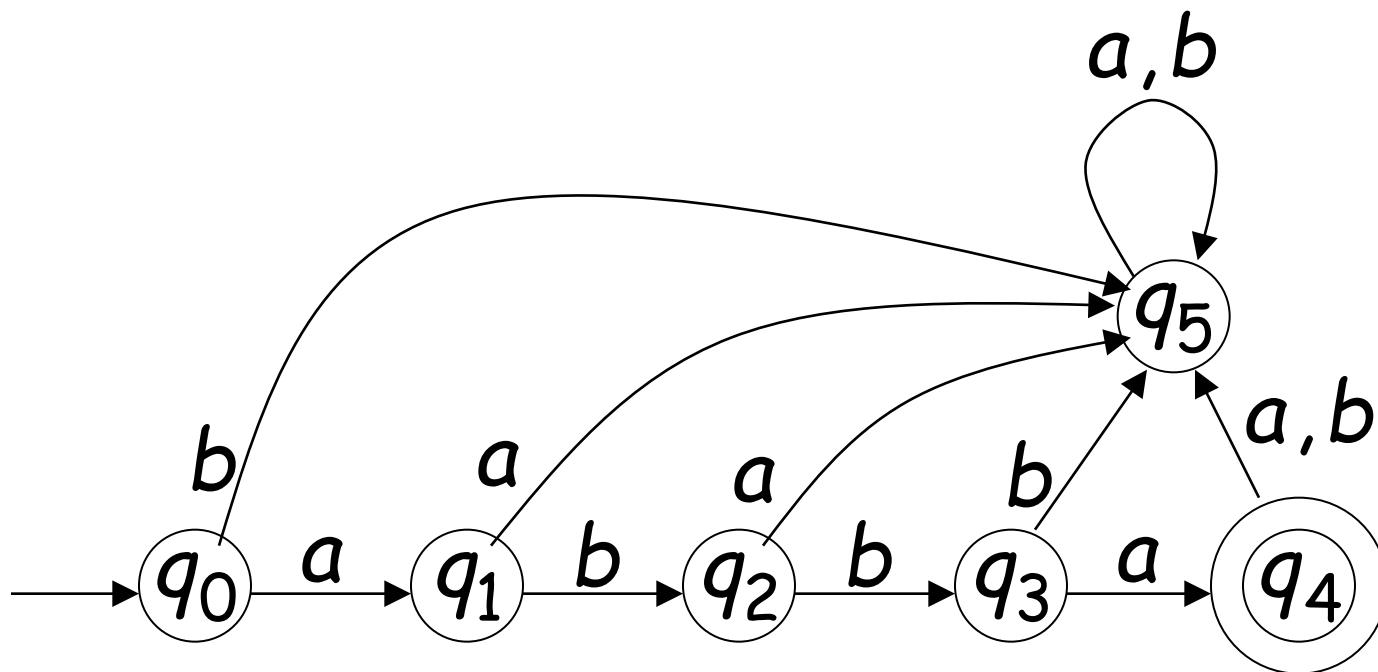
(λ)

Input finito



rigettato

Linguaggio accettato: $L = \{abba\}$



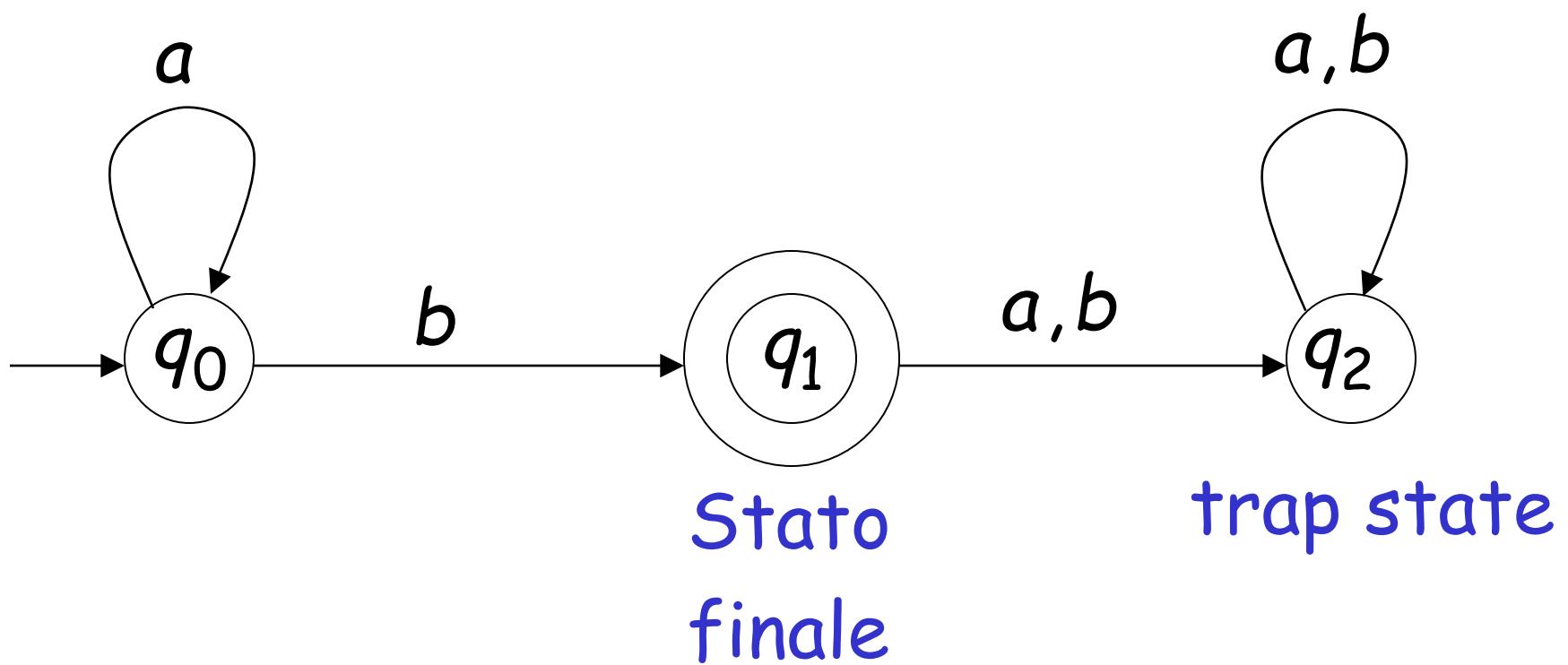
Per accettare una stringa :

Devono essere esaminati tutti i caratteri di Input e l'ultimo stato è uno stato finale

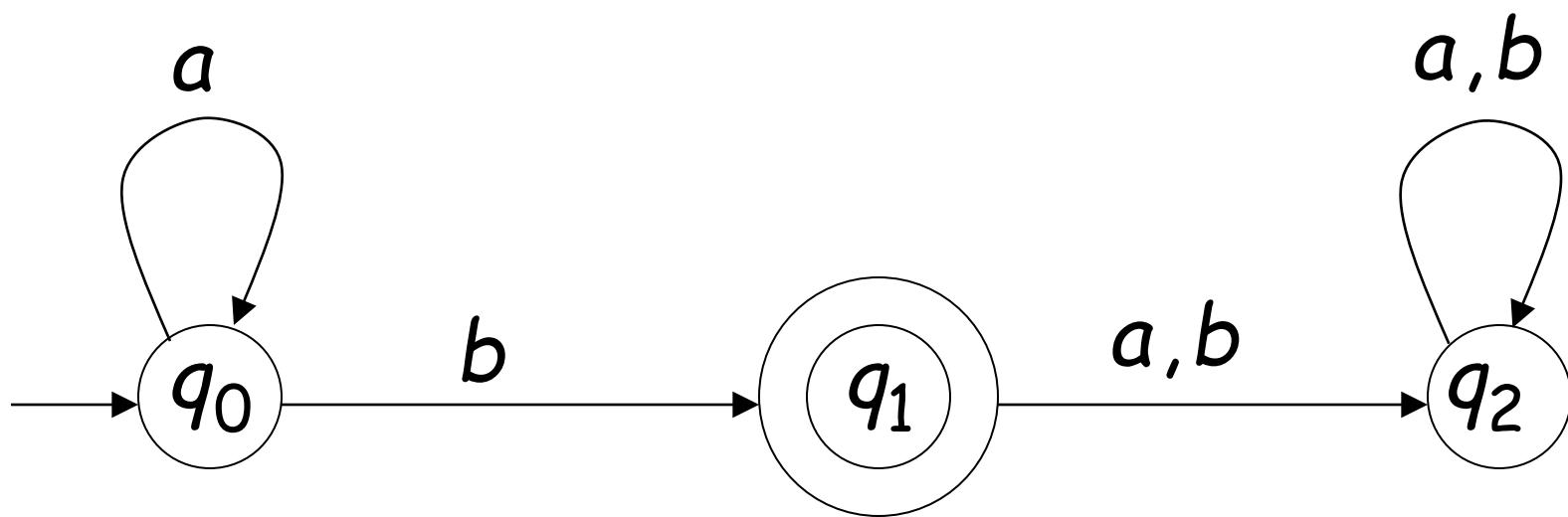
Per rigettare una stringa :

Tutti i caratteri di input sono stati esaminati
E non si è raggiunto uno stato finale

Un altro esempio

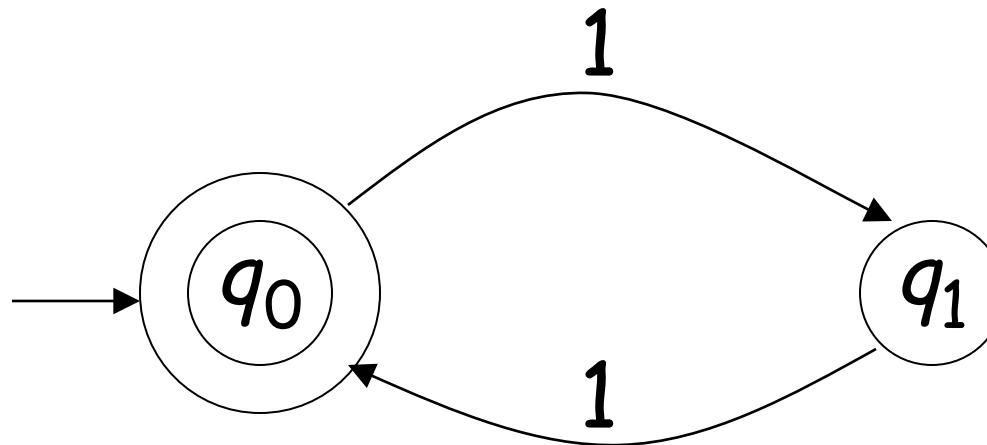


Language Accepted: $L = \{a^n b : n \geq 0\}$



Un altro esempio

Alfabeto: $\Sigma = \{1\}$



Linguaggio accettato:

$$\begin{aligned} EVEN &= \{x : x \in \Sigma^* \text{ and } x \text{ is even}\} \\ &= \{\lambda, 11, 1111, 111111, \dots\} \end{aligned}$$

Definizione formale

un automa deterministico formale(DFA)

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

Q : insieme degli stati

Σ : alfabeto di input $\lambda \notin \Sigma$

δ : funzione di transizione

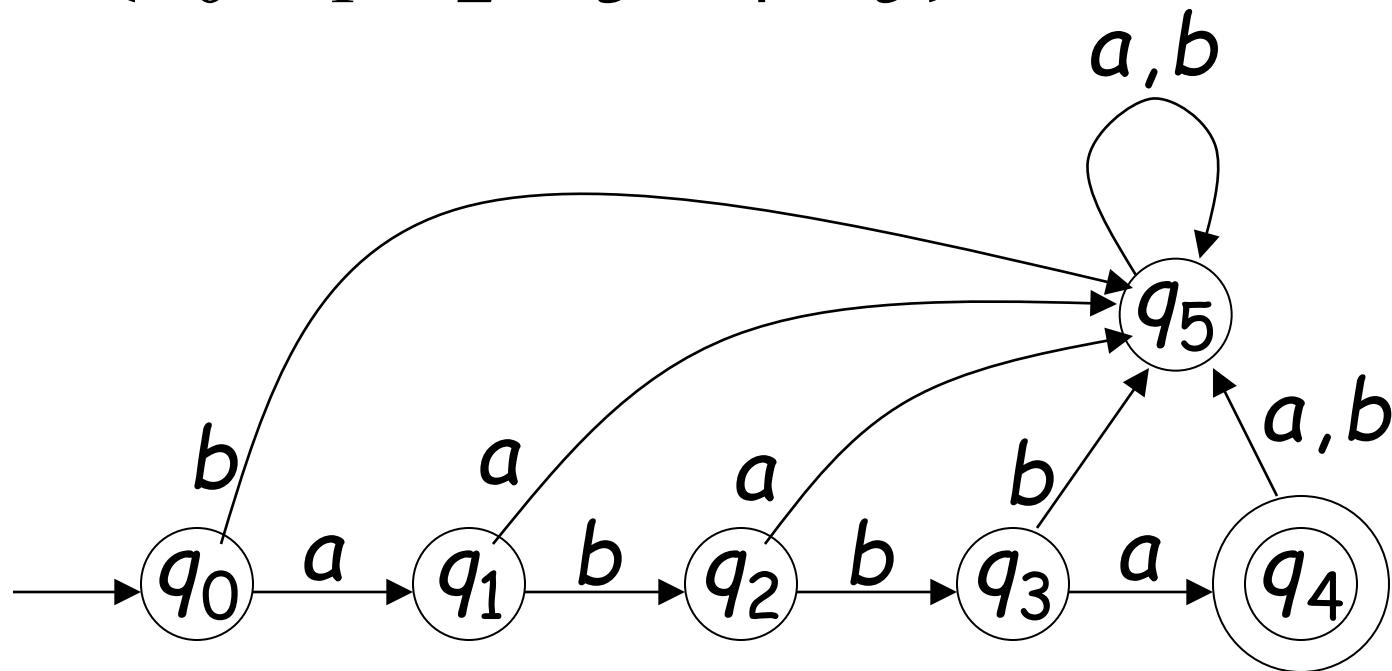
q_0 : stato iniziale

F : insieme degli stati di accettazione
(finale)

Insieme degli stati Q

esempio

$$Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$$

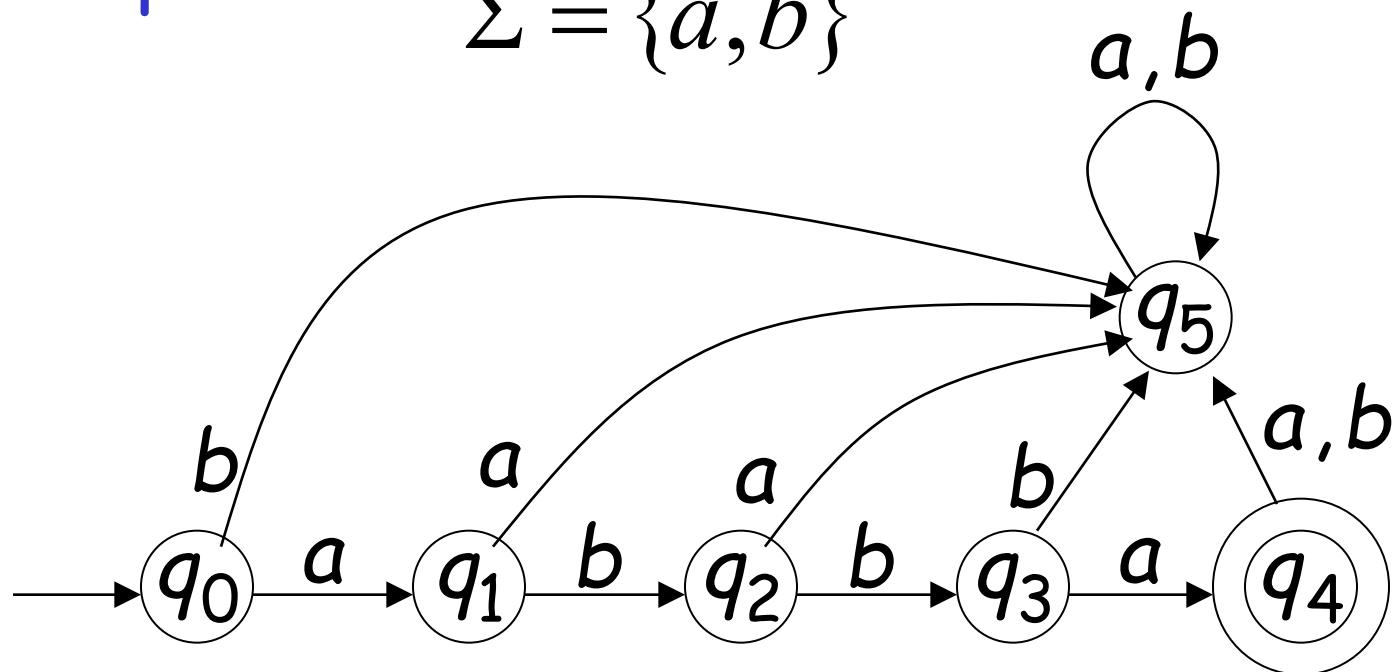


Alfabeto di input Σ

$\lambda \notin \Sigma$: l'alfabeto di input non contiene λ

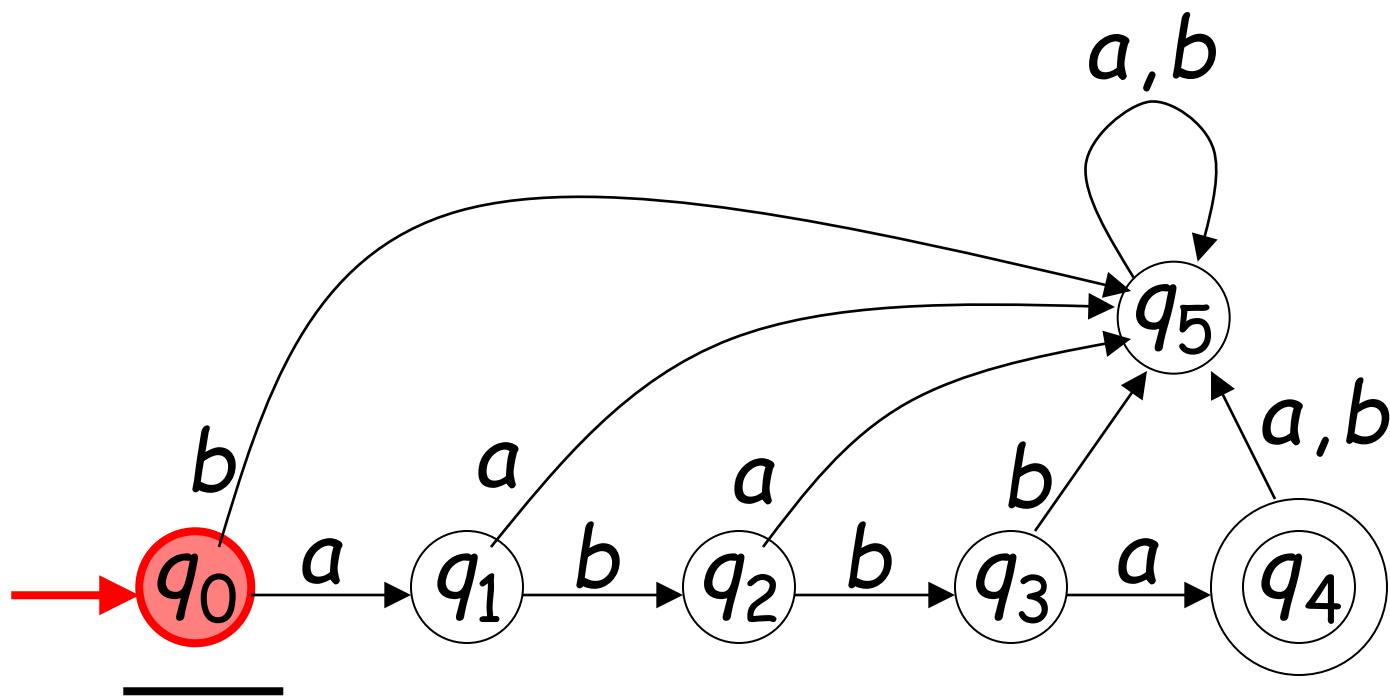
esempio

$$\Sigma = \{a, b\}$$



Stato iniziale q_0

esempio

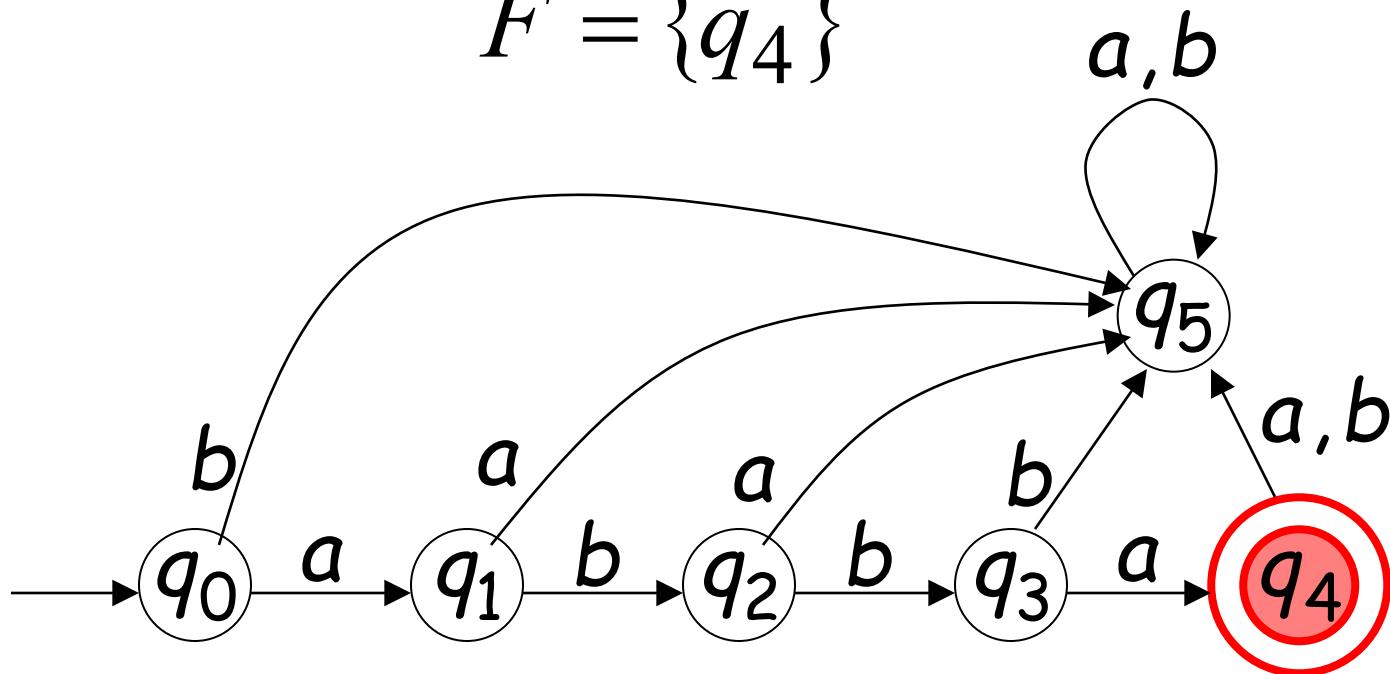


Insieme stati finali

$$F \subseteq Q$$

esempio

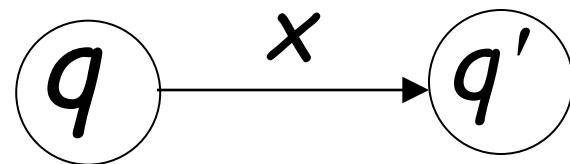
$$F = \{q_4\}$$



Funzione di transizione

$$\delta: Q \times \Sigma \rightarrow Q$$

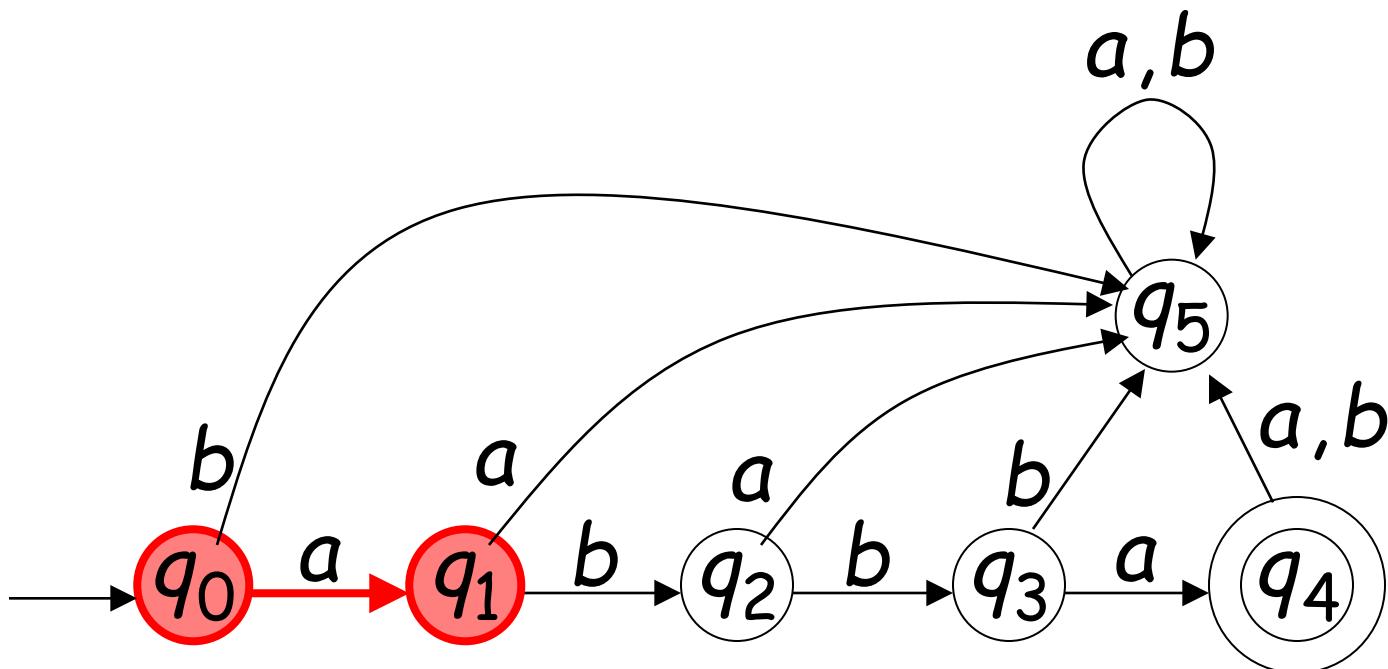
$$\delta(q, x) = q'$$



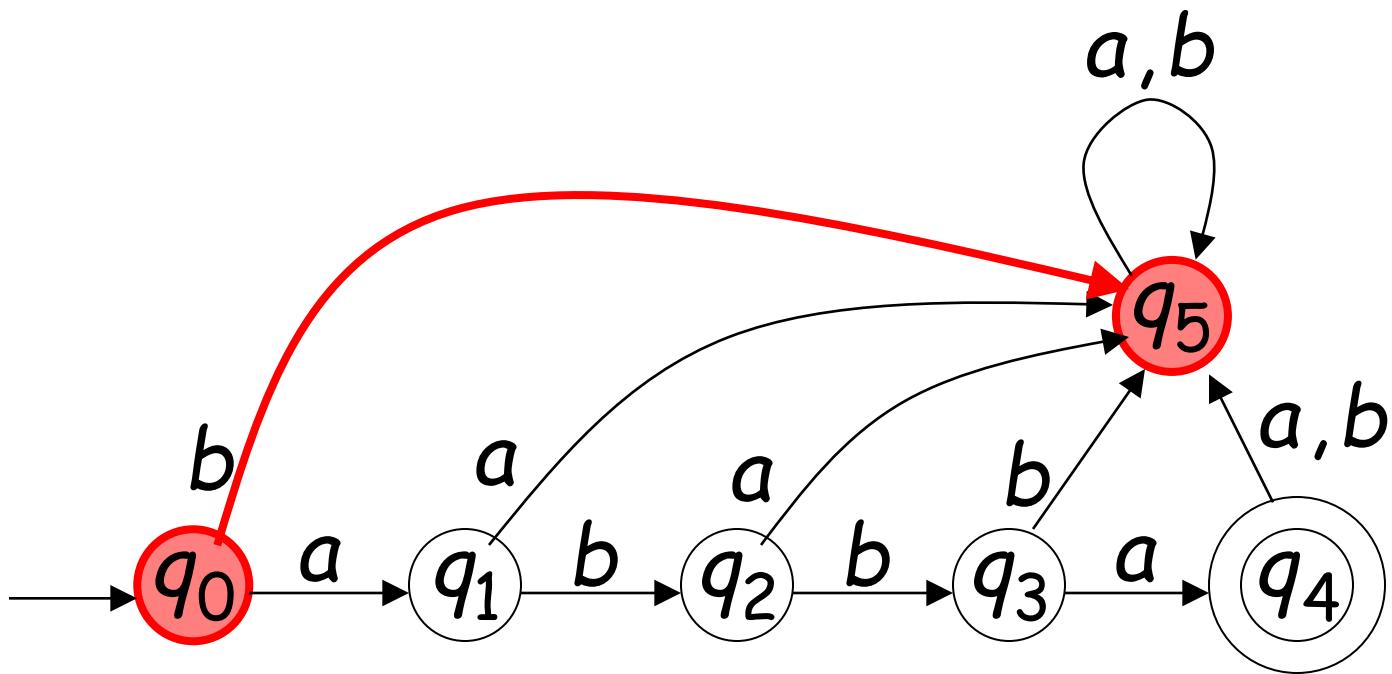
Describe il risultato della
Transizione dallo stato q
Con simbolo x

esempio:

$$\delta(q_0, a) = q_1$$



$$\delta(q_0, b) = q_5$$



$$\delta(q_2, b) = q_3$$

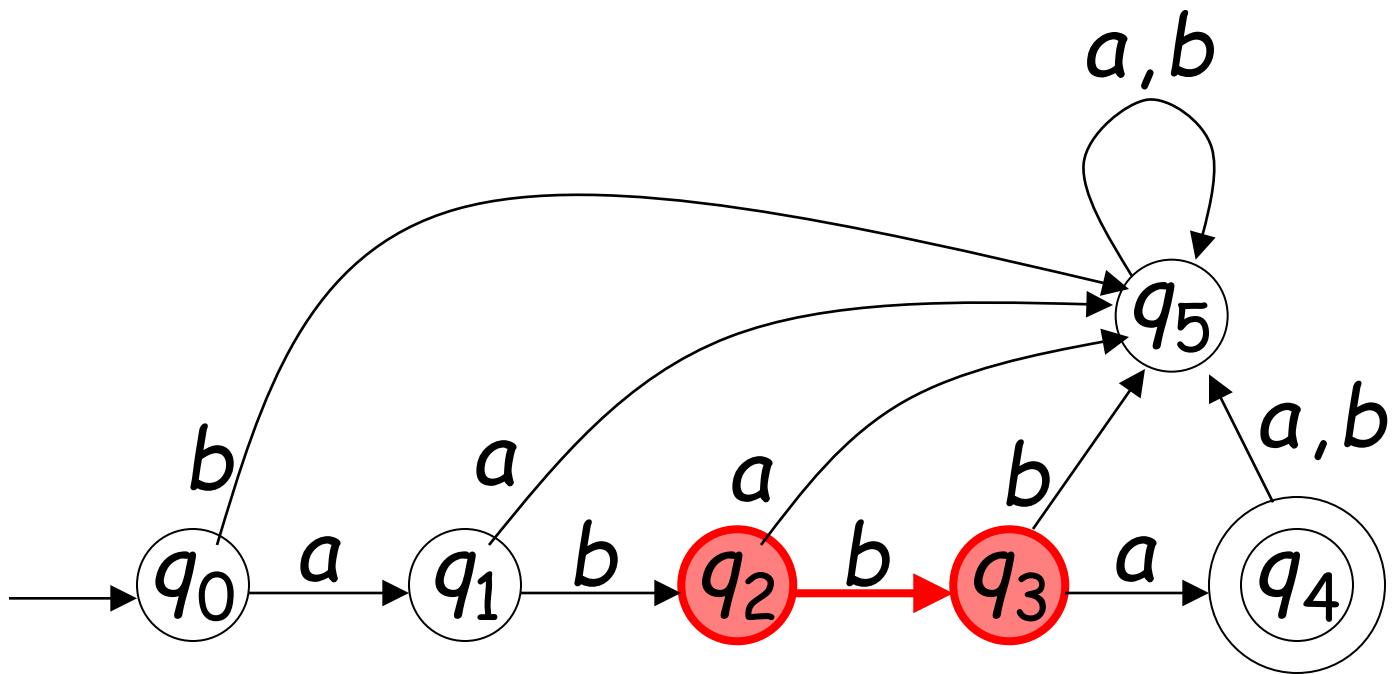
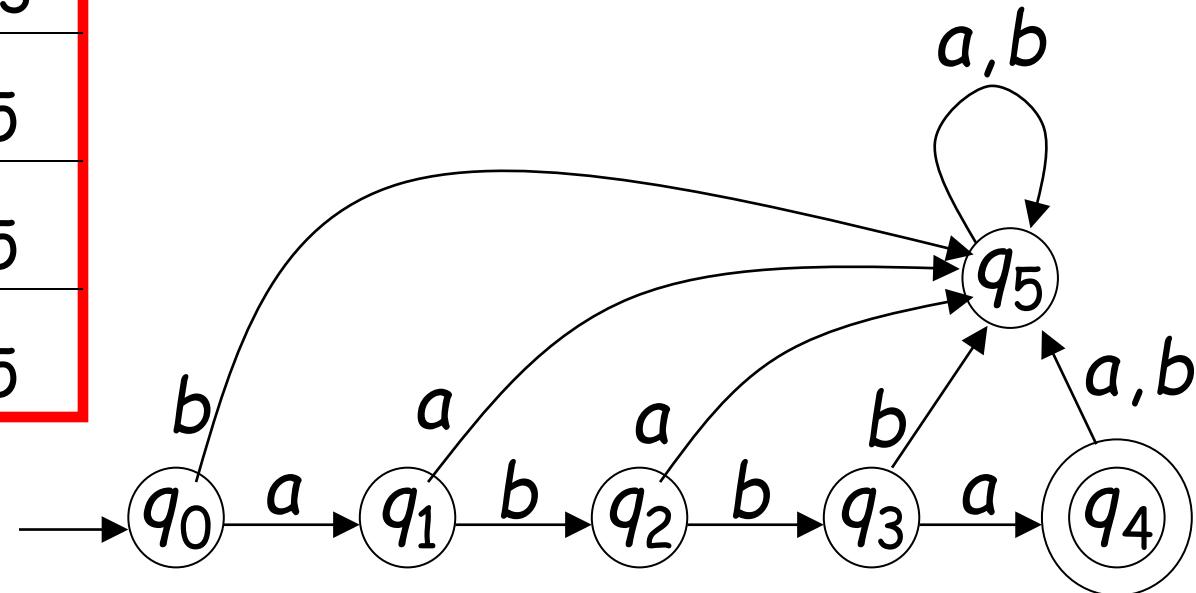


Tavola di transizione per δ

symbols

δ	a	b
q_0	q_1	q_5
q_1	q_5	q_2
q_2	q_5	q_3
q_3	q_4	q_5
q_4	q_5	q_5
q_5	q_5	q_5



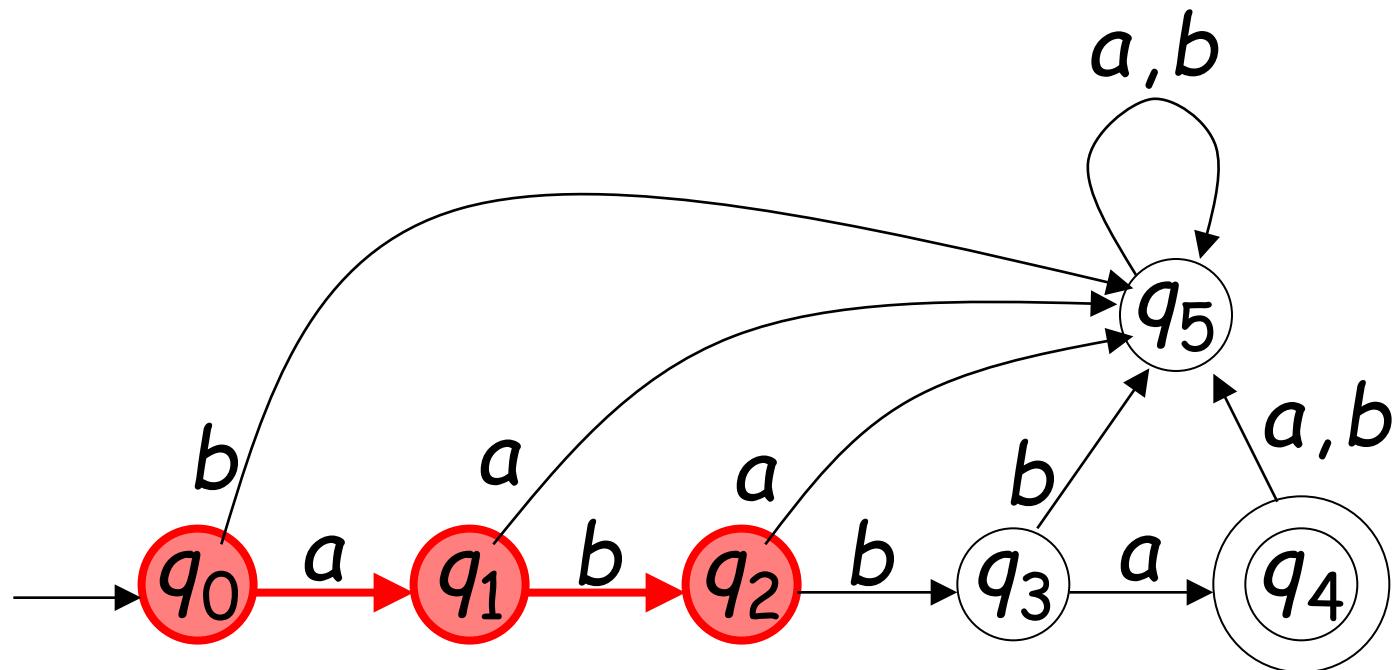
Funzione estesa di transizione

$$\delta^*: Q \times \Sigma^* \rightarrow Q$$

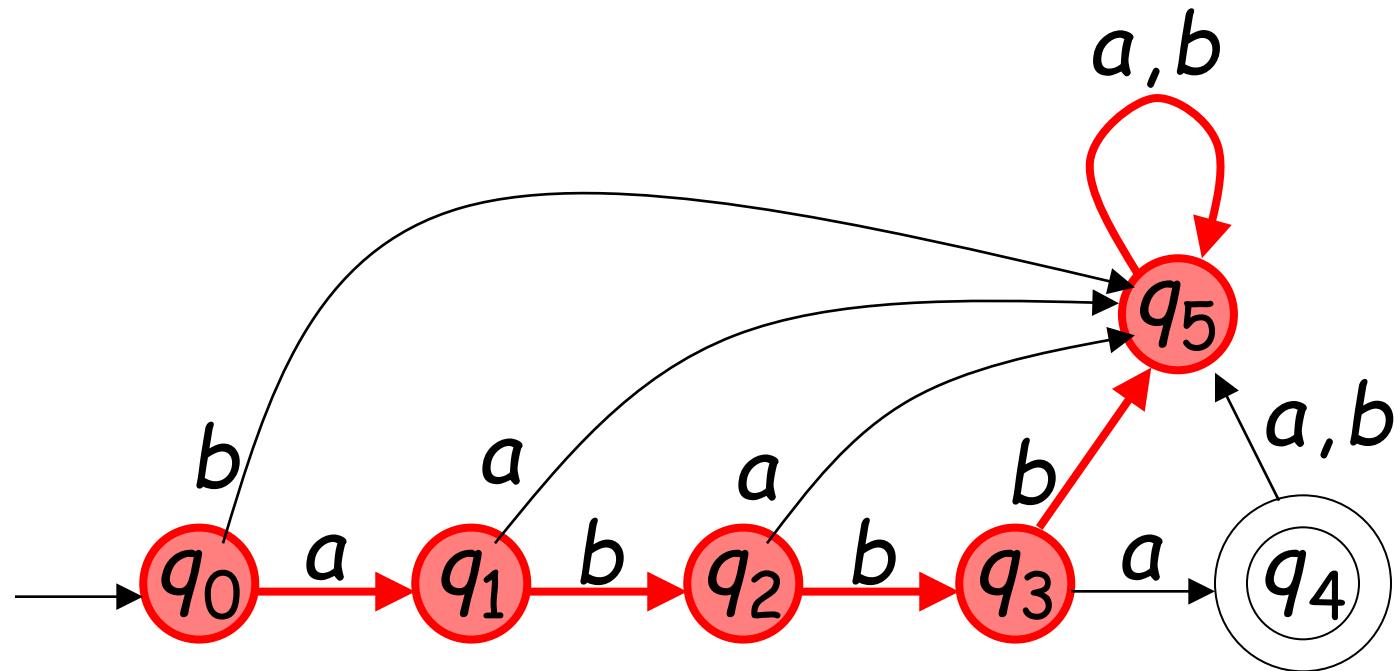
$$\delta^*(q, w) = q'$$

Describe lo stato che risulta dopo aver
Esaminata la stringa w
a partire dallo stato q

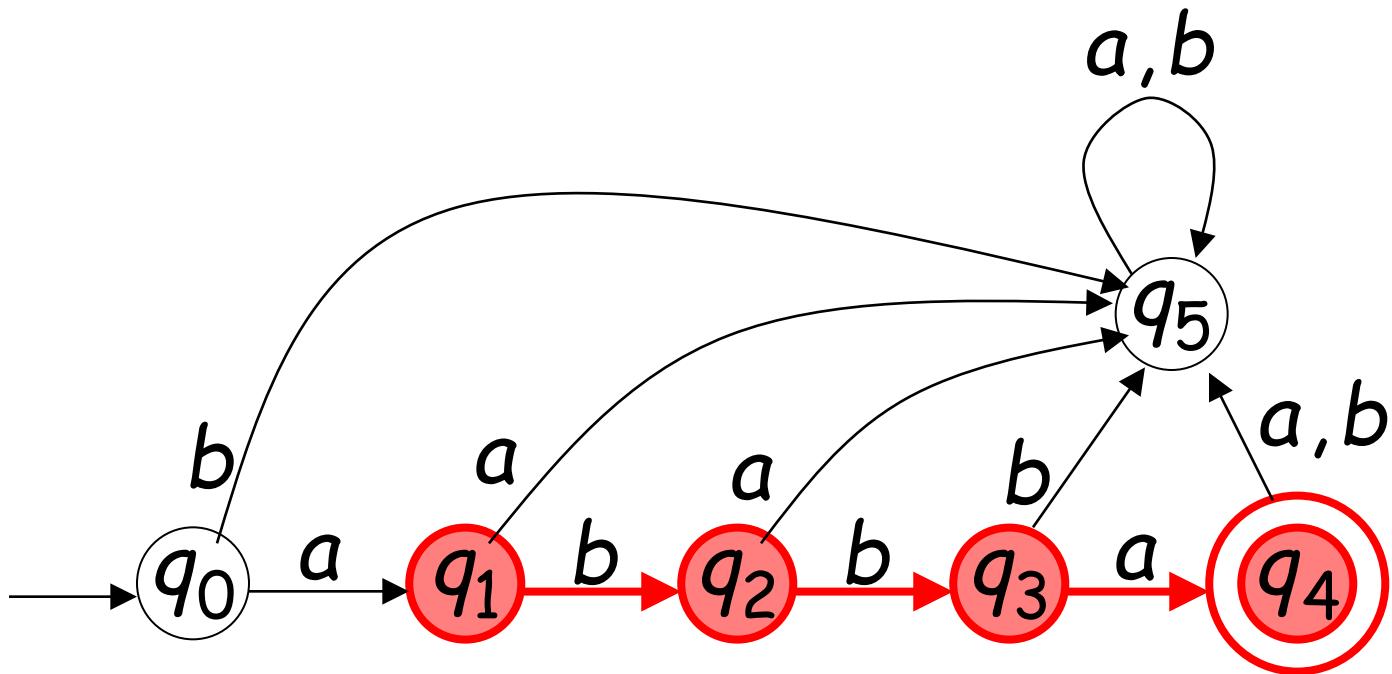
esempio: $\delta^*(q_0, ab) = q_2$



$$\delta^*(q_0, abbbbaa) = q_5$$



$$\delta^*(q_1, bba) = q_4$$



Caso speciale:

$$\begin{aligned}\Delta^*(q, aw) &= \Delta^*(\Delta(q, a), w); \\ \Delta^*(q, \lambda) &= q\end{aligned}$$

Per ogni stato q

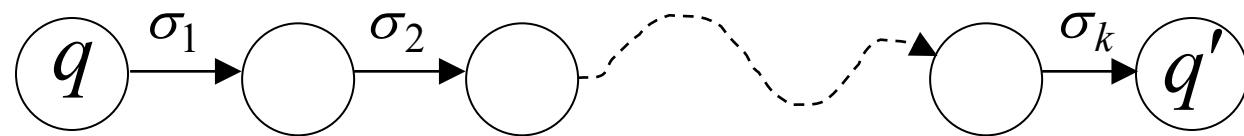
$$\delta^*(q, \lambda) = q$$

:

$$\delta^*(q, w) = q'$$

Implica che vi è un cammino di transizione

$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$

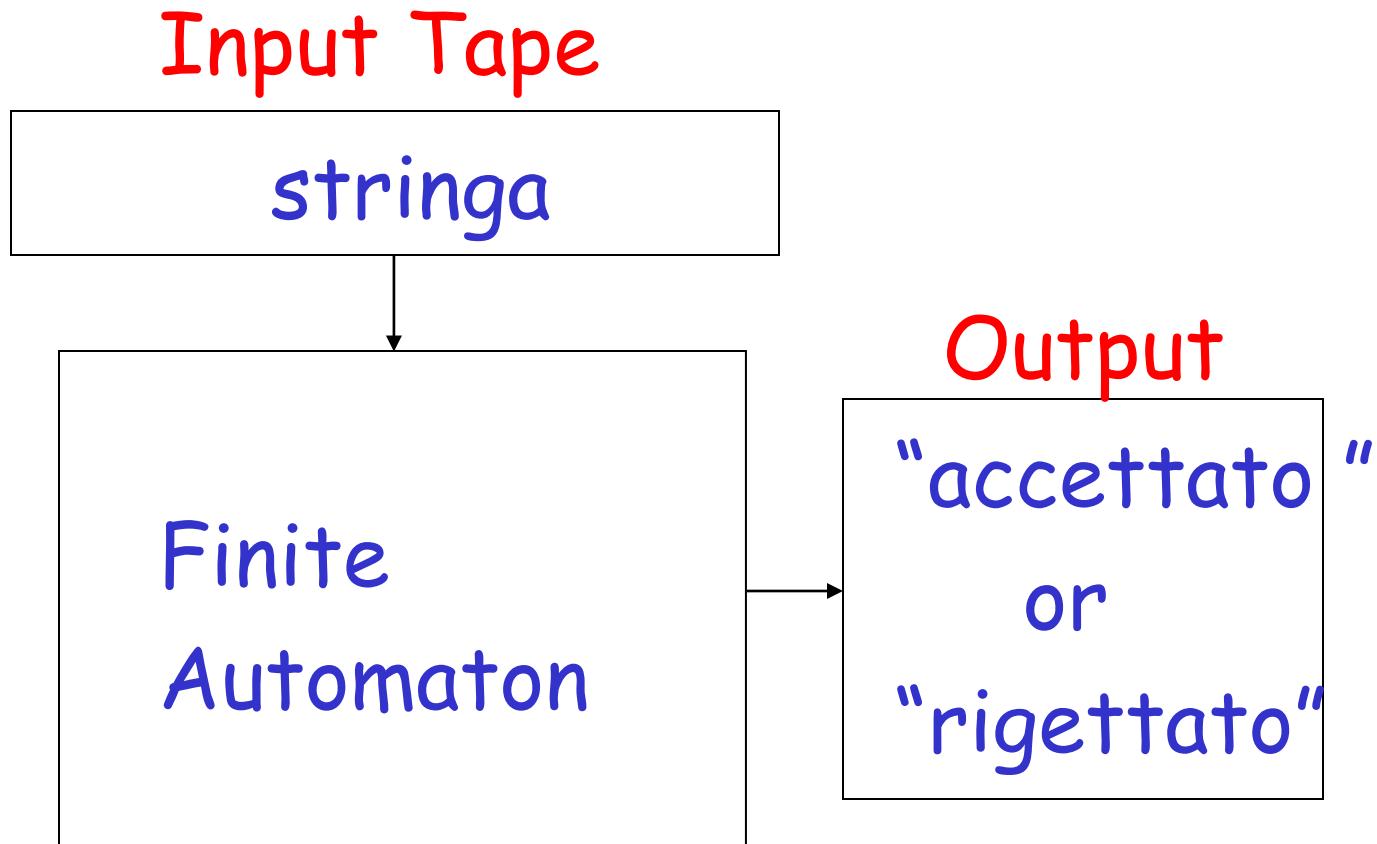


Alcuni stati possono essere ripetuti



Complessità costante sull'input

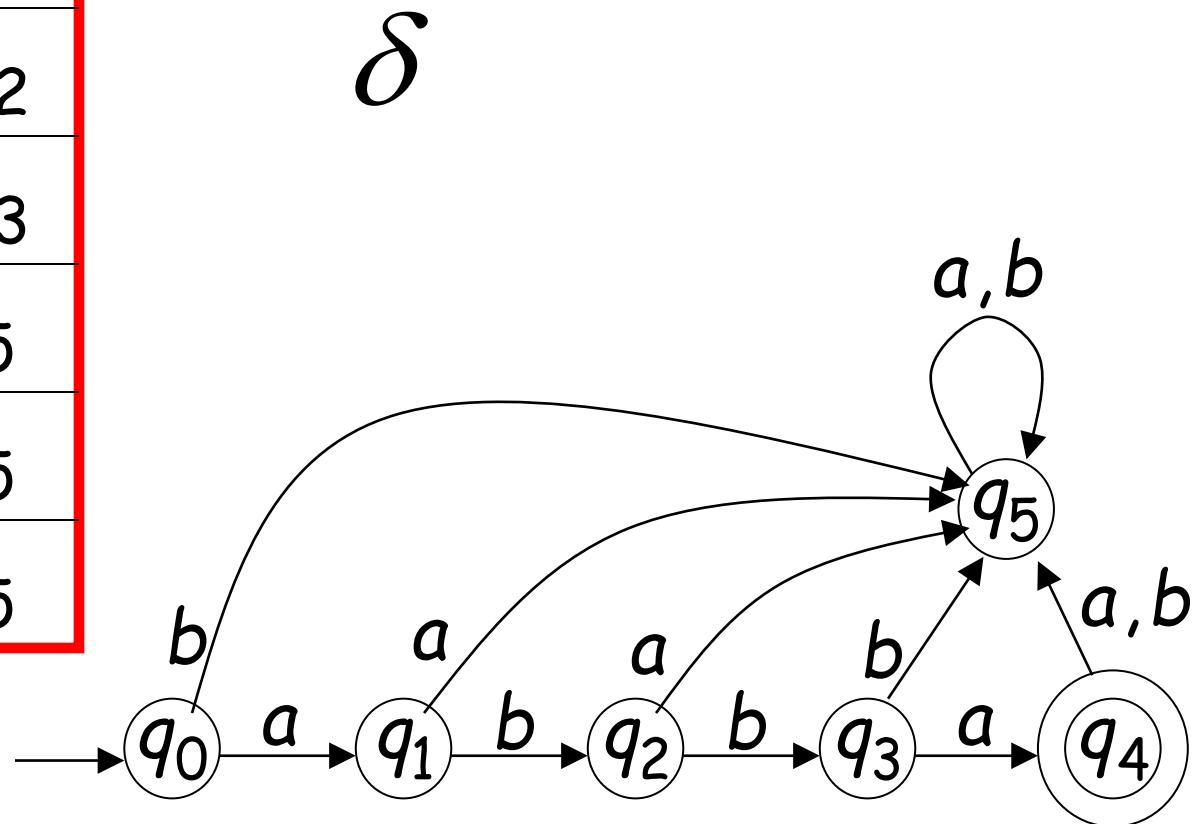
Deterministic Finite Automaton (DFA)



$$U(\text{automa}, \text{input}) = \text{automa}(\text{input})$$

symbols

δ	a	b
q_0	q_1	q_5
q_1	q_5	q_2
q_2	q_5	q_3
q_3	q_4	q_5
q_4	q_5	q_5
q_5	q_5	q_5



$U(\text{automa}, \text{input}) = \text{automa}(\text{input})$

Linguaggio accettato da un DFA

Linguaggio di un DFA: M

È denotato come $L(M)$

E contiene tutte le stringhe

Accettate da M

Un linguaggio L'

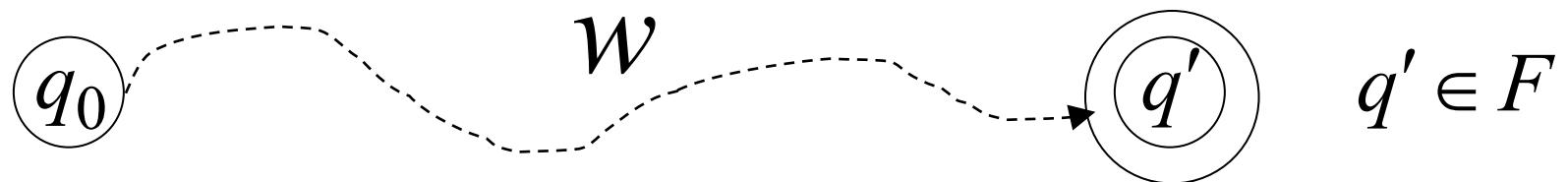
È accettato (o riconosciuto)

Da un DFA M se $L(M) = L'$

Per un DFA $M = (Q, \Sigma, \delta, q_0, F)$

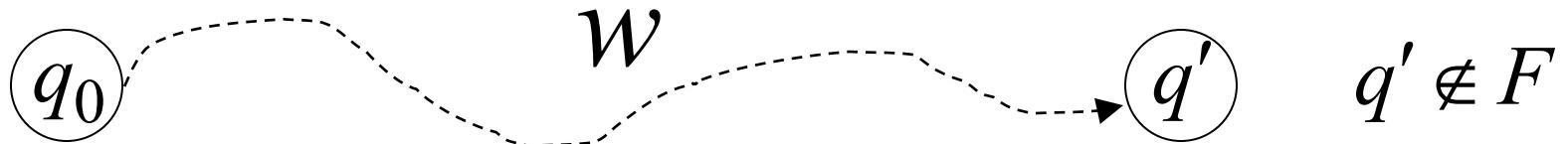
Il linguaggio accettato da M :

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



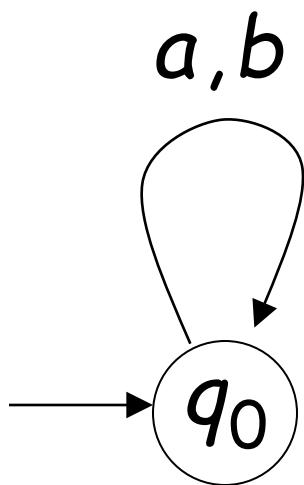
Linguaggio rifiutato da \mathcal{M} :

$$\overline{L(\mathcal{M})} = \{ w \in \Sigma^* : \delta^*(q_0, w) \notin F \}$$



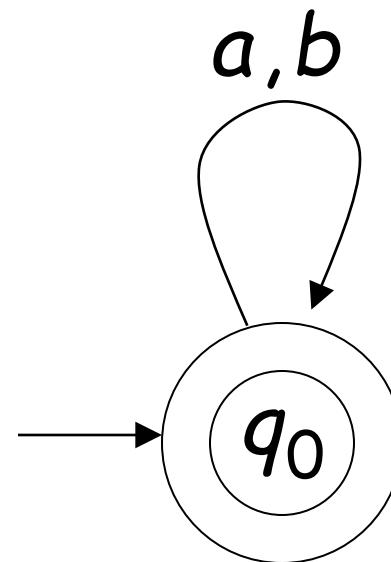
DFA esempi

$$\Sigma = \{a, b\}$$



$$L(M) = \{ \}$$

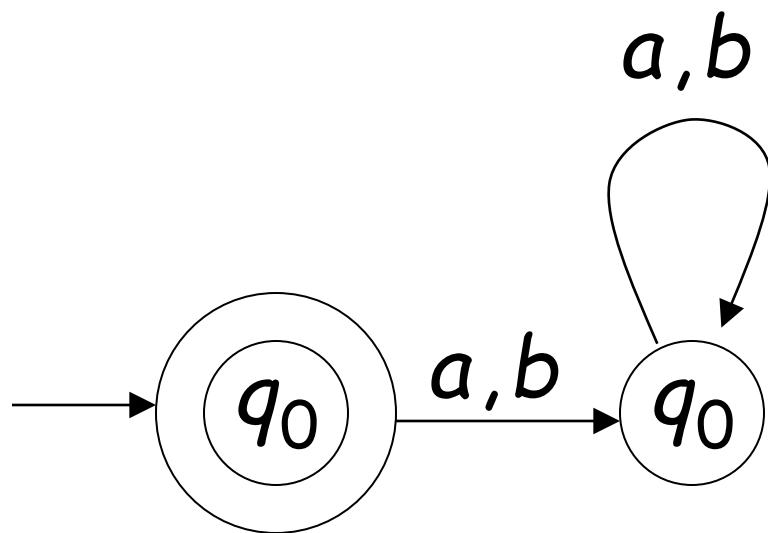
Linguaggio vuoto



$$L(M) = \Sigma^*$$

Tutte le stringhe

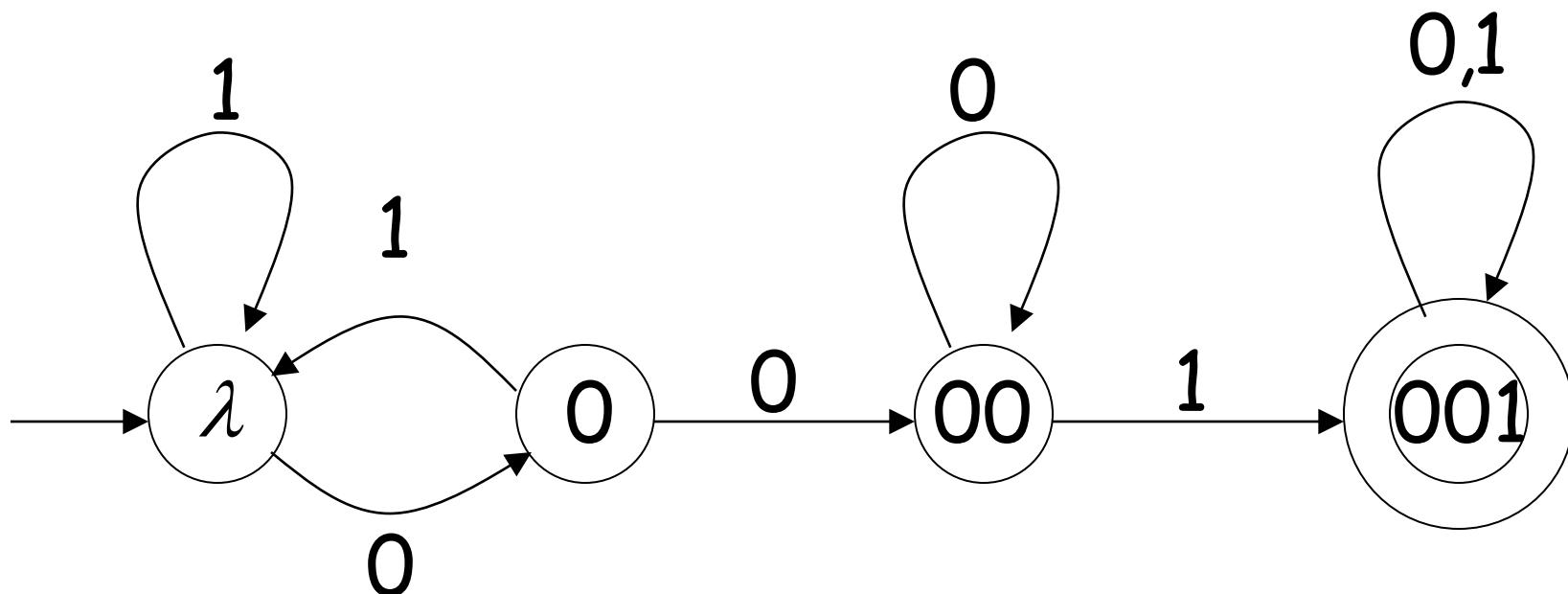
$$\Sigma = \{a, b\}$$



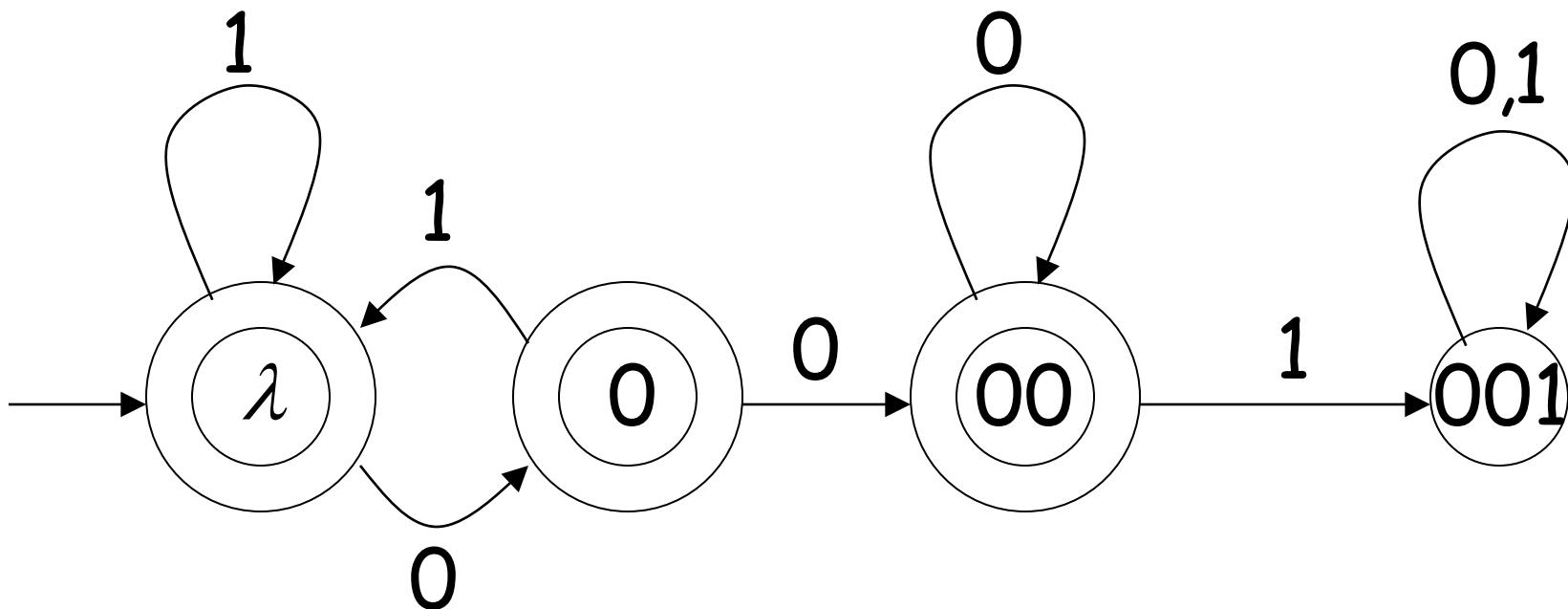
$$L(M) = \{\lambda\}$$

Linguaggio che riconosce le
Stringa vuota

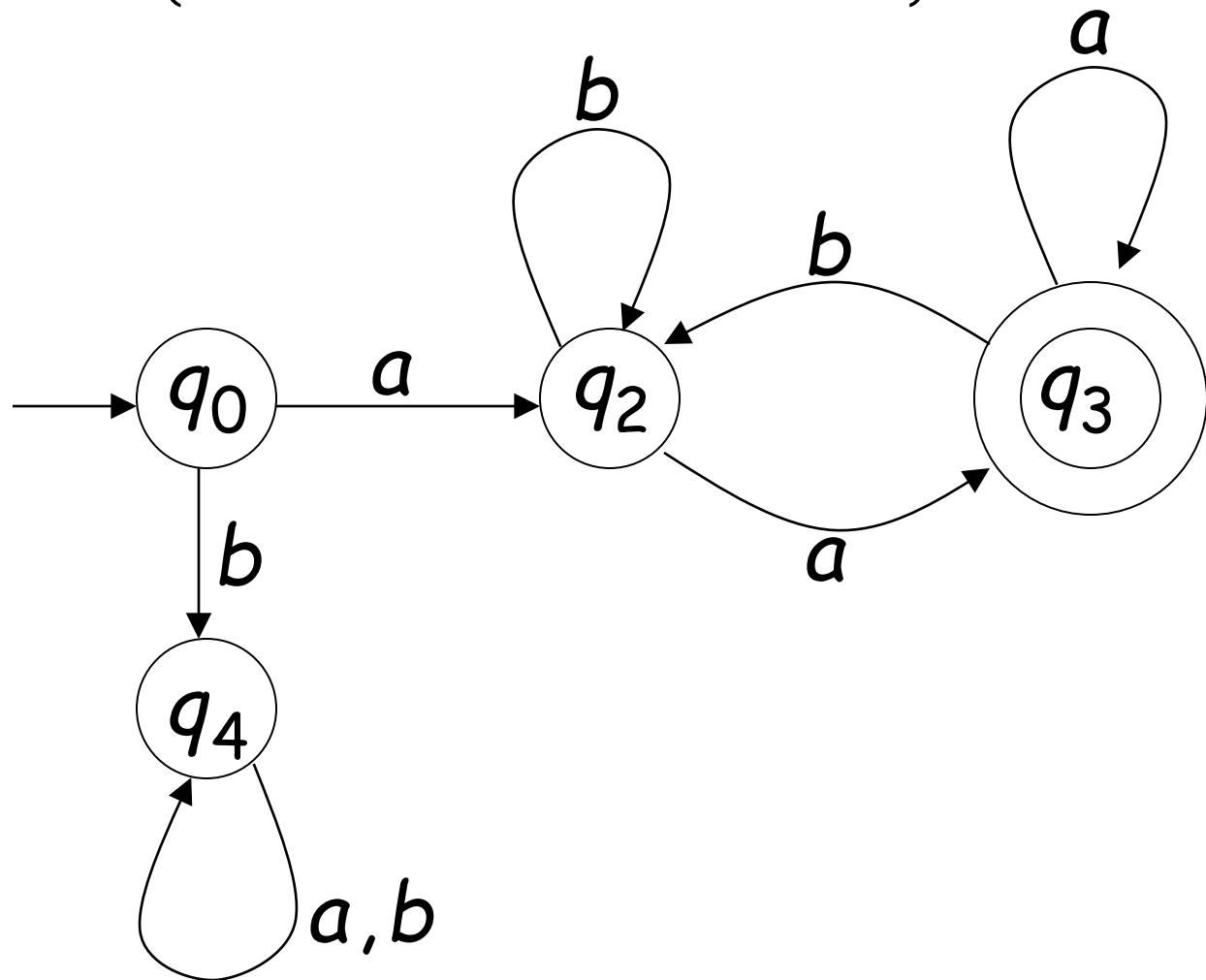
$L(M) = \{ \text{tutte le stringhe binarie}$
 che contengono
 $\text{la sottostringa } 001\}$



$L(M) = \{ \text{tutte le stringhe binarie che non Contengono } 001 \}$



$$L(M) = \{awba : w \in \{a,b\}^*\}$$



Linguaggi regolari

Definizione:

Un linguaggio L è regolare se esiste un DFA M che lo accetta ($L(M) = L$)

I linguaggi accettati da tutti i DFA formano la famiglia dei linguaggi regolari

Esempi di linguaggi regolari:

$\{abba\}$ $\{\lambda, ab, abba\}$

$\{a^n b : n \geq 0\}$ $\{awa : w \in \{a,b\}^*\}$

{ tutte stringhe $\{a,b\}^*$ con prefisso ab }

{ all binary strings without substring 001}

{ $x : x \in \{1\}^*$ and x is even}

{ } $\{\lambda\}$ $\{a,b\}^*$

Abbiamo visto in precedenza gli automi regolari che li definiscono

Esistono linguaggi che non sono regolari:

$$L = \{a^n b^n : n \geq 0\}$$

$$\text{ADDITION} = \{x + y = z : x = 1^n, y = 1^m, z = 1^k, n + m = k\}$$

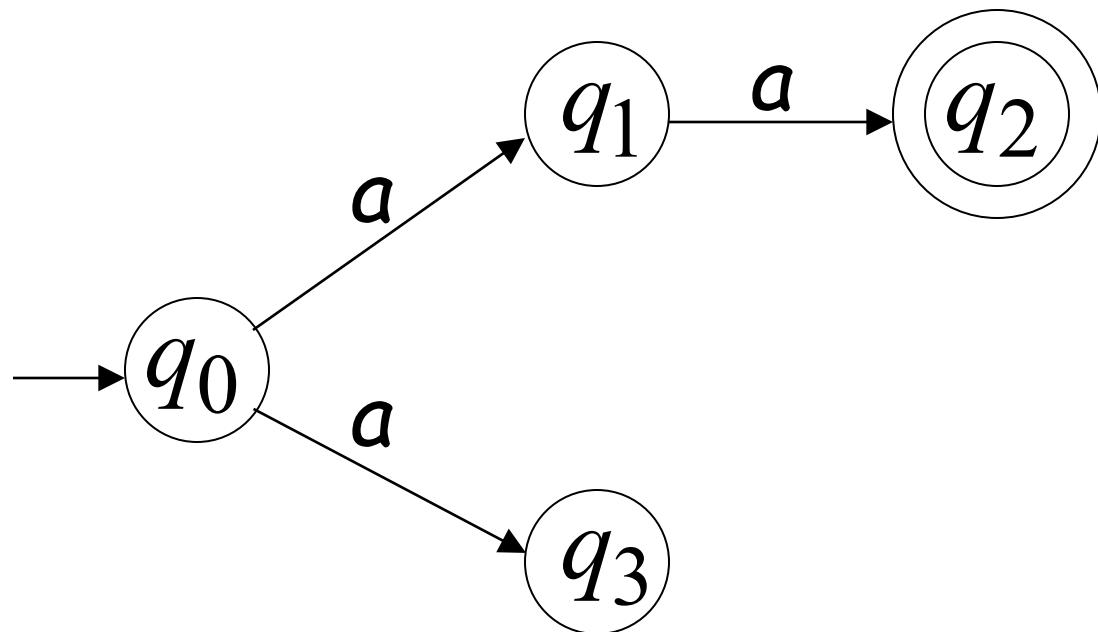
Non esiste nessun DFA che accetta
Questo linguaggio (vedremo più avanti)

Non-Deterministic Finite Automata

Automa finito deterministico
calcolo finito e deterministico
sequenziale, un segmento di Ing
dell'input

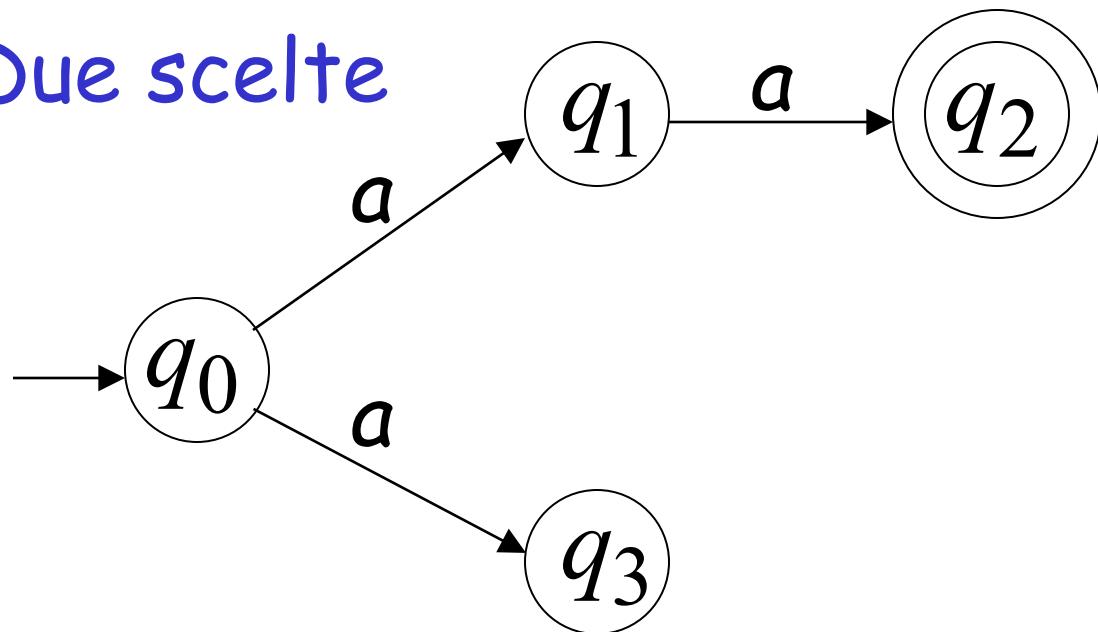
Automi non deterministici (NFA)

alfabeto = $\{a\}$

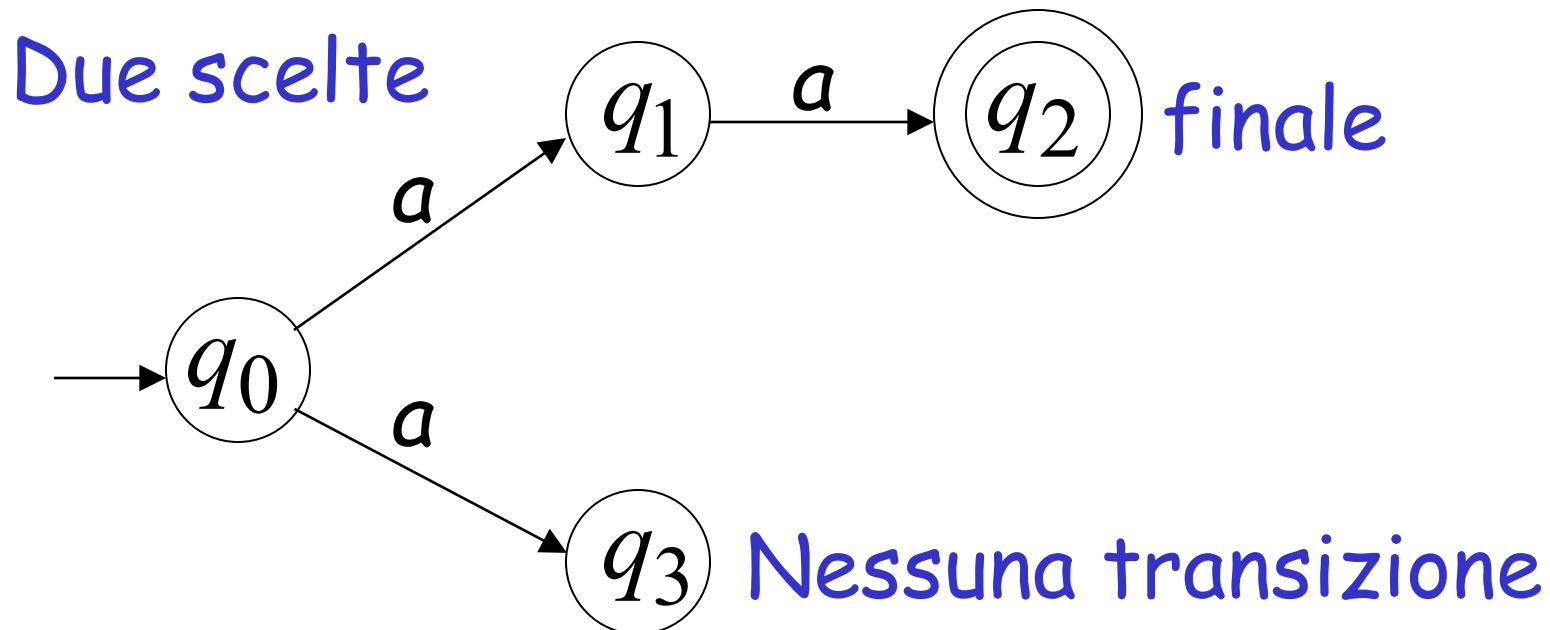


alfabeto = $\{a\}$

Due scelte



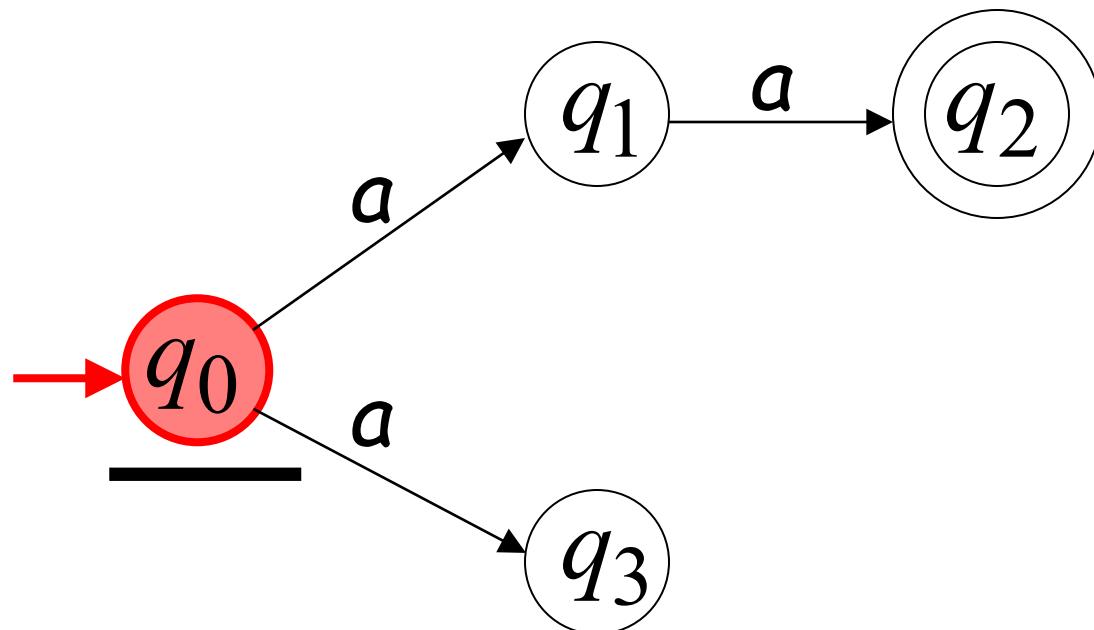
alfabeto = $\{a\}$



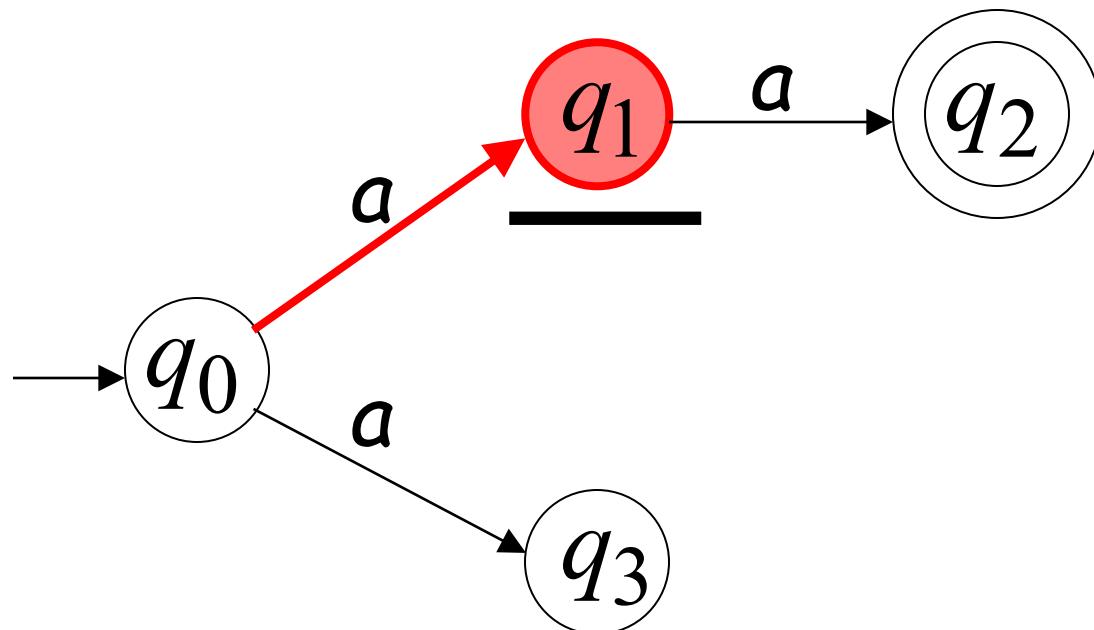
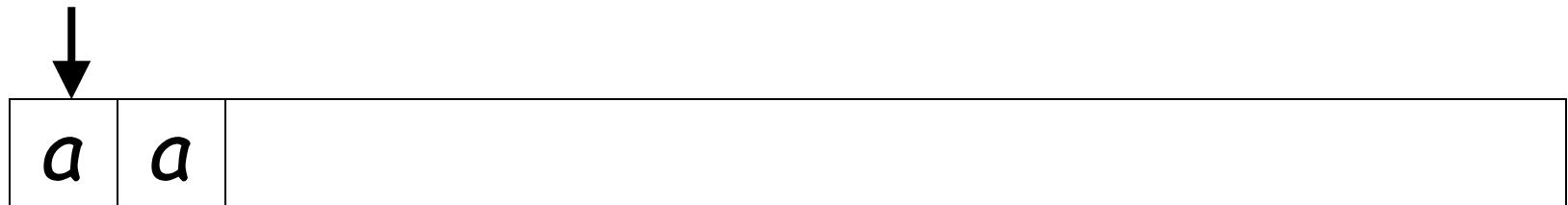
Prima delle due scelte



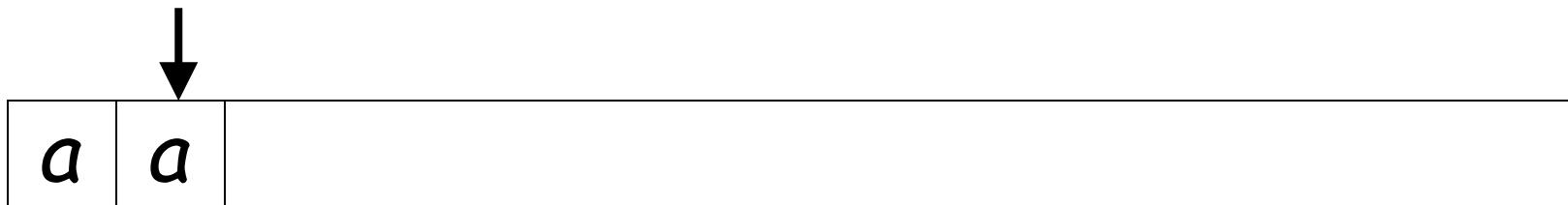
a	a	
-----	-----	--



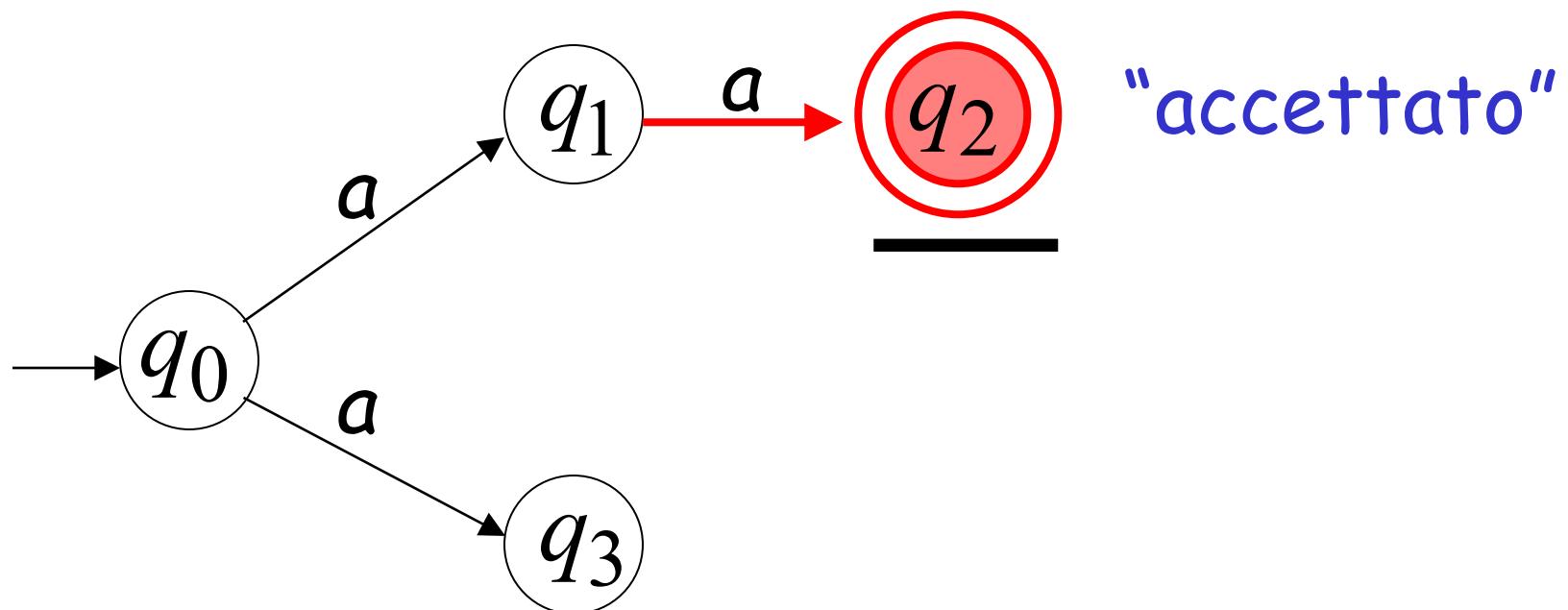
Prima scelta



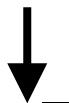
Prima scelta



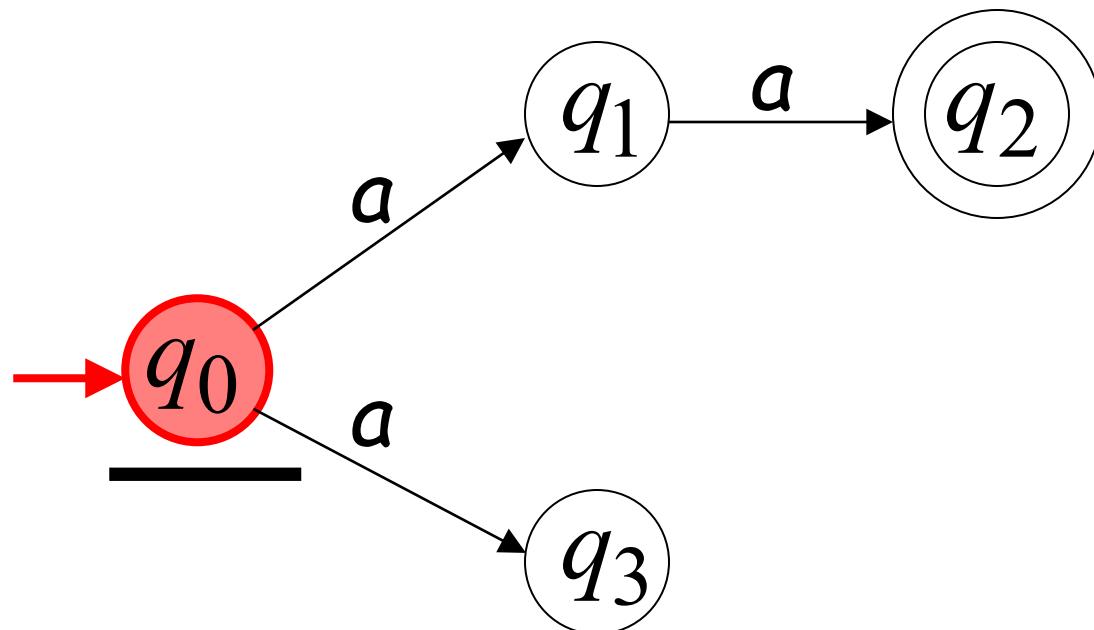
Abbiamo consumato tutto l'input



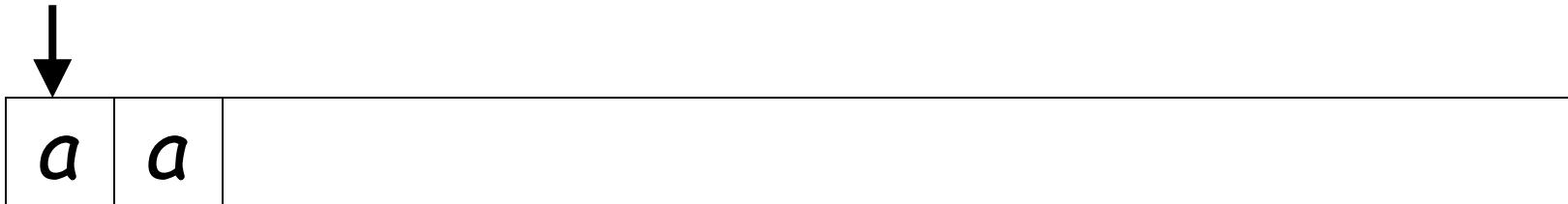
Seconda scelta



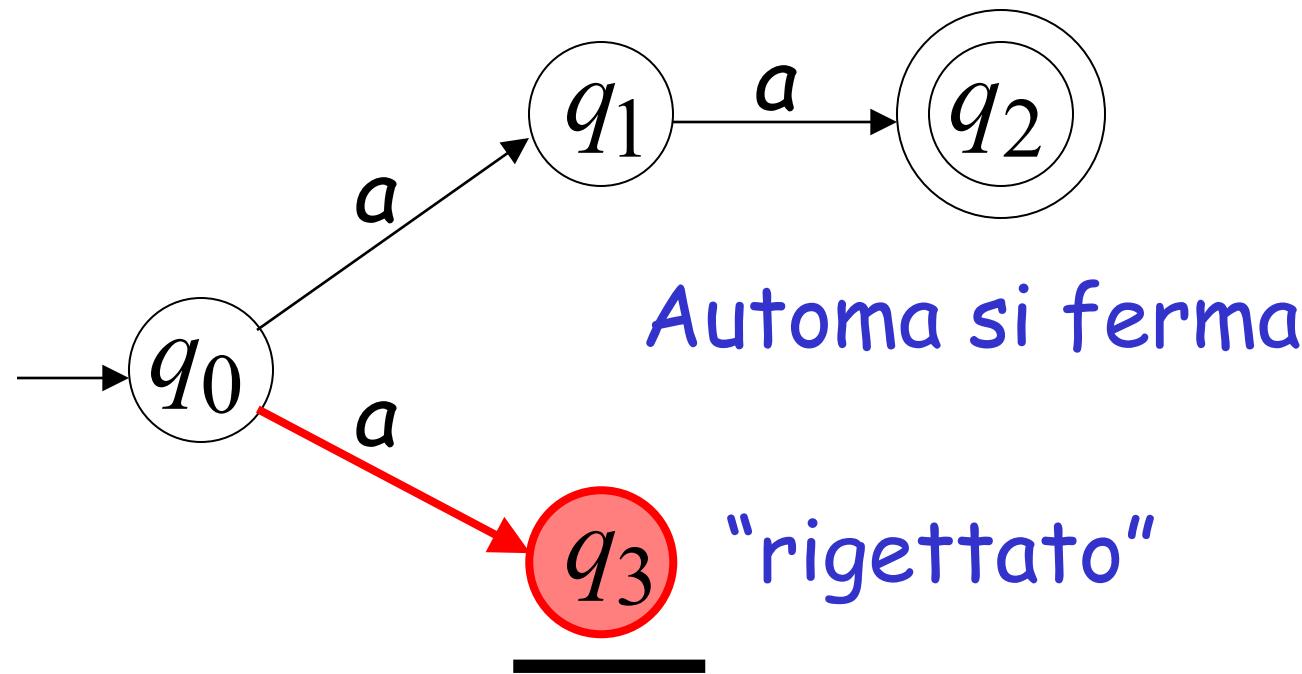
a	a	
-----	-----	--



Seconda scelta



Input non può essere tutto usato

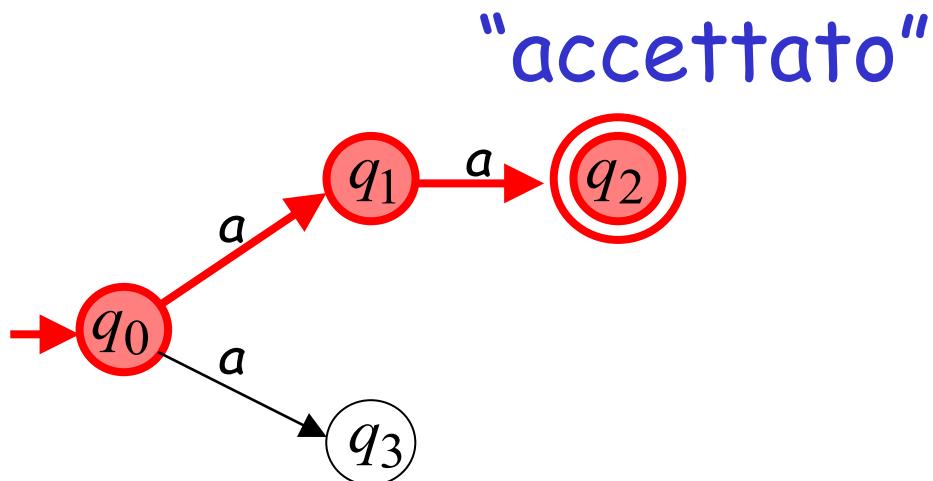


un NFA accetta una stringa:

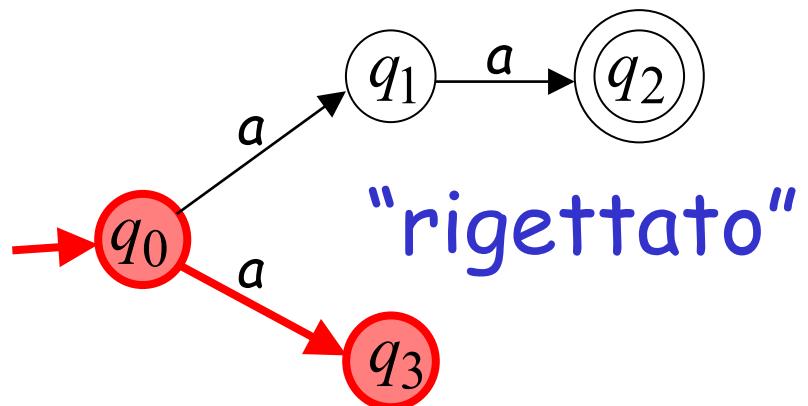
Se esiste una computazione che accetta
la stringa

Tutta la stringa di input è stata letta e l'automa
Si trova in uno stato finale

aa È accettato dal NFA:



Perchè la
Computazione
accetta *aa*

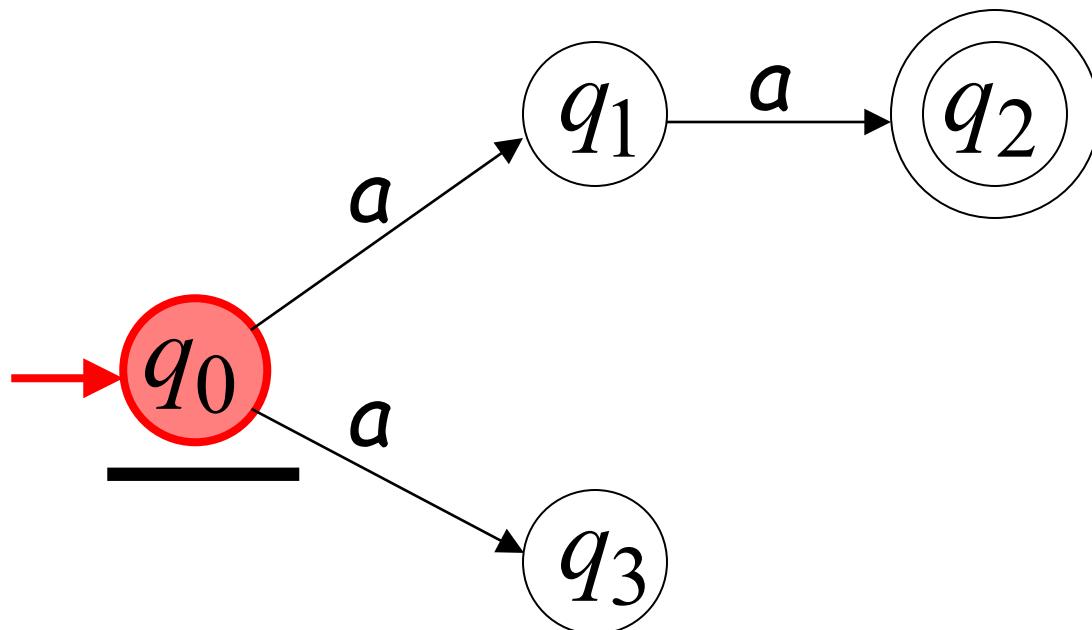


Questa computazione
è ignorata

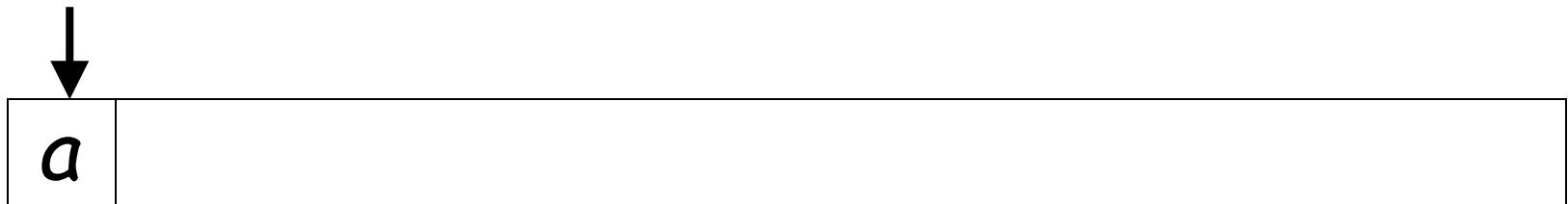
Esempio computazione che rigettà



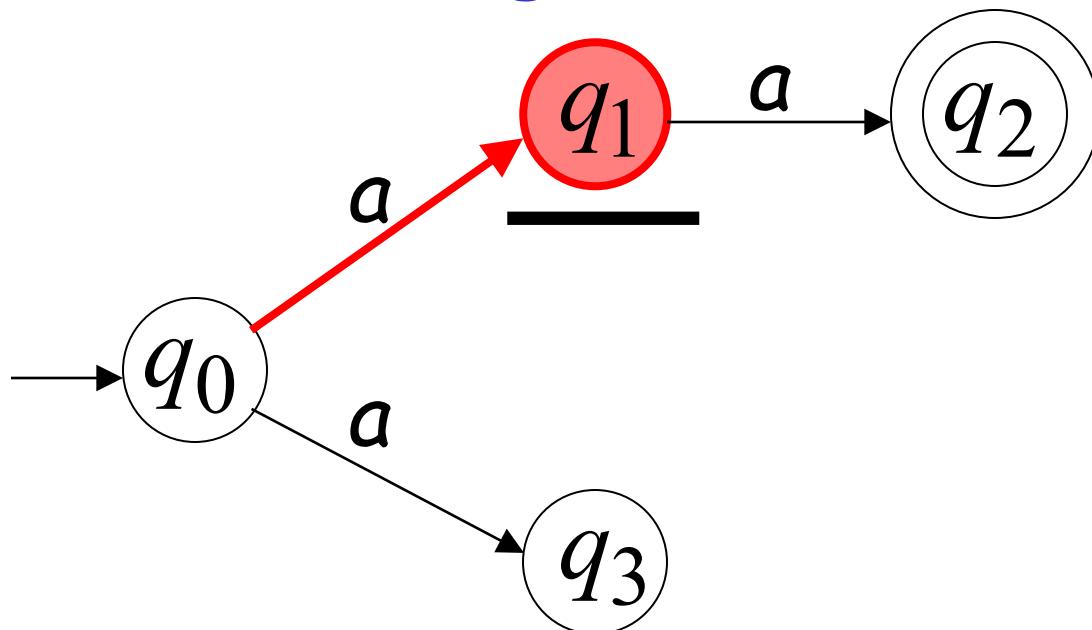
a	
-----	--



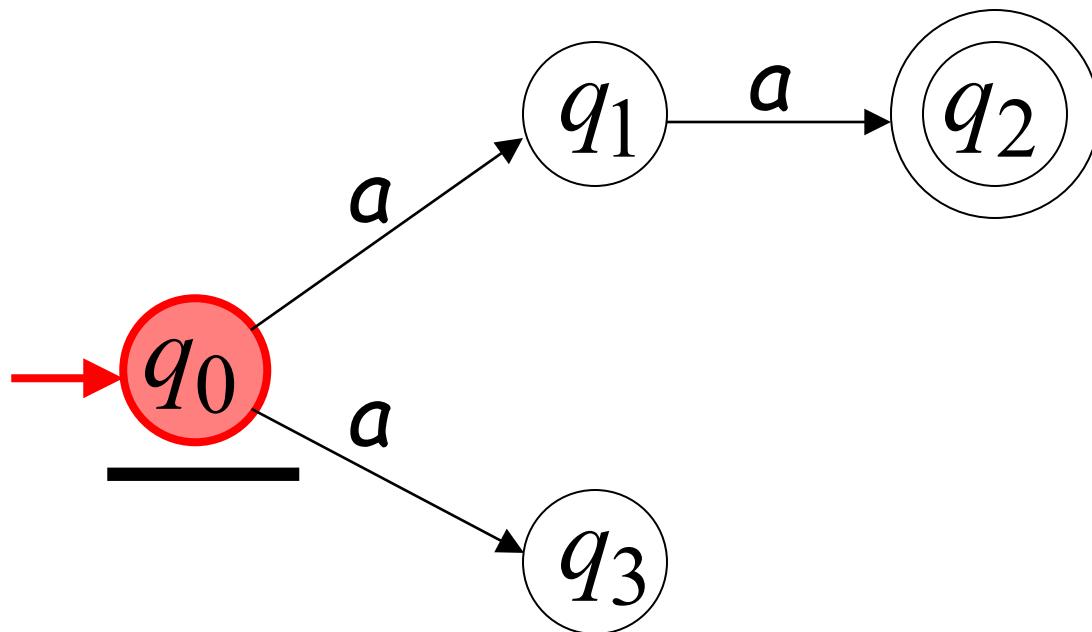
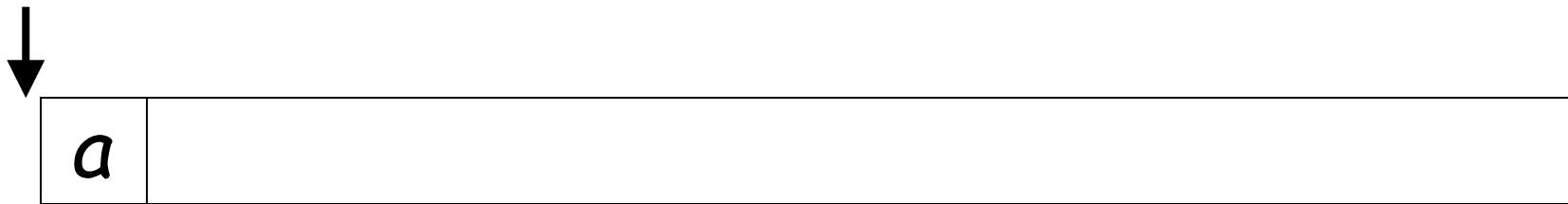
Prima scelta



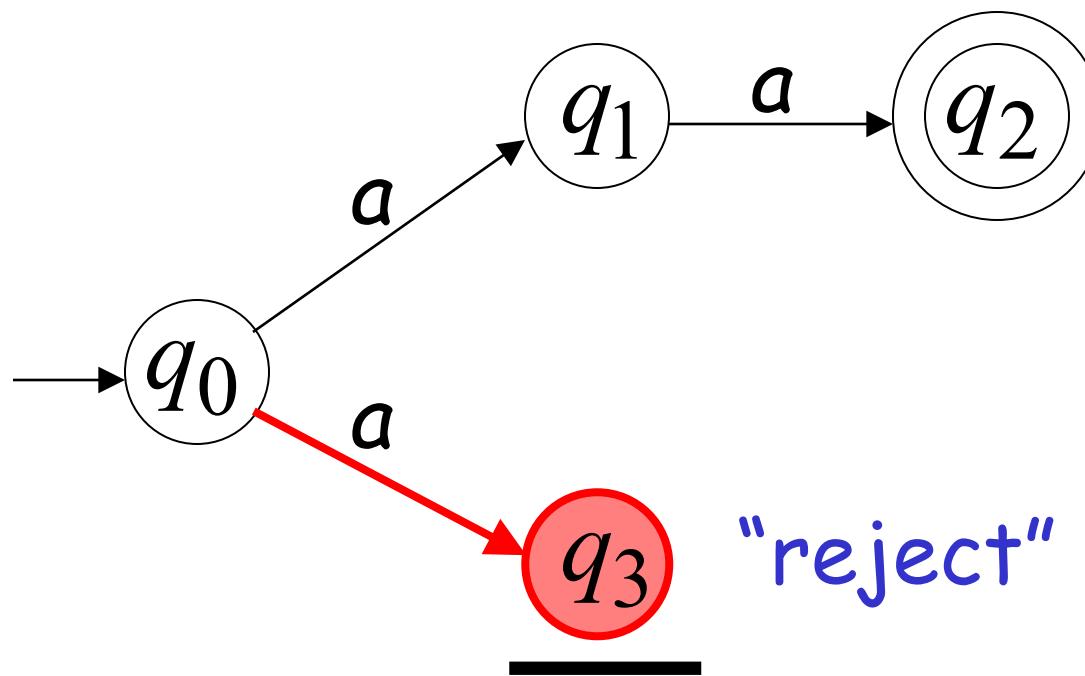
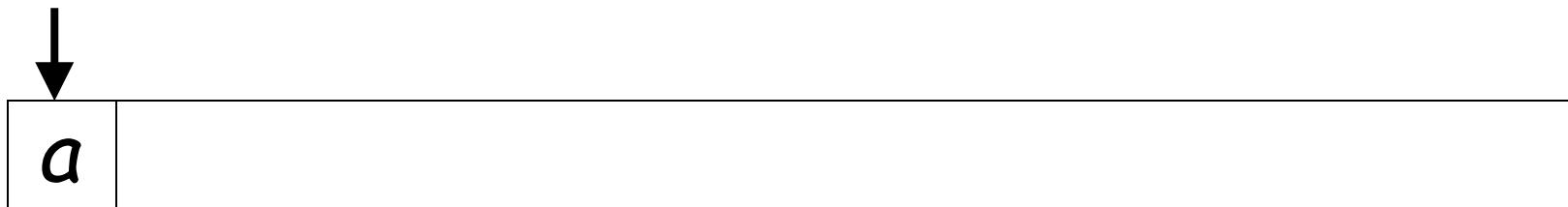
“rigettato”



Seconda scelta



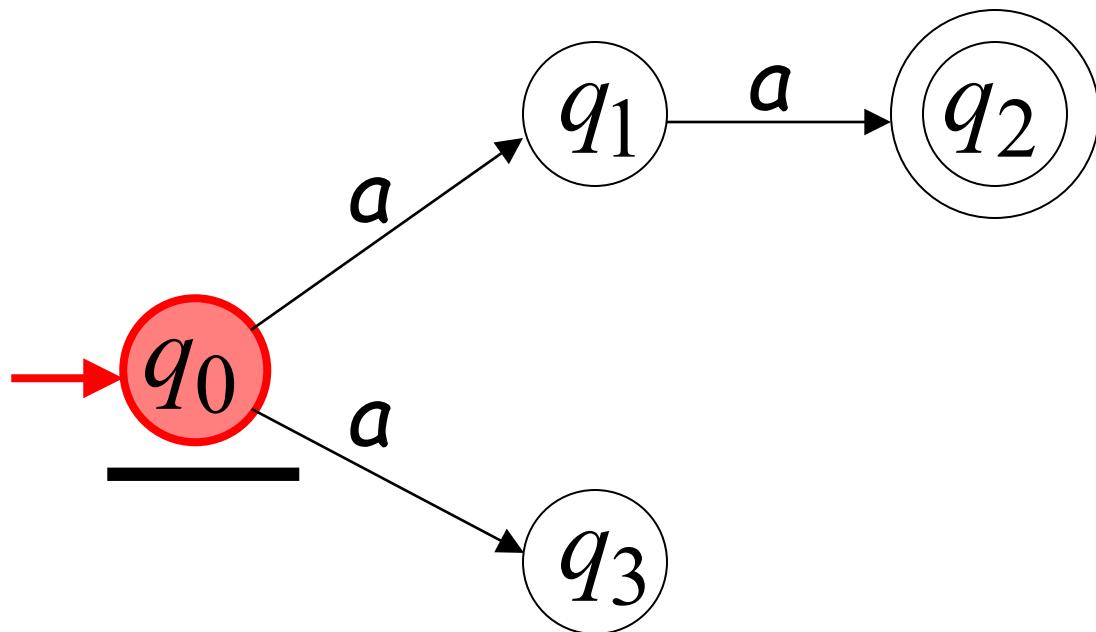
Seconda scelta



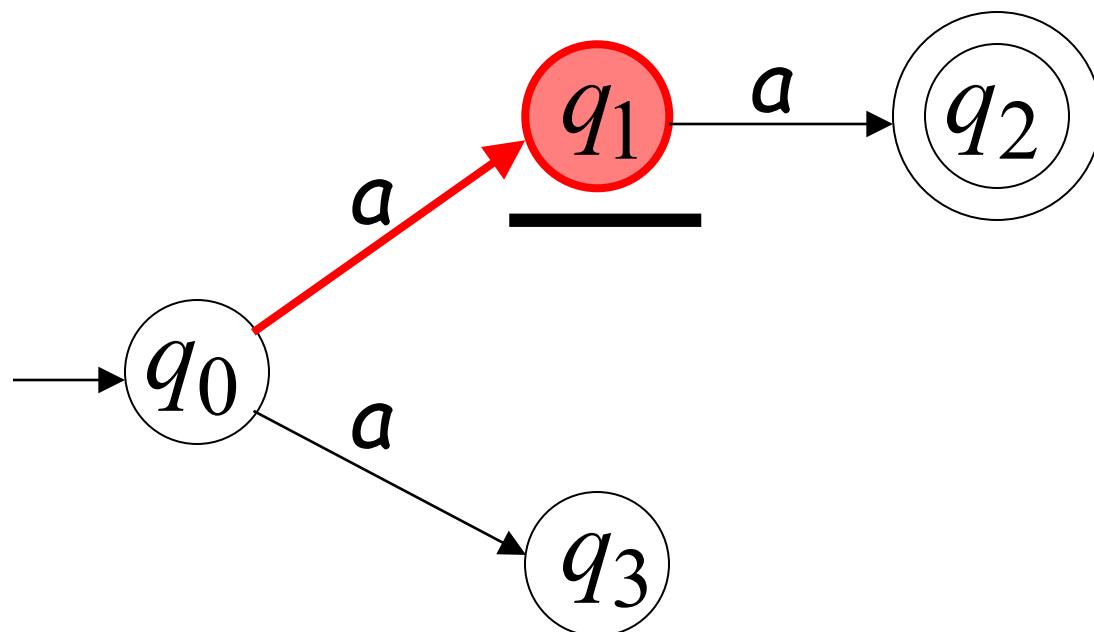
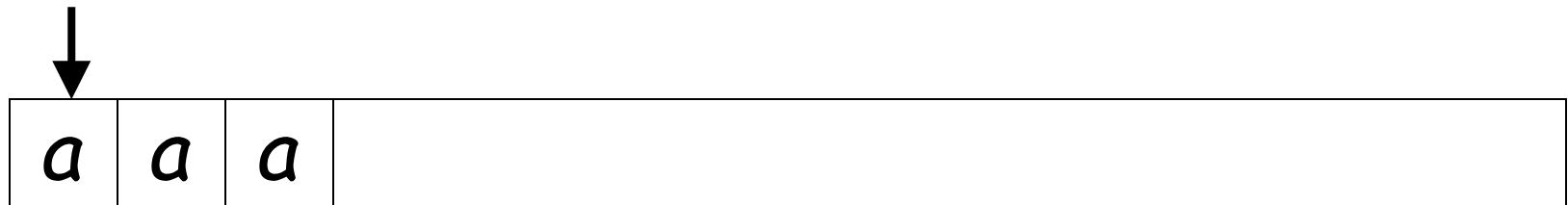
Un altro esempio



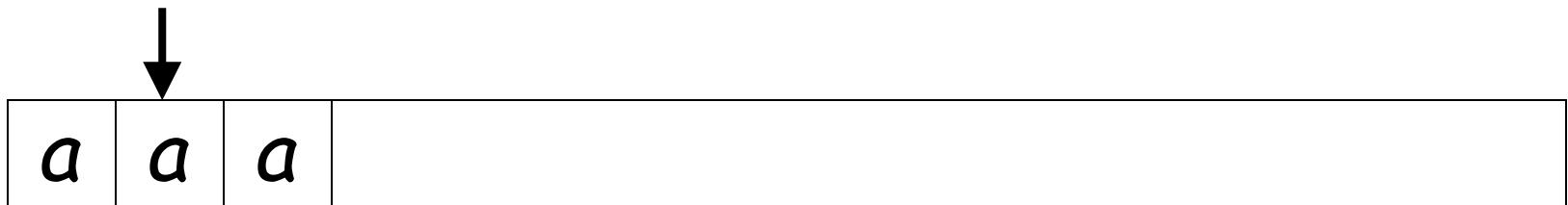
a	a	a	
-----	-----	-----	--



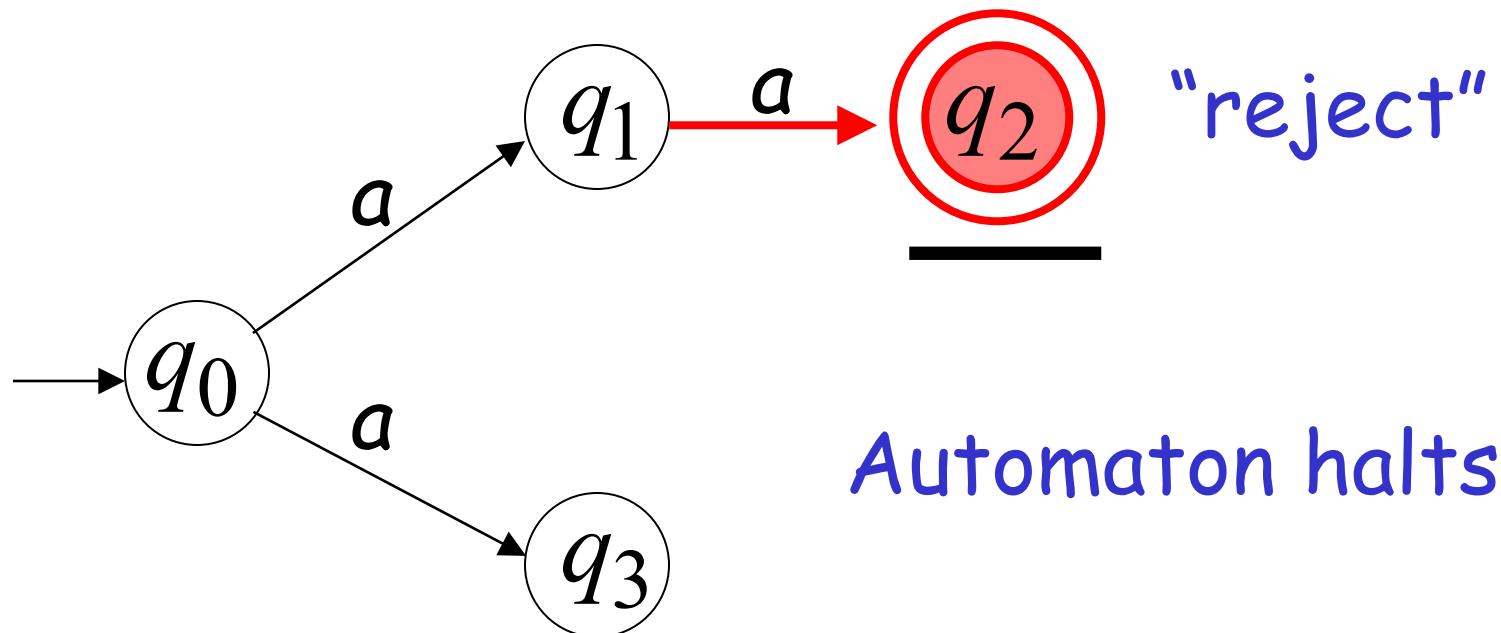
Prima scelta



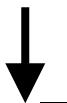
First Choice



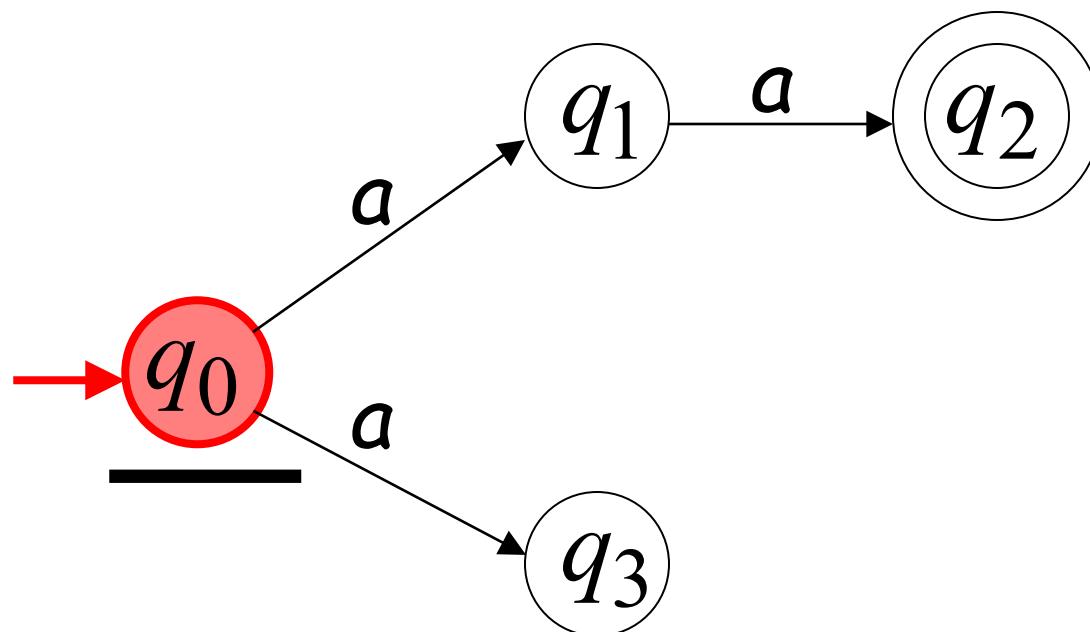
Input cannot be consumed



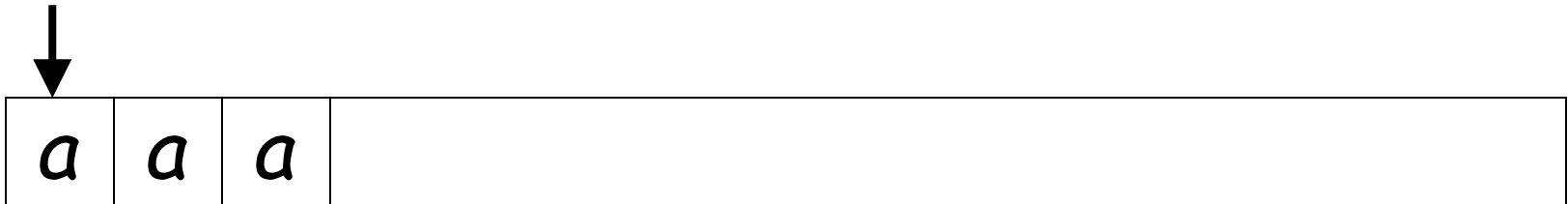
Second Choice



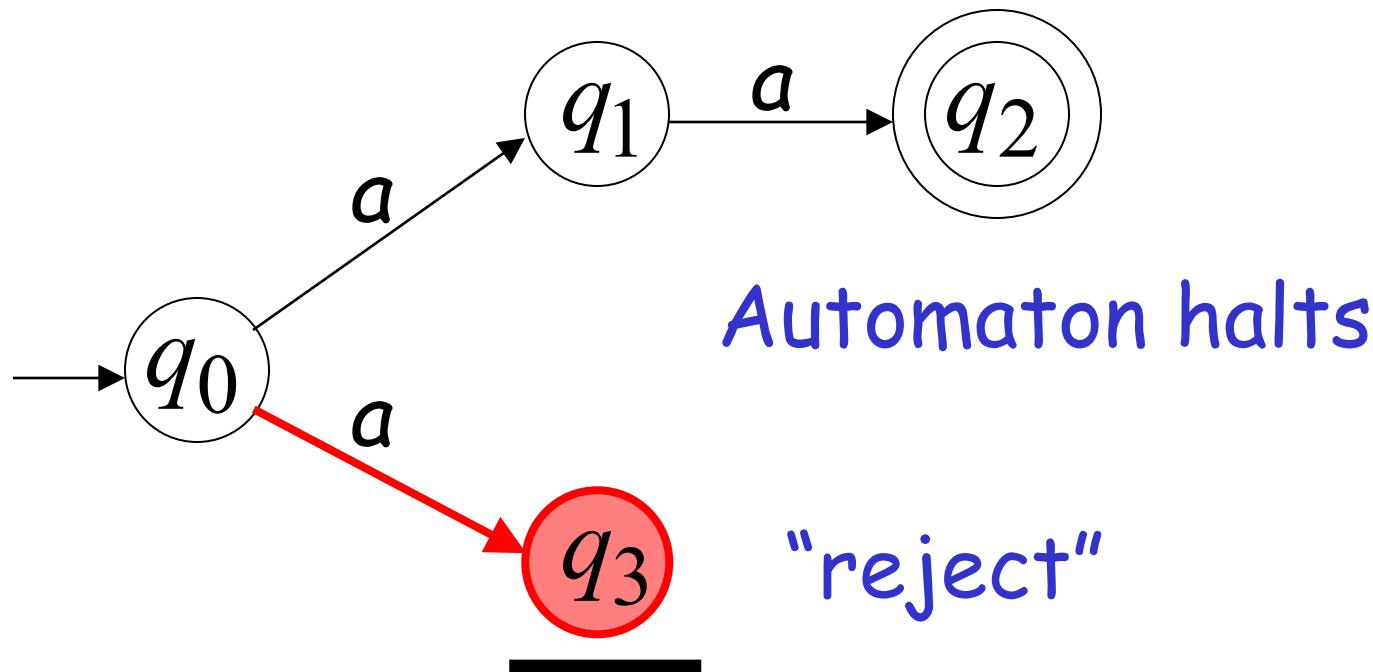
a	a	a	
-----	-----	-----	--



Second Choice



Input non viene tutto consumato



An NFA rejects a string:

Se non vi è una computazione del NFA
che accetta la stringa.

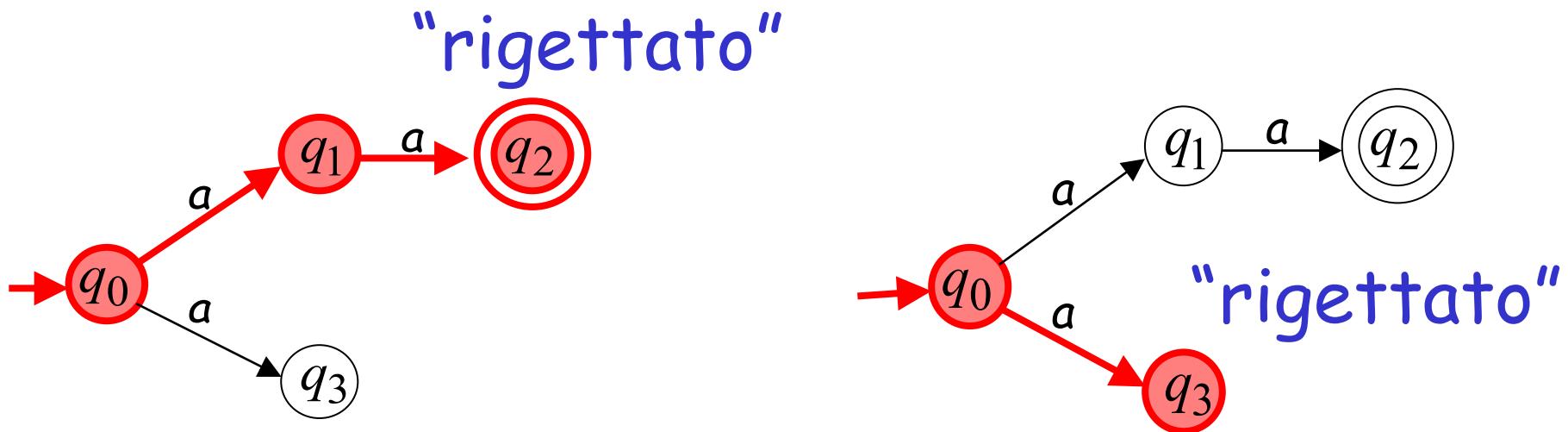
Per ogni computazione:

- tutto l'input è consumato e l'automa
- non ha raggiunto uno stato finale

O

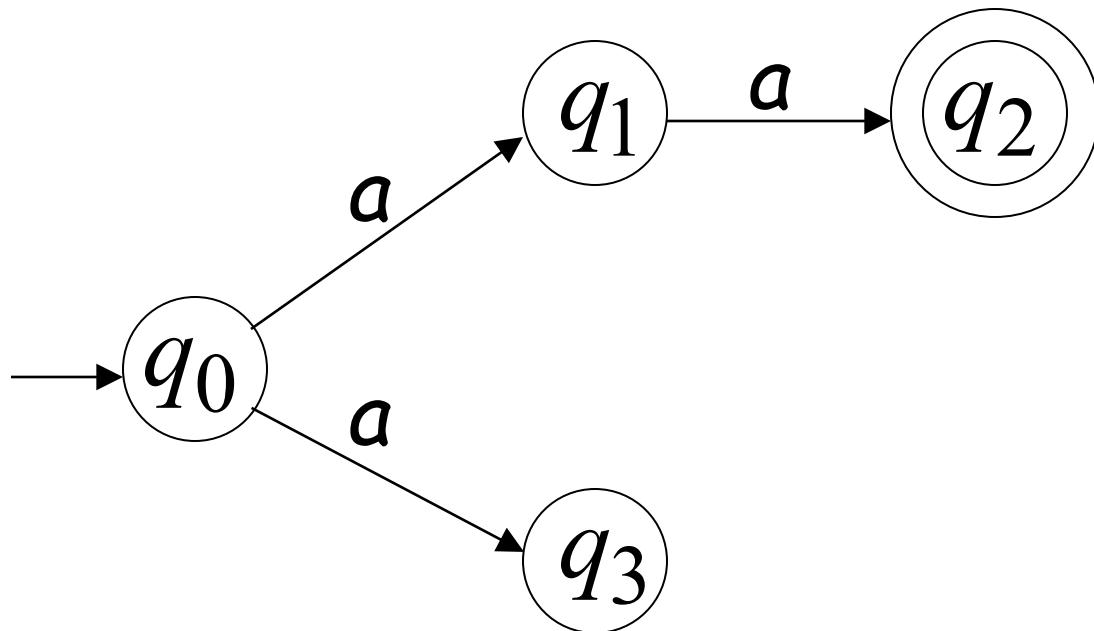
- L'input non è stato tutto consumato

aaa È rigettato dal NFA:

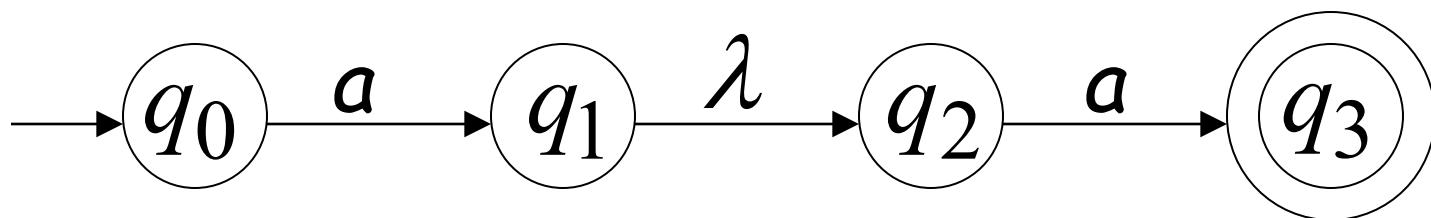


Tutte le possibili computazioni
non raggiungono uno stato finale

Linguaggio accettato: $L = \{aa\}$

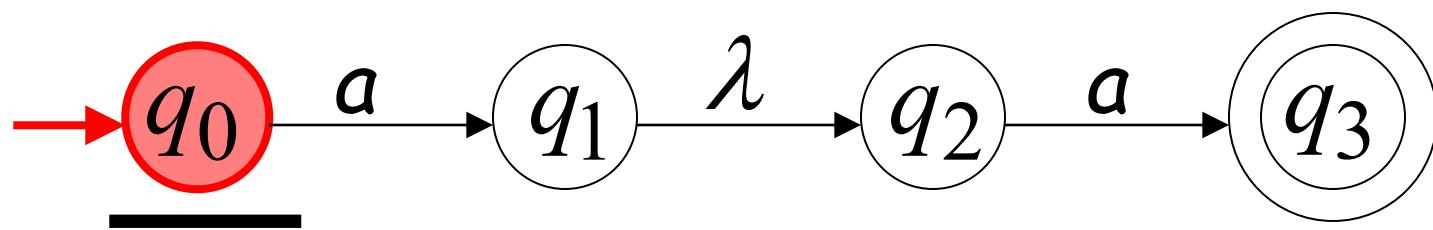


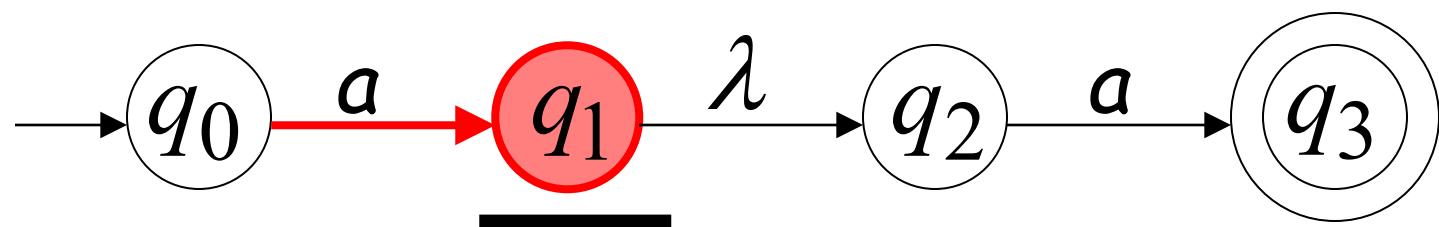
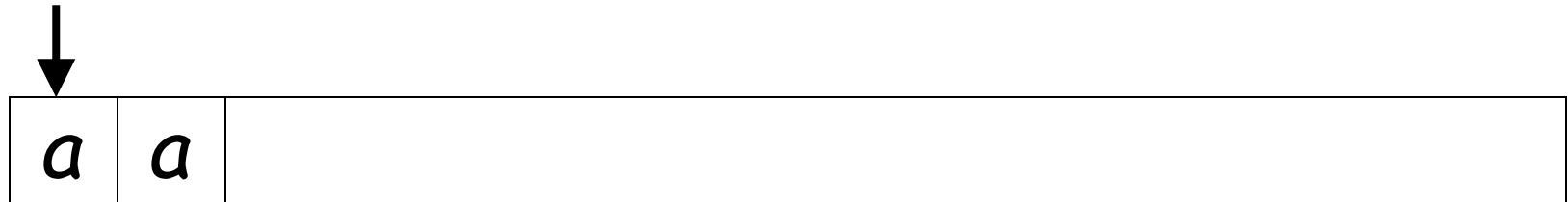
Lambda transizione



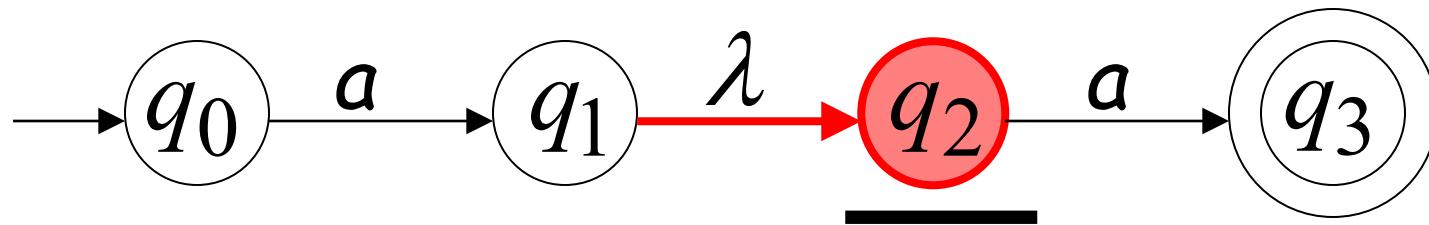
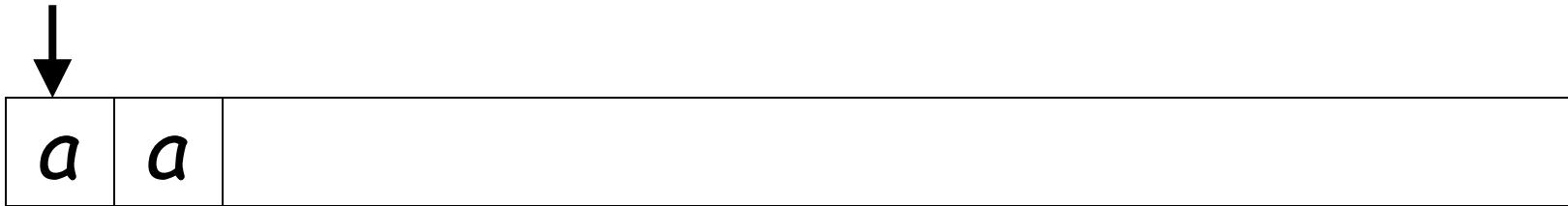


a	a	
-----	-----	--





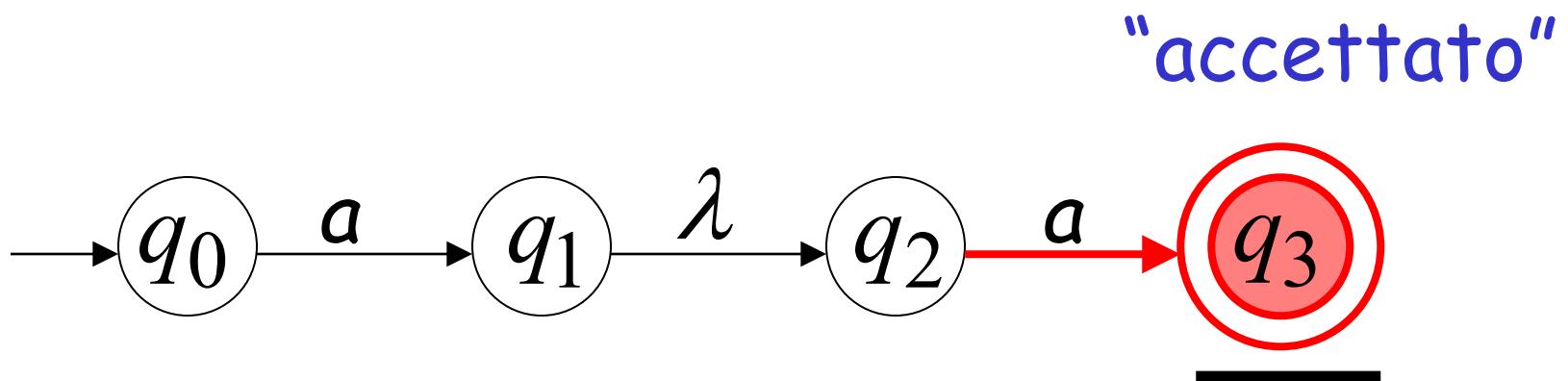
La testina dell'input non si muove



Tutto l'input è esaminato



a	a	
-----	-----	--

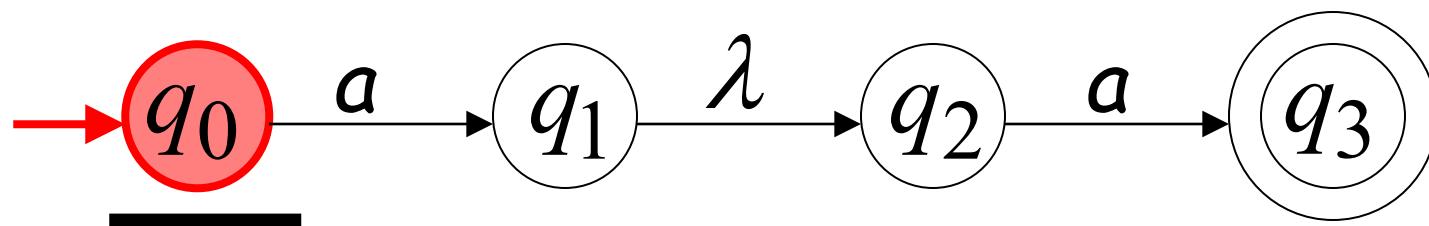


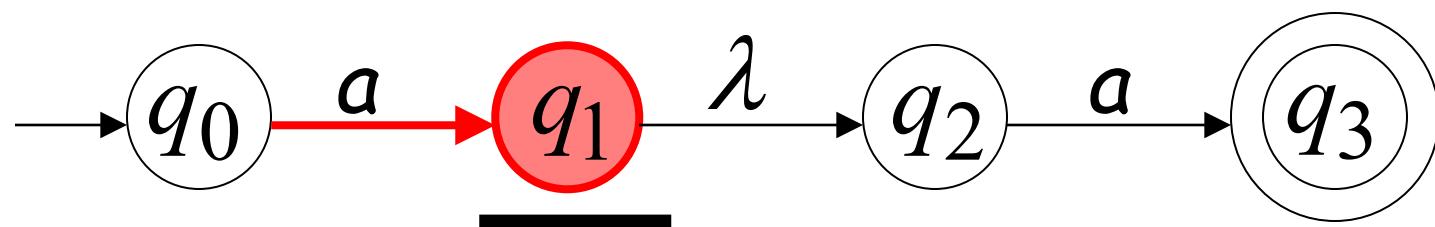
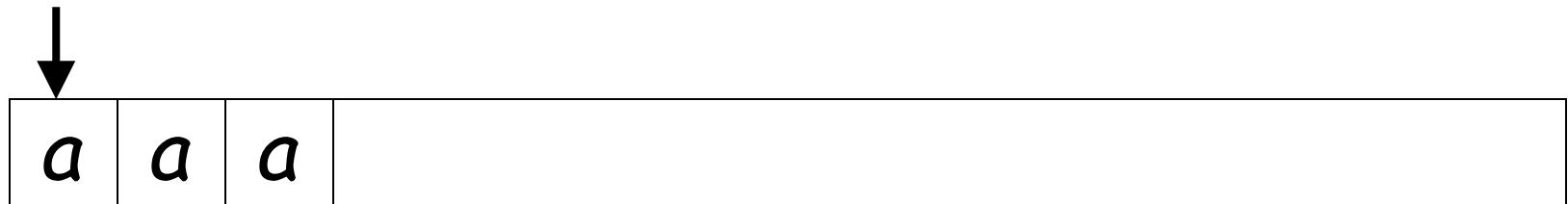
stringa aa è accettata

Esempio di non accettazione (rigettato)

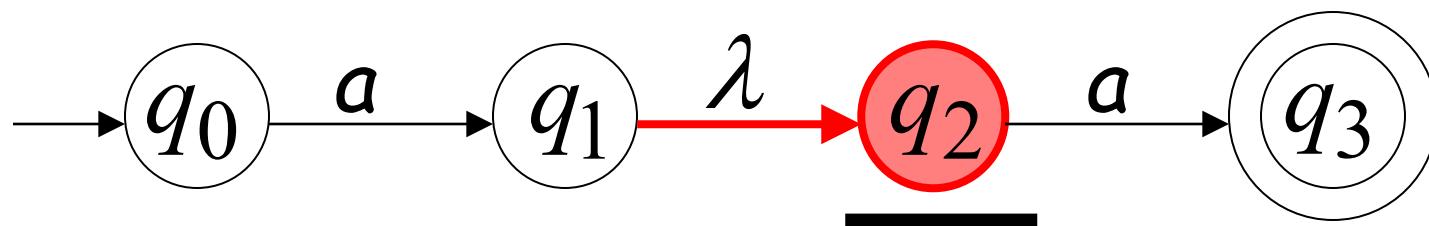
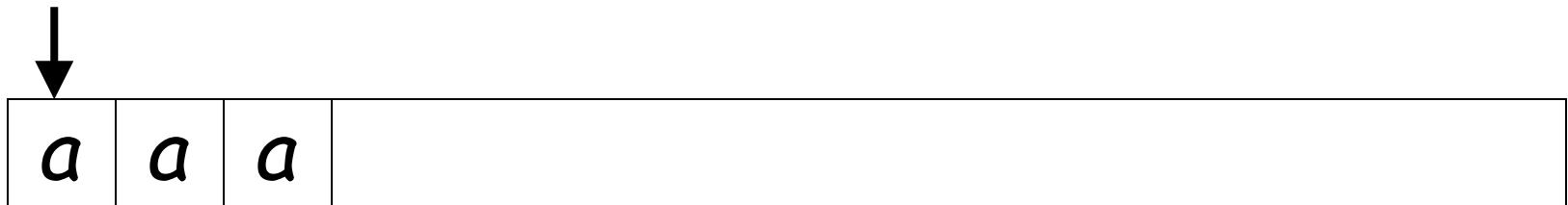


a	a	a	
-----	-----	-----	--





(la testina non si muove)



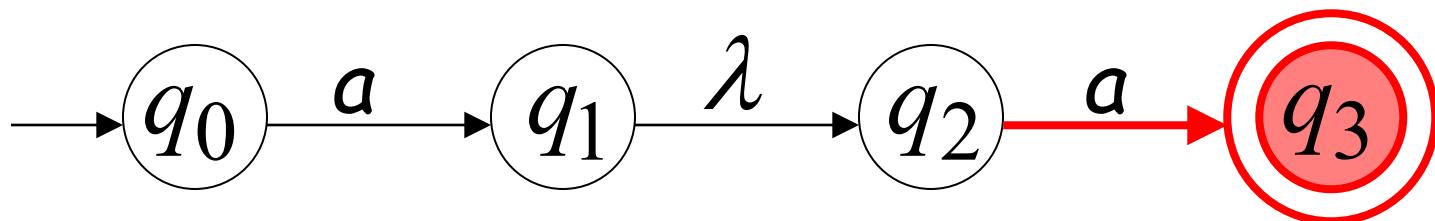
Input non viene analizzato tutto



a	a	a	
---	---	---	--

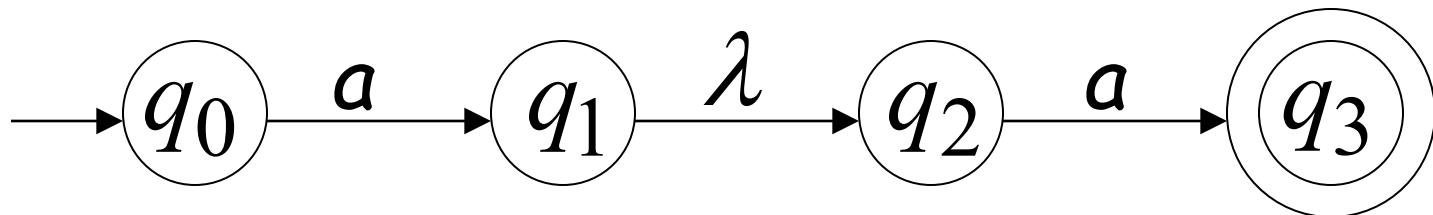
Automa si ferma

“rigettato”



stringa aaa è rigettata

Linguaggio accettato: $L = \{aa\}$

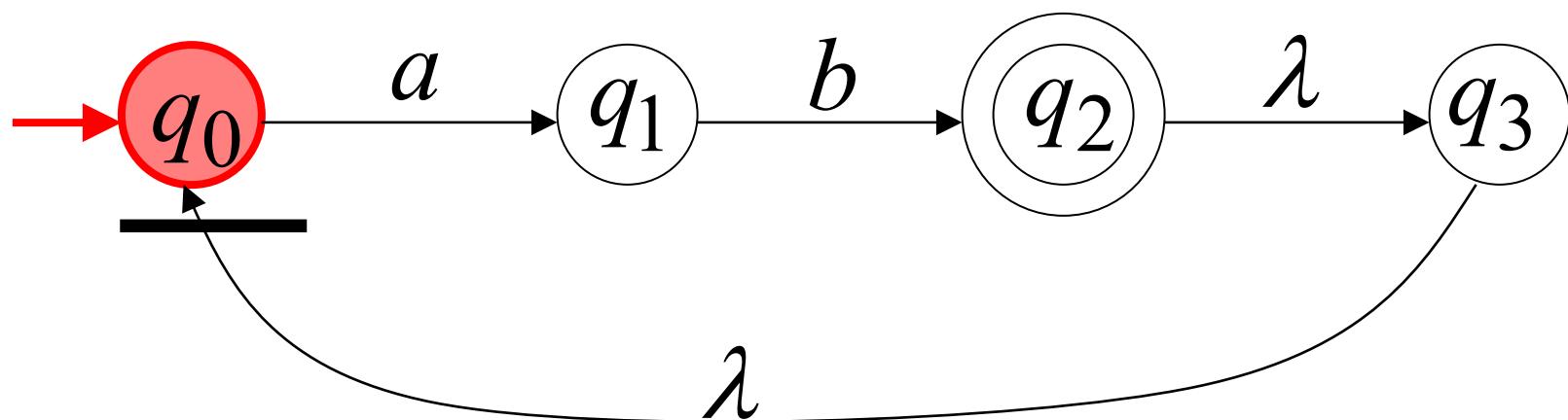


Esiste una computazione si
Per ogni computazione no

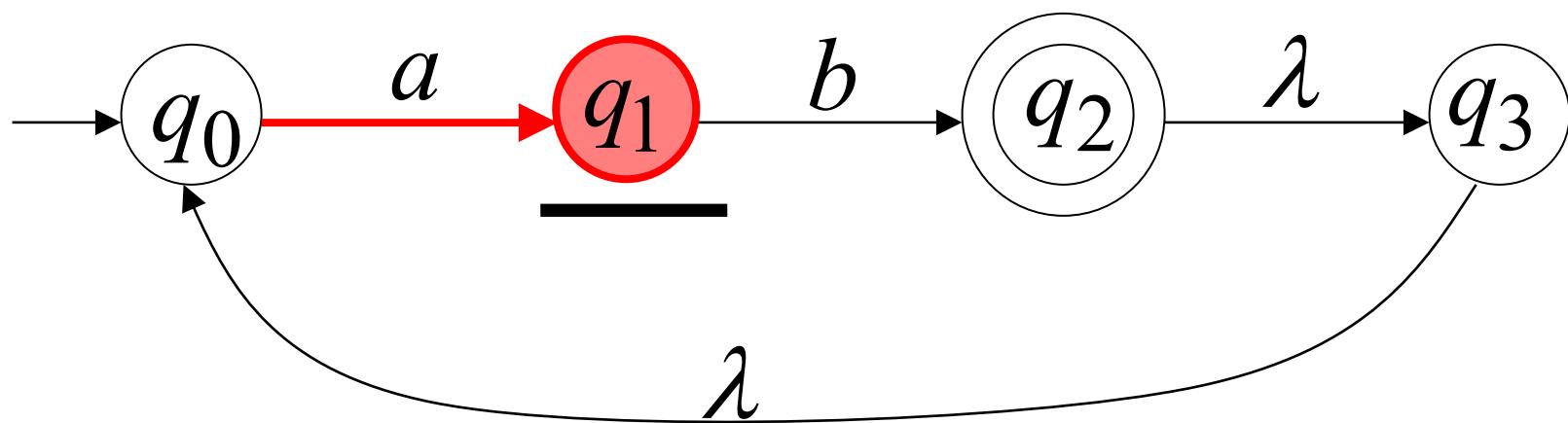
Un altro NFA



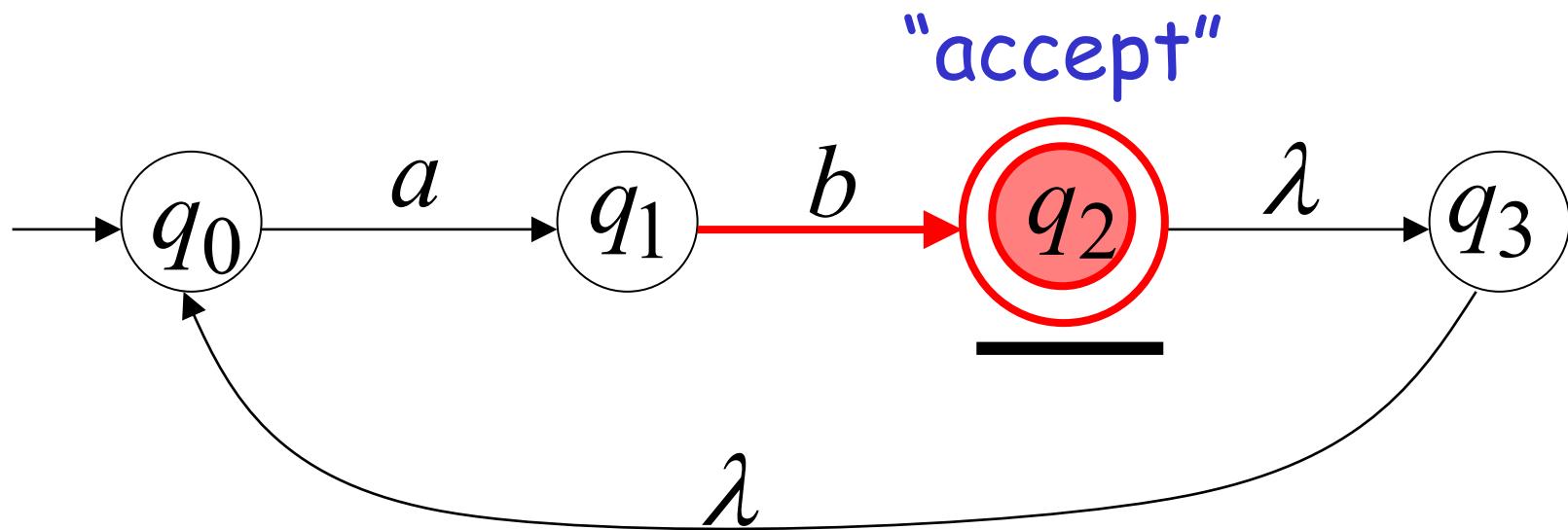
a	b	
-----	-----	--



a	b	
-----	-----	--



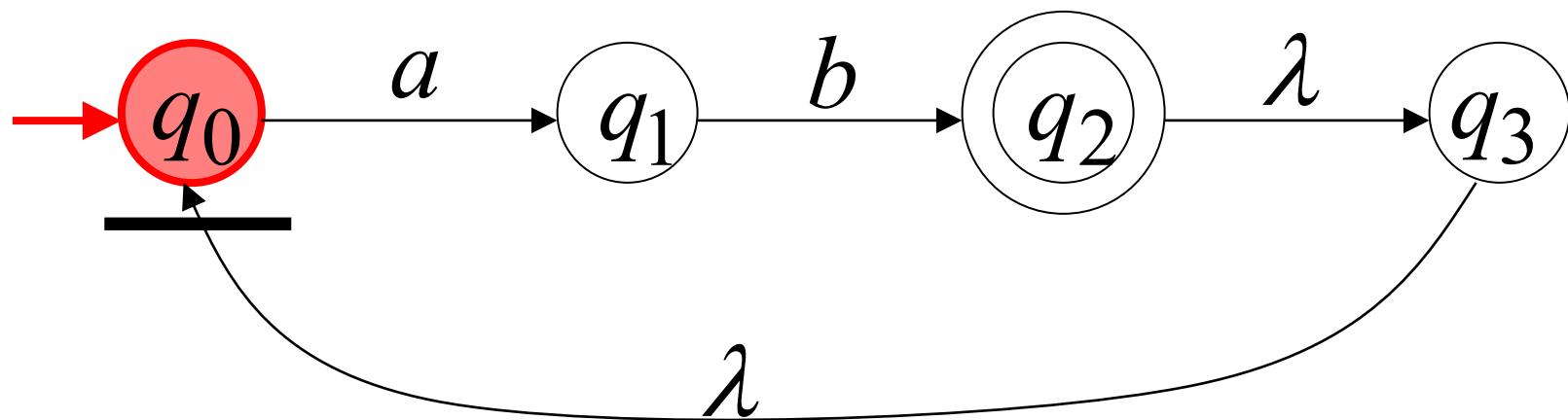
a	b	
-----	-----	--

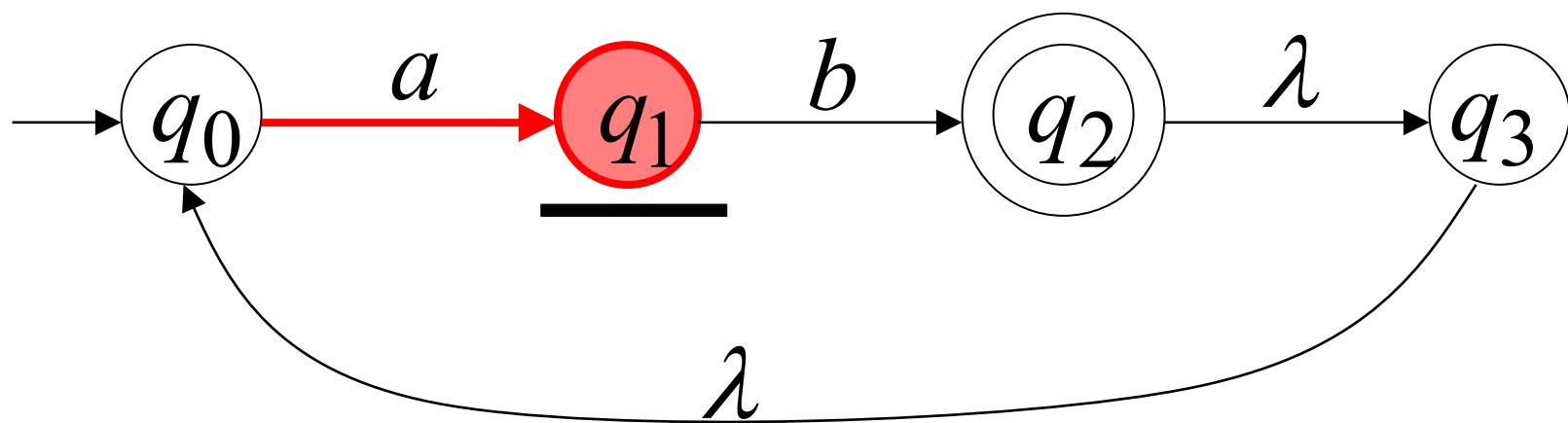
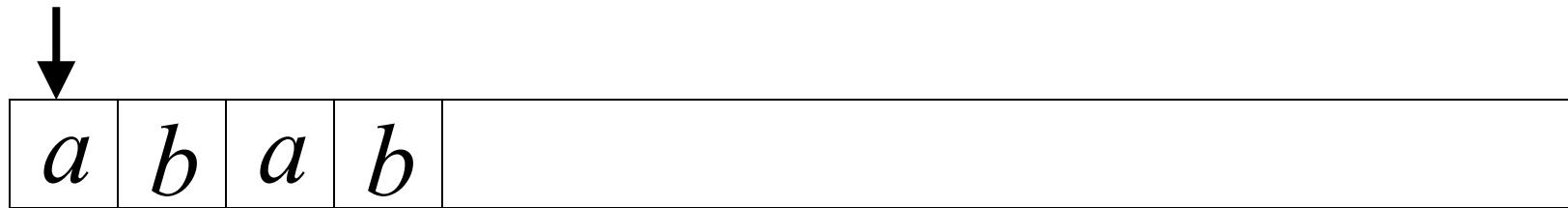


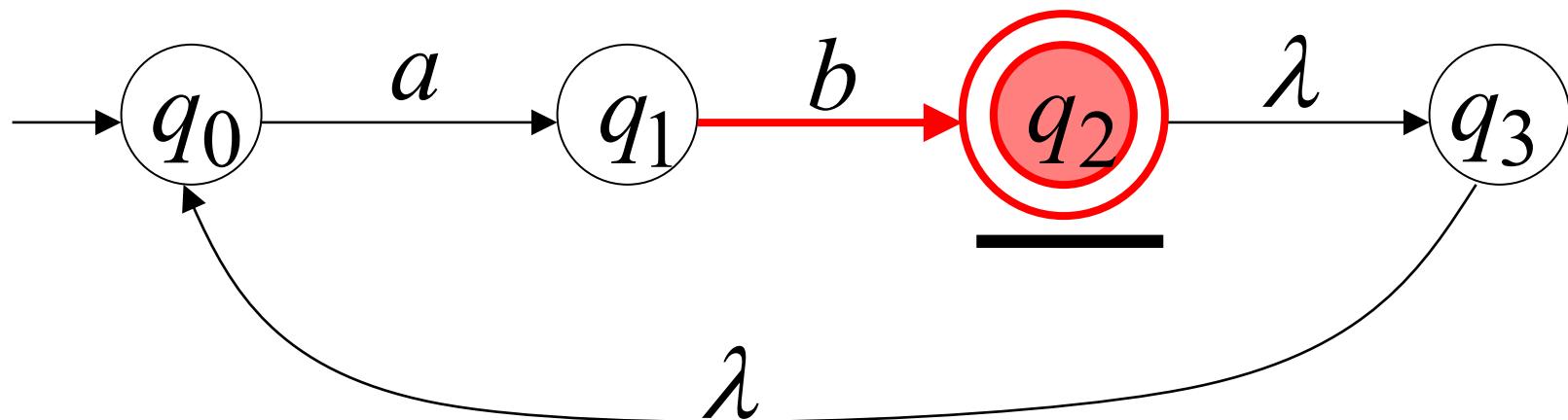
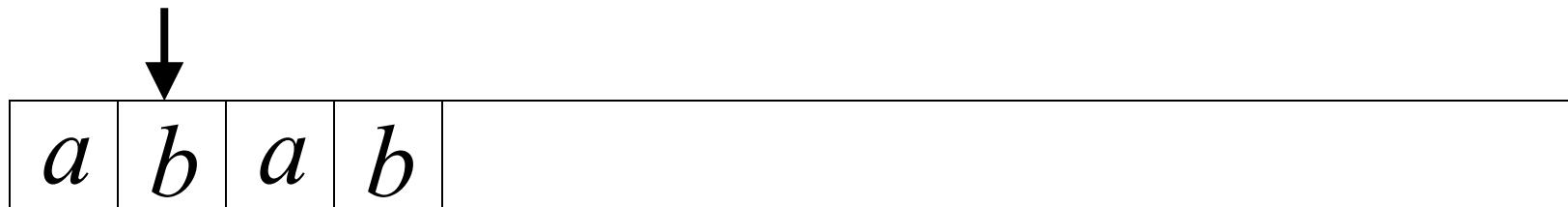
Un'altra stringa

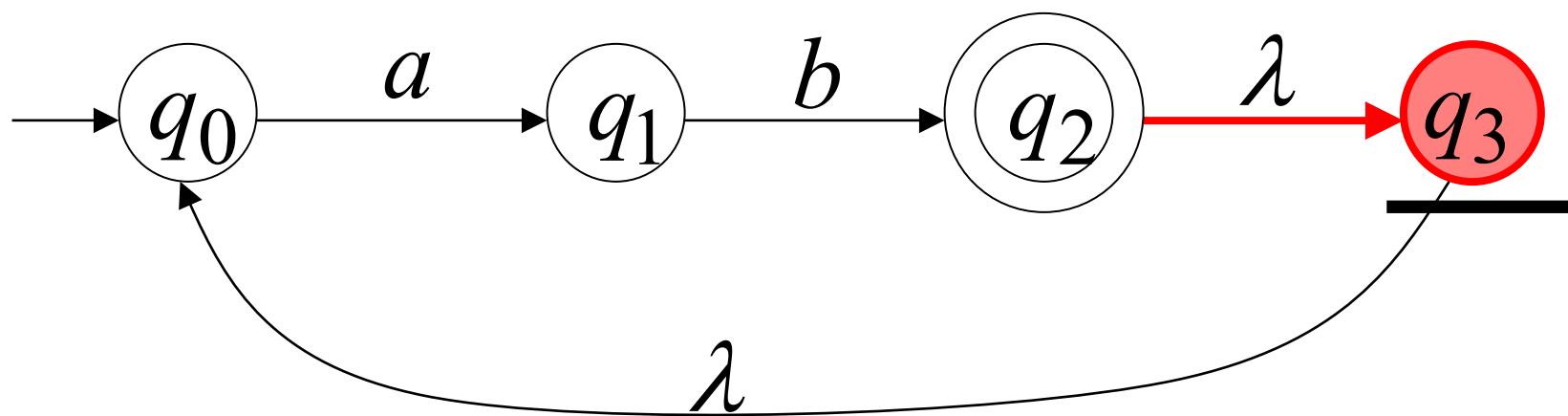
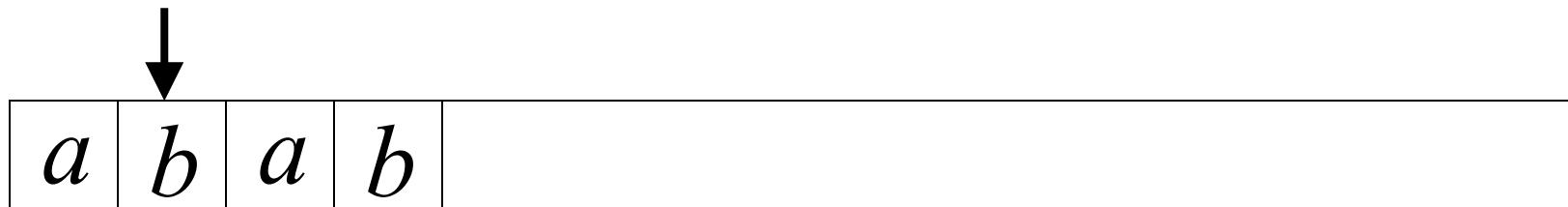


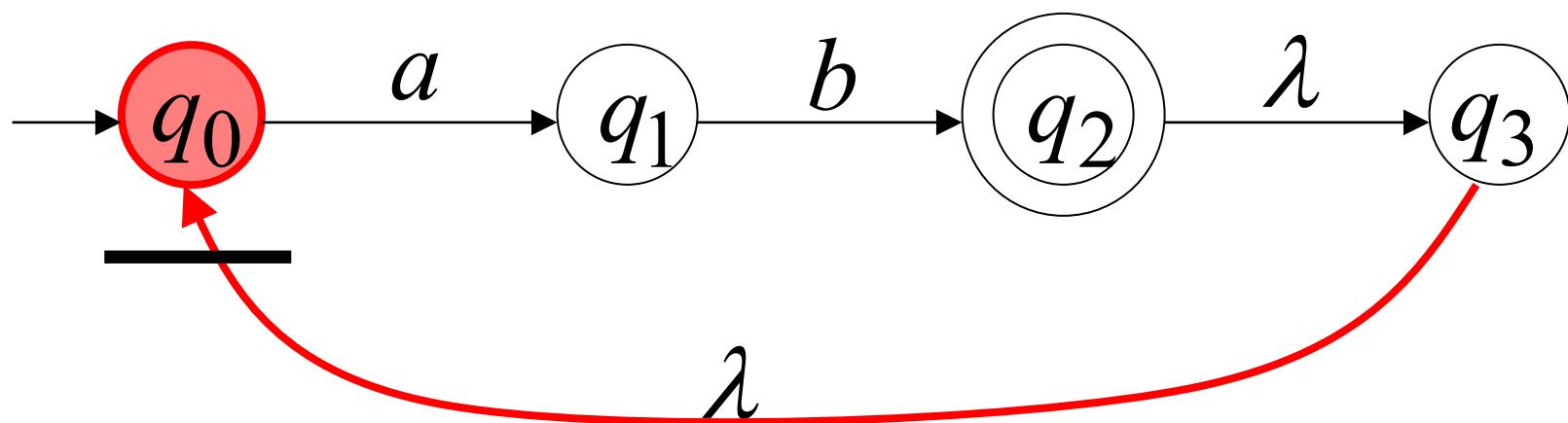
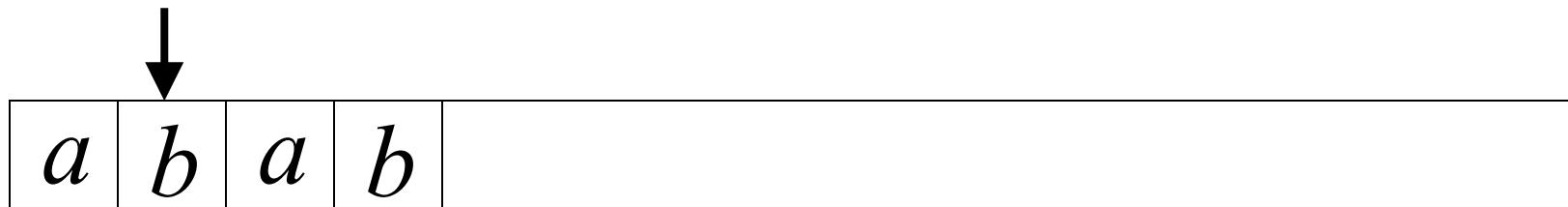
a	b	a	b	
-----	-----	-----	-----	--

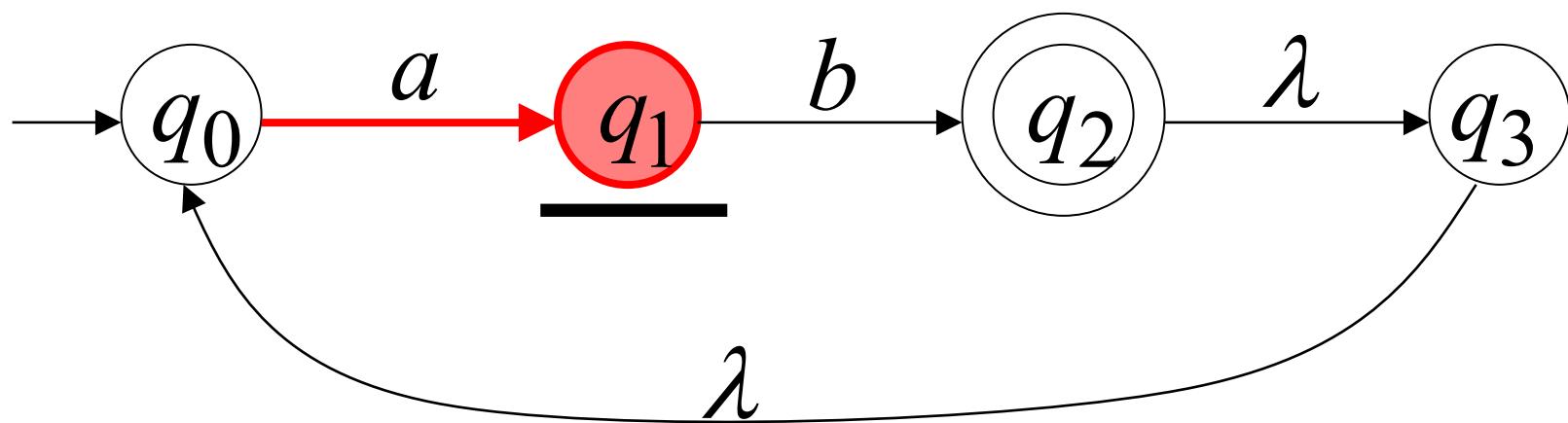
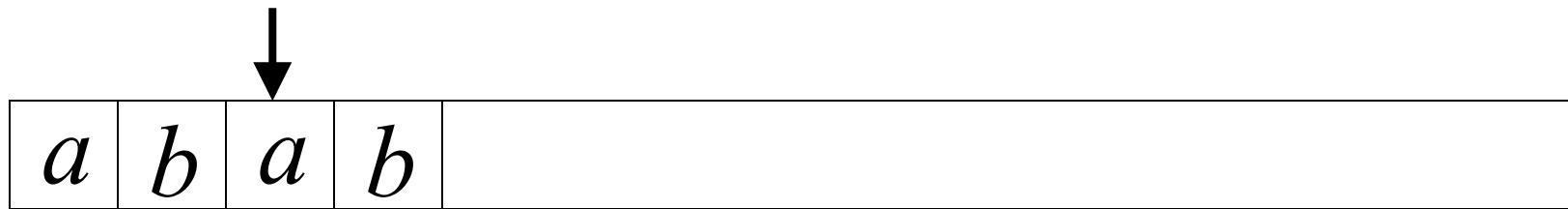


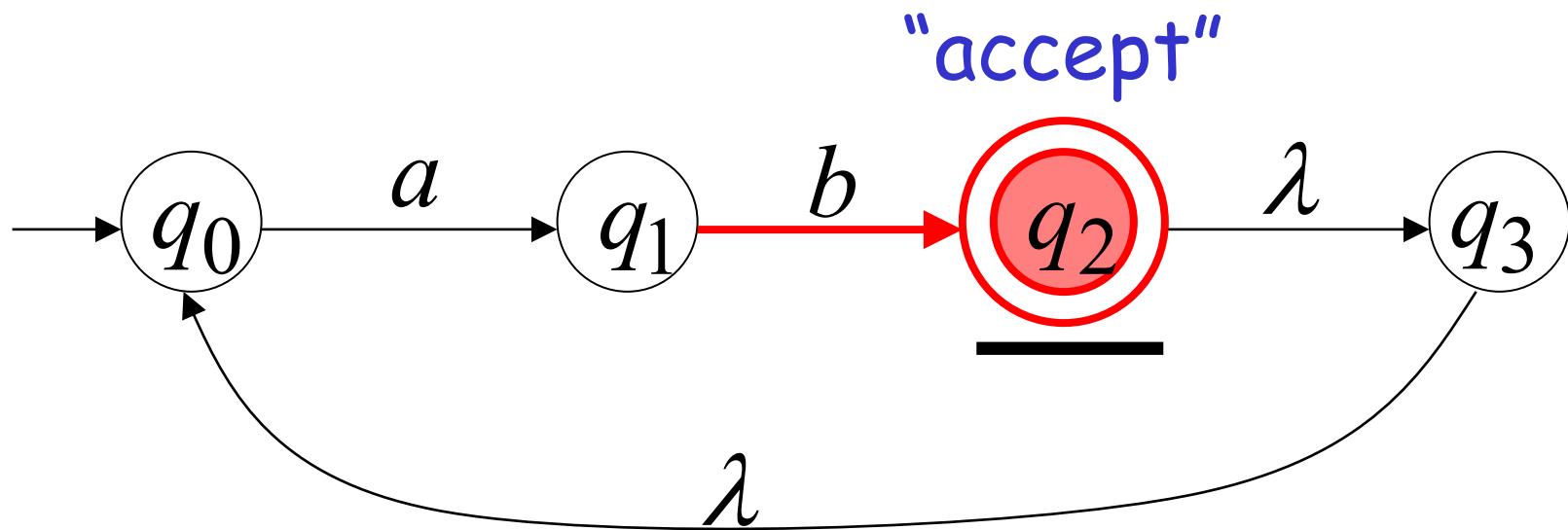
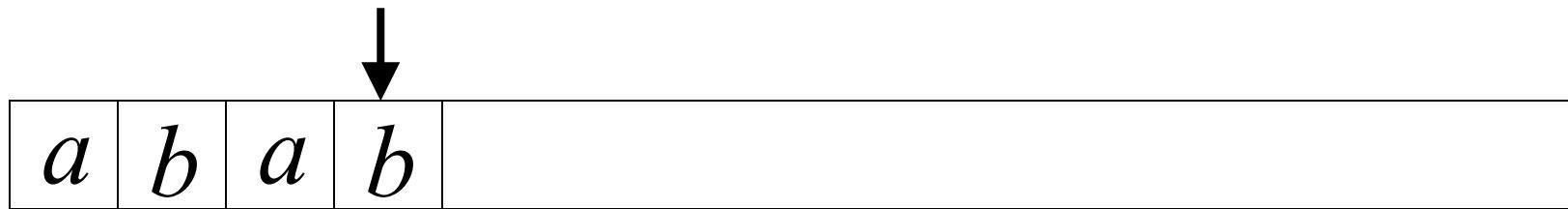








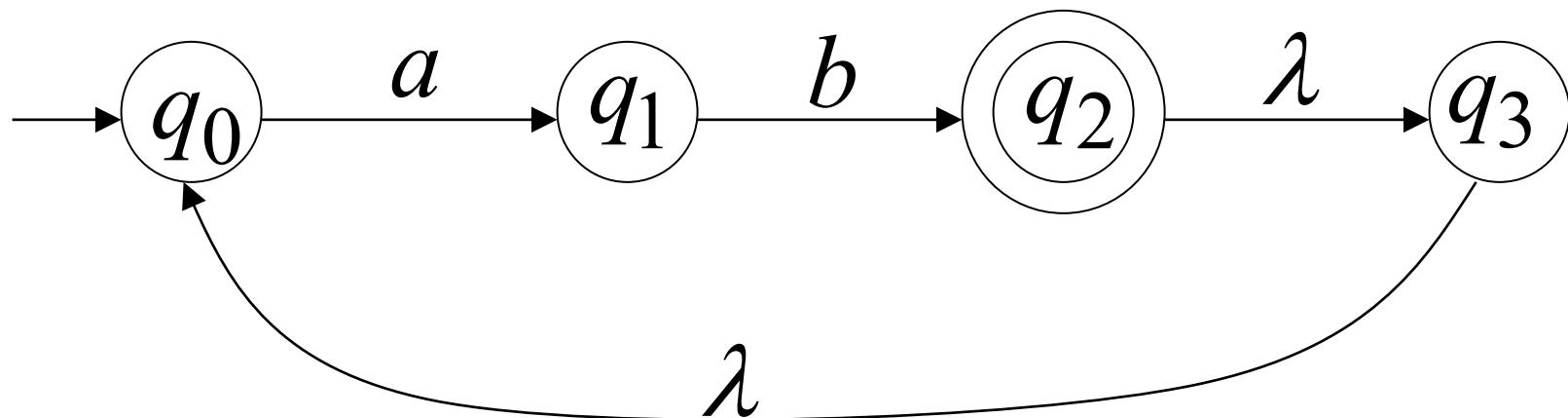




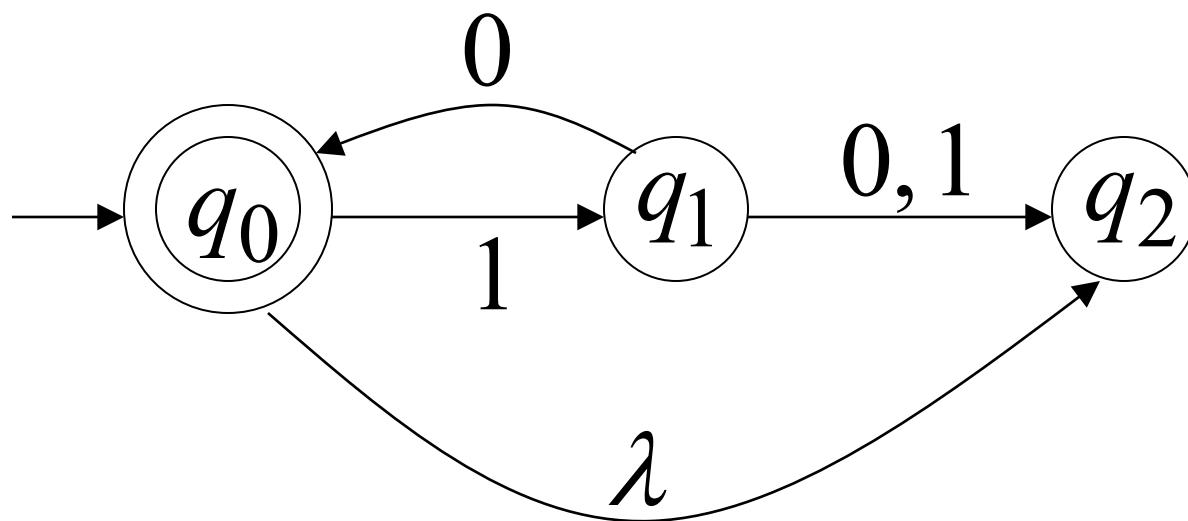
Linguaggio accettato

$$L = \{ab, abab, ababab, \dots\}$$

$$= \{ab\}^+$$

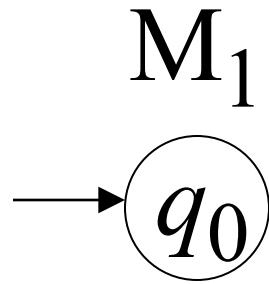


NFA esempio

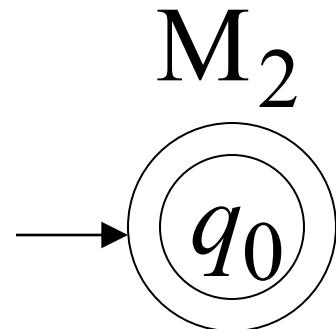


Remarks:

- Il simbolo λ non appare mai
 - sul nastro di input
 - Semplici automata:



$$L(M_1) = \{\}$$



$$L(M_2) = \{\lambda\}$$

Formal Definition of NFAs

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

Q : Set of states, i.e. $\{q_0, q_1, q_2\}$

Σ : Input alphabet, i.e. $\{a, b\}$ $\lambda \notin \Sigma$

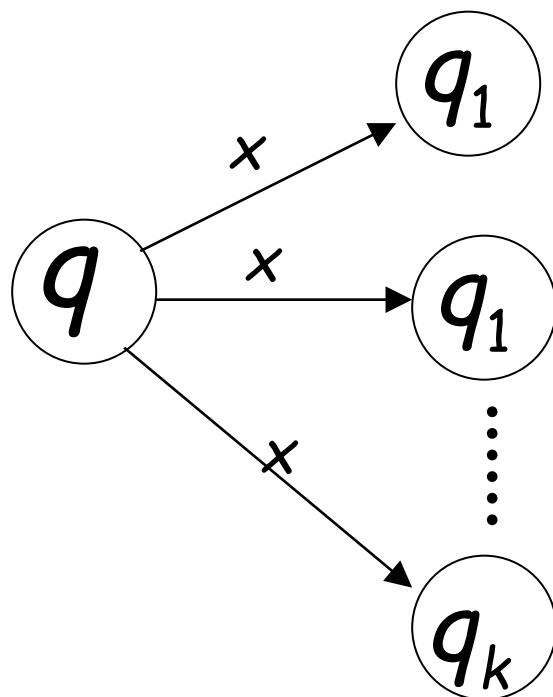
δ : Transition function

q_0 : Initial state

F : Accepting states

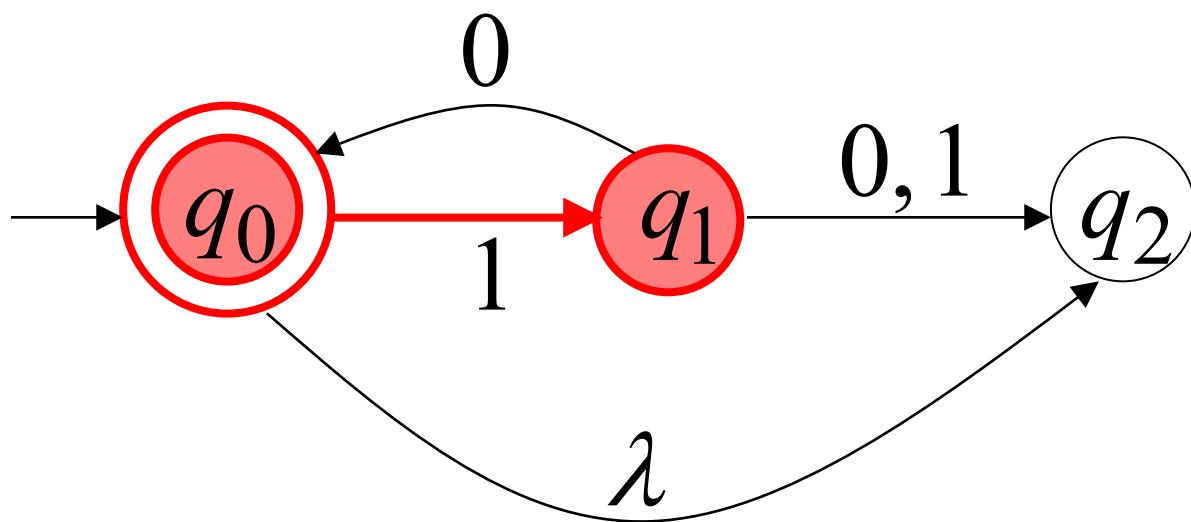
Funzione di transizione δ

$$\delta(q, x) = \{q_1, q_2, \dots, q_k\}$$

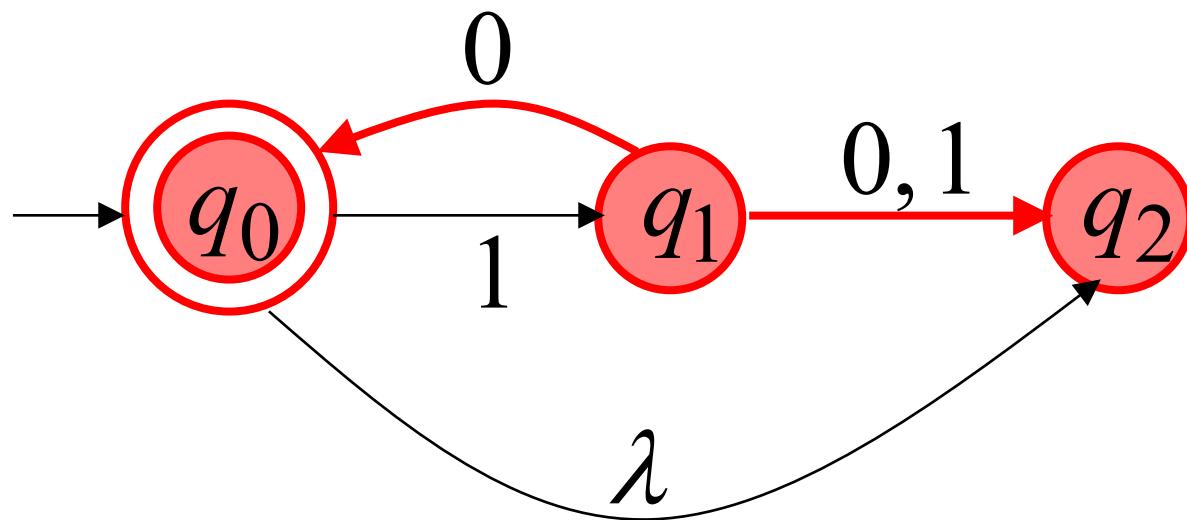


Stati risultanti con
una transizione
con simbolo x

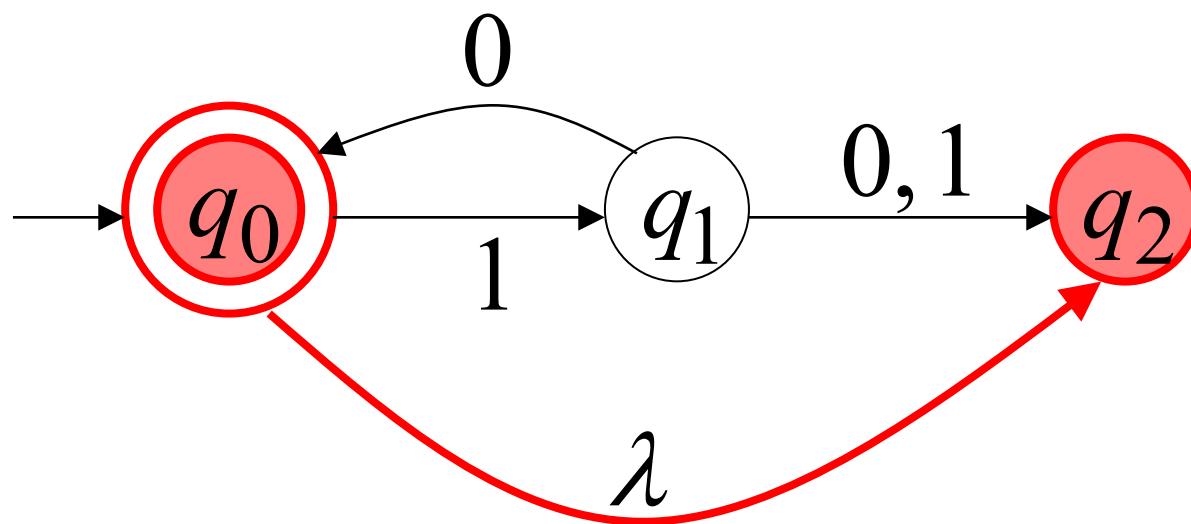
$$\delta(q_0, 1) = \{q_1\}$$



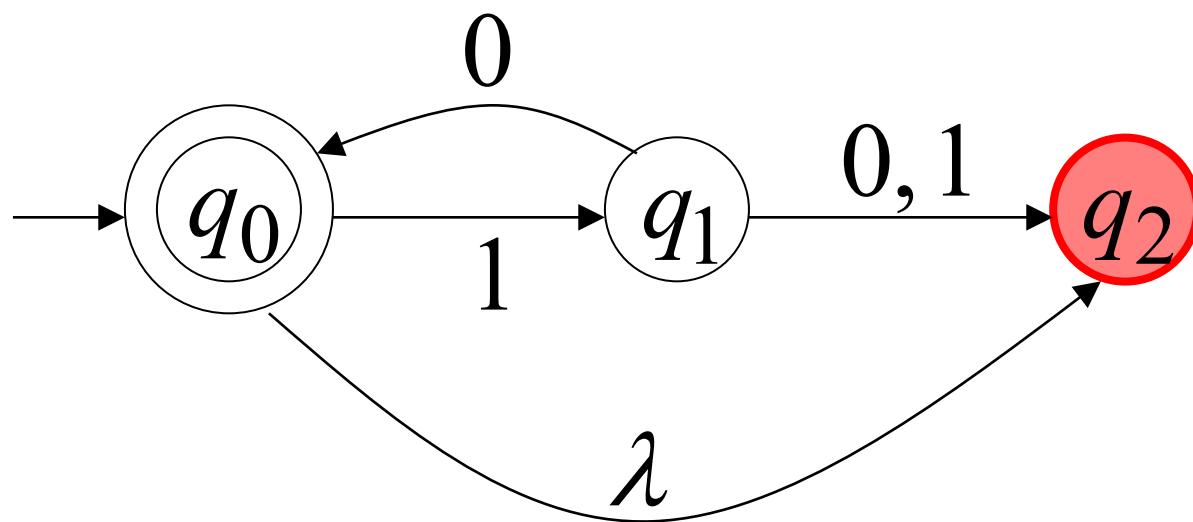
$$\delta(q_1, 0) = \{q_0, q_2\}$$



$$\delta(q_0, \lambda) = \{q_2\}$$



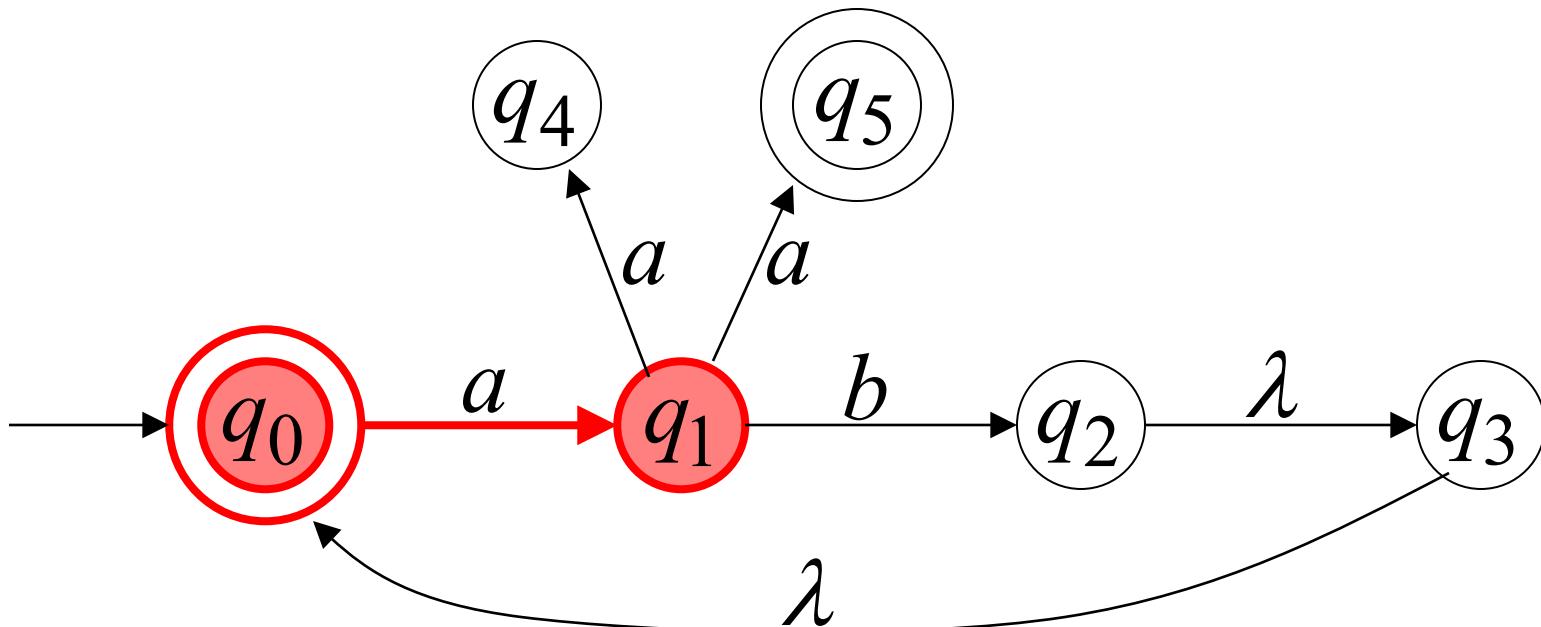
$$\delta(q_2, 1) = \emptyset$$



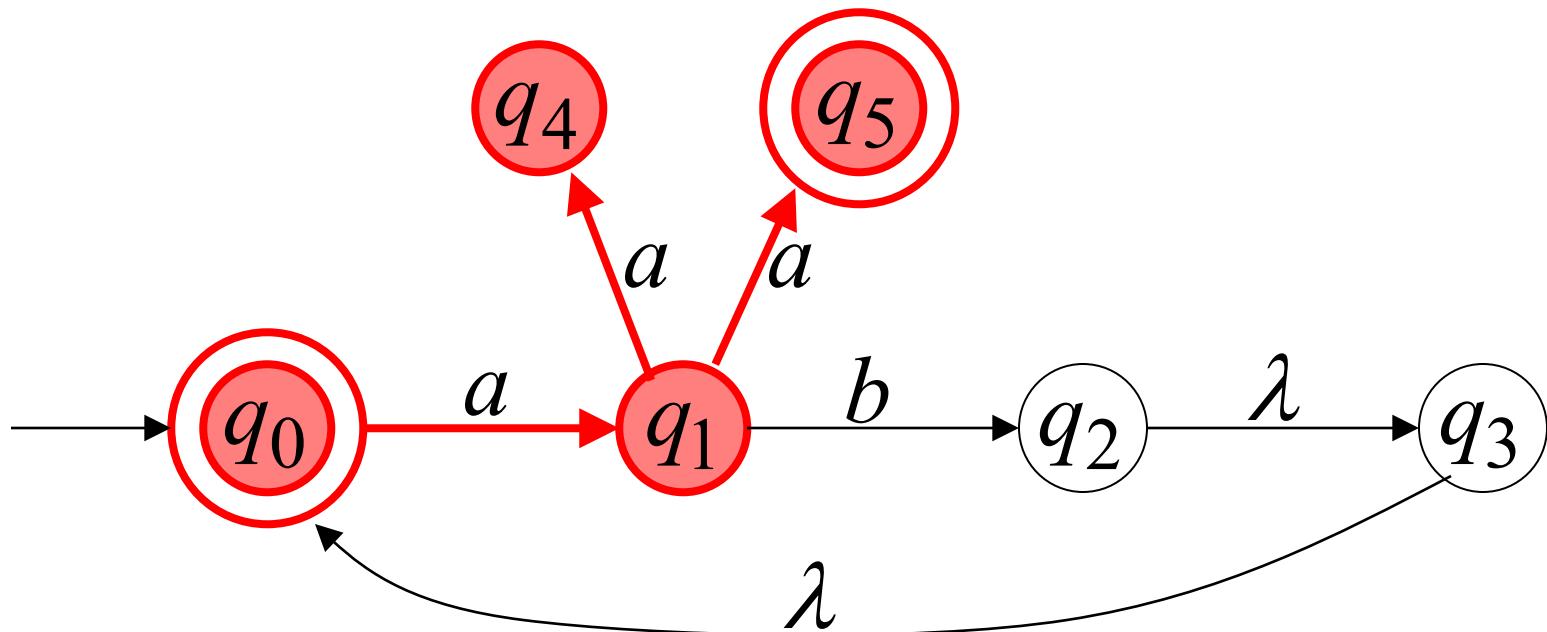
Funzione di transizione estesa δ^*

La stessa cosa δ ma applicata a stringhe

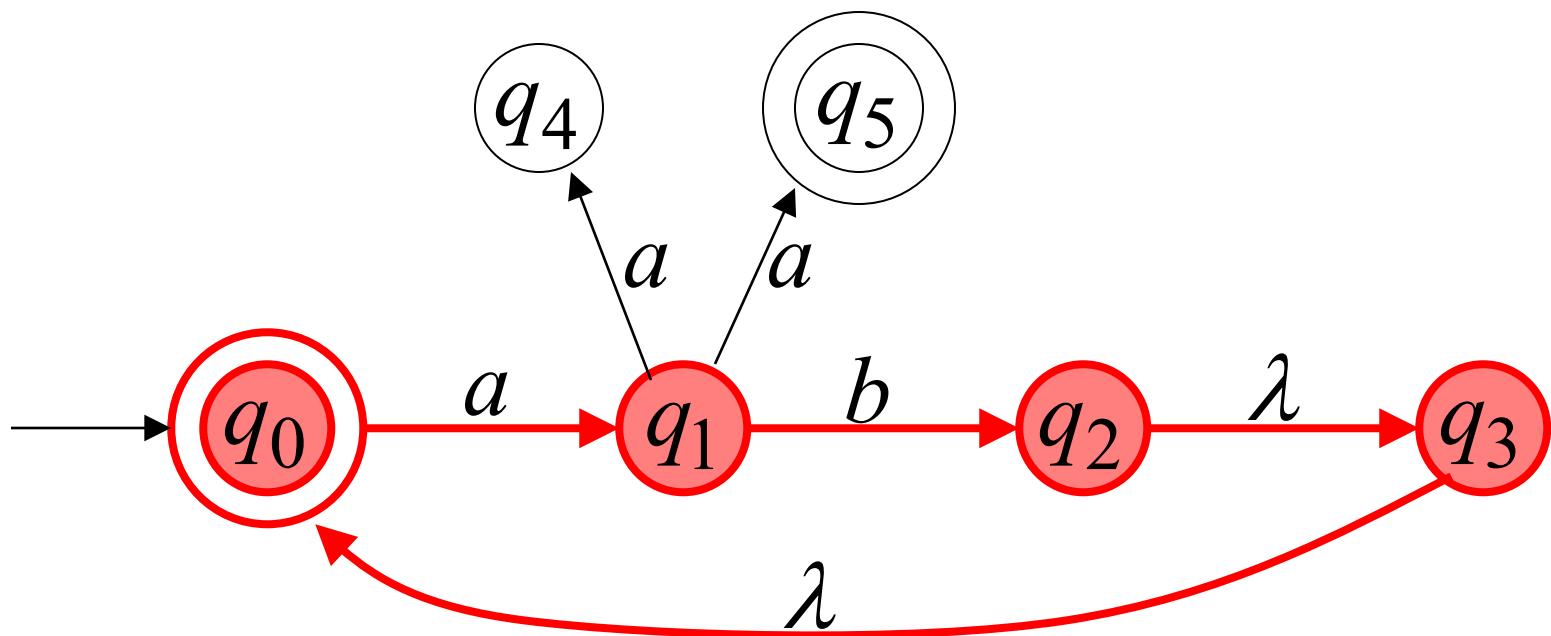
$$\delta^*(q_0, a) = \{q_1\}$$



$$\delta^*(q_0, aa) = \{q_4, q_5\}$$



$$\delta^*(q_0, ab) = \{q_2, q_3, q_0\}$$

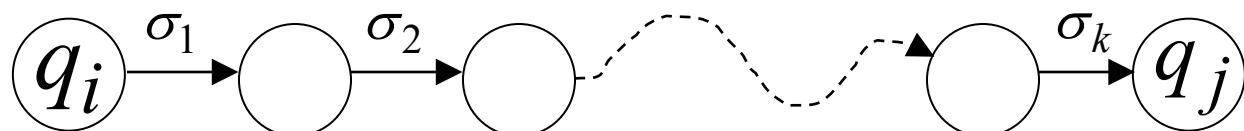


In generale

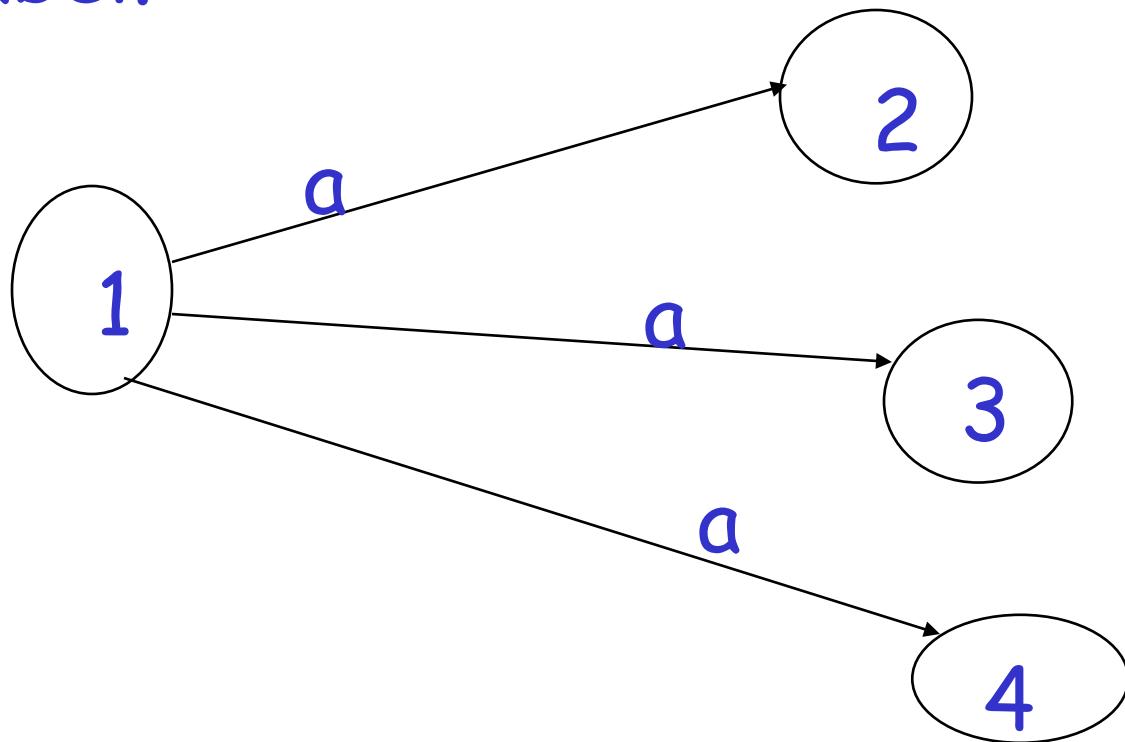
$q_j \in \delta^*(q_i, w)$: vi è un cammino da q_i a q_j
con label w



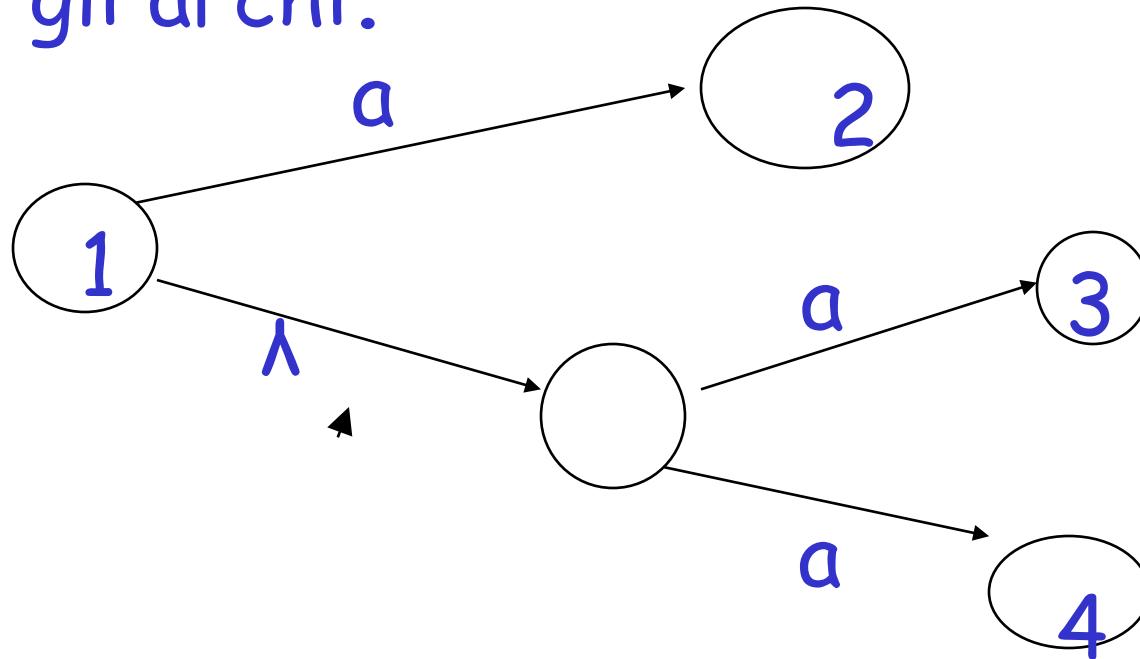
$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



Grado di non determinismo di un nodo per ogni nodo il numero di archi con la stessa label.



