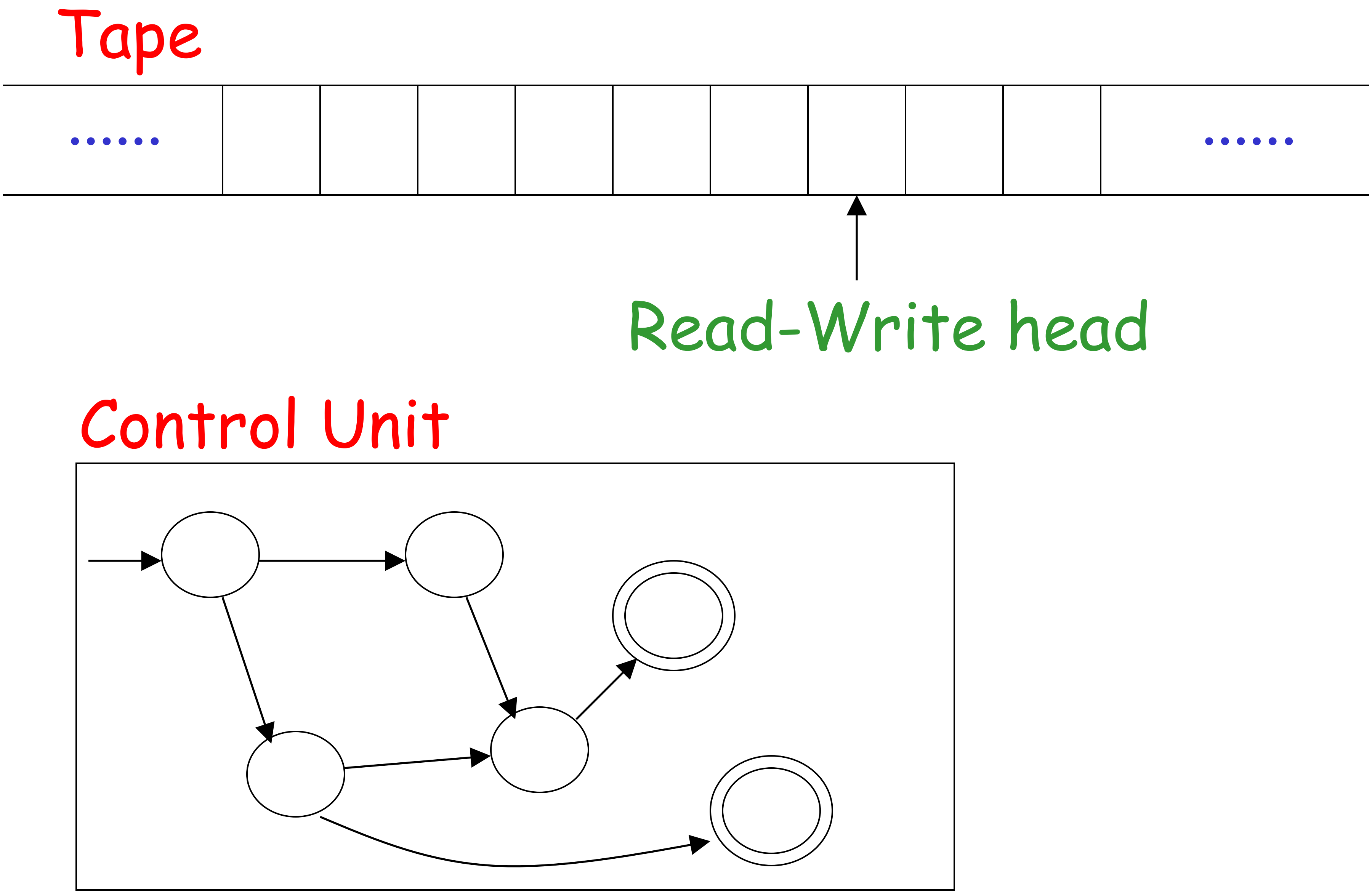
Walton Athletic Club, 1947

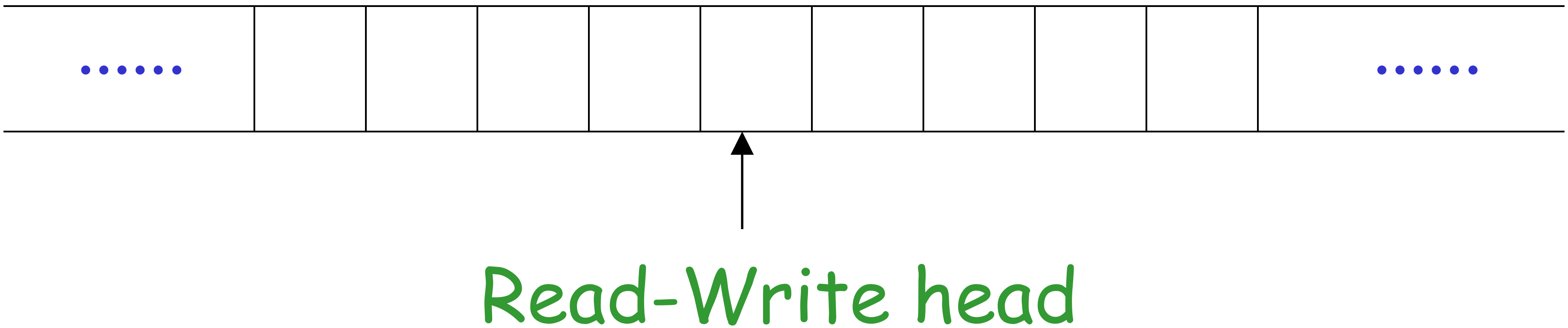
Macchine di Turing

Una macchina di Turing

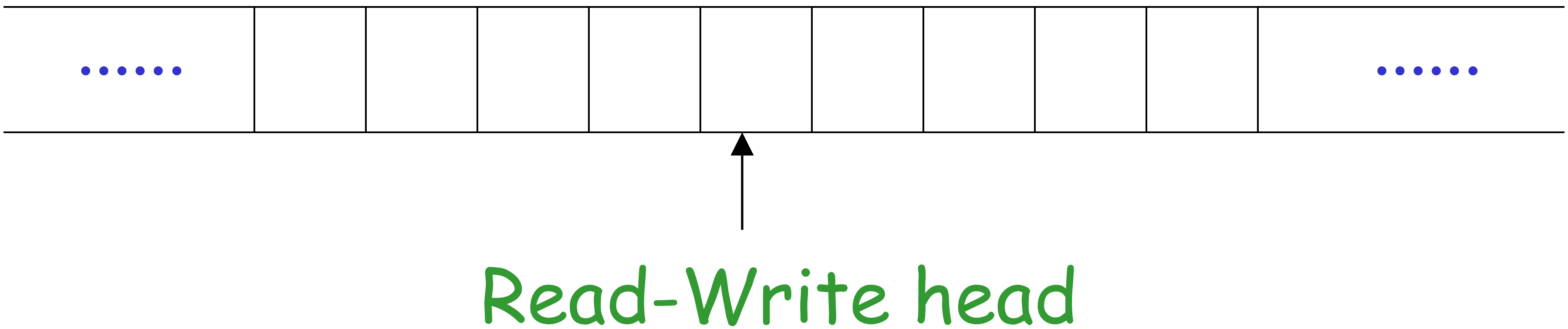


il Tape

No limiti - lunghezza potenzialmente infinita



Le testa si muove Left or Right

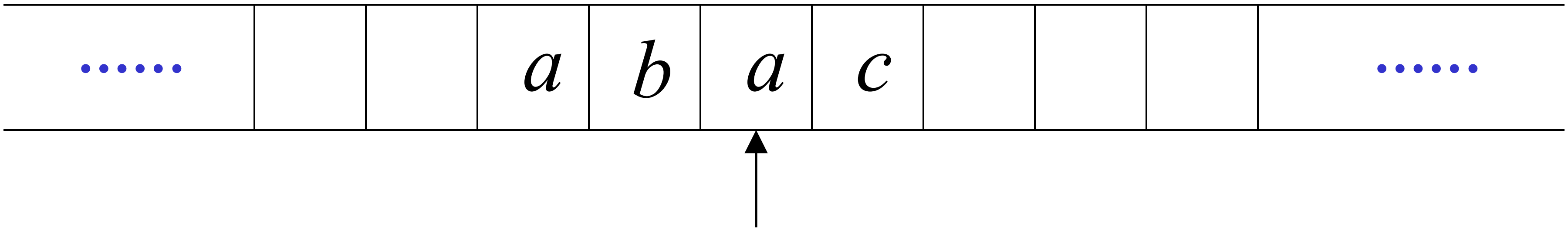


la head ad ogni transizione (time step):

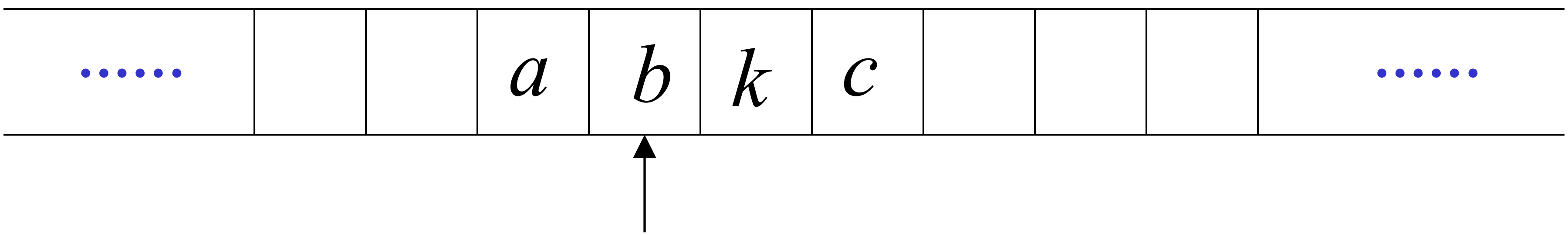
1. legge un simbolo
2. scrive un simbolo
3. si muove Left or Right

esempio:

Time 0

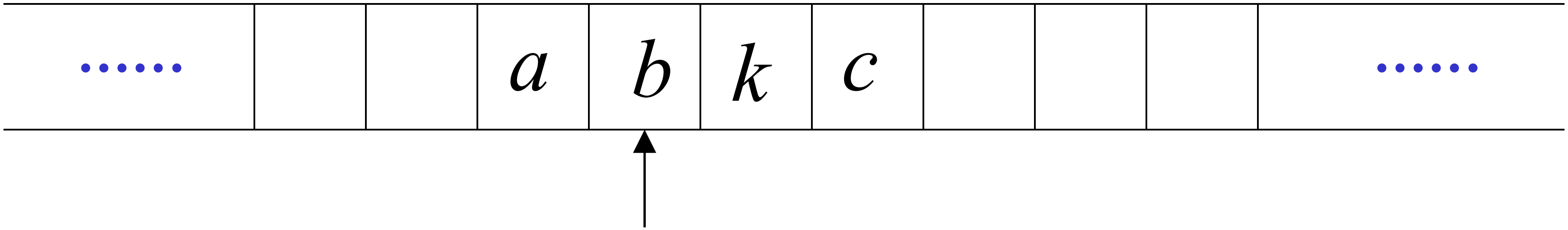


Time 1

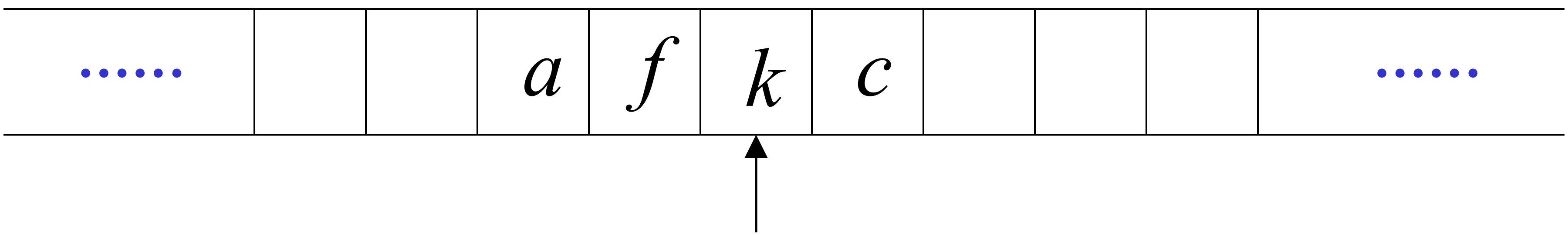


1. Reads a
2. Writes k
3. Moves Left

Time 1

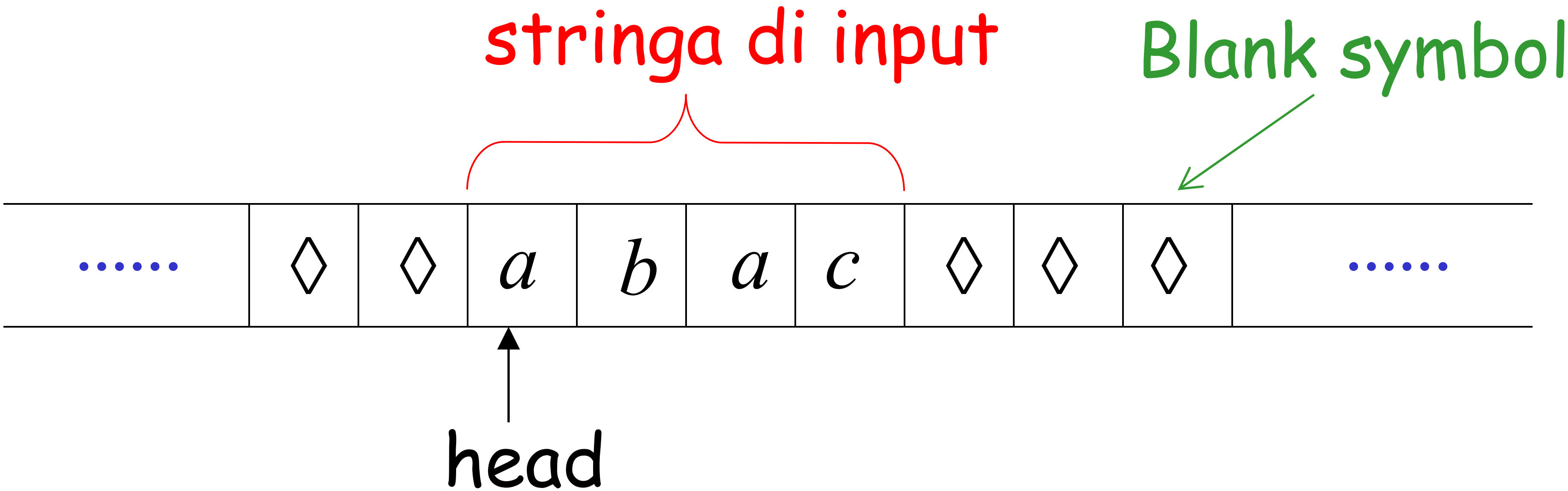


Time 2



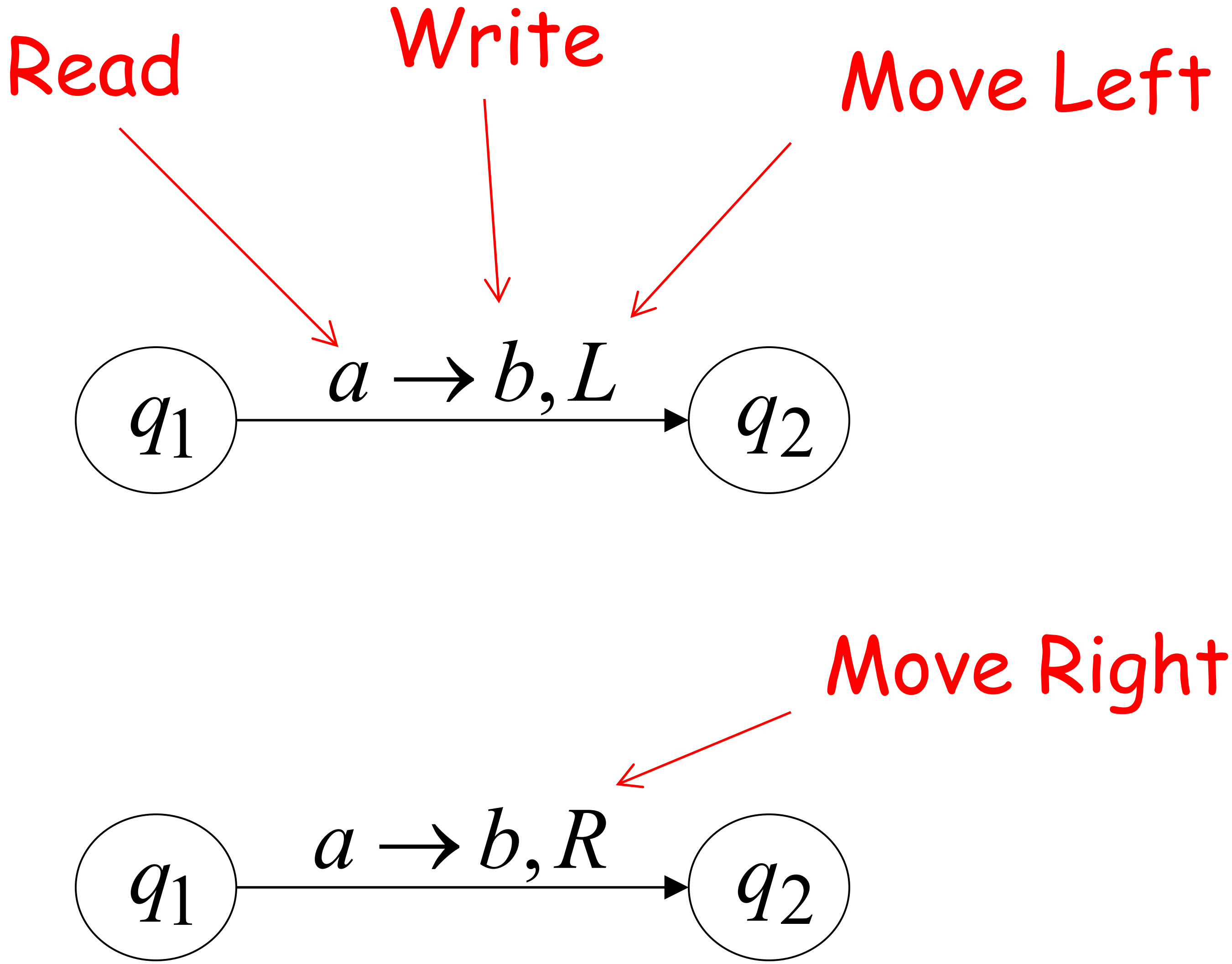
1. Reads b
2. Writes f
3. Moves Right

la String Input



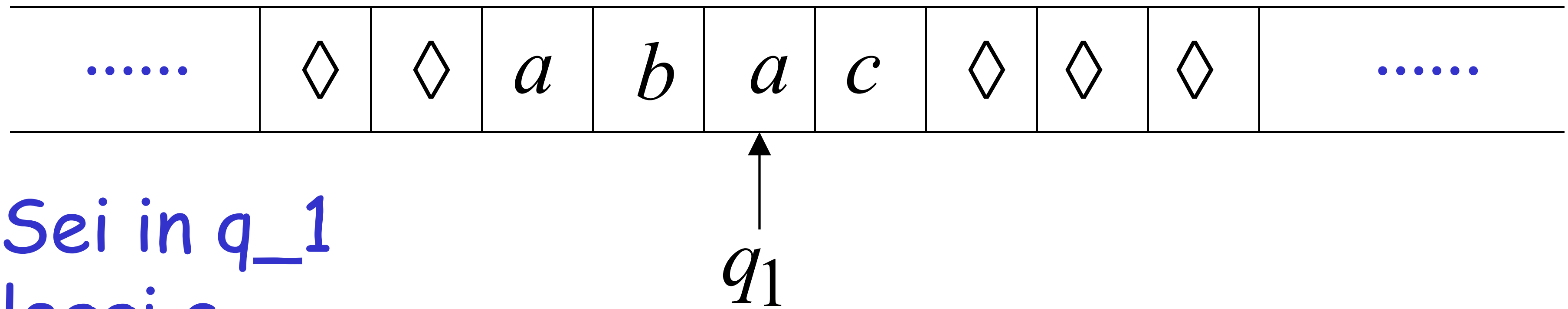
Head parte dalla posizione più a sinistra della stringa di input

Stati & Transizioni



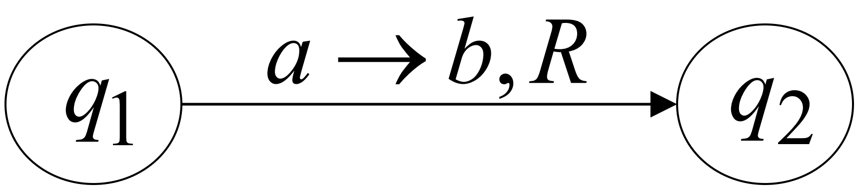
esempio:

Time 1

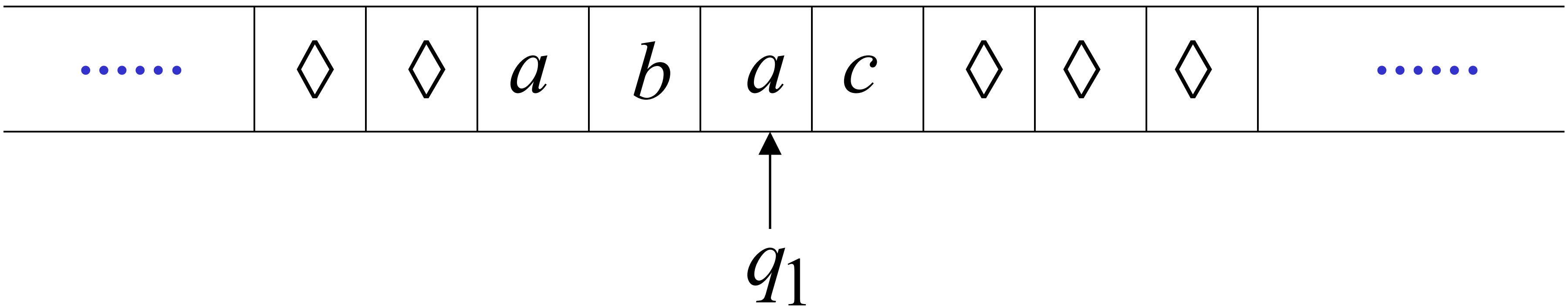


Sei in q_1
leggi a ,
cambia a in b
e vai a right

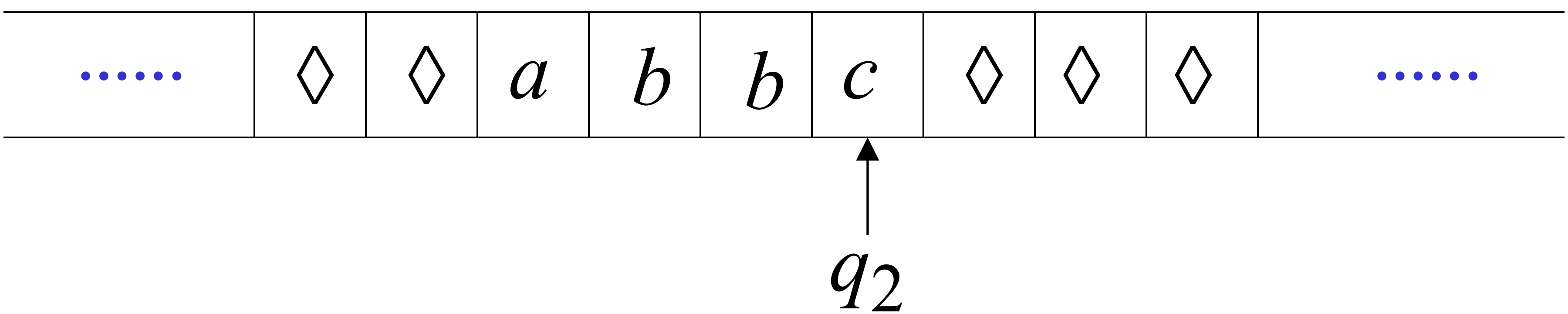
stato corrente

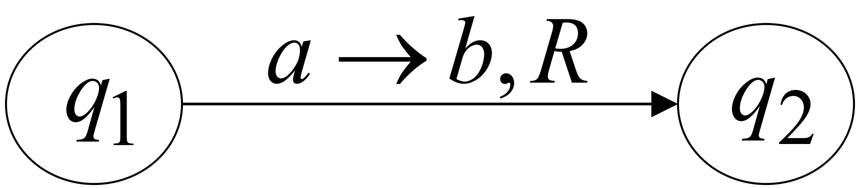


Time 1



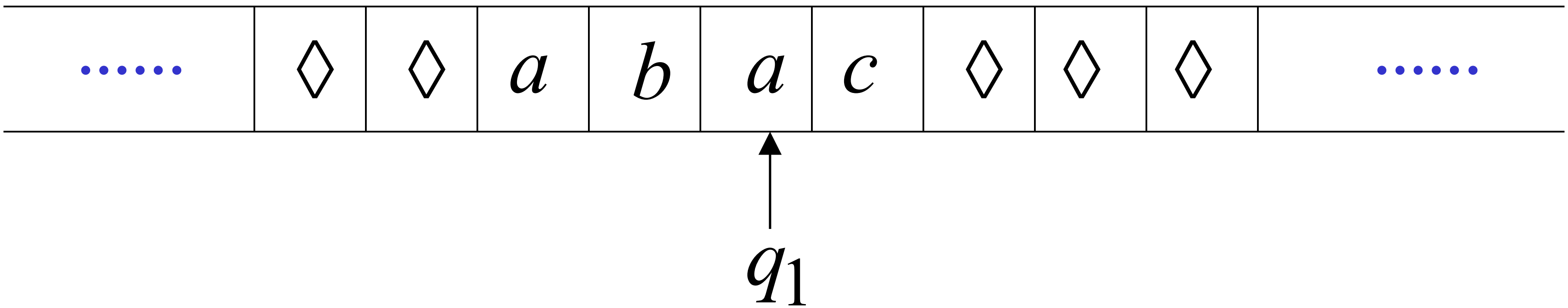
Time 2



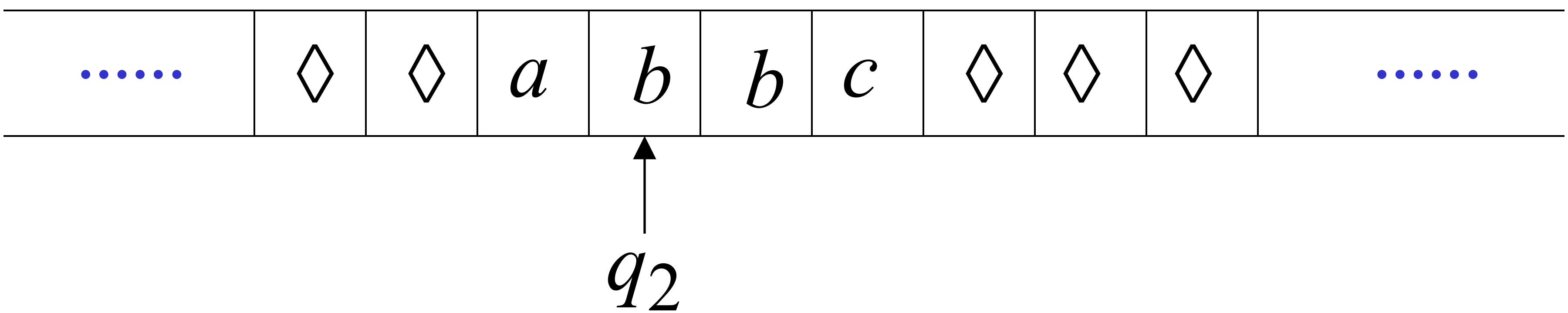


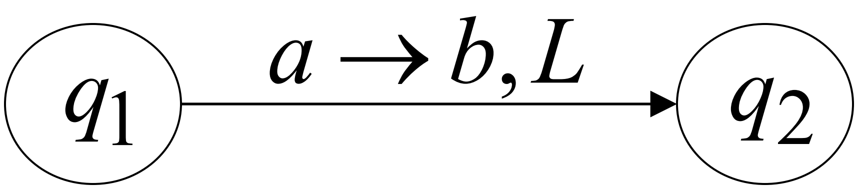
esempio:

Time 1



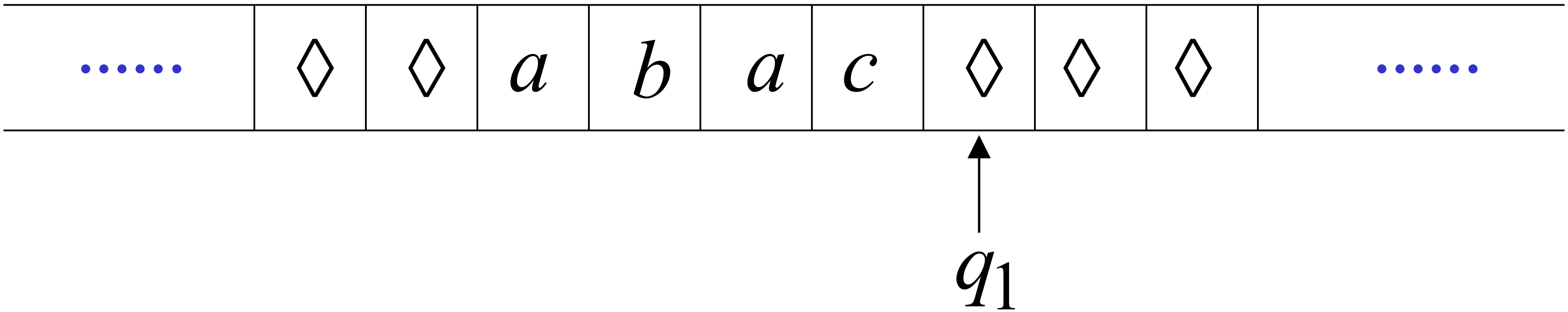
Time 2



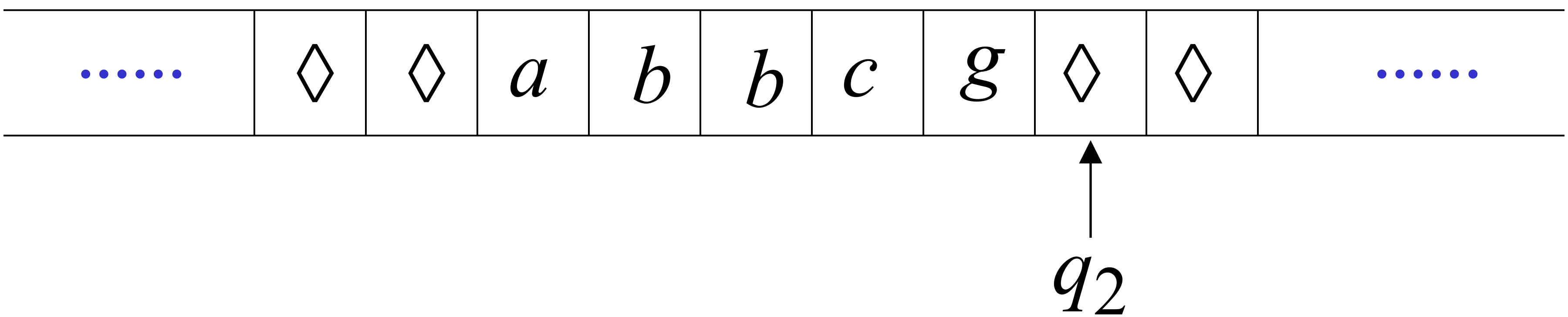


esempio:

Time 1



Time 2

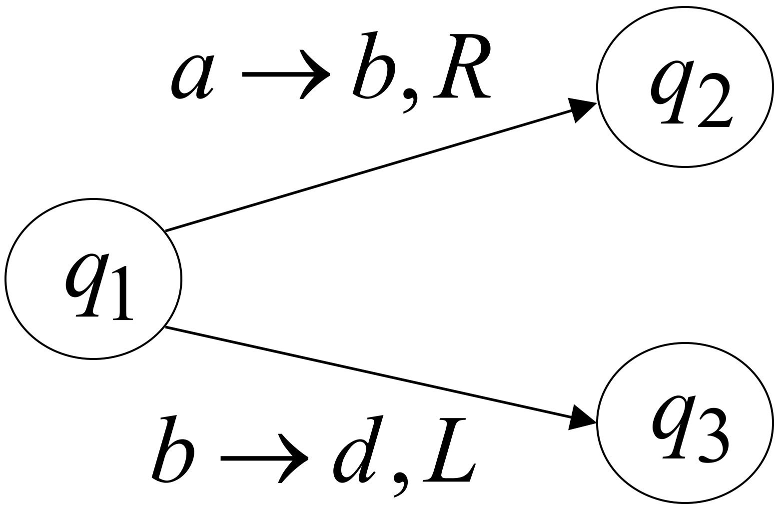


$$q_1 \xrightarrow{\lozenge \to g, R} q_2$$

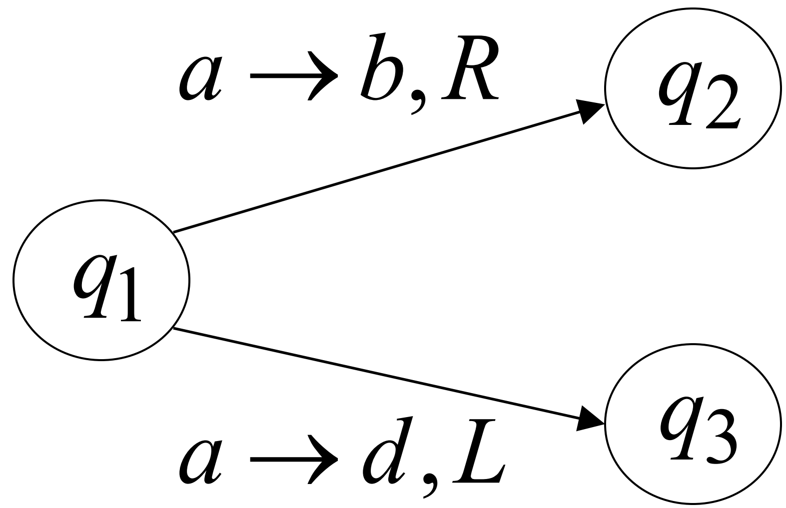
Determinismo

macchine di Turing sono deterministiche

permesso



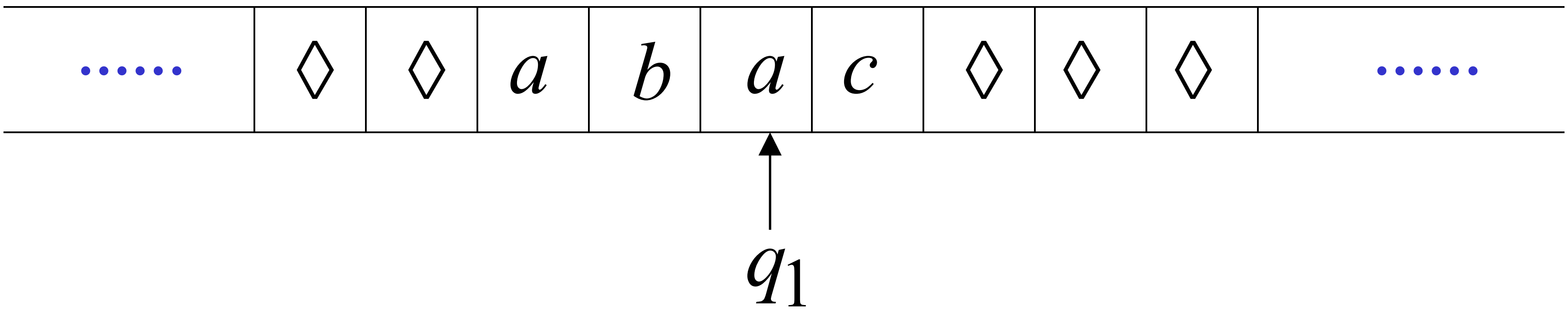
Non permesso

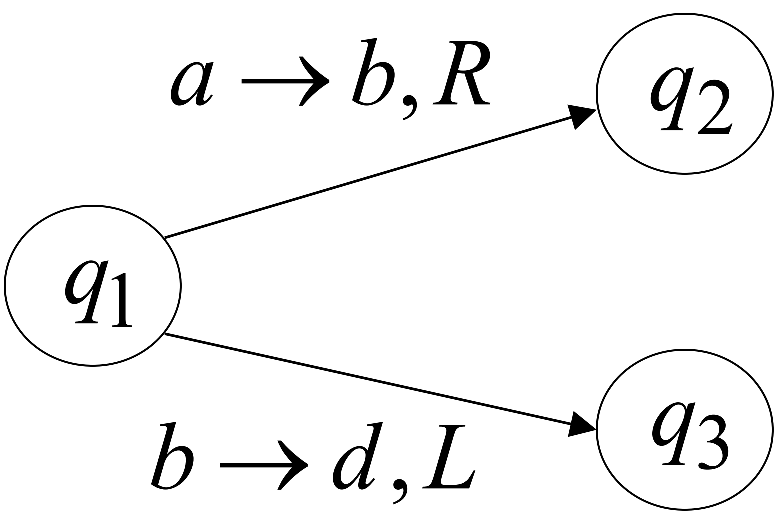


nessuna transizione lambda è permessa

Funzione di Transizione Parziale

esempio:





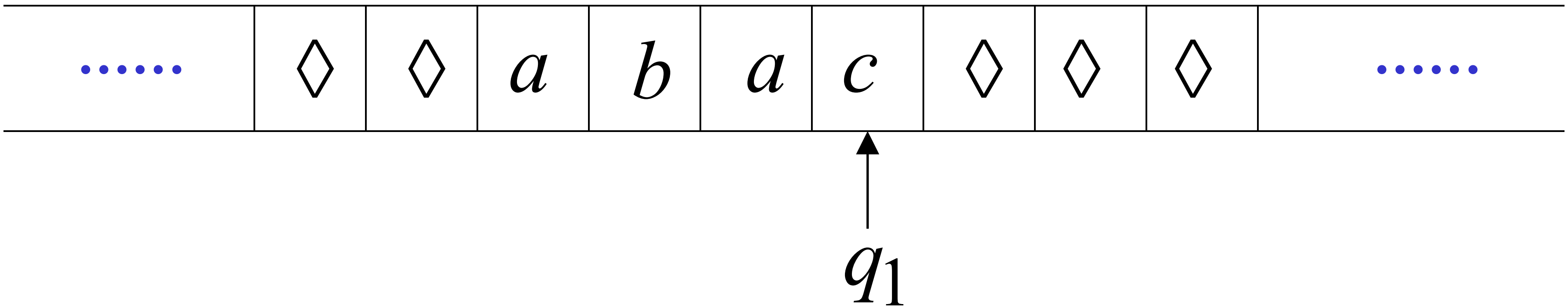
permesso:

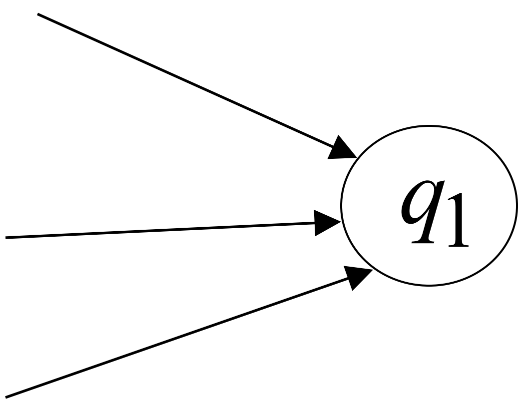
Nessuna transizione per simbolo input c

Halting

La macchina ***si ferma*** nello stato
in cui si trova
se non vi è nessuna transizione
da eseguire

Halt esempio 1:

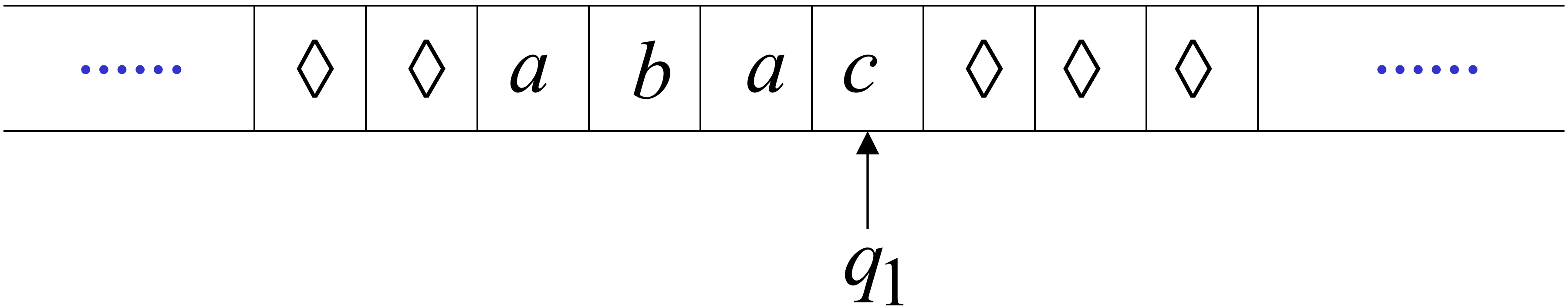


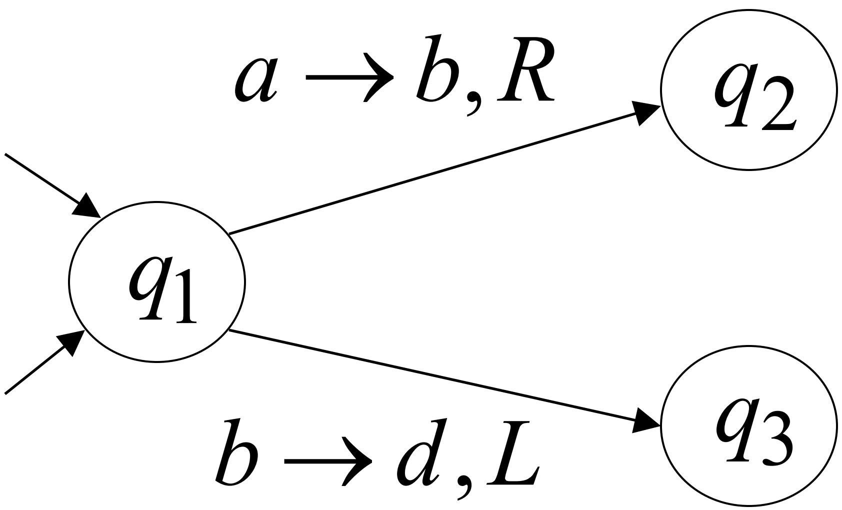


Nessuna transizione da q_1

HALT!!!