Grado di non determinismo di un automa, il grado massimo di non determinismo di tutti gli archi.



The Language of an NFA M

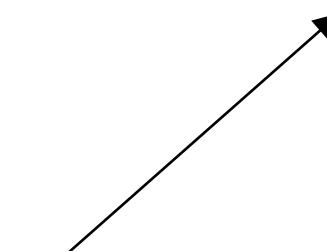
Il linguaggio accettato da M è:

$$L(M) = \{w_1, w_2, \dots, w_n\}$$

dove

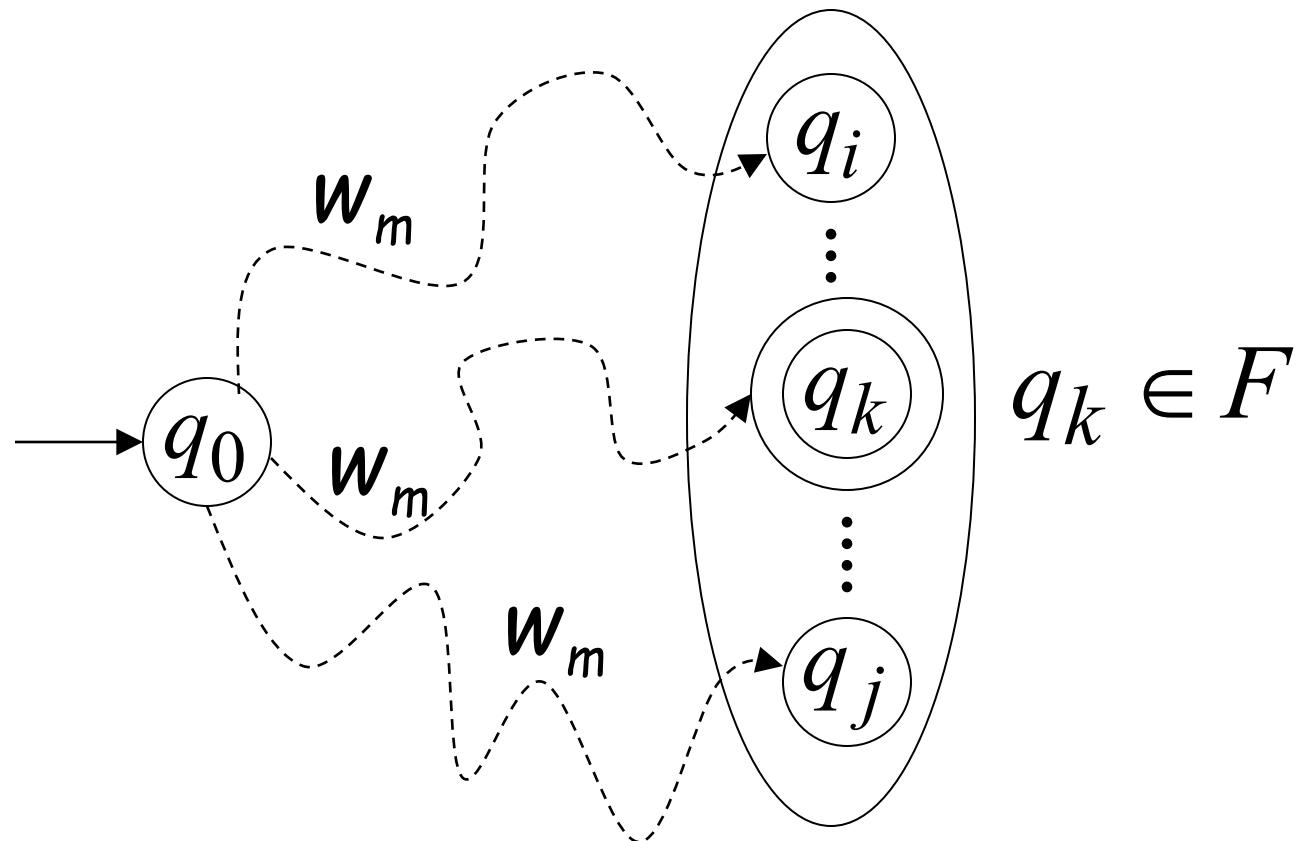
$$\delta^*(q_0, w_m) = \{q_i, \dots, q_k, \dots, q_j\}$$

E vi è un


 $q_k \in F$ (stato finale)

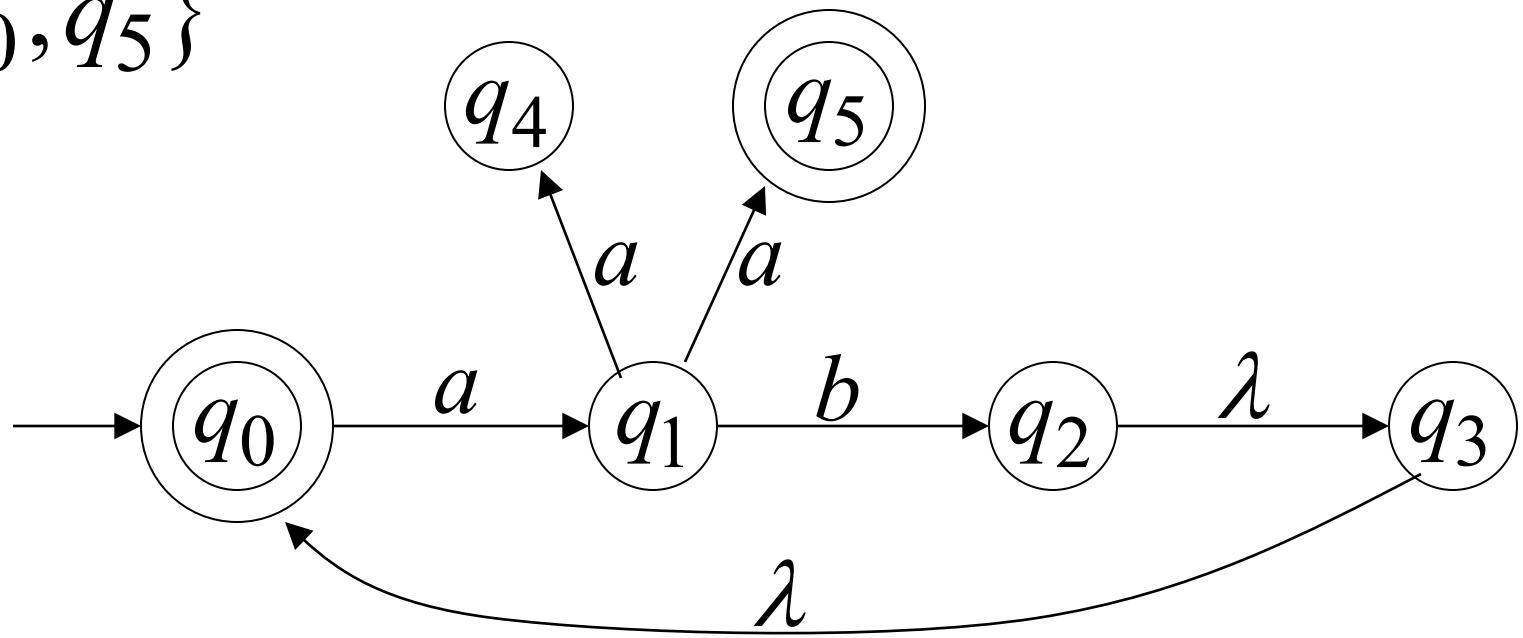
$$w_m \in L(M)$$

$$\delta^*(q_0, w_m)$$



$$q_k \in F$$

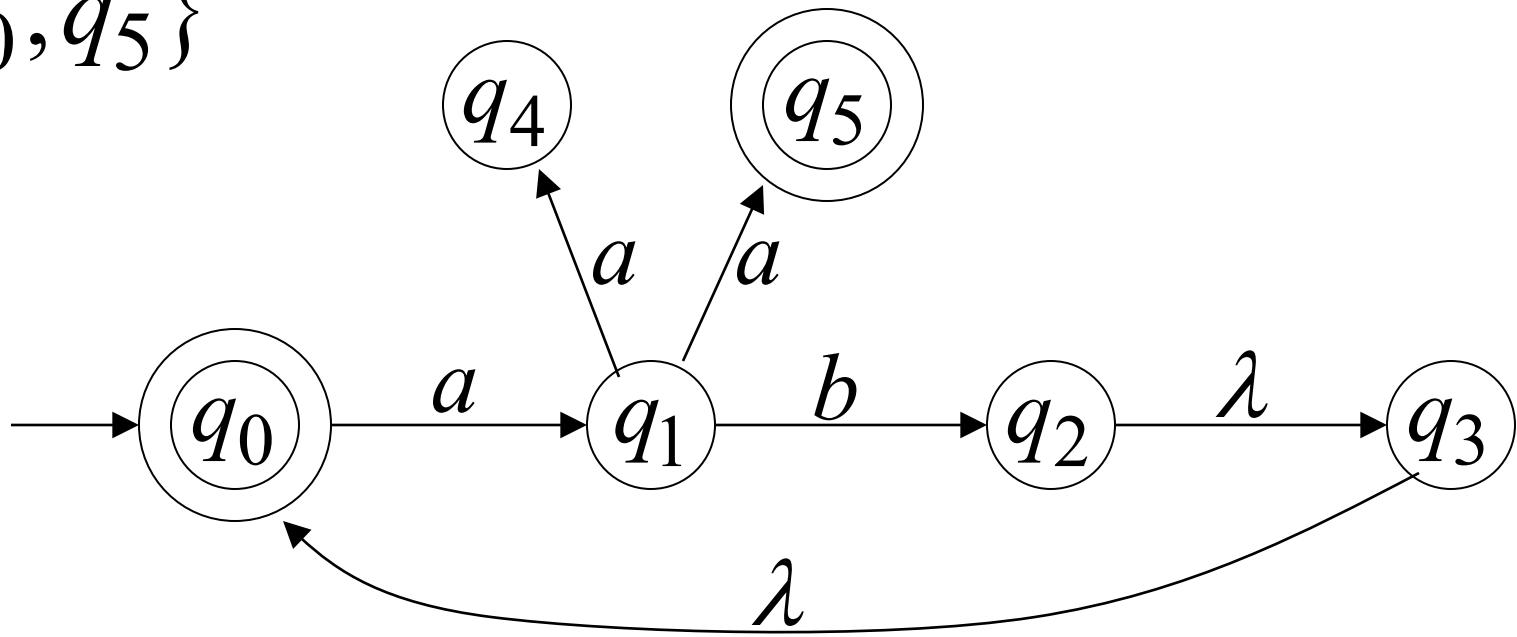
$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, aa) = \{q_4, \underline{q_5}\} \quad \Rightarrow \quad aa \in L(M)$$

$\in F$

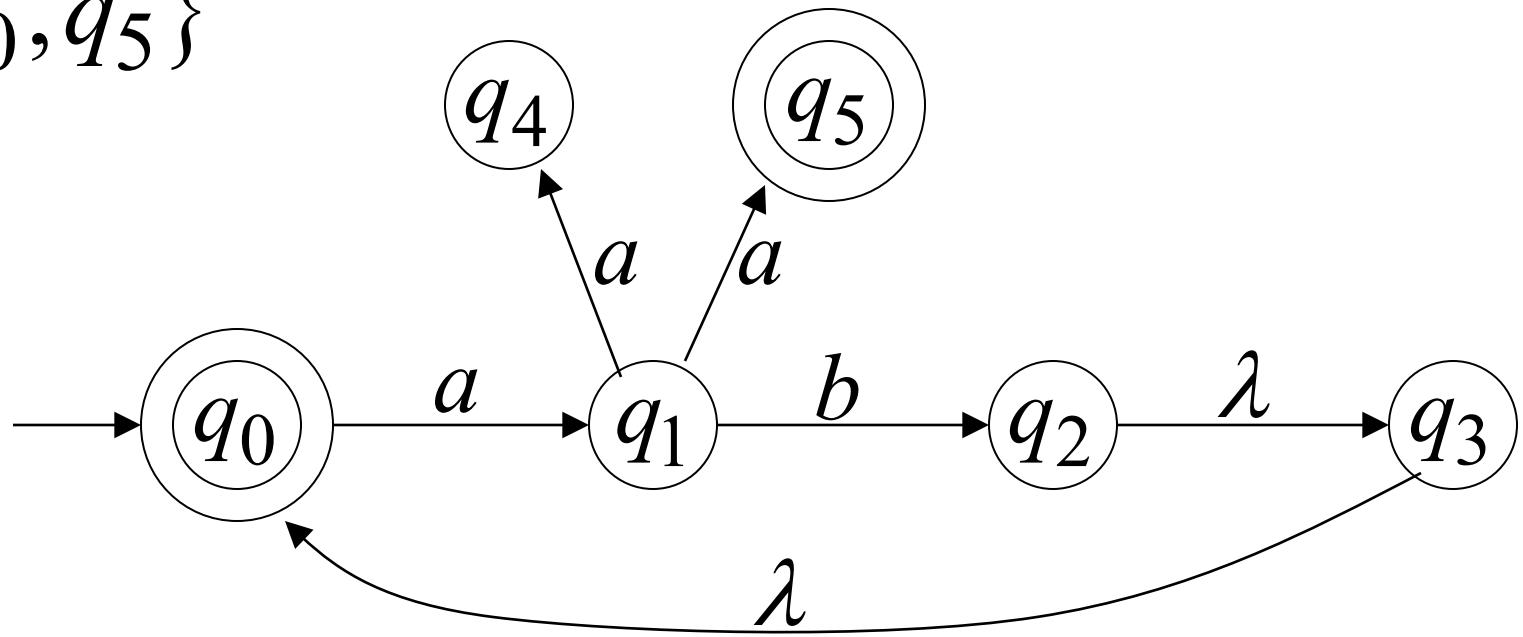
$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, ab) = \{q_2, q_3, \underline{q_0}\} \quad \text{Yellow Arrow} \quad ab \in L(M)$$

$\searrow \in F$

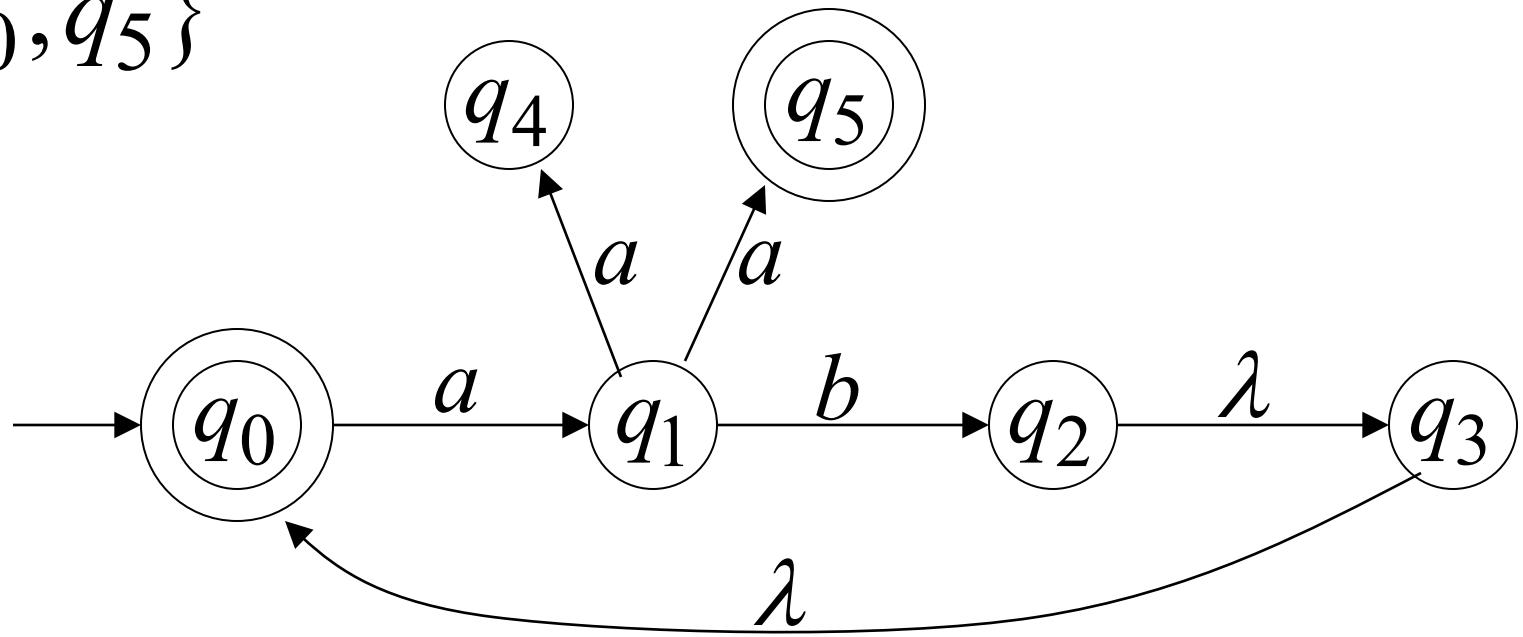
$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, aba) = \{q_4, \underline{q_5}\} \xrightarrow{\text{yellow arrow}} aaba \in L(M)$$

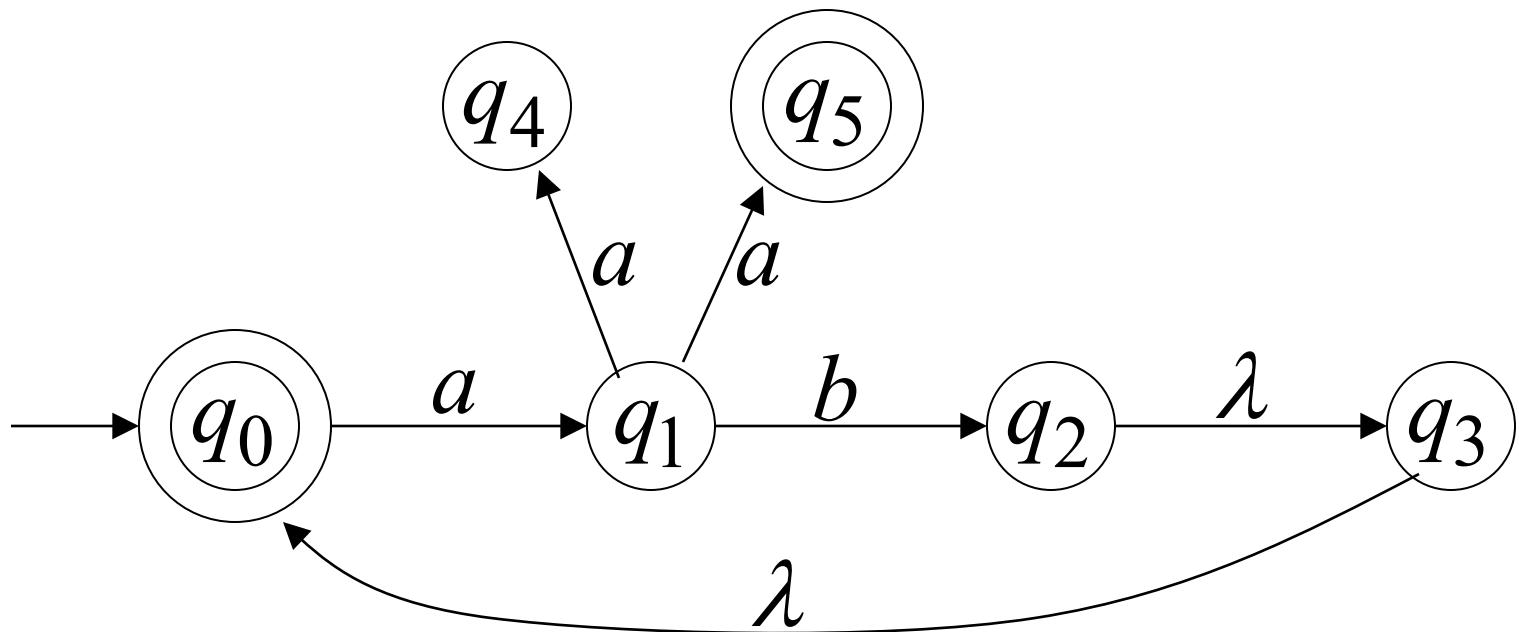
$\searrow \in F$

$$F = \{q_0, q_5\}$$



$$\delta^*(q_0, aba) = \{q_1\} \quad \xrightarrow{\text{yellow arrow}} \quad aba \notin L(M)$$

$\not\in F$



$$L(M) = \{ab\}^* \cup \{ab\}^* \{aa\}$$

$\delta^*(\text{stato}, cW) =$

{

$\delta^*(q, W)$

con q elemento dell'insieme $\{\delta(\text{stato}, c)\}$

}

$q \in \delta^*(q, \lambda)$ Per ogni stato

1 $\delta^*(\text{stato}, cW) =$

{

$\delta^*(q, W)$

con $q \in \{\delta(\text{stato}, c)\}$

}

2 $q \in \delta^*(q, \lambda)$ Per ogni stato

NFA accettano i linguaggi regolari

Equivalenza tra macchine

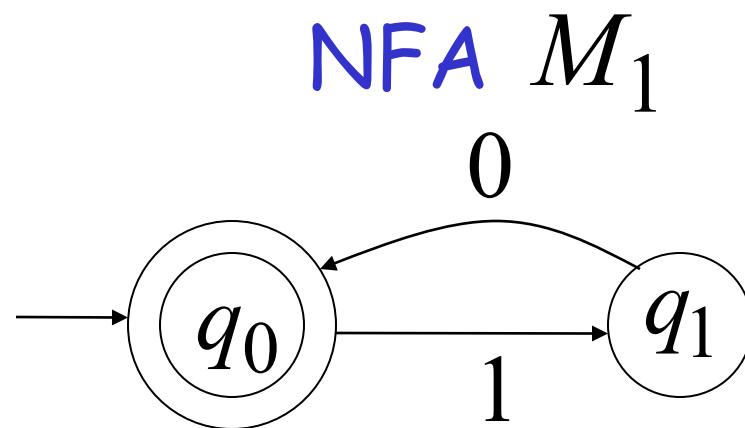
Definizione:

macchina M_1 è equivalente alla macchina M_2

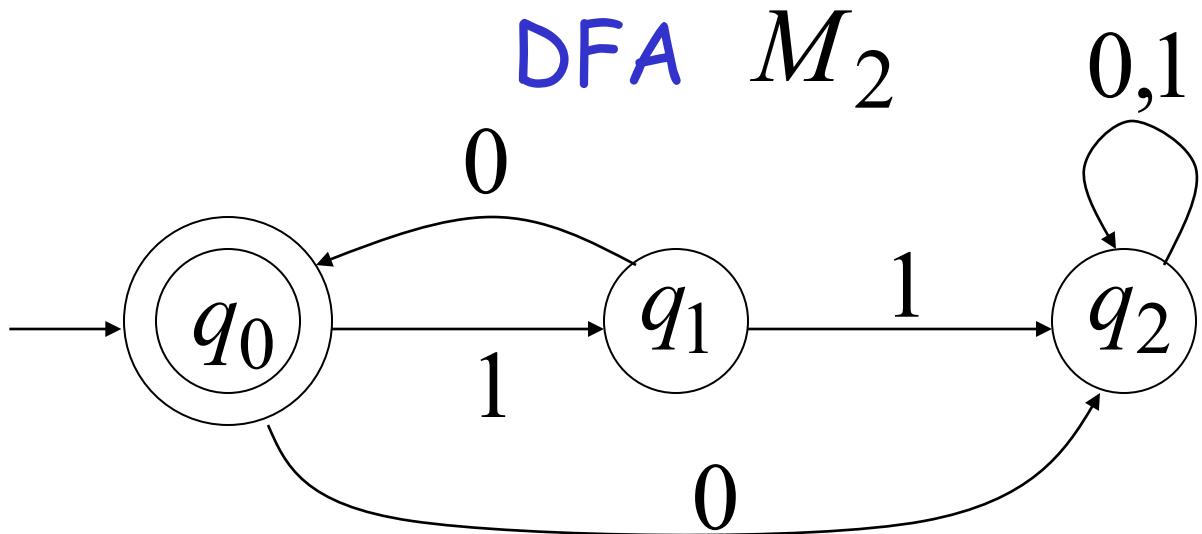
se $L(M_1) = L(M_2)$

Esempio di macchine equivalenti

$$L(M_1) = \{10\}^*$$



$$L(M_2) = \{10\}^*$$



Teorema:

$$\left\{ \begin{array}{l} \text{Linguaggi} \\ \text{Accettati} \\ \text{da NFA} \end{array} \right\} = \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

Linguaggi
Accettati da un
DFA

NFA e DFA hanno lo stesso potere di computazione,
Accettano gli stessi linguaggi.

dimostrazione: mostreremo

$$\left\{ \begin{array}{l} \text{Linguaggi a} \\ \text{Accettati} \\ \text{da NFA} \end{array} \right\} \equiv \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

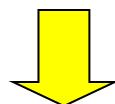
AND

$$\left\{ \begin{array}{l} \text{Linguaggi a} \\ \text{Accettati} \\ \text{da NFA} \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

Parte prima

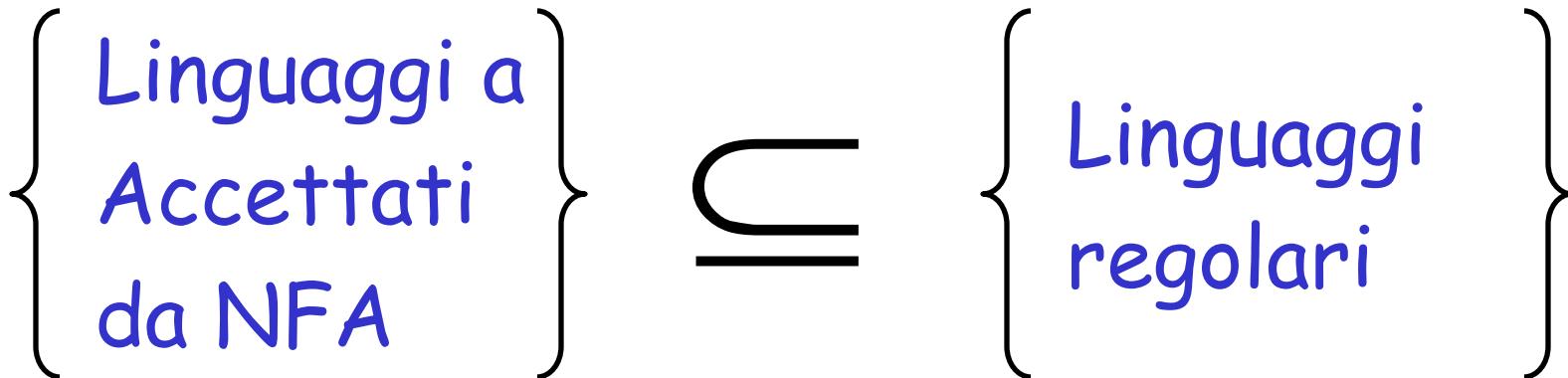
$$\left\{ \begin{array}{l} \text{Linguaggi a} \\ \text{Accettati} \\ \text{da NFA} \end{array} \right\} \equiv \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

ogni DFA è banalmente un NFA

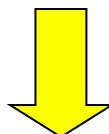


Ogni linguaggio L accettato da un DFA
È anche accettato da un NFA

Parte seconda



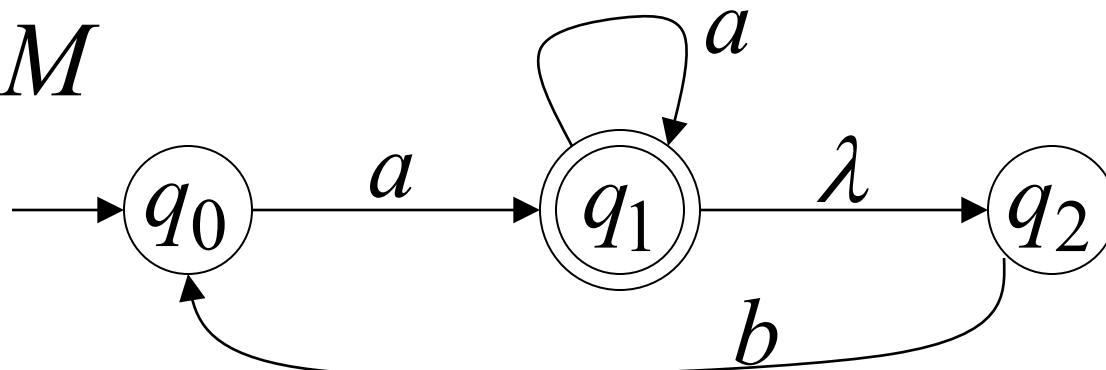
Ogni nfa può essere tradotto in un nfa



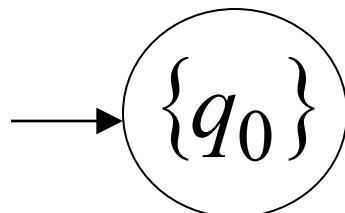
Ogni linguaggio L accettato da un NFA
È anche accettato da un DFA

Conversione da NFA a DFA

NFA M

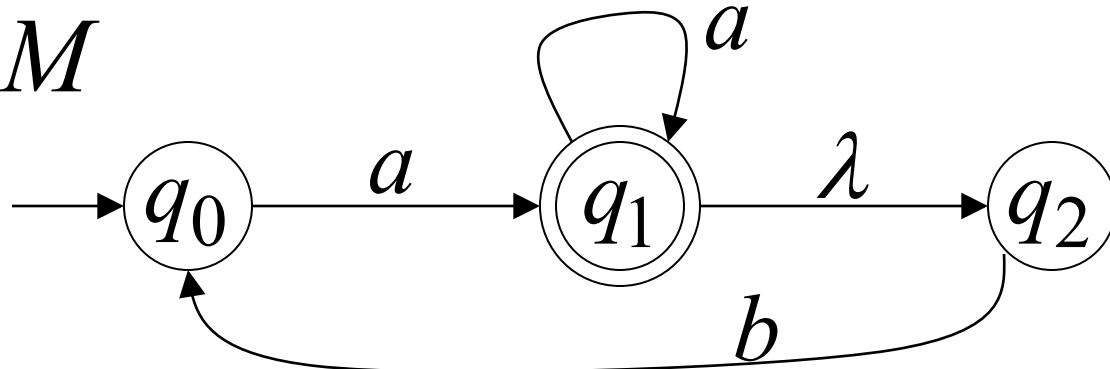


DFA M'

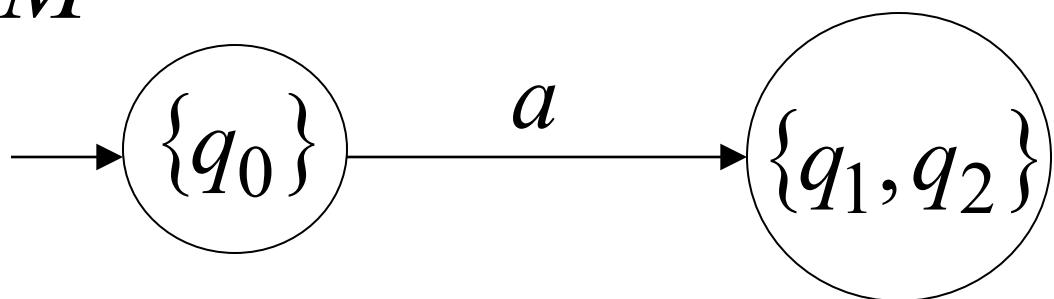


$$\delta^*(q_0, a) = \{q_1, q_2\}$$

NFA M

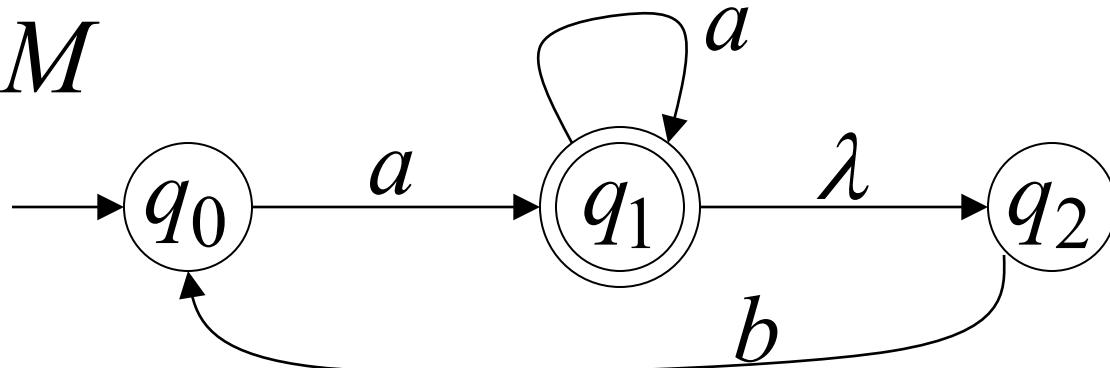


DFA M'

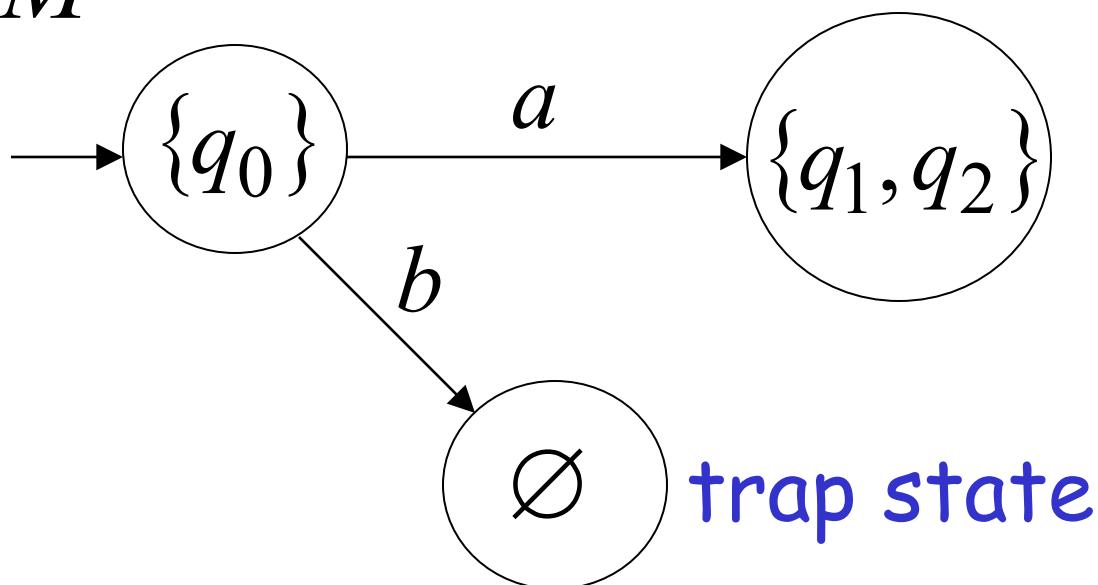


$$\delta^*(q_0, b) = \emptyset \quad \text{Insieme vuoto}$$

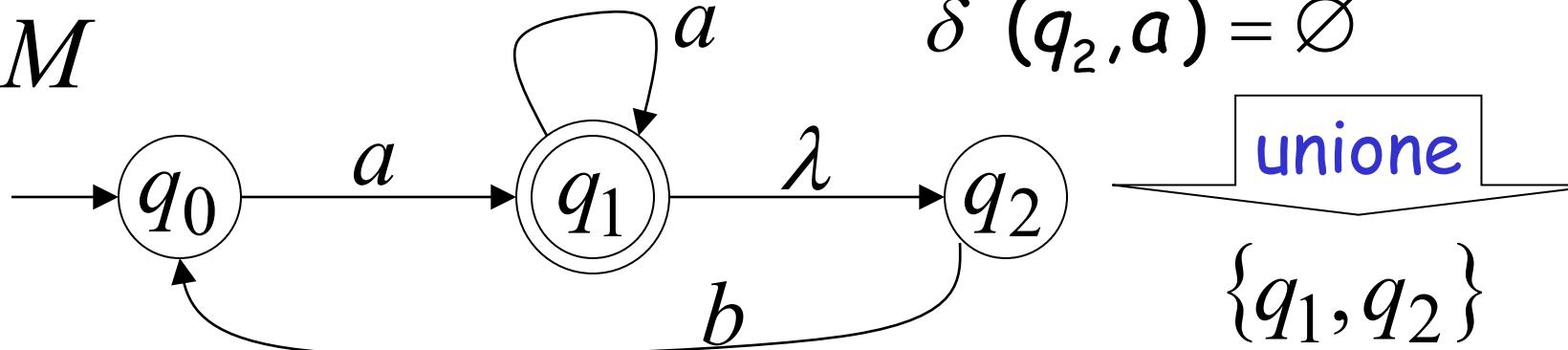
NFA M



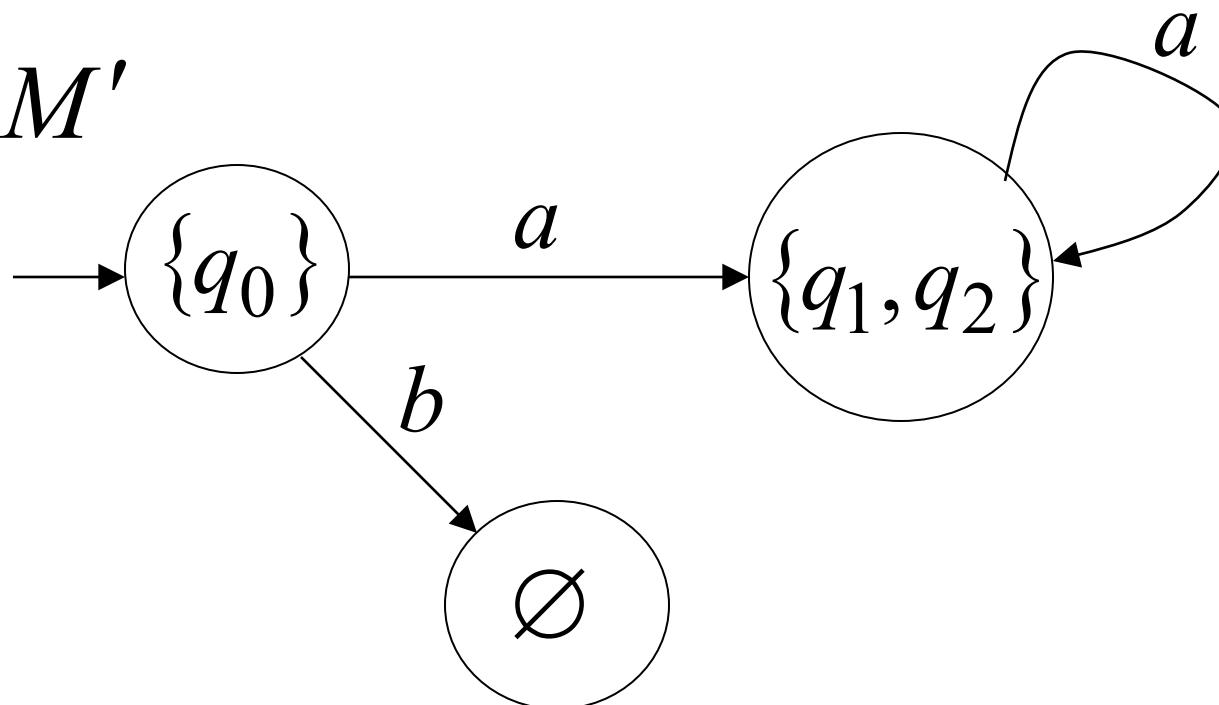
DFA M'



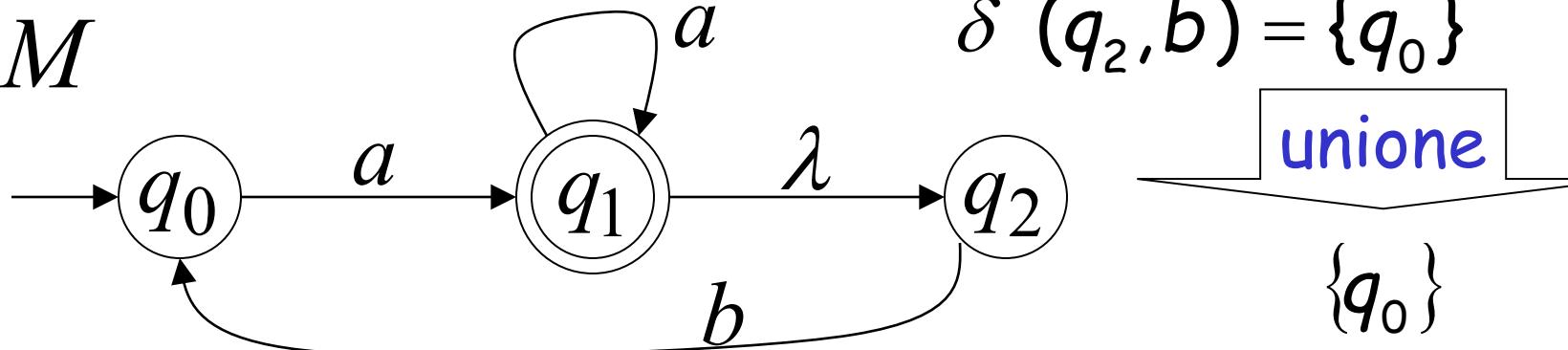
NFA M



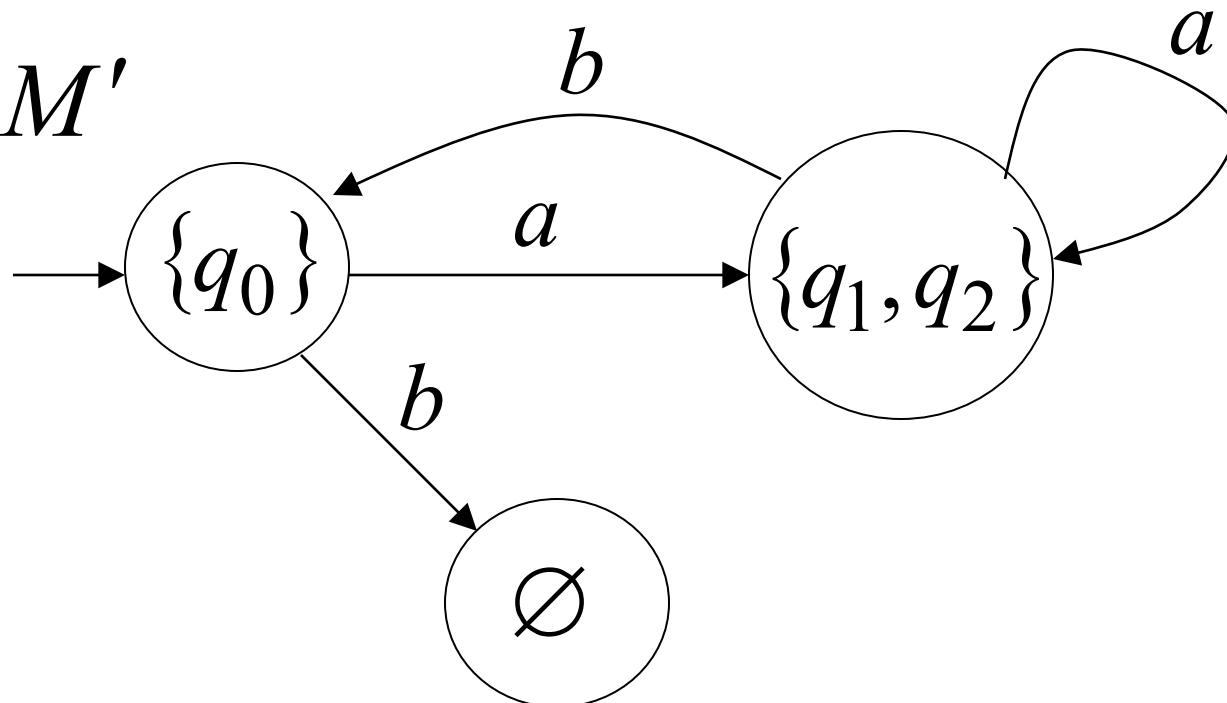
DFA M'



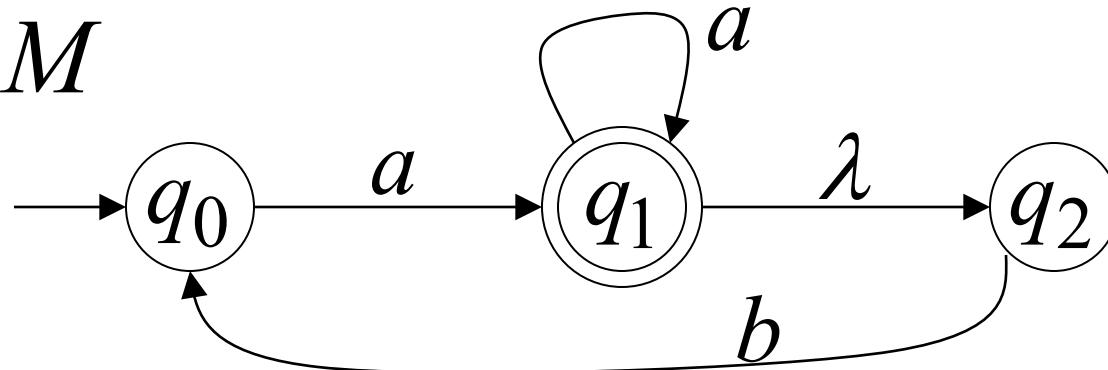
NFA M



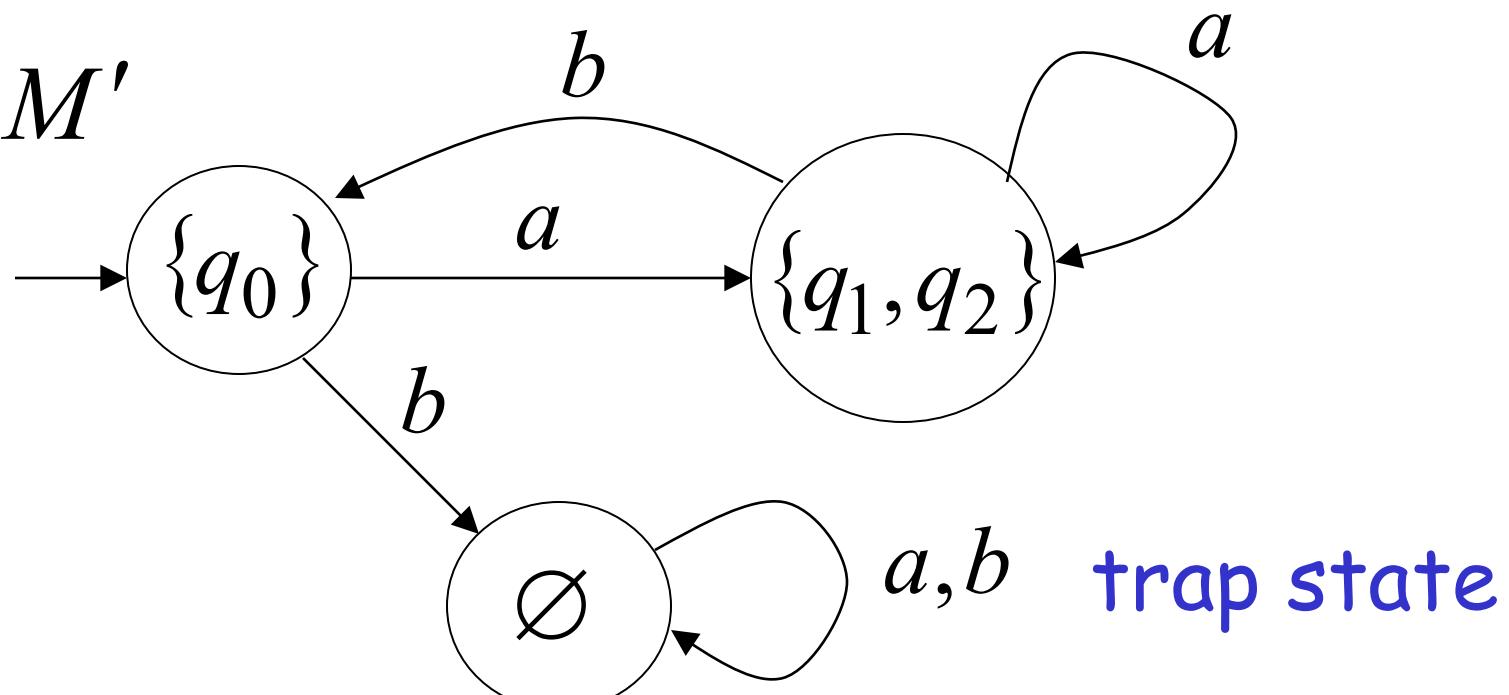
DFA M'



NFA M

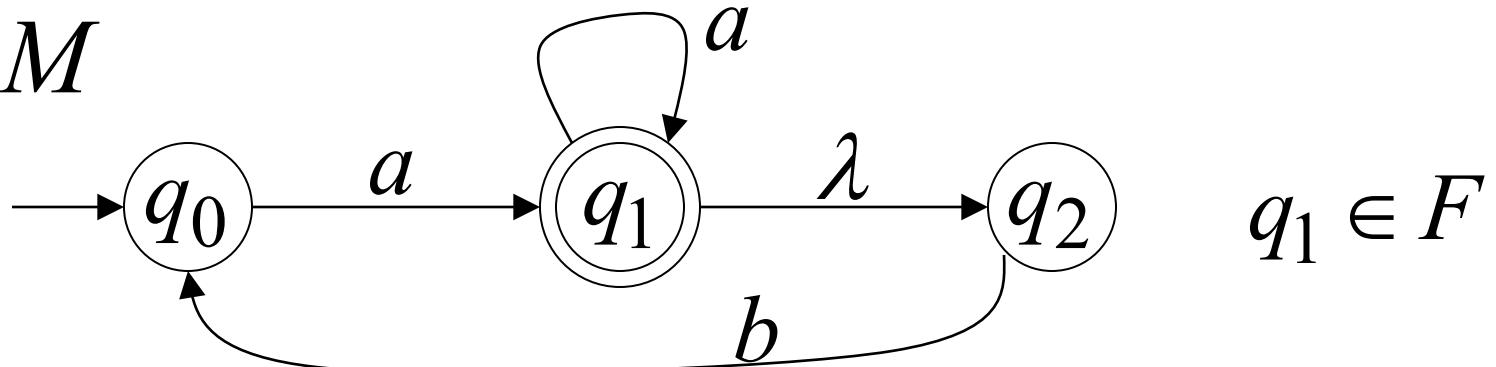


DFA M'



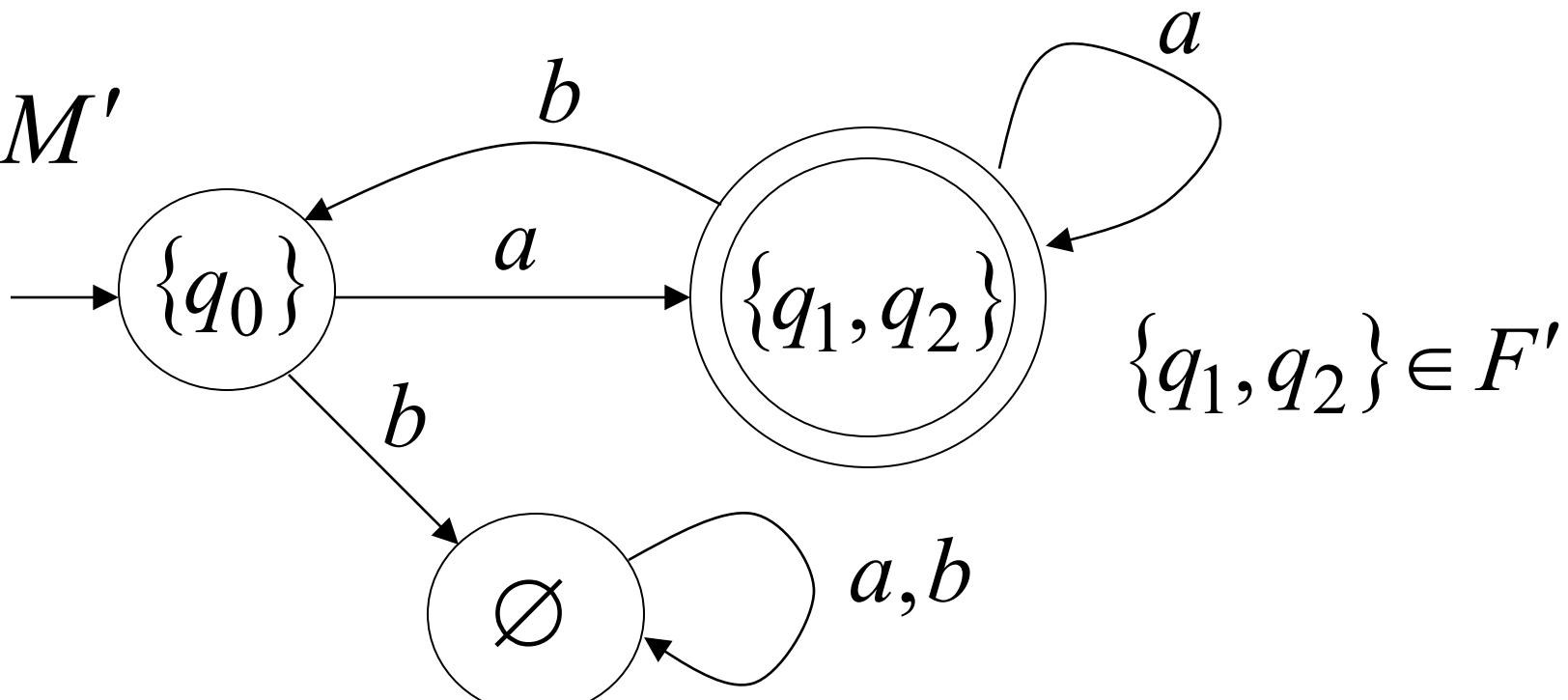
Fine della costruzione

NFA M



$$q_1 \in F$$

DFA M'



$$\{q_1, q_2\} \in F'$$

Procedura generale

Input: NFA M

Output: un equivalente DFA M'
con $L(M) = L(M')$

NFA ha gli stati

q_0, q_1, q_2, \dots

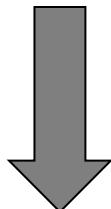
DFA ha gli stati definiti dall'insieme delle parti

$\emptyset, \{q_0\}, \{q_1\}, \{q_0, q_1\}, \{q_1, q_2, q_3\}, \dots$

Step della procedura

step

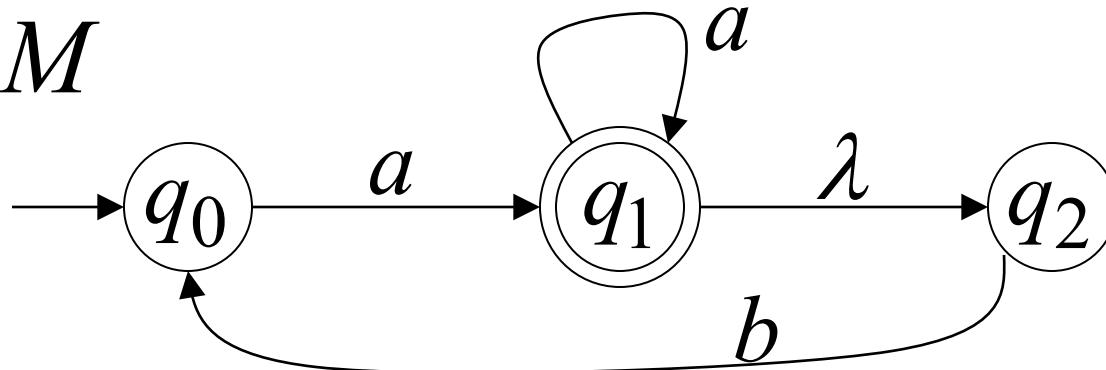
1. Stato iniziale NFA: q_0



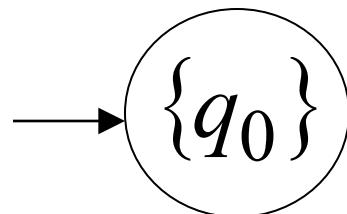
stato iniziale del DFA: $\{q_0\}$

esempio

NFA M



DFA M'



step

2. per ogni stato DFA $\{q_i, q_j, \dots, q_m\}$

calcolo nel NFA

$$\left. \begin{array}{l} \delta^*(q_i, a) \\ \cup \delta^*(q_j, a) \\ \dots \\ \cup \delta^*(q_m, a) \end{array} \right\} = \{q'_k, q'_l, \dots, q'_n\}$$

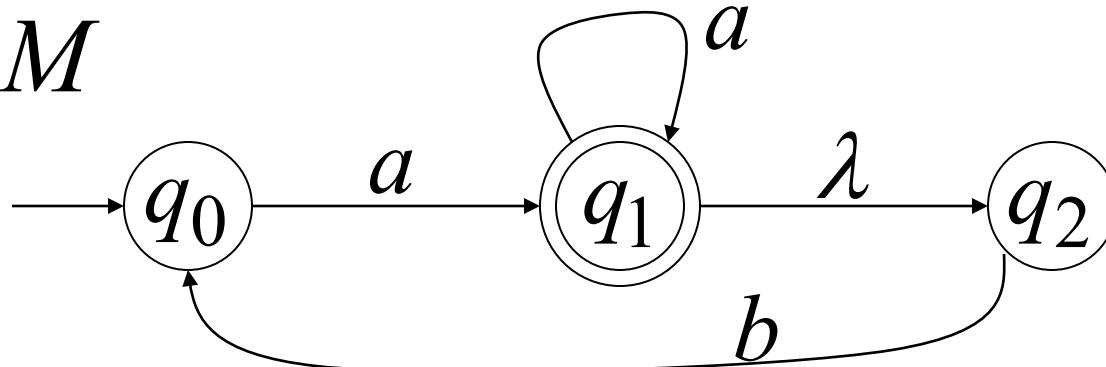
unione

addiziona questa nuova transizione al DFA

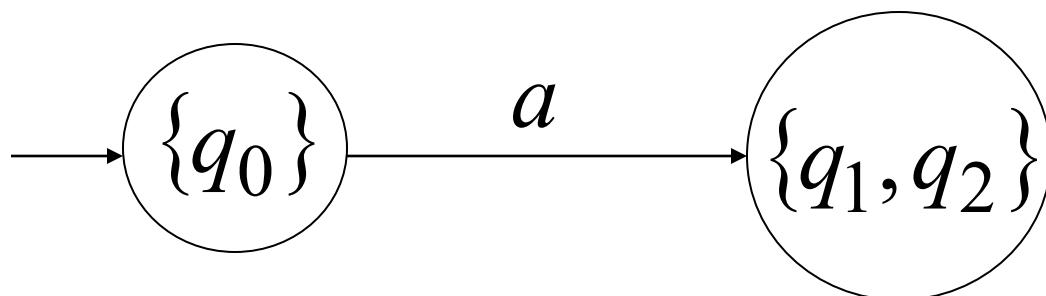
$$\delta(\{q_i, q_j, \dots, q_m\}, a) = \{q'_k, q'_l, \dots, q'_n\}$$

esempio $\delta^*(q_0, a) = \{q_1, q_2\}$

NFA M



DFA M' $\delta(\{q_0\}, a) = \{q_1, q_2\}$

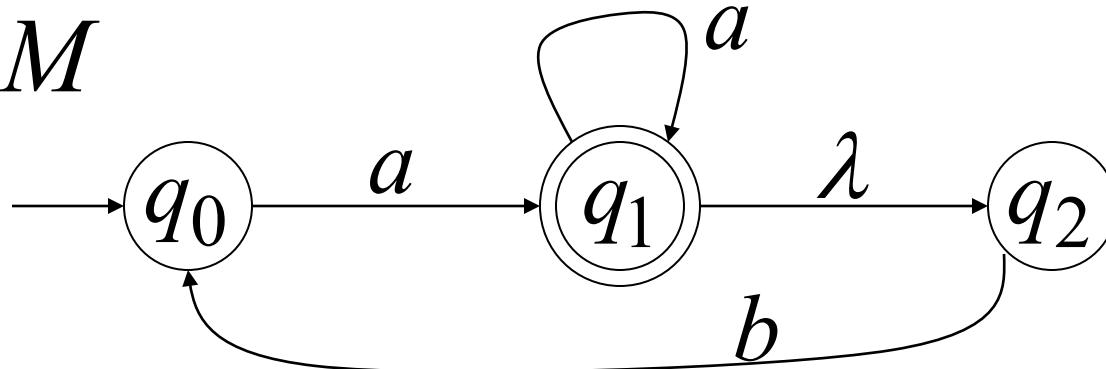


step

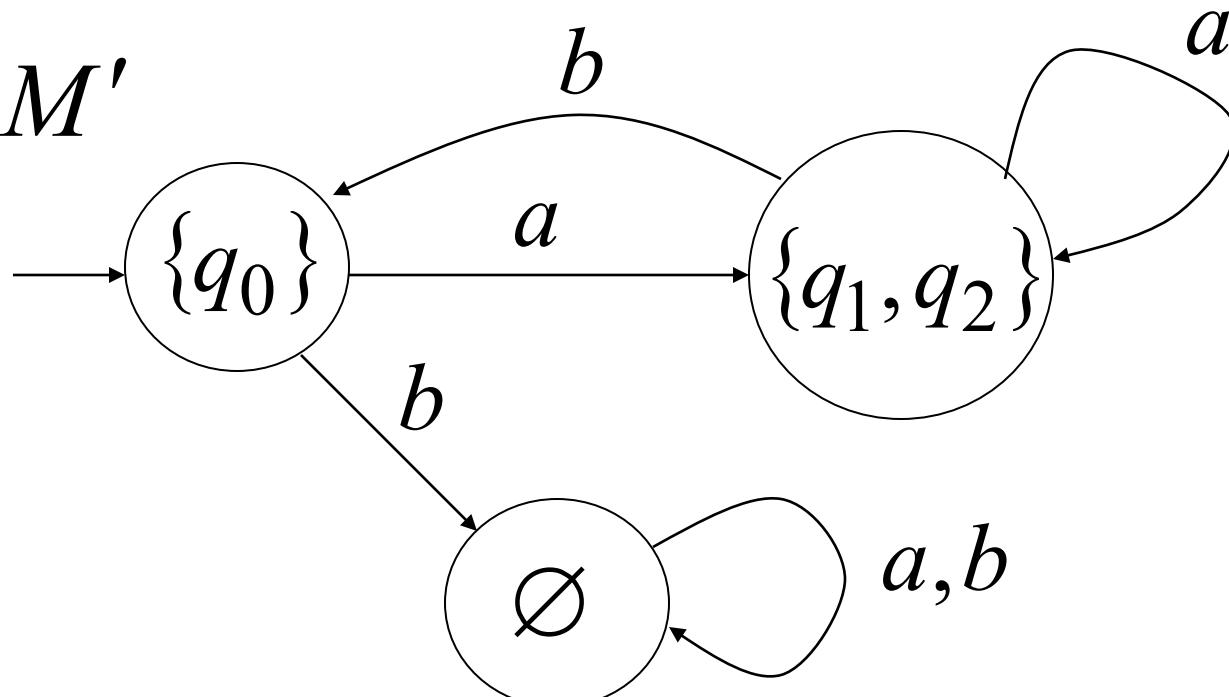
3. Ripeti lo step 2 per ogni stato nel DFA e simboli nell'alfabeto finchè non vi sono più stati che possono essere addizionati al DFA

esempio

NFA M



DFA M'



step

4.

$$\{q_i, q_j, \dots, q_m\}$$

Per ogni stato DFA

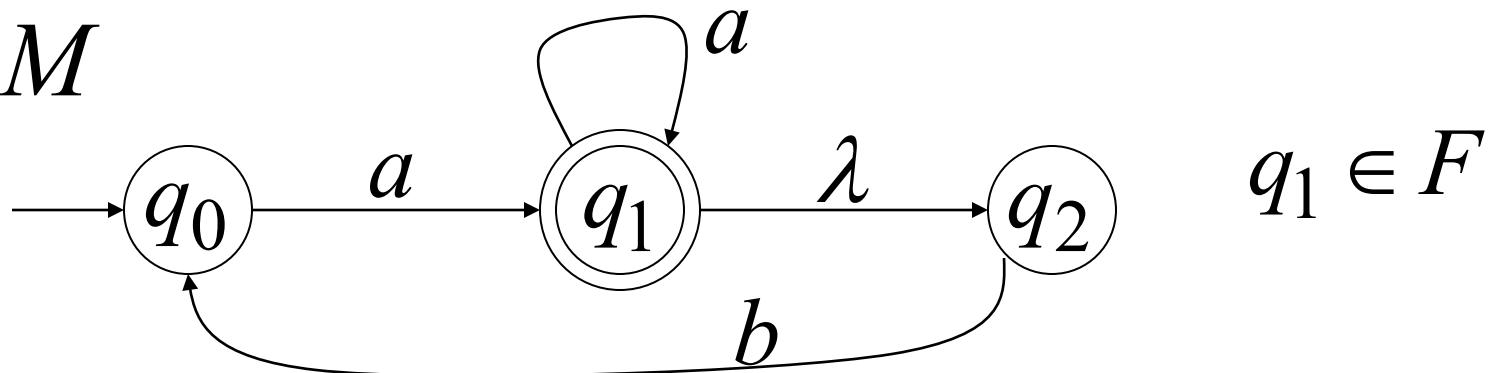
Se qualche q_j è uno stato di accettazione del NFA

Allora $\{q_i, q_j, \dots, q_m\}$

è uno stato di accettazione del DFA

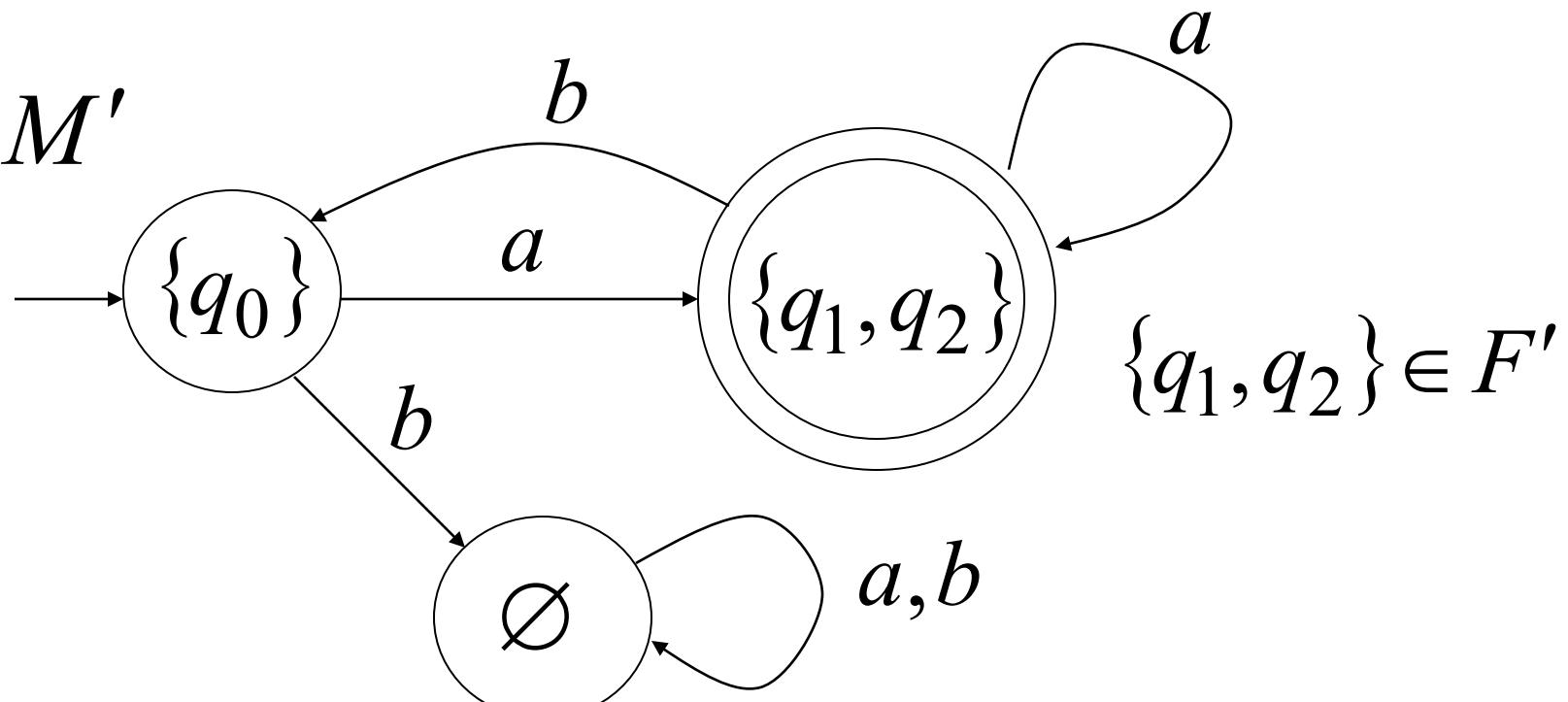
Example

NFA M



$$q_1 \in F$$

DFA M'

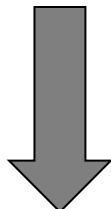


$$\{q_1, q_2\} \in F'$$

Step della procedura

step

1. Stato iniziale NFA: q_0



stato iniziale del DFA: $\{q_0\}$

step

2. per ogni stato DFA $\{q_i, q_j, \dots, q_m\}$

calcolo nel NFA

$$\left. \begin{array}{l} \delta^*(q_i, a) \\ \cup \delta^*(q_j, a) \\ \dots \\ \cup \delta^*(q_m, a) \end{array} \right\} = \text{unione } \{q'_k, q'_l, \dots, q'_n\}$$

addiziona questa nuova transizione al DFA

$$\delta(\{q_i, q_j, \dots, q_m\}, a) = \{q'_k, q'_l, \dots, q'_n\}$$

step

3. Ripeti lo step 2 per ogni stato nel DFA e simboli nell'alfabeto finchè non vi sono più stati che possono essere addizionati al DFA

step

4. Per ogni stato del DFA $\{q_i, q_j, \dots, q_m\}$

se è presente uno stato q_j finale,
accettante, del NFA

allora, $\{q_i, q_j, \dots, q_m\}$

è uno stato accettante del DFA

Lemma:

Se traduciamo un NFA M in un DFA M'

Allora i due automata sono equivalenti:

$$L(M) = L(M')$$

dimostrazione:

Dobbiamo dimostrare che: $L(M) \subseteq L(M')$

$$L(M) \stackrel{e}{\supseteq} L(M')$$

Mostriamo che: $L(M) \subseteq L(M')$

NFA contenuto in DFA

Dobbiamo provare che:

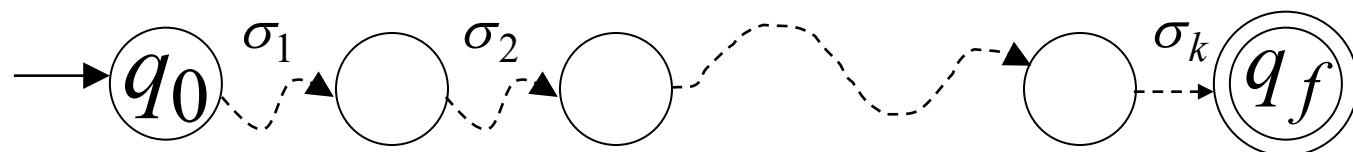
$$w \in L(M) \quad \longrightarrow \quad w \in L(M')$$

considera $w \in L(M)$ NFA



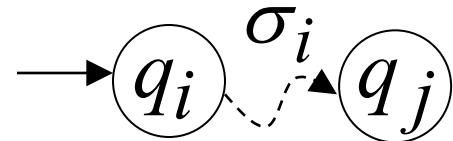
simboli

$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



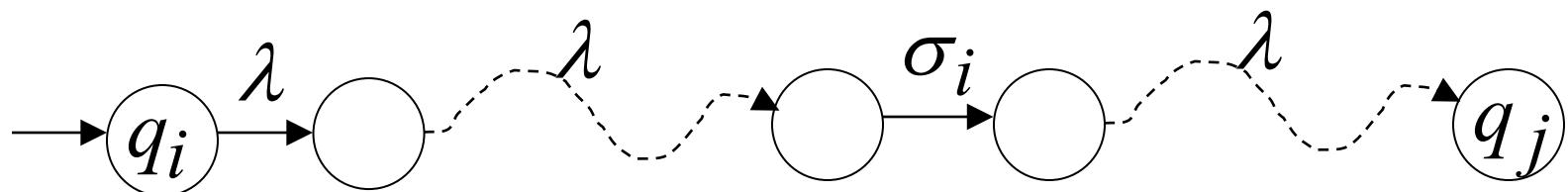
Simboli, Ing 1

ricordiamo



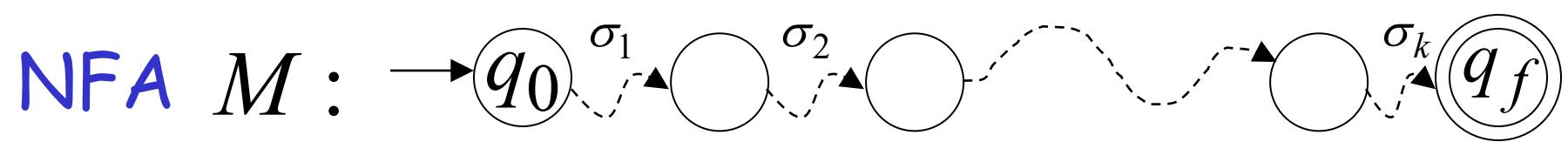
Denota un sotto cammino tale che

simboli

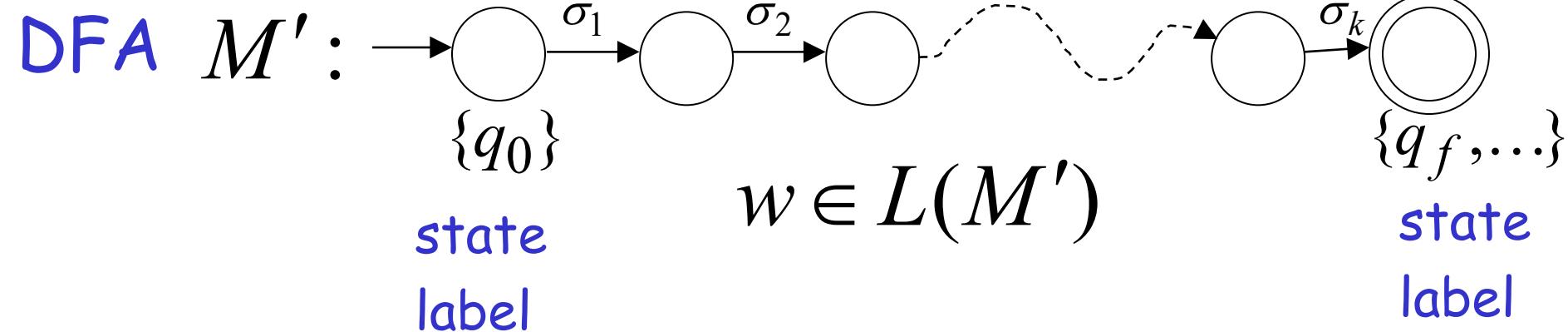


Mostriamo che se $w \in L(M)$

$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$

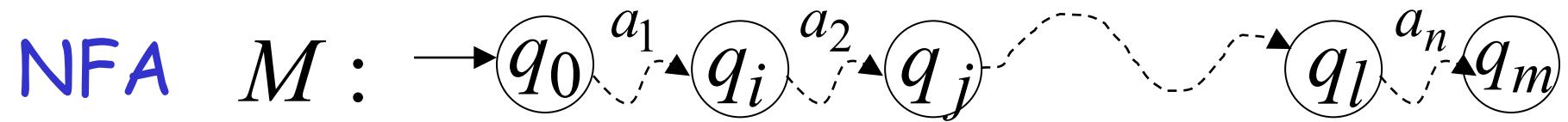


allora

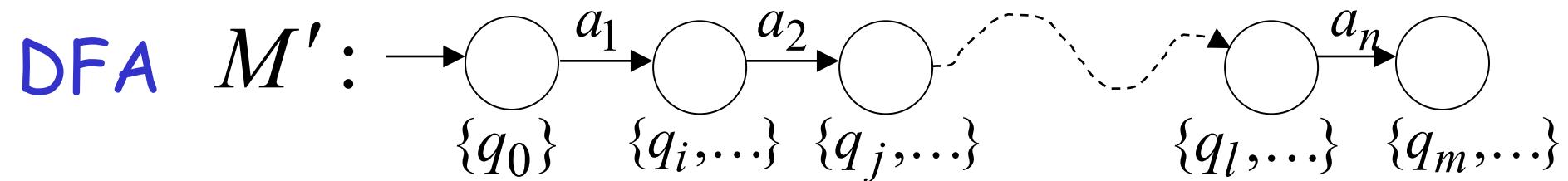


In modo piu generale ,
mostreremo che se in M :

(stringa arbitraria) $\mathcal{V} = a_1 a_2 \cdots a_n$

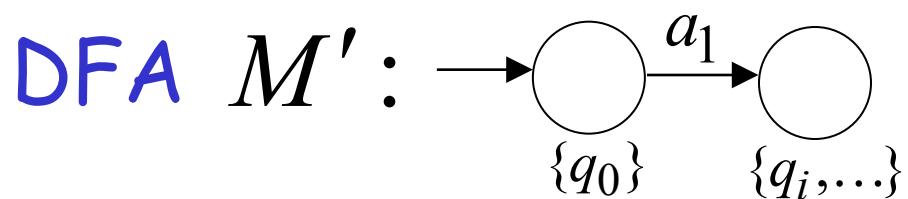
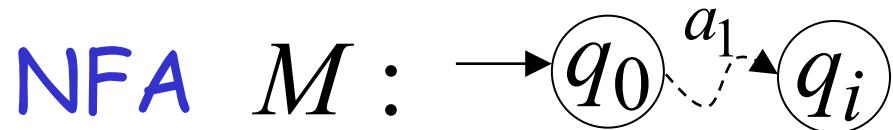


allora



Dimostrazione per induzione su $|v|$

Base induzione: $|v| = 1$ $v = a_1$

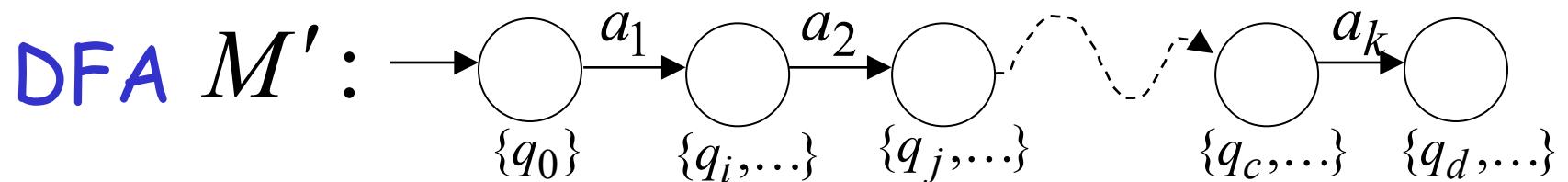
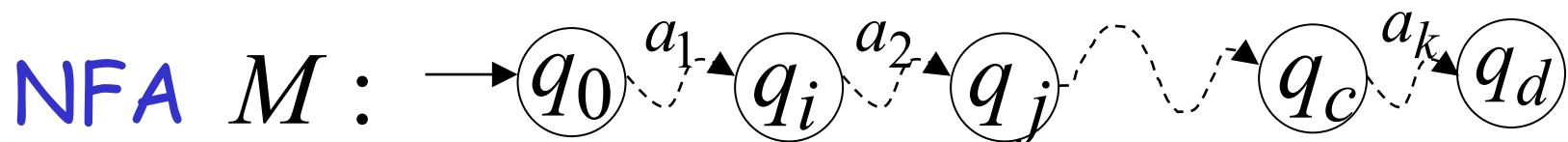


[vero per come costruito M']

Ipotesi induttiva: $1 \leq v \leq k$

$$v = a_1 a_2 \cdots a_k$$

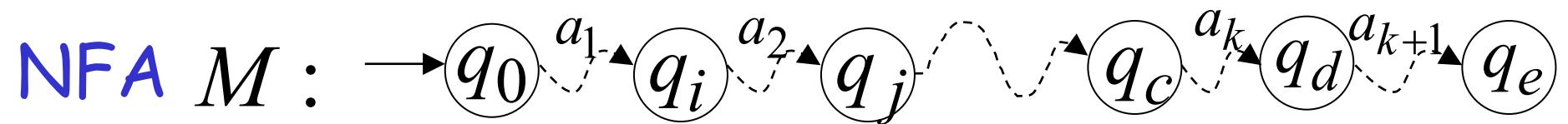
Supponiamo valga



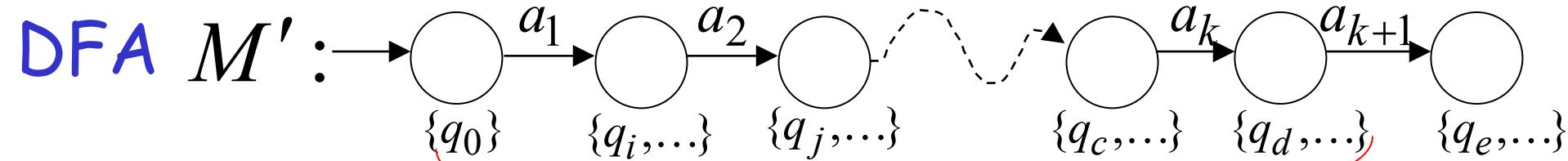
Step induttivo: $|v| = k + 1$

$$v = \underbrace{a_1 a_2 \cdots a_k}_{v'} a_{k+1} = v' a_{k+1}$$

Vero per costruzione di M'



v'

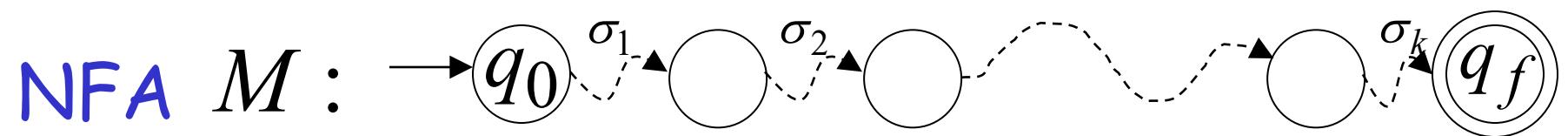


v'

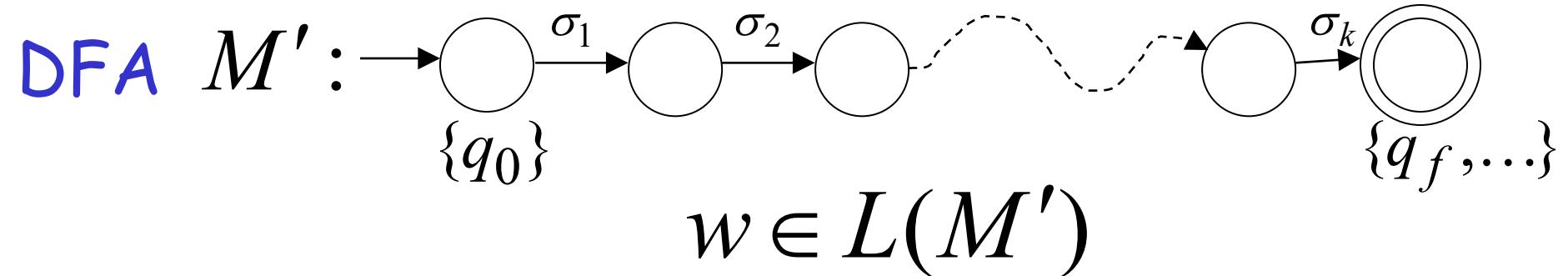
Quindi se

$$w \in L(M)$$

$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



allora



allora: $L(M) \subseteq L(M')$ dimostrato

e $L(M) \supseteq L(M')$ banale

quindi: $L(M) = L(M')$

Fine lemma

Espressioni regolari

Ricordo: un linguaggio è regolare
se è riconosciuto da un NFA
(=DFA)

Definizione sintattica

L'espressioni regolari di base : \emptyset , λ , α

Date le espressioni regolari r_1 e r_2

$$\left. \begin{array}{l} r_1 + r_2 \\ r_1 \cdot r_2 \\ r_1^* \\ (r_1) \end{array} \right\}$$

Sono espressioni regolari

Una semantica: Linguaggi associati alle espressioni regolari

Per le espressioni regolari di base:

$$L(\emptyset) = \emptyset$$

$$L(\lambda) = \{\lambda\}$$

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

passo

per le espressioni regolari r_1 e r_2

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1) L(r_2)$$

$$L(r_1^*) = (L(r_1))^*$$

$$L((r_1)) = L(r_1)$$

Linguaggi associati alle espressioni regolari

$L(r)$: linguaggio associato all'espressione r

esempio

$$L((a + b \cdot c)^*) = \{\lambda, a, bc, aa, abc, bca, \dots\}$$

Espressioni regolari definiscono solo e
soltanto i linguaggi regolari?

proprietà dei linguaggi regolari e automi

per linguaggi regolari L_1 e L_2
dimostreremo che:

Unione: $L_1 \cup L_2$

Concatenazione: $L_1 L_2$

Star: L_1^*

Reversal: L_1^R

Complemento: $\overline{L_1}$

Intersezione: $L_1 \cap L_2$

sono linguaggi
regolari

diremo: linguaggi regolari sono **chiusi sotto**

Unione: $L_1 \cup L_2$

Concatenazione: $L_1 L_2$

Star: L_1^*

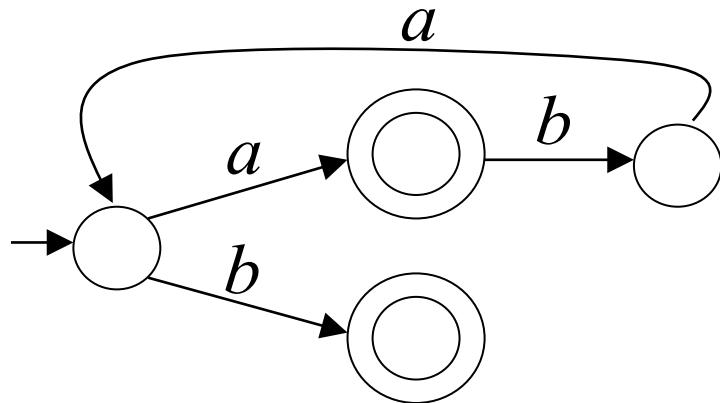
Reversal: L_1^R

Complemento: $\overline{L_1}$

Intersezione: $L_1 \cap L_2$

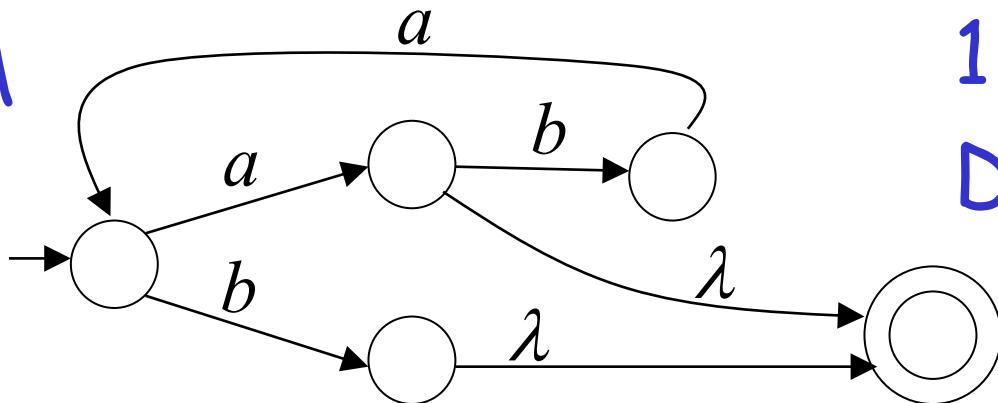
Useremo nfa con un solo stato finale

NFA



2 stati di
accettazione

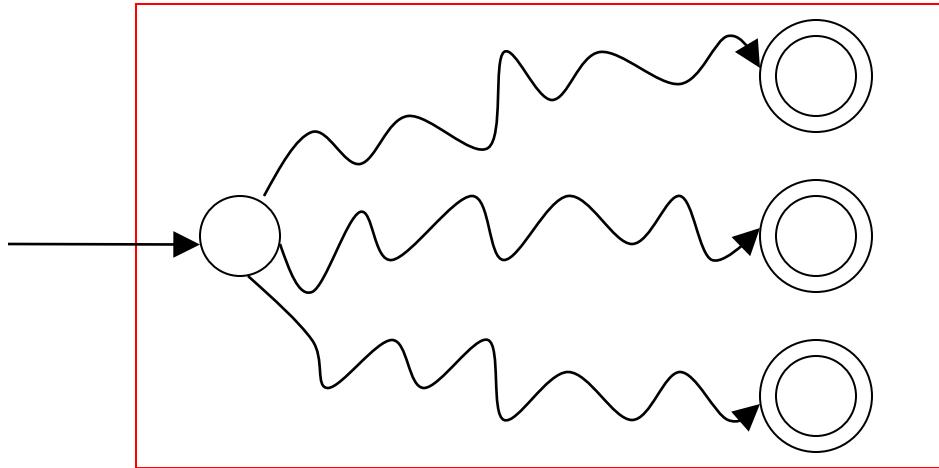
Equivalent
NFA



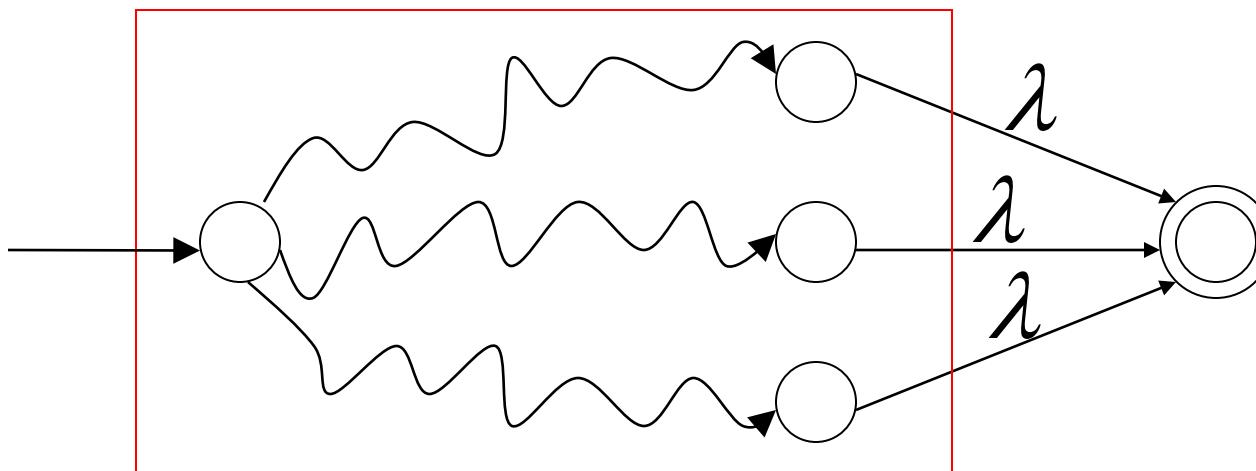
1 solo stato
Di accettazione

In Generale

NFA



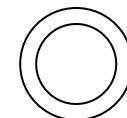
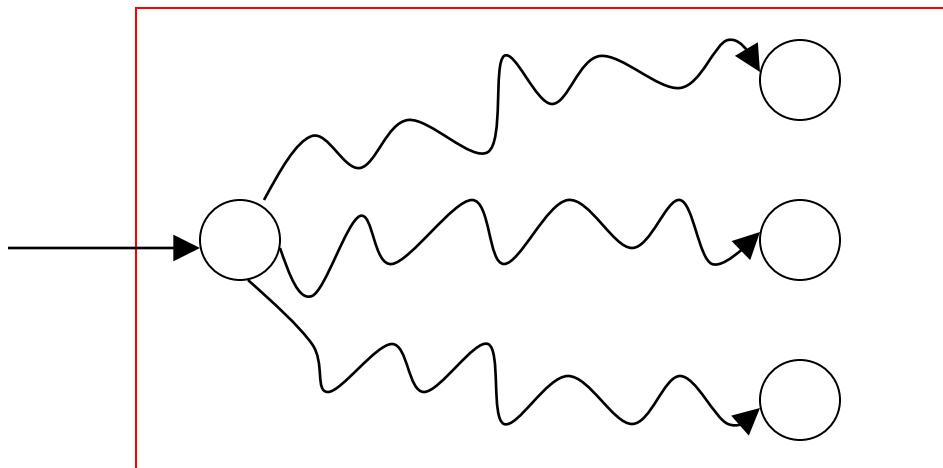
Equivalent NFA



Un solo
Stato di
accettazione

Caso estremo

NFA senza stato di accettazione



Addizioniamo
Uno stato di
Accettazione
Senza transizione

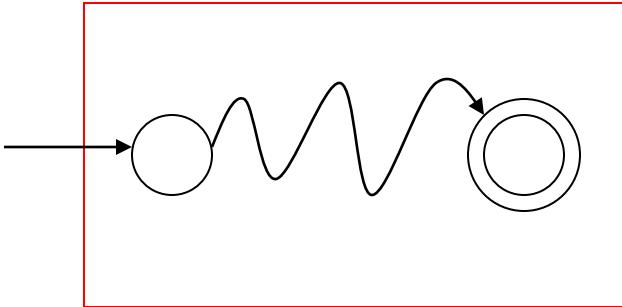
Prendiamo due linguaggi

Linguaggio regolare L_1 linguaggio regolare L_2

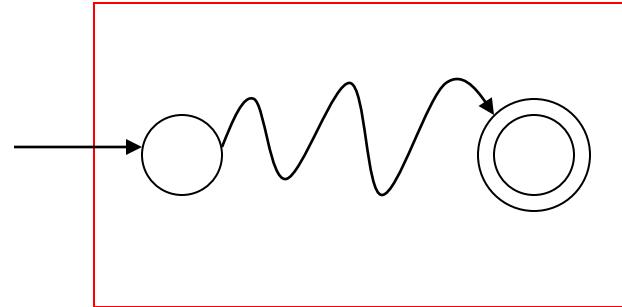
$$L(M_1) = L_1$$

$$L(M_2) = L_2$$

NFA M_1



NFA M_2



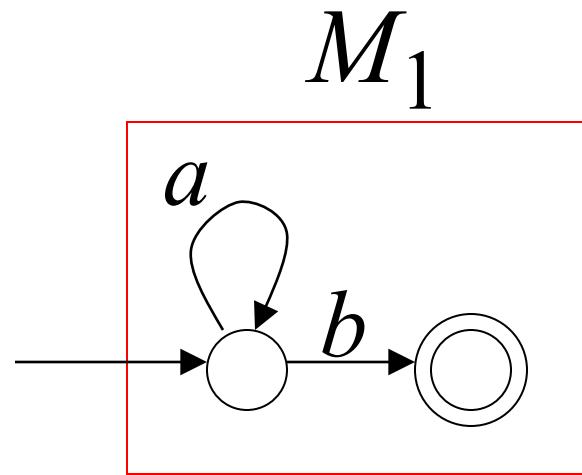
Un solo stato di accettazione

Un solo stato di accettazione

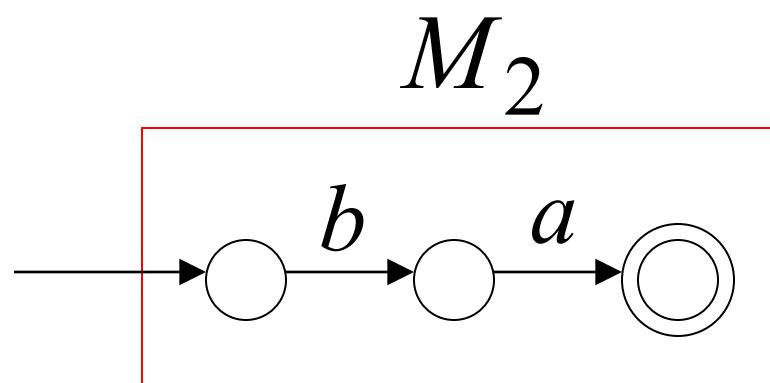
Esempio

$$n \geq 0$$

$$L_1 = \{a^n b\}$$

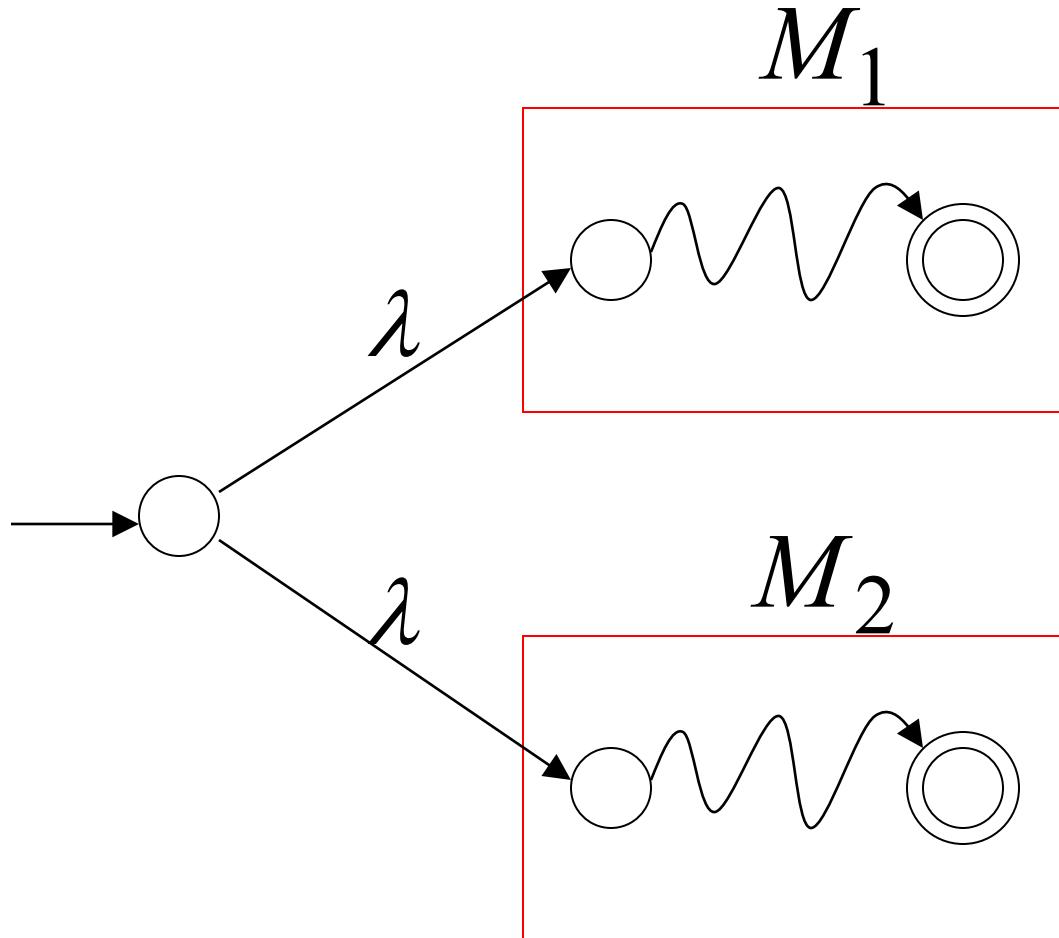


$$L_2 = \{ba\}$$



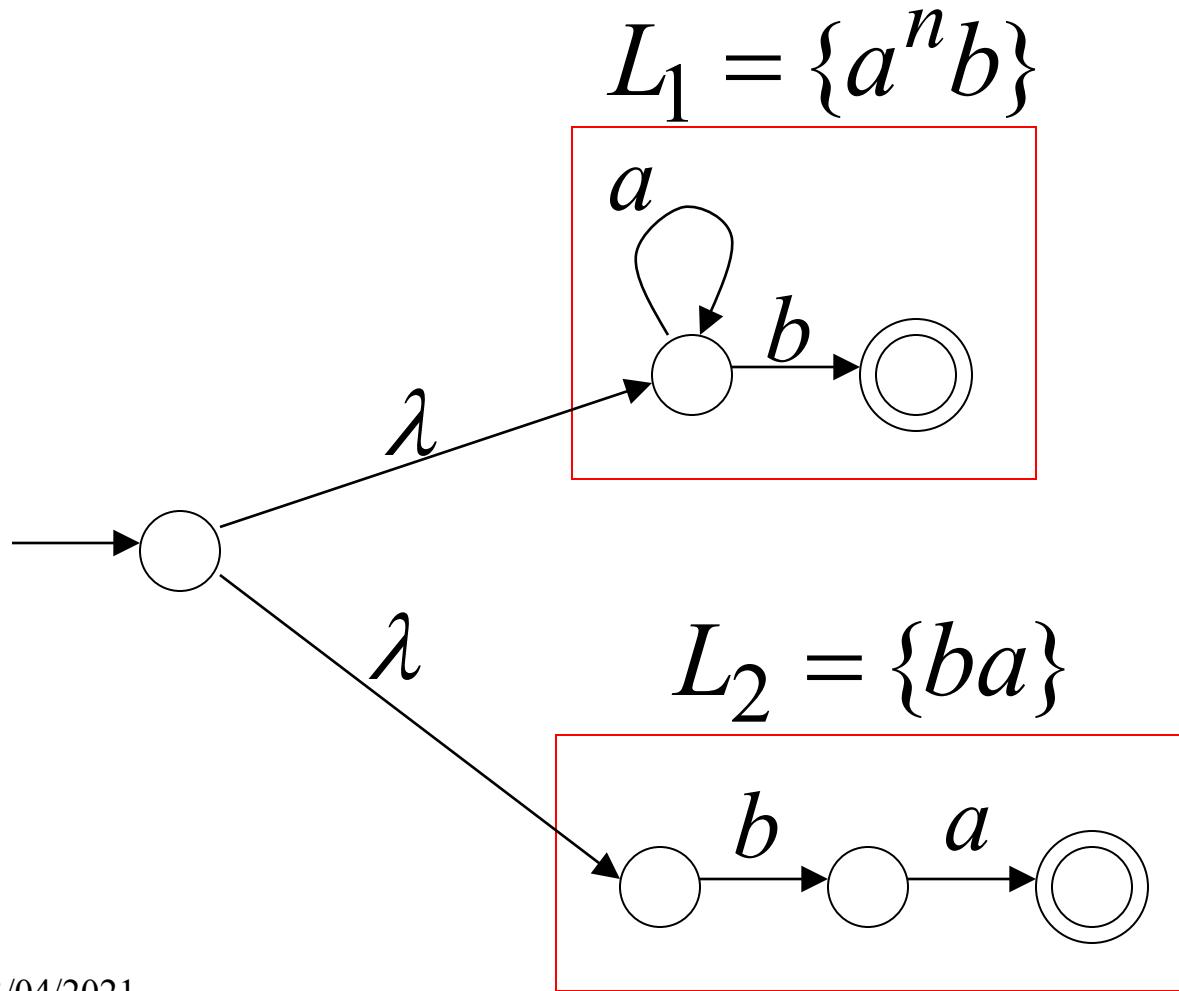
Unione

NFA per $L_1 \cup L_2$



Esempio

NFA per $L_1 \cup L_2 = \{a^n b\} \cup \{ba\}$



Evitiamo le transizioni con le lambda transizioni.

Mostriamo che a partire da due automi (N_1, N_2) , si può costruire l'automa unione dei due linguaggi definiti dagli automi precedenti.

Gli stati del nuovo automa sono l'unione degli stati precedenti, K_1 e K_2 , più un nuovo stato iniziale q'_0 .

Funzione transizione
dell'automa unione, N , a
partire dalle delta di
 N_1 e N_2 .

$$\delta_N(q, a) = \delta_{N_1}(q, a), q \in K_1, a \in \Sigma_1$$

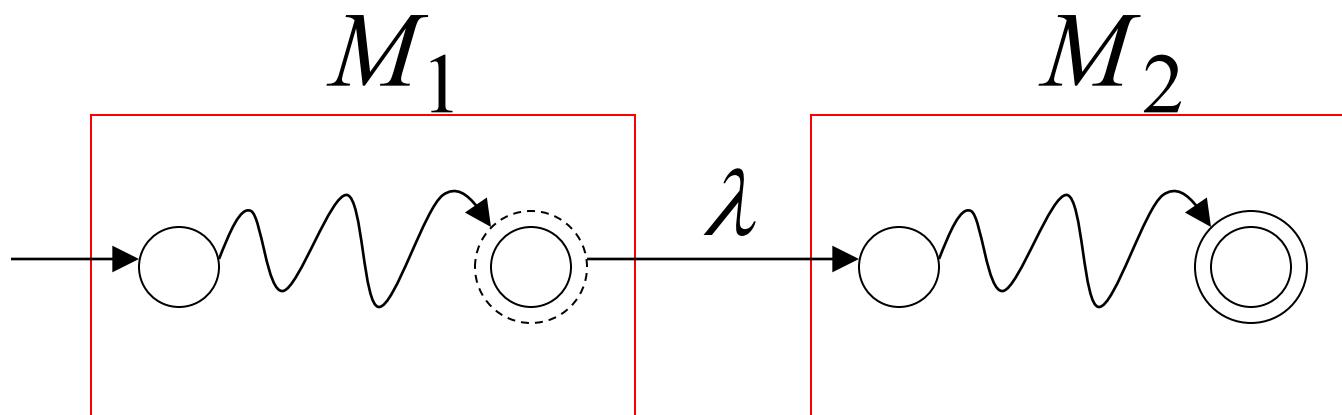
$$\delta_N(q, a) = \delta_{N_2}(q, a), q \in K_2, a \in \Sigma_2$$

$$\delta_N(q'_0, a) = \delta_{N_1}(q_{0_1}, a) \cup \delta_{N_2}(q_{0_2}, a), a \in \Sigma$$

Provare che la
definizione
precedente
definisce l'unione
di due automi.

Concatenazione

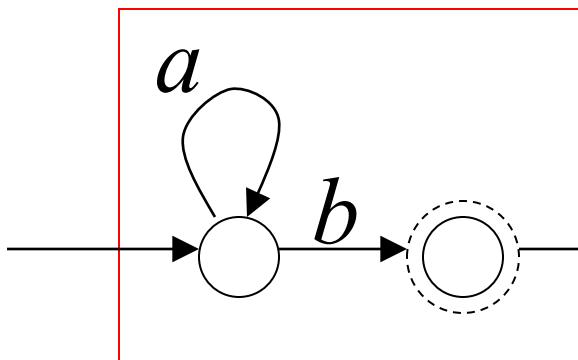
NFA per $L_1 L_2$



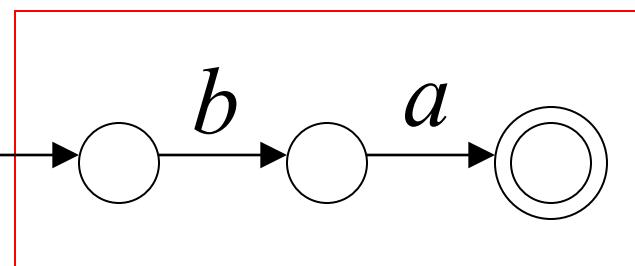
esempio

NFA per $L_1 L_2 = \{a^n b\} \{ba\} = \{a^n bba\}$

$$L_1 = \{a^n b\}$$



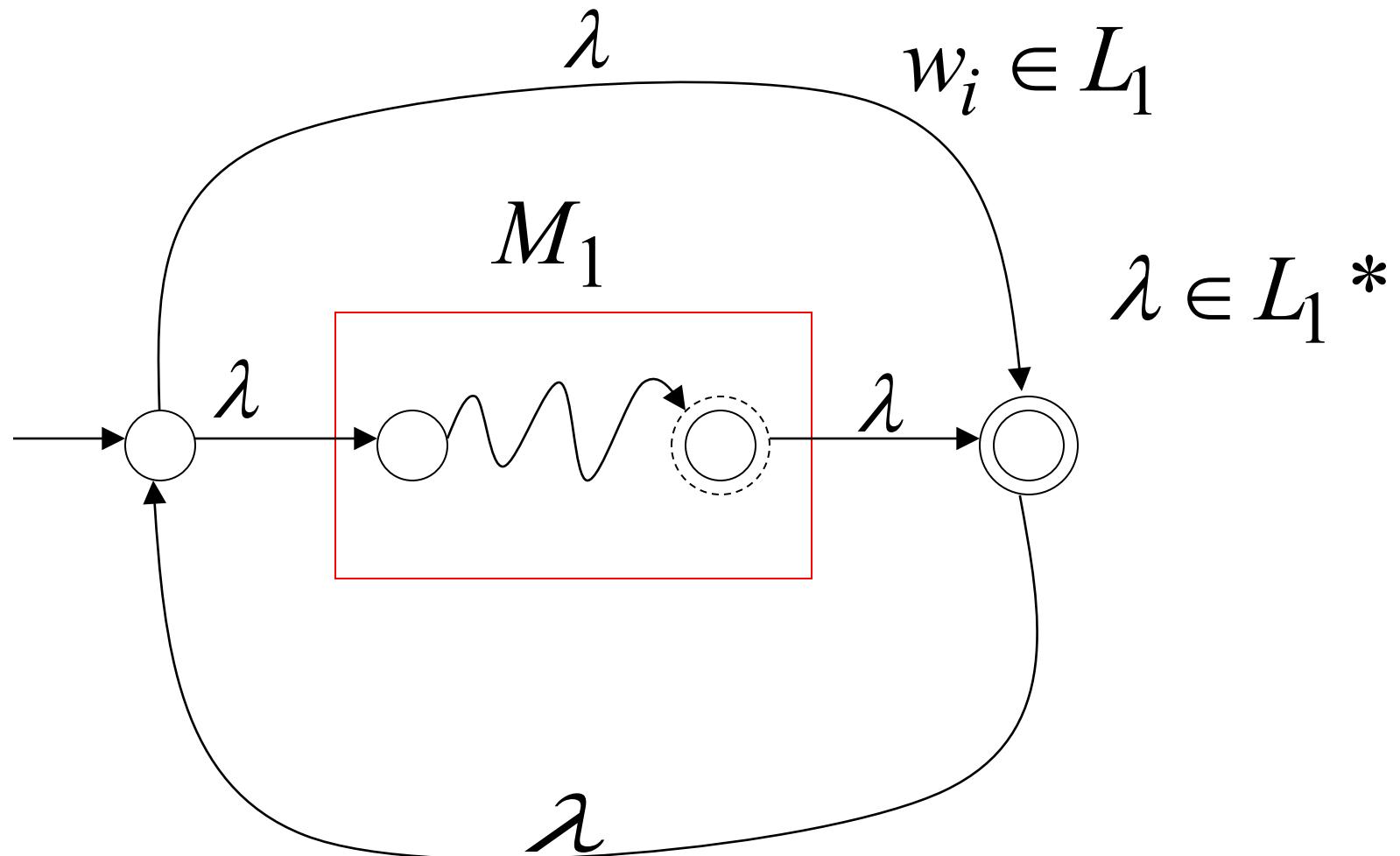
$$L_2 = \{ba\}$$



Star

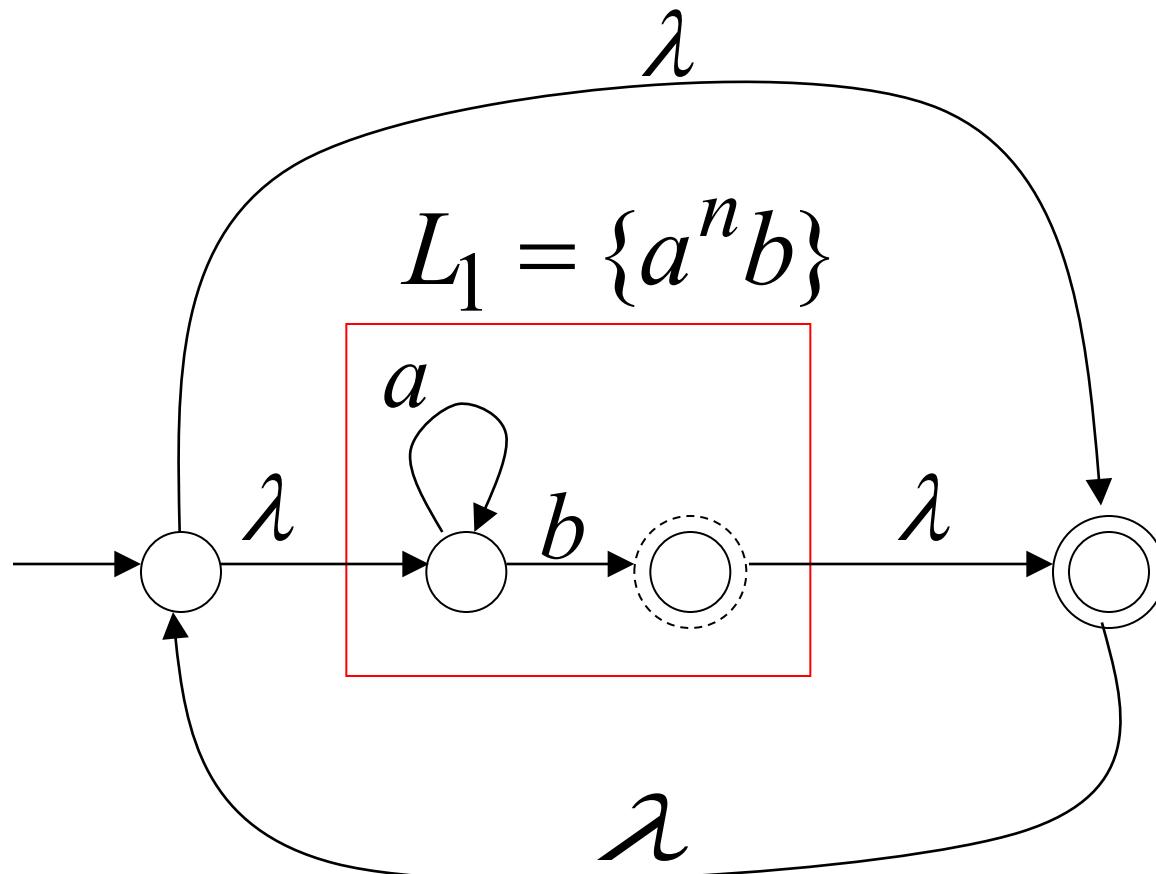
NFA per L_1^*

$w = w_1 w_2 \cdots w_k$



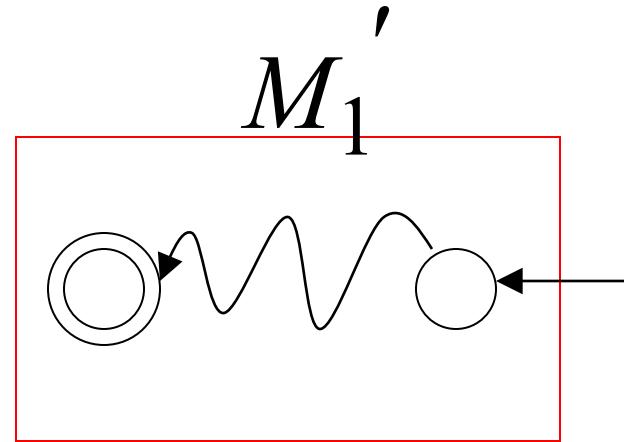
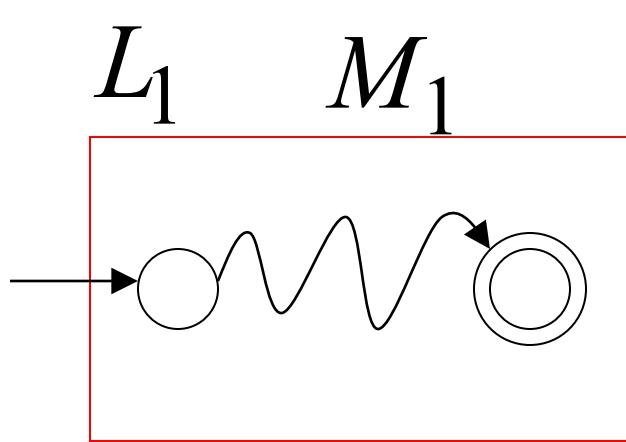
esempio

NFA per $L_1^* = \{a^n b\}^*$



Reverse

NFA per L_1^R

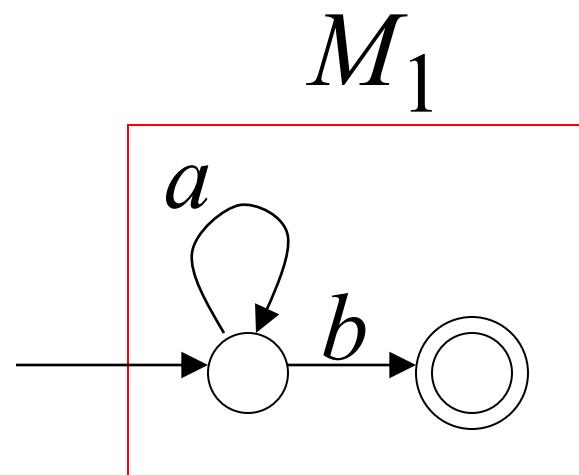


1. Reverse tutte le transizioni

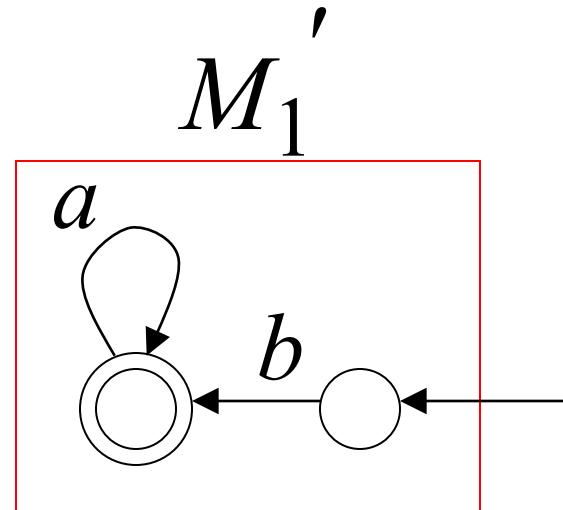
2. Stato iniziale quello finale,
quello finale stato iniziale

esempio

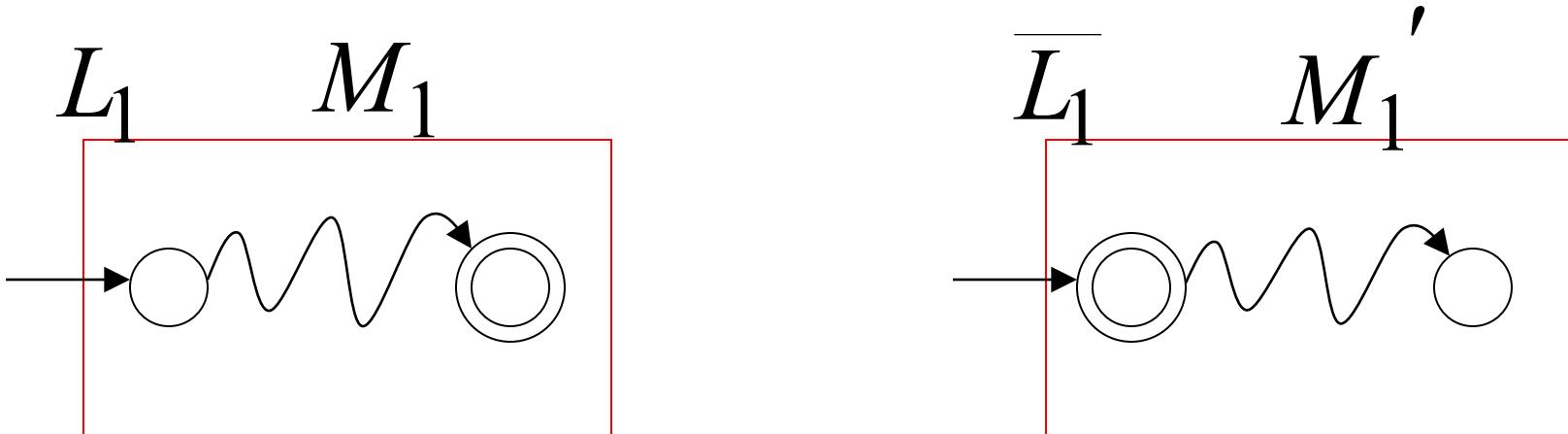
$$L_1 = \{a^n b\}$$



$${L_1}^R = \{ba^n\}$$



Complemento

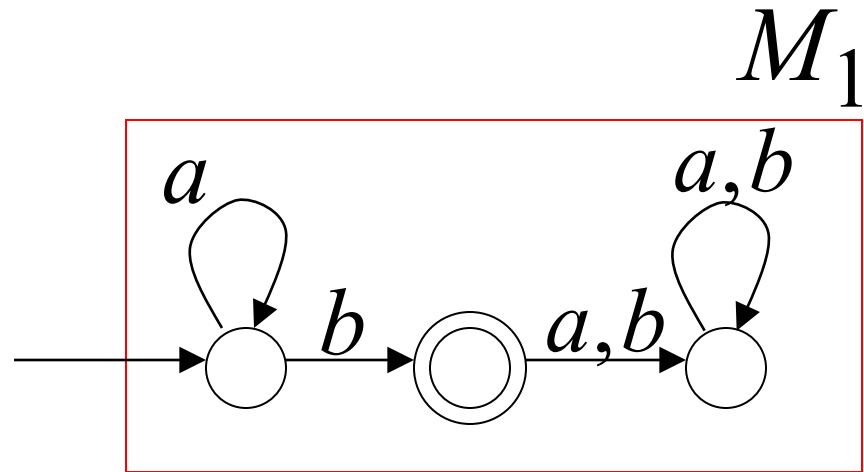


prendiamo il **DFA** che accetta L_1

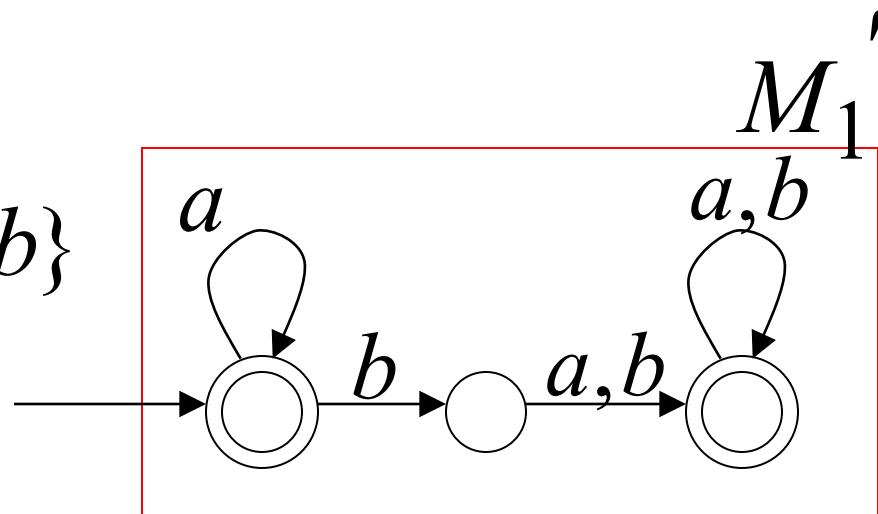
1. Stati non finale diventano finale,
e vice-versa, resta lo stato iniziale.

esempio

$$L_1 = \{a^n b\}$$



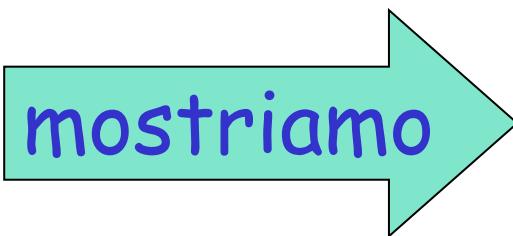
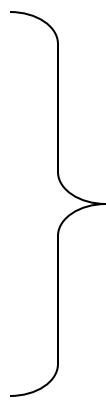
$$\overline{L_1} = \{a,b\}^* - \{a^n b\}$$



Intersezione

L_1 regolari

L_2 regolari



$L_1 \cap L_2$

regolari

leggi DeMorgan : $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

L_1 , L_2 regolari

→ $\overline{L_1}$, $\overline{L_2}$ regolari

→ $\overline{L_1} \cup \overline{L_2}$ regolari

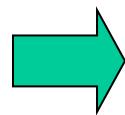
→ $\overline{\overline{L_1} \cup \overline{L_2}}$ regolari

→ $L_1 \cap L_2$ regolari

esempio

$$L_1 = \{a^n b\} \quad \text{regolari}$$

$$L_2 = \{ab, ba\} \quad \text{regolari}$$



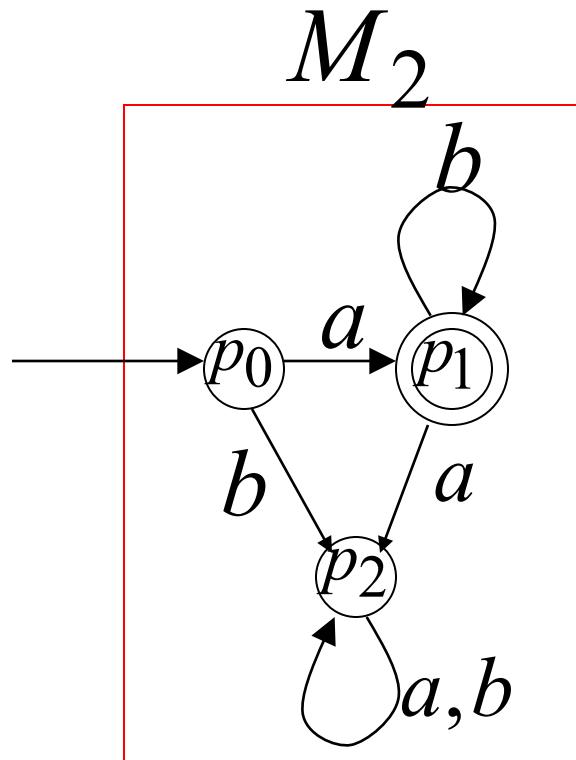
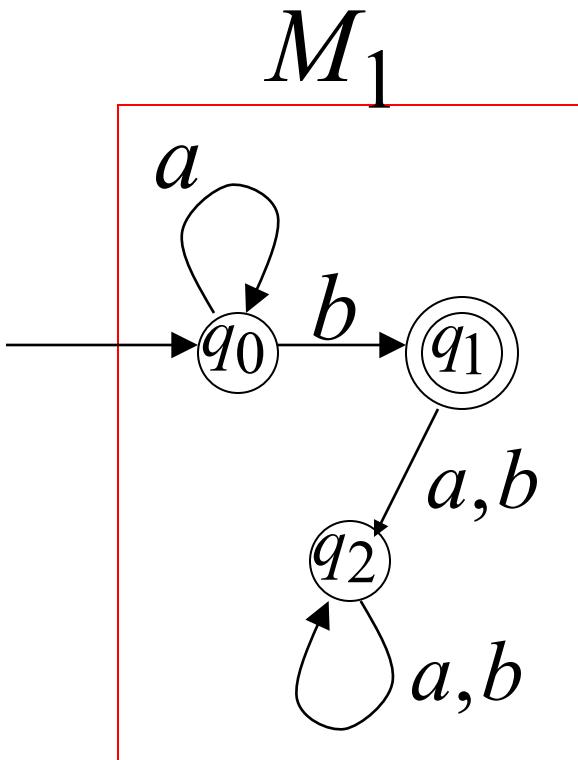
$$L_1 \cap L_2 = \{ab\}$$

regolari

esempio:

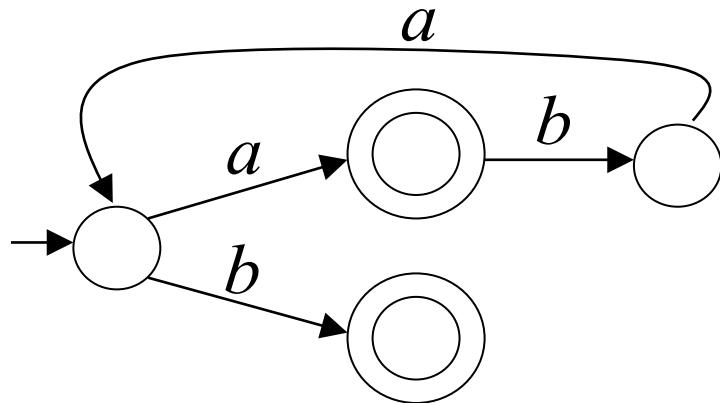
$$L_1 = \{a^n b\} \quad n \geq 0$$

$$L_2 = \{ab^m\} \quad m \geq 0$$



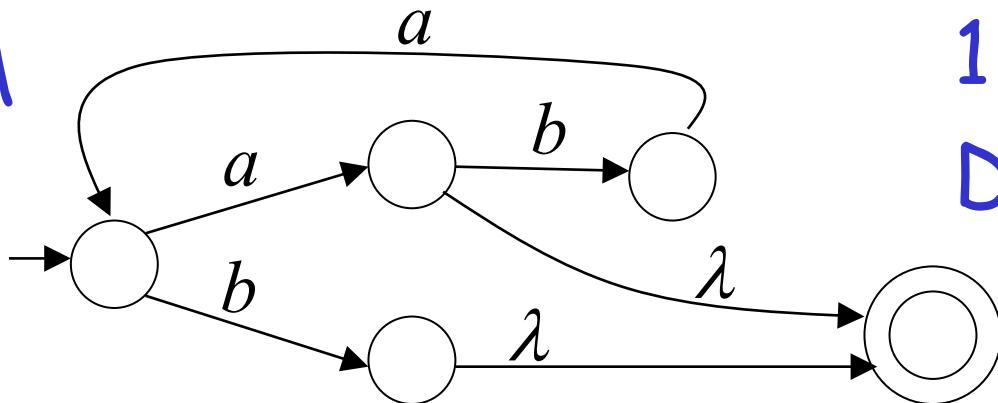
Useremo nfa con un solo stato finale

NFA



2 stati di
accettazione

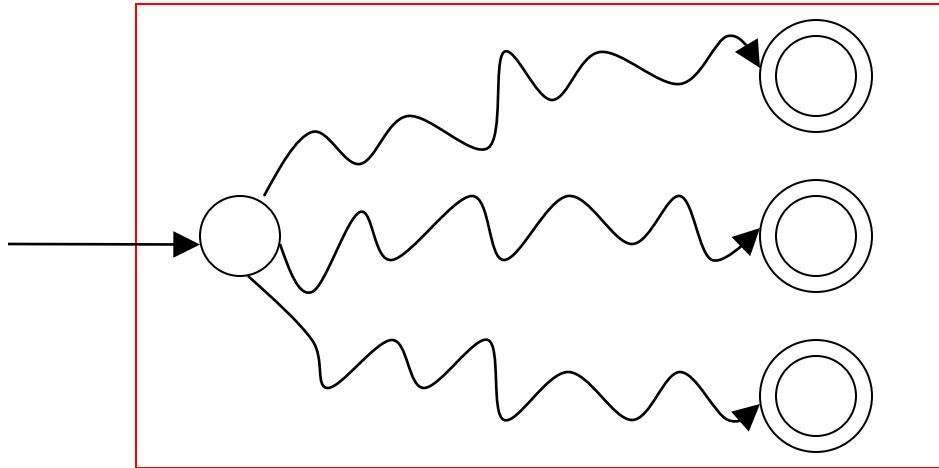
Equivalent
NFA



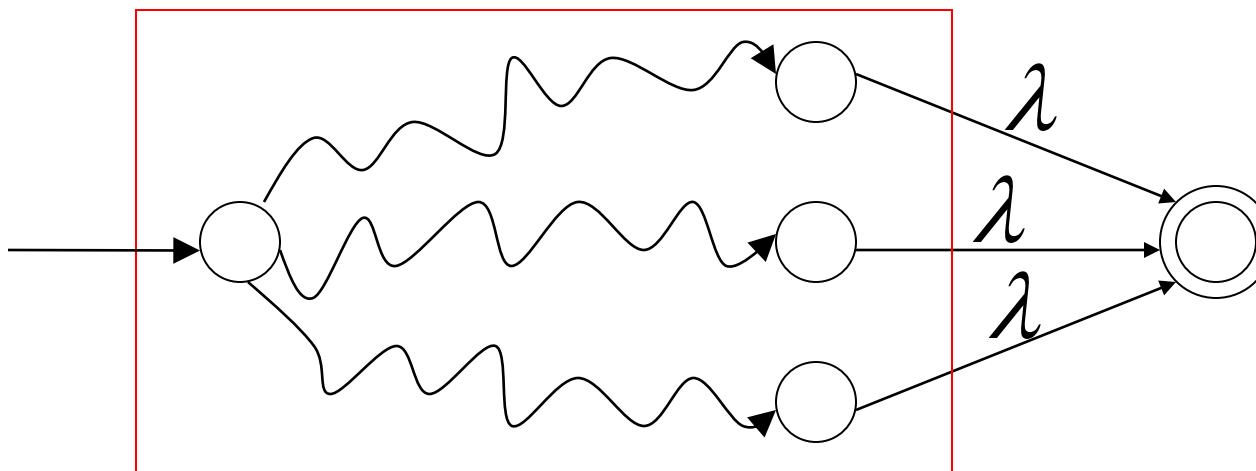
1 solo stato
Di accettazione

In Generale

NFA

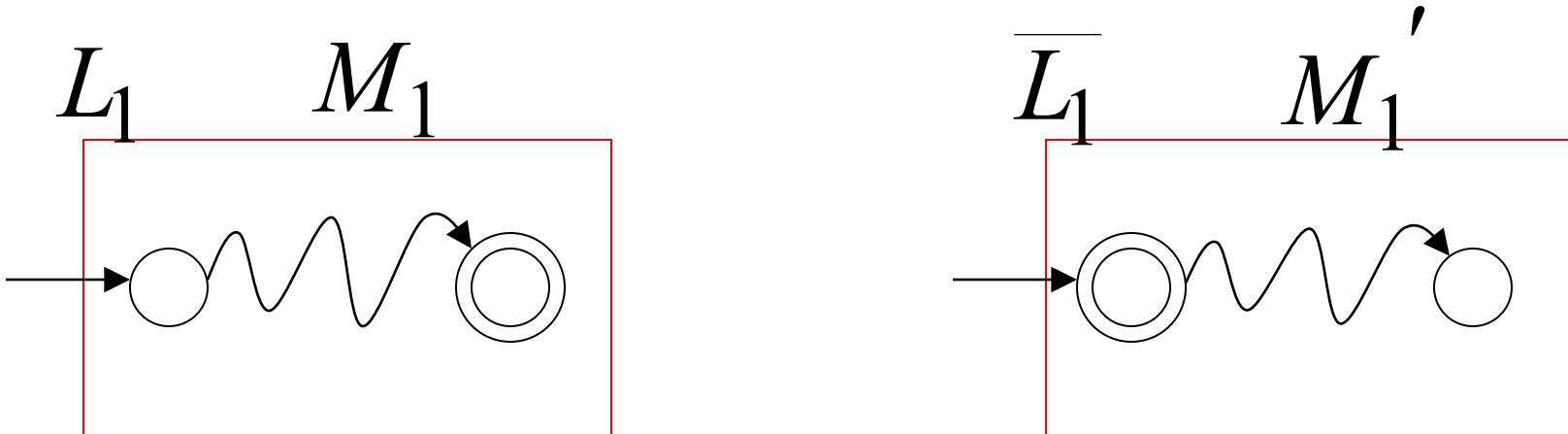


Equivalent NFA



Un solo
Stato di
accettazione

Complemento



1 prendiamo il **DFA** che accetta L_1

2. Stati non finale diventano finale,
e vice-versa

Chiusura rispetto intersezione

macchina M_1

DFA per L_1

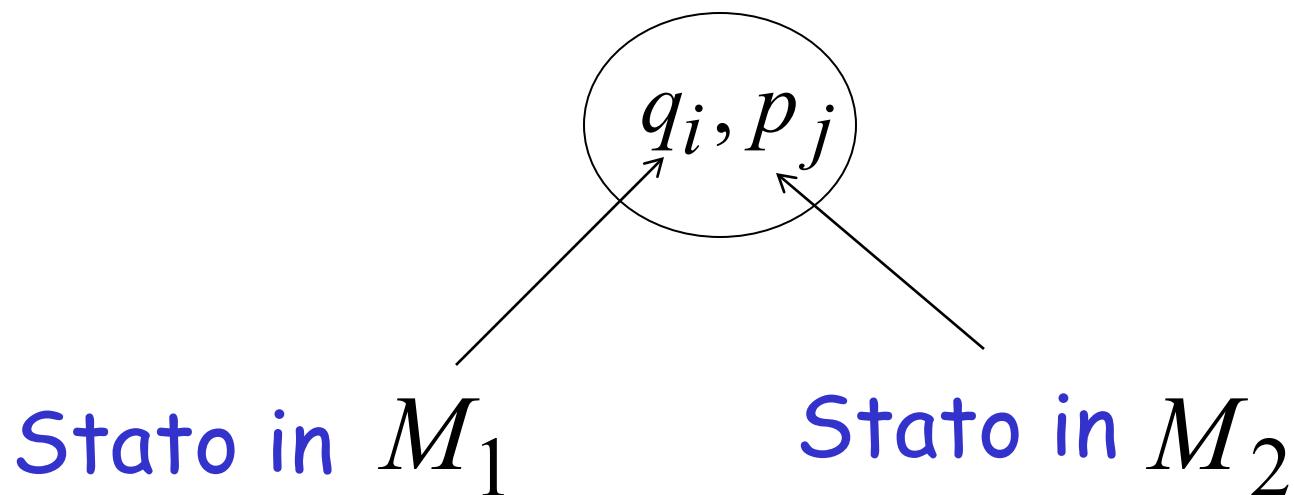
macchina M_2

DFA per L_2

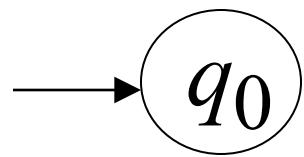
Costruiamo un DFA M che accetta $L_1 \cap L_2$

M Simula in parallelo M_1 e M_2

Stati in M

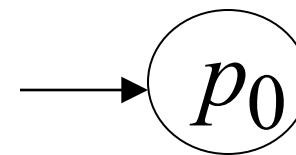


DFA M_1



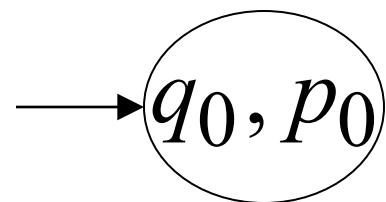
stato iniziale

DFA M_2



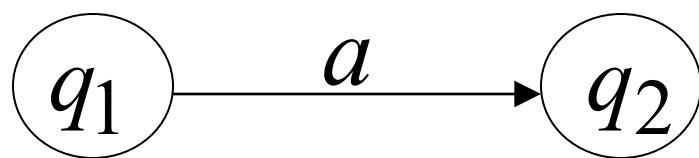
stato iniziale

DFA M



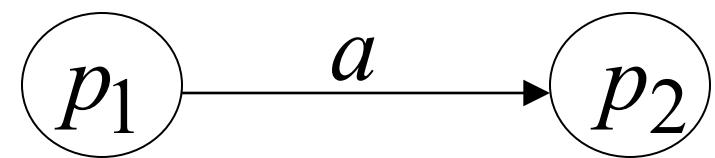
nuovo stato iniziale

DFA M_1

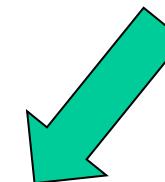
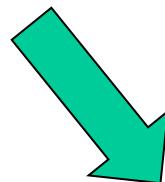


transizione

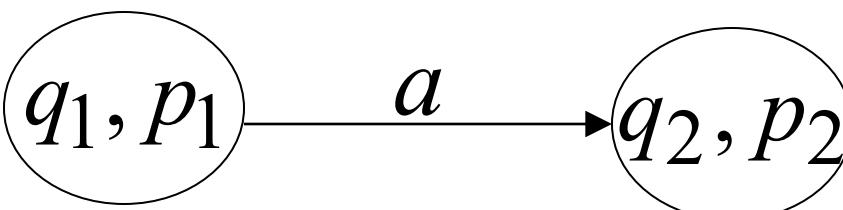
DFA M_2



transizione

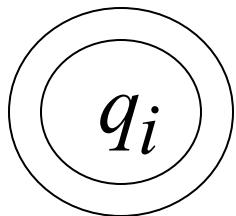


DFA M



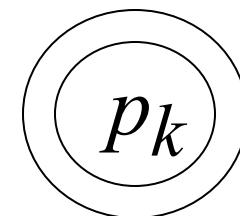
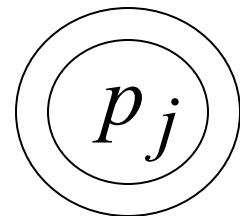
Nuova transizione

DFA M_1



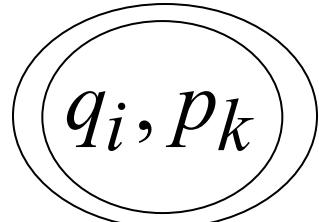
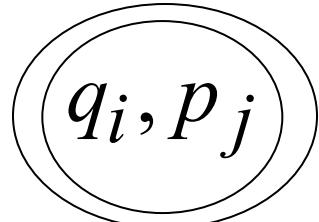
accettazione stato

DFA M_2



accettazione stati

DFA M



nuovo accettazione stati

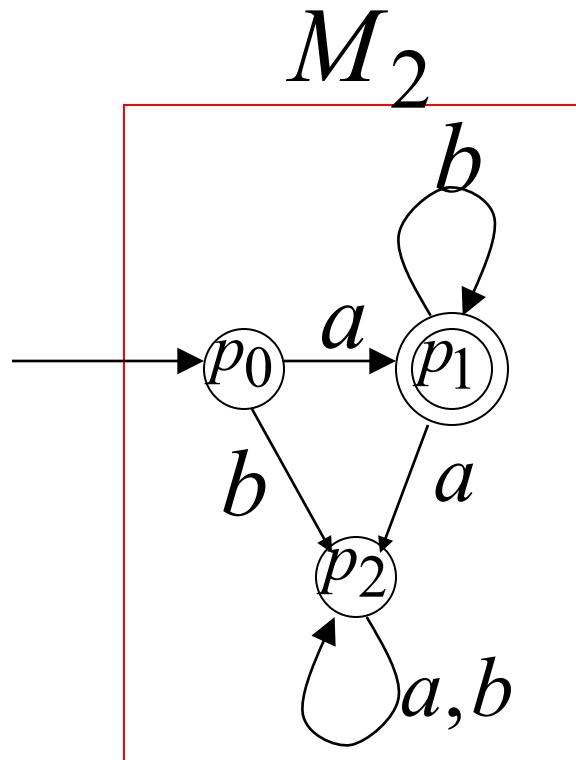
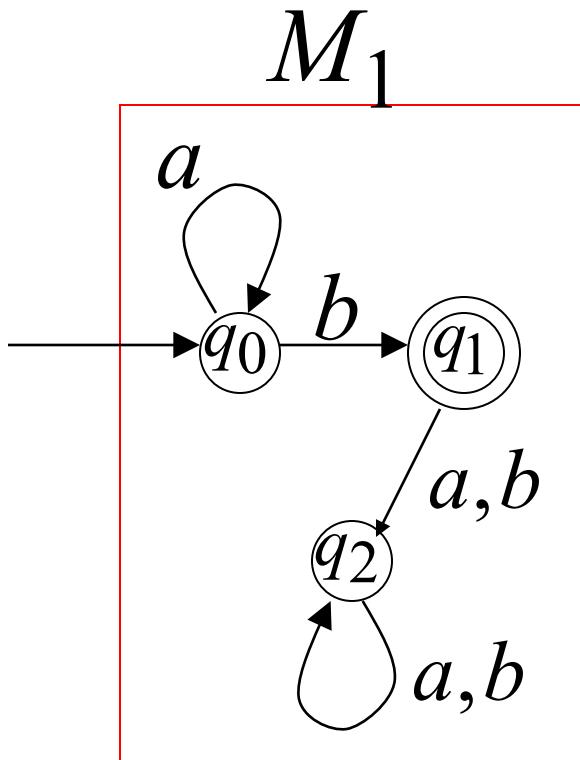
esempio:

$$n \geq 0$$
$$L_1 = \{a^n b\}$$



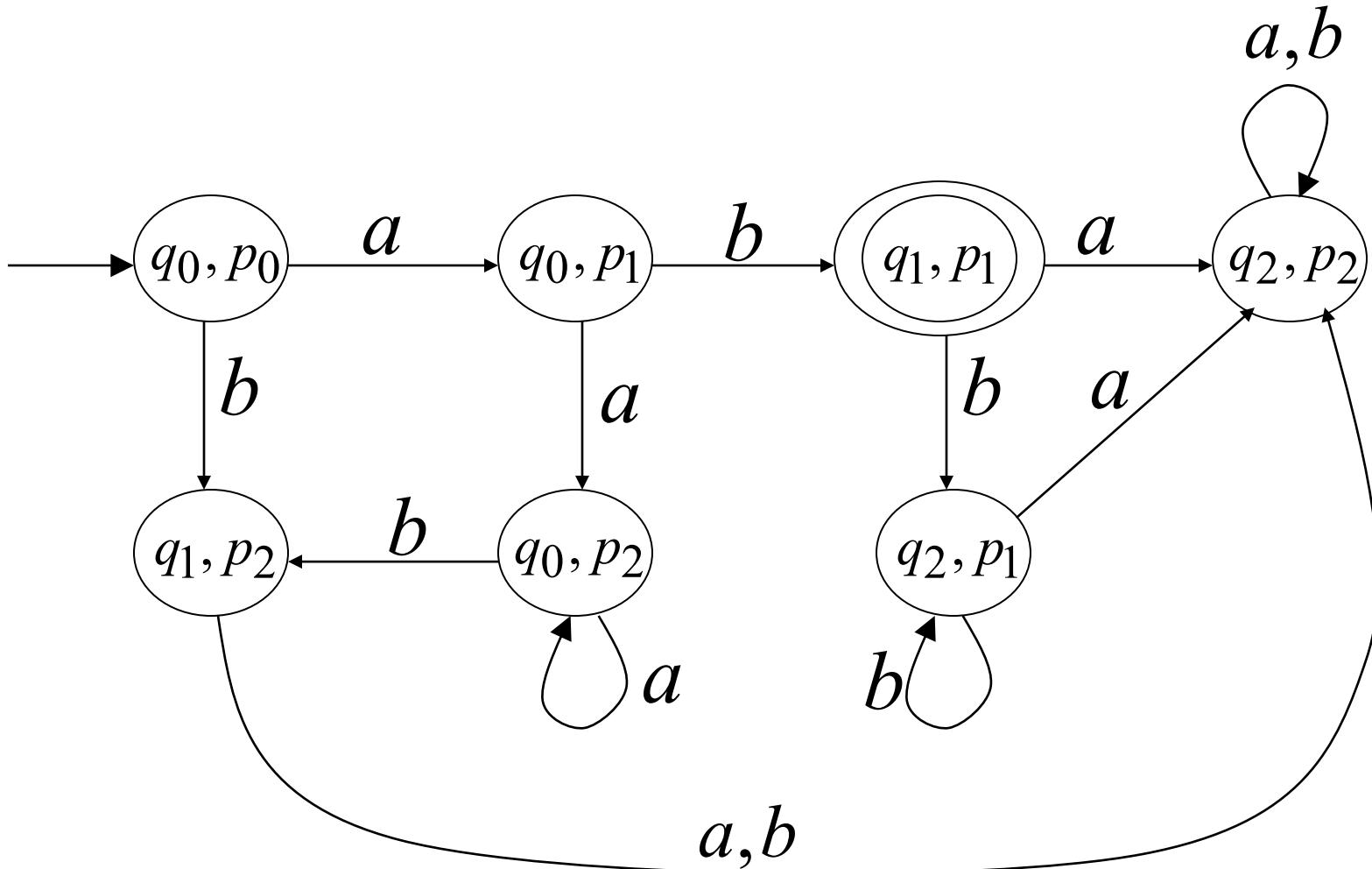
0

$$m \geq 0$$
$$L_2 = \{ab^m\}$$



Intersezione automata

$$L = \{a^n b\} \cap \{ab^n\} = \{ab\}$$



Se appartenne ad entrambi

Sia la stringa di lunghezza n

Esistono due cammini di lunghezza n , uno per ogni automa. Dallo stato iniziale a quello finale.

Se Ing 1 vero, dimostrare vero per $n+1$.

Ultimo tratto da n a $n+1$ arco nei due automa e arco
automa costruito. Considera stringa n e considera
come stato finale quello prima dello stato finale, vedi
arco che riconosce il carattere n , simula i due automi.
Continua ad andare indietro fino a raggiungere lo stato
iniziale.

Nell'automa costruito esiste un cammino
di $\text{Ing } n$ che li simula

M Simula in parallelo M_1 e M_2

M accettazione stringa w Se e solo se:

M_1 accetta w string

e M_2 accetta w string

$$L(M) = L(M_1) \cap L(M_2)$$

Espressioni regolari e linguaggi regolari

Teorema

$$\left\{ \begin{array}{l} \text{Linguaggi} \\ \text{Generati da} \\ \text{Espressioni regolari} \end{array} \right\} = \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

Dimostrazione - Parte 1

$$\left\{ \begin{array}{l} \text{Linguaggi} \\ \text{Generati da} \\ \text{Espressioni regolari} \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

per ogni espressione regolare r
il linguaggio $L(r)$ è regolare

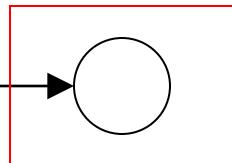
Dimostrazione per induzione sulla lunghezza

r

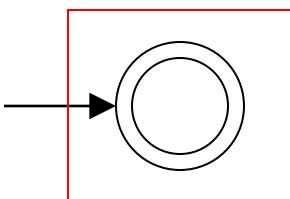
Base induzione

Espressioni regolari di base: $\emptyset, \lambda, \alpha$
corrispondente

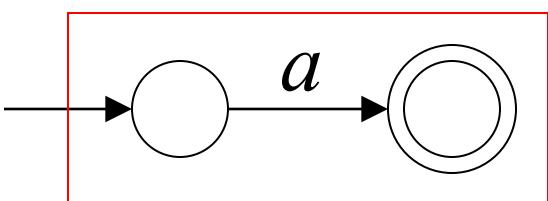
NFAs



$$L(M_1) = \emptyset = L(\emptyset)$$



$$L(M_2) = \{\lambda\} = L(\lambda)$$



$$L(M_3) = \{a\} = L(a)$$

Linguaggi
regolari

Ipotesi induttiva

supponi

Per le espressioni regolari r_1 e r_2 ,
 $L(r_1)$ e $L(r_2)$ sono linguaggi regolari.

Esistono due automi uno per
ogni linguaggio

Passo induttivo

Proviamo che:

$$\left. \begin{array}{l} L(r_1 + r_2) \\ L(r_1 \cdot r_2) \\ L(r_1^*) \\ L((r_1)) \end{array} \right\}$$

Sono linguaggi
regolari

Ricorda che, per def. di espressione regolare

$$L(r_1 + r_2) = L(r_1) \cup L(r_2)$$

$$L(r_1 \cdot r_2) = L(r_1) L(r_2)$$

$$L(r_1^*) = (L(r_1))^*$$

$$L((r_1)) = L(r_1)$$

Per ipotesi induttiva :

$L(r_1)$ e $L(r_2)$ sono linguaggi regolari

Inoltre sappiamo, slides precedenti:

I linguaggi regolari sono chiusi rispetto:

Unione

$$L(r_1) \cup L(r_2)$$

Concatenazione

$$L(r_1) L(r_2)$$

Star

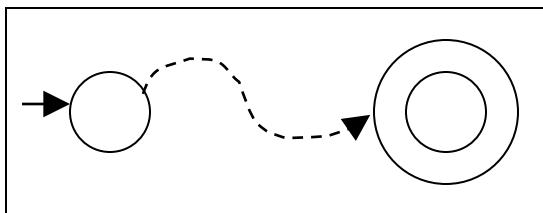
$$(L(r_1))^*$$

Usando la chiusura delle operazioni
Possiamo costruire un NFA M tale che:

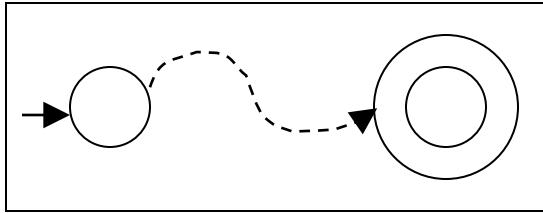
$$L(M) = L(r)$$

esempio: $r = r_1 + r_2$

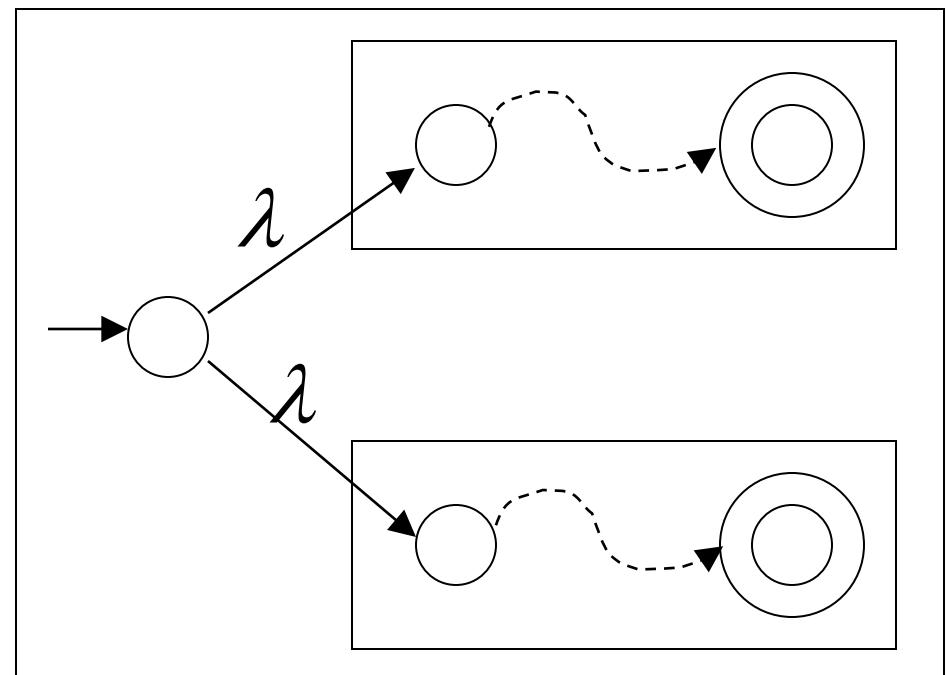
$$L(M_1) = L(r_1)$$



$$L(M_2) = L(r_2)$$



$$L(M) = L(r)$$



Stella e puntino.

esercizio

Stella: torna indietro con
lambda.

Puntino: collega i finali del primo
con l'iniziale con un lambda.