Halt esempio 2:

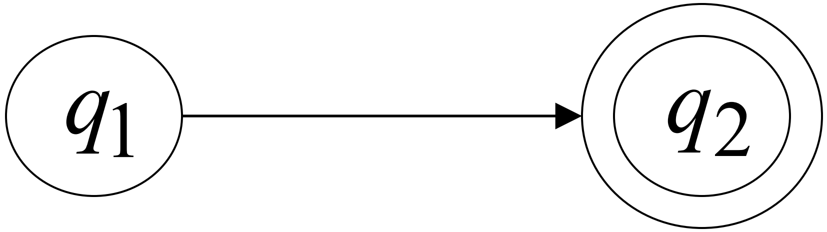




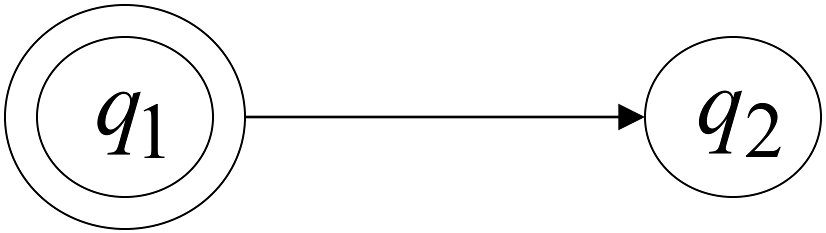
Nessuna transizione possibile da q_1 sul simbolo c

HALT!!!

Stati di accettazione



permesso



Non permesso

- Stati di accettazione non hanno transizioni in uscita
- La macchina si ferma e accetta.

Accettazione

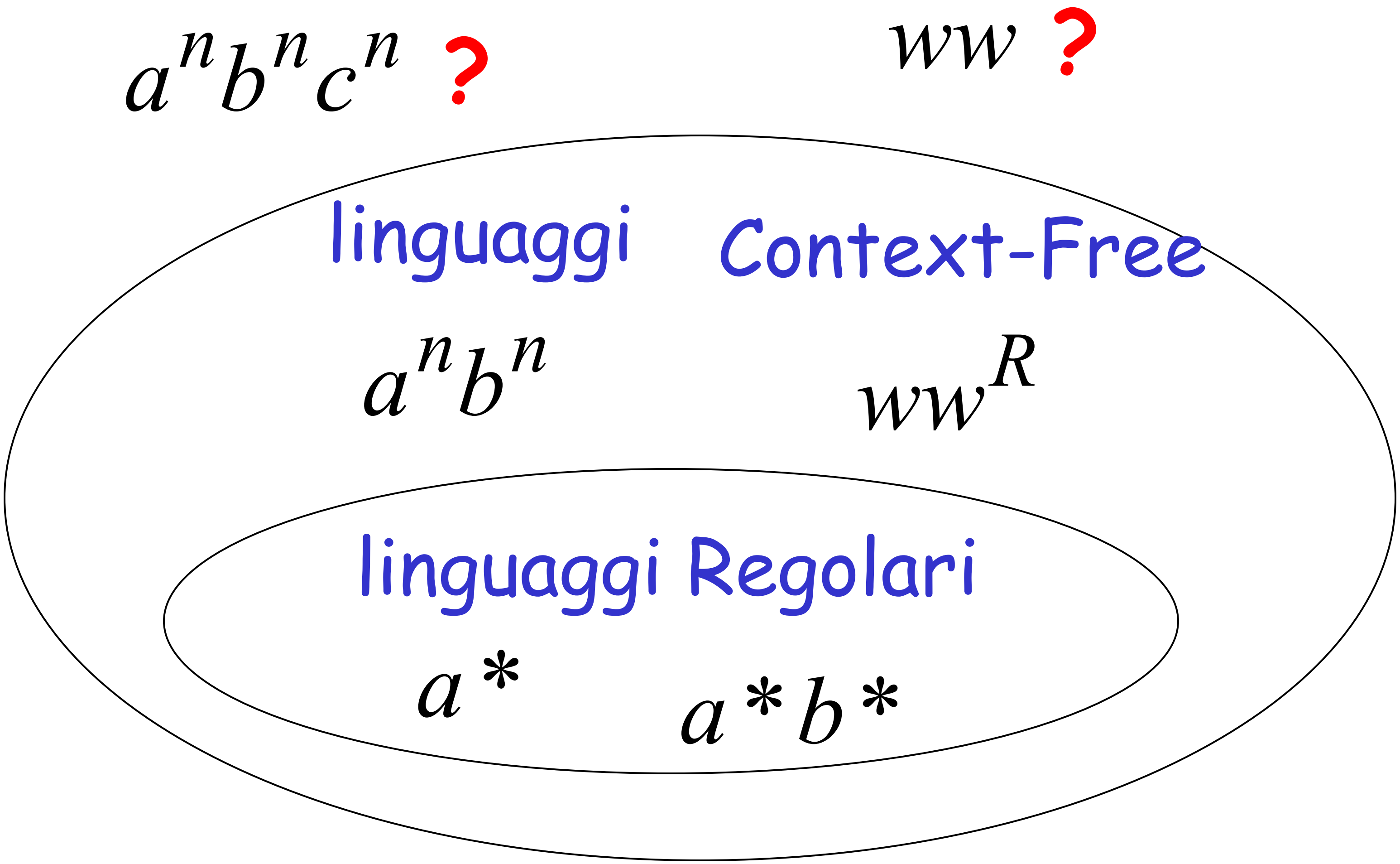
Accettare stringa In Input → se macchina si ferma in uno stato di accettazione

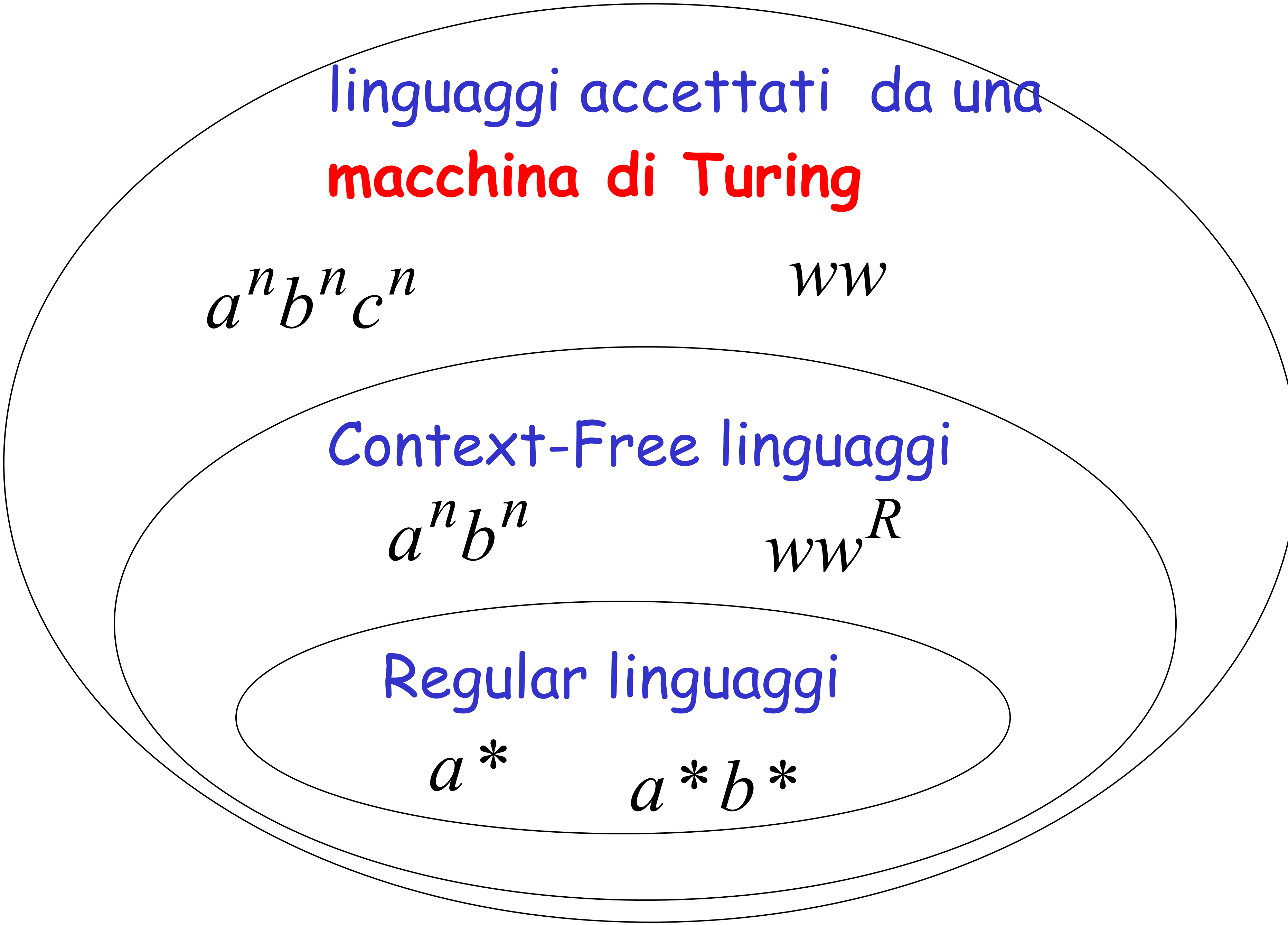
RIGETTARE stringa In Input → se macchina si ferma in uno stato di NON- accettazione o se la macchina entra in un *infinite loop*

Osservazione:

Nell'accettare una stringa di input, non è necessario esaminare tutti i simboli nella stringa.

La gerarchia dei linguaggi

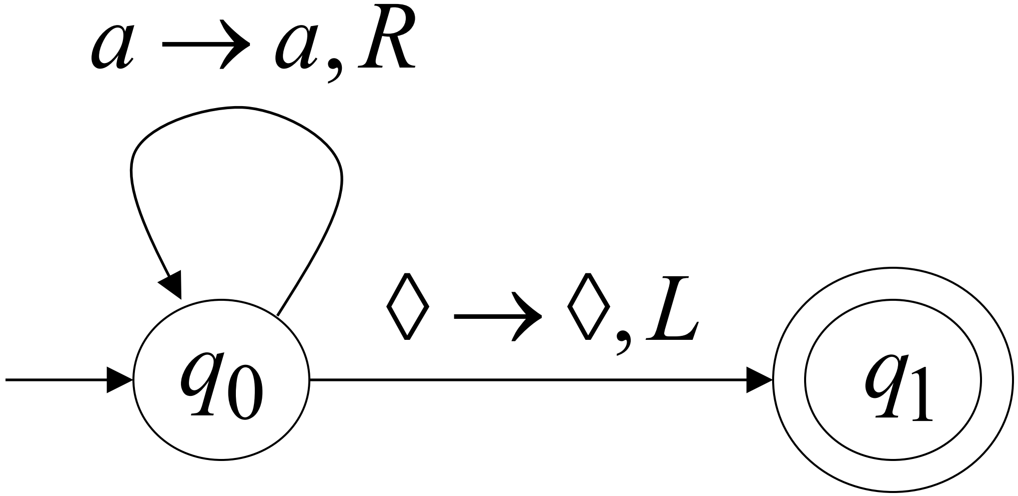



linguaggi accettati da una
macchina di Turing
$a^n b^n c^n$
ww
Context-Free linguaggi
$a^n b^n$
ww^R
Regular linguaggi
a^*
$a^* b^*$

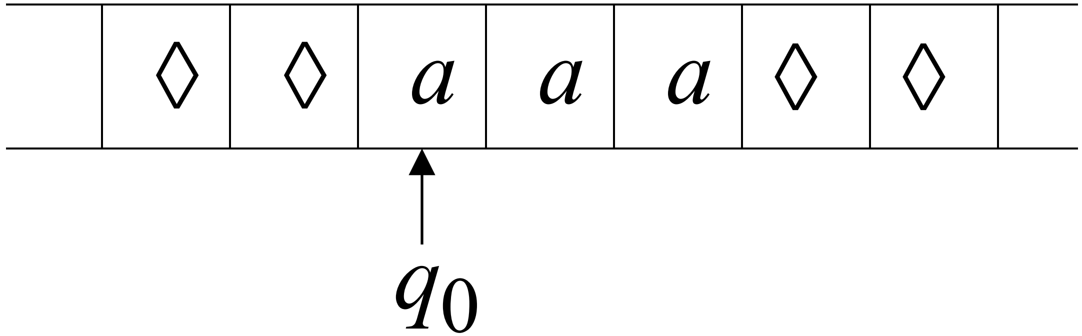
macchinaTuring esempio

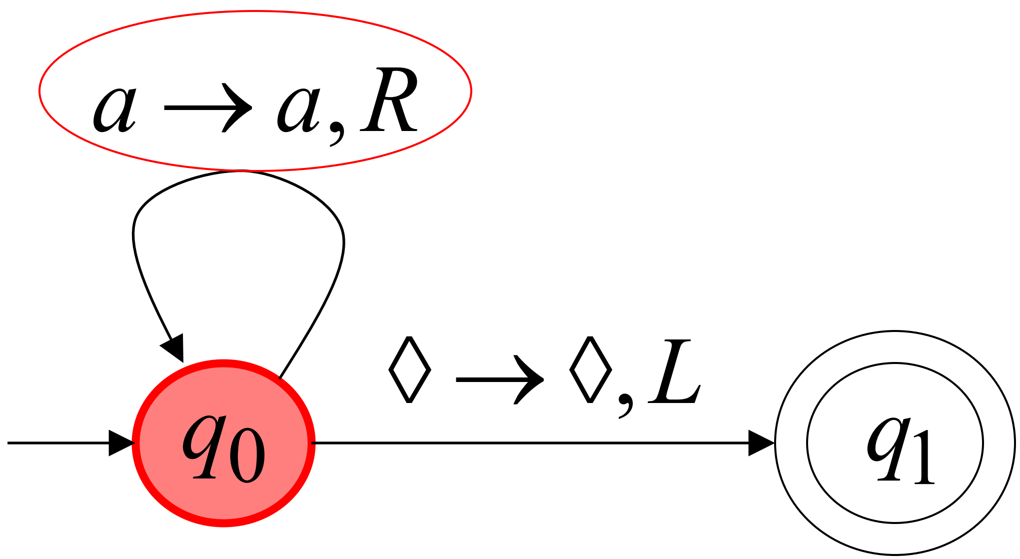
Input alphabet $\Sigma = \{a, b\}$

Accetta il linguaggio: a^*

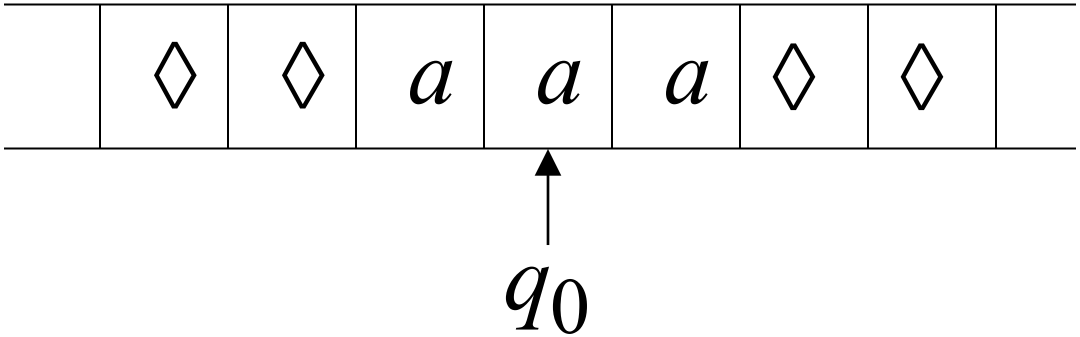


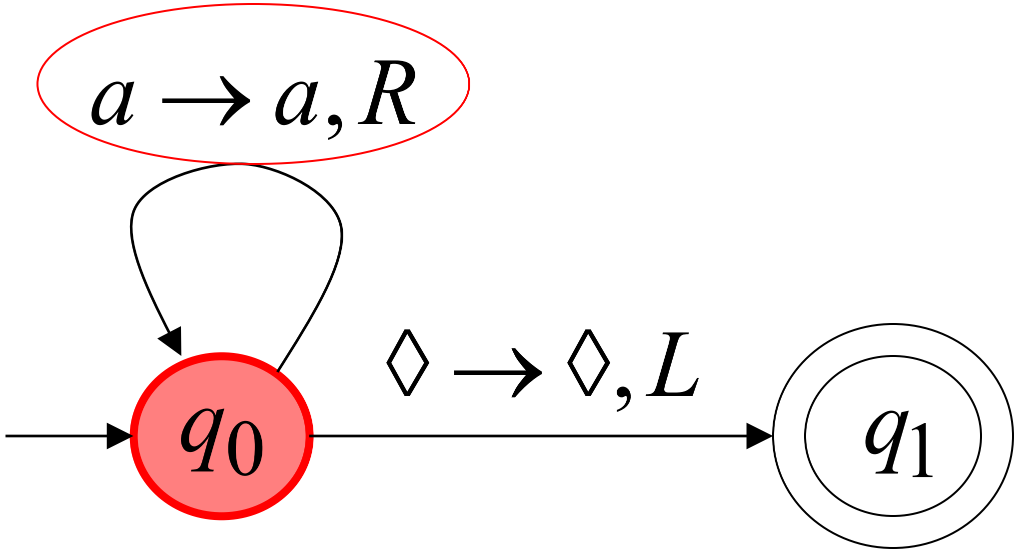
Time 0



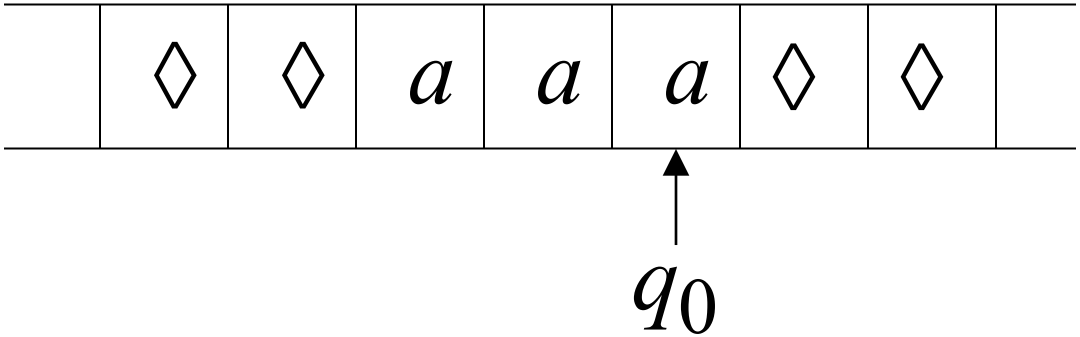


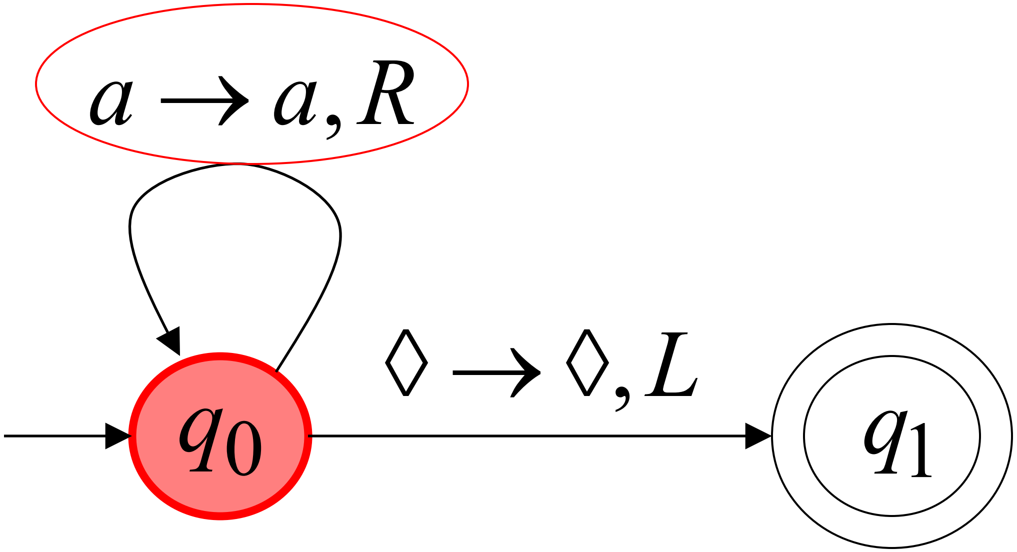
Time 1



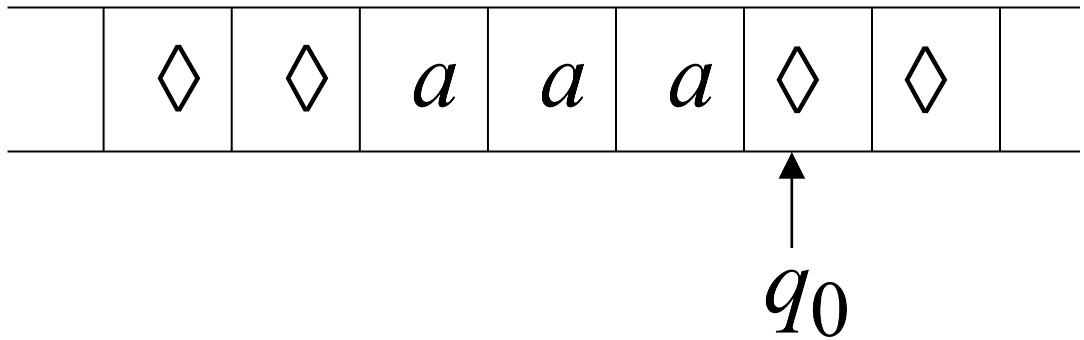


Time 2

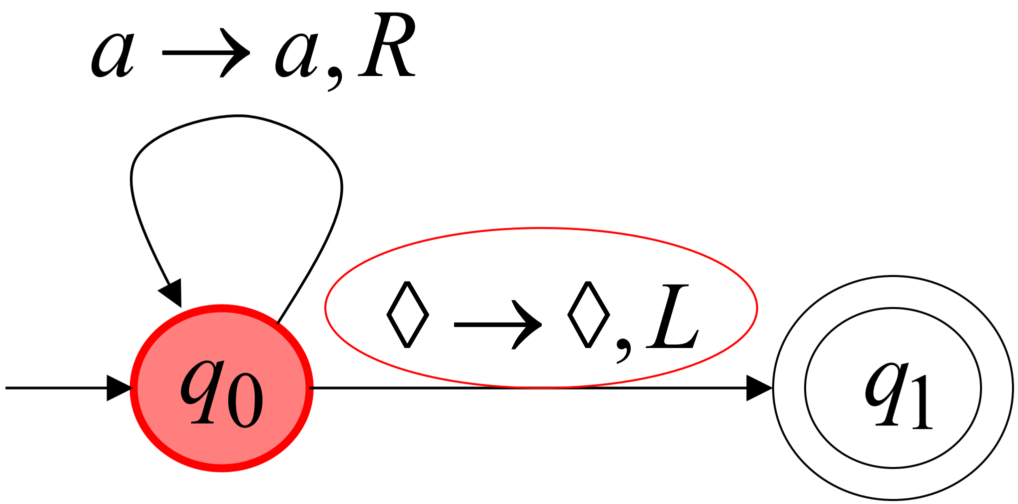




Time 3

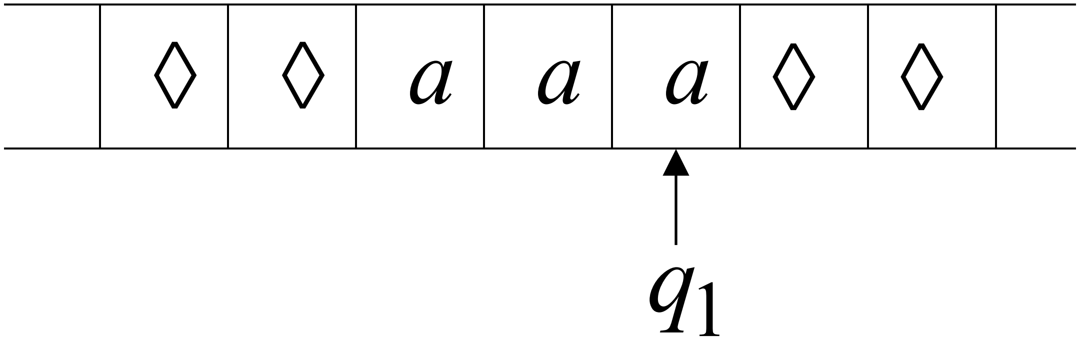


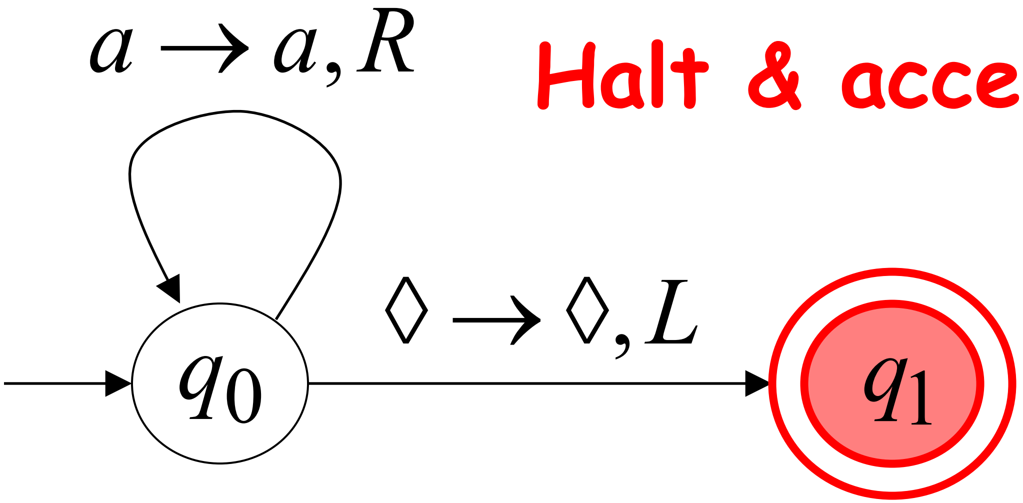
$\lozenge$ $\lozenge$ a a a $\lozenge$ $\lozenge$
q_0



$a \rightarrow a, R$
$\lozenge \rightarrow \lozenge, L$
q_0
q_1

Time 4

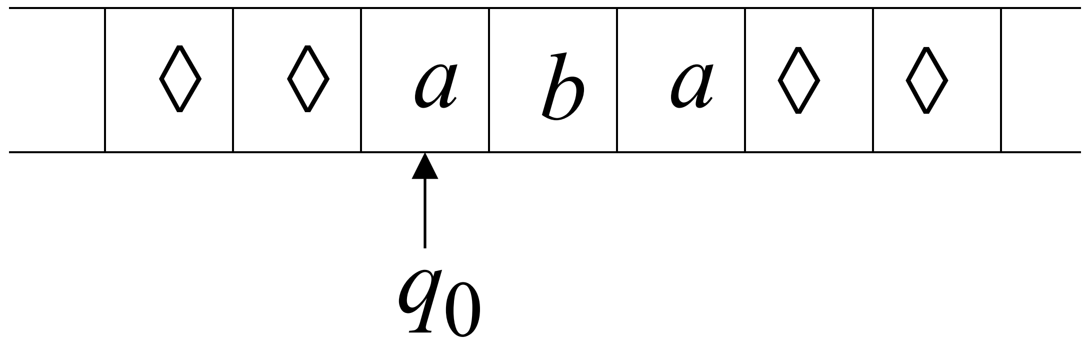


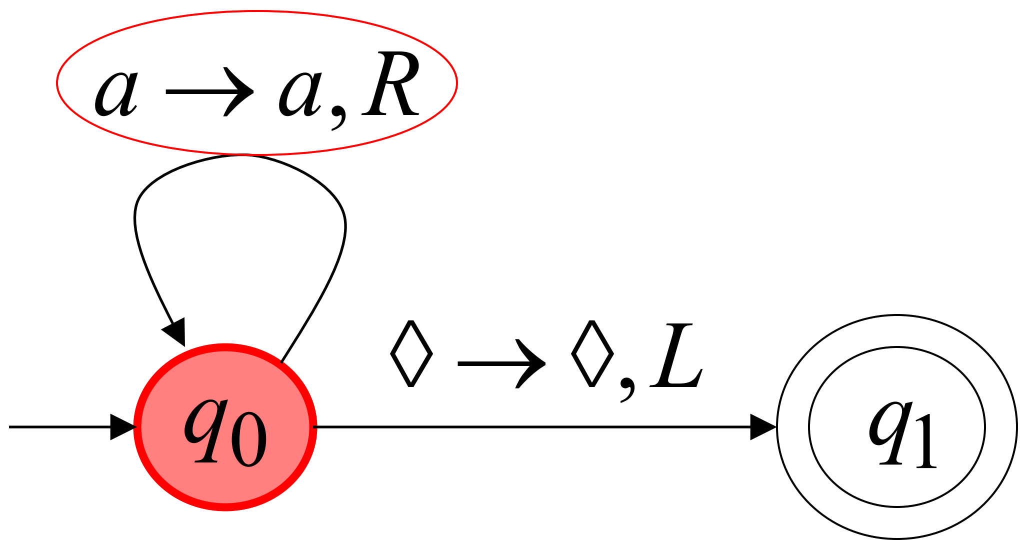


Halt & accettazione

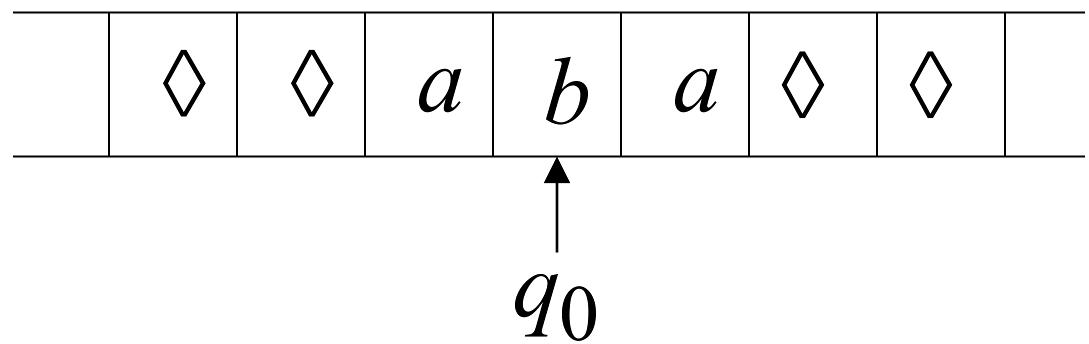
Rejection esempio

Time 0



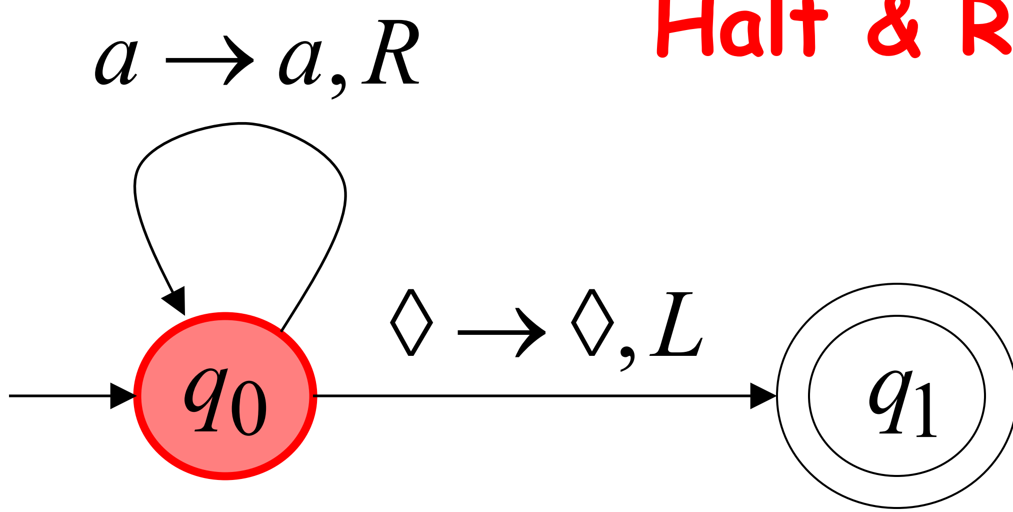


Time 1



Nessuna transizione

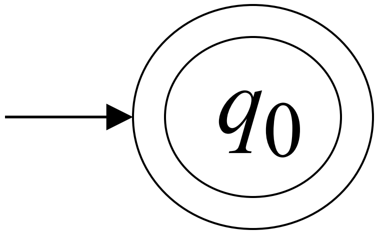
Halt & Reject



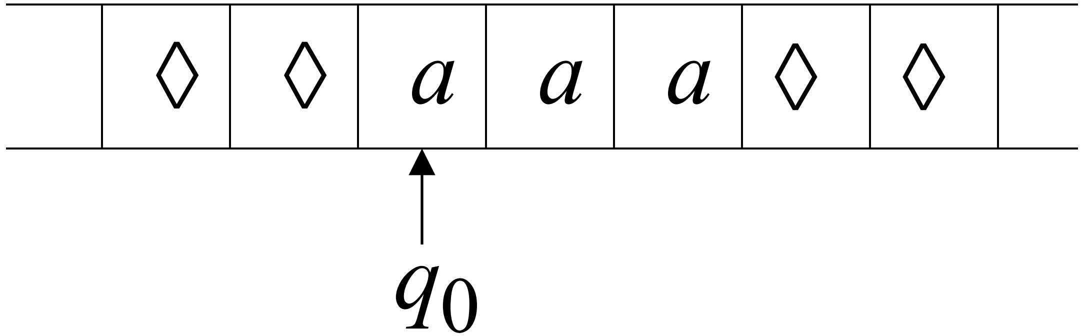
Una macchina semplice per linguaggio a^*

Ma con alfabeto di input $\Sigma = \{a\}$

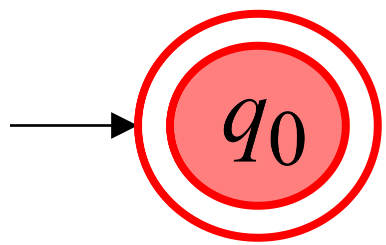
Accetta il linguaggio: a^*



Time 0



Halt & accettazione

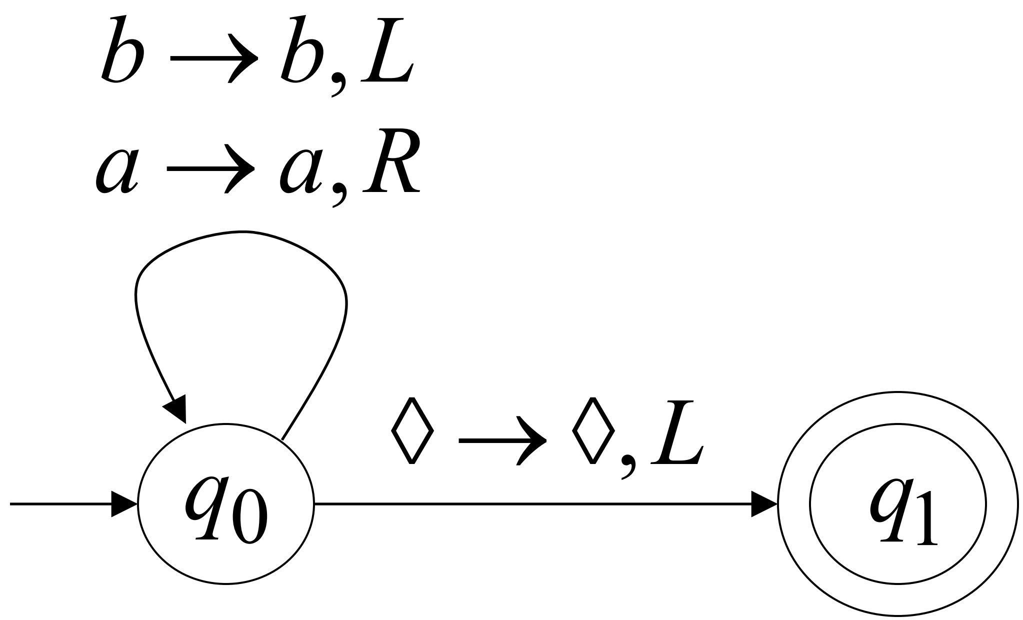


Non è necessario esaminare l'input

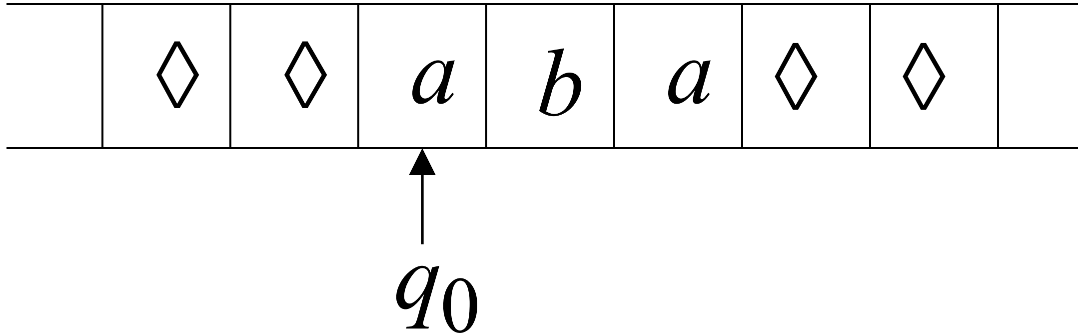
esempio Infinito Loop

una macchina di Turing

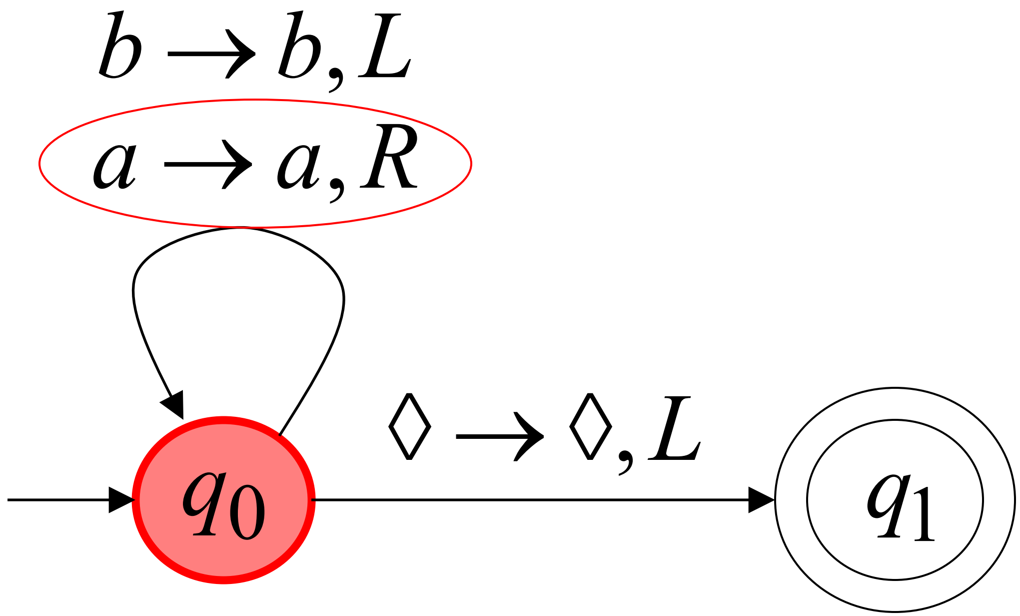
Per il linguaggio $a^*+b(a+b)^*$



Time 0

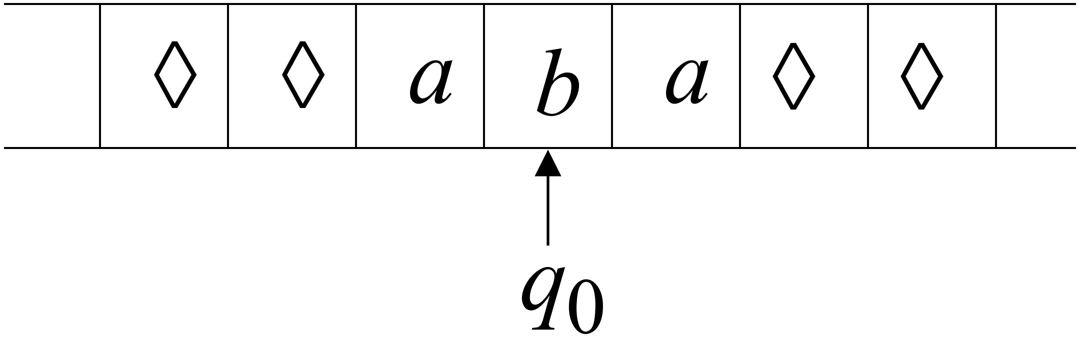


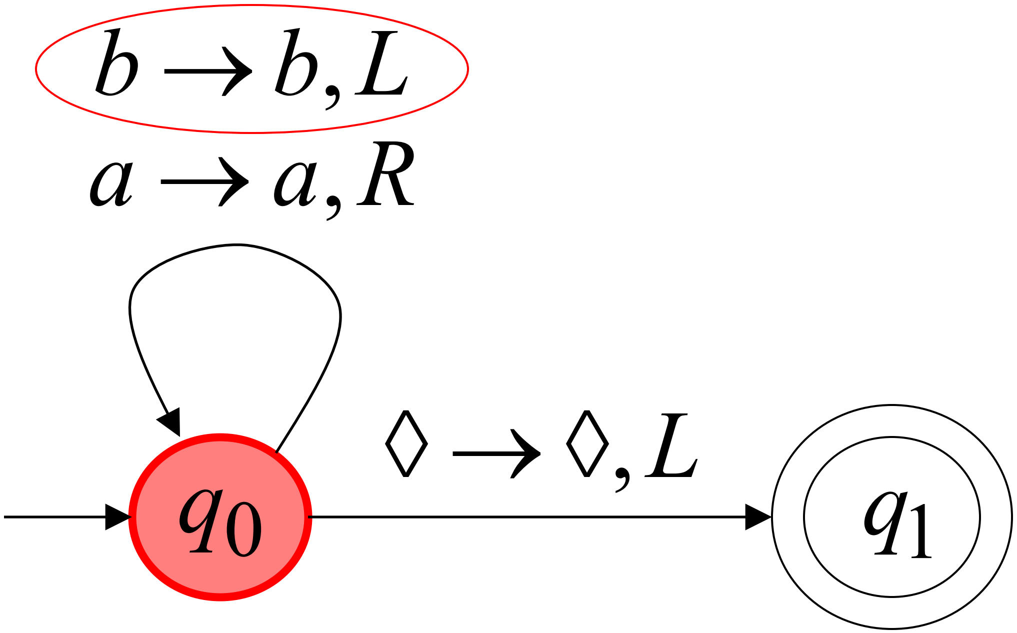
◊ ◊ a b a ◊ ◊
q_0



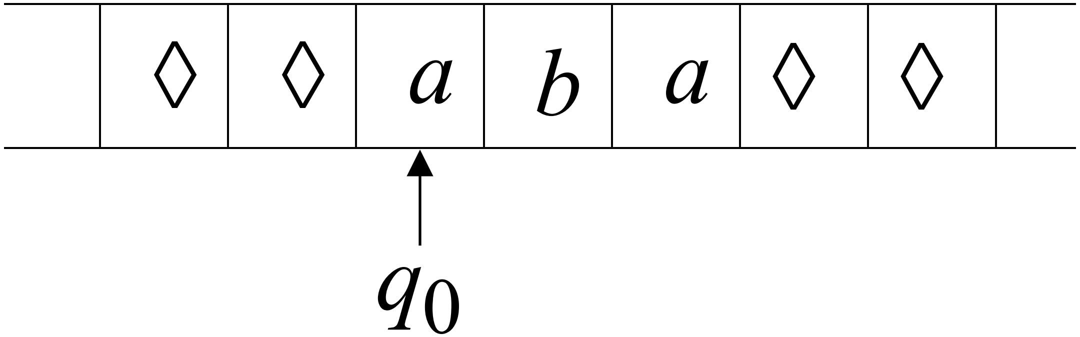
$b \rightarrow b, L$
$a \rightarrow a, R$
$\Diamond \rightarrow \Diamond, L$
q_0
q_1

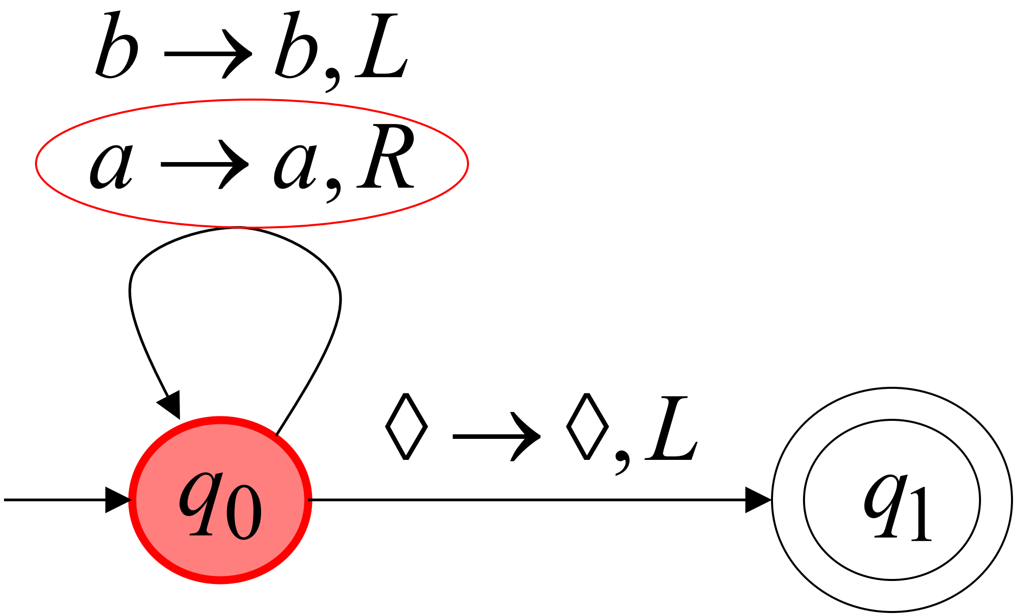
Time 1


◊
◊
a
b
a
◊
◊
q_0


$b \rightarrow b, L$
$a \rightarrow a, R$
$\Diamond \rightarrow \Diamond, L$
q_0
q_1

Time 2


◊
◊
a
b
a
◊
◊
q_0

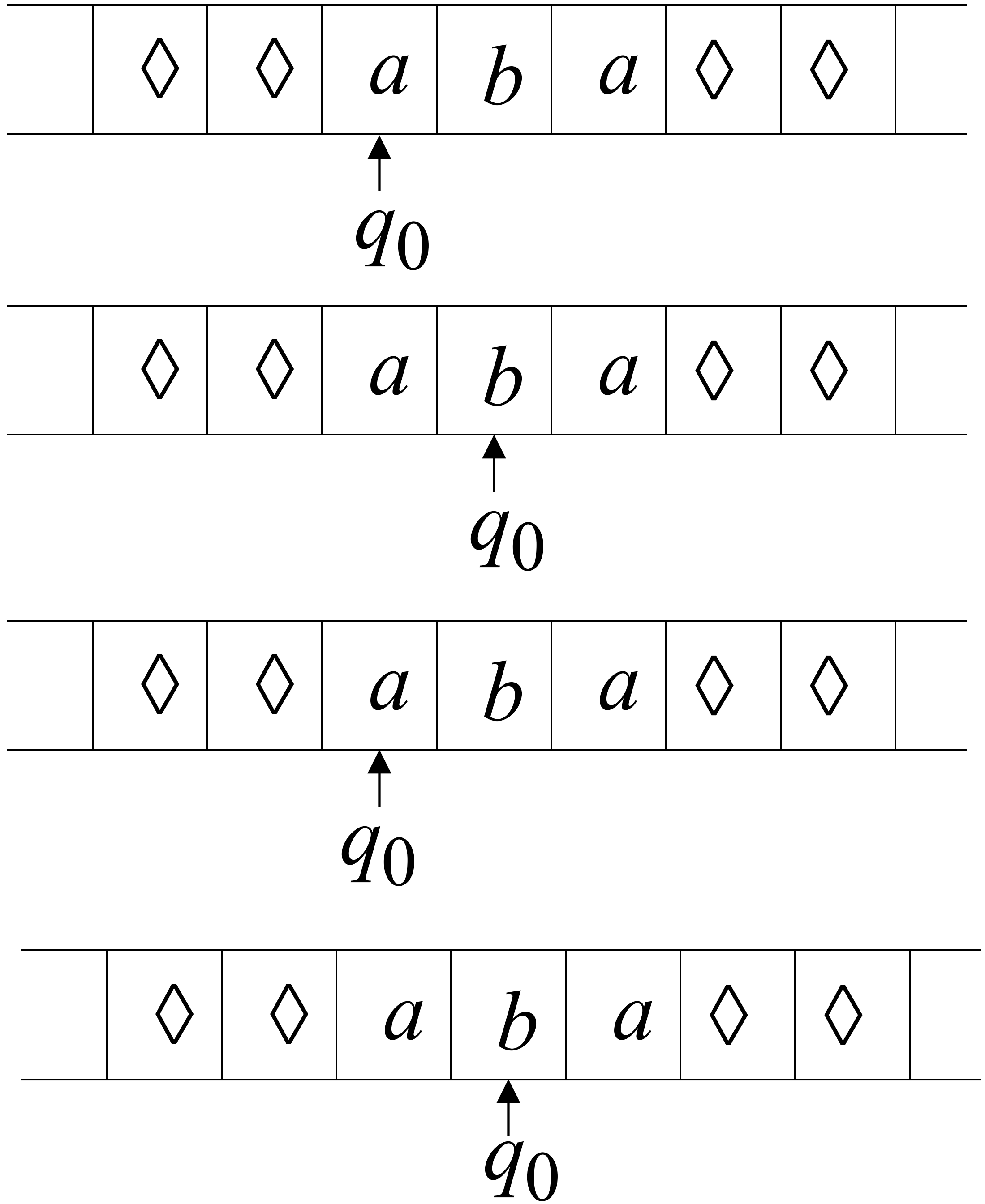

$b \rightarrow b, L$
$a \rightarrow a, R$
$\lozenge \rightarrow \lozenge, L$
q_0
q_1

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 piu a sinistra cambiala in x

trova la b piu 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 ->X q_1

q_1 devo scavalcare tutte le a tutte le Y u b

b->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

XXXaYYbb

XXXaYYYb

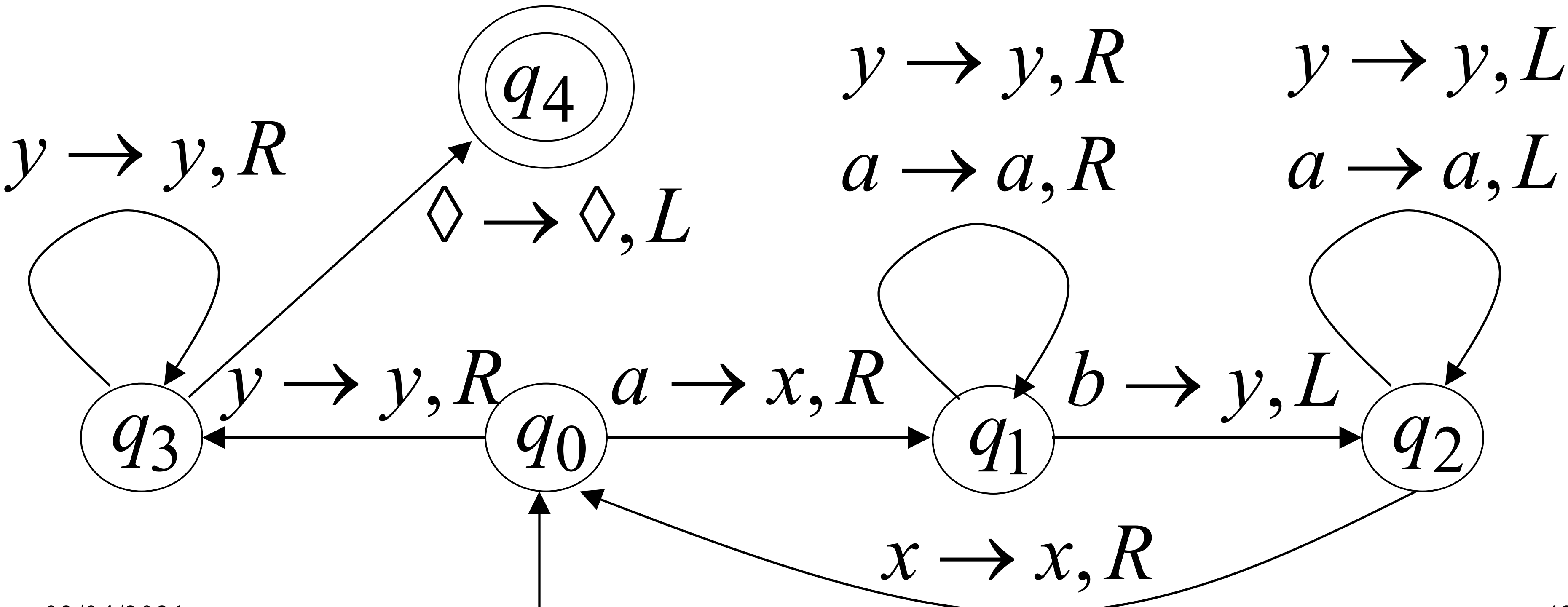
XXXXYYYb

XXXXYYYYb

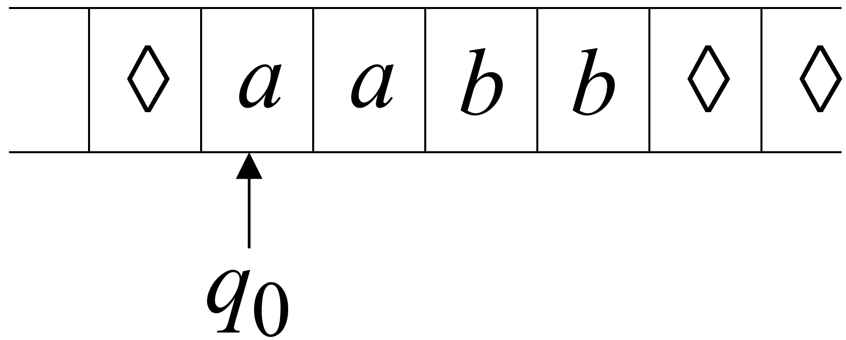
Turing macchina esempio

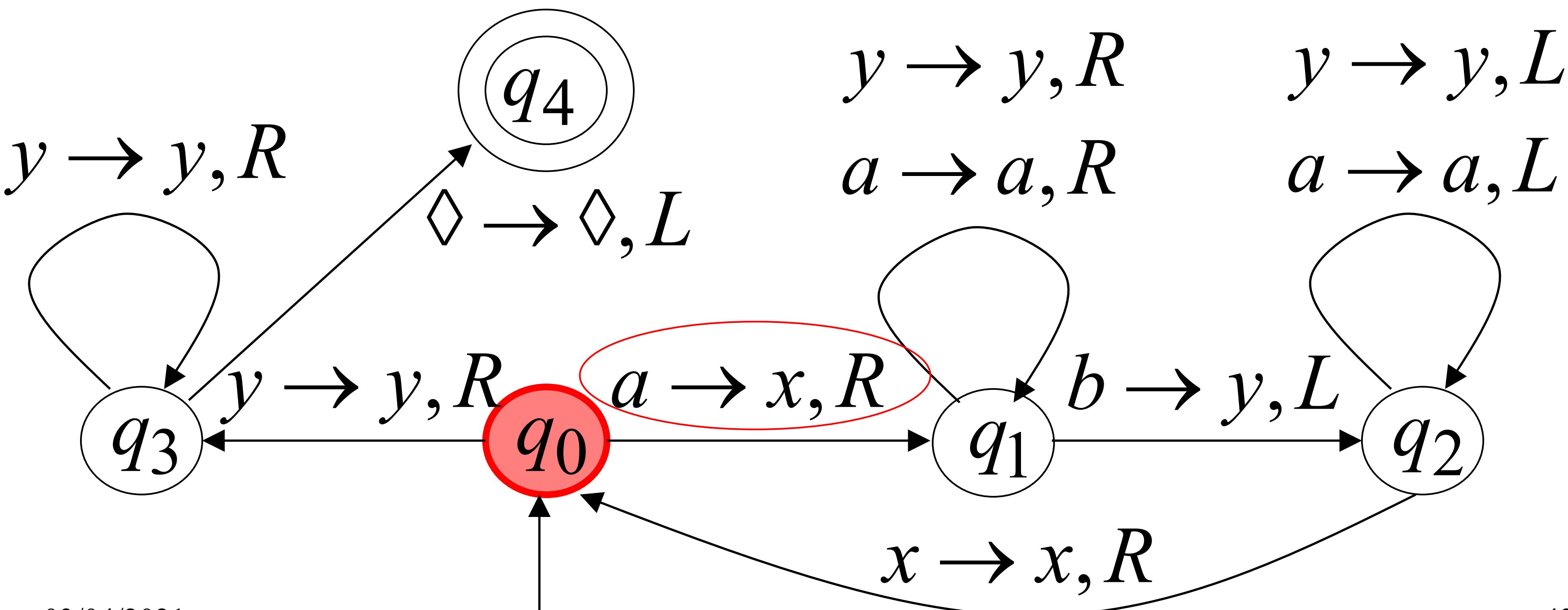
Machina di Turing per il linguaggio $\{a^n b^n\}$

$n \geq 1$

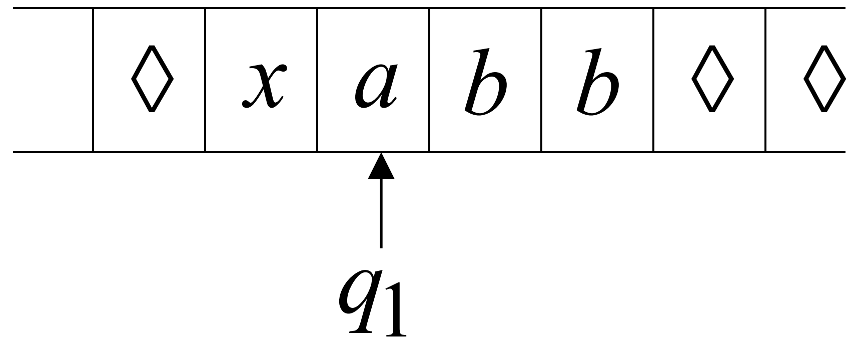


Time 0

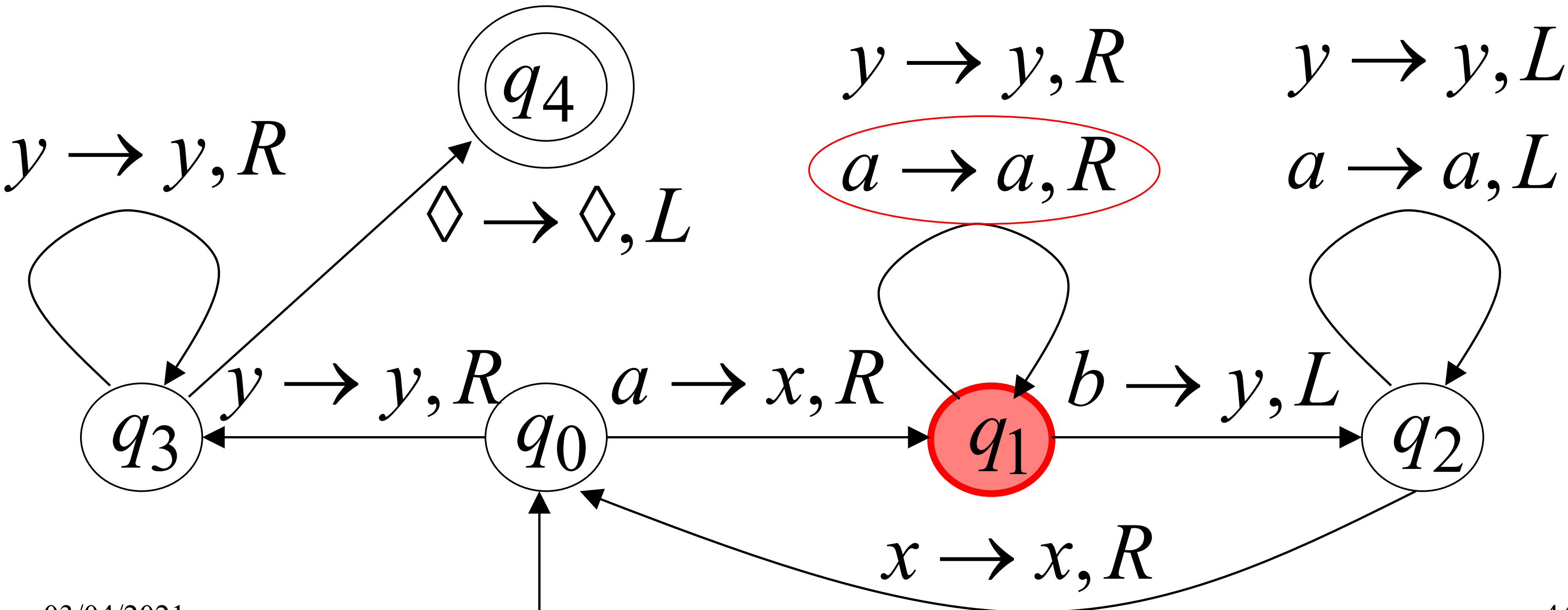

$\Diamond$ a a b b $\Diamond$ $\Diamond$
q_0


q_4
$y \rightarrow y, R$
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
q_3
q_0
q_1
q_2
$x \rightarrow x, R$

Time 1

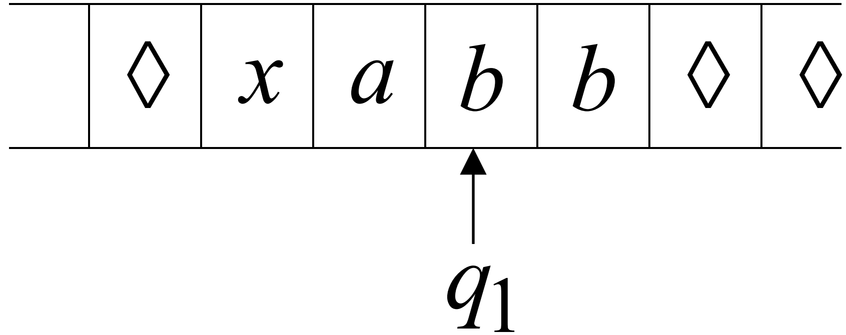


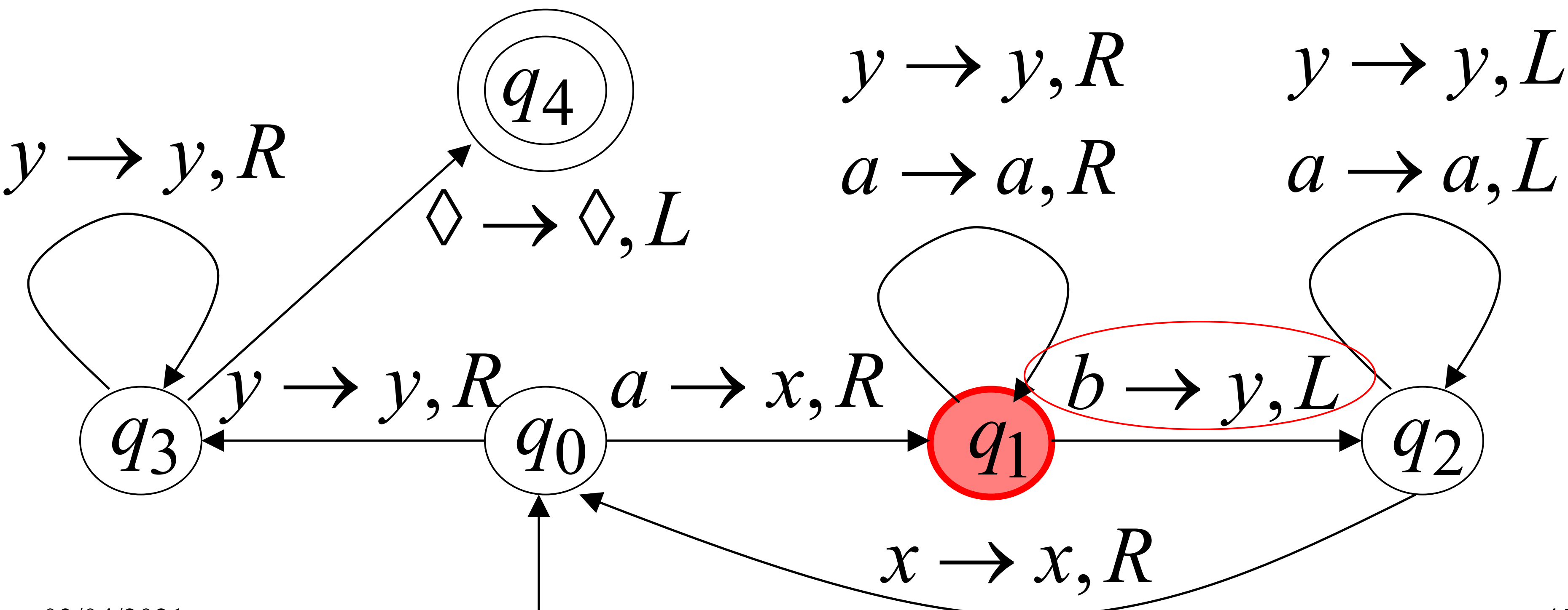
◊ x a b b ◊ ◊
q_1



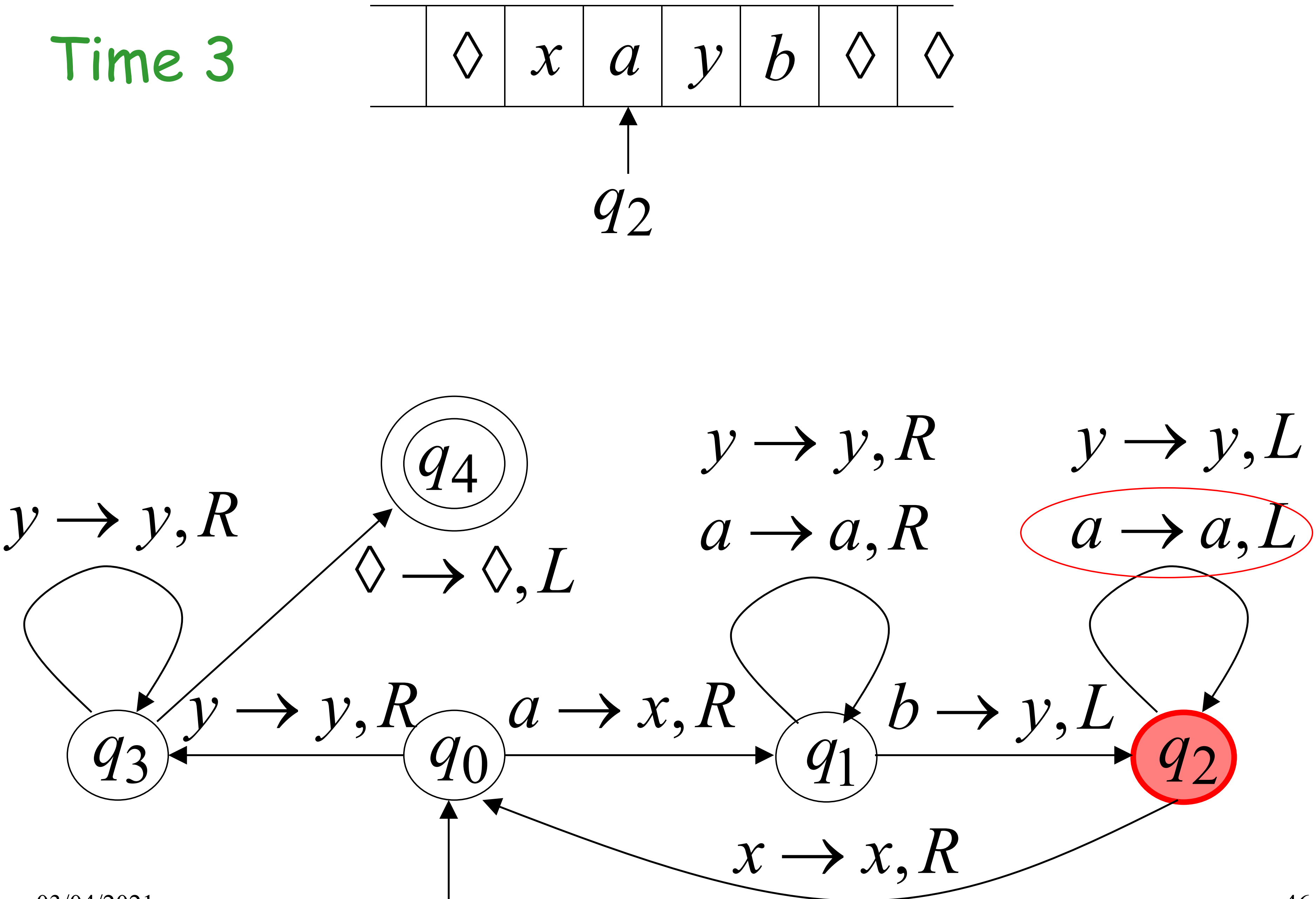
q_4
$y \rightarrow y, R$
◊ $\rightarrow$ ◊$, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

Time 2


◊ x a b b ◊ ◊
q_1

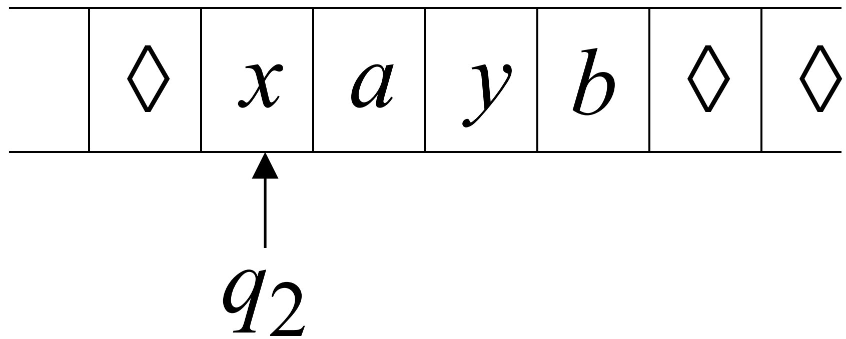

q_4
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
$y \rightarrow y, R$
$a \rightarrow x, R$
$b \rightarrow y, L$
q_3
q_0
q_1
q_2
$x \rightarrow x, R$