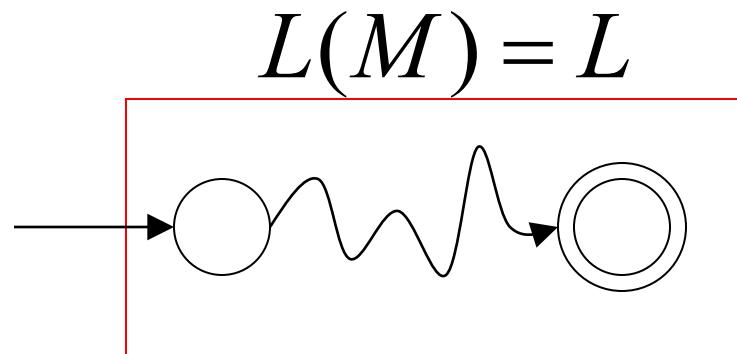
dim - Part 2

$$\left\{ \begin{array}{l} \text{Linguaggi} \\ \text{Generati da} \\ \text{Espressioni regolari} \end{array} \right\} \supseteq \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

Per ogni linguaggio regolare L esiste
una espressione regolare r con $L(r) = L$

Convertiremo un NFA che accetta L
In una espressione regolare

Poichè L è regolare , allora esiste un NFA M che lo accettà

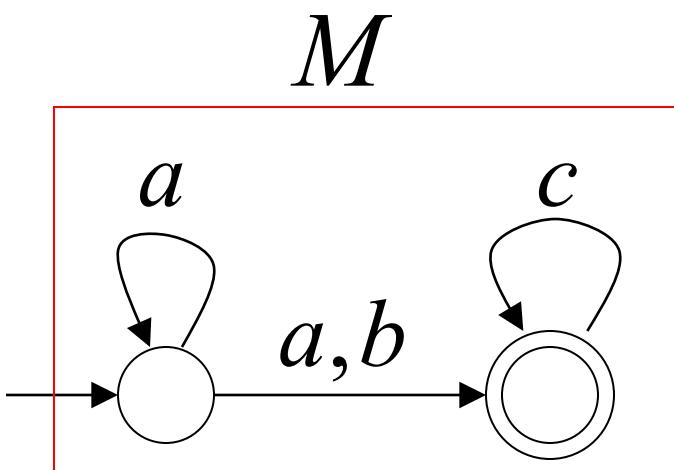


Prendiamo l'automa con
un solo stato finale

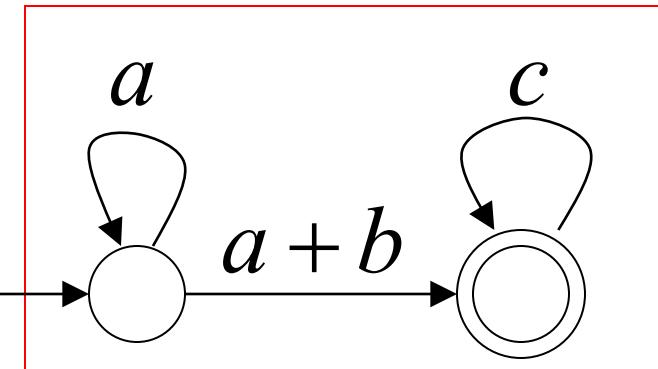
da M costruiamo l'equivalente
Generalized Transition Graph

Nel quale i caratteri di transizione, transition labels, sono espressioni regolari

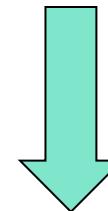
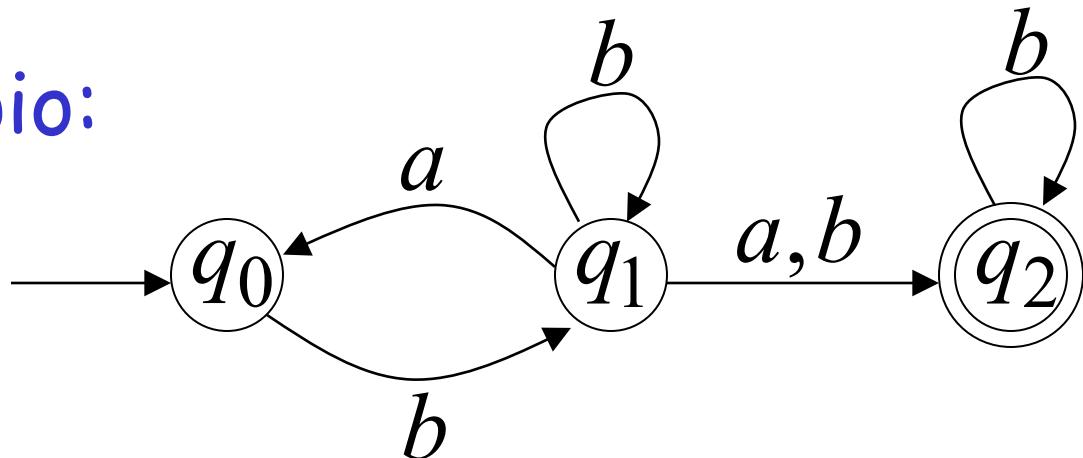
Esempio:



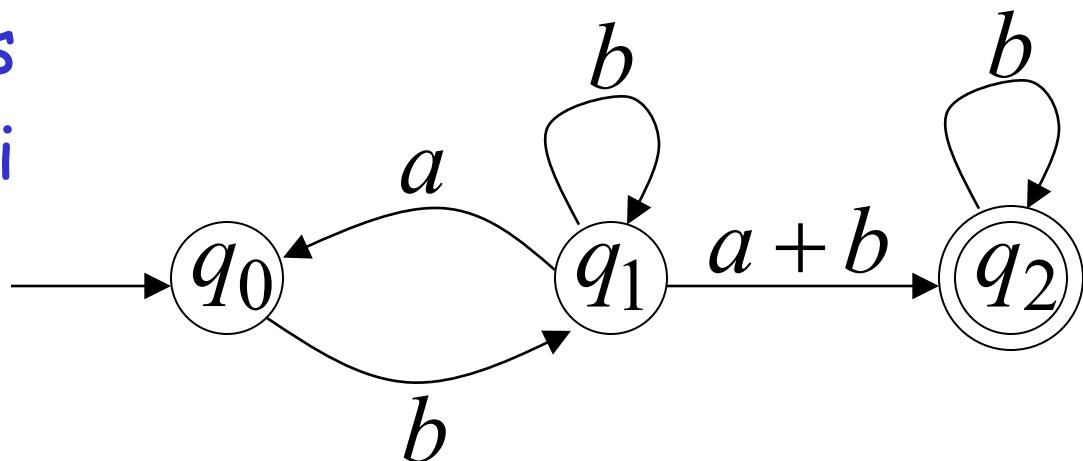
Il corrispondente
Generalized transition graph



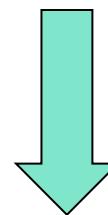
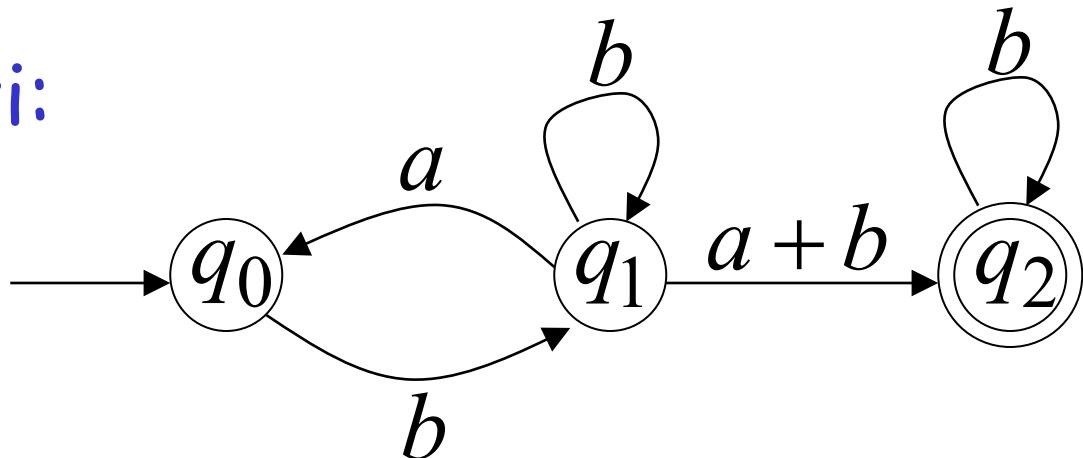
Un altro esempio:



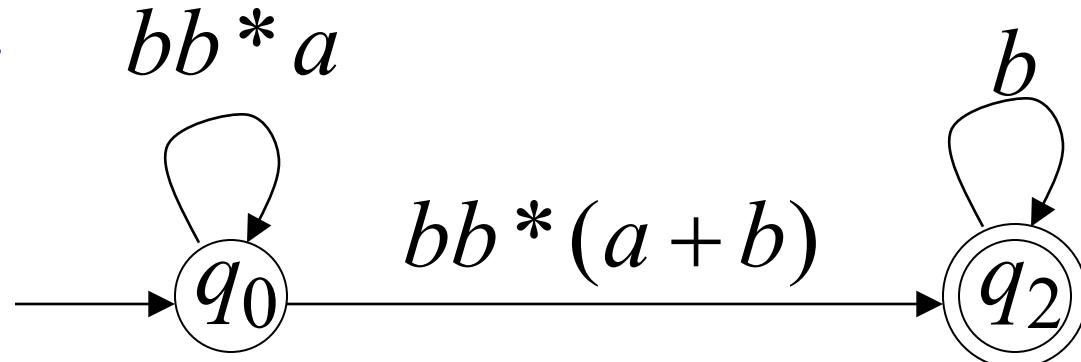
Transition labels
Sono espressioni
regolari



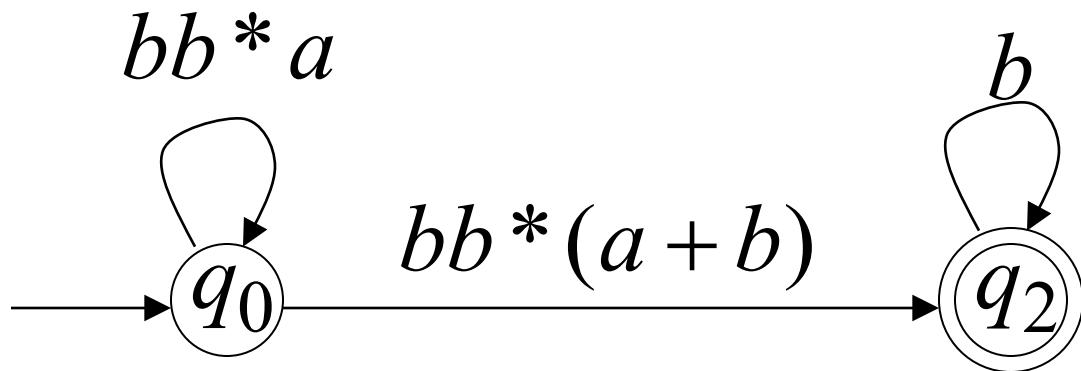
Ridurre gli stati:



Transition labels
sono espressioni
regolari



Espressione regolare che si ottiene:



$$r = (bb^* a)^* bb^* (a + b) b^*$$

$$L(r) = L(M) = L$$

Stato iniziale solo archi uscenti, nessuno rientrante

Solo uno finale tutti entranti e nessun uscente.

Per gli altri stati sono presenti archi uscenti per tutti gli altri stati ed entranti da tutti gli altri stati e su se stesso. Se non esiste un arco da q_i a q_j creiamo un arco con label insieme vuoto \emptyset

Se $k=2$ slide precedente

Altimenti

prendiamo lo stato da eliminare q

Per ogni q_i e q_j collegati via q

$$\delta^*(q_1, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$$

vai da q_i a q , R_1

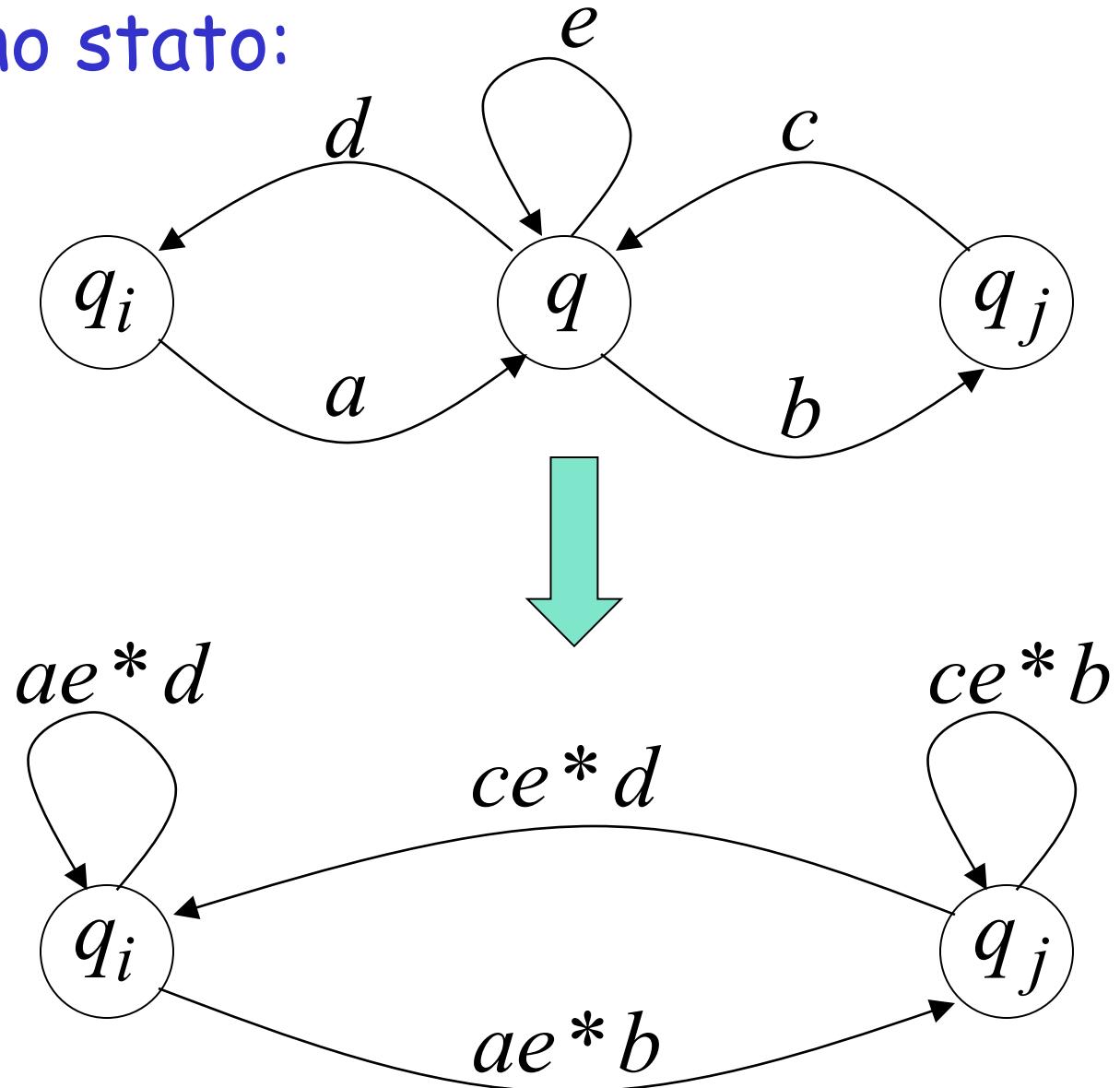
gira su q , $(R_2)^*$

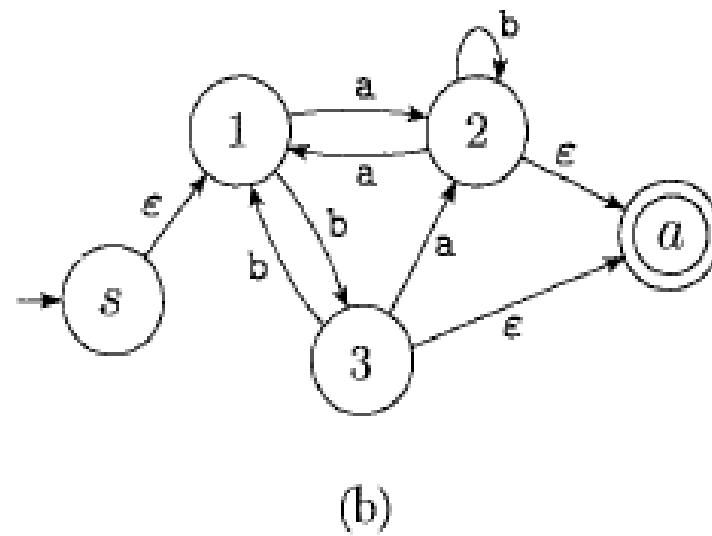
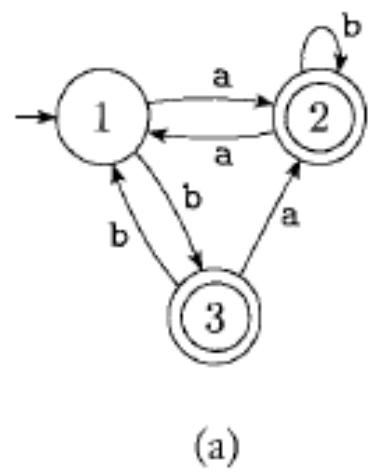
vai da q a q_j , R_3

direttamente da q_i a q_j , R_4

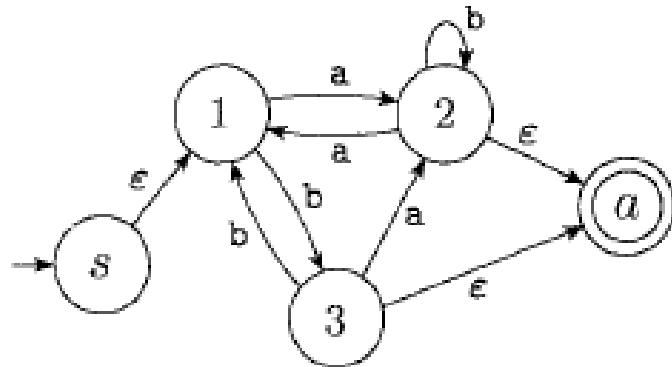
In generale

Rimuovere uno stato:

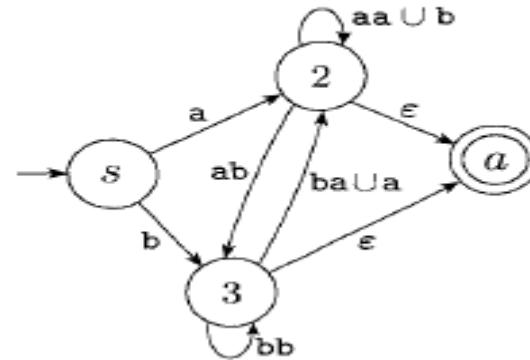




Per ogni q_i e q_j collegati via q

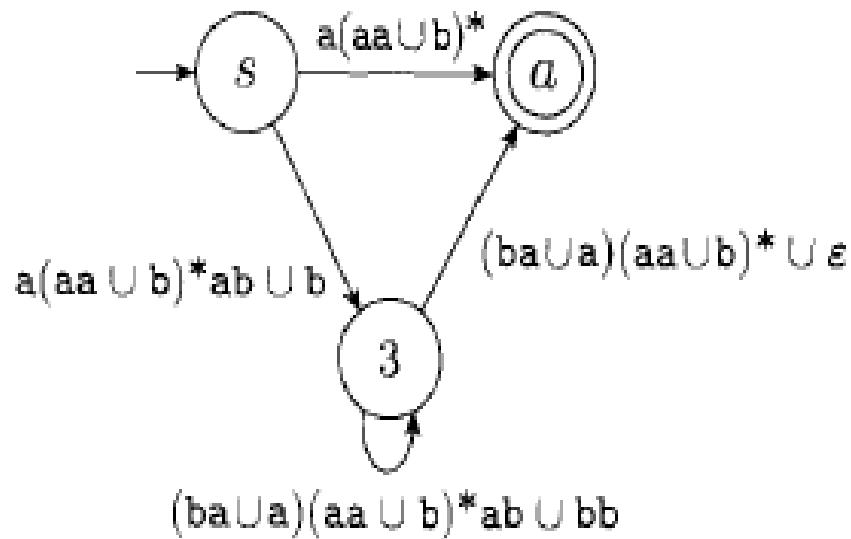
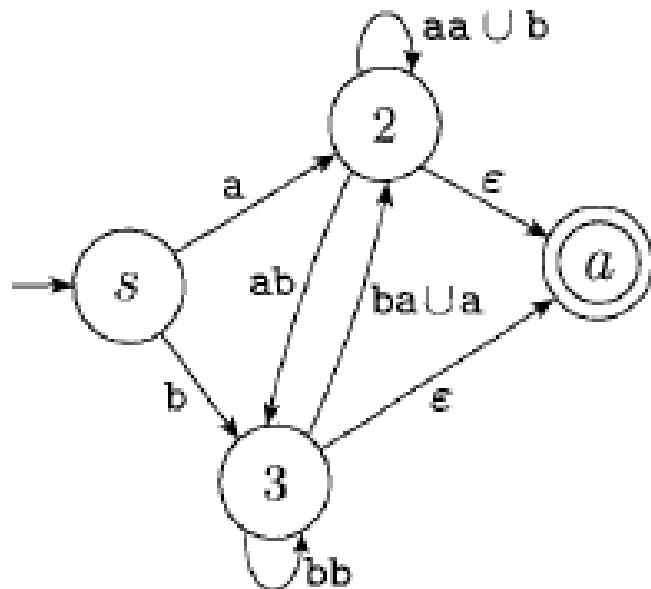


(b)

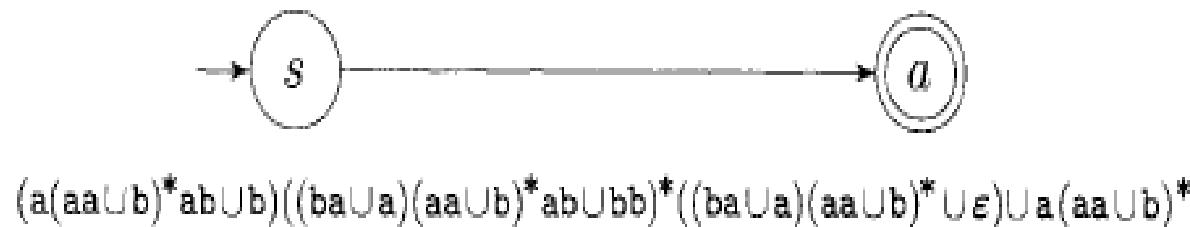
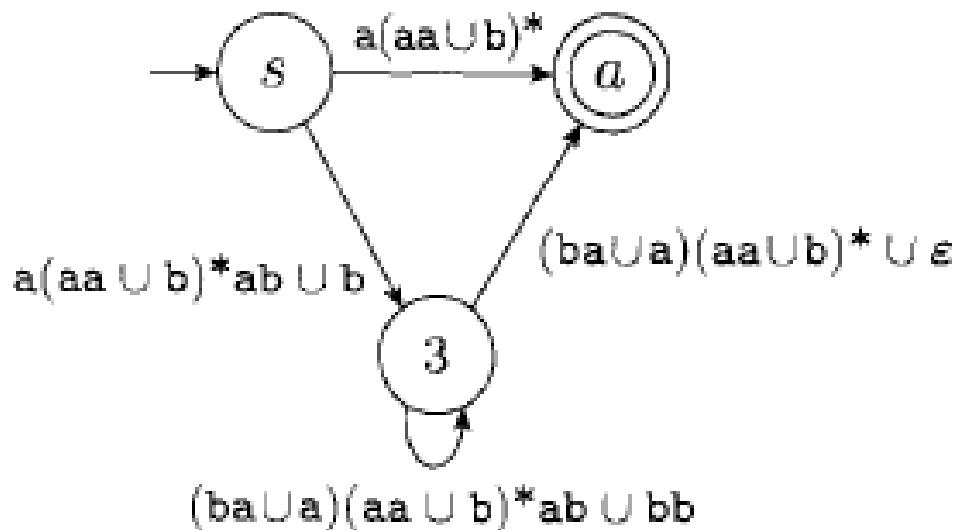


vai da q_i a q , R_1
gira su q , $(R_2)^*$
via da q a q_j , R_3
direttamente da q_i a q_j , R_4

$$\delta^*(q_1, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$$



vai da q_i a q_j , R_1
 gira su q_j , $(R_2)^*$
 vai da q_j a q_i , R_3
 direttamente da q_i a q_j , R_4



Algoritmo: Eliminare uno stato alla volta fino a che restano 2 stati.

Dimostrazione algoritmo funziona ovvero l'automa iniziale G e G' , meno uno stato, accettano lo stesso linguaggio

G' con due stati allora otteniamo espressione regolare.

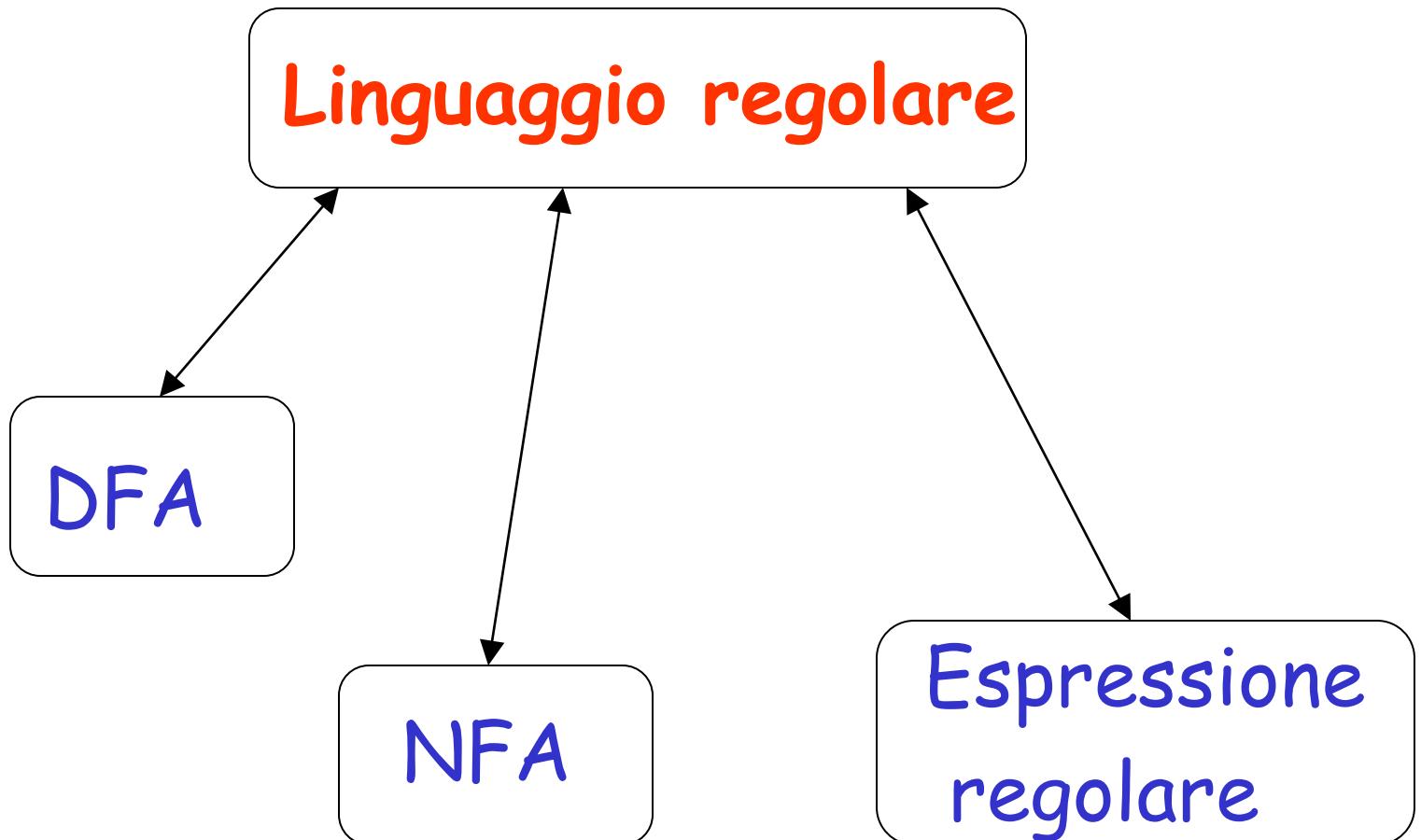
Vero per $k-1$ provare per $k+1$.

Prendiamo una stringa che viene accettata, esiste un cammino che accetta la stringa se non usa lo stato da eliminare bene G e G' accettano la stringa. In slang:

«Se G non usa lo stato da eliminare: bene G'
«si fa lo stesso giro» »;

Se G usa lo stato da eliminare allora in G' lo stato in oggetto non esiste ma nei nuovi archi tutte le sottostringhe che venivano riconosciute tramite lo stato eliminato sono descritte dalle espressioni regolari sugli archi

Presentazione standard di un linguaggio regolare



esercizi
slide successive

esempio

Epressione regolare: $(a + b) \cdot a^*$

$$\begin{aligned}L((a + b) \cdot a^*) &= L((a + b)) L(a^*) \\&= L(a + b) L(a^*) \\&= (L(a) \cup L(b)) (L(a))^* \\&= (\{a\} \cup \{b\}) (\{a\})^* \\&= \{a, b\} \{\lambda, a, aa, aaa, \dots\} \\&= \{a, aa, aaa, \dots, b, ba, baa, \dots\}\end{aligned}$$

Linguaggi associati alle espressioni regolari

$L(r)$: linguaggio associato all'espressione r

esempio

$$L((a + b \cdot c)^*) = \{\lambda, a, bc, aa, abc, bca, \dots\}$$

Esempio

Espressione regolare $r = (a + b)^*(a + bb)$

$$L(r) = \{a, bb, aa, abb, ba, bbb, \dots\}$$

esempio

Espressione regolare $r = (aa)^*(bb)^*b$

$$L(r) = \{a^{2n}b^{2m}b : n, m \geq 0\}$$

esempio

Espressione regolare $r = (0 + 1)^* 00 (0 + 1)^*$

$L(r) = \{ \text{tutte le stringhe che contengono } 00 \}$

esempio

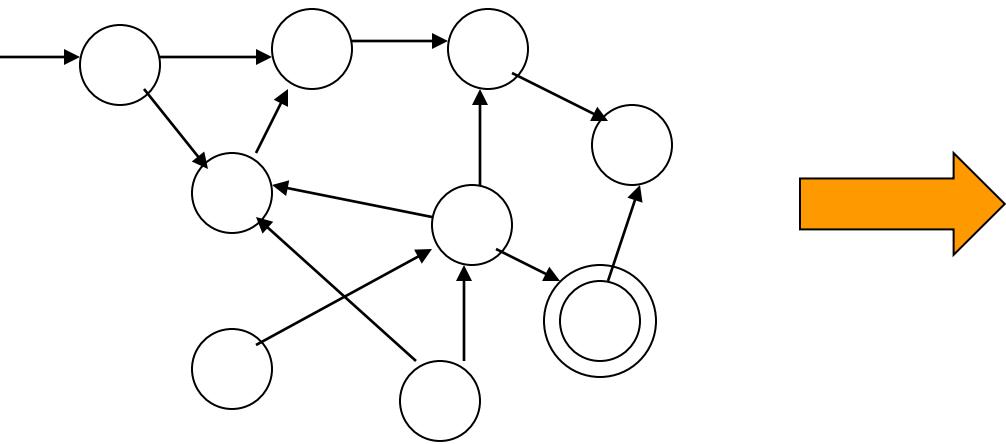
Espressione regolare $r = (1 + 01)^*(0 + \lambda)$

$L(r)$

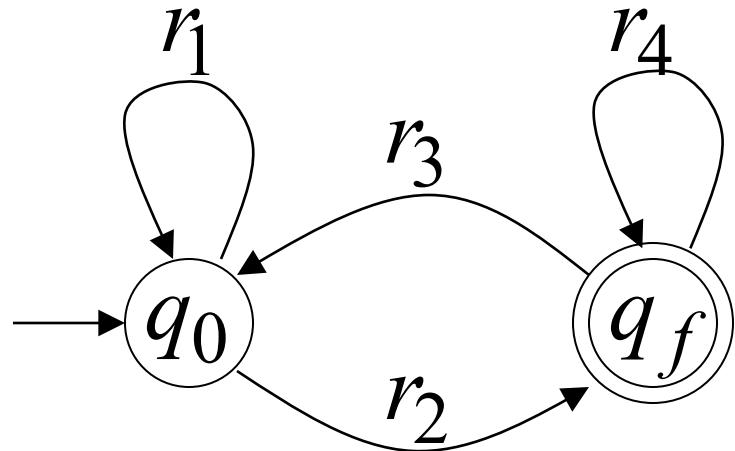
= { tutte le stringhe senza sottostringhe 00 }

Ripetere il processo finchè
Due stati restano il grafo risultante
sarà il seguente

Grafo iniziale



Grafo risultante

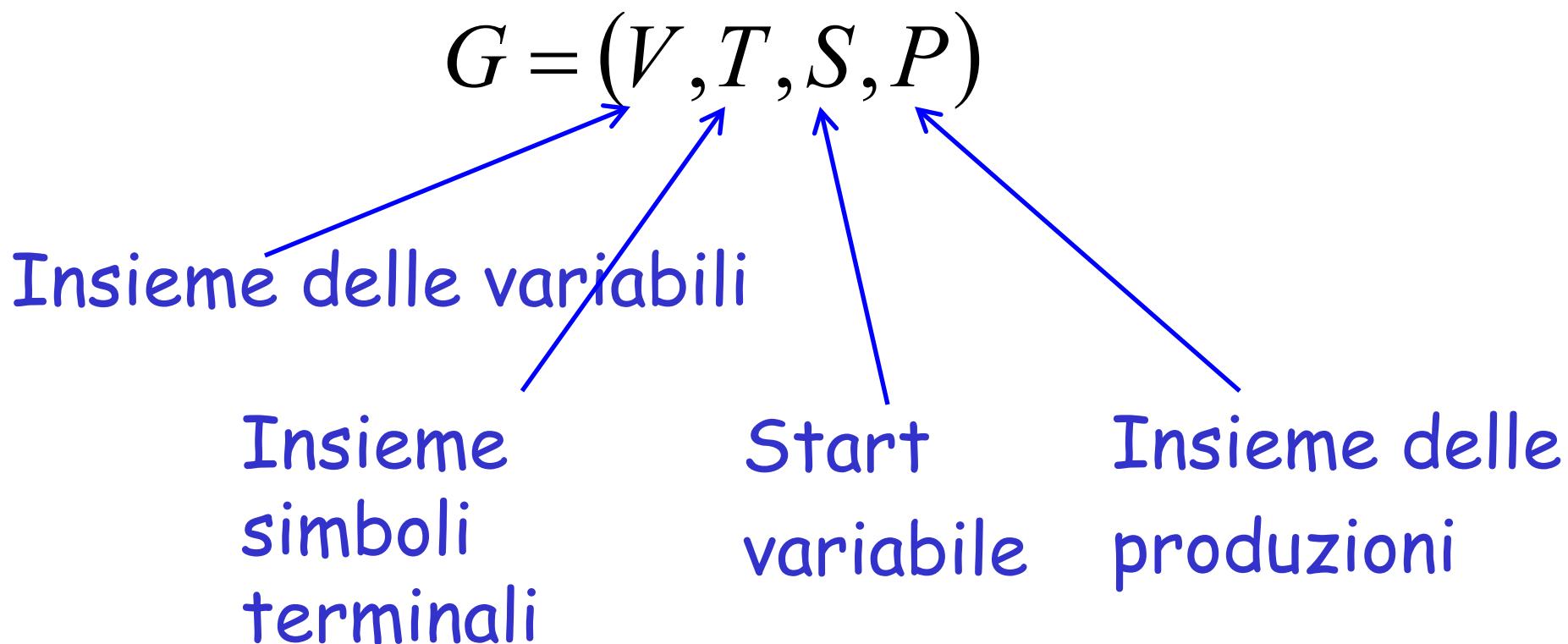


L'espressione regolare risultante:

$$r = r_1 * r_2 (r_4 + r_3 r_1 * r_2) *$$
$$L(r) = L(M) = L$$

Definizione formale

Grammatica:



$$G = (V, T, S, P)$$

Tutte le produzioni P sono della forma:

$$A \rightarrow S$$

Stringhe di
Variabili e non
terminali

Linguaggio di una grammatica:

Per una grammatica G con start S

$$L(G) = \{w : S \xrightarrow{*} w, \quad w \in T^*\}$$



Stringhe di terminali o λ

terminali

Grammatica lineare

Le grammatiche con al massimo una variabile
sul lato destro della produzione

Esempio:

$$S \rightarrow aSb$$

$$S \rightarrow Ab$$

$$S \rightarrow \lambda$$

$$A \rightarrow aAb$$

$$A \rightarrow \lambda$$

Grammatica non lineare

Grammatica G : $S \rightarrow SS$

$S \rightarrow \lambda$

$S \rightarrow aSb$

$S \rightarrow bSa$

$$L(G) = \{w: n_a(w) = n_b(w)\}$$

Numeri di a nella stringa w

Grammatica lineare

Grammatica $G : S \rightarrow A$

$$A \rightarrow aB \mid \lambda$$

$$B \rightarrow Ab$$

$$L(G) = \{a^n b^n : n \geq 0\}$$

Grammatica lineare a destra

Tutte le produzioni hanno la forma

$$A \rightarrow xB$$

o

$$A \rightarrow x$$

esempio: $S \rightarrow abS$

$$S \rightarrow a$$

Stringa di terminali

Grammatiche lineare sinistra

Tutte le produzioni hanno
la forma:

$$A \rightarrow Bx$$

o

$$A \rightarrow x$$



$$S \rightarrow Aab$$

Esempio:

$$A \rightarrow Aab \mid B$$

$$B \rightarrow a$$

Stringhe
di terminali

Grammatica regolare

Grammatiche regolari

Una grammatica regolare è qualsiasi grammatica lineare a destra o a sinistra

Esempio:

G_1

$$S \rightarrow abS$$

$$S \rightarrow a$$

G_2

$$S \rightarrow Aab$$

$$A \rightarrow Aab \mid B$$

$$B \rightarrow a$$

I linguaggi generati da una grammatica regolare è un linguaggio regolare

Examples:

G_1

$S \rightarrow abS$

$S \rightarrow a$

$L(G_1) = (ab)^* a$

G_2

$S \rightarrow Aab$

$A \rightarrow Aab \mid B$

$B \rightarrow a$

$L(G_2) = aab(ab)^*$

Grammatiche regolari
generano linguaggi regolari

Teorema

$$\left\{ \begin{array}{l} \text{Linguaggi} \\ \text{Generati da} \\ \text{grammatiche} \\ \text{regolari} \end{array} \right\} = \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

Teorema - Part 1

$$\left\{ \begin{array}{l} \text{Linguaggi} \\ \text{Generati da} \\ \text{grammatiche} \\ \text{regolari} \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

Ogni grammatica regolare
Genera un linguaggio generale

Teorema - Part 2

$$\left\{ \begin{array}{l} \text{Linguaggi} \\ \text{Generati da} \\ \text{grammatiche} \\ \text{regolari} \end{array} \right\} \supseteq \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

Ogni linguaggio regolare
È generato da una grammatica regolare

Proof - Part 1

$$\left\{ \begin{array}{l} \text{Linguaggi} \\ \text{Generati da} \\ \text{grammatiche} \\ \text{regolari} \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

Il linguaggio $L(G)$ generato da
Una grammatica regolare G è regolare

Il caso della Grammatica

sia G una right-linear grammatica

proveremo: $L(G)$ è regolare

idea: costruiamo una NFA M
con $L(M) = L(G)$

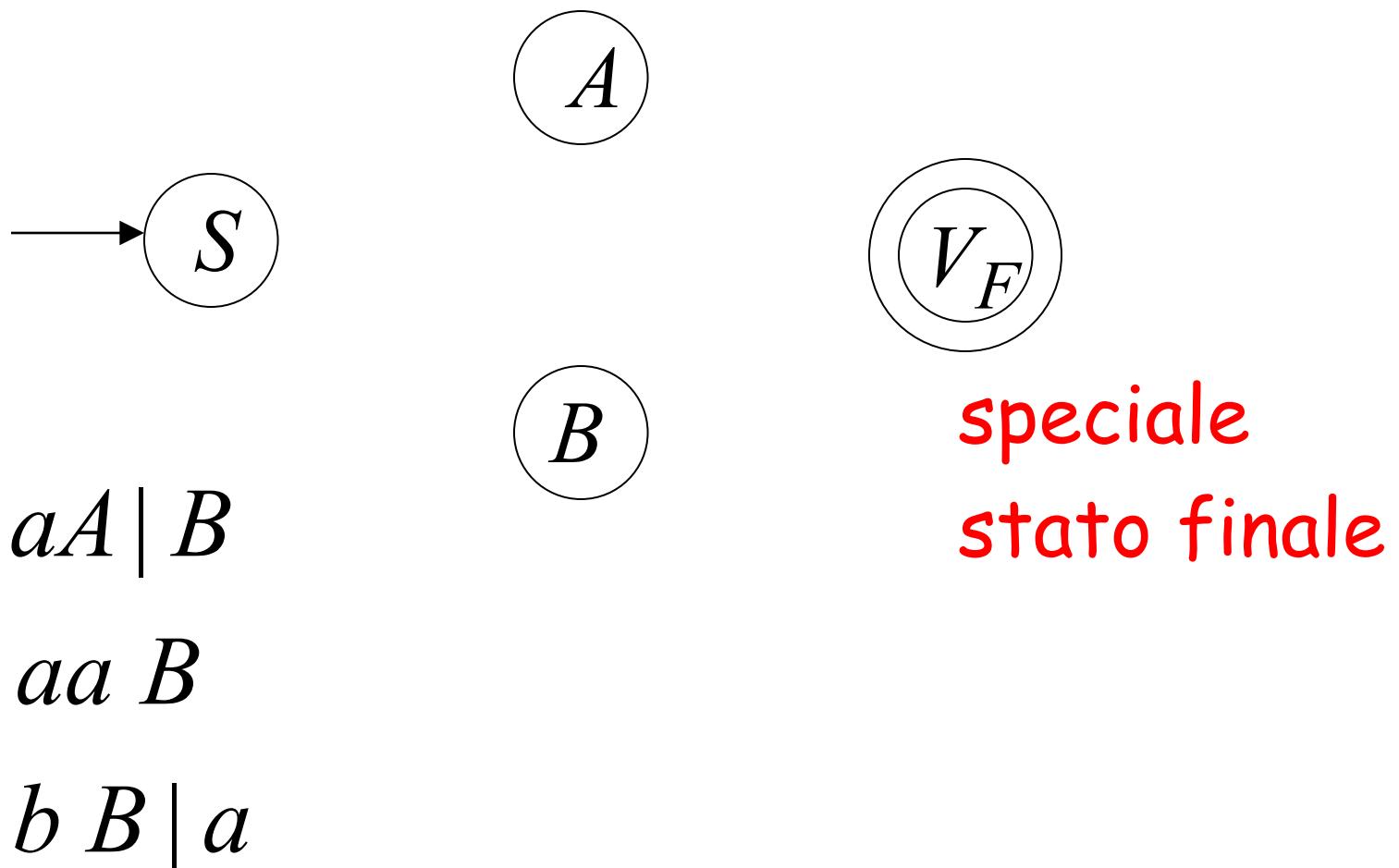
Grammatica G è right-linear

Esempio: $S \rightarrow aA \mid B$

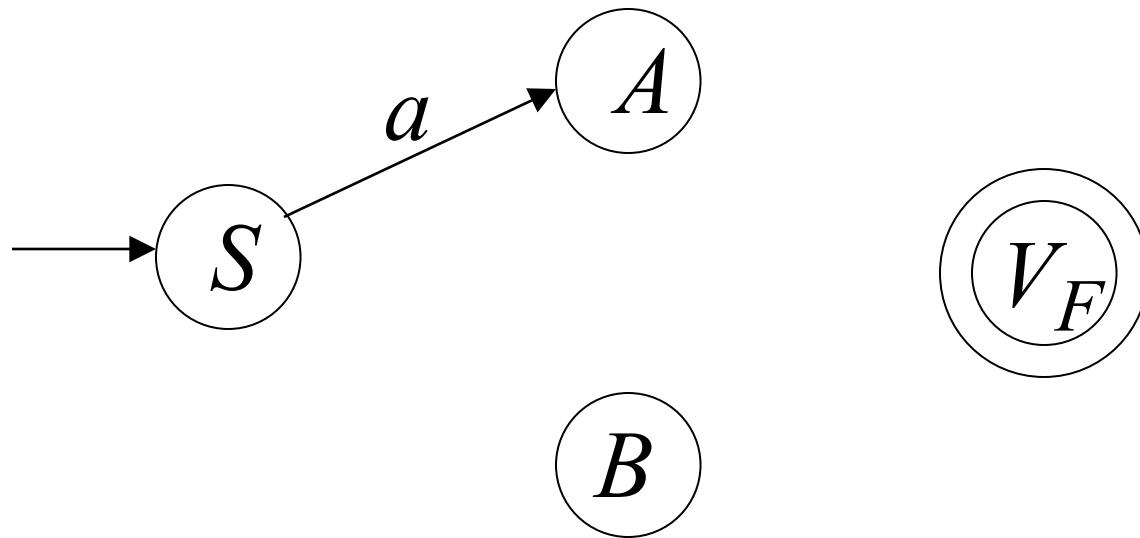
$A \rightarrow aa \ B$

$B \rightarrow b \ B \mid a$

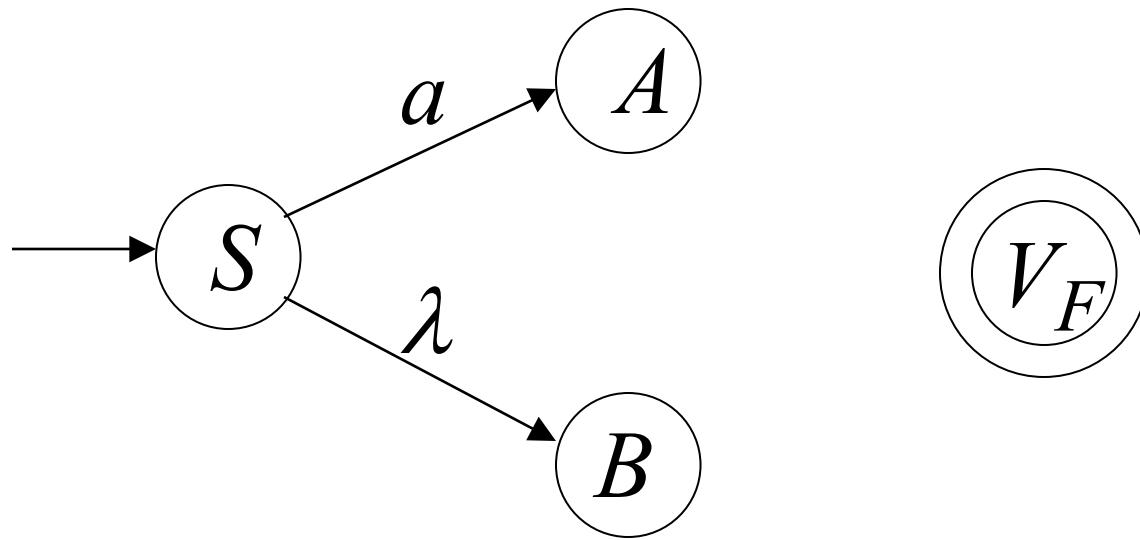
Construiamo NFA M tale che
ogni stato è una variabile della grammatica :



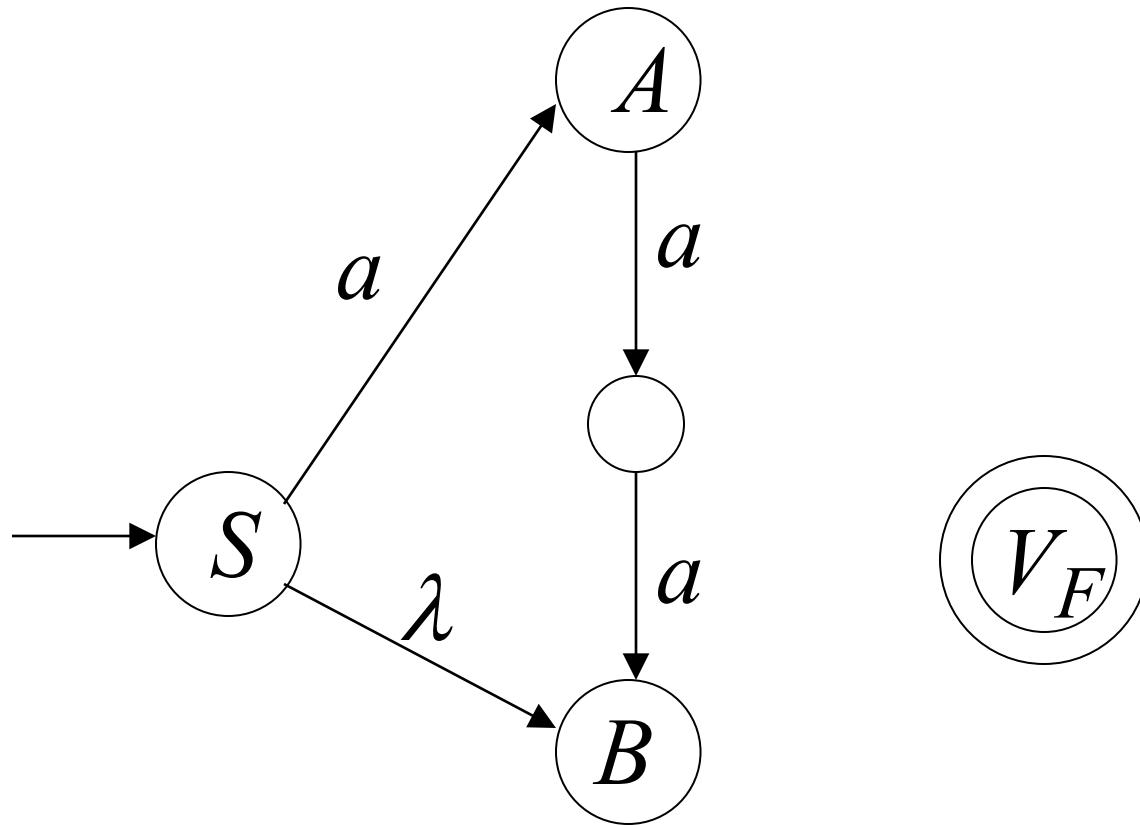
Addizioniamo un arco per ogni produzione:



$$S \rightarrow aA$$

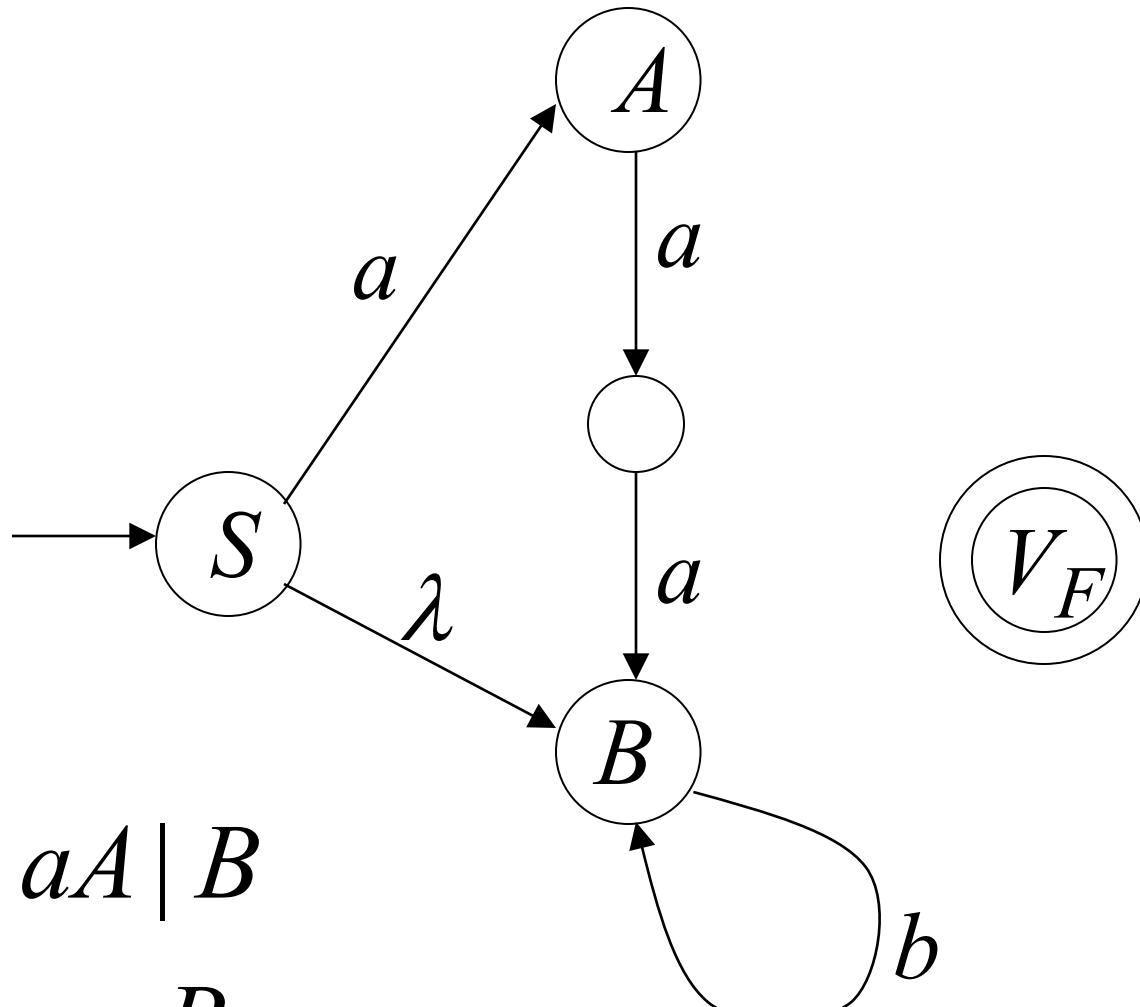


$$S \rightarrow aA \mid B$$

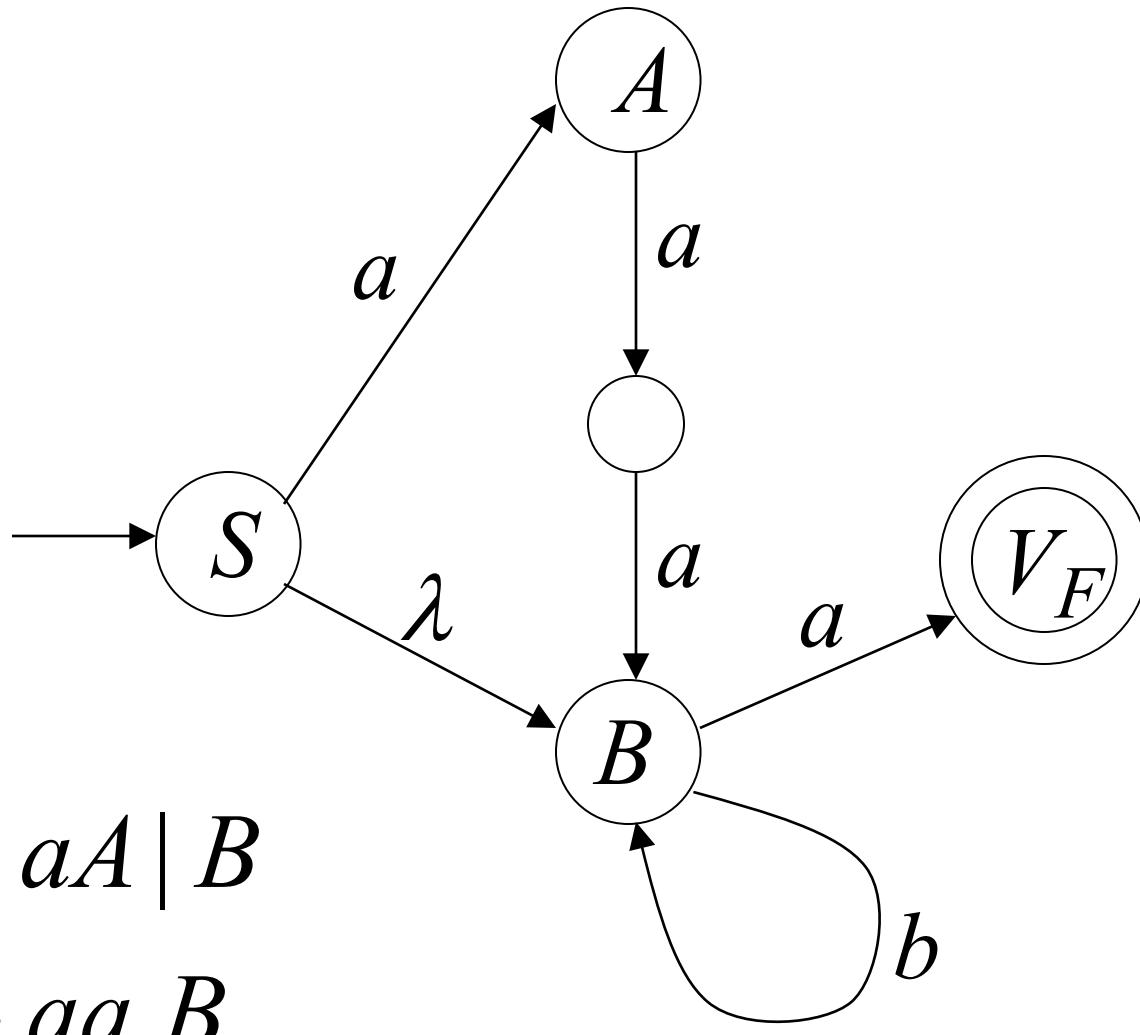


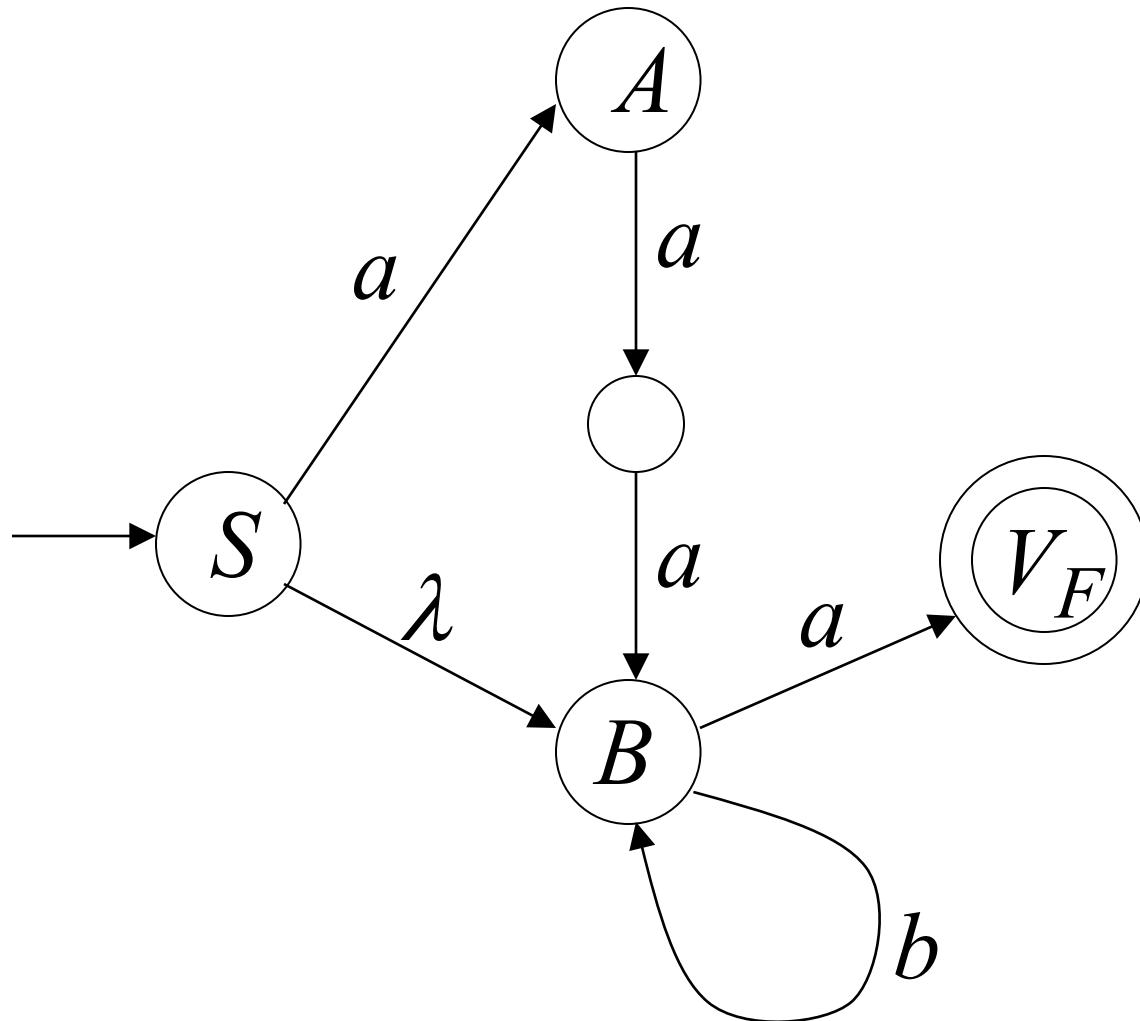
$$S \rightarrow aA \mid B$$

$$A \rightarrow aa \ B$$


$$S \rightarrow aA \mid B$$
$$A \rightarrow aa \ B$$
$$B \rightarrow bB$$

03/04/2021


$$S \rightarrow aA \mid B$$
$$A \rightarrow aa \mid B$$
$$B \rightarrow bB \mid a$$


$$S \Rightarrow aA \Rightarrow aaaB \Rightarrow aaabB \Rightarrow aaaba$$

NFA M

Grammatica

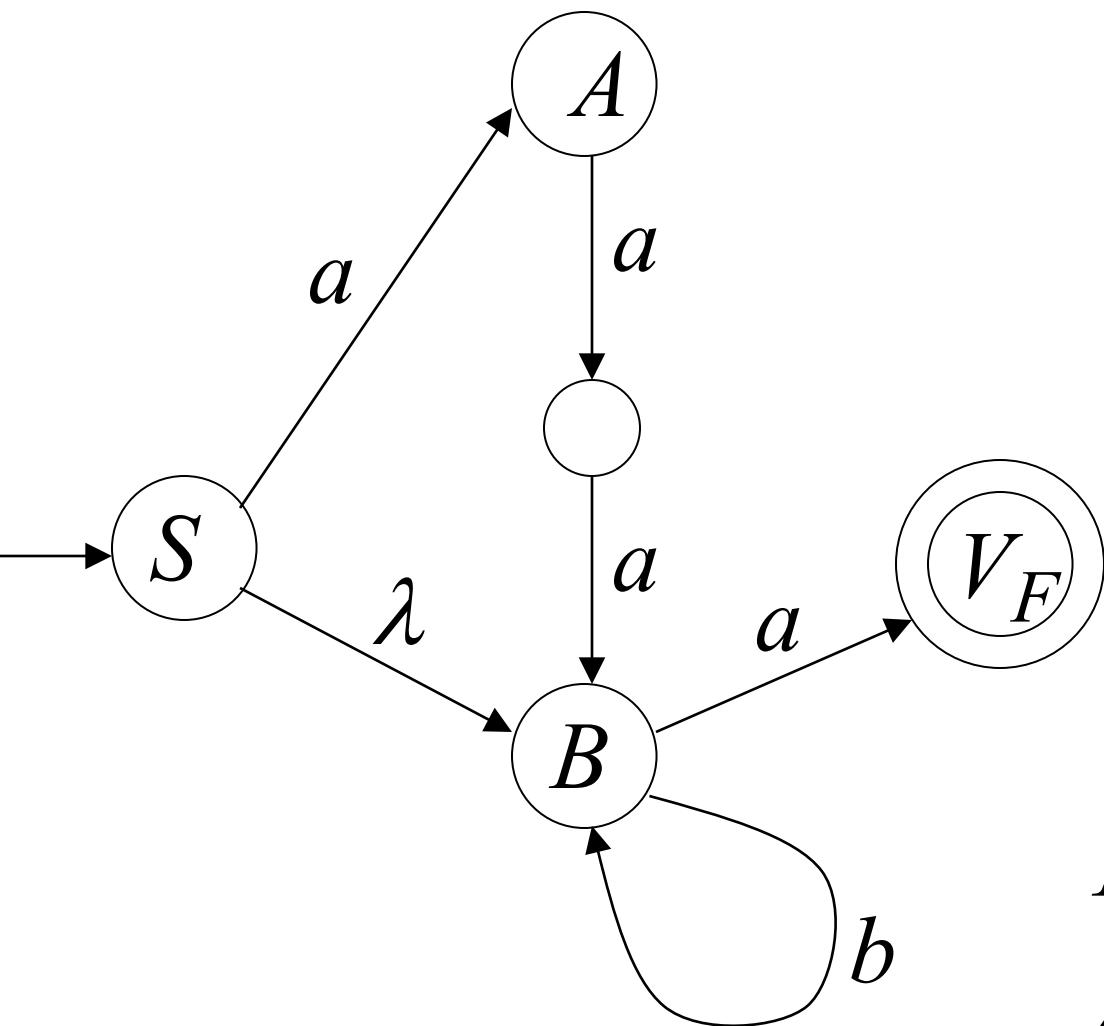
G

$$S \rightarrow aA \mid B$$

$$A \rightarrow aa \mid B$$

$$B \rightarrow bB \mid a$$

$$\begin{aligned} L(M) &= L(G) = \\ &aaab^*a + b^*a \end{aligned}$$



In Generale
una right-linear grammatica G

Ha le variabili: V_0, V_1, V_2, \dots

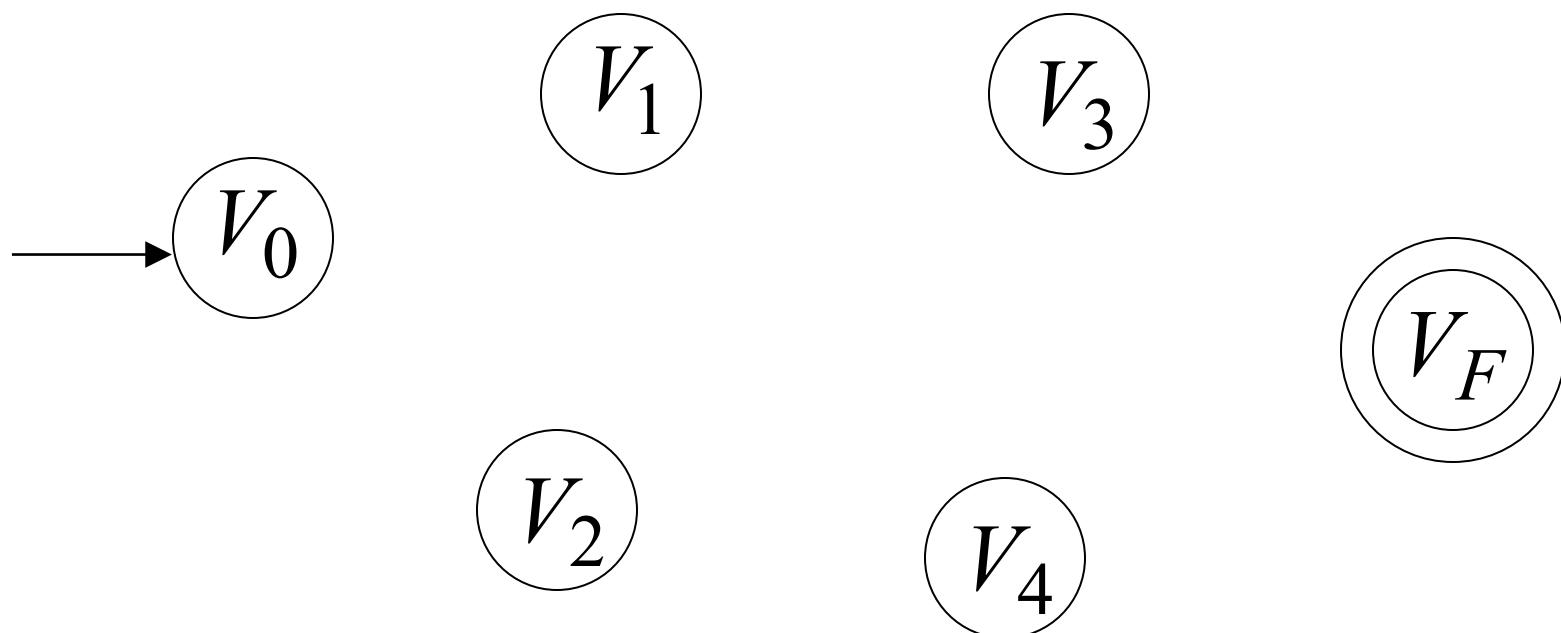
E le produzioni: $V_i \rightarrow a_1 a_2 \cdots a_m V_j$

or

$V_i \rightarrow a_1 a_2 \cdots a_m$

Costruiamo un NFA M tale che:

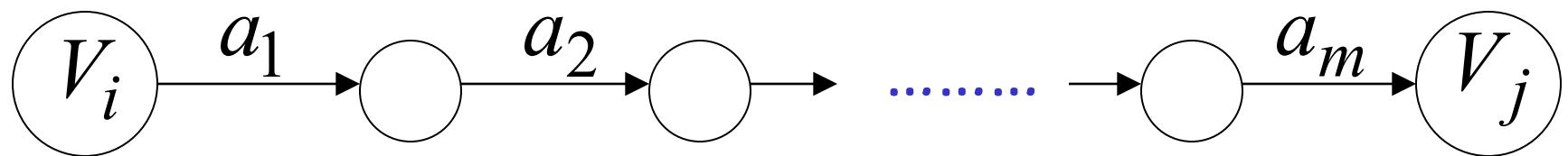
Ogni variabile V_i corrisponde ad un nodo:



stato finale

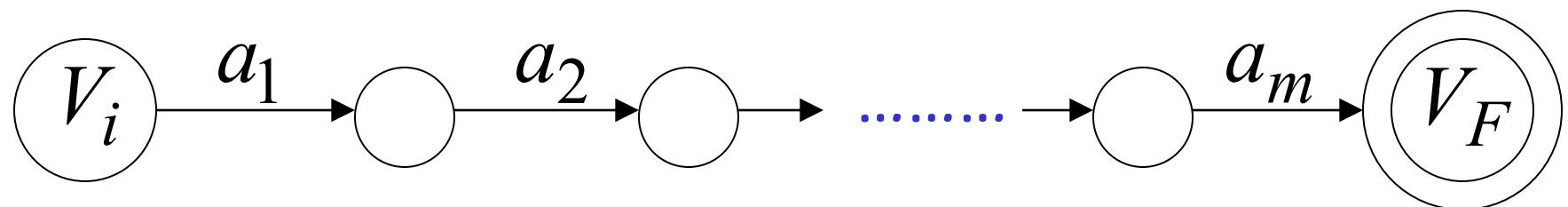
per ogni produzione: $V_i \rightarrow a_1 a_2 \cdots a_m V_j$

Addizioniamo transizioni e nodi intermedi

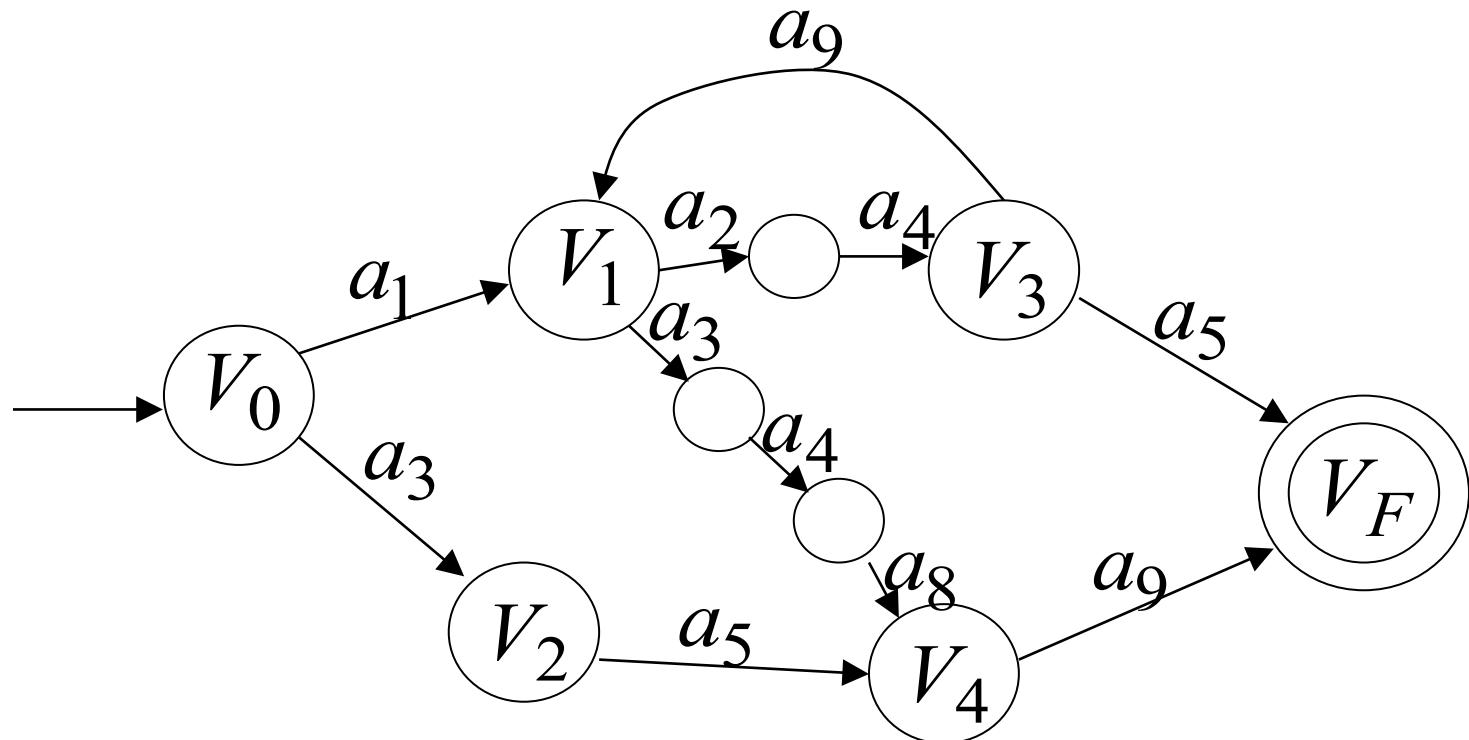


per ogni produzione: $V_i \rightarrow a_1 a_2 \cdots a_m$

Addizioniamo transizioni e nodi intermedi



otteniamo NFA M come questo:



Vale che:

$$L(G) = L(M)$$

Il caso di una Left-Linear grammatica

Fate voi

dimostrazione - Part 2

$$\left\{ \begin{array}{l} \text{Linguaggi} \\ \text{Generati da} \\ \text{grammatiche} \\ \text{regolari} \end{array} \right\} \supseteq \left\{ \begin{array}{l} \text{Linguaggi} \\ \text{regolari} \end{array} \right\}$$

ogni linguaggio regolare L è generato
da qualche grammatica regolare G

qualsiasi linguaggio regolare L è generato da una grammatica regolare G

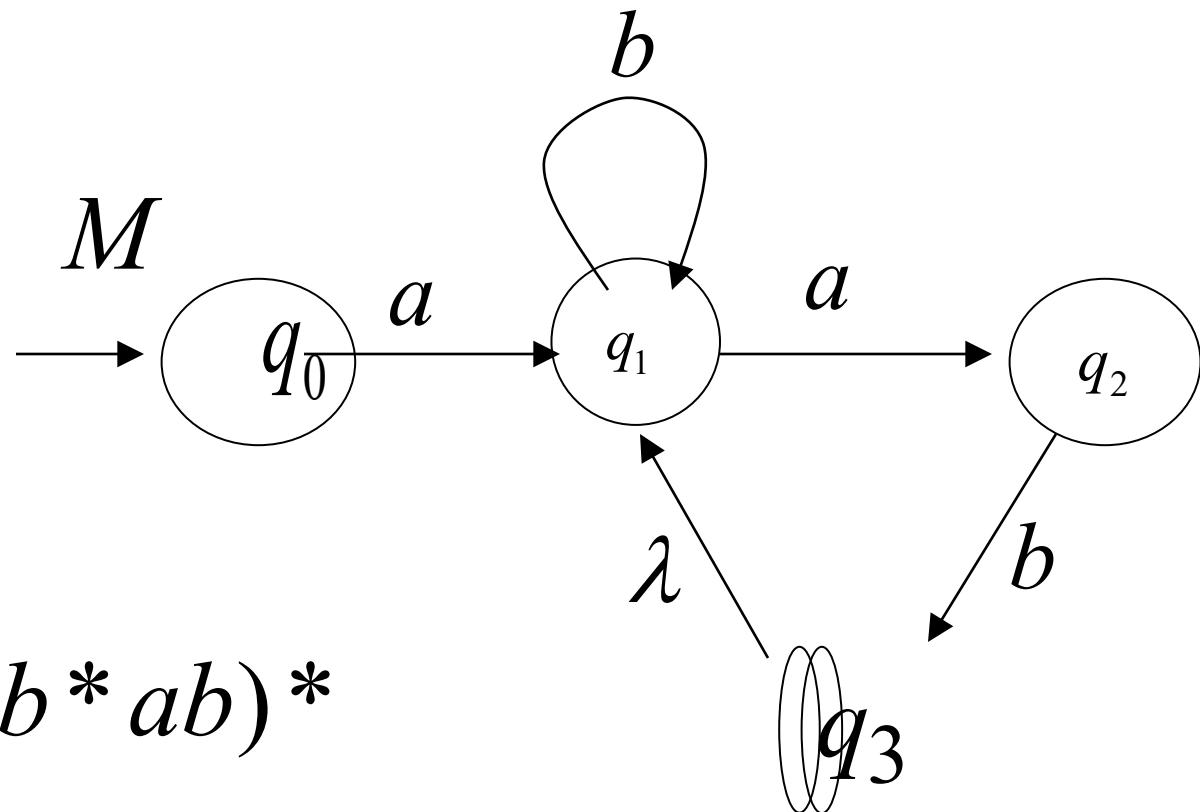
idea:

sia M NFA con $L = L(M)$.

costruiamo da M una grammatica G regolare tale che $L(M) = L(G)$

Poichè L è regolare
è un NFA M tale che $L = L(M)$

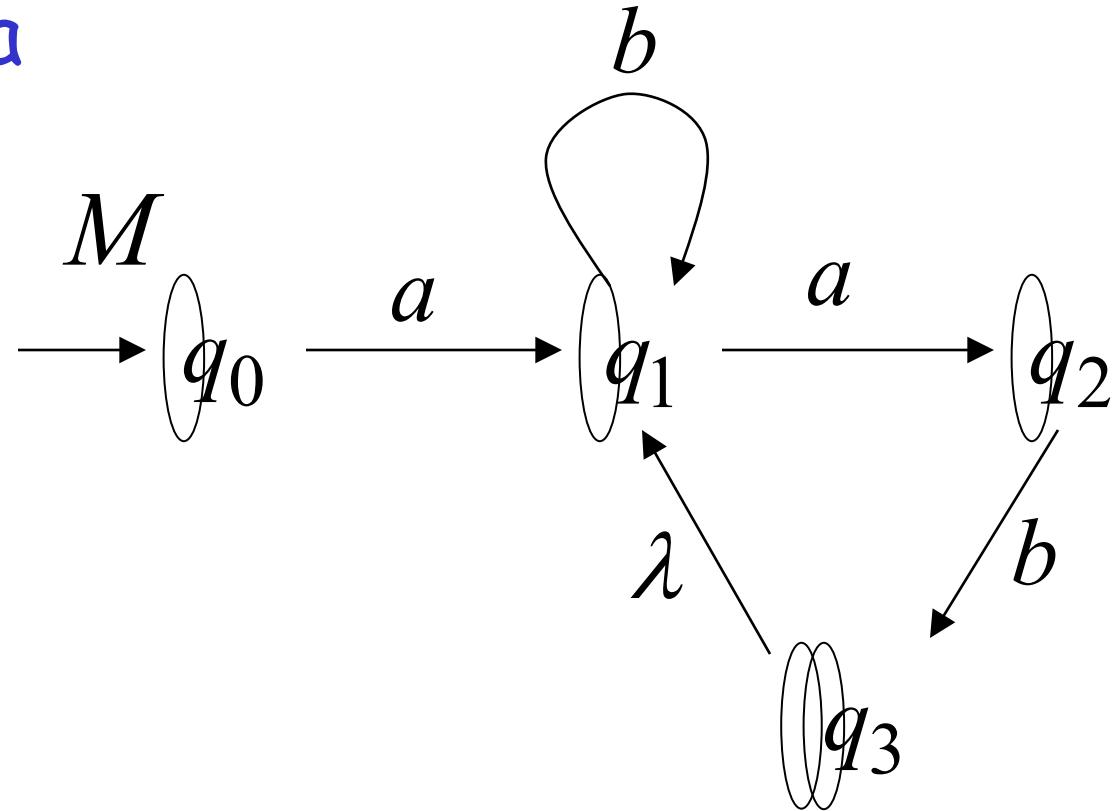
Esempio:



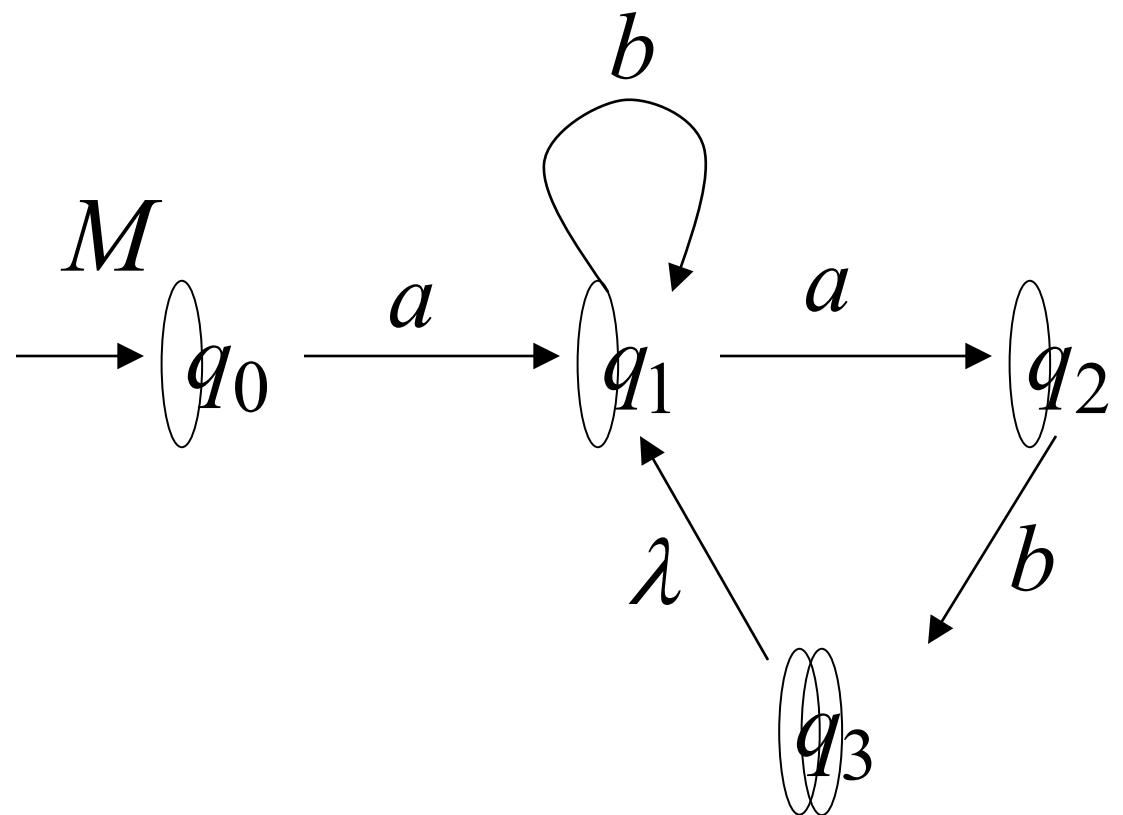
$$L = ab^*ab(b^*ab)^*$$

$$L = L(M)$$

convertiamo M in una right-linear grammatica



$$q_0 \rightarrow aq_1$$

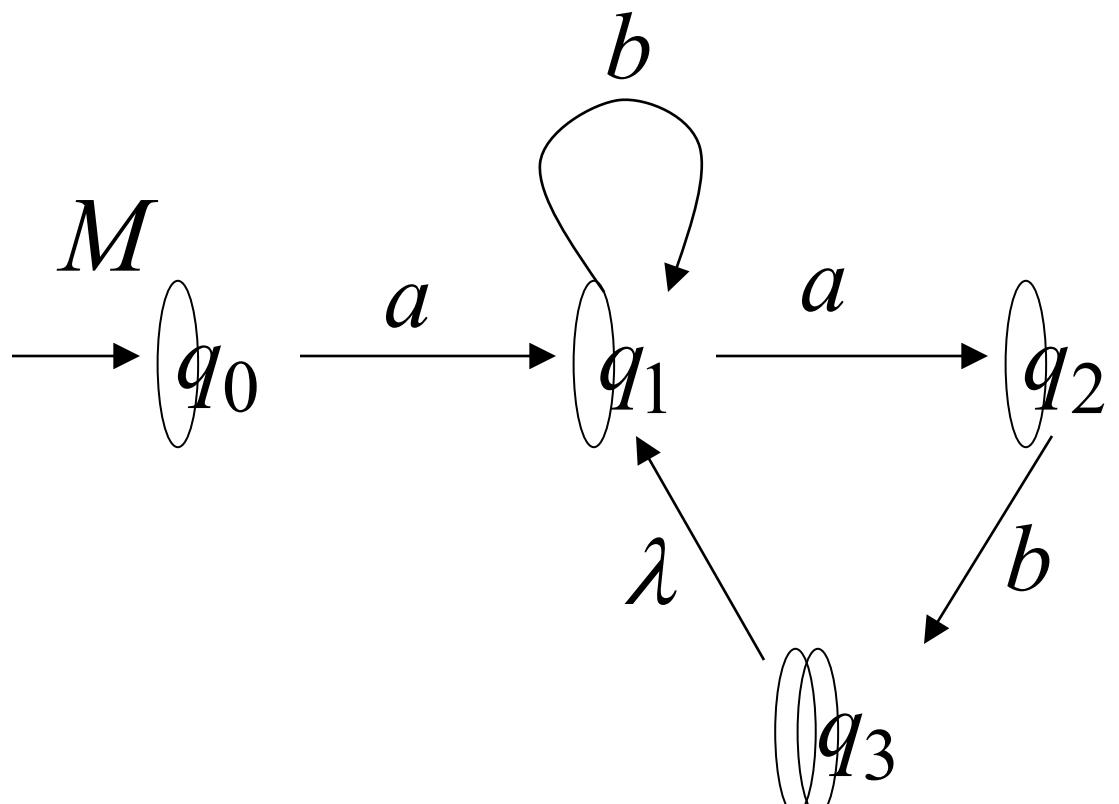


$$q_0 \rightarrow aq_1$$

$$q_1 \rightarrow bq_1$$

$$q_1 \rightarrow aq_2$$

$$\begin{aligned}q_0 &\rightarrow aq_1 \\q_1 &\rightarrow bq_1 \\q_1 &\rightarrow aq_2 \\q_2 &\rightarrow bq_3\end{aligned}$$



$$L(G) = L(M) = L$$

G

$$q_0 \rightarrow aq_1$$

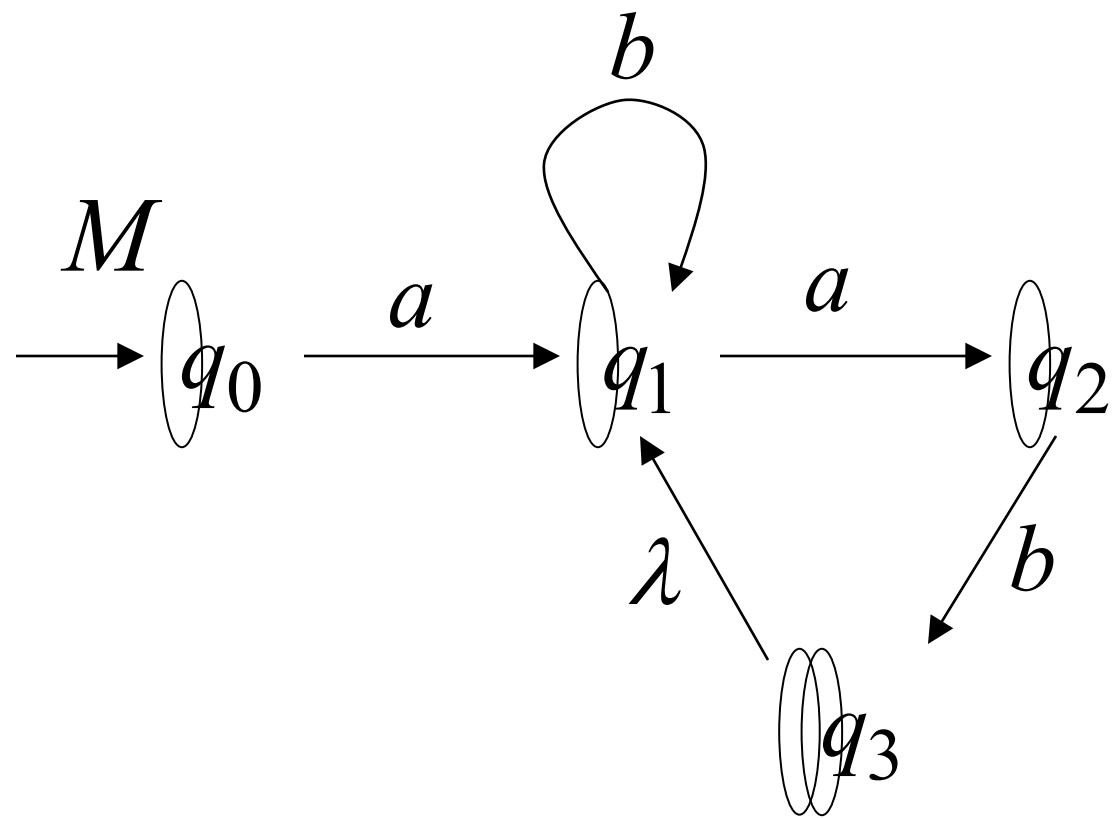
$$q_1 \rightarrow bq_1$$

$$q_1 \rightarrow aq_2$$

$$q_2 \rightarrow bq_3$$

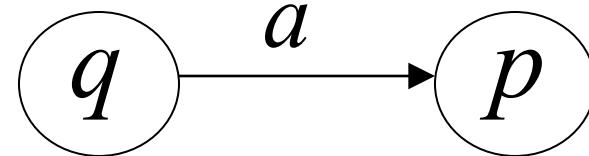
$$q_3 \rightarrow q_1$$

$$q_3 \rightarrow \lambda$$

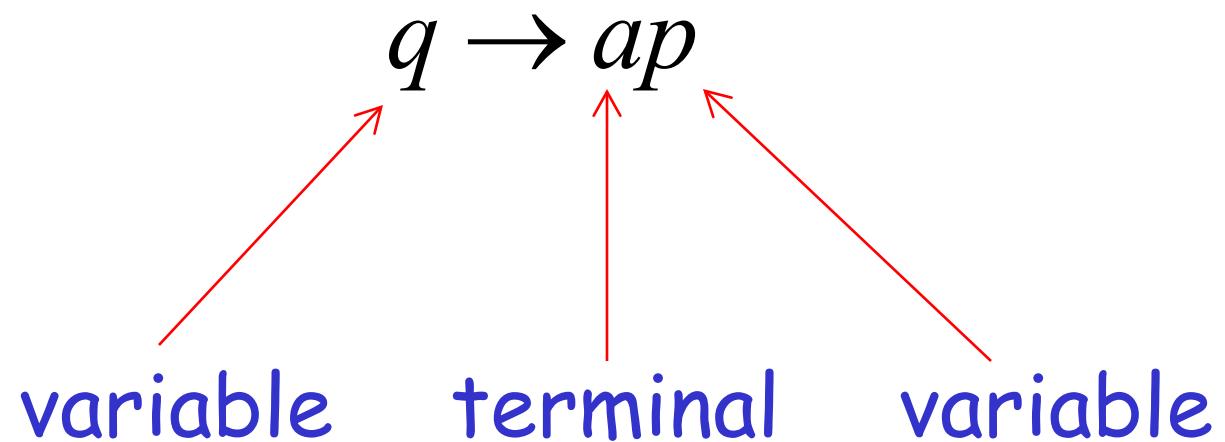


In Generale

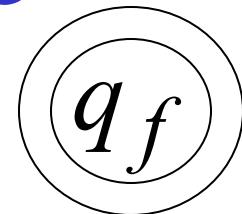
per qualsiasi transizione:



addizioniamo la produzione:



per qualsiasi stato finale :



Addizioniamo la
produzione:

$$q_f \rightarrow \lambda$$

Since G è right-linear grammatica

G è grammatica regolare

con

$$L(G) = L(M) = L$$

linguaggi Non-regolari

(Pumping Lemma)

linguaggi Non-regolari

$$\{a^n b^n : n \geq 0\}$$

$$\{vv^R : v \in \{a,b\}^*\}$$

linguaggi regolari

$$a^*b$$

$$b^*c + a$$

$$b + c(a+b)^*$$

etc...

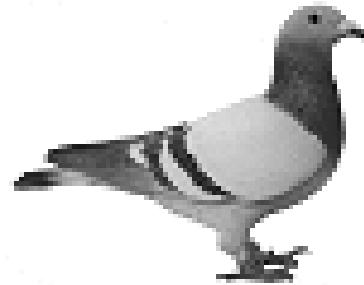
Come possiamo provare che un linguaggio L
non è regolare?

Dobbiamo provare che non vi è
Nessun DFA or NFA or RE
che lo accetta

Difficulty: non è facile da provare
(perchè vi sono infiniti dfa, nfa e re)

Solution: usare il Pumping Lemma !!!

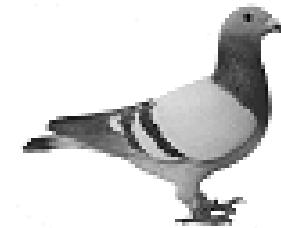
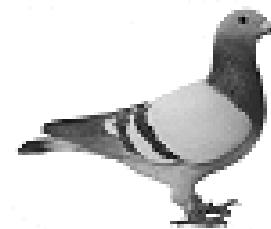
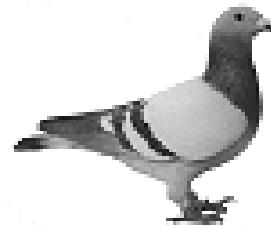
L



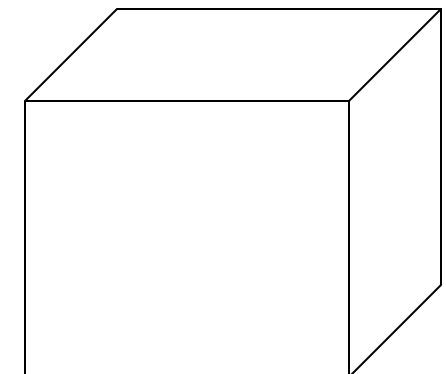
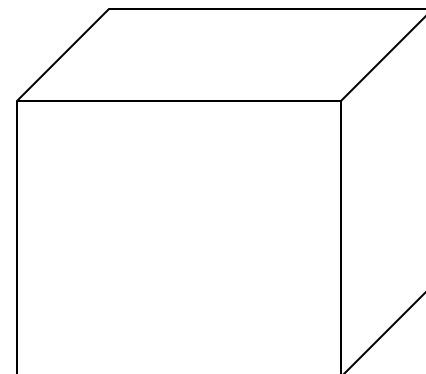
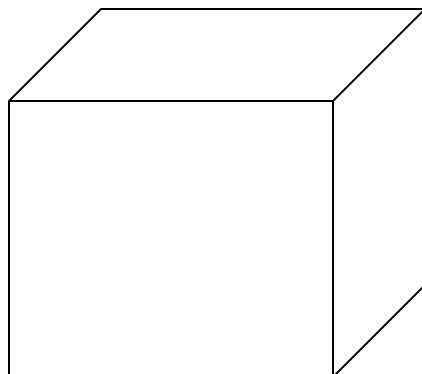
il Pigeonhole Principle

Capelli. Persone.

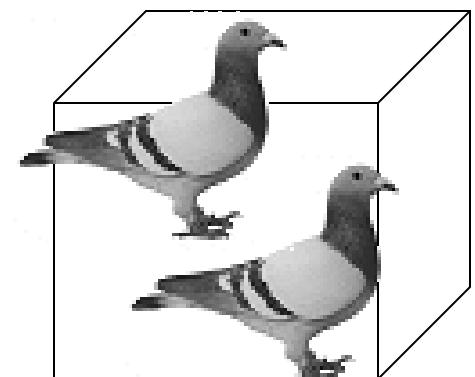
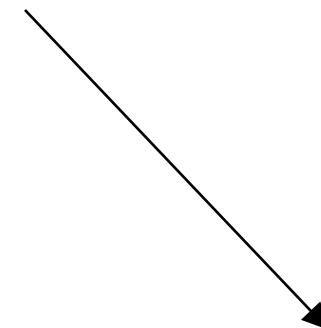
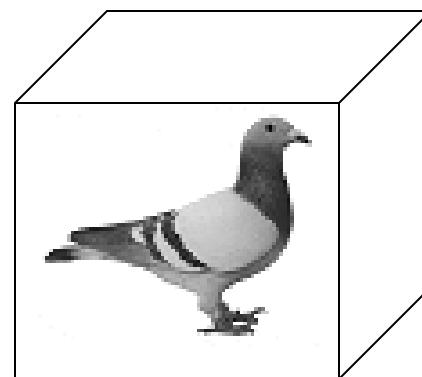
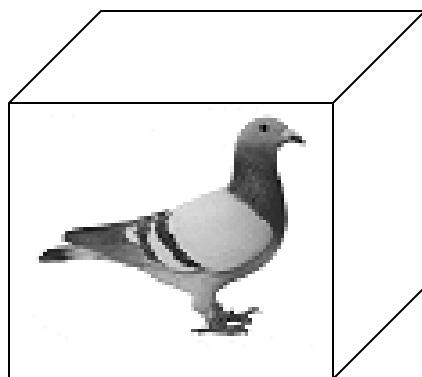
4 pigeons



3 pigeonholes



a pigeonhole deve
Contenere due pigeons



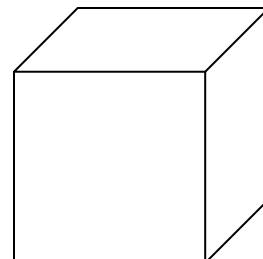
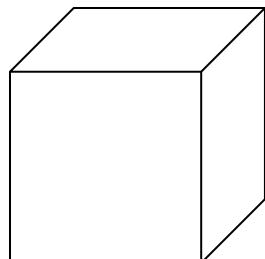
n pigeons



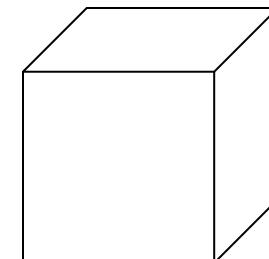
.....



m pigeonholes



.....



$n > m$

il Pigeonhole Principle

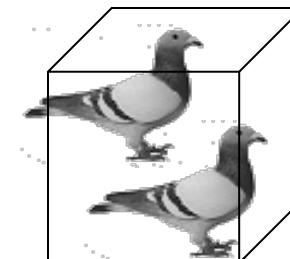
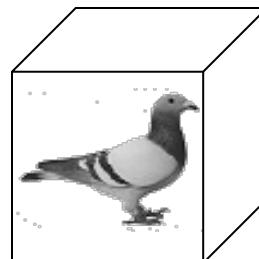
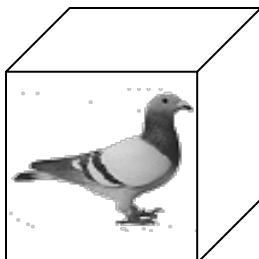
n pigeons

m pigeonholes

$$n > m$$

a pigeonhole deve
Containere minimo
due pigeons

.....

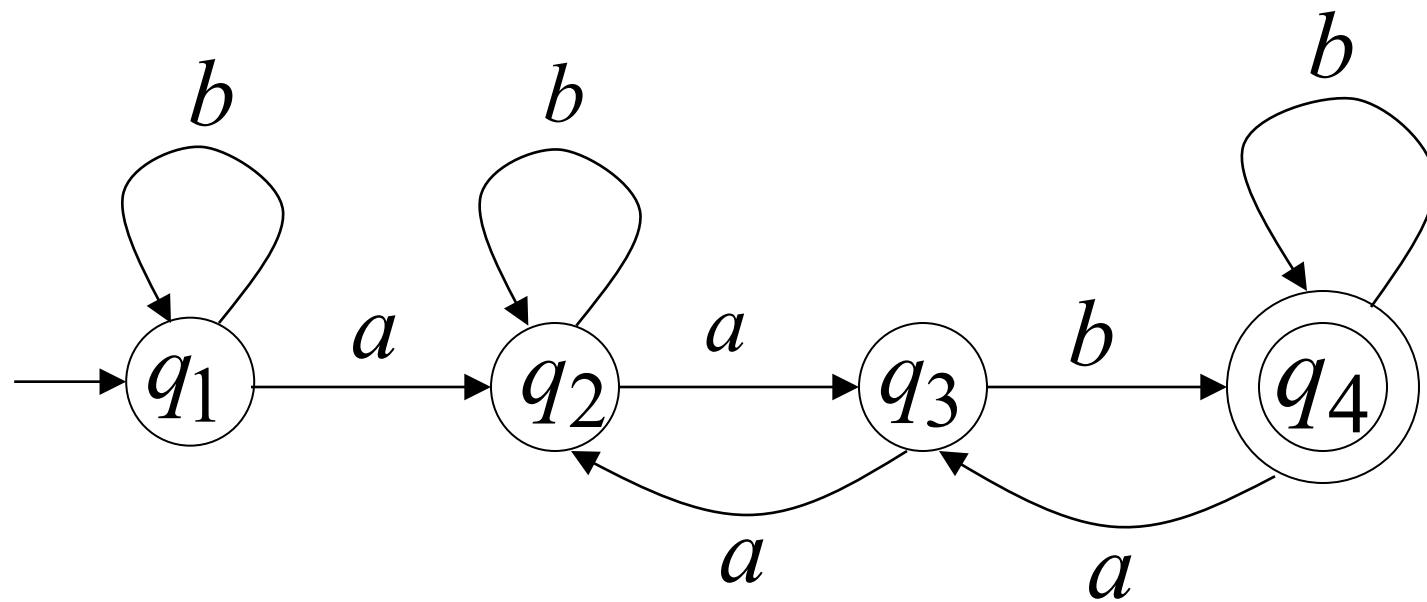


il Pigeonhole Principle

e_i

DFA

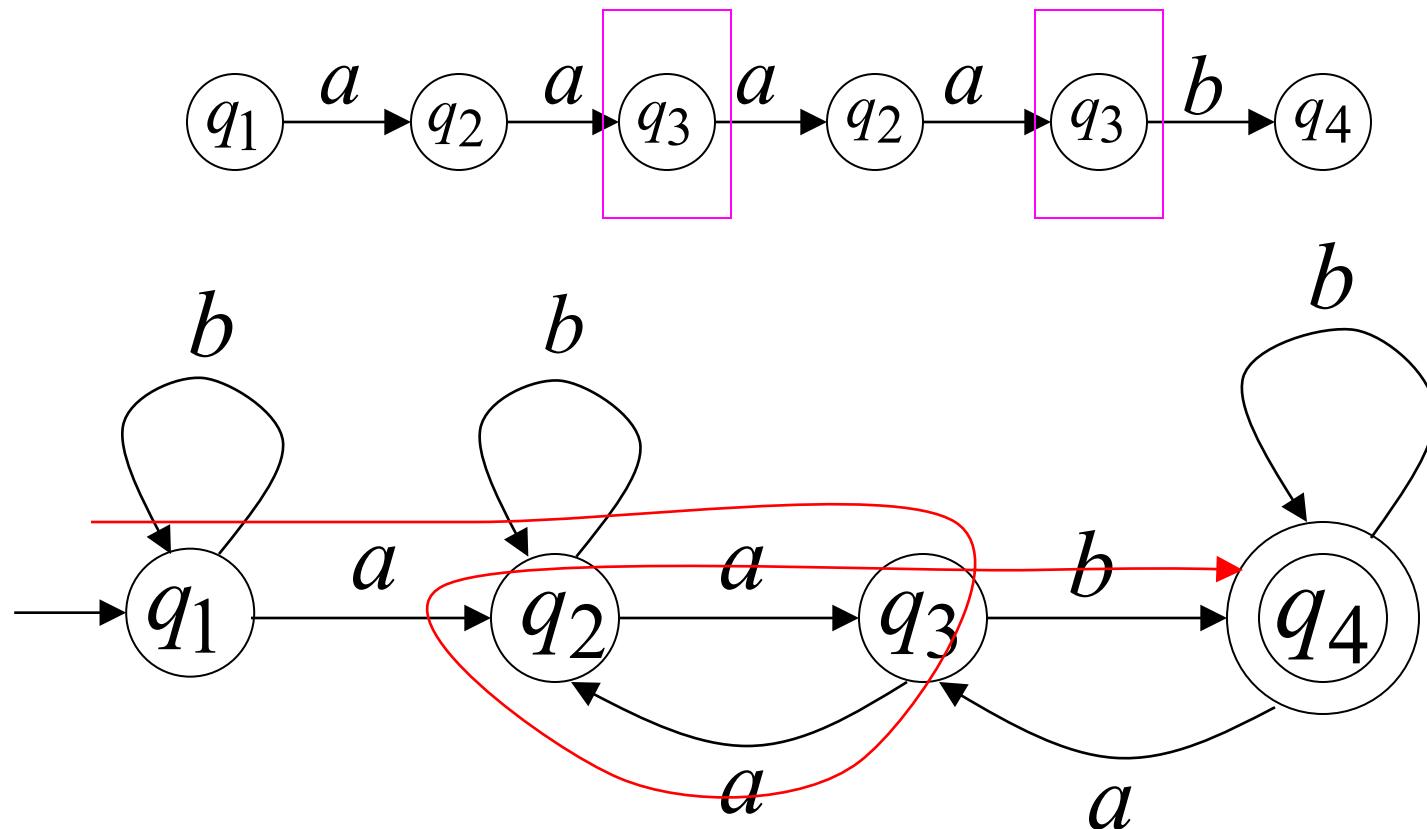
considera un DFA con 4 stati



considera il cammino di una "stringa lunga":
(lunghezza almeno 4)

$aaaab$

uno stato è ripetuto nel cammino di $aaaab$



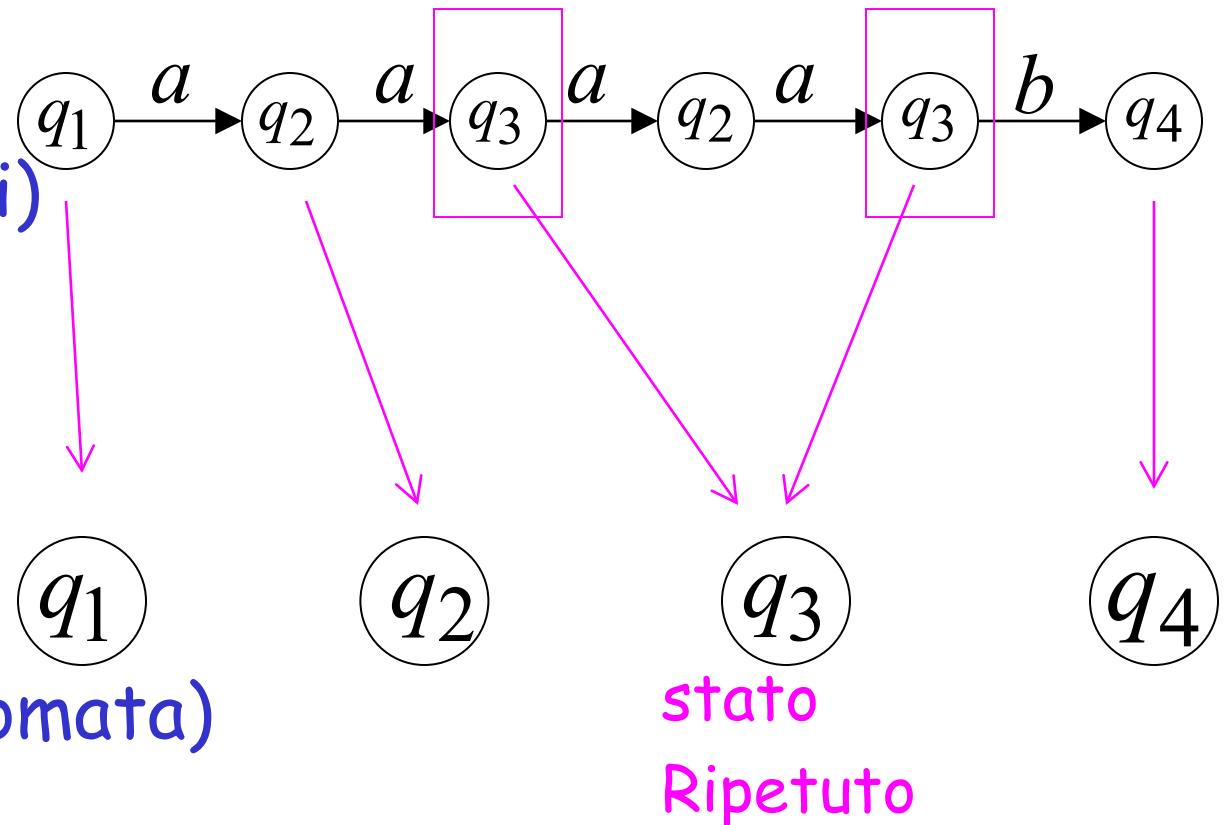
il stato è ripetuto da a, risultato del pigeonhole principle

Pigeons:
(cammino stati)

Sono più degli

Cesti:
(stati dell'automata)

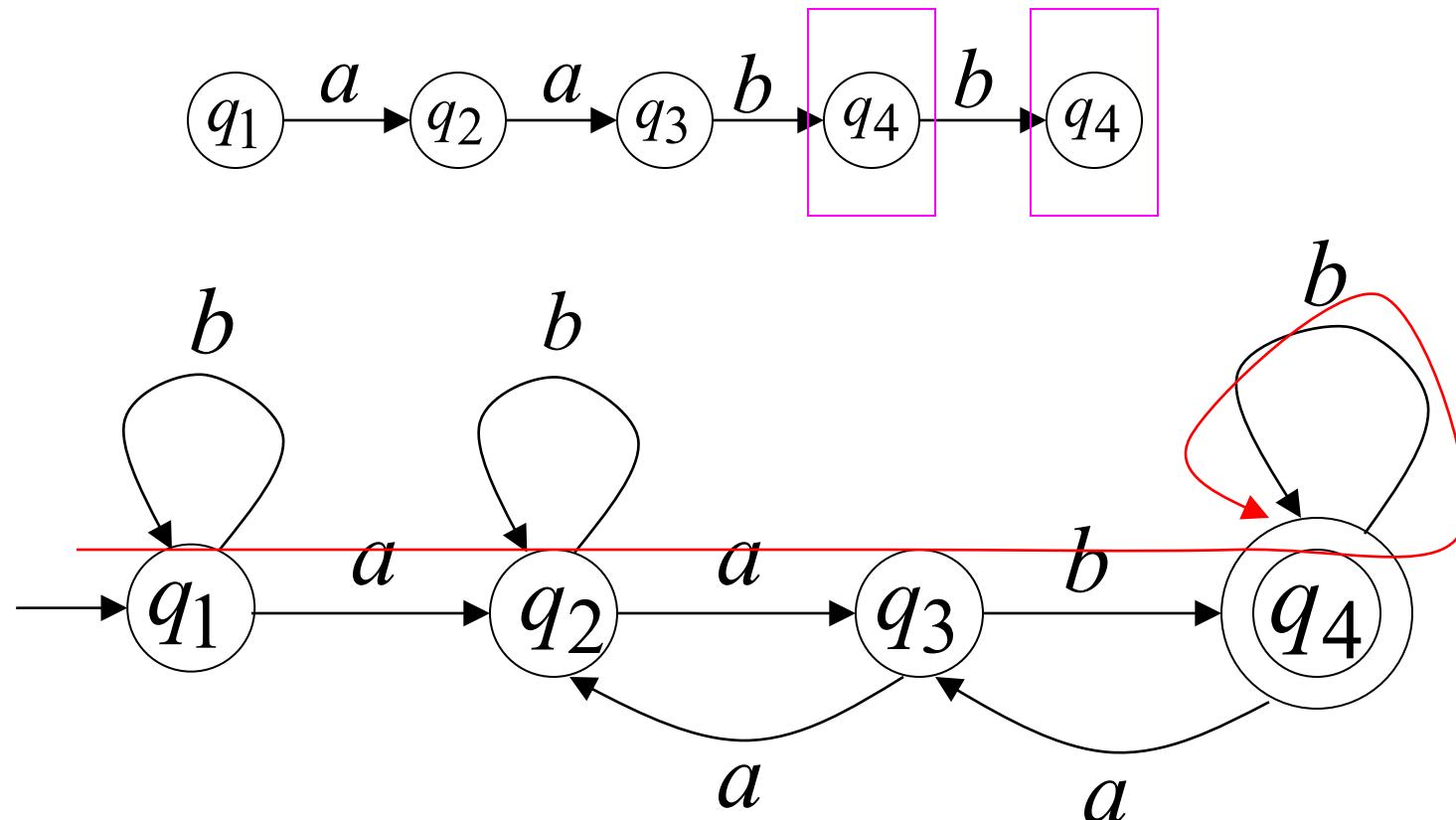
cammino di
aaaab



considera il cammino di a "long" stringa:
(lunghezza almeno 4) $aabb$

Dal pigeonhole principle:

uno stato è ripetuto nel cammino di $aabb$



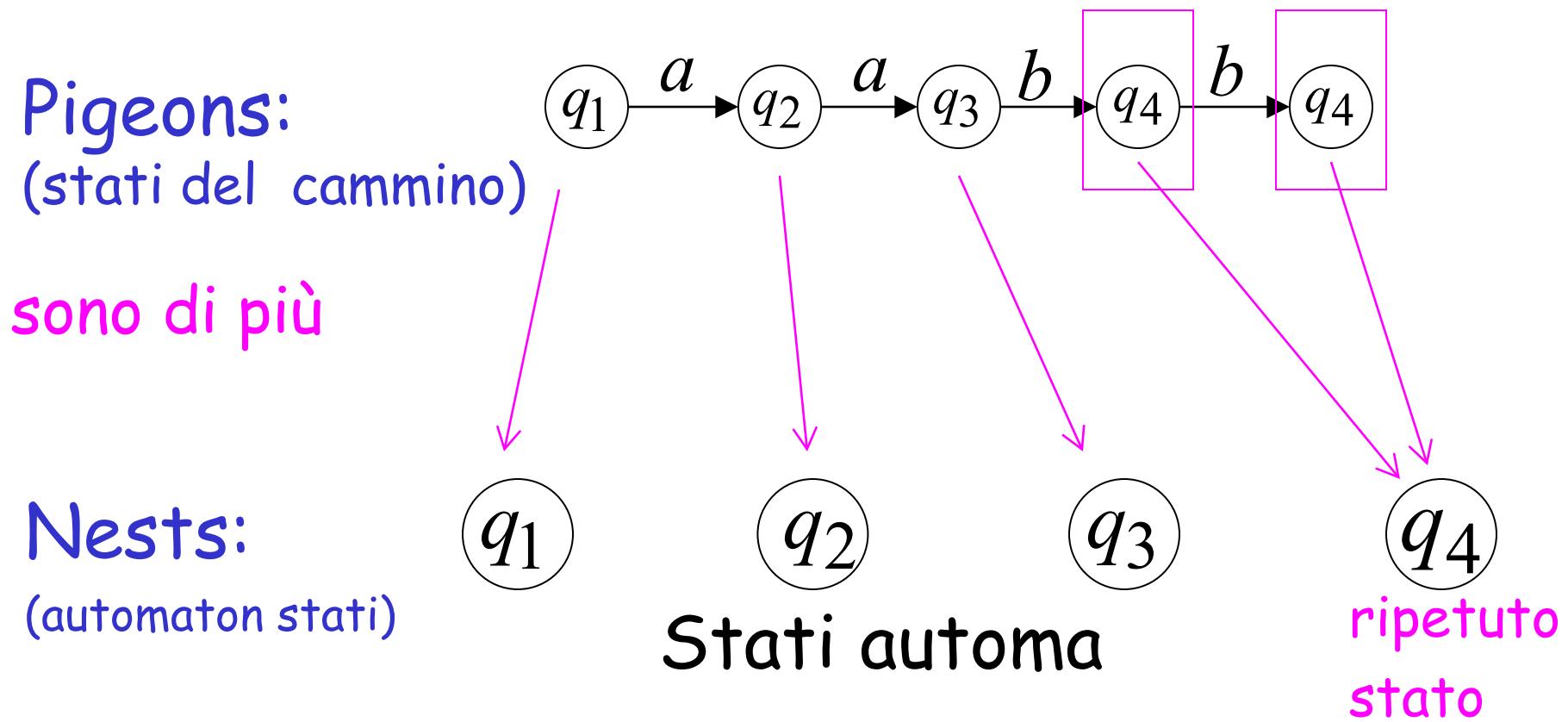
il stato è ripetuto come risultato del pigeonhole principle

Pigeons:
(stati del cammino)

sono di più

Nests:
(automaton stati)

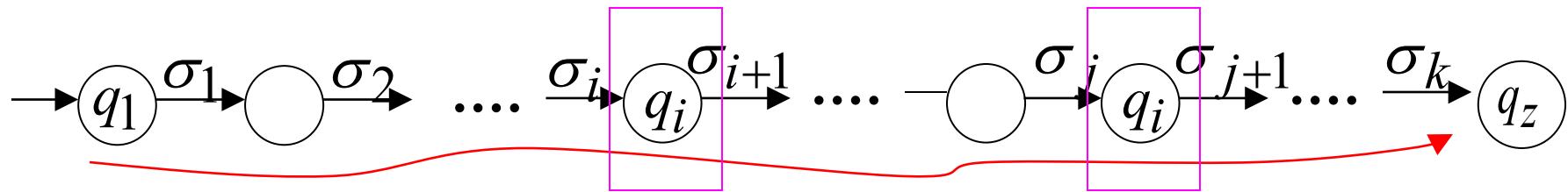
cammino di $aabb$



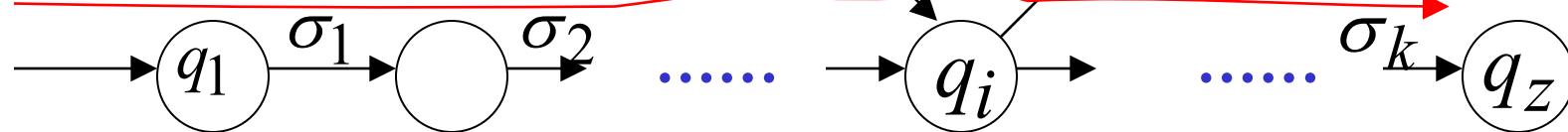
In Generale: se $|w| \geq \# \text{states of DFA}$

Per il pigeonhole principle,
uno stato è ripetuto nel cammino W

cammino di $w = \sigma_1 \sigma_2 \cdots \sigma_k$

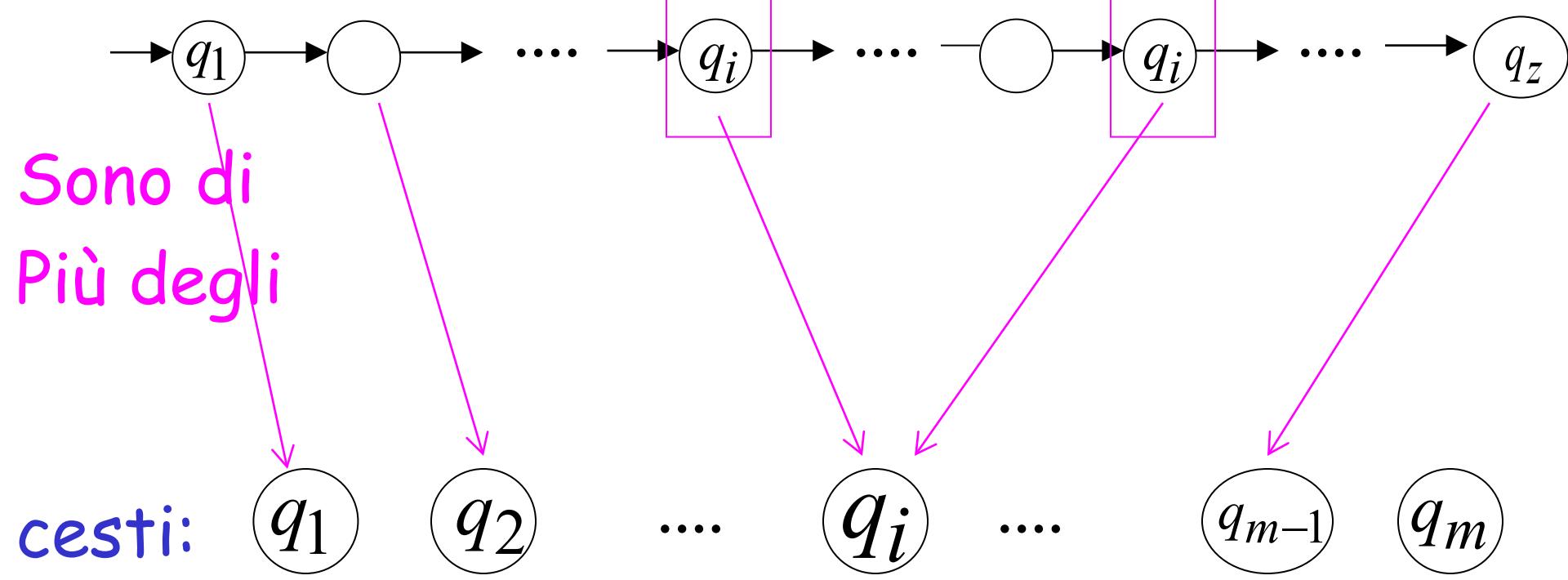


arbitrario DFA



$$|w| \geq \#\text{states of DFA} = m$$

Pigeons: (stati del cammino) cammino di w



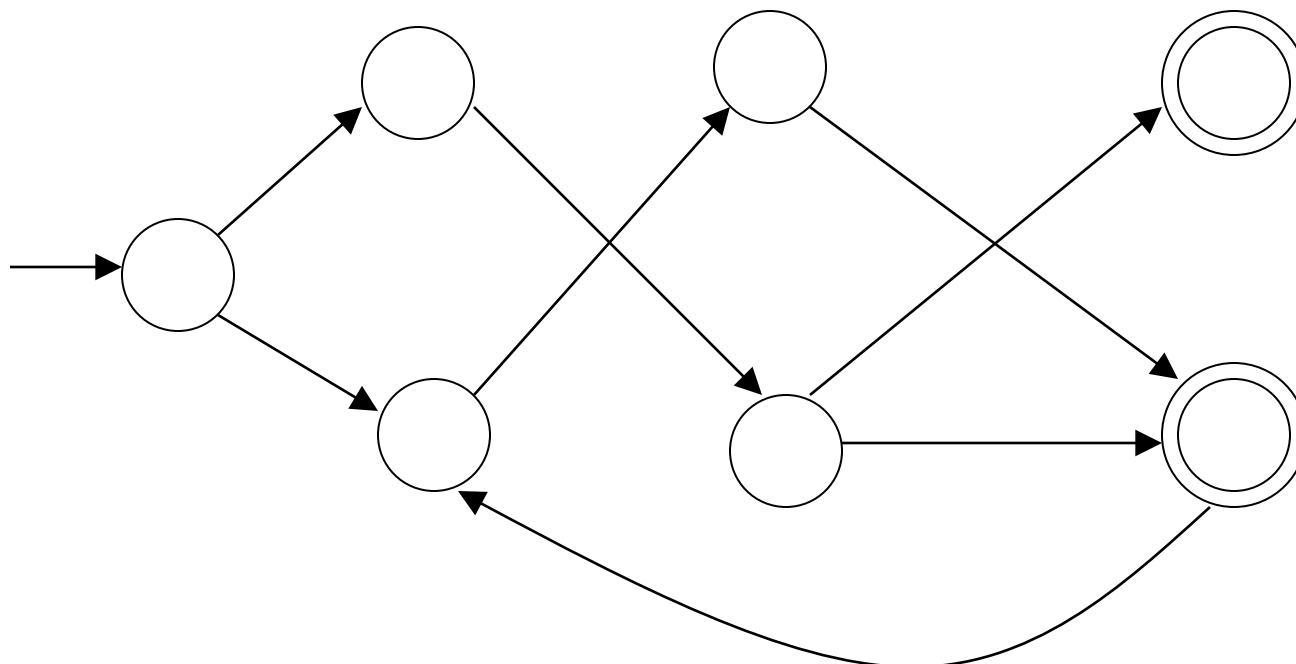
cestini:
(stati dell
automata)

uno stato è
ripetuto

il Pumping Lemma

prendi un linguaggio regolare infinito L
(contiene un numero infinito di stringhe)

Sia un DFa che accetta L



m
stati

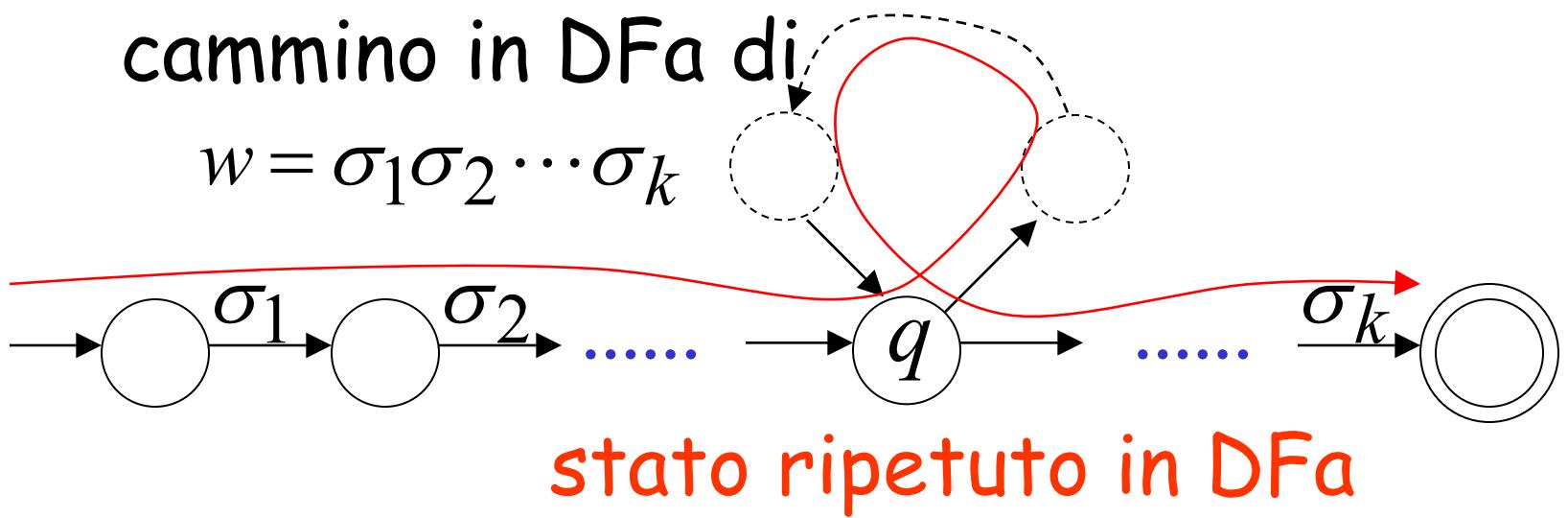
prendiamo una stringa $w \in L$ con $|w| \geq m$

(numero di
stati del DFa)

Almeno uno stato è ripetuto
nel cammino di w

cammino in DFa di

$$w = \sigma_1 \sigma_2 \cdots \sigma_k$$



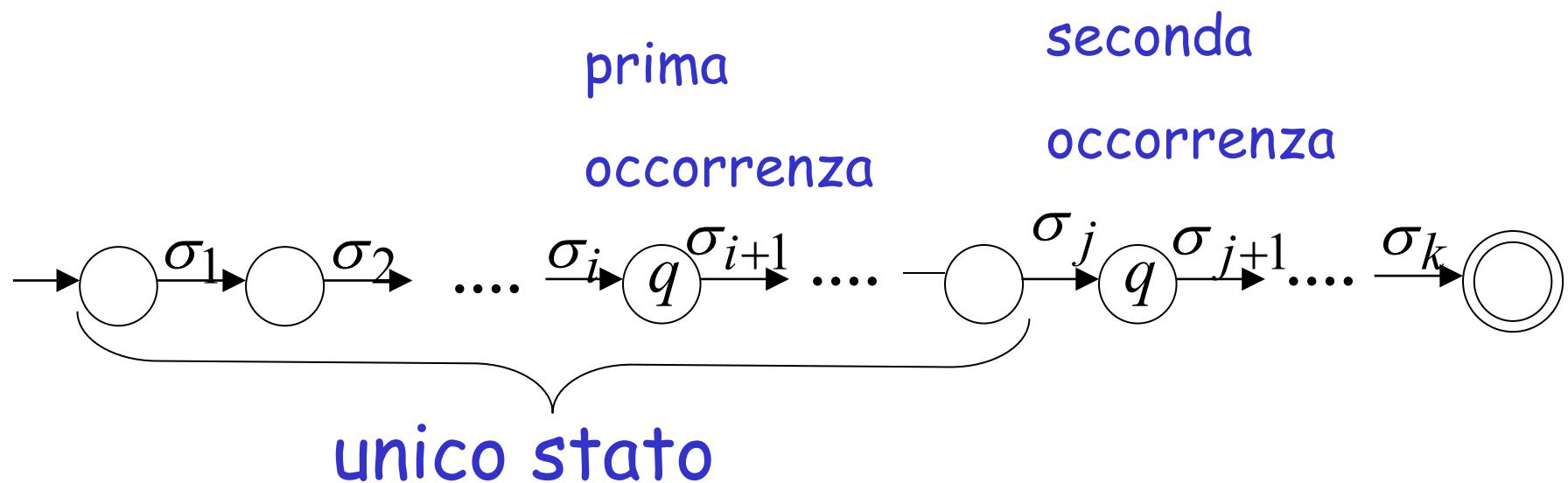
Ci saranno molti stati ripetuti

prendiamo il primo stato ripetuto

q

In una dimensione il cammino di:

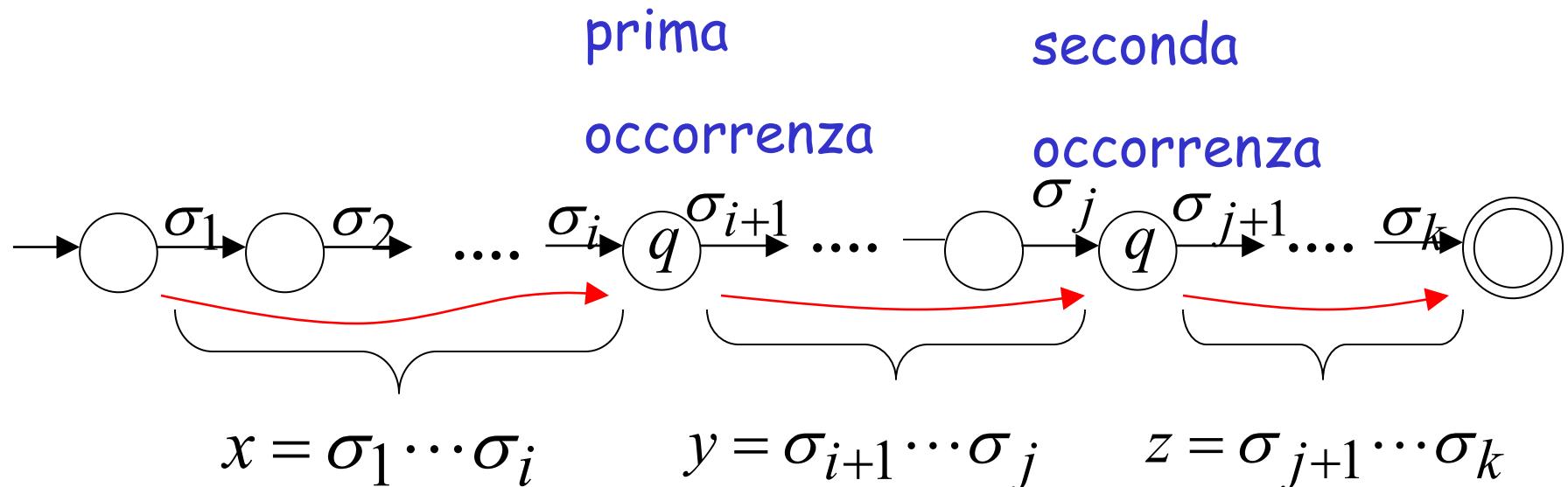
w



Possiamo scrivere

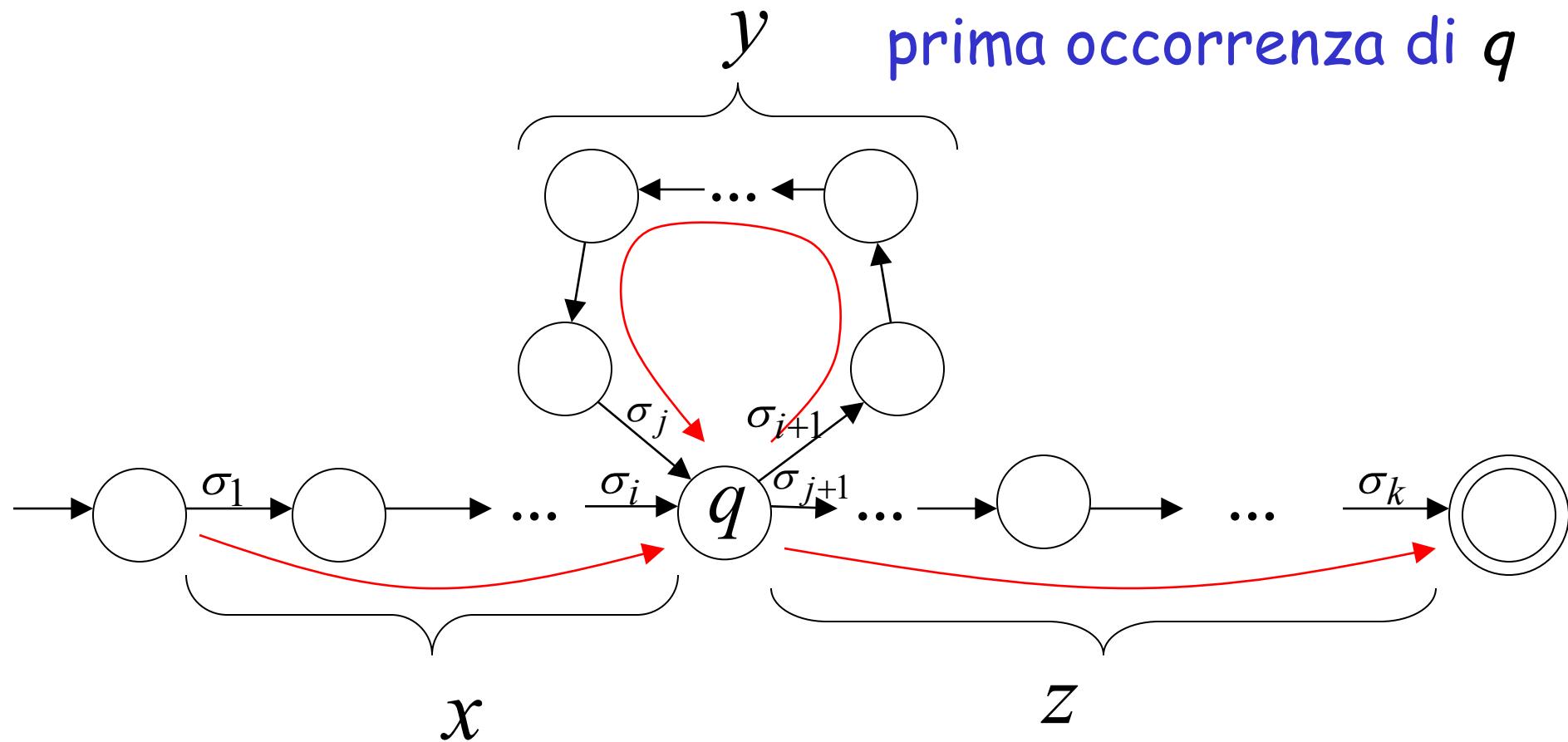
$$w = xyz$$

Una dimensione del cammino di : w

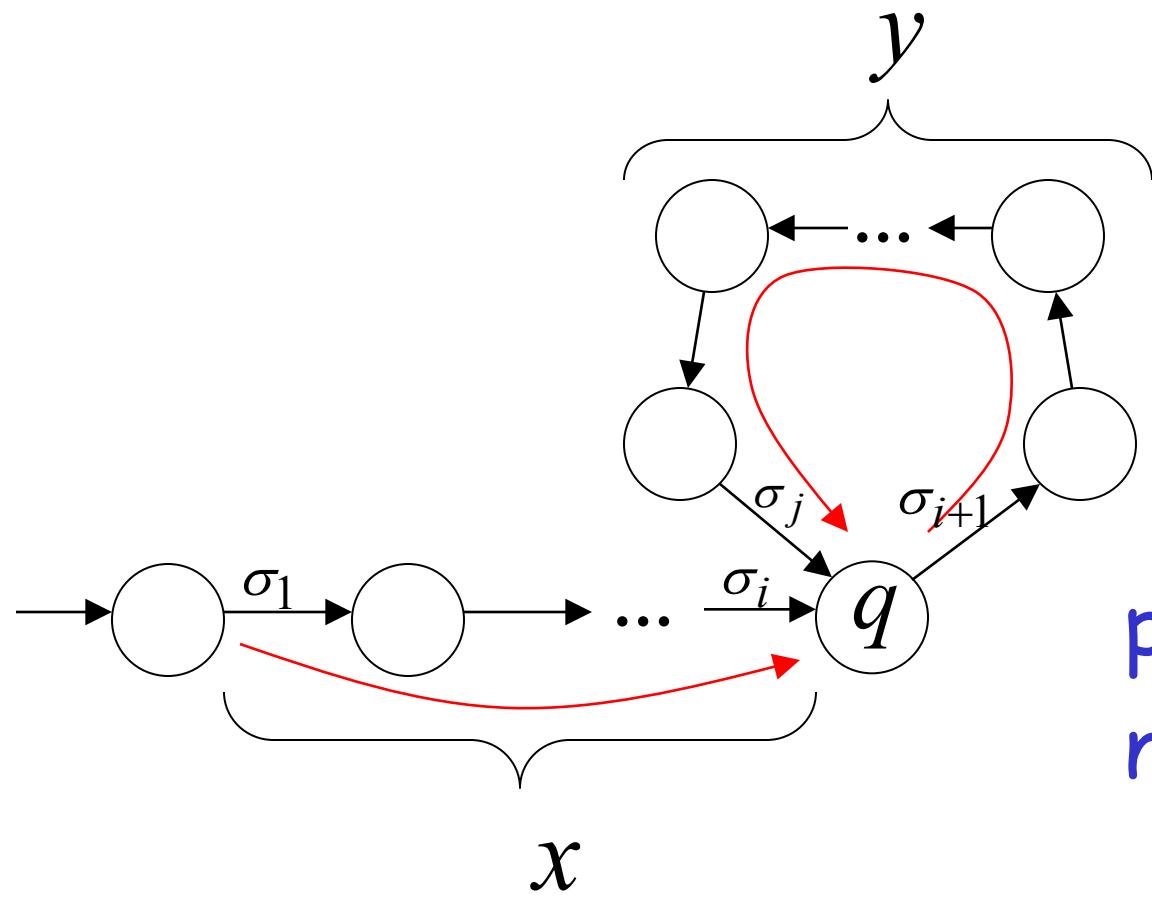


Nel DFa: $w = x \ y \ z$

contiene solo
prima occorrenza di q



osservazione: lunghezza $|x y| \leq m$ numero di stati del DFA

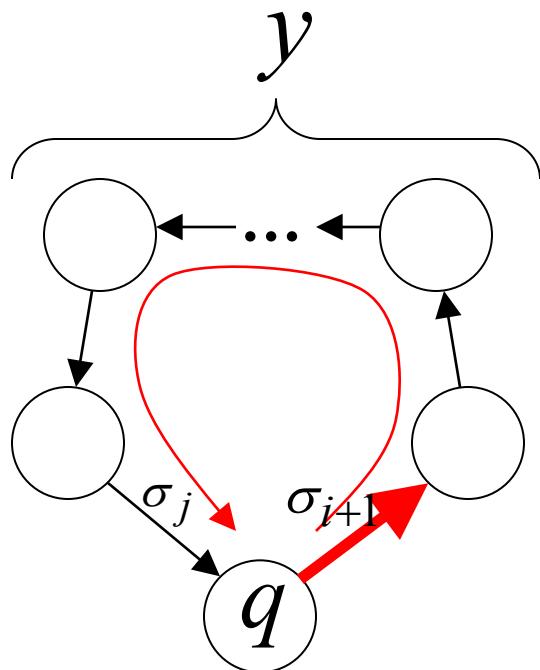


unici stati

poichè, in xy
nessuno è stato
ripetuto (eccetto q)

osservazione: lunghezza $|y| \geq 1$

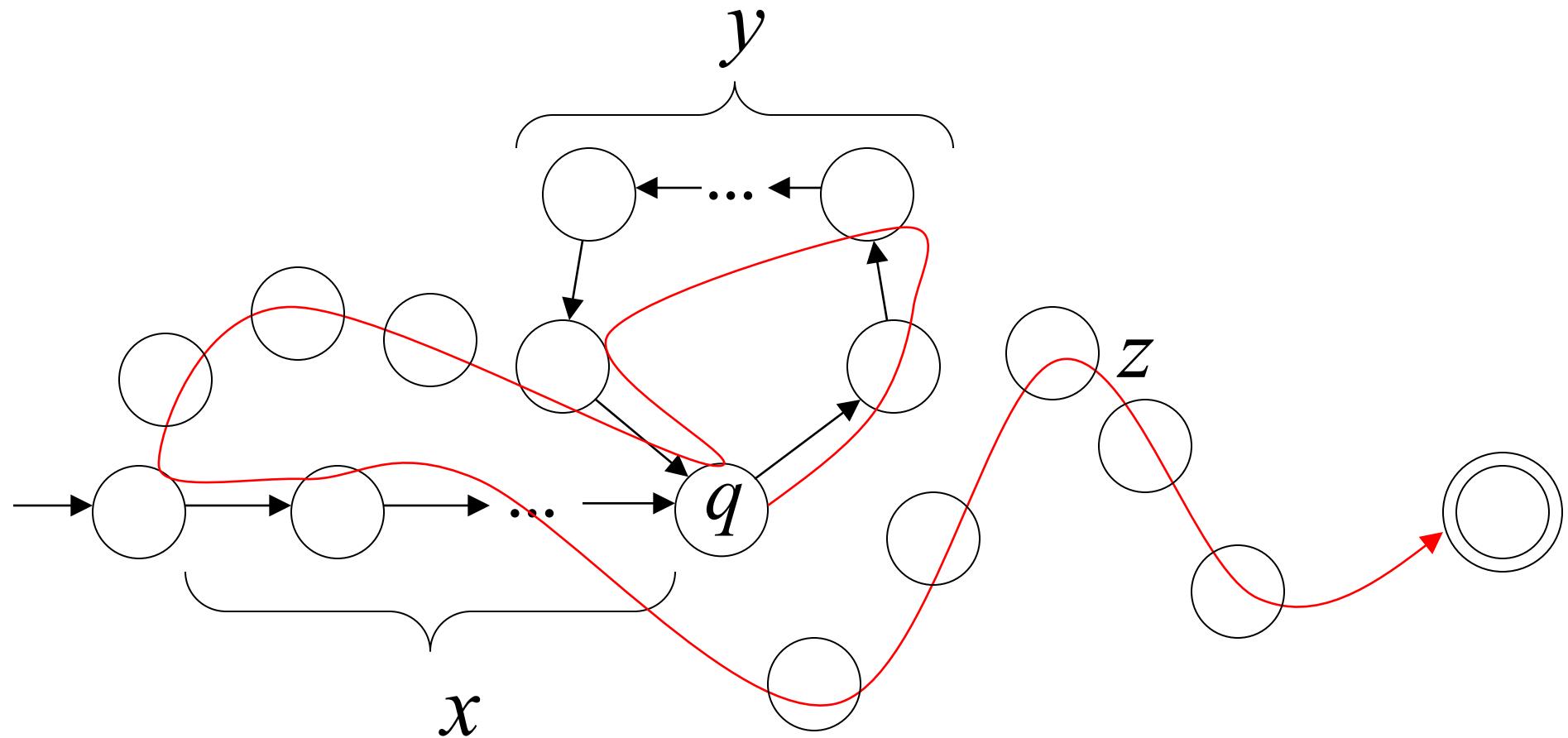
Vi è almeno un loop



Non badiamo alla forma della stringa

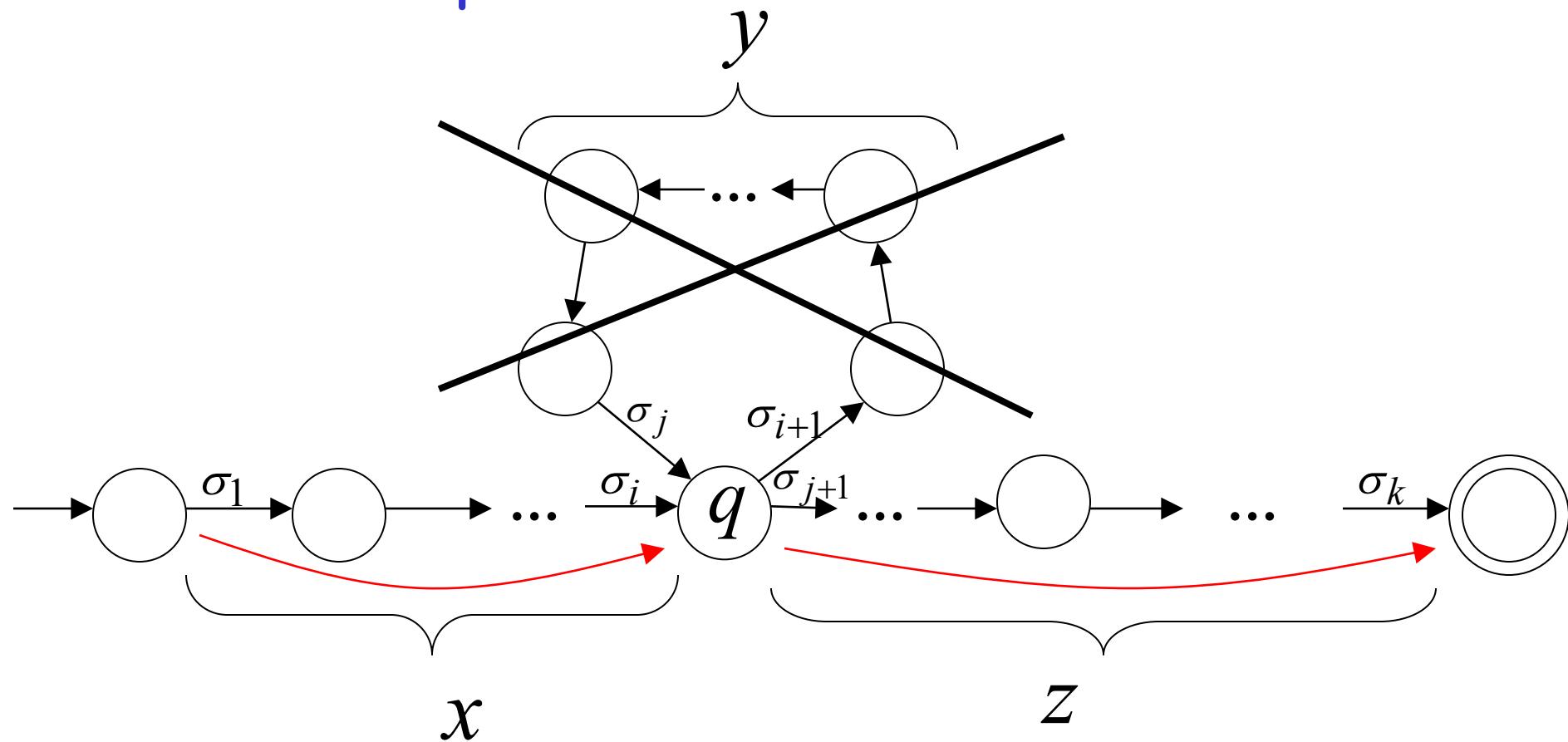
z

z può avere pezzi di cammino di x and y



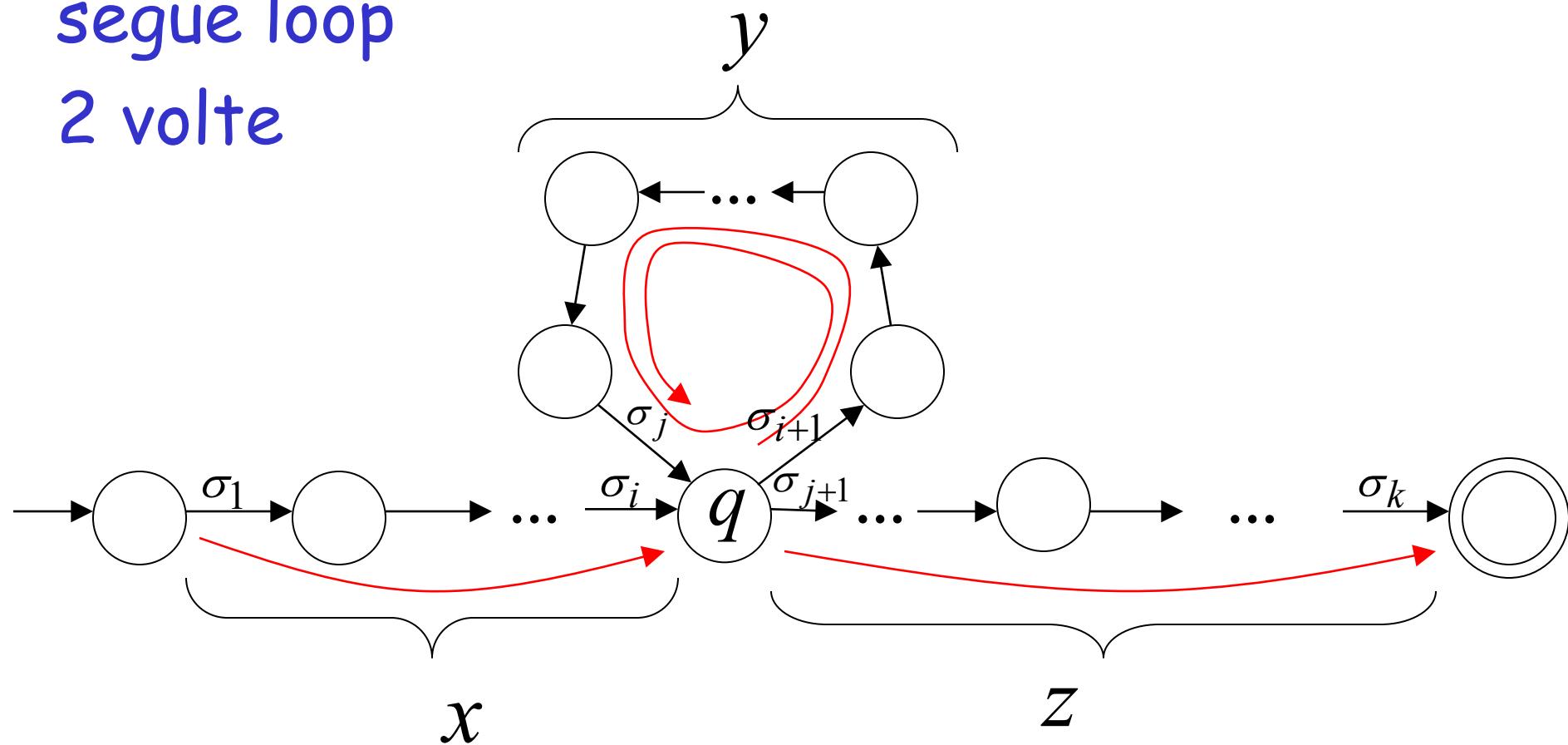
stringa addizionale: la stringa $x z$
è accettata

Non fa il loop



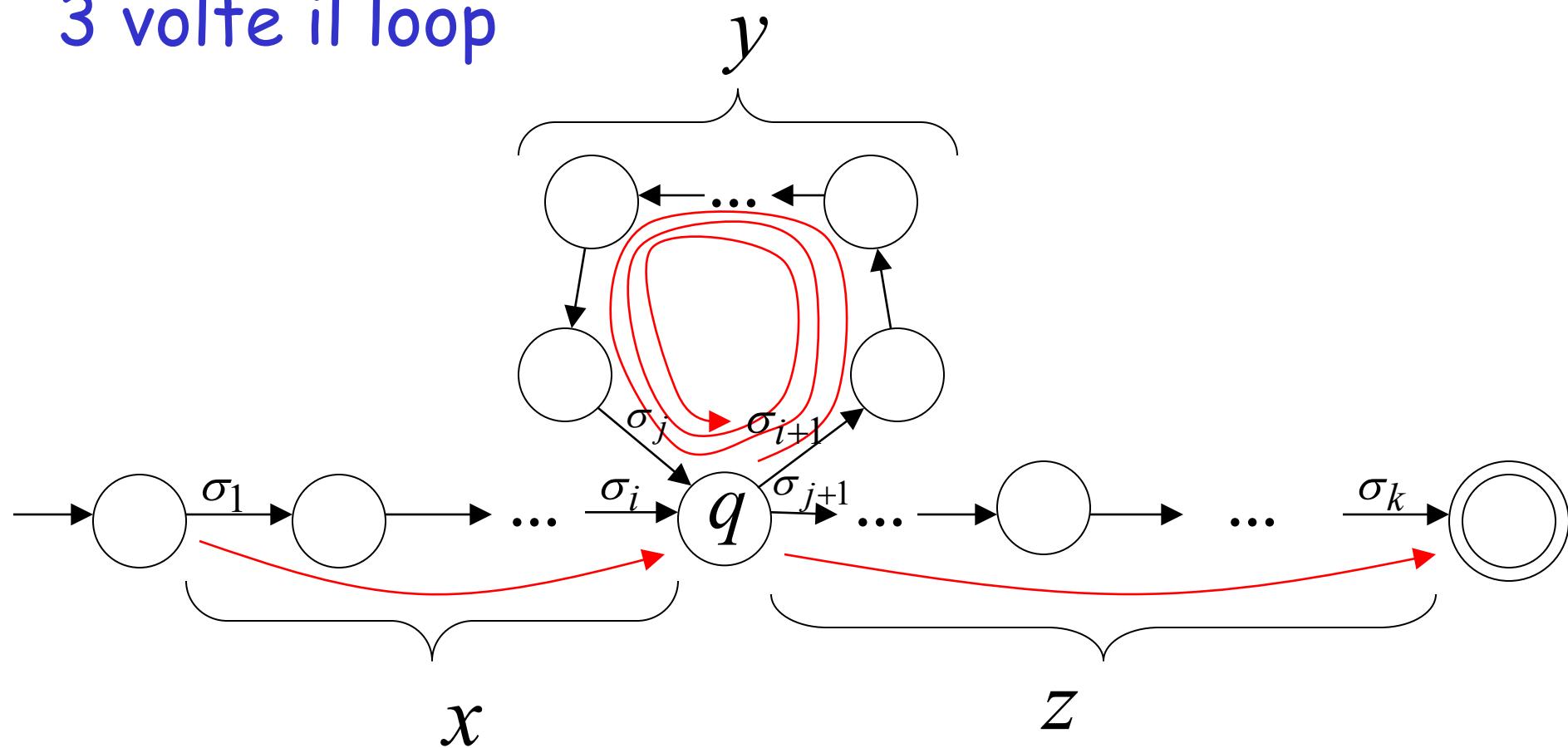
stringa addizionale : la stringa $x y y z$
è accettata

segue loop
2 volte



addizionale stringa: la stringa $x y y y z$
è accettata

3 volte il loop



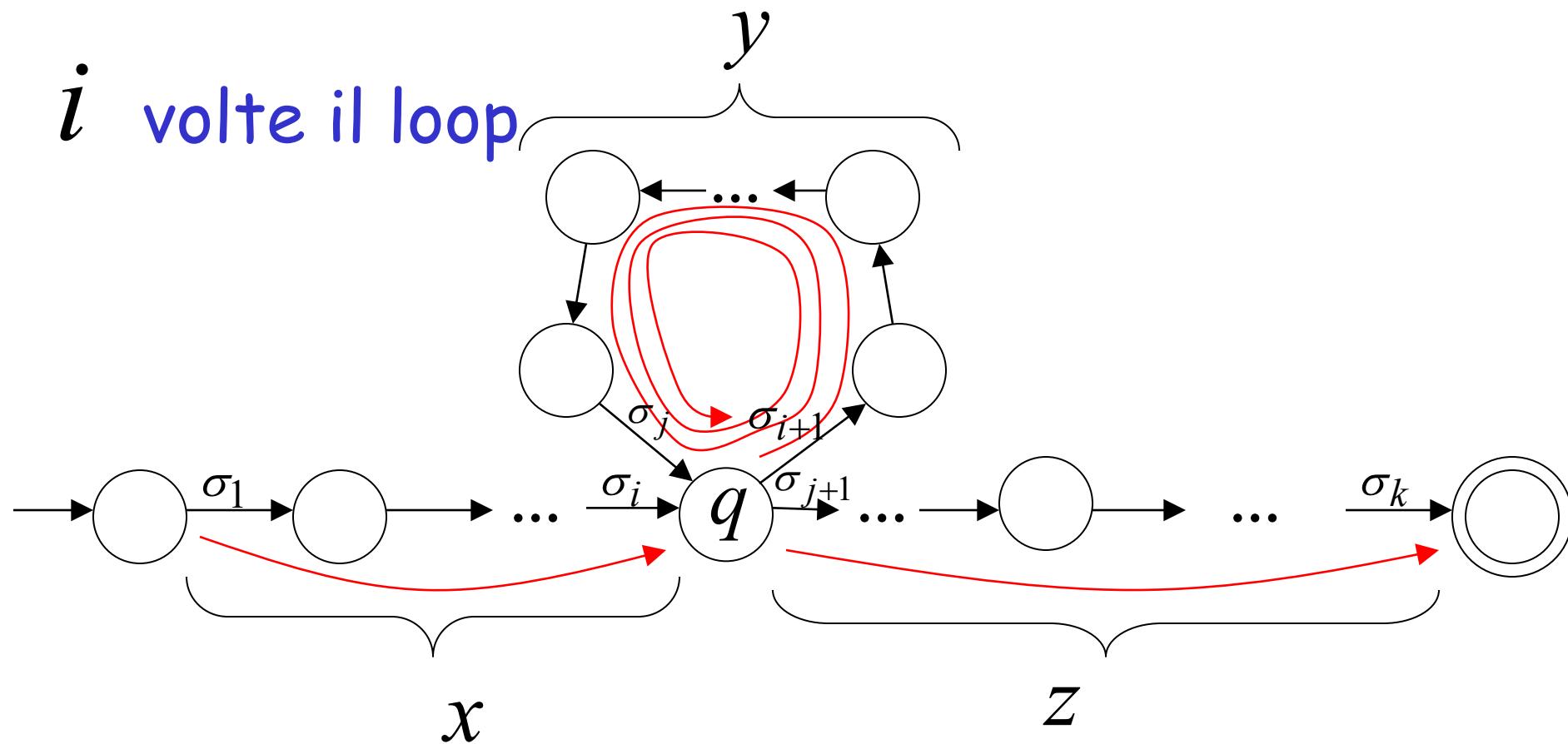
In Generale:

la stringa

$x y^i z$

è accettata $i = 0, 1, 2, \dots$

i volte il loop

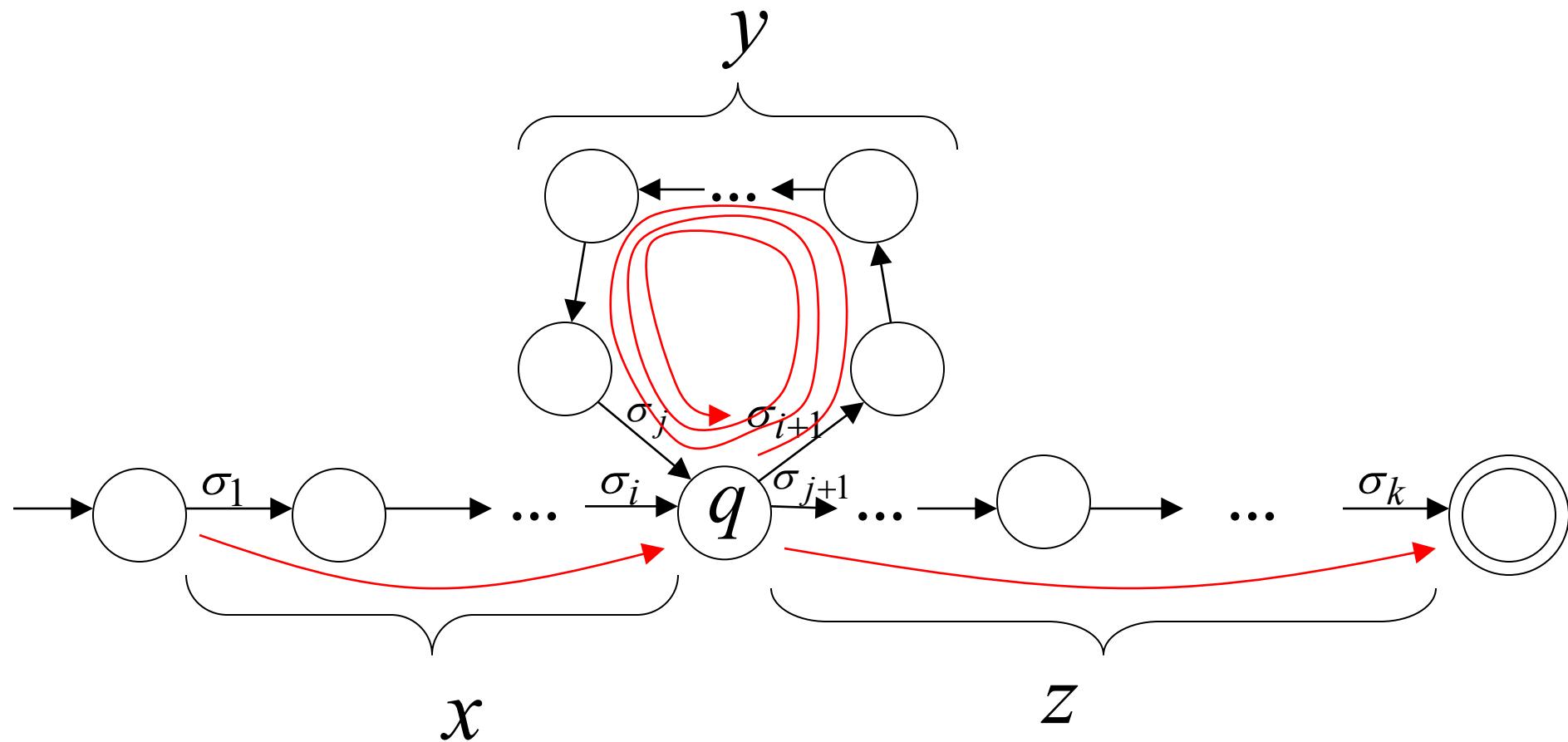


quindi:

$$x \ y^i \ z \in L$$

$$i = 0, 1, 2, \dots$$

linguaggio accettato dal DFA



il Pumping Lemma:

- dato un linguaggio regolare infinito L
- esiste un intero m (lunghezza critica)
- per ogni stringa $w \in L$ con lunghezza $|w| \geq m$
- possiamo scrivere $w = x y z$
- con $|x y| \leq m$ e $|y| \geq 1$
- tale che: $x y^i z \in L \quad i = 0, 1, 2, \dots$

nel libro sipster :

lunghezza Critica = m lunghezza Pumping p

applicazioni applicazioni

del
Pumping Lemma

osservazione:

ogni linguaggio di dimensione finita è regolare

(possiamo facilmente costruire un NFA
che accetta ogni stringa nel linguaggio)

quindi, ogni linguaggio non-regolare
è di dimensione infinita

(contiene un infinito numero di stringhe)

supponiamo vogliamo provare che

Un linguaggio infinito L non è regolare

1. assumiamo l' opposto: L è regolare
2. il pumping lemma deve valere per L
3. usiamo il pumping lemma per ottenere una contraddizione
4. quindi, L non è regolare

Spiegazione Step 3: come avere una contraddizione

1. Let m sia la lunghezza critica for L
2. Scegliamo una stringa particolare $w \in L$ che soddisfa la condizione di lunghezza $|w| \geq m$
3. scrivere $w = xyz$
4. mostriamo che $w' = xy^i z \notin L$ Per qualche $i \neq 1$
5. Questo ci dà una contraddizione, poichè dal pumping lemma $w' = xy^i z \in L$

Note: È sufficiente mostrare che
solo una stringa $w \in L$
genera una contraddizione

Non dobbiamo ottenere
contradizioni per ogni $w \in L$

Esempi di applicazioni del Pumping Lemma

teorema: il linguaggio $L = \{a^n b^n : n \geq 0\}$ non è regolare

dim: Usa il Pumping Lemma

$$L = \{a^n b^n : n \geq 0\}$$

assumiamo per contraddizione
che L è un linguaggio regolare

Since L è infinito
Possiamo applicare il Pumping Lemma

$$L = \{a^n b^n : n \geq 0\}$$

sia m la lunghezza critica per L

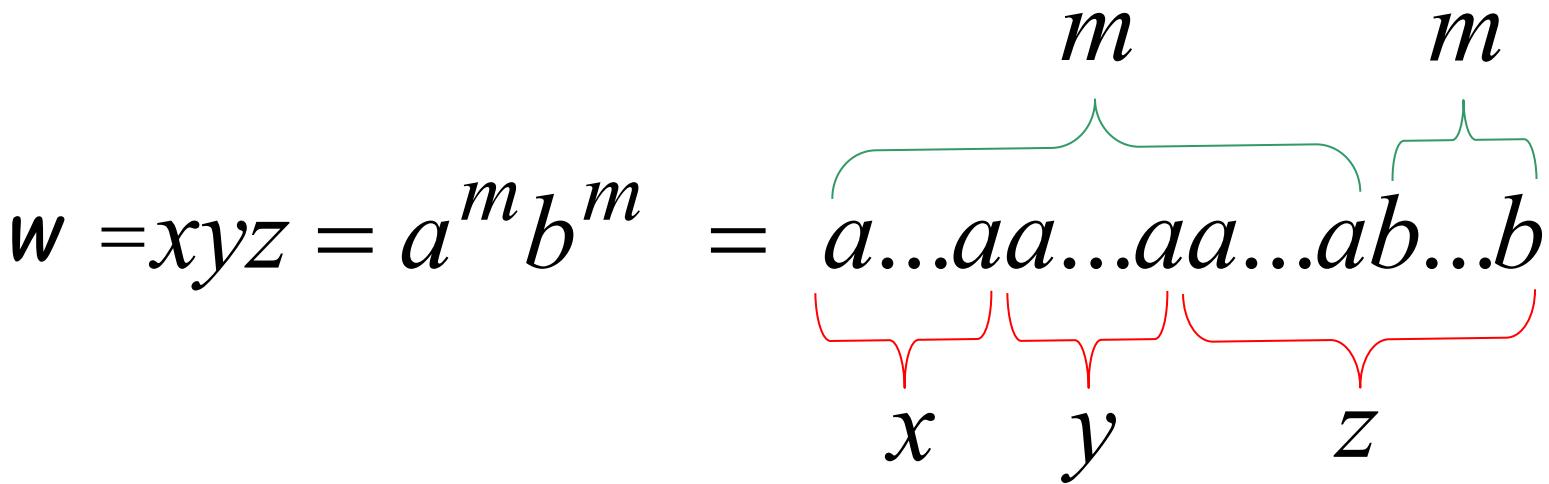
Prendiamo a stringa w such che: $w \in L$
e lunghezza $|w| \geq m$

prendiamo $w = a^m b^m$

Dal Pumping Lemma:

possiamo scrivere $w = a^m b^m = x y z$

Con lunghezza $|x y| \leq m, |y| \geq 1$



allora: $y = a^k, 1 \leq k \leq m$

$$x \ y \ z = a^m b^m \quad y = a^k, \quad 1 \leq k \leq m$$

dal Pumping Lemma:

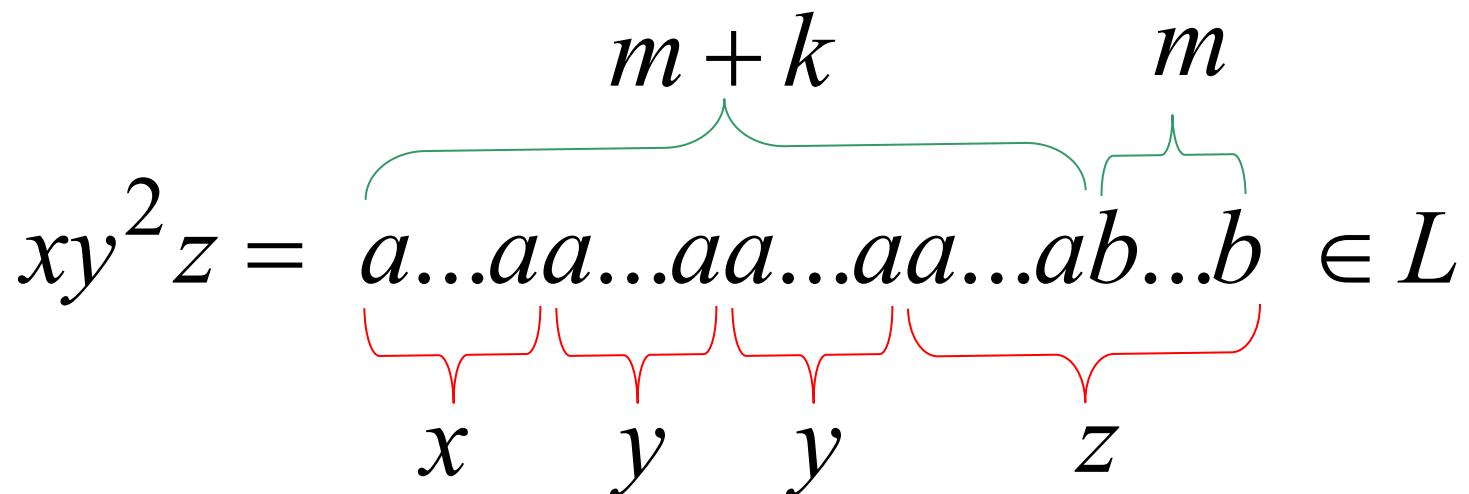
$$x \ y^i \ z \in L$$

$$i = 0, 1, 2, \dots$$

allora: $x \ y^2 \ z \in L$

$$x \ y \ z = a^m b^m \quad y = a^k, \quad 1 \leq k \leq m$$

Dal Pumping Lemma: $x \ y^2 \ z \in L$



allora: $a^{m+k} b^m \in L$

aaabbb =xyz

1 caso $x=aa$ $y=a$ $z=bbb$

aa aa bbb

2 caso $x=aaab$ $y=b$ $z=b$

aaab bb b

3 caso

$x=aa$ $y=ab$ $z=bb$

aa ababab bb

$$a^{m+k}b^m \in L \quad k \geq 1$$

MA: $L = \{a^n b^n : n \geq 0\}$



$$a^{m+k}b^m \notin L$$

contradizione!!!

quindi: l'assunzione che che L
è un linguaggio regolare
non è vera

Conclusione: L non è un linguaggio regolare

END dim

linguaggio Non-regolare $\{a^n b^n : n \geq 0\}$

Linguaggio regolare

$L(a^* b^*)$

a^*b^*

$aayabybb = xyz$ $y=a$

aa aa a bbb

aaa bbbbb

$y=ab$

aaabb



23922 - ROMA - Part. del frammento di bassorilievo, rappres. le Stagioni - Museo Vaticano Ripr. int. - Anders

Precisazioni unioni e intersezione automi

Per studenti 2020

Chiusura rispetto intersezione

automa M_1

DFA per L_1

automa M_2

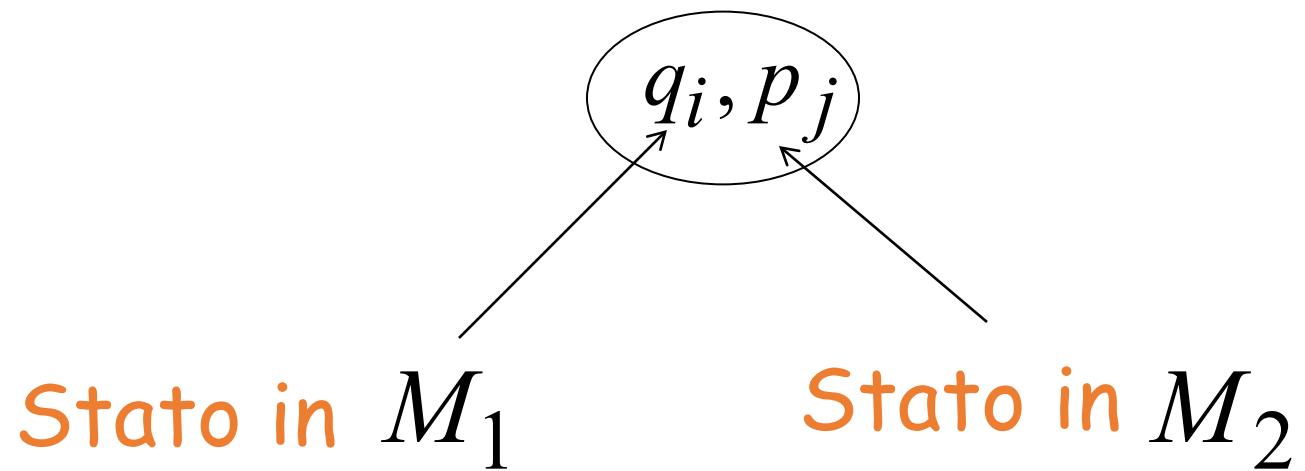
DFA per L_2

«Automi completi»

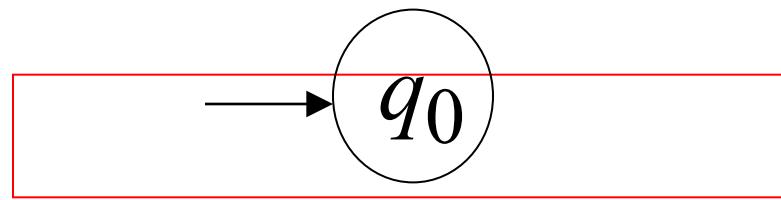
Costruiamo un DFA M che accetta $L_1 \cap L_2$

M Simula in parallelo M_1 e M_2

Stati in M

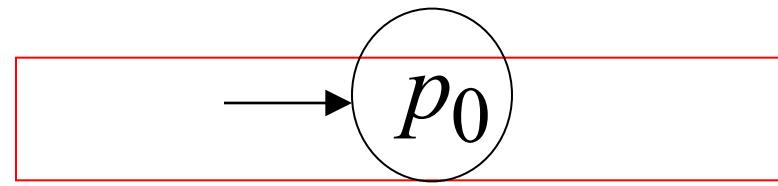


DFA M_1



stato iniziale

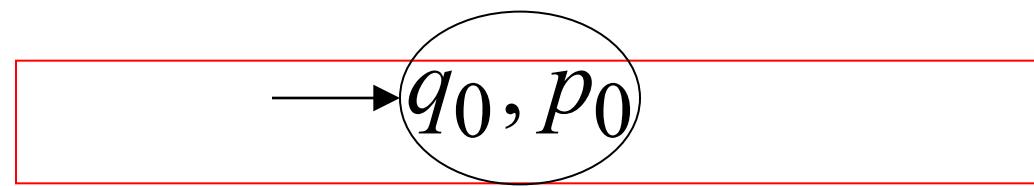
DFA M_2



stato iniziale

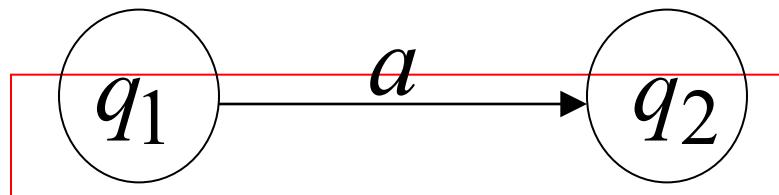


DFA M



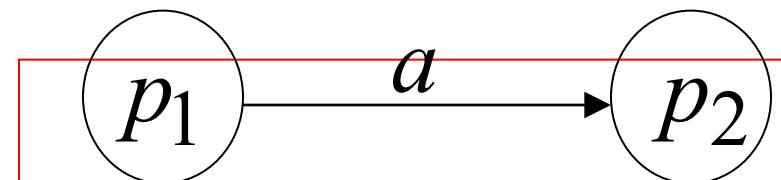
nuovo stato iniziale

DFA M_1



transizione

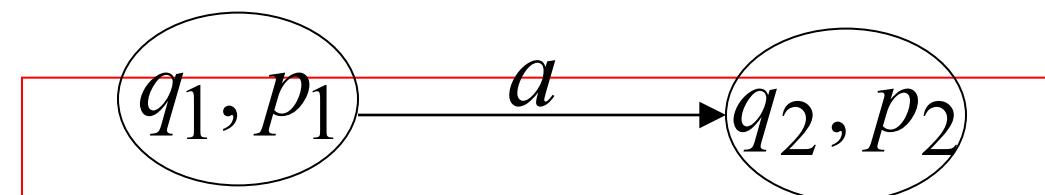
DFA M_2



transizione

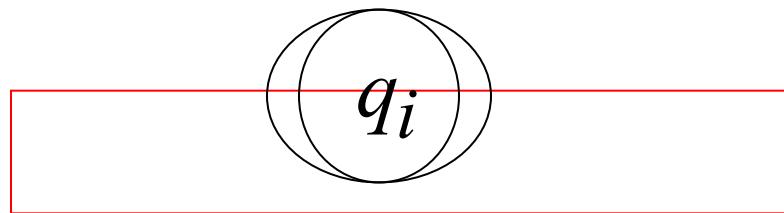


DFA M



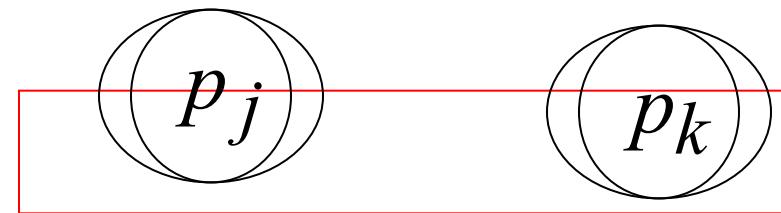
Nuova transizione

DFA M_1



accettazione stato

DFA M_2



accettazione stati

DFA M

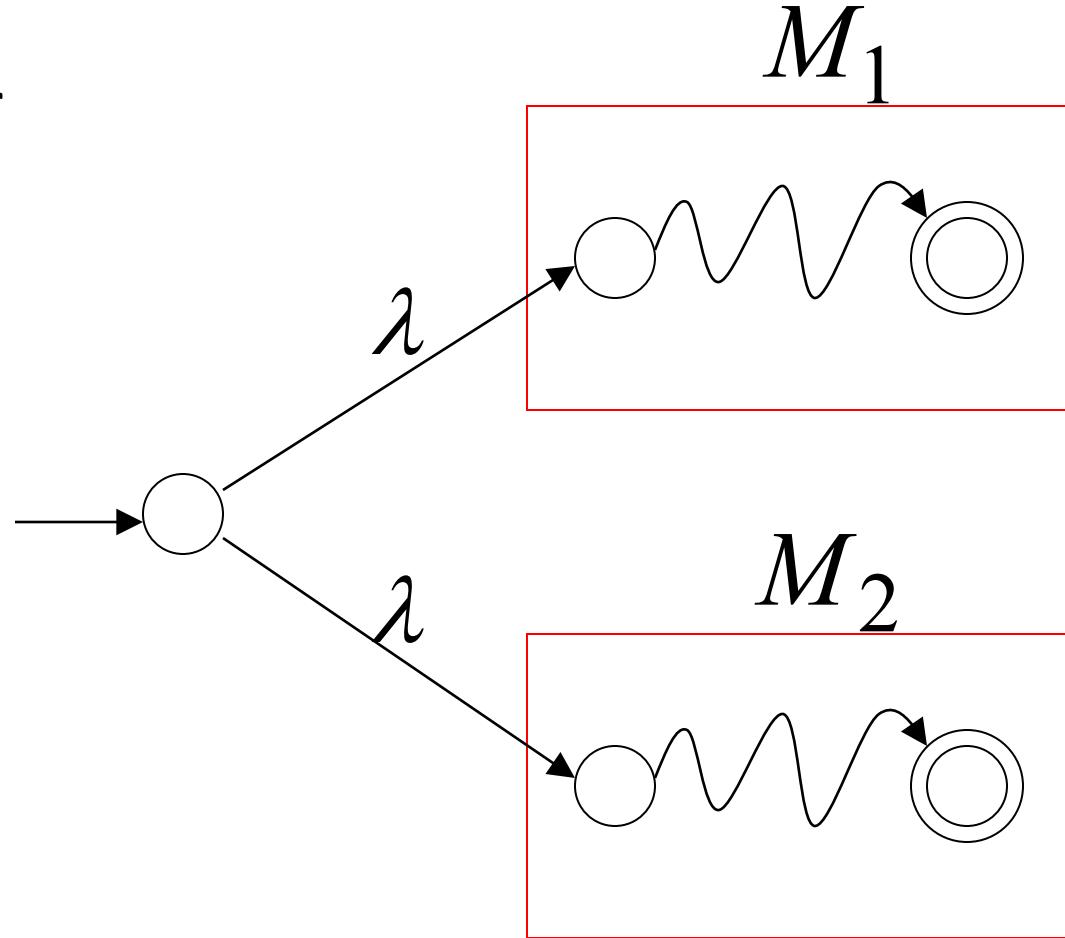


nuovo accettazione stati

Unione

$$L_1 \cup L_2$$

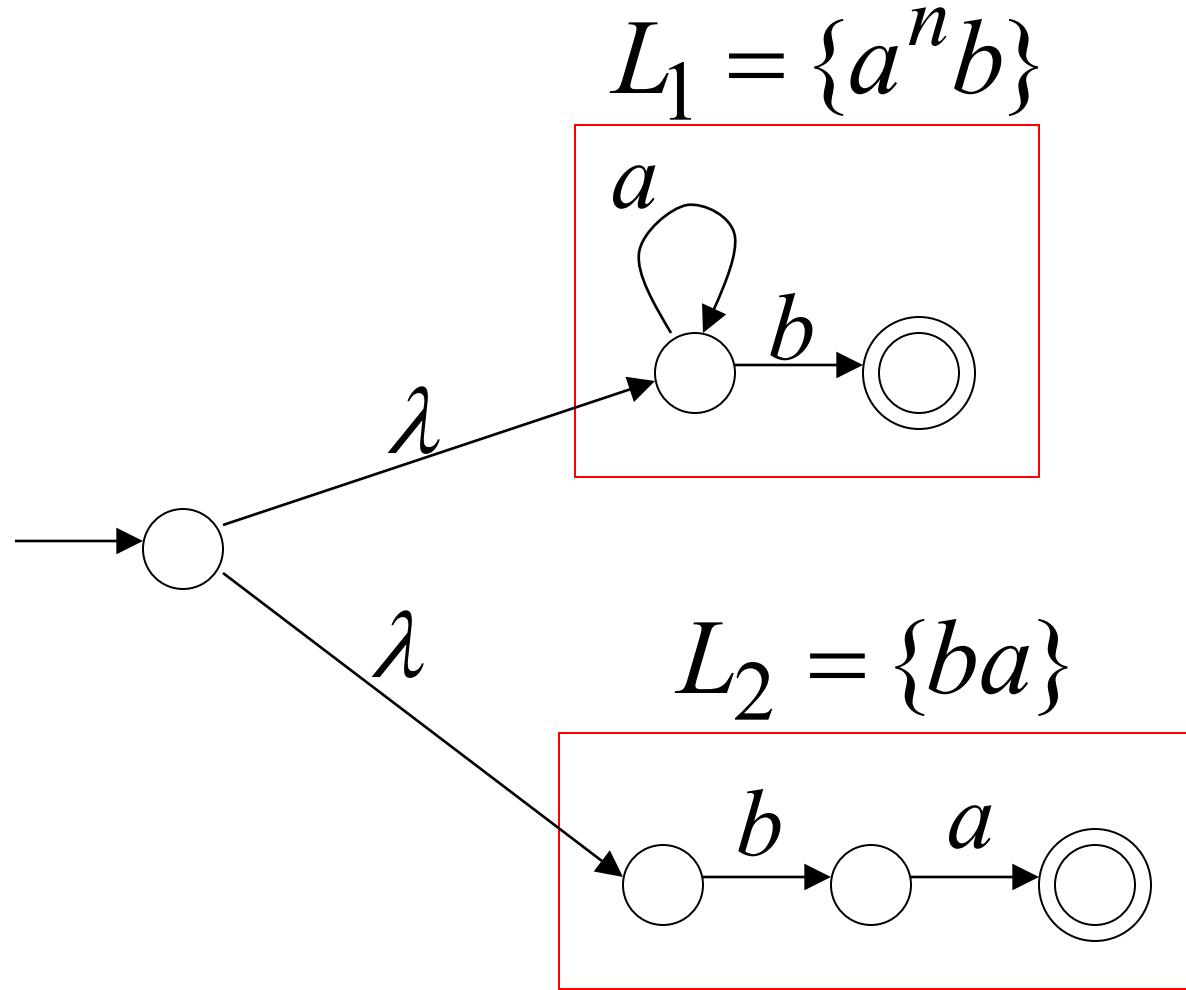
- NFA per



Esempio

NFA per $L_1 \cup L_2 = \{a^n b\} \cup \{ba\}$

-

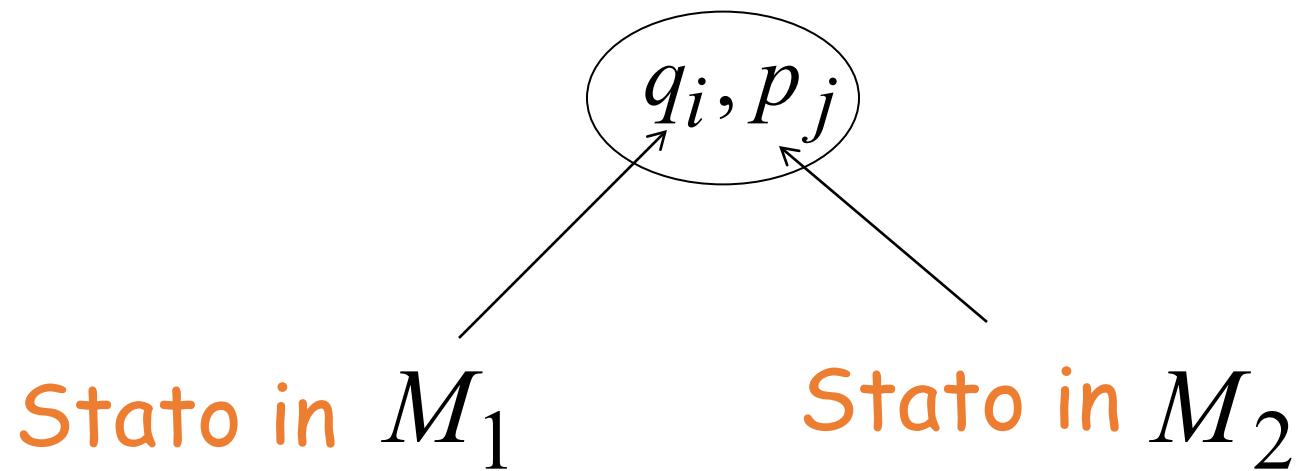


Potremmo definire l'unione in un altro modo.
dati due automi «completi»

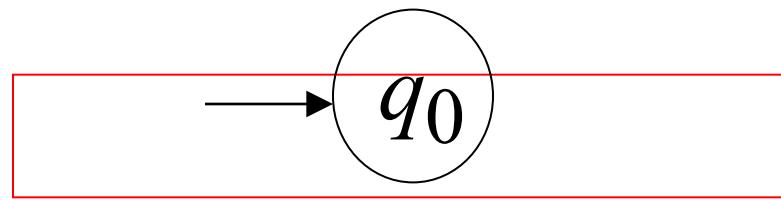
.