Time 3

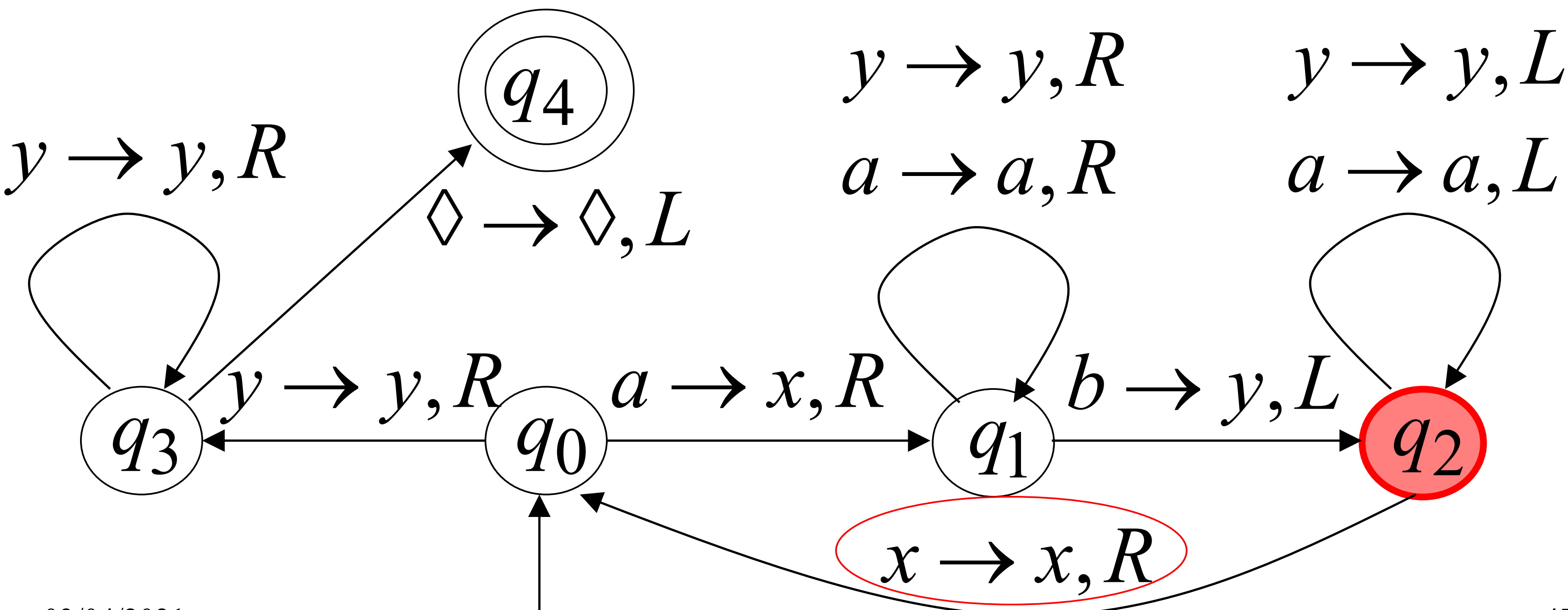


◊
x
a
y
b
◊
◊
q_2
q_4
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
q_3
$y \to y, R$
q_0
$a \to x, R$
q_1
$b \to y, L$
q_2
$x \to x, R$

Time 4

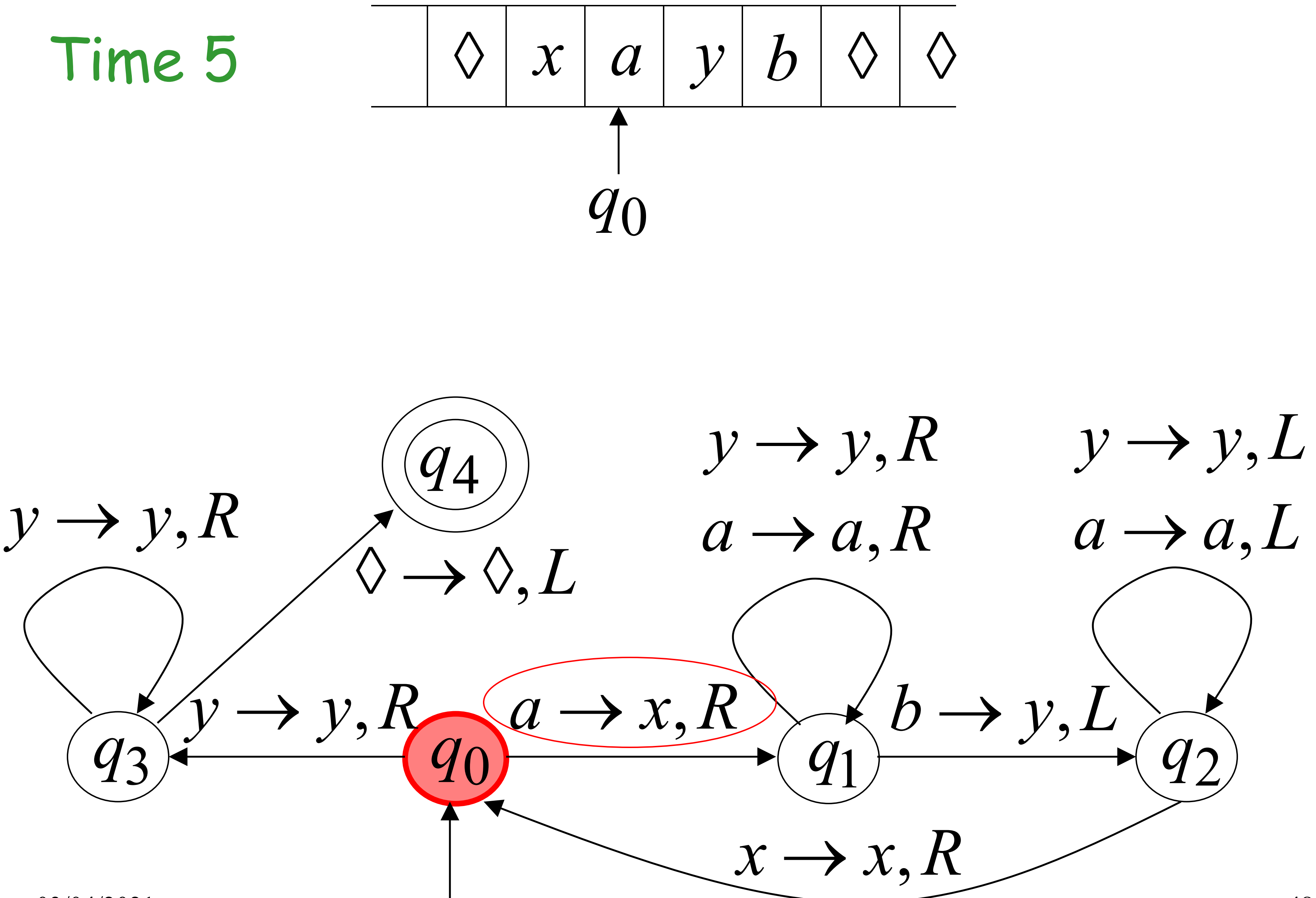


◊ x a y b ◊ ◊
q_2

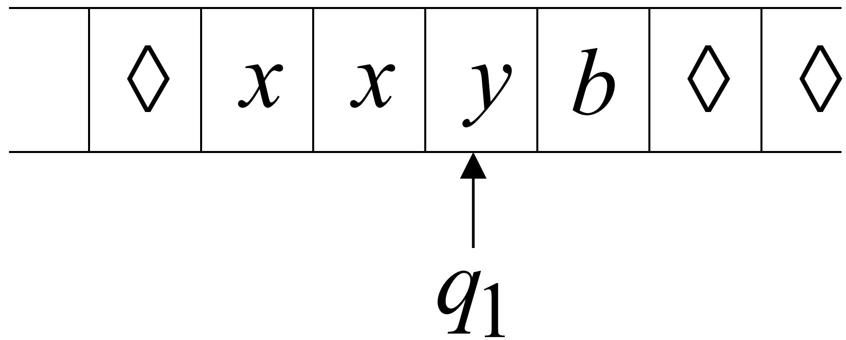


q_4
$y \rightarrow y, R$
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

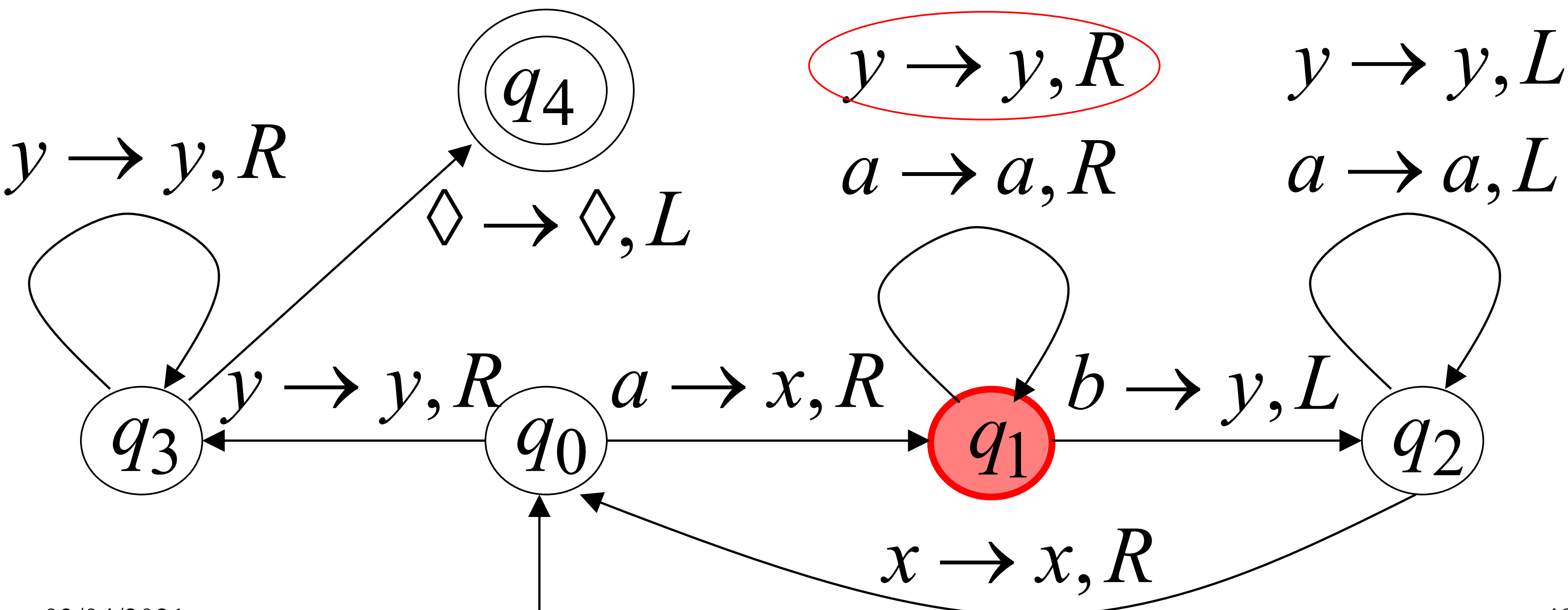
Time 5


◊
x
a
y
b
◊
◊
q_0
$y \to y, R$
q_4
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
q_3
$y \to y, R$
q_0
$a \to x, R$
q_1
$b \to y, L$
q_2
$x \to x, R$

Time 6

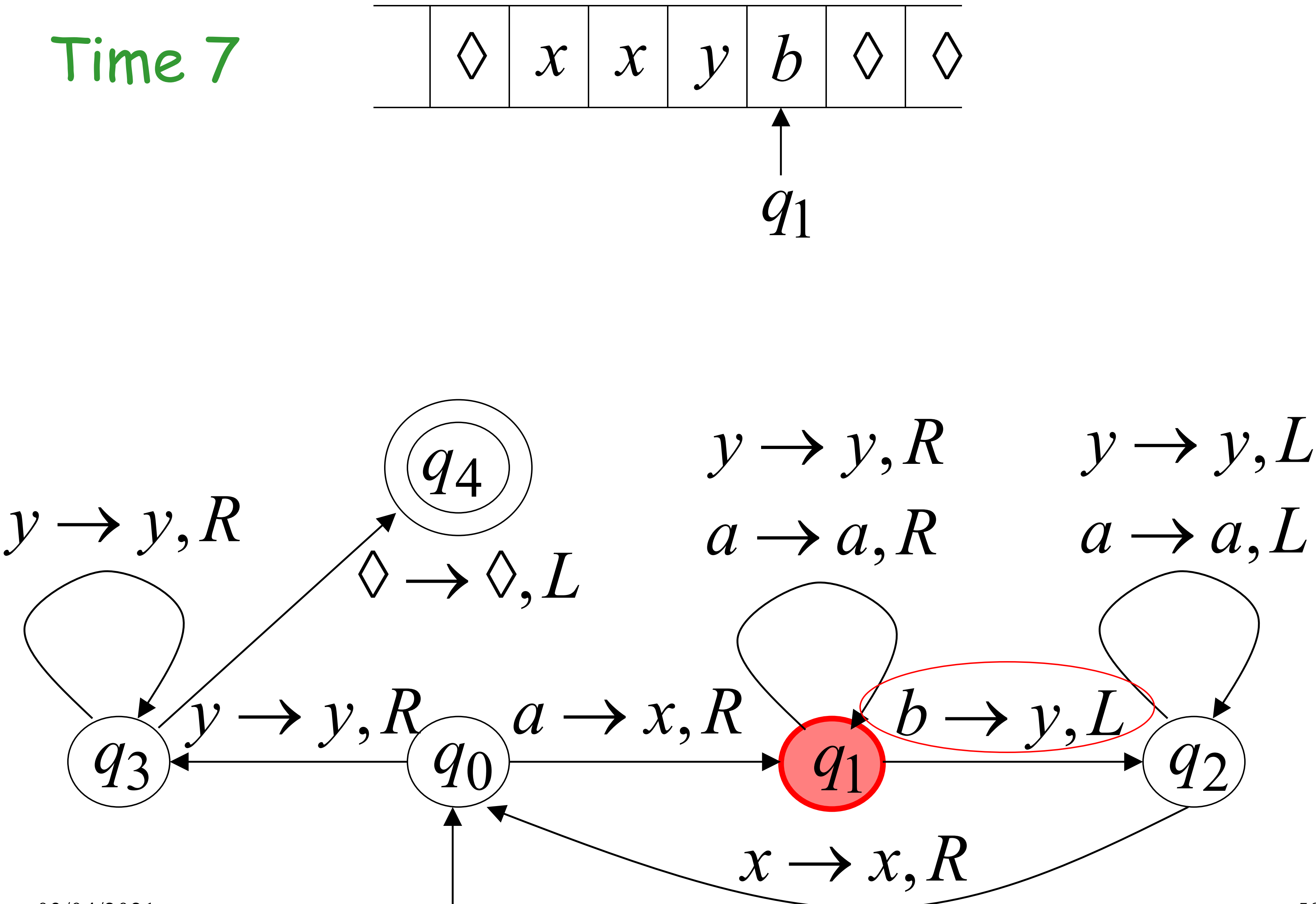


◊
x
x
y
b
◊
◊
q_1



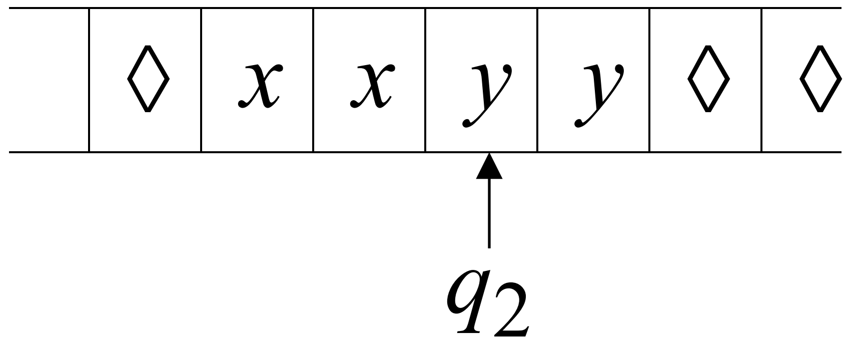
q_4
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
q_3
q_0
q_1
q_2
$x \to x, R$

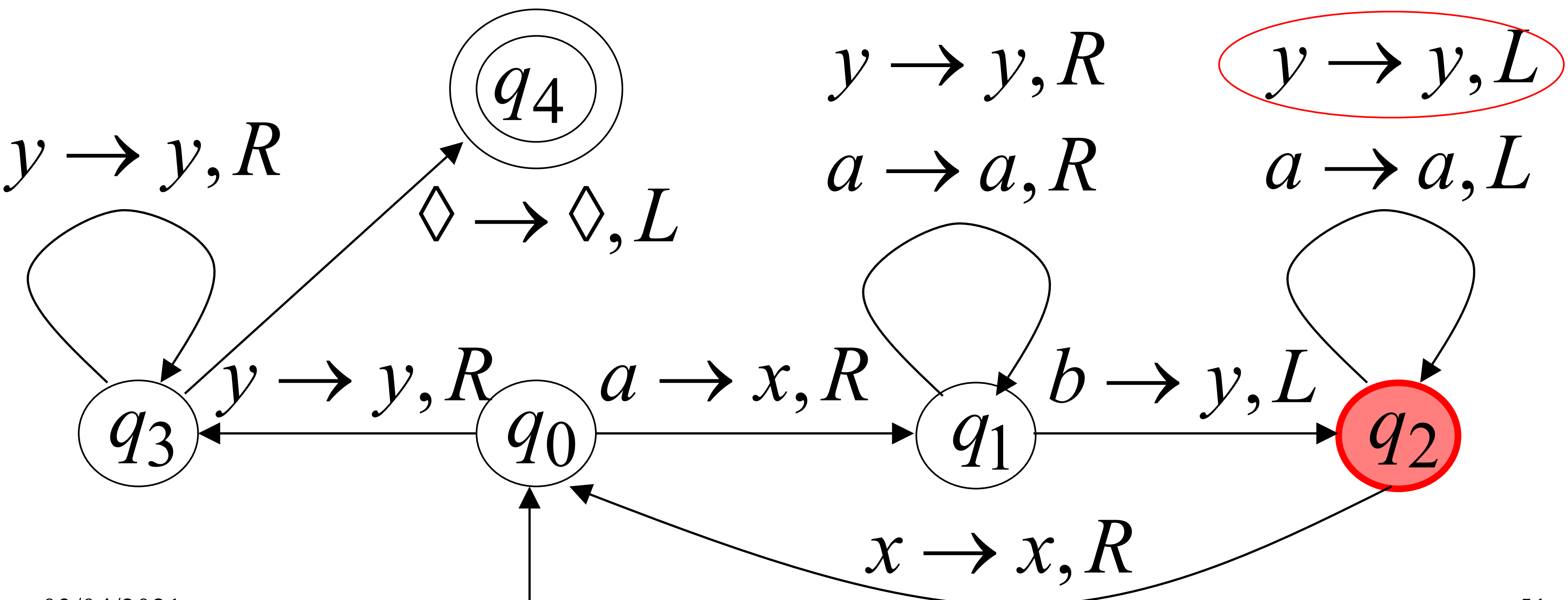
Time 7



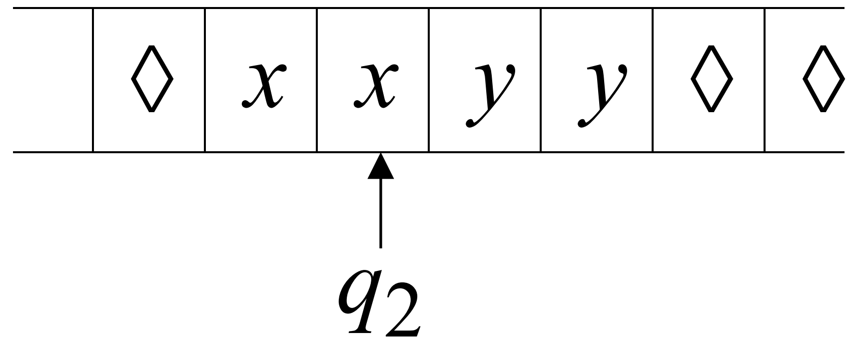
◊ x x y b ◊ ◊
q_1
q_4
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
q_3
q_0
q_1
q_2
$x \to x, R$

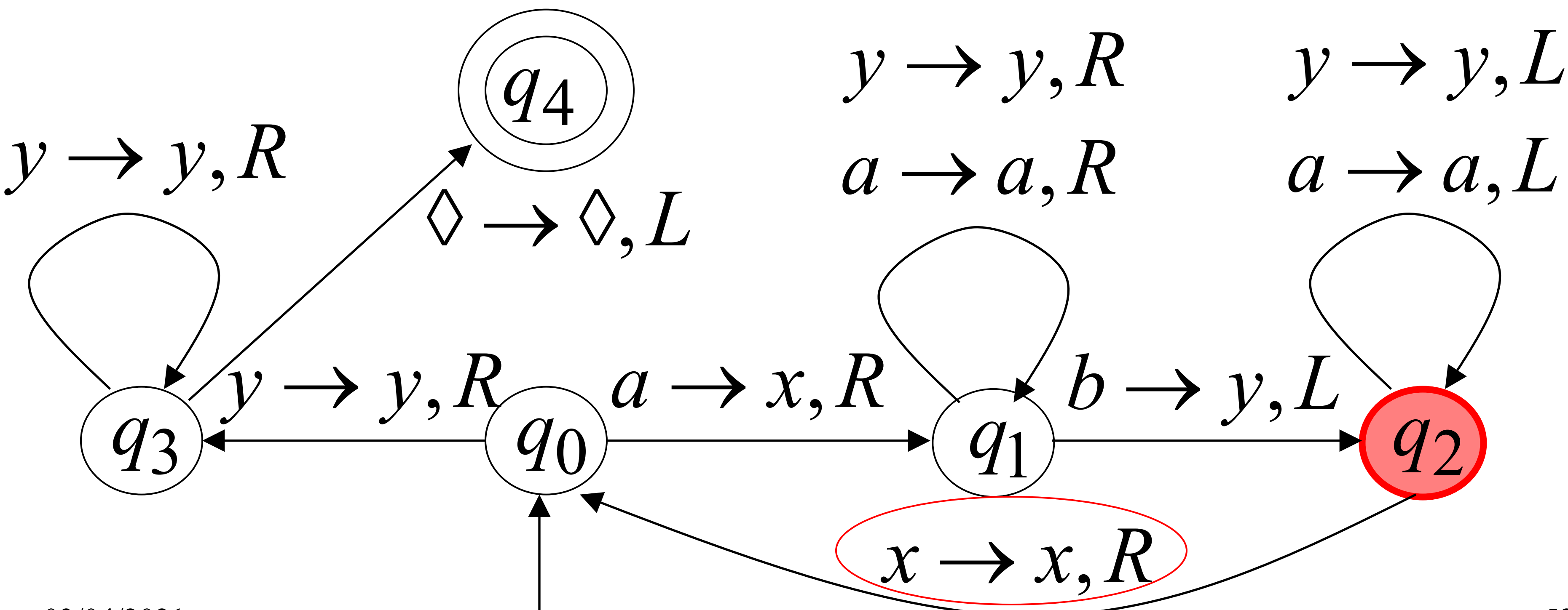
Time 8


◊ x x y y ◊ ◊
q_2

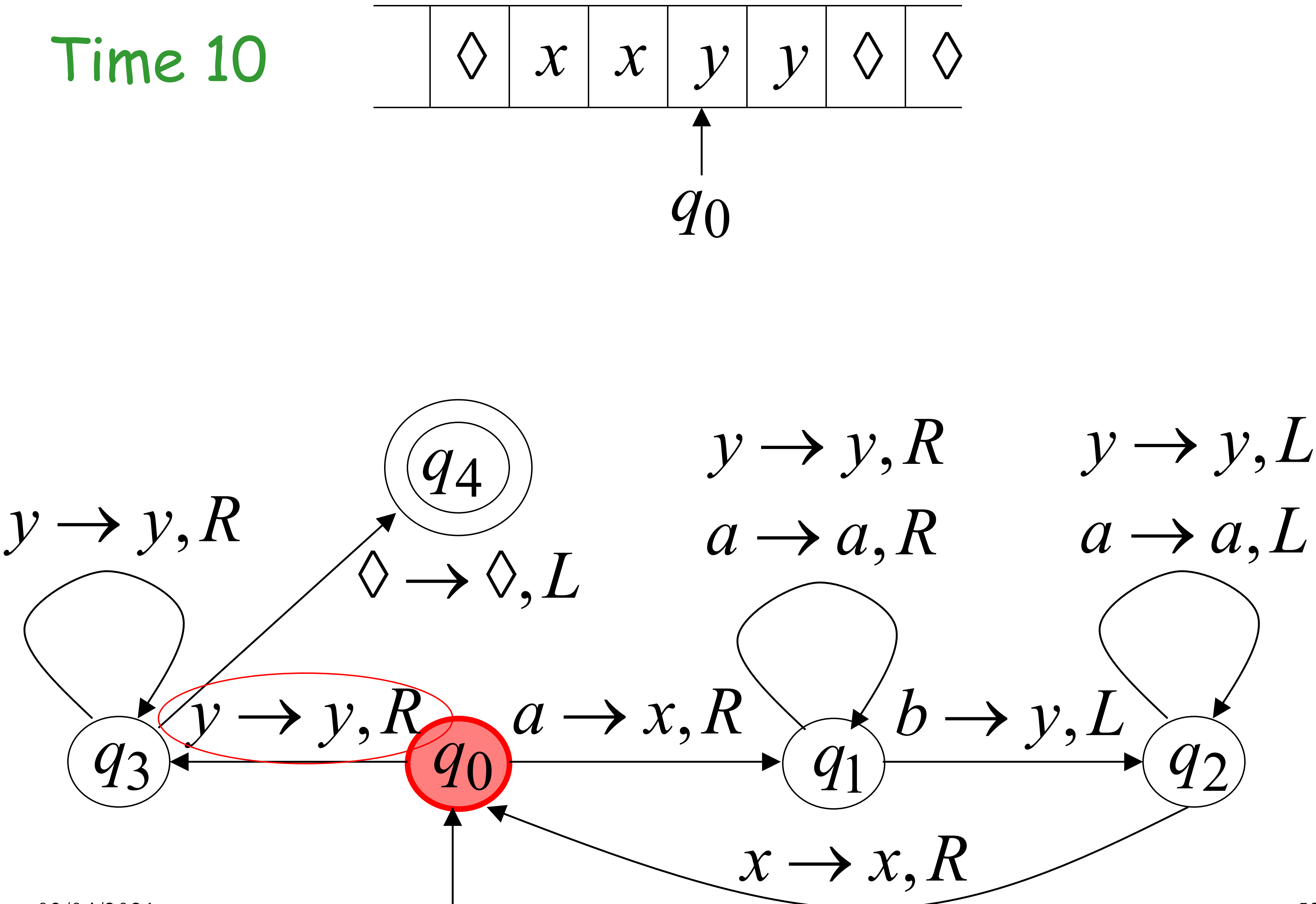

q_4
$y \to y, R$
$\Diamond \to \Diamond, L$
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
q_3
$y \to y, R$
q_0
$a \to x, R$
q_1
$b \to y, L$
q_2
$x \to x, R$

Time 9

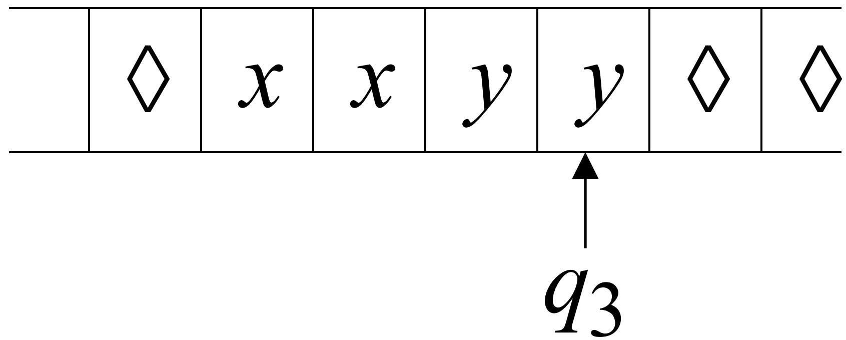

$\Diamond$ x x y y $\Diamond$ $\Diamond$
q_2

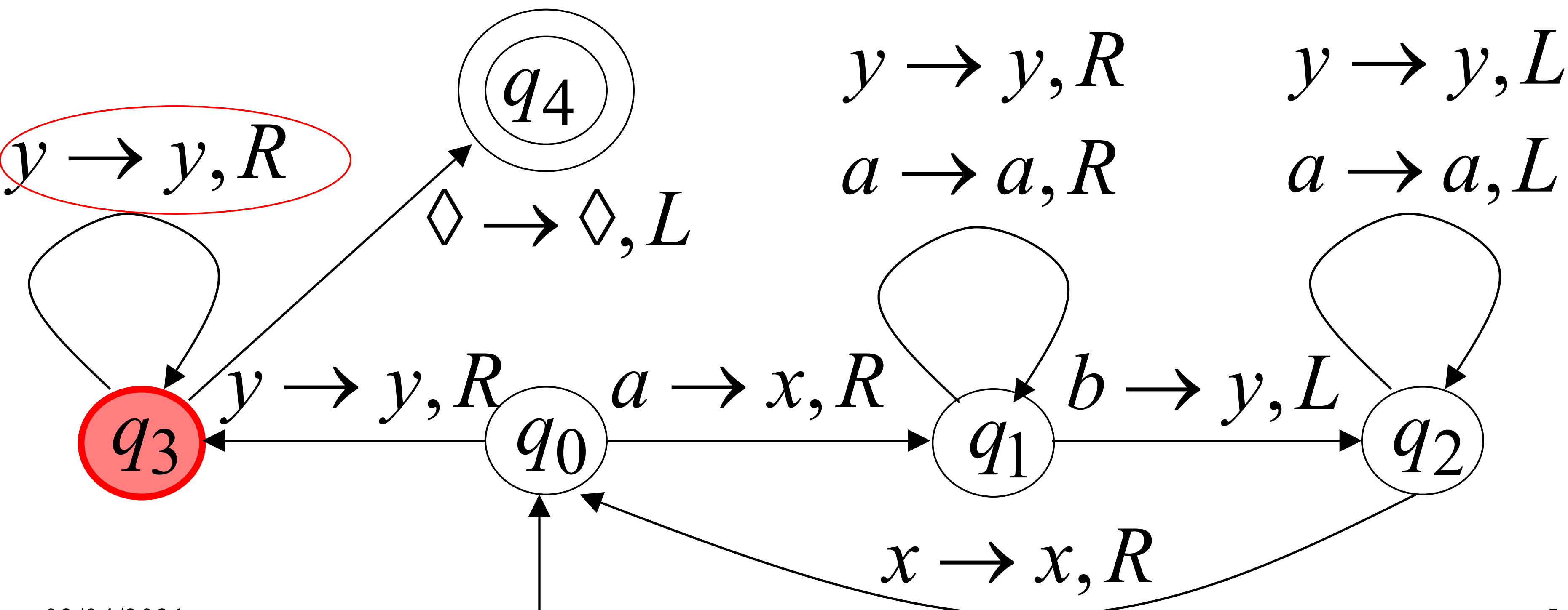

q_4
$\Diamond \rightarrow \Diamond, L$
$y \rightarrow y, R$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

Time 10

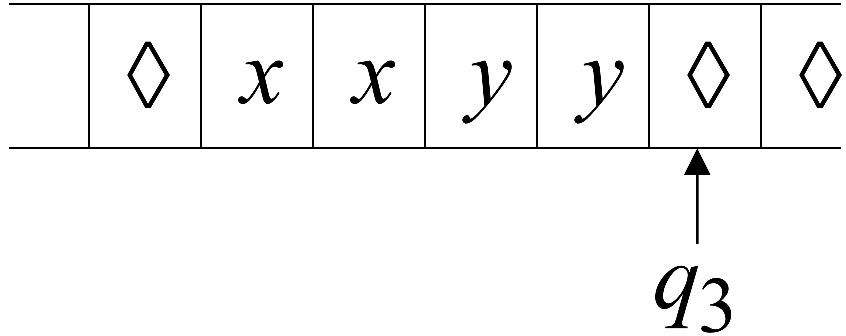

◊
x
x
y
y
◊
◊
q_0
q_4
$y \to y, R$
$a \to a, R$
$y \to y, L$
$a \to a, L$
$y \to y, R$
$\lozenge \to \lozenge, L$
$y \to y, R$
$a \to x, R$
$b \to y, L$
q_3
q_0
q_1
q_2
$x \to x, R$

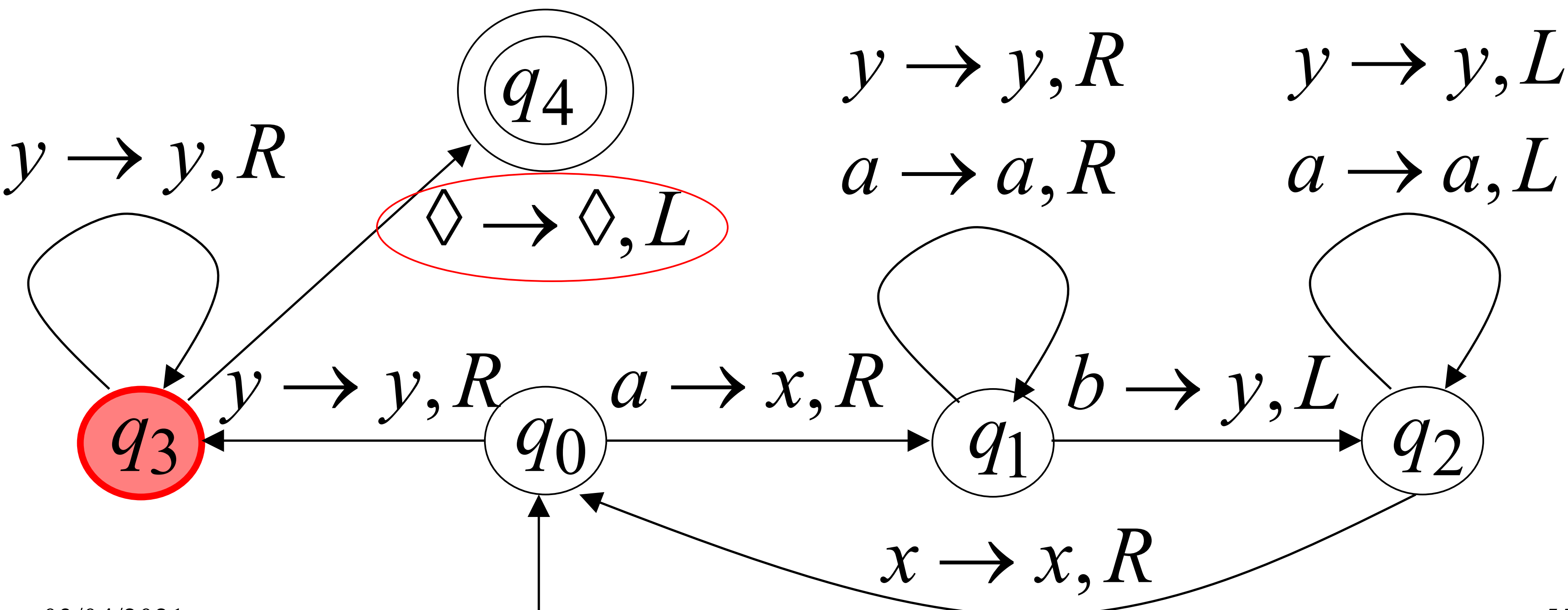
Time 11



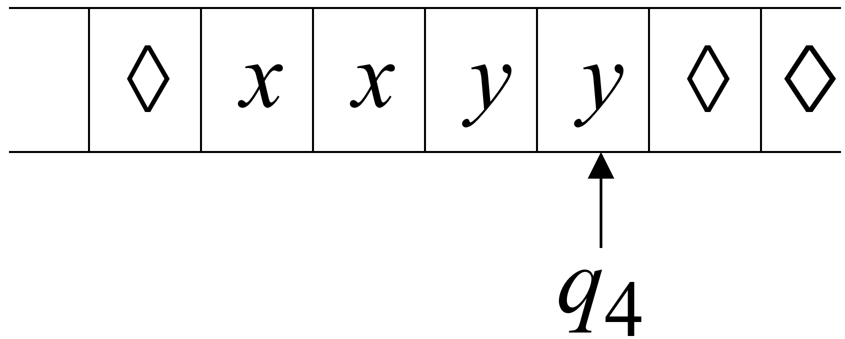


Time 12

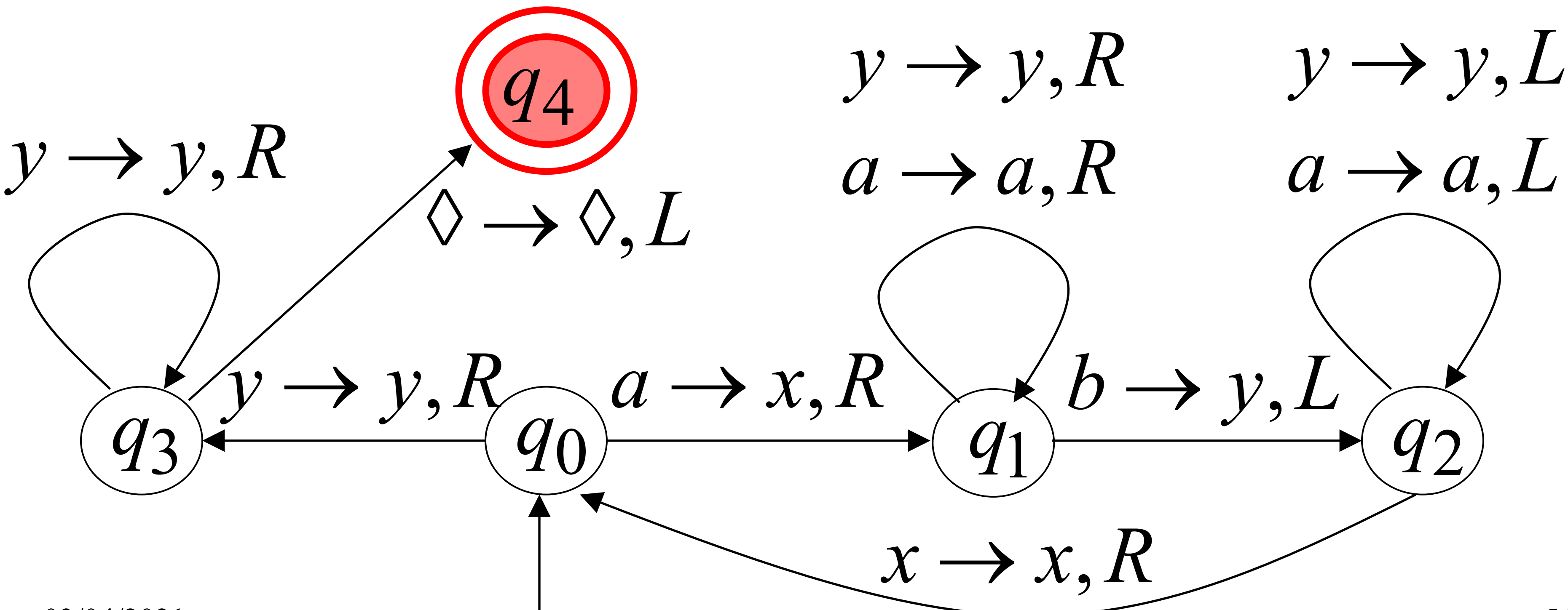

◊
x
x
y
y
◊
◊
q_3


q_4
$y \rightarrow y, R$
$\lozenge \rightarrow \lozenge, L$
$y \rightarrow y, R$
$a \rightarrow a, R$
$y \rightarrow y, L$
$a \rightarrow a, L$
q_3
$y \rightarrow y, R$
q_0
$a \rightarrow x, R$
q_1
$b \rightarrow y, L$
q_2
$x \rightarrow x, R$

Time 13



Halt & accettazione



Osservazione:

Se modifichiamo

La macchina per il linguaggio $\{a^n b^n\}$

Facilmente possiamo costruire

Una macchina per il linguaggio $\{a^n b^n c^n\}$

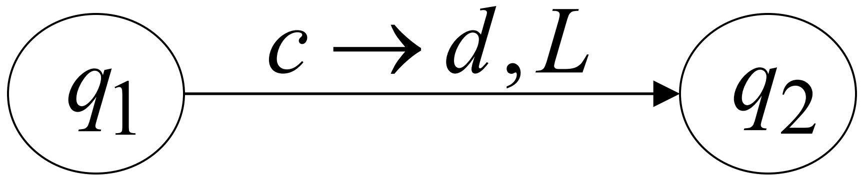
Definizione formale di macchina di turing

Funzione Transizione

$$q_1 \xrightarrow{\;a \rightarrow b, R\;} q_2$$

$$\delta(q_1, a) = (q_2, b, R)$$

Funzione Transizione



$$\delta(q_1, c) = (q_2, d, L)$$

Turing macchina:

stati

Input alfabeto

Tape alfabeto

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

Transition function

iniziale stato

blank

accettazi stati

Tipo della delta

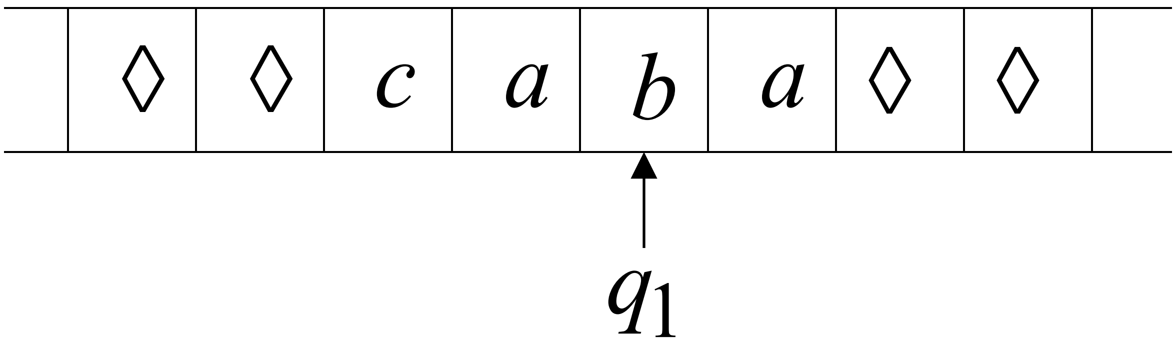
Input

Stati xAI U AN->

Stati x AI U AN x Op

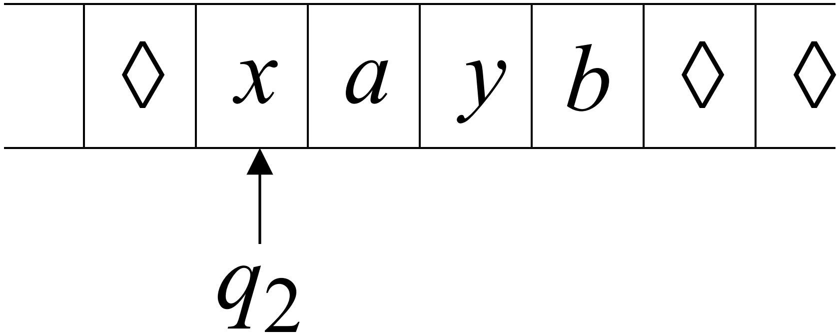
Op =L. R

Configurazione

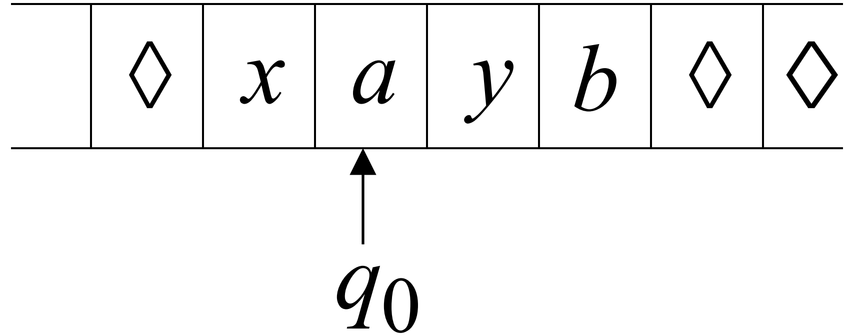


descrizione istantanea: $ca\ q_1\ ba$

Time 4



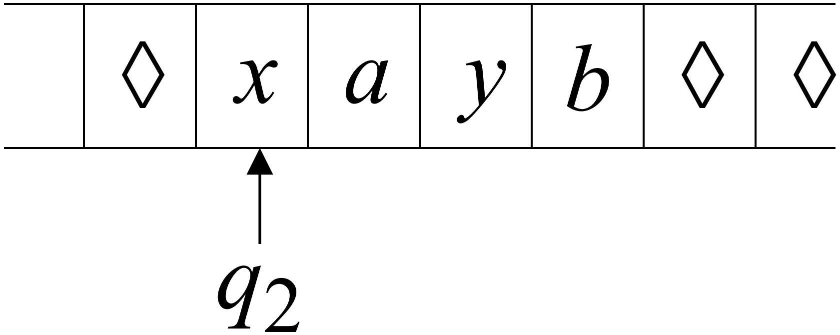
Time 5



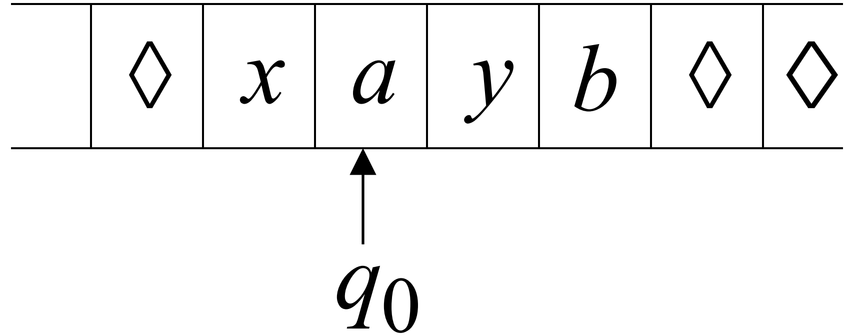
Una mossa $\quad q_2\ xayb \succ x\ q_0\ ayb$

(dà)

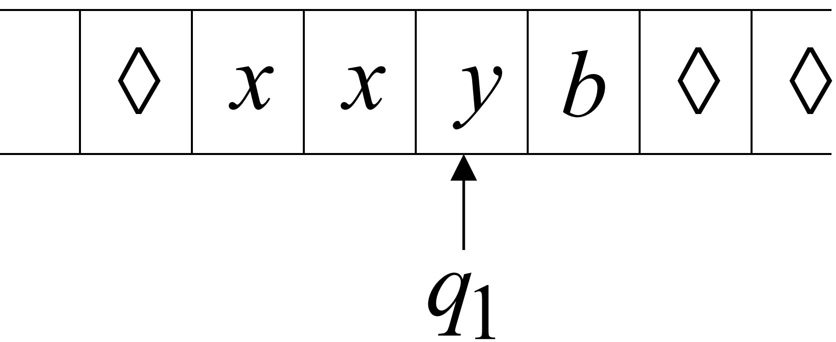
Time 4



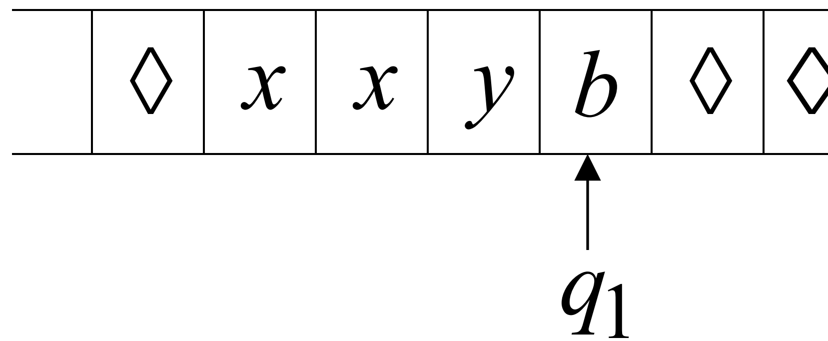
Time 5



Time 6



Time 7



Un calcolo

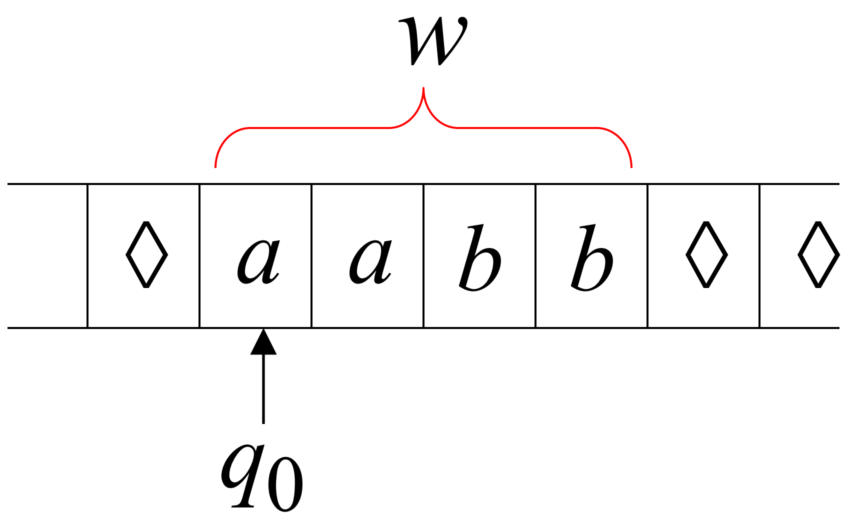
$$q_2\ xayb \succ x\ q_0\ ayb \succ xx\ q_1\ yb \succ xxy\ q_1\ b$$

$$q_2\ xayb \succ x\ q_0\ ayb \succ xx\ q_1\ yb \succ xxy\ q_1\ b$$

Notazione equivalente: $q_2\ xayb \overset{*}{\succ} xxy\ q_1\ b$

configurazione Iniziale : $q_0 \; w$

stringa di input



il linguaggio accettato

Per ogni macchina Turing M

$$L(M) = \{w : \quad q_0\, w \overset{*}{\succ} x_1\, q_f\, x_2\}$$

Accettato in forma standard

q_o w da* w 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

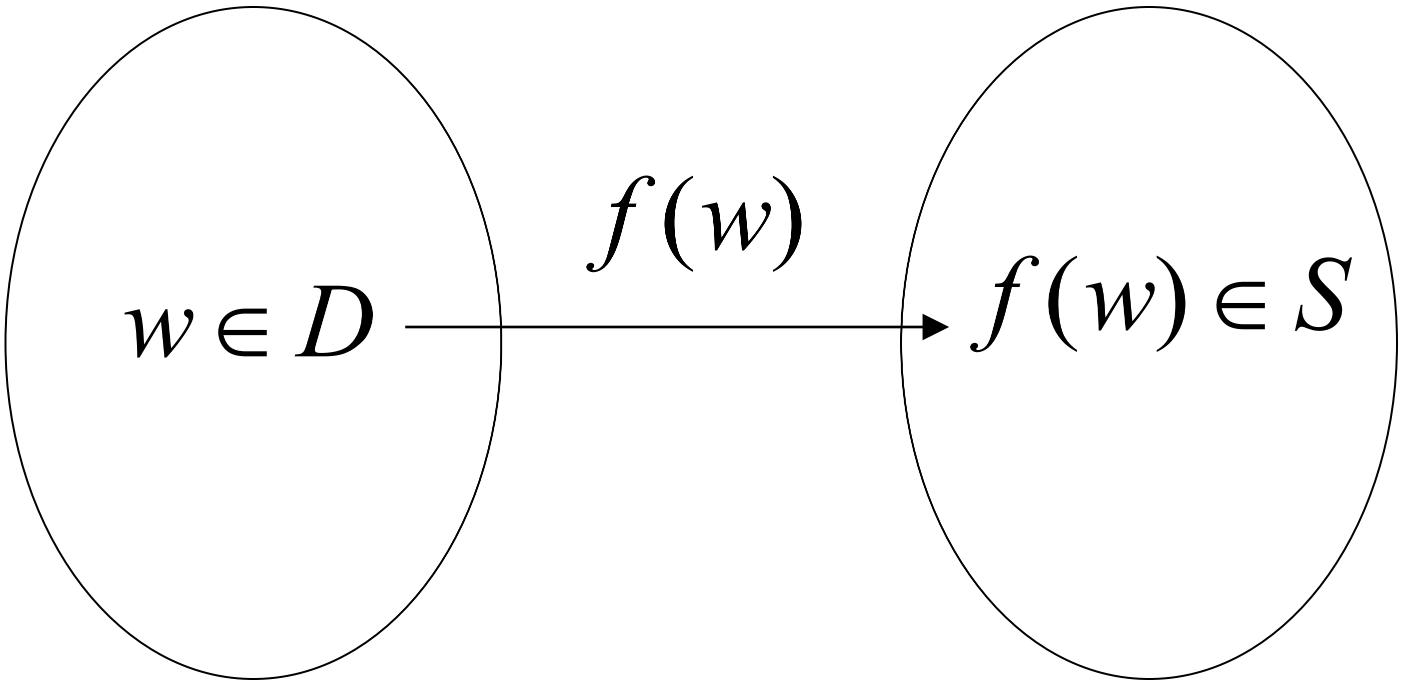
Non raggiunge uno stato finale. Ma questo non vuol dire che la macchina si ferma

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

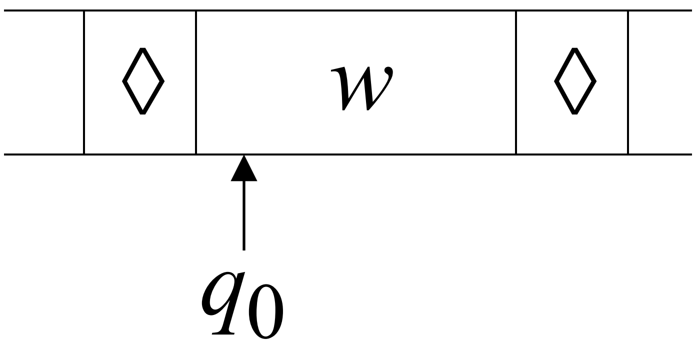
Useremo rapresentazione **unaria** :

Più facile da usare con le macchine di Turing

Definizione:

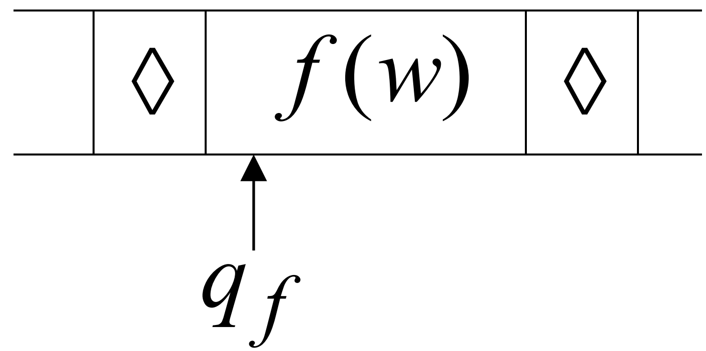
Una funzione f è calcabile se
vi è una macchina di Turing M tale che:

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:

$$q_0\, w \overset{*}{\succ} q_f\, f(w)$$

configurazione iniziale

configurazione Finale

Per tutte $w \in D$ dominio

esempio

la funzione $f(x, y) = x + y$ è calcolabile

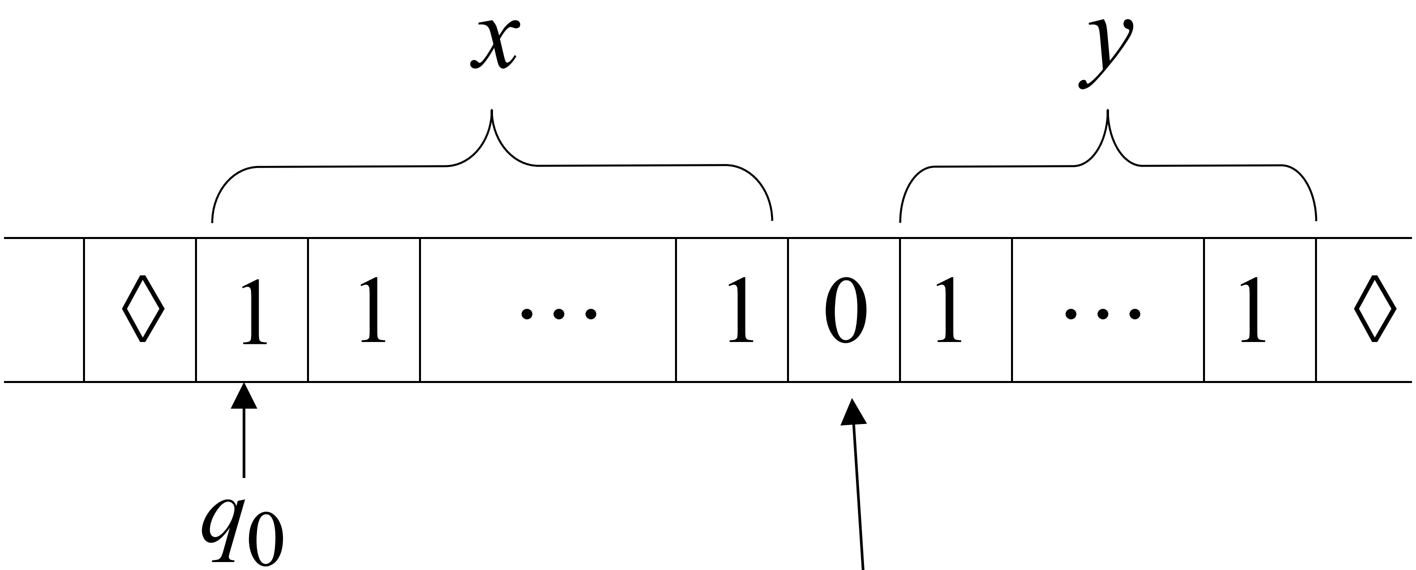
x, y Sono interi

macchina Turing :

stringa di input: $x0y$ unario

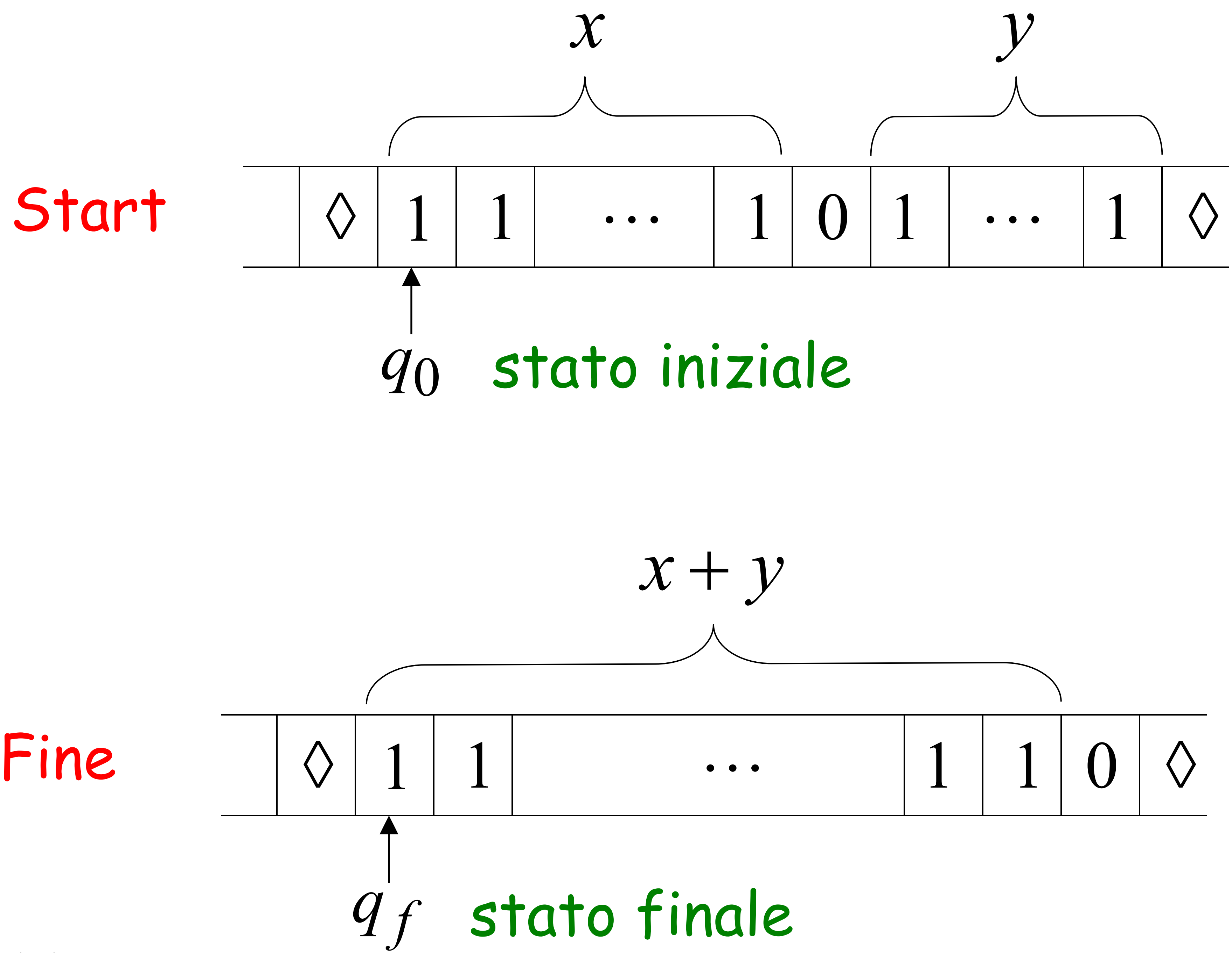
Output stringa: $xy0$ unario

Start



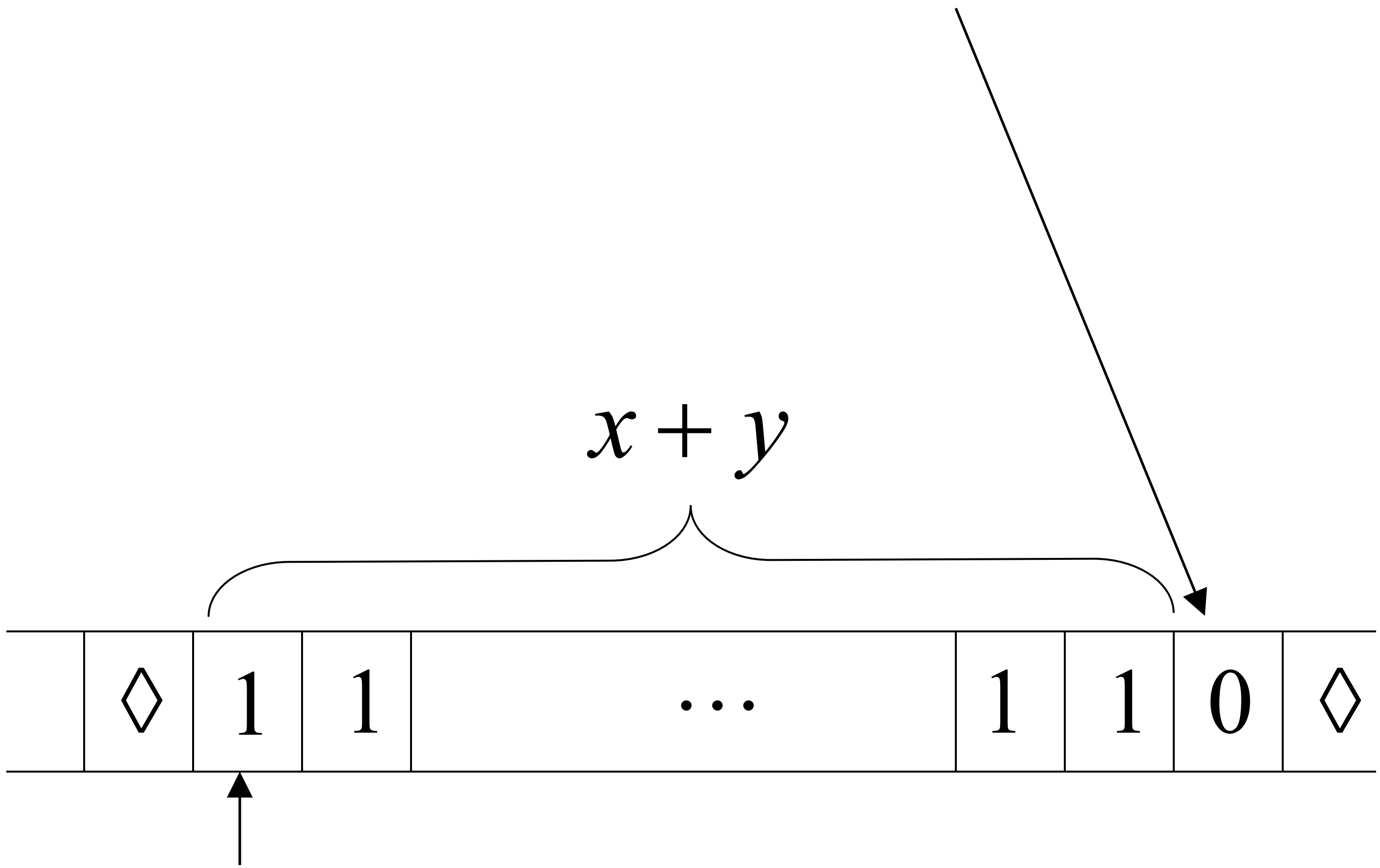
iniziale stato

il 0 è il delimitatore che
Separa I due numeri


x
y
Start
◊ 1 1 ⋯ 1 0 1 ⋯ 1 ◊
q_0 stato iniziale
$x+y$
Fine
◊ 1 1 ⋯ 1 1 0 ◊
q_f stato finale

lo 0 ci può aiutare se usiamo
il risultato per un altra operazione

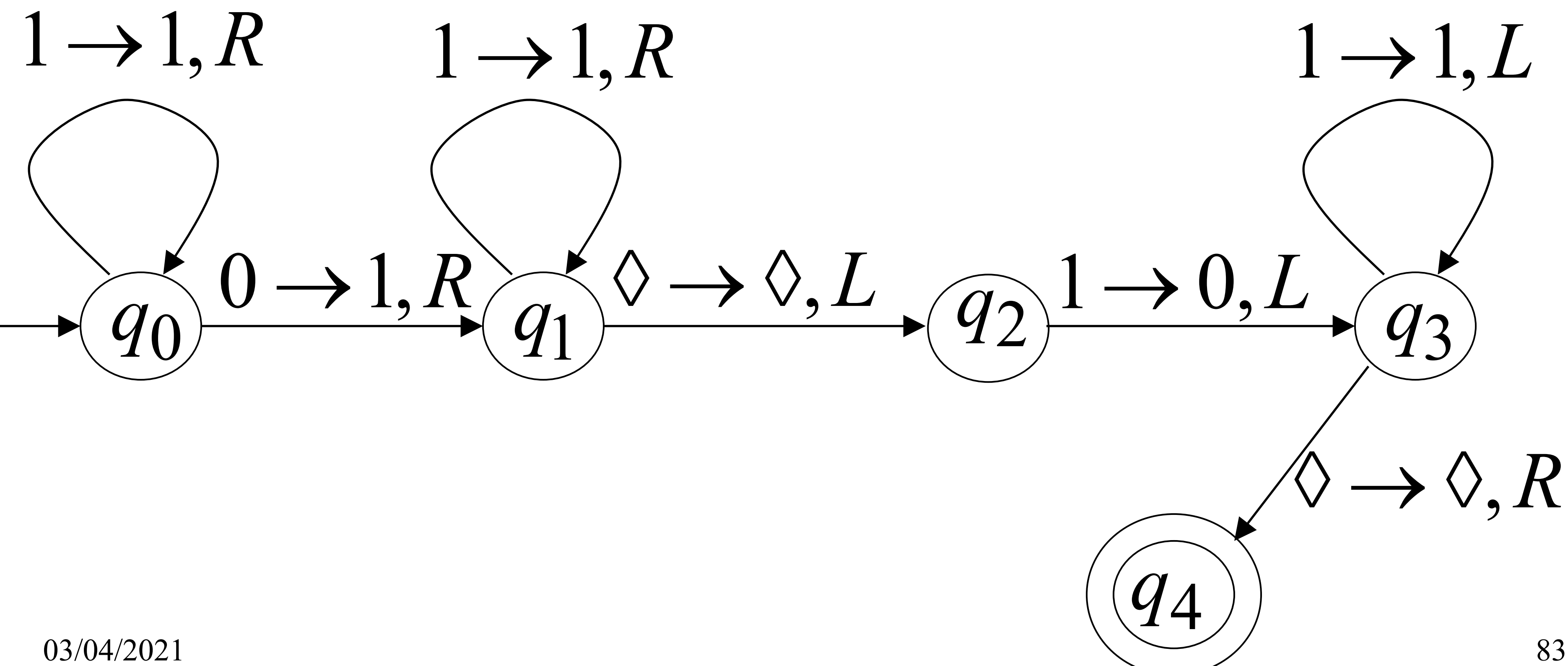
Fine



q_f stato finale

macchina Turing per la funzione $f(x,y)=x+y$

Ricordarsi di eliminare due 1 alla fine



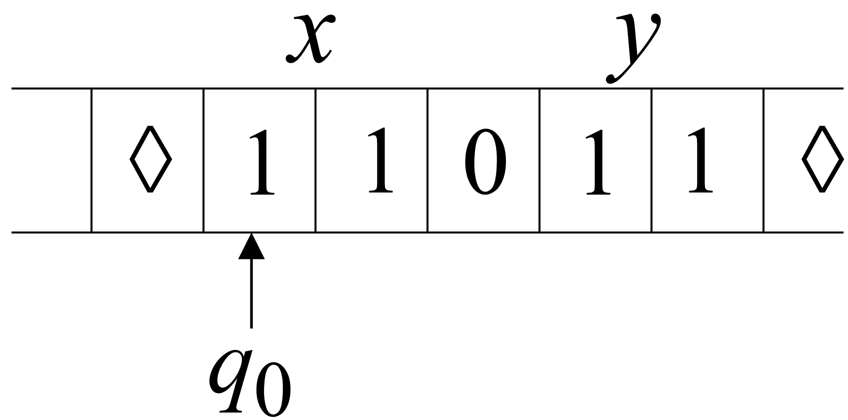
Consideriamo i numeri naturali
senza lo zero
Quindi basta avere n= 1alla n

esempio di esecuzione:

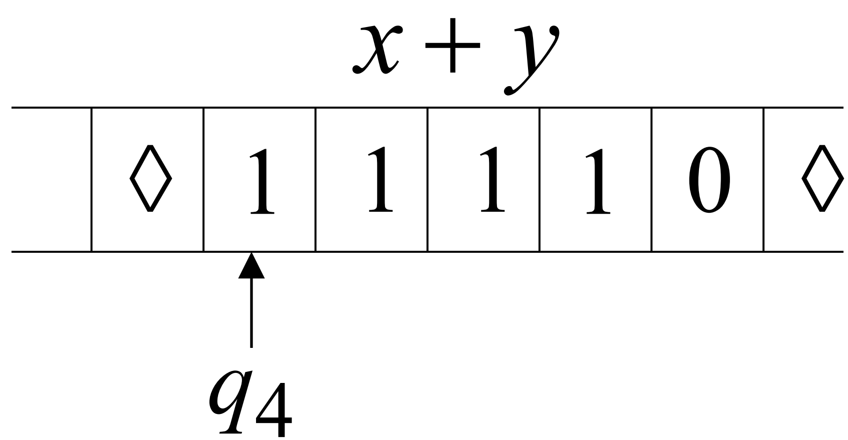
$x = 11$ (=2)

$y = 11$ (=2)