 M_1 M_2

Stati in M

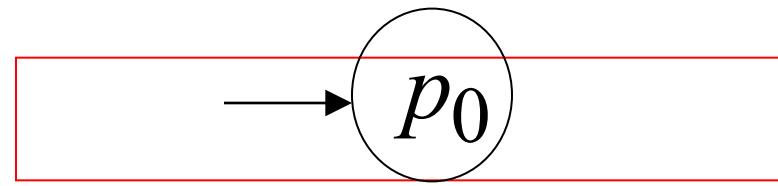


DFA M_1



stato iniziale

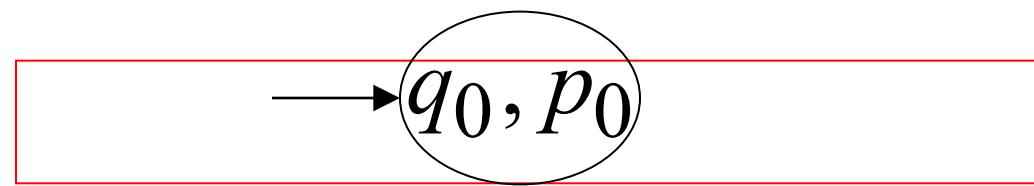
DFA M_2



stato iniziale

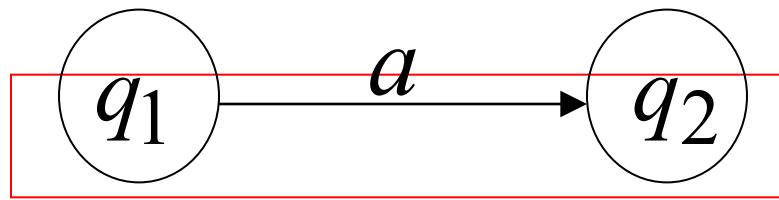


DFA M



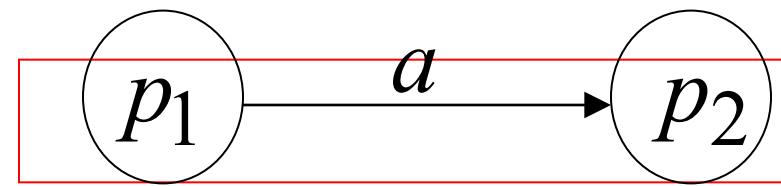
nuovo stato iniziale

DFA M_1



transizione

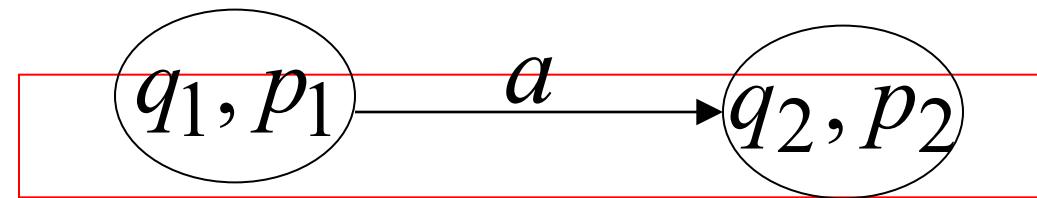
DFA M_2



transizione

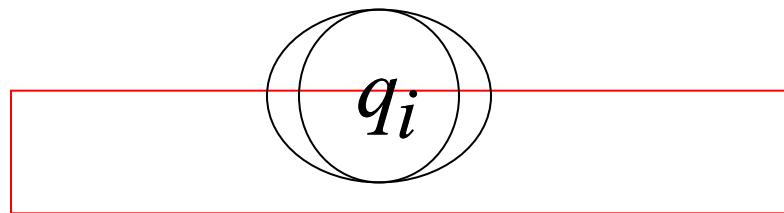


DFA M



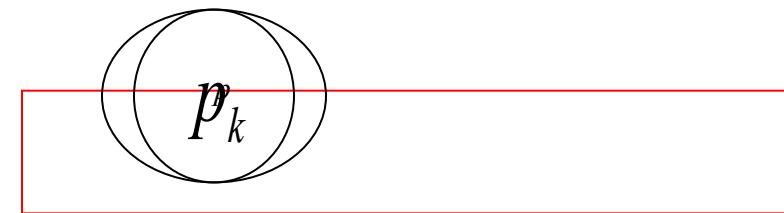
Nuova transizione

DFA M_1



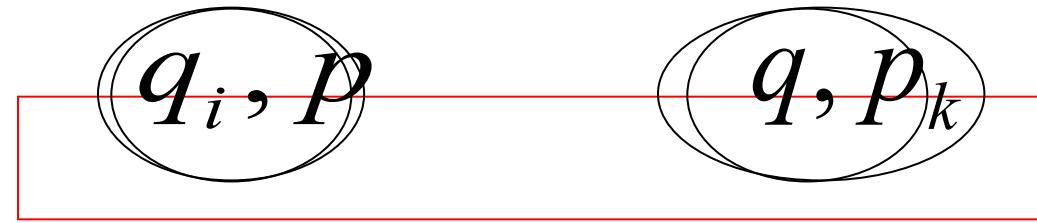
accettazione stato

DFA M_2



accettazione stato

DFA M



nuovi stati di accettazione

Con P uno stato
qualsiasi di M_2

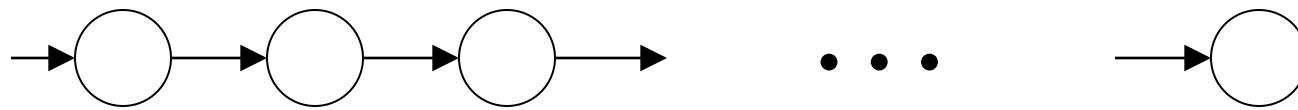
Con q uno stato
qualsiasi di M_1

Provare che la definizione
precedente definisce l'unione di
due automi.
Parliamo come.

Complessità temporale

Considera una Turing Machine deterministica
 M che decide un linguaggio L

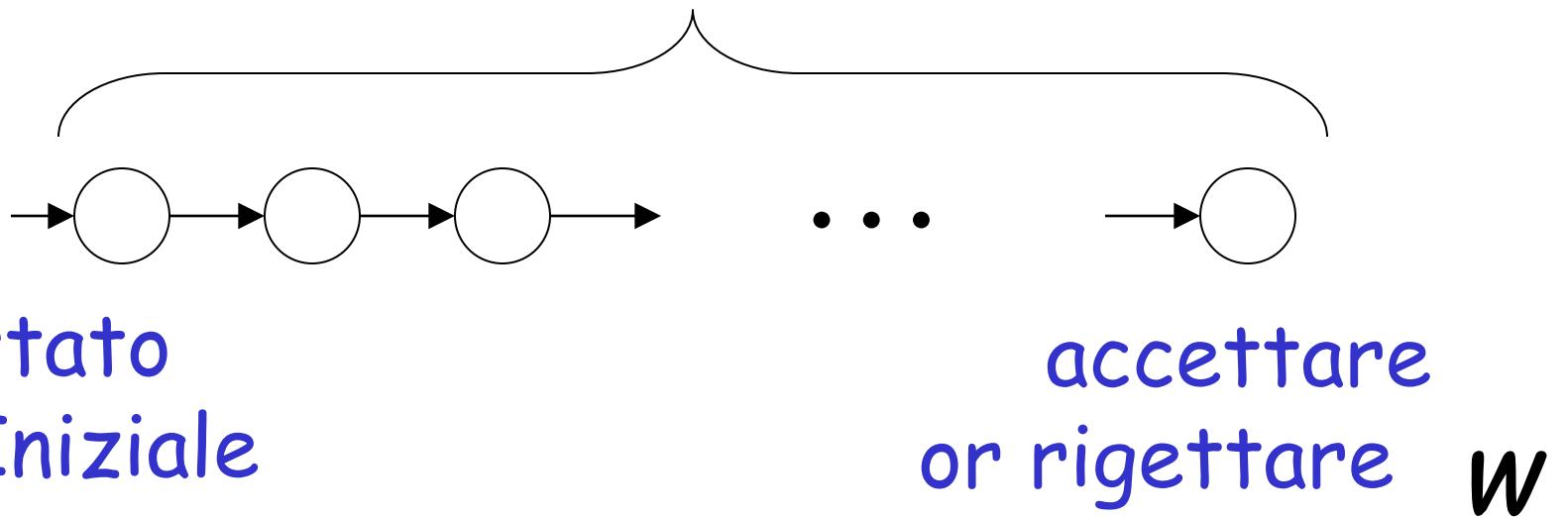
Per ogni stringa w la computazione di M termina usando una quantità finita di transizioni



Iniziale
stato

Accetta
o rifiuta w

Tempo di Decisione = #transizioni



Consideriamo ora, tutte le stringhe di Lunghezza n

$T_M(n)$ = massimo tempo richiesto per decidere (calcolare) Una qualsiasi stringa di lunghezza n

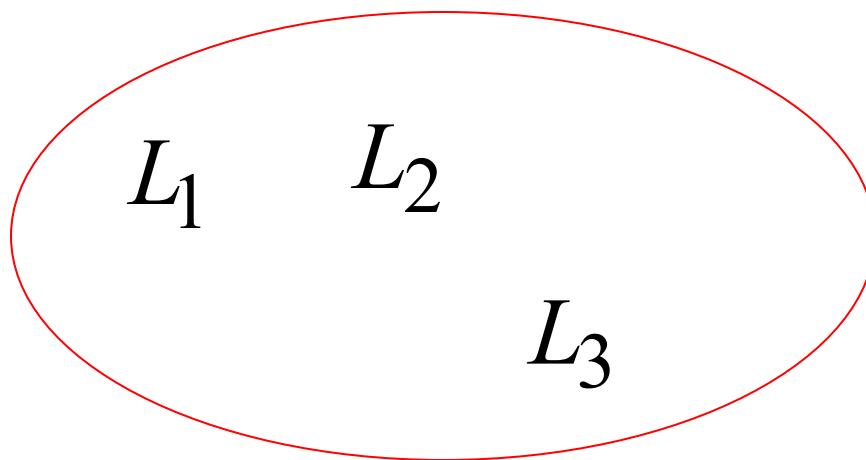
Let f and g be functions $f, g: \mathcal{N} \rightarrow \mathcal{R}^+$. Say that $f(n) = O(g(n))$ if positive integers c and n_0 exist such that for every integer $n \geq n_0$

$$f(n) \leq c g(n).$$

When $f(n) = O(g(n))$ we say that $g(n)$ is an ***upper bound*** for $f(n)$, or more precisely, that $g(n)$ is an ***asymptotic upper bound*** for $f(n)$, to emphasize that we are suppressing constant factors.

Classe Complessità temporale : $TIME(T(n))$

Tutti i linguaggi decidibili da una
Turing Machine deterministica
in tempo $O(T(n))$



Esempio: $L_1 = \{a^n b : n \geq 0\}$

Può essere deciso in tempo $O(n)$

$TIME(n)$

$L_1 = \{a^n b : n \geq 0\}$

Altri esempi nella stessa classe

$TIME(n)$

$$L_1 = \{a^n b : n \geq 0\}$$

$$\{ab^n aba : n, k \geq 0\}$$

$$\{b^n : n \text{ è pari}\}$$

$$\{b^n : n = 3k\}$$

Esempi nella classe

$TIME(n^2)$

$\{a^n b^n : n \geq 0\}$

$\{ww^R : w \in \{a,b\}^*\}$

$\{ww : w \in \{a,b\}^*\}$

M_1 = “On input string w :

1. Scan across the tape and *reject* if a 0 is found to the right of a 1.
2. Repeat if both 0s and 1s remain on the tape:
 3. Scan across the tape, crossing off a single 0 and a single 1.
 4. If 0s still remain after all the 1s have been crossed off, or if 1s still remain after all the 0s have been crossed off, *reject*. Otherwise, if neither 0s nor 1s remain on the tape, *accept*.”

In stages 2 and 3, the machine repeatedly scans the tape and crosses off a 0 and 1 on each scan. Each scan uses $O(n)$ steps. Because each scan crosses off two symbols, at most $n/2$ scans can occur. So the total time taken by stages 2 and 3 is $(n/2)O(n) = O(n^2)$ steps.

Esempi nella classe:

TIME (n^3)

Anche su sipster

CYK algorithm

$L_2 = \{\langle G, w \rangle : w \text{ è generata da una grammatica}$
context-free $G\}$

Moltiplicazione tra matrici

$L_3 = \{\langle M_1, M_2, M_3 \rangle : n \times n \text{ matrices}$
and $M_1 \times M_2 = M_3\}$

algoritmi tempo Polinomiale : $TIME(n^k)$

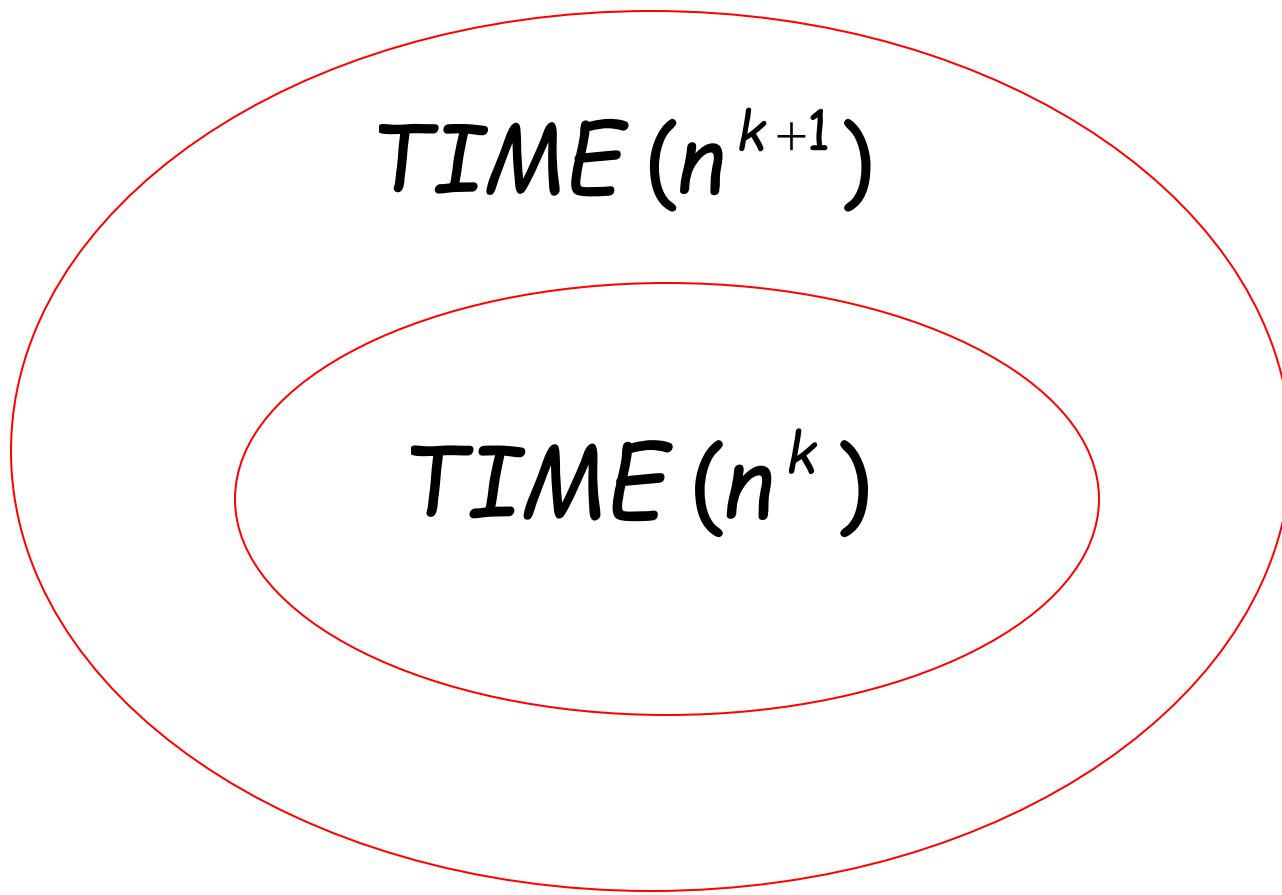
costante $k > 0$

Rappresentano gli algoritmi trattabili:

Per piccoli k possiamo decidere
il risultato velocemente

chiaramente:

$$TIME(n^{k+1}) \supset TIME(n^k)$$



architetture

Multitape = singolo tape in
tempo quadratico

Anche sul sipster

La Classe di Complessità temporale P

$$P = \bigcup_{k>0} TIME(n^k)$$

Representa:

- Algoritmi che usano tempo polinomiale
- Problemi “trattabili”

Class P

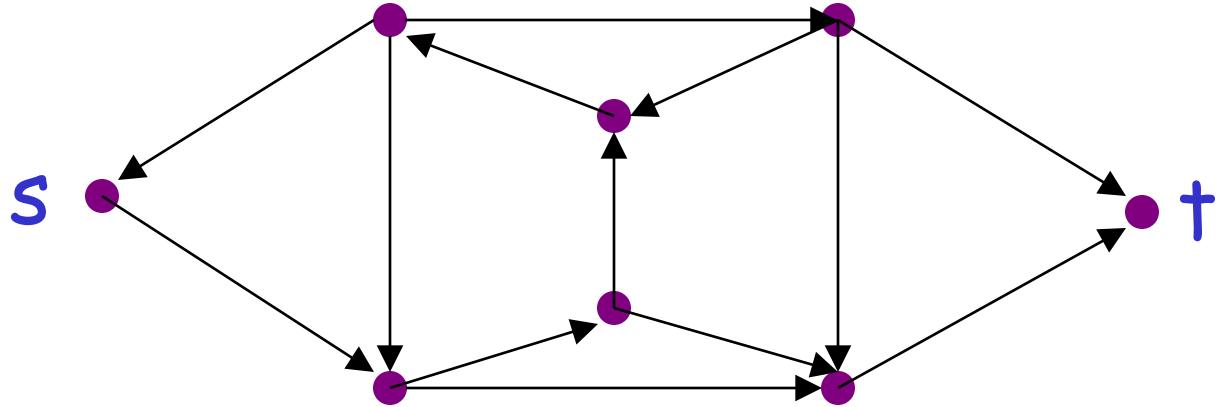
$\{a^n b\}$

$\{a^n b^n\}$

$\{ww\}$

CYK-algorithm

Dato un grafo e due nodi: esiste
un cammino da un nodo all'altro?



Dato un grafo e due nodi: esiste un cammino da un nodo all'altro?

PROOF A polynomial time algorithm M for $PATH$ operates as follows.

M = “On input $\langle G, s, t \rangle$ where G is a directed graph with nodes s and t :

1. Place a mark on node s .
2. Repeat the following until no additional nodes are marked:
 3. Scan all the edges of G . If an edge (a, b) is found going from a marked node a to an unmarked node b , mark node b .
 4. If t is marked, *accept*. Otherwise, *reject*.

Step 1 : 1

Step 4: 1

Step 2,3 : m (numero dei nodi)

Qualche problema non in P

algoritmi calcolabili
in tempo esponenziale

$TIME(2^{poly(k)})$

$TIME(2^{n^k})$

Rappresentano algoritmi intrattabili

Per alcuni input ci possono volere
secoli per trovare la soluzione

Ricordiamo:

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

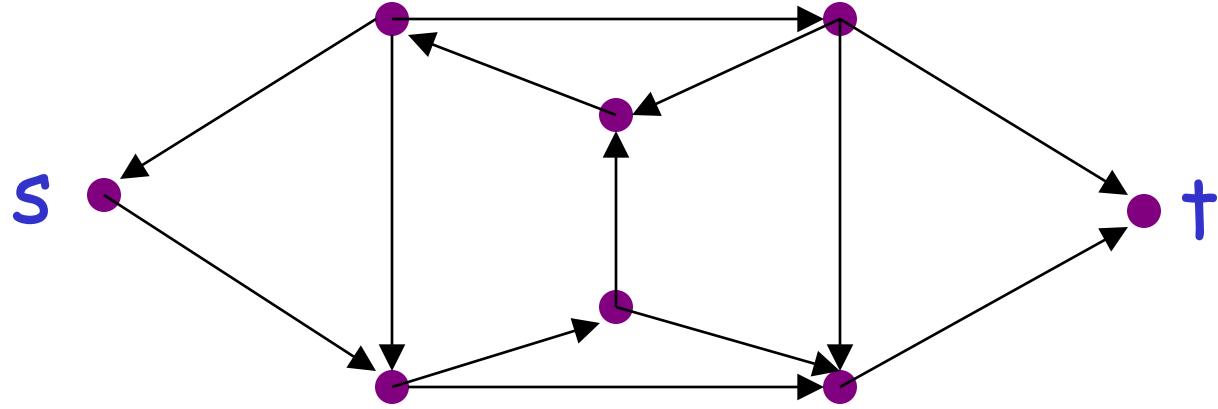
NonDeterminismo =
Determinismo in tempo
esponenziale (dim anche
sipster)

uno

Problema: Cammino Hamiltonian

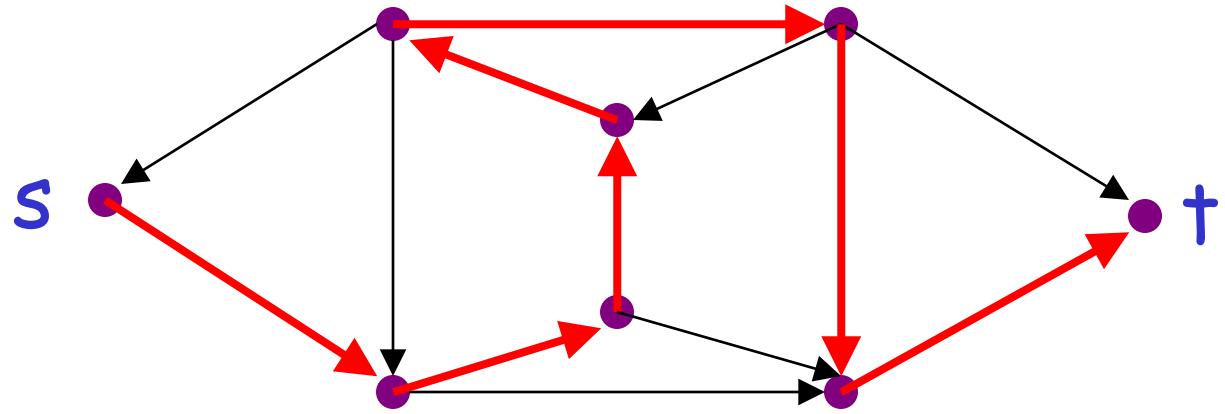
un cammino in un grafo
(orientato o non orientato)
è detto **hamiltoniano** se
esso tocca tutti i vertici
del grafo
una e una sola volta.

Problema del Cammino Hamiltonian



domanda: vi è un Cammino Hamiltoniano
da s a t?

tocca tutti i vertici del grafo
una e una sola volta.



si

tocca tutti i vertici del
grafo una e una sola
volta.

Non esistono algoritmi efficienti per la risoluzione del «problema del cammino hamiltoniano», l'unico metodo di risoluzione è rappresentato dall'enumerazione totale, ovvero nell'enumerazione di tutti i possibili cammini sul grafo fino a che non si trovi il cammino voluto.

ab, ba Permutazione di n elementi

Arriva c

Possibili posizioni: Avanti, Dietro, mezzo(posizione 2); tre nuovi elementi per ogni elemento di partenza
cab, acb, abc, cba, bca, bac (3!)

Arriva d prendiamo per esempio abc
Avanti, Dietro, posizione 2, posizione 3 (Aa2b3cD); sono 4 nuovi elementi per ogni elemento di partenza

Per ogni tripla

$$6 * 4 = 24 = 4!$$

Una soluzione :

Lavorare su tutti i cammini

$TIME(2^{poly(k)})$

$L = \{<G,s,t> : \text{vi è un Cammino Hamiltonian in } G \text{ da } s \text{ a } t\}$

$n! = n \times (n-1) \times \dots$

$n \times n \times n \dots$

$L \in TIME(n!) \approx TIME(2^{n^k})$ $k=2$

Permutazione di
n elementi

tempo Esponenziale

problema Intrattabile

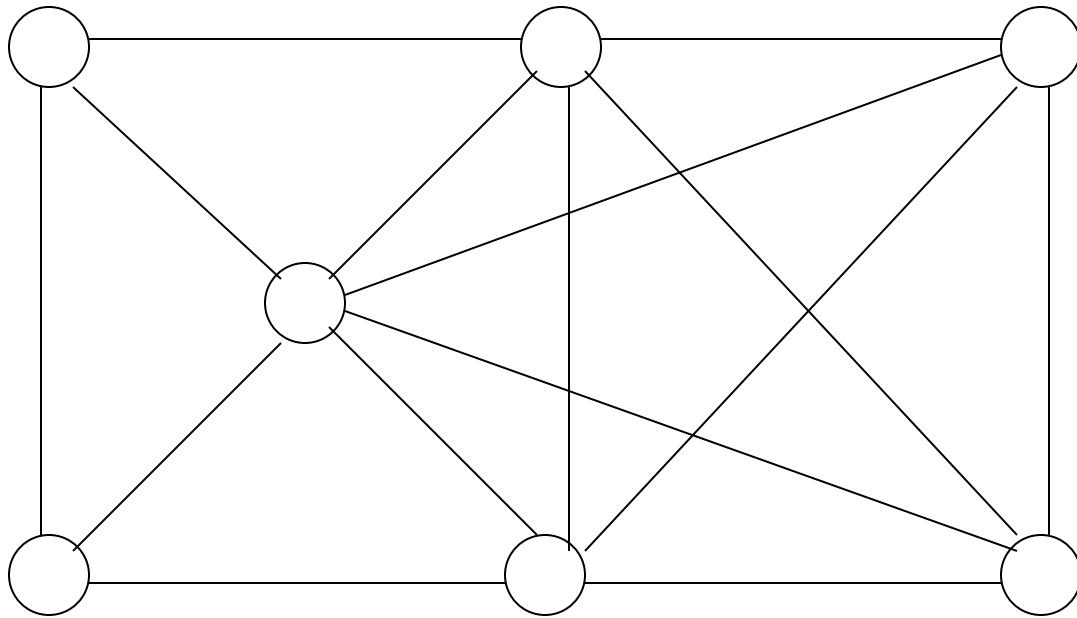
due

problema della cricca

dato un grafo e dato un k

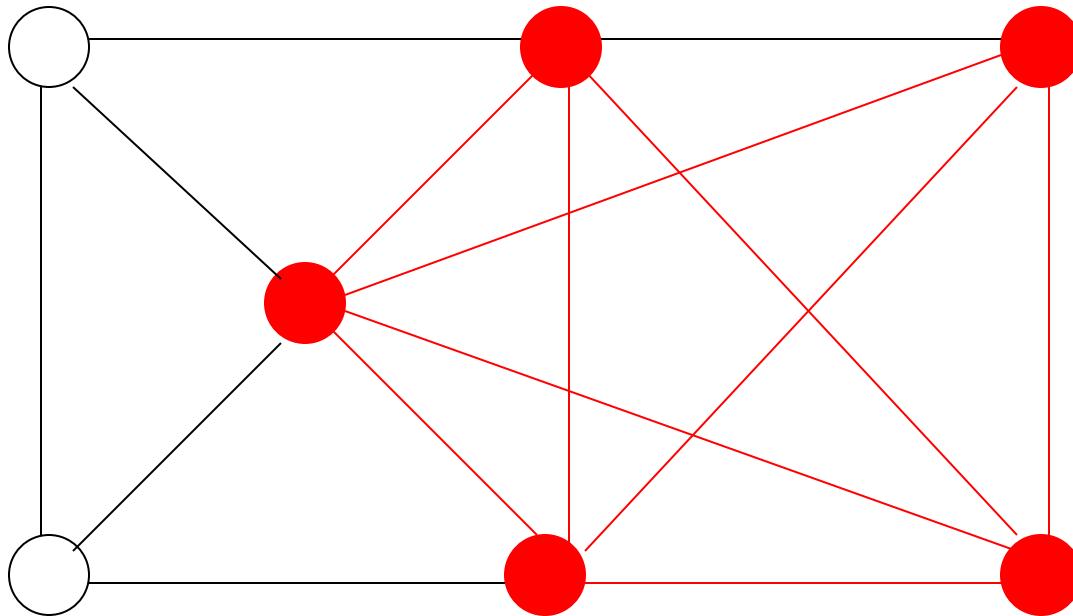
trovare un insieme di k nodi dove
ciascun elemento è connesso con tutti
gli altri.

Esempio: Il problema della cricca



Esiste una cricca di grado 5?

Il problema della cricca



Esiste una cricca di grado 5.

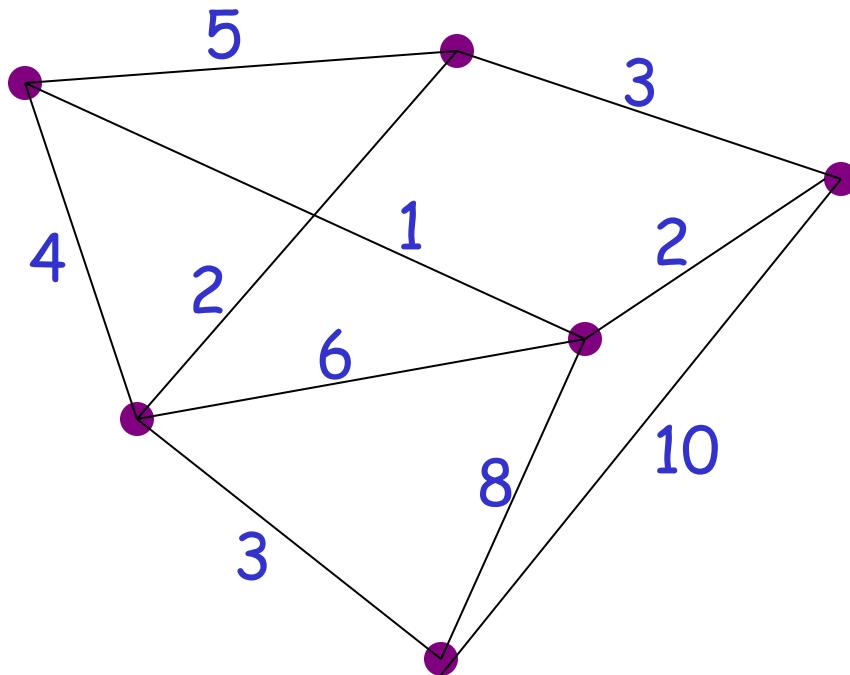
Tutte le permutazioni di 5 elementi
e verificare che ogni elemento è
connesso con tutti gli altri.

$5! \times 5!$

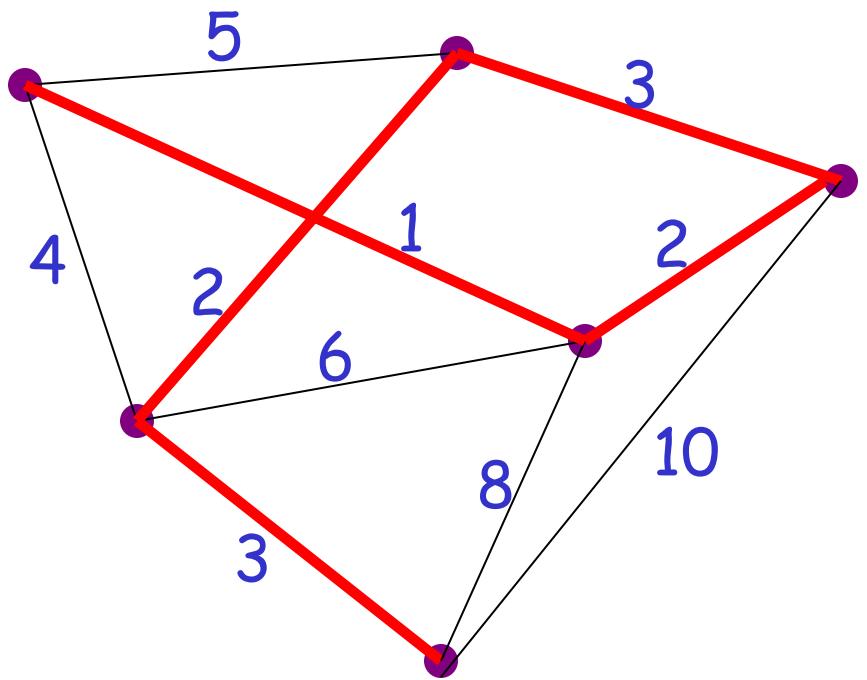
Per precisione $5! \times (5-1) \times (4-1) \dots \times 1$

tre

Esempio: commesso viaggiatore



domanda: quale è la via più veloce
Per connettere tutte le città?



quale è la via più veloce
Per connettere tutte le città?

Una soluzione : ricerca tutti i cammini,
Hamiltoniani, ovvero tutti i cammini
che toccano tutti i vertici una sola volta.

ricorda $L \in TIME(n!) \approx TIME(2^{n^k})$

quindi

$L = \{\text{shortest hamiltonian paths}\}$

quattro: il Problema della soddisfacibilità

espressioni Booleani in
Conjunctive Normal Form:

$t_1 \wedge t_2 \wedge t_3 \wedge \cdots \wedge t_k$ clausole

$t_i = x_1 \vee \bar{x}_2 \vee x_3 \vee \cdots \vee \bar{x}_p$
Variabili

domanda: è l'espressione soddisfacibile?

Esempio:

$$(\bar{x}_1 \vee x_2) \wedge (x_1 \vee x_3)$$

soddisfacibile:

$$x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$$

$$(\bar{x}_1 \vee x_2) \wedge (x_1 \vee x_3) = 1$$

Esempio: $(x_1 \vee x_2) \wedge \bar{x}_1 \wedge \bar{x}_2$

Non soddisfacibile

$(x_1 \vee x_2) \wedge \bar{x}_1$

soddisfacibile

$$L = \{w : w \text{ soddisfacibile}\}$$

$$L \in TIME(2^{n^k})$$

Algoritmo:

ricerca , in modo esaustivo,
su tutti i possibili valori delle variabili
Tavole di verità, n variabili , 2^n

Non-determinismo: prima definizione

La classe dei linguaggi: $NTIME(T(n))$

Turing Machine Non-Deterministica:
i rami di computazione sono limitati
da un $T(n)$

Linguaggi decidibili da una mdTuring
non deterministica in tempo $O(T(n))$

Non-determinismo: seconda definizione

A *verifier* for a language A is an algorithm V , where

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}.$$

We measure the time of a verifier only in terms of the length of w , so a *polynomial time verifier* runs in polynomial time in the length of w . A language A is *polynomially verifiable* if it has a polynomial time verifier.

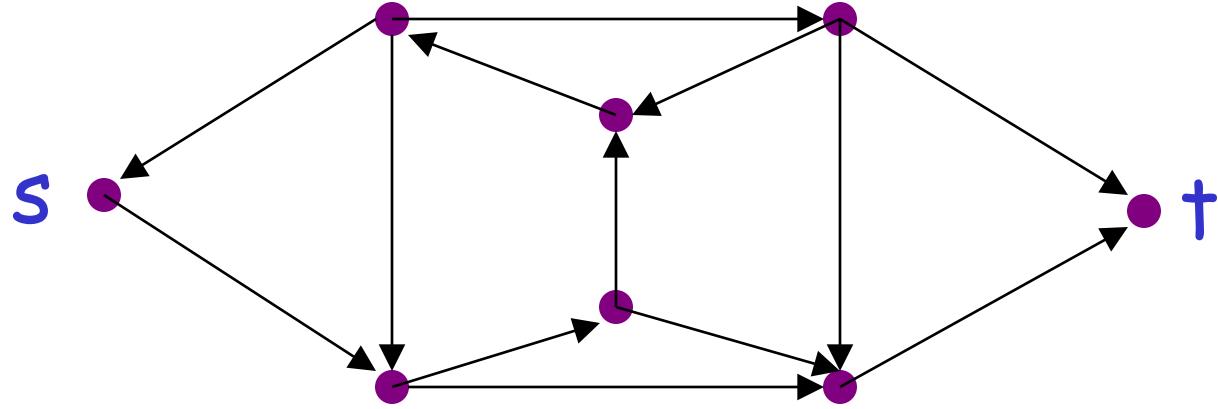
Decide per ogni stringa (w) di
lunghezza n con l'aiuto di una
stringa c in tempo $O(T(n))$

uno

Problema: Cammino Hamiltonian

un cammino in un grafo
(orientato o non orientato)
è detto **hamiltoniano** se
esso tocca tutti i vertici
del grafo
una e una sola volta.

Problema del Cammino Hamiltonian



domanda: vi è un Cammino Hamiltoniano
da s a t?

tocca tutti i vertici del grafo
una e una sola volta.

Problema del Cammino Hamiltoniano

Esempio

Stringa giusta che ci dà il cammino.

due

problema della cricca

dato un grafo e dato un k

trovare un insieme di k nodi dove
ciascun elemento è connesso con tutti
gli altri.

Esempio

.

problema della cricca, dato un k trovare un insieme di k elementi dove ciascun elemento è connesso con tutti gli altri. Stringa giusta che ci dà la cricca.

il Problema della soddisfabilità

Valori di verità giusti

quattro: il Problema della soddisfacibilità

espressioni Booleani in
Conjunctive Normal Form:

$t_1 \wedge t_2 \wedge t_3 \wedge \cdots \wedge t_k$ clausole

$t_i = x_1 \vee \bar{x}_2 \vee x_3 \vee \cdots \vee \bar{x}_p$
Variabili

domanda: è l'espressione soddisfacibile?

Esempio

.

il Problema della soddisfabilità

Valori di verità giusti

Due definizioni sono uguali?

Verificatore \rightarrow NdT

Assumiamo che V è limitata da n^k .

Prendiamo tutte le stringhe di lunghezza n^k :

NdT: Su input w di lunghezza n :

1. Non deterministicamente seleziona stringa c di lung al massimo n^k ;
2. Calcola V su (w, c) ;
3. Se V accetta allora accetta altrimenti rifiuta

NdT \rightarrow Verificatore

Trovare la
stringa

Ricorda che se NdT accetta s allora esiste un cammino, c, che ci porta all'accettazione.

Sia Ndt **N** costruiamo un verificatore V
V: input (w,c)

1. Simula NdT sull'input w scegliendo il cammino indicato da c.
2. Se V accetta allora accetta,
altrimenti rigetta

Esempio: $L = \{ww\}$

algoritmo Non-Deterministico
per accettare una stringa ww :

- Usiamo una two-tape Turing machine
- Congetturiamo la metà della stringa e la copiamo w sul secondo nastro
- Compariamo i due nastri

$$L = \{ww\}$$

tempo necessario

Usiamo una two-tape Turing machine

Congetturiamo la metà della stringa $O(|w|)$
e la copiamo w sul secondo nastro

Compariamo i due nastri $O(|w|)$

tempo totale: $O(|w|)$

$NTIME(n)$

$L = \{ww\}$

$$L = \{w : \text{expression } w \text{ is satisfiable}\}$$
$$L \in NP$$

Il problema della soddisfacibilità è un
 NP - Problem

$$L = \{w : \text{expression } w \text{ is satisfiable}\}$$

Tempo necessario per n variabili:

- Congetturiamo un assegnazione $O(n)$ di valore alle variabili
 - Verifichiamo che questo assegnamento $O(n)$ sia soddisfacibile
- Total tempo: $O(n)$

In modo simile possiamo definire

$$NTIME(T(n))$$

Per ogni funzione temporale : $T(n)$

Esempi: $NTIME(n^2), NTIME(n^3), \dots$

La classe NP Non-Deterministic Polynomial time

$$NP = \bigcup NTIME(n^k)$$

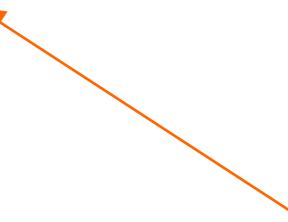
Per ogni k

$$P = \bigcup TIME(n^k)$$

osservazione:

$$P \subseteq NP$$

Deterministic
Polynomial



Non-Deterministic
Polynomial

Problema aperto: $P = NP$?

Non conosciamo la risposta

qui

Problema aperto: $P = NP$?

Esempio: il problema della soddisfacibilità
ha un algoritmo deterministico
che lo risolva in tempo polinomiale?

Non conosciamo la risposta

Fine cap.

NP-Completezza

Un problema è NP-complete se:

- E' in NP
- Ogni NP problema si può ridurre
al problema di partenza
(in tempo polinomiale)

Osservazione:

Se possiamo risolvere un problema

NP-complete

in tempo Deterministic Polynomial (P tempo)

Allora avremo che :

$$P = NP$$

Osservazione :

Se proviamo che
non possiamo risolvere un problema
NP-complete
in tempo Deterministic Polynomial (P tempo)

Allora abbiamo:

$$P \neq NP$$

Teorema di Cook:

Il problema della soddisfacibilità è
NP-complete

Dimostrazione:

Convertire una Non-Deterministic Turing
Machine

In una espressione booleana
in (congiuntiva) conjunctive normal form

Altri problemi NP-Complete :

- Il commesso viaggiatore
- Vertex cover
- Hamiltonian Path

Tutti questi problemi si possono ridurre
al problema della soddisfabilità

Osservazione:

sarebbe molto strano che NP-complete problemi sono in P

I problemi NP-complete hanno algoritmi in tempo esponenziale

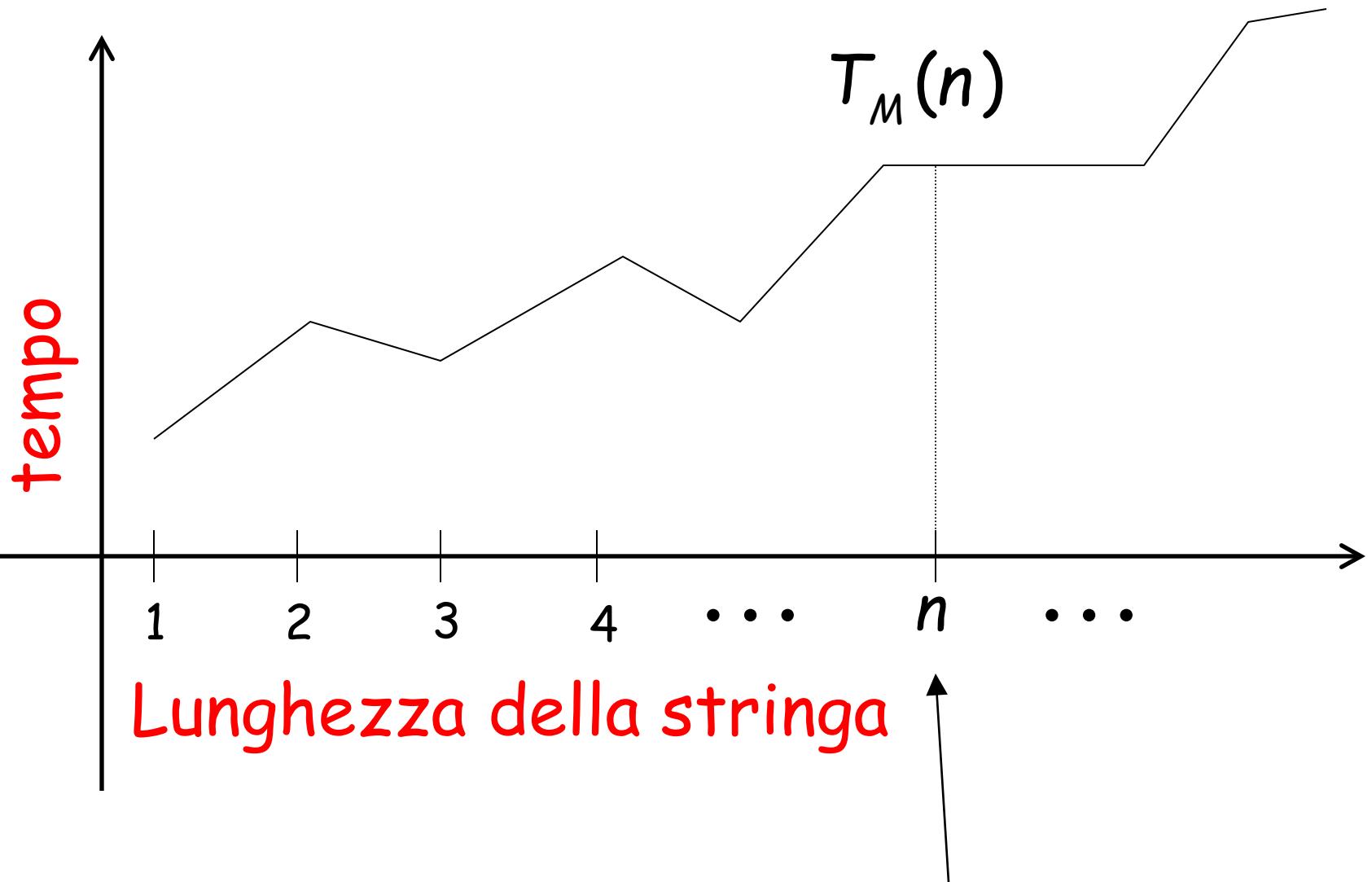
Approssimazioni di questi problemi
Sono in P

tempo complessità:

Il numero di passi (step)
durante una computazione

spazio complessità:

spazio usato
durante una computazione



algoritmi calcolabili
in tempo esponenziale

$TIME(2^{n^k})$

$TIME(2^{2^{poly(k)}})$

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time nondeterministic single-tape Turing machine has an equivalent $2^{O(t(n))}$ time deterministic single-tape Turing machine.

Rappresentano algoritmi intrattabili

Per alcuni input ci possono volere secoli per trovare la soluzione

tempo algoritmi
Non-Deterministic Polynomial :

$$L \in NTIME(n^k)$$

algoritmi

Tempo Polinomiale Non-Deterministico

$$L \in NTIME(n^k)$$

La classe NP
Non-Deterministic Polynomial time

Per ogni k

$$NP = \cup NTIME(n^k)$$

La classe P
Deterministic Polynomial time

Per ogni k

$$P = \cup TIME(n^k)$$

Linguaggi NP-completi

Una funzione $f: A \rightarrow B$

è calcolabile in tempo polinomiale se esiste una macchina M che calcola la funzione f in tempo polinomiale

$$w \xrightarrow{\hspace{1cm}} wOf(w)$$

Un linguaggio A è riducibile in tempo polinomiale a un linguaggio B , $A \leq B$, se esiste una funzione polinomiale f tale che per ogni $x : x \in A$ implica $f(x) \in B$

Nota che se $A \leq B$ e B è polinomiale
allora anche A è polinomiale.

Sia M che decide B ,

Sia f che riduce A a B

Sia x elemento di A

1. Calcola $f(x)$
2. Calcola $M(f(x))$

$A \leq B$

Pari \leftarrow Dispari
Dispari \leftarrow Pari

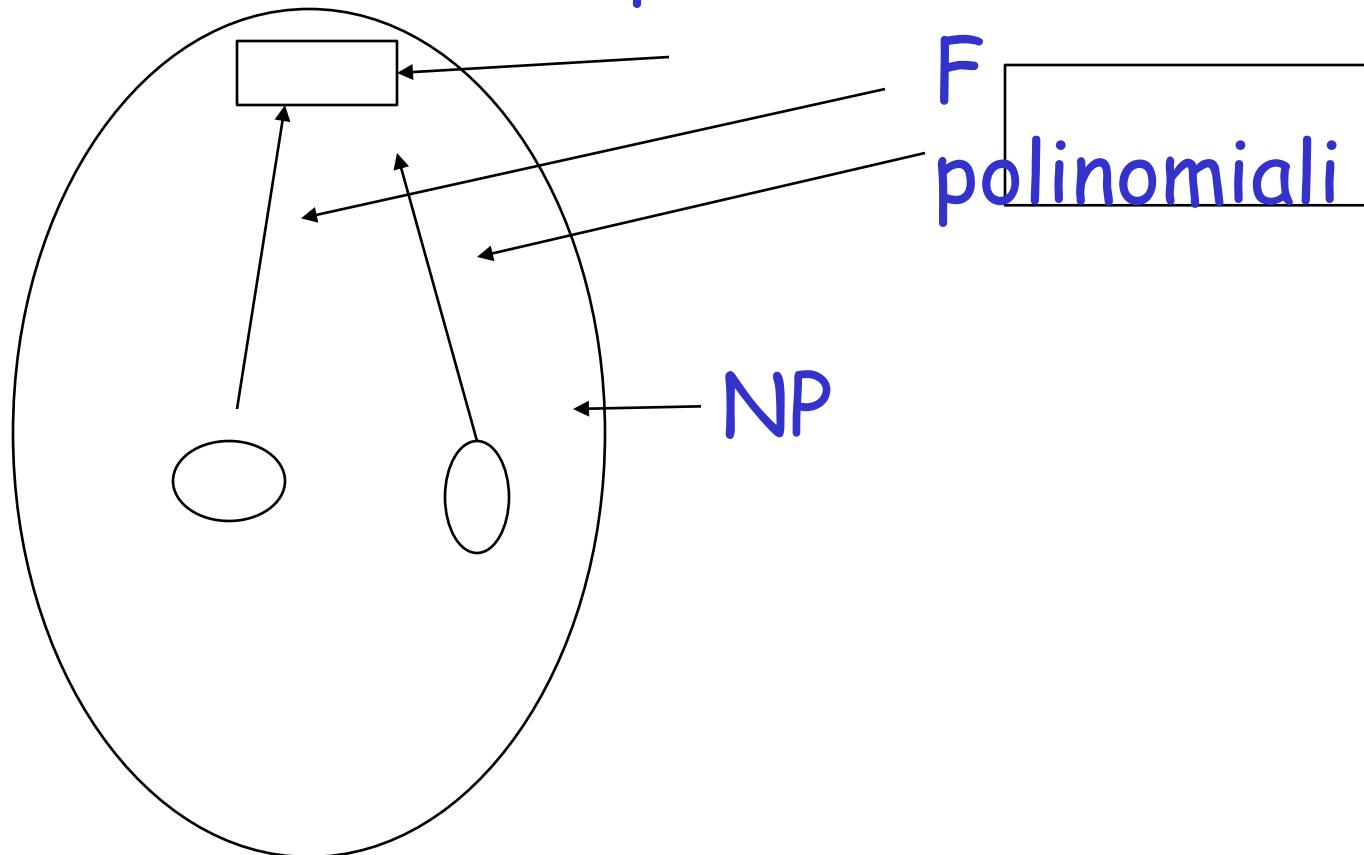
X è elemento di Pari
(div 2 (resto=0))
Dispari
X è elemento di dispari
Non (div 2)

NP-Completezza

Un problema A è NP-completo se:

- A è in NP
- Ogni problema NP è riducibile ad A
(in tempo polinomiale)

NP
completi



F
polinomiali

Osservazione:

Se possiamo risolvere un problema
NP-complete
in tempo Deterministico Polinomiale
allora :

$$P = NP$$

Osservazione :

Se proviamo che
non possiamo risolvere un problema
NP-complete
in tempo Deterministico Polinomiale
Allora :

$$P \neq NP$$

Un Linguaggio L è NP-completo se:

- L è in NP, e
- Ogni Linguaggio in NP può essere ridotto a L in Tempo Polinomiale

Decidibili

NP

P

NP-complete

?

Formule SAT:

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

literal



L: letterale o letterale negato

O: Gruppi di L collegati da \vee

Sat: Gruppi di O collegati da \wedge

Un Linguaggio NP-completo

Teorema di Cook-Levin :

Linguaggio SAT (satisfiability problem)
è NP-complete

Dim:

Part1: SAT è in NP
(già provato)

Part2: ridurre tutti i Linguaggi NP
al problema SAT
in Tempo Polinomiale

Sia $L \in NP$, un arbitrario linguaggio

Definiamo una riduzione Polinomiale di L to SAT

Sia M Macchina di Turing Nondeterministica che decide L in Tempo polinomiale

Per ogni stringa w costruiamo in Tempo Polinomiale una espressione Booleana $\varphi(M, w)$

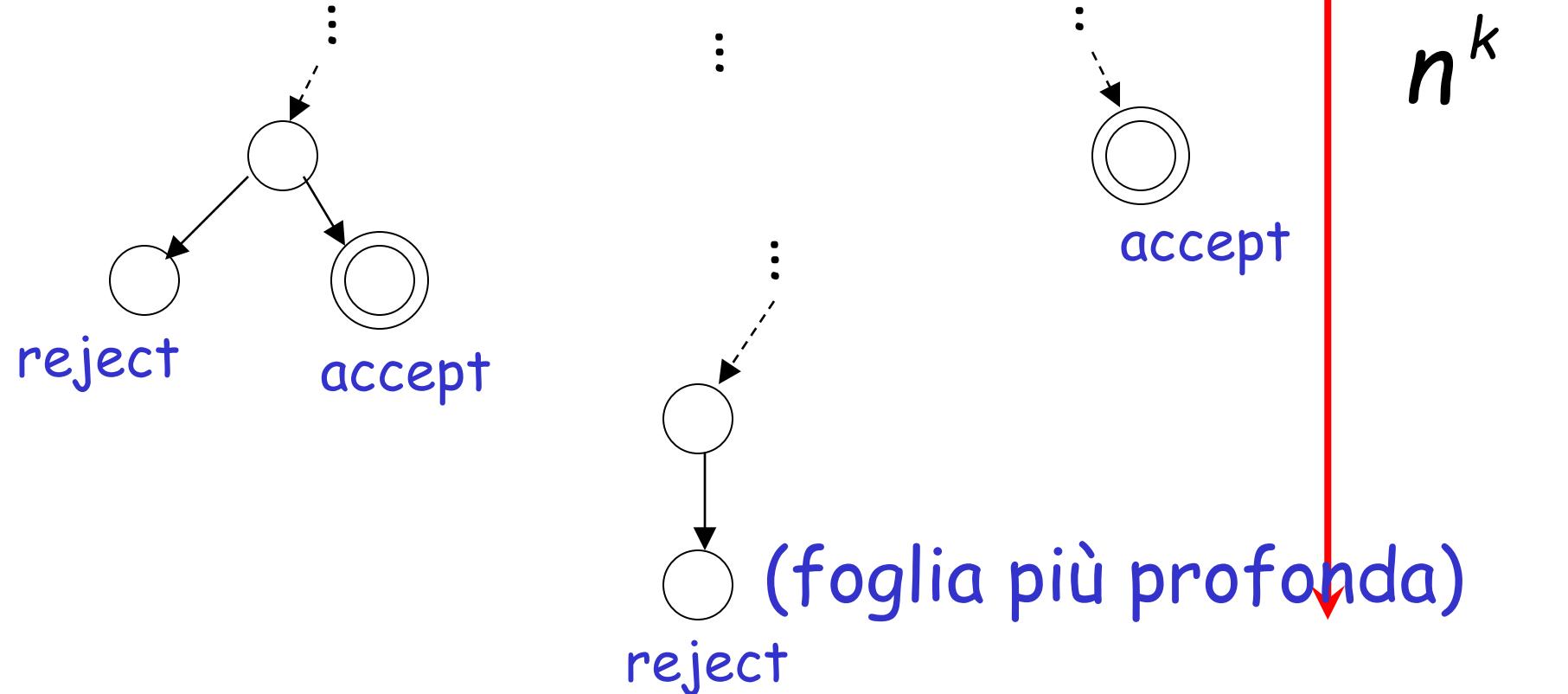
tale che: $w \in L \Leftrightarrow \varphi(M, w)$ è soddisfacibile

Tutte le computaz.

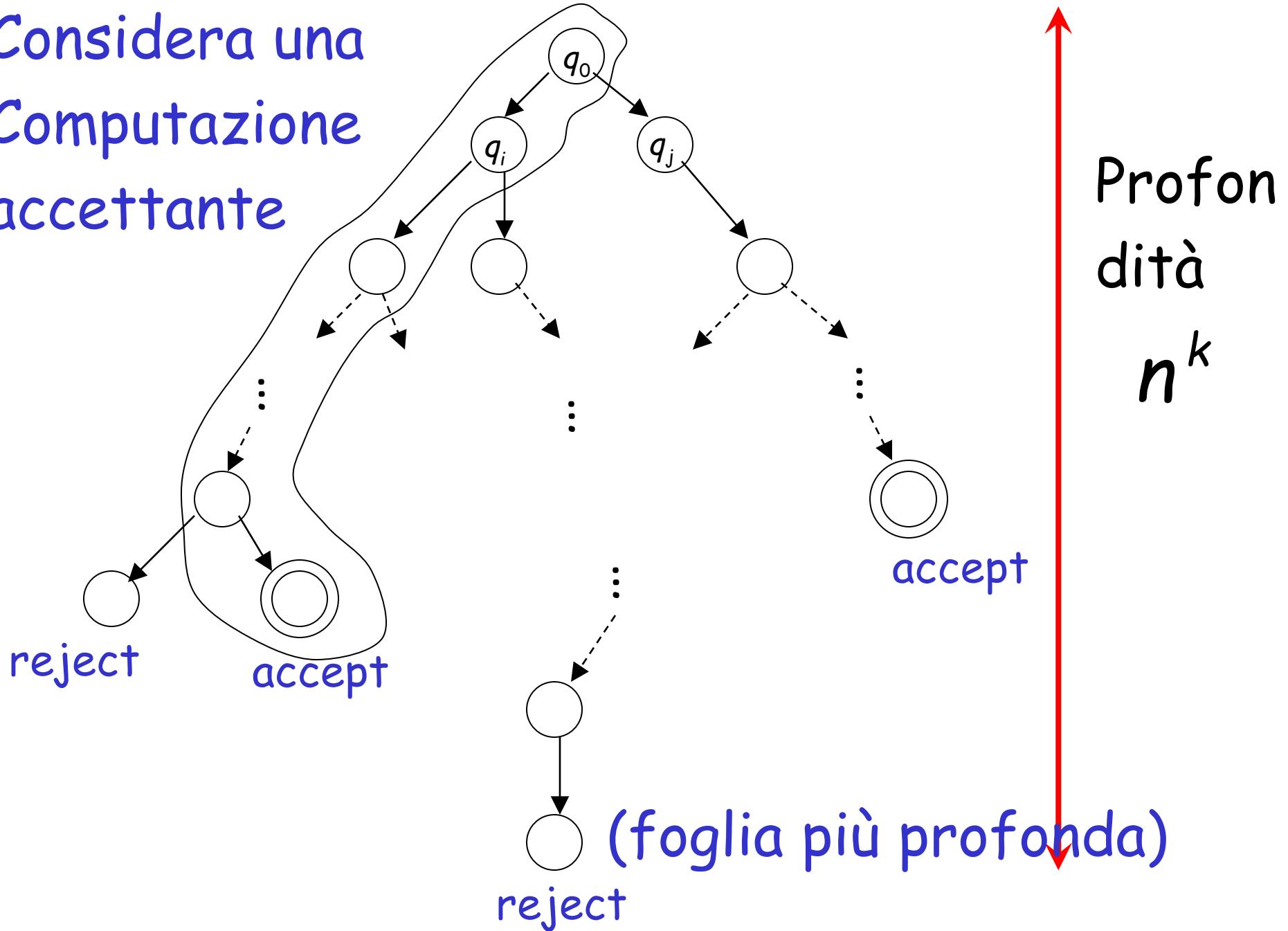
of M sulla stringa

w

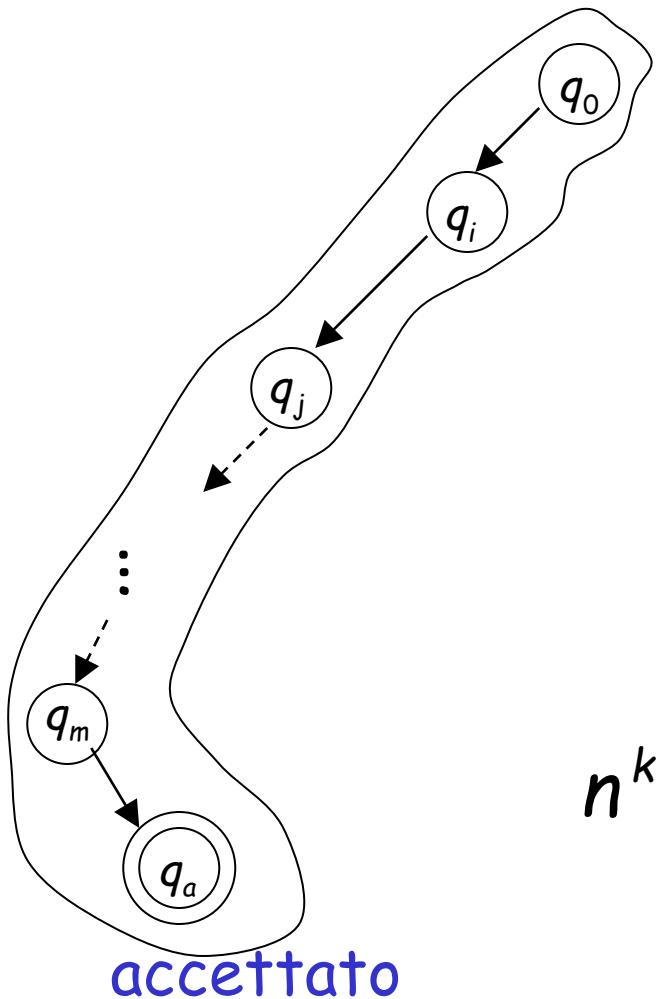
$|w| = n$



Considera una
Computazione
accettante



Cammino di computazione



Sequenze di
configurazioni

Stato iniziale

1: $q_0 \sigma_1 \sigma_2 \cdots \sigma_n$

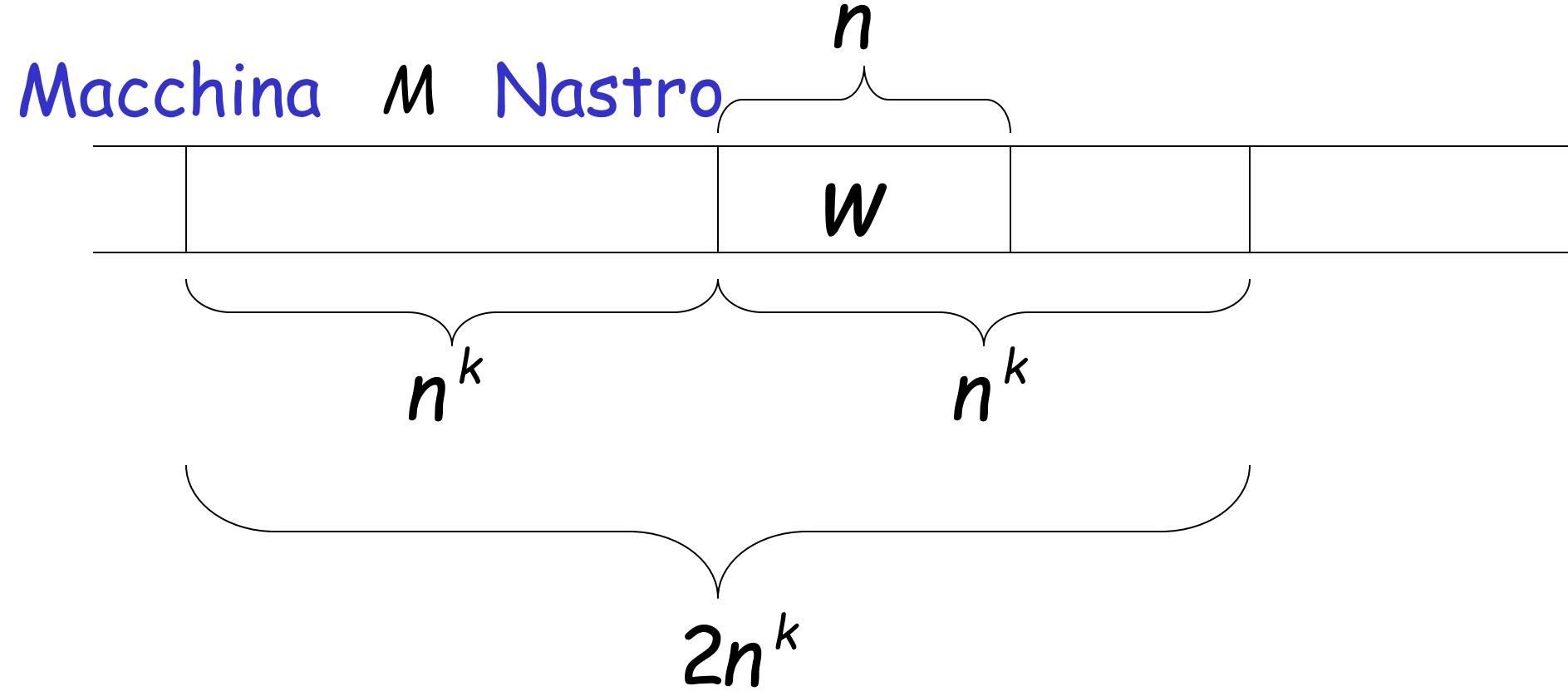
2: $\succ \sigma'_1 q_i \sigma_2 \cdots \sigma_n$

⋮

$n^k \geq x :$ $\succ \sigma'_1 \cdots \sigma'_l q_a \sigma'_{l+1} \cdots \sigma'_{n^k}$

Stato accettazione

$W = \sigma_1 \sigma_2 \cdots \sigma_n$



Massima area di calcolo sul nastro

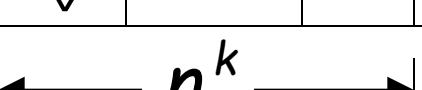
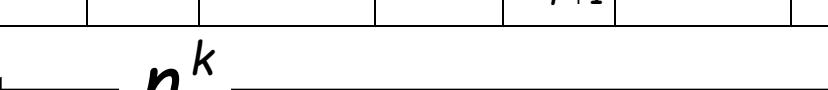
durante n^k steps (passi) temporali

Tableau delle configurazioni

1 :	#	\diamond	...	\diamond	q_0	σ_1	σ_2	...	σ_n	\diamond	...	\diamond	#
2 :	#	\diamond	...	\diamond	σ'_1	q_i	σ_2	...	σ_n	\diamond	...	\diamond	#
⋮ :													
x :	#	\diamond	...	σ''	σ'_1	σ'_2	σ'_3	...	q_a	σ'_{l+1}	...	σ_{n^k}	#
n^k :	#	\diamond	...	σ''	σ'_1	σ'_2	σ'_3	...	q_a	σ'_{l+1}	...	σ_{n^k}	#

Configurazione accettante

Righe identiche

Configurazione accettante

Righe identiche

Alfabeto del Tableau

$$\begin{aligned}C &= \{\#\} \cup \{\text{alfabeto nastro}\} \cup \{\text{insieme degli stati}\} \\&= \{\#\} \cup \{\alpha_1, \dots, \alpha_r\} \cup \{q_1, \dots, q_t\}\end{aligned}$$

Dimensione finita (costante)

Per ogni cella con posizione i, j

E per ogni simbolo nell'alfabeto
del tableau $s \in C$

Definiamo la variabile $x_{i,j,s}$

tale che se la cella i, j contiene il simbolo s
allora $x_{i,j,s} = 1$ altrimenti $x_{i,j,s} = 0$

Esempio:

	$n^k + 3$												
1:	#	◊	...	◊	q_0	σ_1	σ_2	...	σ_n	◊	...	◊	#
2:	#	◊	...	◊	σ'_1	q_i	σ_2	...	σ_n	◊	...	◊	#

$x_{1,1,\#} = 1$ $x_{2,n^k+3,q_i} = 1$
 $x_{1,1,\diamond} = 0$ $x_{2,n^k+3,\#} = 0$

$\varphi(M, w)$ è costruito con le variabili $x_{i,j,s}$

$$\varphi(M, w) = \varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{accept}} \wedge \varphi_{\text{move}}$$

Quando la formula è soddisfatta allora descrive una computazione di accettazione nel tableau della macchina M su input w

φ_{cell}

Ci rende sicuri che ogni cella nel tableau contiene esattamente un simbolo

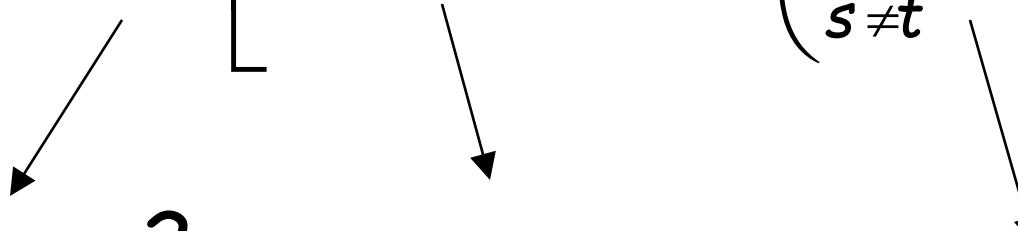
$$\varphi_{\text{cell}} = \bigwedge_{\text{all } i,j} \left[\left(\bigvee_{s \in C} X_{i,j,s} \right) \wedge \left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} \left(\overline{\overline{X}_{i,j,s}} \vee \overline{\overline{X}_{i,j,t}} \right) \right) \right]$$

Ogni cella contiene almeno un simbolo

Ogni cella contiene al massimo un simbolo

Dimensione di φ_{cell} :

$$\varphi_{\text{cell}} = \underset{\text{all } i,j}{\wedge} \left[\left(\bigvee_{s \in C} X_{i,j,s} \right) \wedge \left(\underset{\substack{s,t \in C \\ s \neq t}}{\wedge} \left(\overline{X}_{i,j,s} \vee \overline{X}_{i,j,t} \right) \right) \right]$$



$$(2n^k + 3)^2 \times (|C| + |C|^2)$$
$$= O(n^{2k})$$

φ_{start} il tableau parte con la configurazione iniziale

$$\begin{aligned}\varphi_{\text{start}} = & X_{1,1,\#} \wedge X_{1,2,\diamond} \wedge \cdots \wedge X_{1,n^k+1,\diamond} \\ & \wedge X_{1,n^k+2,q_0} \wedge X_{1,n^k+3,\sigma_1} \wedge \cdots \wedge X_{1,n^k+n+2,\sigma_n} \\ & \wedge X_{1,n^k+n+3,\diamond} \wedge X_{1,2n^k+2,\diamond} \wedge \cdots \wedge X_{1,2n^k+2,\#}\end{aligned}$$

Describe la configurazione iniziale
Nella riga 1 del tableau

Dimensione di φ_{start} :

$$\begin{aligned}\varphi_{\text{start}} = & X_{1,1,\#} \wedge X_{1,2,\diamond} \wedge \cdots \\ & \wedge X_{1,n^k+1,\diamond} \wedge X_{1,n^k+2,q_0} \wedge X_{1,n^k+3,\sigma_1} \wedge \cdots \\ & \wedge X_{1,2n^k+2,\diamond} \wedge X_{1,2n^k+3,\#}\end{aligned}$$

↓

$$2n^k + 3 = O(n^k)$$

φ_{accept}

Ci dà la sicurezza che La
computazione raggiunge stato di
Accettazione

$$\varphi_{\text{accept}} = \bigvee_{\substack{\text{all } i, j \\ \text{all } q \in F}} X_{i,j,q}$$



Stati di accettazione

Uno stato di accettazione deve apparire
Da qualche parte nel tableau

Size of φ_{accept} :

$$\varphi_{\text{accept}} = \bigvee_{\substack{\text{all } i,j \\ \text{all } q \in F}} X_{i,j,q}$$



$$(2n^k + 3)^2 = O(n^{2k})$$

φ_{move}

Ci rende sicuri che il tableau ci da una sequenza valida di configurazioni

φ_{move} è espresso in termini di windows legali

Tableau

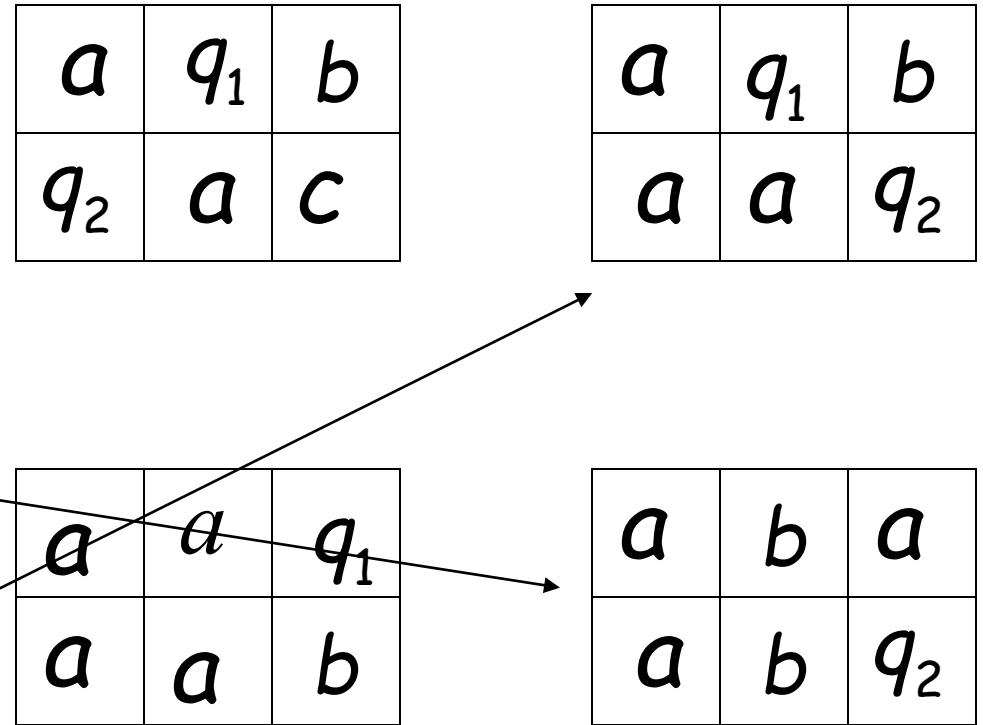
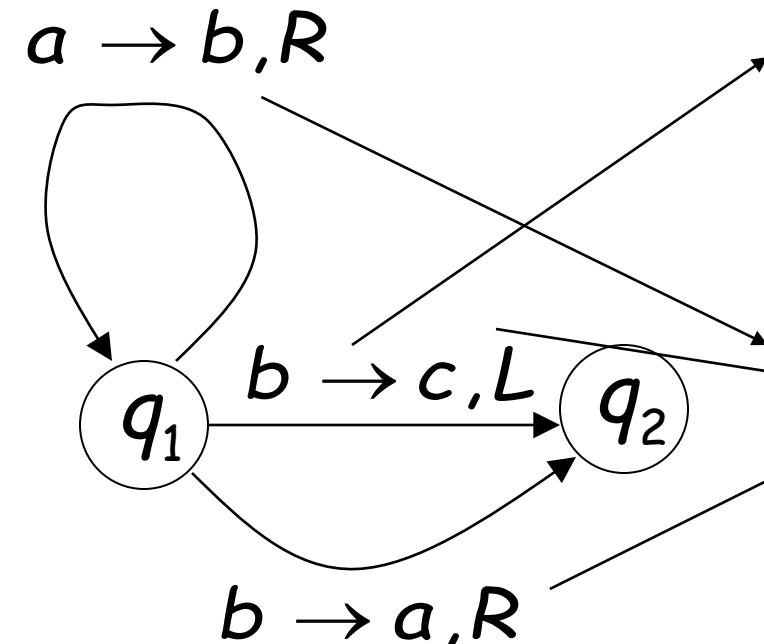
Per ogni
i,j,
elemento
centrale
superiore

Window

a	q_1	b
q_2	a	c

2x6 area di celle

Possibili windows Legali



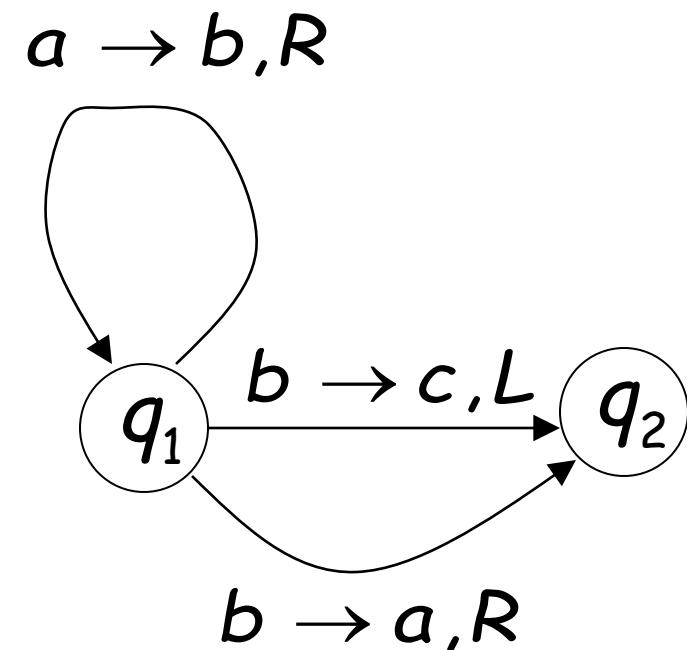
window Legali obbediscono alle transizioni

Altre finestre legali

#	b	a
#	b	a

b	b	b
c	b	b

Possibili window illegali



La b in a?

a	b	a
a	a	a

a	q_1	b
q_1	a	a

Due stati

b	q_1	b
q_2	b	q_2

Se $\delta(q_1, b) = (q_1, c, R)$

	<i>j</i>		
<i>i</i>	<i>a</i>	<i>q</i> ₁	<i>b</i>
	<i>q</i> ₂	<i>a</i>	<i>c</i>
	<i>(è legale)</i>		

Formula:

$$\begin{aligned}
 & x_{i,j,a} \wedge x_{i,j+1,q_1} \wedge x_{i,j+2,b} \\
 & \wedge x_{i+1,j,q_2} \wedge x_{i+1,j+1,a} \wedge x_{i+1,j+2,c}
 \end{aligned}$$

$$\varphi_{\text{move}} = \bigwedge_{\text{all } i,j} (\text{window } (i,j) \text{ is legal})$$

window (i,j) è legale:

	j	i	j	i	j	i	...
j							
i	a	q_1	b	a	q_1	b	
q_2	a	c	a	a	q_2	a	
((is legal))	e	(is legal)	e	(is legal))			

Tutte le possibili celle legali
nella posizione (i, j)

Dimensione di φ_{move} :

Dimensione di una formula per una window
Legale in una cella i,j : 6

- X Numero di possibili legal window
in una cella i,j : al max $|C|^6$
 - X Numero di possibili celle: $(2n^k + 3)n^k$
- $$= 6 \cdot |C|^6 \cdot (2n^k + 3)n^k = O(n^{2k})$$

Dimensione di $\varphi(M, w)$:

$$\begin{aligned}\varphi(M, w) &= \varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{accept}} \wedge \varphi_{\text{move}} \\ &\quad \swarrow \qquad \downarrow \qquad \downarrow \qquad \downarrow \\ O(n^{2k}) + O(n^k) + O(n^{2k}) + O(n^{2k}) \\ &= O(n^{2k})\end{aligned}$$

Costruita in Tempo $O(n^{2k})$

Quindi Polinomiale in n

$$\varphi(M, w) = \varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{accept}} \wedge \varphi_{\text{move}}$$

Abbiamo che:

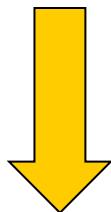
$$w \in L \Leftrightarrow \varphi(M, w) \text{ is satisfiable}$$

poichè,

$w \in L \Leftrightarrow \varphi(M, w)$ is satisfiable

e

$\varphi(M, w)$ è costruita
in Tempo Polinomiale



L è riducibile in tempo Polinomiale a SAT

END OF Dim

Osservazione 1:

La $\varphi(M, w)$ formula può essere convertita in CNF (conjunctive normal form) formula in Tempo Polinomiale

$$\varphi(M, w) = \varphi_{\text{cell}} \wedge \varphi_{\text{start}} \wedge \varphi_{\text{accept}} \wedge \varphi_{\text{move}}$$

gia CNF

NOT CNF

```
graph LR; A[gia CNF] --> B["\varphi_{cell}"]; A --> C["\varphi_{start}"]; A --> D["\varphi_{accept} \wedge \varphi_{move}"]; E[NOT CNF] --> D;
```

Ma può essere convertita
in CNF (distribuitività)

leggi: Distributività

$$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$$

$$P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$$

Osservazione 2:

La $\varphi(M, w)$ formula può essere
Convertita in una formula 3CNF
in Tempo Polinomiale

$$(a_1 \vee a_2 \vee \cdots \vee a_l)$$



$$(a_1 \vee a_2 \vee z_1) \wedge (\overline{z_1} \vee a_3 \vee z_2) \wedge (\overline{z_2} \vee a_4 \vee z_3) \wedge \cdots \wedge (\overline{z_{l-3}} \vee a_{l-1} \vee z_l)$$

Nuove z

3CNF formula:

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

The diagram shows a 3CNF formula with four clauses separated by '&'. Each clause contains three literals separated by 'vee' (\vee). A bracket under the first two clauses is labeled 'clause'. An arrow points from the word 'literal' to the first literal in the second clause, x_3 .

Ogni clausola ha tre letterali

Linguaggio:

3CNF-SAT = { $w : w$ è una formula
3CNF soddisfacibile}

CNF-SAT e
3CNF-SAT sono
Linguaggi NP-completi

(sappiamo che sono NP Linguaggi)

Esempio di riduzione Tempo-Polinomiale:

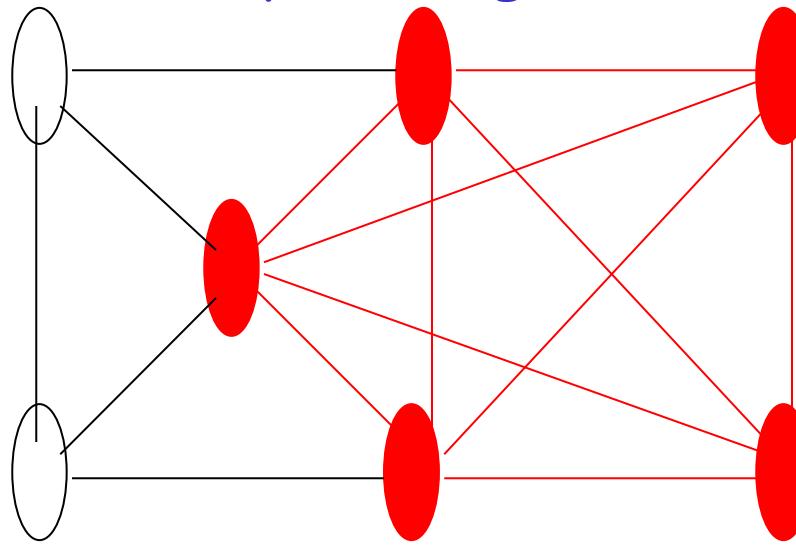
Ridurremo il problema

3CNF-satisfiability

al

problema CLIQUE

Una 5-cricca (Clique) nel grafo G



Linguaggio:

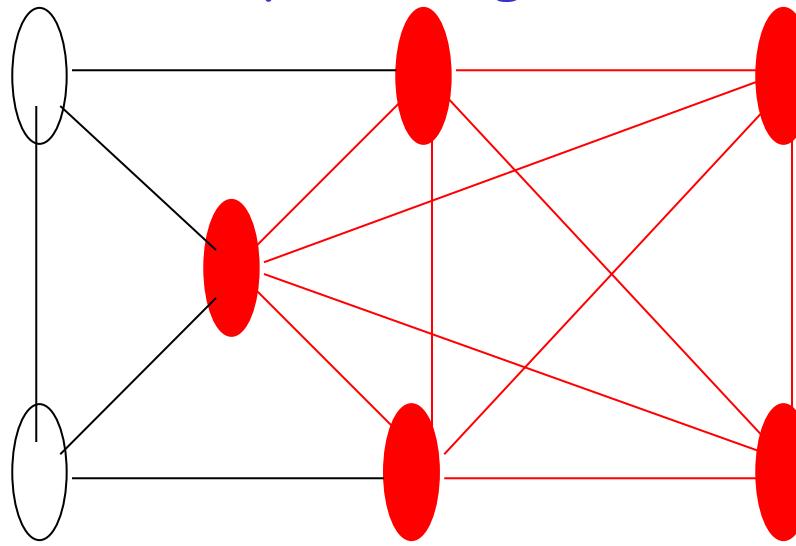
$\text{CLIQUE} = \{<G, k> : \text{grafo } G$
 $\text{contiene un } k \text{ -clique}\}$

Teorema: 3CNF-SAT è riducibile
In Tempo Polinomiale
a CLIQUE

Dim: Una riduzione in Tempo Polinomiale
da un problema all'altro

Transformare una formula in un grafo

Una 5-cricca (Clique) nel grafo G



Linguaggio:

$\text{CLIQUE} = \{<G, k> : \text{grafo } G$
 $\text{contiene un } k \text{ -clique}\}$

Teorema: 3CNF-SAT è riducibile
In Tempo Polinomiale
a CLIQUE

Dim: Una riduzione in Tempo Polinomiale
da un problema all'altro

Transformare una formula in un grafo

3CNF formula:

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

The diagram shows a 3CNF formula with four clauses separated by '&'. The first clause is $(x_1 \vee \overline{x_2} \vee \overline{x_3})$. The second clause is $(x_3 \vee \overline{x_5} \vee x_6)$, with x_3 underlined. The third clause is $(x_3 \vee \overline{x_6} \vee x_4)$. The fourth clause is $(x_4 \vee x_5 \vee x_6)$. A bracket under the second and third clauses is labeled 'clause'. An arrow points from the word 'literal' to the term $\overline{x_5}$ in the second clause.

Ogni clausola ha tre letterali

Linguaggio:

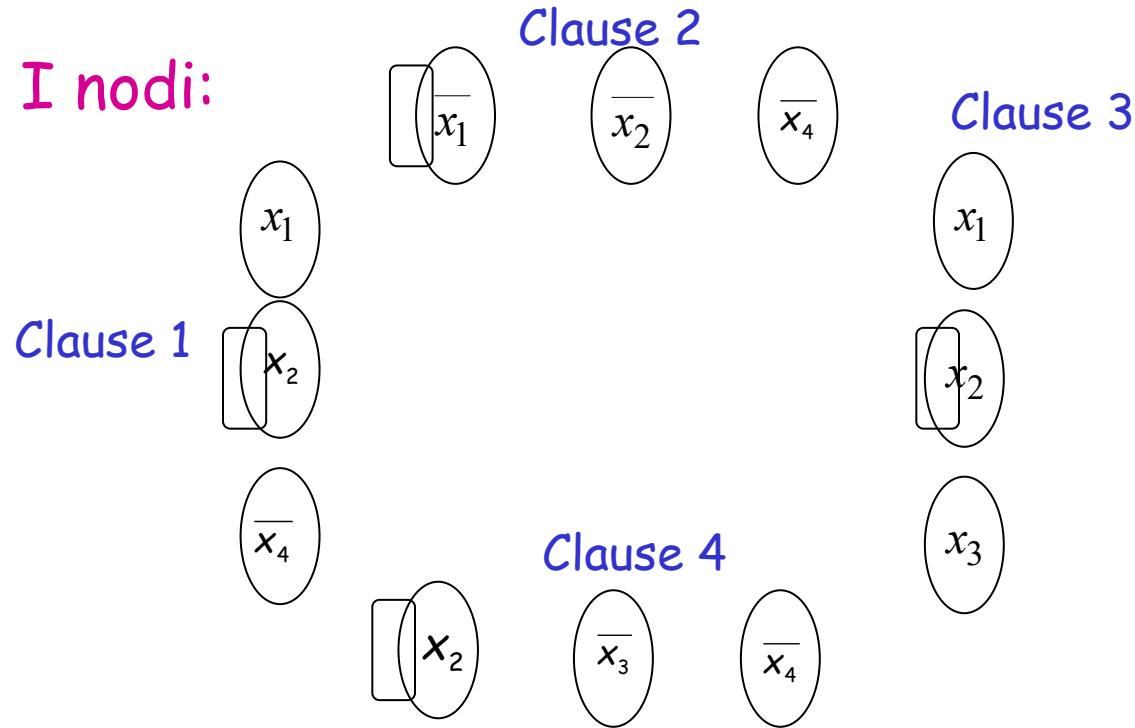
3CNF-SAT = { $w : w$ è una formula
3CNF soddisfacibile }

Transformare una formula in un grafo

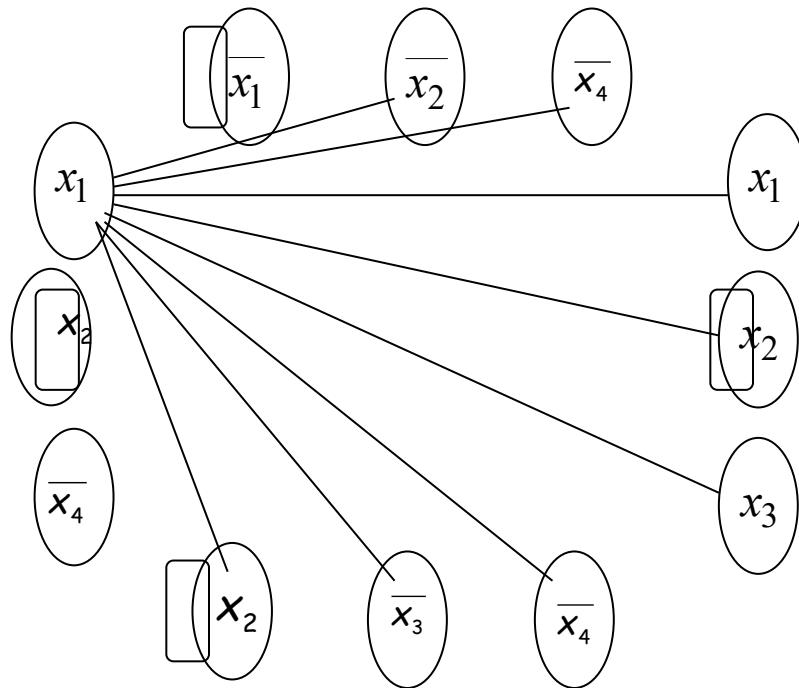
Esempio:

$$(x_1 \vee x_2 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3 \vee \overline{x}_4)$$

Creare I nodi:



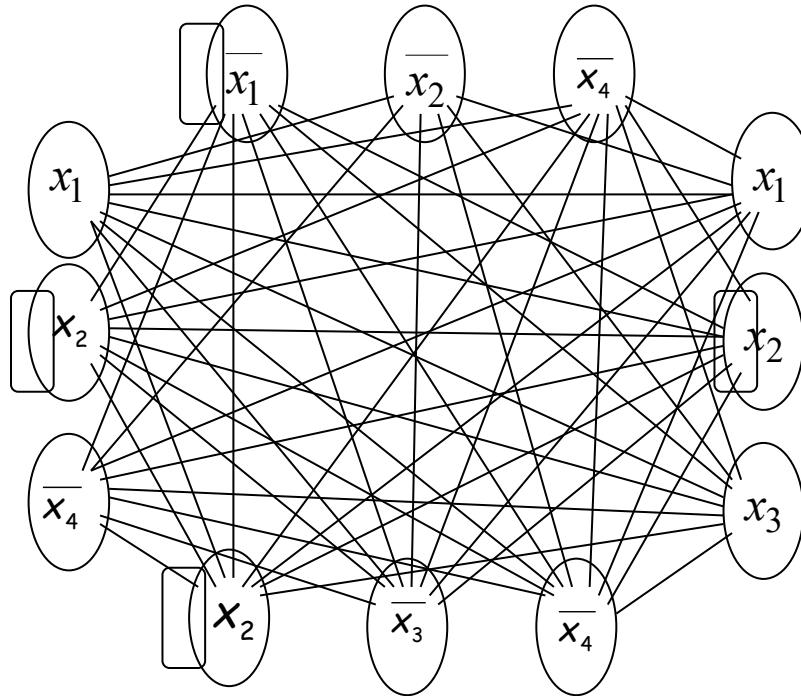
$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$



Addizionare un arco da un letterale ξ

A ogni altro letterale in ogni clausola salvo a $\overline{\xi}$

$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4})$$



Grafo risultante

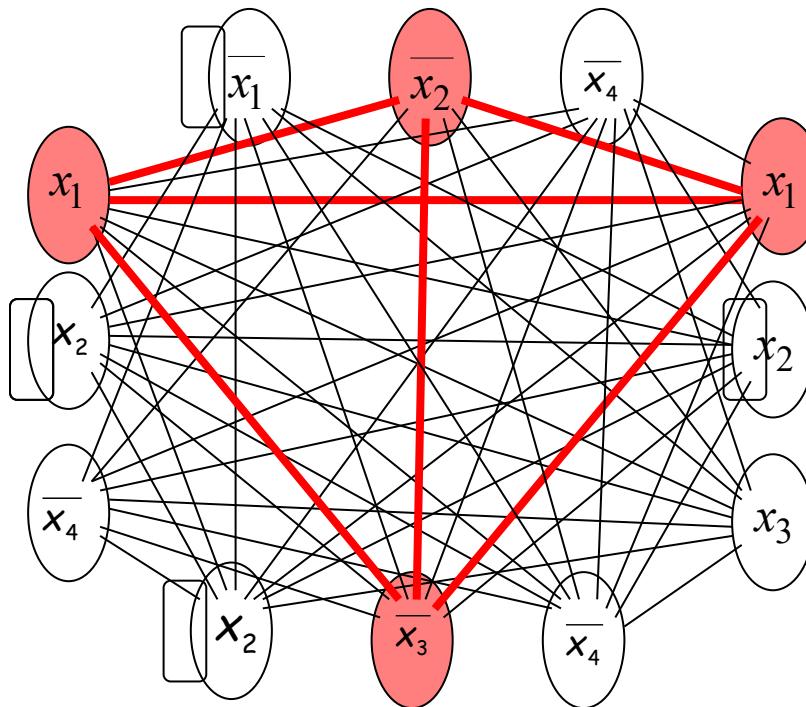
$$(x_1 \vee x_2 \vee \overline{x_4}) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_4}) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee \overline{x_3} \vee \overline{x_4}) = 1$$

$$x_1 = 1$$

$$x_2 = 0$$

$$x_3 = 0$$

$$x_4 = 1$$



La formula è soddisfacibile se e solo se il grafo ha
un 4-clique

End of Dim

Teorema:

If: a. Linguaggio A is NP-complete
b. Linguaggio B is in NP
c. A è riducibile in Tempo Polinomiale a B

Then: B is NP-complete

Dim:

Ogni linguaggio L in NP

È riducibile in Tempo Polinomiale a A

allora, L è riducibile in Tempo Polinomiale
a B

(somma di due riducibilità Polinomiali dà
una riduzione Polinomiale)

Corollario: CLIQUE è NP-complete

Dim:

- a. 3CNF-SAT è NP-complete
- b. CLIQUE è in NP (già visto)
- c. 3CNF-SAT è riducibile Polinomiale a CLIQUE
(già visto)

Applichiamo il teorema precedente

A=3CNF-SAT e B=CLIQUE

Fine chap

Osservazione:

SE si dimostra che un Linguaggio
NP-complete
è in P allora:

$$P = NP$$

Complessità spaziale
o meglio
Complessità nello spazio

- Considera una Turing Machine deterministica
- M che decide un linguaggio L

La complessità spaziale di M (deterministica) è una funzione $f:N \rightarrow N$, dove $f(n)$ è il numero massimo di celle che M usa su ogni input di lunghezza n .

Per ogni stringa w la computazione di $M(w)$ termina usando una quantità finita di spazio $f(n)$, (senza limitazione di tempo)

- Considera una Turing Machine non deterministica
- M che decide un linguaggio L

La complessità spaziale di M è una funzione $f:N \rightarrow N$, dove $f(n)$ è il numero massimo di celle che per ogni ramo della computazione M usa su ogni input di lunghezza n .

Consideriamo tutte le stringhe di Lunghezza n e T una funzione da N->N

$Space_M(T(n))$ = Linguaggi decidibili (calcolabili) in spazio $O(T(n))$ deterministicamente per una qualsiasi stringa di lunghezza n

$NSpace_MT((n))$ = Linguaggi decidibili (calcolabili) in spazio $O(T(n))$ non deterministicamente per una qualsiasi stringa di lunghezza n

Esempio: $L_1 = \{a^n b : n \geq 0\}$

Può essere deciso in spazio $O(n)$

Altri esempi nella stessa classe

Space(n)

$$L_1 = \{a^n b : n \geq 0\}$$

$$\{ab^n aba : n, k \geq 0\}$$

$$\{b^n : n \text{ is even}\}$$

$$\{b^n : n = 3k\}$$

Esempi nella classe

$Space(n)$

$$\{a^n b^n : n \geq 0\}$$

$$\{ww^R : w \in \{a,b\}^*\}$$

$$\{ww : w \in \{a,b\}^*\}$$

Macchine calcolabili
in spazio Polinomiale
deterministico .

$Space(O(n^k))$

costante $k > 0$

Macchine calcolabili
in spazio Polinomiale
Non deterministico.

$NSpace(O(n^k))$

costante $k > 0$

chiaramente:

$$Space(n^{k+1}) \supset Space(n^k)$$

$$NSpace(n^{k+1}) \supset NSpace(n^k)$$

La Classi di Complessità spaziale

$$P\text{Space} = \bigcup_{k>0} P\text{Space}(n^k)$$

Rapresenta:

- Algoritmi deterministici che usano spazio polinomiale
 - Problemi “trattabili nello spazio”

$$NP\text{Space} = \bigcup_{k>0} NP\text{Space}(n^k) \quad NP$$

Rapresenta:

- Algoritmi non deterministici che usano spazio polinomiale

tempo complessità:

Il numero di passi (step)
durante una computazione

spazio complessità:

spazio usato
durante una computazione

Ricorda che una macchina di Turing deterministica M simula una macchina di Turing M' non deterministica con una complessità di tempo esponenziale rispetto alla complessità di M .

Teorema di Savitch: Per ogni funzione $f: \mathbb{N} \rightarrow \mathbb{R}^+$, dove $f(n) \geq n$, abbiamo
 $\text{NSPACE}(f(n)) \subseteq \text{SPACE}(f^2(n))$.

una MdT deterministica può simulare una TM non deterministica usando soltanto una piccola parte di spazio aggiuntivo.

Questo è dovuto al fatto che lo spazio può essere riutilizzato, mentre il tempo no.

Se potessimo utilizzare lo stesso concetto con il tempo avremmo una prova che $P=NP$

Teorema di Savitch:

Per ogni funzione $f: \mathbb{N} \rightarrow \mathbb{R}^+$, dove $f(n) \geq n$,
abbiamo

$$\text{NSPACE}(O(f(n))) \subseteq \text{SPACE}(O(f^2(n))).$$

teorema di Savitch dimostra che nello spazio la
complessità di questa simulazione,
non determinismo \rightarrow determinismo,
aumenta in modo al più quadratico.

Si consideri una **NMdT** (non deterministica) calcolata in
spazio $f(n)$.

Cerchiamo di simularla il comportamento con una MdT (deterministica) con
spazio polinomiale $f^2(n)$.

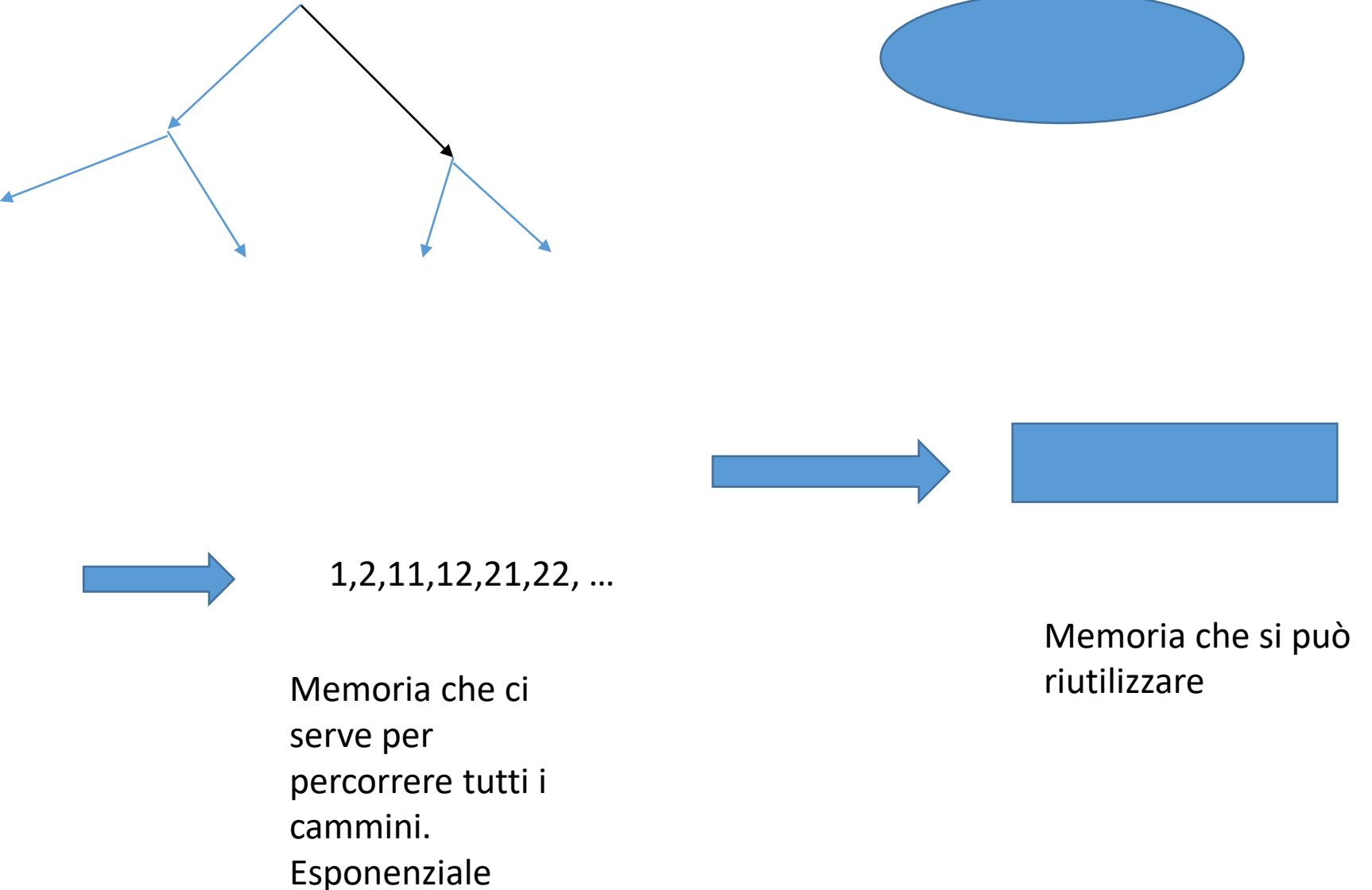
Un primo tentativo potrebbe essere quello di simulare tutti i possibili rami di computazione (di lunghezza finita) di **NMdT**, ma questo non permette di sovrascrivere l'area di memoria utilizzata, perché occorre tenere traccia delle scelte nondeterministiche fatte su un ramo specifico per scegliere un prossimo ramo.

1, 2 11,12, 111,112,121, 122

Ogni passo deve tenere traccia del cammino percorso. Nota che se usiamo uno spazio $f(n)$ il tempo che possiamo usare su quello spazio è $2^{\Omega(f(n))}$ step (perché?) ed ogni step può essere una scelta non deterministica e quindi può accadere di dover memorizzare stringhe di lunghezza $2^{\Omega(f(n))}$

Le scelte fatte per percorrere tutti i possibili cammini sono esponenziali in numero, anche se lo spazio necessario per calcolarli è polinomiale.
Dunque, questo tentativo richiede spazio esponenziale.

Metodo
Non
determinismo
Verso
determinismo
Perché non
funziona

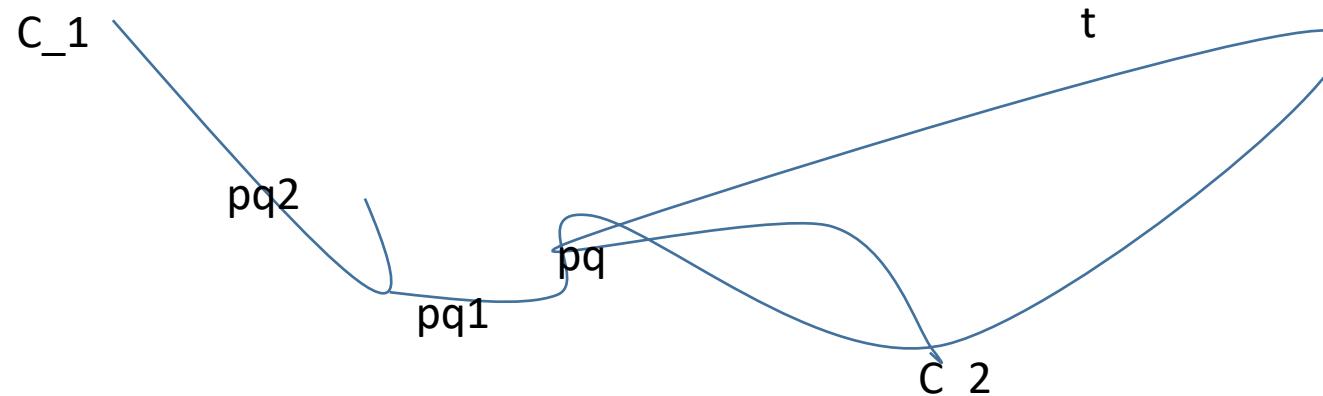


YIELDABILITY PROBLEM

Verificare se una **MNdT** con input w può passare da una configurazione iniziale c_1 a quella finale c_2 in un numero di passi minore o uguale a un numero dato t (time). (dove t verrà scelto come il numero massimo di operazioni che **MNdT** necessita per accettare w).

Si consideri una **MdT** non deterministica arbitraria.

Preso un intero t positivo, e due configurazioni c_1 e c_2 di **NMdT** Diremo che **c1 produce c2** in un numero minore o uguali di passi t se **NMdT** può produrre c_2 a partire da c_1 in t (o meno passi).



input c_1, c_2 e t , dove $t \geq 0$.

c_1, c_2 sono configurazioni che usano al più uno spazio $f(n)$ (se lo spazio occupato è minore possiamo raggiungere spazio $f(n)$ aggiungendo caratteri blank);

sia t una potenza del 2 ($t=2^p$ per qualche p maggiore o uguale a 0), possibile?.

CANYIELD ha come input anche tutti gli stati e i collegamenti tra di loro ovvero la delta

$\text{CANYIELD}(c_1, c_2, 2^t) =$

1. Se $p=0$, allora "test (se $c_1=c_2$ " oppure " c_1 produce c_2 in un solo step)" *(in base alle delta di MNdT)*.

Accept se il test ha successo, altrimenti **Reject**.

2. Se $t>0$, allora per ciascuna (tutte) configurazione c_m di N (che usano spazio $\leq f(n)$):

3. Run $\text{CANYIELD}(c_1, c_m, 2^{t-1})$.

4. Run $\text{CANYIELD}(c_m, c_2, 2^{t-1})$.

5. Se lo step 3 e 4 entrambi accept, allora accept.

6. Se non hanno entrambi accettato allora, reject

Dimostrazione del teorema di Savitch

Passiamo a definire la MdT deterministica che simula **MNdT**. Prima abbiamo bisogno di fare una assunzione semplificativa:

1. Quando **MNdT** accetta, prima di fermarsi pulisce il nastro e ritorna all'inizio del nastro, dove entrerà in una (fissata) configurazione chiamata c_{accept} .

Notazione: Con w indichiamo l'input di N , n è la lunghezza di w e c_{start} è la configurazione iniziale di **MNdT**.

Denotiamo con d una costante tale che **MNdT** non usa più di $d^*f(n)$ configurazioni per computare w .

In pratica, $2^{d^*f(n)}$ fornisce un upper bound per il tempo di esecuzione di **MNdT** su w (perché).

Con queste assunzioni, abbiamo che **MNdT** accetta w se e solo se si può andare da c_{start} a c_{accept} in al massimo $2^{d*f(n)}$ passi.

Analizziamo la seguente MdT deterministica:

M := “”Su input w :
L’output è il risultato di
CANYIELD(c_{start} , c_{accept} , $2^{d*f(n)}$) .”

Quindi **M** simula **NMdT**

Analizziamo la complessità di spazio di M.

- Ogni volta che CANYIELD si autochiama (ricorsione) memorizza lo stato corrente (i valori c_1, c_2 e t) così da poter essere richiamati (usati) al ritorno dalla ricorsione.
- Ciascun livello della ricorsione quindi usa spazio $O(f(n))$ aggiuntivo.
- Il numero delle chiamate ricorsive è invece pari a $d*f(n)$.

Dunque, lo spazio totale usato da N: $d*f(n)*O(f(n))=O(f^2(n))$

Ritorniamo per un momento sull'affermazione

M = “Su input w : L'output è il risultato di **CANYIELD(c_{start} ,
 c_{accept} , $2^{d*f(n)}$)**”

M deve sapere il valore di $f(n)$ quando invoca CANYIELD, questo lo dobbiamo calcolare. Quindi la complessità aumenta.

Un modo semplice per risolvere questo problema è quello di modificare M (chiamata M') in modo che provi per $f(n) = 1, 2, 3, \dots$ Per ogni valore $f(n) = i$

,

M' usa CANYIELD per accettare e determinare se la configurazione è raggiungibile.

Per garantire che M' non continui ad aumentare i (e quindi a consumare spazio) nei casi in cui $N\text{MdT}$ non accetta w , si effettua il seguente controllo prima di passare da i a $i+1$ (nuovo valore di $f(n)$):

Utilizzando CANYIELD, controlla che $M\text{NdT}$ può raggiungere una configurazione che utilizza più di i celle del nastro (da c_{start}). Se questo non accade: non ha alcun senso cercare per $i+1$ quindi M' va in reject.

Per memorizzare il valore i ($f(n)$): occorre spazio $O(\log f(n))$. Inoltre, questo spazio può essere riciclato ad ogni iterazione.
Dunque, la complessità dello spazio di M' rimane $O(f^2(n))$.

Fine corso

Questo è quello che conosciamo:

$$P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME.$$

Tuttavia non conosciamo se qualcuna delle inclusioni possa essere sostituita con un uguaglianza.

Sappiamo che

$$P \neq EXPTIME.$$

Quindi sappiamo che l'ultima delle tre inclusioni deve essere necessariamente un' inclusione (non una uguaglianza).

:

$P \subseteq NP \subseteq PSPACE = NPSPACE \subseteq EXPTIME$.

$P \neq EXPTIME$.

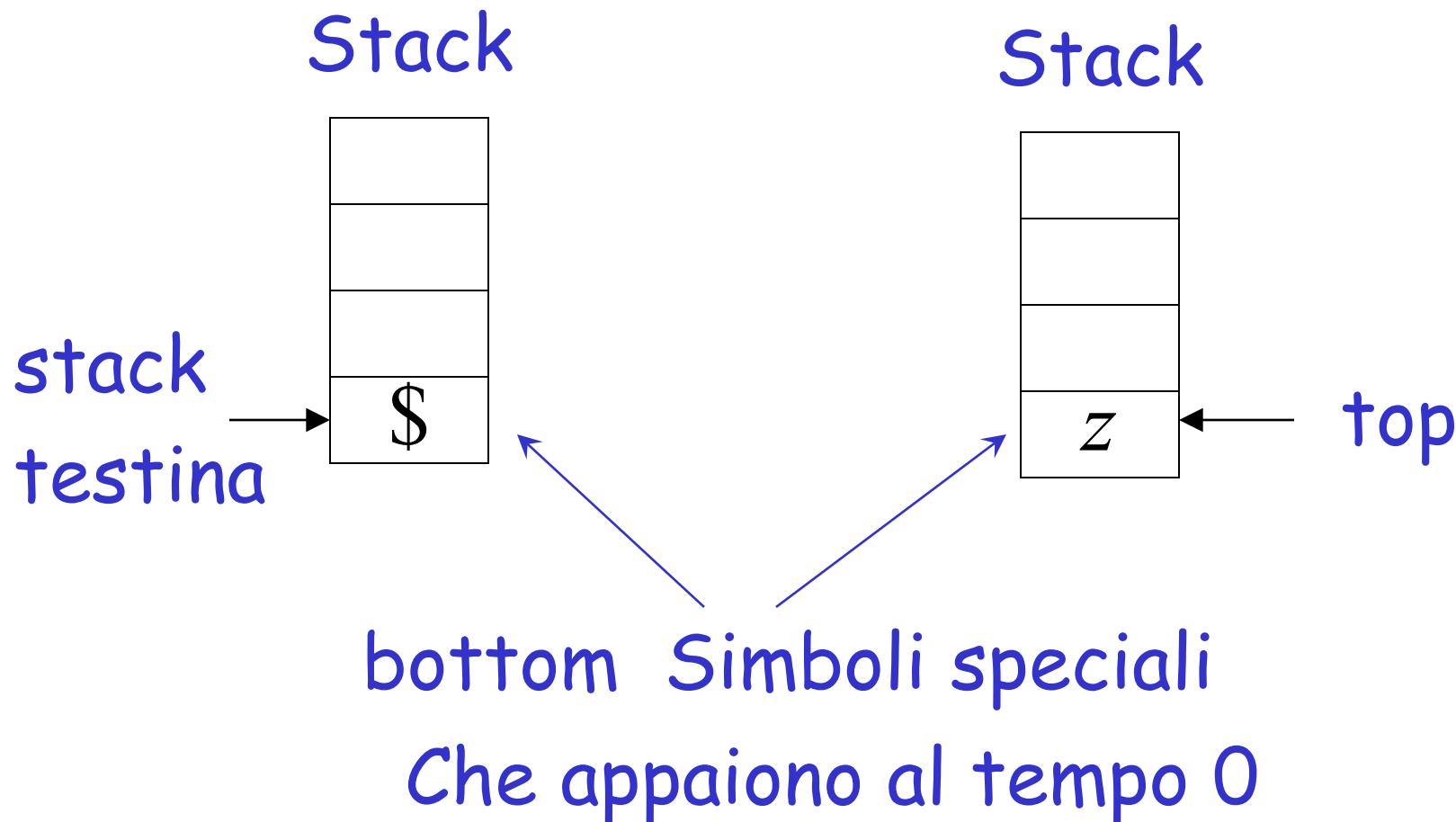
Questo corso è dedicato alla
città e agli abitanti che mi
hanno ospitato in questo
periodo.



Pushdown Automata

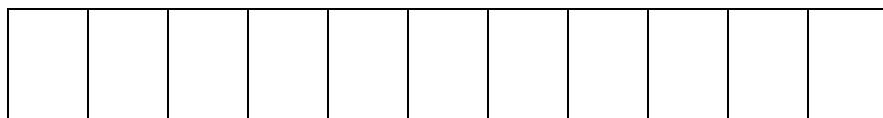
PDA

Initial Stack Symbol

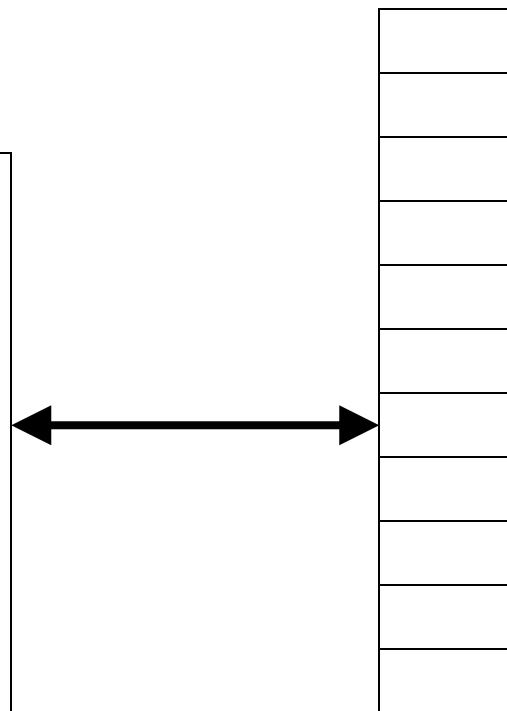
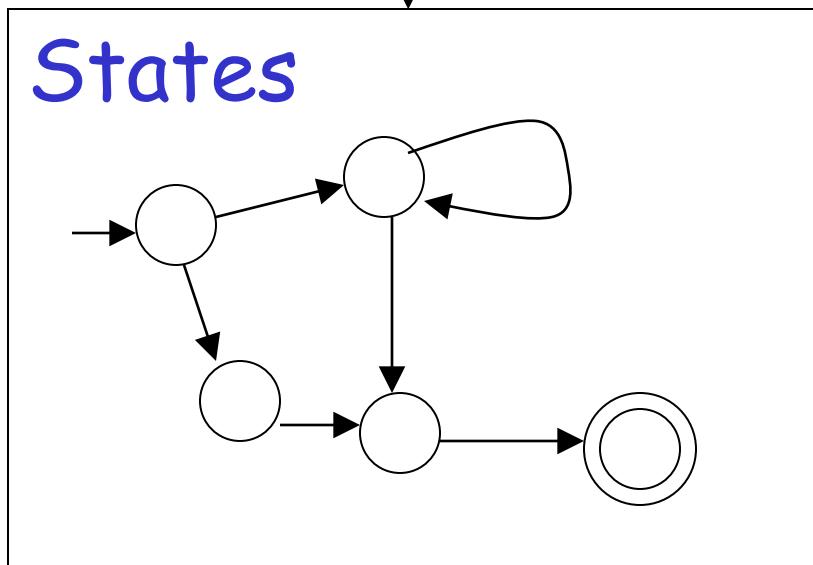


Pushdown Automaton -- PDA

Input String



Stack

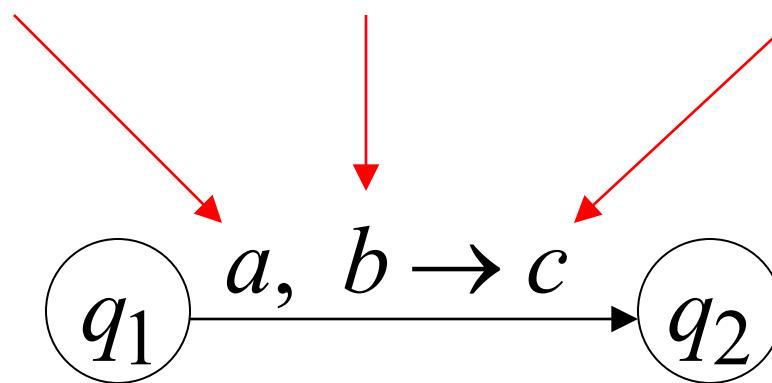


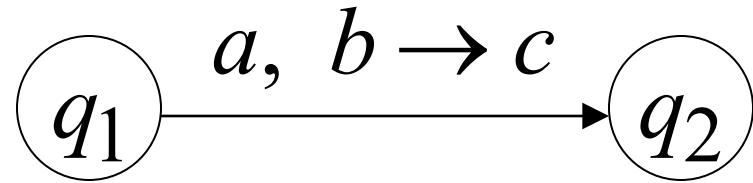
Gli stati

Input
simbolo

Pop
simbolo

Push
simbolo

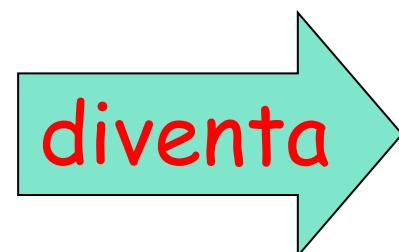




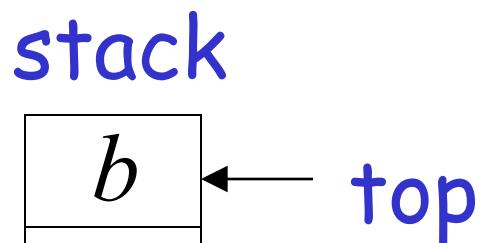
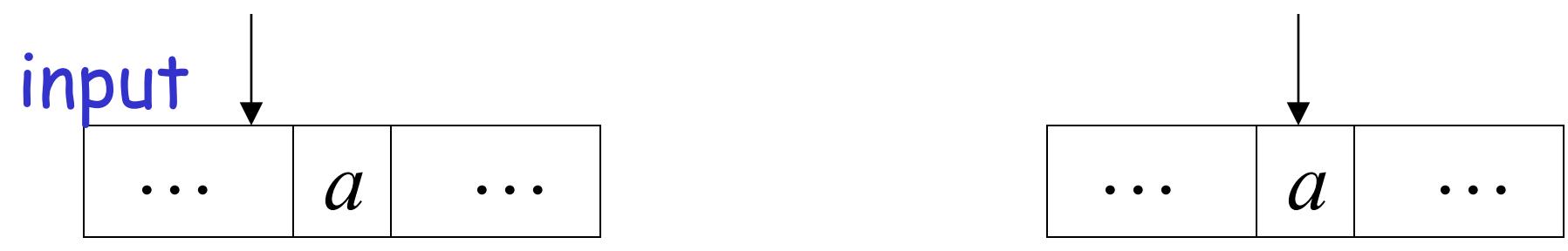
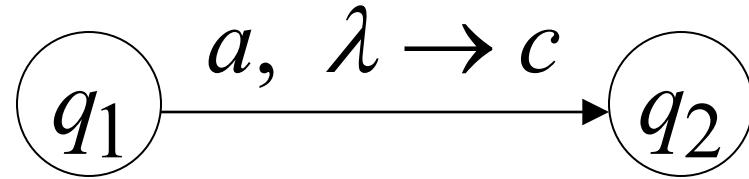
stack

b
h
e
$\$$

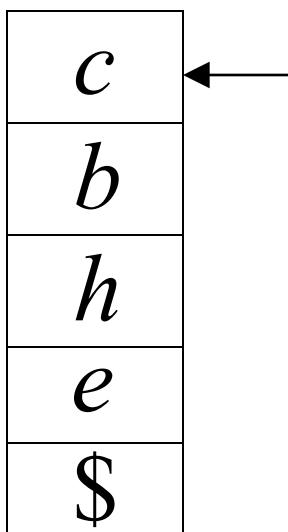
top

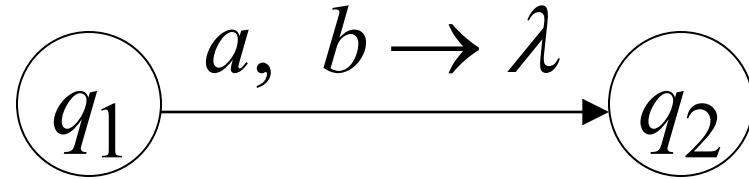


c
h
e
$\$$

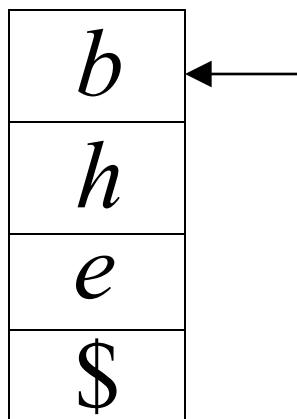


Push

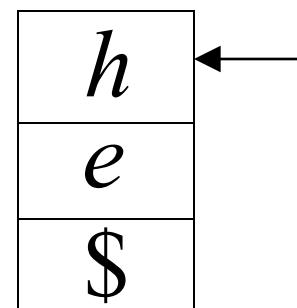
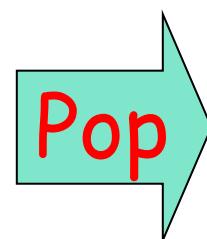


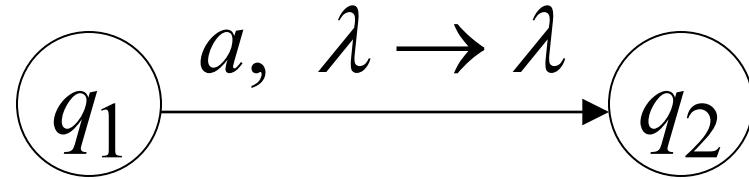


stack



top

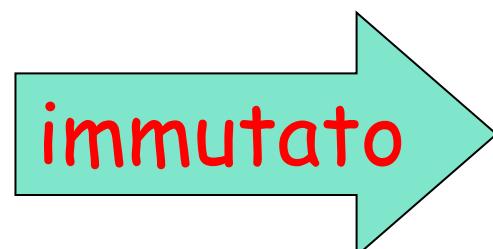




stack

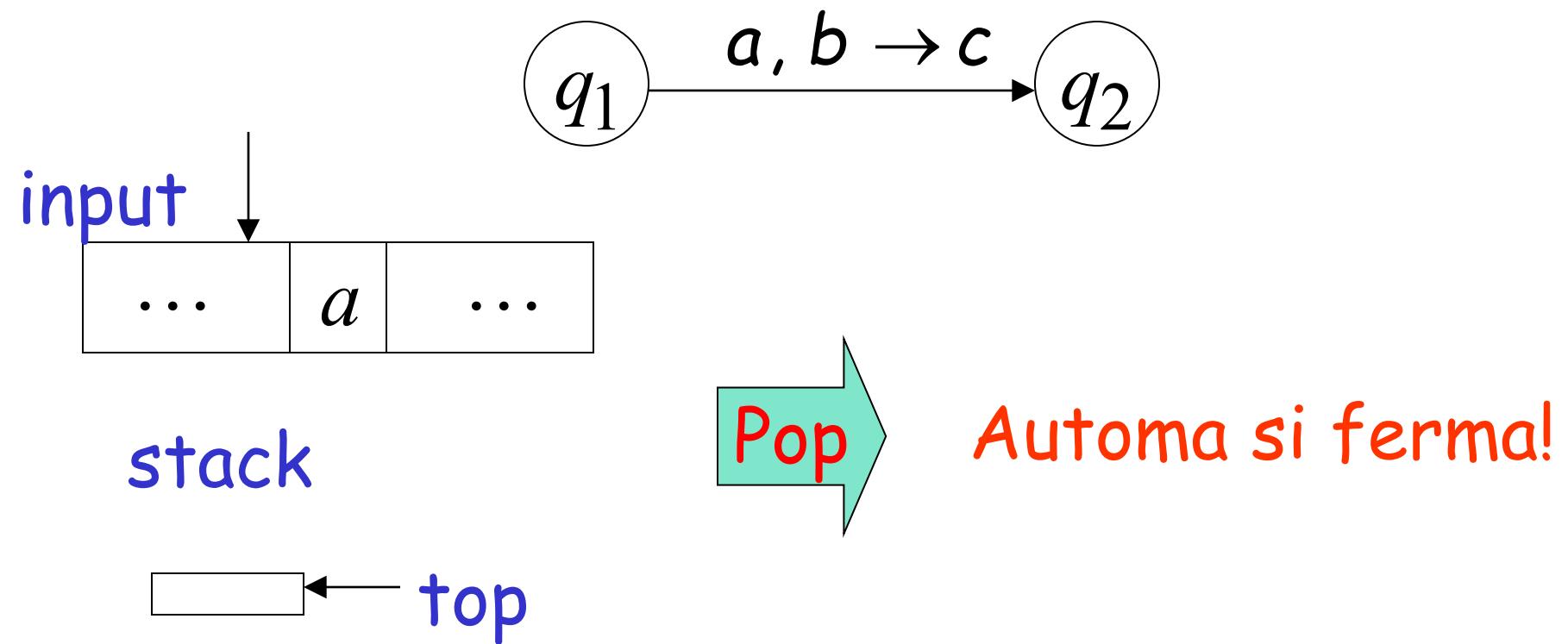
b
h
e
\$

top



b
h
e
\$

Pop da uno stack vuoto

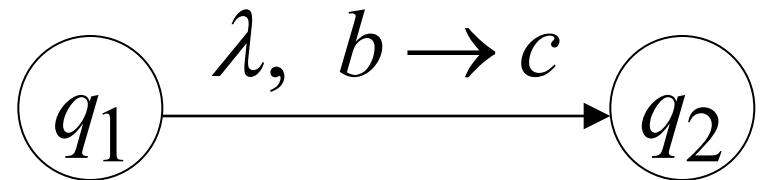
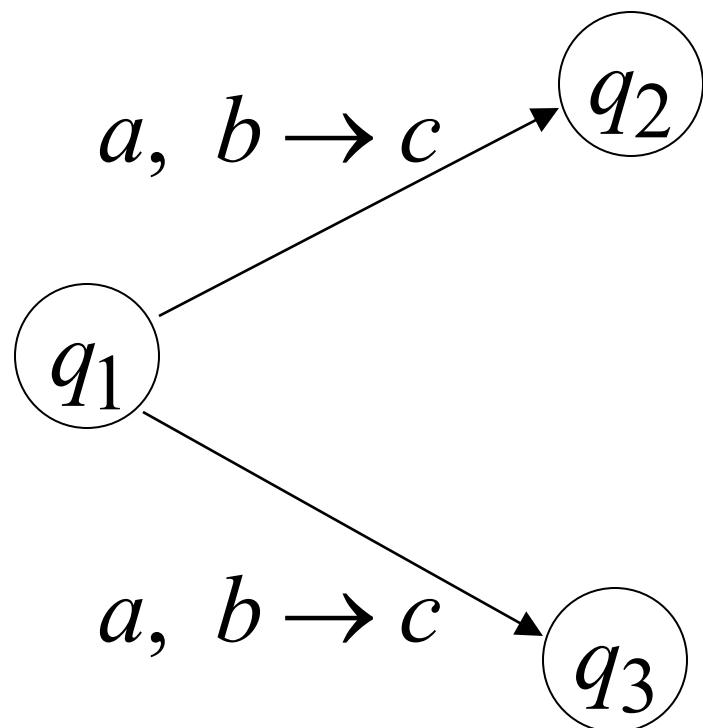


Se l'automata tenta di fare un pop
da uno stack vuoto allora si ferma
la computazione e rigetta l'input

Non-Determinismo

PDAs sono non-deterministici

Permettono transizioni non deterministiche

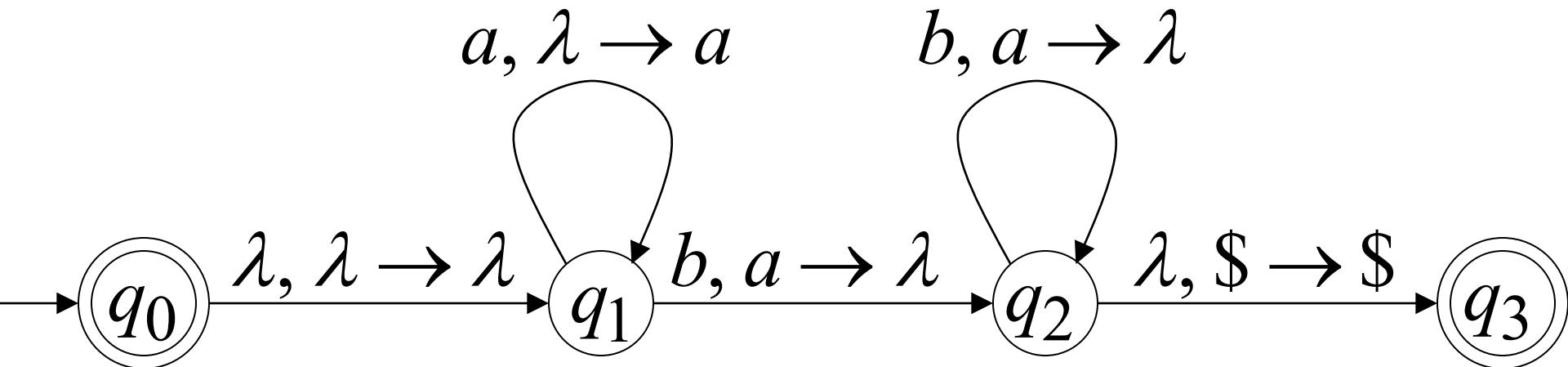


λ – transition

Esempio di PDA

PDA M :

$$L(M) = \{a^n b^n : n \geq 0\}$$



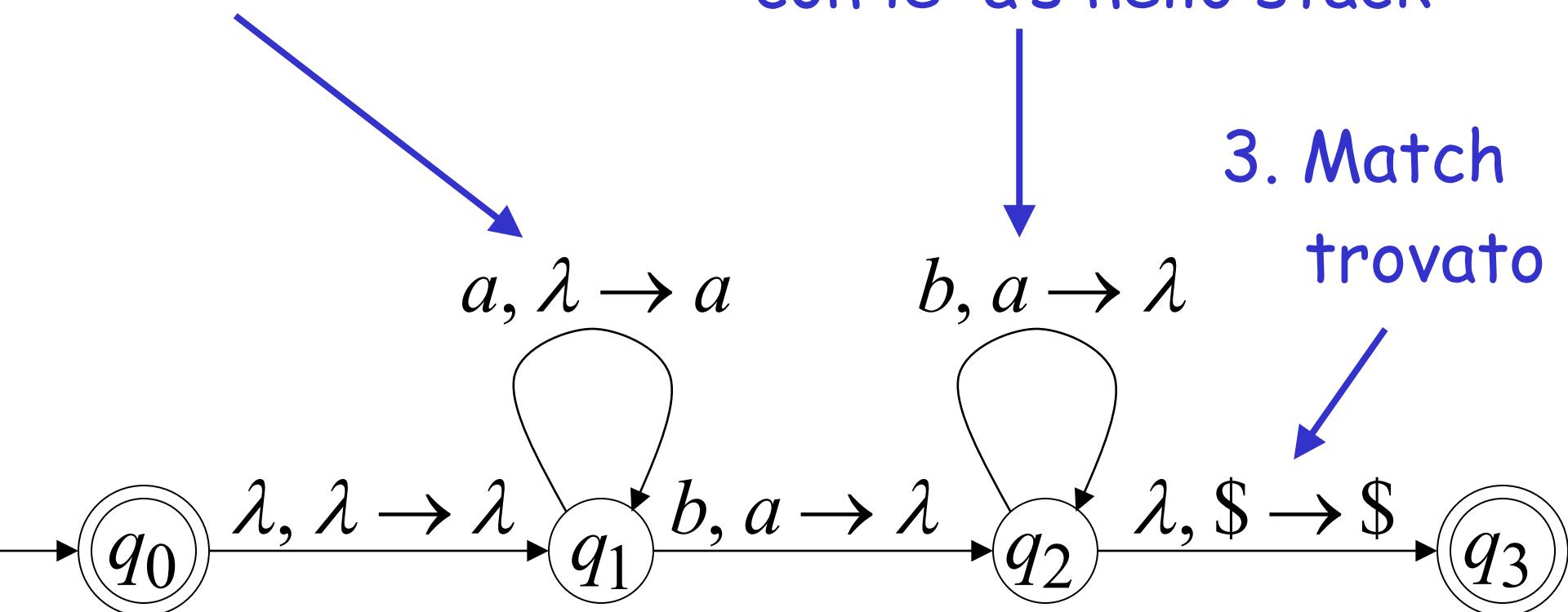
$$L(M) = \{a^n b^n : n \geq 0\}$$

Idea di base:

1. Push le a's
nello stack

2. Verifica le b's in input
con le a's nello stack

3. Match
trovato

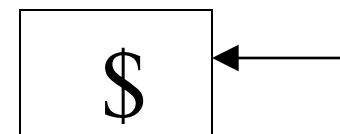


Esempio di esecuzione:

Time 0

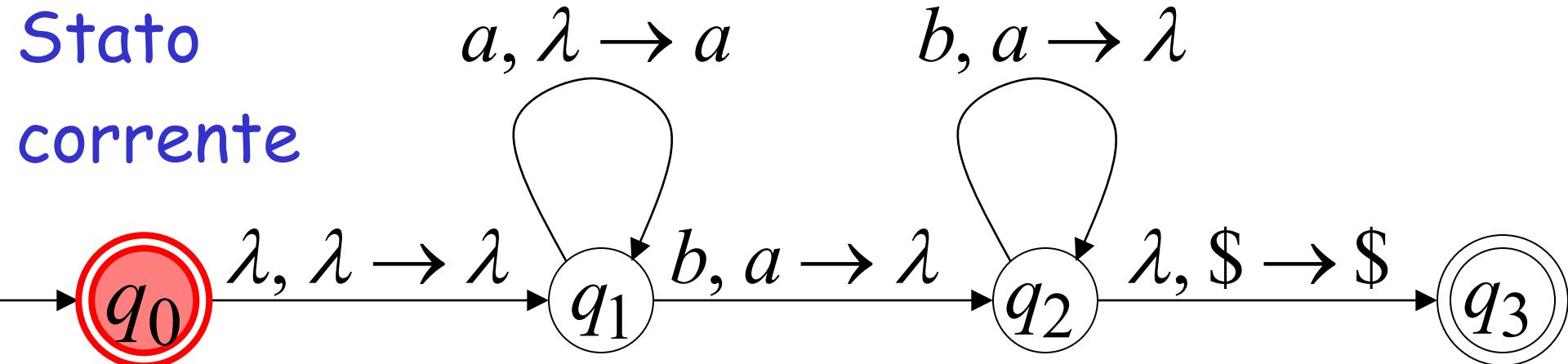
Input

a	a	a	b	b	b
-----	-----	-----	-----	-----	-----



Stack

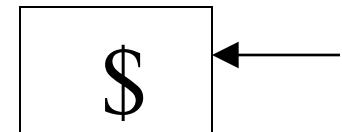
Stato
corrente



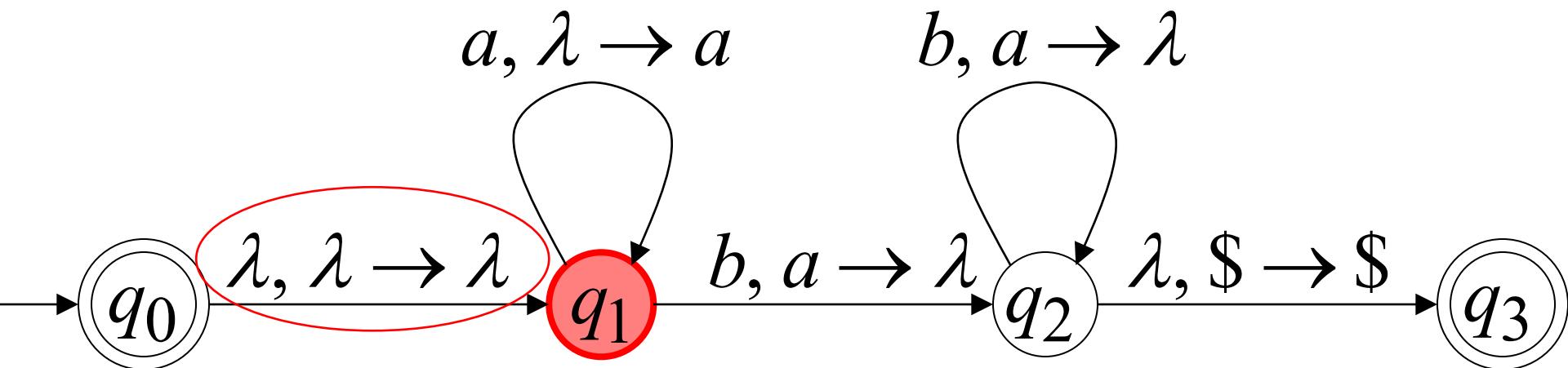
Time 1

Input

a	a	a	b	b	b
-----	-----	-----	-----	-----	-----



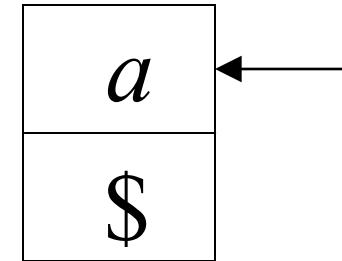
Stack



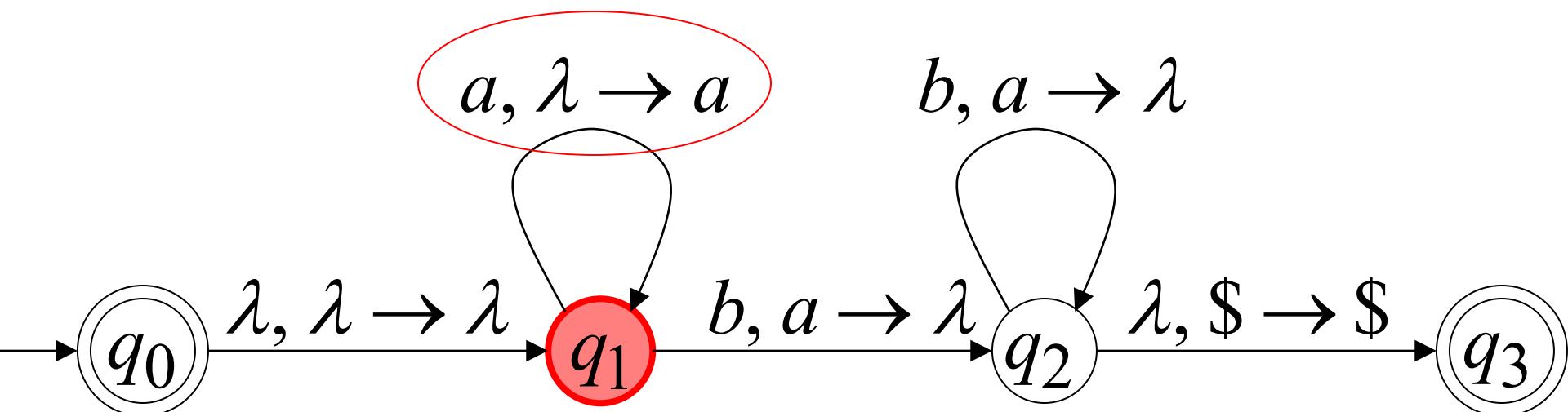
Time 2

Input

a	a	a	b	b	b
-----	-----	-----	-----	-----	-----



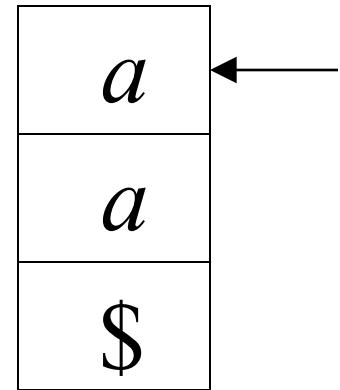
Stack



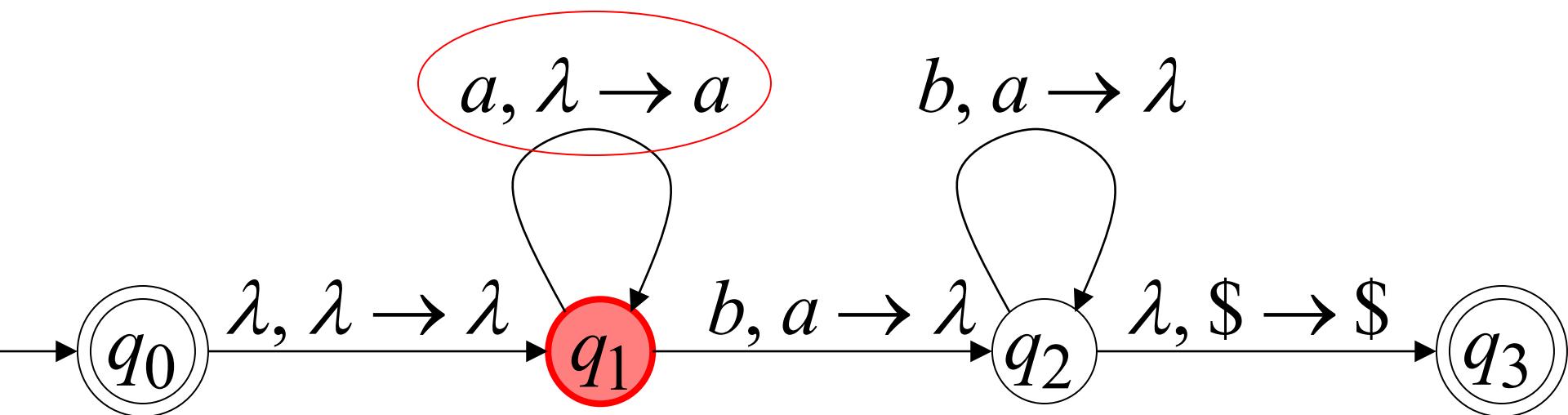
Time 3

Input

a	a	a	b	b	b
---	---	---	---	---	---



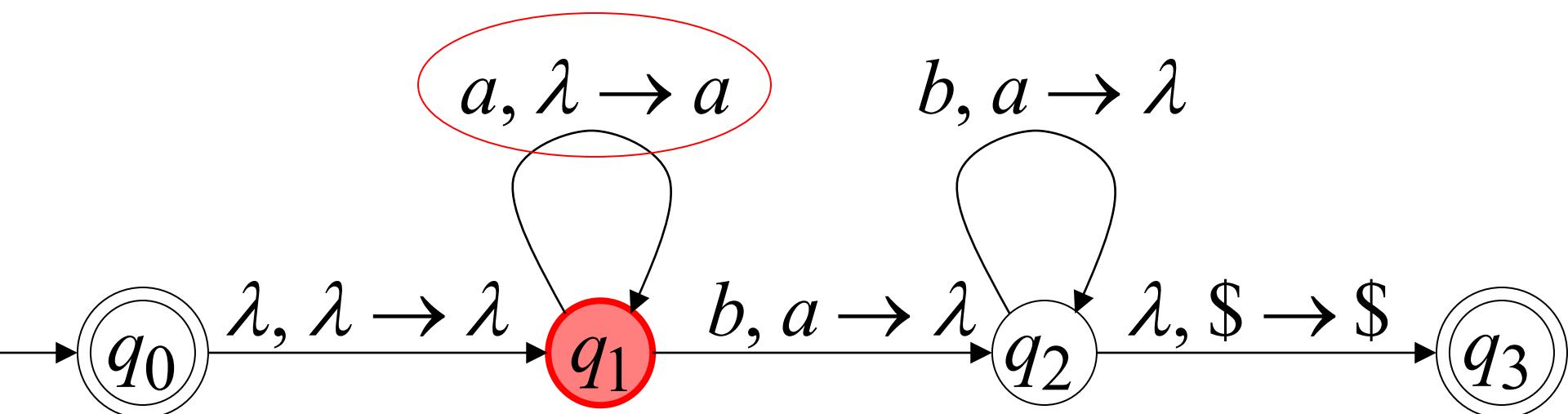
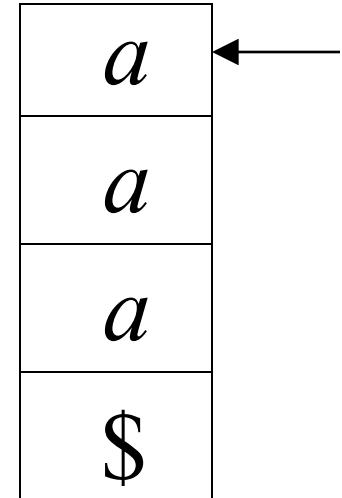
Stack



Time 4

Input

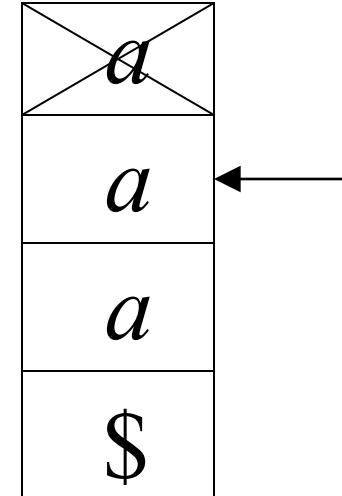
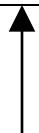
a	a	a	b	b	b
---	---	---	---	---	---



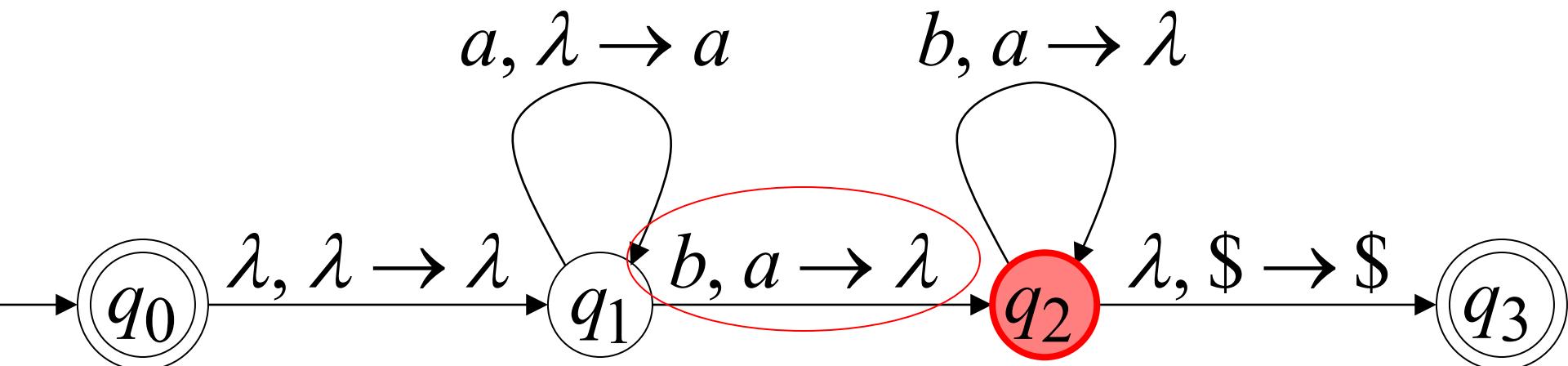
Time 5

Input

a	a	a	b	b	b
---	---	---	---	---	---



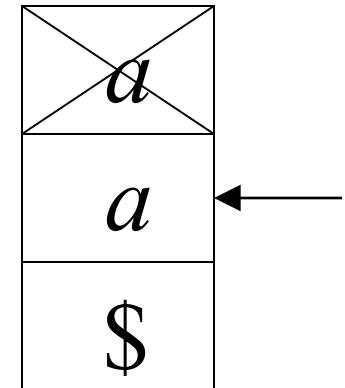
Stack



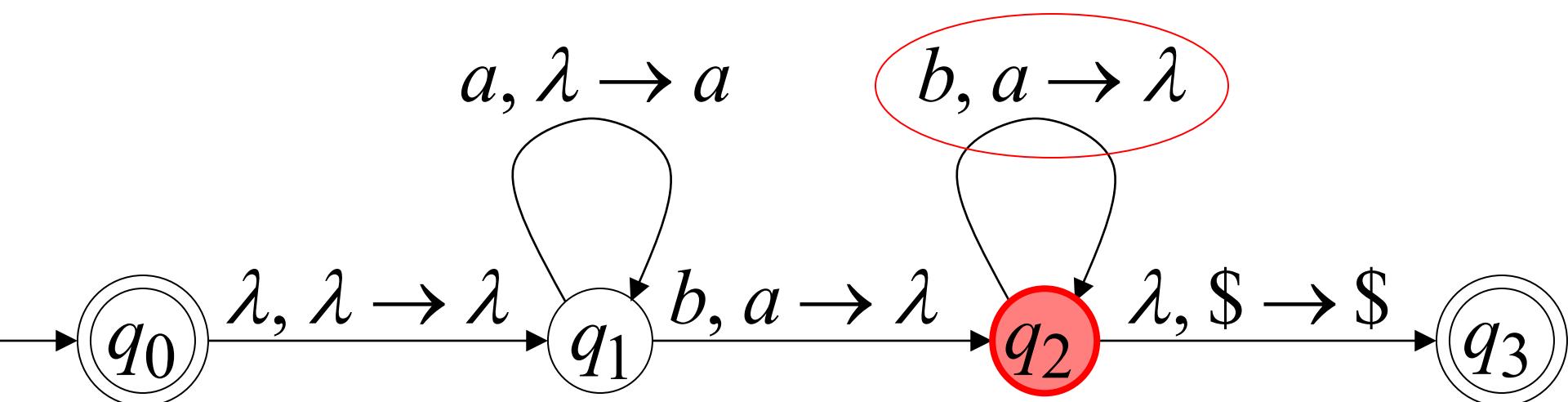
Time 6

Input

a	a	a	b	b	b
-----	-----	-----	-----	-----	-----



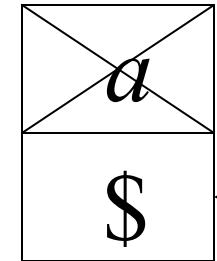
Stack



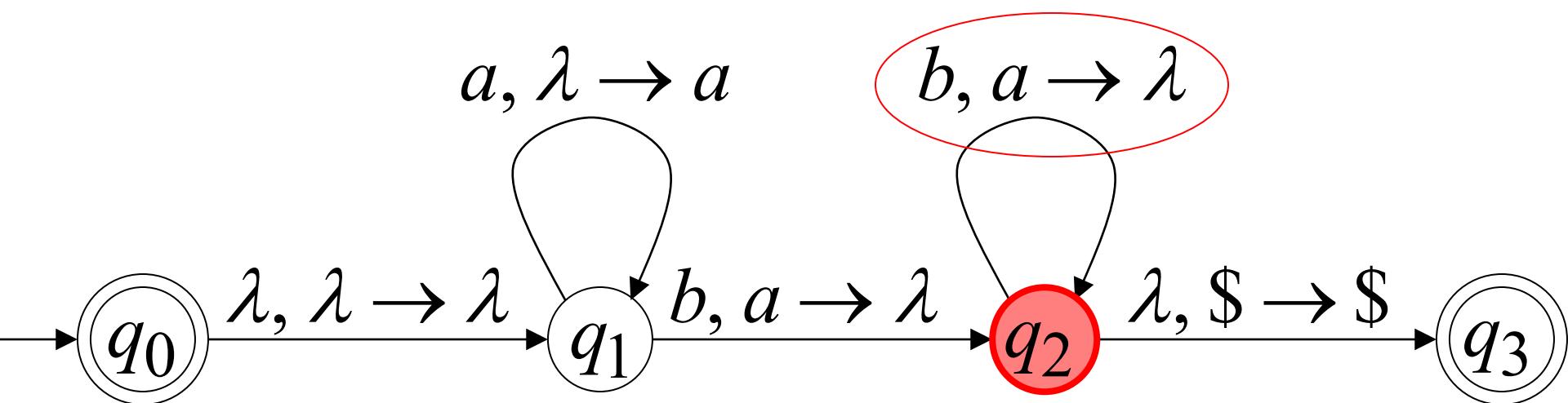
Time 7

Input

a	a	a	b	b	b
-----	-----	-----	-----	-----	-----



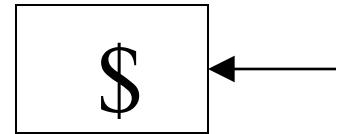
Stack



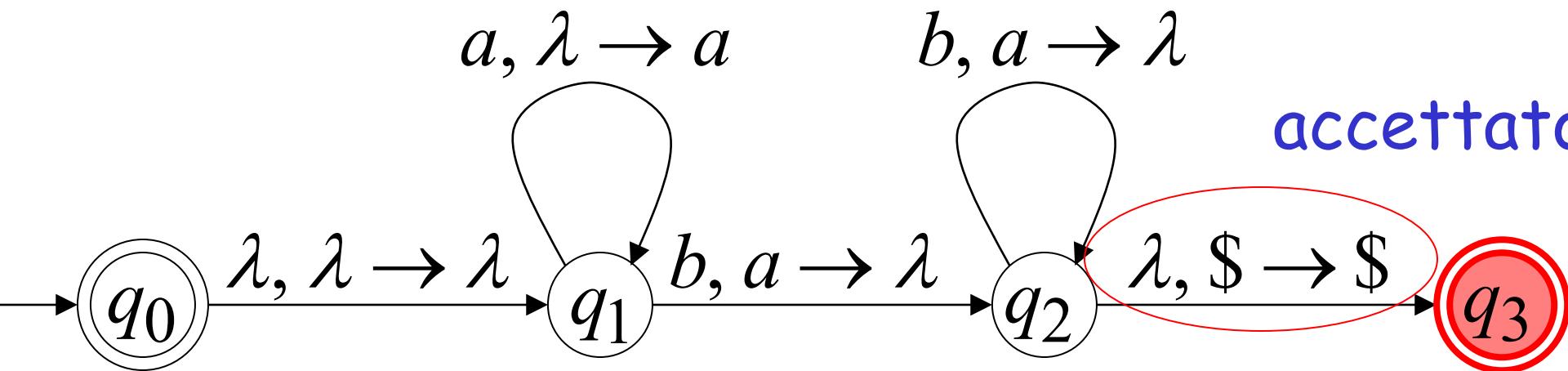
Time 8

Input

a	a	a	b	b	b
-----	-----	-----	-----	-----	-----



Stack



Una stringa è accettata se
vi è una computazione tale che:

tutti gli input sono "consumati"

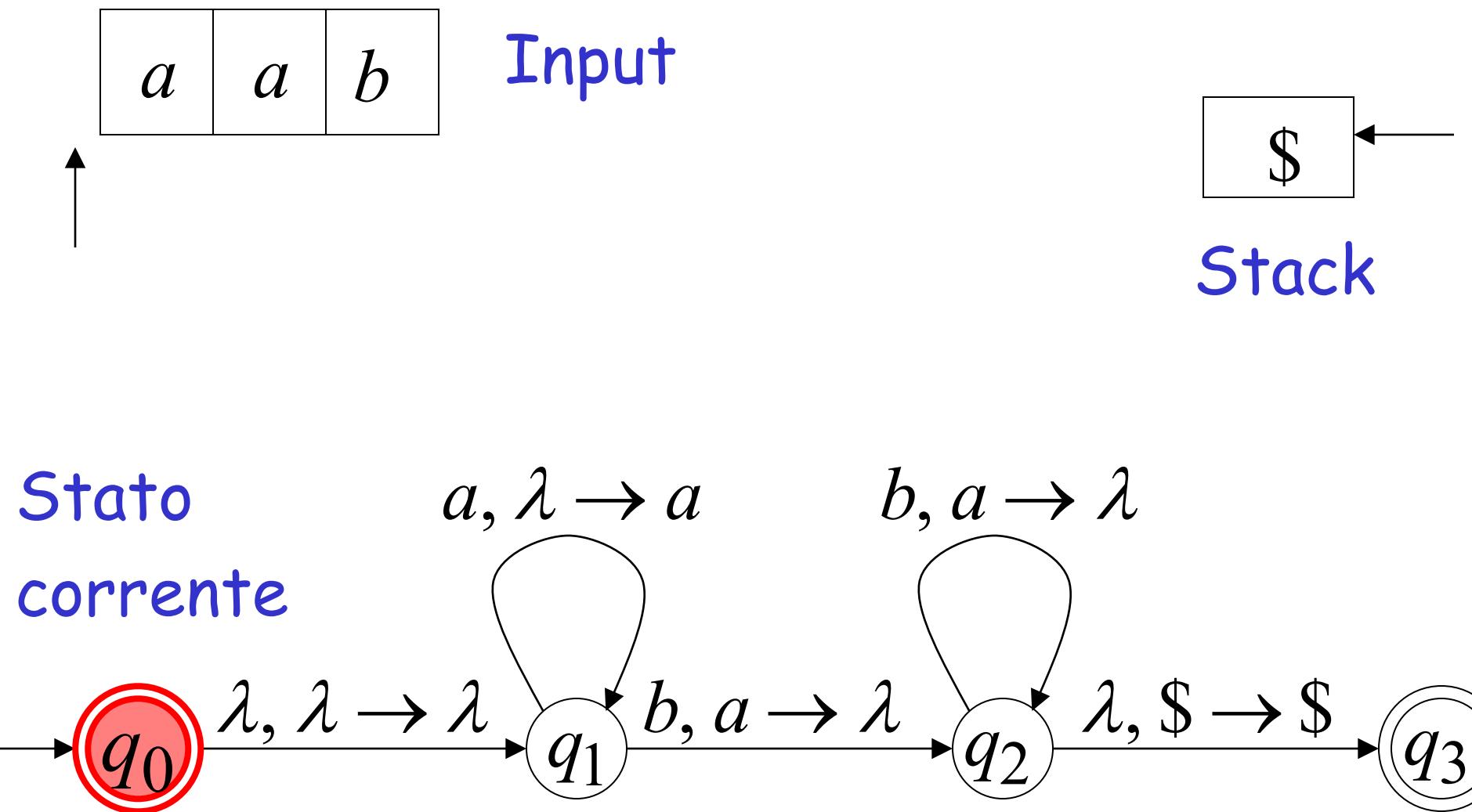
E

lo stato raggiunto è uno stato di accettazione

Non teniamo conto di quello che c'è nello
Stack alla fine dello stato di accettazione

Esempio di
non accettazione:

Time 0

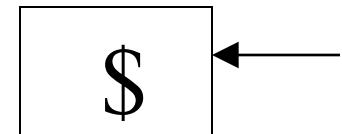


non accettazione :

Time 1

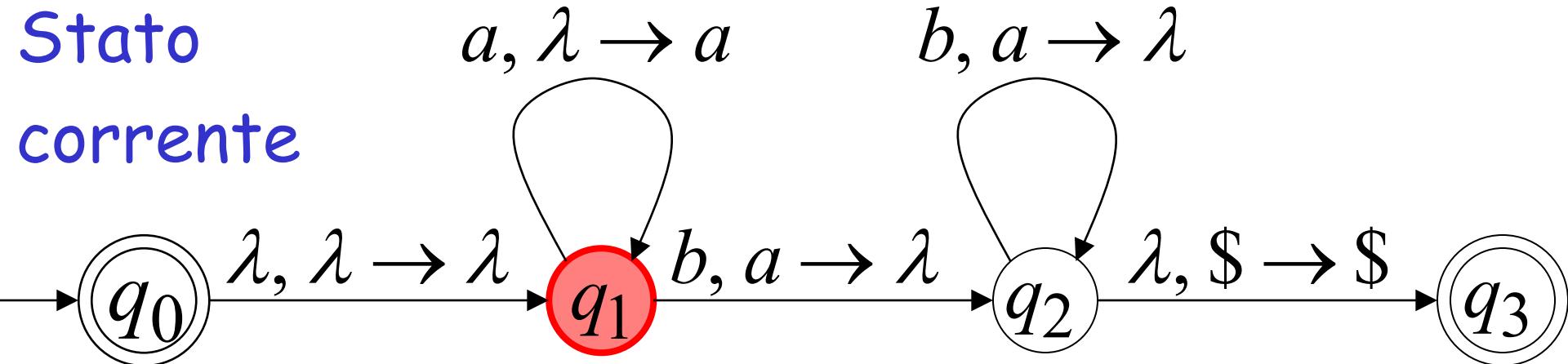
Input

a	a	b
-----	-----	-----



Stack

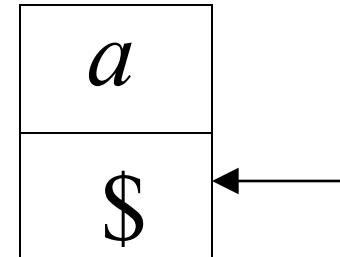
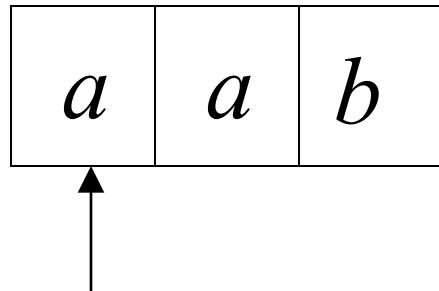
Stato
corrente



non accettazione :

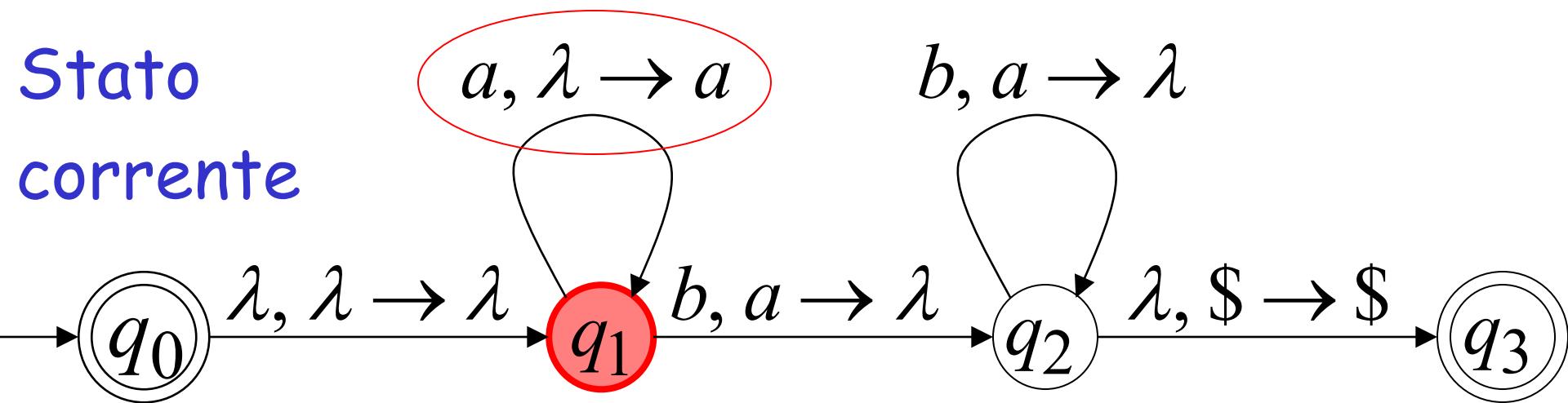
Time 2

Input



Stack

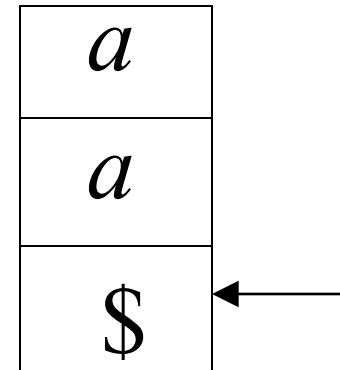
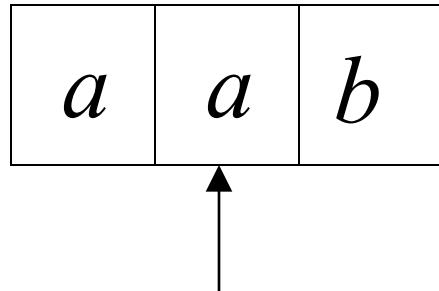
Stato
corrente



non accettazione :

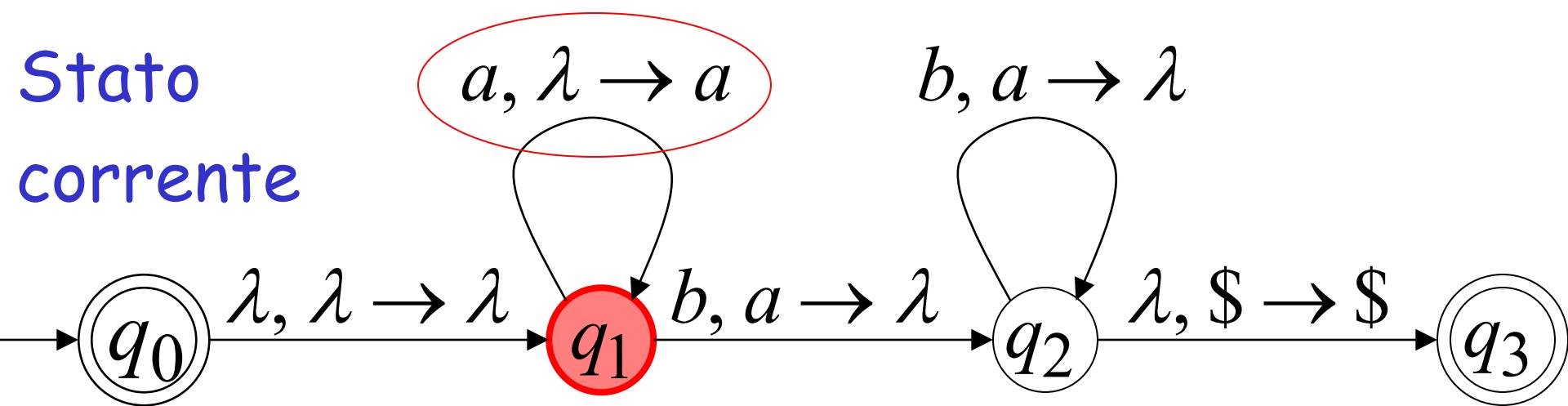
Time 3

Input



Stack

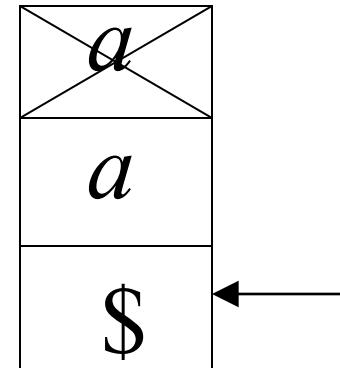
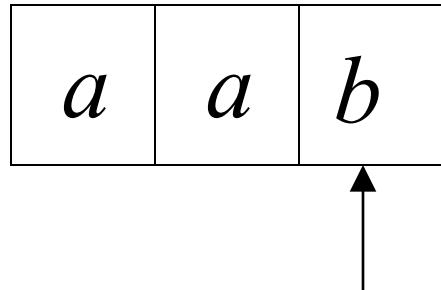
Stato
corrente



non accettazione :

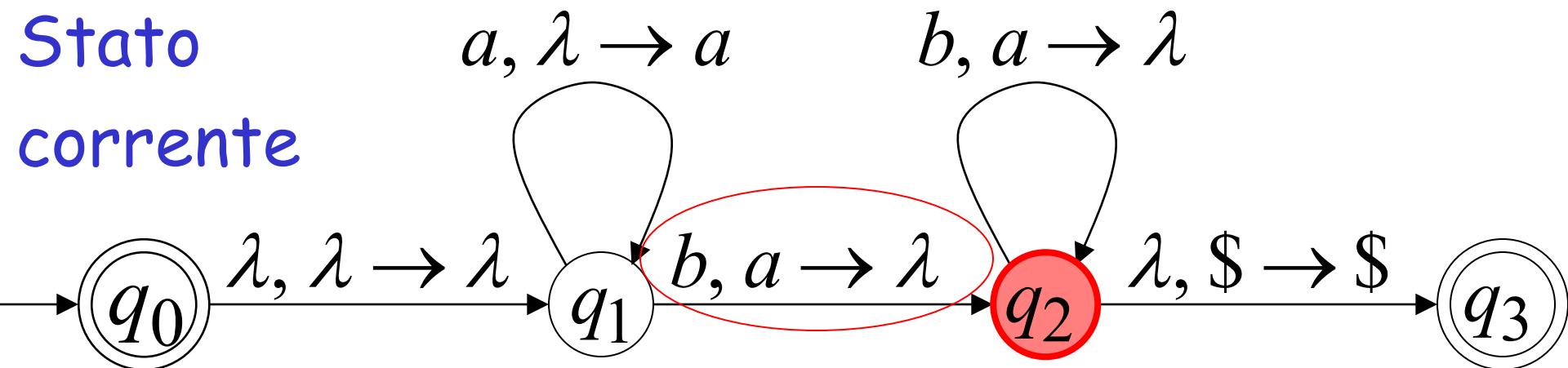
Time 4

Input



Stack

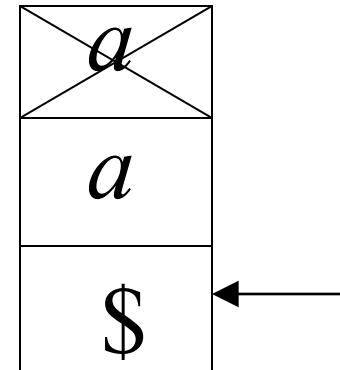
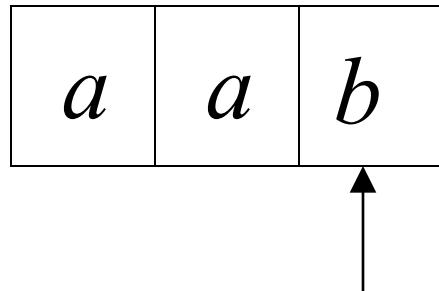
Stato
corrente



non accettazione :

Time 4

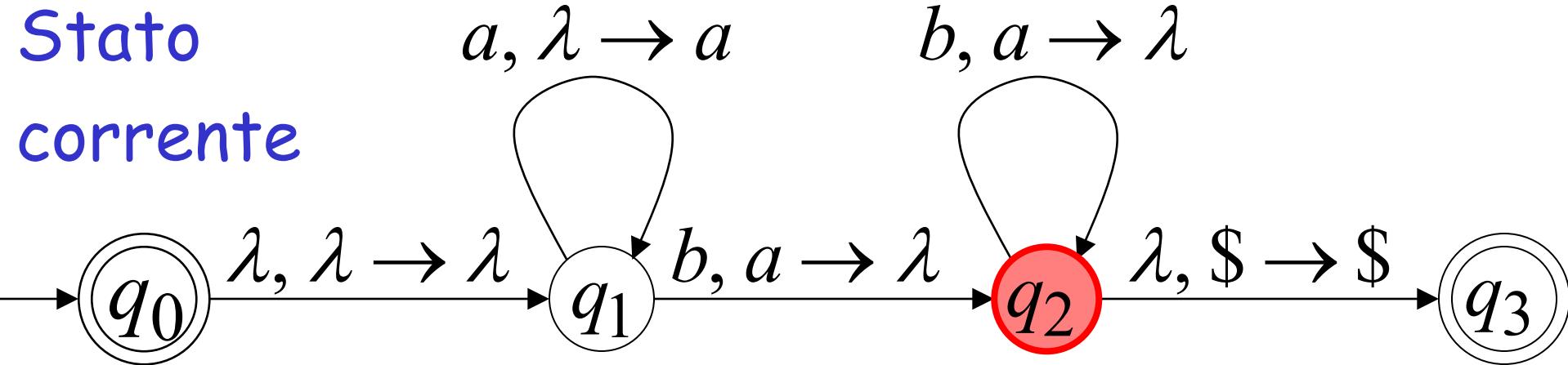
Input



Stack

reject

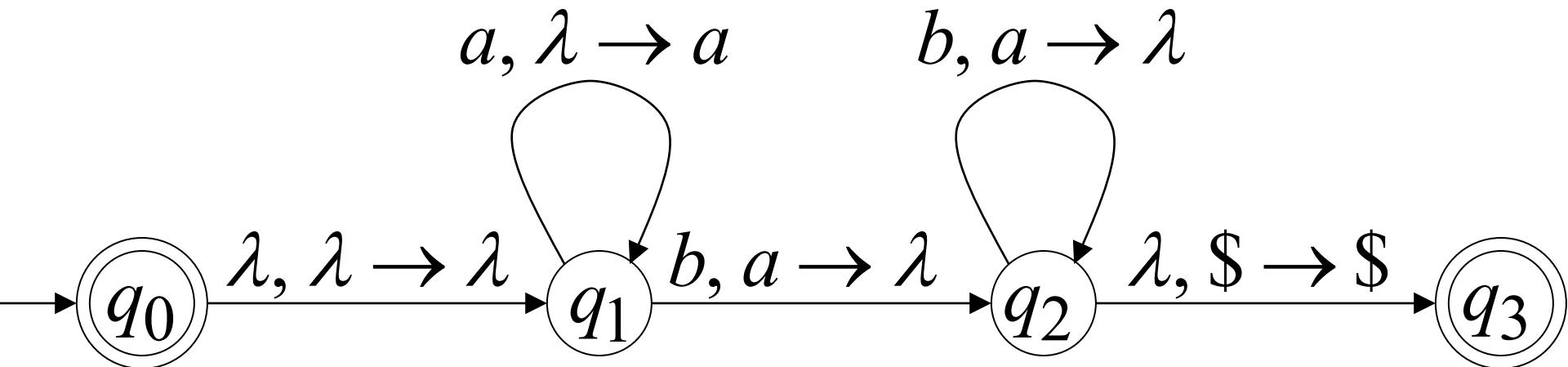
Stato
corrente



Non esiste una computazione accettante

Per aab .