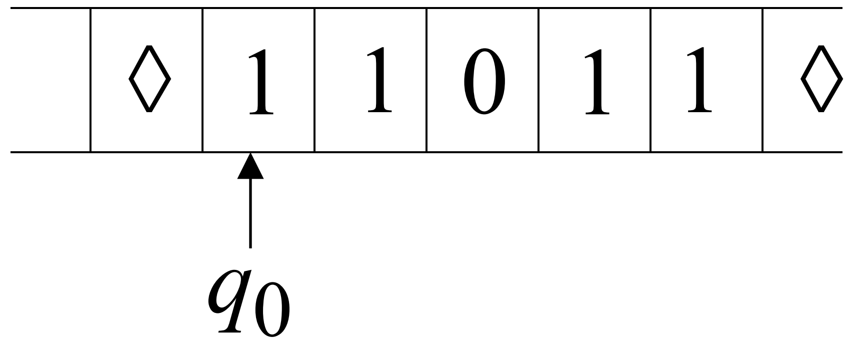
Time 0

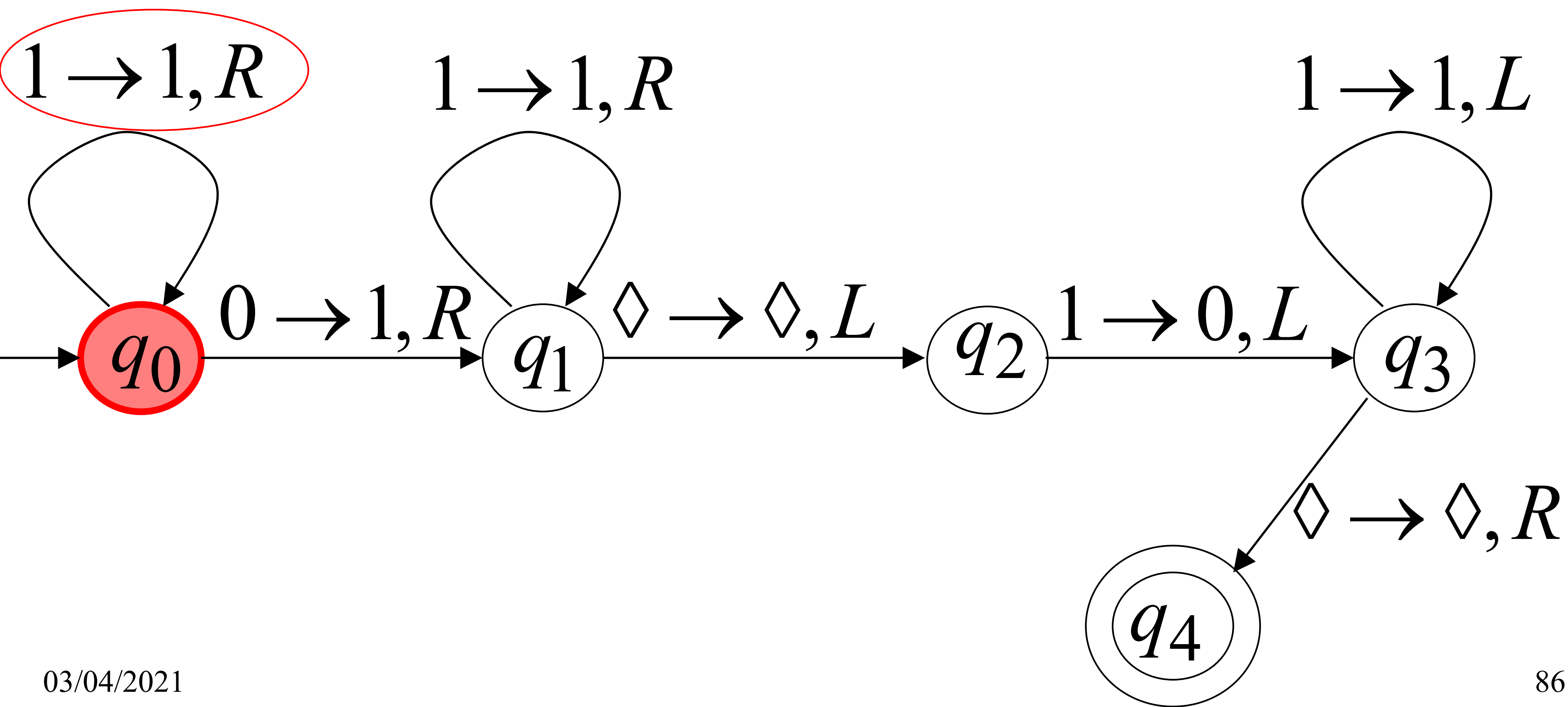


Final Result

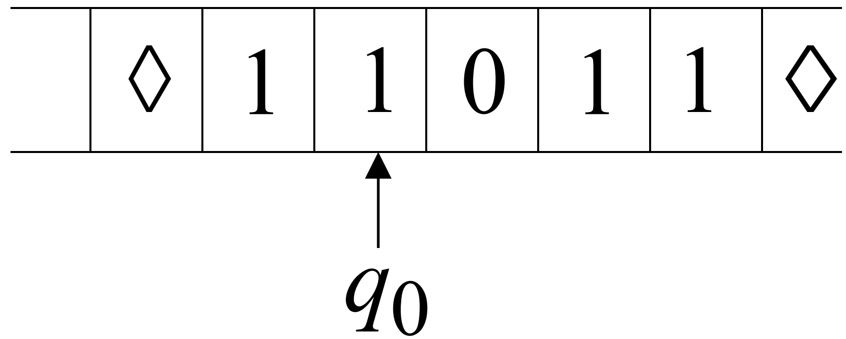


Time 0

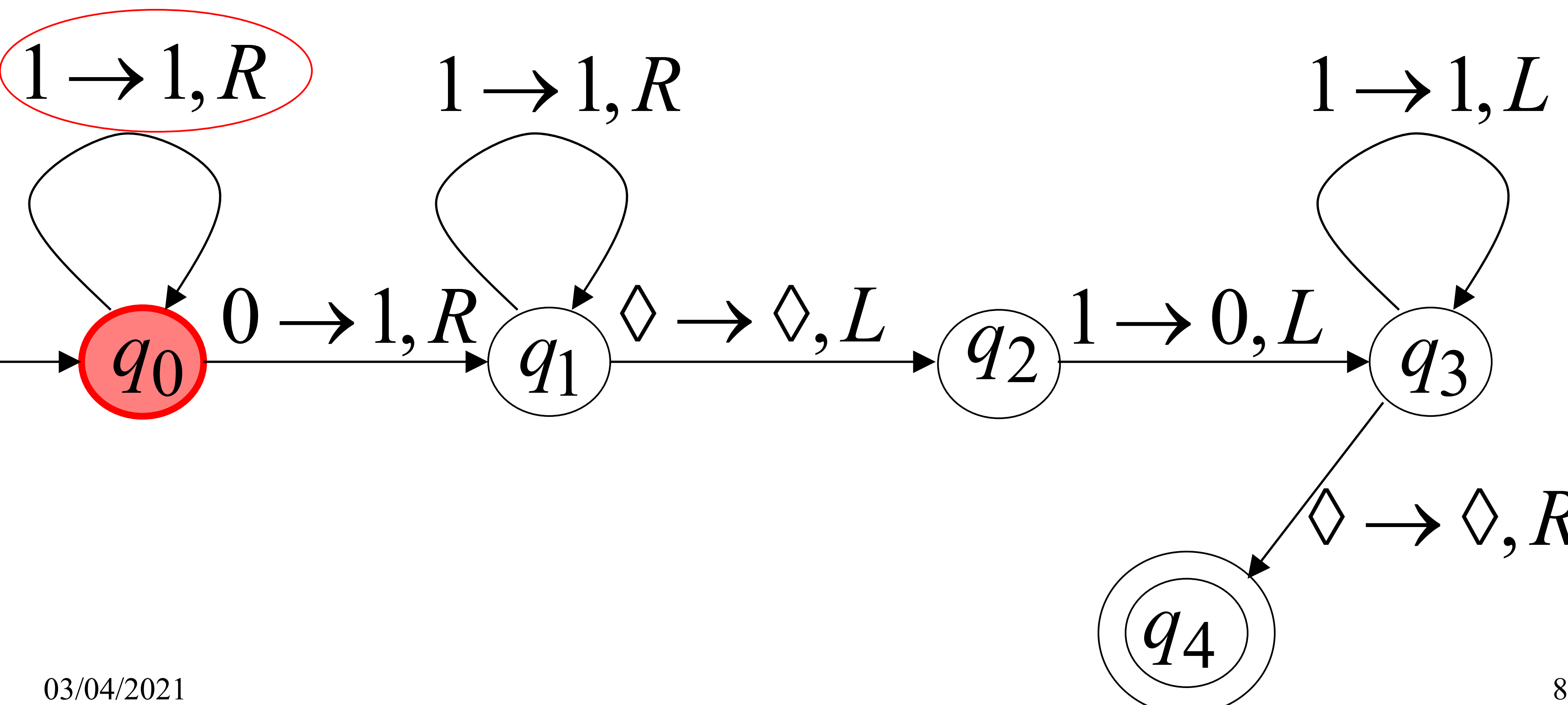

◊ 1 1 0 1 1 ◊
q_0


$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$\Diamond \rightarrow \Diamond, L$
q_2
$1 \rightarrow 0, L$
q_3
$\Diamond \rightarrow \Diamond, R$
q_4

Time 1

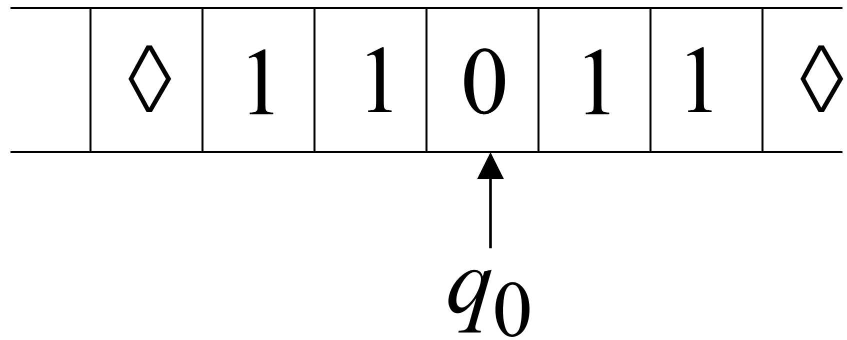


◊ 1 1 0 1 1 ◊
q_0

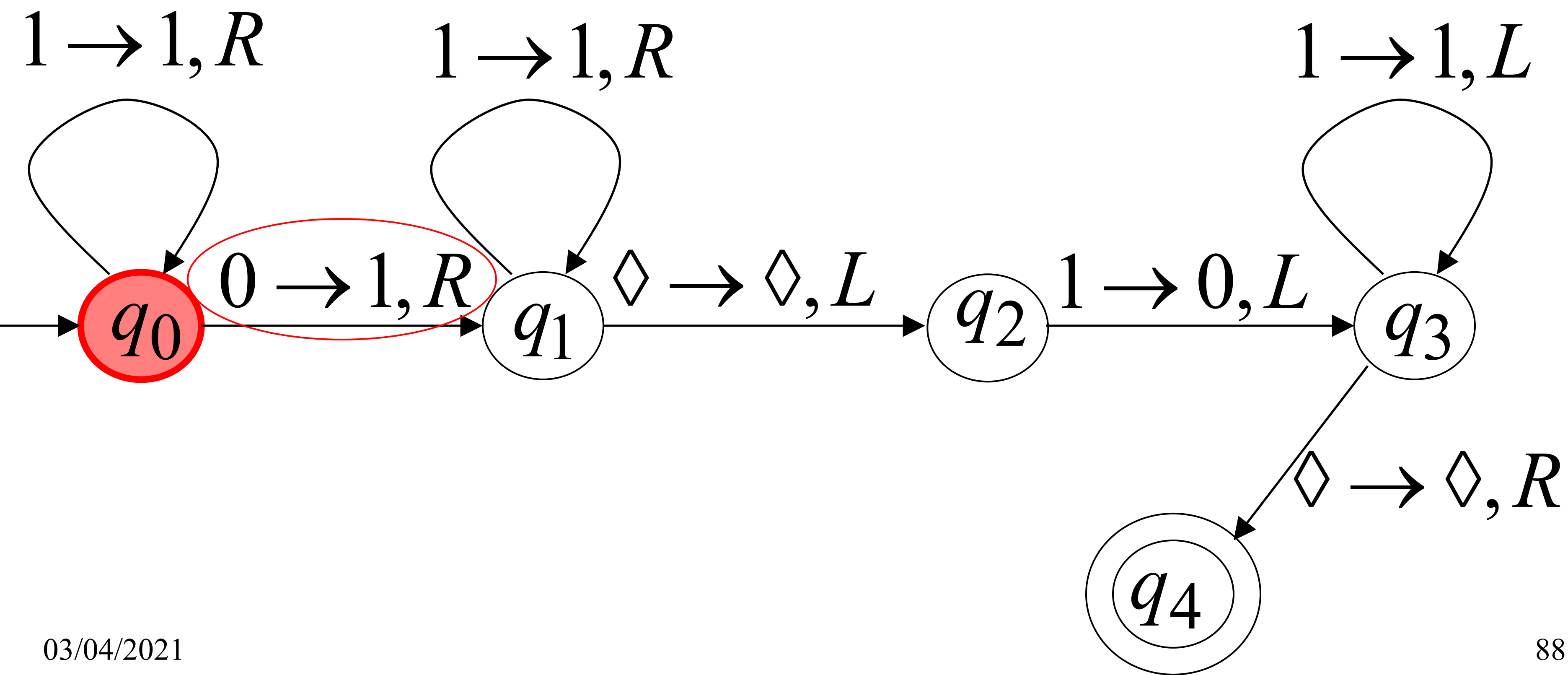


$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$\Diamond \rightarrow \Diamond, L$
q_2
$1 \rightarrow 0, L$
q_3
$\Diamond \rightarrow \Diamond, R$
q_4

Time 2

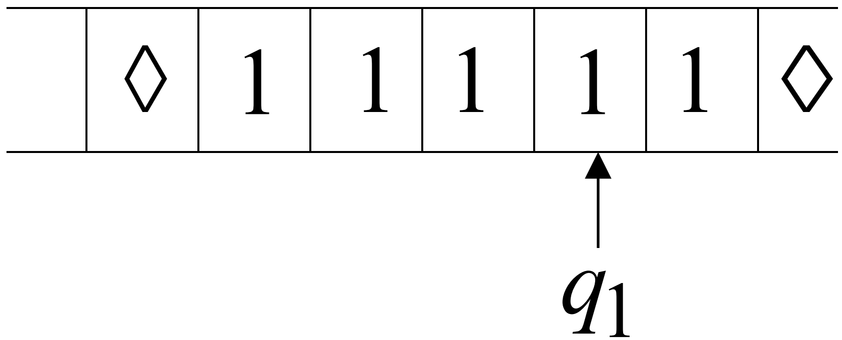


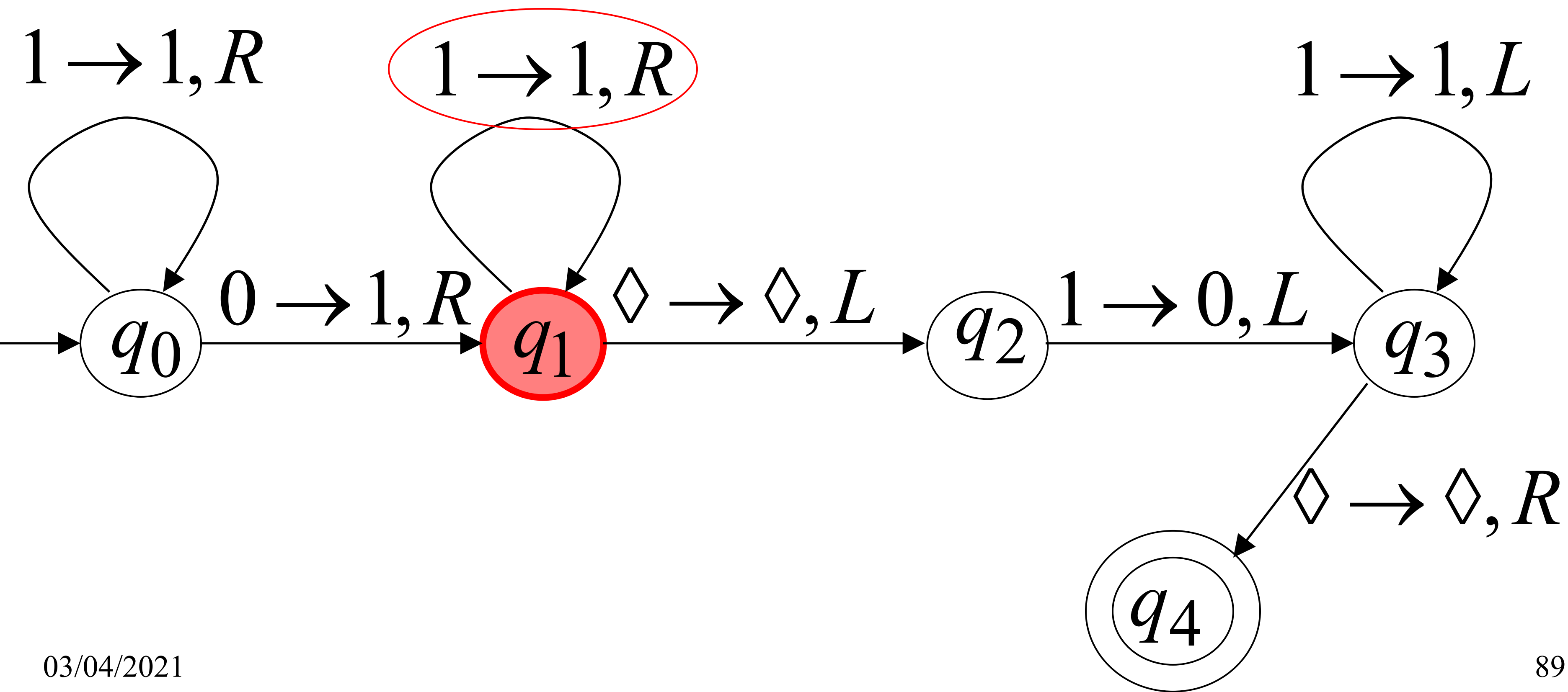
◊ 1 1 0 1 1 ◊
q_0



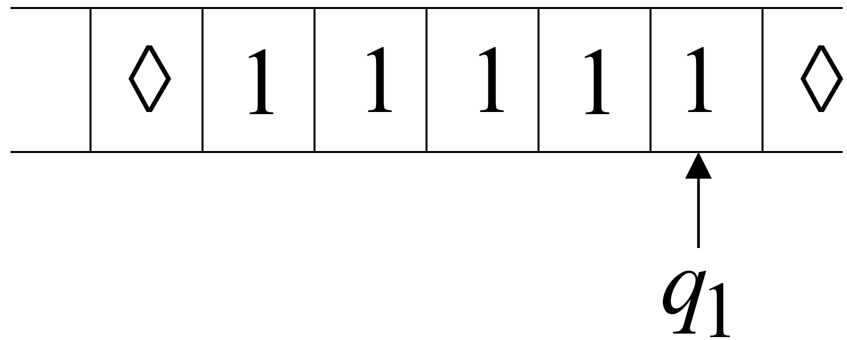
$1 \to 1, R$
$1 \to 1, R$
$1 \to 1, L$
$0 \to 1, R$
$\Diamond \to \Diamond, L$
$1 \to 0, L$
$\Diamond \to \Diamond, R$
q_0
q_1
q_2
q_3
q_4

Time 3

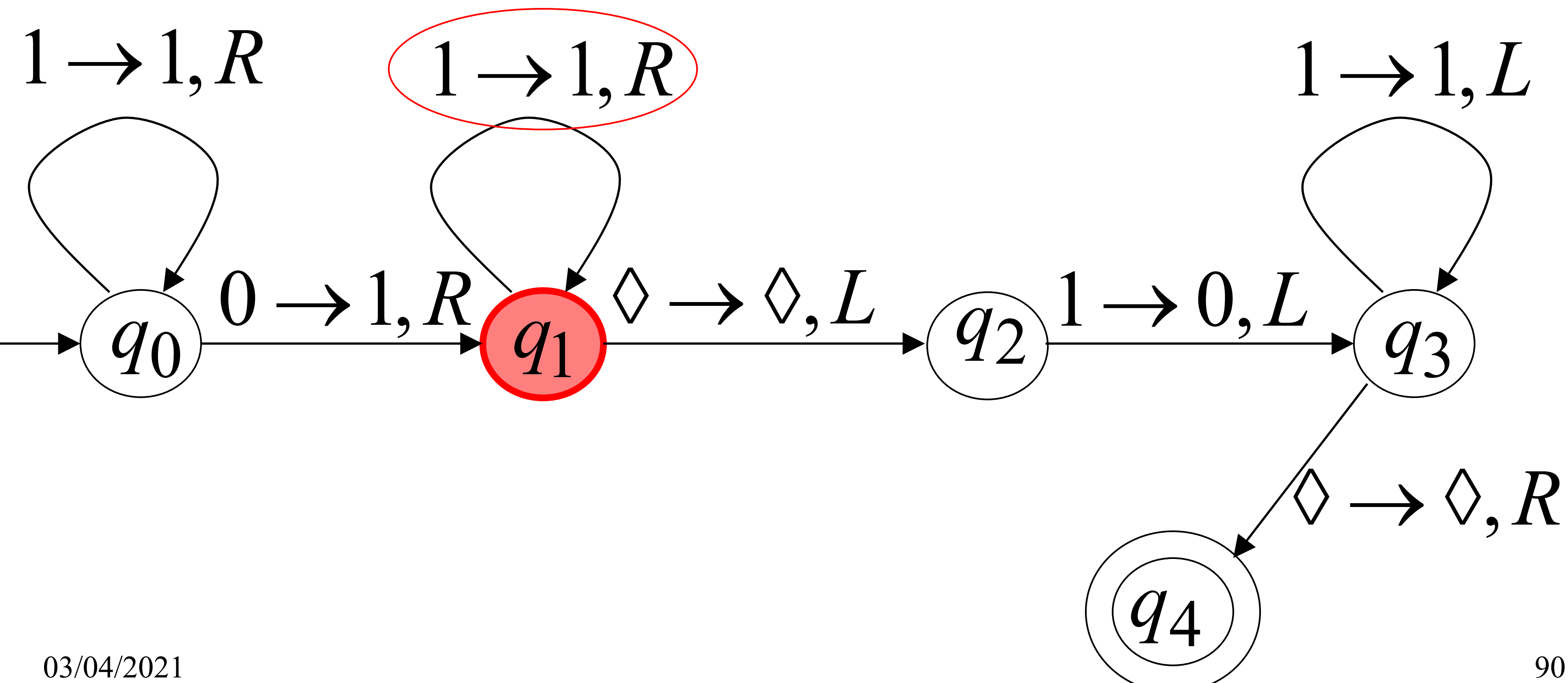

◊ 1 1 1 1 1 ◊
q_1


$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
$0 \rightarrow 1, R$
$\lozenge \rightarrow \lozenge, L$
$1 \rightarrow 0, L$
$\lozenge \rightarrow \lozenge, R$
q_0
q_1
q_2
q_3
q_4

Time 4

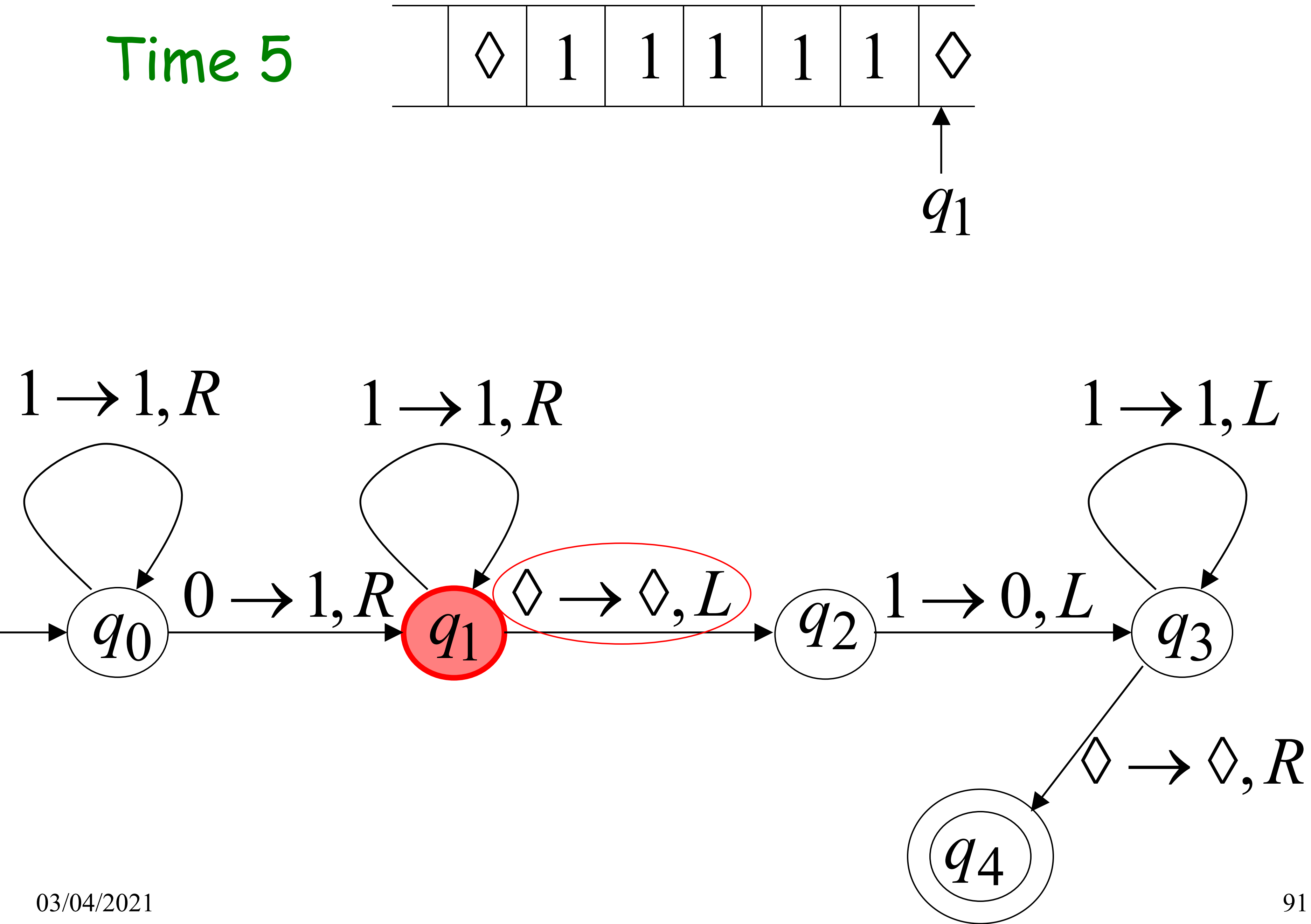


◊
1
1
1
1
1
◊
q_1

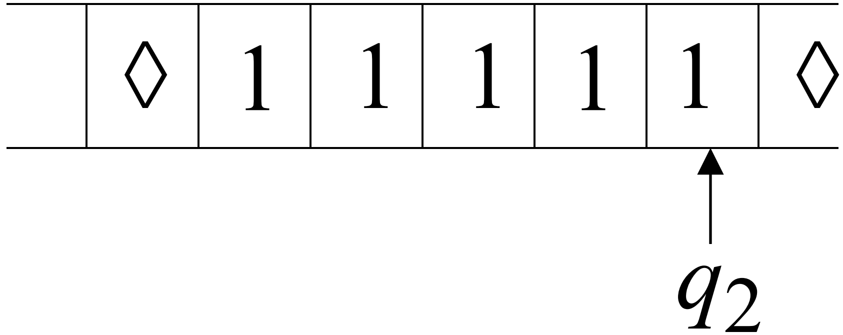


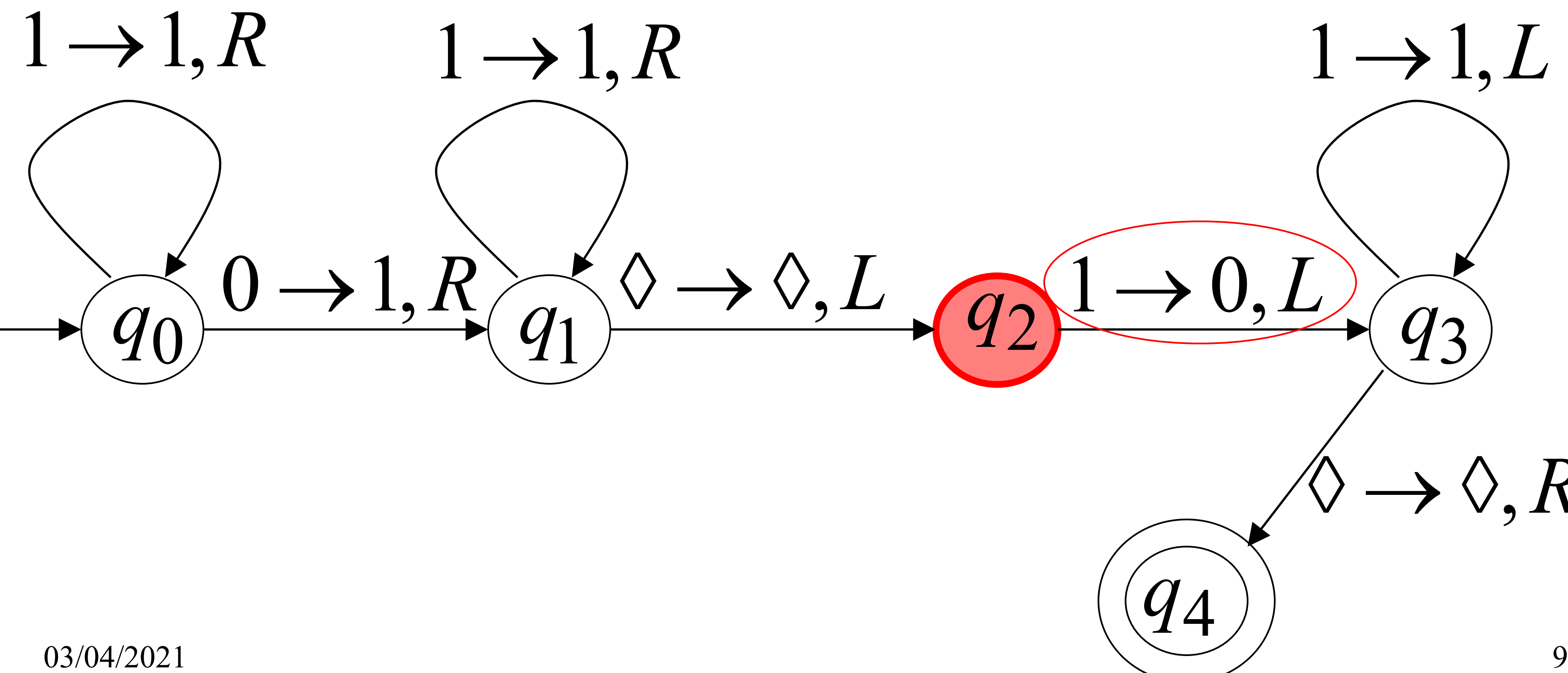
$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
$0 \rightarrow 1, R$
$\lozenge \rightarrow \lozenge, L$
$1 \rightarrow 0, L$
$\lozenge \rightarrow \lozenge, R$
q_0
q_1
q_2
q_3
q_4

Time 5

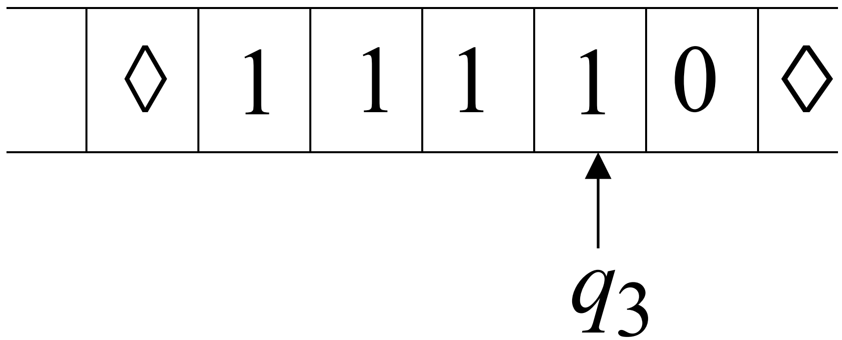

◊
1
1
1
1
1
◊
q_1
$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
$0 \rightarrow 1, R$
$\Diamond \rightarrow \Diamond, L$
$1 \rightarrow 0, L$
$\Diamond \rightarrow \Diamond, R$
q_0
q_1
q_2
q_3
q_4

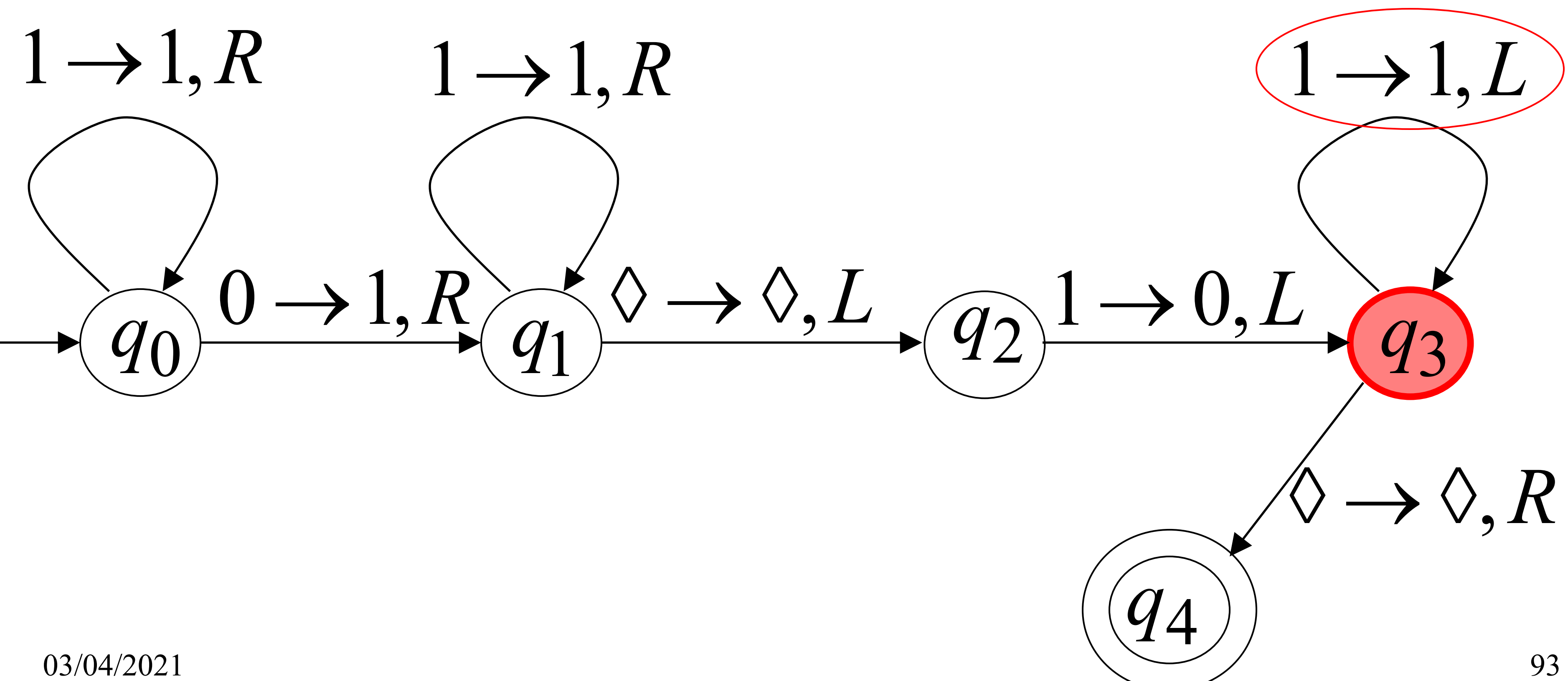
Time 6


◊
1
1
1
1
1
◊
q_2

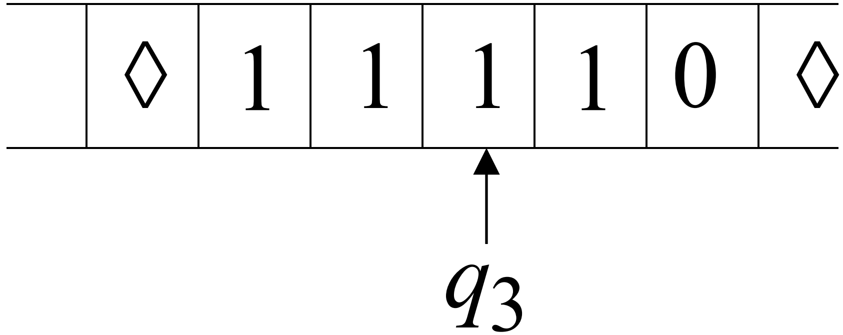

$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$\Diamond \rightarrow \Diamond, L$
q_2
$1 \rightarrow 0, L$
q_3
$\Diamond \rightarrow \Diamond, R$
q_4

Time 7

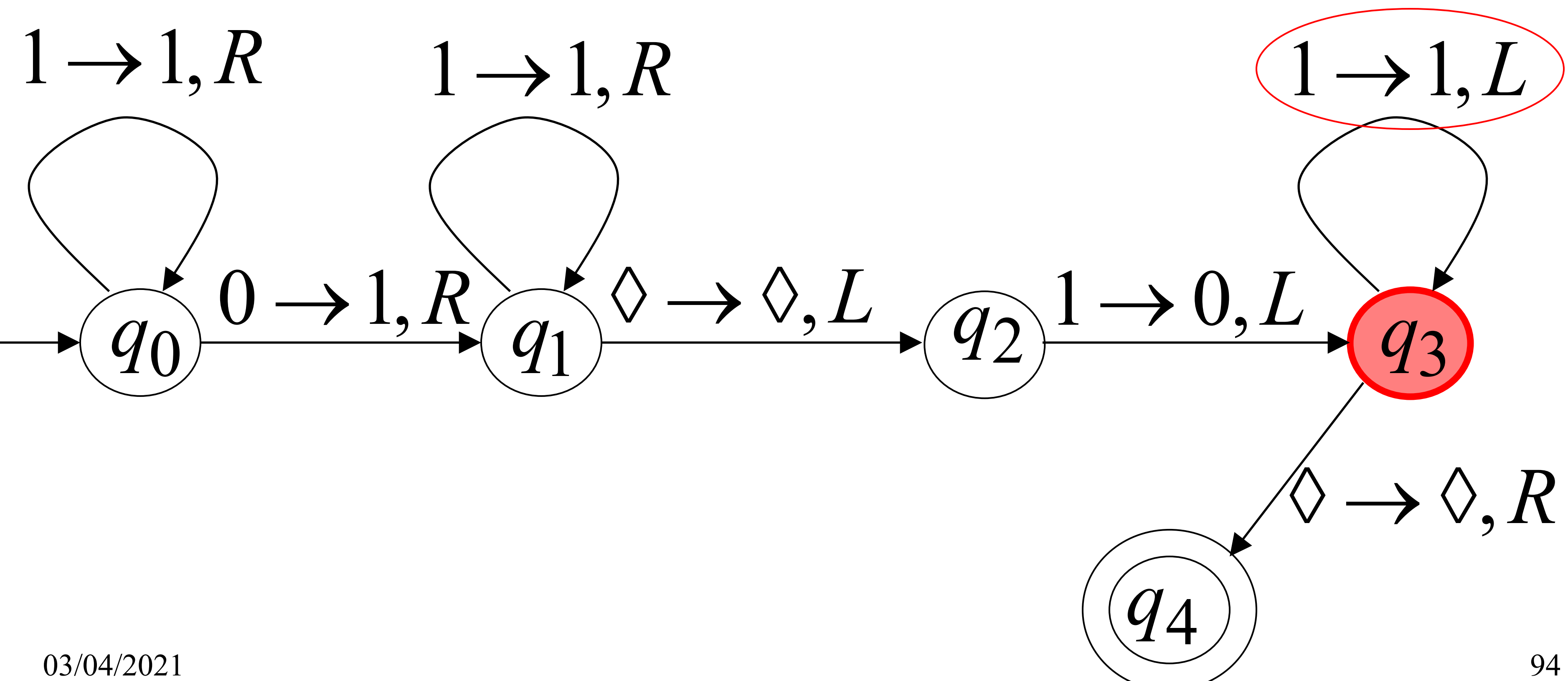

◊ 1 1 1 1 0 ◊
q_3


$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$\Diamond \rightarrow \Diamond, L$
q_2
$1 \rightarrow 0, L$
q_3
$\Diamond \rightarrow \Diamond, R$
q_4

Time 8

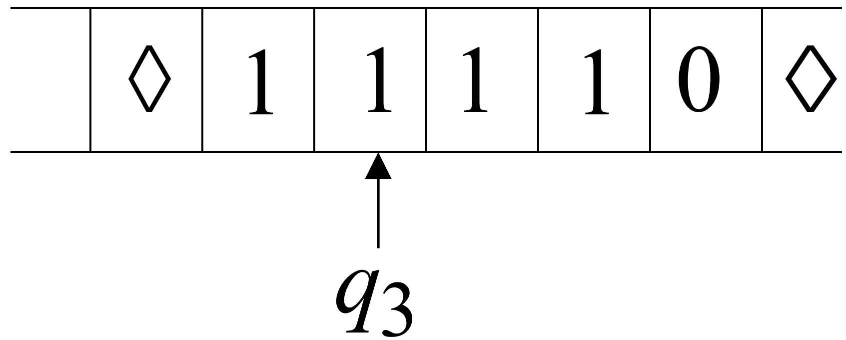


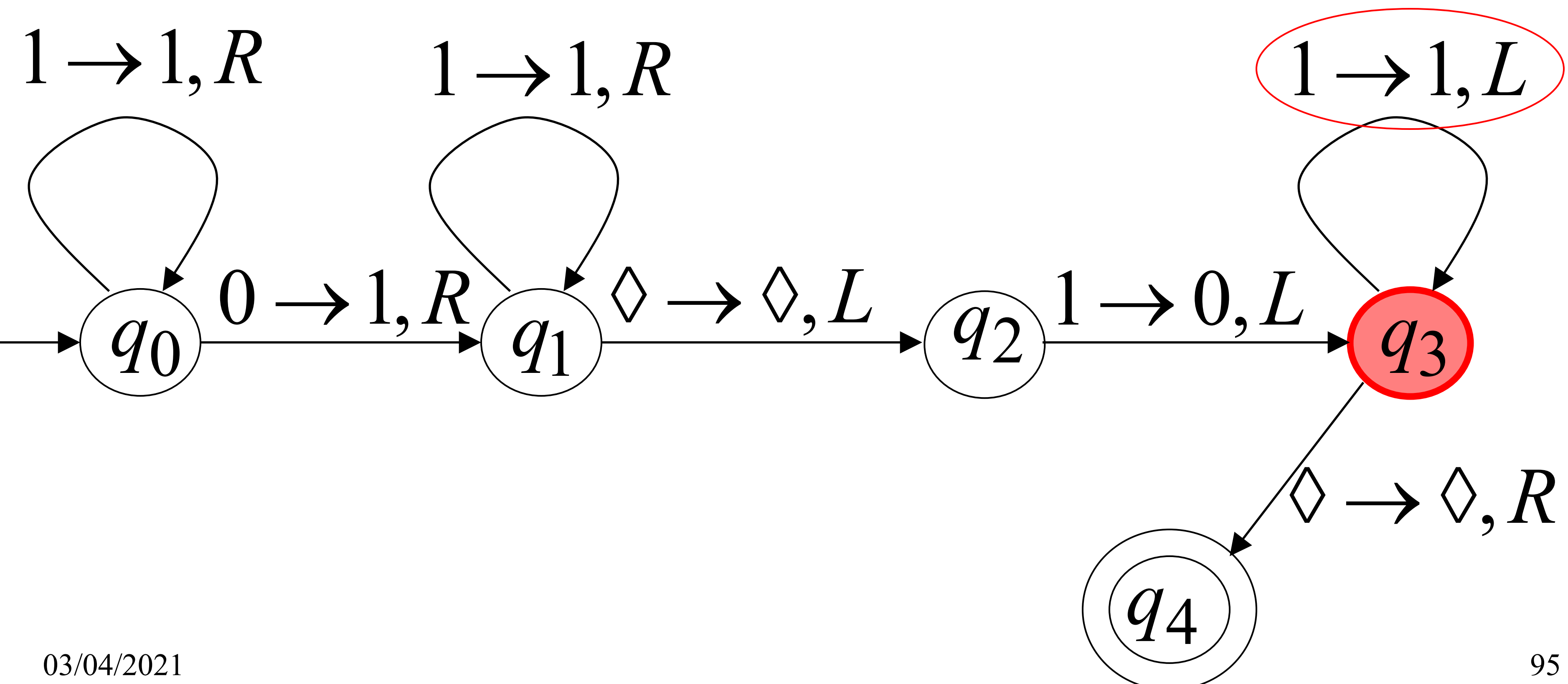
◊
1
1
1
1
0
◊
q_3



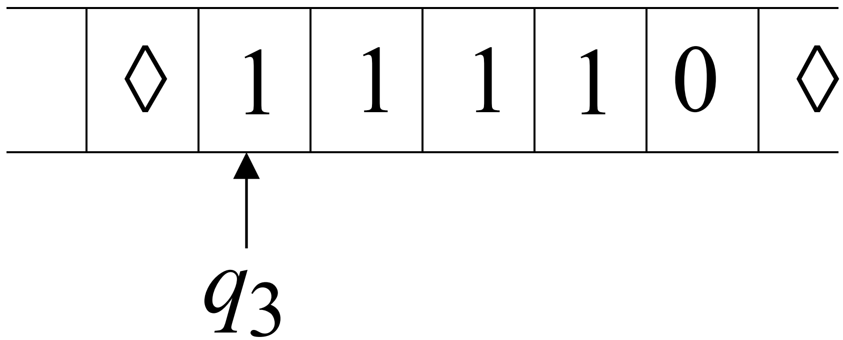
$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
q_0
$0 \rightarrow 1, R$
q_1
$\lozenge \rightarrow \lozenge, L$
q_2
$1 \rightarrow 0, L$
q_3
$\lozenge \rightarrow \lozenge, R$
q_4

Time 9

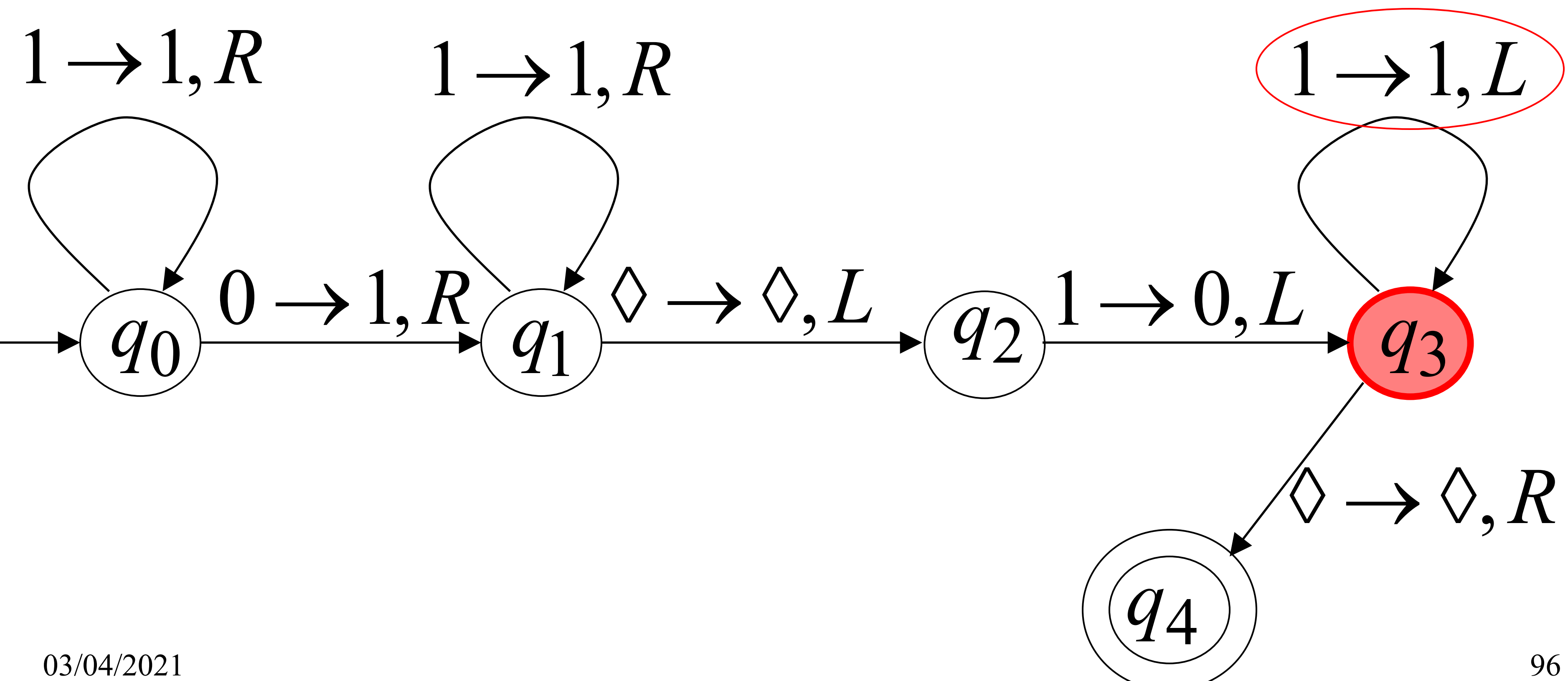

◊
1
1
1
1
0
◊
q_3


$1 \to 1, R$
$1 \to 1, R$
$1 \to 1, L$
$0 \to 1, R$
$\lozenge \to \lozenge, L$
$1 \to 0, L$
$\lozenge \to \lozenge, R$
q_0
q_1
q_2
q_3
q_4

Time 10

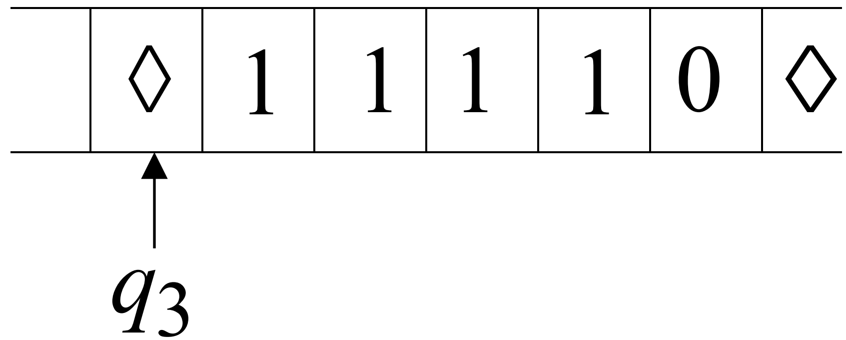


◊
1
1
1
1
0
◊
q_3

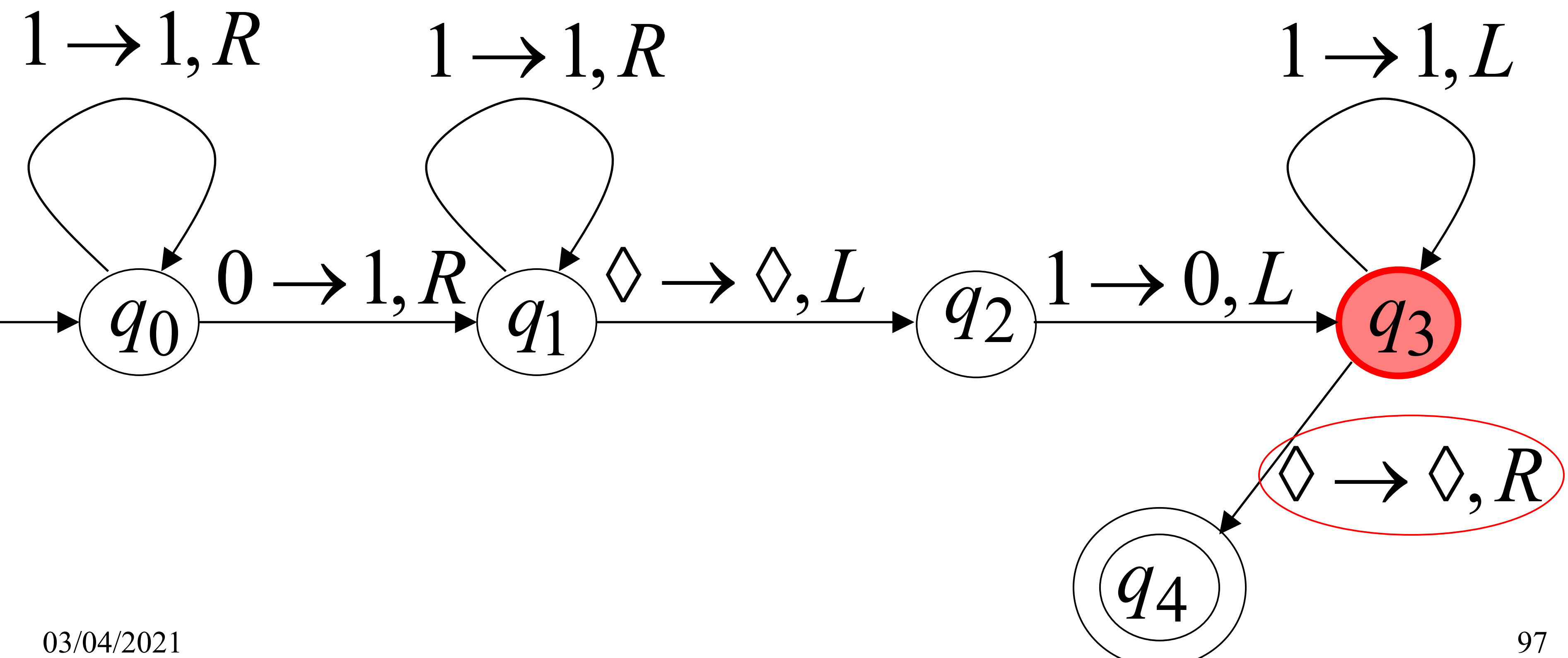


$1 \to 1, R$
$1 \to 1, R$
$1 \to 1, L$
$0 \to 1, R$
$\lozenge \to \lozenge, L$
$1 \to 0, L$
$\lozenge \to \lozenge, R$
q_0
q_1
q_2
q_3
q_4

Time 11

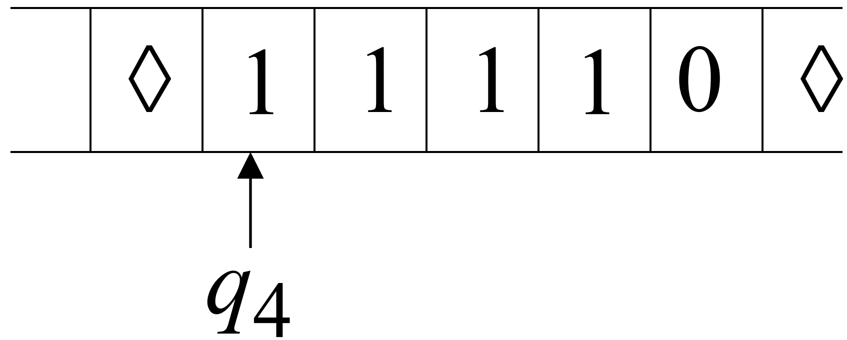


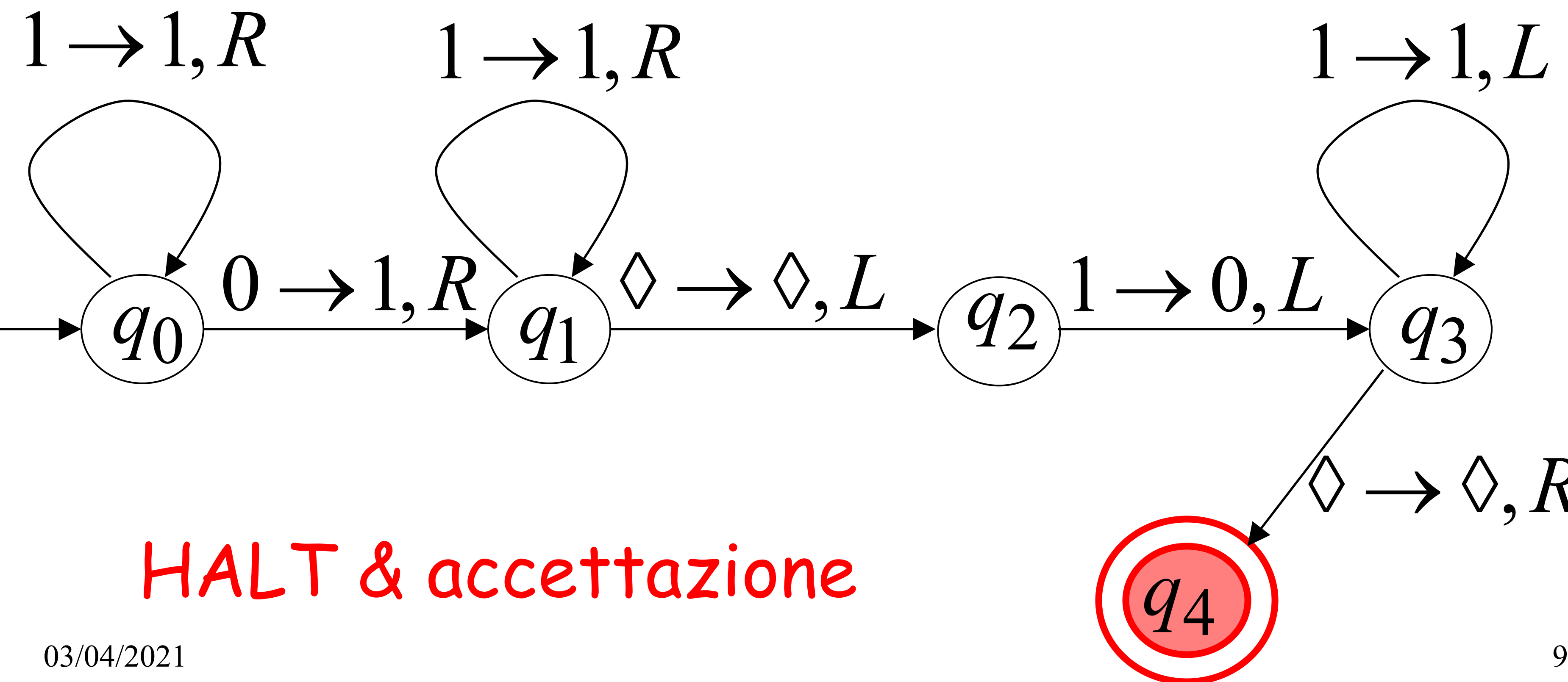
◊ 1 1 1 1 0 ◊
q_3



$1 \rightarrow 1, R$
$1 \rightarrow 1, R$
$1 \rightarrow 1, L$
$0 \rightarrow 1, R$
$\Diamond \rightarrow \Diamond, L$
$1 \rightarrow 0, L$
$\Diamond \rightarrow \Diamond, R$
q_0
q_1
q_2
q_3
q_4

Time 12





HALT & accettazione

Un altro esempio

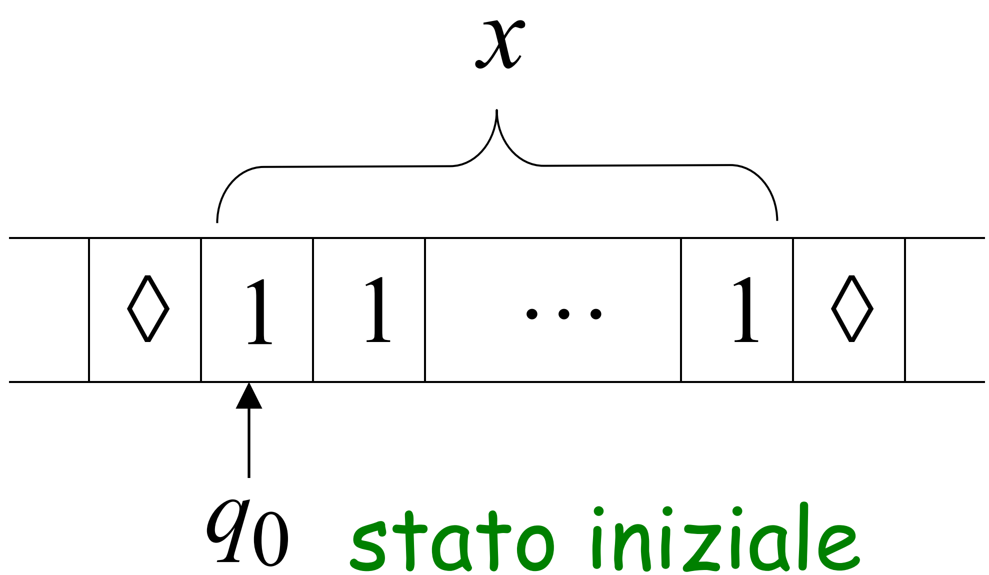
La funzione Che raddoppia il numero di 1 è calcolabile

Macchina di Turing :

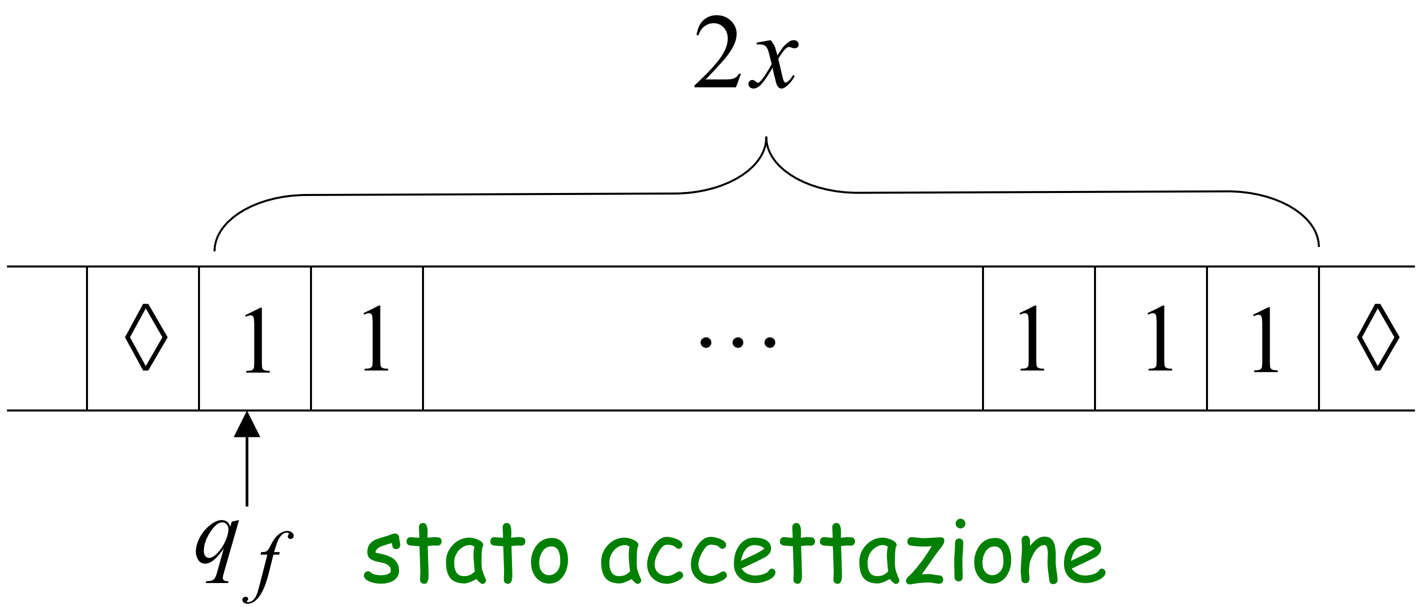
stringa di input: x unario

Output string: xx unario

Start



Finish

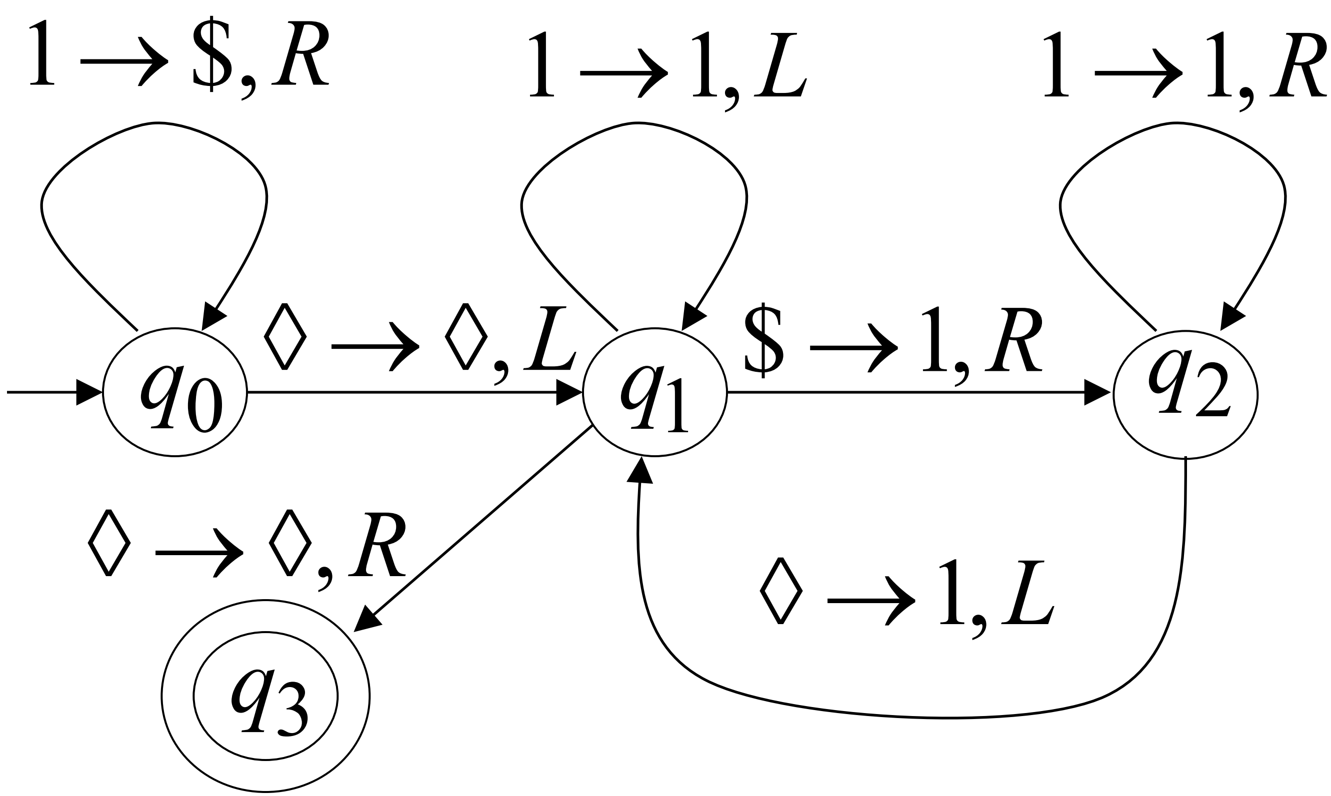


macchina Turing Pseudocodice per $f(x) = 2x$

- ogni 1 diventa $
- Repeat:
  - trova il $ più a destra, cambia in 1
  - vai alla fine a destra, inserisci 1

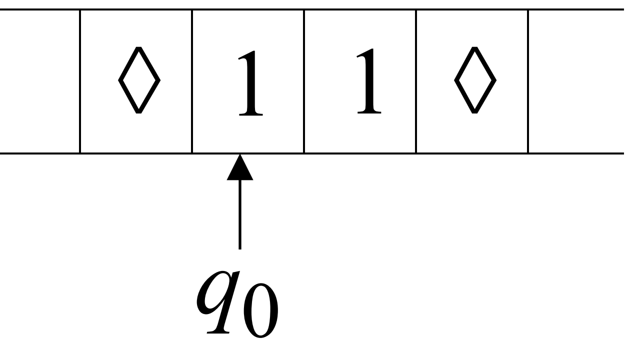
Until no $ rimangono

Turing macchina per $f(x) = xx$

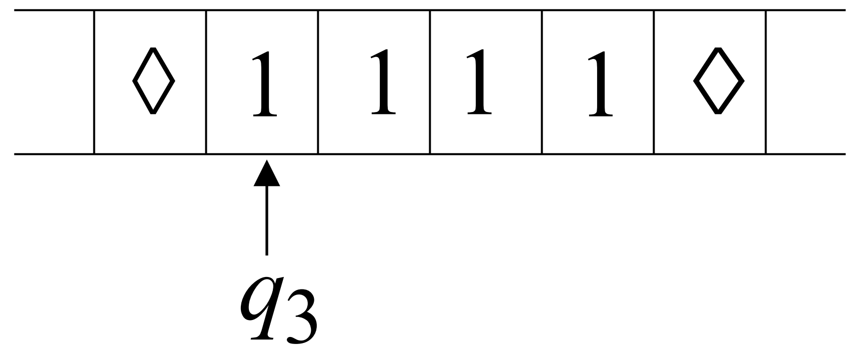


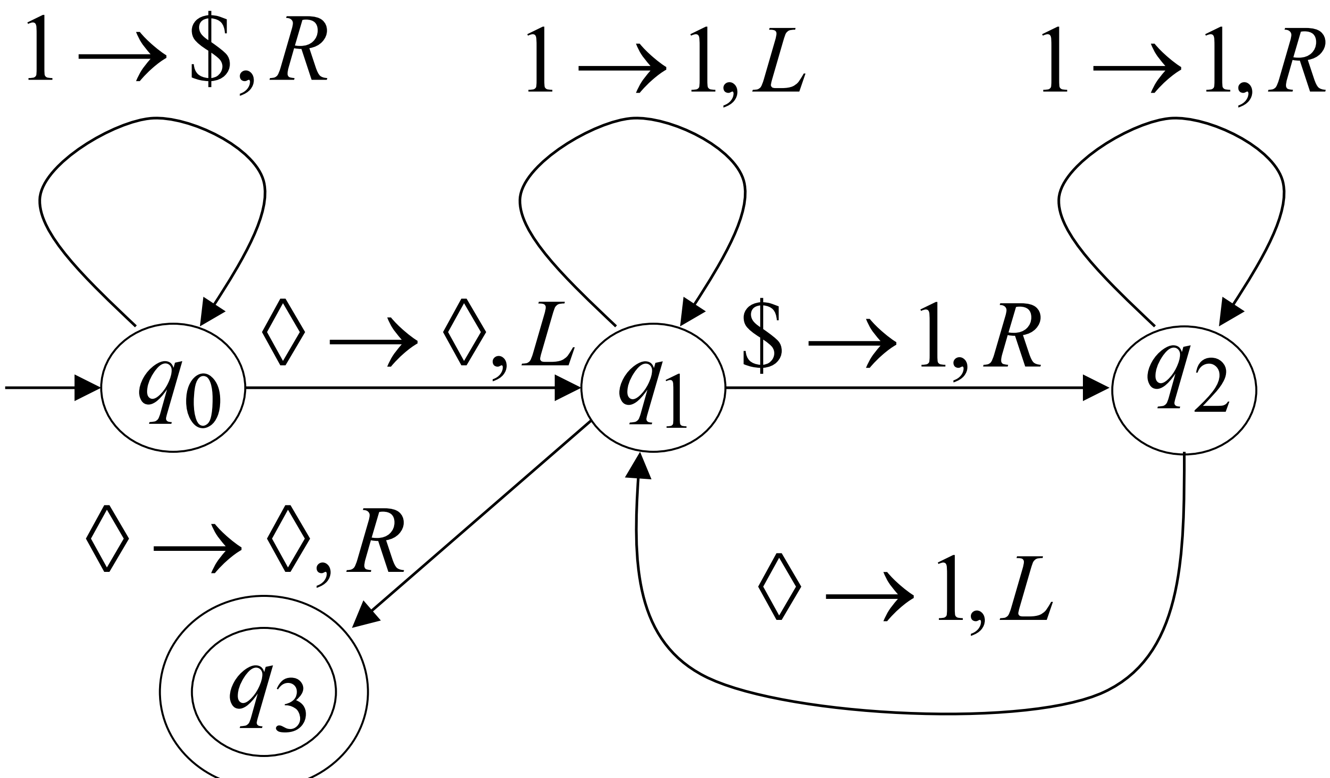
esempio

Start



Finish





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

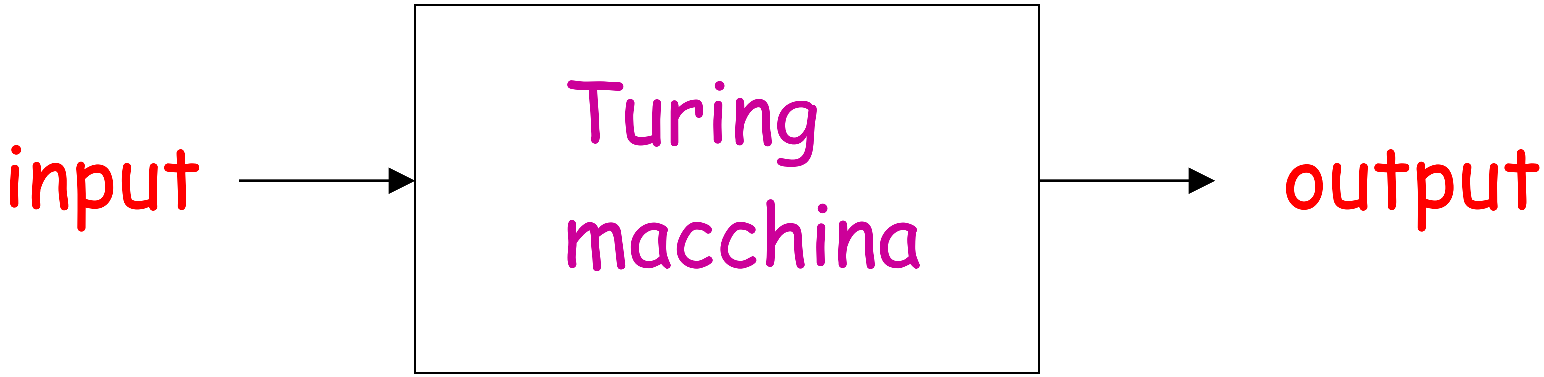
cancella tape, scrivi 1 $(x > y)$

else

cancella tape, scrivi 0 $(x \leq y)$

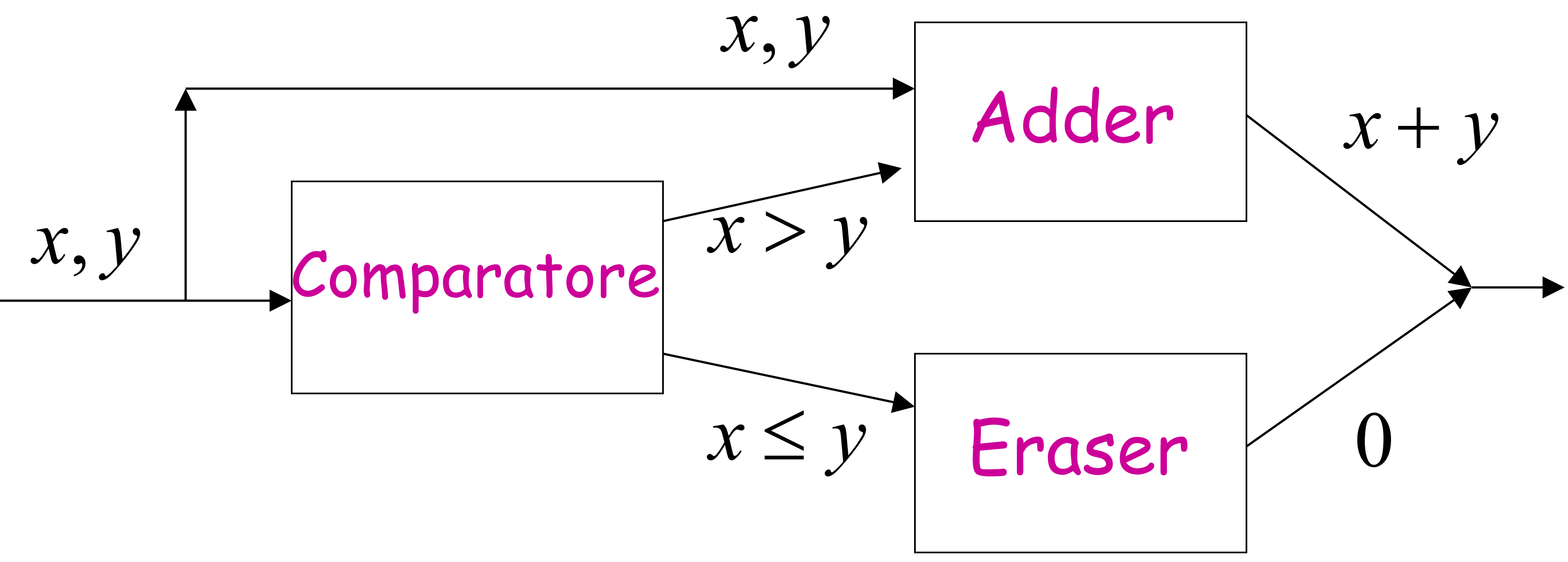
Mettere insieme macchine di turing

Block Diagram



esempio:

$$f(x,y) = \begin{cases} x+y & \text{if } x>y \\ 0 & \text{if } x \le y \end{cases}$$



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

$$\Sigma, \not{b}, K, \delta, q_0, q_f$$

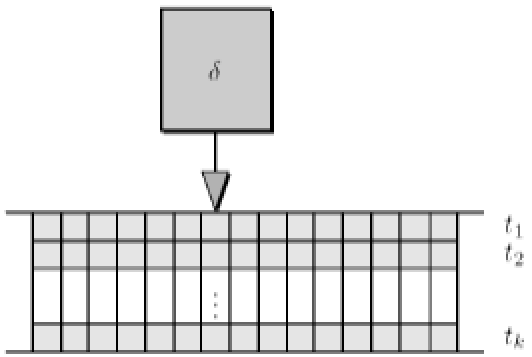
ove la fuzione δ è definita come

$$\delta_m : (K - \{q_f\}) \times \Sigma_{\not{b}}^m \rightarrow K \times \Sigma_{\not{b}}^m \times \{d, s, i\}$$

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 anzichè operare su **simboli scalari** opera su **simboli vettoriali**.