La stringa aab è rigettata dal PDA.



Un altro esempio di PDA

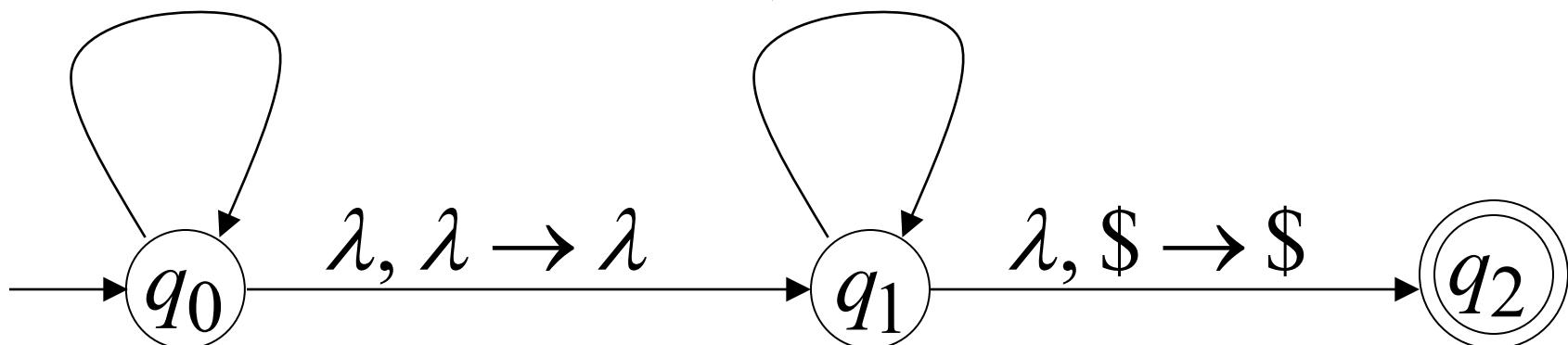
PDA M : $L(M) = \{vv^R : v \in \{a,b\}^*\}$

$$a, \lambda \rightarrow a$$

$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$



aby aa y a y aba=xyz

abcy cba

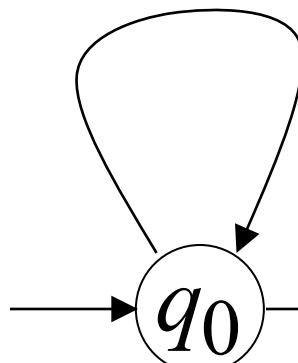
Basic Idea:

$$L(M) = \{vv^R : v \in \{a,b\}^*\}$$

1. Push v
Nello stack



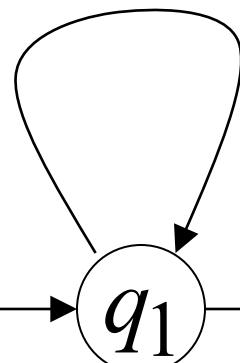
$$\begin{aligned} a, \lambda &\rightarrow a \\ b, \lambda &\rightarrow b \end{aligned}$$



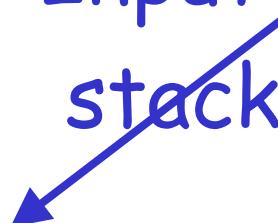
2. Congettura
metà
Dell'input

$$\lambda, \lambda \rightarrow \lambda$$

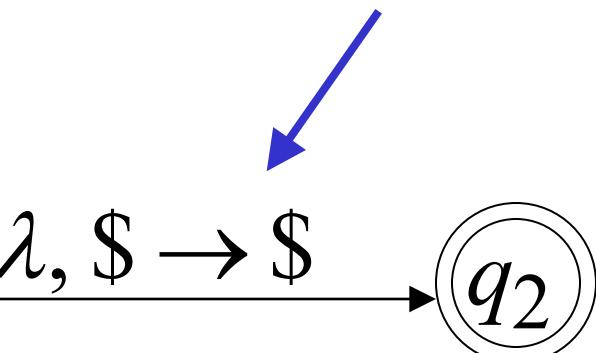
$$\begin{aligned} a, a &\rightarrow \lambda \\ b, b &\rightarrow \lambda \end{aligned}$$



3. verifica v^R in
Input con v nello
stack



4. Verifica
trovata

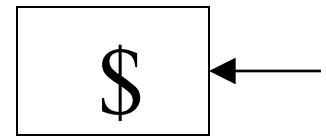


esecuzione:

Time 0

Input

a	b	b	a
-----	-----	-----	-----

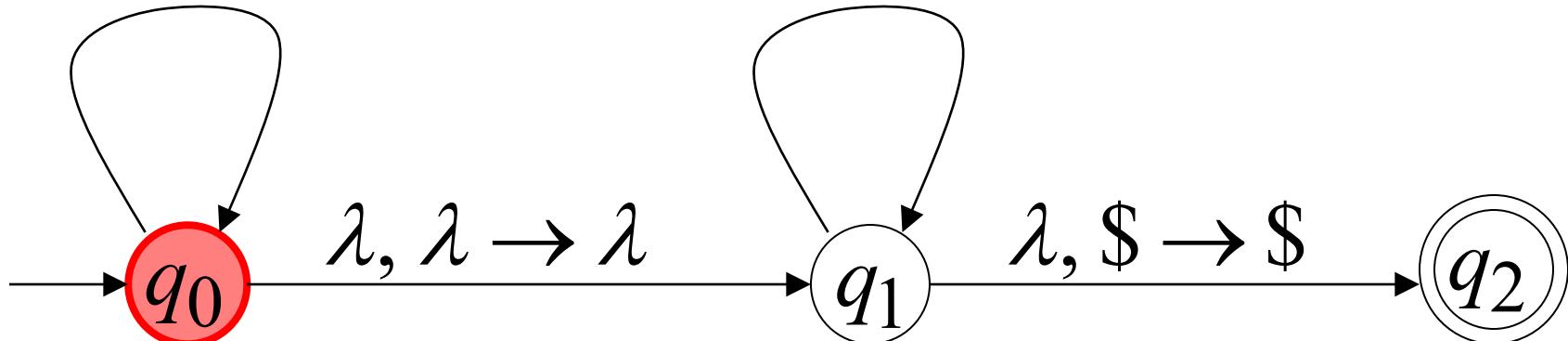


$$a, \lambda \rightarrow a$$

$$a, a \rightarrow \lambda$$

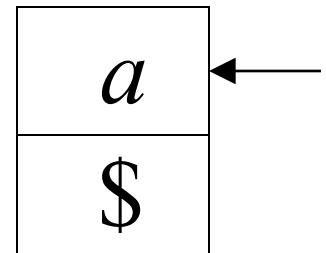
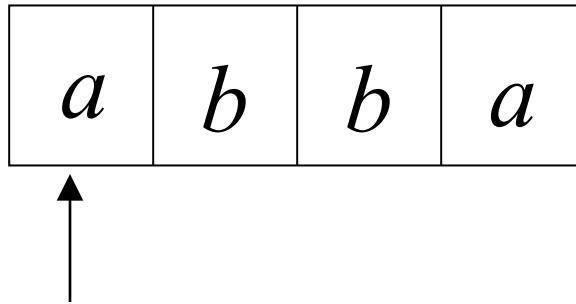
$$b, \lambda \rightarrow b$$

$$b, b \rightarrow \lambda$$



Time 1

Input



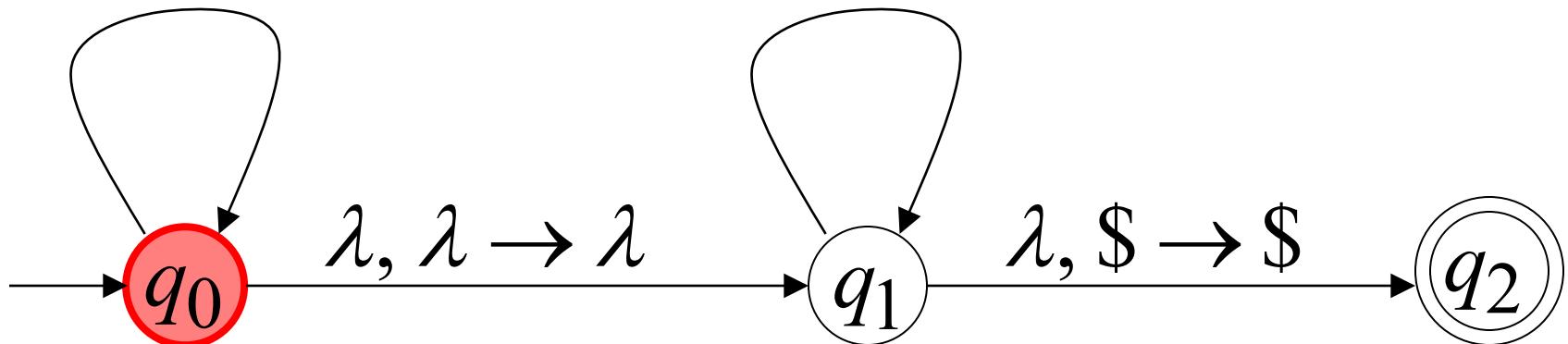
$a, \lambda \rightarrow a$

$b, \lambda \rightarrow b$

$a, a \rightarrow \lambda$

$b, b \rightarrow \lambda$

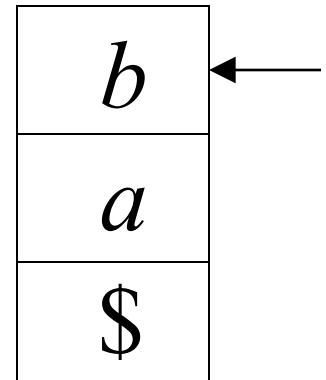
Stack



Time 2

Input

a	b	b	a
---	---	---	---



Stack

$$a, \lambda \rightarrow a$$

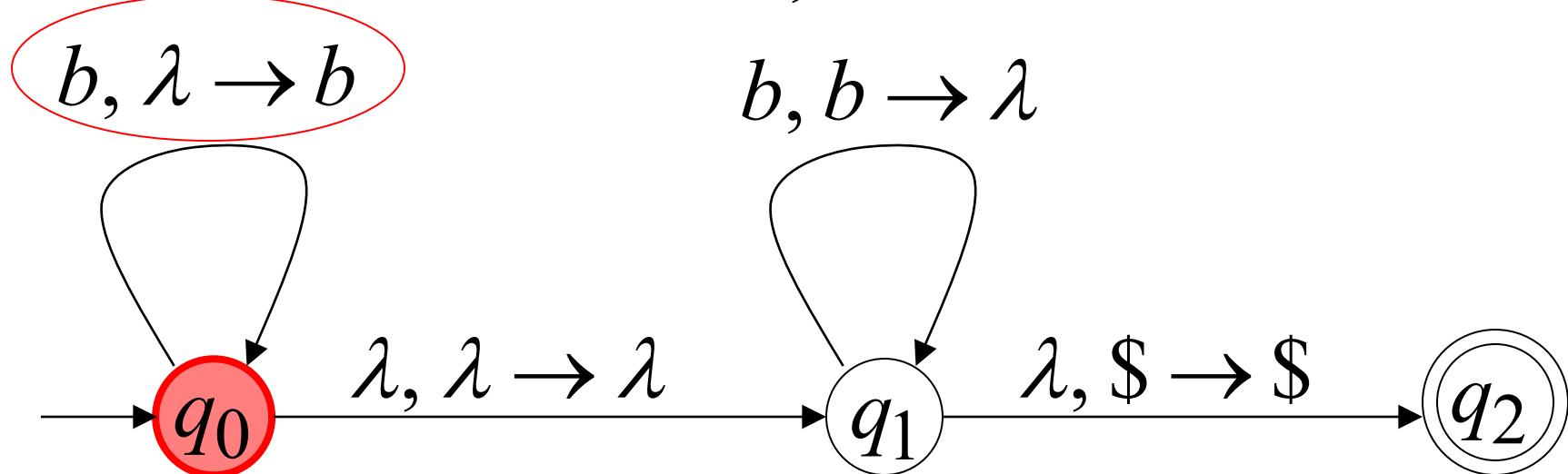
$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$

$$\lambda, \lambda \rightarrow \lambda$$

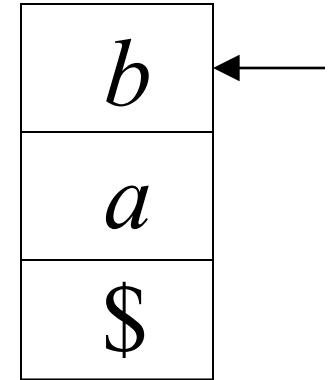
$$\lambda, \$ \rightarrow \$$$



Time 3

Input

a	b	b	a
---	---	---	---



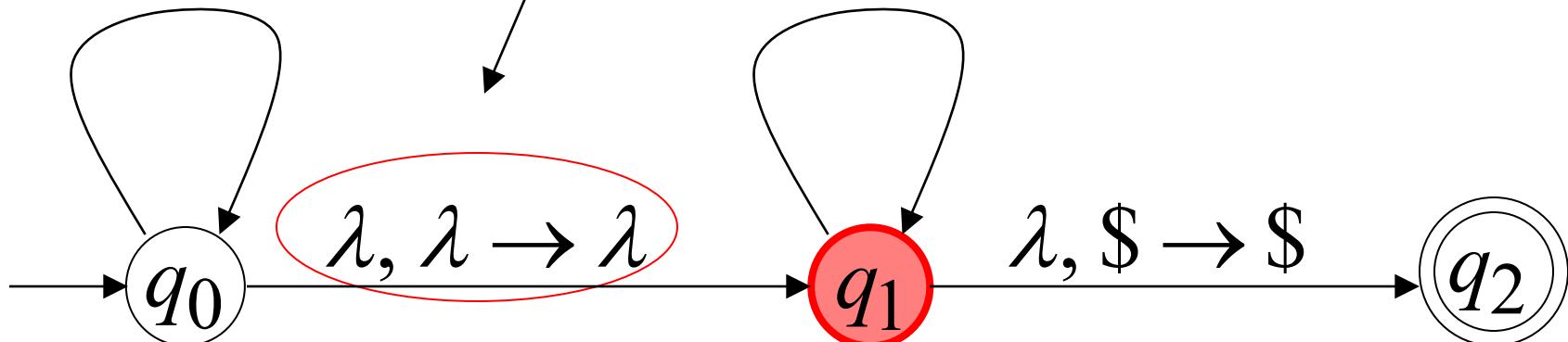
Congettura sei
Metà input

$$a, \lambda \rightarrow a$$

$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

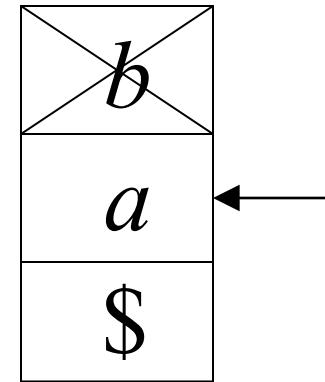
$$b, b \rightarrow \lambda$$



Time 4

Input

a	b	b	a
-----	-----	-----	-----



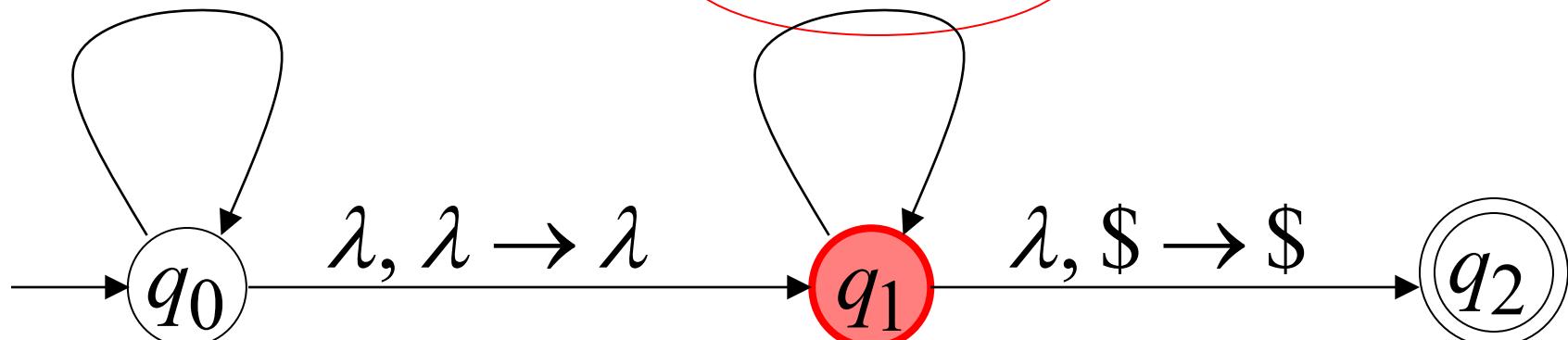
Stack

$$a, \lambda \rightarrow a$$

$$a, a \rightarrow \lambda$$

$$b, \lambda \rightarrow b$$

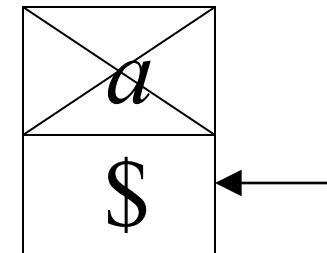
$$b, b \rightarrow \lambda$$



Time 5

Input

a	b	b	a
-----	-----	-----	-----



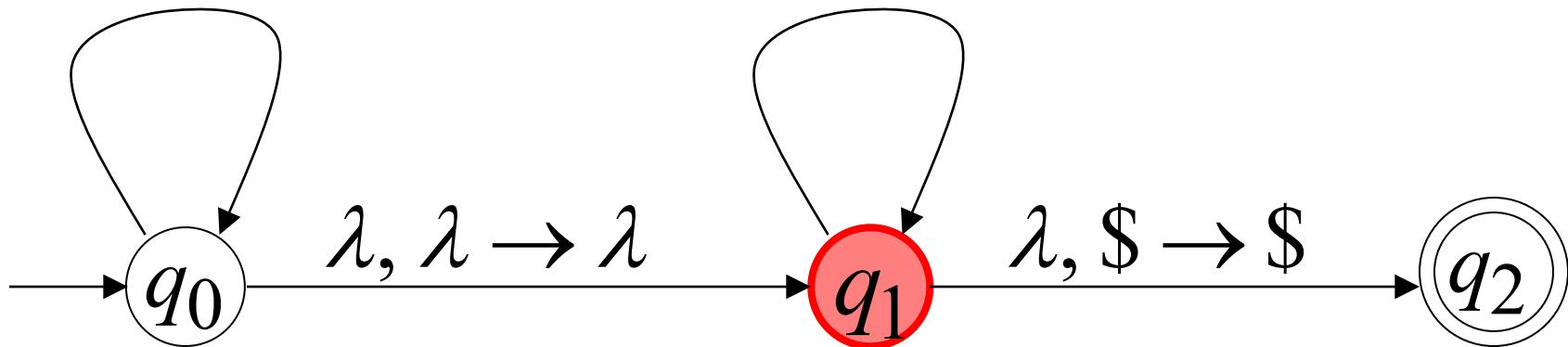
Stack

$$a, \lambda \rightarrow a$$

$$a, a \rightarrow \lambda$$

$$b, \lambda \rightarrow b$$

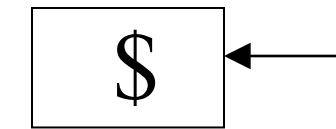
$$b, b \rightarrow \lambda$$



Time 6

Input

a	b	b	a
-----	-----	-----	-----



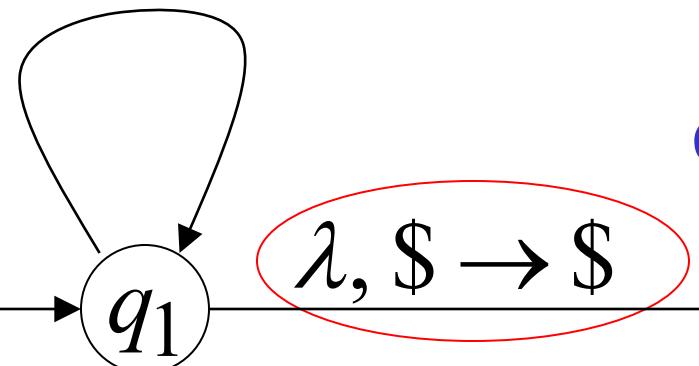
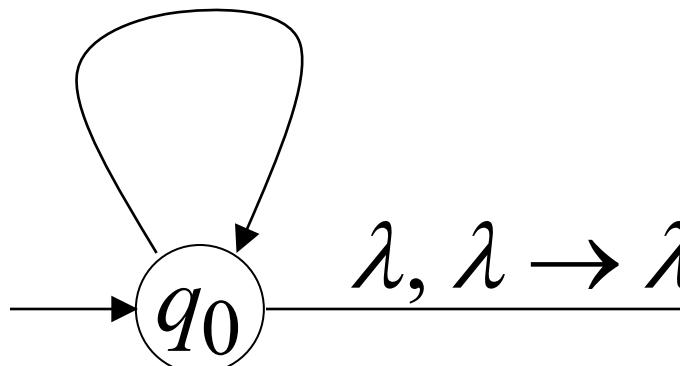
Stack

$$a, \lambda \rightarrow a$$

$$a, a \rightarrow \lambda$$

$$b, \lambda \rightarrow b$$

$$b, b \rightarrow \lambda$$



accept



esecuzione:

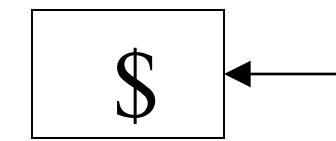
Time 0

Input

a	b	b	a
-----	-----	-----	-----



\$



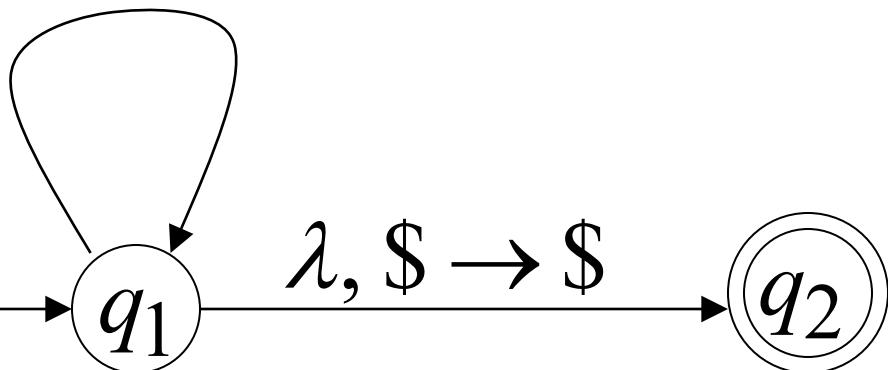
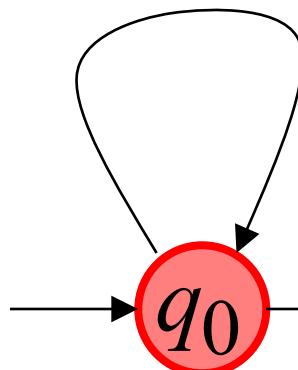
Stack

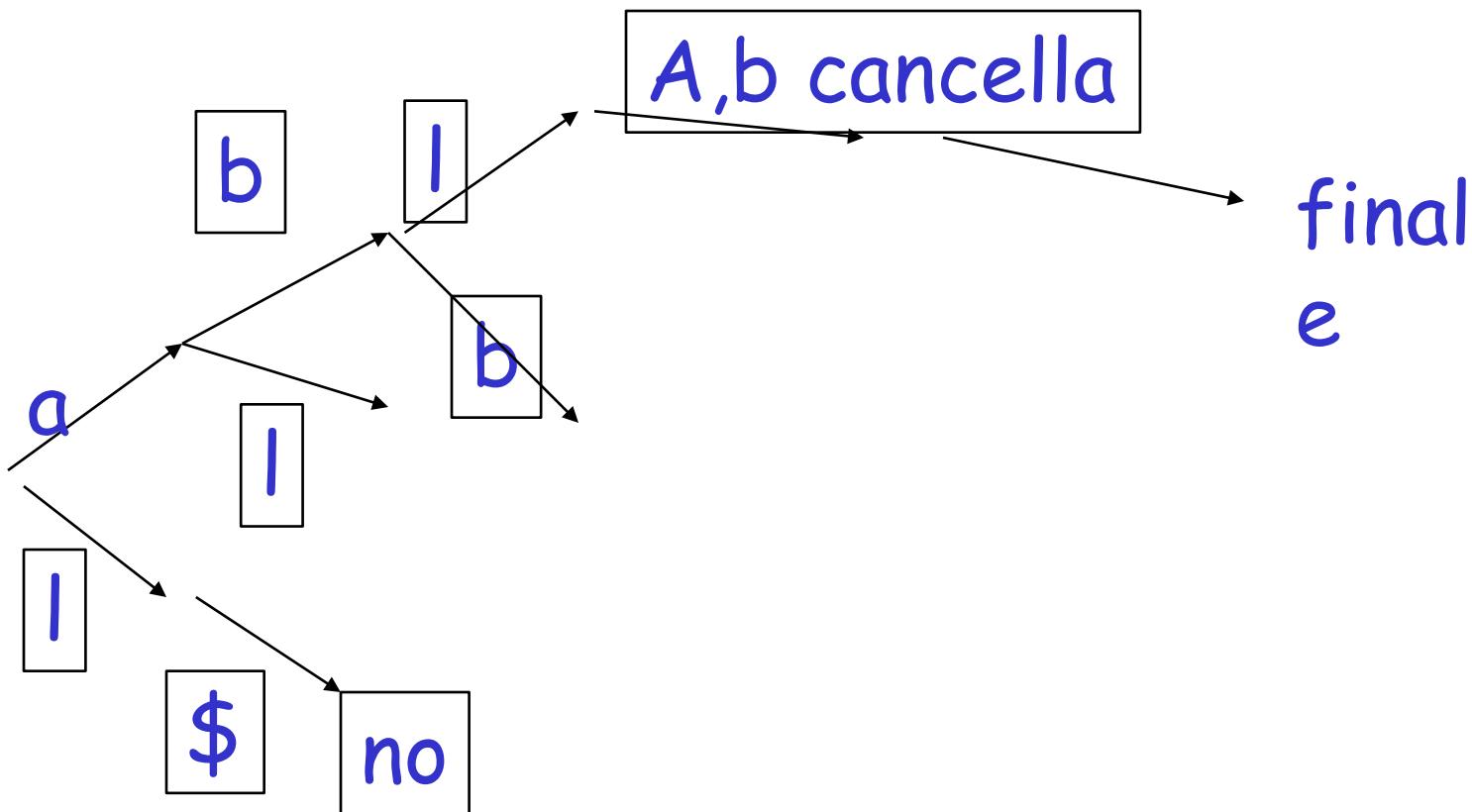
$$a, \lambda \rightarrow a$$

$$a, a \rightarrow \lambda$$

$$b, \lambda \rightarrow b$$

$$b, b \rightarrow \lambda$$



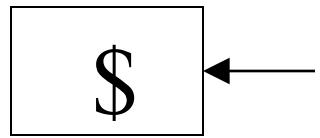


Altro esempio:

Time 0

Input

a	b	b	b
-----	-----	-----	-----

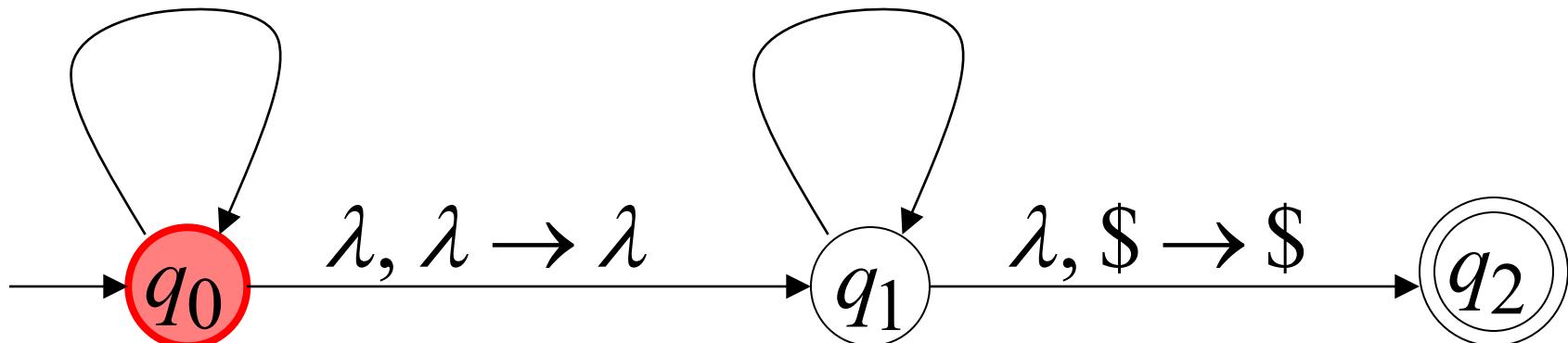


$$a, \lambda \rightarrow a$$

$$a, a \rightarrow \lambda$$

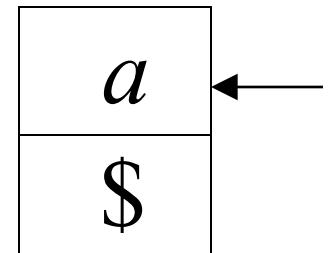
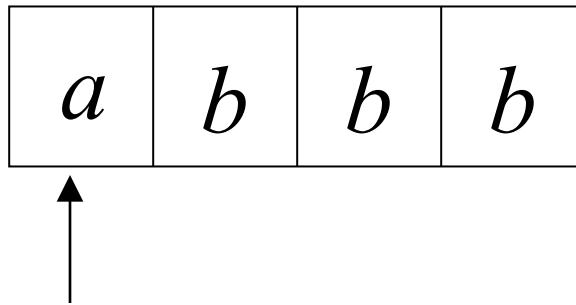
$$b, \lambda \rightarrow b$$

$$b, b \rightarrow \lambda$$



Time 1

Input



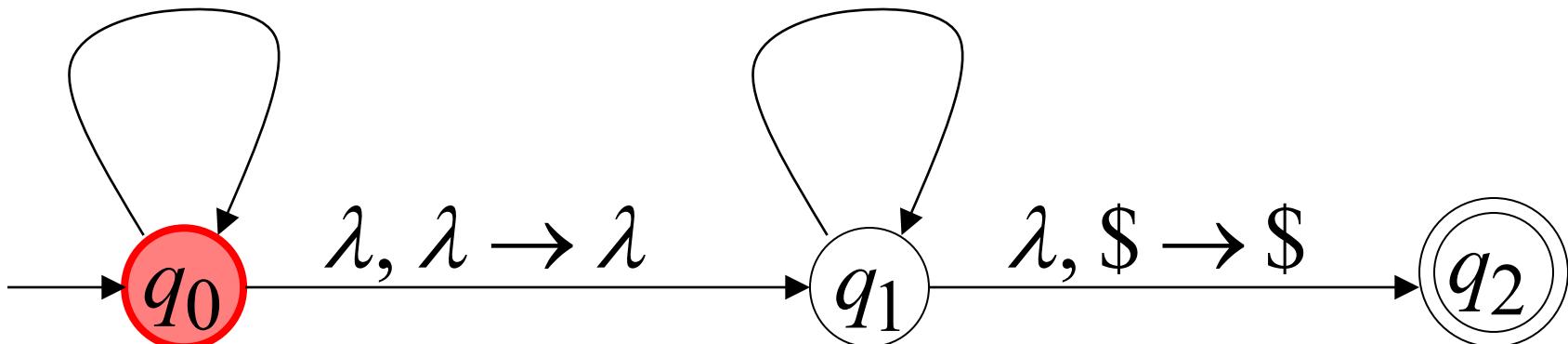
Stack

$a, \lambda \rightarrow a$

$a, a \rightarrow \lambda$

$b, \lambda \rightarrow b$

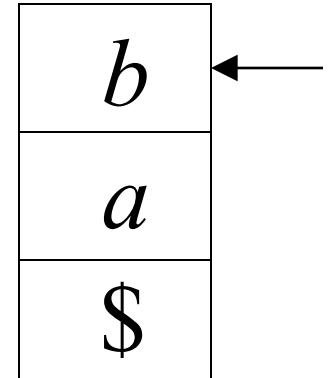
$b, b \rightarrow \lambda$



Time 2

Input

a	b	b	b
---	---	---	---



Stack

$$a, \lambda \rightarrow a$$

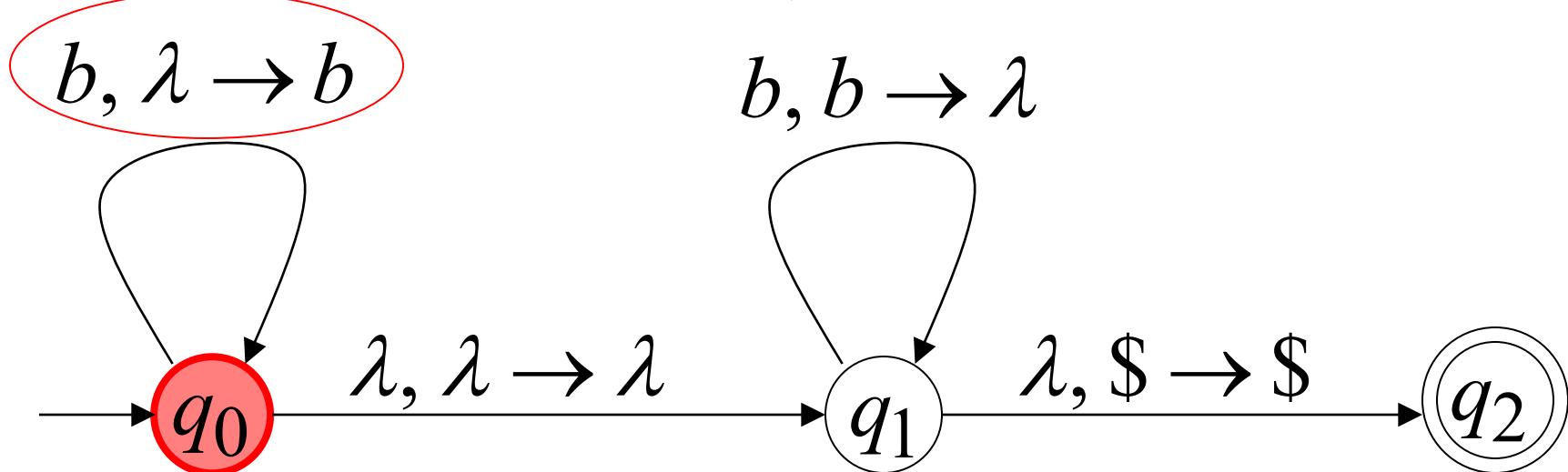
$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$

$$\lambda, \lambda \rightarrow \lambda$$

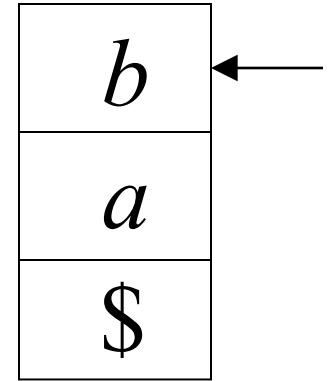
$$\lambda, \$ \rightarrow \$$$



Time 3

Input

a	b	b	b
---	---	---	---



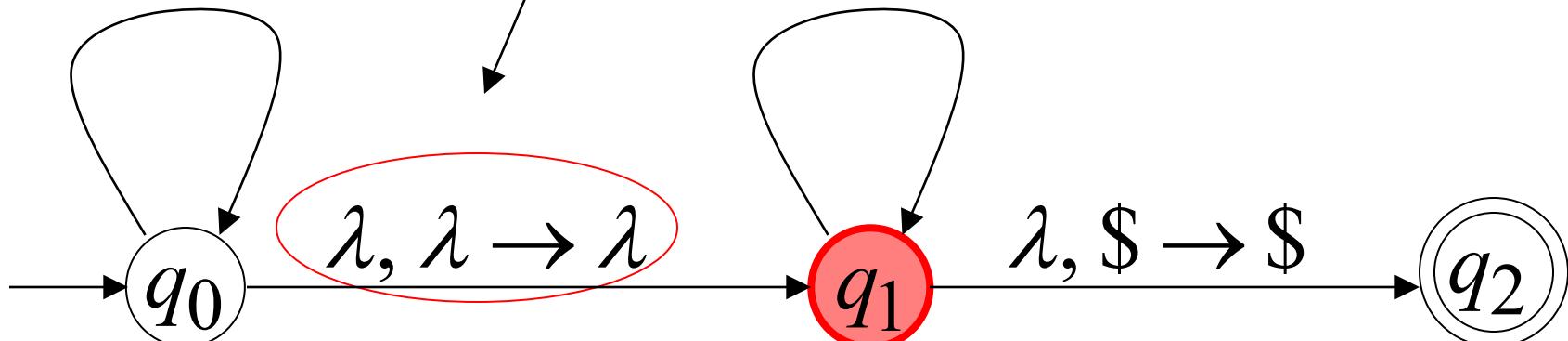
Congettura sei
A metà dell'input

$$a, \lambda \rightarrow a$$

$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

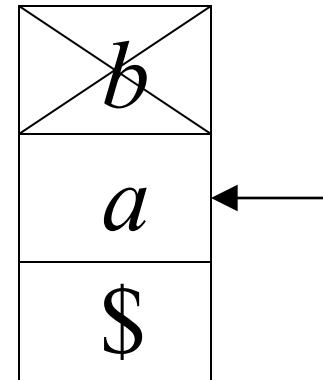
$$b, b \rightarrow \lambda$$



Time 4

Input

a	b	b	b
-----	-----	-----	-----



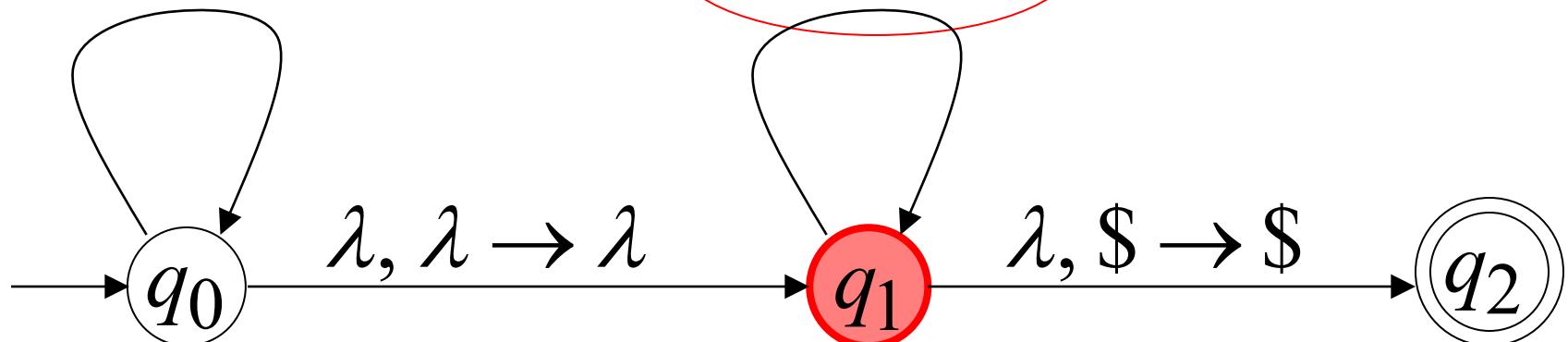
Stack

$$a, \lambda \rightarrow a$$

$$a, a \rightarrow \lambda$$

$$b, \lambda \rightarrow b$$

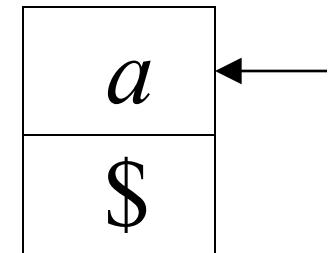
$$b, b \rightarrow \lambda$$



Input

a	b	b	b
-----	-----	-----	-----

Input non è
stato consumato

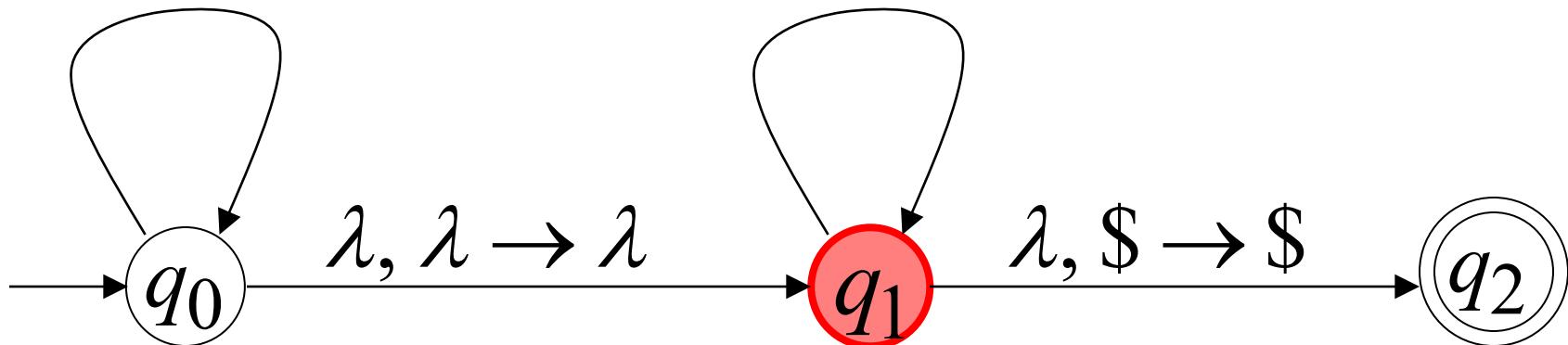


$$a, \lambda \rightarrow a$$

$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$



Time 5

Non ci sono possibili transizioni.

Stack

Un'altra computazione sulla stessa stringa:

Input

a	b	b	b
---	---	---	---

Time 0

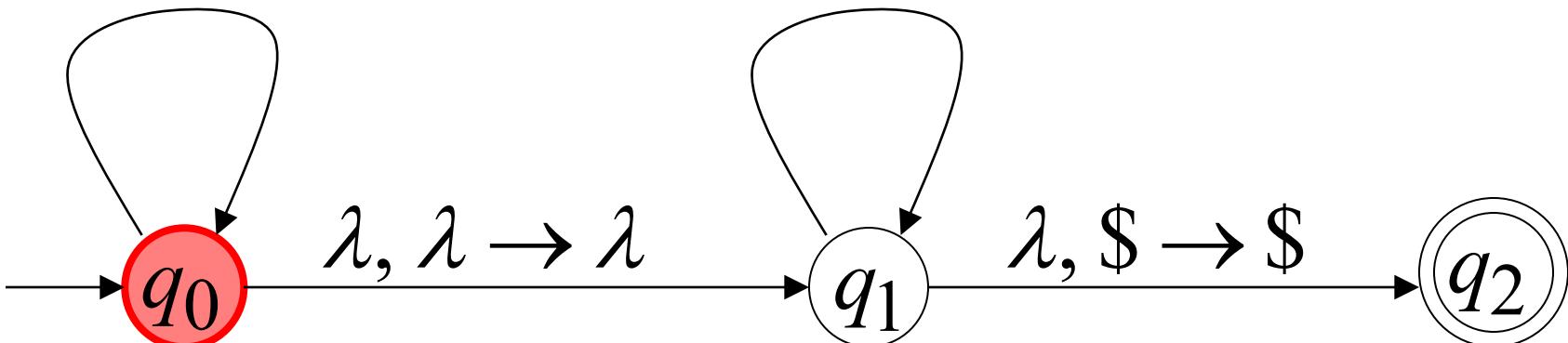


$$a, \lambda \rightarrow a$$

$$a, a \rightarrow \lambda$$

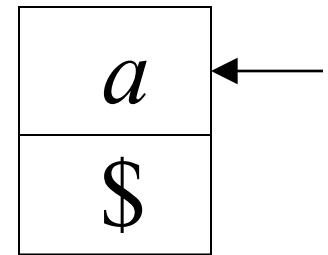
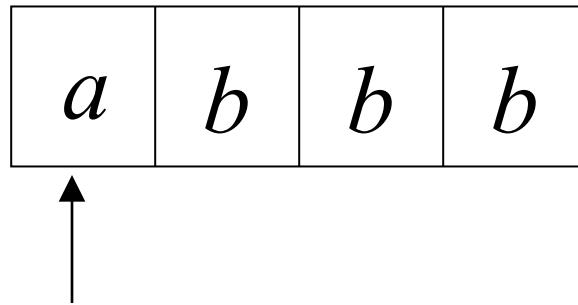
$$b, \lambda \rightarrow b$$

$$b, b \rightarrow \lambda$$



Time 1

Input



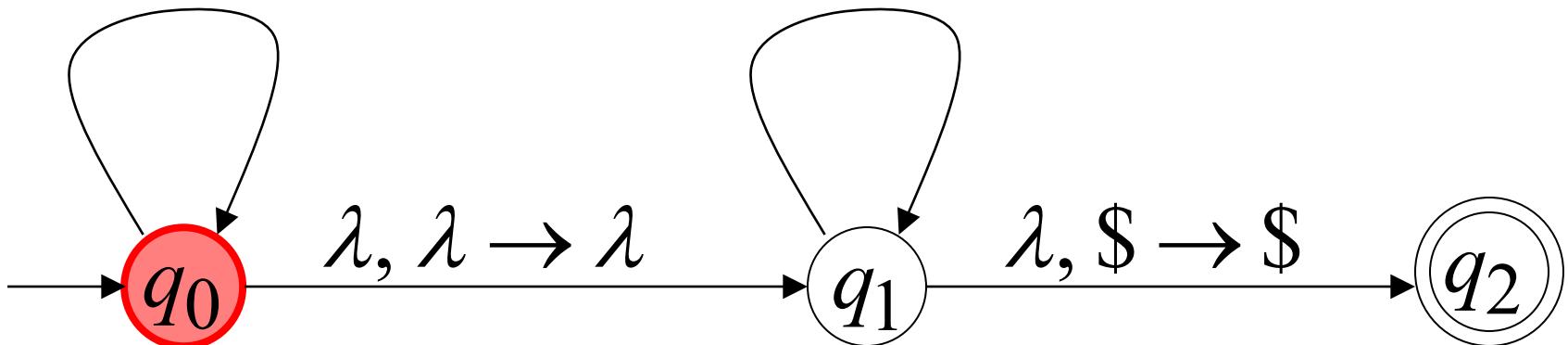
Stack

$a, \lambda \rightarrow a$

$a, a \rightarrow \lambda$

$b, \lambda \rightarrow b$

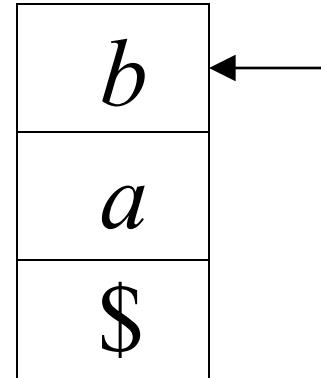
$b, b \rightarrow \lambda$



Time 2

Input

a	b	b	b
---	---	---	---



Stack

$$a, \lambda \rightarrow a$$

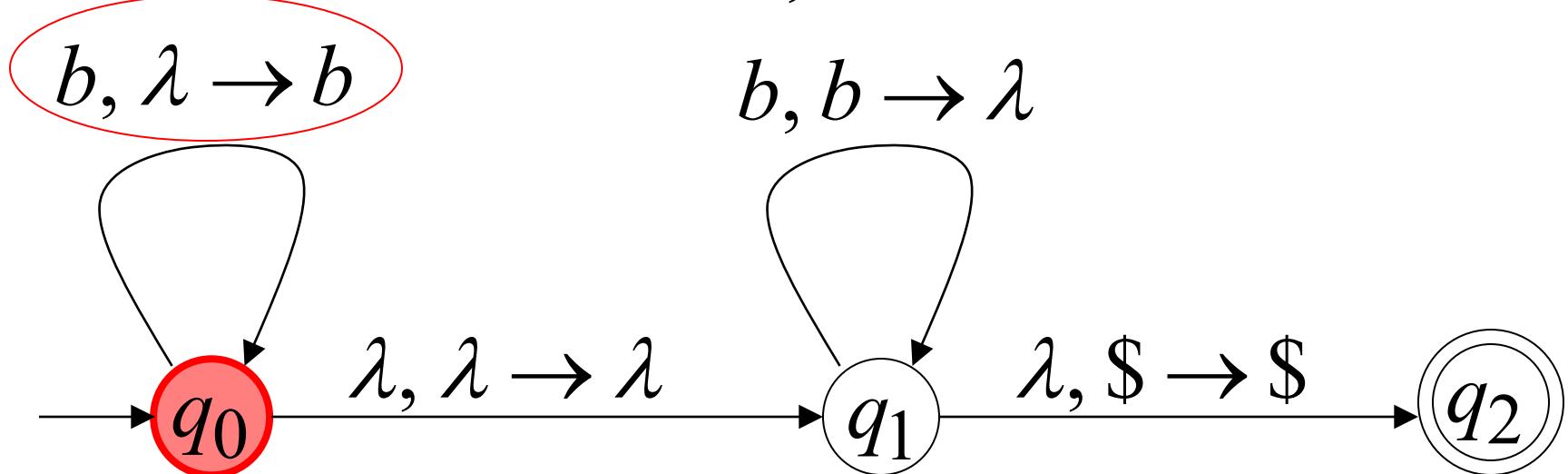
$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$

$$\lambda, \lambda \rightarrow \lambda$$

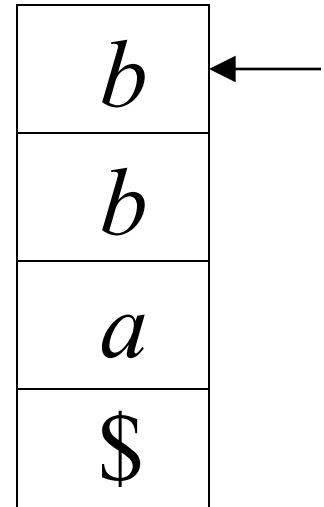
$$\lambda, \$ \rightarrow \$$$



Input

a	b	b	b
---	---	---	---

Time 3



Stack

$$a, \lambda \rightarrow a$$

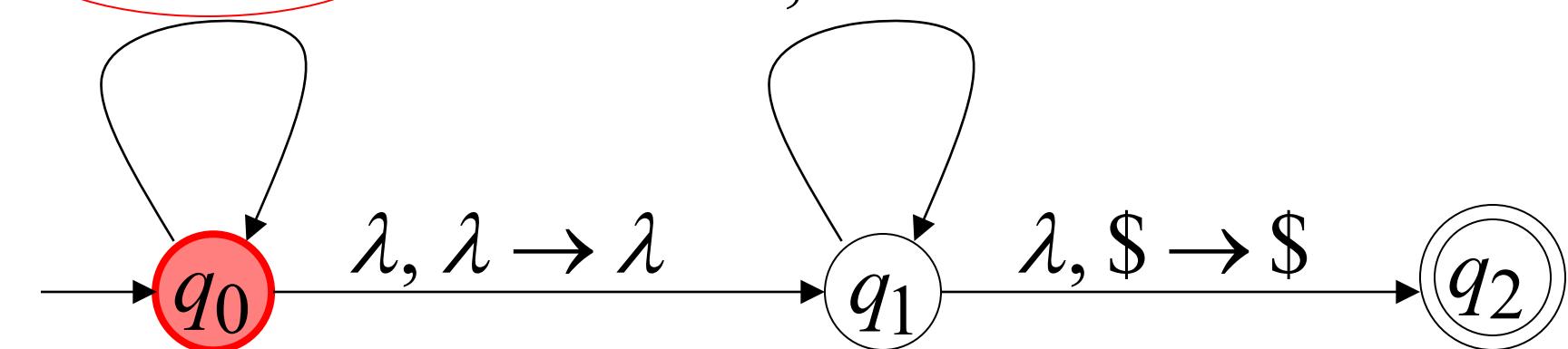
$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$

$$\lambda, \lambda \rightarrow \lambda$$

$$\lambda, \$ \rightarrow \$$$



Input

a	b	b	b
-----	-----	-----	-----

Time 4

b
b
b
a
\$

Stack

$$a, \lambda \rightarrow a$$

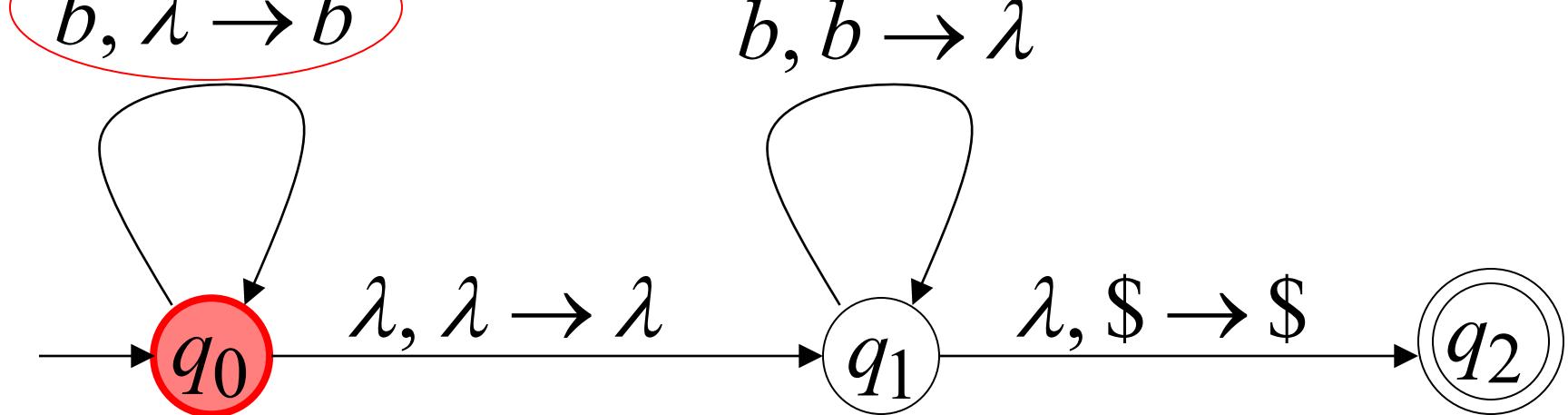
$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$

$$\lambda, \lambda \rightarrow \lambda$$

$$\lambda, \$ \rightarrow \$$$



Input

a	b	b	b
---	---	---	---

Time 5

No stato di
accettazione

b
b
b
a
\$

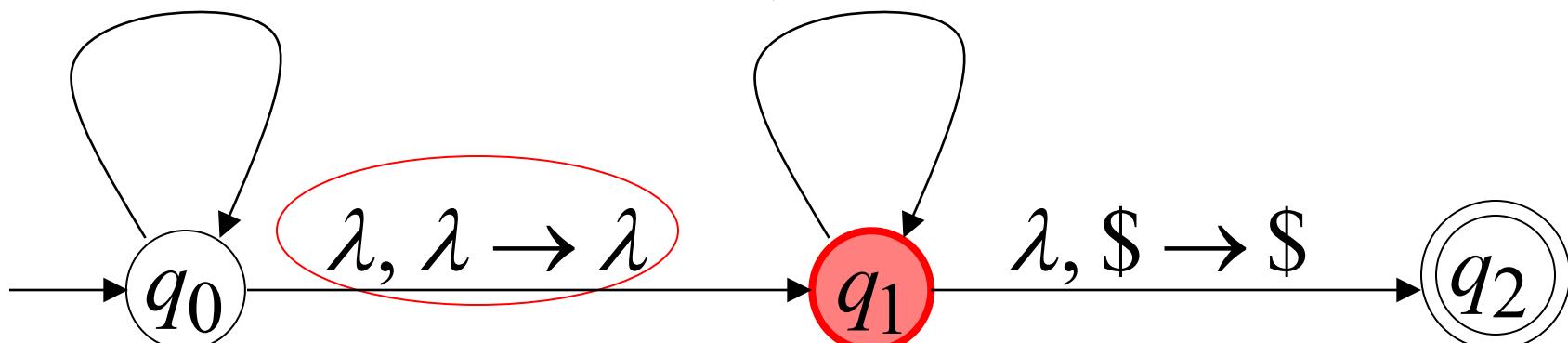
Stack

$$a, \lambda \rightarrow a$$

$$a, a \rightarrow \lambda$$

$$b, \lambda \rightarrow b$$

$$b, b \rightarrow \lambda$$



Non esiste nessuna computazione
che accetta $abbb$

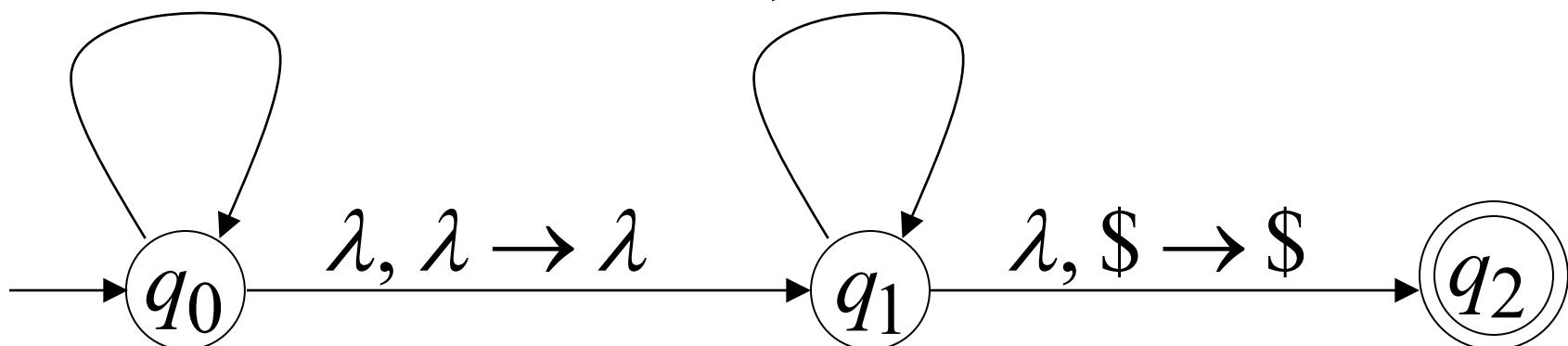
$$abbb \notin L(M)$$

$$a, \lambda \rightarrow a$$

$$b, \lambda \rightarrow b$$

$$a, a \rightarrow \lambda$$

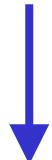
$$b, b \rightarrow \lambda$$



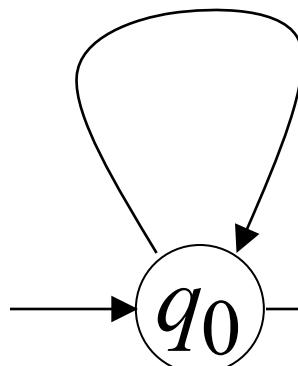
Basic Idea:

$$L(M) = \{vv^R : v \in \{a,b\}^*\}$$

1. Push v
Nello stack

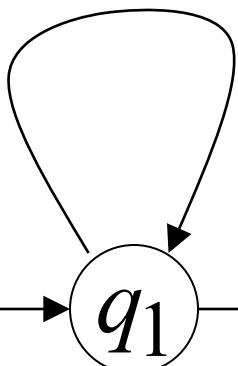


$$\begin{aligned} a, \lambda &\rightarrow a \\ b, \lambda &\rightarrow b \end{aligned}$$



2. Congettura
metà
Dell'input

$$\begin{aligned} a, a &\rightarrow \lambda \\ b, b &\rightarrow \lambda \end{aligned}$$



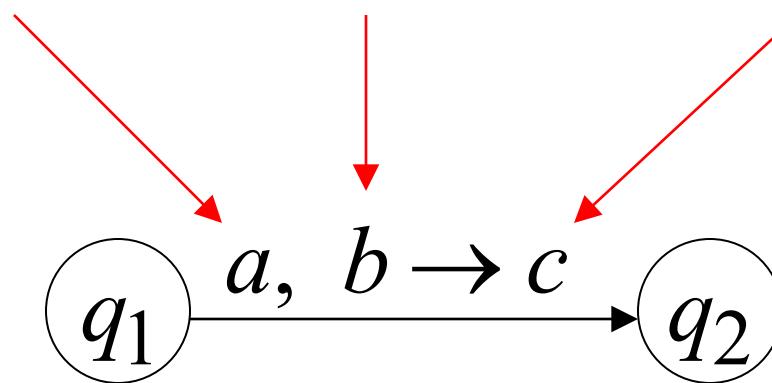
3. verifica v^R in
Input con v nello
stack

4. Verifica
trovata

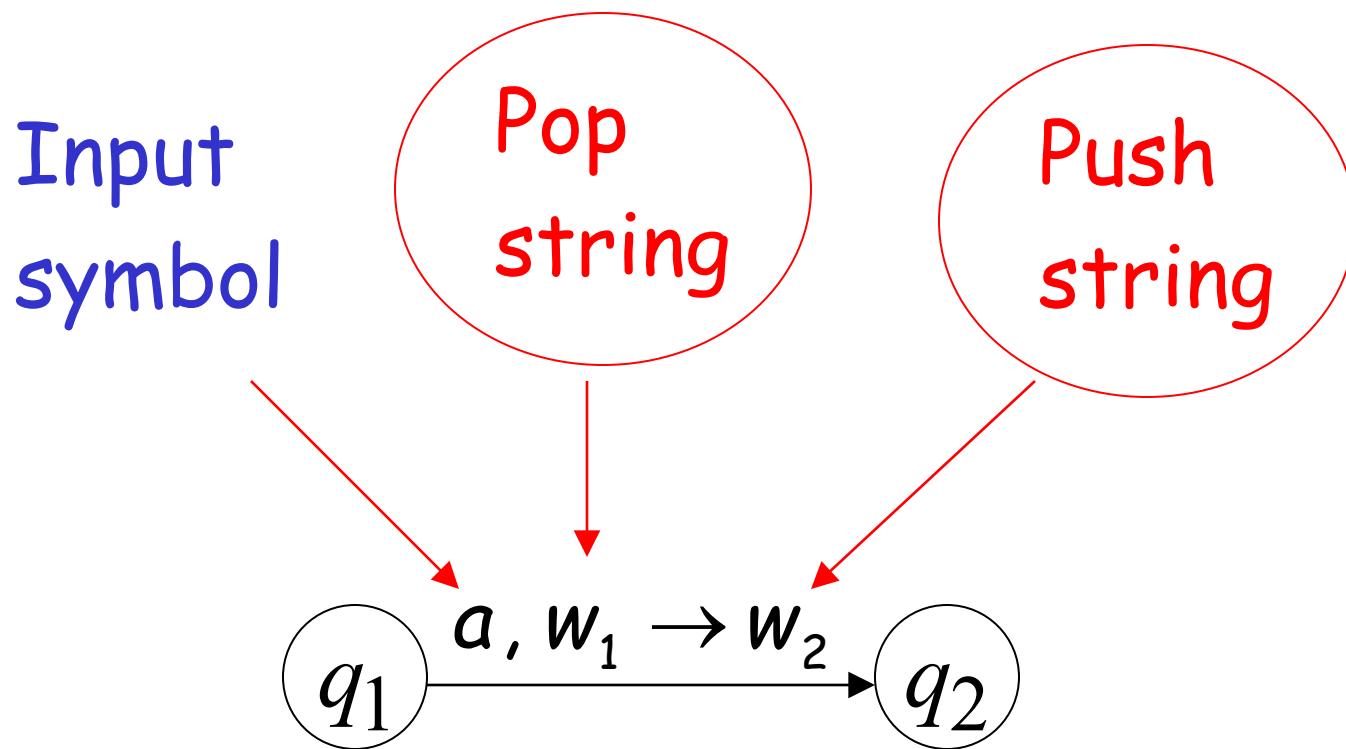
Input
simbolo

Pop
simbolo

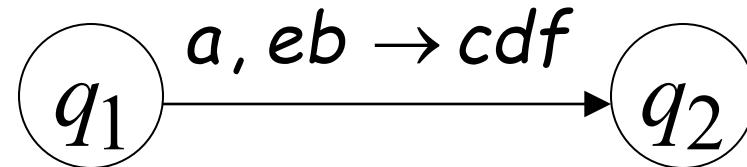
Push
simbolo



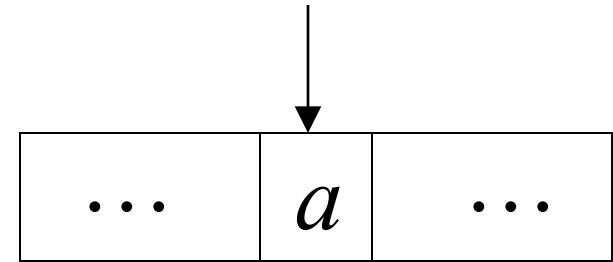
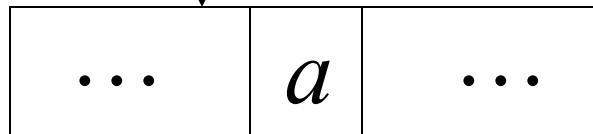
Pushing & Popping Strings



Esempio:

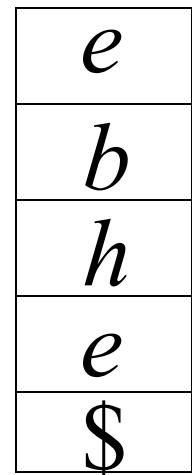


input



stack

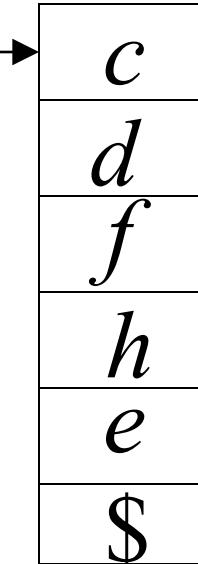
pop
string



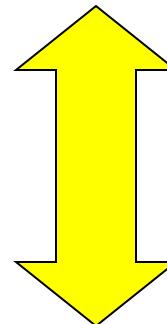
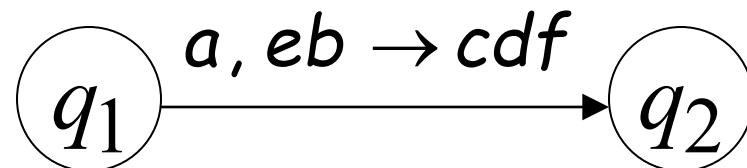
top

diventa

top

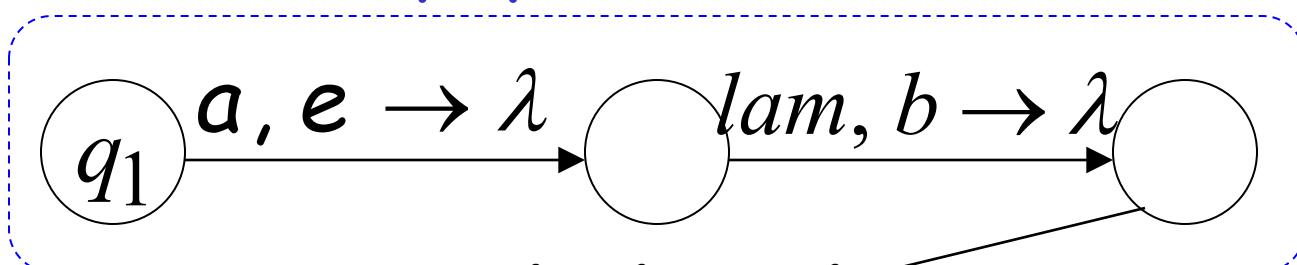


push
string

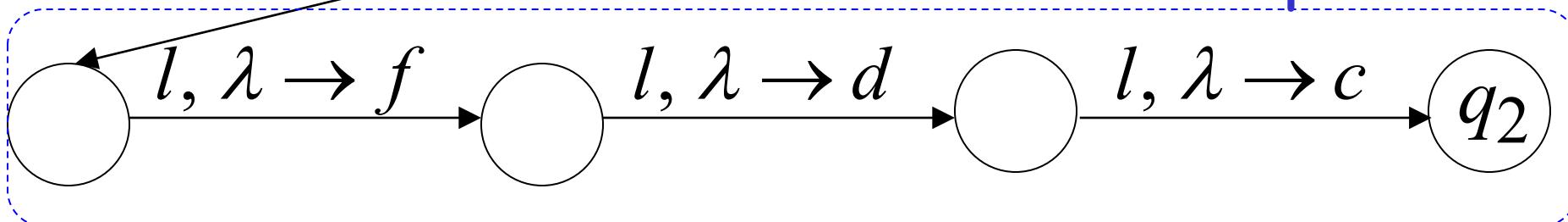


Transizioni
equivalenti

pop



push



altro PDA esempio

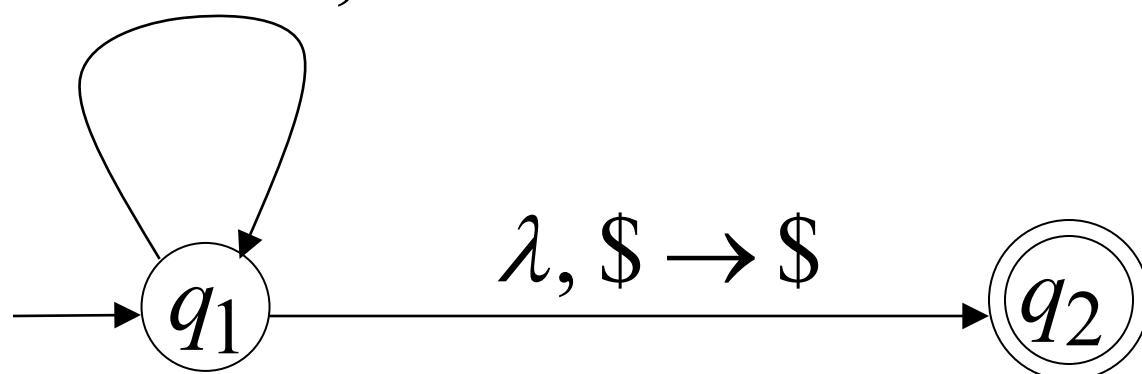
$$L(M) = \{w \in \{a, b\}^*: n_a(w) = n_b(w)\}$$

PDA M

$$a, \$ \rightarrow 0\$ \quad b, \$ \rightarrow 1\$$$

$$a, 0 \rightarrow 00 \quad b, 1 \rightarrow 11$$

$$a, 1 \rightarrow \lambda \quad b, 0 \rightarrow \lambda$$



esecuzione:

Time 0

Input

a	b	b	b	a	a
-----	-----	-----	-----	-----	-----



$a, \$ \rightarrow 0\$$

$a, 0 \rightarrow 00$

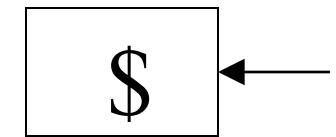
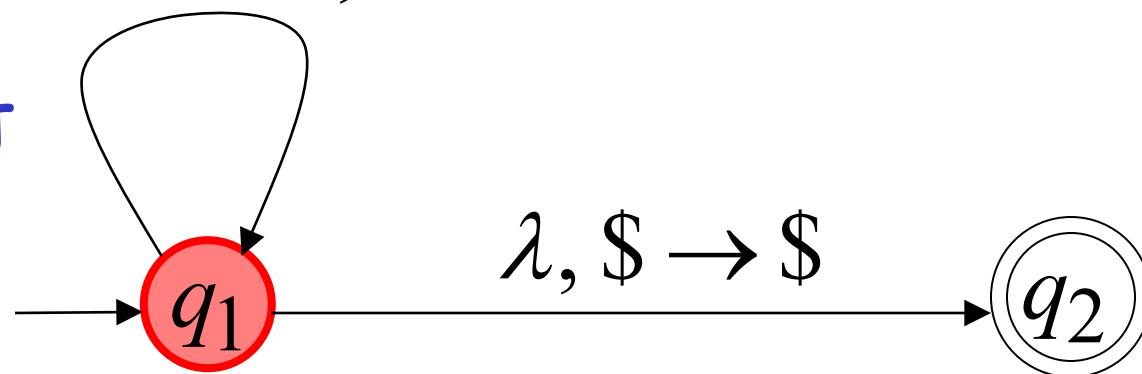
$a, 1 \rightarrow \lambda$

$b, \$ \rightarrow 1\$$

$b, 1 \rightarrow 11$

$b, 0 \rightarrow \lambda$

current
state

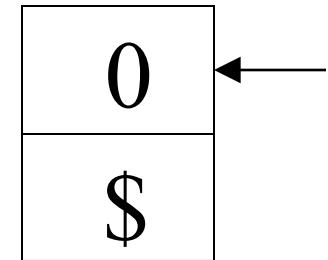


Stack

Time 1

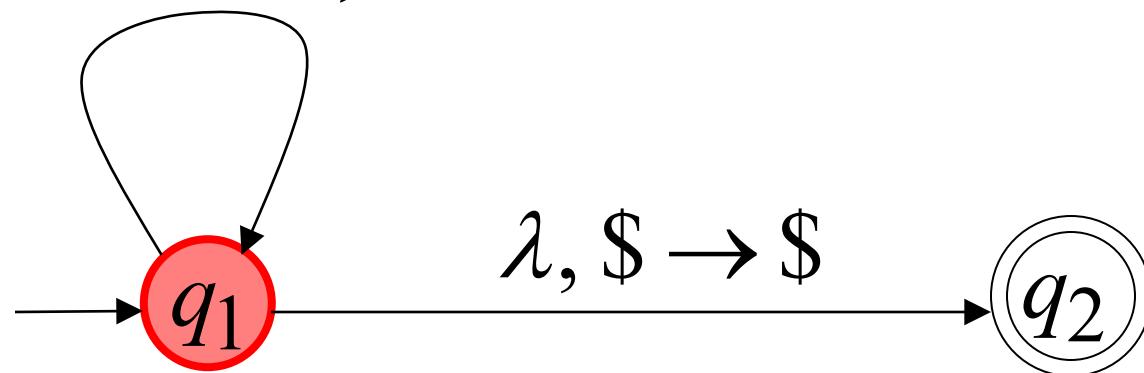
Input

a	b	b	b	a	a
-----	-----	-----	-----	-----	-----



Stack

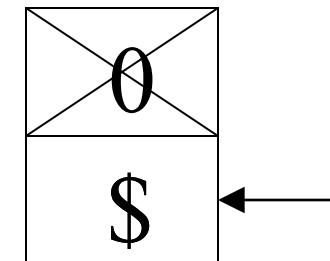
$$\begin{array}{ll} a, \$ \rightarrow 0\$ & b, \$ \rightarrow 1\$ \\ a, 0 \rightarrow 00 & b, 1 \rightarrow 11 \\ a, 1 \rightarrow \lambda & b, 0 \rightarrow \lambda \end{array}$$



Time 3

Input

a	b	b	b	a	a
-----	-----	-----	-----	-----	-----

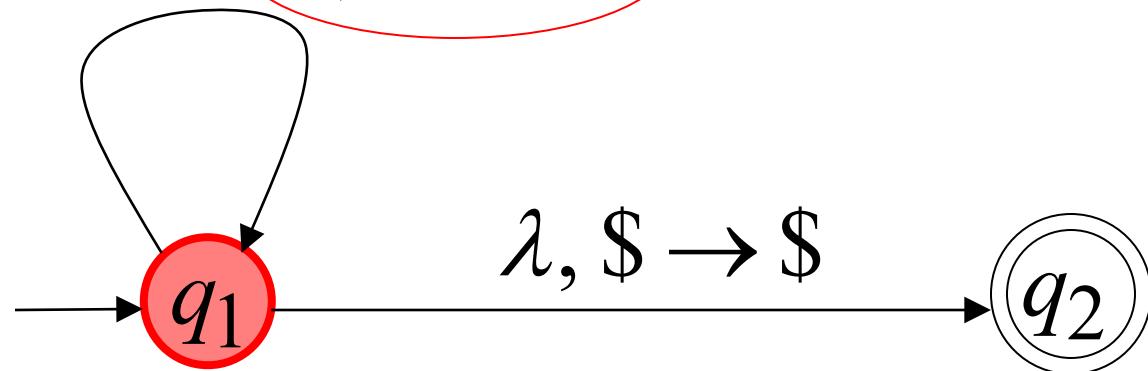


Stack

$a, \$ \rightarrow 0\$$ $b, \$ \rightarrow 1\$$

$a, 0 \rightarrow 00$ $b, 1 \rightarrow 11$

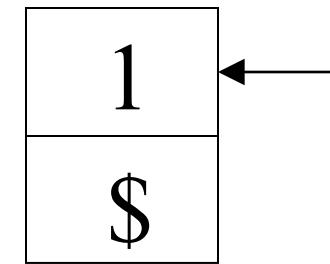
$a, 1 \rightarrow \lambda$ $b, 0 \rightarrow \lambda$



Time 4

Input

a	b	b	b	a	a
-----	-----	-----	-----	-----	-----



Stack

$a, \$ \rightarrow 0\$$

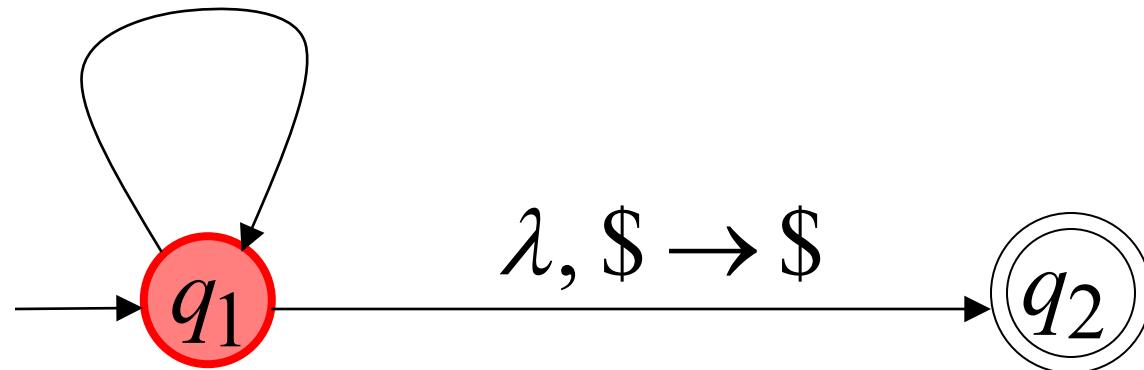
$a, 0 \rightarrow 00$

$a, 1 \rightarrow \lambda$

$b, \$ \rightarrow 1\$$

$b, 1 \rightarrow 11$

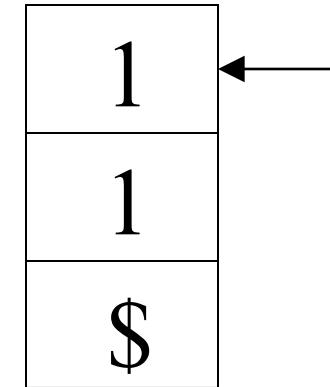
$b, 0 \rightarrow \lambda$



Time 5

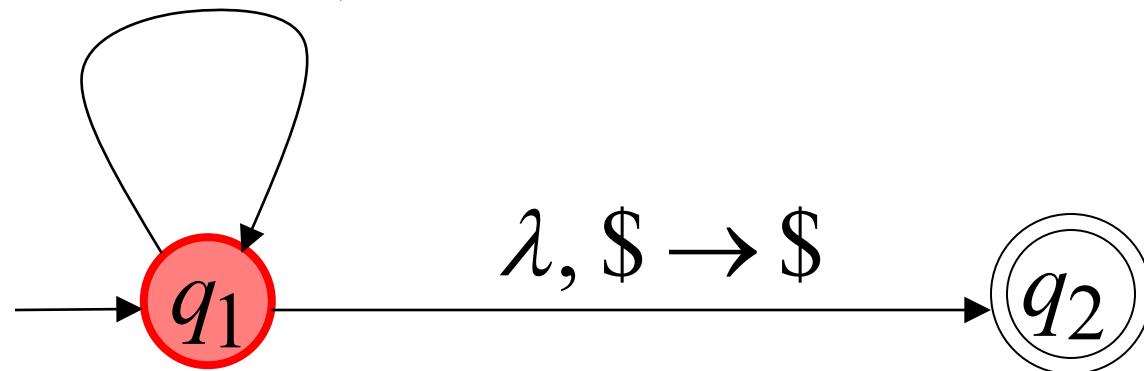
Input

a	b	b	b	a	a
-----	-----	-----	-----	-----	-----



Stack

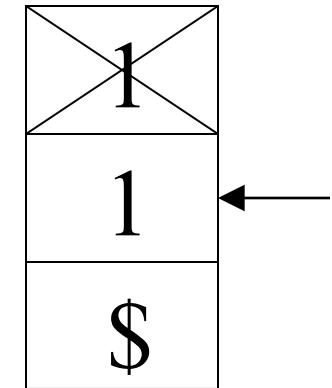
$$\begin{array}{ll} a, \$ \rightarrow 0\$ & b, \$ \rightarrow 1\$ \\ a, 0 \rightarrow 00 & b, 1 \rightarrow 11 \\ a, 1 \rightarrow \lambda & b, 0 \rightarrow \lambda \end{array}$$



Time 6

Input

a	b	b	b	a	a
-----	-----	-----	-----	-----	-----

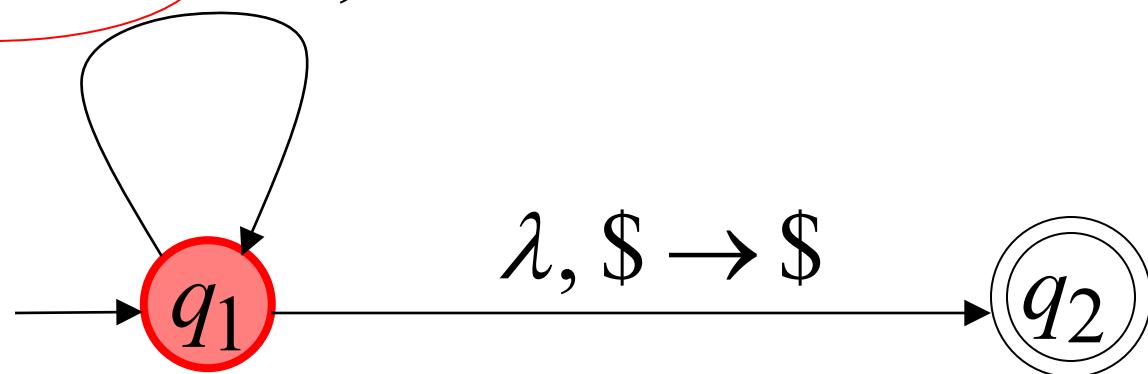


Stack

$$a, \$ \rightarrow 0\$ \quad b, \$ \rightarrow 1\$$$

$$a, 0 \rightarrow 00 \quad b, 1 \rightarrow 11$$

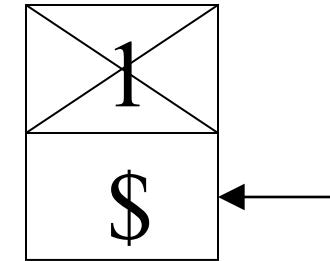
$$a, 1 \rightarrow \lambda \quad b, 0 \rightarrow \lambda$$



Time 7

Input

a	b	b	b	a	a
-----	-----	-----	-----	-----	-----

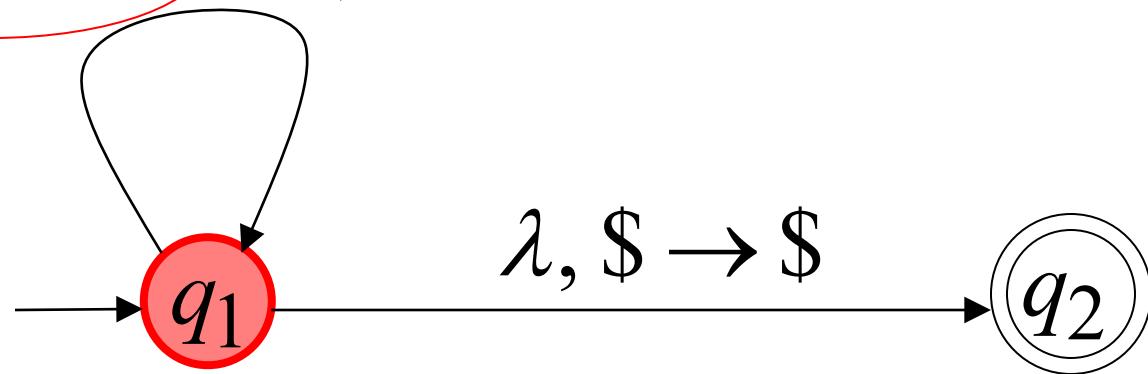


Stack

$$a, \$ \rightarrow 0\$ \quad b, \$ \rightarrow 1\$$$

$$a, 0 \rightarrow 00 \quad b, 1 \rightarrow 11$$

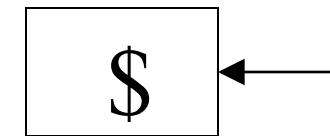
$$a, 1 \rightarrow \lambda \quad b, 0 \rightarrow \lambda$$



Time 8

Input

a	b	b	b	a	a
-----	-----	-----	-----	-----	-----

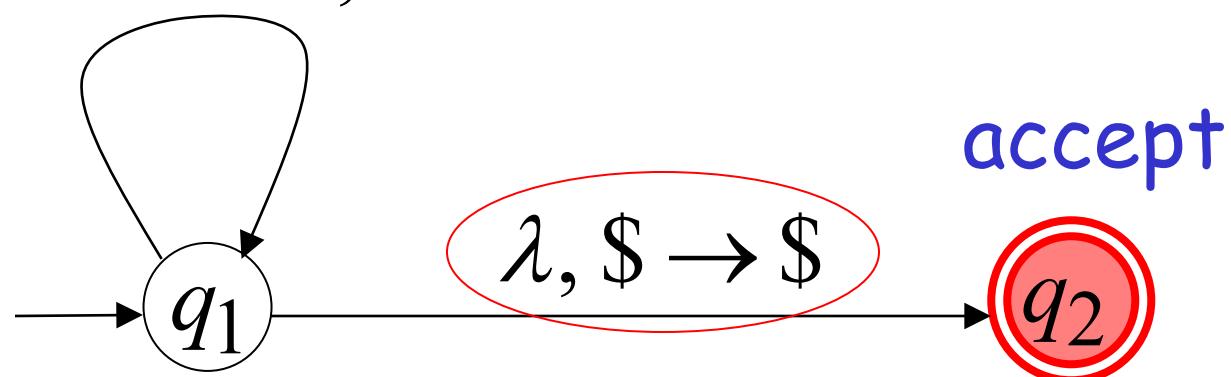


$a, \$ \rightarrow 0\$$ $b, \$ \rightarrow 1\$$

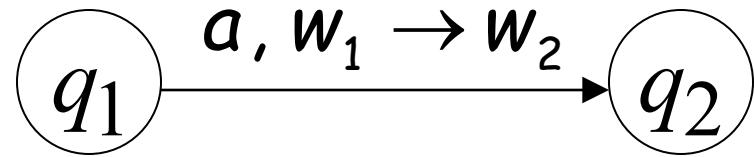
$a, 0 \rightarrow 00$ $b, 1 \rightarrow 11$

$a, 1 \rightarrow \lambda$ $b, 0 \rightarrow \lambda$

Stack

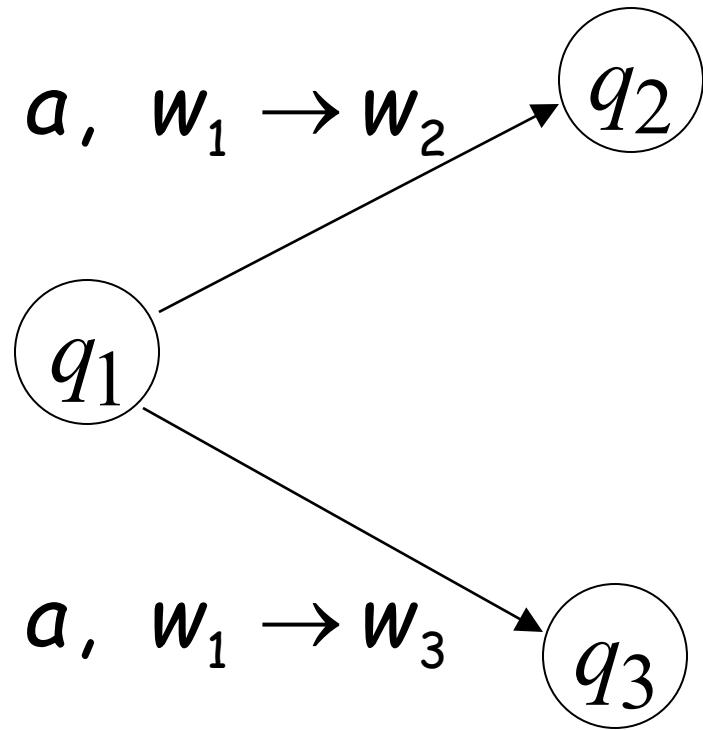


Formalismo per I PDA



Funzione di transizione:

$$\delta(q_1, a, w_1) = \{(q_2, w_2)\}$$

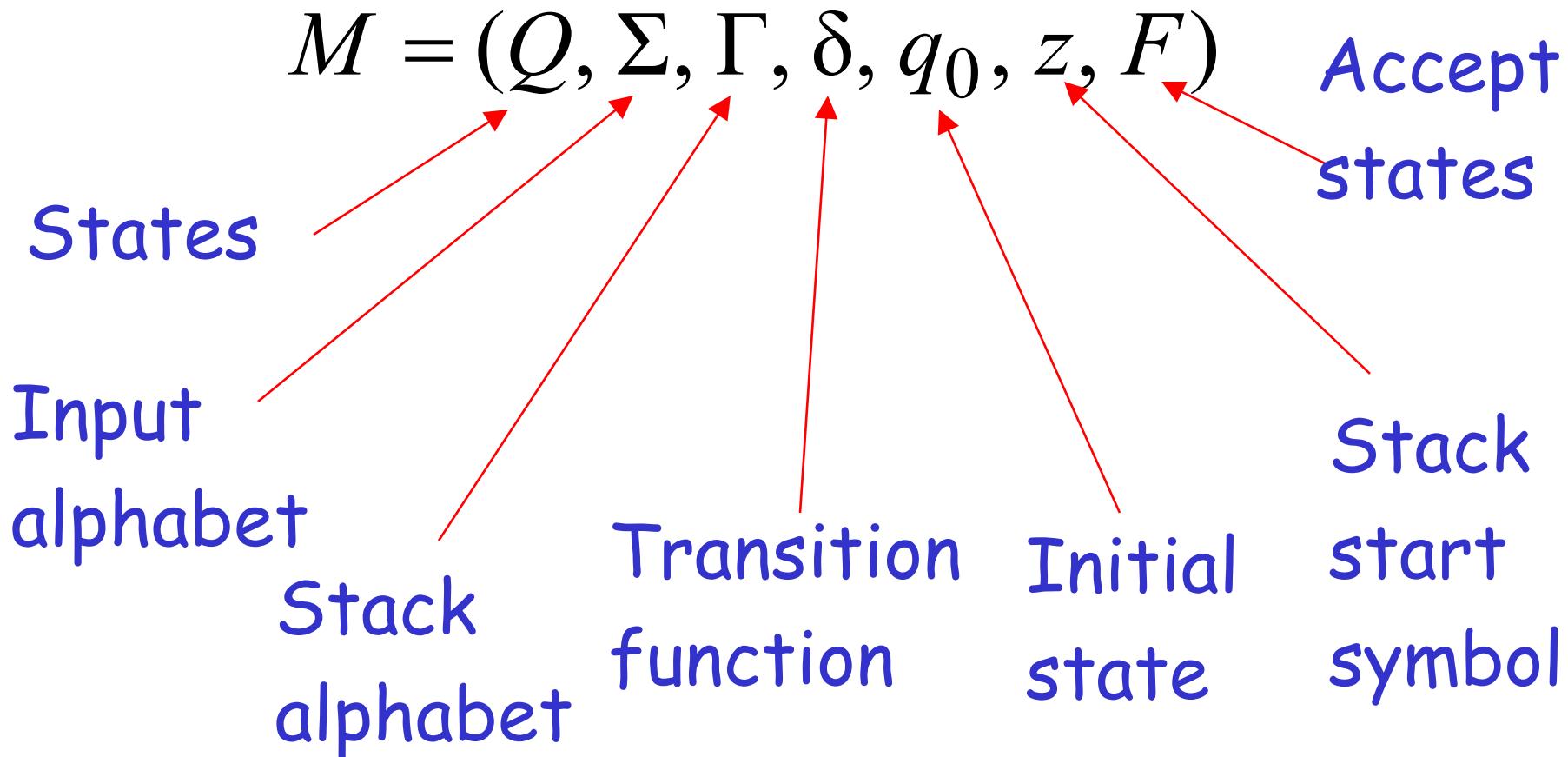


Funzione di transizione :

$$\delta(q_1, a, w_1) = \{(q_2, w_2), (q_3, w_3)\}$$

Formal Definition

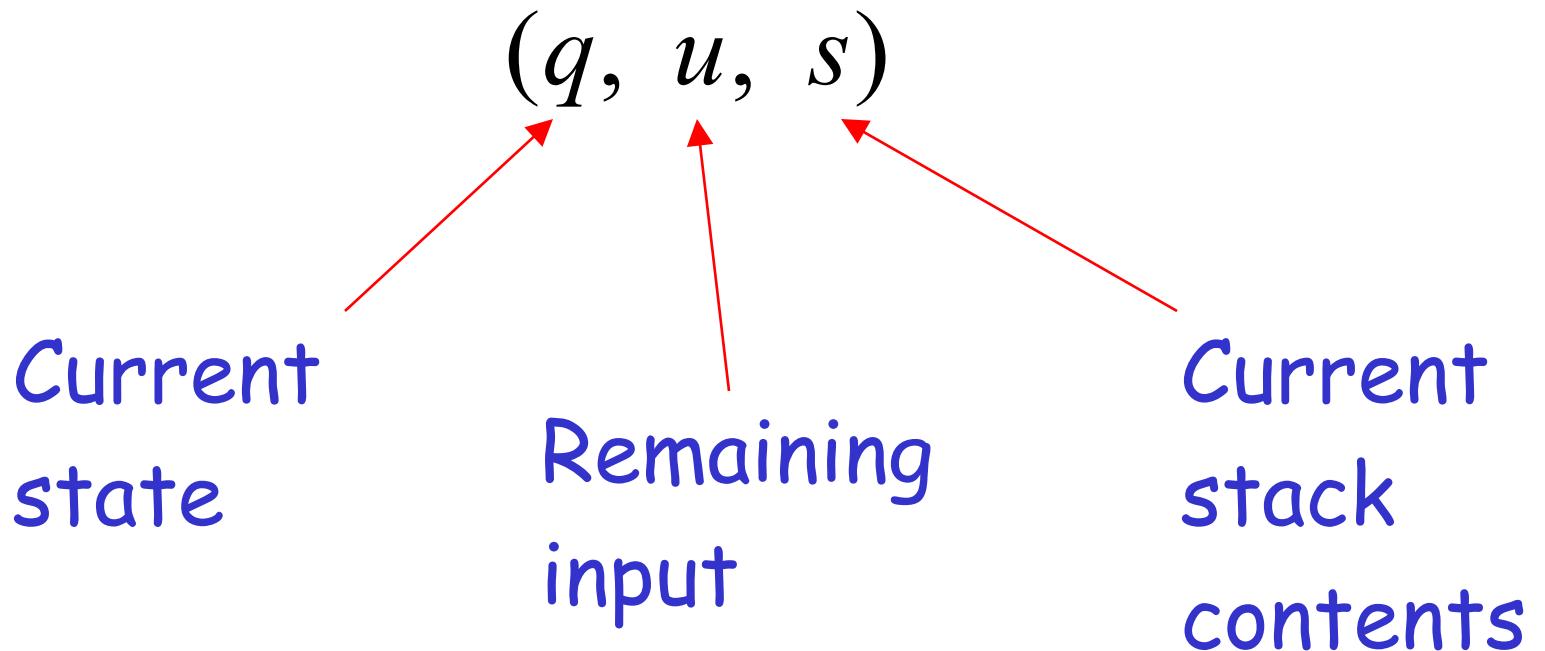
Pushdown Automaton (PDA)



Delta

: State \times Input \times Stack \rightarrow P { (State, Stack) }

Instantaneous Description



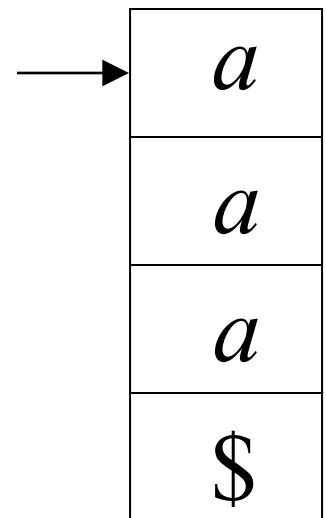
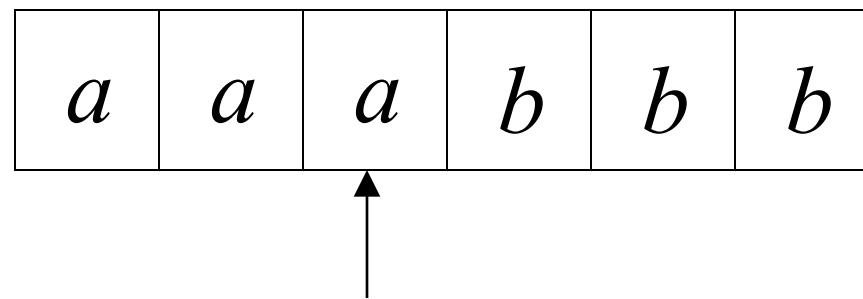
Example:

Instantaneous Description

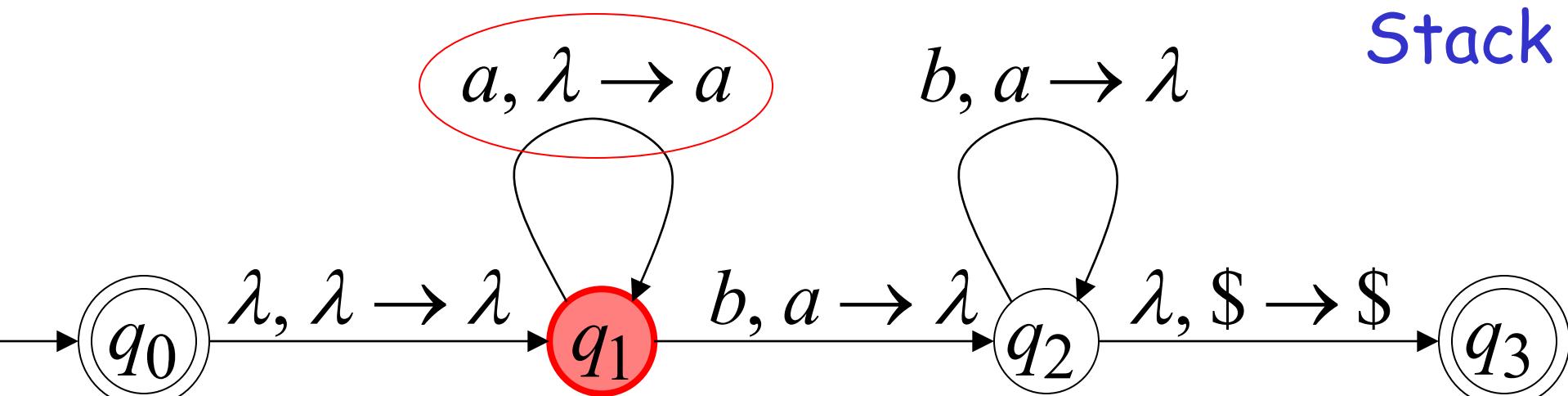
$$(q_1, bbb, aaa\$)$$

Time 4:

Input



Stack



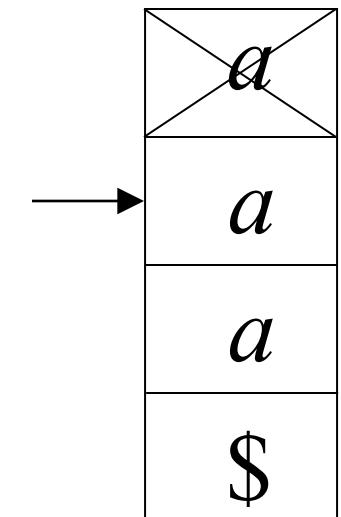
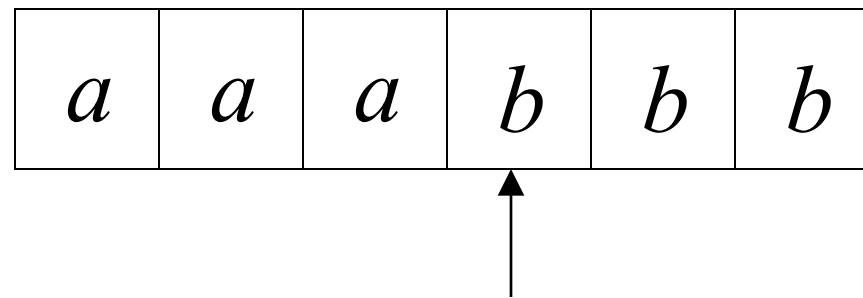
Example:

Instantaneous Description

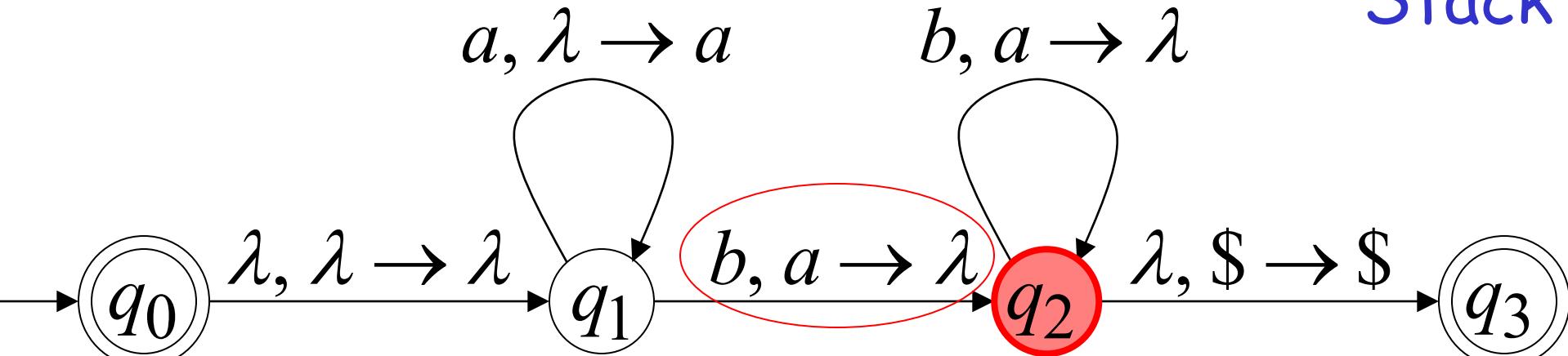
$(q_2, bb, aa\$)$

Time 5:

Input



Stack



scriviamo:

$$(q_1, bbb, aaa\$) \succ (q_2, bb, aa\$)$$

Time 4

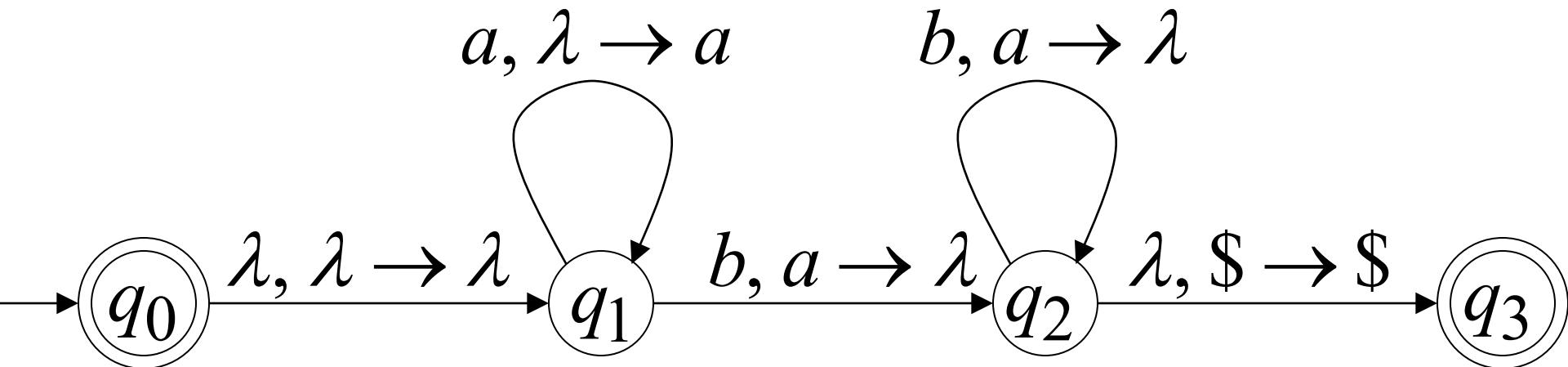
Time 5

Una computazione:

$(q_0, aaabbb, \$) \succ (q_1, aaabbb, \$) \succ$

$(q_1, aabbb, a\$) \succ (q_1, abbb, aa\$) \succ (q_1, bbb, aaa\$) \succ$

$(q_2, bb, aa\$) \succ (q_2, b, a\$) \succ (q_2, \lambda, \$) \succ (q_3, \lambda, \$)$



$$(q_0, aaabbb, \$) \succ (q_1, aaabbb, \$) \succ$$
$$(q_1, aabbb, a\$) \succ (q_1, abbb, aa\$) \succ (q_1, bbb, aaa\$) \succ$$
$$(q_2, bb, aa\$) \succ (q_2, b, a\$) \succ (q_2, \lambda, \$) \succ (q_3, \lambda, \$)$$

Per convenienza scriviamo:

$$(q_0, aaabbb, \$) \stackrel{*}{\succ} (q_3, \lambda, \$)$$

Language of PDA

Linguaggio $L(M)$ accettato da PDA M :

$$L(M) = \{w : (q_0, w, z) \stackrel{*}{\succ} (q_f, \lambda, s)\}$$

Initial state

Accept state

Stack può essere anche non vuoto, quindi s qualsiasi.

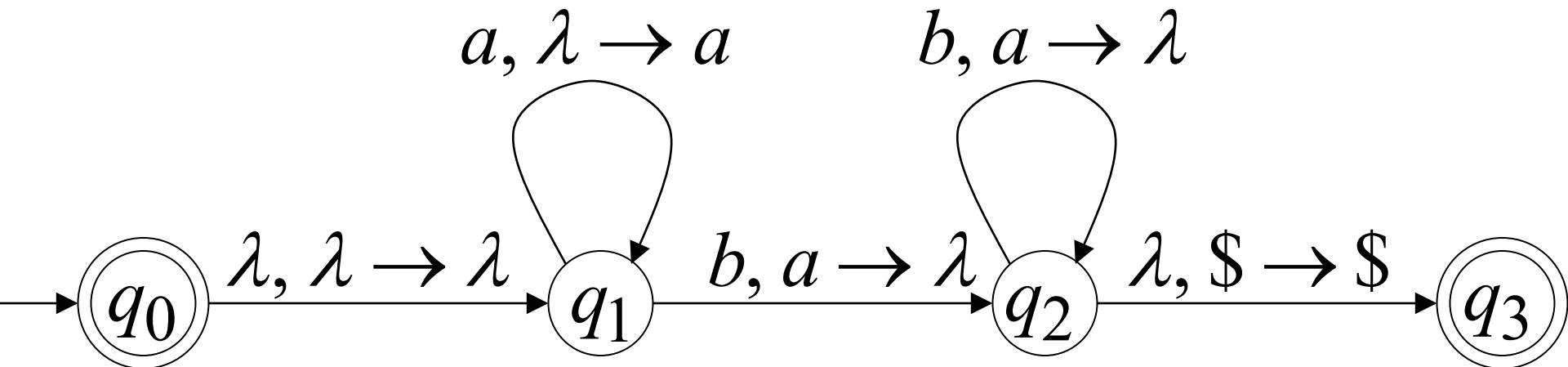
Esempio:

$$(q_0, aaabbb, \$) \xrightarrow{*} (q_3, \lambda, \$)$$



$$aaabbb \in L(M)$$

PDA M :

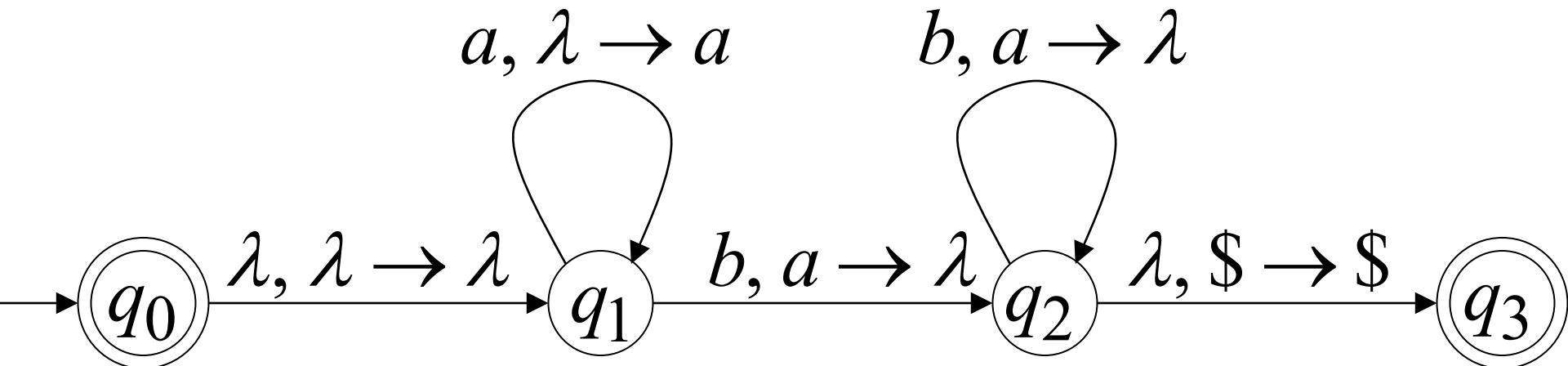


$$(q_0, a^n b^n, \$) \xrightarrow{*} (q_3, \lambda, \$)$$



$$a^n b^n \in L(M)$$

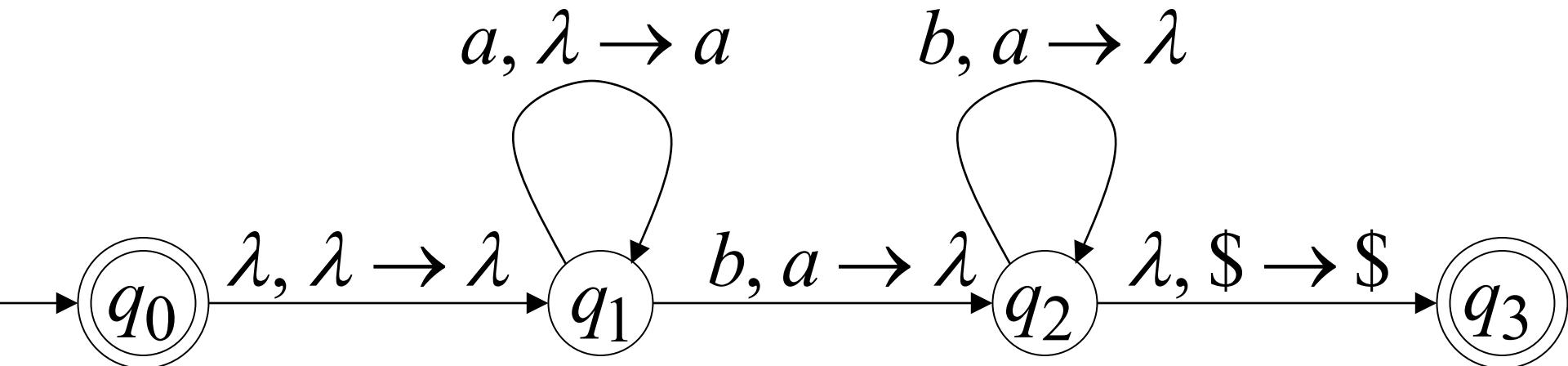
PDA M :



quindi:

$$L(M) = \{a^n b^n : n \geq 0\}$$

PDA M :

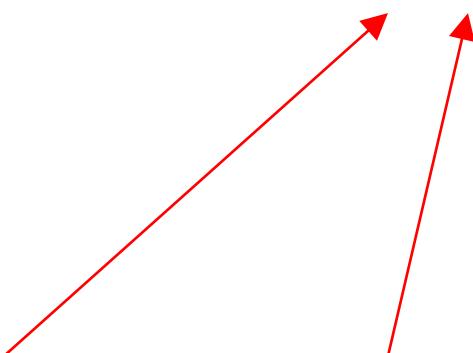


Normal Forms per grammatiche Context-free

Context free

Ogni produzioni ha la forma:

$$A \rightarrow \text{stringa}$$

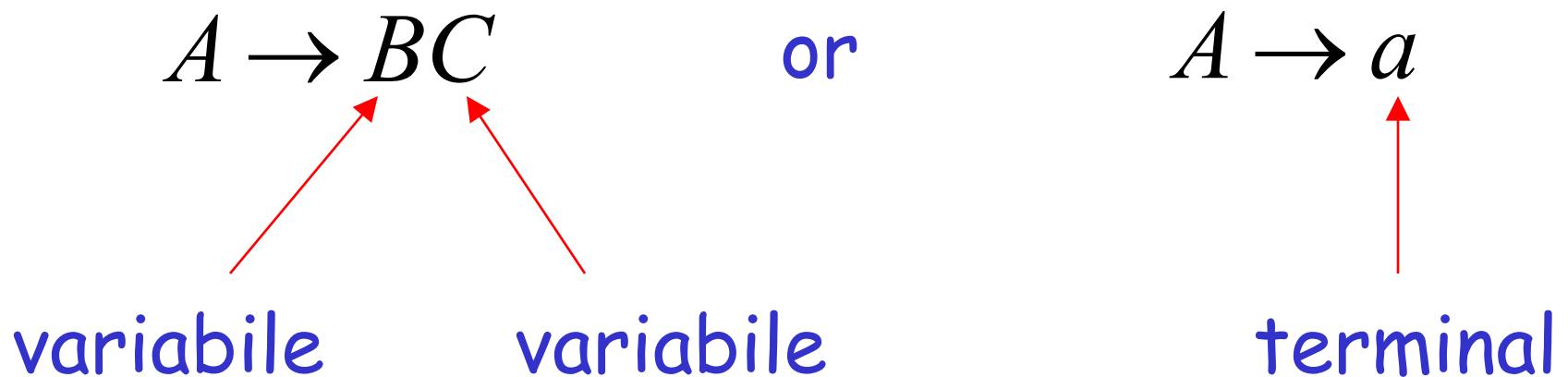


variabile

Terminali o costanti

Chomsky Normal Form

Ogni produzioni ha la forma:



esempi:

$$S \rightarrow AS$$

$$S \rightarrow a$$

$$A \rightarrow SA$$

$$A \rightarrow b$$

Chomsky
Normal Form

$$S \rightarrow AS$$

$$S \rightarrow AAS$$

$$A \rightarrow SA$$

$$A \rightarrow aa$$

Not Chomsky
Normal Form

Conversione nella Chomsky Normal Form

esempio:

$$S \rightarrow ABa$$

$$A \rightarrow aab$$

$$B \rightarrow Ac$$

Not Chomsky
Normal Form

Convertiamo questa grammatica nella
Chomsky Normal Form

Introduciamo nuove variabili per i terminali:

$$T_a, T_b, T_c$$

$$S \rightarrow ABa$$

$$A \rightarrow aab$$

$$B \rightarrow Ac$$



$$S \rightarrow ABT_a$$

$$A \rightarrow T_a T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$

Introduciamo una nuova variabile intermedia

Per rompere la prima produzione:

$$S \rightarrow ABT_a$$

$$A \rightarrow T_a T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$



$$V_1$$

$$S \rightarrow AV_1$$

$$V_1 \rightarrow BT_a$$

$$A \rightarrow T_a T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$

Introduciamo la variabile intermedia : V_2

$$S \rightarrow AV_1$$

$$V_1 \rightarrow BT_a$$

$$A \rightarrow T_a T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$

$$S \rightarrow AV_1$$

$$V_1 \rightarrow BT_a$$

$$A \rightarrow T_a V_2$$

$$V_2 \rightarrow T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$



grammatica in Chomsky Normal Form:

$$S \rightarrow AV_1$$

$$V_1 \rightarrow BT_a$$

$$A \rightarrow T_a V_2$$

$$V_2 \rightarrow T_a T_b$$

$$B \rightarrow AT_c$$

$$T_a \rightarrow a$$

$$T_b \rightarrow b$$

$$T_c \rightarrow c$$

Iniziale grammatica

$$S \rightarrow ABa$$

$$A \rightarrow aab$$

$$B \rightarrow Ac$$

In generale:

Per ogni grammatica context-free
(che non produce λ)
non in Chomsky Normal Form

Possiamo ottenere:

una grammatica equivalente
in Chomsky Normal Form

La procedura

First remove:

variabili che si possono
annulare

(variabili inutili, optional)

Poi, per ogni simbolo : a

Nuova variabile: T_a

Nuova produzione $T_a \rightarrow a$

Nelle produzioni con lunghezza maggiore o uguale a due

a T_a

produzioni della forma non terminale

Non necessitano di cambio!

Rimpiazza
ogni produzione

$$A \rightarrow C_1 C_2 \cdots C_n$$

con

$$A \rightarrow C_1 V_1$$

$$V_1 \rightarrow C_2 V_2$$

...

$$V_{n-2} \rightarrow C_{n-1} C_n$$

Nuove variabili intermedie : V_1, V_2, \dots, V_{n-2}

Observations

- Chomsky normal forms are good for parsing and proving theorems
- It is easy to find the Chomsky normal form for any context-free grammar

The Pumping Lemma for CFL's

Statement

Intuition

- ◆ Recall the pumping lemma for regular languages.
- ◆ It told us that if there was a string long enough to cause a cycle in the DFA for the language, then we could “pump” the cycle and discover an infinite sequence of strings that had to be in the language.

Intuition – (2)

- ◆ For CFL's the situation is a little more complicated.
- ◆ We can always find **two** pieces of any sufficiently long string to “pump” in tandem.
 - ◆ **That is:** if we repeat each of the two pieces the same number of times, we get another string in the language.

Statement of the CFL Pumping Lemma

For every context-free language L

There is an integer n , such that

For every string z in L of length $\geq n$

There exists $z = uvwxy$ such that:

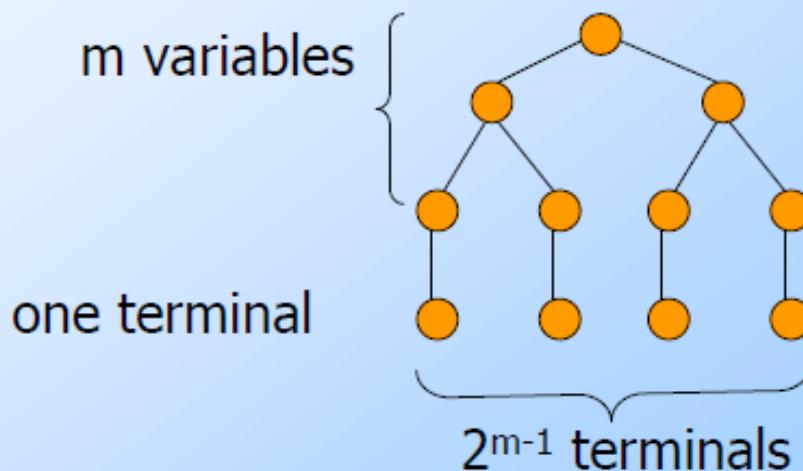
1. $|vwx| \leq n$.
2. $|vx| > 0$.
3. For all $i \geq 0$, $uv^iwx^i y$ is in L .

Proof of the Pumping Lemma

- ◆ Start with a CNF grammar for $L - \{\epsilon\}$.
- ◆ Let the grammar have m variables.
- ◆ Pick $n = 2^m$.
- ◆ Let $|z| \geq n$.
- ◆ We claim ("*Lemma 1*") that a parse tree with yield z must have a path of length $m+2$ or more.

Proof of Lemma 1

- ◆ If all paths in the parse tree of a CNF grammar are of length $\leq m+1$, then the longest yield has length 2^{m-1} , as in:

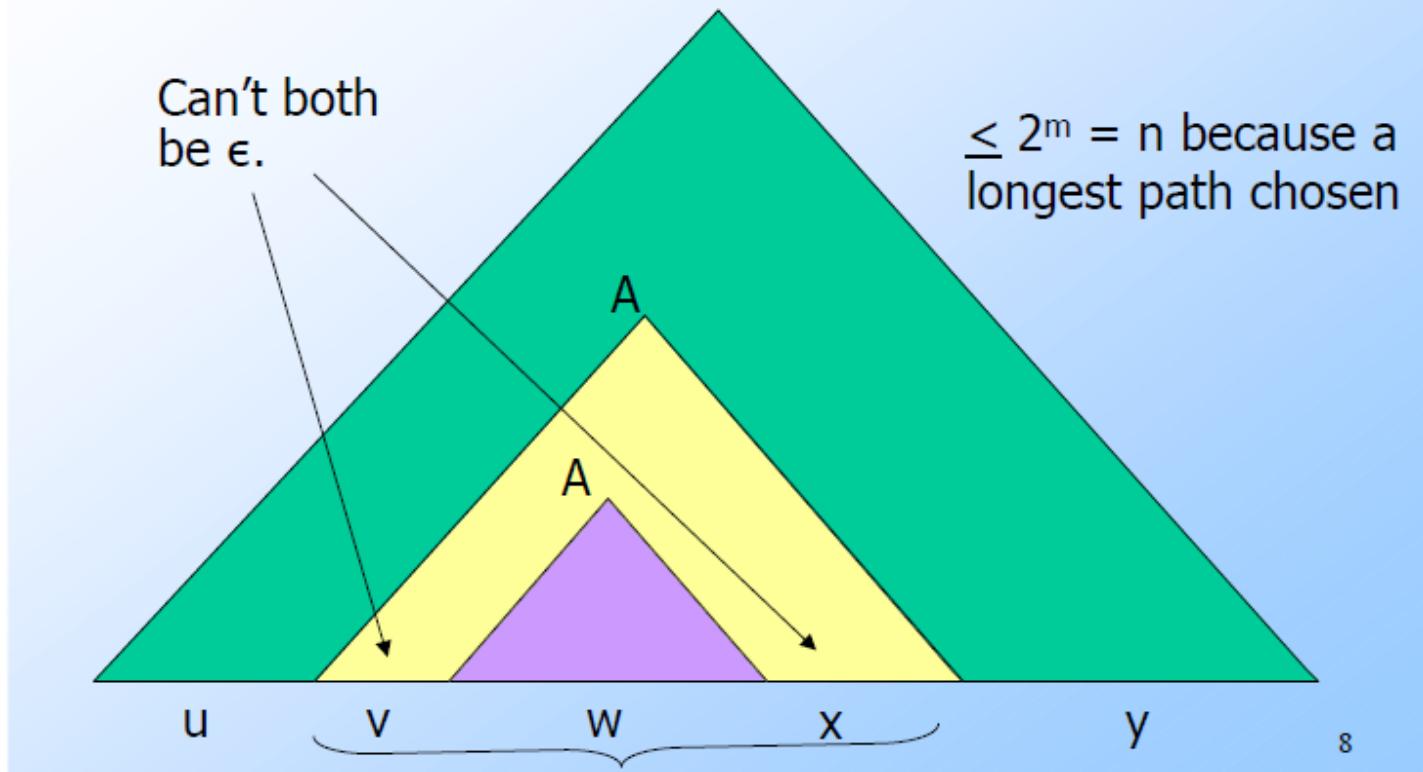


6

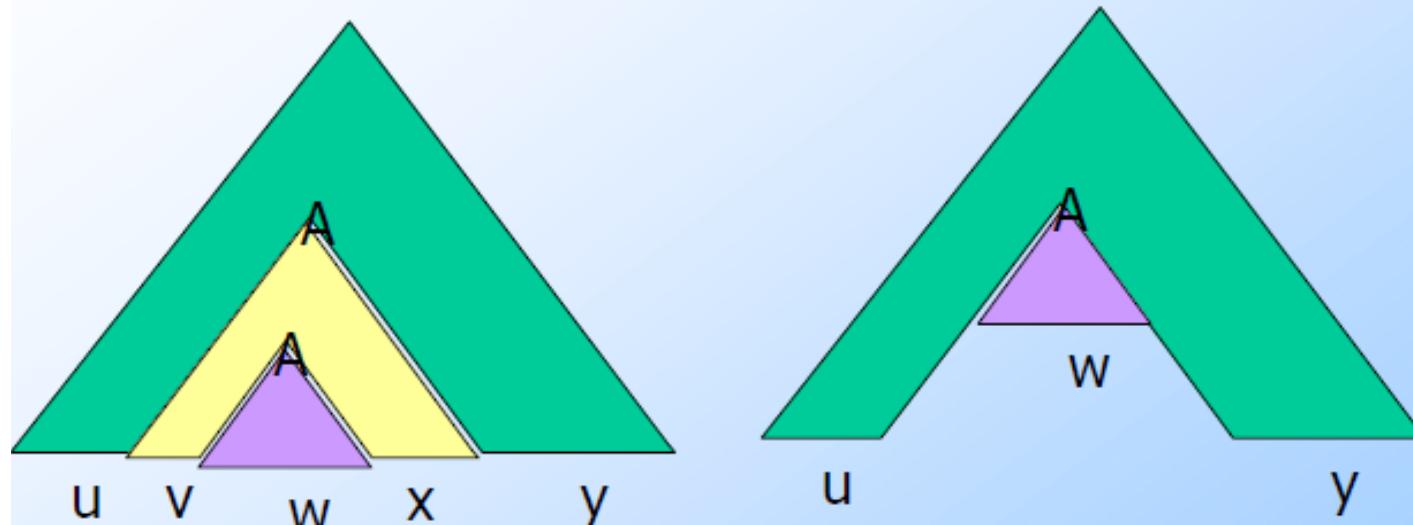
Back to the Proof of the Pumping Lemma

- ◆ Now we know that the parse tree for z has a path with at least $m+1$ variables.
- ◆ Consider some longest path.
- ◆ There are only m different variables, so among the **lowest** $m+1$ we can find two nodes with the same label, say A .
- ◆ The parse tree thus looks like:

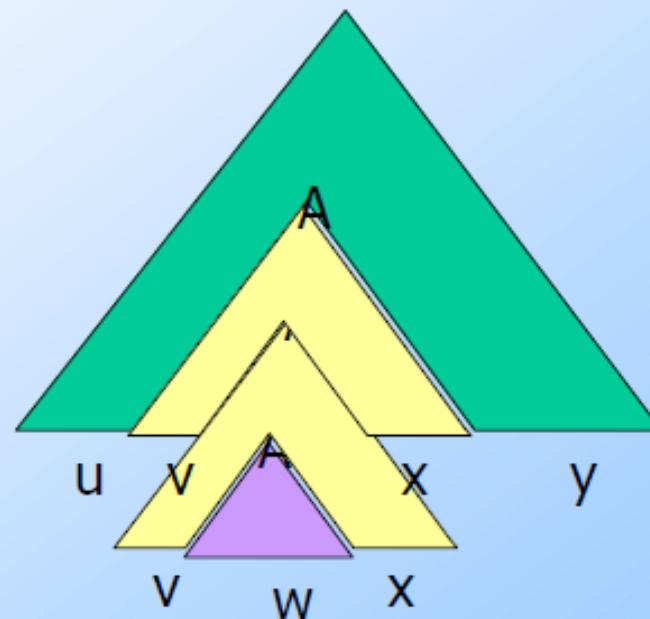
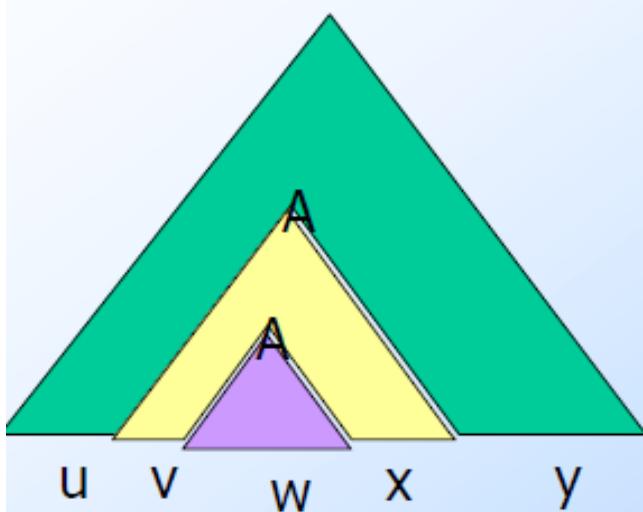
Parse Tree in the Pumping-Lemma Proof



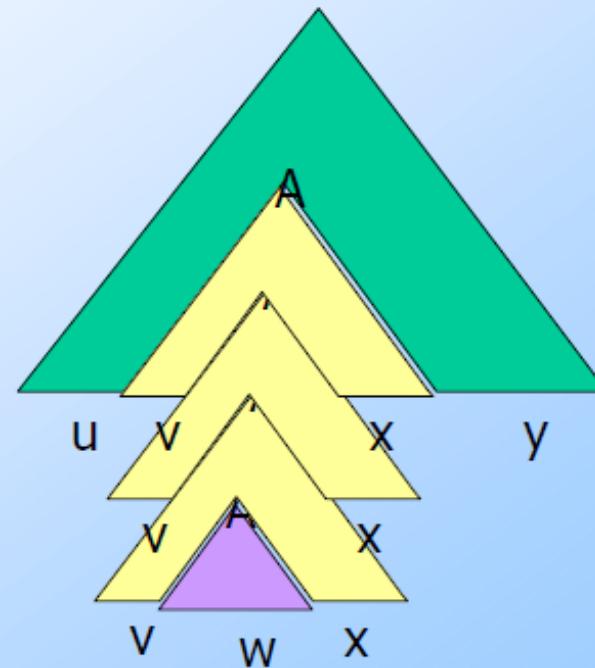
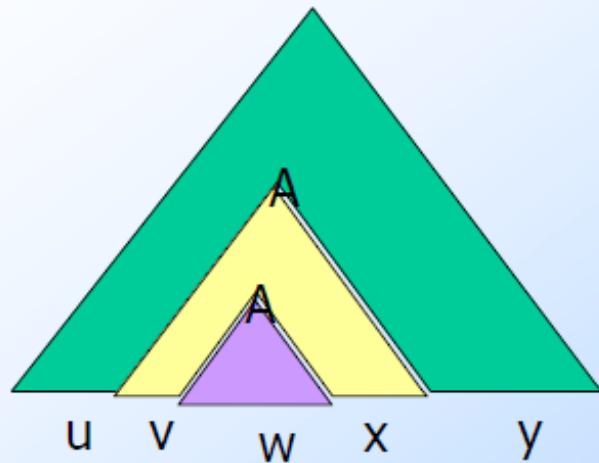
Pump Zero Times



Pump Twice



Pump Thrice Etc., Etc.



applicazioni del Pumping Lemma

Gli stati dell'automa , Variabili della grammatica e vale l'opposto.

Le variabili context free sono gli stati del PDA?

$W y Z x K$, $W yy Z xx K$, $W yyy Z xxx K$

Qualcosa di più complicato dobbiamo considerare sia la variabile che genera la y (Y) e sia la variabile che genera la x (X)

Stato YX ?

Quando ho un linguaggio se «provo» che le parole possono essere scritte

X $\underline{y_i}$ Y

A^* a, aa, aaa,

A^*B^* due pezzi le A e le B indipendenti

W $\underline{y_i}$ Z $\underline{x_i}$ K

$A_n B_n$ no regolare

AA BB non indipendenti

Il Pumping Lemma:

- dato un linguaggio regolare infinito L
- esiste un intero m (lunghezza critica)
- Per ogni stringa $w \in L$ con lunghezza $|w| \geq m$
- possiamo scrivere $w = x y z$
- con $|x y| \leq m$ e $|y| \geq 1$
- tale che: $x y^i z \in L$ $i = 0, 1, 2, \dots$

Teorema: Il linguaggio

$$L = \{vv^R : v \in \Sigma^*\} \quad \Sigma = \{a, b\}$$

non è regolare

Proof: Usiamo il Pumping Lemma

$$L = \{vv^R : v \in \Sigma^*\}$$

assumiamo per contraddizione
che L sia un linguaggio regolare

poichè L è infinito
Possiamo applicare il Pumping Lemma

$$L = \{vv^R : v \in \Sigma^*\}$$

sia m la lunghezza critica per L

Prendiamo una stringa w tale che: $w \in L$

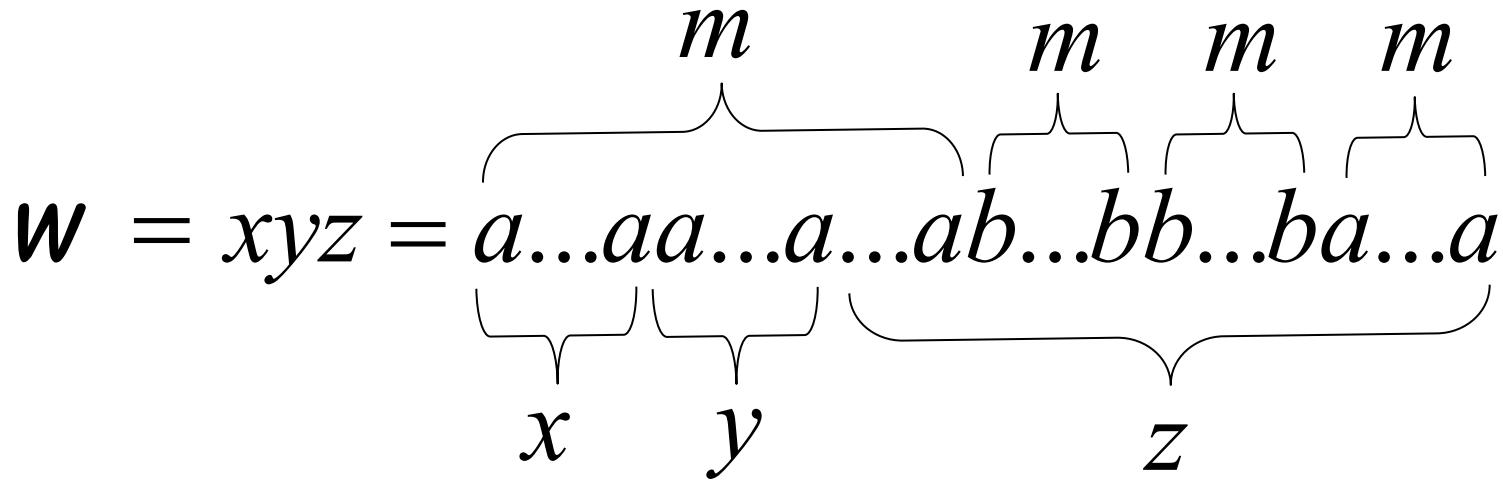
con lunghezza
 $|w| \geq m$

prendiamo $w = a^m b^m b^m a^m$

Dal Pumping Lemma:

Possiamo scrivere: $w = a^m b^m b^m a^m = x y z$

con lunghezza: $|x y| \leq m, |y| \geq 1$



allora: $y = a^k, 1 \leq k \leq m$

$$x \ y \ z = a^m b^m b^m a^m \quad y = a^k, \quad 1 \leq k \leq m$$

dal Pumping Lemma:

$$x \ y^i \ z \in L$$

$$i = 0, 1, 2, \dots$$

allora: $x \ y^2 \ z \in L$

$$x y z = a^m b^m b^m a^m \quad y = a^k, \quad 1 \leq k \leq m$$

Dal Pumping Lemma:

$$x y^2 z \in L$$

$$xy^2z = \underbrace{a \dots aa \dots aa \dots a}_{x} \underbrace{\dots}_{y} \underbrace{\dots}_{y} \underbrace{ab \dots bb \dots ba \dots a}_{z} \quad \square L$$

$m+k$
 $m \quad m \quad m$

allora: $a^{m+k} b^m b^m a^m \in L$

$$a^{m+k} b^m b^m a^m \in L \quad k \geq 1$$

ma:

$$L = \{vv^R : v \in \Sigma^*\}$$



$$a^{m+k} b^m b^m a^m \notin L$$

CONTRADIZIONE!!!

Considerare i casi
con y tra le b, tra le
a e tra ab.

$$xy^2z = \underbrace{a \dots aa \dots aa \dots a}_{x} \underbrace{\dots}_{y} \underbrace{\dots}_{y} \underbrace{ab \dots bb \dots ba \dots a}_{z} \quad \square L$$

$m+k$
 $m \quad m \quad m$

quindi:

L'assunzione che L è
un linguaggio regolare non
è vera

Conclusione: L Non è un linguaggio regolare

END OF PROOF

Teorema: il linguaggio

$$L = \{a^n b^l c^{n+l} : n, l \geq 0\}$$

non è regolare

Proof: Usiamo il Pumping Lemma

$$L = \{a^n b^l c^{n+l} : n, l \geq 0\}$$

assumiamo per contraddizione
che L sia un linguaggio regolare

poichè L è infinito allora
possiamo applicare il Pumping Lemma

$$L = \{a^n b^l c^{n+l} : n, l \geq 0\}$$

sia m la lunghezza critica di L

Prendiamo una stringa w tale che: $w \in L$

e lunghezza $|w| \geq m$

prendiamo $w = a^m b^m c^{2m}$

Dal Pumping Lemma:

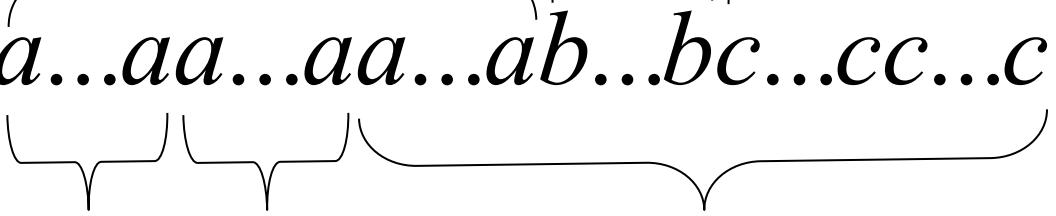
possiamo scrivere $w = a^m b^m c^{2m} = x y z$

con lunghezzas

$$|x y| \leq m, |y| \geq 1$$

$$w = xyz = \underbrace{a \dots aa \dots aa}_{x} \dots \underbrace{ab \dots bc \dots cc \dots c}_{z} \underbrace{\dots}_{y}$$

m m $2m$



allora: $y = a^k, 1 \leq k \leq m$

$$x \ y \ z = a^m b^m c^{2m} \quad y = a^k, \quad 1 \leq k \leq m$$

Dal Pumping Lemma: $x \ y^i \ z \in L$
 $i = 0, 1, 2, \dots$

allora: $x \ y^0 \ z = xz \quad \square \quad L$

$$x \ y \ z = a^m b^m c^{2m} \quad y = a^k, \quad 1 \leq k \leq m$$

Dal Pumping Lemma: $xz \in L$

$$xz = a \dots a a \dots a b \dots b c \dots c c \dots c \in L$$

$m-k$ m $2m$

x z

allora: $a^{m-k} b^m c^{2m} \in L$

$$a^{m-k} b^m c^{2m} \in L \quad k \geq 1$$

ma:

$$L = \{a^n b^l c^{n+l} : n, l \geq 0\}$$



$$a^{m-k} b^m c^{2m} \notin L$$

Contradizione.

Vedere gli altri casi. La y tra le b, tra le c, tra ab, tra bc.

La nostra assunzione che L sia un linguaggio regolare non è vera

Conclusione: L non è un linguaggio regolare

END OF PROOF

Teorema: il linguaggio $L = \{a^{n!} : n \geq 0\}$

Non è regolare

$$n! = 1 \cdot 2 \cdots (n-1) \cdot n$$

dimostrazione: Usiamo il Pumping Lemma

$$L = \{a^{n!} : n \geq 0\}$$

assumiamo che L
sia un linguaggio regolare

poichè L è infinito
Possiamo applicare il Pumping Lemma

$$L = \{a^{n!} : n \geq 0\}$$

sia m la lunghezza critica of L

prendiamo una stringa w tale che: $w \in L$

lunghezza $|w| \geq m$

prendiamo $w = a^{m!}$

Dal Pumping Lemma:

Possiamo scrivere

$$w = a^{m!} = x \ y \ z$$

con lunghezza

$$|x \ y| \leq m, \ |y| \geq 1$$

$$w = xyz = a^{m!} = \overbrace{a \dots a}^m \dots \overbrace{a \dots a}^{m!-m} \dots \overbrace{a \dots a}^m$$

$x \quad y \quad z$

allora: $y = a^k, \ 1 \leq k \leq m$

$$x \ y \ z = a^{m!} \quad y = a^k, \quad 1 \leq k \leq m$$

Dal Pumping Lemma: $x \ y^i \ z \in L$
 $i = 0, 1, 2, \dots$

allora: $x \ y^2 \ z \in L$

$$x \ y \ z = a^{m!}$$

$$y = a^k, \quad 1 \leq k \leq m$$

Dal Pumping Lemma:

$$x \ y^2 \ z \in L$$

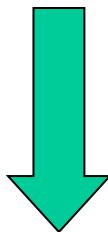
$$xy^2z = \overbrace{a \dots aa \dots aa \dots aa \dots aa \dots aa \dots a}^{m+k} \in L$$

$m + k$
 $x \quad y \quad y \quad z \quad m!-m$

allora: $a^{m!+k} \in L$

$$a^{m!+k} \in L \quad 1 \leq k \leq m$$

poichè: $L = \{a^{n!} : n \geq 0\}$



Deve esistere p tale che:

$$m!+k = p!$$

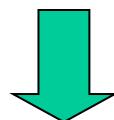
ovvero: $m!+k \leq m!+m$ per $m > 1$

$$\leq m!+m!$$

$$< m!m + m!$$

$$= m!(m+1)$$

$$= (m+1)!$$



$$m!+k < (m+1)!$$



$m!+k \neq p!$ Per ogni p

$$a^{m!+k} \in L \quad 1 \leq k \leq m$$

ma:

$$L = \{a^{n!} : n \geq 0\}$$



$$a^{m!+k} \notin L$$

contradizione

quindi:

La nostra assunzione che L
È un linguaggio regolare
Non è vera

Conclusione: L Non è un linguaggio regolare

END OF PROOF

Applicazioni del Pumping Lemma context free

The Pumping Lemma:

Per un linguaggio infinito context-free L

Esiste un intero m tale che

per ogni stringa $w \in L$, $|w| \geq m$

possiamo scrivere $w = uvxyz$

Con lunghezze $|vxy| \leq m$ and $|vy| \geq 1$

E deve essere:

$uv^i xy^i z \in L$, for all $i \geq 0$

$m \geq$

(2 _ numero delle variabili
della grammatica) - 1

Perché?

linguaggi Non-context free

$\{a^n b^n c^n : n \geq 0\}$

$\{vv : v \in \{a,b\}^*\}$

linguaggi Context-free

$\{a^n b^n : n \geq 0\}$

$\{ww^R : w \in \{a,b\}^*\}$

Perché?

Teorema: il linguaggio

$$L = \{vv : v \in \{a,b\}^*\}$$

non è context free

Dim.: Usiamo il Pumping Lemma
Per i linguaggi context-free

$$L = \{vv : v \in \{a,b\}^*\}$$

Assumiamo per assurdo che L
è context-free

poichè L è context-free e infinito
Possiamo applicare il pumping lemma

$$L = \{vv : v \in \{a,b\}^*\}$$

Pumping Lemma ci dà un magico numero m tale che da lì in poi due pezzi della stringa si ripetono.

Prendiamo

una stringa di L con lunghezza almeno m

sia: $a^m b^m a^m b^m \in L$

$$L = \{vv : v \in \{a,b\}^*\}$$

possiamo scrivere: $a^m b^m a^m b^m = uvxyz$

con lunghezze $|vxy| \leq m$ e $|vy| \geq 1$

Pumping Lemma dice:

$uv^i xy^i z \in L$ per tutti $i \geq 0$

$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Esaminiamo tutti i possibili “posti”

Dove la stringa vxy può essere

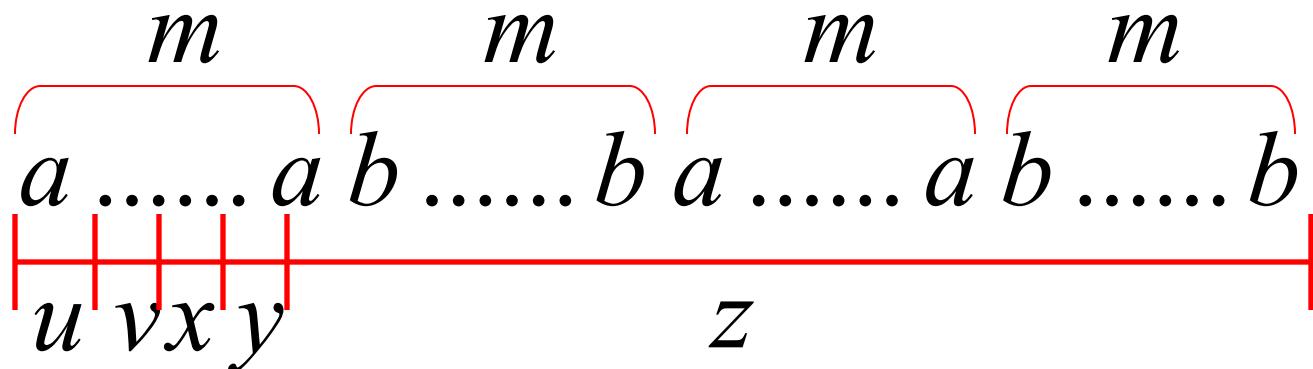
in $a^m b^m a^m b^m$

$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 1: vxy È nel primo a^m

$$v = a^{k_1} \quad y = a^{k_2} \quad k_1 + k_2 \geq 1$$

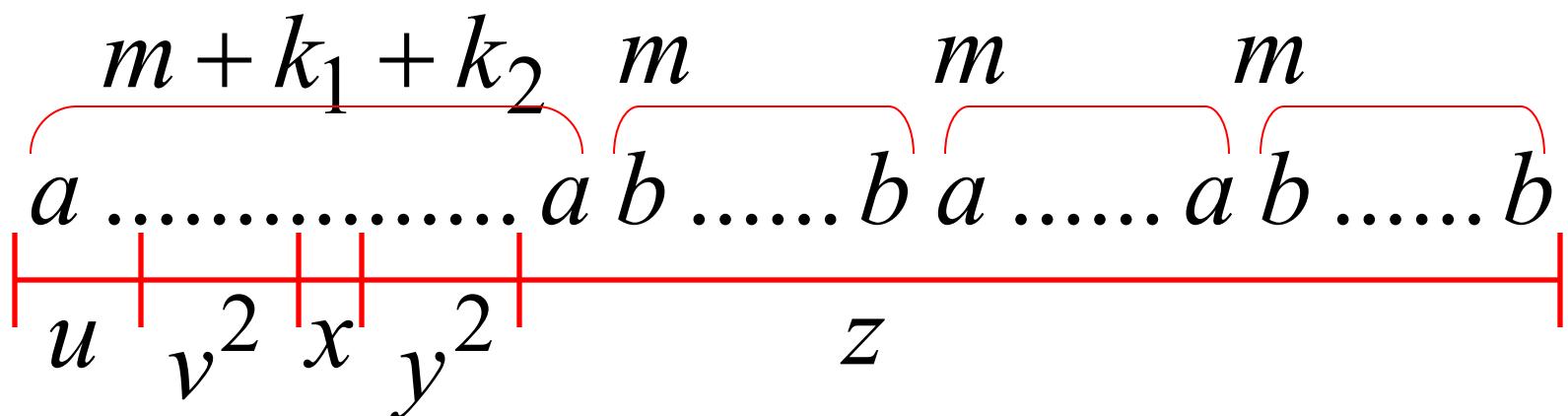


$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 1: vxy è nel primo a^m

$$v = a^{k_1} \quad y = a^{k_2} \quad k_1 + k_2 \geq 1$$



$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 1: vxy è nel primo a^m

$$a^{m+k_1+k_2} b^m a^m b^m = u v^2 x y^2 z \notin L$$

$$k_1 + k_2 \geq 1$$

$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 1: vxy è nel primo a^m

$$a^{m+k_1+k_2} b^m a^m b^m = uv^2 xy^2 z \notin L$$

Ma dal pumping lemma abbiamo: $uv^2 xy^2 z \in L$

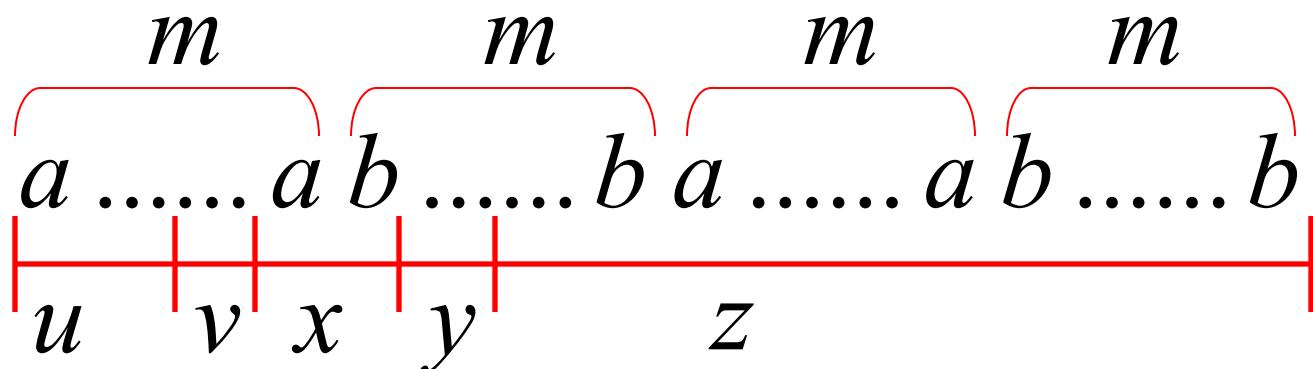
Contradizione!!!

$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 2: v È nel primo a^m
 y È nel primo b^m

$$v = a^{k_1} \quad y = b^{k_2} \quad k_1 + k_2 \geq 1$$



$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

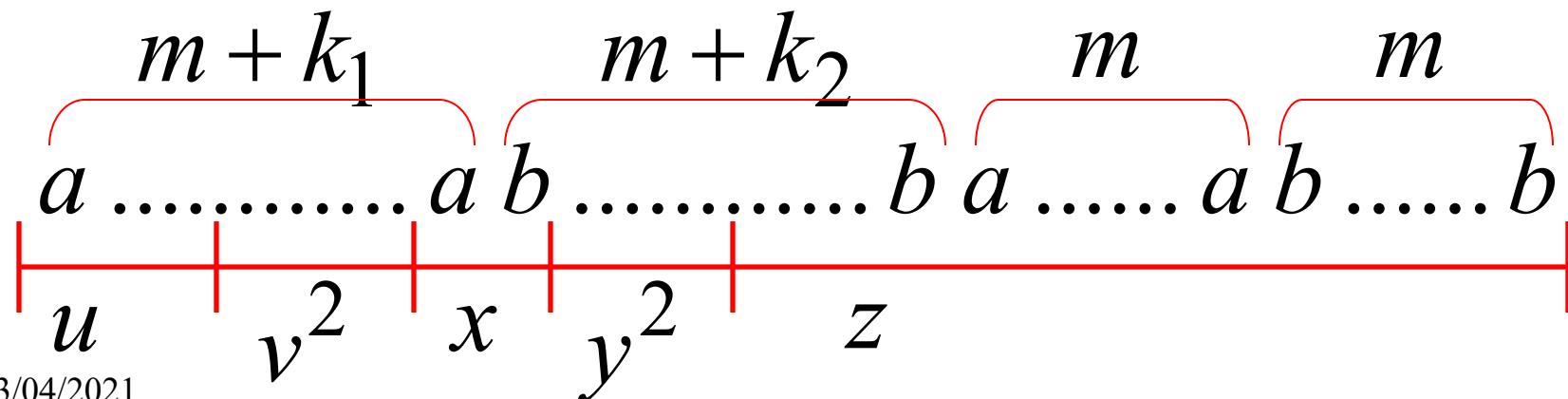
Case 2: v è nel primo

y è nel primo

$$a^m$$

$$b^m$$

$$v = a^{k_1} \quad y = b^{k_2} \quad k_1 + k_2 \geq 1$$



$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 2: v è nel primo a^m
 y è nel primo b^m

$$a^{m+k_1} b^{m+k_2} a^m b^m = uv^2 xy^2 z \notin L$$

$$k_1 + k_2 \geq 1$$

$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 2: v È nel primo a^m
 y È nel primo b^m

$$a^{m+k_1} b^{m+k_2} a^m b^m = uv^2 xy^2 z \notin L$$

Dal Pumping Lemma:

$$uv^2 xy^2 z \in L$$

Contradizione!!!

$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

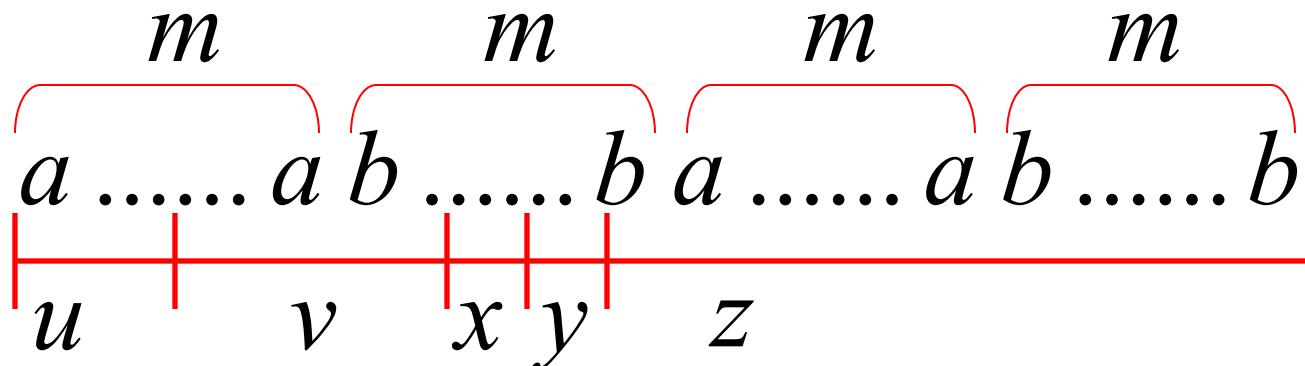
Case 3: v Sovrappone sul primo

y È nel primo

$$a^m b^m$$

$$b^m$$

$$v = a^{k_1} b^{k_2} \quad y = b^{k_3} \quad k_1, k_2 \geq 1$$

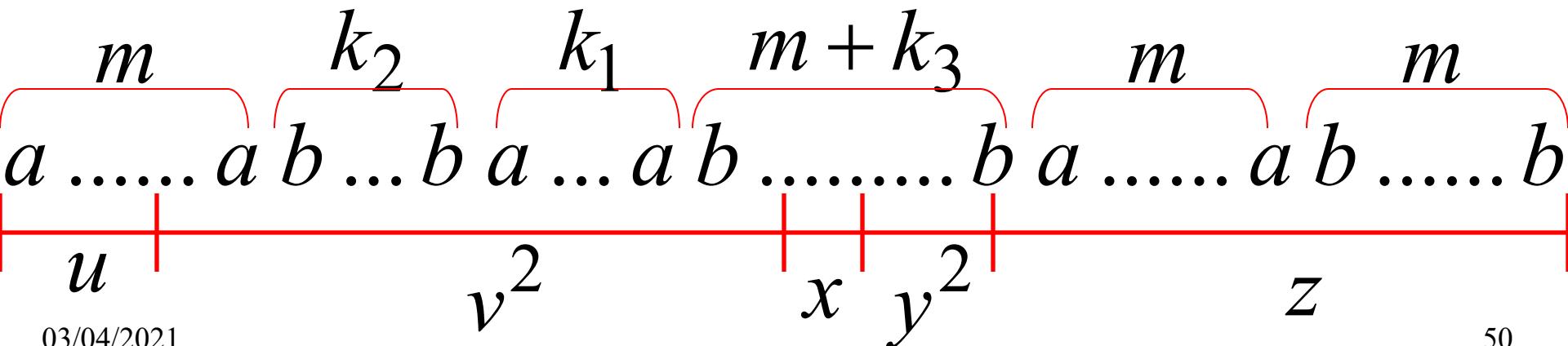


$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 3: v Sovrappone sul primo $a^m b^m$
 y è nel primo b^m

$$v = a^{k_1} b^{k_2} \quad y = b^{k_3} \quad k_1, k_2 \geq 1$$



$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 3: v Sovrappone sul primo $a^m b^m$
 y È nel primo b^m

$$a^m b^{k_2} a^{k_1} b^{m+k_3} a^m b^m = uv^2 xy^2 z \notin L$$

$$k_1, k_2 \geq 1$$

$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 3: v Sovrappone sul primo $a^m b^m$
 y È nel primo b^m

$$a^m b^{k_2} a^{k_1} b^{k_3} a^m b^m = uv^2 xy^2 z \notin L$$

dal Pumping Lemma: $uv^2 xy^2 z \in L$

assurdo!!!

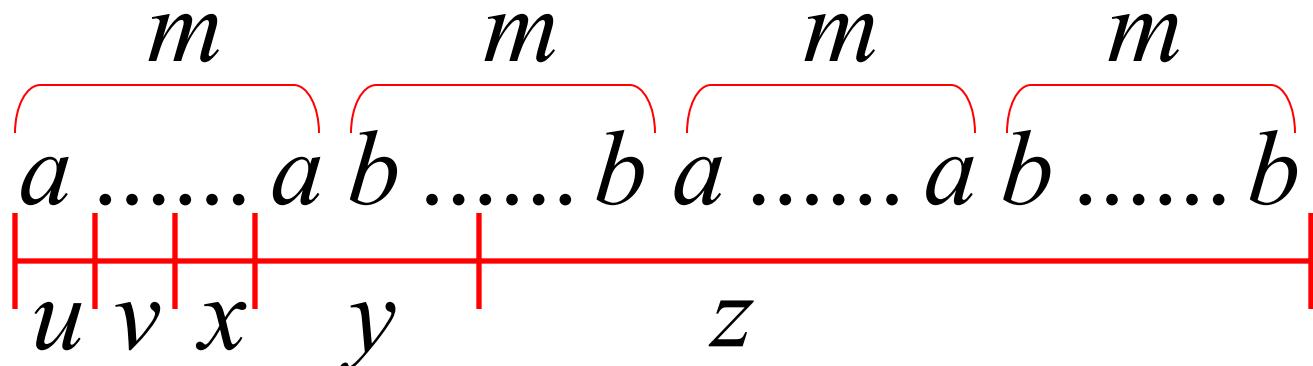
$$L = \{vv : v \in \{a,b\}^*\}$$

$$a^m b^m a^m b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Case 4: v È nel primo a^m

y Sovrappone $a^m b^m$

Analisi simile al caso 3



Altri casi:

vxy È dentro $a^m b^m a^m b^m$

or

$a^m b^m a^m b^m$

or

$a^m b^m a^m b^m$

Analisi simile al caso 1:

$a^m b^m a^m b^m$

Altri casi:

vxy sovrappone $a^m b^m a^m b^m$

or

$a^m b^m a^m b^m$

Analisi simile ai casi 2,3,4:

$a^m b^m a^m b^m$

Vi sono altri casi da considerare

Poichè $|vxy| \leq m$, è impossibile vxy sovrapporre a

$$a^m b^m a^m b^m$$

O, esclusivo

$$a^m b^m a^m b^m$$

O, esclusivo

$$a^m b^m a^m b^m$$

In tutti i casi raggiungiamo un **assurdo**

quindi:

Il punto di partenza che

$$L = \{vv : v \in \{a,b\}^*\}$$

è context-free è sbagliato

Conclusione: L Non è context-free

Linguaggi Non-context free

$\{a^n b^n c^n : n \geq 0\}$

$\{ww : w \in \{a,b\}^*\}$

$\{a^{n!} : n \geq 0\}$

linguaggi Context-free

$\{a^n b^n : n \geq 0\}$

$\{ww^R : w \in \{a,b\}^*\}$

linguaggi Non-context free

$\{a^n b^n c^n : n \geq 0\}$

$\{ww : w \in \{a,b\}^*\}$

$\{a^{n^2} b^n : n \geq 0\}$

$\{a^{n!} : n \geq 0\}$

linguaggi Context-free

$\{a^n b^n : n \geq 0\}$

$\{ww^R : w \in \{a,b\}^*\}$

Teorema: il linguaggio

$$L = \{a^{n^2} b^n : n \geq 0\}$$

non è context free

Dim:

Usiamo il Pumping Lemma
per linguaggi context-free

$$L = \{a^{n^2} b^n : n \geq 0\}$$

assumiamo per assurdo che L
è context-free

poichè L è context-free ed è infinito
possiamo applicare il pumping lemma

$$L = \{a^{n^2} b^n : n \geq 0\}$$

Pumping Lemma ci da m

Prendiamo una stringa di L

Con lunghezza almeno m

sia: $a^{m^2} b^m \in L$

$$L = \{a^{n^2} b^n : n \geq 0\}$$

Prendiamo m:

$$a^{m^2} b^m = uvxyz$$

con lunghezze

$$|vxy| \leq m \quad e \quad |vy| \geq 1$$

Pumping Lemma dice:

$$uv^i xy^i z \in L \quad \text{Per tutte le} \quad i \geq 0$$

$$L = \{a^{n^2} b^n : n \geq 0\}$$

$$a^{m^2} b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Esaminiamo tutte le possibili posizioni

Della stringa vxy in $a^{m^2} b^m$

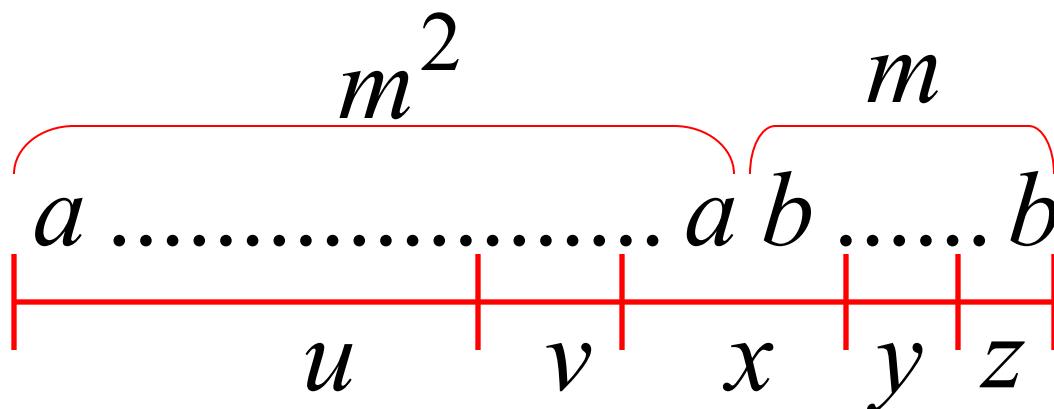
$$L = \{a^{n^2} b^n : n \geq 0\}$$

$$a^{m^2} b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Caso interessante:

v è in a^m
 y è in b^m

sia $v = a^{k_1} y = b^{k_2}$ con $1 \leq k_1 + k_2 \leq m$



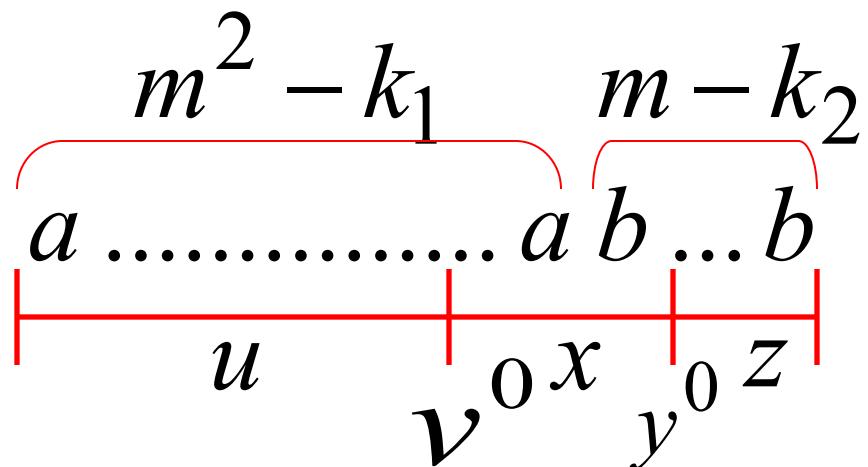
$$L = \{a^{n^2} b^n : n \geq 0\}$$

$$a^{m^2} b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Sotto caso in cui: $v = a^{k_1} \quad y = b^{k_2}$

Con $1 \leq k_1 + k_2 \leq m$

Sia $i=0$
abbiamo:



$$L = \{a^{n^2} b^n : n \geq 0\}$$

$$a^{m^2} b^m = uvxyz \quad |vxy| \leq m \quad |vy| \geq 1$$

Sia $i=0$:

$$v = a^{k_1} \quad y = b^{k_2} \quad 1 \leq k_1 + k_2 \leq m$$

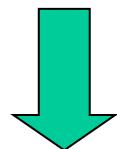
$$a^{m^2 - k_1} b^{m - k_2} = u v^0 x y^0 z$$

Vediamo il rapporto tra il numero di a e di b quando i=0

$$\begin{aligned}(m - k_2)^2 &\leq (m - 1)^2 \\&= m^2 - 2m + 1 \\&< m^2 - k_1 \\&\downarrow\end{aligned}$$

$$m^2 - k_1 \neq (m - k_2)^2$$

$$m^2 - k_1 \neq (m - k_2)^2$$



$$a^{m^2-k_1} b^{m-k_2} = uv^0 xy^0 z \notin L$$

Ma via PL:

$$uv^0 xy^0 z \in L$$

assurdo!!!

Casi

Caso 1. v e y sono una serie di a , quindi pumping v e y aumentano le a ma non le b

Caso 2. v e y sono una serie di b , quindi pumping v e y aumentano le b ma non le a .

Caso 3 (interessante) visto prima

v è una serie di a e y è una serie di b .

Si potrebbe pensare che crescono secondo le regole del linguaggio. Ma le a dovrebbero crescere rispetto alle b spettando il fatto che le tutte le a sono di lunghezza quadratico rispetto al numero delle b . Questo non è possibile perché le v , quindi le a , crescono linearmente (allo stesso modo) delle y , ovvero delle b .

dal Pumping Lemma: $uv^0xy^0z \in L$

$$uv^0xy^0z \in L$$

$$a^{m^2-k_1}b^{m-k_2} = uv^0xy^0z \notin L$$

In tutti i casi otteniamo un **assurdo**

quindi:

L'assunzione che

$$L = \{a^{n^2} b^n : n \geq 0\}$$

è context-free è sbagliata

Conclusion: L Non è context-free

PDA sono equivalenti
ai
linguaggi Context-Free

Teorema:

$$\left\{ \begin{array}{l} \text{Context-Free} \\ \text{linguaggi} \\ (\text{grammatiche}) \end{array} \right\} = \left\{ \begin{array}{l} \text{linguaggi} \\ \text{accettati da} \\ \text{PDA} \end{array} \right\}$$

dimostrazione - Step 1:

$$\left\{ \begin{array}{l} \text{Context-Free} \\ \text{linguaggi} \\ (\text{grammatiche}) \end{array} \right\} \subseteq \left\{ \begin{array}{l} \text{linguaggi} \\ \text{accettati by} \\ \text{PDAs} \end{array} \right\}$$

Traduci ogni grammatica context-free G
In un PDA M con: $L(G) = L(M)$

dimostrazione - step 1

trasformare
grammatiche Context-Free
in
PDAs

Prendiamo una grammatica context-free G



Tradurremo G in un PDA M tale che:

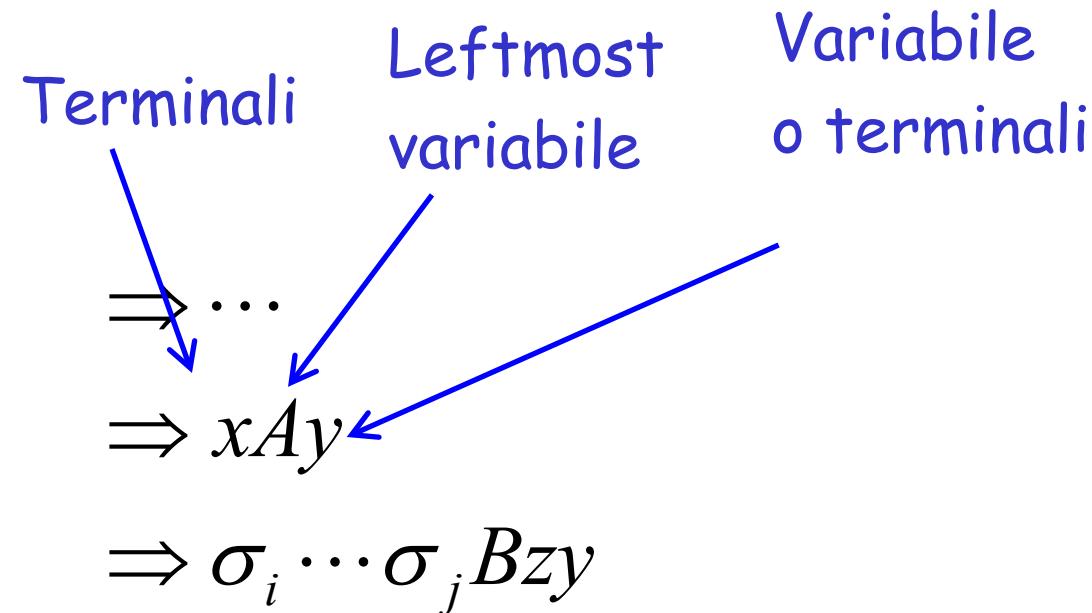
$$L(G) = L(M)$$

Def. : Una derivazione $S \Rightarrow a_1 \Rightarrow a_2 \Rightarrow \dots \Rightarrow a_{n-1} \Rightarrow a_n$ si dice **leftmost** (sinistra) se:
 $\forall i=1,\dots,n-1 \text{ si ha } a_i = uX\beta_i \text{ e } a_{i+1} = u\gamma\beta_i,$ con $u \in \Sigma^*, X \in N, (X \rightarrow \gamma)$ in P

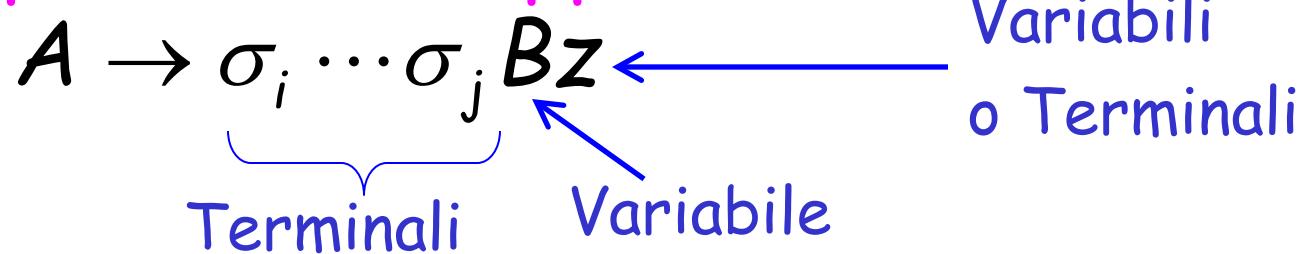
Nel primo caso si scrive : $a_i \xrightarrow[L]{} a_j \quad (i < j).$

Useremo solo leftmost

Grammatica consideriamo le Derivazioni Leftmost



produzione Applicata



Procedura di conversione:

per ogni produzione in G

per ogni terminale in G

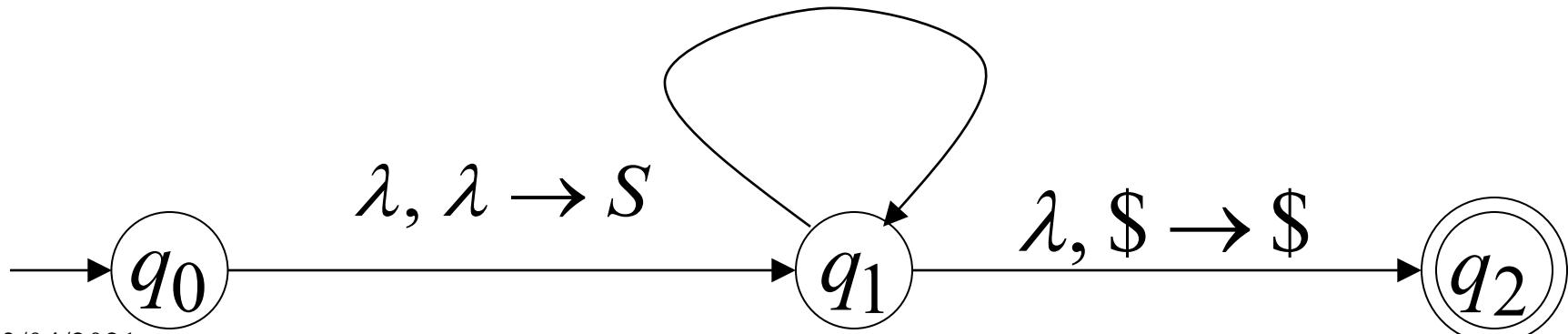
$$A \rightarrow w$$

Addiziona
le transizioni

$$\lambda, A \rightarrow w$$

$$a, a \rightarrow \lambda$$

$$a$$



grammatica

$$S \rightarrow aSTb$$

$$S \rightarrow b$$

$$T \rightarrow Ta$$

$$T \rightarrow \lambda$$

esempio

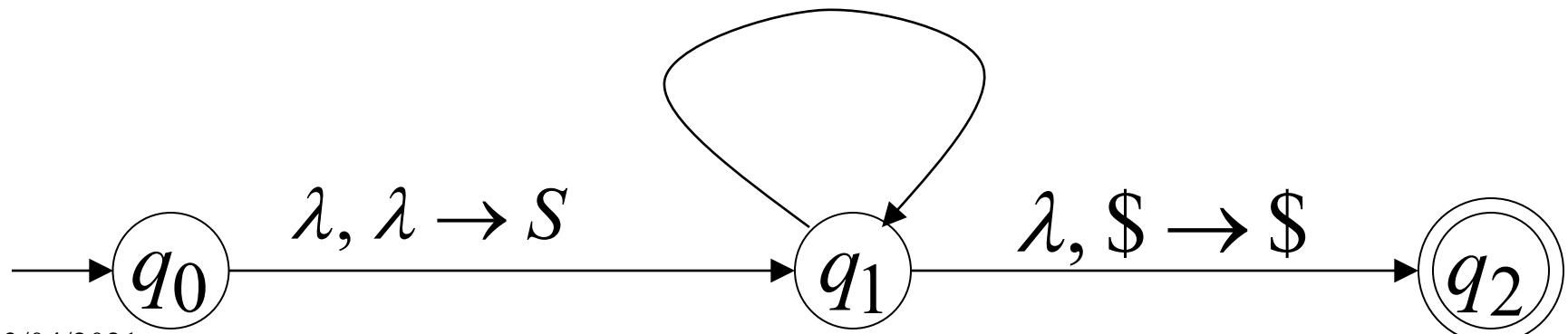
PDA

$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

$$\lambda, T \rightarrow Ta \quad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda \quad b, b \rightarrow \lambda$$



Esempio:

Input

a	b	a	b
-----	-----	-----	-----

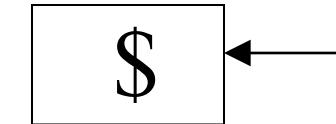


Time 0

$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

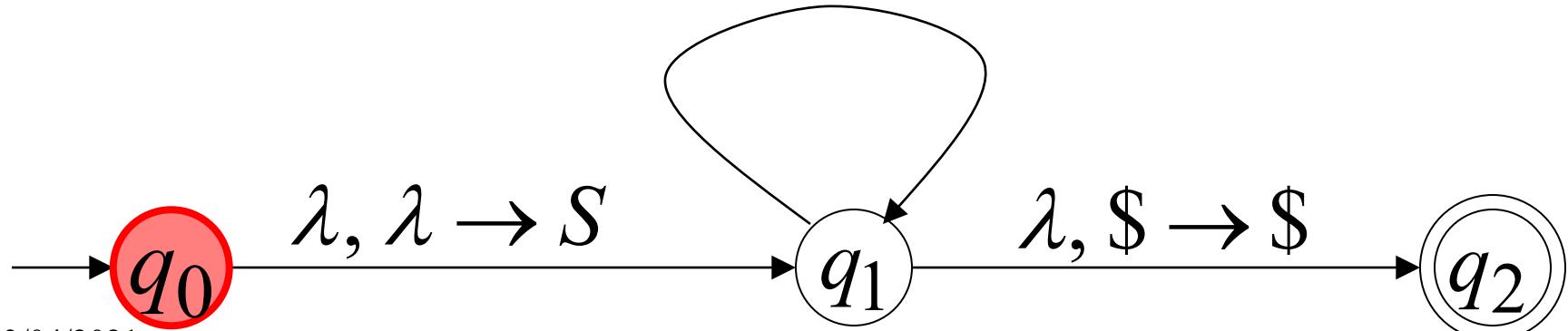
\$



Stack

$$\lambda, T \rightarrow Ta \quad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda \quad b, b \rightarrow \lambda$$



derivazione: S

Input

a	b	a	b
---	---	---	---



Time 1

$$\lambda, S \rightarrow aSTb$$

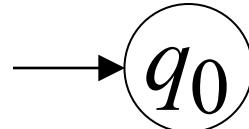
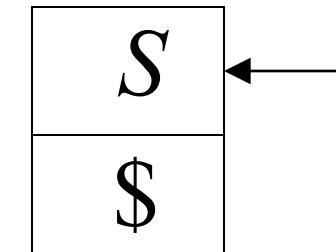
$$\lambda, S \rightarrow b$$

$$\lambda, T \rightarrow Ta$$

$$\lambda, T \rightarrow \lambda$$

$$a, a \rightarrow \lambda$$

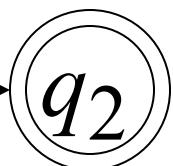
$$b, b \rightarrow \lambda$$



$$\lambda, \lambda \rightarrow S$$

q_1

$$\lambda, \$ \rightarrow \$$$



derivazione: $S \Rightarrow aSTb$

Input

a	b	a	b
---	---	---	---

Time 2

a
S
T
b
\$

Stack

$$\lambda, S \rightarrow aSTb$$

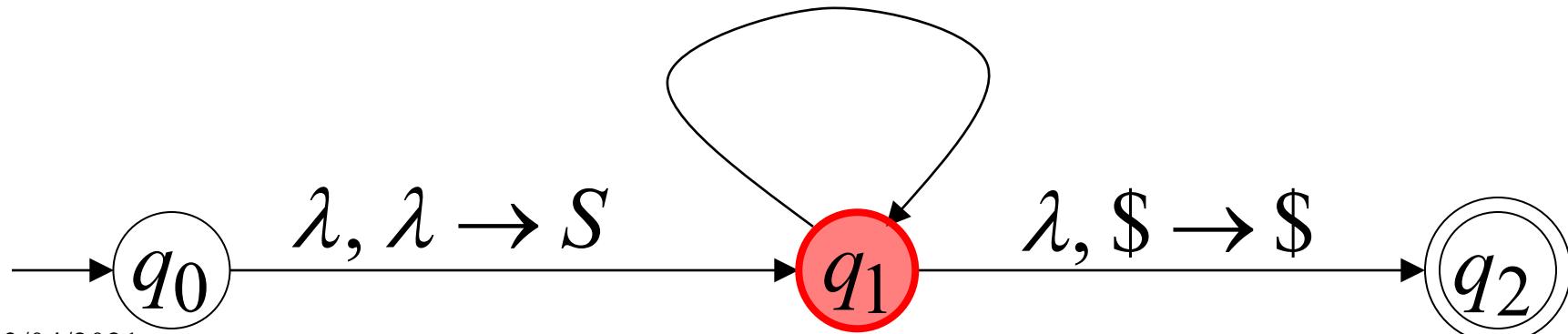
$$\lambda, S \rightarrow b$$

$$\lambda, T \rightarrow Ta$$

$$a, a \rightarrow \lambda$$

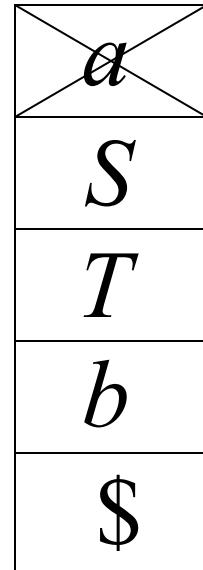
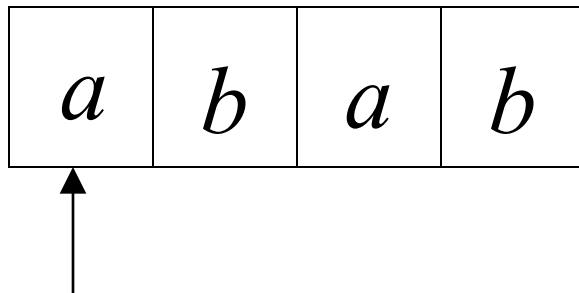
$$\lambda, T \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$



derivazione: $S \Rightarrow aSTb$

Input



Time 3

$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

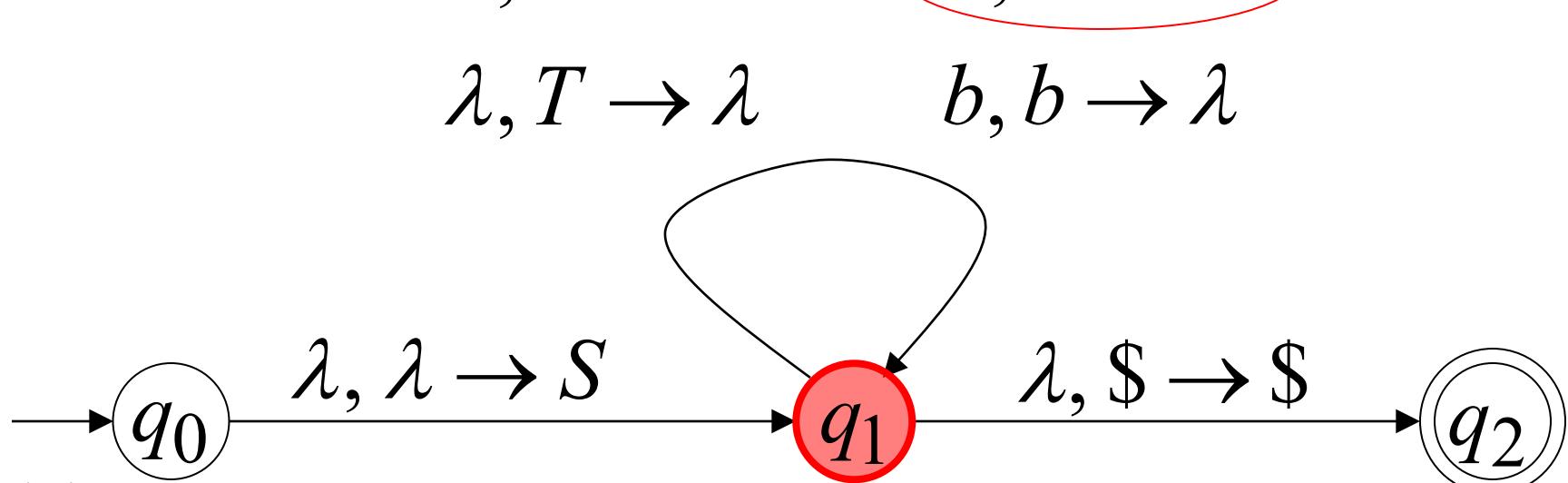
$$\lambda, T \rightarrow Ta$$

$$a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda$$

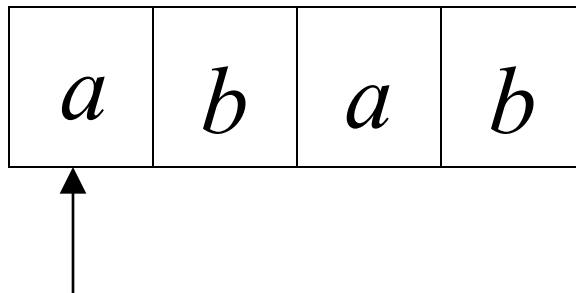
$$b, b \rightarrow \lambda$$

Stack



derivazione: $S \Rightarrow aSTb \Rightarrow abTb$

Input



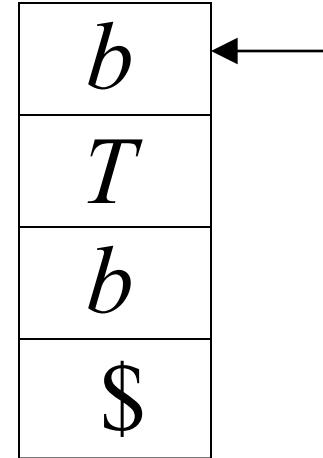
Time 4

$$\lambda, S \rightarrow aSTb$$

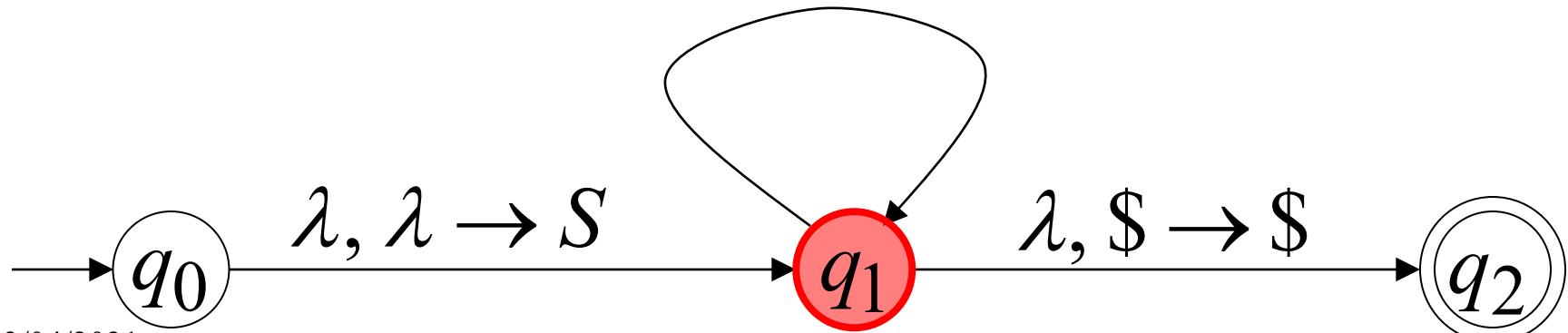
$$\lambda, S \rightarrow b$$

$$\lambda, T \rightarrow Ta \quad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda \quad b, b \rightarrow \lambda$$

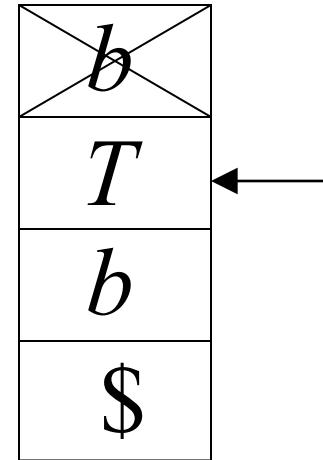
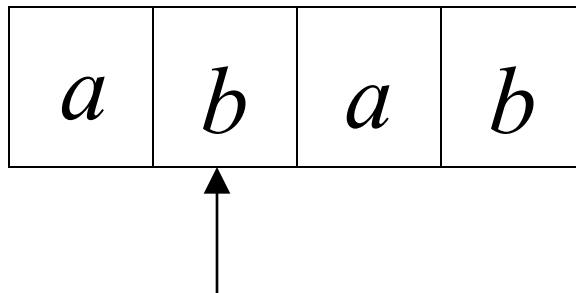


Stack



derivazione: $S \Rightarrow aSTb \Rightarrow abTb$

Input



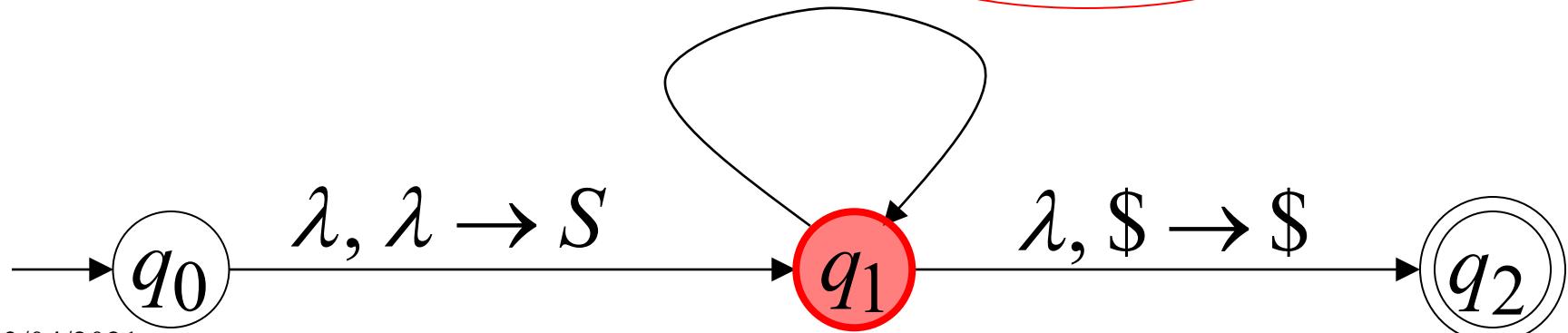
Time 5

$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

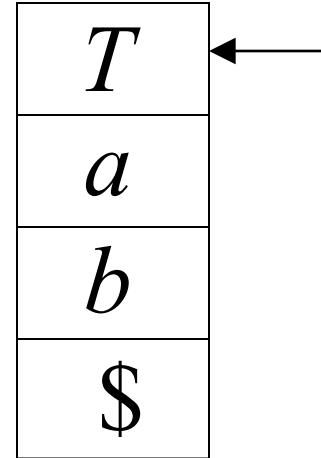
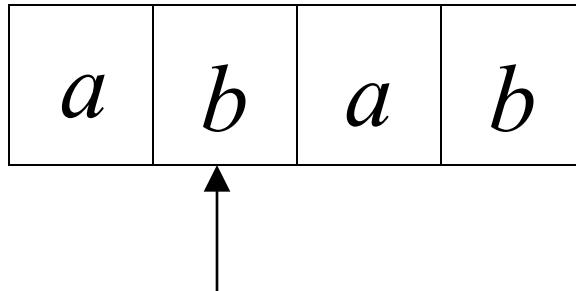
$$\lambda, T \rightarrow Ta \quad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda \quad b, b \rightarrow \lambda$$



derivazione: $S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab$

Input



Time 6

$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

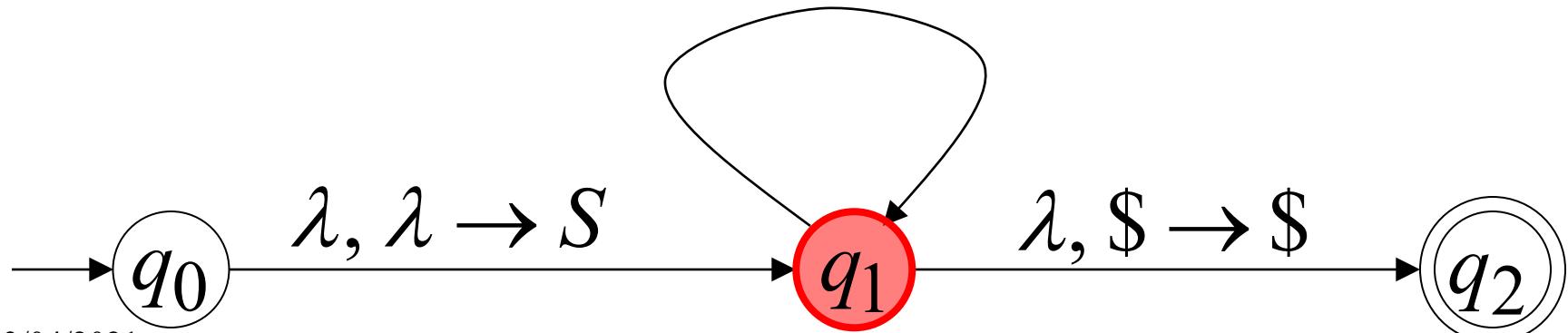
$$\lambda, T \rightarrow Ta$$

$$a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda$$

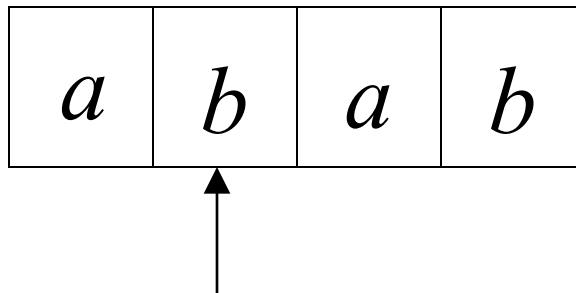
$$b, b \rightarrow \lambda$$

Stack



derivazione: $S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab \Rightarrow abab$

Input



Time 7

$$\lambda, S \rightarrow aSTb$$

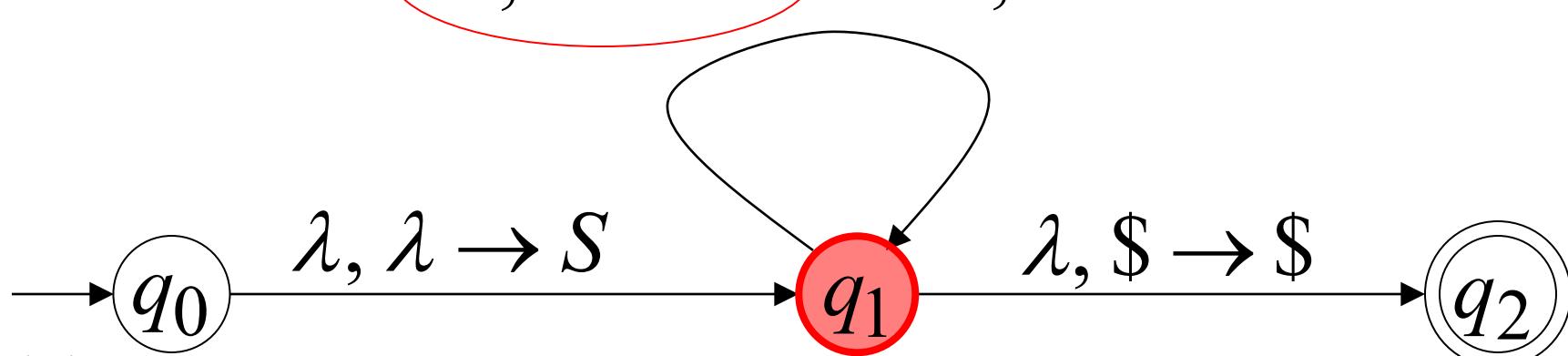
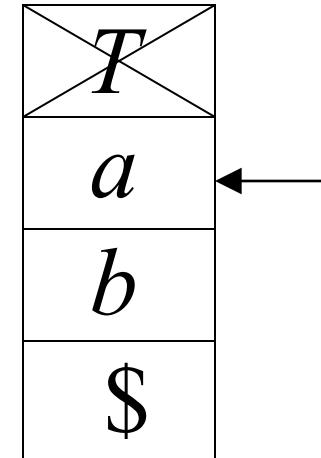
$$\lambda, S \rightarrow b$$

$$\lambda, T \rightarrow Ta \quad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda$$

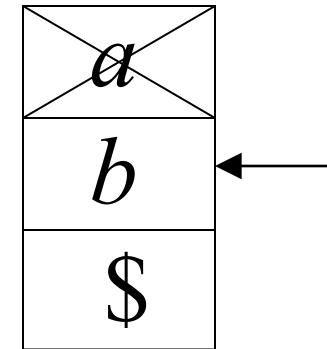
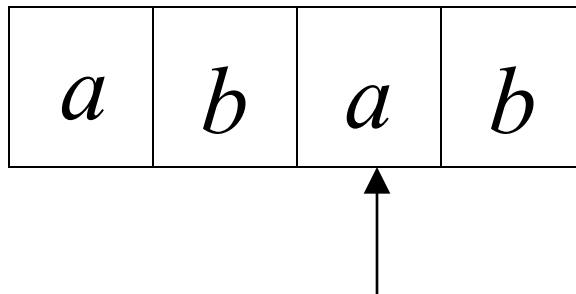
$$b, b \rightarrow \lambda$$

Stack



derivazione: $S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab \Rightarrow abab$

Input



Time 8

$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

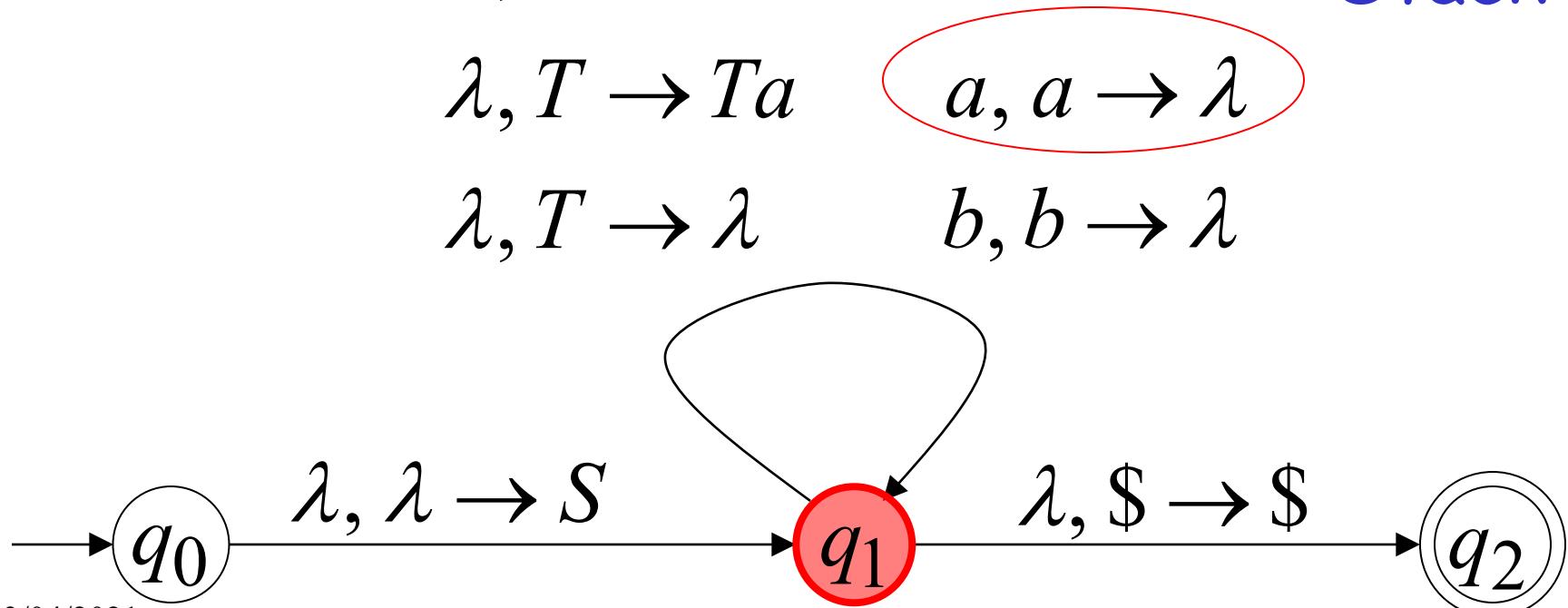
$$\lambda, T \rightarrow Ta$$

$$a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$

Stack



derivazione: $S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab \Rightarrow abab$

Input

a	b	a	b
-----	-----	-----	-----



Time 9

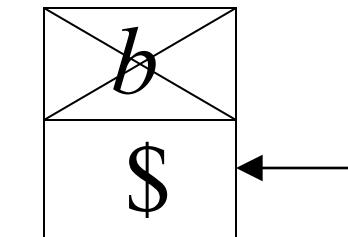
$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

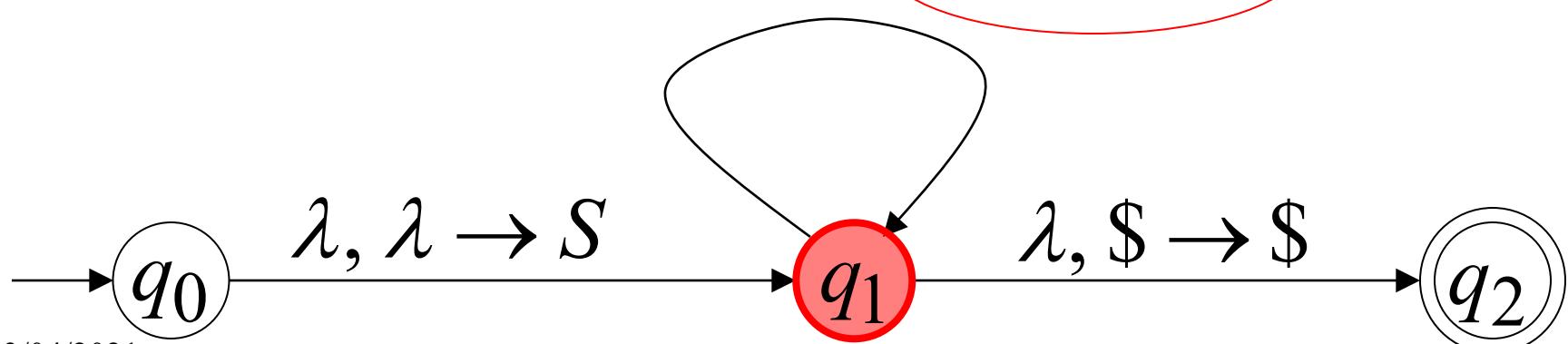
$$\lambda, T \rightarrow Ta \quad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda$$

$$b, b \rightarrow \lambda$$



Stack



derivazione: $S \Rightarrow aSTb \Rightarrow abTb \Rightarrow abTab \Rightarrow abab$

Input

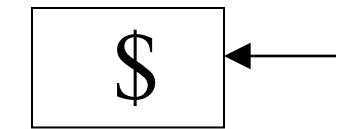
a	b	a	b
-----	-----	-----	-----



$$\lambda, S \rightarrow aSTb$$

Time 10

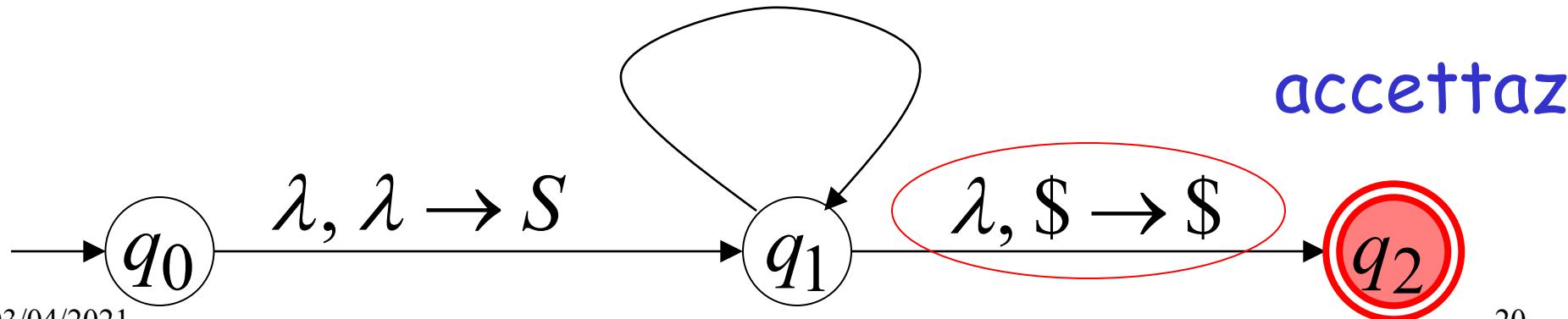
$$\lambda, S \rightarrow b$$



Stack

$$\lambda, T \rightarrow Ta \quad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda \quad b, b \rightarrow \lambda$$



Procedura di conversione:

per ogni produzione in G

per ogni terminale in G

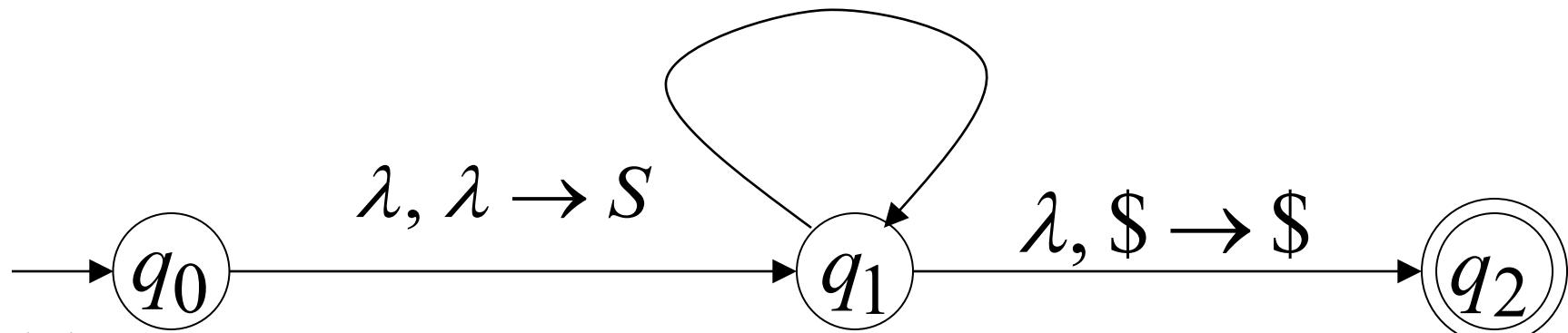
$$A \rightarrow w$$

Addiziona
le transizioni

$$\lambda, A \rightarrow w$$

$$a, a \rightarrow \lambda$$

$$a$$



grammatica

$$S \rightarrow aSTb$$

$$S \rightarrow b$$

$$T \rightarrow Ta$$

$$T \rightarrow \lambda$$

esempio

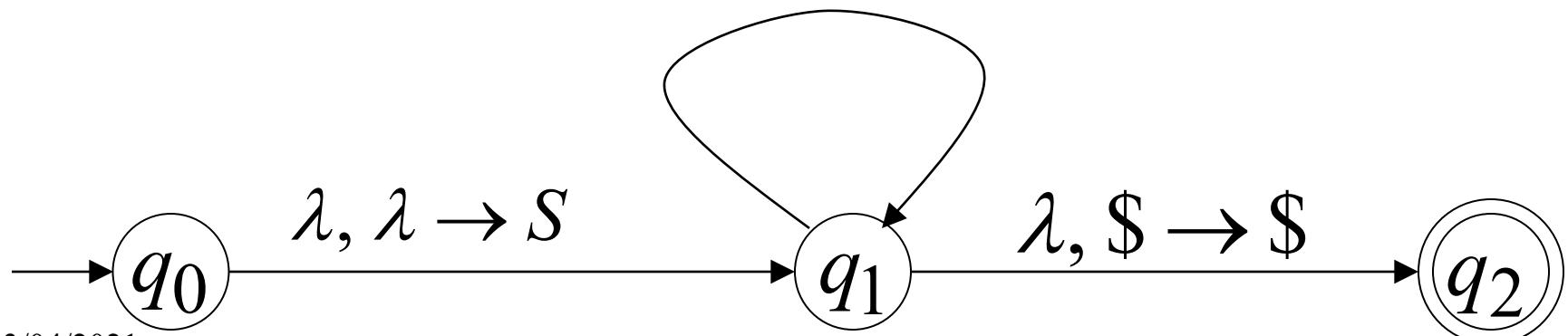
PDA

$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

$$\lambda, T \rightarrow Ta \quad a, a \rightarrow \lambda$$

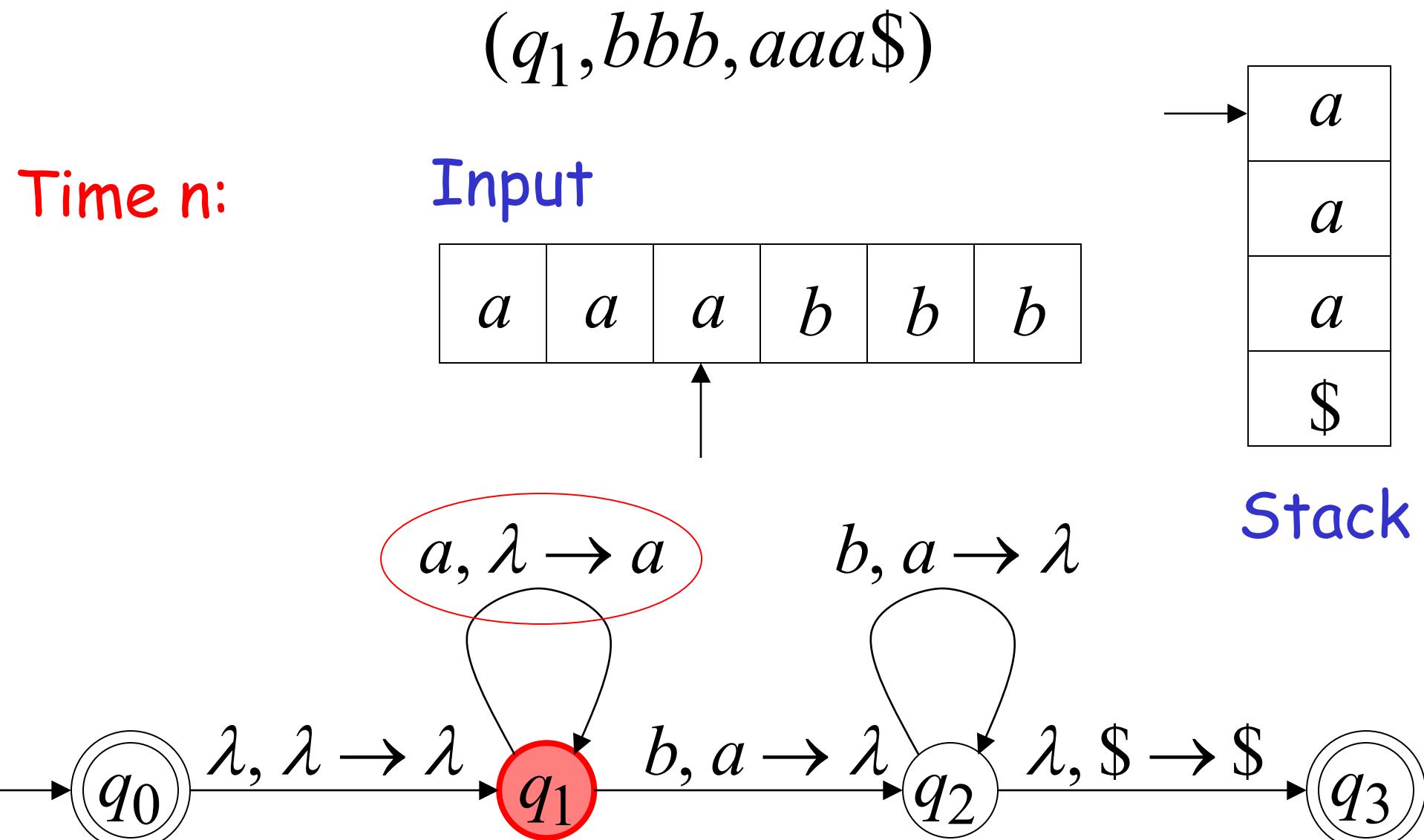
$$\lambda, T \rightarrow \lambda \quad b, b \rightarrow \lambda$$



Proviamo a dimostrare il teorema

Ricordiamo la notazione di configurazione istantanea e di passo di computazione.

configurazione istantanea definizione:

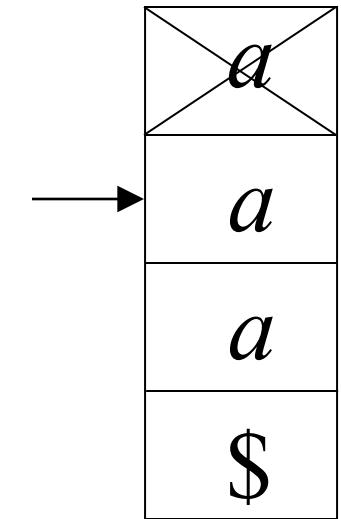
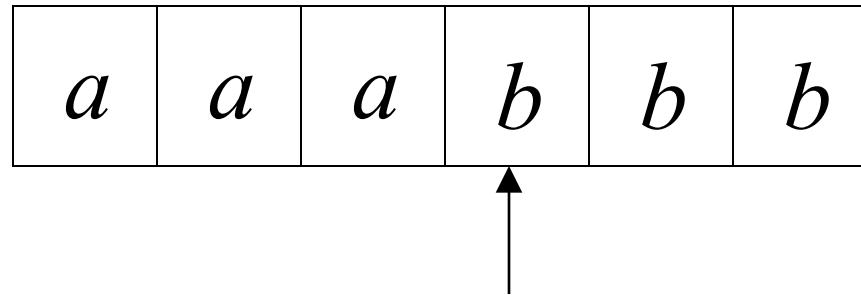


configurazione istantanea

$(q_2, bb, aa\$)$

Time $n+1$:

Input



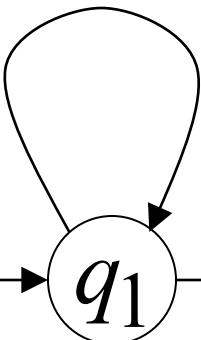
Stack

$a, \lambda \rightarrow a$

$b, a \rightarrow \lambda$



$\lambda, \lambda \rightarrow \lambda$



$b, a \rightarrow \lambda$



$\lambda, \$ \rightarrow \$$



passo di computazione.

$$(q_1, bbb, aaa\$) \succ (q_2, bb, aa\$)$$

Time n

Time n+1

PDA simula le derivazioni leftmost

grammatica
Leftmost derivazione

S

$\Rightarrow \dots$

$\Rightarrow [\sigma_1 \dots \sigma_k] [X_1 \dots X_m]$

$\Rightarrow \dots$

$\Rightarrow \sigma_1 \dots \sigma_k \sigma_{k+1} \dots \sigma_n$

simboli
esaminati

PDA calcolo

$(q_0, \sigma_1 \dots \sigma_k \sigma_{k+1} \dots \sigma_n, \$)$

$\succ (q_1, [\sigma_1 \dots \sigma_k] \sigma_{k+1} \dots \sigma_n, S\$)$

$\succ \dots$

$\succ (q_1, \sigma_{k+1} \dots \sigma_n, [X_1 \dots X_m] \$)$

$\succ \dots$

$\succ (q_2, \lambda, \$)$

Contenuti
Dello Stack

Procedura di conversione:

per ogni produzione in G

per ogni terminale in G

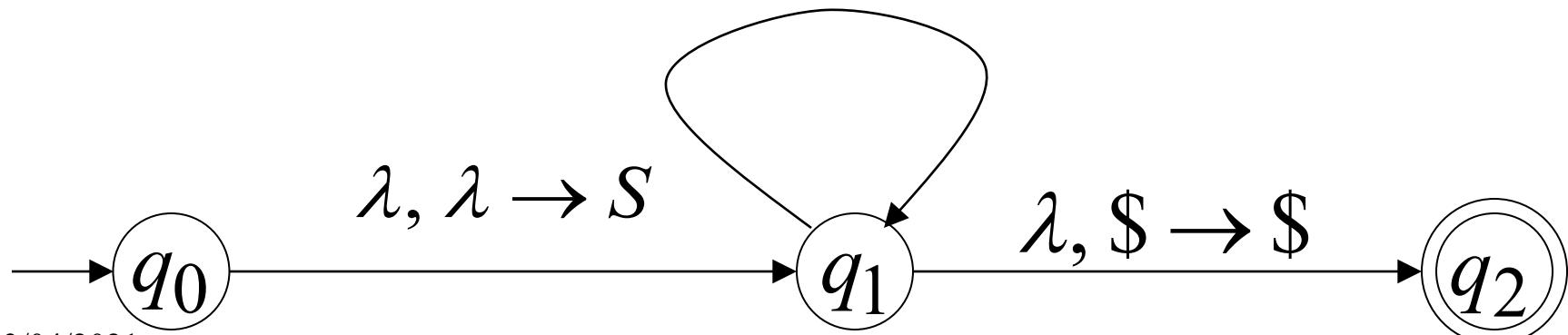
$$A \rightarrow w$$

Addiziona
le transizioni

$$\lambda, A \rightarrow w$$

$$a, a \rightarrow \lambda$$

$$a$$



grammatica

$$S \rightarrow aSTb$$

$$S \rightarrow b$$

$$T \rightarrow Ta$$

$$T \rightarrow \lambda$$

esempio

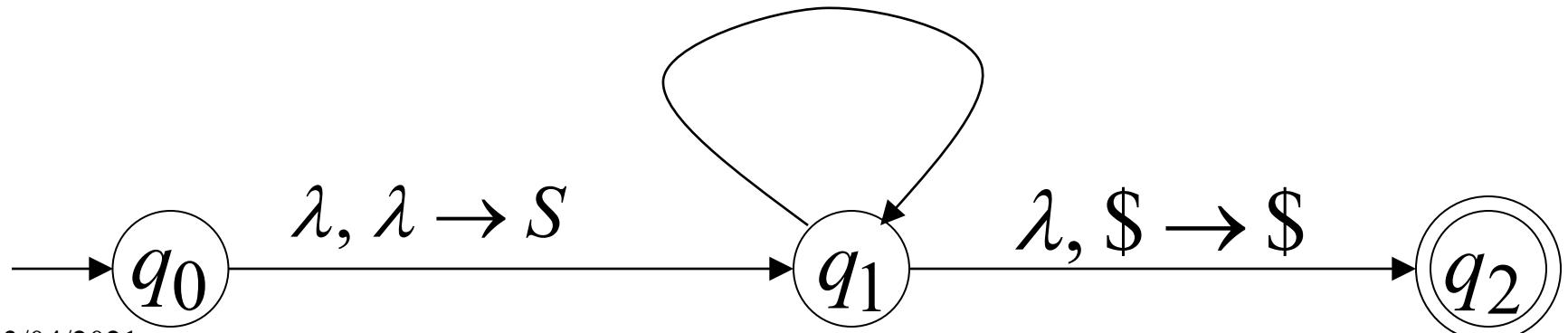
PDA

$$\lambda, S \rightarrow aSTb$$

$$\lambda, S \rightarrow b$$

$$\lambda, T \rightarrow Ta \quad a, a \rightarrow \lambda$$

$$\lambda, T \rightarrow \lambda \quad b, b \rightarrow \lambda$$



grammatica

Derivazione Leftmost

Calcolo PDA

S

$\Rightarrow aSTb$

$\Rightarrow abTb$

$\Rightarrow abTab$

$\Rightarrow abab$

$(q_0, abab, \$)$

$\succ (q_1, abab, S\$)$

$\succ (q_1, bab, STb\$)$

$\succ (q_1, bab, bTb\$)$

$\succ (q_1, ab, Tb\$)$

$\succ (q_1, ab, Tab\$)$

$\succ (q_1, ab, ab\$)$

$\succ (q_1, b, b\$)$

$\succ (q_1, \lambda, \$)$

$\succ (q_2, \lambda, \$)$

grammatica

Leftmost derivazione

$\Rightarrow \dots$

$\Rightarrow xAy$

$\Rightarrow \sigma_i \dots \sigma_j Bzy$

calcolo del PDA

$\gamma \dots$

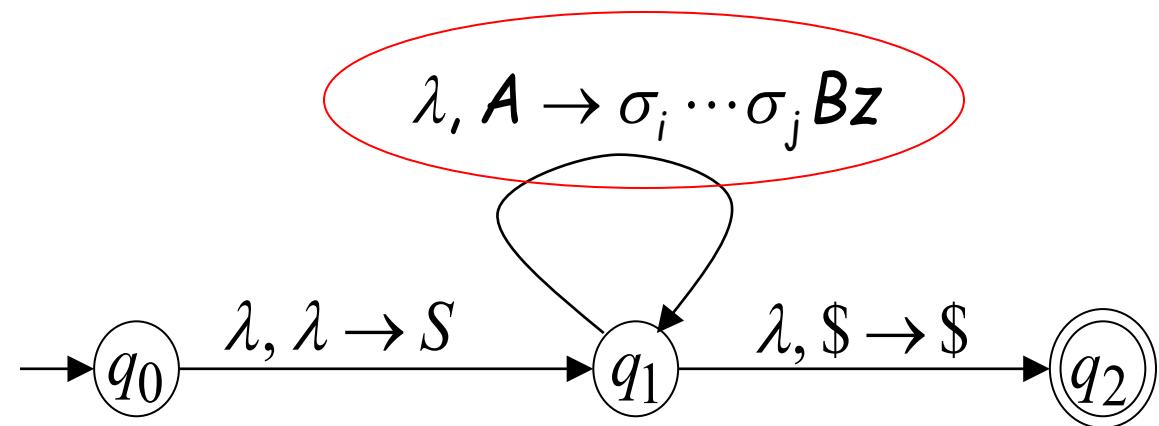
$\succ (q_1, \sigma_i \dots \sigma_n, Ay \$)$

$\succ (q_1, \sigma_i \dots \sigma_n, \sigma_i \dots \sigma_j Bzy \$)$

Produzione Applicata

$A \rightarrow \sigma_i \dots \sigma_j Bz$

Transizione applicata



grammatica

Leftmost derivazione

$\Rightarrow \dots$

$\Rightarrow xAy$

$\Rightarrow \sigma_i \dots \sigma_j Bzy$

$\succ \dots$

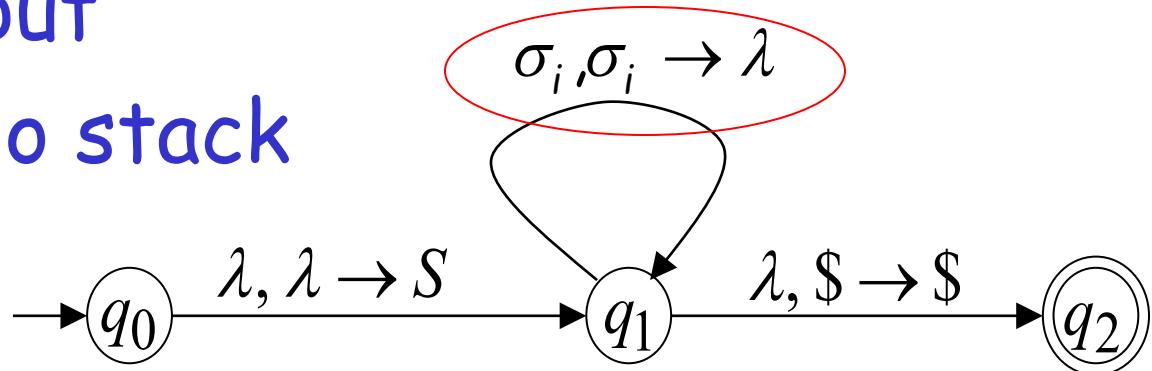
$\succ (q_1, \sigma_i \dots \sigma_n, Ay \$)$

$\succ (q_1, \sigma_i \dots \sigma_n, \sigma_i \dots \sigma_j Bzy \$)$

$\succ (q_1, \sigma_{i+1} \dots \sigma_n, \sigma_{i+1} \dots \sigma_j Bzy \$)$

leggi σ_i dall'input
E rimuovilo dallo stack

Transizione applicata



grammatica

Leftmost derivazione

$\Rightarrow \dots$

$\Rightarrow xAy$

$\Rightarrow \sigma_i \dots \sigma_j Bzy$

tutti simboli $\sigma_i \dots \sigma_j$

Sono stati rimossi

Dal top dello stack

PDA calcolo

$\succ \dots$

$\succ (q_1, \sigma_i \dots \sigma_n, Ay \$)$

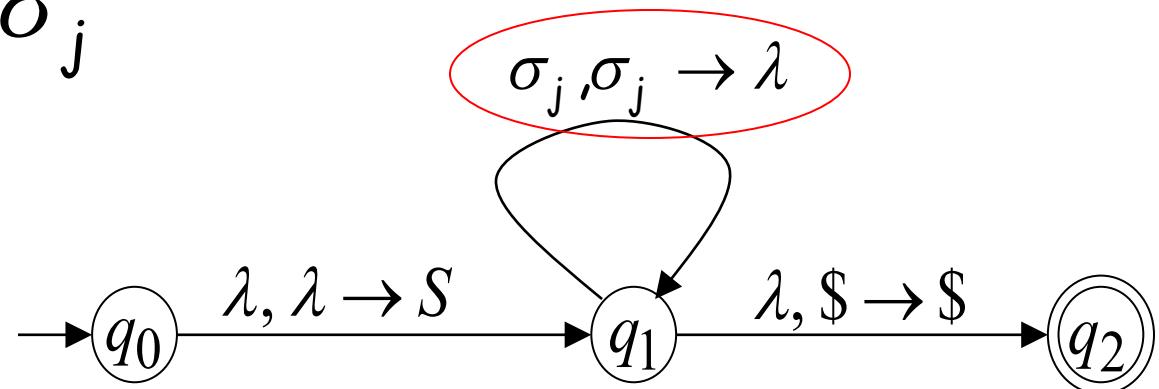
$\succ (q_1, \sigma_i \dots \sigma_n, \sigma_i \dots \sigma_j Bzy \$)$

$\succ (q_1, \sigma_{i+1} \dots \sigma_n, \sigma_{i+1} \dots \sigma_j Bzy \$)$

$\succ \dots$

$\succ (q_1, \sigma_{j+1} \dots \sigma_n, Bzy \$)$

ultima Transizione applicata



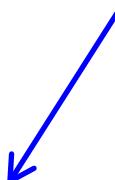
Il processo viene ripetuto con successiva
variabile leftmost

$\Rightarrow \dots$

$\Rightarrow xAy$

$\Rightarrow \sigma_i \dots \sigma_j Bzy$

$\Rightarrow \sigma_i \dots \sigma_j \sigma_{j+1} \dots \sigma_k Cpzy$



$\succ \dots$

$\succ (q_1, \sigma_{j+1} \dots \sigma_n, Bzy \$)$

$\succ (q_1, \sigma_{j+1} \dots \sigma_n, \sigma_{j+1} \dots \sigma_k Cpzy \$)$

$\succ \dots$

$\succ (q_1, \sigma_{k+1} \dots \sigma_n, Cpzy \$)$

Produzione applicata

$B \rightarrow \sigma_{j+1} \dots \sigma_k Cp$

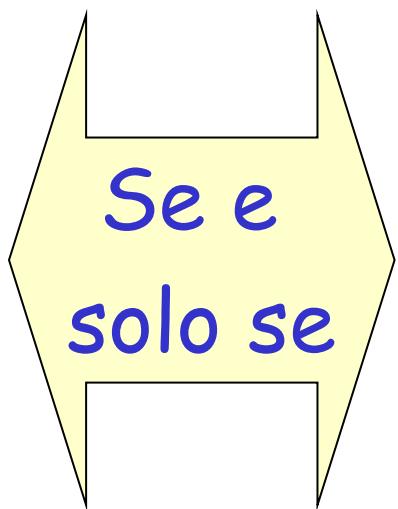
E così via

dimostrato che:

grammatica G

Genera la
stringa w

$S \xrightarrow{*} w$



PDA M
accetta w

$(q_0, w, \$) \succ (q_2, \lambda, \$)$

quindi

$L(G) = L(M)$

Proof - step 2

Tradurre
i PDA
in
grammatiche Context-Free

Prendi un qualsiasi PDA M

Traduremmo M

In una grammatica G context-free

tale che: $L(M) = L(G)$

Prima di tutto modifica PDA M tale che:

1. PDA ha un solo stato di accettazione
2. Svuota tutta la pila prima di accettare
3. Per ogni transizione o
 - push un simbolo
 - oppure
 - pop un simbolo

ma non entrambe le cose

Inoltre abbiamo:

- - nuovo simbolo iniziale @
- - stato di accettazione nello stack solo @

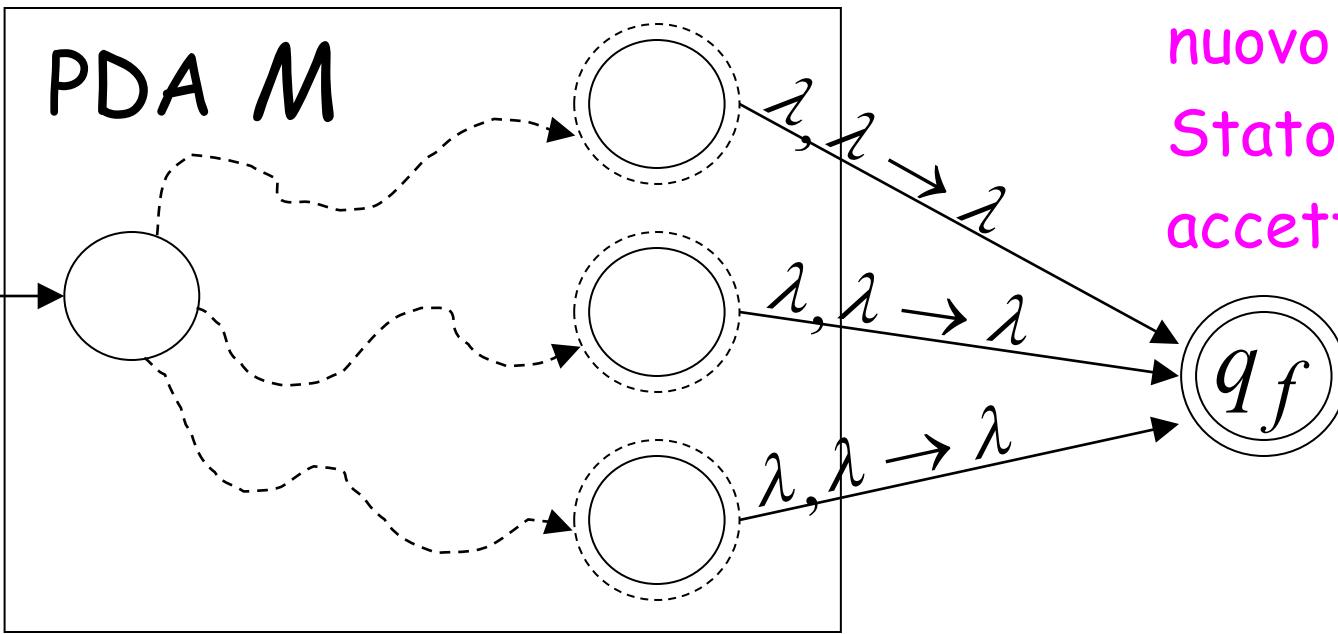
1. il PDA deve avere un solo stato di accettazione

PDA M_1

vecchi
Stati di
accettazione

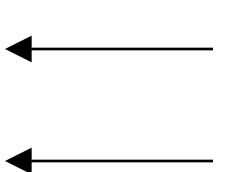
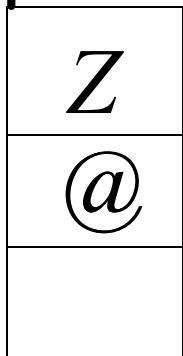
PDA M

nuovo
Stato di
accettazione



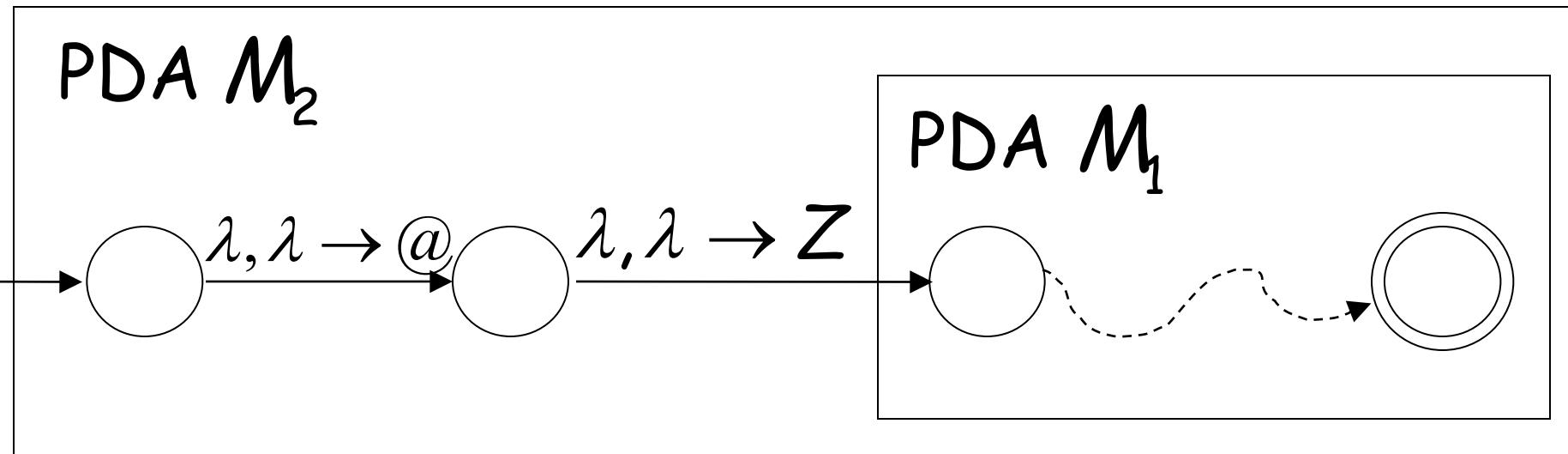
2. Nuovo simbolo iniziale dello stack

Top of stack



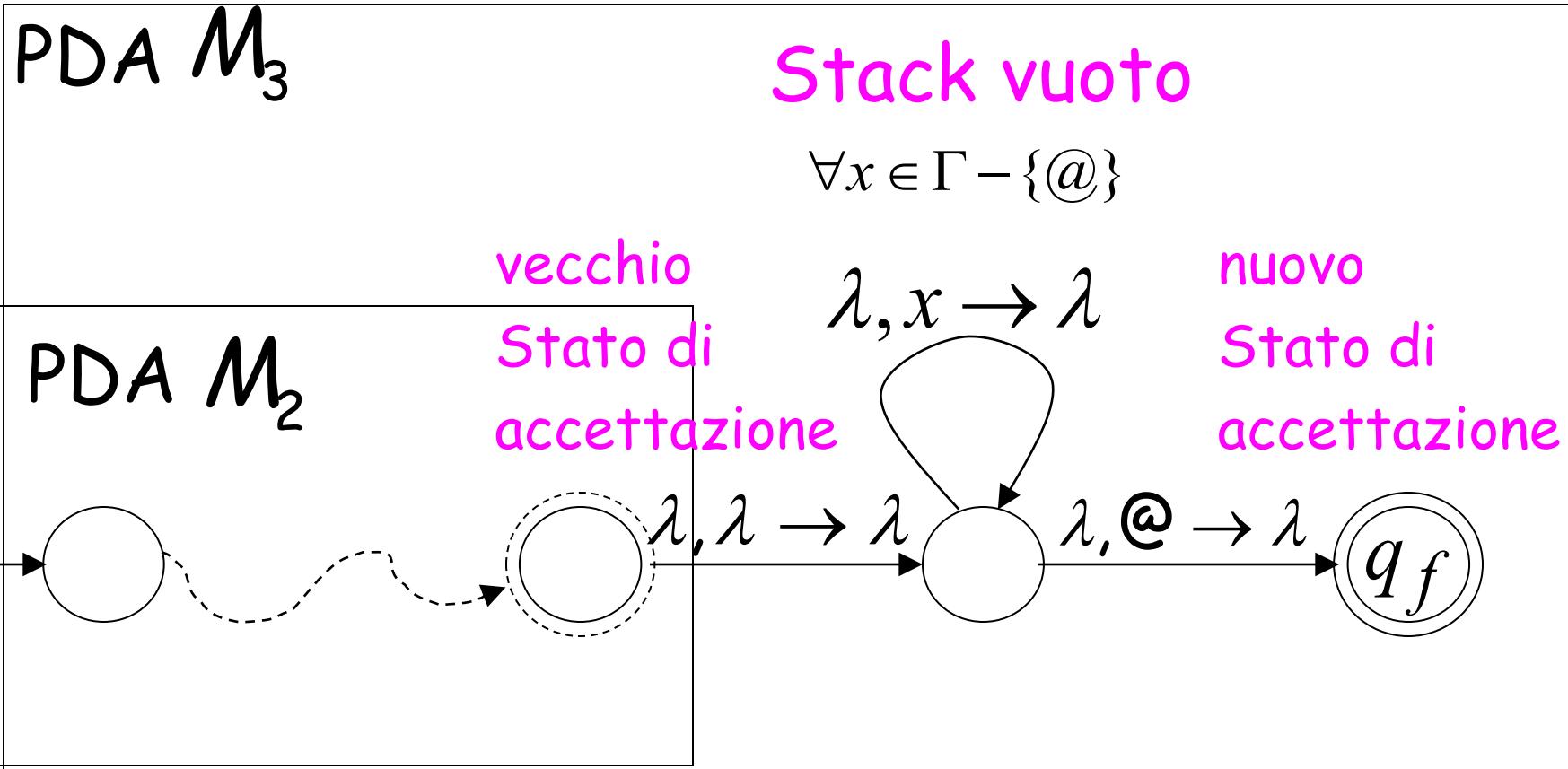
Vecchio simbolo iniziale

Simbolo ausiliario stack



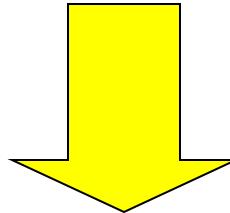
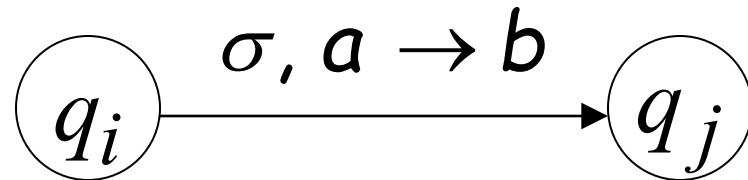
M_1 pensa ancora che Z è il simbolo iniziale

3. Nello stato di accettazione
lo stack contiene
solo il simbolo @

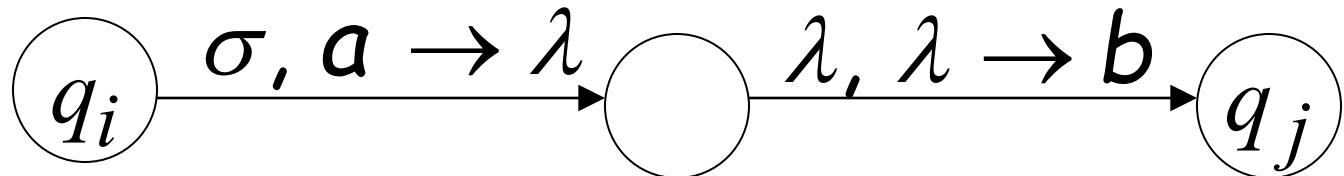


4. Ogni transizione ha un push oppure un pop
Mai le due cose insieme.

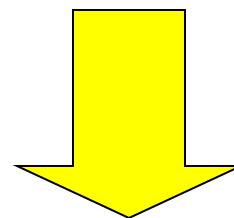
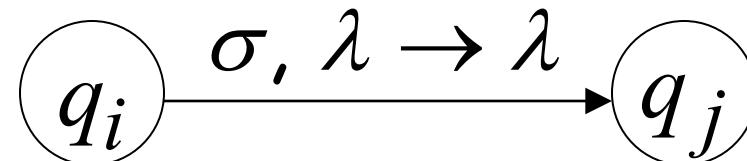
PDA M_3



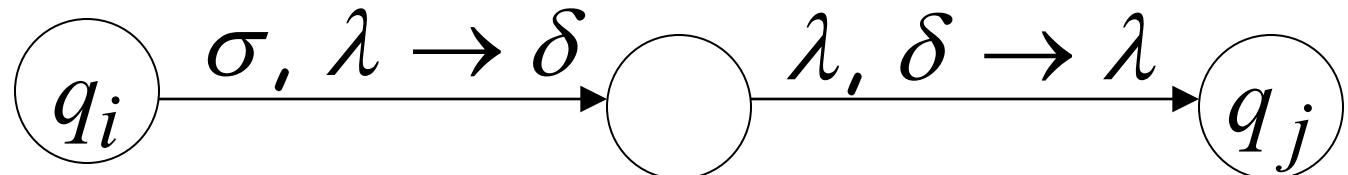
PDA M_4



PDA M_3



PDA M_4

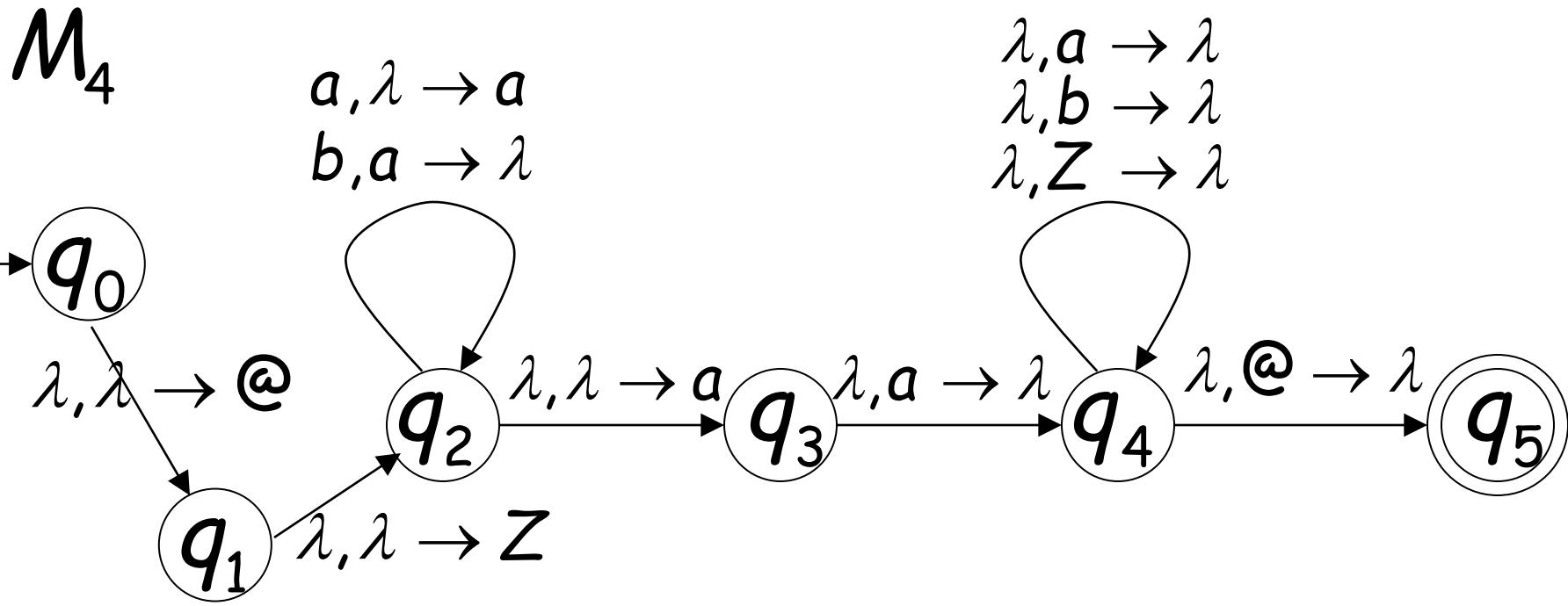
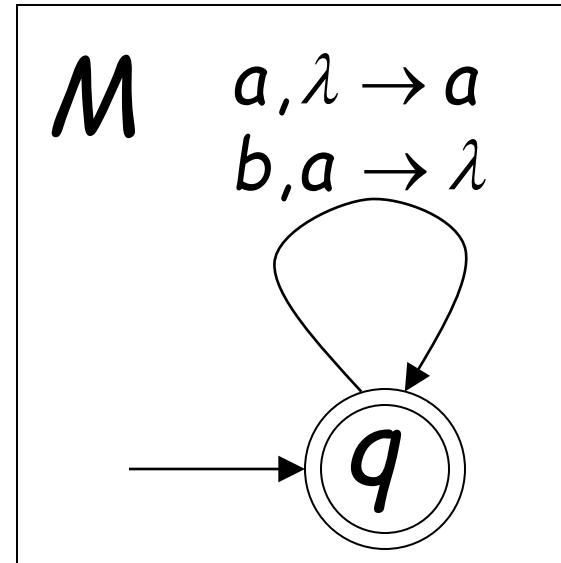


dove δ è un qualsiasi simbolo
dell'alfabeto di input

PDA M_4 è il PDA completamente modificato secondo le regole precedenti

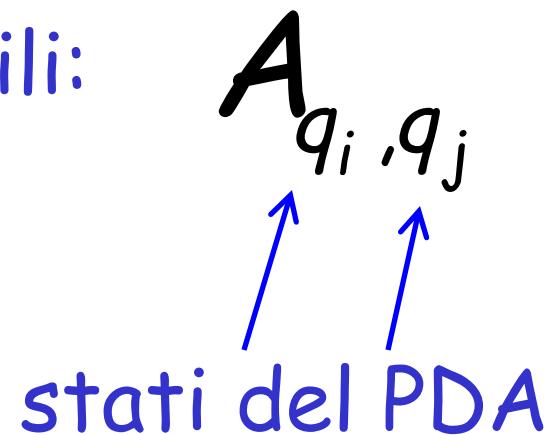
Notiamo che il nuovo simbolo iniziale @ non è mai usato in nessuna transizione

Esempio:



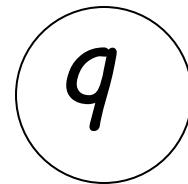
Costruzione della grammatica

variabili:



PDA

caso 1: per ogni stato

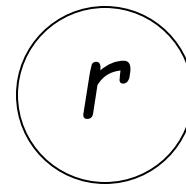
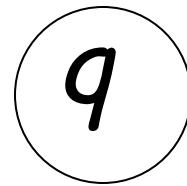
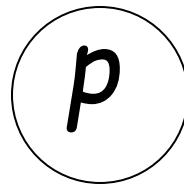


grammatica

$$A_{qq} \rightarrow \lambda$$

PDA

caso 2: per ogni tre stati

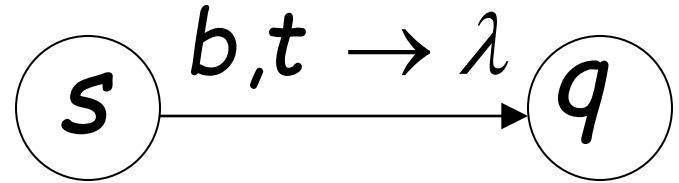
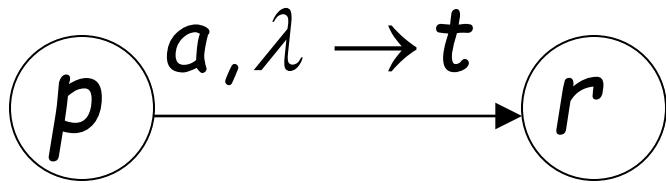


grammatica

$$A_{pq} \rightarrow A_{pr} A_{rq}$$

PDA

caso 3: per ogni coppia di transizioni

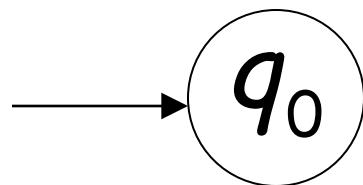


grammatica

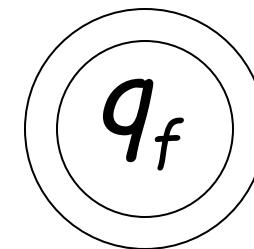
$$A_{pq} \rightarrow a A_{rs} b$$

PDA

Stato Iniziale



Stato accettazione



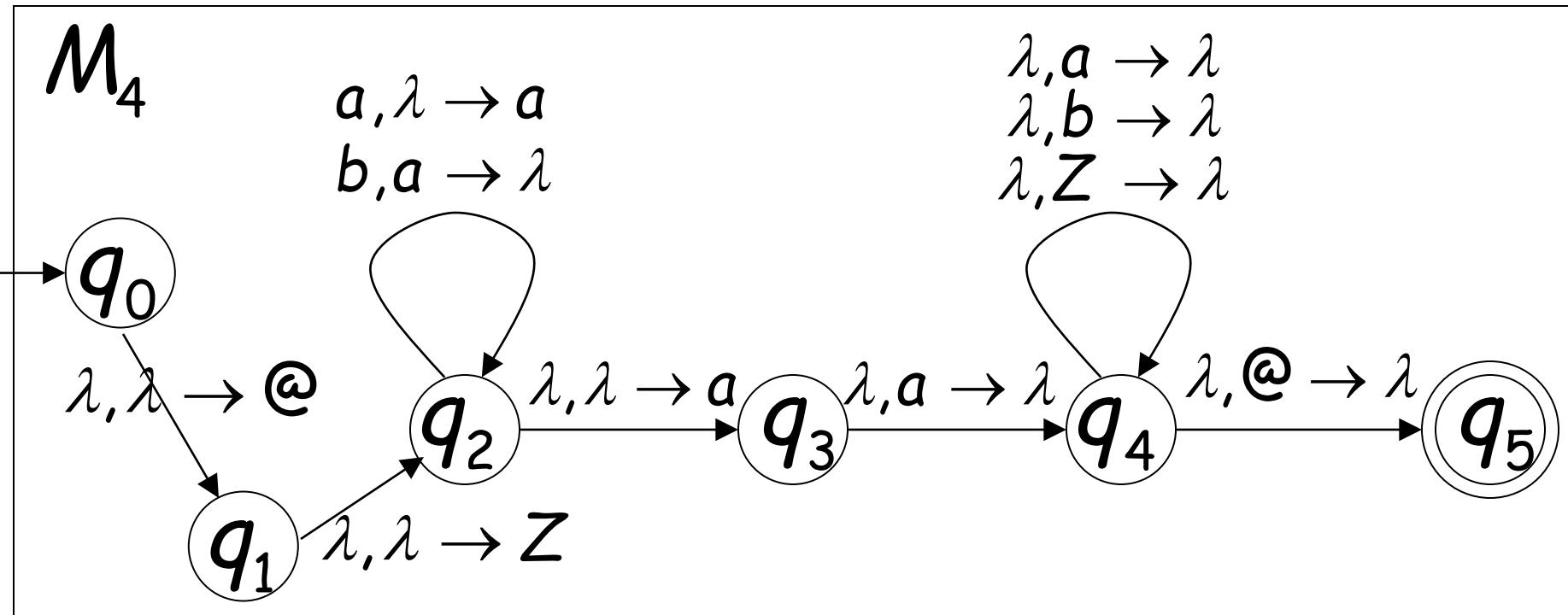
grammatica

Variabile Start

$A_{q_0 q_f}$

Esempio:

PDA



grammatica

caso 1: da stati singoli

$$A_{q_0q_0} \rightarrow \lambda$$

$$A_{q_1q_1} \rightarrow \lambda$$

$$A_{q_2q_2} \rightarrow \lambda$$

$$A_{q_3q_3} \rightarrow \lambda$$

$$A_{q_4q_4} \rightarrow \lambda$$

$$A_{q_5q_5} \rightarrow \lambda$$

caso 2: da triple di stati

$A_{q_0q_0} \rightarrow A_{q_0q_0} A_{q_0q_0} \mid A_{q_0q_1} A_{q_1q_0} \mid A_{q_0q_2} A_{q_2q_0} \mid A_{q_0q_3} A_{q_3q_0} \mid A_{q_0q_4} A_{q_4q_0} \mid A_{q_0q_5} A_{q_5q_0}$

$A_{q_0q_1} \rightarrow A_{q_0q_0} A_{q_0q_1} \mid A_{q_0q_1} A_{q_1q_1} \mid A_{q_0q_2} A_{q_2q_1} \mid A_{q_0q_3} A_{q_3q_1} \mid A_{q_0q_4} A_{q_4q_1} \mid A_{q_0q_5} A_{q_5q_1}$

⋮

$A_{q_0q_5} \rightarrow A_{q_0q_0} A_{q_0q_5} \mid A_{q_0q_1} A_{q_1q_5} \mid A_{q_0q_2} A_{q_2q_5} \mid A_{q_0q_3} A_{q_3q_5} \mid A_{q_0q_4} A_{q_4q_5} \mid A_{q_0q_5} A_{q_5q_5}$

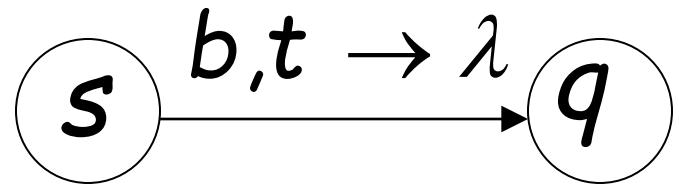
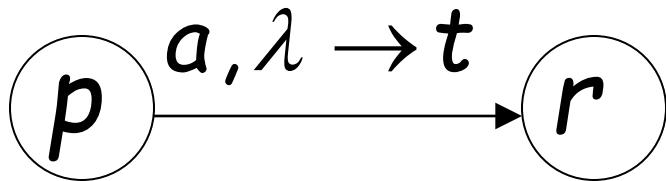
⋮

$A_{q_5q_5} \rightarrow A_{q_5q_0} A_{q_0q_5} \mid A_{q_5q_1} A_{q_1q_5} \mid A_{q_5q_2} A_{q_2q_5} \mid A_{q_5q_3} A_{q_3q_5} \mid A_{q_5q_4} A_{q_4q_5} \mid A_{q_5q_5} A_{q_5q_5}$

Variabile Start $A_{q_0q_5}$

PDA

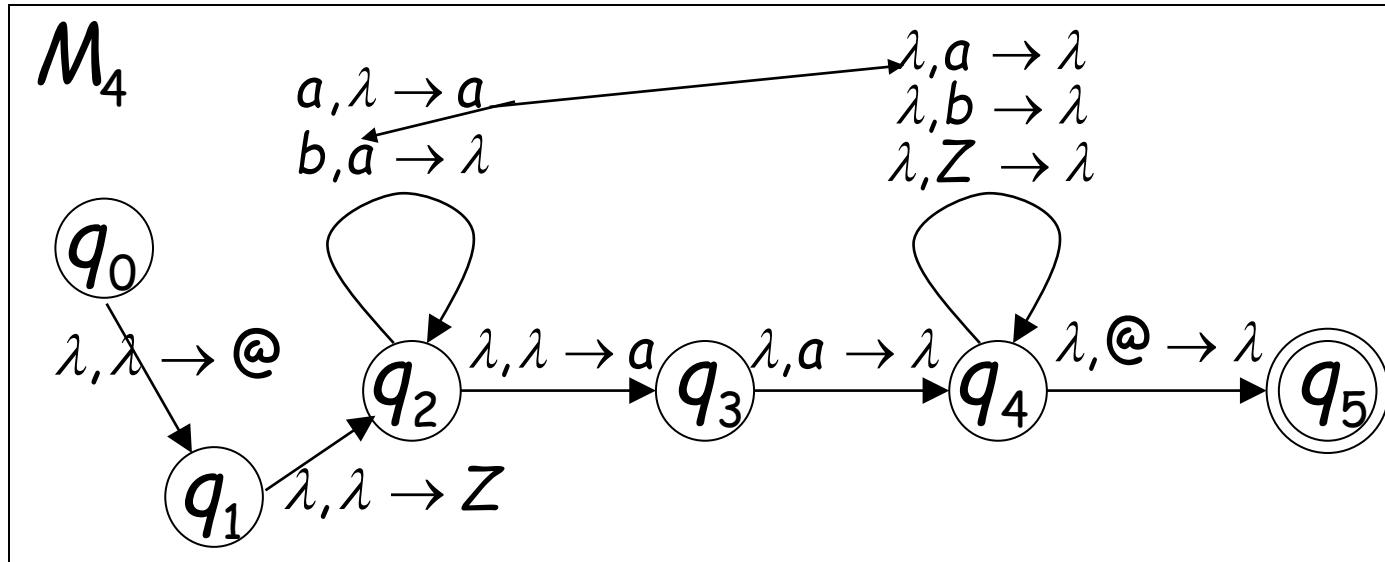
caso 3: per ogni coppia di transizioni



grammatica

$$A_{pq} \rightarrow a A_{rs} b$$

caso 3: da coppie di transizioni



$$\begin{array}{l}
 A_{q_0 q_5} \rightarrow A_{q_1 q_4} \quad A_{q_2 q_4} \rightarrow a A_{q_2 q_4} \quad A_{q_2 q_2} \rightarrow A_{q_3 q_2} b \\
 A_{q_1 q_4} \rightarrow A_{q_2 q_4} \quad A_{q_2 q_2} \rightarrow a A_{q_2 q_2} b \quad A_{q_2 q_4} \rightarrow A_{q_3 q_3} \\
 \quad \quad \quad A_{q_2 q_4} \rightarrow a A_{q_2 q_3} \quad A_{q_2 q_4} \rightarrow A_{q_3 q_4}
 \end{array}$$

Supponiamo che il PDA M è stato tradotto
In una grammatica context-free G

Dobbiamo provare $L(G) = L(M)$

O in modo equivalente

$$L(G) \subseteq L(M)$$

$$L(G) \supseteq L(M)$$

$$L(G) \subseteq L(M)$$

Dobbiamo mostrare che se G ha una derivazione:

$$A_{q_0 q_f}^* \Rightarrow^* W \quad (\text{stringa di terminali})$$

Allora vi è un calcolo in M che accetta W :

$$(q_0, w, @) \xrightarrow{*} (q_f, \lambda, @)$$

partiamo con una p e una q qualsiasi.

Se in G vi è una derivazione:

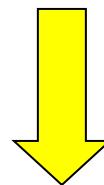
$$A_{pq} \xrightarrow{*} W$$

Allora vi è un calcolo in M :

$$(p, w, \lambda) \xrightarrow{*} (q, \lambda, \lambda)$$

Dopo aver provato
il passo precedente
abbiamo:

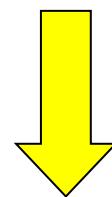
$$A_{q_0 q_f} \xrightarrow{*} W$$



$$(q_0, w, \lambda) \xrightarrow{*} (q_f, \lambda, \lambda)$$

Poichè non c'è nessuna
transizione

Con il simbolo @



$$(q_0, w, @) \xrightarrow{*} (q_f, \lambda, @)$$

Lemma:

se $A_{pq} \xrightarrow{*} w$ (stringa di terminali)

Allora vi è un calcolo
dallo stato p allo stato q
sulla stringa w

Che lascia lo stack vuoto:

$(p, w, \lambda) \xrightarrow{*} (q, \lambda, \lambda)$

Dim intuitiva:

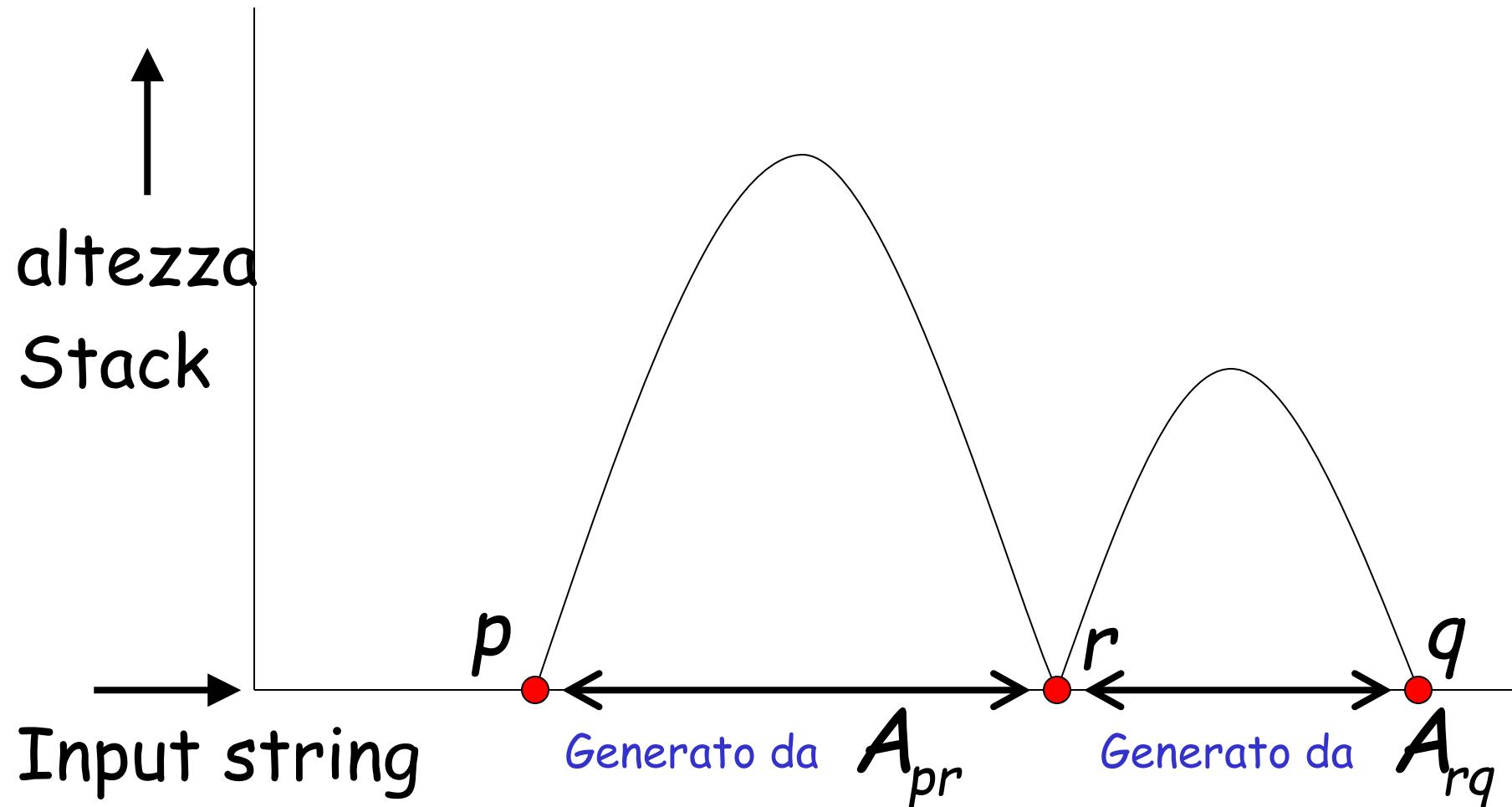
$$A_{pq} \Rightarrow \cdots \Rightarrow W$$

Case 1: $A_{pq} \Rightarrow A_{pr} A_{rq} \Rightarrow \cdots \Rightarrow W$

Case 2: $A_{pq} \Rightarrow a A_{rs} b \Rightarrow \cdots \Rightarrow W$

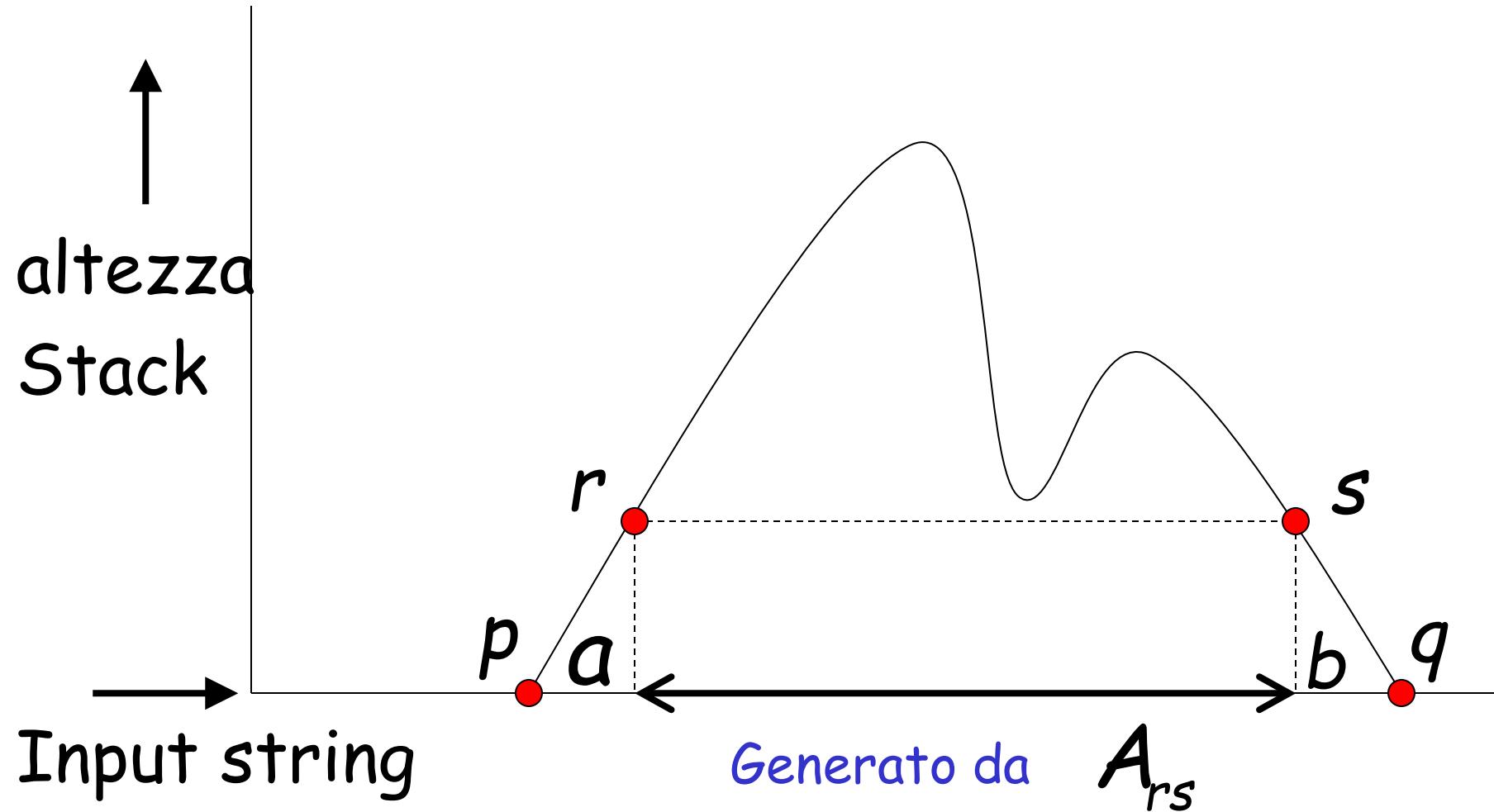
Type 2

Case 1: $A_{pq} \Rightarrow A_{pr} A_{rq} \Rightarrow \dots \Rightarrow W$



Type 3

Case 2: $A_{pq} \Rightarrow aA_{rs}b \Rightarrow \dots \Rightarrow W$



Formale :

Proviamo l'asserto per induzione

Sul numero di step della derivazione:

$$A_{pq} \Rightarrow \cdots \Rightarrow W$$

Numero di step

base:

$$A_{pq} \Rightarrow W$$

(Uno step di derivazione)

caso 1 produzione che deve essere usata è:

$$A_{pp} \rightarrow \lambda$$

Quindi $p = q$ e $w = \lambda$

Questo calcolo nel PDA esiste (banale):

$$(p, \lambda, \lambda) \xrightarrow{*} (p, \lambda, \lambda)$$

Ipotesi induttiva:

$$A_{pq} \Rightarrow \cdots \Rightarrow W$$

con k Step di derivazione

Quindi abbiamo, per ipotesi induttiva:

$$(p, w, \lambda) \xrightarrow{*} (q, \lambda, \lambda)$$

Step induttivo

Data:

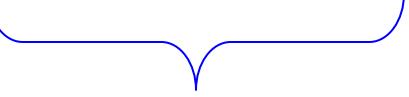
$$A_{pq} \Rightarrow \cdots \Rightarrow W$$

con $k + 1$ step
di derivazione

Dobbiamo provare:

$$(p, w, \lambda) \xrightarrow{*} (q, \lambda, \lambda)$$

$$A_{pq} \Rightarrow \cdots \Rightarrow W$$



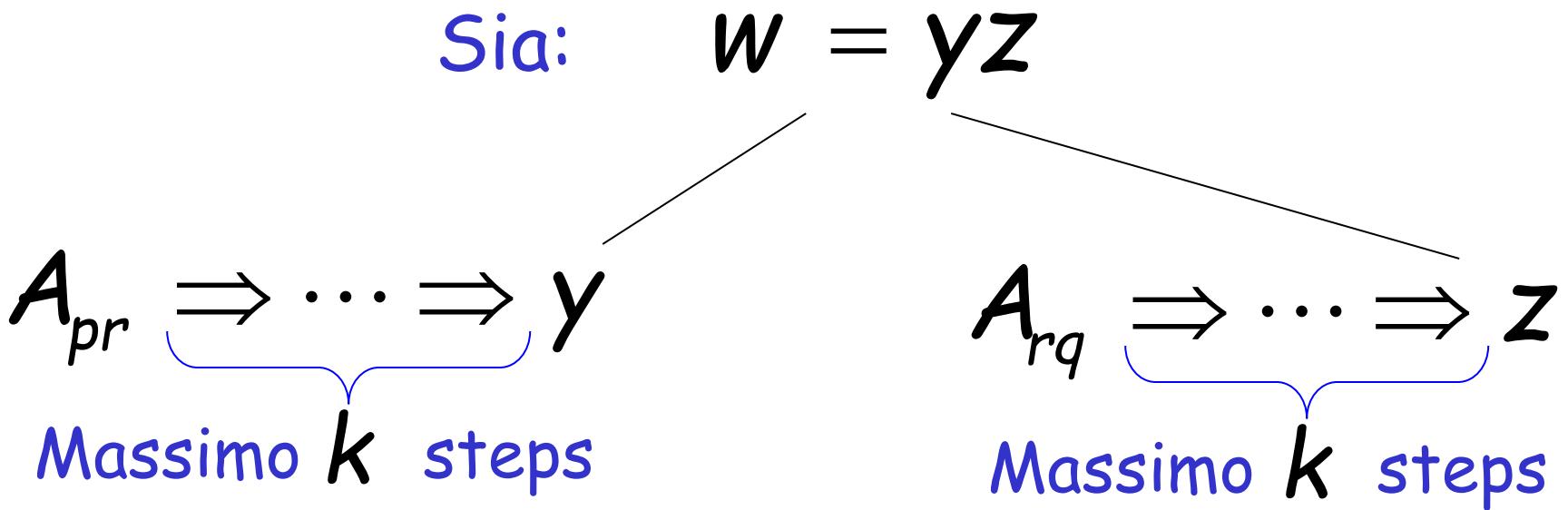
$k + 1$ derivazione steps

Case 1: $A_{pq} \Rightarrow A_{pr} A_{rq} \Rightarrow \cdots \Rightarrow W$

Case 2: $A_{pq} \Rightarrow a A_{rs} b \Rightarrow \cdots \Rightarrow W$

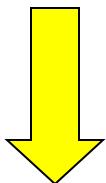
Case 1: $A_{pq} \Rightarrow A_{pr} A_{rq} \Rightarrow \dots \Rightarrow w$

$k + 1$ steps



$$A_{pr} \Rightarrow \cdots \Rightarrow y$$

massimo k steps

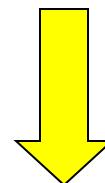


Per ipotesi induttiva
in PDA:

$$(p, y, \lambda) \xrightarrow{*} (r, \lambda, \lambda)$$

$$A_{rq} \Rightarrow \cdots \Rightarrow z$$

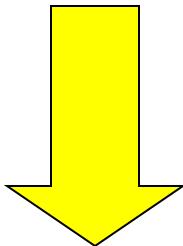
massimo k steps



Per ipotesi induttiva,
in PDA:

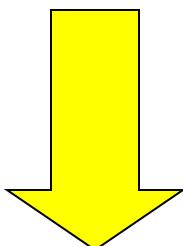
$$(r, z, \lambda) \xrightarrow{*} (q, \lambda, \lambda)$$

$$(p, y, \lambda) \stackrel{*}{\succ} (r, \lambda, \lambda) \quad (r, z, \lambda) \stackrel{*}{\succ} (q, \lambda, \lambda)$$



$$(p, yz, \lambda) \stackrel{*}{\succ} (r, z, \lambda) \stackrel{*}{\succ} (q, \lambda, \lambda)$$

poichè $w = yz$



$$(p, w, \lambda) \stackrel{*}{\succ} (q, \lambda, \lambda)$$

Case 2: $A_{pq} \Rightarrow aA_{rs}b \Rightarrow \dots \Rightarrow w$

$k + 1$ steps

Possiamo scrivere

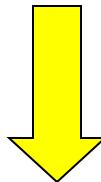
$$w = ayb$$

$A_{rs} \Rightarrow \dots \Rightarrow y$

Massimo k steps

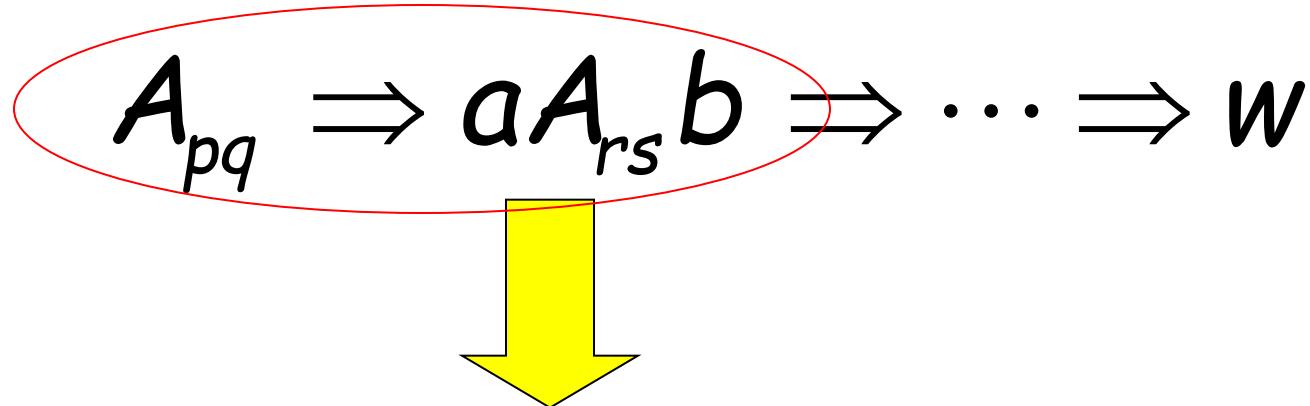
$$A_{rs} \Rightarrow \cdots \Rightarrow y$$

Massimo k steps



Per ipotesi induttiva ,
il PDA calcola:

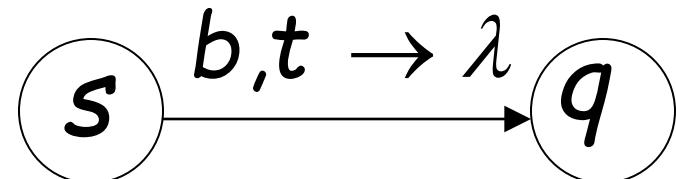
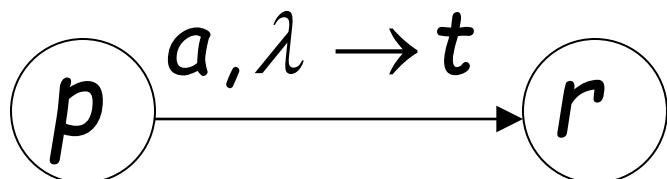
$$(r, y, \lambda) \xrightarrow{*} (s, \lambda, \lambda)$$

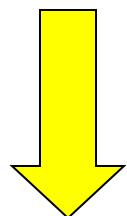
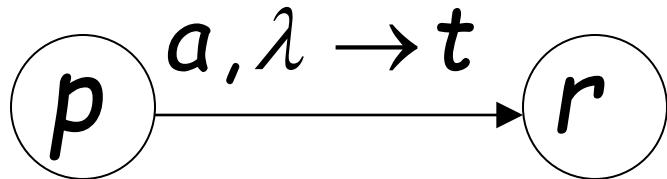


La grammatica contiene la produzione

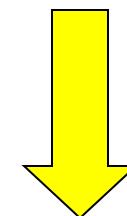
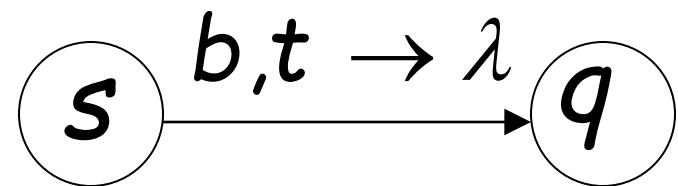
$$A_{pq} \rightarrow aA_{rs}b$$

e il PDA contiene la transizione





$$(p, ayb, \lambda) \succ (r, yb, t)$$



$$(s, b, t) \succ (q, \lambda, \lambda)$$

sappiamo

$$(r, y, \lambda) \stackrel{*}{\succ} (s, \lambda, \lambda) \quad \longrightarrow \quad (r, yb, t) \stackrel{*}{\succ} (s, b, t)$$

$$(p, ayb, \lambda) \succ (r, yb, t)$$

Inoltre sappiamo

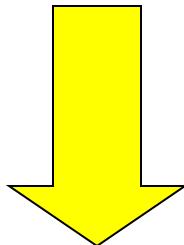
$$(s, b, t) \succ (q, \lambda, \lambda)$$

quindi:

$$(p, ayb, \lambda) \succ (r, yb, t) \stackrel{*}{\succ} (s, b, t) \succ (q, \lambda, \lambda)$$

$$(p, ayb, \lambda) \succ (r, yb, t) \stackrel{*}{\succ} (s, b, t) \succ (q, \lambda, \lambda)$$

poichè $w = ayb$



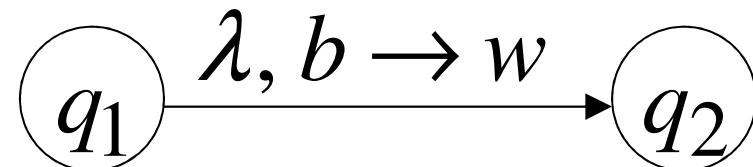
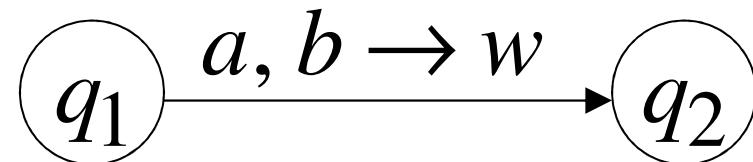
$$(p, w, \lambda) \stackrel{*}{\succ} (q, \lambda, \lambda)$$

Fine

*Determinismo vs
non Determinismo
nei push down.*

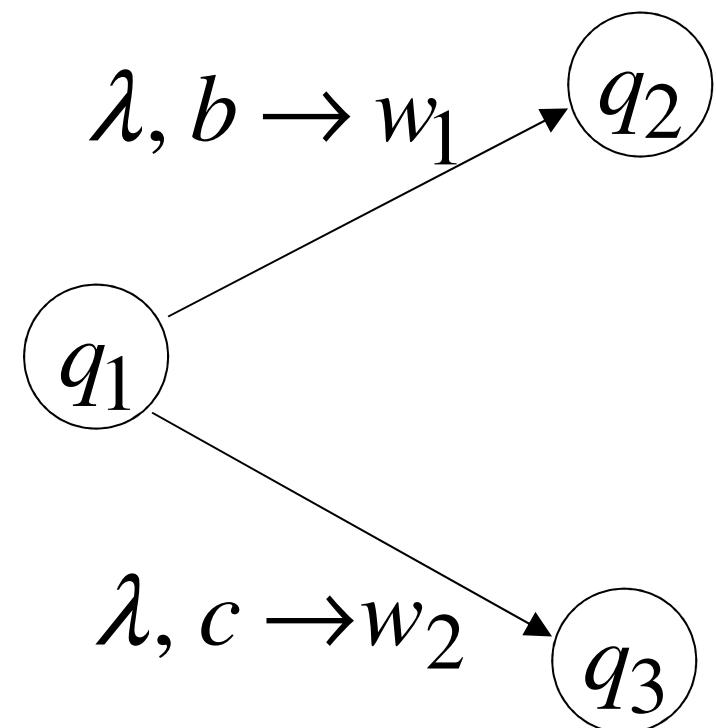
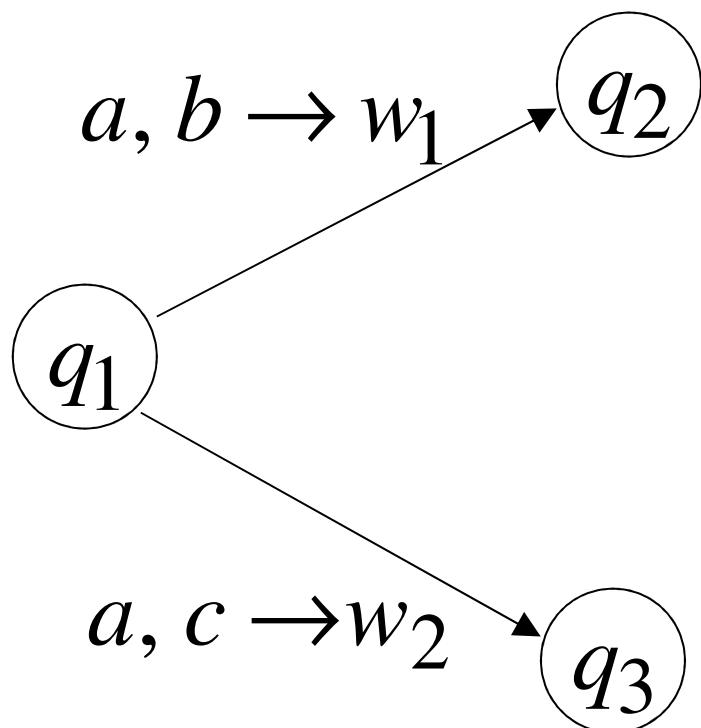
deterministico PDA: DPDA

Transizioni permesse:



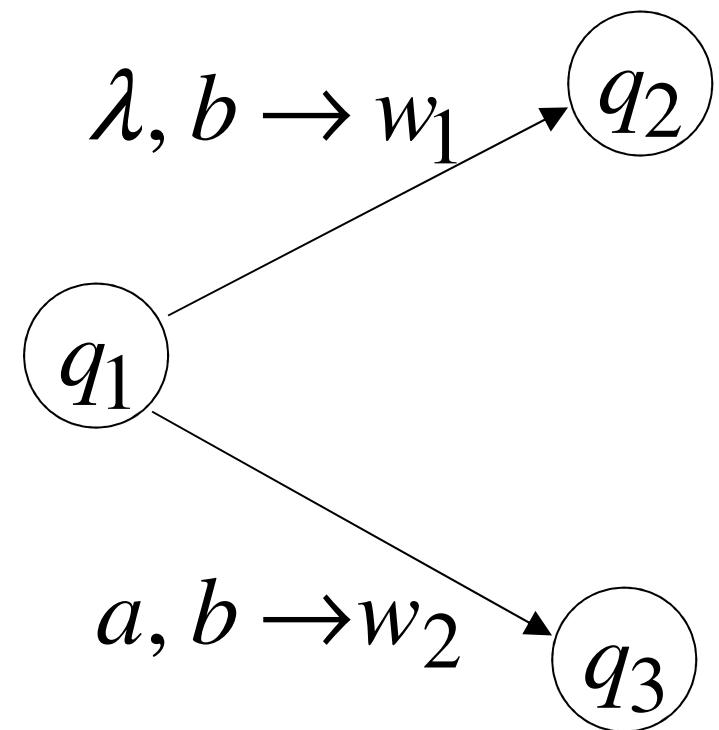
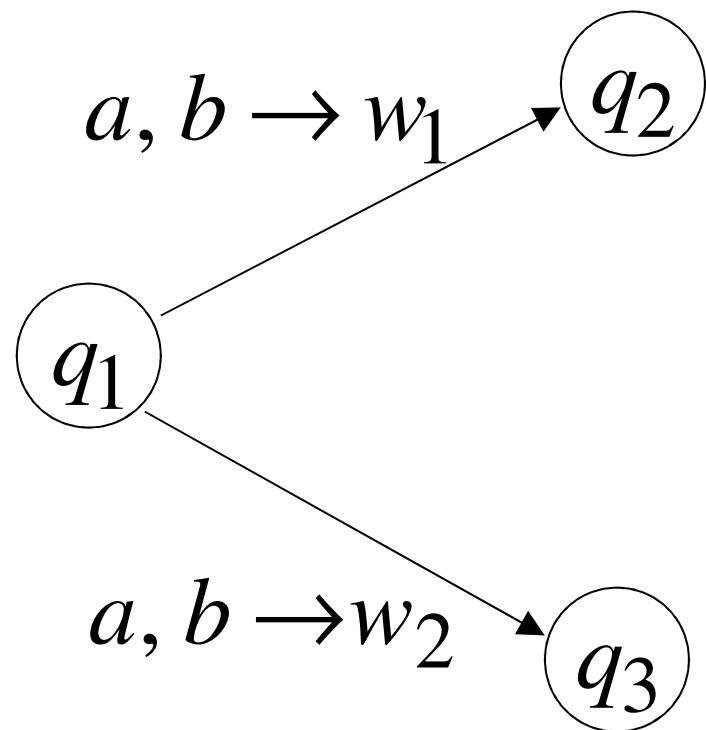
(scelte deterministiche)

Transizioni permesse:



scelte deterministiche

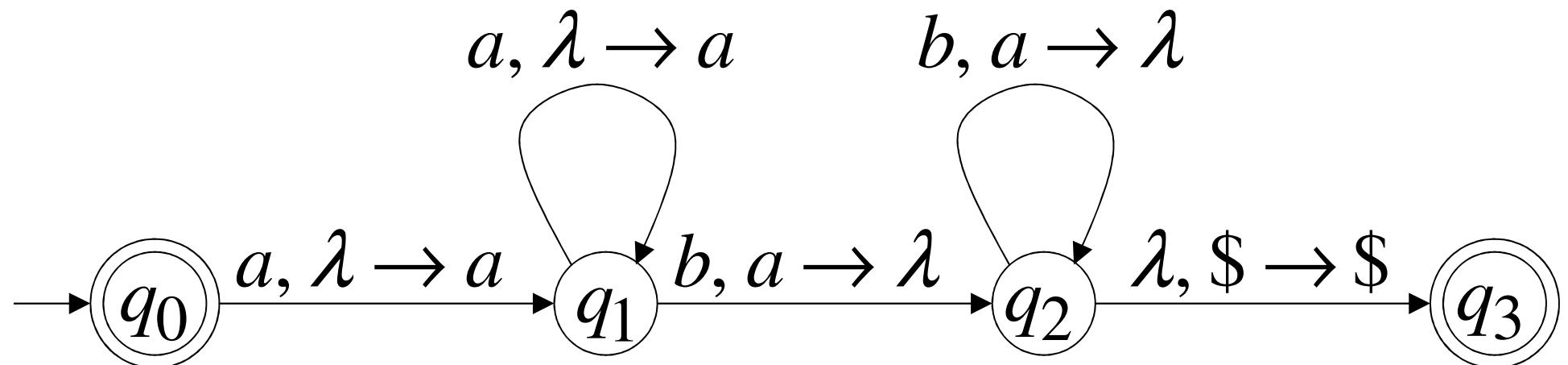
Non permesse:



(scelte non deterministiche)

Deterministico PDA esempio

$$L(M) = \{a^n b^n : n \geq 0\}$$



Definition:

Un linguaggio L è **deterministico context-free**

Se esiste un DPDA che lo accetta

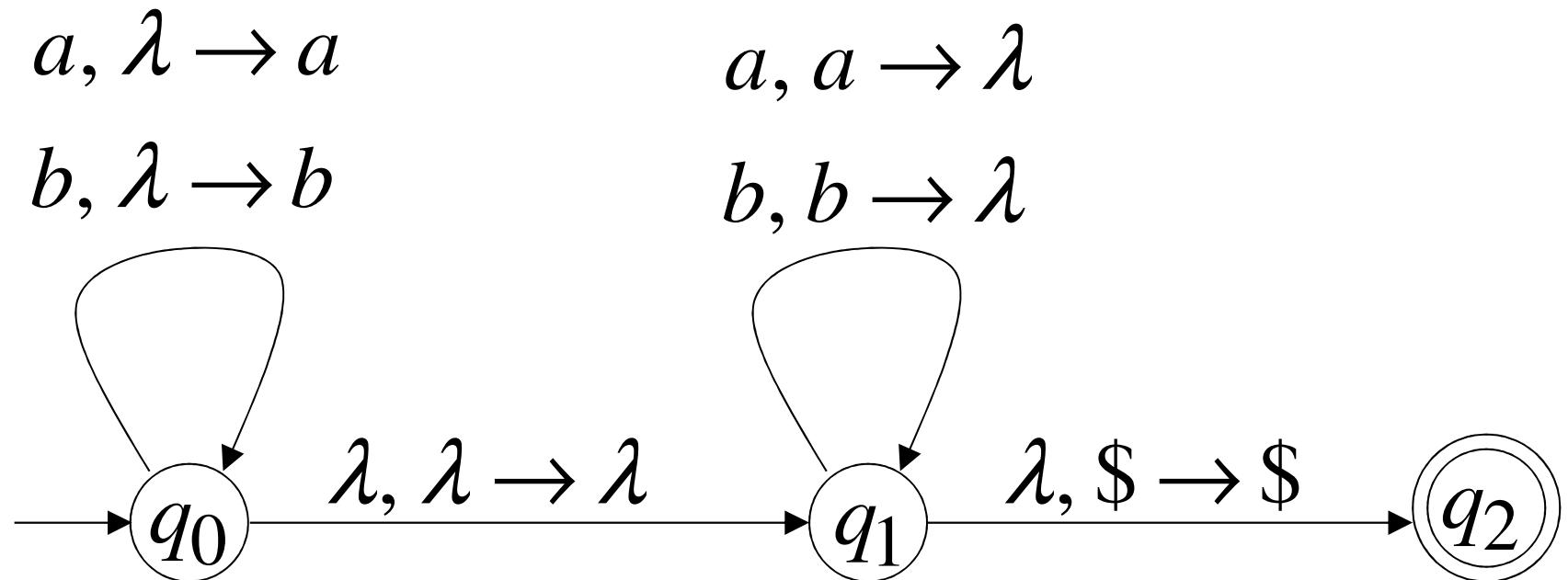
Esempio:

Il linguaggio $L(M) = \{a^n b^n : n \geq 0\}$

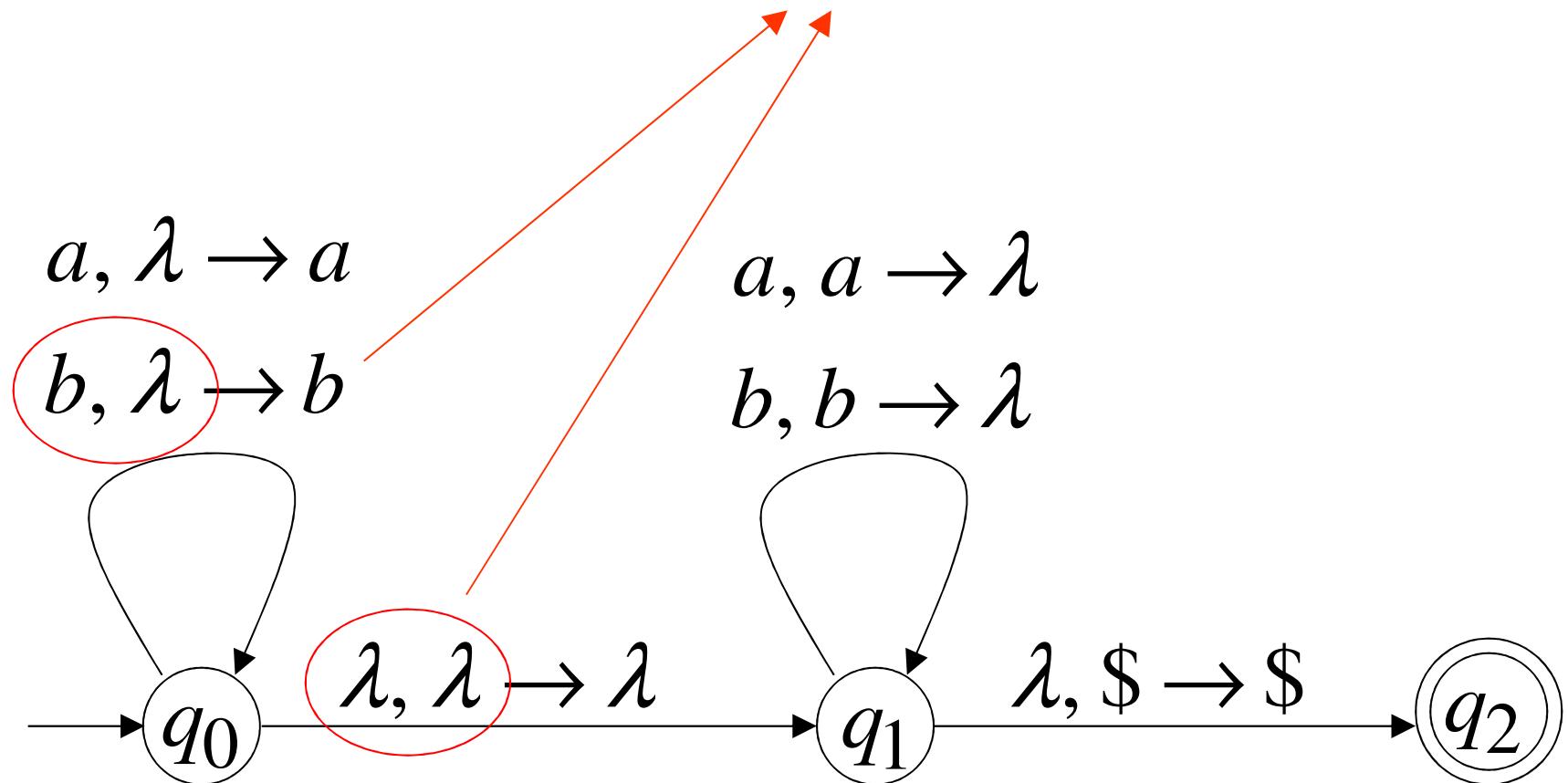
è deterministico context-free

Esempio di Non-DPDA (PDA)

$$L(M) = \{vv^R : v \in \{a,b\}^*\}$$



Non è permesso in DPDA

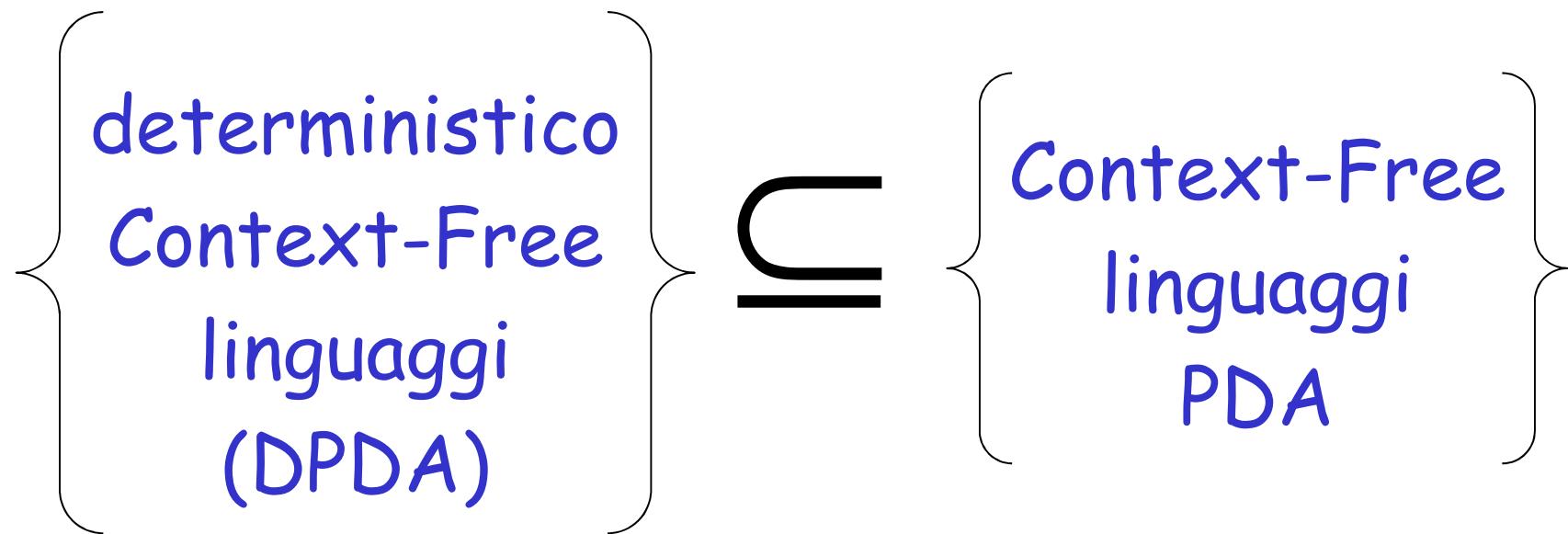


I PDA

hanno più potere dei

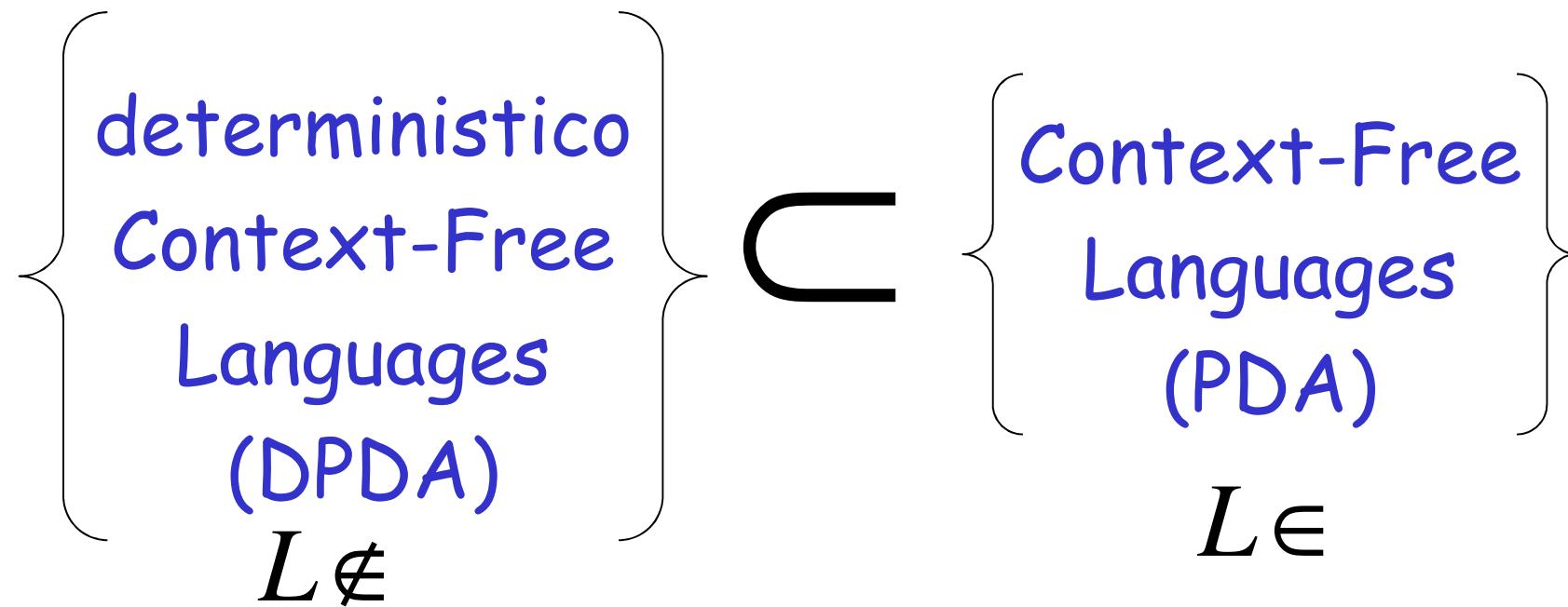
DPDA

Vale la relazione:



Poichè ogni DPDA è anche un PDA

Dimostriamo che :



Definiremo un linguaggio
context-free L che non è
Accettato da un DPDA

Il linguaggio è :

$$L = \{a^n b^n\} \cup \{a^n b^{2n}\} \quad n \geq 0$$

Dobbiamo dimostrare che :

- L è context free
- L **non è** deterministico context-free

$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

Il linguaggio L è context-free

Grammatica Context-free per : L

$$S \rightarrow S_1 \mid S_2 \quad \{a^n b^n\} \cup \{a^n b^{2n}\}$$

$$S_1 \rightarrow aS_1b \mid \lambda \quad \{a^n b^n\}$$

$$S_2 \rightarrow aS_2bb \mid \lambda \quad \{a^n b^{2n}\}$$

Teorema:

Il linguaggio $L = \{a^n b^n\} \cup \{a^n b^{2n}\}$

non è deterministico context-free

(nessun DPDA accetta L)

Dim : Assumiamo per assurdo che

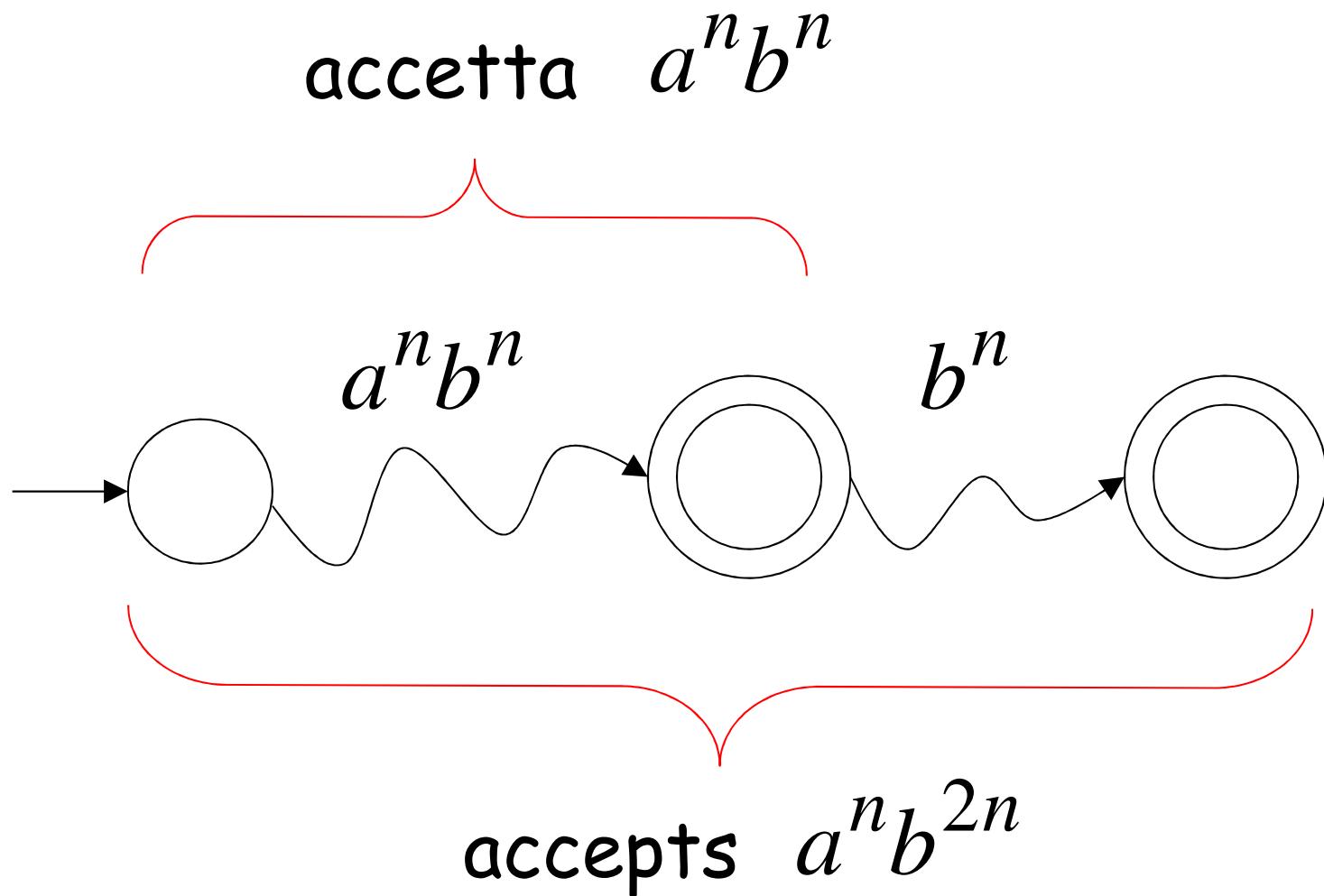
$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

è deterministico context free

quindi:

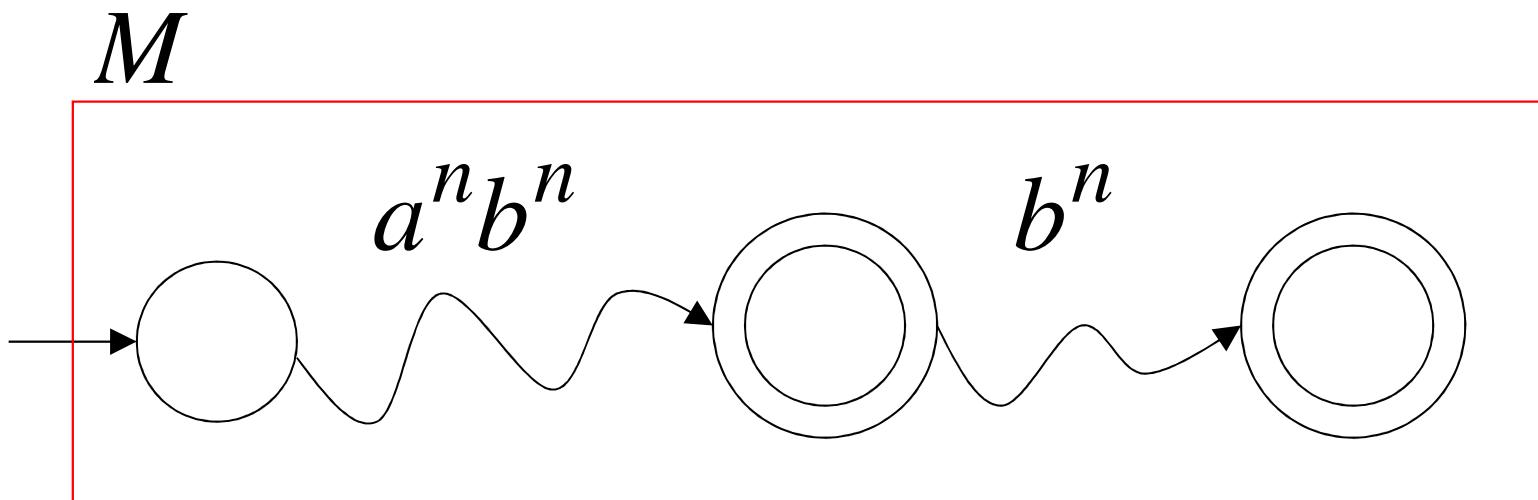
Esiste un DPDA M che accetta L

DPDA M con $L(M) = \{a^n b^n\} \cup \{a^n b^{2n}\}$

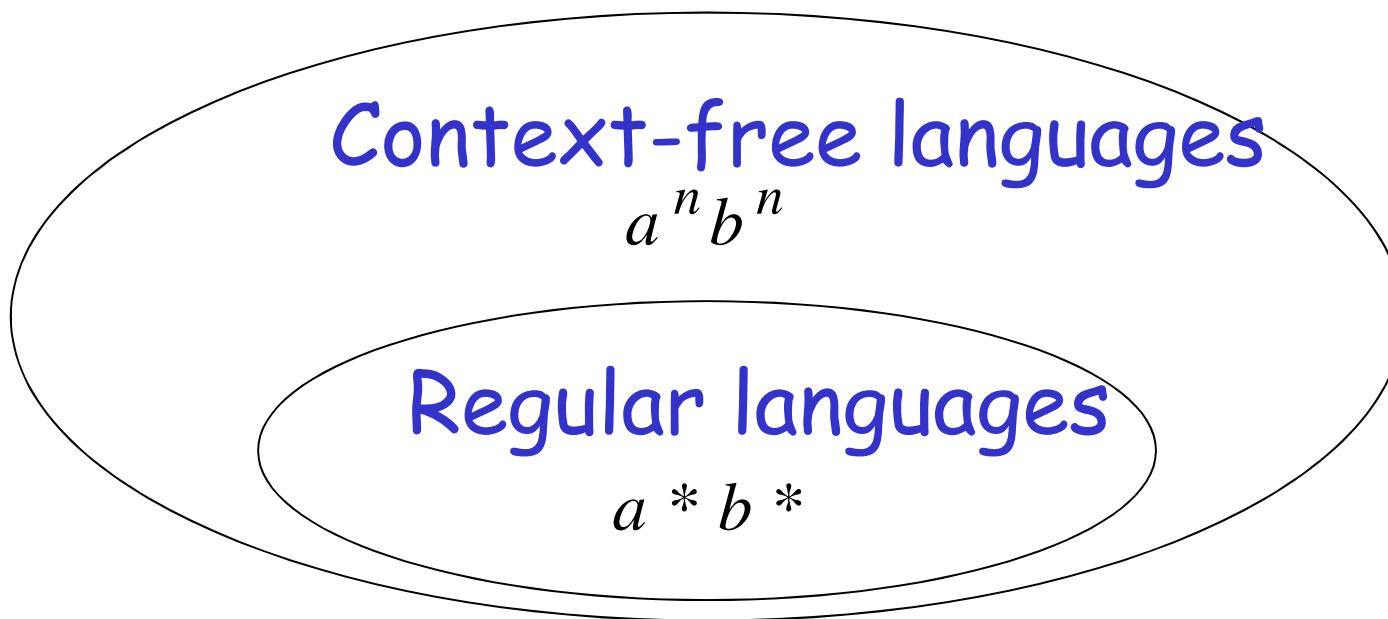


DPDA M con $L(M) = \{a^n b^n\} \cup \{a^n b^{2n}\}$

Un tale cammino
deve esistere a causa del determinismo



Fatto 1: Il linguaggio $\{a^n b^n c^n\}$
not è context-free



(si prova per pumping lemma "per i" context free)

Fatto 2: Il linguaggio $L \cup \{a^n b^n c^n\}$
non è context-free

$$(L = \{a^n b^n\} \cup \{a^n b^{2n}\})$$

(usando pumping lemma per linguaggi context-free)

Ora costruiamo un PDA che accetta:

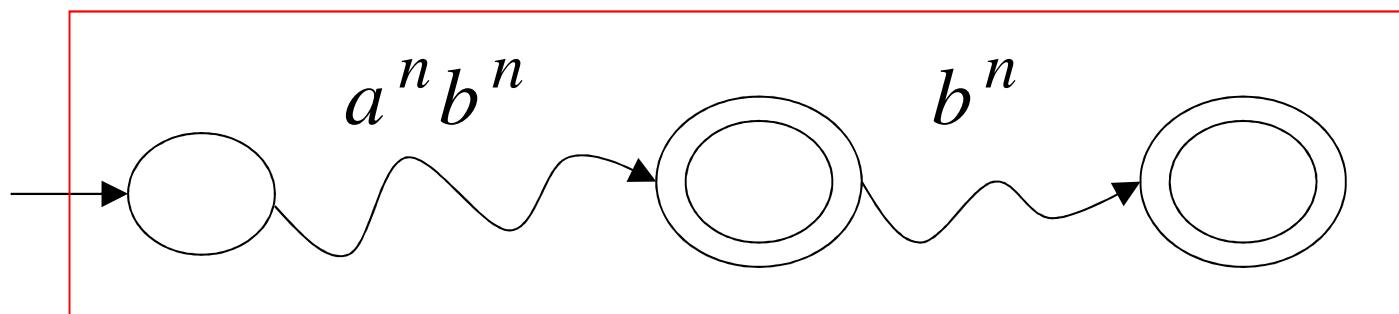
$$L \cup \{a^n b^n c^n\}$$

$$(L = \{a^n b^n\} \cup \{a^n b^{2n}\})$$

Che è una contraddizione !

DPDA M

$$L(M) = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

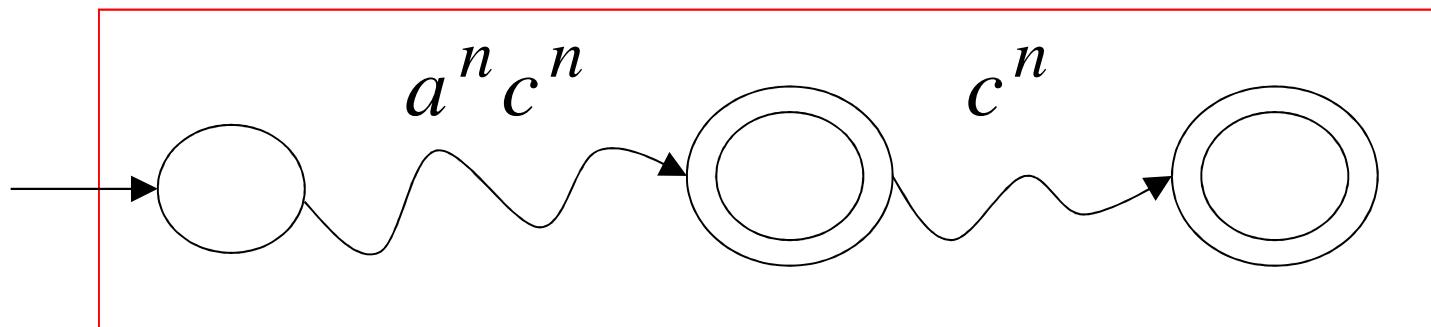


modifichiamo
 M

Al posto
di b
poniamo c

DPDA M'

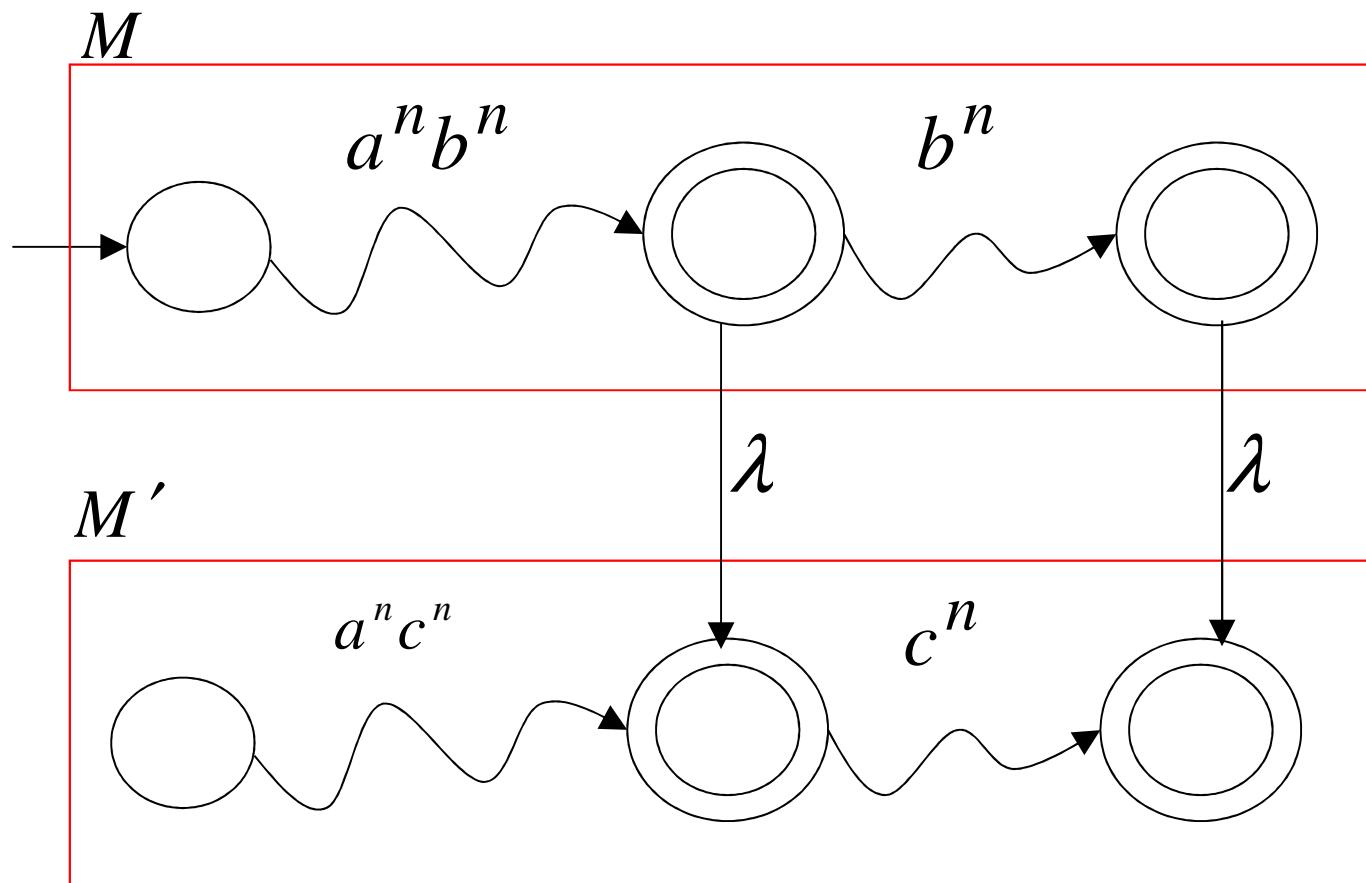
$$L(M') = \{a^n c^n\} \cup \{a^n c^{2n}\}$$



un PDA che accetta $L \cup \{a^n b^n c^n\}$

Connettiamo lo stato finale M

Con lo stato finale di M'



poichè $L \cup \{a^n b^n c^n\}$ è accettato da PDA

È context-free

Contradizione!

(poichè $L \cup \{a^n b^n c^n\}$ non è context-free)

quindi:

$$L = \{a^n b^n\} \cup \{a^n b^{2n}\}$$

Non è deterministico context free

non esiste DPDA che lo accetta

Fine context free

The CYK Algorithm

- J. Cocke
- D. Younger,
- T. Kasami

The CYK Algorithm

- *Il problema dell'appartenenza:*
 - Problema:
 - Data una grammatica grammar \mathbf{G} e una stringa w
 - $\mathbf{G} = (V, \Sigma, P, S)$ dove
 - » V insieme finito di variabili
 - » Σ (alfabeto) insieme finito di simboli terminali
 - » P insieme finito di produzioni
 - » S simbolo iniziale (elemento distintivo di V)
 - » V e Σ sono insiemi disgiunti
 - \mathbf{G} genera un linguaggio, $L(\mathbf{G})$,
 - Domanda :
 - w appartiene al $L(\mathbf{G})$?

The CYK Algorithm

- La grammatica è scritta in Chomsky Normal Form
- Viene usata una tecnica chiamata “dynamic programming” o “table-filling algorithm”

Chomsky Normal Form

- *Normal Form è descritta da un insieme di condizioni che ogni regola della grammatica deve soddisfare.*
- Context-free grammar è in CNF, ogni regola ha la seguente forma:
 - $A \rightarrow BC$ al massimo due simboli sul lato destro
 - $A \rightarrow a$ a simbolo terminale
 - $S \rightarrow \lambda$ stringa vuotaDove $B, C \in V - \{S\}$

Costruire una Triangular Table

$X_{5, 1}$				
$X_{4, 1}$	$X_{4, 2}$			
$X_{3, 1}$	$X_{3, 2}$	$X_{3, 3}$		
$X_{2, 1}$	$X_{2, 2}$	$X_{2, 3}$	$X_{2, 4}$	
$X_{1, 1}$	$X_{1, 2}$	$X_{1, 3}$	$X_{1, 4}$	$X_{1, 5}$

w_1 w_2 w_3 w_4 w_5

Tavola per una stringa ‘w’ che ha lunghezza 5

Esempio CYK Algorithm

- Prendiamo la seguente grammatica:
 - CNF grammatica G
 - $S \rightarrow AB \mid BC$
 - $A \rightarrow BA \mid a$
 - $B \rightarrow CC \mid b$
 - $C \rightarrow AB \mid a$
 - w sia **baaba**
 - E' **baaba** in $L(G)$?

Constructing The Triangular Table

{B}	{A, C}	{A, C}	{B}	{A, C}	
b	a	a	b	a	

Calcolare la riga più bassa

Costruire la Triangular Table

- $X_{2,1} = (X_{1,1}, X_{1,2})$
- $\rightarrow \{B\}\{A,C\} = \{BA, BC\}$
- Step:
 - Trovare, se esistono, le regole che producono BA or BC
 - Sono due : S e A
 - $X_{2,1} = \{S, A\}$

$S \rightarrow AB \mid BC$
 $A \rightarrow BA \mid a$
 $B \rightarrow CC \mid b$
 $C \rightarrow AB \mid a$

$\{S, A\}$				
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$
b	a	a	b	a

Constructing The Triangular Table

- $X_{2,2} = (X_{1,2}, X_{1,3})$
 - $\rightarrow \{A, C\}\{A, C\} = \{AA, AC, CA, CC\} = Y$
 - Step:
 - Trovare, se esistono, le regole che producono Y
 - Esiste una : B
 - $X_{2,2} = \{B\}$
- $S \rightarrow AB \mid BC$
 $A \rightarrow BA \mid a$
 $B \rightarrow CC \mid b$
 $C \rightarrow AB \mid a$

$\{S, A\}$	$\{B\}$			
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$
b	a	a	b	a

- $X_{2,3} = (X_{1,3}, X_{1,4})$
 - $\rightarrow \{A, C\}\{B\} = \{AB, CB\} = Y$
 - Steps:
 - Trovare, se esistono, le regole che producono Y
 - sono: S e C
 - $X_{2,3} = \{S, C\}$
- $S \rightarrow AB \mid BC$
 $A \rightarrow BA \mid a$
 $B \rightarrow CC \mid b$
 $C \rightarrow AB \mid a$

$\{S, A\}$	$\{B\}$	$\{S, C\}$		
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$

b **a** **a** **b** **a**

- $X_{2,4} = (X_{1,4}, X_{1,5})$
 - $\rightarrow \{B\}\{A, C\} = \{BA, BC\} = Y$
 - Steps:
 - Trovare, se esistono, le regole che producono Y
 - Cono: S and A
 - $X_{2,4} = \{S, A\}$
- $S \rightarrow AB \mid BC$
 $A \rightarrow BA \mid a$
 $B \rightarrow CC \mid b$
 $C \rightarrow AB \mid a$

Constructing The Triangular Table

{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}

b **a** **a** **b** **a**

- $X_{3,1} = (X_{1,1}, X_{2,2}), (X_{2,1}, X_{1,3})$
 - $\rightarrow \{B\}\{B\} \cup \{S, A\}\{A, C\} = \{BB, SA, SC, AA, AC\} = Y$
 - Steps:
 - Trovare, se esistono, le regole che producono Y
 - Nessuna
 - $X_{3,1} = \emptyset$
 - Nessun elemento in questo insieme**
- $S \rightarrow AB \mid BC$
 $A \rightarrow BA \mid a$
 $B \rightarrow CC \mid b$
 $C \rightarrow AB \mid a$

\emptyset				
$\{S, A\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$	
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$

b **a** **a** **b** **a**

- $X_{3,2} = (X_{1,2}, X_{2,3}), (X_{2,2}, X_{1,3})$
- $\rightarrow \{A, C\}\{S, C\} \cup \{B\}\{B\} = \{AS, AC, CS, CC, BB\} = Y$
- Step:
 - Trovare, se esistono, le regole che producono Y
 - una: B
 - $X_{2,4} = \{B\}$

$$\begin{aligned}
 S &\rightarrow AB \mid BC \\
 A &\rightarrow BA \mid a \\
 B &\rightarrow CC \mid b \\
 C &\rightarrow AB \mid a
 \end{aligned}$$

\emptyset	$\{B\}$			
$\{S, A\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$	
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$
b	a	a	b	a

- $X_{3,3} = (X_{1,3}, X_{2,4}), (X_{2,3}, X_{1,5})$
 - $\rightarrow \{A,C\}\{S,A\} \cup \{S,C\}\{A,C\}$
 $= \{AS, AA, CS, CA, SA, SC, CA, CC\} = Y$
 - Step:
 - Trovare, se esistono, le regole che producono Y
 - una: B
 - $X_{3,5} = \{B\}$
- $S \rightarrow AB \mid BC$
 $A \rightarrow BA \mid a$
 $B \rightarrow CC \mid b$
 $C \rightarrow AB \mid a$

\emptyset	$\{B\}$	$\{B\}$		
$\{S, A\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$	
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$
b	a	a	b	a

- $X_{4,1} = (X_{1,1}, X_{3,2}), (X_{2,1}, X_{2,3})$
- $(X_{3,1}, X_{1,4})$
- Step:
 - Trovare, se esistono, le regole che producono Y
 - una: B
 - $X_{4,1} = \{?\}$

$$\begin{aligned}
 S &\rightarrow AB \mid BC \\
 A &\rightarrow BA \mid a \\
 B &\rightarrow CC \mid b \\
 C &\rightarrow AB \mid a
 \end{aligned}$$

- $X_{4,2} = (X_{1,2}, X_{3,3}), (X_{2,2}, X_{2,4})$
- $(X_{3,2}, X_{1,5})$
- Step:
 - Trovare, se esistono, le regole che producono Y
 - una: B
 - $X_{4,1} = \{?\}$

$$\begin{aligned}
 S &\rightarrow AB \mid BC \\
 A &\rightarrow BA \mid a \\
 B &\rightarrow CC \mid b \\
 C &\rightarrow AB \mid a
 \end{aligned}$$

Finale Triangular Table

$\{S, A, C\}$	$\leftarrow x_{5,1}$					
\emptyset	$\{S, A, C\}$					
\emptyset	$\{B\}$	$\{B\}$				
$\{S, A\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$			
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$		
b	a	a	b	a		

- Tavola per la stringa 'w' ha lunghezza 5
- The algorithm popola la triangular table

domanda

- sia **G** la grammatical CNF
 - $S \rightarrow AB \mid BC$
 - $A \rightarrow BA \mid a$
 - $B \rightarrow CC \mid b$
 - $C \rightarrow AB \mid a$
- **w** is ababa
- Domanda: **ababa** è in $L(G)$?

- E' baaba in $L(G)$?

Si

Possiamo vedere che S è nell'insieme X_{1n} dove
‘n’= 5

la cella $X_{51} = (S, A, C)$ allora

$S \in X_{15}$ allora $baaba \in L(G)$

Costruire una Triangular Table

- Ogni riga corrisponde a una lunghezza delle sottostringhe.
 - La riga più in basso – Stringhe di lung 1
 - Seconda riga – Stringhe di lung 2
 -
 -
 - Riga più in alto – la stringa ‘w’

Costruire una Triangular Table

- $X_{i,i}$ è l'insieme delle variabili tale che $A \in A \rightarrow w_i$ una produzione di G
- Comparare al massimo n coppie di insieme calcolati in precedenza
$$(X_{i,i}, X_{i+1,j}), (X_{i,i+1}, X_{i+2,j}) \dots (X_{i,j-1}, X_{j,j})$$

teorema

- The CYK Algorithm calcola correttamente X_{ij} per tutti i e j ; allora w è in $L(G)$ iff S è in X_{1n} .
- Perchè? Spiegazione, esercizio scrivere la dimostrazione.
- Complessità $O(n^3)$.

- let the input be a string L consisting of n characters: $a_1 \dots a_n$.
- let the grammar contain r nonterminal symbols $R_1 \dots R_r$, with start symbol R_1 .
- let $P[n,n,r]$ be an array of booleans. Initialize all elements of P to false.
- **for each** $s = 1$ to n
- **for each** unit production $R_v \rightarrow a_s$
- set $P[1,s,v] = \text{true}$; Generata la prima riga-
- **for each** $L = 2$ to n
- **for each** $s = 1$ to $n-L+1$
- **for each** $p = 1$ to $L-1$
- **for each** production $R_a \rightarrow R_b R_c$
- **if** $P[p,s,b]$ and $P[L-p,s+p,c]$ **then** set $P[L,s,a] = \text{true}$
- **if** $P[n,1,1]$ is true **then** L is member of language
- **else** L is not member of language

$\{S, A, C\}$	$\leftarrow X_{5,1}$				
\emptyset	$\{S, A, C\}$				
\emptyset	$\{B\}$	$\{B\}$			
$\{S, A\}$	$\{B\}$	$\{S, C\}$	$\{S, A\}$		
$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$	
b	a	a	b	a	

$X_{5, 1}$	$(5)n - L+1=s$			
$X_{4, 1}$	$X_{4, 2}$	$(4)n - L+1=s$		
$X_{3, 1}$	$X_{3, 2}$	$X_{3, 3}$	$(3)n-L+1 =s$	
$X_{2, 1}$	$X_{2, 2}$	$X_{2, 3}$	$X_{2, 4}$	$(2)n - L+1=s$
$X_{1, 1}$	$X_{1, 2}$	$X_{1, 3}$	$X_{1, 4}$	$X_{1, 5}$

```

for each  $L = 2$  to  $n$ 
  for each  $s = 1$  to  $n-L+1$ 
    for each  $p = 1$  to  $L-1$ 
      for each production  $R_a \rightarrow R_b R_c$ 
        if  $P[p,s,b]$  and  $P[L-p,s+p,c]$  then set  $P[L,s,a] = \text{true}$ 

```

$X_{5, 1}$				
$X_{4, 1}$	$X_{4, 2}$			
$X_{3, 1}$	$X_{3, 2}$	$X_{3, 3}$	$L=3, s=1$	
$X_{2, 1}$	$X_{2, 2}$	$X_{2, 3}$	$X_{2, 4}$	
$X_{1, 1}$	$X_{1, 2}$	$X_{1, 3}$	$X_{1, 4}$	$X_{1, 5}$

w_1 w_2 w_3 w_4 w_5

Tavola per una stringa ‘w’ che ha lunghezza 5

```

for each  $L = 2$  to  $n$ 
  for each  $s = 1$  to  $n-L+1$ 
    for each  $p = 1$  to  $L-1$ 
      for each production  $R_a \rightarrow R_b R_c$ 
        if  $P[p,s,b]$  and  $P[L-p,s+p,c]$  then set  $P[L,s,a] = \text{true}$ 

```

$X_{5, 1}$					
$X_{4, 1}$	$X_{4, 2}$ Δ	$L=4, s=2$			
$X_{3, 1}$	$X_{3, 2}$ Δ	$X_{3, 3}$ \square			
$X_{2, 1}$	$X_{2, 2}$ \circ	$X_{2, 3}$	$X_{2, 4}$ \bullet		
$X_{1, 1}$	$X_{1, 2}$ \square	$X_{1, 3}$	$X_{1, 4}$	$X_{1, 5}$ Δ	
w_1	w_2	w_3	w_4	w_5	

Tavola per una stringa ‘w’ che ha lunghezza 5

```

for each  $L = 2$  to  $n$ 
for each  $s = 1$  to  $n-L+1$ 
    for each  $p = 1$  to  $L-1$ 
for each production  $R_a \rightarrow R_b R_c$ 
if  $P[p, s, b]$  and  $P[L-p, s+p, c]$  then set  $P[L, s, a] = \text{true}$ 

```

$X_{5, 1}$				
$X_{4, 1}$	$X_{4, 2}$			
$X_{3, 1}$	$X_{3, 2}$	$X_{3, 3}$		
$X_{2, 1}$	$X_{2, 2}$	$X_{2, 3}$	$X_{2, 4}$	
$X_{1, 1}$	$X_{1, 2}$	$X_{1, 3}$	$X_{1, 4}$	$X_{1, 5}$

w_1 w_2 w_3 w_4 w_5

Tavola per una stringa ‘w’ che ha lunghezza 5



Walton Athletic Club, 1947

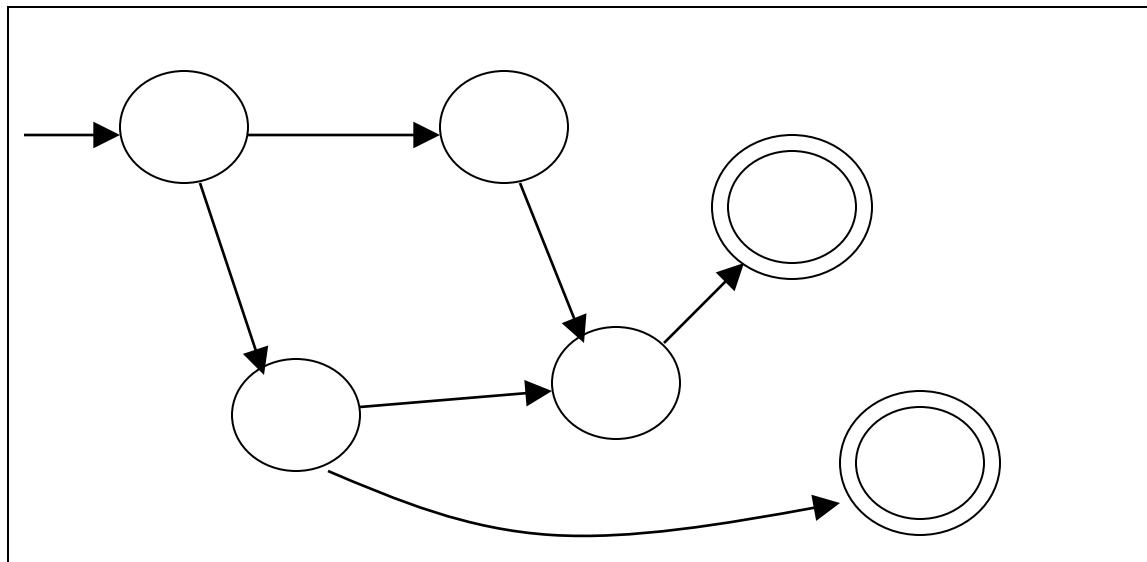
Macchine di Turing

Una macchina di Turing

Tape

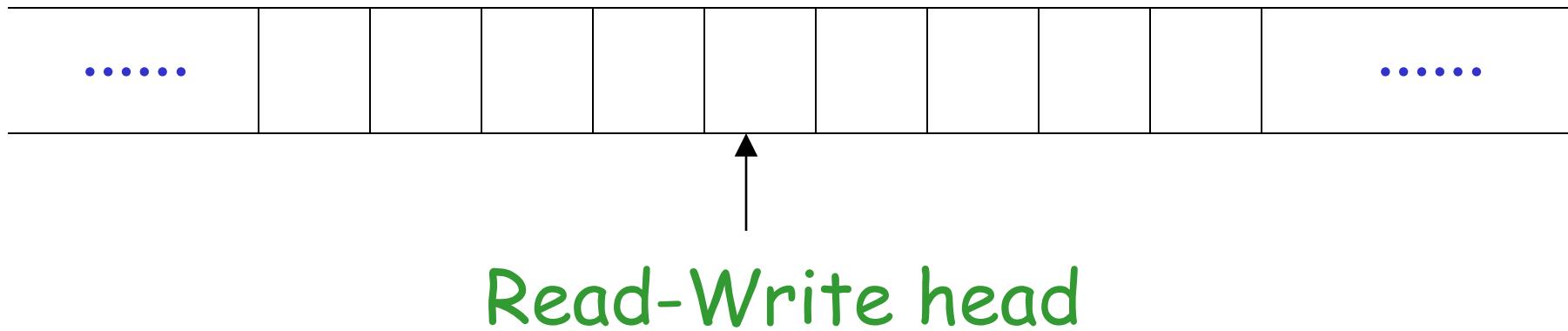


Control Unit

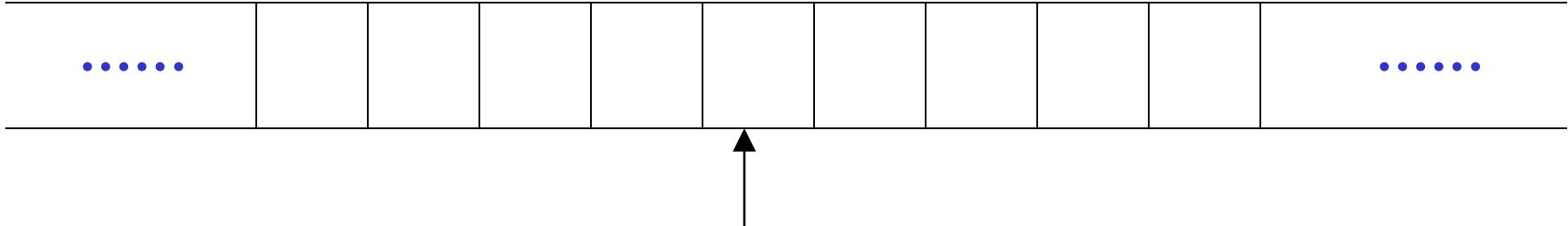


il Tape

No limiti - lunghezza potenzialmente infinita



Le testa si muove Left or Right



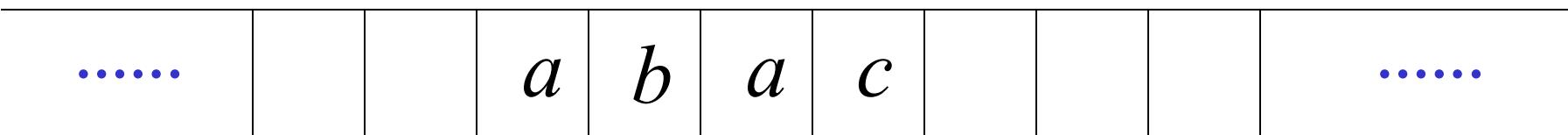
Read-Write head

la head ad ogni transizione (time step):

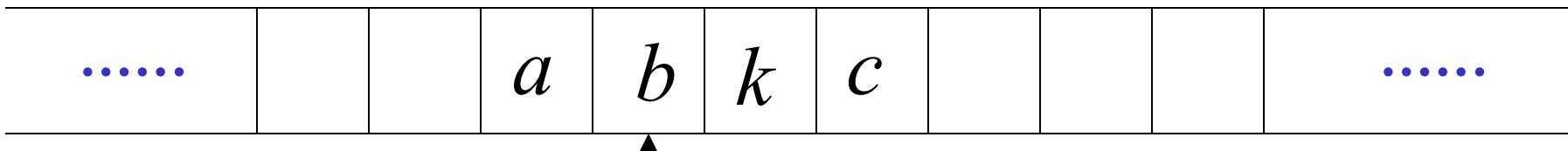
1. legge un simbolo
2. scrive un simbolo
3. si muove Left or Right

esempio:

Time 0



Time 1

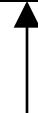
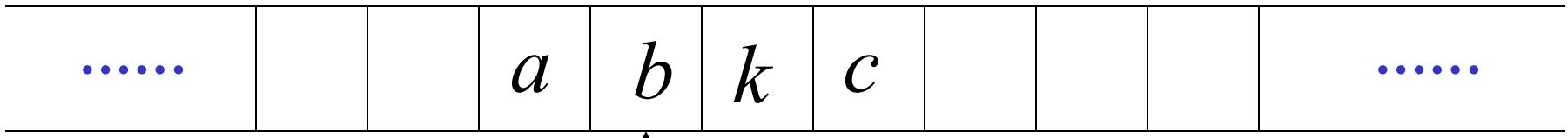


1. Reads a

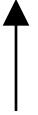
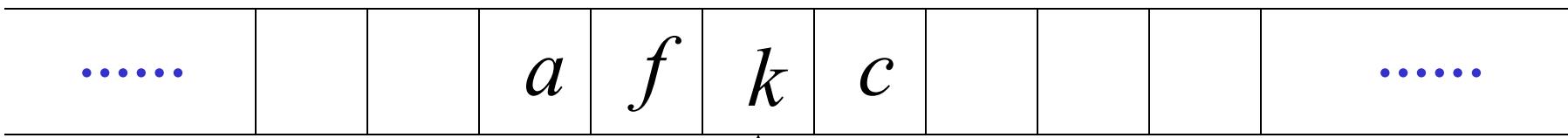
2. Writes k

3. Moves Left

Time 1



Time 2

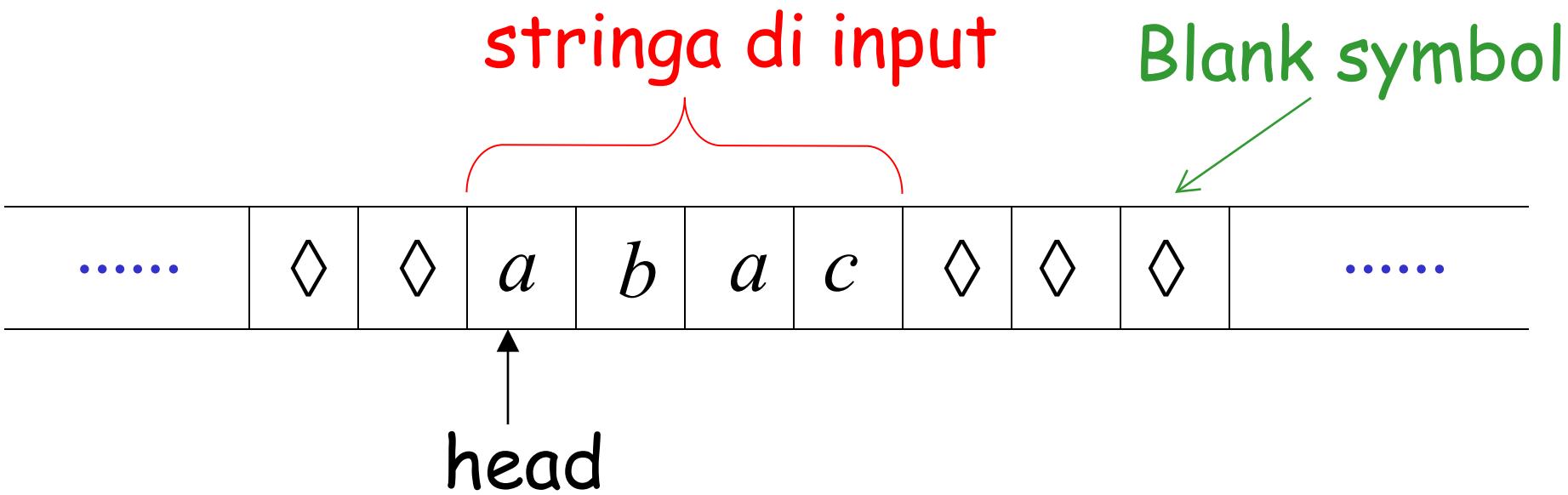


1. Reads b

2. Writes f

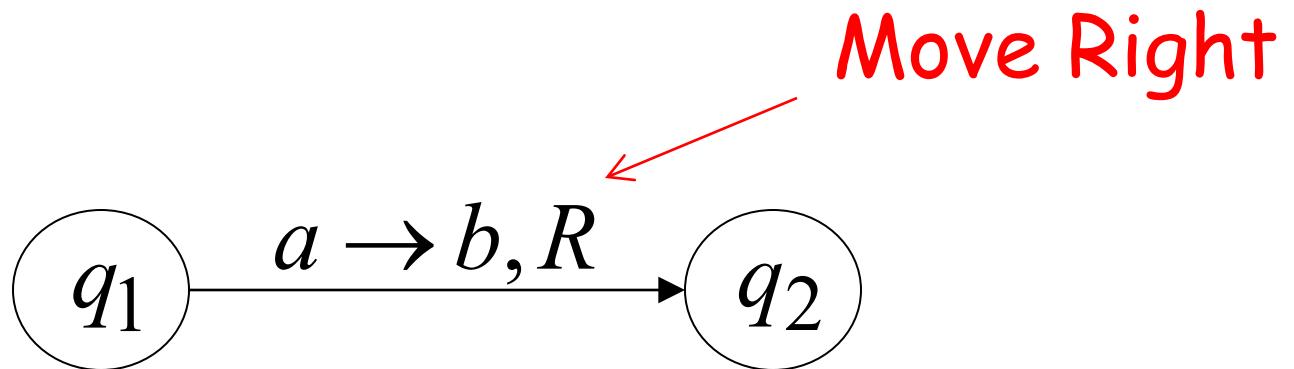
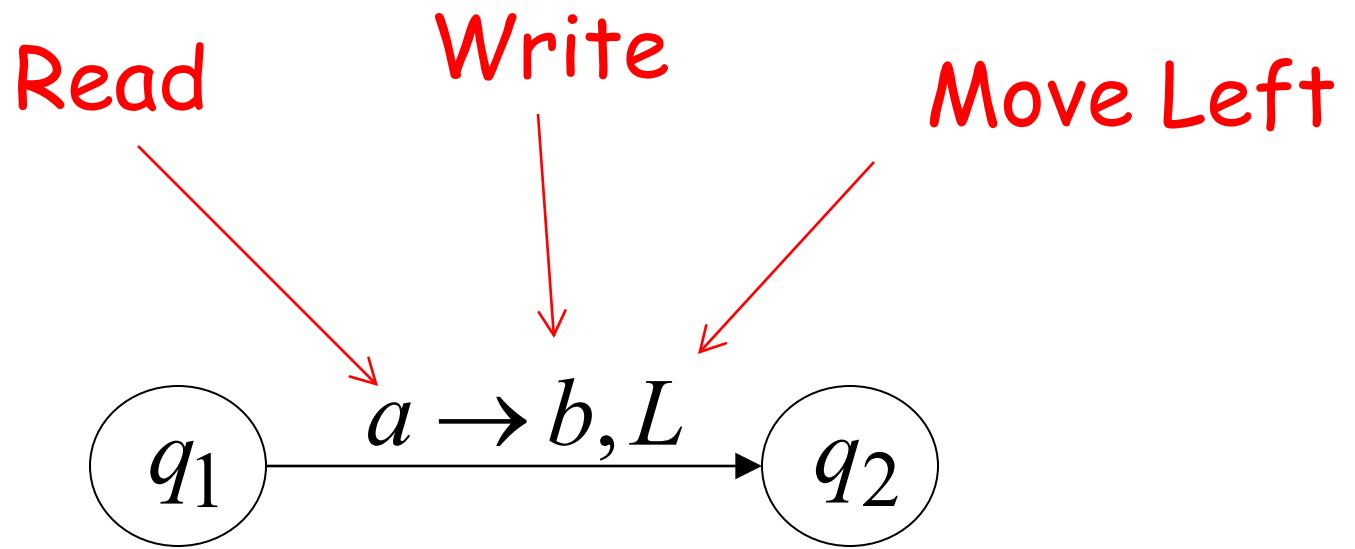
3. Moves Right

la String Input



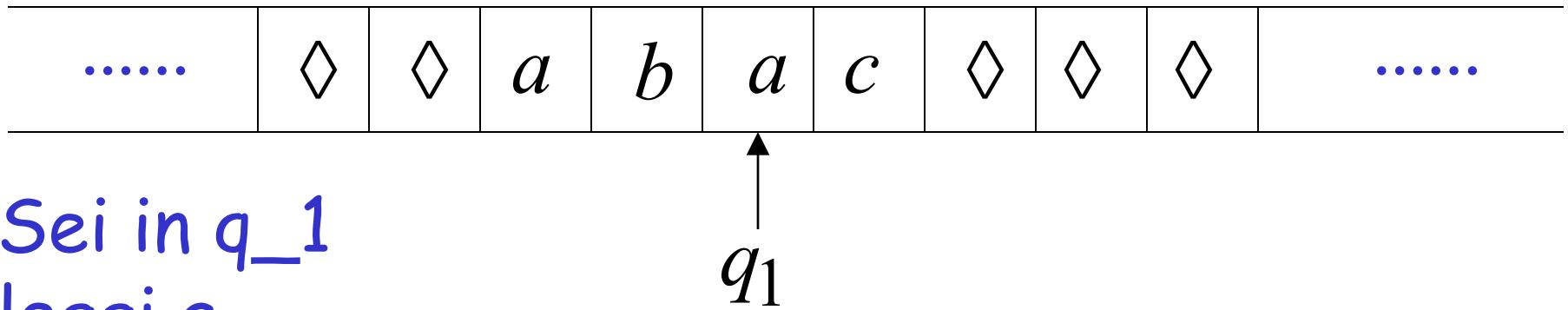
Head parte dalla posizione più a sinistra
della stringa di input

Stati & Transizioni

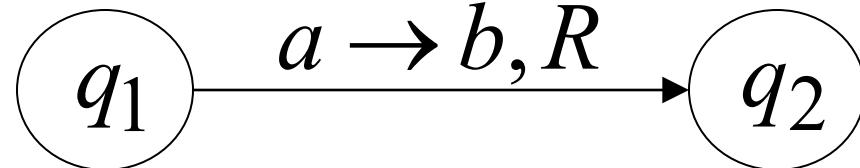


esempio:

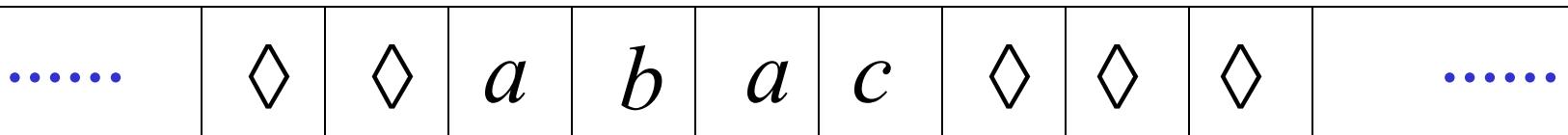
Time 1



Sei in q_1
leggi a ,
cambia a in b stato corrente
e vai a right

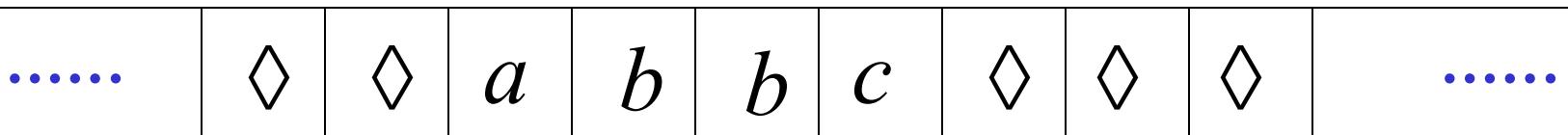


Time 1

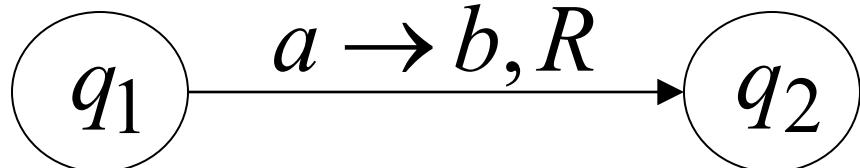


q_1

Time 2

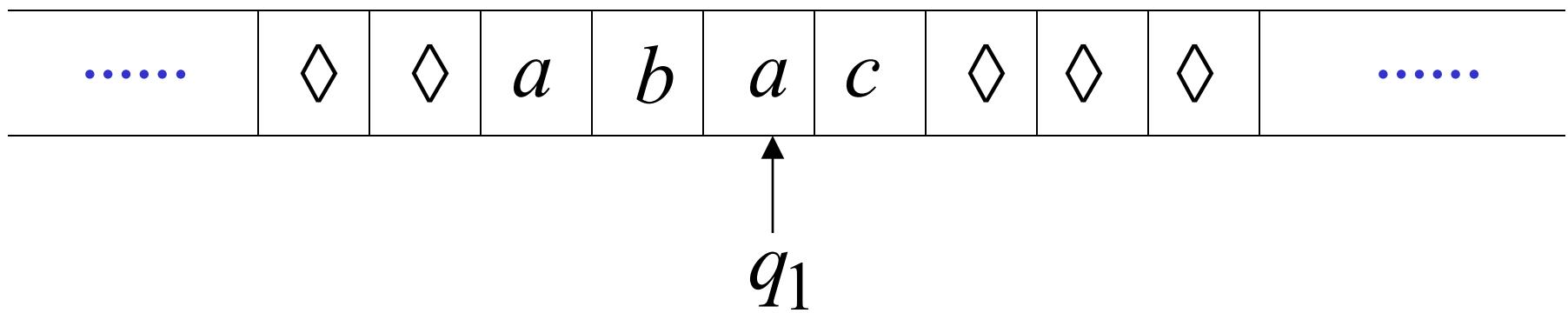


q_2

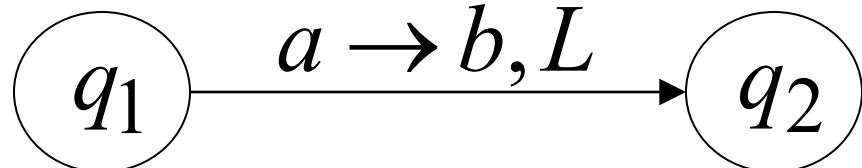
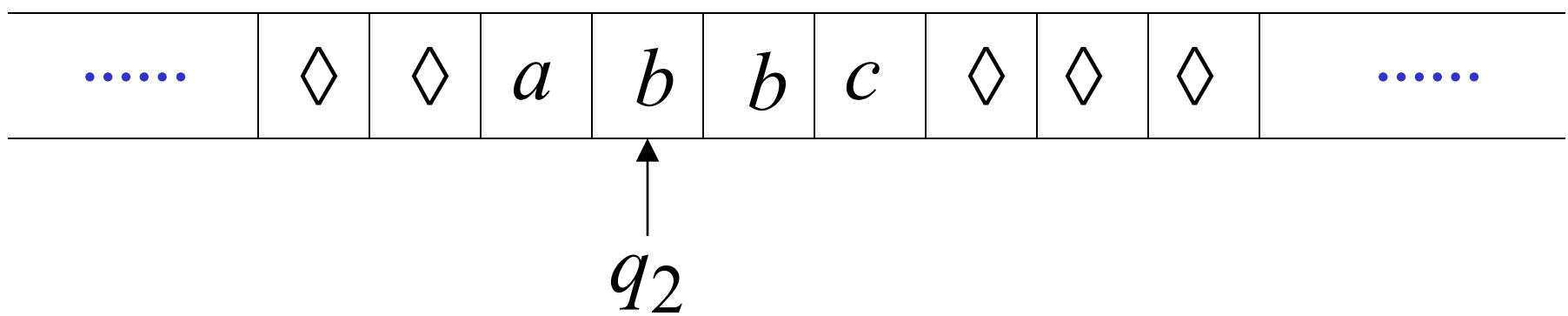


esempio:

Time 1



Time 2



esempio:

Time 1

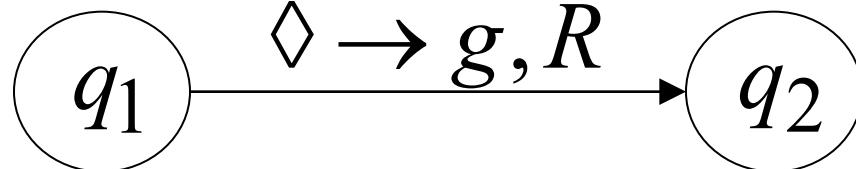
.....	◊	◊	a	b	a	c	◊	◊	◊
-------	---	---	---	---	---	---	---	---	---	-------

↑
 q_1

Time 2

.....	◊	◊	a	b	b	c	g	◊	◊
-------	---	---	---	---	---	---	---	---	---	-------

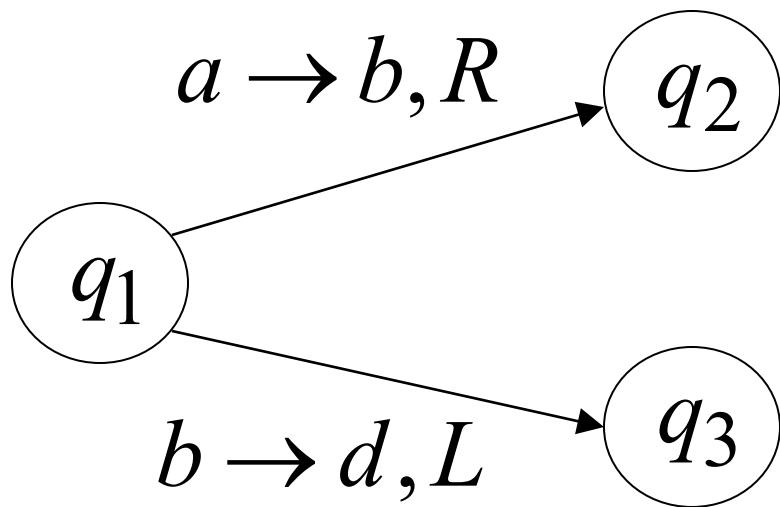
↑
 q_2



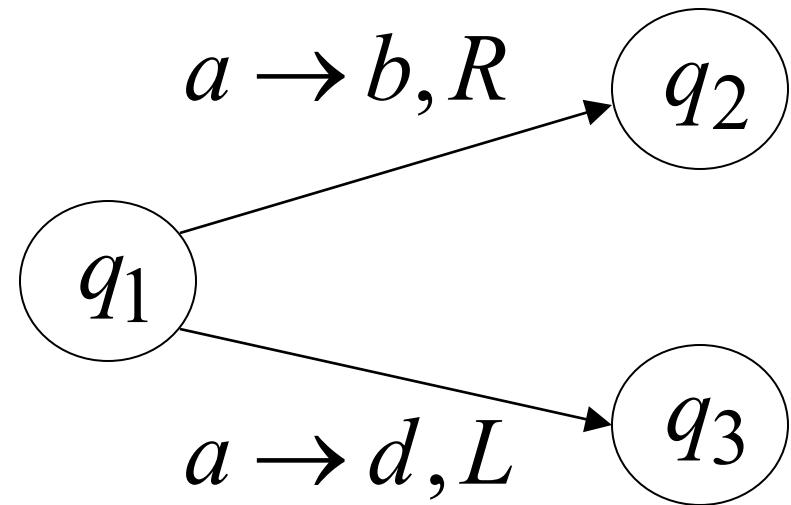
Determinismo

macchine di Turing sono deterministiche

permesso



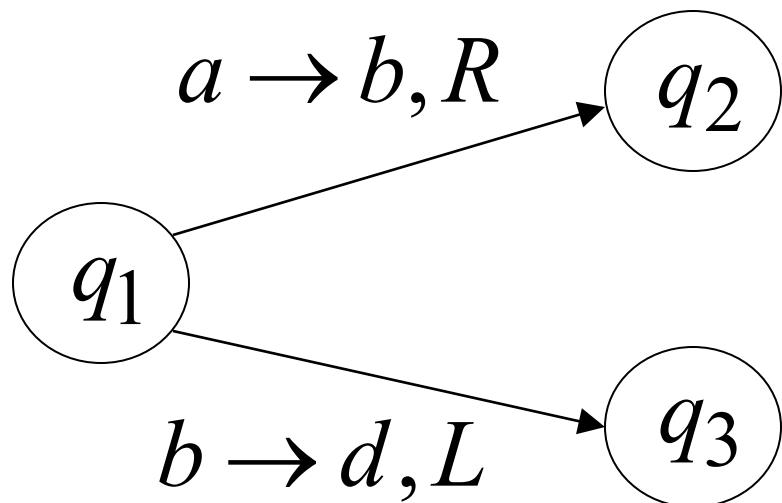
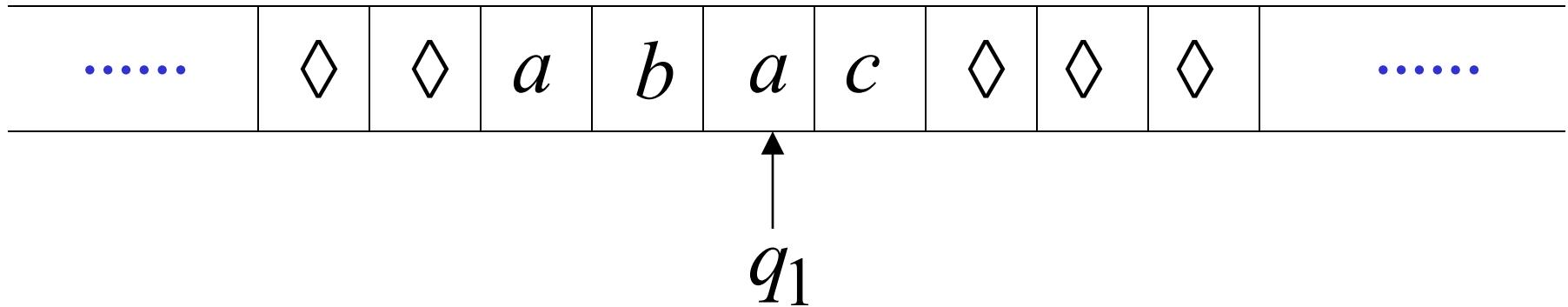
Non permesso



nessuna transizione lambda è permessa

Funzione di Transizione Parziale

esempio:



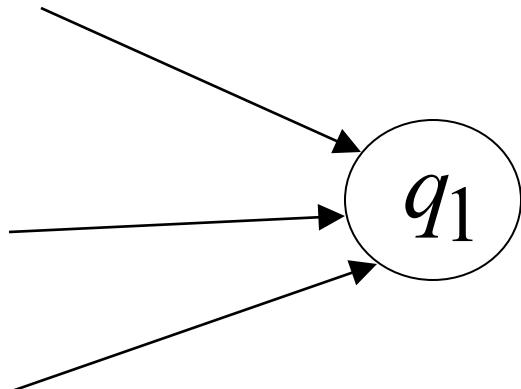
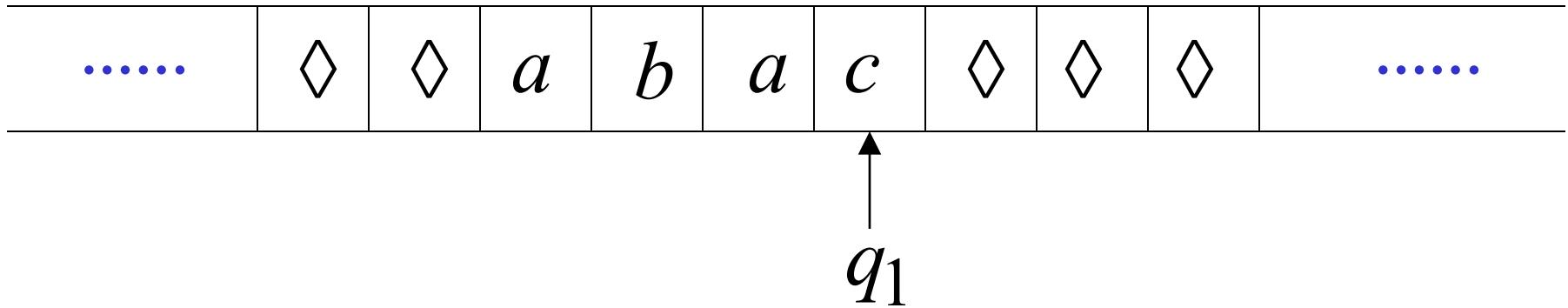
permesso:

Nessuna transizione
per simbolo input c

Halting

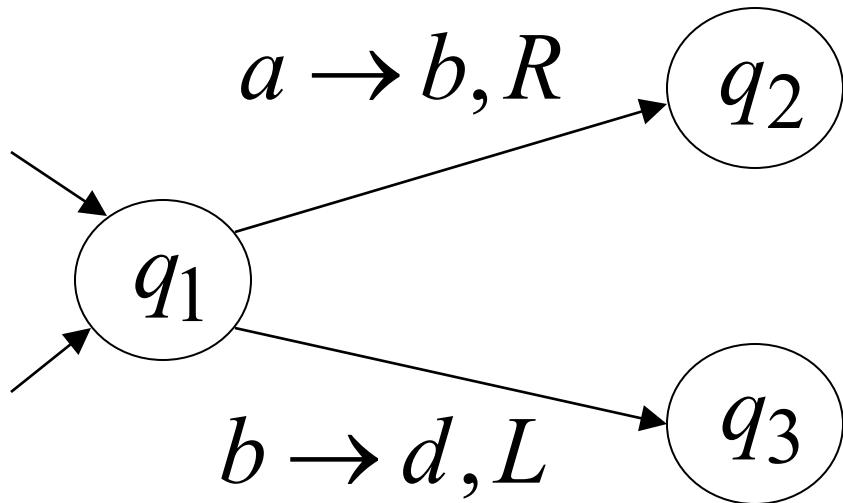
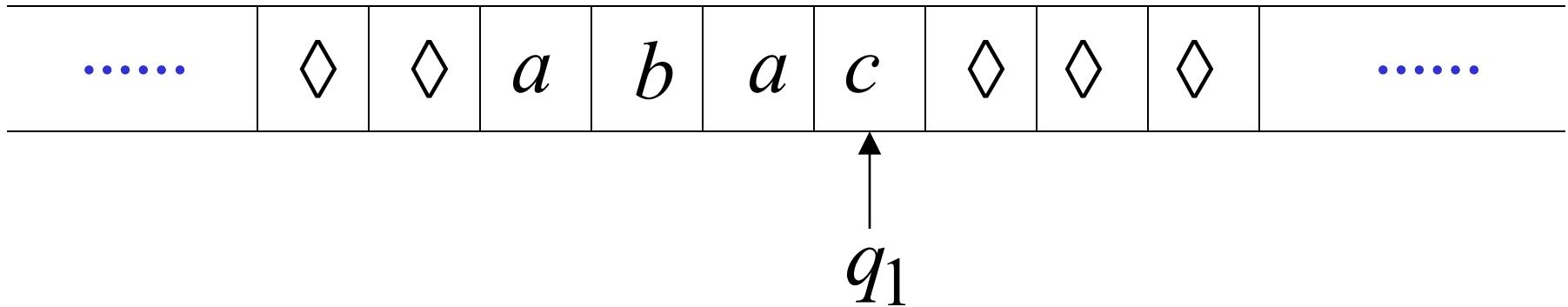
La macchina *si ferma* nello stato
in cui si trova
se non vi è nessuna transizione
da eseguire

Halt esempio 1:



Nessuna transizione da q_1
HALT!!!