Data una macchine di turing $\mathcal{M}_m = (\Sigma, \not{b}, K, \delta^m, q_0, q_f)$ multitraccia con m tracce si ha che:

- l'alfabeto $\Sigma = \Sigma_1 \times \ldots \times \Sigma_m$, ove ogni Σ_i rappresenta l'alfabeto di simboli della traccia i.

  Quindi un generico elemento (carattere) $\sigma \in \Sigma$ sarà del tipo: $\sigma = (\sigma_1, \ldots, \sigma_m)$, $1 \leq i \leq m$ $\sigma_i \in \Sigma_i$.

  In particolare il simbolo $\underbrace{(\not{b}, \ldots, \not{b})}_{m}$ rappresenta il simbolo di blank dell'alfabeto $\Sigma_{\not{b}}$.

  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 $\sigma = (\sigma_1, \ldots, \sigma_m)$ 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:

$$\delta^m : (K - \{q_f\}) \times \Sigma_{\not{b}} \rightarrow K \times \Sigma_{\not{b}} \times \{d, s, i\}, \quad \delta^m(q_i, \sigma) = (q_j, \sigma'), \quad \sigma, \sigma' \in \Sigma_{\not{b}}$$

Equivalenza MT-multi-traccia e MT-singola-traccia

Teorema Una Macchine 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 $\mathcal{M}^m = (\Sigma, \not{b}, K, \delta^m, q_0, q_f)$ la macchina di Turing multitraccia, ove $\Sigma = \Sigma_1 \times \ldots \times \Sigma_m$. Si definisca una MT singola traccia $\mathcal{M} = (\Lambda, \not{b}, K', \delta, q'_0, q'_f)$ tale che:

$$|\Lambda| = |\Sigma_1| \times \ldots \times |\Sigma_m|$$

cioè, la cardinalità dell'alfabeto Λ è pari al prodotto delle cardinalità degli alfabeti delle singole tracce.

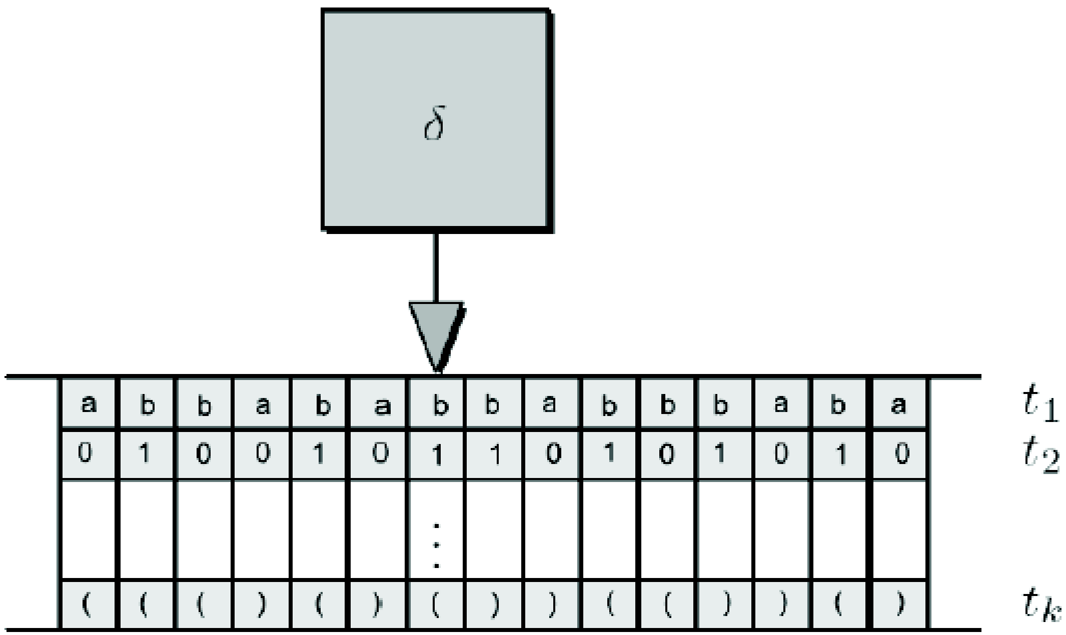
Si definisca inoltre, una funzione **iniettiva** $\varphi : \Sigma \to \Lambda$ che associ ad ogni simbolo di Σ un simbolo di Λ (poichè $|\Lambda| \geq |\Sigma|$ è possibile definire una tale funzione).

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

$$\Sigma, \not{b}, K, \delta_m, q_0, q_f$$

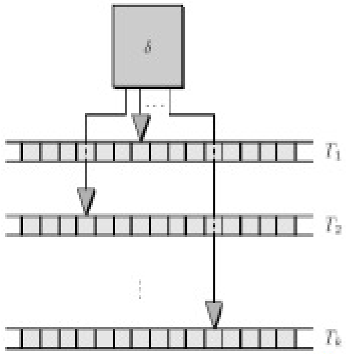
ove $\Sigma, \not{b}, K, q_0, q_f$ sono definiti come nel caso di una machina di Turing a singolo nastro e δ_m è la funzione di transizione definita come:

$$\delta_m : (K - \{q_f\}) \times \Sigma_{\not{b}}^m \rightarrow K \times \Sigma_{\not{b}}^m \times \{d, s, i\}^m$$

cioè la funzione δ_m definisce le transizione 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

$\cdots$	$\not b$	$\not b$	$\not b$	$\not b$	$\downarrow$	$\not b$	$\not b$	$\not b$	$\not b$	$\cdots$	t_1
$\cdots$	$\not b$	$\not b$	$\not b$	a	b	b	a	$\not b$	$\not b$	$\cdots$	t_2
$\cdots$	$\not b$	$\not b$	$\not b$	$\not b$	$\downarrow$	$\not b$	$\not b$	$\not b$	$\not b$	$\cdots$	t_3
$\cdots$	$\not b$	$\not b$	$\not b$	b	a	b	c	b	$\not b$	$\cdots$	t_4
$\cdots$	$\not b$	$\not b$	$\downarrow$	$\not b$	$\not b$	$\not b$	$\not b$	$\not b$	$\not b$	$\cdots$	t_5
$\cdots$	$\not b$	d	e	d	d	e	$\not b$	$\not b$	$\not b$	$\cdots$	t_6
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\cdots$	$\not b$	$\not b$	$\not b$	$\not b$	$\not b$	$\downarrow$	$\not b$	$\not b$	$\not b$	$\cdots$	t_{2k-1}
$\cdots$	f	f	g	h	g	$\not b$	$\not b$	$\not b$	$\not b$	$\cdots$	t_{2k}

ure 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 $\mathcal{M}^{(k)} = (\Sigma, \flat, K, q_0, F, \delta)$ 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:

$$\mathcal{M}' = (\Sigma', \flat, K', q_0', F', \delta')$$

ove l'alfabeto Σ' è definito come:

$$\Sigma' = \{\flat, \downarrow\} \times \Sigma_1 \times \ldots \{\flat, \downarrow\} \times \Sigma_k$$

cioè, è composto di k coppie di simboli (λ_i, σ_i) di cui $\lambda_i \in \{\flat, \downarrow\}$ 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

$\cdots$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\downarrow$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\cdots$	t_1
$\cdots$	$\bar{b}$	$\bar{b}$	$\bar{b}$	a	b	b	a	$\bar{b}$	$\bar{b}$	$\cdots$	t_2
$\cdots$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\downarrow$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\cdots$	t_3
$\cdots$	$\bar{b}$	$\bar{b}$	$\bar{b}$	b	a	b	c	b	$\bar{b}$	$\cdots$	t_4
$\cdots$	$\bar{b}$	$\bar{b}$	$\downarrow$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\cdots$	t_5
$\cdots$	$\bar{b}$	d	e	d	d	e	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\cdots$	t_6
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
$\cdots$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\downarrow$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\cdots$	t_{2k-1}
$\cdots$	f	f	g	h	g	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\bar{b}$	$\cdots$	t_{2k}

ure 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 $\not{b}$
- $\forall i, 2 \leq i \leq k$ 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:

$$\delta^{(k)}(q_i, a_{i_1}, \ldots, a_{i_k}) = (q_j, a_{j_1}, \ldots, a_{j_k}, d_1, \ldots, d_k)$$

deve eseguire i seguenti passi:

1. rintracciare le posizione dei k simboli $\downarrow$ rappresentanti le posizione 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 macchine $\mathcal{M}^{(k)}$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 $x$aa cor $x \in \{$a,b$\}^*$.

Sia $\mathcal{M}$ la macchina di Turing non-deterministica definita nel seguente modo:

$$\mathcal{M} = (\{a, b\}, \not{b}, \{q_0, q_1\}, \delta, q_0, q_f)$$

ove la funzione δ è definita dalla seguente matrice:

δ	a	b	$\not{b}$
q_0	$(q_0, a, d), (q_1, a, d)$	(q_0, b, d)	$-$
q_1	(q_2, a, d)	$-$	$-$
q_2	$-$	$-$	$(q_f, \not{b}, i)$

Esercizio: scrivere la computazione sulla stringa abbabaa.

Macchine di Turing NON Deterministiche

Una macchina di Turing **non deterministica** è definita da 6-upla:

$$\mathcal{M} = (\Sigma, \not{b}, K, \delta, q_0, q_f)$$

ove la funzione di transizione δ è definita nel seguente modo:

$$\delta : (K - \{q_f\}) \times \Sigma_{\not{b}} \rightarrow \mathcal{P}(K \times \Sigma_{\not{b}} \times \{d, s, i\})$$

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 confgurazioni 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 macchine di Turing non deterministica $\mathcal{M}$ tramite una deterministica $\mathcal{M}^{(3)}$ si ottiene visitando l'albero delle computazione 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:

$$\Sigma_{j=1}^{k} j \cdot d^j \in \mathcal{O}(kd^k)$$

Quindi una MTND può essere simulata da una MT deterministica multi-nastro in un tempo esponenziale nel numero dei passi della macchine non deterministica.

Macchina Turing Universale

Una limitazione delle macchine di Turing

Turing Machines sono "hardwired"

$\underbrace{}_{}$

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

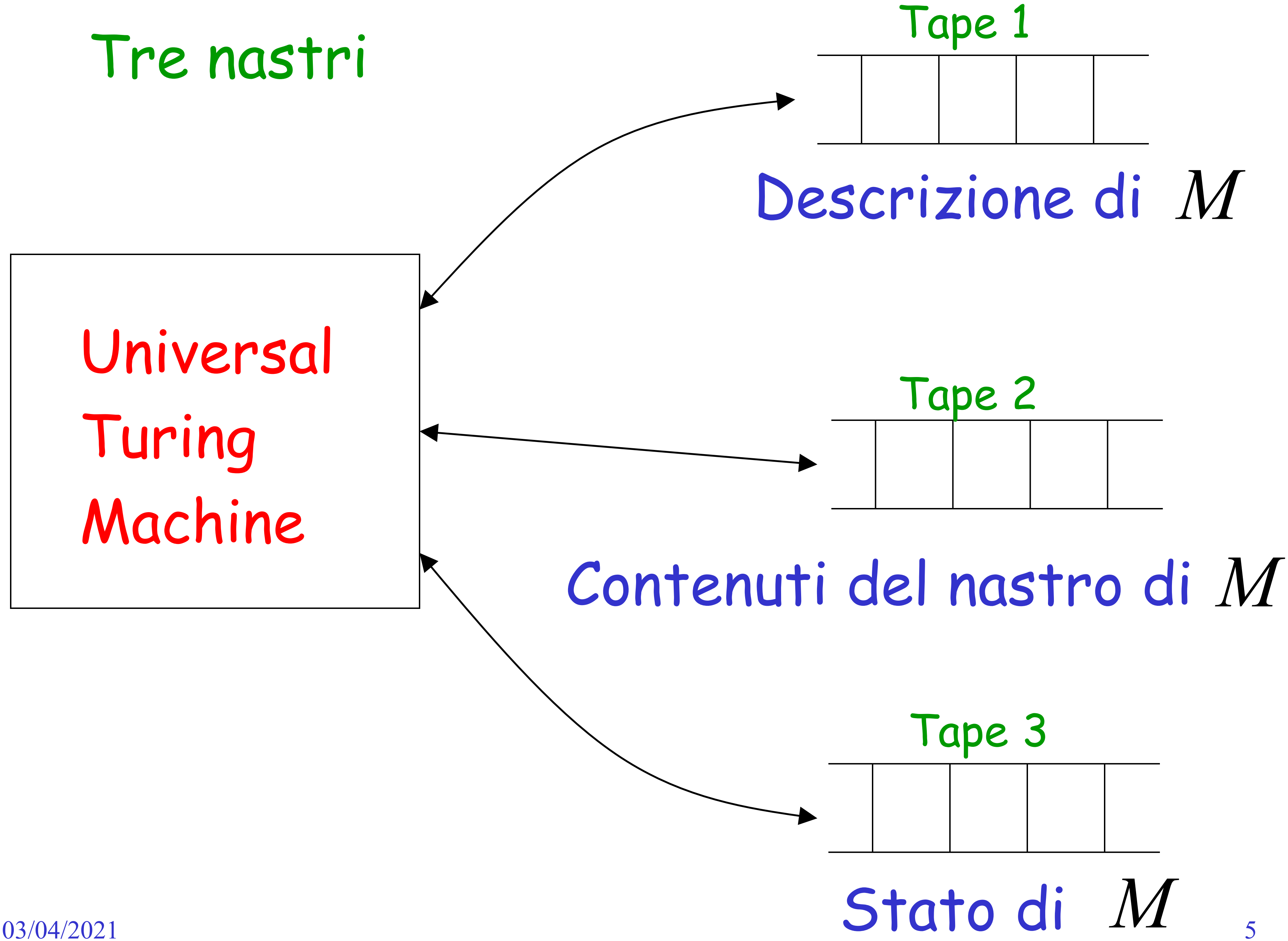
- Simula ogni altra Macchina di Turing M

Input della Universal Turing Machine:

Descrizione delle transizioni di M

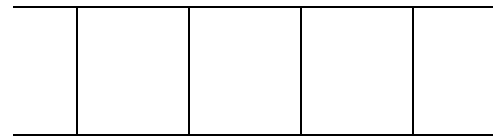
stringa di input di M

Tre nastri



Universal
Turing
Machine
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	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
codifica:	1	11	111	1111	

Codifica degli stati

Stati:	q_1	q_2	q_3	q_4	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
codifica:	1	11	111	1111	

Codifica dei movimenti della Head

Mossa:	L	R
	$\downarrow$	$\downarrow$
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:

$$\delta(q_1, a) = (q_2, b, L) \qquad \delta(q_2, b) = (q_3, c, R)$$

Codifica:

$$10101101101 \quad 00 \quad 1101101110111011$$

separatore

Tape 1 della Universal Turing Machine contiene:

Codifica binaria
della macchina da simulare M

Tape 1

1 0 1 0 11 0 11 0 10011 0 1 10 111 0 111 0 1100…

Nastro due, tre : simuliamo

(q_1,c, q_new, c_new, op)

Consideriamo tutti gli stati e tutti i caratteri (perché? Si può fare diversamente?)

Una lista
{(q,c, q_new, c_new, op) Per tutti gli stati q, 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, nuovoc, op);
Ovvero I=n; c=nuovoc
c=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) $0,\ 2,\ 4,\ 6,\ \ldots$

corrispondenza: $\downarrow\ \downarrow\ \downarrow\ \downarrow$

Interi positivi: $1,\ 2,\ 3,\ 4,\ \ldots$

$2n$ Corrisponde a $n+1$

Esempio: L'insieme dei numeri razionali

Numeri razionali: $\frac{1}{2}, \frac{3}{4}, \frac{7}{8}, \ldots$

approccio banale

Nominatore 1

Numeri razionali: $\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \ldots$

Corrispondenza: $\downarrow \quad \downarrow \quad \downarrow$

interi positivi: $1, \ 2, \ 3, \ \ldots$

Non funziona:

Non analizzeremo mai

Numeri con nominatore 2: $\frac{2}{1}, \frac{2}{2}, \frac{2}{3}, \ldots$

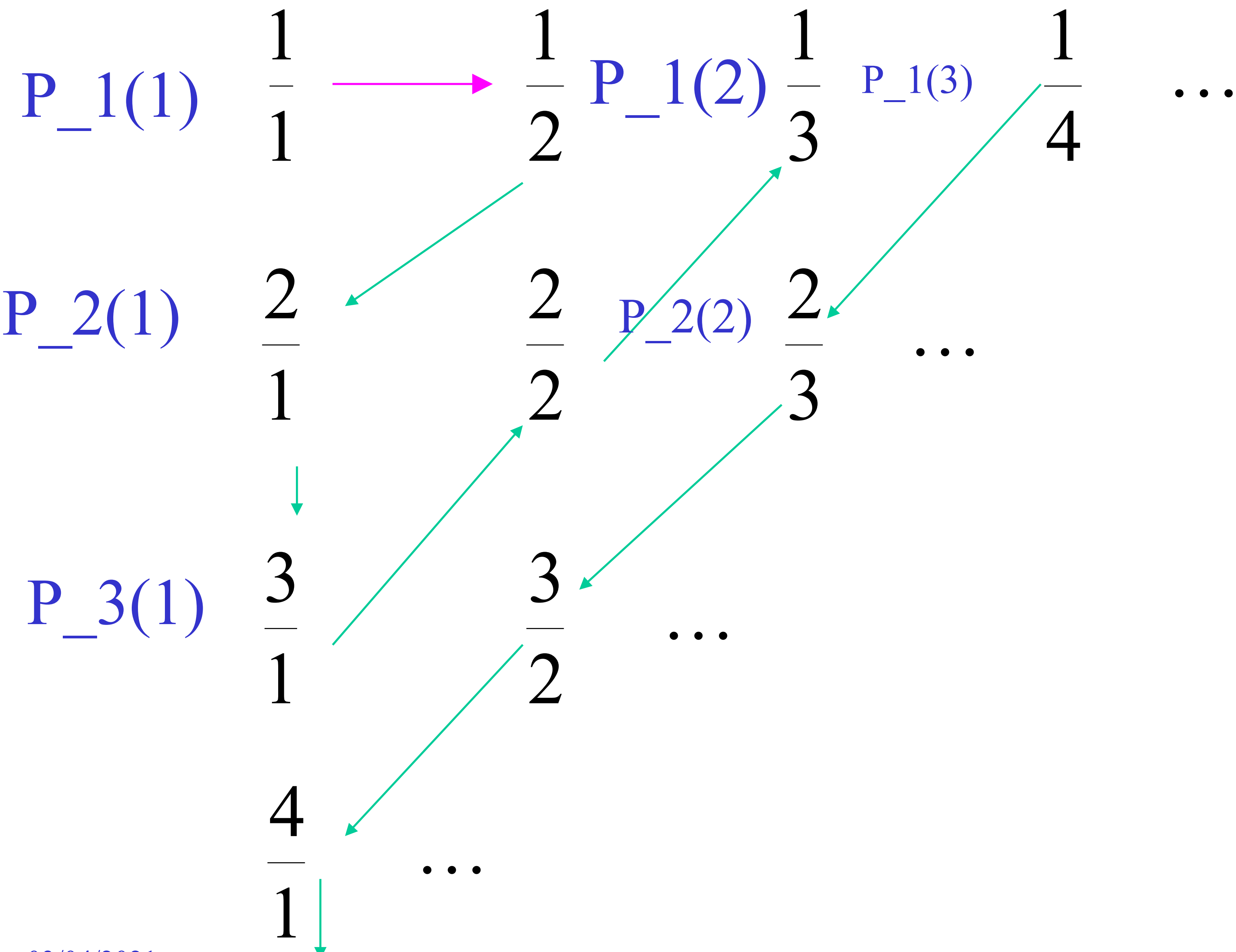
Approccio migliore

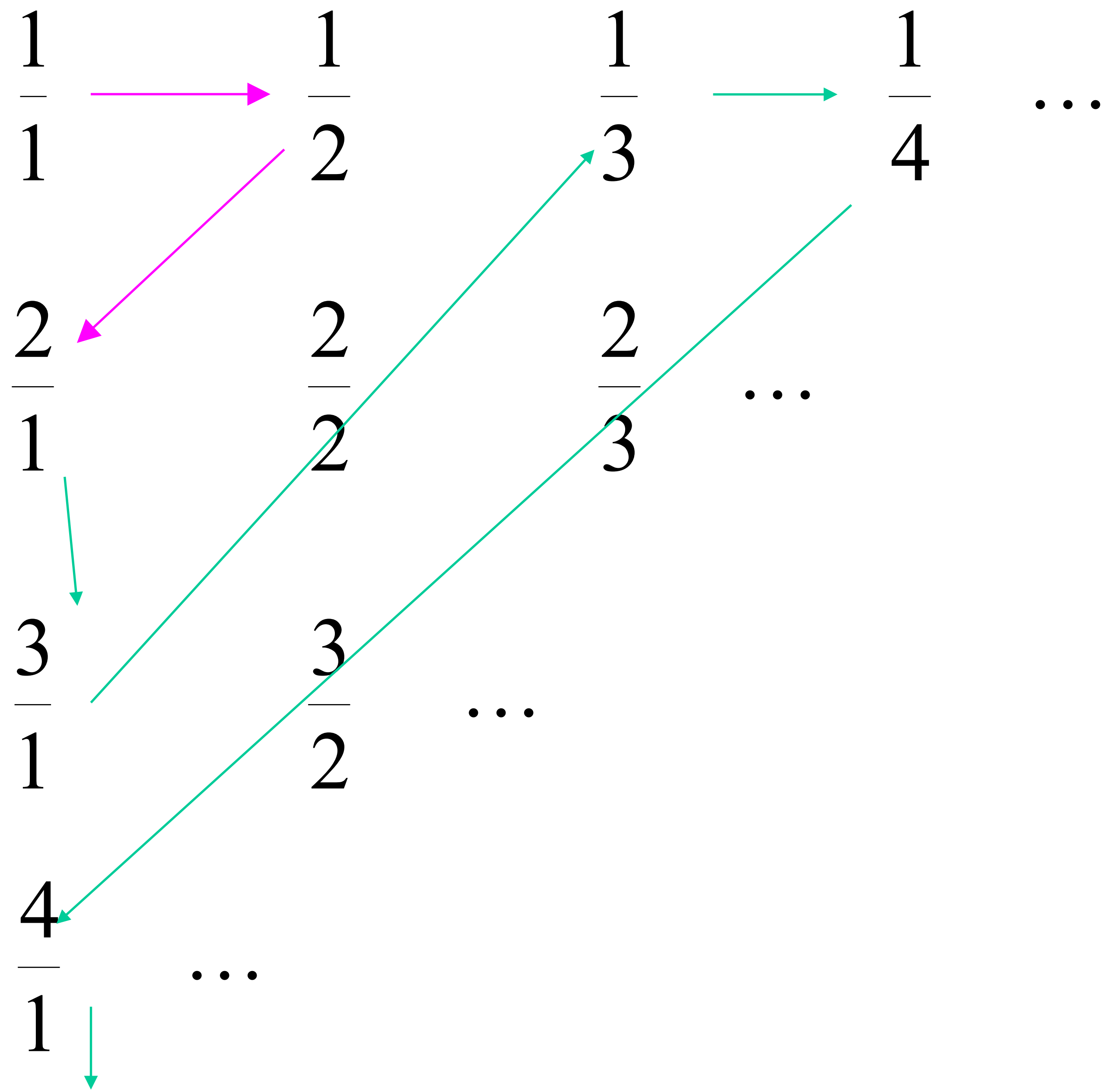
P_1 $\quad \frac{1}{1} \quad \frac{1}{2} \quad \frac{1}{3} \quad \frac{1}{4} \quad \cdots$

P_2 $\quad \frac{2}{1} \quad \frac{2}{2} \quad \frac{2}{3} \quad \cdots$

$\frac{3}{1} \quad \frac{3}{2} \quad \cdots$

$\frac{4}{1} \quad \cdots$


P_1(1)
1/1
1/2
P_1(2)
1/3
P_1(3)
1/4
…
P_2(1)
2/1
2/2
P_2(2)
2/3
…
P_3(1)
3/1
3/2
…
4/1
…


1/1
1/2
1/3
1/4
2/1
2/2
2/3
3/1
3/2
4/1

Oppure?

$$
\begin{array}{ccccc}
\frac{1}{1} & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots \\[2em]
\frac{2}{1} & \frac{2}{2} & \frac{2}{3} & \cdots & \\[2em]
\frac{3}{1} & \frac{3}{2} & \cdots & & \\[2em]
\frac{4}{1} & \cdots & & &
\end{array}
$$

P_1(1)

P_1(2) P_2(1)

P_1(3) P_2(2) P_3(1)

P_1(4) P_2(3) P_3(2) P_4(1)

I=1

J=1,I

P_I(J)

Numeri razionali: $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{2}{2}, \ldots$

Corrispondenza:

interi positivi: $1, \; 2, \; 3, \; 4, \; 5, \; \ldots$

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 $s_1, s_2, s_3, \ldots \in S$

Enumerator Machine per S

output (sul nastro) $s_1, s_2, s_3, \ldots$

Tempo finito: $t_1, t_2, t_3, \ldots$

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

$$s_1 = a$$

$$s_2 = aa$$

$$\vdots \quad aaa$$

$$aaaa$$

$$\ldots\ldots$$

No buono:

le stringhe con primo carattere b non vengono fuori

Procedura migliore : **Ordine proprio** (Canonical, proper, Order)

1. Produci tutte le stringhe di lunghezza 1

2. Produci tutte le stringhe di lunghezza 2

3. Produci tutte le stringhe di lunghezza 3

4. Produci tutte le stringhe di lunghezza 4

……….

$s_1 = a$
$s_2 = b$
c
$\vdots$

lunghezza 1

aa
ab
ac
ba
bb
bc
ca
cb
cc

lunghezza 2

Produce stringhe in
Proper Order:

aaa
aab
aac
$\ldots\ldots$

lunghezza 3

Codifica degli stati

Stati:	q_1	q_2	q_3	q_4	$\cdots$
	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	
codifica:	1	11	111	1111	

Codifica dei movimenti della Head

Mossa:	L	R
	$\downarrow$	$\downarrow$
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 | **Turing Machines**

0
1
00
01
⋮
1 0 1 0 11 0 11 0 0
1 0 1 0 11 0 11 0 1 $\xrightarrow{s_1}$ 1 0 1 0 11 0 11 0 1
⋮
1 0 11 0 1010010101101 $\xrightarrow{s_2}$ 1 0 11 0 1010010101101
⋮

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

$$S = \{s_1, s_2, s_3, \ldots\}$$

Elementi di S

Gli elementi dell'insieme delle parti 2^S hanno la forma:

$$\emptyset$$

$$\{s_1, s_3\}$$

$$\{s_5, s_7, s_9, s_{10}\}$$

……

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	$\cdots$
$\{s_1\}$	1	0	0	0	$\cdots$
$\{s_2, s_3\}$	0	1	1	0	$\cdots$
$\{s_1, s_3, s_4\}$	1	0	1	1	$\cdots$

Osservazione:

Ogni stringa binaria infinita corrisponde a un elemento dell' insieme delle parti
:

esempio: $1\,0\,0\,1\,1\,1\,0\ \cdots$

Corrispondente a : $\{s_1, s_4, s_5, s_6, \ldots\} \in 2^S$

assumiamo (per assurdo)

Che l' Insieme delle parti 2^S

è enumerabile

allora: possiamo enumerare gli elementi dell' insieme delle parti

$$2^S = \{t_1, t_2, t_3, \ldots\}$$

Insieme delle Parti elementi	Supponiamo che la seguente sia **Codifica Binaria**				
t_1	1	0	0	0	0 $\cdots$
t_2	1	1	0	0	0 $\cdots$
t_3	1	1	0	1	0 $\cdots$
t_4	1	1	0	0	1 $\cdots$
$\cdots$			$\cdots$		

Prendiamo la diagonale e complementiamola

$$
\begin{array}{llllll}
t_1 & 1 & 0 & 0 & 0 & 0 \ \cdots \\
t_2 & 1 & 1 & 0 & 0 & 0 \ \cdots \\
t_3 & 1 & 1 & 0 & 1 & 0 \ \cdots \\
t_4 & 1 & 1 & 0 & 0 & 1 \ \cdots
\end{array}
$$

Stringa binaria: $\boldsymbol{t} = 0011\ldots$

Stringa binaria

$$t = 0011\ldots$$

Corrisponde ad un elemento dell'Insieme delle parti 2^S :

$$t = \{s_3, s_4, \ldots\} \in 2^S$$

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 contradizione 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:

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

infinito e countable

(possiamo enumerare le stringhe
in ordine proprio)

Considera alfabeto : $A=\{a,b\}$

L'insieme delle stringhe:

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

infinito e countable

Ogni linguaggio è un sottoinsieme di S :

$$L=\{aa,ab,aab\}$$

Considera l'alfabeto : $A = \{a, b\}$

L'insieme delle stringhe:

$$S = A^* = \{a, b\}^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$$

infinito e countable

Ricorda: L'insieme delle parti di S contiene tutti i linguaggi:

$$2^S = \{\emptyset, \{\lambda\}, \{a\}, \{a, b\}, \{aa, b\}, \ldots, \{aa, ab, aab\}, \ldots\}$$

uncountable

Considera alfabeto : $A = \{a, b\}$

countable

Turing machines: $M_1 \quad M_2 \quad M_3 \quad \cdots$

accetta

Linguaggi accettati da Turing Machines: $L_1 \quad L_2 \quad L_3 \quad \cdots$

countable

Denota: $X = \{L_1, L_2, L_3, \ldots\}$
countable

Nota: $X \subseteq 2^S$

$\left(S = \{a, b\}^*\right)$

Linguaggi accettati da Turing machines: X countable

Tutti i possibili linguaggi: 2^S uncountable

quindi: $X \neq 2^S$

$$\left(\text{since } X \subseteq 2^S, \text{ we have } X \subset 2^S\right)$$

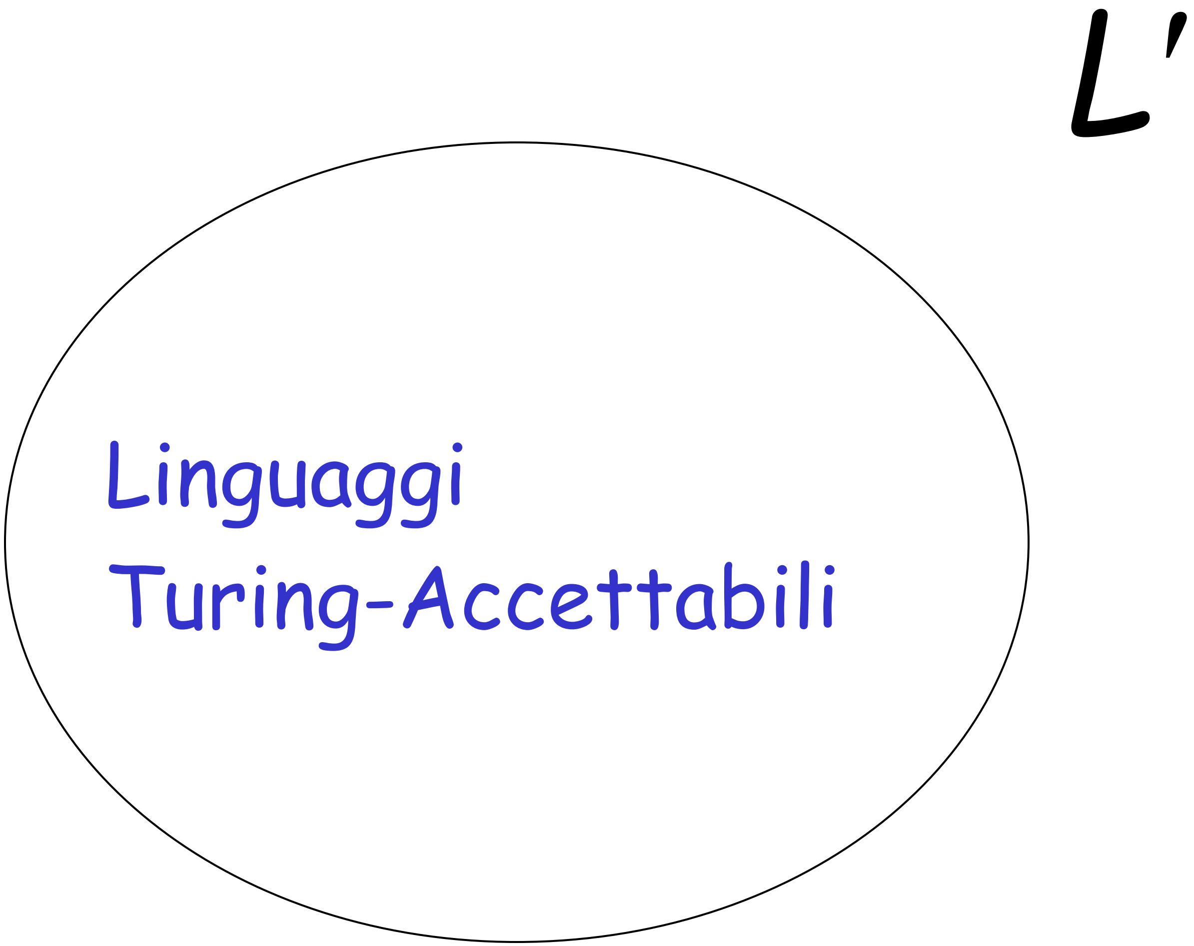
Conclusione:

Esiste un linguaggio L' non accettato da nessuna Turing Machine:

$$X \subset 2^S \Longrightarrow \exists L' \in 2^S \text{ and } L' \notin X$$

(linguaggio L' non può essere descritto da nessun algoritmo)

Linguaggi Non Turing-Accettabili



Nota che: $X = \{L_1, L_2, L_3, \ldots\}$

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:

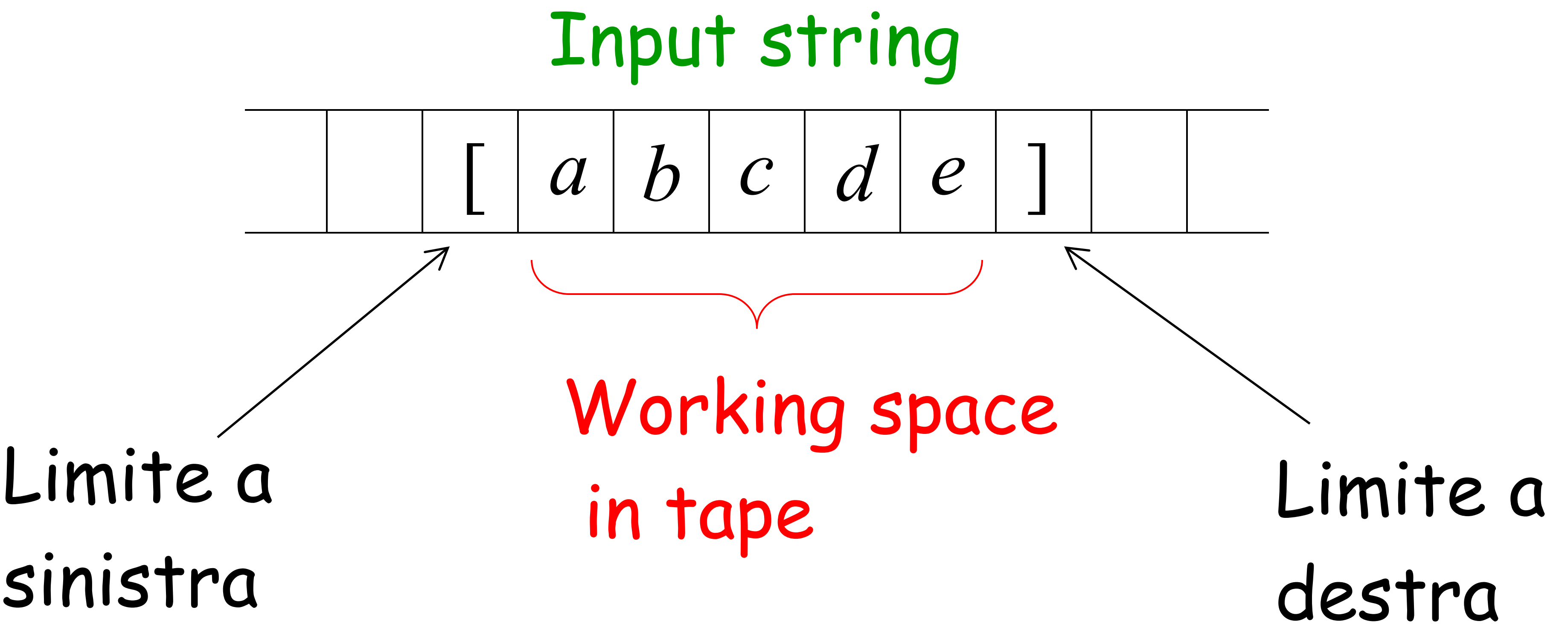
Come una Macchina di Turing

con una differenza:

Lo spazio dove è memorizzato l'input

è il solo spazio che può essere utilizzato

Linear Bounded Automa (LBA)



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:

$$L = \{a^n b^n c^n\} \qquad\qquad L = \{a^{n!}\}$$

LBA hanno più potere dei PDA
(pushdown automata)

LBA hanno meno potere delle Turing Machines

Grammatica Context-Sensitive :

produzioni

$$u \to v$$

Stringhe di variabili
e terminali

Stringhe di variabil
e terminali

e: $|u| \leq |v|$

Il linguaggio $\{a^n b^n c^n\}$

è context-sensitive:

$$S \to abc \mid aAbc$$

$$Ab \to bA$$

$$Ac \to Bbcc$$

$$bB \to Bb$$

$$aB \to aa \mid aaA$$

Teorema:

Un linguaggio L è context sensitive

$\Longrightarrow$

è accettato da Linear-Bounded automa

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