Halt esempio 2:



Nessuna transizione
possibile da q_1
sul simbolo c

HALT!!!

Stati di accettazione



permesso



Non permesso

- Stati di accettazione non hanno transizioni in uscita
- La macchina si ferma e accetta.

Accettazione

Accettare stringa

In Input



se macchina si ferma
in uno stato di
accettazione

RIGETTARE stringa

In Input

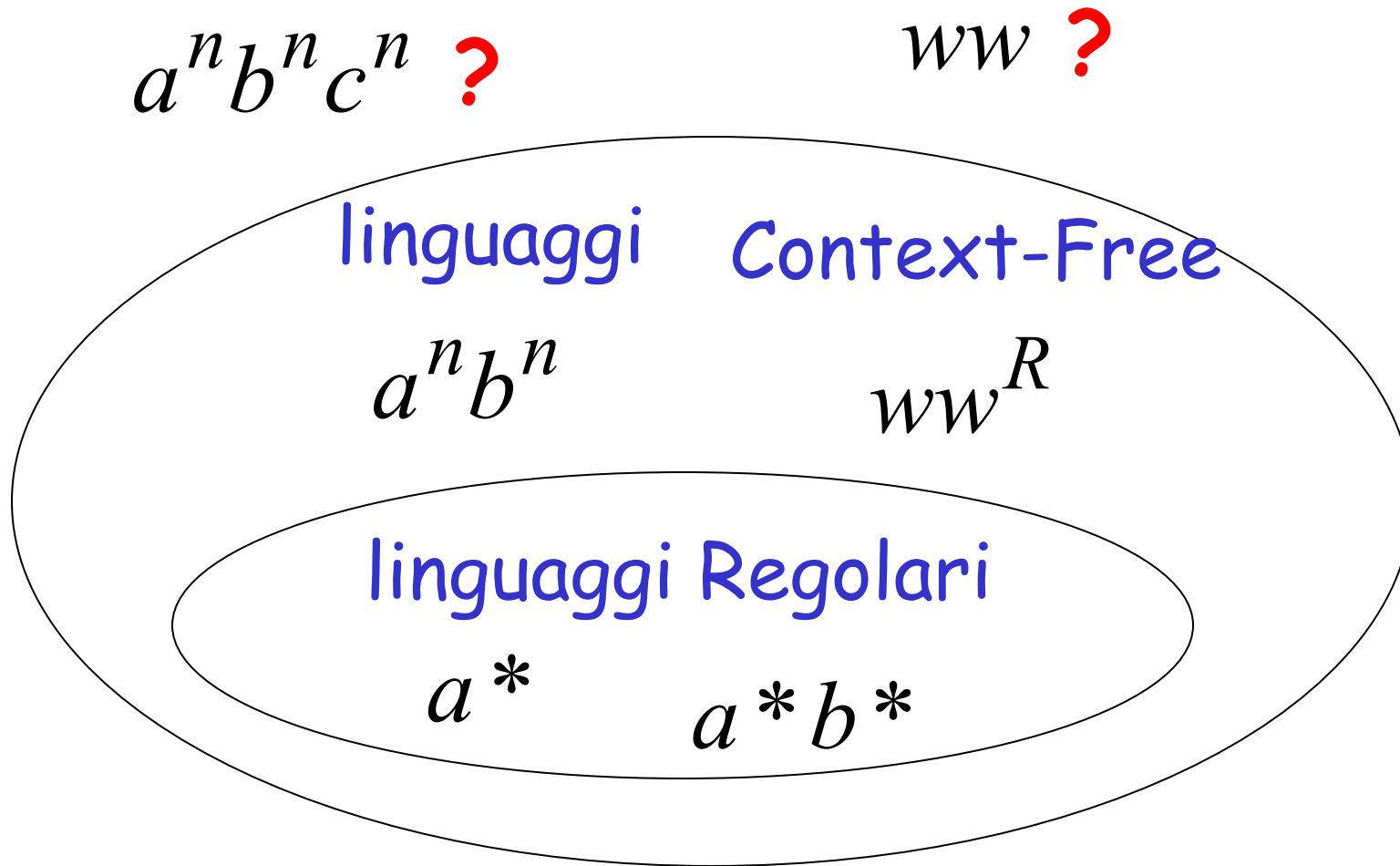


se macchina si ferma
in uno stato di
NON- accettazione o
se la macchina entra
in un infinite loop

Osservazione:

Nell'accettare una stringa di input,
non è necessario esaminare tutti
i simboli nella stringa.

La gerarchia dei linguaggi



linguaggi accettati da una macchina di Turing

$a^n b^n c^n$

ww

Context-Free linguaggi

$a^n b^n$

ww^R

Regular linguaggi

a^*

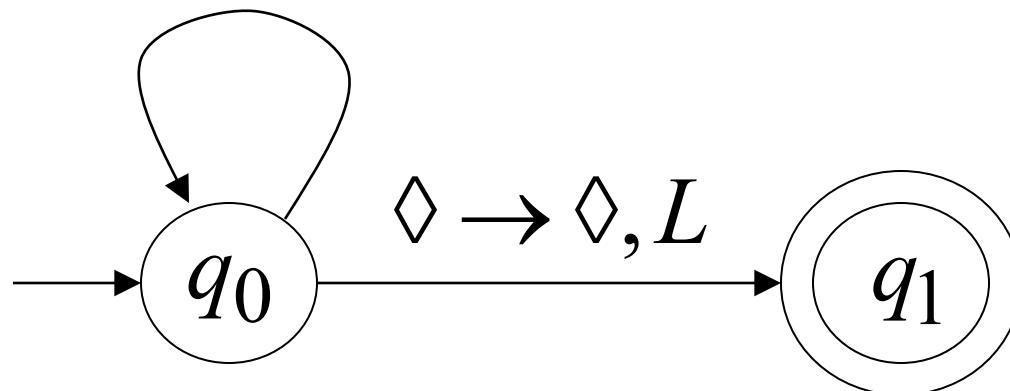
$a^* b^*$

macchina Turing esempio

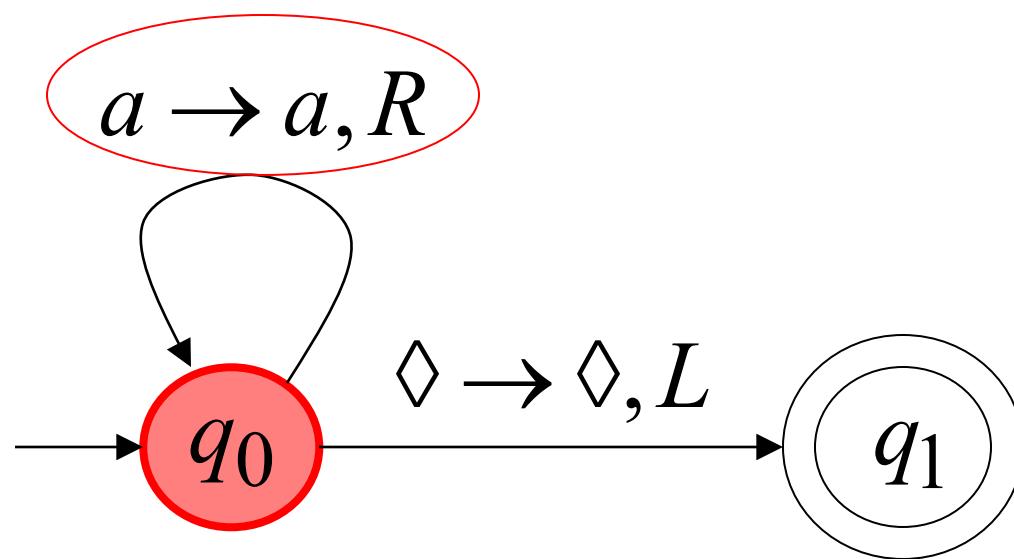
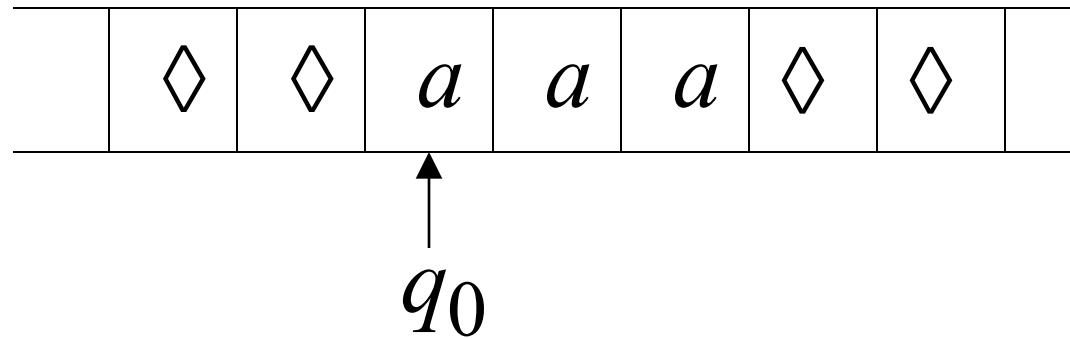
Input alphabet $\Sigma = \{a, b\}$

Accetta il linguaggio: a^*

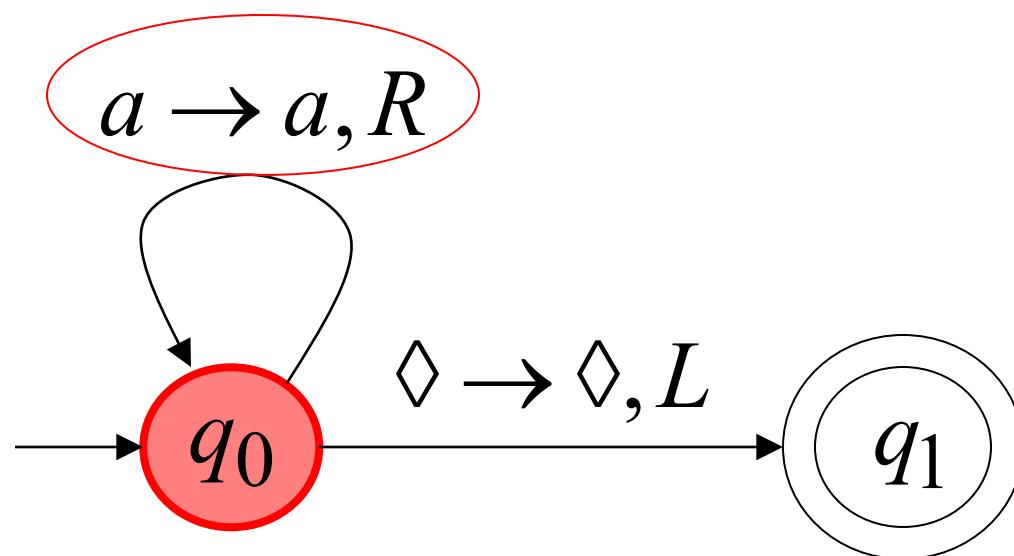
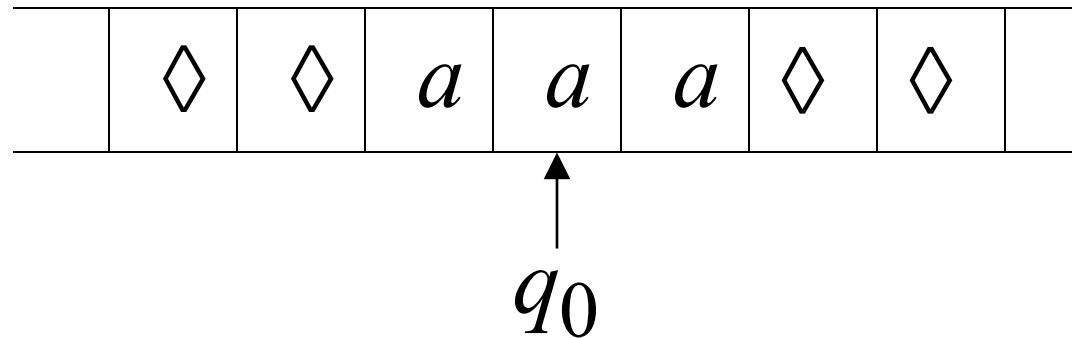
$$a \rightarrow a, R$$



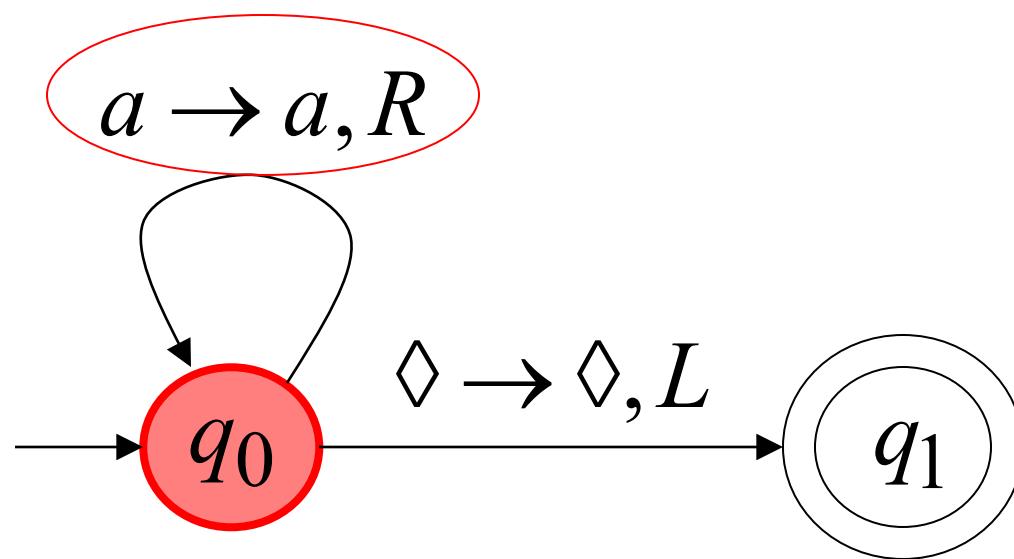
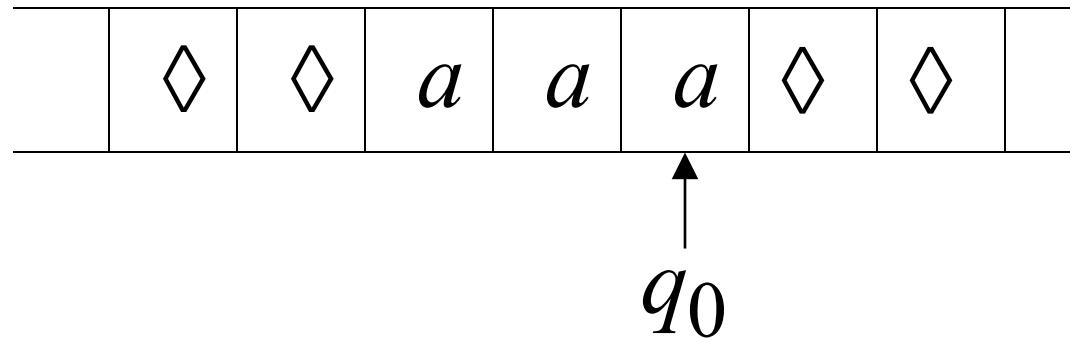
Time 0



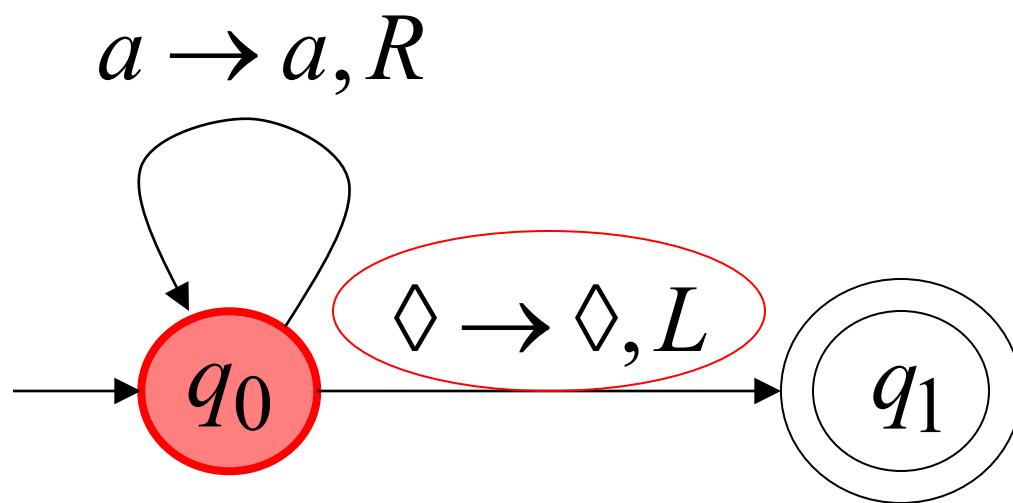
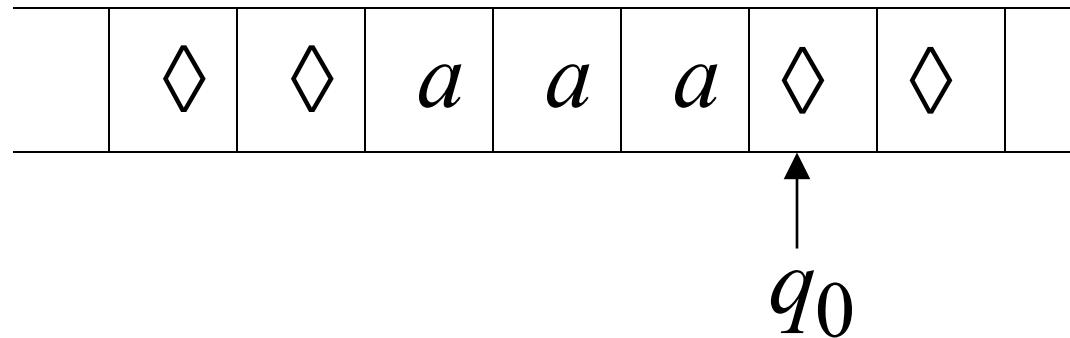
Time 1



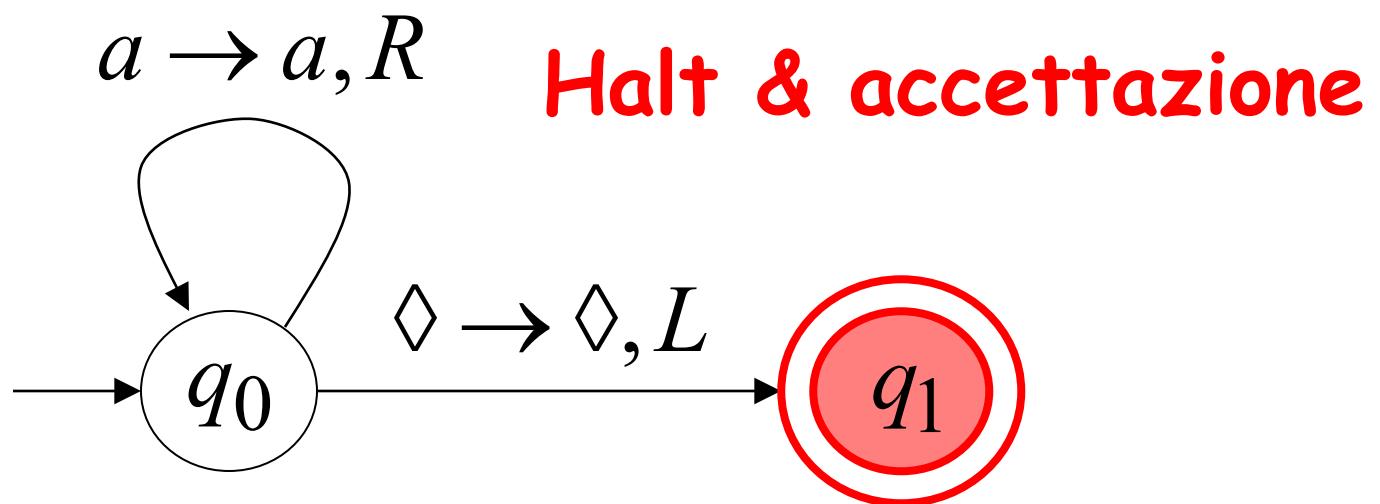
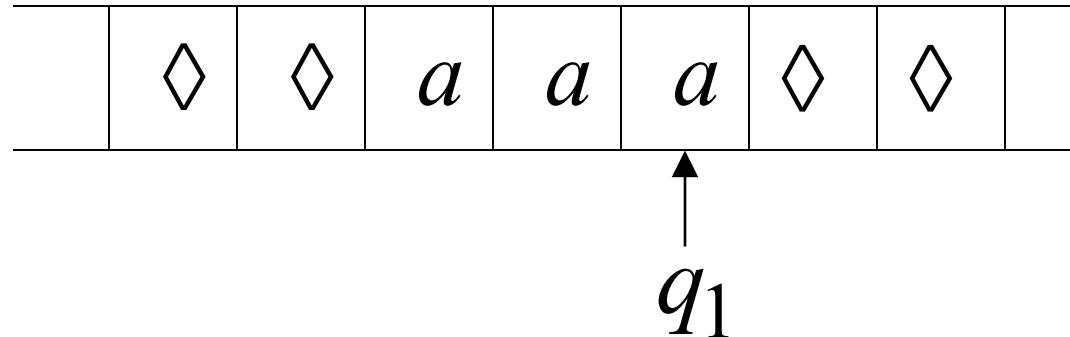
Time 2



Time 3

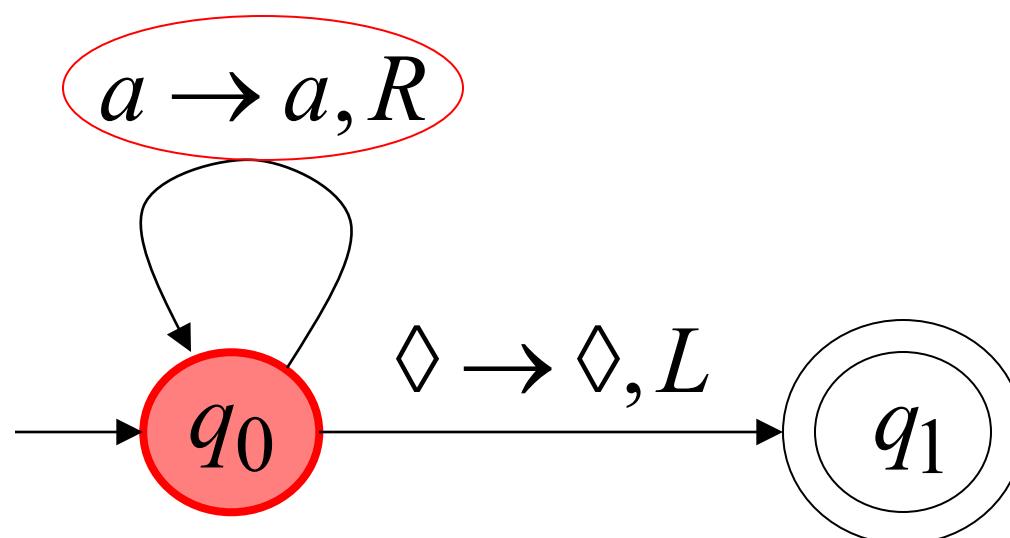
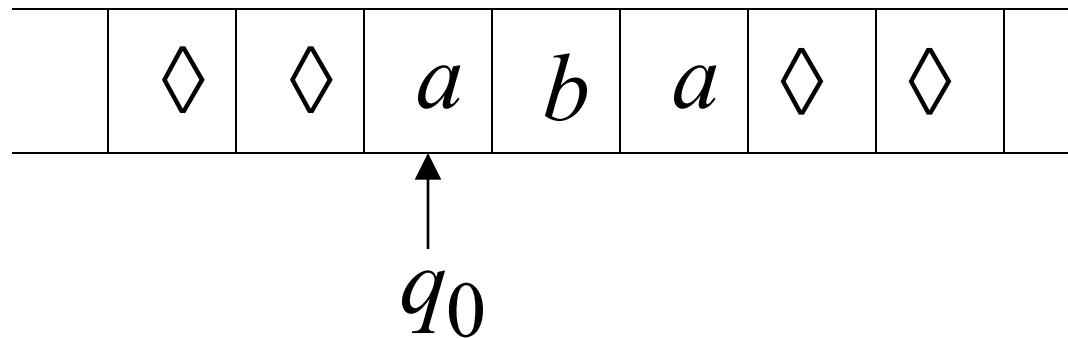


Time 4

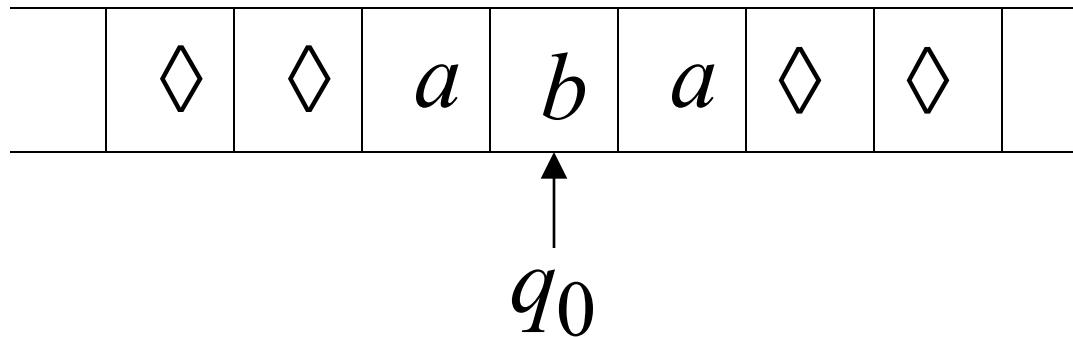


Rejection esempio

Time 0

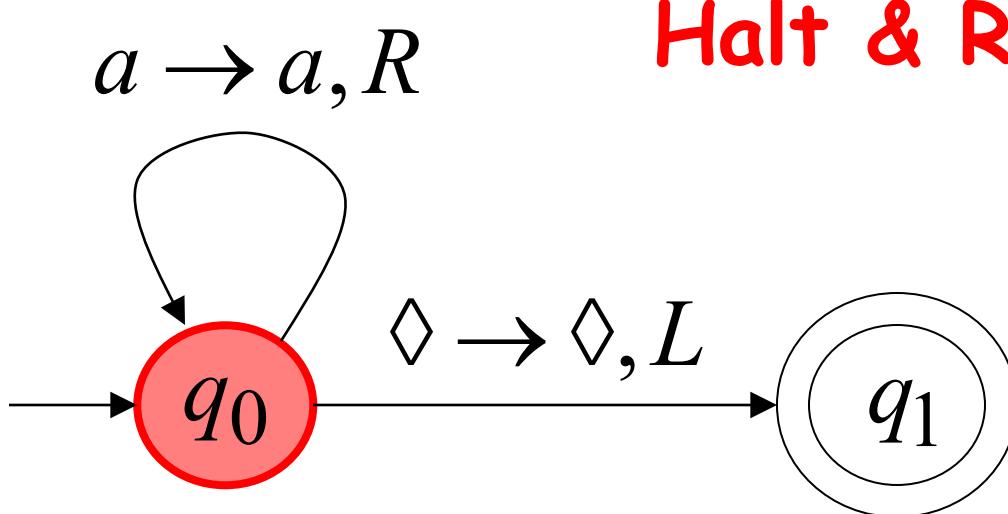


Time 1



Nessuna transizione

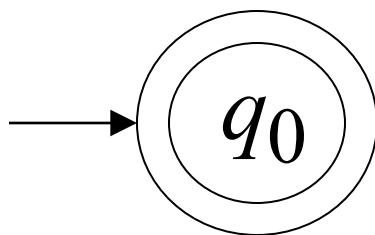
Halt & Reject



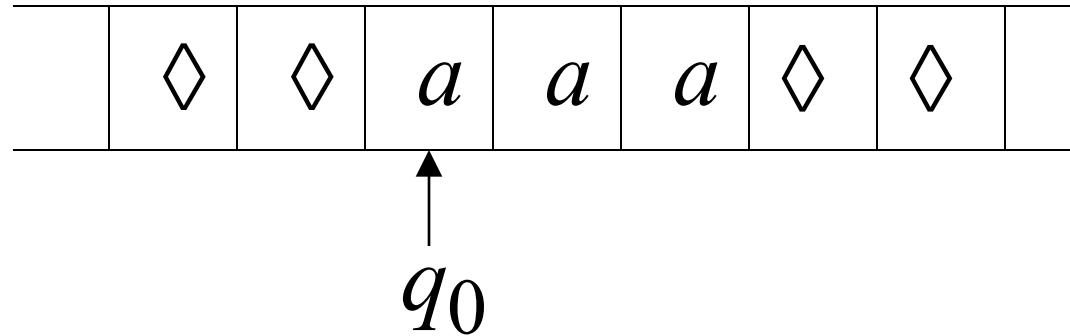
Una macchina semplice per linguaggio a^*

Ma con alfabeto di input $\Sigma = \{a\}$

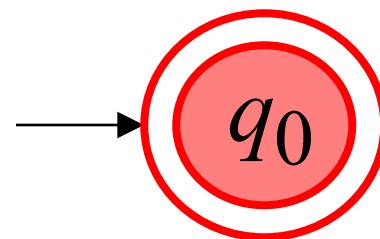
Accetta il linguaggio: a^*



Time 0



Halt & accettazione



Non è necessario esaminare l'input

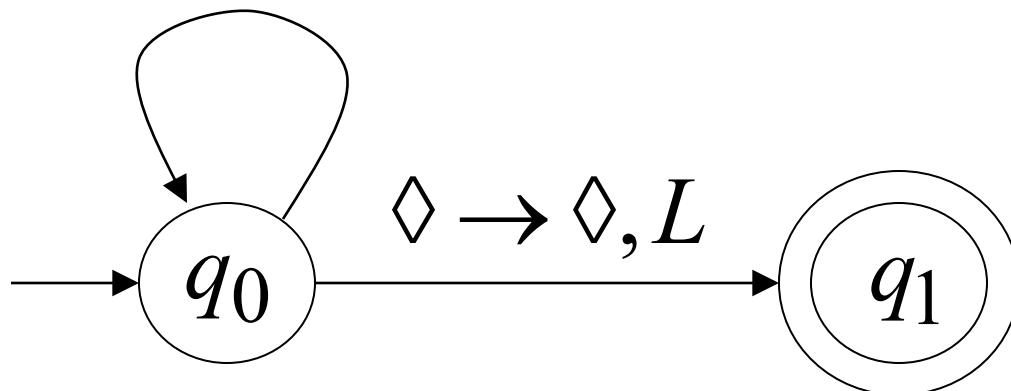
esempio Infinito Loop

una macchina di Turing

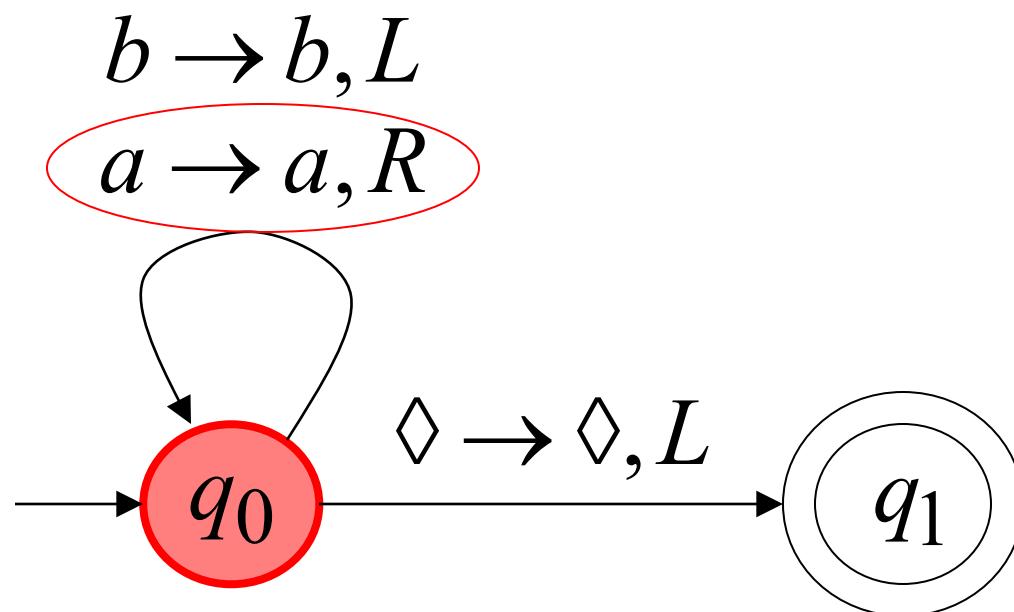
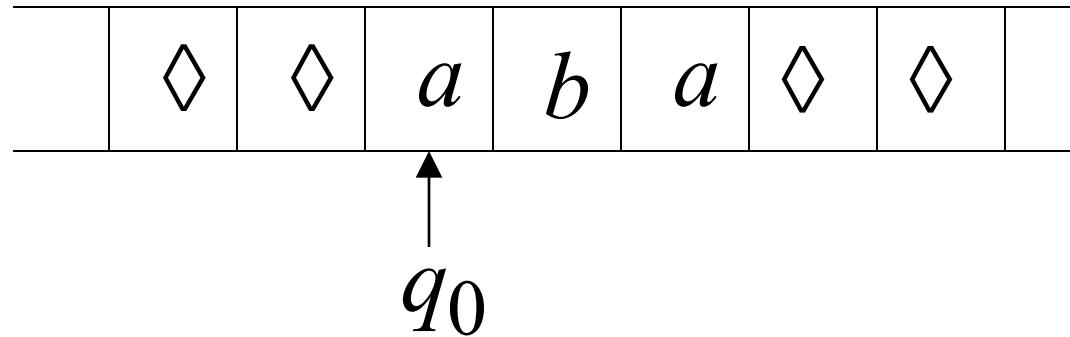
Per il linguaggio $a^* + b(a+b)^*$

$$b \rightarrow b, L$$

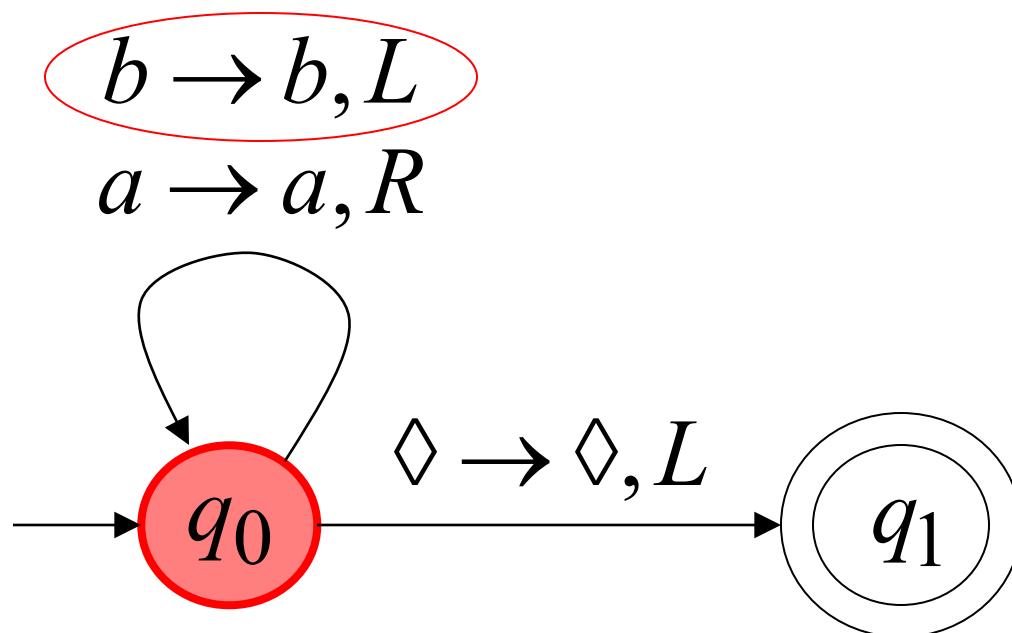
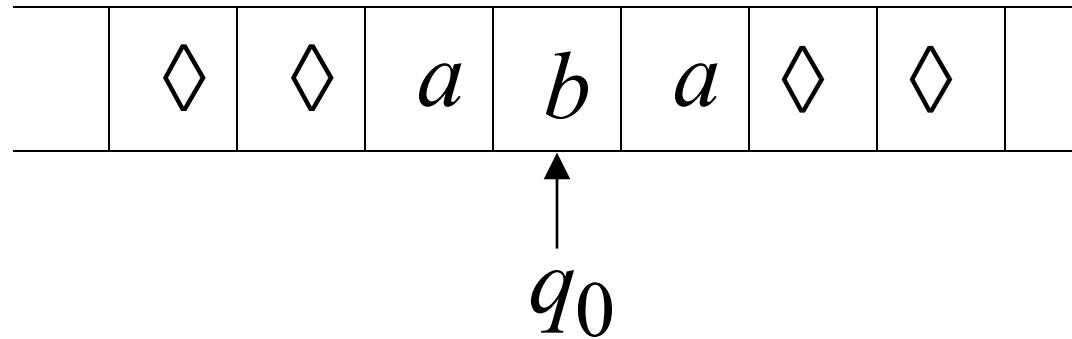
$$a \rightarrow a, R$$



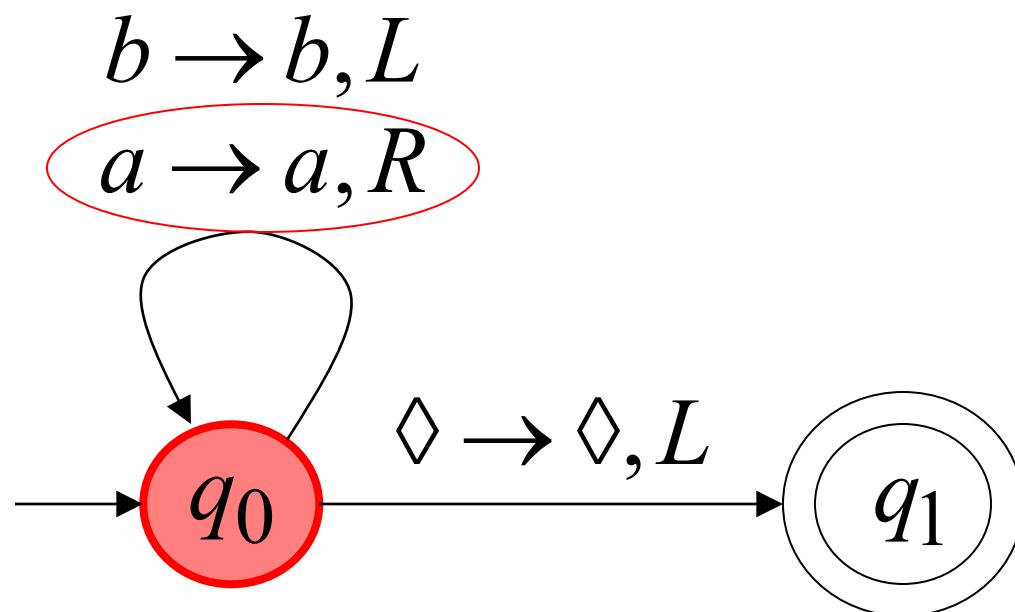
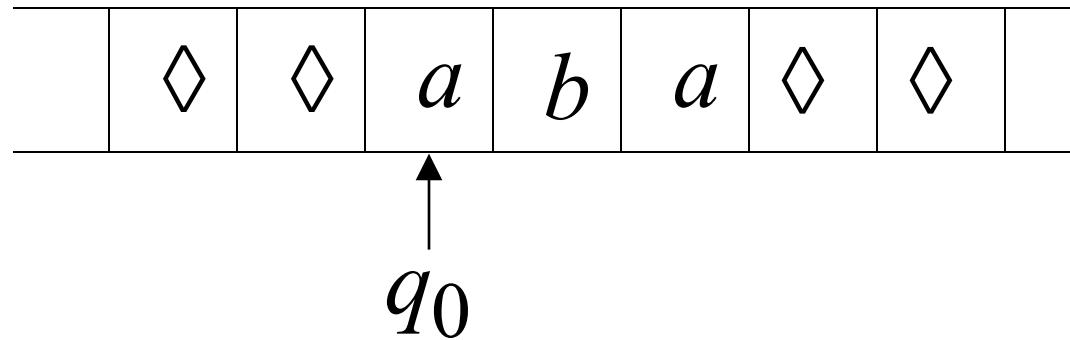
Time 0



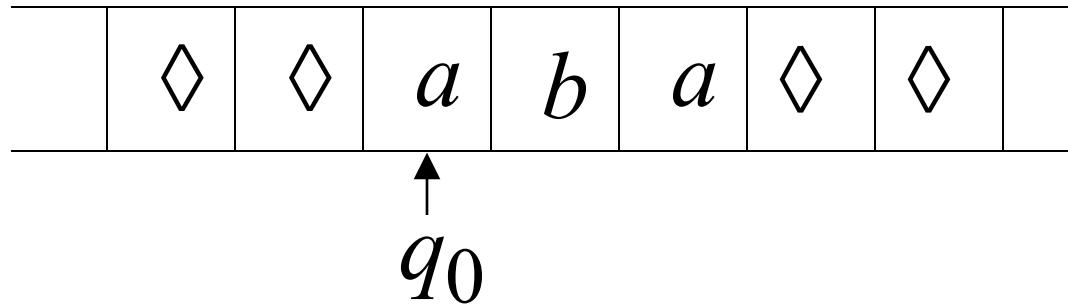
Time 1



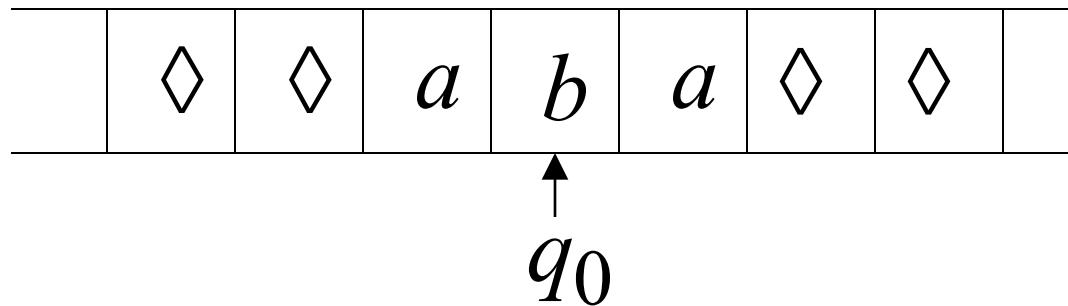
Time 2



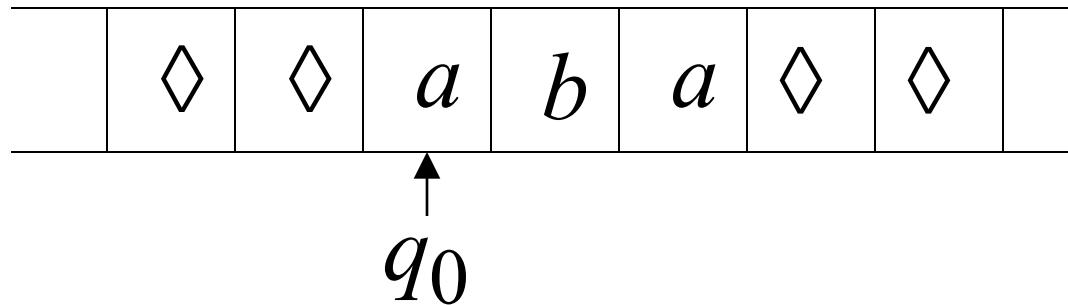
Time 2



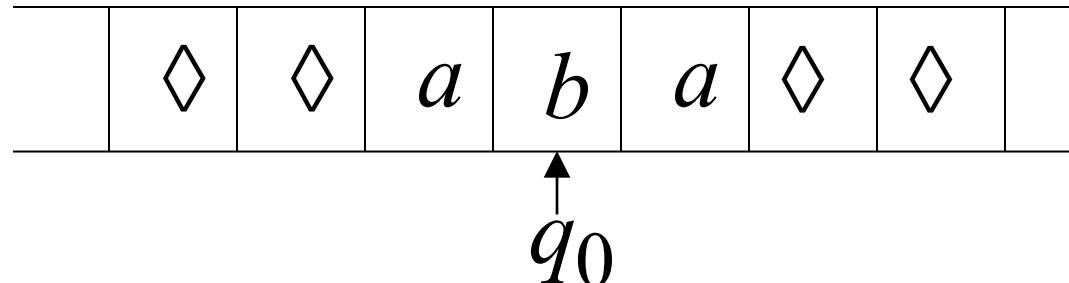
Time 3



Time 4



Time 5



Infinite loop

A causa dell' infinite loop:

- lo stato accettazione non può essere raggiunto
- La macchina non si ferma
- La stringa di input è "rejected-rigettata"

Basic Idea:

$$\{a^n b^n\}$$

Match **a** con le **b**:

Repeat:

La **a** più a sinistra cambiala in **x**

trova la **b** più a sinistra cambiala con **y**

Until non vi sono più **a o b**

Se rimane una **a** o una **b** reject

--- xxayyb

$q_0 a \rightarrow X q_1$

q_1 devo scavalcare
tutte le a tutte le Y u b
 $b \rightarrow Y$ vai stato q_2

q_2 vai a L scavalcando
tutte le Y e tutte le a
fino a raggiungere una X
Spostare R q_0

aaaabbbb

Xaaabbbb

XaaaYbbb

XXaaYbbb

XXaaYYbb

XXXaaYYbb

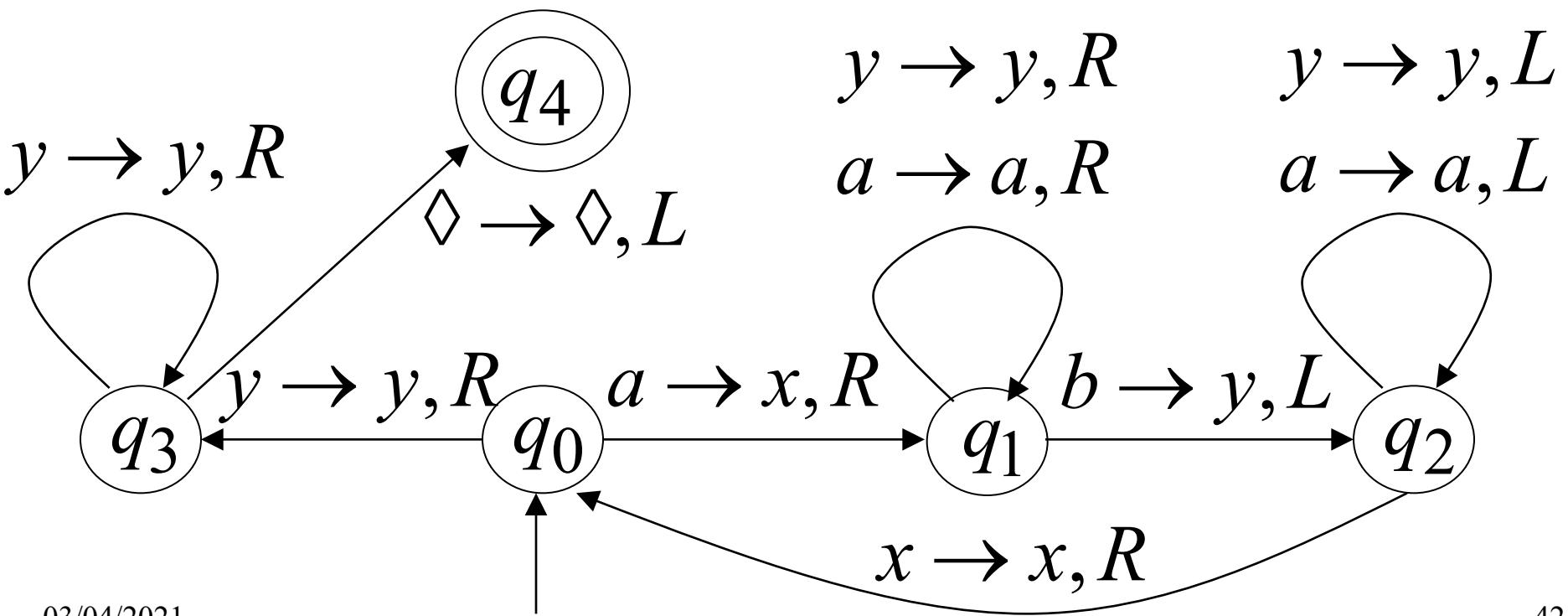
XXXaaYYYb

XXXXYYYb

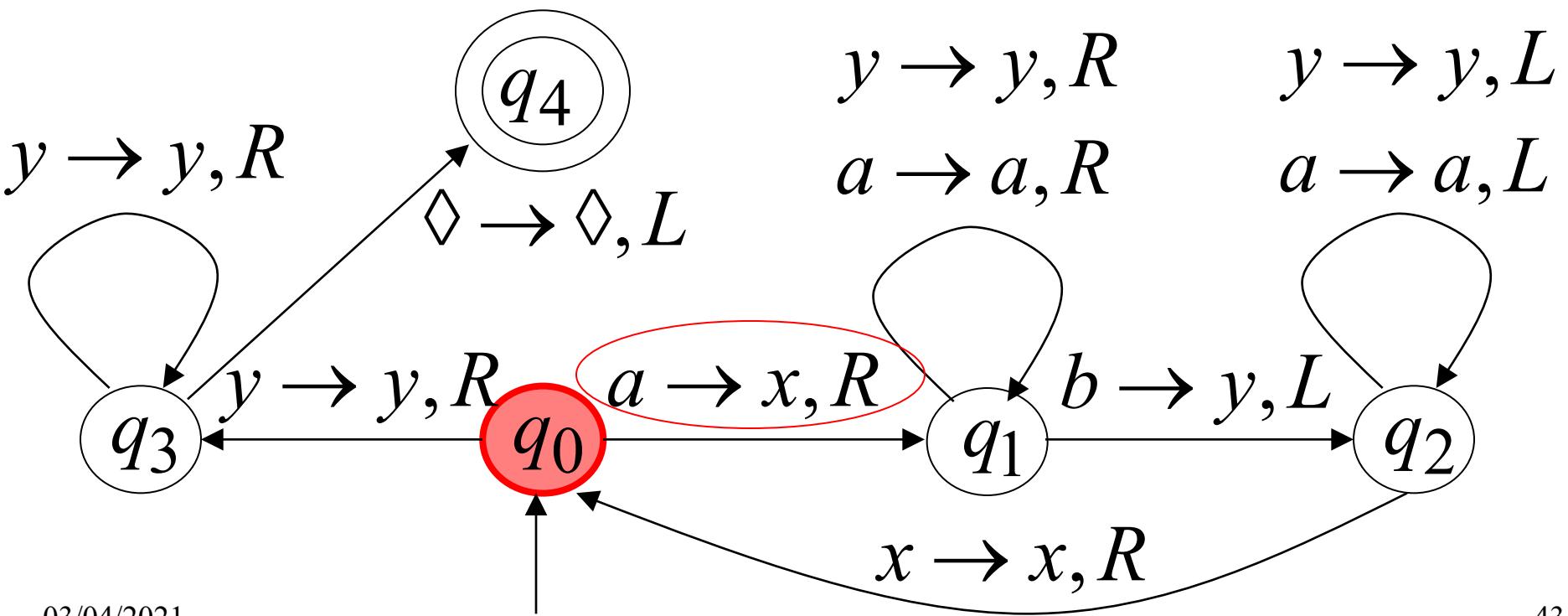
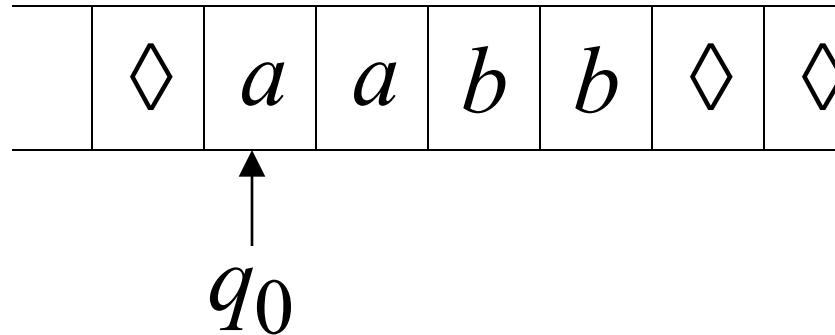
XXXXYYYb

Turing macchina esempio

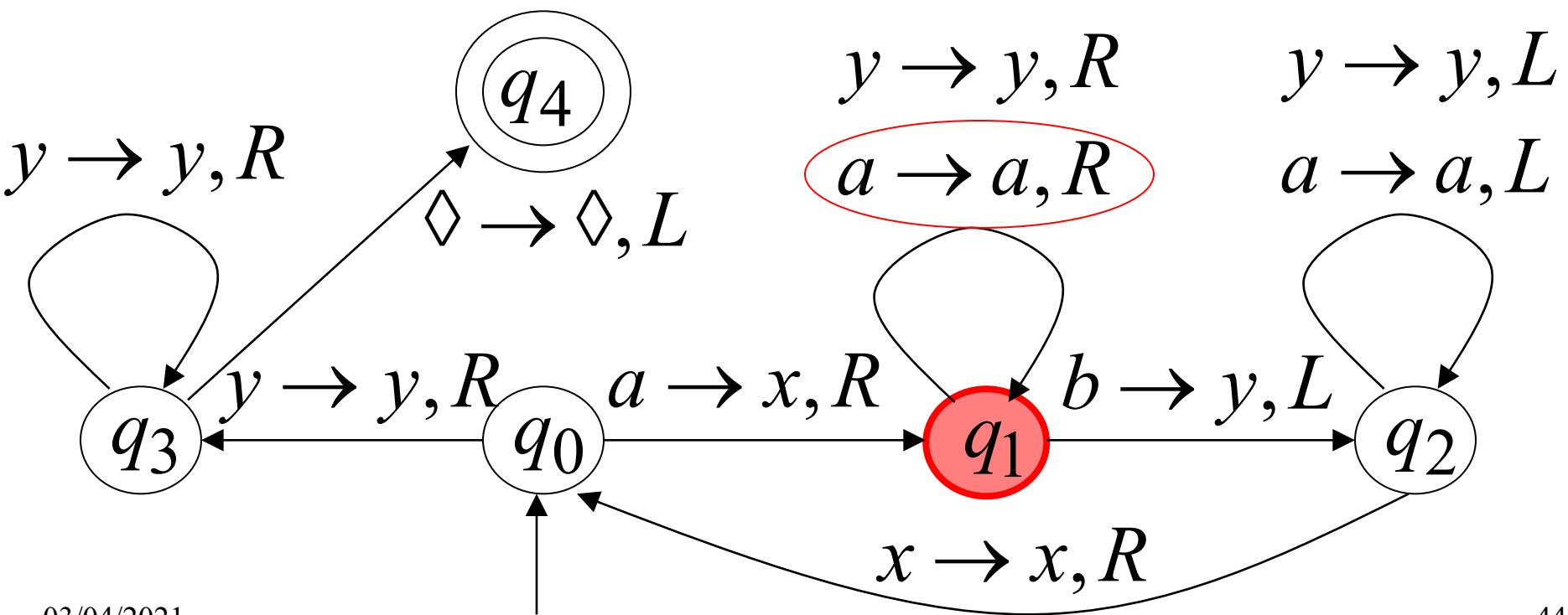
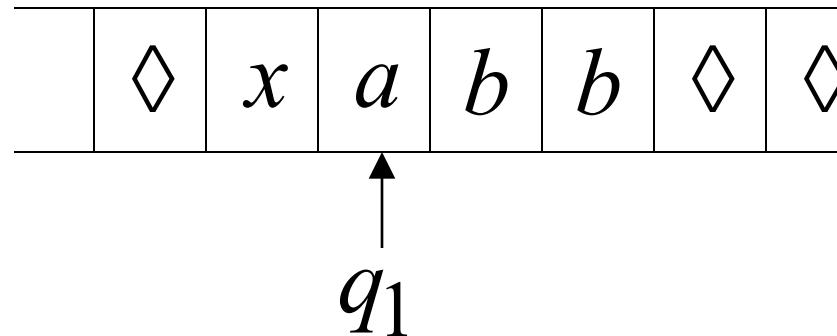
Machina di Turing per il linguaggio $\{a^n b^n\}$
 $n \geq 1$



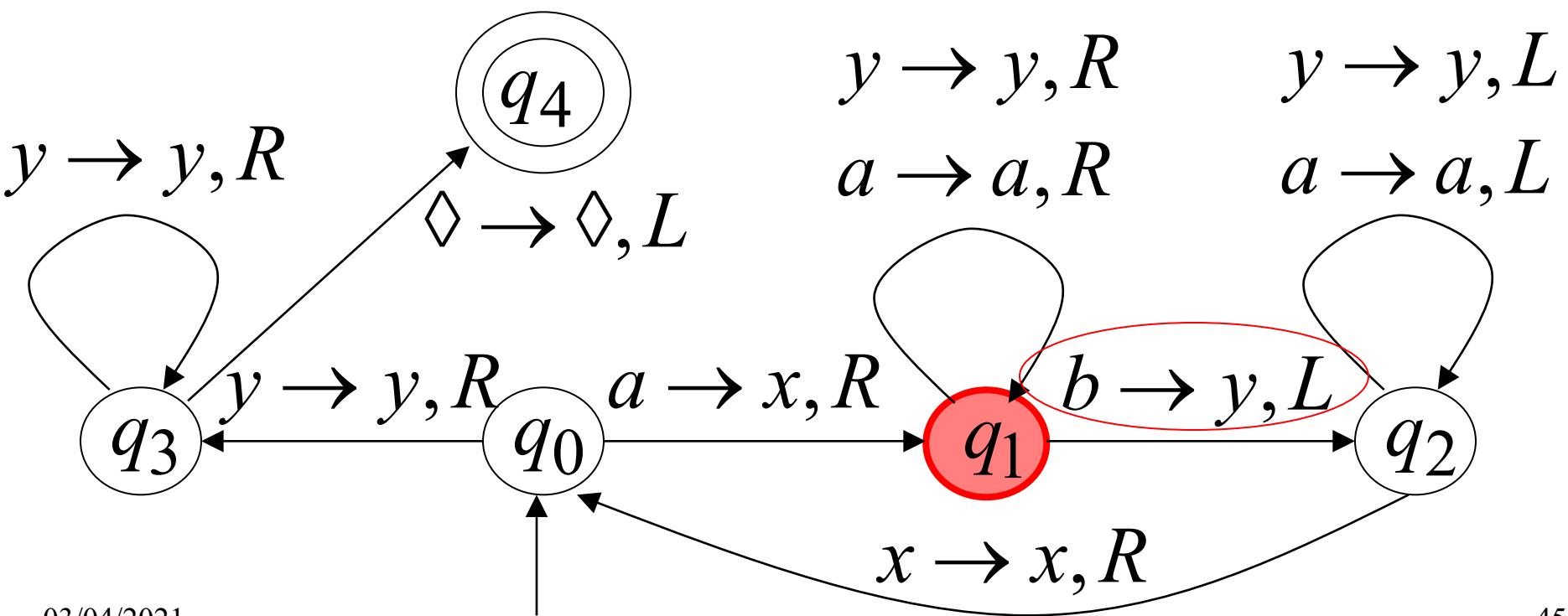
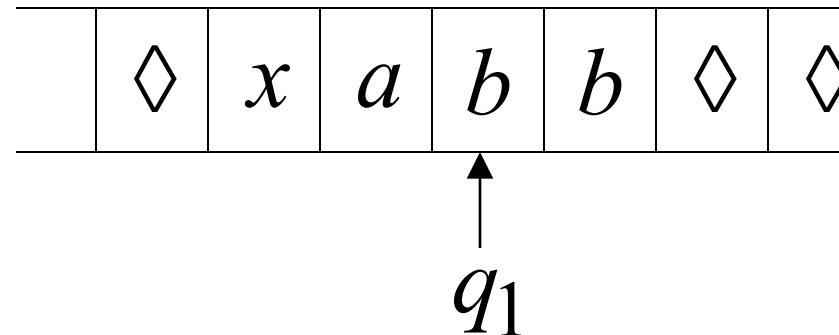
Time 0



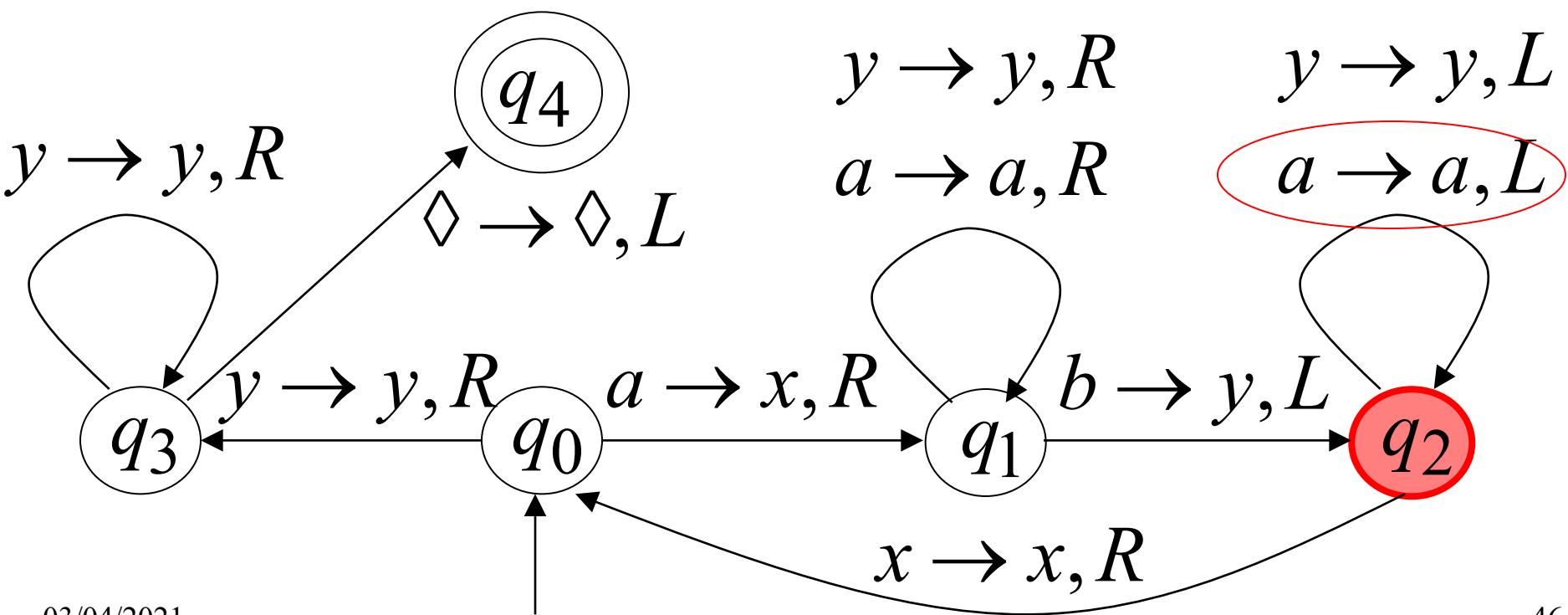
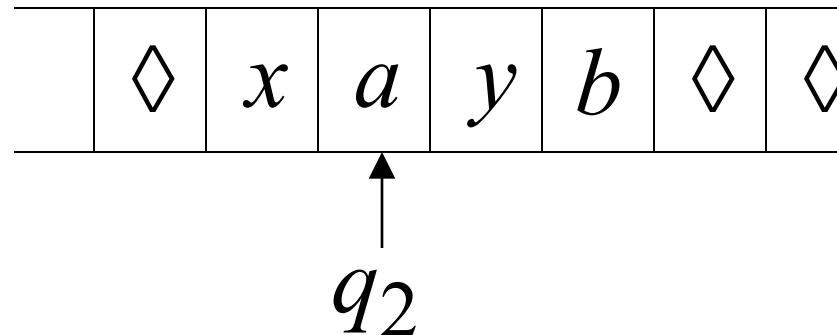
Time 1



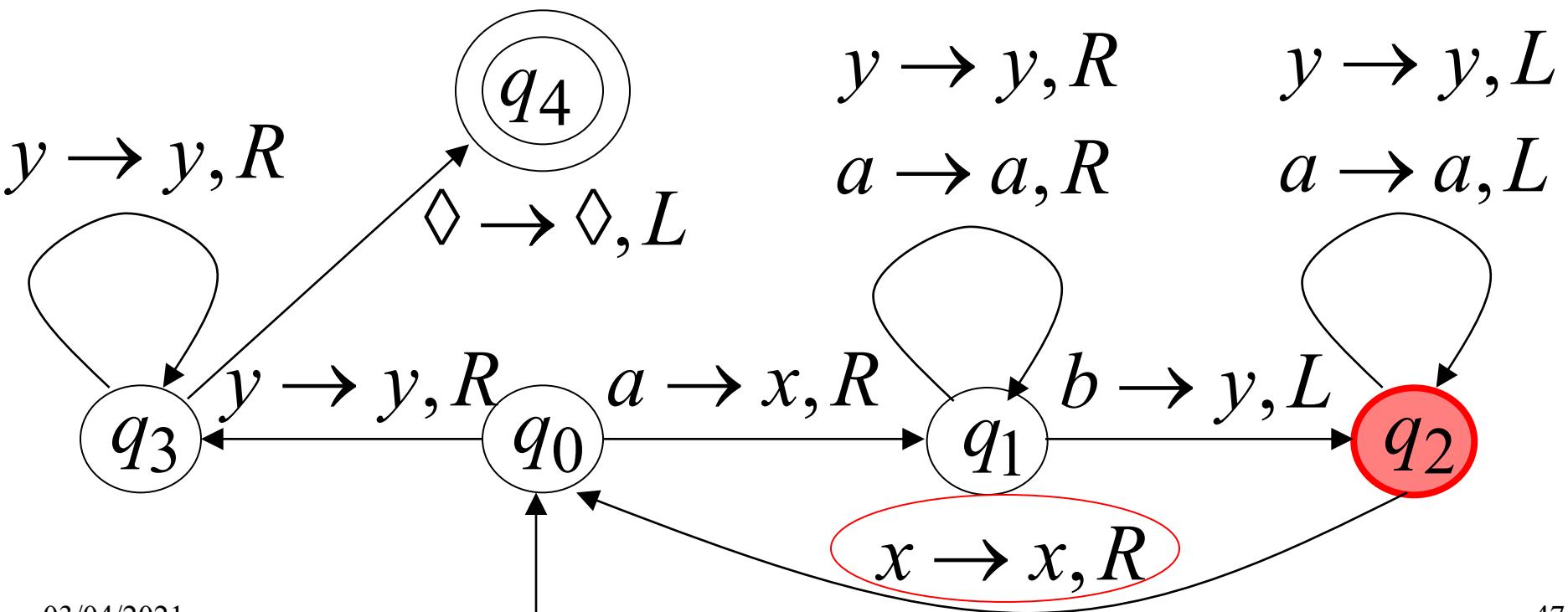
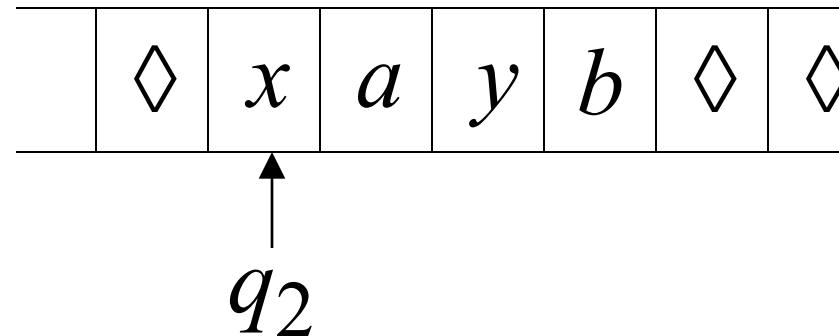
Time 2



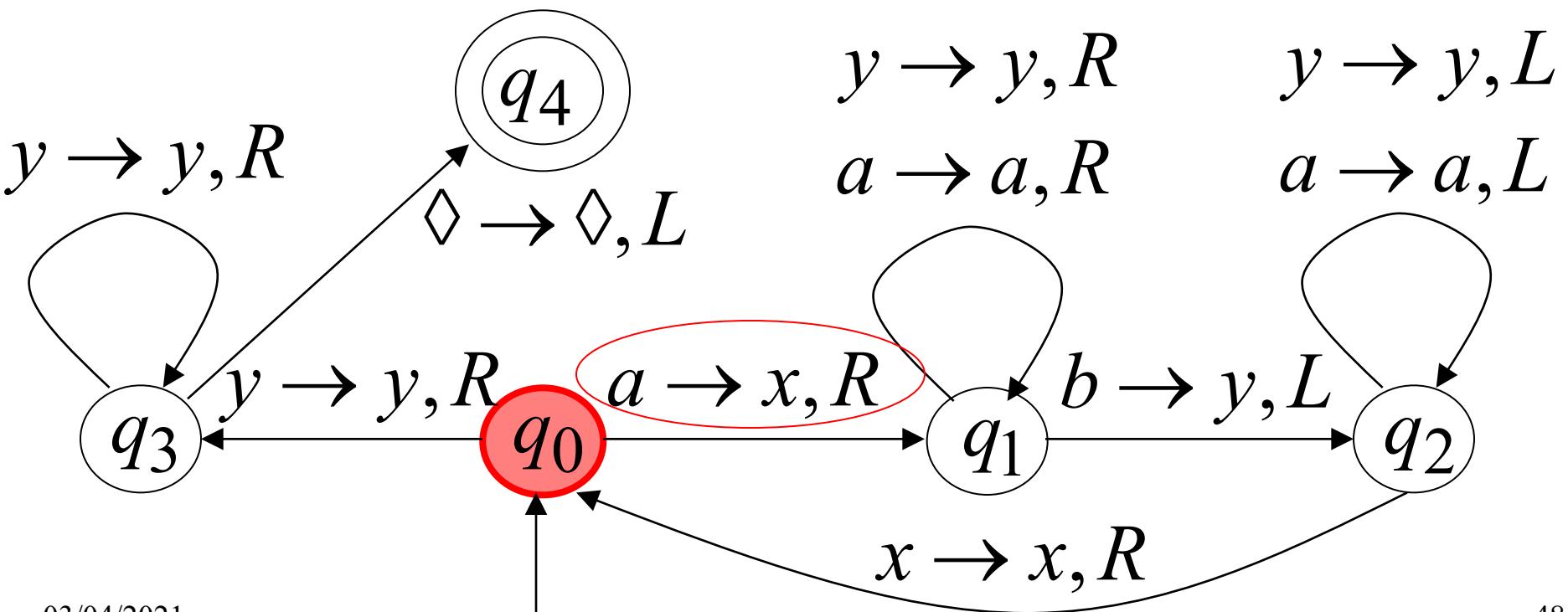
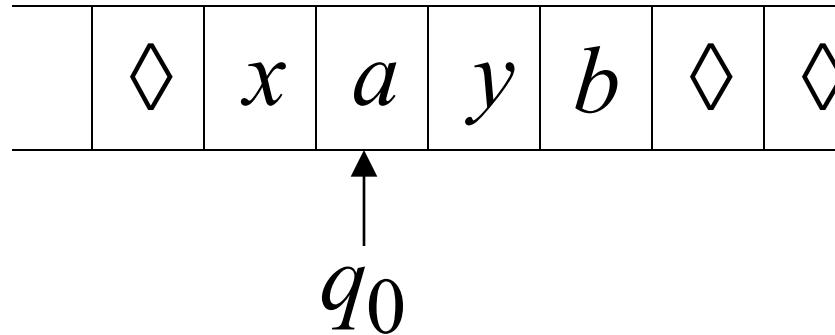
Time 3



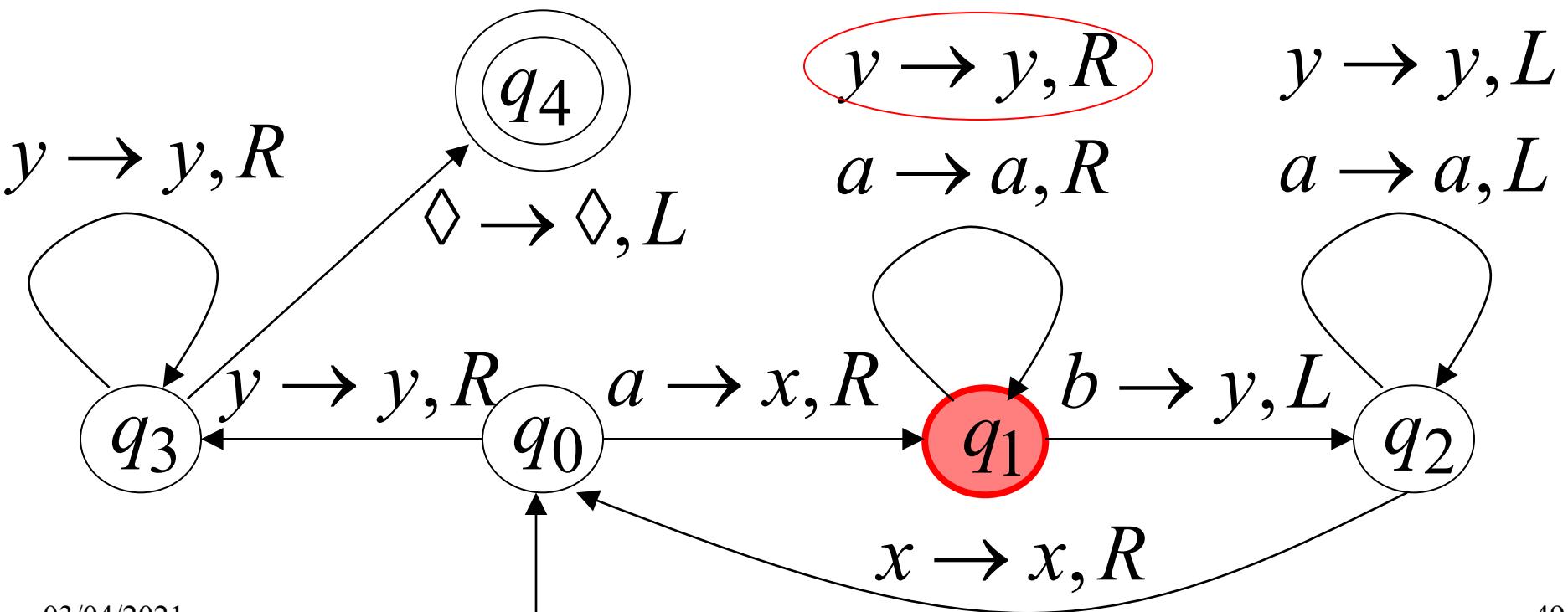
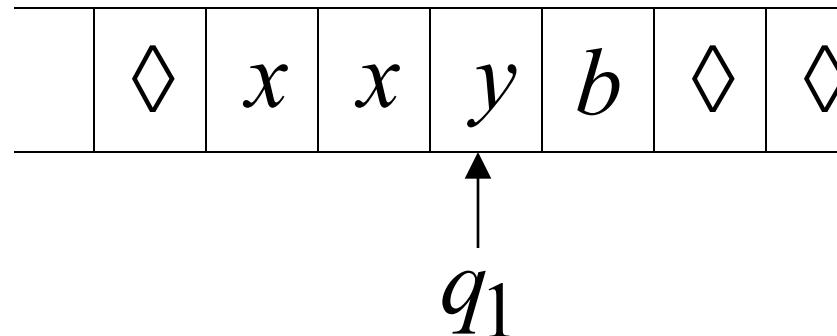
Time 4



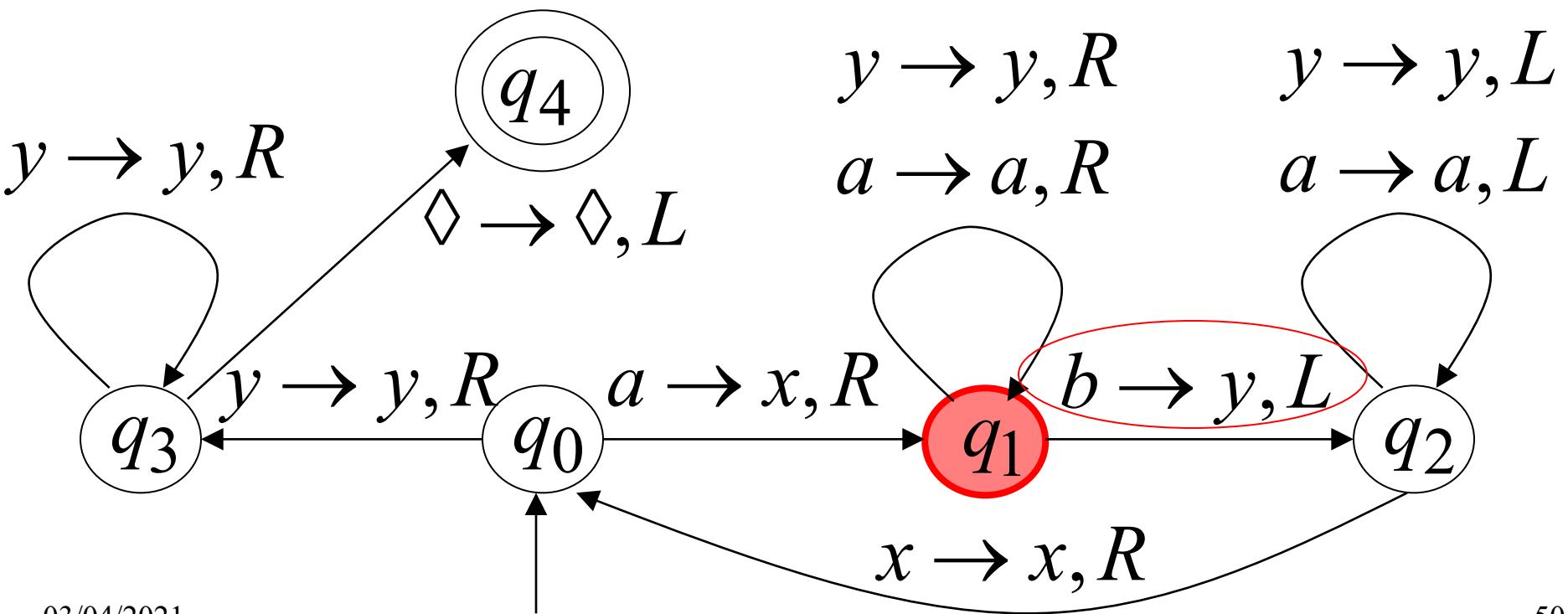
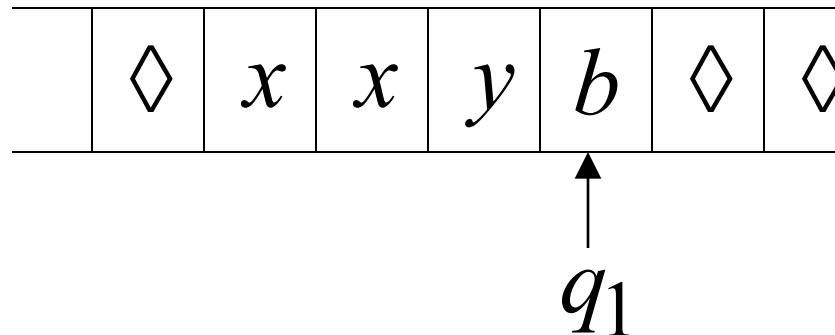
Time 5



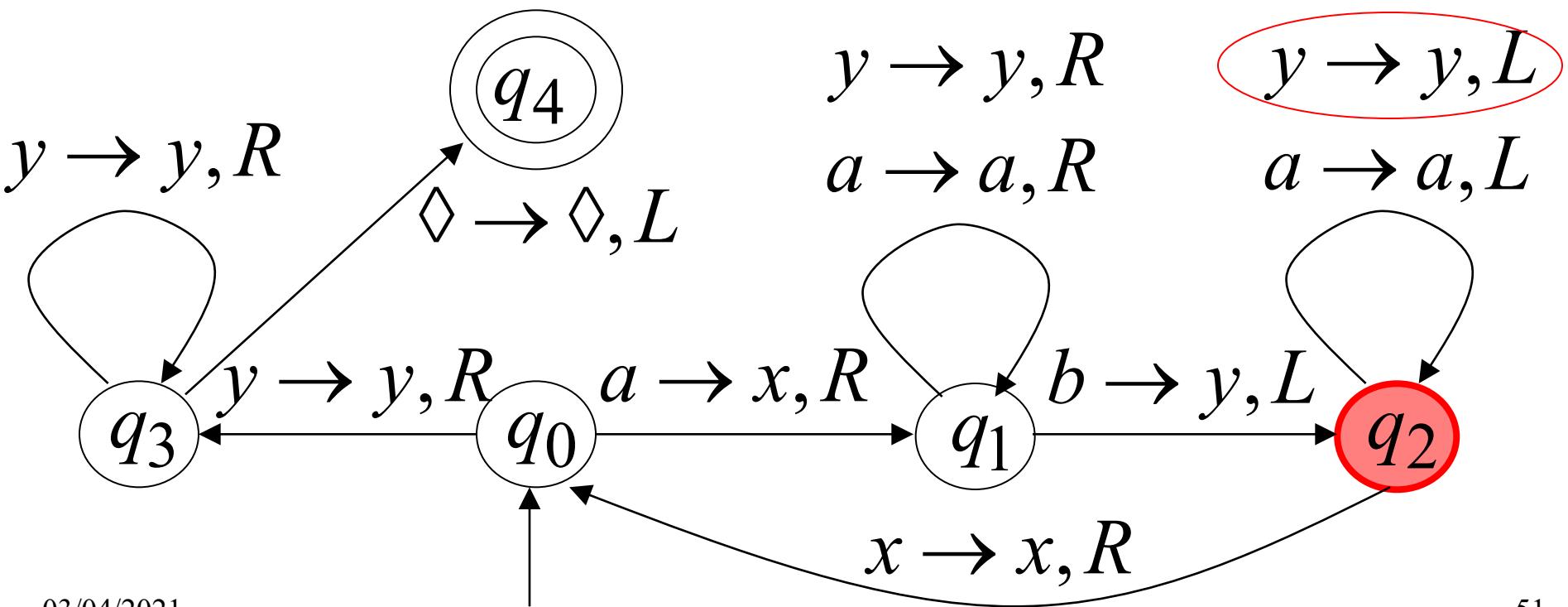
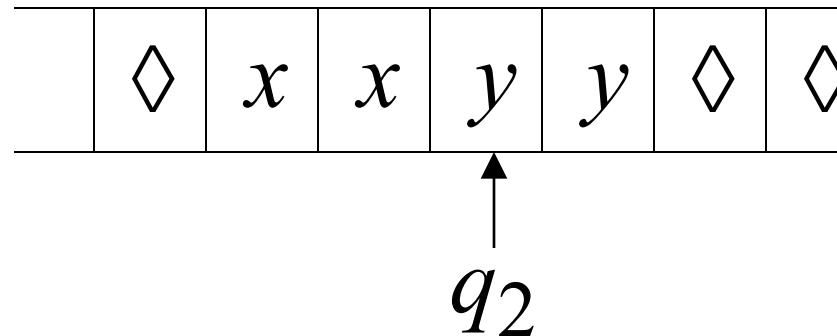
Time 6



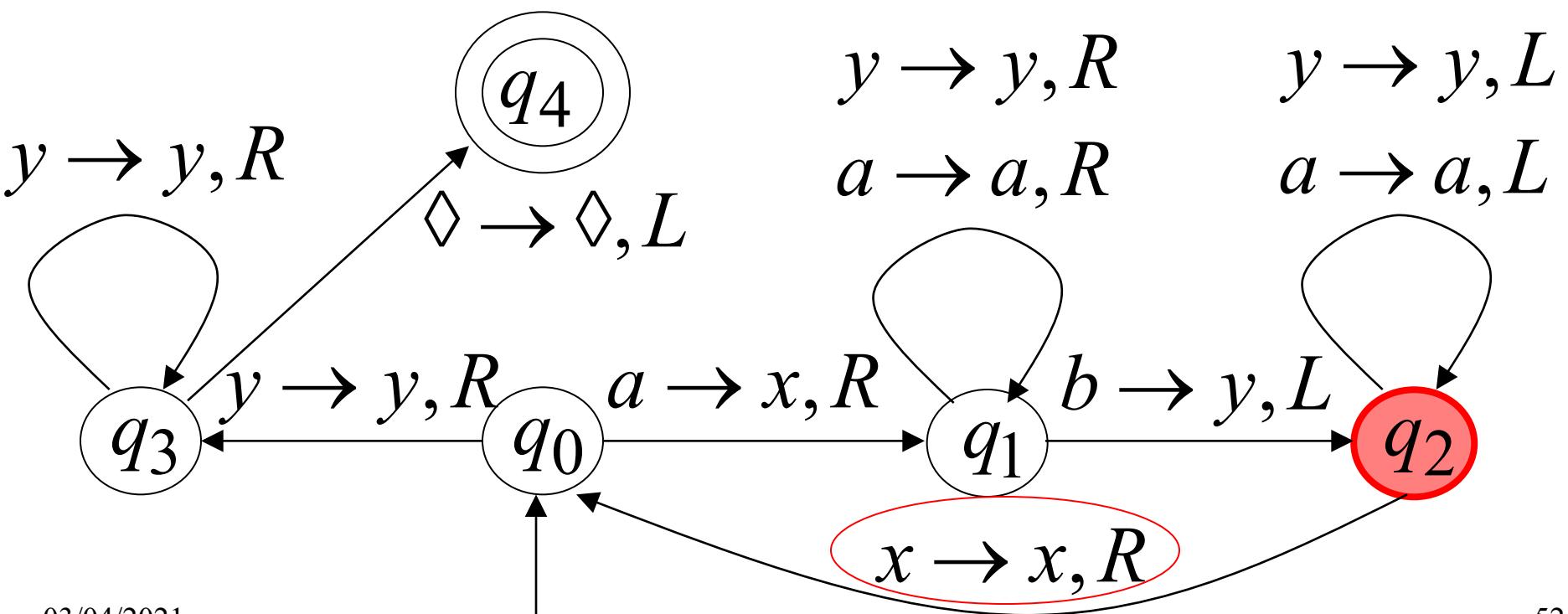
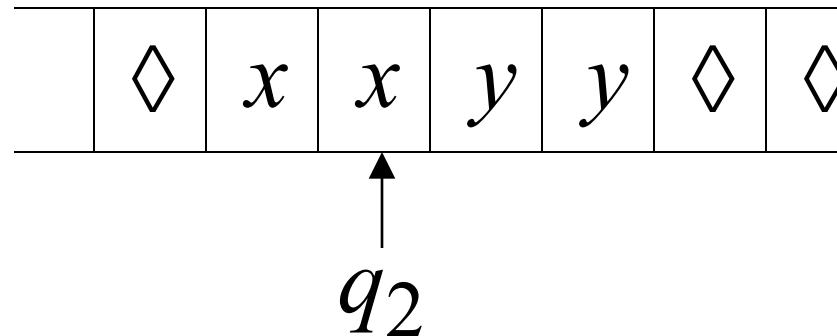
Time 7



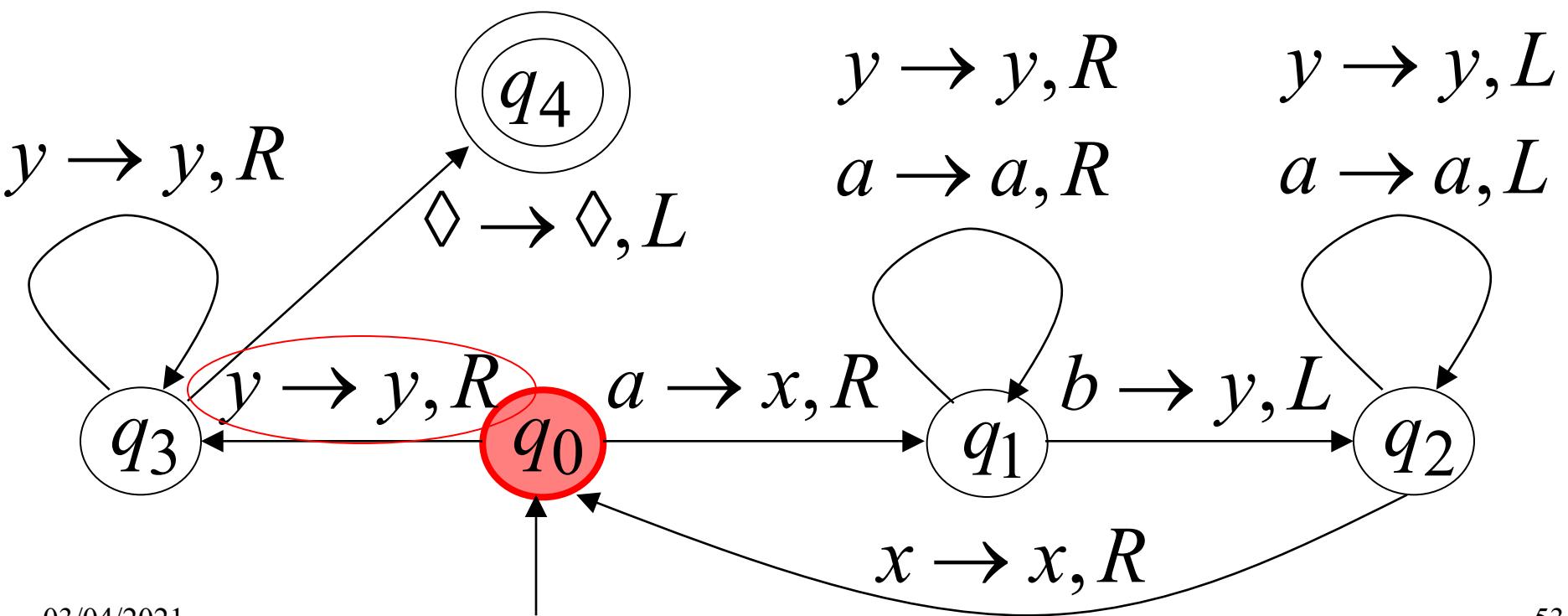
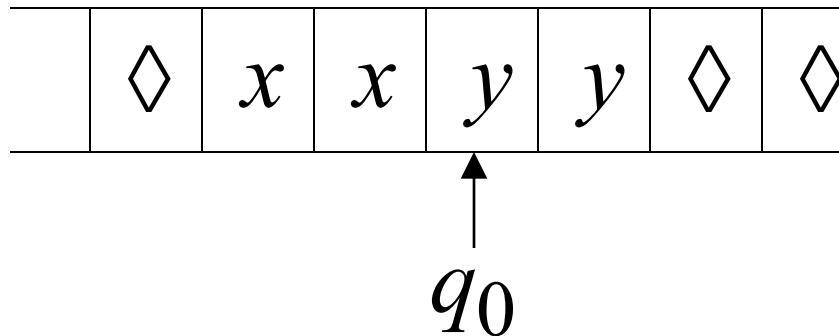
Time 8



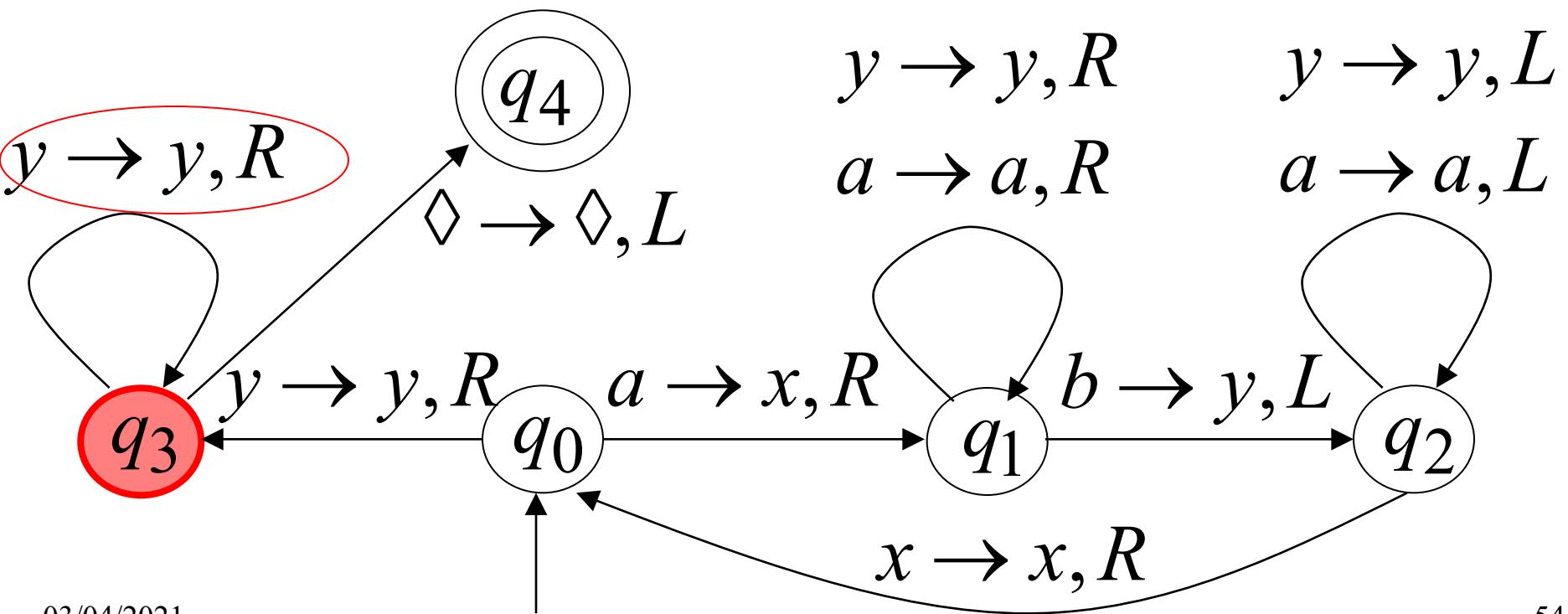
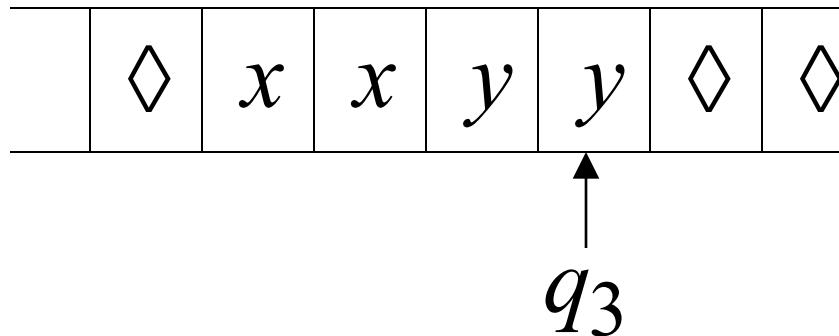
Time 9



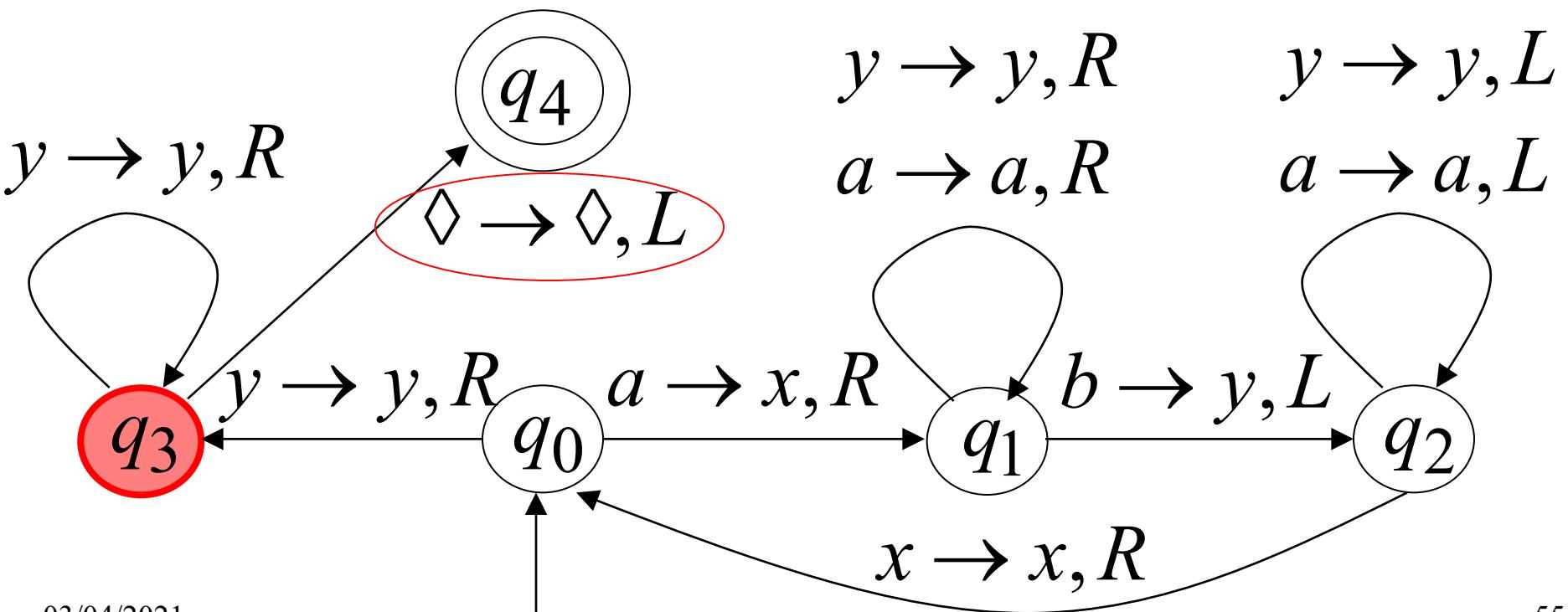
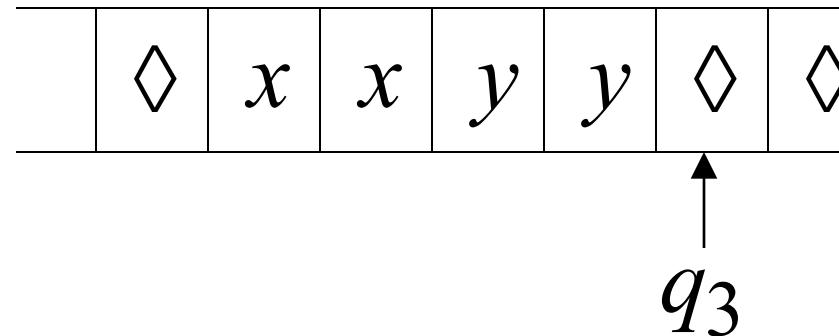
Time 10



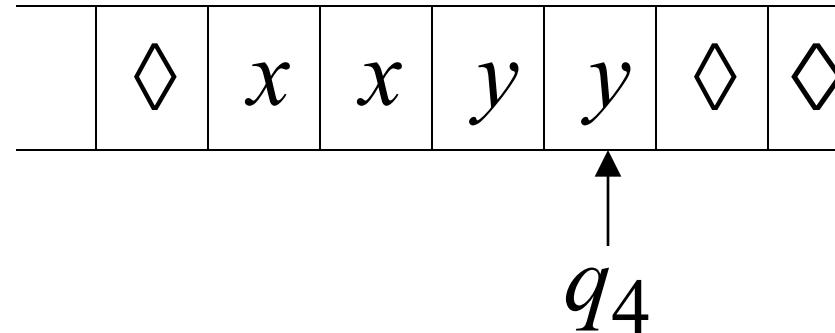
Time 11



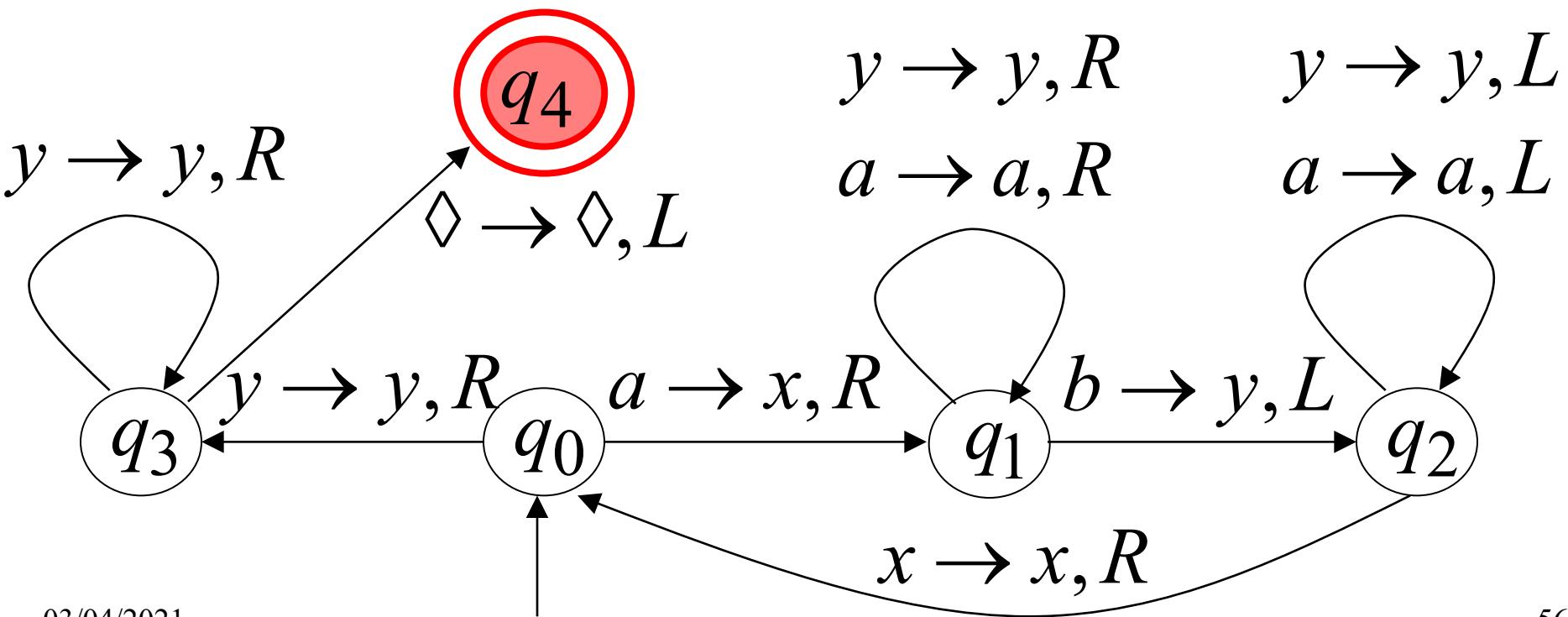
Time 12



Time 13



Halt & accettazione



Osservazione:

Se modifichiamo

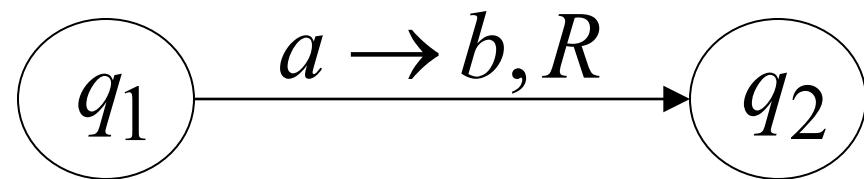
La macchina per il linguaggio $\{a^n b^n\}$

Facilmente possiamo costruire

Una macchina per il linguaggio $\{a^n b^n c^n\}$

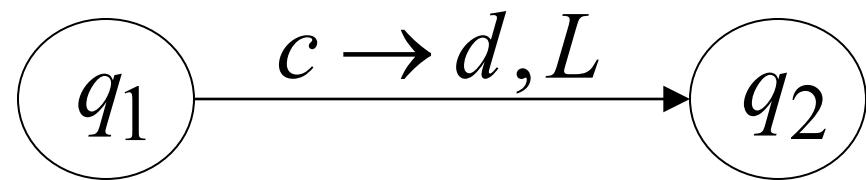
Definizione formale di macchina di turing

Funzione Transizione



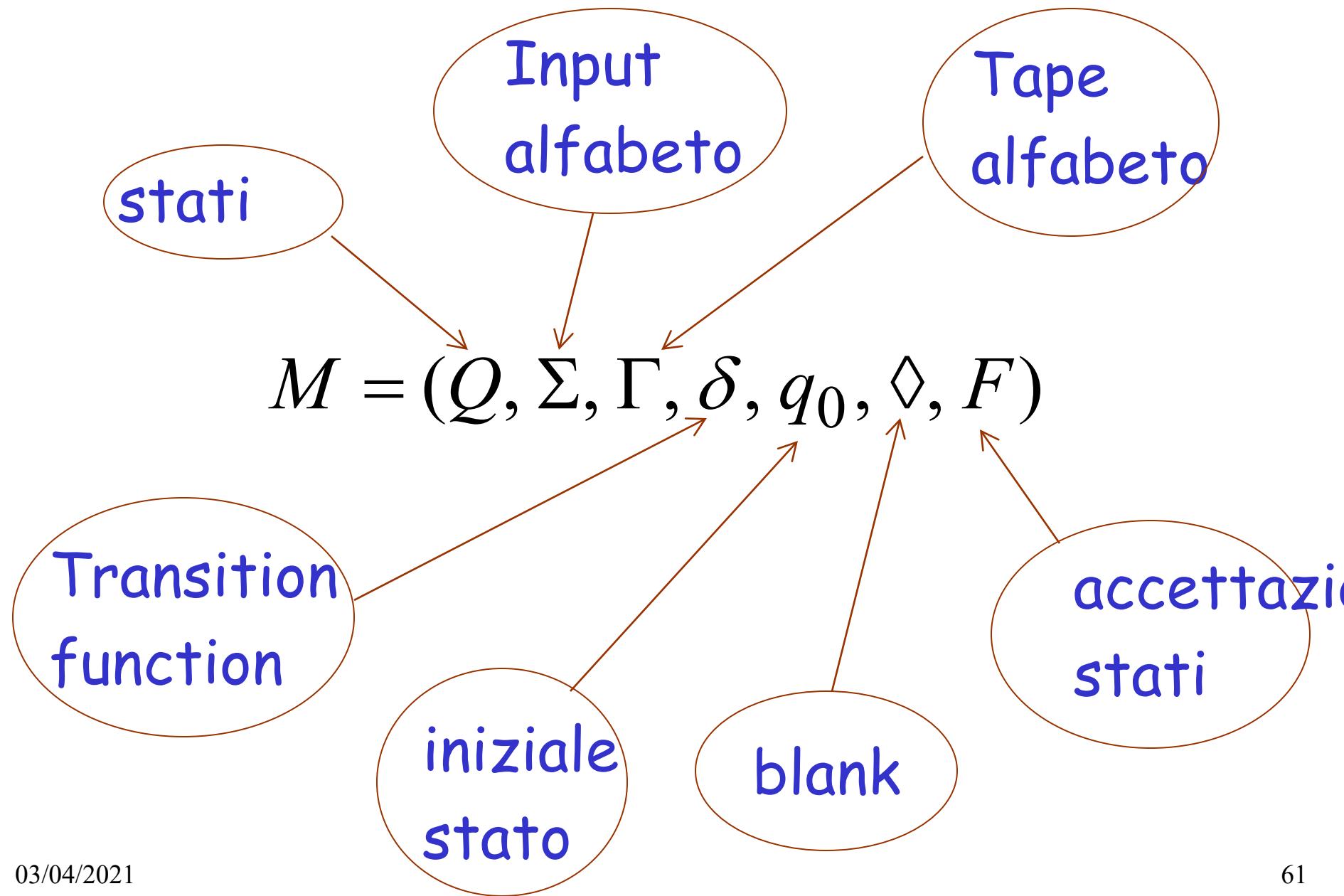
$$\delta(q_1, a) = (q_2, b, R)$$

Funzione Transizione



$$\delta(q_1, c) = (q_2, d, L)$$

Turing macchina:



Tipo della delta

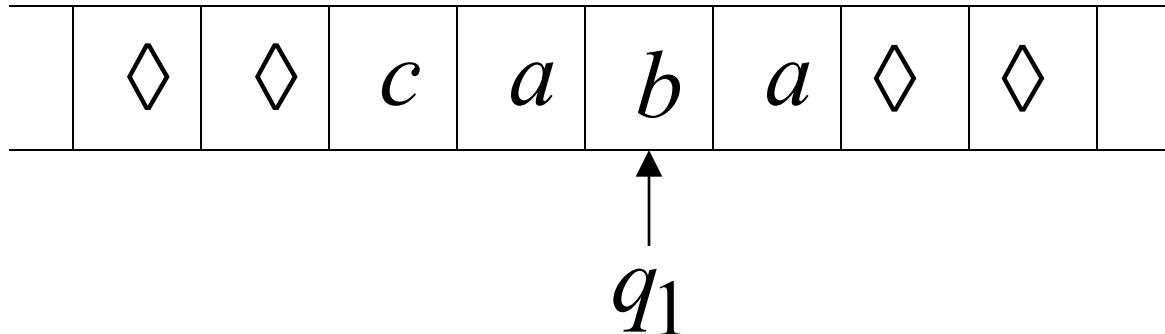
Input

Stati \times AI U AN ->

Stati \times AI U AN \times Op

Op = L, R

Configurazione

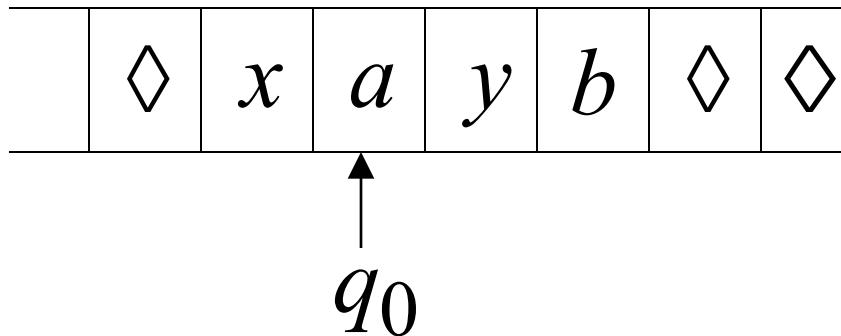
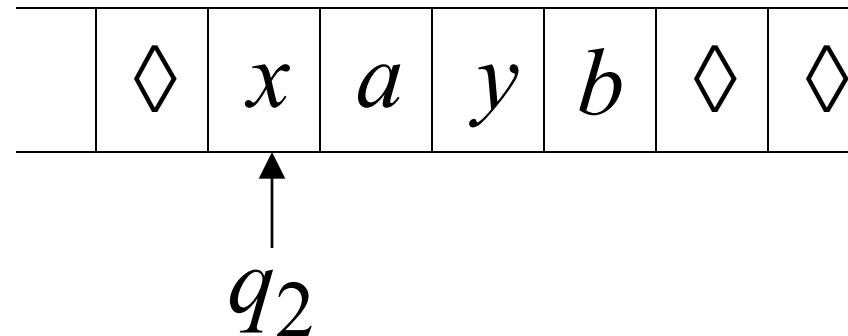


descrizione istantanea:

$ca\ q_1\ ba$

Time 4

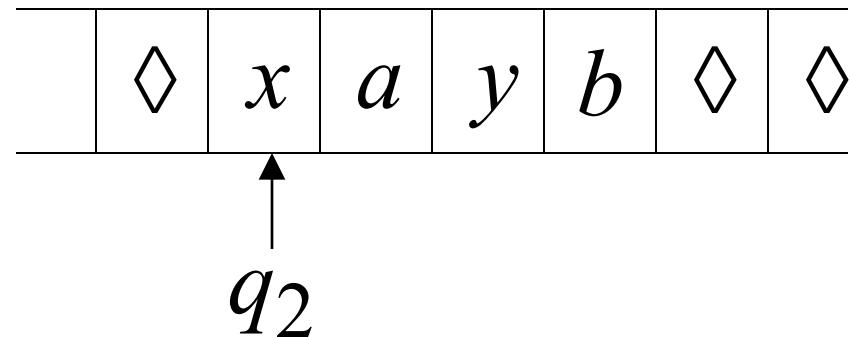
Time 5



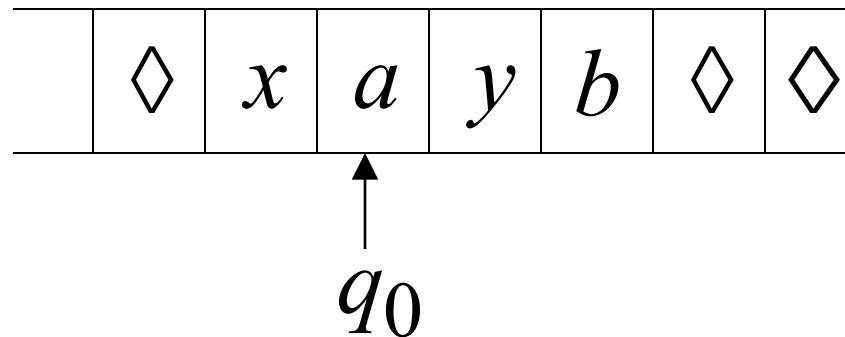
Una mossa $q_2 \ xayb \succ x \ q_0 \ ayb$

(dà)

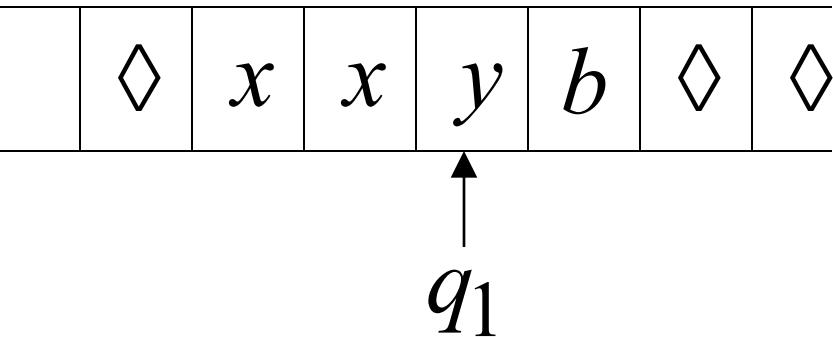
Time 4



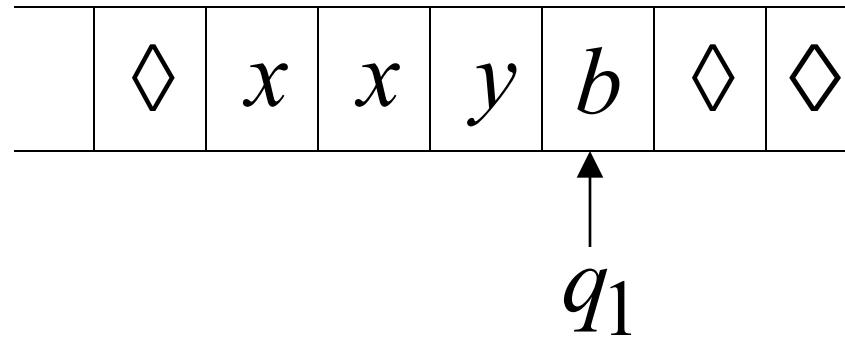
Time 5



Time 6



Time 7



Un calcolo

$q_2 \ xayb \succ x q_0 \ ayb \succ xx q_1 \ yb \succ xxy q_1 \ b$

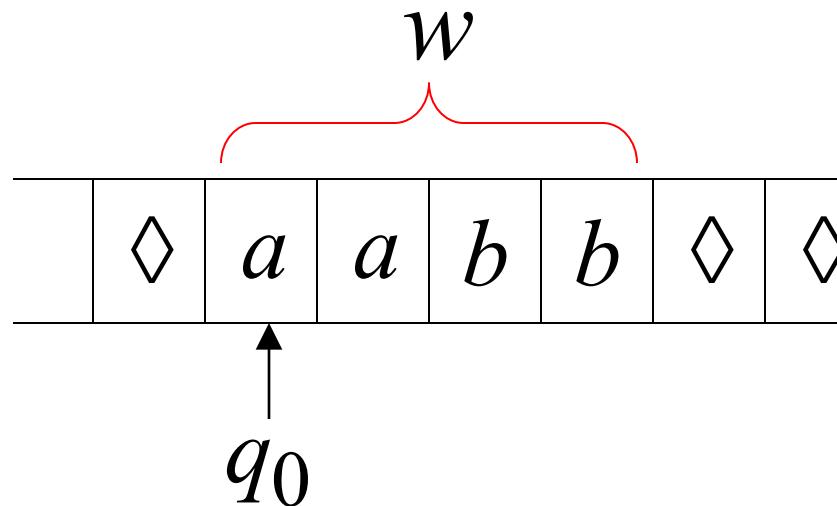
$$q_2 \ xayb \succ x \ q_0 \ ayb \succ xx \ q_1 \ yb \succ xxy \ q_1 \ b$$

*

Notazione equivalente: $q_2 \ xayb \succ^* xxy \ q_1 \ b$

configurazione Iniziale : q_0 w

stringa di input



il linguaggio accettato

Per ogni macchina Turing M

$$L(M) = \{w : q_0 w \xrightarrow{*} x_1 q_f x_2\}$$

Accettato in forma
standard
 $q_0 w \xrightarrow{*} q_f$
stato iniziale

*

accettazione stato

Se un linguaggio L è accettato da
Una macchina di Turing M
A noi diciamo che L è:

- Turing Riconoscibile

Alfabeto

Altri nomi usati:

- Turing accettati
- Recursivamente Enumerabili

Alfabeto L definito a partire da
quell'alfabeto

L turing riconoscibile

w elemento A^* $M(w)$ raggiunge
lo stato finale se w appartiene
ad L

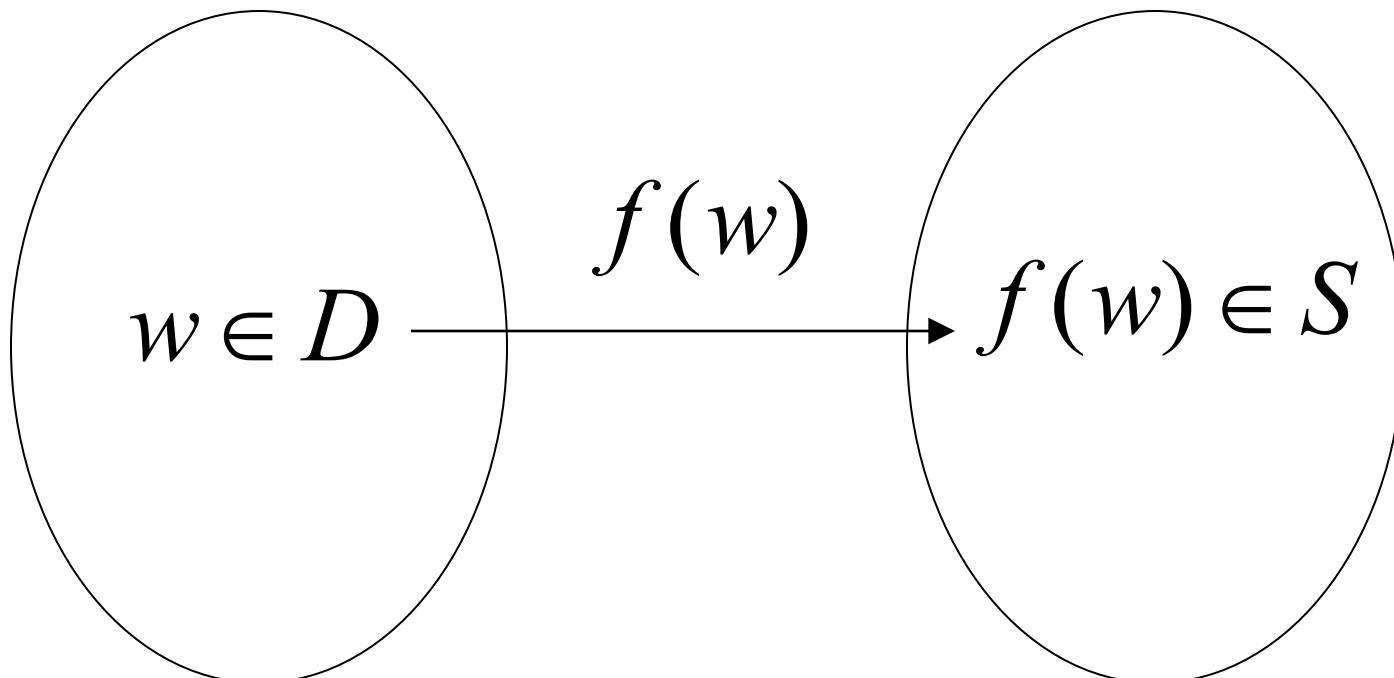
non lo raggiunge ? Altrimenti
Non raggiunge uno stato finale.
Ma questo non vuol dire che la
macchina si ferma

Calcolare funzioni con macchine di Turing

Una funzione $f(w)$ ha:

Dominio: D

Regione dei risultati: S



Una funzione può avere molti parametri:

esempio: funzione Addizione

$$f(x, y) = x + y$$

dominio degli interi

Decimali: 5

Binari: 101

Unario: 11111

0->1

1->11

n-> n+1 1

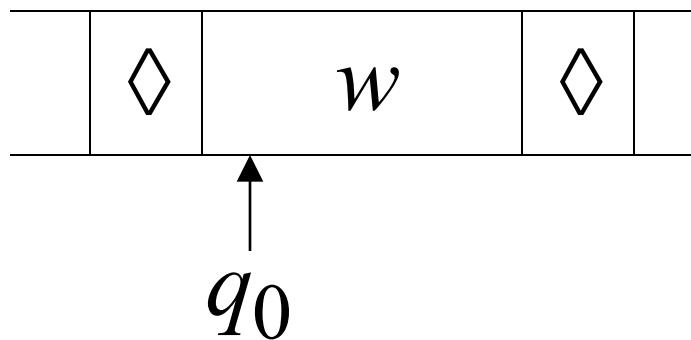
Useremo rappresentazione **unaria** :

Più facile da usare con le macchine di Turing

Definizione:

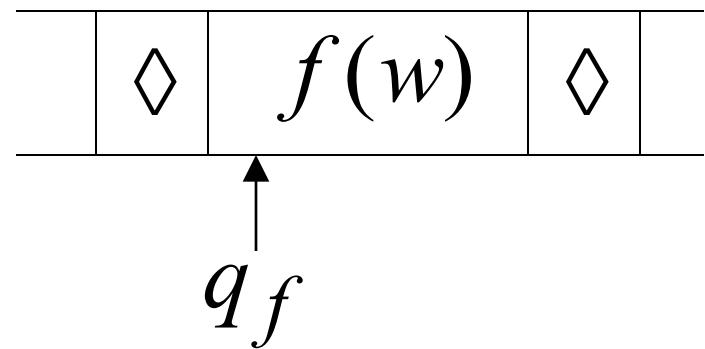
Una funzione f è calcabile se
vi è una macchina di Turing M tale che:

Configurazione iniziale



stato iniziale

Configurazione Finale



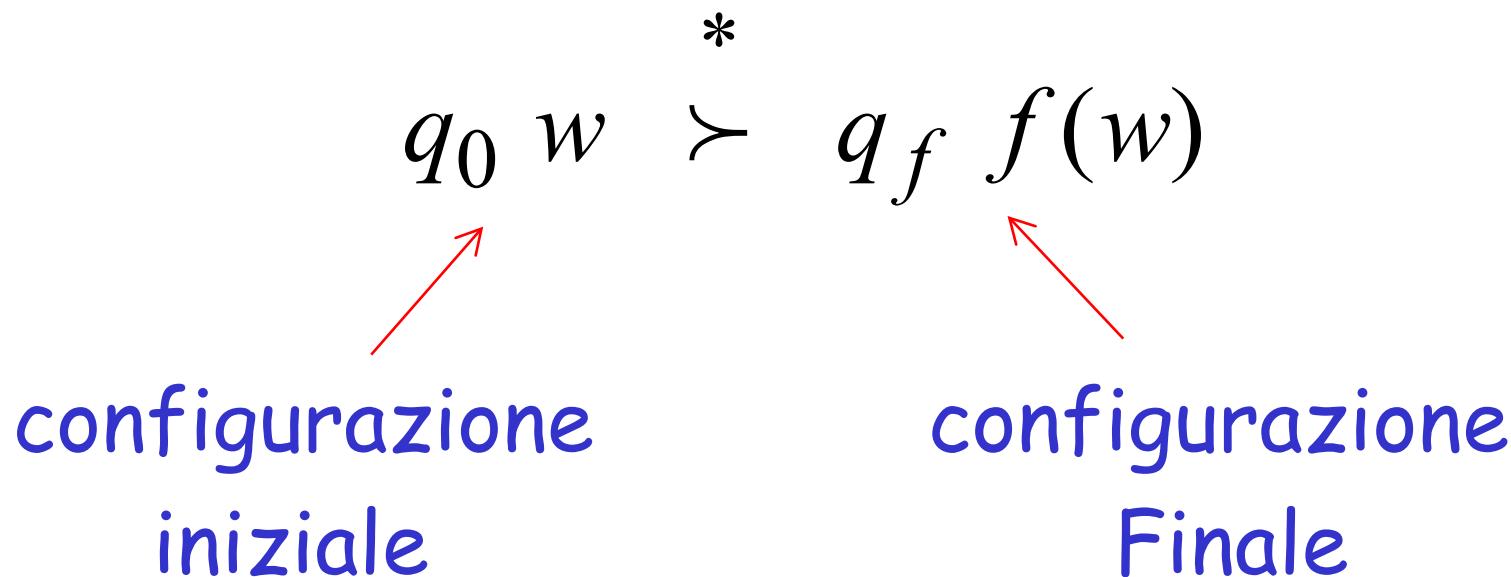
Stato di accettazione

Per tutti $w \in D$ dominio

Calcolare in modo standard

Inltre parole :

A Una funzione f è calcabile se
Vi è una macchina di Turing M tale che:



Per tutte $w \in D$ dominio

esempio

la funzione $f(x, y) = x + y$ è calcolabile

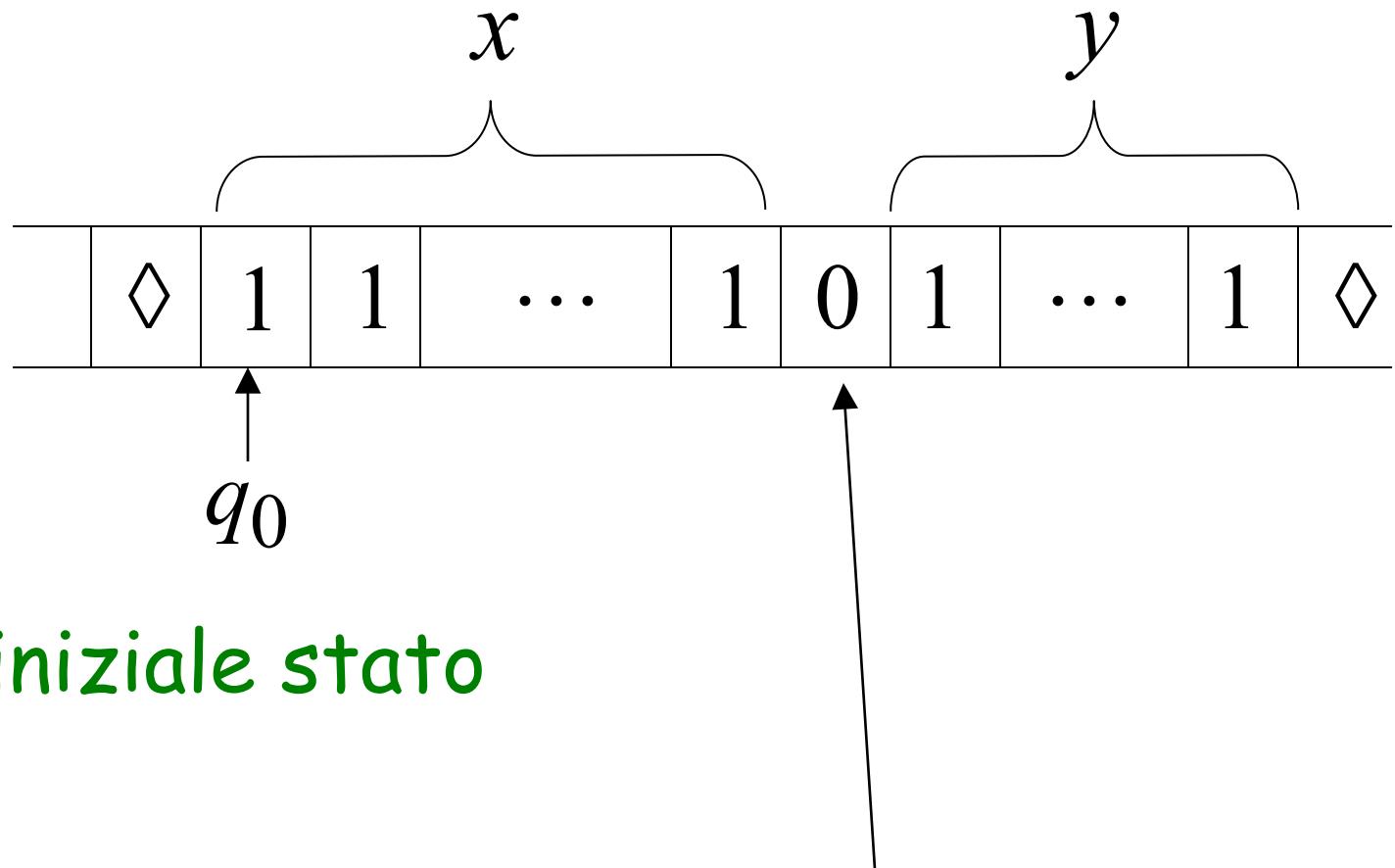
x, y Sono interi

macchina Turing :

stringa di input: $x0y$ unario

Output stringa: $xy0$ unario

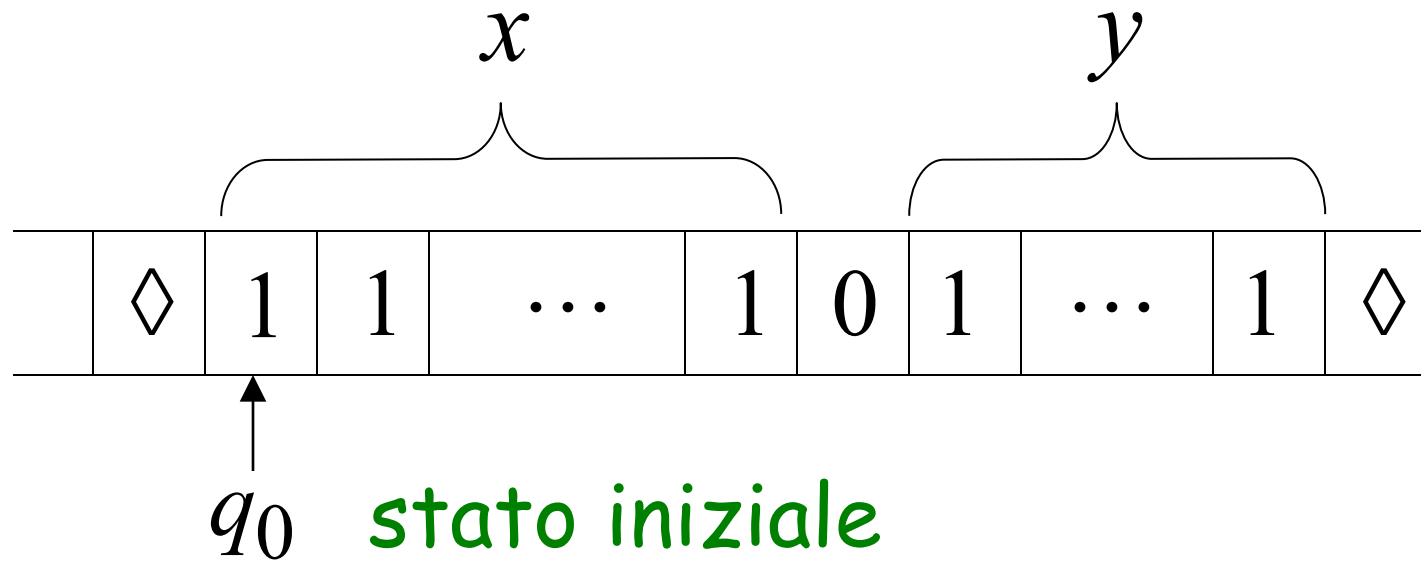
Start



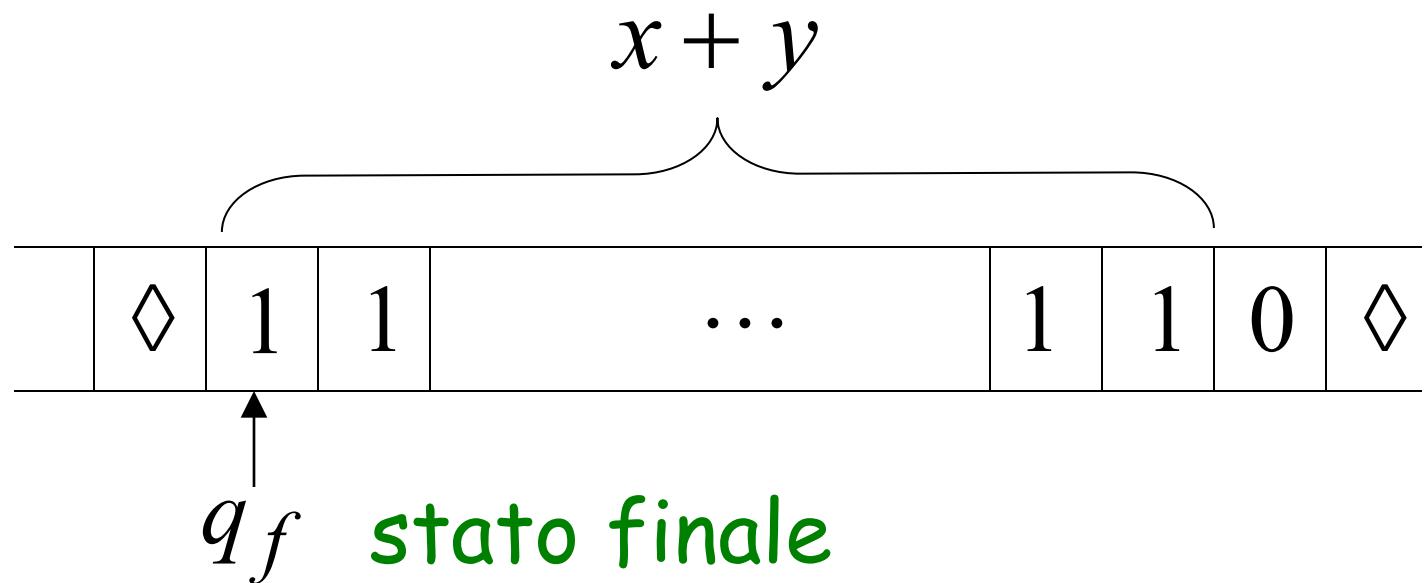
iniziale stato

il 0 è il delimitatore che
Separa I due numeri

Start

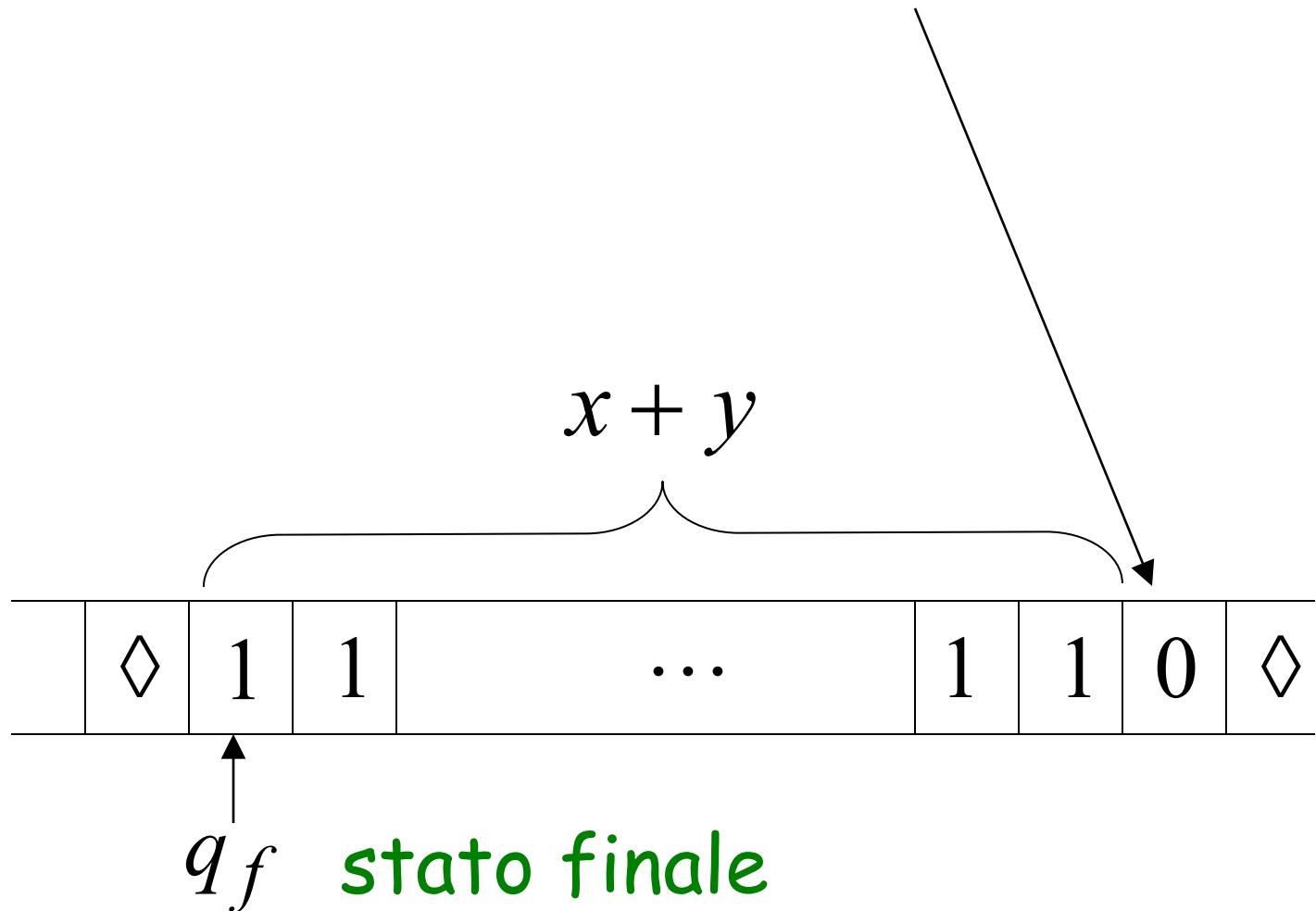


Fine



Io 0 ci può aiutare se usiamo
il risultato per un'altra operazione

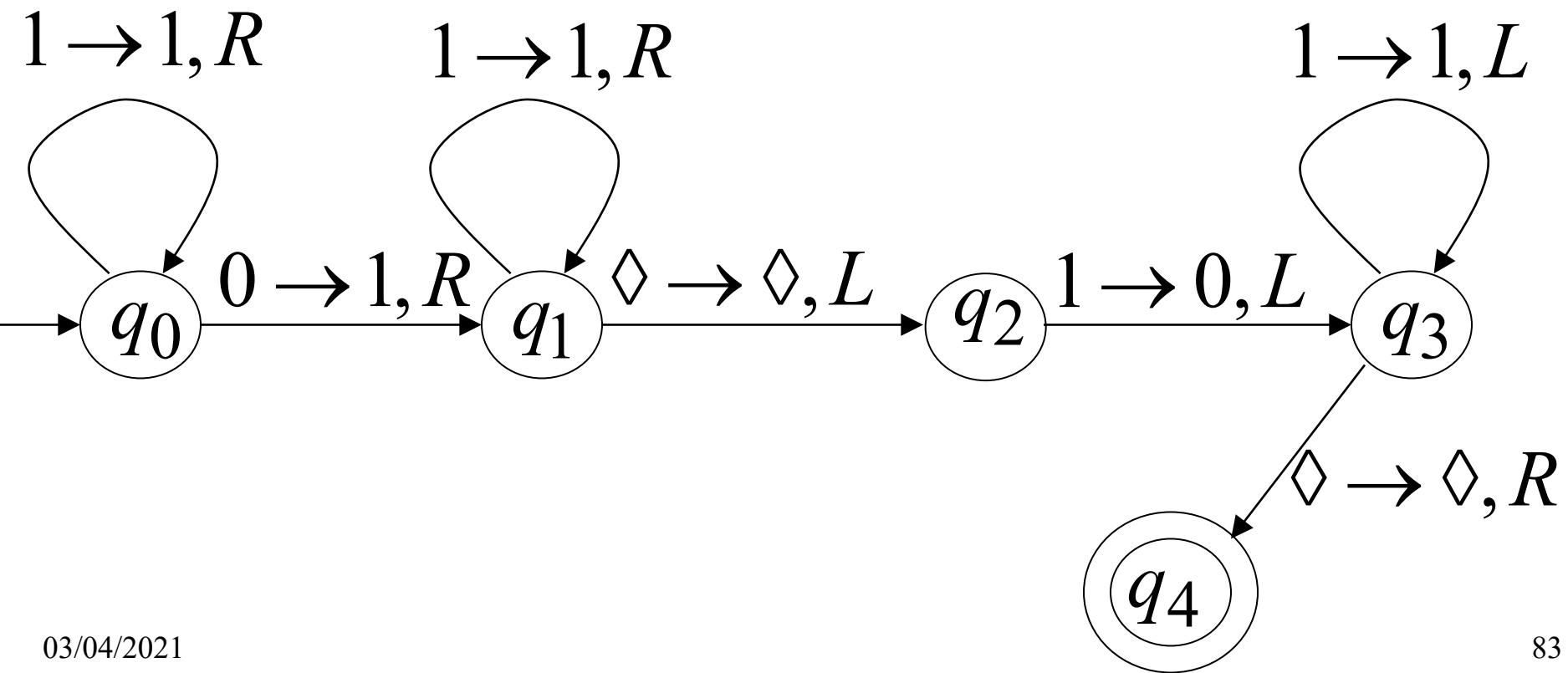
Fine



macchina Turing per la funzione

$$f(x, y) = x + y$$

Ricordarsi di
eliminare due 1
alla fine



Consideriamo i numeri naturali
senza lo zero

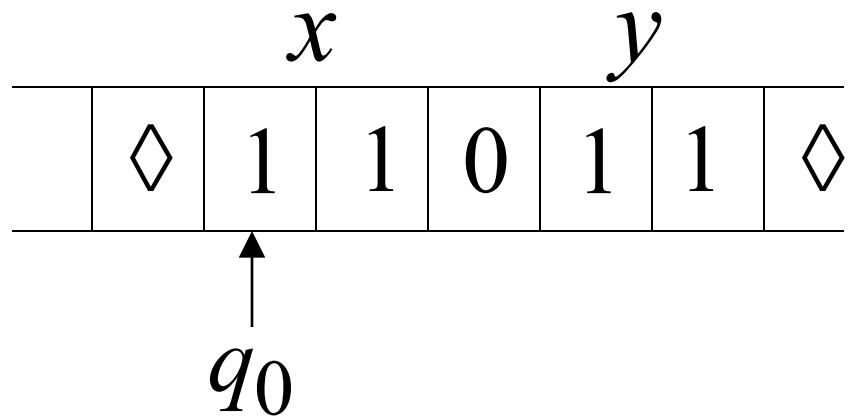
Quindi basta avere $n=1$ alla n

esempio di esecuzione:

Time 0

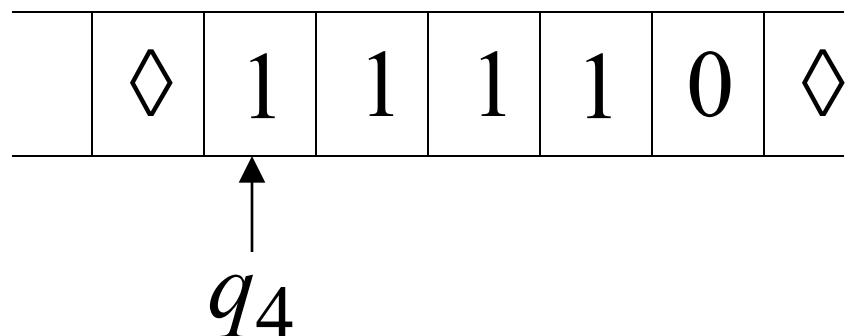
$$x = 11 \quad (=2)$$

$$y = 11 \quad (=2)$$

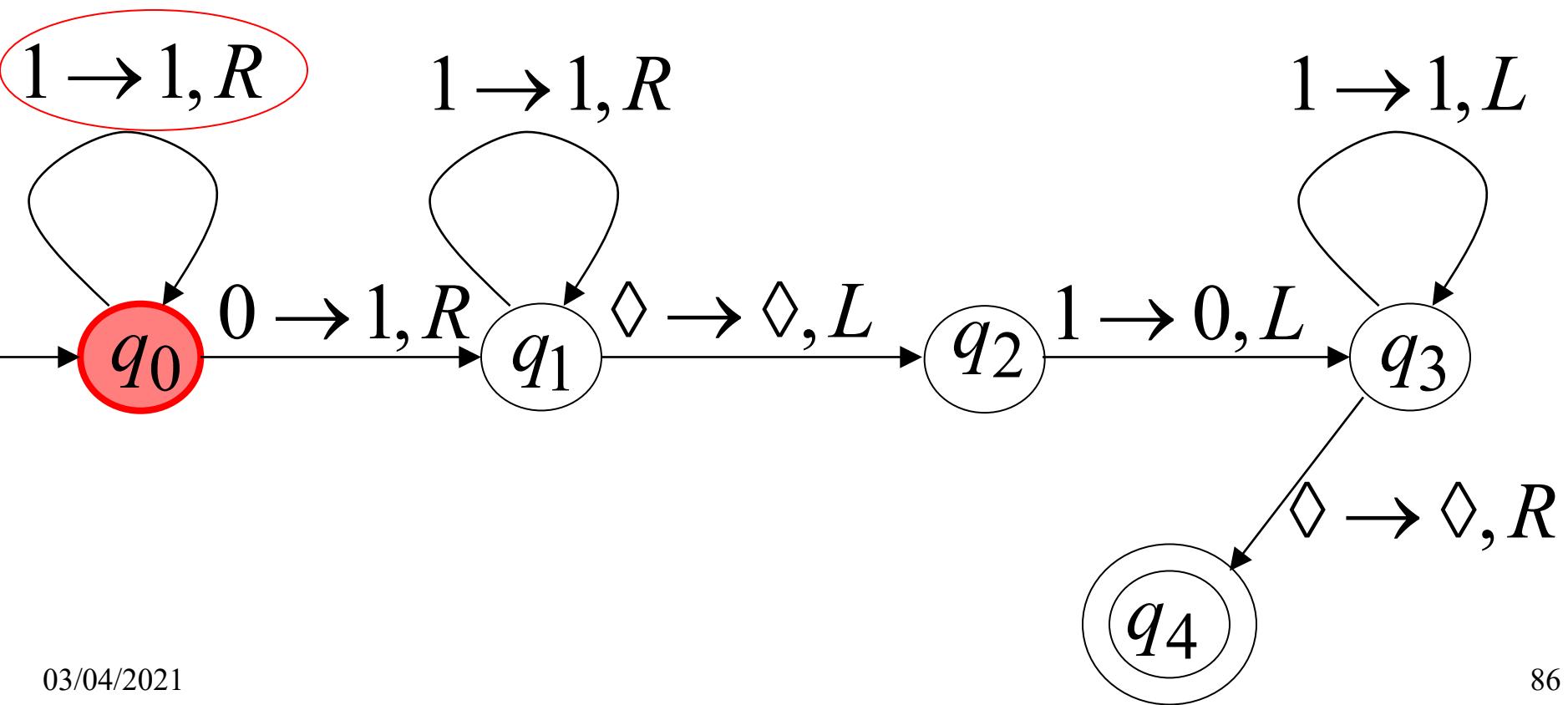
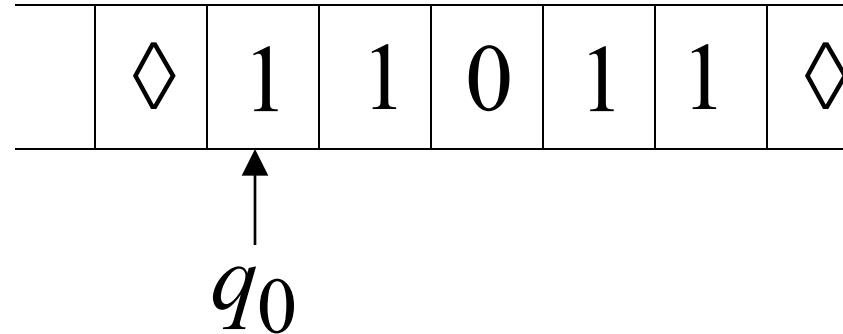


Final Result

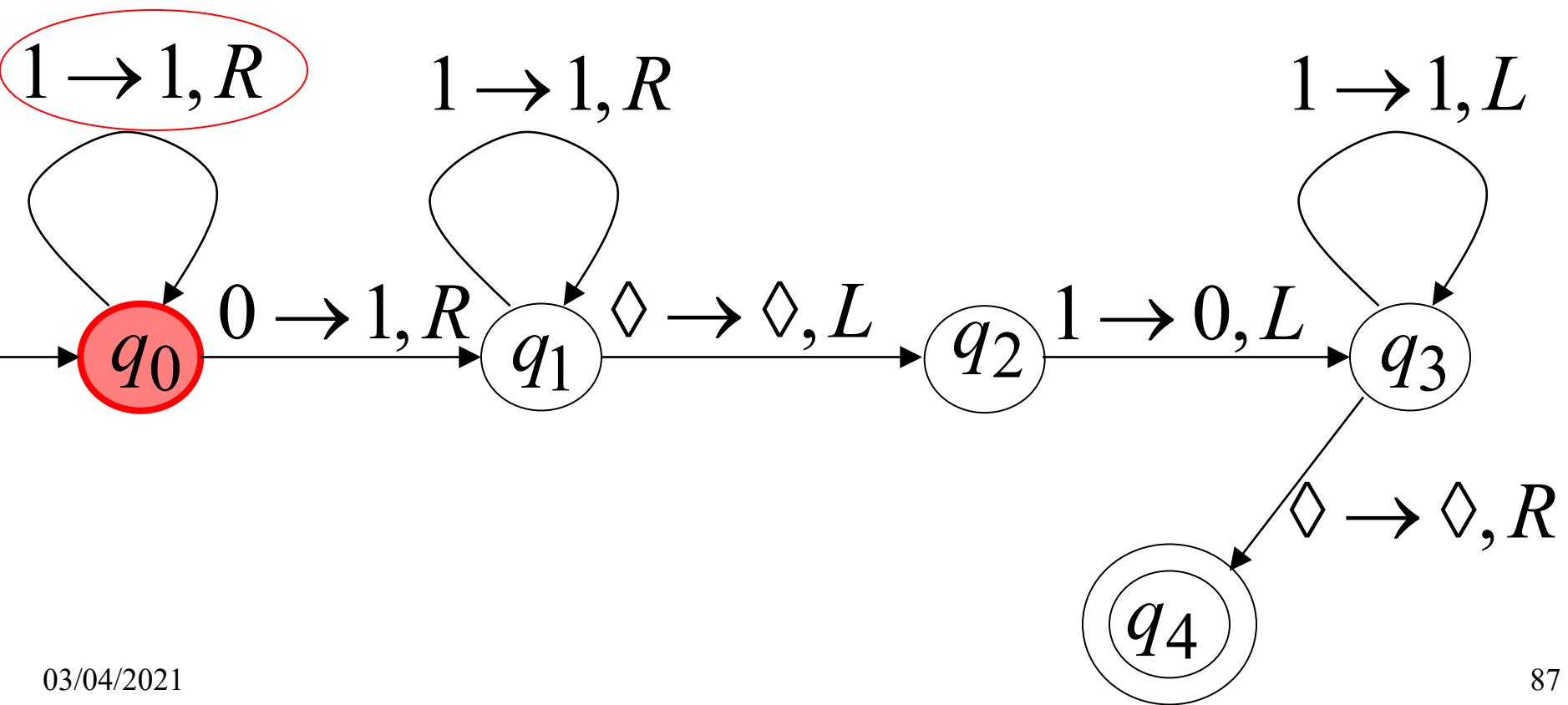
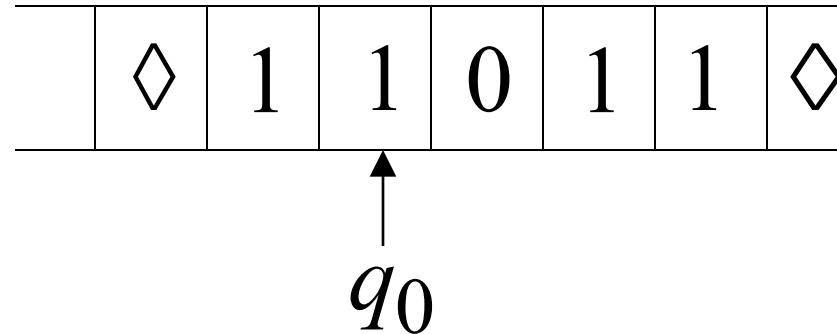
$$x + y$$



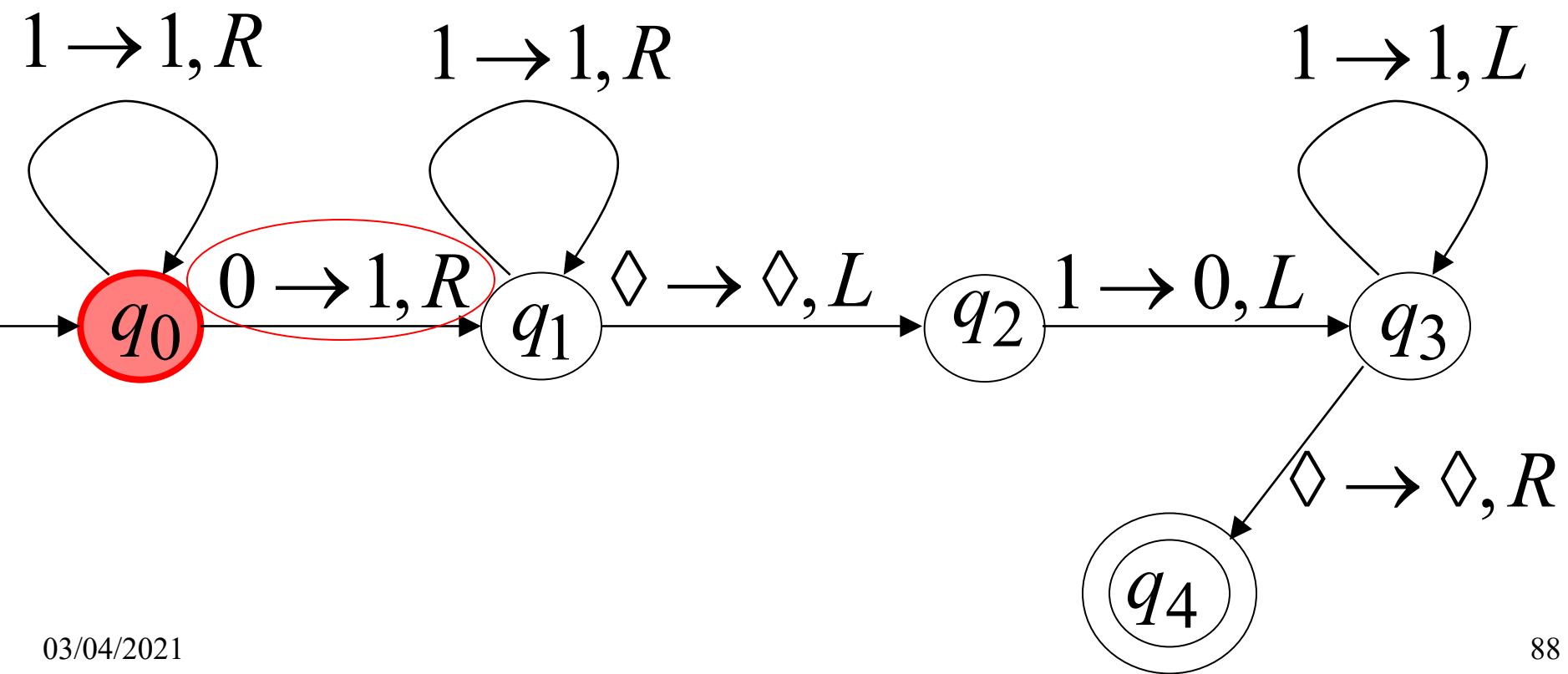
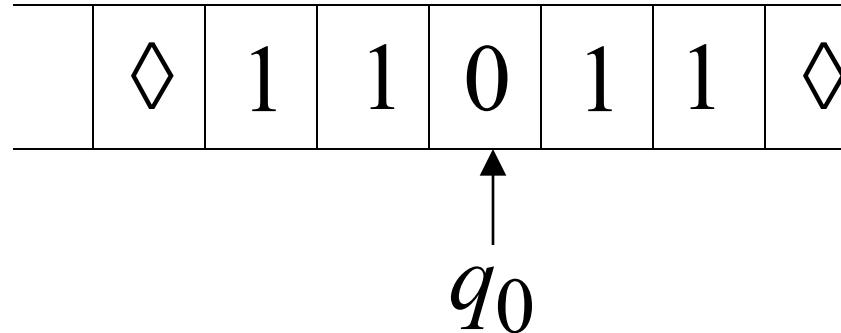
Time 0



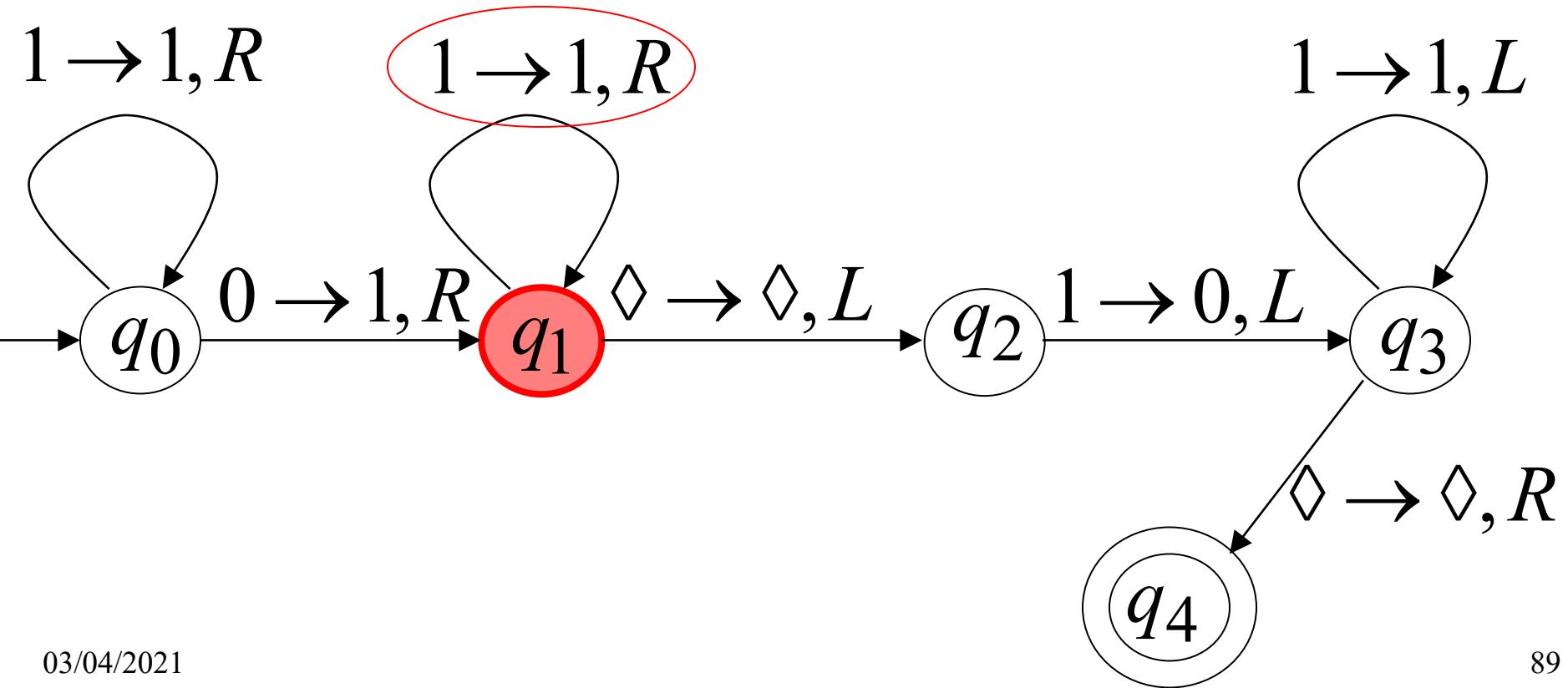
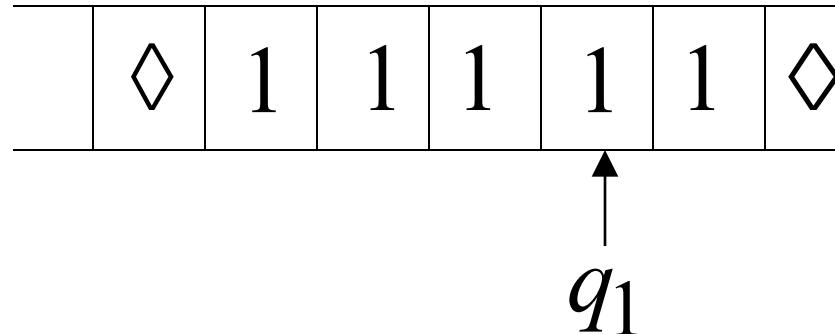
Time 1



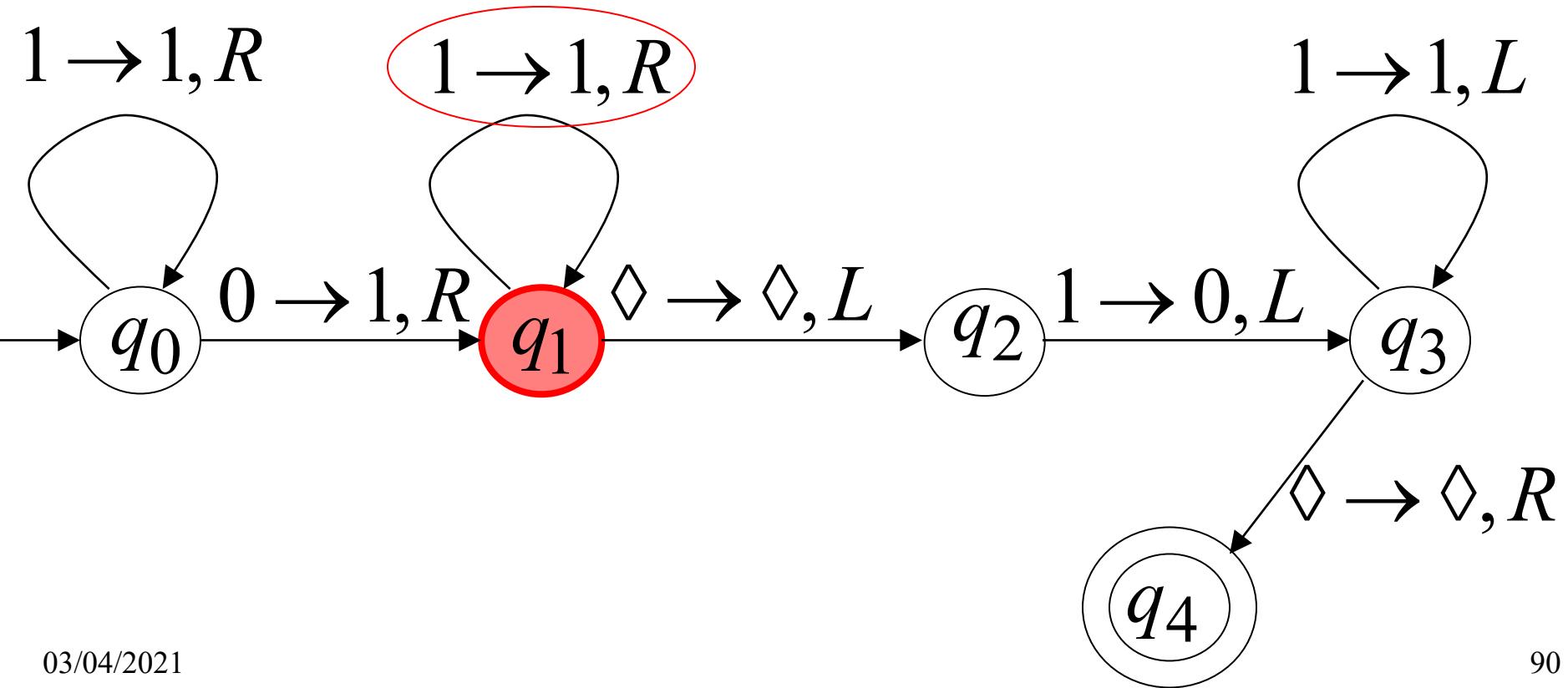
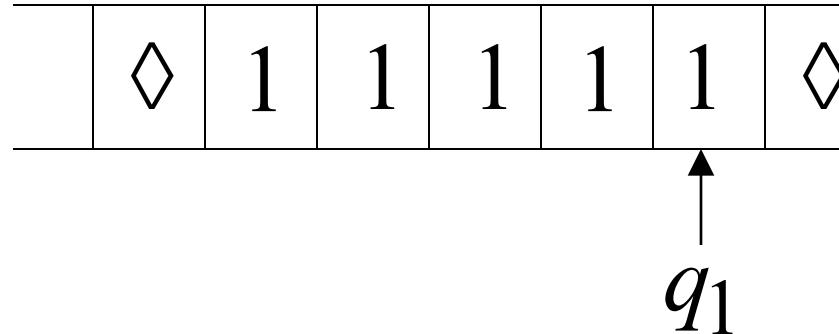
Time 2



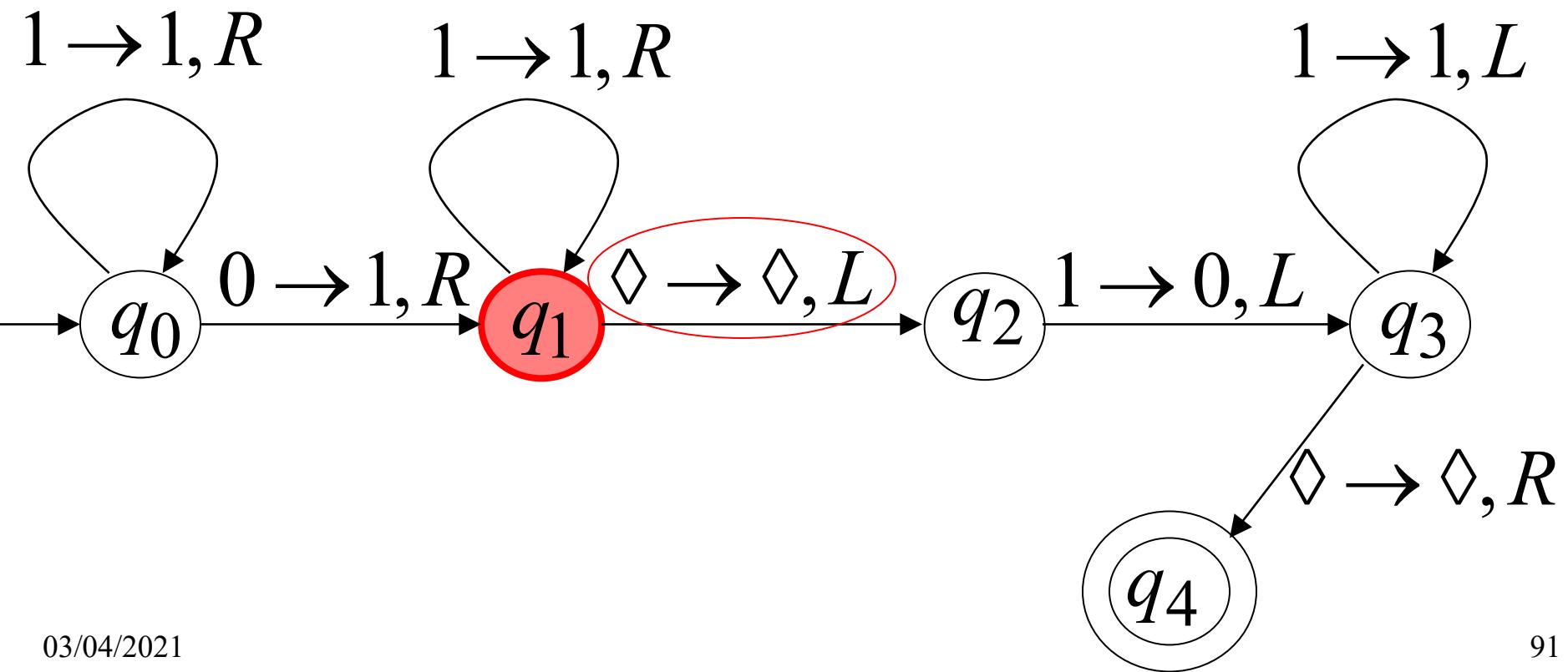
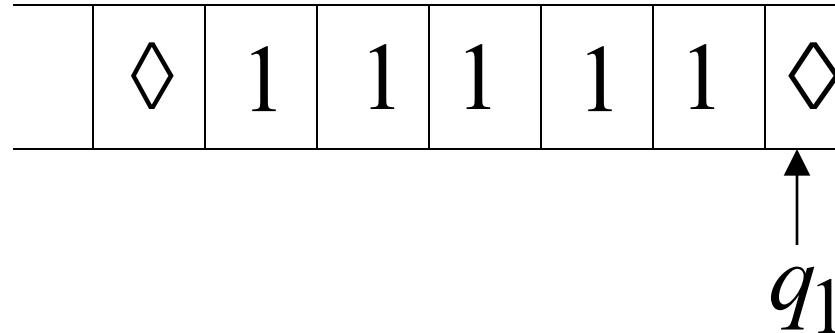
Time 3



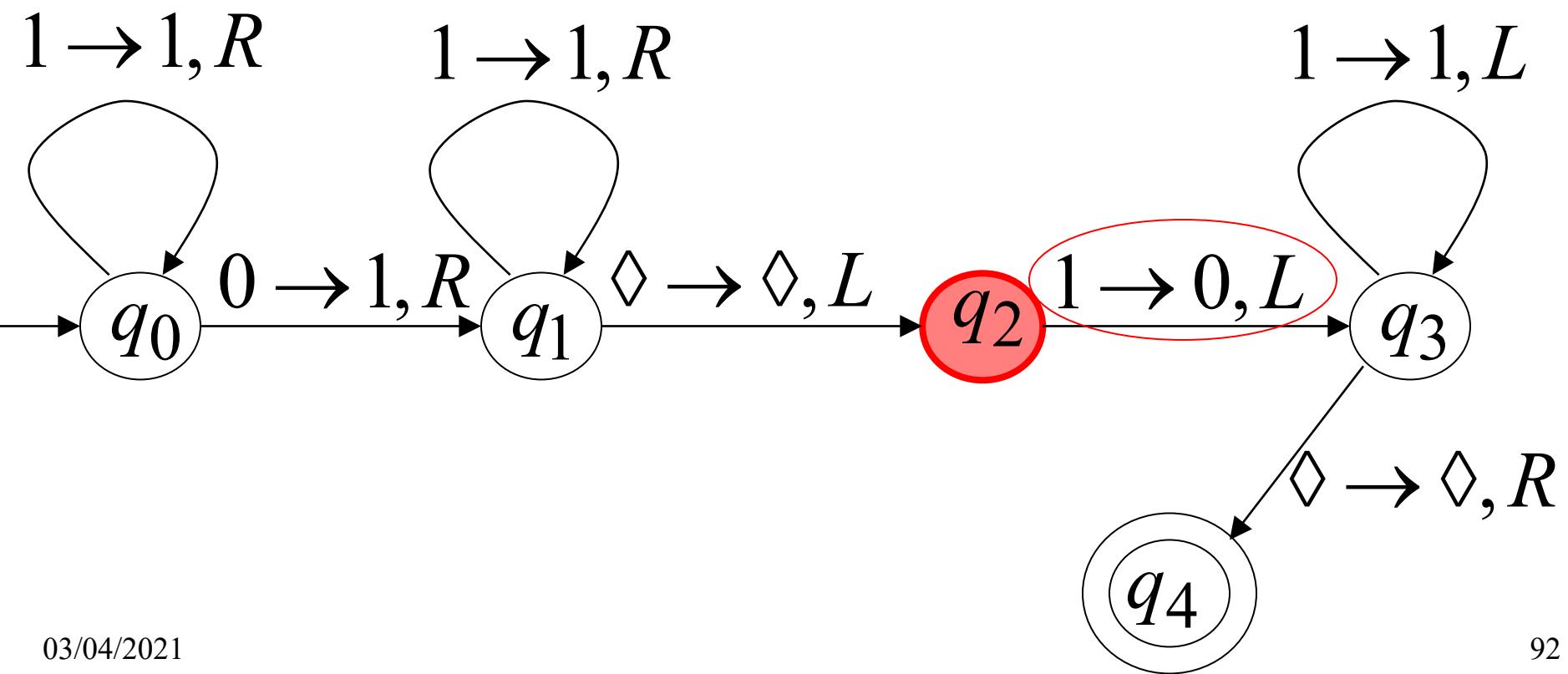
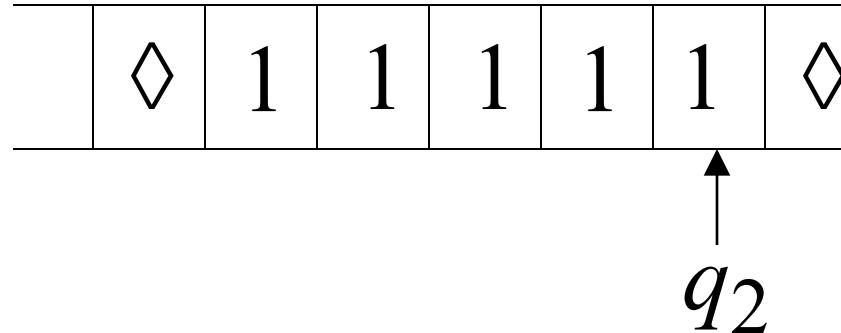
Time 4



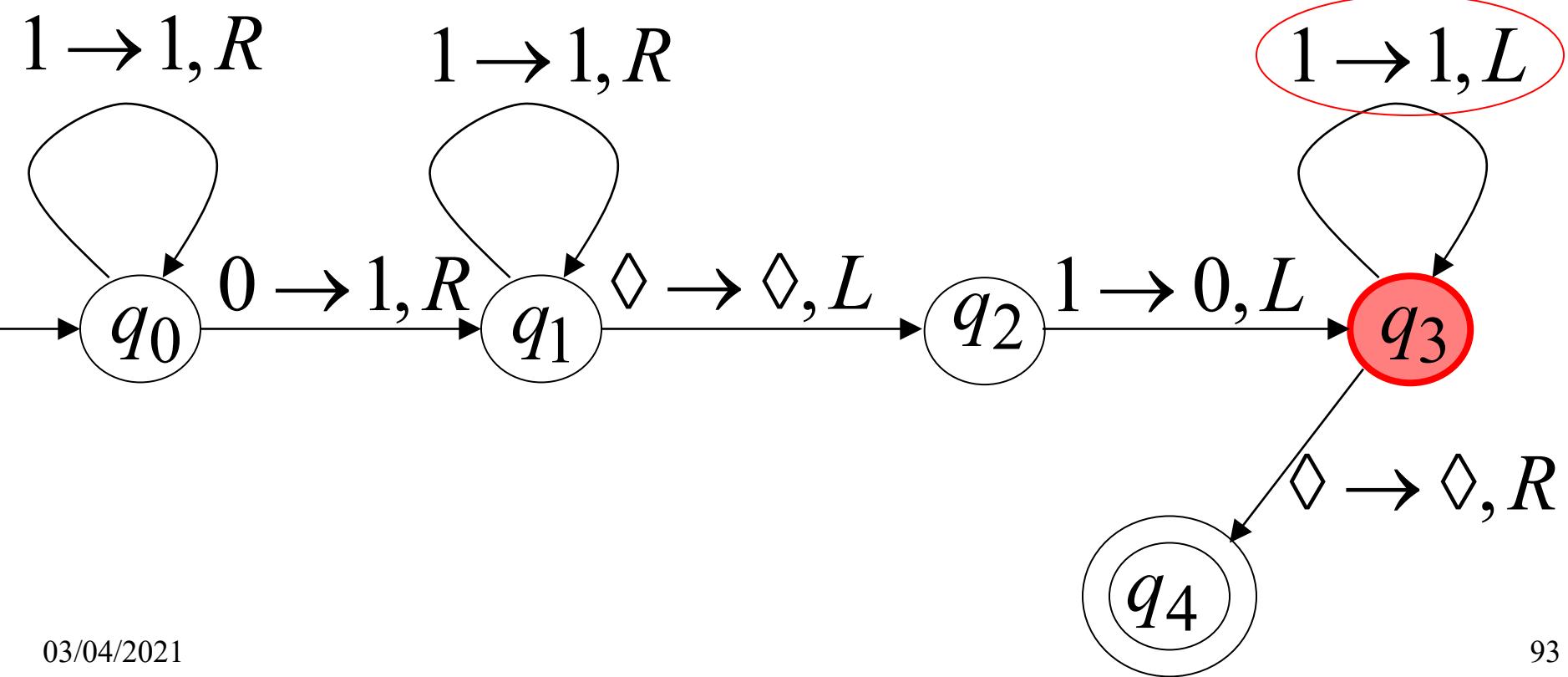
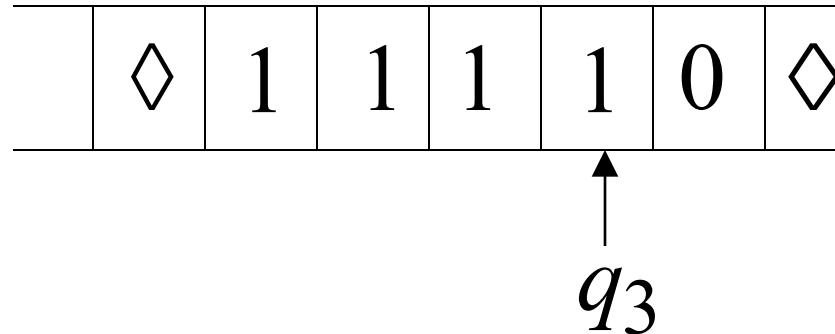
Time 5



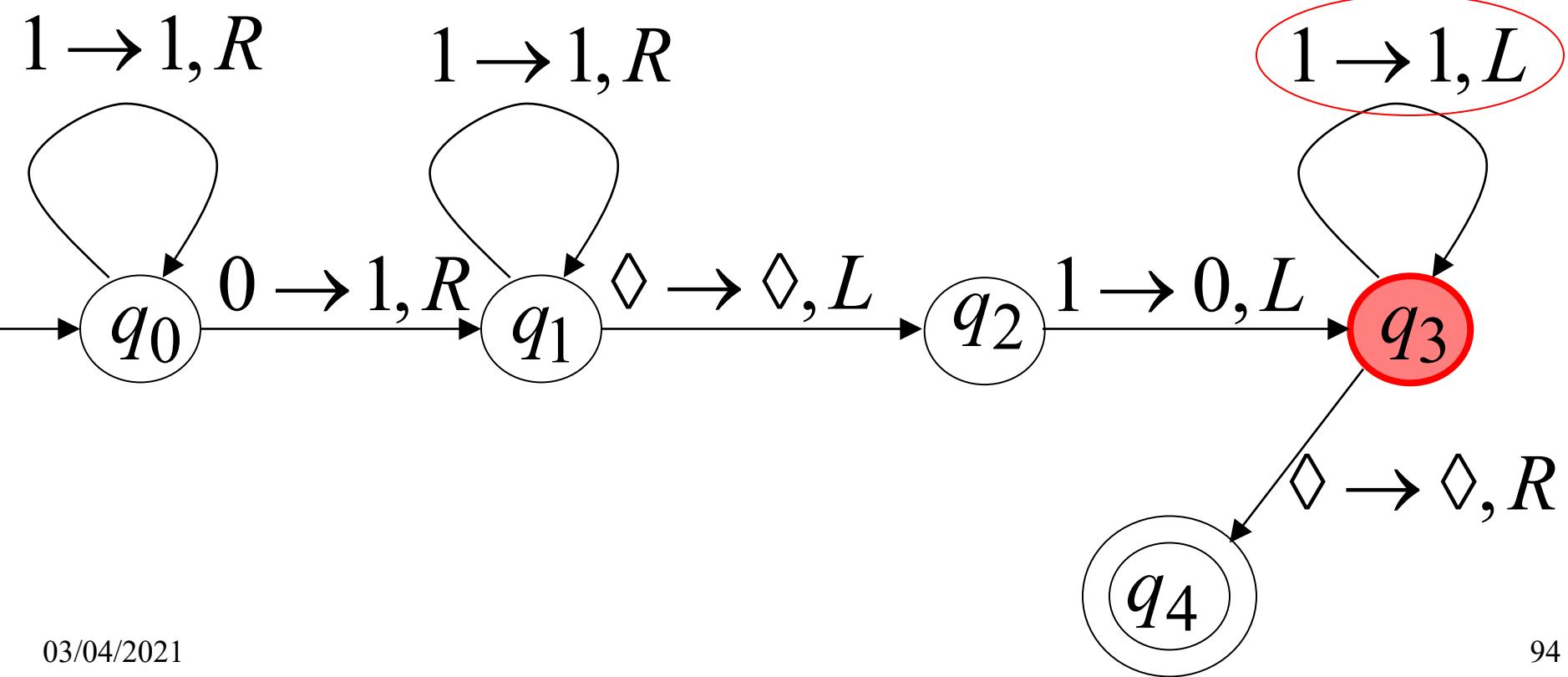
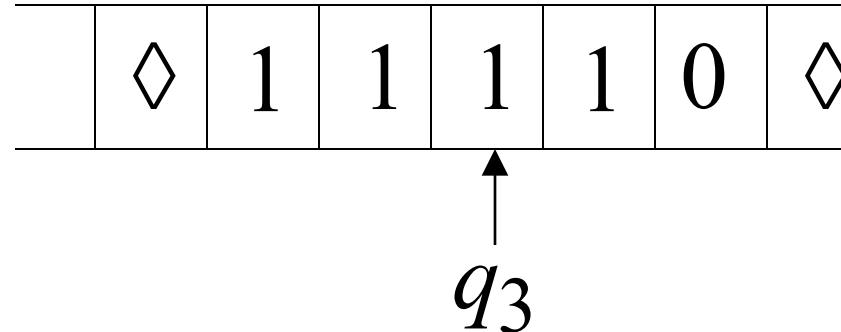
Time 6



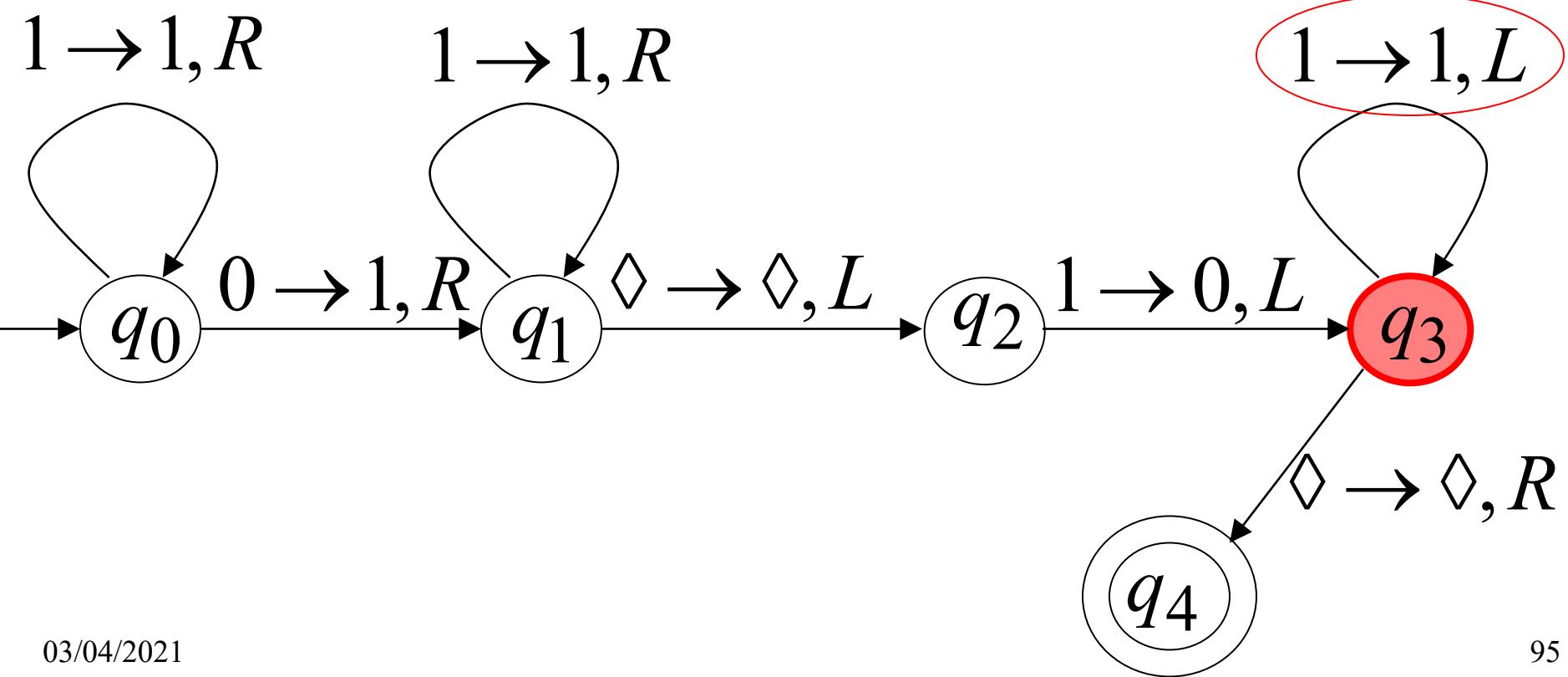
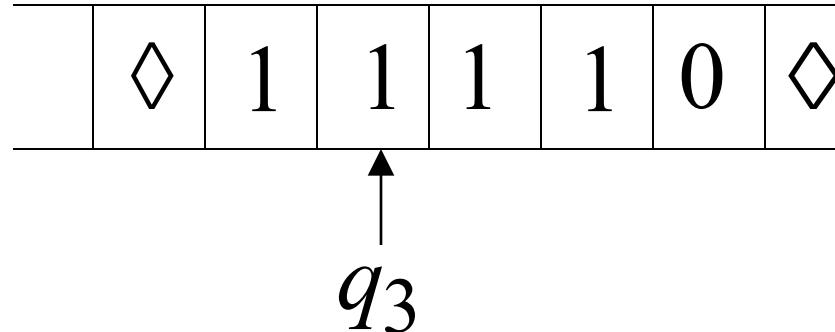
Time 7



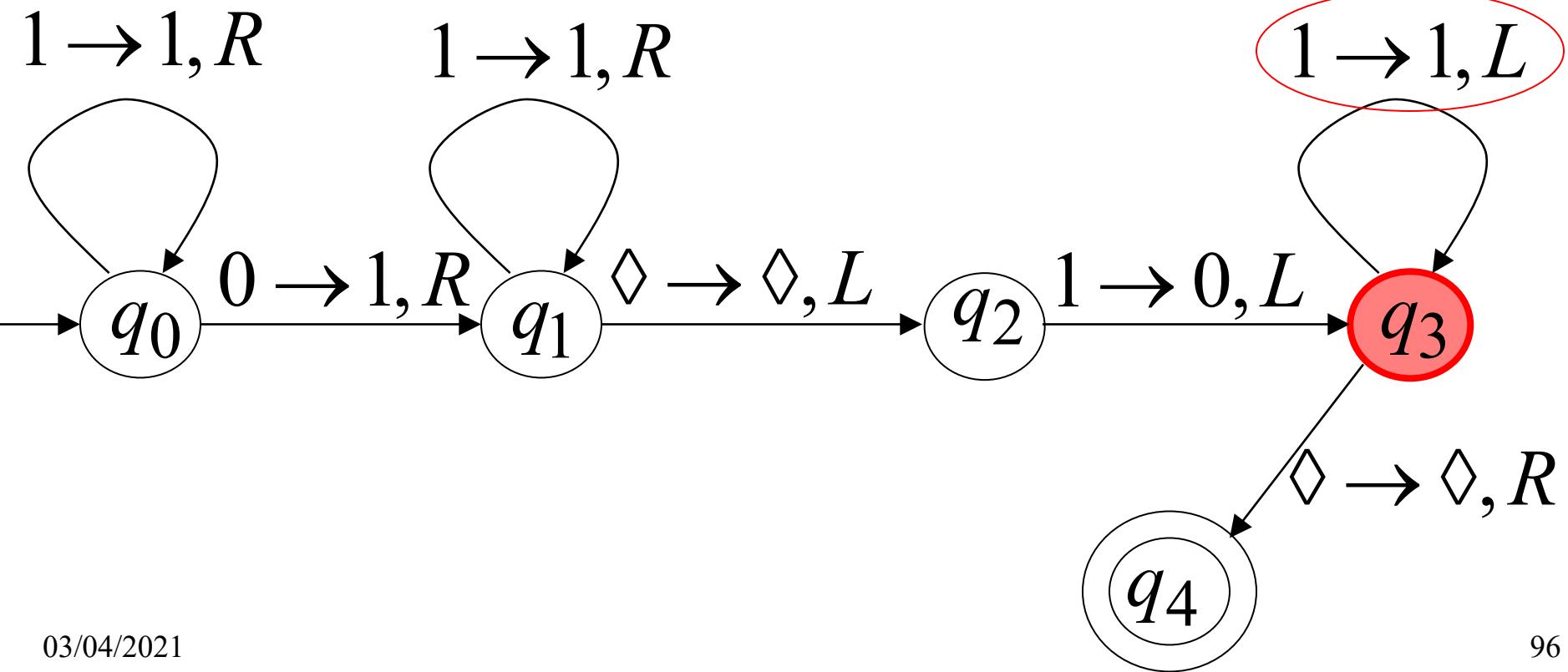
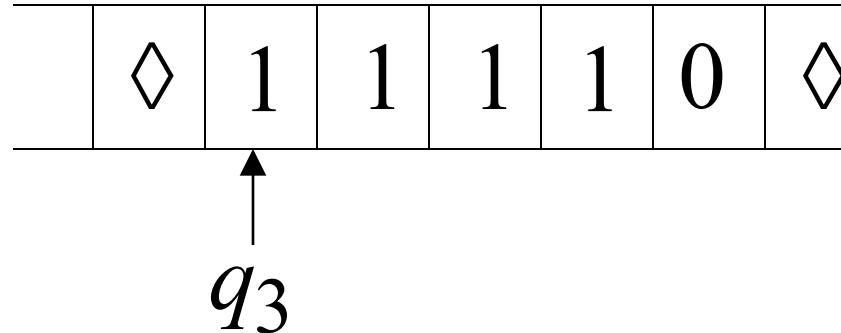
Time 8



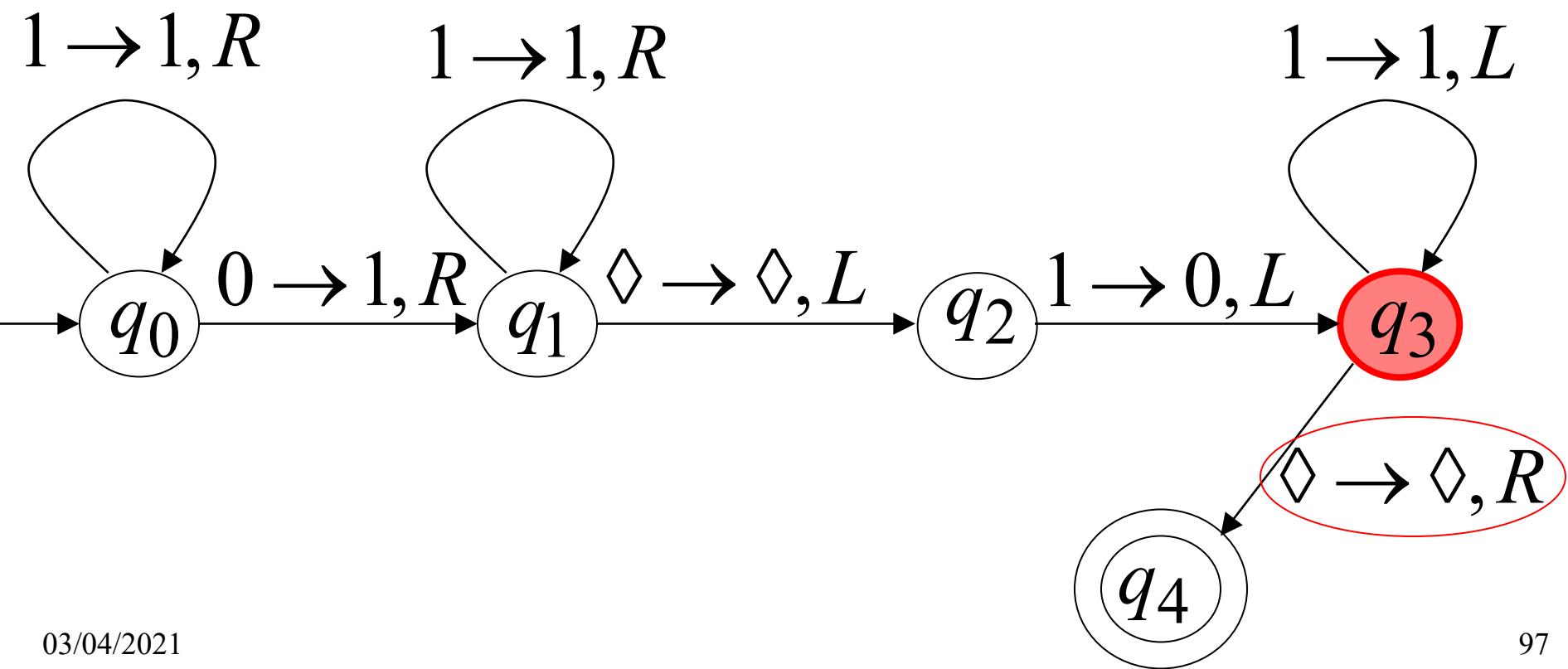
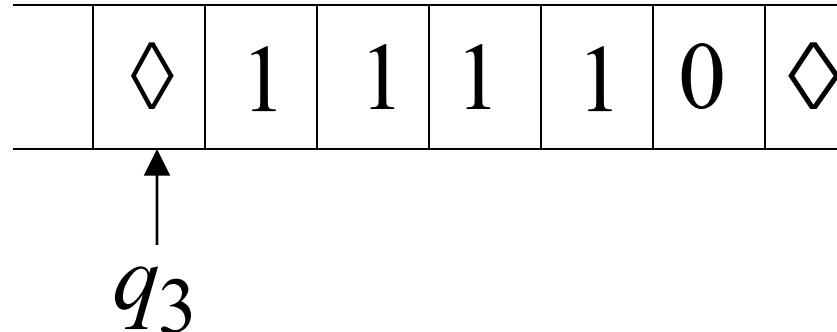
Time 9



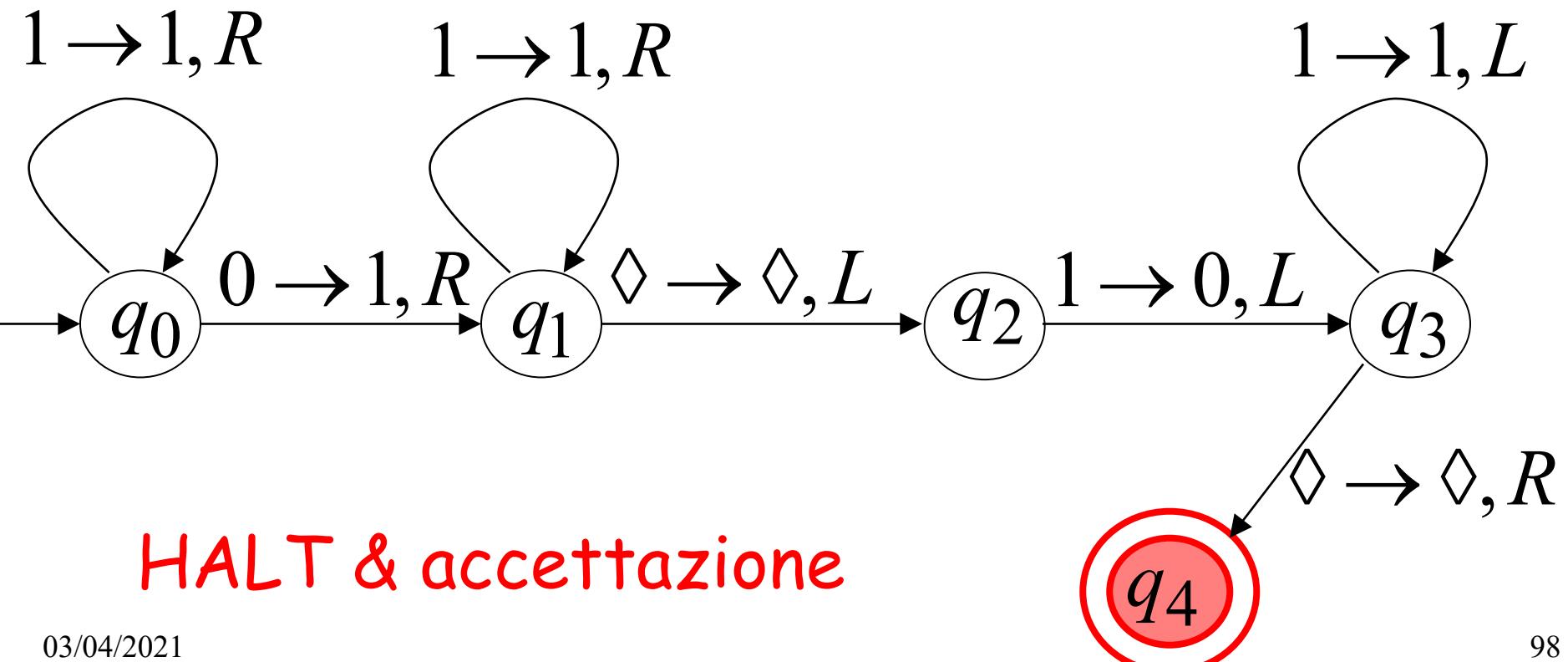
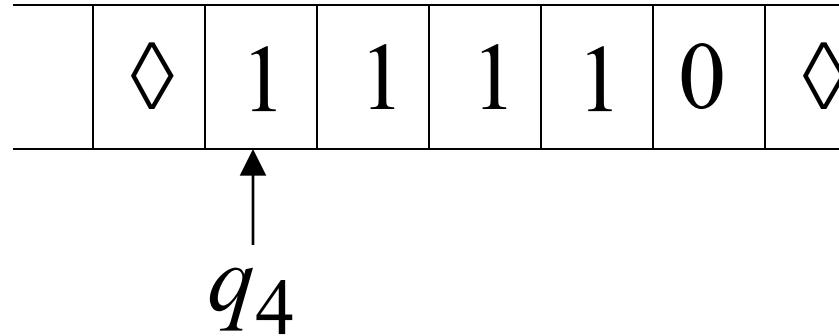
Time 10



Time 11



Time 12



Un altro esempio

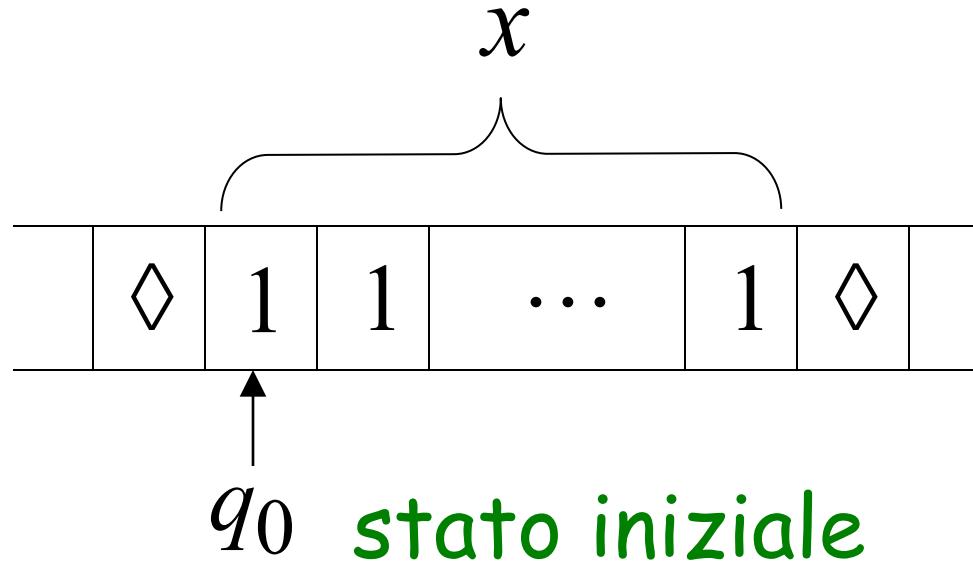
La funzione Che
raddoppia è calcolabile
il numero
di 1

Macchina di Turing :

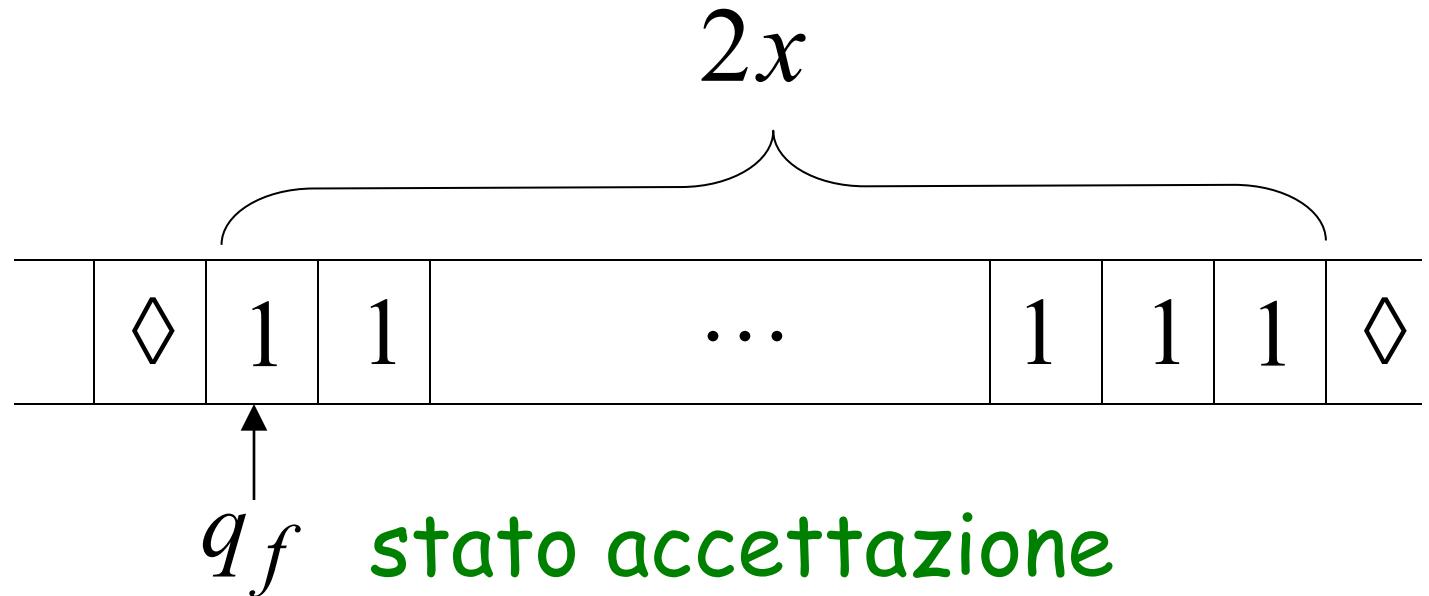
stringa di input: x unario

Output string: xx unario

Start



Finish



macchina Turing

Pseudocodice per

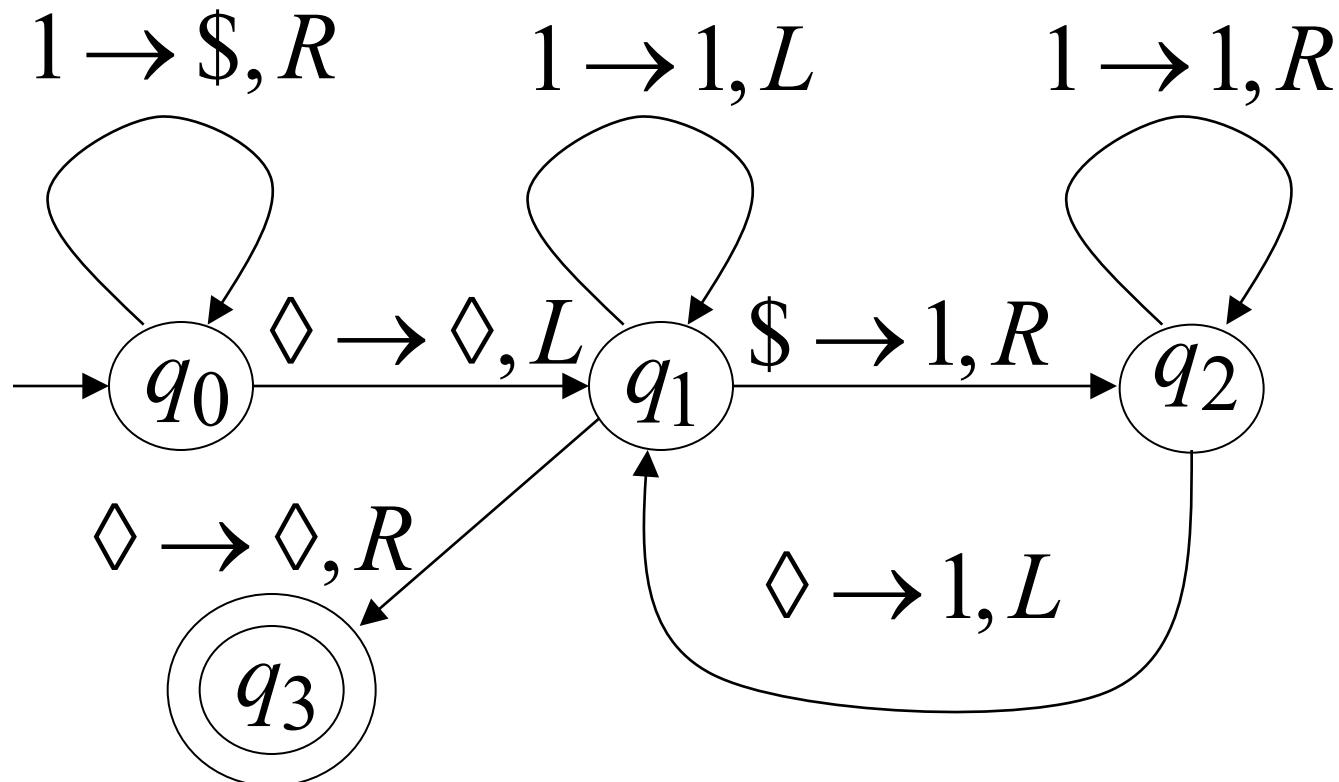
$$f(x) = 2x$$

- ogni 1 diventa \$
- Repeat:
 - trova il \$ più a destra, cambia in 1
 - vai alla fine a destra, inserisci 1

Until no \$ rimangono

Turing macchina per

$$f(x) = xx$$



esempio

Start

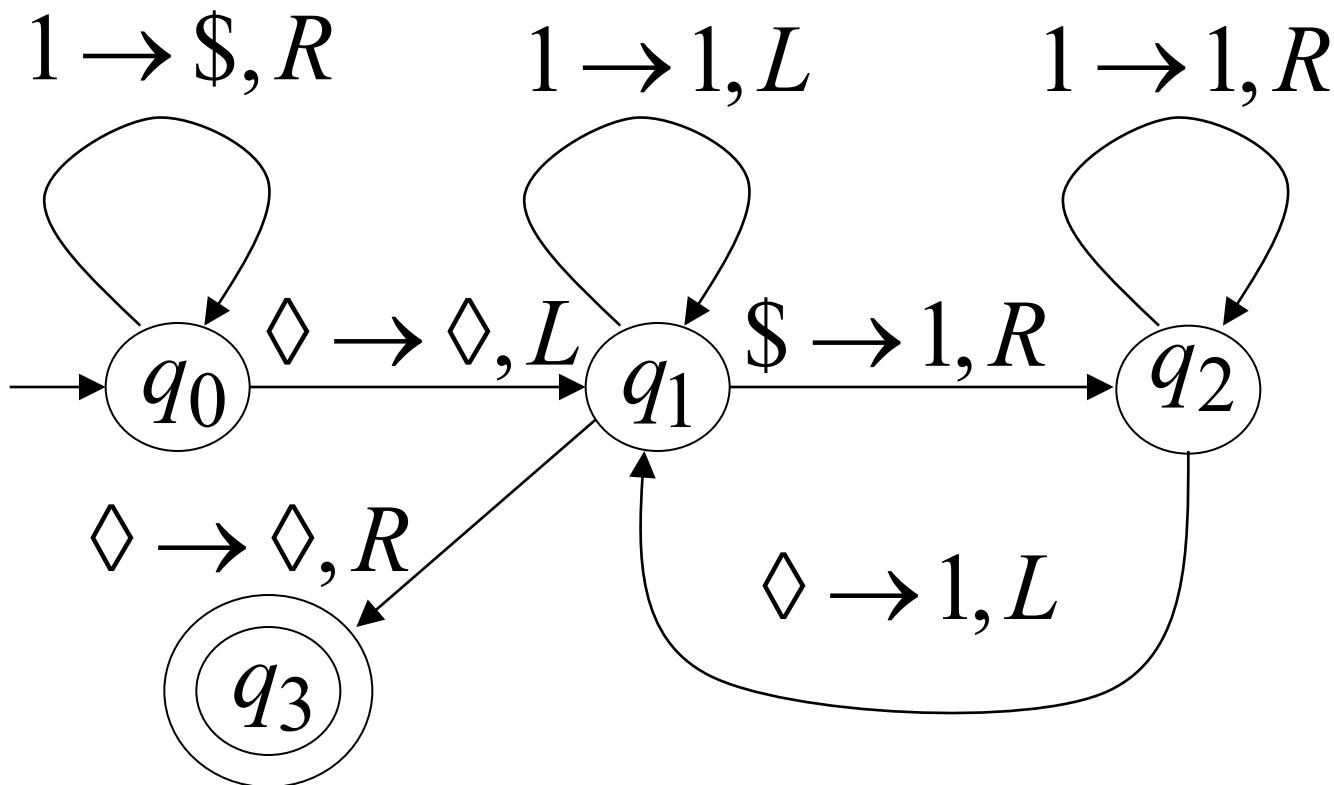
Finish

\diamond	1	1	\diamond	
------------	---	---	------------	--

q_0

	\diamond	1	1	1	1	\diamond
--	------------	---	---	---	---	------------

q_3



Copia a distanza di una stringa

Es ►1111 dà 111101111

altro esempio

La funzione
È calcolabile

$$f(x, y) = \begin{cases} 1 & \text{if } x > y \\ 0 & \text{if } x \leq y \end{cases}$$

Input: $x0y$

Output: 1 or 0

macchina di Turing Pseudocodice:

- Repeat

verifica ogni 1 da x con 1 fda y

Until tutti gli 1 di x or y sono verificate

- If un 1 da x non è verificato

cancella tape, scrivi 1 ($x > y$)

else

cancella tape, scrivi 0 ($x \leq y$)

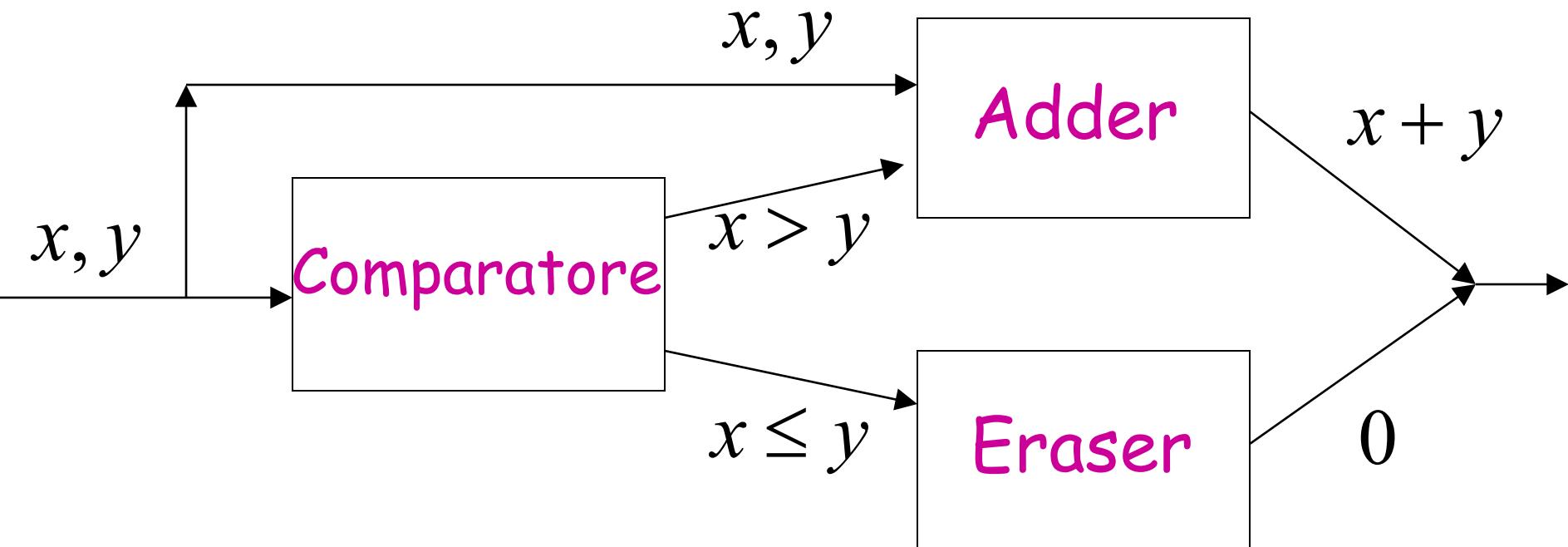
Mettere insieme macchine di
turing

Block Diagram



esempio:

$$f(x, y) = \begin{cases} x + y & \text{if } x > y \\ 0 & \text{if } x \leq y \end{cases}$$



Ricordiamoci sempre
Calcolo standard salvando gli input.

Vari modelli

Lezioni tratte dal gambosi etc, vai su ada.

Macchine di Turing Multitraccia

Definizione Una macchina di Turing multitraccia \mathcal{M} ad m tracce è definita come una 6-upla

$$\Sigma, \emptyset, K, \delta, q_0, q_f$$

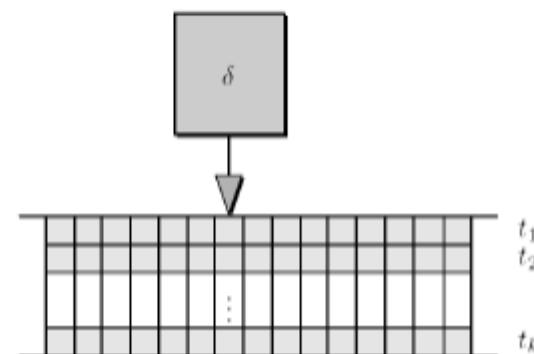
ove la funzione δ è definita come

$$\delta_m : (K - \{q_f\}) \times \Sigma_{\emptyset}^m \rightarrow K \times \Sigma_{\emptyset}^m \times \{d, s, i\}$$

Quindi, una macchina di Turing multitraccia è in grado di scrivere e leggere caratteri **vettoriali** ma la testina si sposta contemporaneamente su tutte le tracce.

Da un punto di vista fisico si può immaginare di avere una macchina composta da un nastro suddiviso in m tracce ed una singola testina.

L'uso di MT-multitraccia permette di avere maggior potere computazionale?



Macchine di Turing Multi-traccia

Una macchina di Turing **multi-traccia** consiste di un nastro suddiviso in **tracce** disposte in modo tale che la testina, con una singola operazione può accedere a tutte le celle di tutte le tracce in corrispondenza della testina.

Possiamo considerare la macchina multi-traccia come una macchina che anziché operare su **simboli scalari** opera su **simboli vettoriali**.

Data una macchina di turing $\mathcal{M}_m = (\Sigma, \emptyset, K, \delta^m, q_0, q_f)$ multitaccia con m tracce si ha che:

- l'alfabeto $\Sigma = \Sigma_1 \times \dots \times \Sigma_m$, ove ogni Σ_i rappresenta l'alfabeto di simboli della traccia i .

Quindi un generico elemento (carattere) $\sigma \in \Sigma$ sarà del tipo: $\sigma = (\sigma_1, \dots, \sigma_m)$, $1 \leq i \leq m$ $\sigma_i \in \Sigma_i$.

In particolare il simbolo $(\underbrace{\emptyset, \dots, \emptyset}_m)$ rappresenta il simbolo di blank dell'alfabeto Σ_\emptyset .

Poichè ogni elemento di Σ_i può essere combinato con gli altri elementi di Σ_j , per ogni $j \neq i$, il numero di caratteri $\sigma = (\sigma_1, \dots, \sigma_m)$ diversi che potranno comparire sul nastro saranno: $|\Sigma| = \prod_{i=1}^m |\Sigma_i|$.

- la funzione di transizione δ^m sarà una funzione del tipo:

$$\delta^m : (K - \{q_f\}) \times \Sigma_\emptyset \rightarrow K \times \Sigma_\emptyset \times \{d, s, i\}, \quad \delta^m(q_i, \sigma) = (q_j, \sigma'), \quad \sigma, \sigma' \in \Sigma_\emptyset$$

Equivalenza MT-multi-traccia e MT-singola-traccia

Teorema Una Macchina di Turing singolo nastro multi-traccia \mathcal{M}^m con m tracce può essere simulata da una macchina di Turing singolo nastro mono-traccia \mathcal{M} .

Dimostrazione Sia $\mathcal{M}^m = (\Sigma, \emptyset, K, \delta^m, q_0, q_f)$ la macchina di Turing multitraccia, ove $\Sigma = \Sigma_1 \times \dots \times \Sigma_m$. Si definisca una MT singola traccia $\mathcal{M} = (\Lambda, \emptyset, K', \delta, q'_0, q'_f)$ tale che:

$$|\Lambda| = |\Sigma_1| \times \dots \times |\Sigma_m|$$

cioè, la cardinalità dell'alfabeto Λ è pari al prodotto delle cardinalità degli alfabeti delle singole tracce.

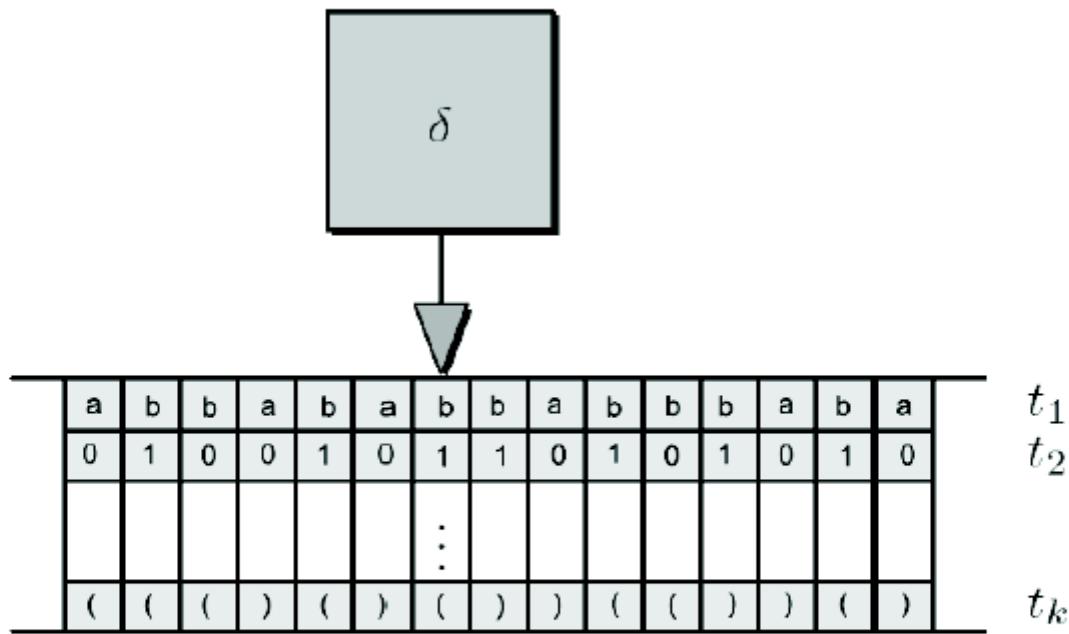
Si definisca inoltre, una funzione **iniettiva** $\varphi : \Sigma \rightarrow \Lambda$ che associa ad ogni simbolo di Σ un simbolo di Λ (poichè $|\Lambda| \geq |\Sigma|$ è possibile definire una tale funzione).

La funzione di transizione δ sarà definita in modo tale che in corrispondenza di una transizione $\delta^m(q_i, \sigma) = (q_j, \sigma', v)$ della macchina \mathcal{M}^m , la macchina \mathcal{M} esegua la transizione:

$$\delta(q_i, \lambda) = (q_j, \lambda', v), \quad \lambda = \varphi(\sigma), \quad \lambda' = \varphi(\sigma')$$

L'alfabeto su cui opera la macchina singola traccia \mathcal{M} è un alfabeto in cui ogni simbolo rappresenta la **codifica** di un vettore di simboli dell'alfabeto della macchina multi-traccia \mathcal{M}^m .

Equivalenza MT-multi-traccia e MT-singola-traccia



La testina della macchina di Turing multi-traccia punta al carattere $\sigma = \begin{bmatrix} b \\ 1 \\ \vdots \\ (\end{bmatrix}$

Macchine di Turing Multinastro

Definizione Una macchina di Turing ad m nastri è definita da una 6-upla

$$\Sigma, \emptyset, K, \delta_m, q_0, q_f$$

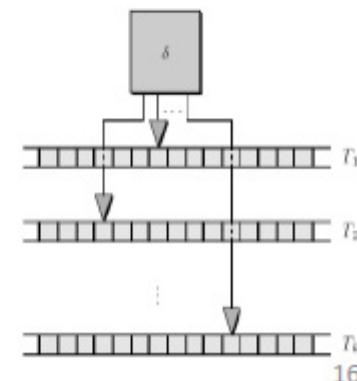
ove $\Sigma, \emptyset, K, q_0, q_f$ sono definiti come nel caso di una macchina di Turing a singolo nastro e δ_m è la funzione di transizione definita come:

$$\delta_m : (K - \{q_f\}) \times \Sigma_{\emptyset}^m \rightarrow K \times \Sigma_{\emptyset}^m \times \{d, s, i\}^m$$

cioè la funzione δ_m definisce le transizioni della MT su ogni nastro.

Da un punto di vista fisico si può immaginare di avere una macchina composta da m nastri ed m testine, una per ogni nastro.

L'uso di MTM permette di avere un maggior potere computazionale ?



Equivalenza MT-Multinastro e MT-singolo-nastro

\dots	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\downarrow	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_1
\dots	\bar{b}	\bar{b}	\bar{b}	a	b	b	a	\bar{b}	\bar{b}	\bar{b}	\dots	t_2
\dots	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\downarrow	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_3
\dots	\bar{b}	\bar{b}	\bar{b}	b	a	b	c	b	\bar{b}	\dots	t_4	
\dots	\bar{b}	\bar{b}	\downarrow	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_5
\dots	\bar{b}	d	e	d	d	e	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_6
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	
\dots	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\downarrow	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_{2k-1}
\dots	f	f	g	h	g	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_{2k}

Figure 1: Macchina di Turing singolo nastro multitraccia che simula una Macchina di Turing multinastro

Equivalenza MT-Multinastro e MT-singolo-nastro

Teorema Sia data una macchina di Turing $\mathcal{M}^{(k)}$ con k nastri, allora esiste una macchina di Turing \mathcal{M} a singolo nastro che la simula.

Dimostrazione Sia $\mathcal{M}^{(k)}$ la MT multi nastro così definita $\mathcal{M}^{(k)} = (\Sigma, \emptyset, K, q_0, F, \delta)$ ove supponiamo per ogni nastro i , $1 \leq i \leq k$ che l'alfabeto usato sia Σ_i .

Costruiamo una MT singolo nastro, avente $2k$ tracce, definita nel seguente modo:

$$\mathcal{M}' = (\Sigma', \emptyset, K', q'_0, F', \delta')$$

ove l'alfabeto Σ' è definito come:

$$\Sigma' = \{\emptyset, \downarrow\} \times \Sigma_1 \times \dots \times \{\emptyset, \downarrow\} \times \Sigma_k$$

cioè, è composto di k coppie di simboli (λ_i, σ_i) di cui $\lambda_i \in \{\emptyset, \downarrow\}$ e $\sigma_i \in \Sigma_i$ per ogni $1 \leq i \leq k$.

Il nastro di \mathcal{M}' risulta allora composto nel seguente modo:

- ▷ $\forall i, 1 \leq i \leq k$, la traccia pari di indice $2i$ contiene la stringa presente sul nastro di indice i
- ▷ $\forall i, 1 \leq i \leq k$, la traccia dispari di indice $2i - 1$ contiene una stringa composta del solo simbolo \downarrow rappresentante la posizione della testina del nastro di indice i

Equivalenza MT-Multinastro e MT-singolo-nastro

\dots	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\downarrow	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_1
\dots	\bar{b}	\bar{b}	\bar{b}	a	b	b	a	\bar{b}	\bar{b}	\bar{b}	\dots	t_2
\dots	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\downarrow	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_3
\dots	\bar{b}	\bar{b}	\bar{b}	b	a	b	c	b	\bar{b}	\bar{b}	\dots	t_4
\dots	\bar{b}	\bar{b}	\downarrow	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_5
\dots	\bar{b}	d	e	d	d	e	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_6
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	
\dots	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\downarrow	\bar{b}	\bar{b}	\bar{b}	\dots	t_{2k-1}
\dots	f	f	g	h	g	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\bar{b}	\dots	t_{2k}

Figure 1: Macchina di Turing singolo nastro multitraccia che simula una Macchina di Turing multinastro

All'inizio della computazione supponiamo che il nastro di \mathcal{M}' sia configurato nel seguente modo:

- ▷ la traccia 1 contiene una stringa con il solo simbolo \downarrow in corrispondenza del primo carattere a sinistra della traccia 2
- ▷ la traccia 2 contiene la stringa di input della macchina $\mathcal{M}^{(k)}$
- ▷ $\forall i, 2 \leq i \leq k$ le tracce pari di indice $2i$ contengono solamente il simbolo β
- ▷ $\forall i, 2 \leq i \leq k$ le tracce dispari di indice $2i - 1$ contengono solamente il simbolo \downarrow in corrispondenza del simbolo più a sinistra della traccia 2, cioè quella contenente la stringa di input della macchina multi-traccia $\mathcal{M}^{(k)}$

Per simulare la funzione di transizione di $\delta^{(k)}$ della MT multi-nastro \mathcal{M}' , la funzione di transizione δ' deve riscrivere $2k$ simboli, uno per traccia. Quindi, in particolare, per simulare una transizione del tipo:

$$\delta^{(k)}(q_i, a_{i_1}, \dots, a_{i_k}) = (q_j, a_{j_1}, \dots, a_{j_k}, d_1, \dots, d_k)$$

deve eseguire i seguenti passi:

1. rintracciare le posizioni dei k simboli \downarrow rappresentanti le posizioni delle k testine della macchina $\mathcal{M}^{(k)}$ nello stato q_i
2. riscrivere i k simboli puntati dalle testine dei k nastri
3. posizionare i k simboli \downarrow nella posizione delle testine della macchina $\mathcal{M}^{(k)}$ nello stato q_j
4. transire di stato

Quindi ad ogni passo di $\mathcal{M}^{(k)}$, la macchina \mathcal{M}' deve eseguire un numero di passi proporzionale alla distanza, in termini di numero di celle, tra i due simboli \downarrow più lontani.

Ad ogni passo i simboli \downarrow possono al più allontanarsi di 2 celle, quindi dopo t passi, nel caso pessimo, si saranno allontanati di $2t$ celle.

Quindi se la macchina multi-nastro $\mathcal{M}^{(k)}$ compie t passi, il numero di passi eseguiti dalla macchina singolo-nastro \mathcal{M}' per simularla sarà:

$$\Sigma_{i=1}^t 2i = 2\Sigma_{i=1}^t i = 2\frac{t(t+1)}{2} = t^2 + t = \mathcal{O}(t^2)$$

Per quanto riguarda la dimensione dell'alfabeto Σ' , osserviamo che quella dell'alfabeto dei nastri dispari è 2, mentre quella dei nastri pari, per ogni nastro i è $|\Sigma_i|$.

Quindi, la dimensione dell'alfabeto Σ' usato da \mathcal{M}' sarà pari al prodotto di tutte le cardinalità degli alfabeti dei singoli nastri:

$$\prod_{i=1}^k 2|\Sigma_i| = \mathcal{O}((\max|\Sigma_i|)^k)$$

Quindi una MT multinastro può essere simulata da una MT singolo-nastro multi-traccia in un tempo quadratico usando un alfabeto di cardinalità esponenziale nel numero dei nastri.

Macchine di Turing NON Deterministiche

Esempio: Definire una macchina di Turing non deterministica che riconosca le stringhe del tipo xaa con $x \in \{a,b\}^*$.

Sia \mathcal{M} la macchina di Turing non-deterministica definita nel seguente modo:

$$\mathcal{M} = (\{a, b\}, \emptyset, \{q_0, q_1\}, \delta, q_0, q_f)$$

ove la funzione δ è definita dalla seguente matrice:

δ	a	b	\emptyset
q_0	$(q_0, a, d), (q_1, a, d)$	(q_0, b, d)	—
q_1	(q_2, a, d)	—	—
q_2	—	—	(q_f, \emptyset, i)

Esercizio: scrivere la computazione sulla stringa abbabaa.

Macchine di Turing NON Deterministiche

Una macchina di Turing **non deterministica** è definita da 6-upla:

$$\mathcal{M} = (\Sigma, \emptyset, K, \delta, q_0, q_f)$$

ove la funzione di transizione δ è definita nel seguente modo:

$$\delta : (K - \{q_f\}) \times \Sigma_\emptyset \rightarrow \mathcal{P}(K \times \Sigma_\emptyset \times \{d, s, i\})$$

cioè da ogni configurazione si può transire in una o più configurazione simultaneamente.

La configurazione successiva **non è univocamente determinata** e la computazione non è più una successione di configurazioni ma secondo un albero di configurazioni.

Il **grado di non determinismo** corrisponde, dato una generica configurazione, al massimo numero di configurazioni generate dalla funzione δ .

Una MT non deterministica si comporta come se ad ogni passo instanziasse nuove MT, ognuna delle quali elabora una delle configurazione diverse prodotte dalla funzione di transizione δ .

Equivalenza MTND e MT deterministiche

Teorema Per ogni MTND \mathcal{M} esiste una MT deterministica $\mathcal{M}^{(3)}$ deterministica a 3 nastri equivalente.

Dimostrazione La simulazione della macchina di Turing non deterministica \mathcal{M} tramite una deterministica $\mathcal{M}^{(3)}$ si ottiene visitando l'albero delle computazioni di \mathcal{M} utilizzando l'algoritmo di visita **breadth first**.

NB: la visita **non** può essere fatta in modo **depth first** poichè visitando un ramo corrispondente ad una computazione **infinita**, l'algoritmo di visita non terminerebbe.

Ad ogni passo di computazione di \mathcal{M} si possono generare al massimo d scelte, ove d è il grado di non determinismo della macchina \mathcal{M} .

Supponiamo di numerare con numeri compresi tra 1 e d le scelte derivanti dalla funzione di transizione di \mathcal{M} .

In tal modo ogni computazione potrà essere identificata come una sequenza di numeri compresi tra 1 e d , ognuno dei quali identifica una delle possibili d scelte generate dalla funzione di transizione di \mathcal{M} .

Non tutte le combinazioni saranno valide poichè non è detto che ad ogni passo la funzione di transizione generi esattamente d scelte.

Dopo i passi di computazione della macchina \mathcal{M} , quindi esistono al più d^i stringhe di lunghezza i che rappresentano particolari computazioni di \mathcal{M} .

Si supponga quindi di organizzare la macchina $\mathcal{M}^{(3)}$ nel seguente modo:

- ▷ il primo nastro contiene la stringa di input
- ▷ il secondo nastro contiene, per ogni passo di computazione i di \mathcal{M} , stringhe di lunghezza i , corrispondenti a sequenze di numeri compresi tra 1 e d . Fissato i il numero di stringhe sarà al più d^i .
- ▷ il terzo nastro eseguirà la simulazione vera e propria

La simulazione avviene secondo il seguente algoritmo:

1. $\forall i \geq 1$ passo di computazione di \mathcal{M} , si generano sul nastro 2 tutte le stringhe di lunghezza i , corrispondenti a possibili sequenze di scelte per computazioni di lunghezza i . La generazione delle stringhe avviene una alla volta.
2. per ogni sequenza di lunghezza i :
 - (a) si copia il contenuto del nastro 1 sul nastro 3
 - (b) si scandisce il nastro 2, e per ogni j , indice di una possibile scelta, si applica la j -esima scelta di δ al nastro 3

Se esiste un cammino di lunghezza l che porta la macchina \mathcal{M} in uno stato finale, allora esiste sicuramente una fase di calcolo di $\mathcal{M}^{(3)}$ che percorre tale cammino.

Se viceversa tale cammino non esiste allora anche $\mathcal{M}^{(3)}$ non raggiungerà mai lo stato finale.

Ad ogni passo j della computazione di \mathcal{M} , la MT $\mathcal{M}^{(3)}$ compie un numero di passi pari alla lunghezza del cammino (j) per il numero dei cammini (d^j), ovvero:

$$j \cdot d^j$$

Se la macchina \mathcal{M} termina in $k \geq 0$ passi, allora la macchina $\mathcal{M}^{(3)}$ esegue al più un numero di passi pari a:

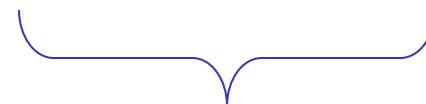
$$\sum_{j=1}^k j \cdot d^j \in \mathcal{O}(kd^k)$$

Quindi una MTND può essere simulata da una MT deterministica multi-nastro in un tempo esponenziale nel numero dei passi della macchina non deterministica.

Macchina Turing Universale

Una limitazione delle macchine di Turing

Turing Machines sono “hardwired”



Eseguono un solo
programma

Computer Reali sono ri-programmabili

Soluzione: Universal Turing Machine

Attributi:

- macchina Riprogrammabile
- Simula ogni altra Macchina di Turing

Universal Turing Machine

- Simula ogni altra Macchina di Turing M

Input della Universal Turing Machine:

Descrizione delle transizioni di M

stringa di input di M

Tre nastri



Tape 1

Descrizione di M

Tape 2

Contenuti del nastro di M

Tape 3

Stato di M

Tape 1

--	--	--	--	--

Descrizione di M

Descriviamo le Turing machines M

Come una stringa di simboli:

codifichiamo M Come una
stringa di simboli

codice Alfabeto

Simboli:

a



b



c



d



...

codifica:

1

11

111

1111

Codifica degli stati

Stati:

q_1

q_2

q_3

q_4

...



codifica:

1

11

111

1111

Codifica dei movimenti della Head

Mossa:

L

R



codifica:

1

11

codifica delle transizione

Transizione: $\delta(q_1, a) = (q_2, b, L)$

codifica:

1 0 1 0 1 1 0 1 1 0 1

separatore

Codifica Turing Machine

Transizione:

$$\delta(q_1, a) = (q_2, b, L)$$

$$\delta(q_2, b) = (q_3, c, R)$$

Codifica:

1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1

↑
separatore

Tape 1 della Universal Turing Machine contiene:

Codifica binaria

della macchina da simulare M

Tape 1

1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 1 0 1 1 1 0 1 1 1 0 1 1 0 0 ...



Nastro due, tre : simuliamo

($q_1, c, q_{\text{new}}, c_{\text{new}}, \text{op}$)

Consideriamo tutti gli stati e tutti i caratteri (perché? Si può fare diversamente?)

Una lista

$\{(q, c, q_{\text{new}}, c_{\text{new}}, \text{op}) \mid \text{Per tutti gli stati } q, \text{ Per tutti i caratteri } c\}$

Al primo posto

q_0 per tutti i caratteri

q_1 per tutti i caratteri

q_n per tutti i caratteri.

In rosso i codici

Primo nastro gli input: Macchina \$ input
Copiamo Macchina secondo nastro ,
Copiamo input terzo nastro

Carattere osservato terzo nastro,
Stato da applicare secondo nastro

I=0

puntatore primo nastro su q_I

puntatore secondo nastro su carattere osservato C

QUI Guardo C e prendo la stringa che comincia co q_I,C

Copio nel terzo nastro (q_I,C,q_new,c_new,op)

/* eseguire l'operazione*/

C diventa c-new nel secondo nastro

Eseguo op sul secondo nastro che sposta la testina

/*Ora cosa devo fare? Da q_I devo passare a q-new */

I=new

Andare su q_I nel primo nastro

/*Come farlo? Devo spostare, sul primo nastro fino a che non trovo q_new*/

Andare Qui se q_I non è finale

Carattere osservato terzo nastro,
Stato da applicare secondo nastro

/ puntatore secondo nastro su q_0, puntatore terzo nastro primo carattere C */*

I=0

c=primo carattere

Label:

il valore di I ci dà lo stato q_I

il valore del carattere osservato, c, ci dà (q_I, c)

Eseguiamo l'operazione $(q_I, c, q_n, \text{nuovoc}, \text{op})$;

Ovvero $I=n$; $c=\text{nuovoc}$

$c=\text{carattere a op dell'osservato}$ op=L,R

Spostiamo il puntatore del terzo nastro secondo op (L,R)

Spostiamo il puntatore del secondo nastro su q_n ;

Vai a Label

Una Turing Machine è descritta
Come una stringa di 0's e 1's

quindi:

L'insieme delle Turing machines
Forma un linguaggio:

Ogni stringa di questo linguaggio è
la Codifica binaria di una Turing Machine

Linguaggio delle Turing Machines

$L = \{ 010100101011,$ (Turing Machine 1)
 $00100100101111,$ (Turing Machine 2)
 $111010011110010101,$
..... }

Insiemi contabili

Countable sets

Insieme infiniti sono:

Countable (enumerabili)

or

Uncountable (non enumerabili)

Countable set:

Esiste una corrispondenza uno a uno
tra
Gli elementi dell'insieme
e
Numeri naturali (interi positivi)

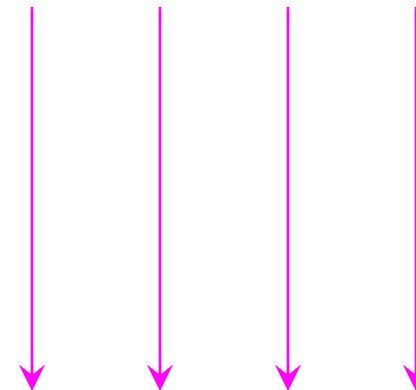
(ogni elemento dell'insieme è associato ad un numero naturale
tale che non esistono due elementi che sono associati
allo stesso numero

Esempio: L'insieme dei numeri pari

Interi pari:
(positive)

0, 2, 4, 6, ...

corrispondenza:



Interi positivi:

1, 2, 3, 4, ...

$2n$ Corrisponde a $n + 1$

Esempio: L'insieme dei numeri razionali

Numeri razionali:

$$\frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \dots$$

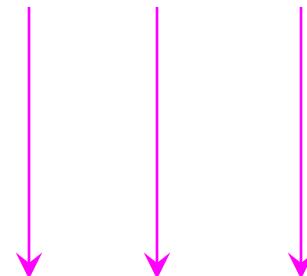
approccio banale

Numeri razionali:

Nominatore 1

$\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \dots$

Corrispondenza:



interi positivi:

1, 2, 3, ...

Non funziona:

Non analizeremo mai

Numeri con nominatore 2:

$\frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \dots$

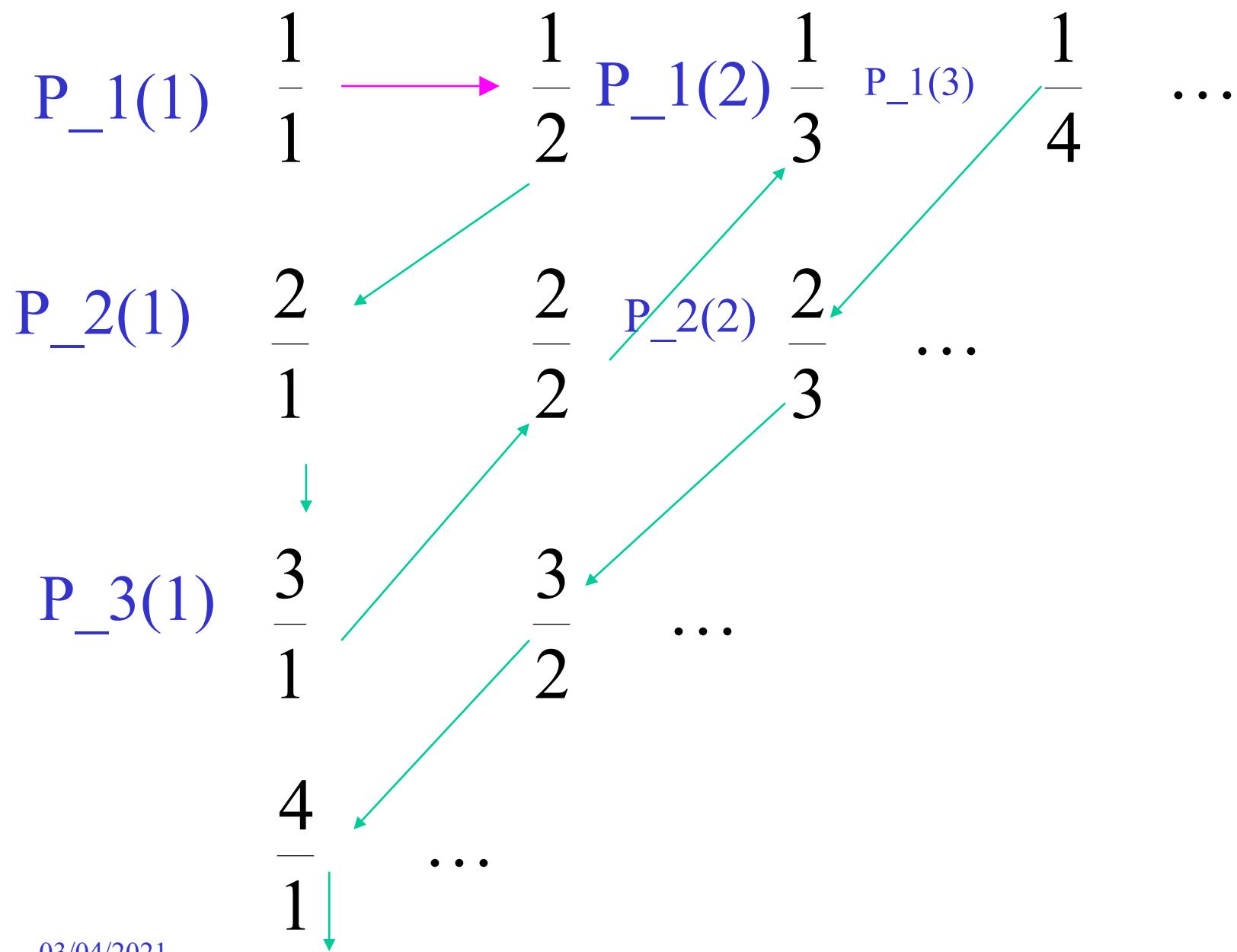
Approccio migliore

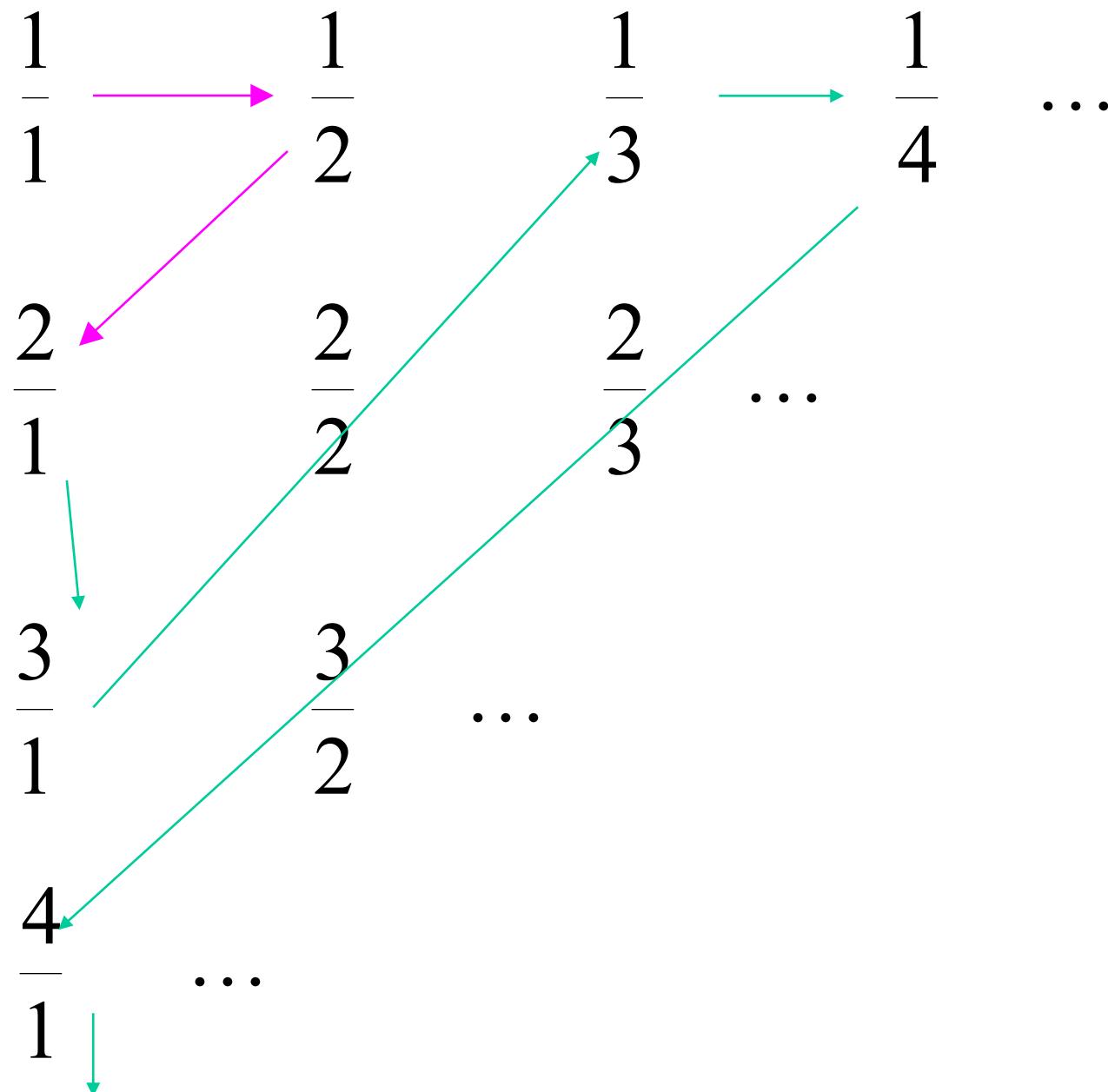
$$P_1 \quad \frac{1}{1} \quad \frac{1}{2} \quad \frac{1}{3} \quad \frac{1}{4} \quad \dots$$

$$P_2 \quad \frac{2}{1} \quad \frac{2}{2} \quad \frac{2}{3} \quad \dots$$

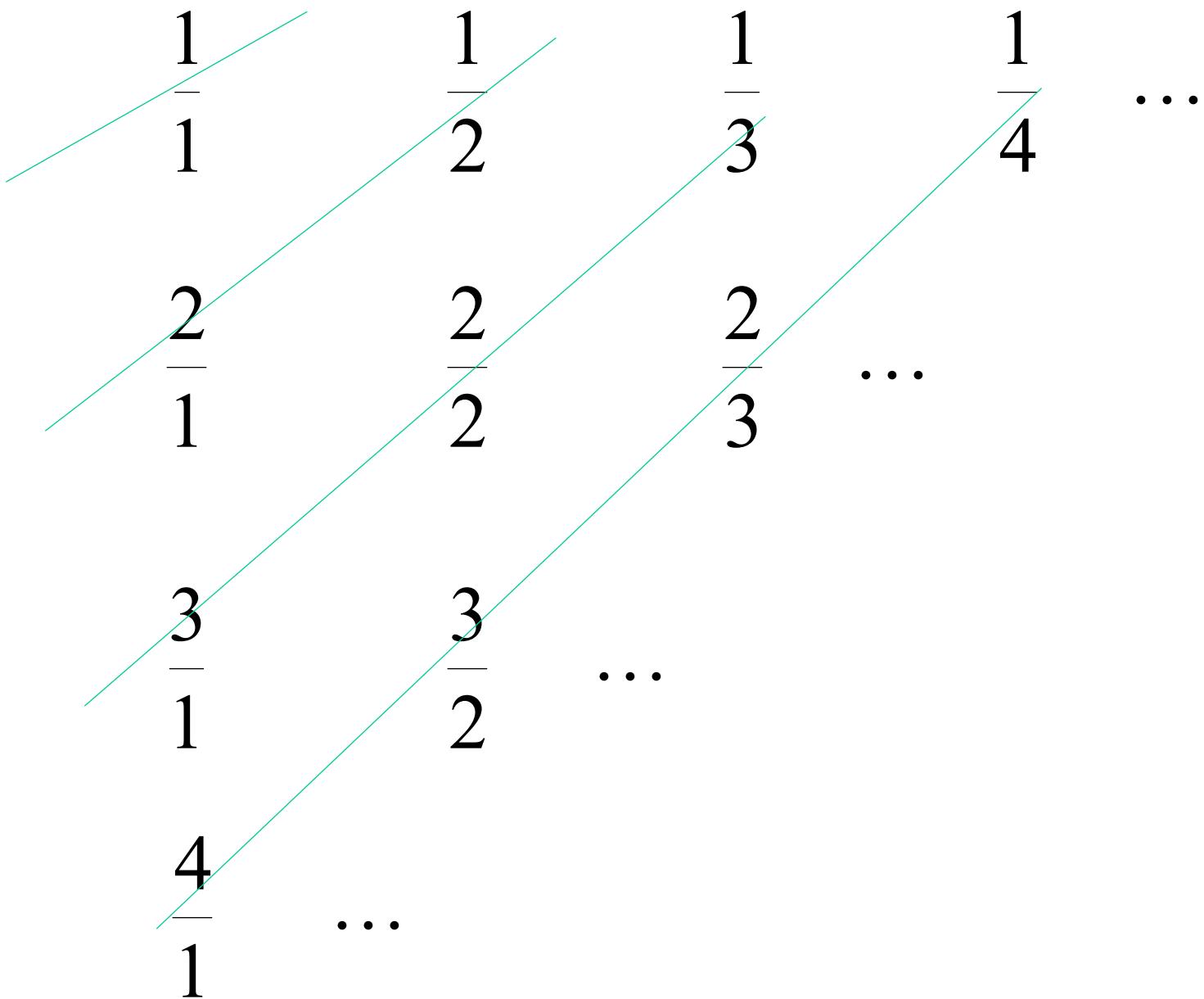
$$\frac{3}{1} \quad \frac{3}{2} \quad \dots$$

$$\frac{4}{1} \quad \dots$$





Oppure?



P₁(1)

P₁(2) P₂(1)

P₁(3) P₂(2) P₃(1)

P₁(4) P₂(3) P₃(2) P₄(1)

I=1

J=1,I

P_{I(J)}

Numeri razionali:

$$\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{2}{2}, \dots$$

Corrispondenza:

interi positivi:

$$1, 2, 3, 4, 5, \dots$$

un insieme è countable se esiste
un enumeration procedure
(enumeratore)
che definisce la corrispondenza con i
numeri naturali

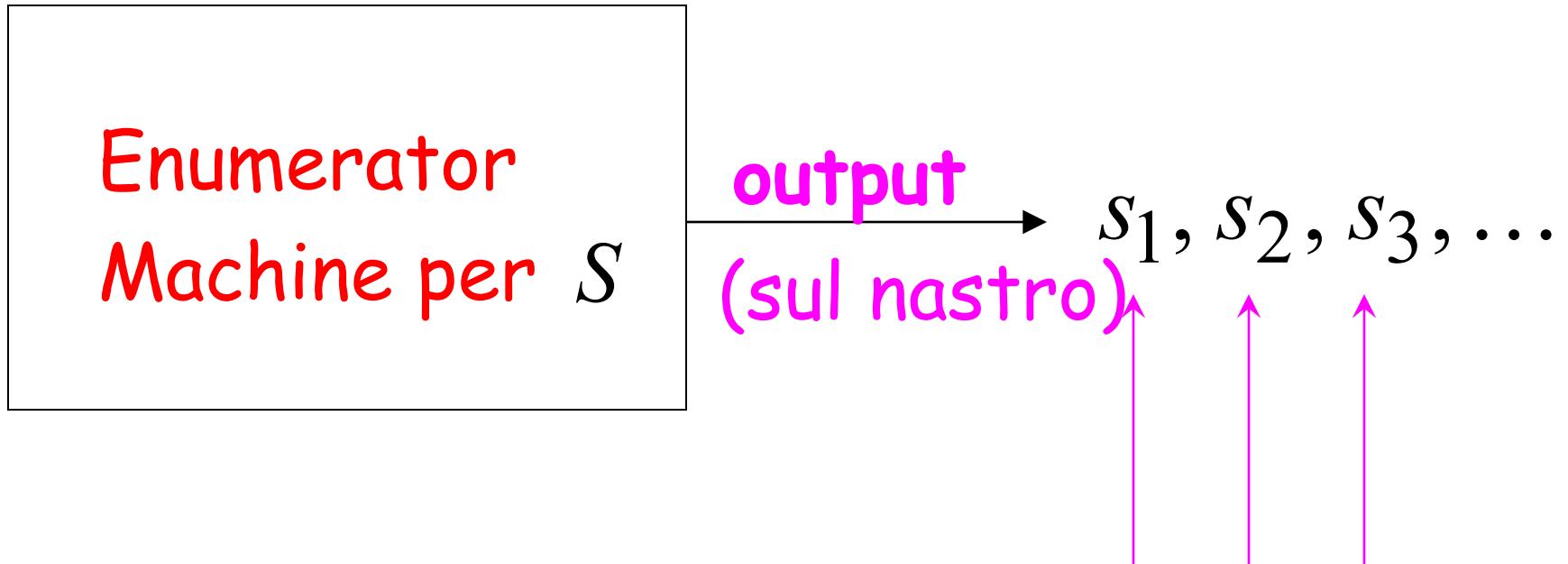
Definizione

Sia S un insieme di stringhe (linguaggio)

Un **enumerator** per S è una Turing Machine
che genera (scrive sul nastro)
tutte le stringhe S una per una

e
Ogni stringa è generata in tempo finito

stringhe $s_1, s_2, s_3, \dots \in S$



Osservazione:

Se esiste per S un enumeratore,
Allora l'insieme è countable

L'enumeratore descrive la corrispondenza
di S con i numeri naturali

Esempio: L'insieme delle stringhe
è countable $S = \{a, b, c\}^+$

Approccio:

Descriviamo un enumeratore per S

Enumeratore banale:

Produrre le stringhe in ordine lessicografico

$s_1 = a$

$s_2 = aa$

$\vdots \quad aaa$

$aaaa$

.....

No buono:

le stringhe con primo carattere b
non vengono fuori

Procedura migliore : **Ordine proprio** **(Canonical, proper, Order)**

1. Producì tutte le stringhe di lunghezza 1
2. Producì tutte le stringhe di lunghezza 2
3. Producì tutte le stringhe di lunghezza 3
4. Producì tutte le stringhe di lunghezza 4
-

$$\begin{aligned}s_1 &= a \\ s_2 &= b \\ \vdots &= c\end{aligned}\quad \left. \right\} \text{lunghezza 1}$$

Produce stringhe in
Proper Order:

$$\begin{aligned}aa \\ ab \\ ac \\ ba \\ bb \\ bc \\ ca \\ cb \\ cc \\ \dots \\ aaa \\ aab \\ aac \\ \dots\end{aligned}\quad \left. \right\} \text{lunghezza 2}\quad \left. \right\} \text{lunghezza 3}$$

Codifica degli stati

Stati:

q_1

q_2

q_3

q_4

...



codifica:

1

11

111

1111

Codifica dei movimenti della Head

Mossa:

L

R



codifica:

1

11

codifica delle transizione

Transizione: $\delta(q_1, a) = (q_2, b, L)$

codifica:

1 0 1 0 1 1 0 1 1 0 1

separatore

Teorema: L'insieme di tutte le Turing Machines è countable

Proof: Ogni macchina di Turing può essere codificata

Con una stringa binaria di 0's e 1's

Definite un enumeration procedure
Per l'insieme delle stringhe che
Descrivono le Turing Machine

Enumerator:

Repeat

1. Genera le stringhe binarie di 0's e 1's in proper order
2. Check se la stringa generate descrive una Turing Machine
 - if YES: print string sull'output tape
 - if NO: ignora string

Binary strings

0

1

00

01

⋮

⋮

1 0 1 0 1 1 0 1 1 0 0

1 0 1 0 1 1 0 1 1 0 1

⋮

⋮

1 0 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1

⋮

⋮

s_1

Turing Machines

1 0 1 0 1 1 0 1 1 0 1

s_2

1 0 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 1

End of Proof

Uncountable Sets

definizione: Un insieme è uncountable L'
se non è countable, ovvero
se non esiste un enumeratore
che lo enumera

Vogliamo provare che vi è un linguaggio
che non è accettato da nessuna macchina di
Turing

Tecnica:

1_ Turing machines sono countable

2_ Linguaggi sono uncountable

(Vi sono più linguaggi che Turing Machines)

Teorema:

Se S è un infinito enumerabile, allora l'insieme delle parti 2^S di S non è enumerabile.

Proof:

Poichè S è enumerabile, possiamo scrivere

$$S = \{s_1, s_2, s_3, \dots\}$$



Elementi di S

Gli elementi dell'insieme delle parti 2^S
hanno la forma:



$$\{s_1, s_3\}$$

$$\{s_5, s_7, s_9, s_{10}\}$$

.....

Assurdo: Sia l'insieme delle parti 2^S
enumerabile

Codifichiamo ogni insieme

con una stringa binaria di 0 e 1.

Elementi

Insieme delle
parti
(in ordine arbitrario)

Codifica Binaria

s_1 s_2 s_3 s_4 ...

$\{s_1\}$ 1 0 0 0 ...

$\{s_2, s_3\}$ 0 1 1 0 ...

$\{s_1, s_3, s_4\}$ 1 0 1 1 ...

Osservazione:

Ogni stringa binaria infinita corrisponde
a un elemento dell' insieme delle parti

:

esempio:

1 0 0 1 1 1 0 ...

```
graph TD; S1 --- 1; S1 --- 0; S4 --- 0; S4 --- 1; S4 --- 1; S5 --- 1; S5 --- 1; S5 --- 1; S6 --- 0; S6 --- ...;
```

Corrispondente a $\{s_1, s_4, s_5, s_6, \dots\} \in 2^S$

assumiamo (per assurdo)

Che l' Insieme delle parti 2^S
è enumerabile

allora: possiamo enumerare gli elementi
dell' insieme delle parti

$$2^S = \{t_1, t_2, t_3, \dots\}$$

Insieme delle
Parti elementi

Supponiamo che la seguente sia
Codifica Binaria

t_1 1 0 0 0 0 ...

t_2 1 1 0 0 0 ...

t_3 1 1 0 1 0 ...

t_4 1 1 0 0 1 ...

...

...

Prendiamo la diagonale e complementiamola

t_1	1	0	0	0	0	...
t_2	1	1	0	0	0	...
t_3	1	1	0	1	0	...
t_4	1	1	0	0	1	...

Stringa binaria: $\mathbf{t} = 0011\dots$

Stringa binaria

$t = 0011\dots$

Corrisponde ad un
elemento dell'Insieme
delle parti 2^S :

$t = \{s_3, s_4, \dots\} \in 2^S$

allora, t deve essere uguale a qualche t_i

$$t = t_i$$

ma,

il i-th bit nella codifica t è
il complemento i-th del bit di t_i , allora:

$$t \neq t_i \quad \text{Per ogni } i$$

Contradizione!!!

Poichè abbiamo ottenuto una contraddizione
a partire dall'ipotesi che 2^S è contabile:

L' insieme delle Parti 2^S
di S è uncountable

FINE

Una applicazione: Linguaggi

Considera l'alfabeto : $A = \{a, b\}$

L'insieme delle stringhe:

$$S = \{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

infinito e countable

(possiamo enumerare le stringhe
in ordine proprio)

Considera alfabeto : $A = \{a, b\}$

L'insieme delle stringhe:

$$S = \{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$$

infinito e countable

Ogni linguaggio è un sottoinsieme di S :

$$L = \{aa, ab, aab\}$$

Considera l'alfabeto : $A = \{a,b\}$

L'insieme delle stringhe:

$S = A^* = \{a,b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$
infinito e countable

Ricorda: L'insieme delle parti di S
contiene tutti i linguaggi:

$2^S = \{\emptyset, \{\lambda\}, \{a\}, \{a,b\}, \{aa,b\}, \dots, \{aa,ab,aab\}, \dots\}$
uncountable

Considera alfabeto : $A = \{a, b\}$

Turing machines:

M_1 M_2 M_3 ...

accetta

Linguaggi accettati da

L_1 L_2 L_3 ...

countable

Denota: $X = \{L_1, L_2, L_3, \dots\}$

countable

Nota: $X \subseteq 2^S$
 $(S = \{a, b\}^*)$

Linguaggi accettati da
Turing machines: X countable

Tutti i possibili linguaggi:

2^S uncountable

quindi: $X \neq 2^S$

(since $X \subseteq 2^S$, we have $X \subset 2^S$)

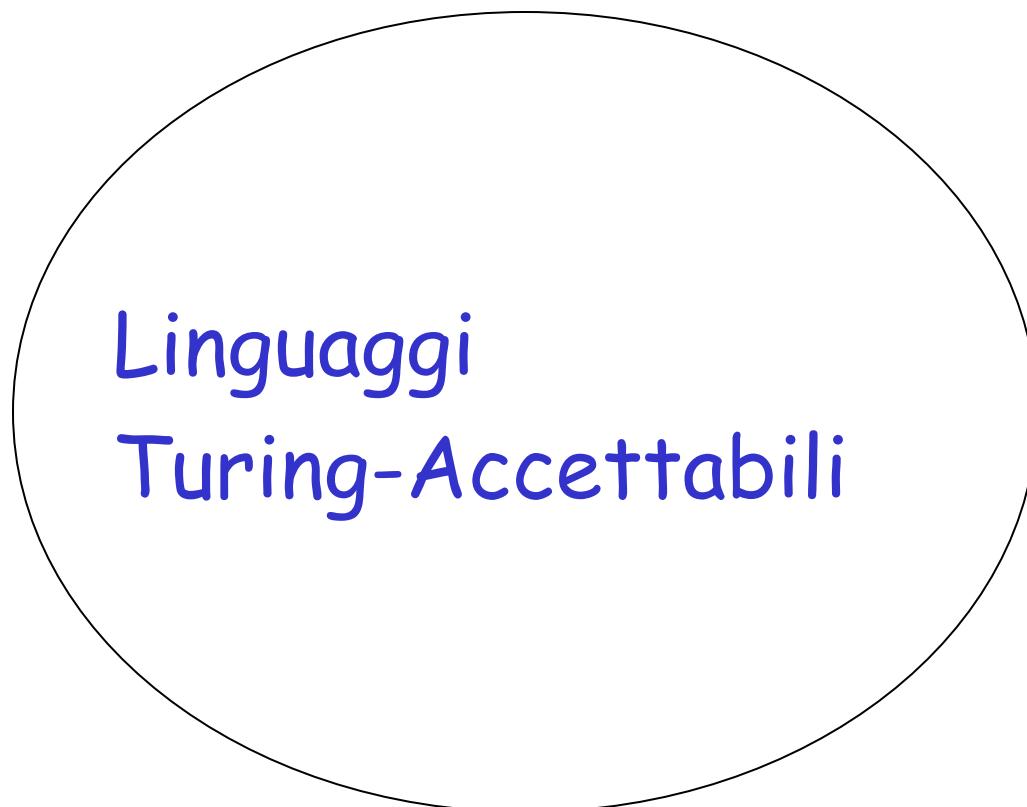
Conclusione:

Esiste un linguaggio L' non accettato da nessuna Turing Machine:

$$X \subset 2^S \rightarrow \exists L' \in 2^S \text{ and } L' \notin X$$

(linguaggio L' non può essere descritto da nessun algoritmo)

Linguaggi Non Turing-Accettabili



L'

Nota che: $X = \{L_1, L_2, L_3, \dots\}$

È un multi-set (elementi possono ripetersi)
Poichè un linguaggio può essere riconosciuto
da più di una Turing machine

Anche se esaminiamo i doppioni il risultato
è di nuovo un insieme countable poichè ogni elemento
corrisponde a un intero positivo

La gerarchia di Chomsky

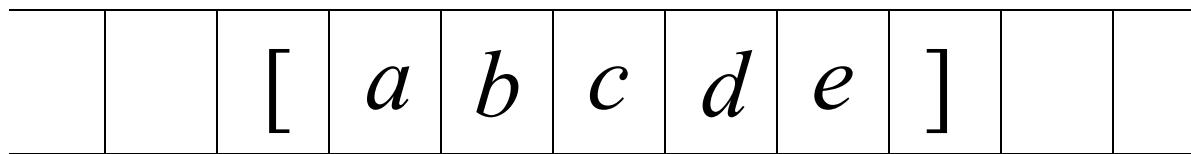
Linear-Bounded Automa:

Come una Macchina di Turing
con una differenza:

Lo spazio dove è memorizzato l'input
è il solo spazio che può essere
utilizzato

Linear Bounded Automa (LBA)

Input string



Working space
in tape

Limite a
sinistra

Limite a
destra

Tutta la computazione si svolge tra i due limiti

Definiamo i LBA come macchine
non deterministiche

Problema aperto:

LBA NonDeterministici
hanno lo stesso potere dei
LBA Deterministici ?

Esempio linguaggio accettato da un LBA:

$$L = \{a^n b^n c^n\} \qquad L = \{a^{n!}\}$$

LBA hanno più potere dei PDA
(pushdown automata)

LBA hanno meno potere delle Turing Machines

Grammatica Context-Sensitive :

produzioni

$$u \rightarrow v$$



Stringhe di variabili
e terminali

Stringhe di variabili
e terminali

e: $|u| \leq |v|$

Il linguaggio $\{a^n b^n c^n\}$

è context-sensitive:

$$S \rightarrow abc \mid aAbc$$

$$Ab \rightarrow bA$$

$$Ac \rightarrow Bbcc$$

$$bB \rightarrow Bb$$

$$aB \rightarrow aa \mid aaA$$

Teorema:

Un linguaggio L è context sensitive



è accettato da Linear-Bounded automata

Devo provare il
contrario

Dato L (data la grammatica per L) devo costruire una LB che riconosce tutte e solo le parole di L

stringa elemento di L la macchina mi deve dire si
stringa non è elemento di L la macchina mi deve dire **no**

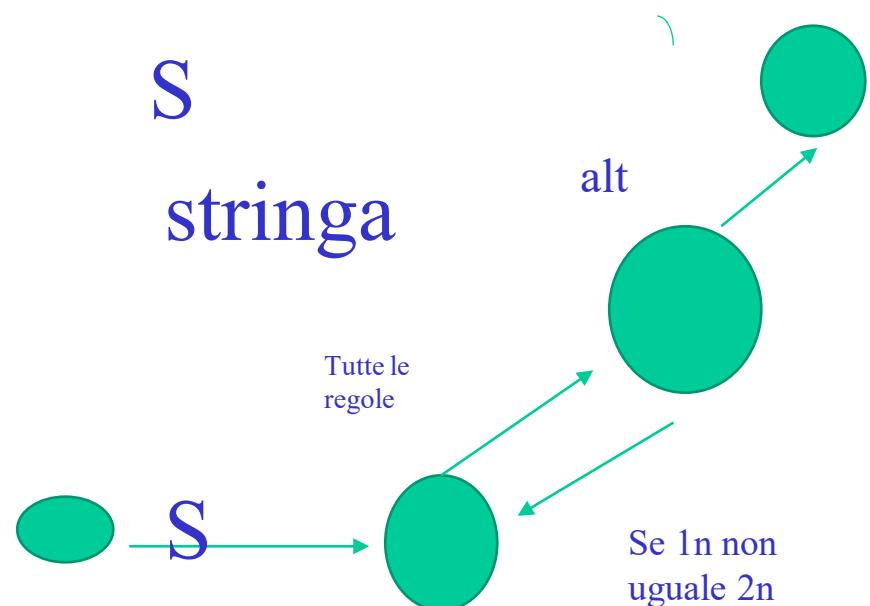
$$S \rightarrow abc \mid aAbc$$

$$Ab \rightarrow bA$$

$$Ac \rightarrow Bbcc$$

$$bB \rightarrow Bb$$

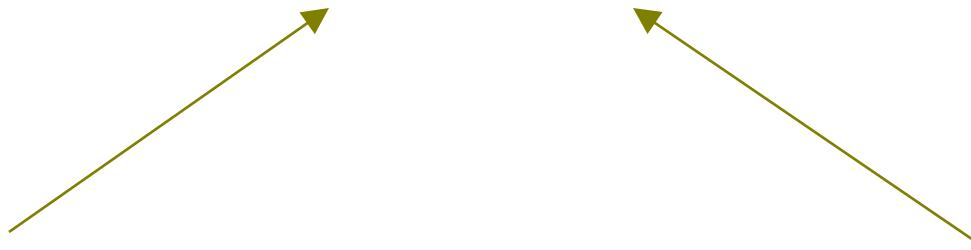
$$aB \rightarrow aa \mid aaA$$



Grammatiche senza limitazioni:

Produzioni

$$u \rightarrow v$$



Stringhe di variabili
e terminali

Stringhe di variabili
e terminali

Teorema:

Un linguaggio L è Turing-Acceptable

Se L è generato da

una grammatica senza restrizione

Perchè? Come fare?

The Chomsky gerarchia

Non Turing-Acceptable

Turing-Acceptable

decidable

Context-sensitive

Context-free

Regular

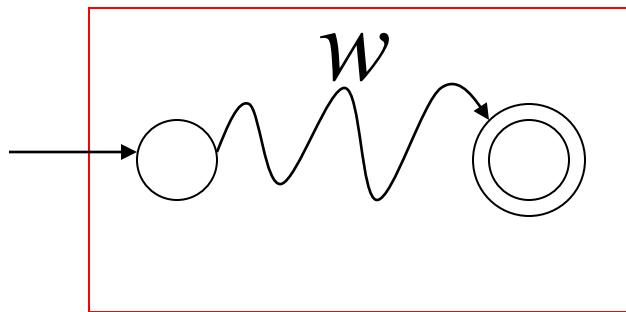
problemi decidibili
(decidibile?)
per linguaggi regolari

appartenenza

Domanda: dato un linguaggio regolare L
e una stringa w
Possiamo verificare se $w \in L$?

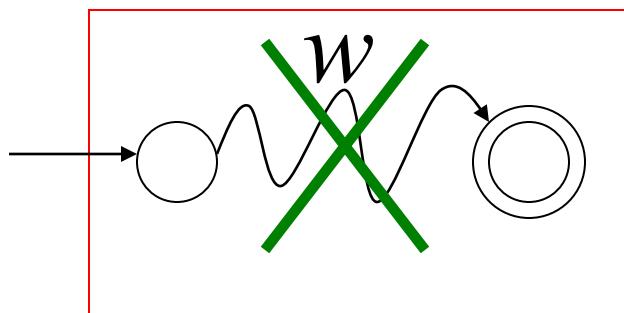
Risposta: Prendiamo un DFA che
accetta L
e verifichiamo se w è
accettato.

DFA



$w \in L$

DFA



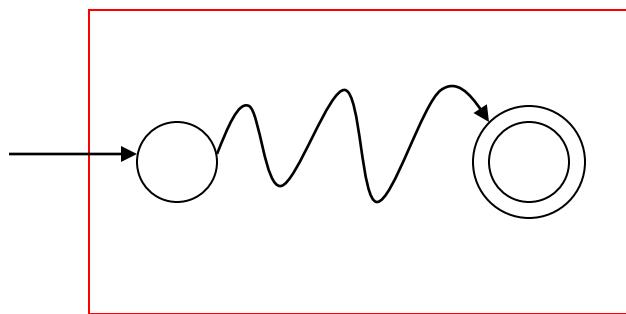
$w \notin L$

Domanda: dato un linguaggio regolare L
Come possiamo verificare
se L è vuoto: $(L = \emptyset)$?

Risposta: Prendiamo il DFA che accettà L

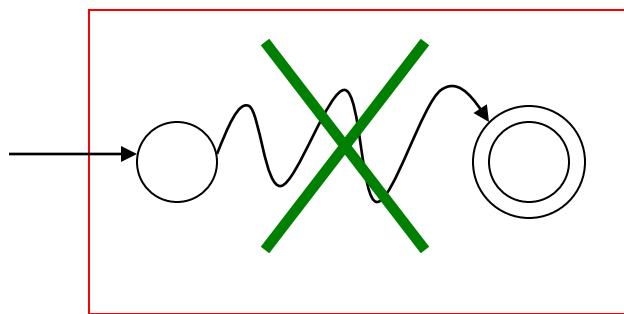
Verifichiamo se esiste almeno
un cammino da uno stato iniziale
ad uno stato di accettazione

DFA



$$L \neq \emptyset$$

DFA



$$L = \emptyset$$

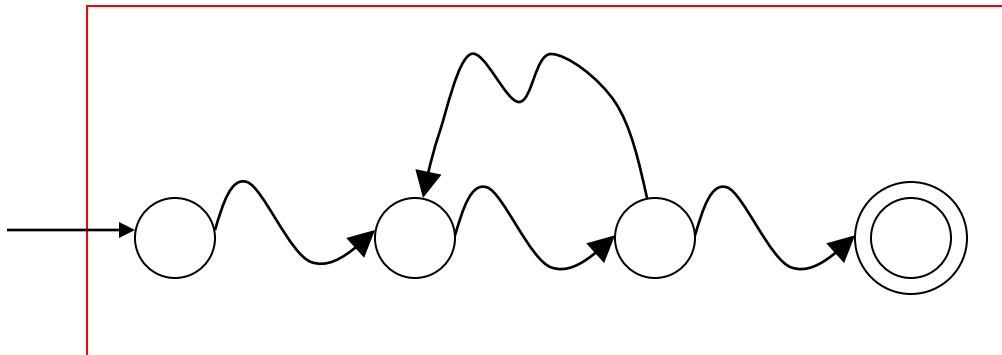
Domanda: Dato un linguaggio regolare L
Come possiamo verificare
Se L è finito?

Prendiamo il DFA che accetta L

Risposta:

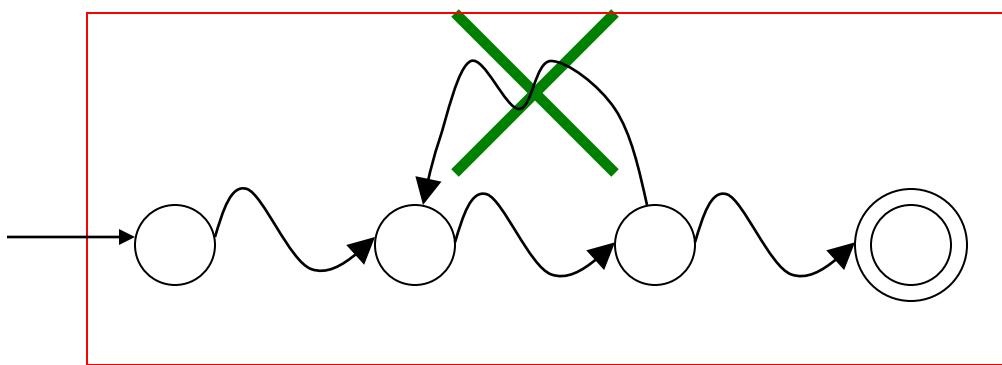
Verifichiamo se vi è un cammino con
un loop.

DFA



L è infinito

DFA



L è finito

Domanda: dato linguaggi regolari L_1 e L_2
Come possiamo verificare che

$$L_1 = L_2$$

Risposta: Trova se

$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$$

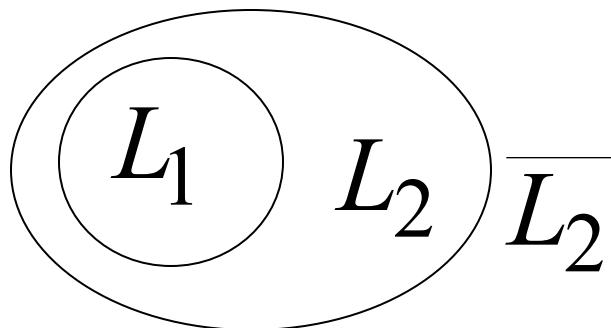
$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$$



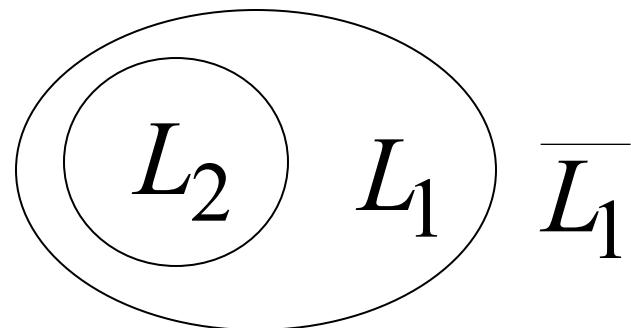
$$L_1 \cap \overline{L_2} = \emptyset$$

and

$$\overline{L_1} \cap L_2 = \emptyset$$



$$L_1 \subseteq L_2$$



$$L_2 \subseteq L_1$$



$$L_1 = L_2$$

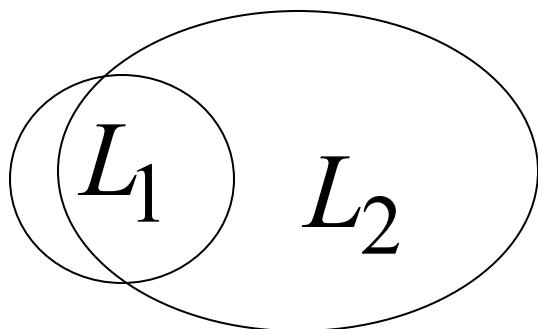
$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) \neq \emptyset$$



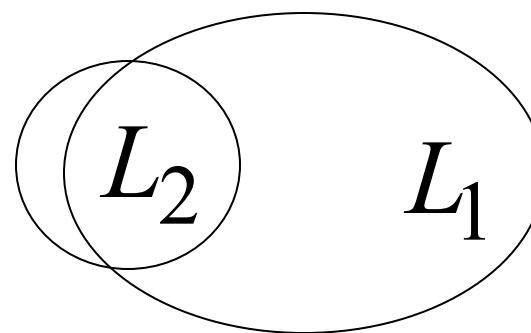
$$L_1 \cap \overline{L_2} \neq \emptyset$$

or

$$\overline{L_1} \cap L_2 \neq \emptyset$$



$$L_1 \subset L_2$$



$$L_2 \subset L_1$$



$$L_1 \neq L_2$$

problemi decidibili per linguaggi Context-Free

appartenenza:

per grammatiche context-free G
Se la stringa $w \in L(G)$

appartenenza : Parsers

- “Exhaustive search parser” o “non determinismo”
 - CYK parsing algorithm

Teorema:

Un linguaggio L è Turing-Acceptable
Se L è generato da
una grammatica senza restrizione

Grammatica definire Turing machine
(linear bounded uguale)



Turing machine  una grammatica senza restrizione

? = Qualsiasi carattere

Stati= non terminali, car =terminali

$$Q \ c := Q \ ? \ c \quad Q \ c = c \ Q$$

$$\Delta(Q, c) = (Q, c, L) \quad Q \ c = Q \ ? \ c$$

$$\Delta(Q, c) = (Q, c, R)$$

Spostandosi trovo il blank
 $\Delta(Q, c) = (Q, c, L)$

$\Delta(Q, b) = (Q, c, R|L)$

F:= fermo la
computazione

La gerarchia di Chomsky

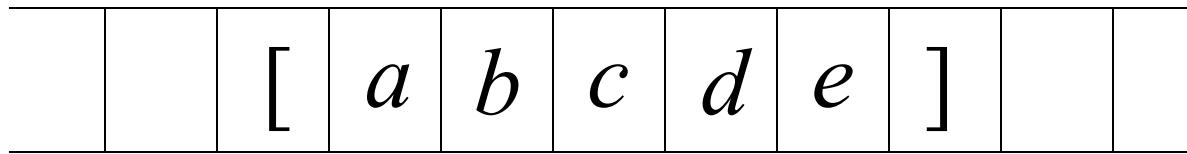
Linear-Bounded Automa:

Come una Macchina di Turing
con una differenza:

Lo spazio dove è memorizzato l'input
È il solo spazio che può essere
utilizzato

Linear Bounded Automa (LBA)

Input string



Working space
in tape

Limite a
sinistra

Limite a
destra

Tutta la computazione si svolge tra I due limiti

Definiamo I LBA come macchine
non deterministiche

Problema aperto:

LBA NonDeterministici
Hanno lo stesso potere dei
LBA Deterministici ?

Esempio linguaggio accettato da un LBA:

$$L = \{a^n b^n c^n\} \qquad L = \{a^{n!}\}$$

LBA hanno più potere dei PDA
(pushdown automata)

LBA hanno meno potere delle Turing Machines

Grammatiche senza limitazioni:

Produzioni

$$u \rightarrow v$$



Stringhe di variabili
e terminali

Stringhe di variabili
e terminali

Esempio di grammatiche senza restrizioni:

$$S \rightarrow aBc$$

$$aB \rightarrow cA$$

$$Ac \rightarrow d$$

Teorema:

Un linguaggio L è Turing-Acceptable
Se e solo se L è generato da
una grammatica senza restrizione

https://en.wikipedia.org/wiki/Unrestricted_grammar

Grammatica Context-Sensitive :

produzioni

$$u \rightarrow v$$



Stringhe di variabili
E terminali

Stringhe di variabili
E terminali

e: $|u| \leq |v|$

Il linguaggio $\{a^n b^n c^n\}$

è context-sensitive:

$$S \rightarrow abc \mid aAbc$$

$$Ab \rightarrow bA$$

$$Ac \rightarrow Bbcc$$

$$bB \rightarrow Bb$$

$$aB \rightarrow aa \mid aaA$$

Theorem:

Un linguaggio L è context sensitive
se e solo se
È accettato da Linear-Bounded automa

osservazione:

Vi è un linguaggio che è context sensitive
e non è decidibile

The Chomsky gerarchia

Non Turing-Acceptable

Turing-Acceptable

decidable

Context-sensitive

Context-free

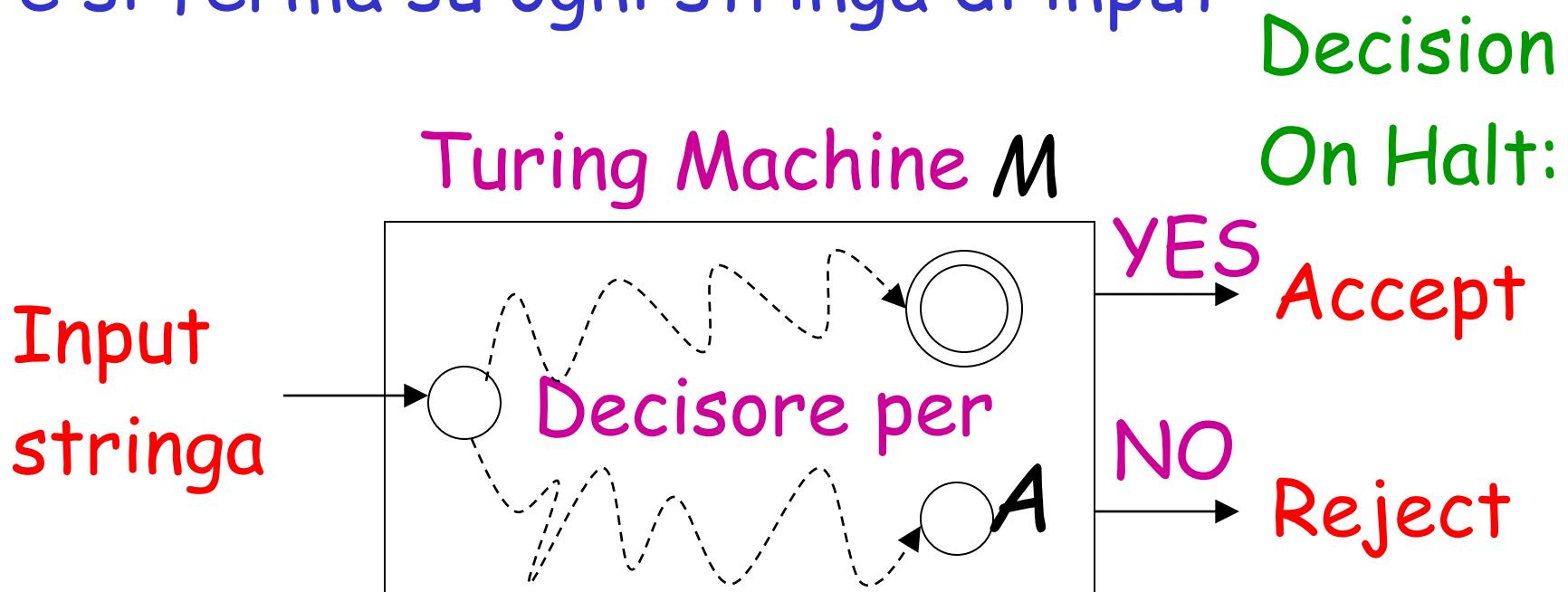
Regular

Problemi indecidibili (unsolvable problems)

linguaggi decidibili

Ricordiamo che:

un linguaggio A è decidibile,
se vi è una Turing machine M (decisore)
che accetta il linguaggio A e
e si ferma su ogni stringa di input



Definizione

Un problema computazionale è decidibile
se il corrispondente linguaggio è decidibile

linguaggi indecidibili

linguaggi indecidibili = linguaggi non decidibili

Non esiste un procedimento di decisione
(decisore):

Non esiste una Turing Machine
che accetta il linguaggio
e prende una decisione (halts)
per ogni stringa di input.

(la macchina può prendere decisioni per qualche stringa
ma non per tutte)

Per un linguaggio indecidibile,
Il corrispondente problema è
indecidibile (unsolvable):

Non esiste una Turing Machine (Algorithm)
che per ogni input
dà una risposta
(yes (appartiene al linguaggio)
or no (non appartiene al linguaggio))
(chiaramente per alcuni input si)

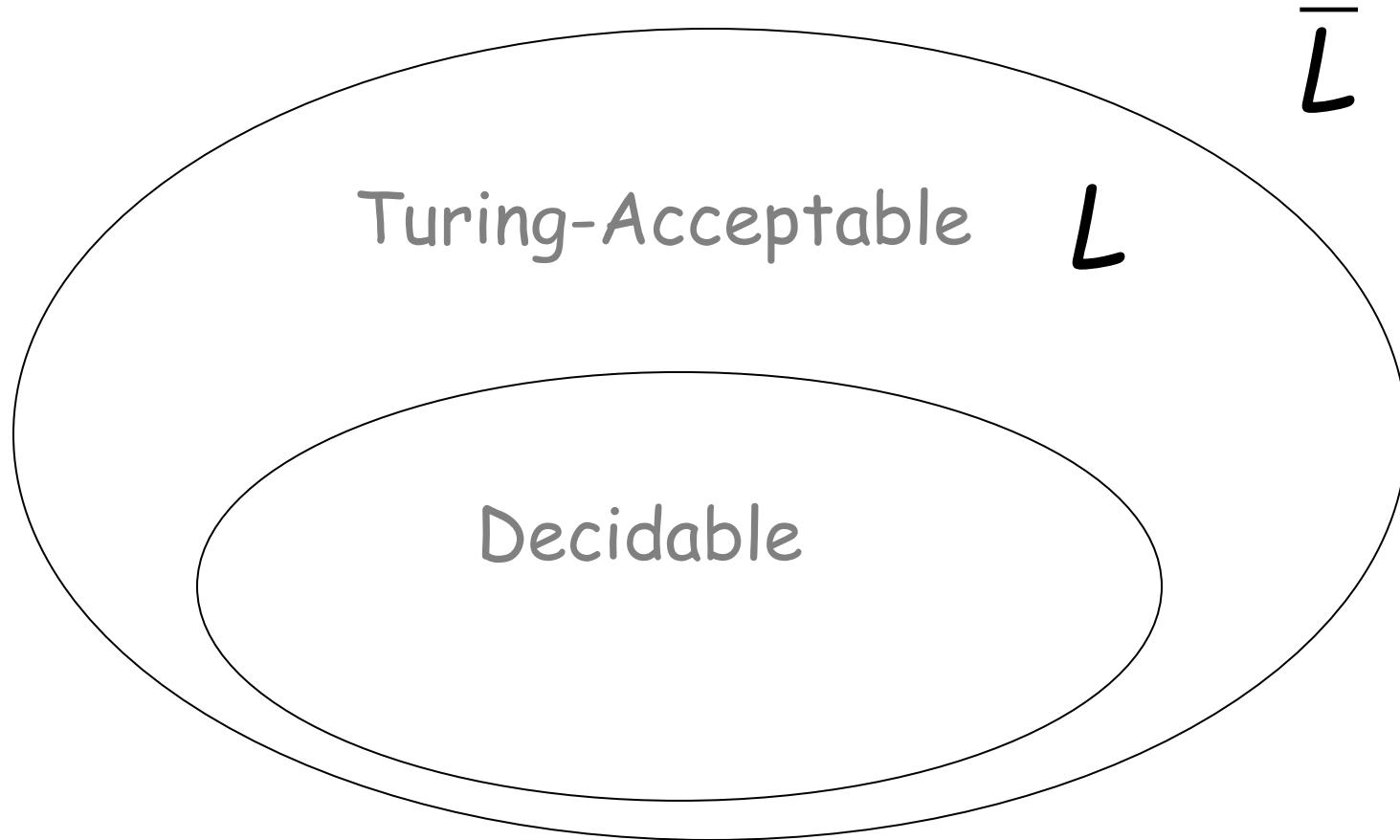
Definizione. Turing accettabile

Abbiamo una Turing Machine (Algorithm) che

1_ per ogni input abbiamo una risposta se la
stringa appartiene al linguaggio(yes)

2_ ma nulla possiamo dire se la stringa non
appartiene al linguaggio

Abbiamo già mostrato che esistono linguaggi indecidibili:



Affronteremo due particolari problemi:

Membership problem

Halting problem

Qui 16 5

Membership Problem

Input: • Turing Machine M
• String w

Question: M accetta w ?
 $w \in L(M)$?

linguaggio corrispondente:

$A_{TM} = \{\langle M, w \rangle : M \text{ è una Turing machine che accetta la stringa } w\}$

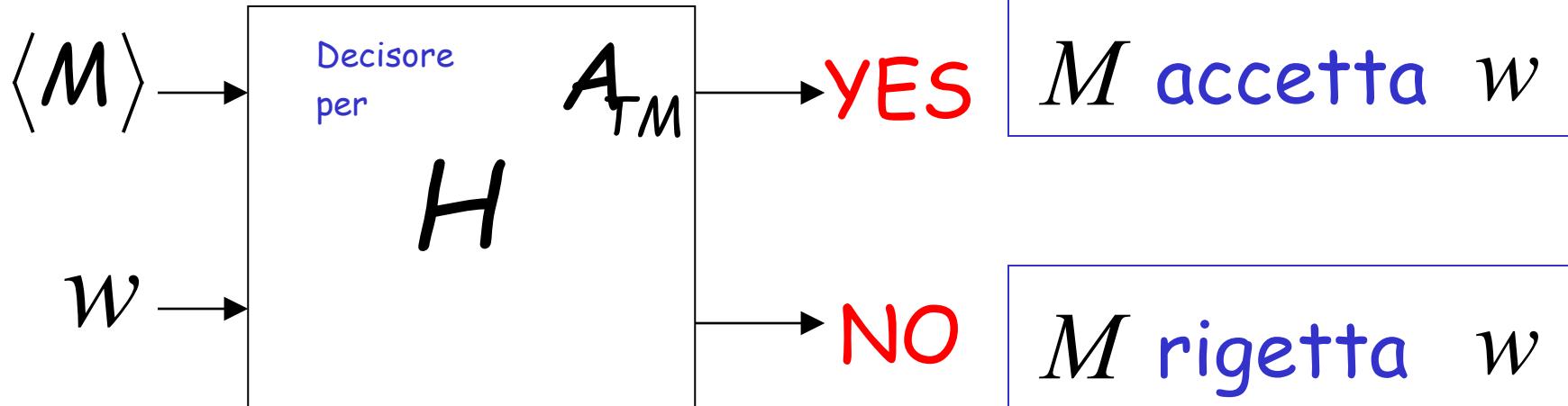
Teorema: A_{TM} è indecidibile

(The membership problem non è decibile)

Supponiamo che A_{TM} sia decidibile

Supponiamo che A_{TM} è decidibile

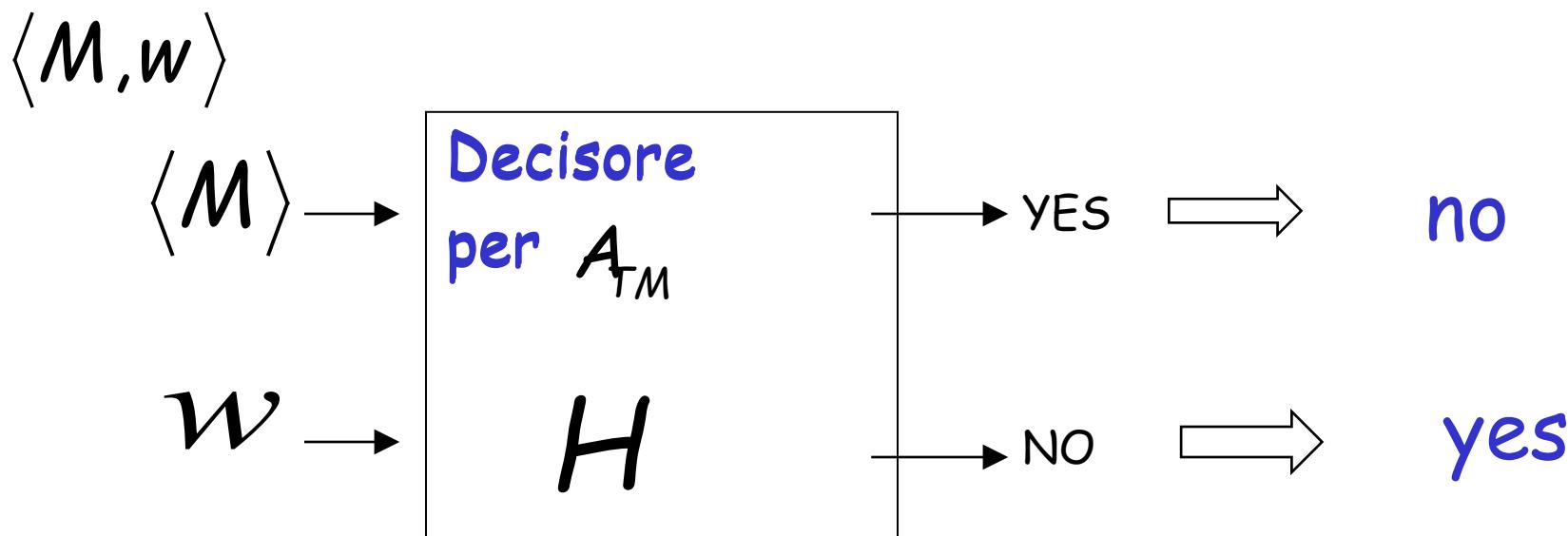
Input
string
 $\langle M, w \rangle$



Cambiamo, diagonalizziamo, definiamo la macchina Diag:

Diag accetta (yes) se H dice no; ovvero se $M(w) = \text{no}$
Diag rigetta (no) se H dice si; ovvero se $M(w) = \text{yes}$

Diag:



Semplifichiamo Diag .

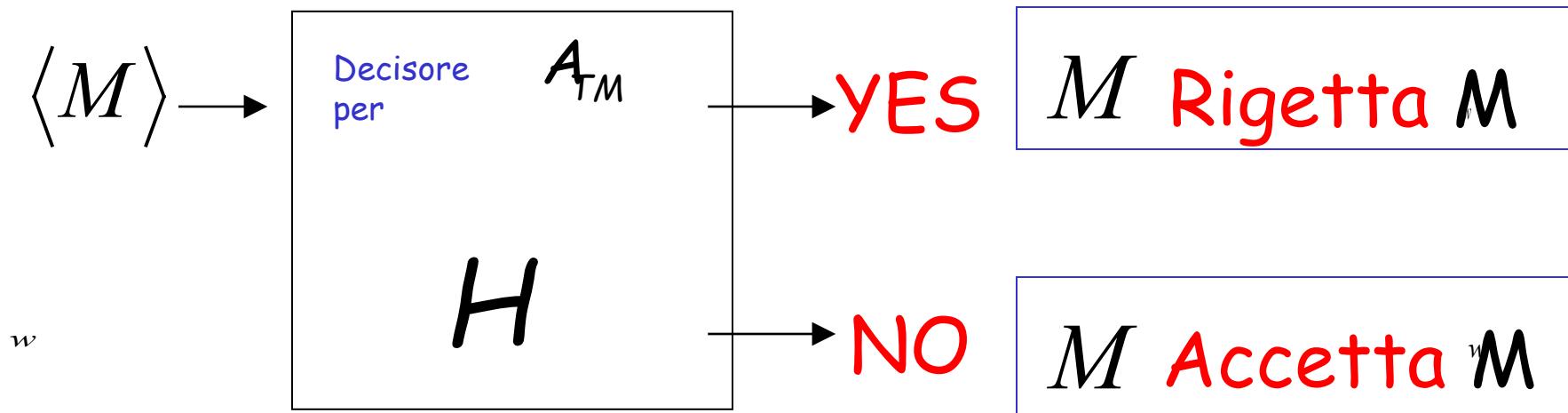
A_M

Definiamo Diag

Descrizione di Diag:

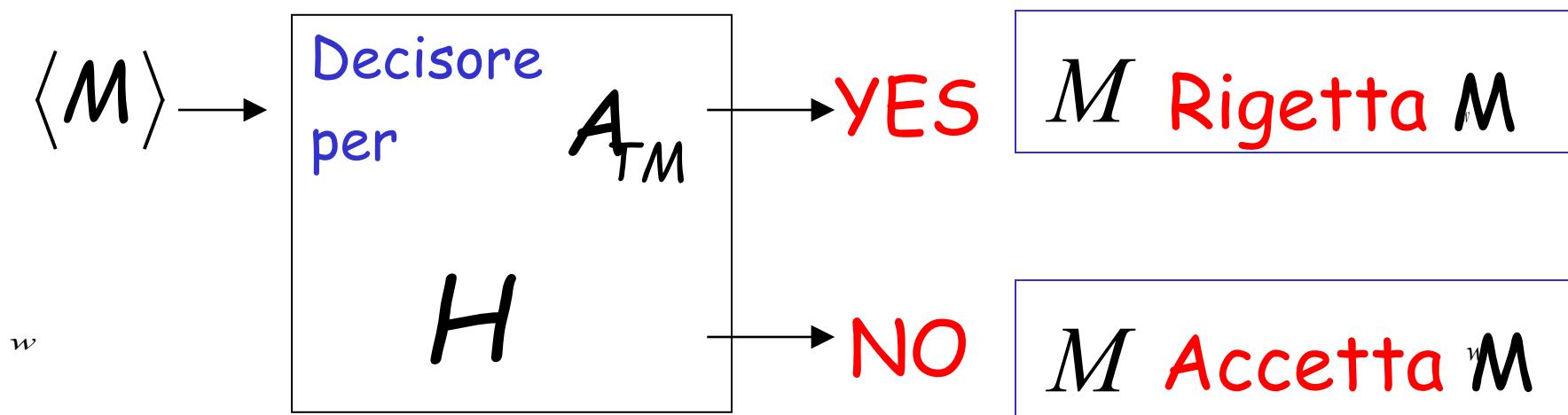
Diag accetta M se M rigetta M

Diag rigetta M se M accetta M



Descrizione di Diag:

Diag accetta (yes) Diag se Diag rigetta Diag (no)
Diag rigetta (no) Diag se Diag accetta Diag (yes)



Descrizione di D:

Diag accetta M se M rigetta M

Diag rigetta M se M accetta M

Al posto di M sostituiamo Diag

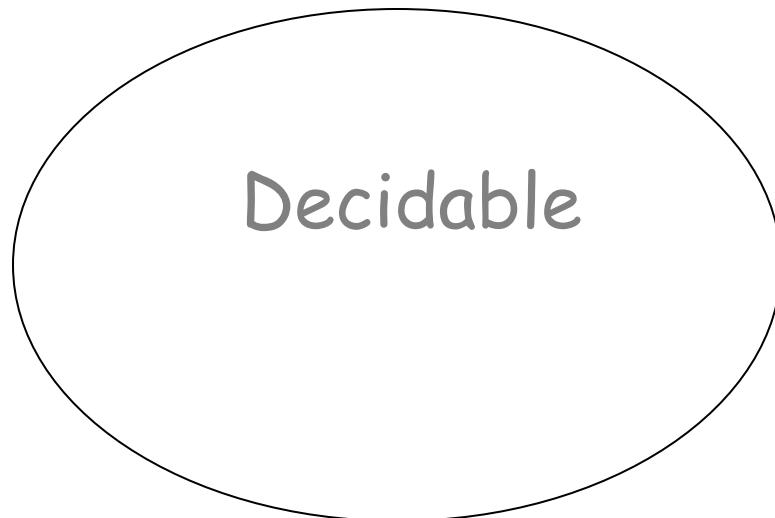
Cosa accade?:

Diag accetta Diag se Diag rigetta Diag (!!)

Diag rigetta Diag se Diag accetta Diag (!!)

Abbiamo mostrato:

A_{TM}



Definizione di Turing accettabile

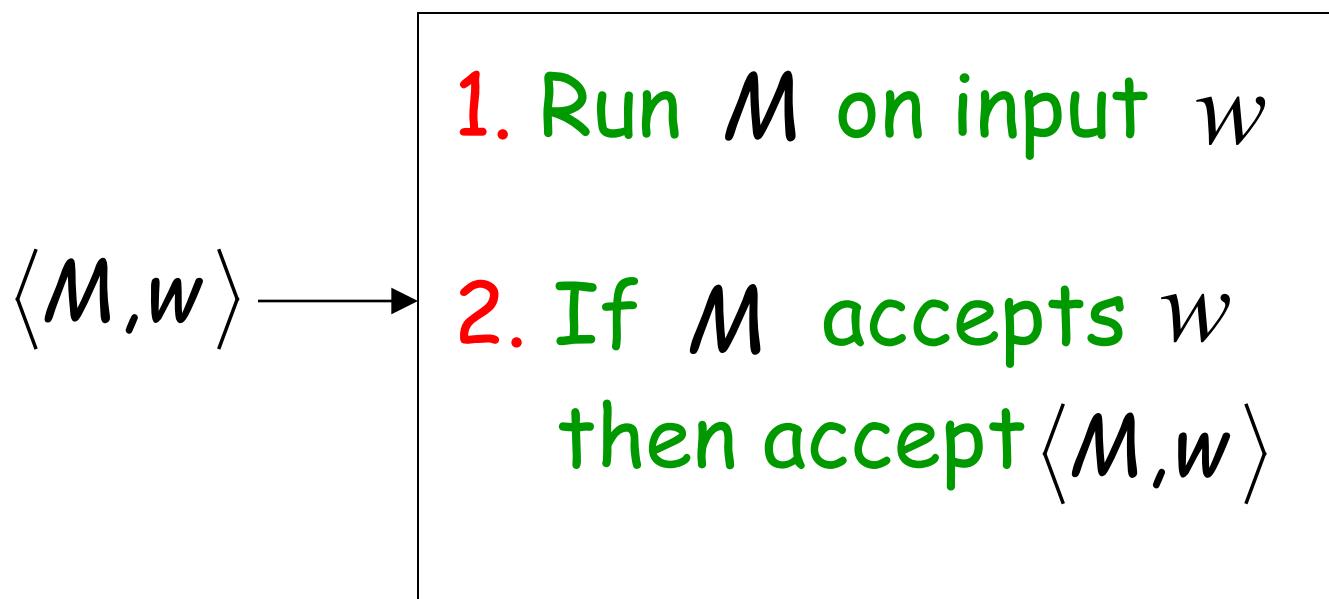
M accetta D se $M(D)$
raggiunge uno stato finale.

Nulla è stato detto su cosa
accade quando M rifiuta D

$A_{TM} = \{\langle M, w \rangle : M \text{ è una Turing machine che accetta la stringa } w\}$

A_{TM} è Turing-Acceptable (semidecidibile)

Turing machine che accetta A_{TM} :



Halting Problem

Input: • Turing Machine M
• String w

domanda: M si ferma nel processo
di calcolo con stringa di input w ?

linguaggio corrispondente:

$HALT_{TM} = \{\langle M, w \rangle : M \text{ è una Turing machine che}$
 $\text{si ferma sull'input } w\}$

$A_{TM} = \{\langle M, w \rangle : M \text{ è una Turing machine che accetta la stringa } w\}$

$HALT_{TM} = \{\langle M, w \rangle : M \text{ è una Turing machine che si ferma sull'input } w\}$

Teorema: HALT_{TM} è indecidibile
(The halting problem non è risolvibile)

dim:

idea di base:

Supponiamo che HALT_{TM} è decidibile;

Proveremo che:

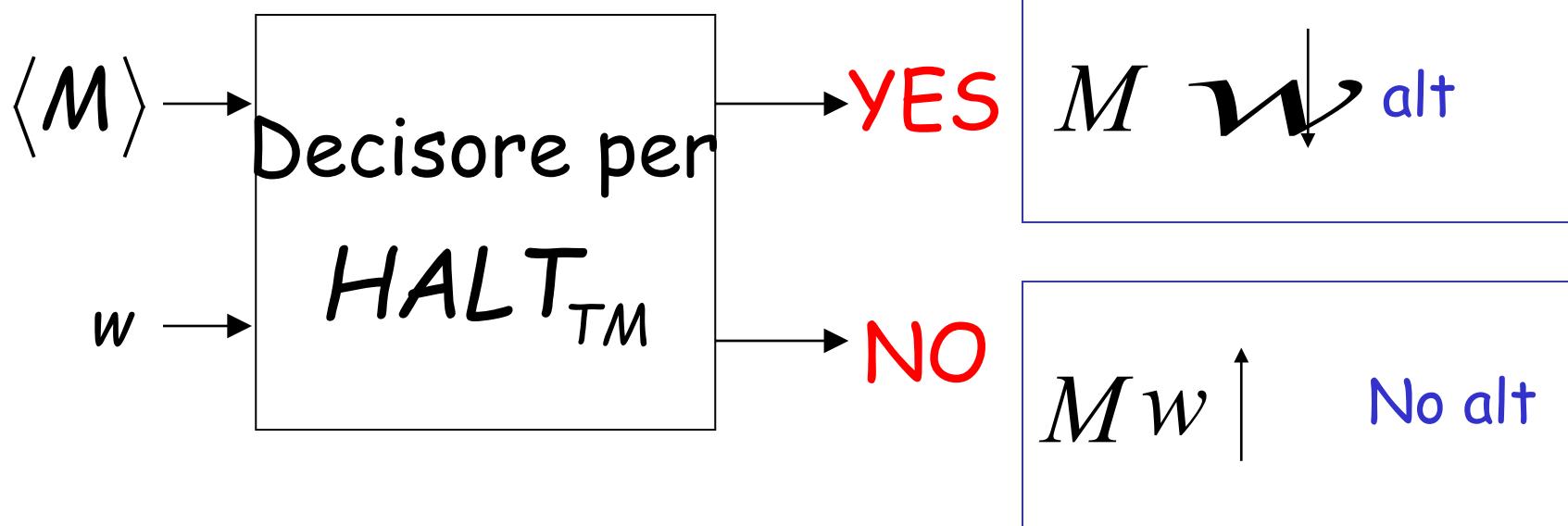
ogni linguaggio Turing-Acceptable
è decidibile

contradizione!

Supponiamo che HALT_{TM} è decidibile

Input
string

$\langle M, w \rangle$



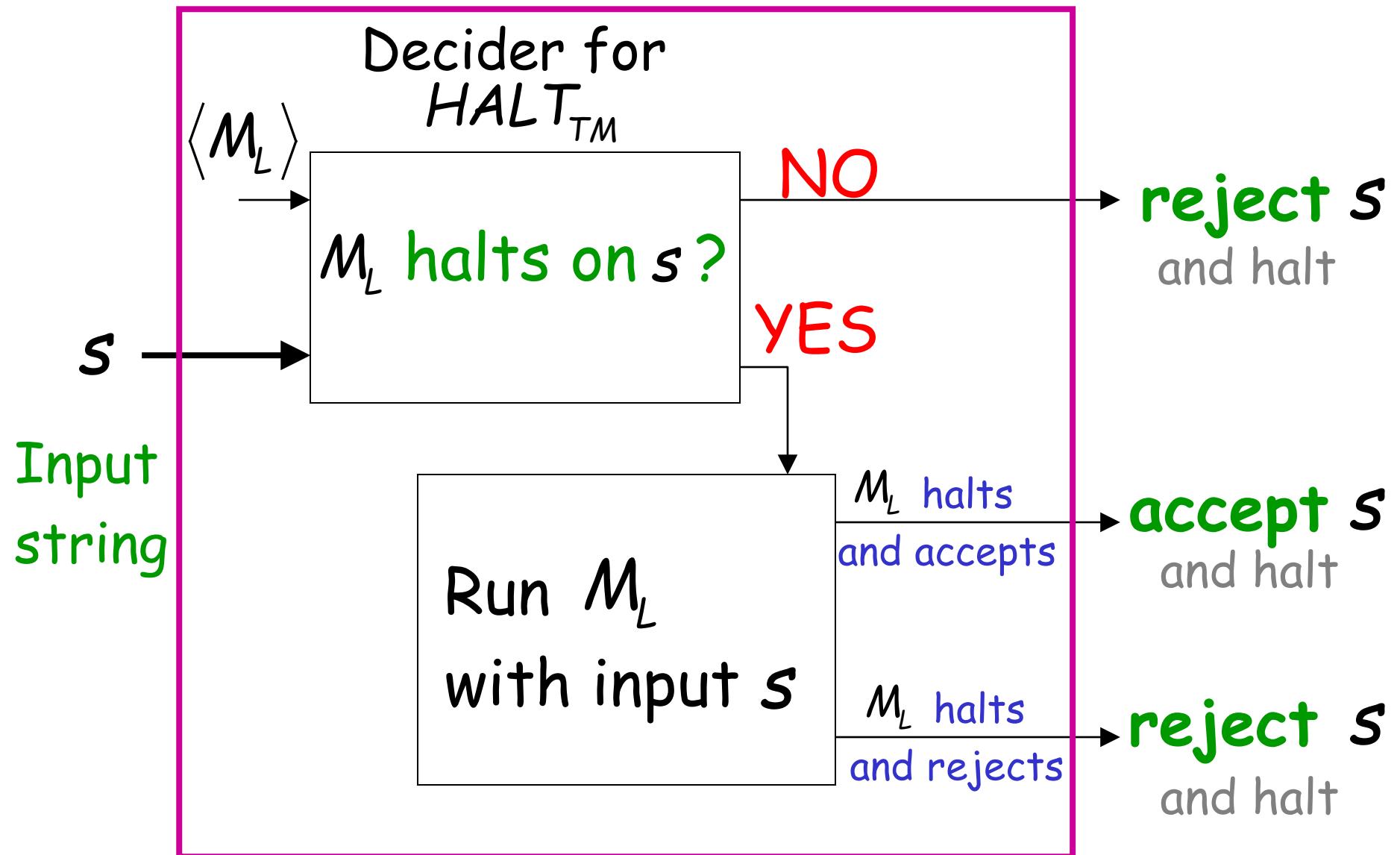
sia L un linguaggio Turing-Accettabile
Turing-Semidecidibile

sia M_L la Turing Machine che accetta L

Proviamo che L è decidibile:

Costruiamo un decisore per L

Decider per L



Quindi L è decidibile

poichè L è stato scelto arbitrariamente, ogni linguaggio (Turing-) semidecidibile è decidibile

Ma vi è un linguaggio Turing-Accettabile (semidecidibile) che è indecidibile (Teor, visto in precedenza)

Contradizione!!!!

END OF PROOF

Uno sguardo sulla diagonalizzazione

Un'altra dimostrazione

Teorema: HALT_{TM} è indecidibile

(The halting problem non è decidibile)

dim:

Idea di base:

Per assurdo: assumiamo che
l'halting problem decidibile;

Cercheremo di ottenere una contraddizione
via diagonalizzazione

Supponi che HALT_{TM} è decidibile

Input
string

$\langle M, w \rangle$

$\langle M \rangle$ →

w →

Decisore
per HALT_{TM}

H

YES

NO

$Mw \downarrow$

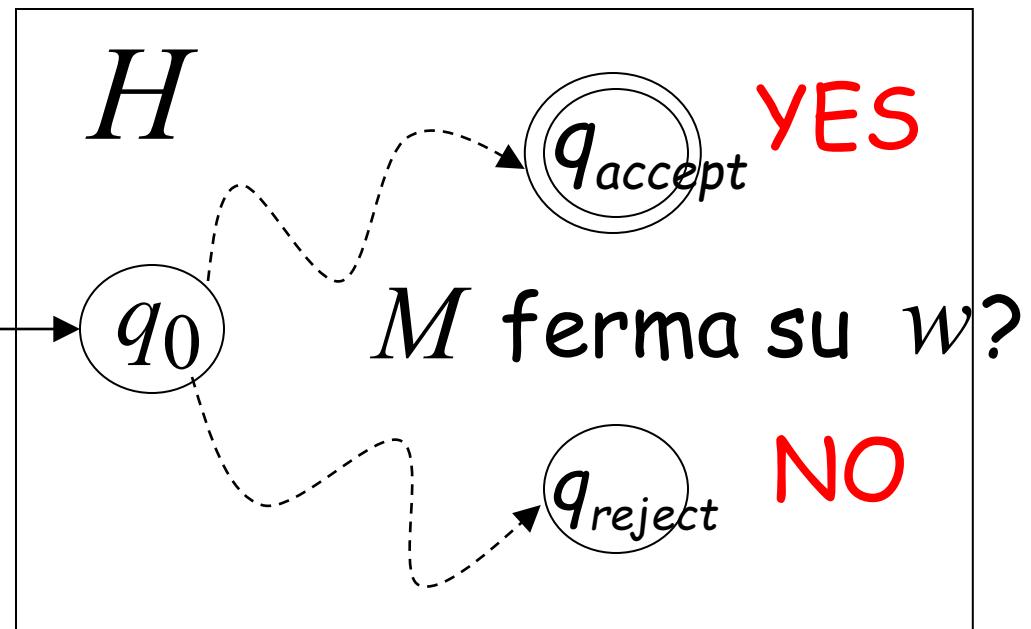
$Mw \uparrow$

Guardiamo
dentro H

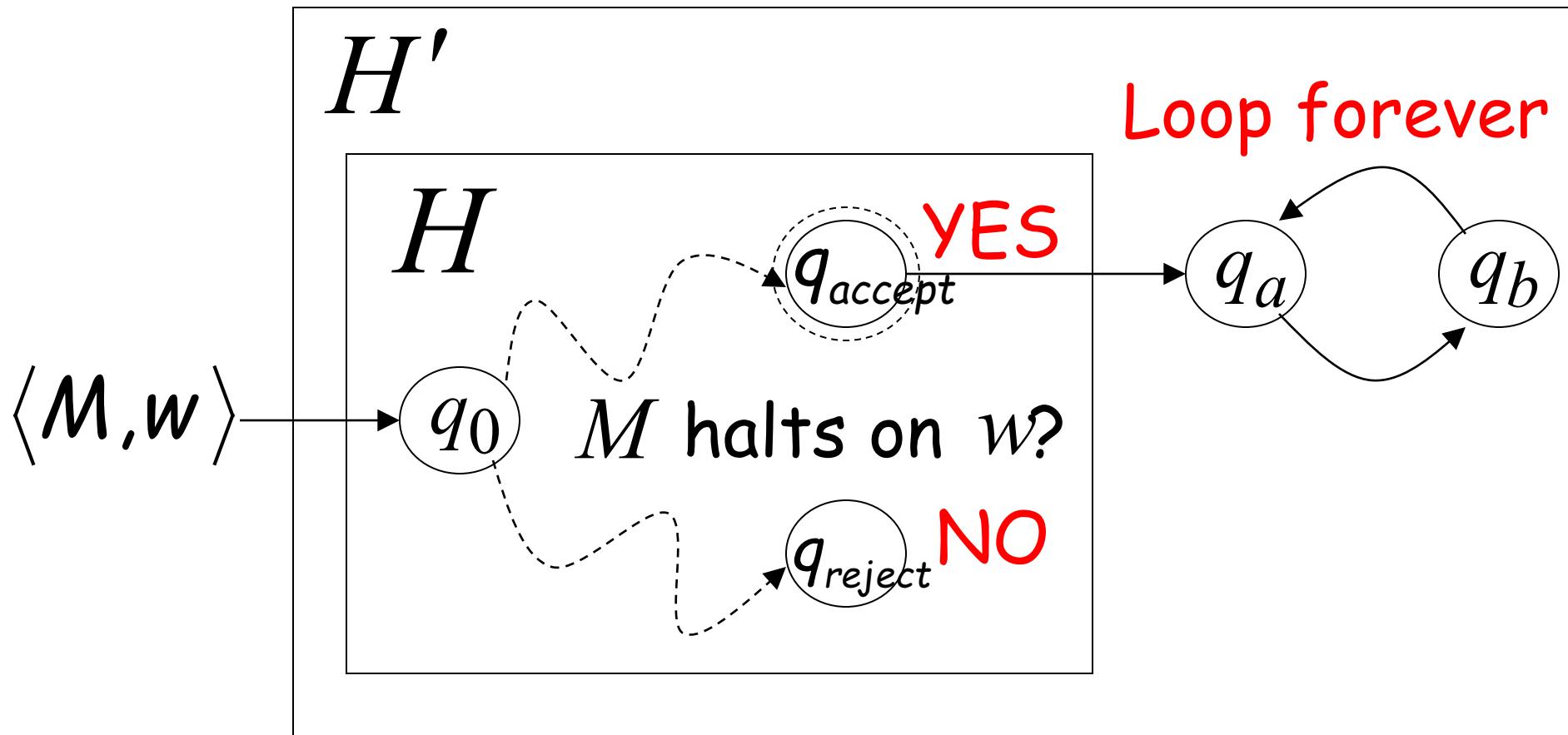
Decider per HALT_{TM}

Input string:

$\langle M, w \rangle$

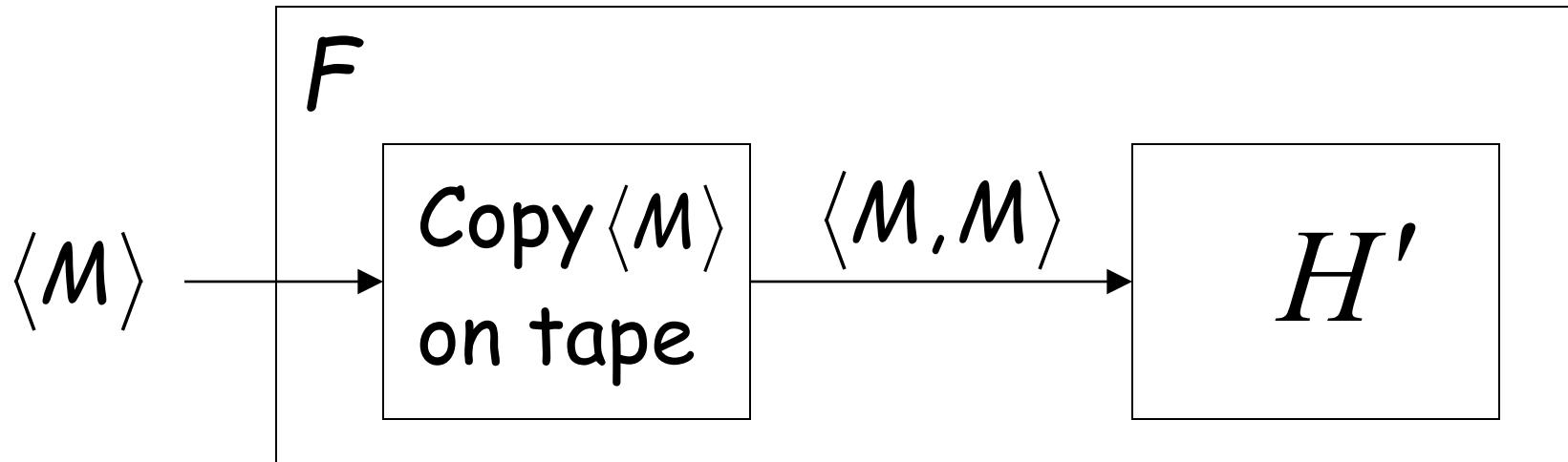


Costruiamo la macchina H' :



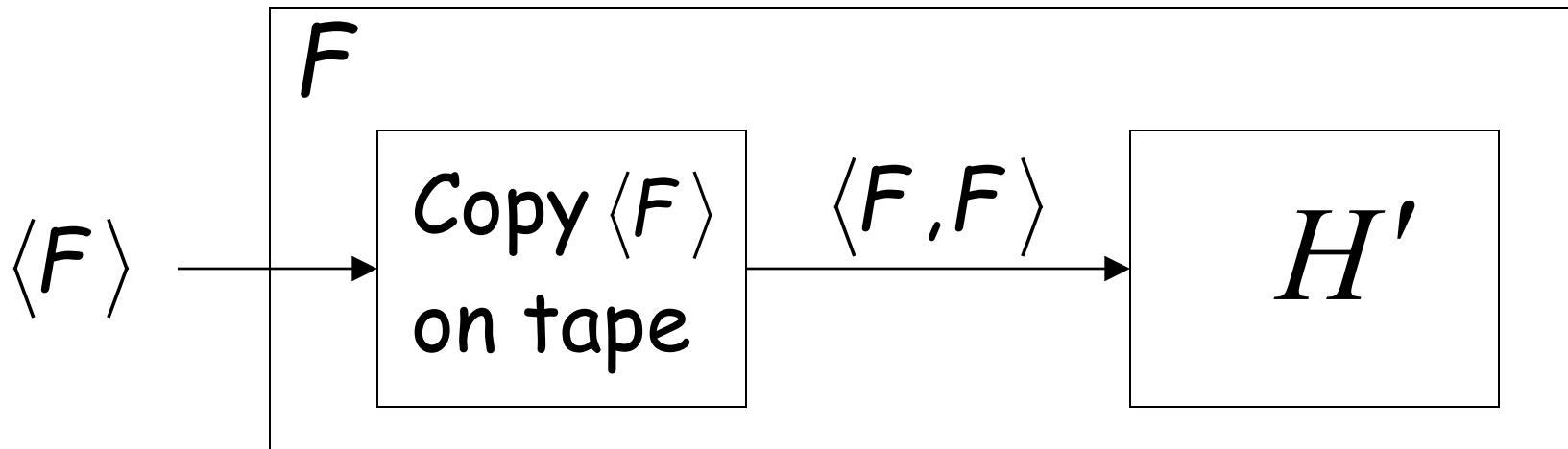
If M halts on input w Then Loop Forever
Else Halt

Costruiamo la macchina F :



If M halts on input $\langle M \rangle$
Then loop forever
Else halt

calcola F con input se stesso



If F halts on input $\langle F \rangle$

Then F loops forever on input $\langle F \rangle$

Else F halts on input $\langle F \rangle$

contradizione!!!

END OF PROOF

Idem precedente

Teorema 5.8 [Indecidibilità del problema della terminazione⁸] *Siano dati un alfabeto Γ ed una codificazione che associa ad ogni macchina di Turing $\mathcal{M} = \langle \Gamma, b, Q, \delta, q_0, F \rangle$ una sua codifica $c_{\mathcal{M}} \in \Gamma^*$. La funzione*

$$h(c_{\mathcal{M}}, x) = \begin{cases} 1 & \text{se } \mathcal{M} \text{ termina su input } x \\ 0 & \text{se } \mathcal{M} \text{ non termina su input } x \end{cases}$$

non è T-calcolabile.

$h(\text{codice}_M, x)$ =1 se M con input x termina
=0 se M con input x non termina

Supponiamo che il predicato sia calcolabile, esista cioè una macchina di Turing h che calcola la funzione h . Costruiamo la macchina h' che calcola il predicato

$h'(\text{codice}_M)$ =1 se M con input Codice_M termina
=0 se M con input Codice_M non termina

h' è la composizione di due macchine:

la prima con input codice_M fornisce codice_M b codice_M,
la seconda è la macchina h che calcola il predicato della terminazione.

In altre parole h' è la macchina che verifica se una MT termina quando le viene fornito in input il proprio codice.

Possiamo ora costruire una nuova macchina h'' che prende in input codice_M e calcola la funzione:

$$\begin{aligned} h''(\text{codice}_M) &= 0 \text{ se } h'(\text{codice}_M) = 0 \\ &= \text{indefinito altrimenti} \end{aligned}$$

$$\begin{aligned}
 h'(\text{codice}_M) &= 1 && \text{se } M(\text{codice}_M) \text{ termina} \\
 &= 0 && \text{se } M(\text{codice}_M) \text{ non termina}
 \end{aligned}$$

▼

$$\begin{aligned}
 h''(\text{codice}_M) &= 0 \text{ se } h'(\text{codice}_M) = 0 \text{ (se } M(\text{codice}_M) \text{ non termina)} \\
 &= \text{indefinito} \quad \text{altrimenti (se } M(\text{codice}_M) \text{ termina)}
 \end{aligned}$$

)

*termina con 0 se h' si è fermata con 0 e
si mette a ciclare, se h' si è fermata con 1*

calcoliamo $h''(\text{codice}_{h''})$:

$$\begin{aligned} h''(\text{codice}_{h''}) &= \text{indefinito} && \text{se } h''(\text{codice}_{h''}) \text{ è definita} \\ &= 0 && \text{se } h''(\text{codice}_{h''}) \text{ è indefinita} \end{aligned}$$

In ogni caso abbiamo una contraddizione. Quindi non può esistere la macchina H.

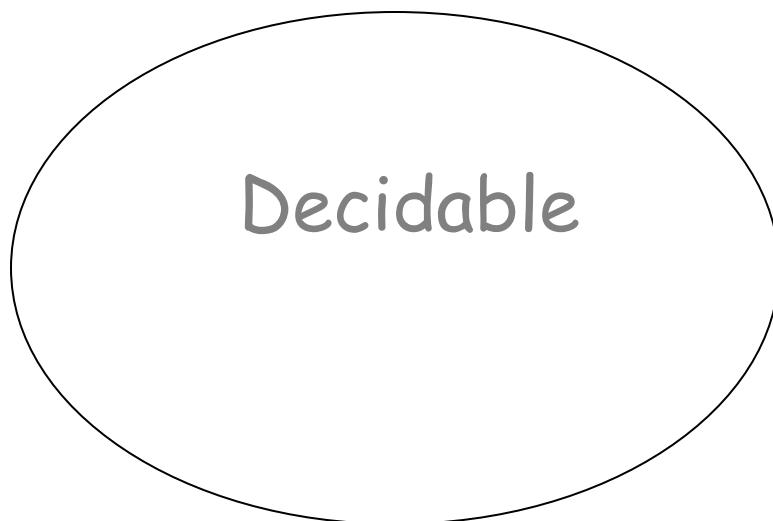
$$\begin{aligned} h''(D_M) &= 0 && \text{se } h'(D_M) = 0 \\ &= \text{indefinito} && \text{altrimenti} \end{aligned}$$

Fine
idem

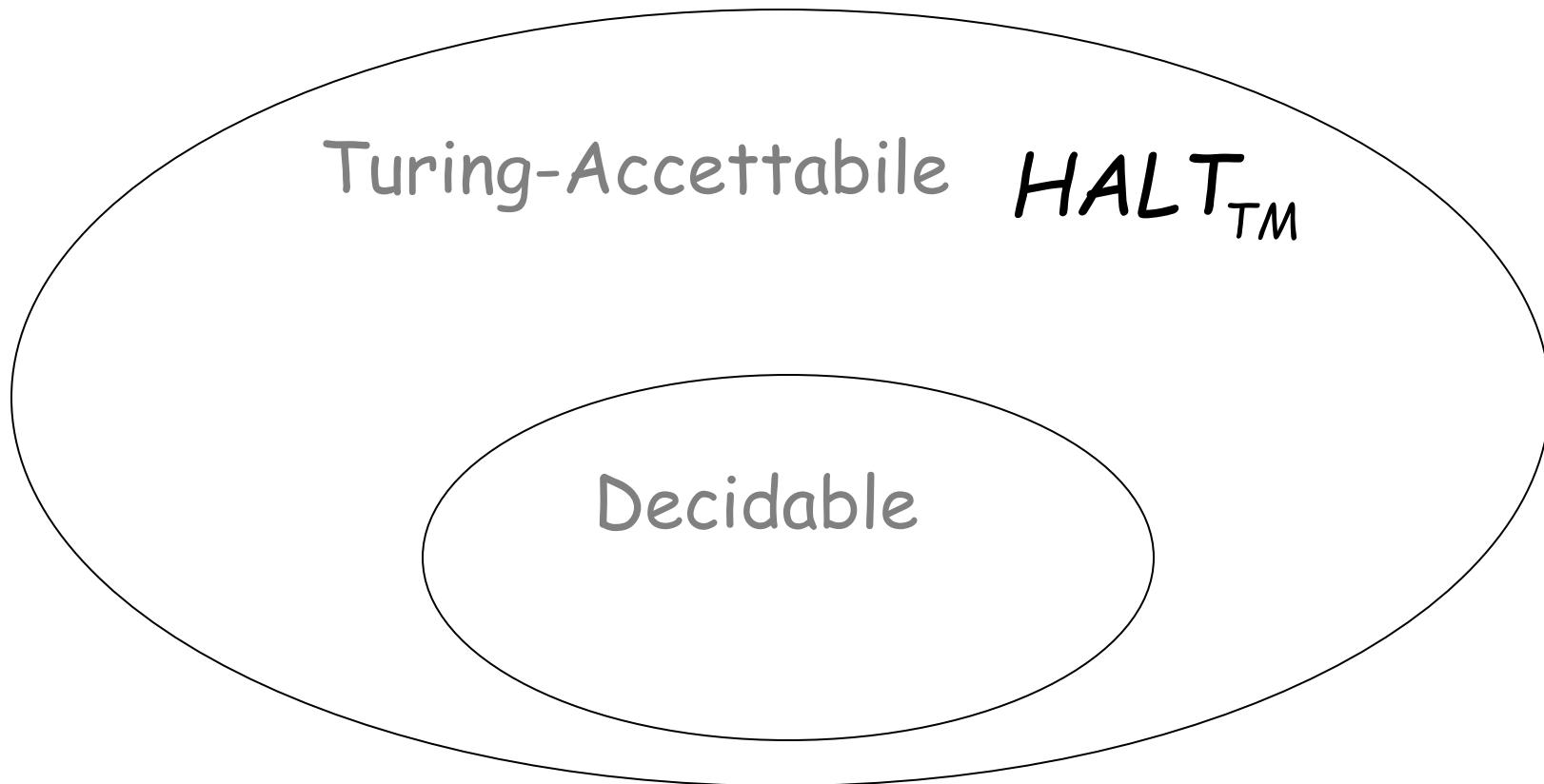
$$\begin{aligned} h'(D_M) &= 1 && \text{se } M(D_M) \text{ termina} \\ &= 0 && \text{se } M(D_M) \text{ non termina} \end{aligned}$$

Abbiamo mostrato

indecidibile HALT_{TM}

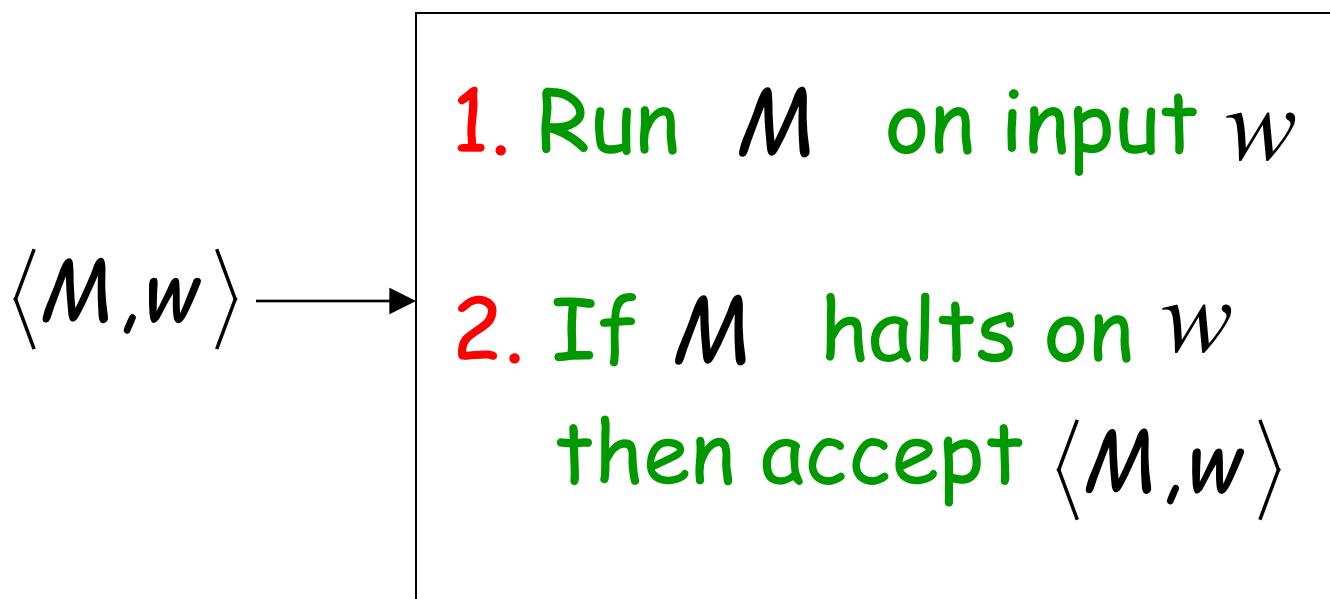


Adesso proviamo che:

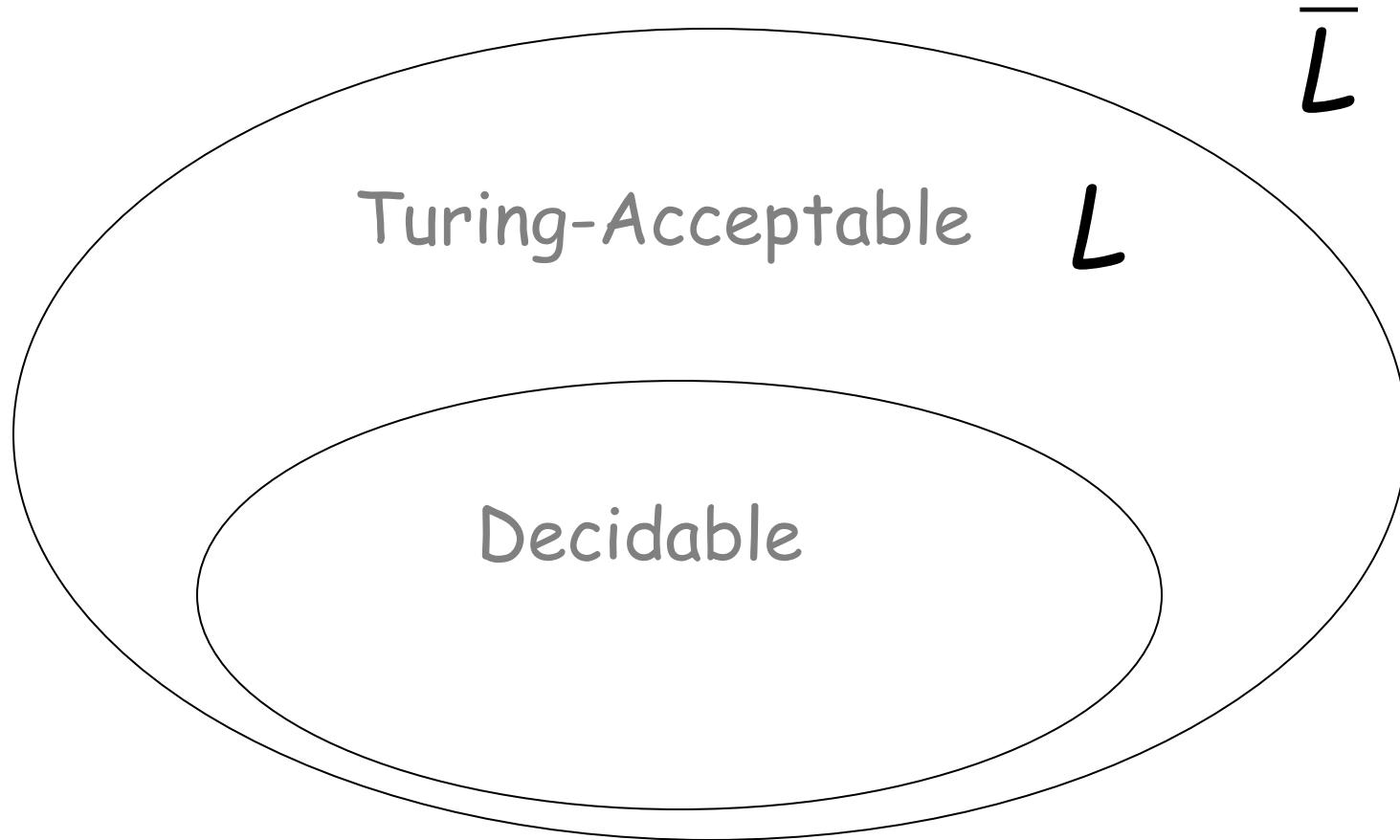


HALT_{TM} è Turing-Acceptable (semidecidibile)

Turing machine che accetta HALT_{TM} :



Abbiamo già mostrato che esistono linguaggi indecidibili:



Tesi di Church Turing

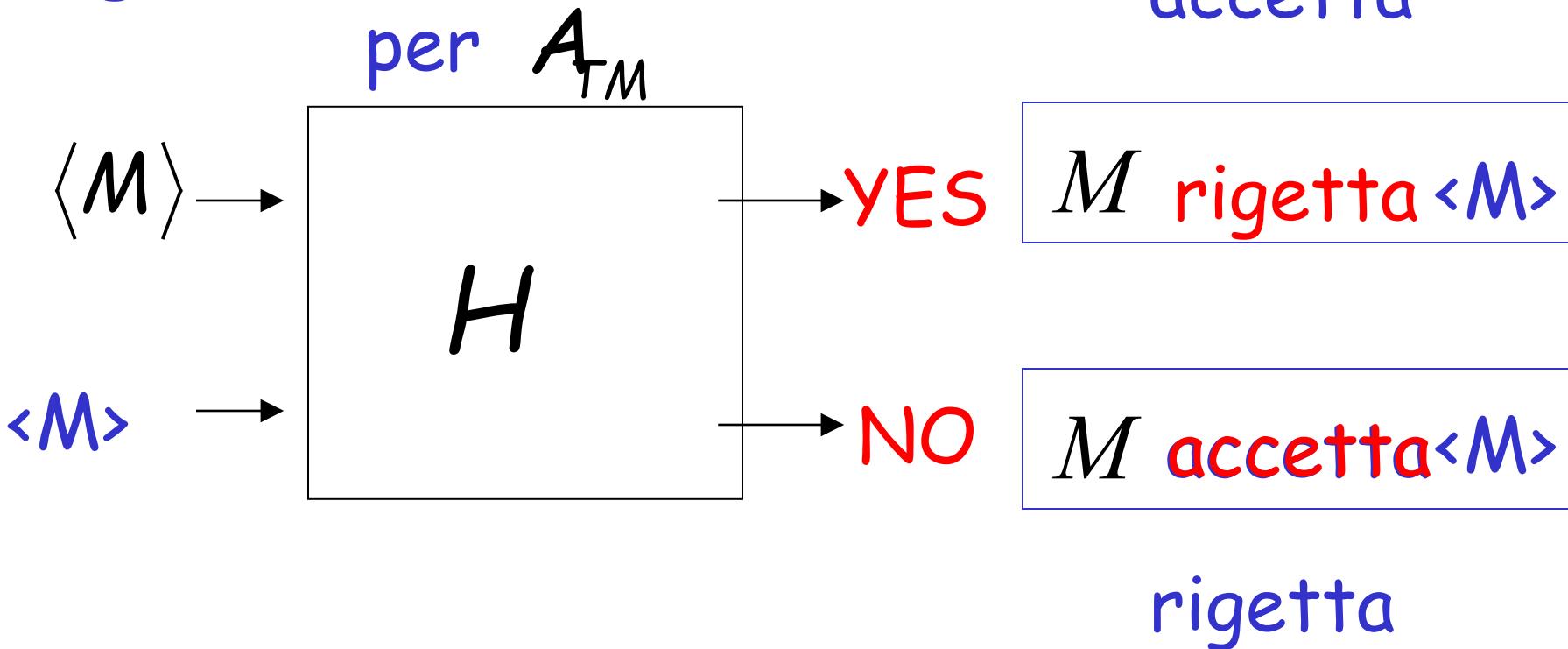
Fine indecidibilità.

$\xrightarrow{A_M}$

Supponiamo $w = \langle M \rangle$ e
calcoliamo $D(\langle M \rangle, \langle M \rangle)$

Input
string

Decisore
per A_{TM}



Teorema: A_{TM} è indecidibile

(The membership problem non è risolvibile)

Proof:

Idea di base:

Assumiamo che A_{TM} è decidibile;

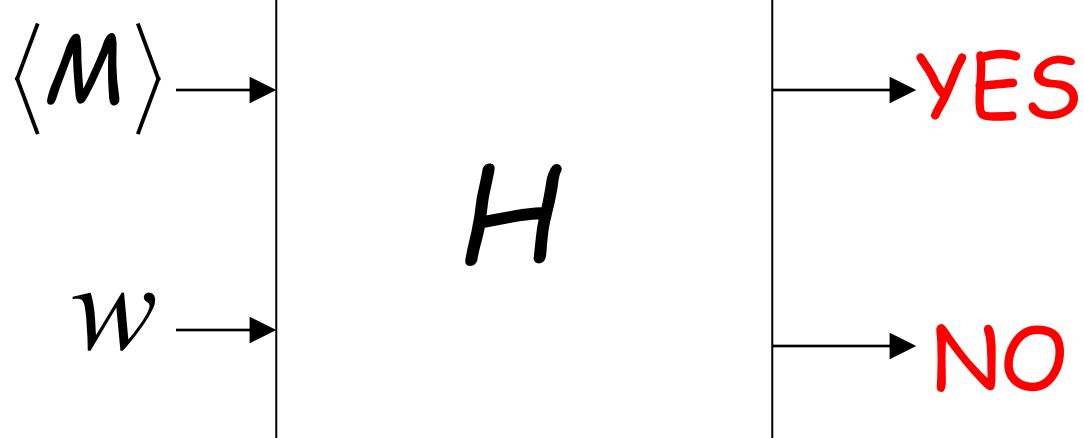
Proveremo che ogni linguaggio
è Turing-Acceptable

assurdo!

Allora avremo un decisore così definito:

Input
string
 $\langle M, w \rangle$

Decisore
per A_{TM}

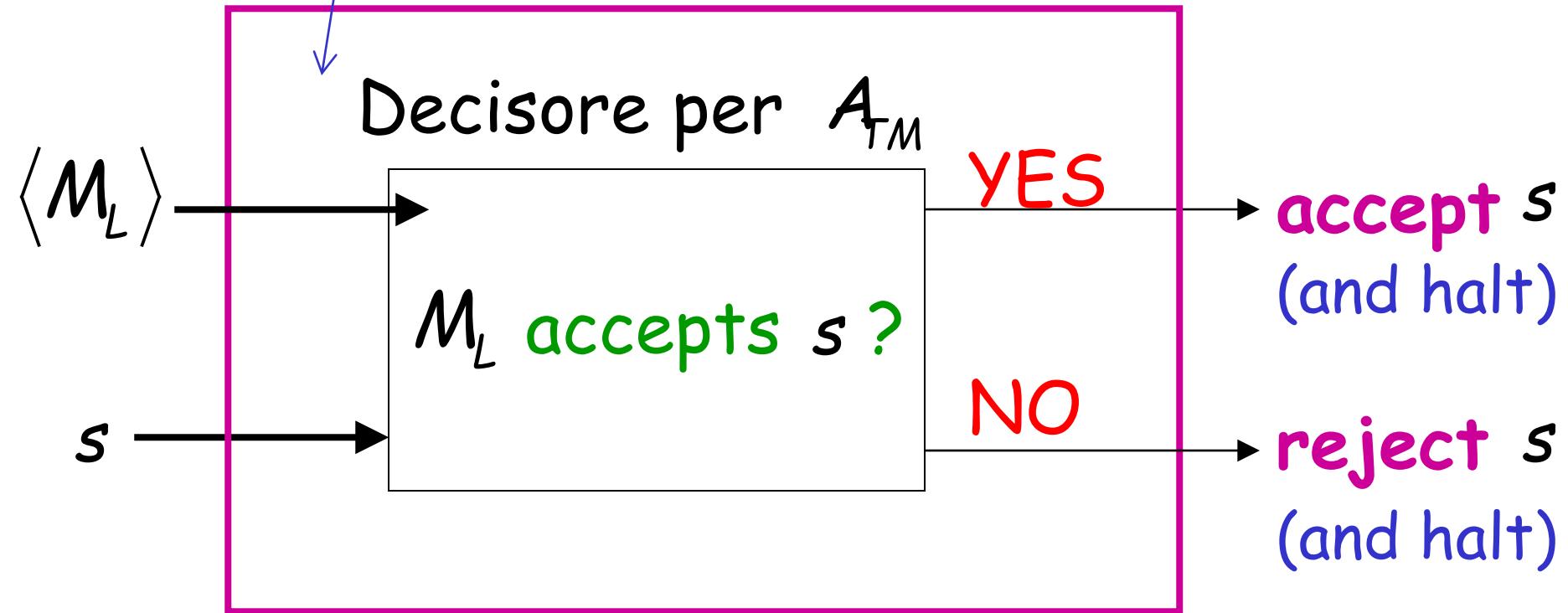


M accetta w

M rigetta w

Descrizione come stringa di M_L

Decisore per L



Sia M_L il decisore per il linguaggio L

Cambiamo, diagonalizziamo, chiamiamo la macchina Diag.

Diag accetta se

A_{TM}

dice yes ovvero se $M(w) = \text{no}$

Diag rigetta se

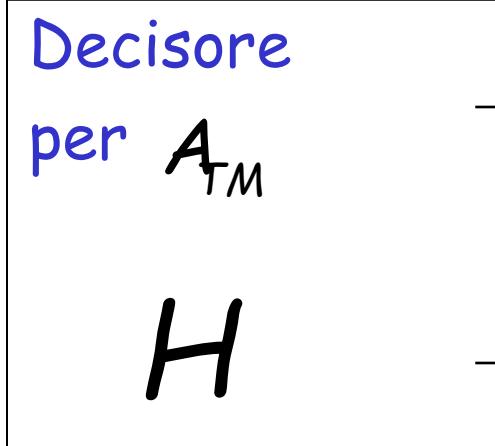
A_{TM}

dice no ovvero se $M(w) = \text{s}$

Diag:

$\langle M, w \rangle$

$\langle M \rangle \rightarrow$



accetta



YES M rigetta w

NO M accetta w

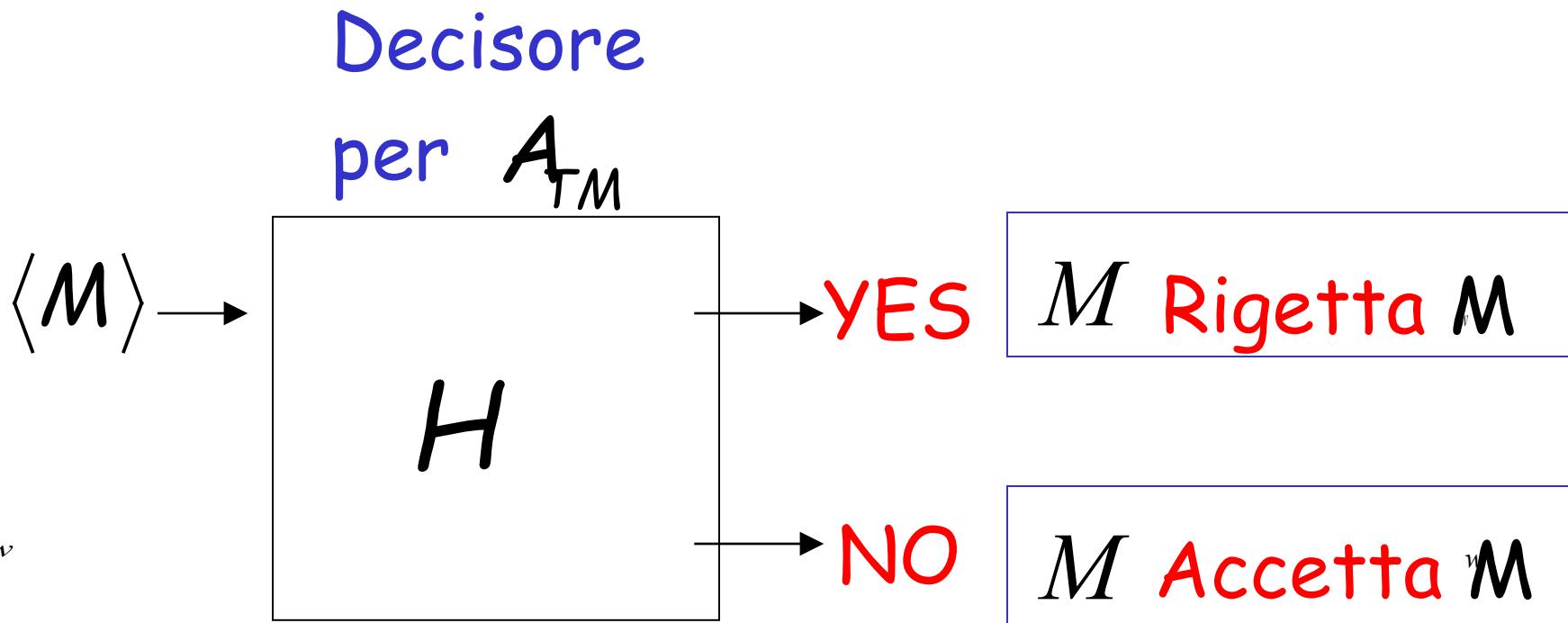
rigetta

Descrizione di Diag:

A_M

Diag accetta M se M rigetta M

Diag rigetta M se M accetta M



Mostriamo che:

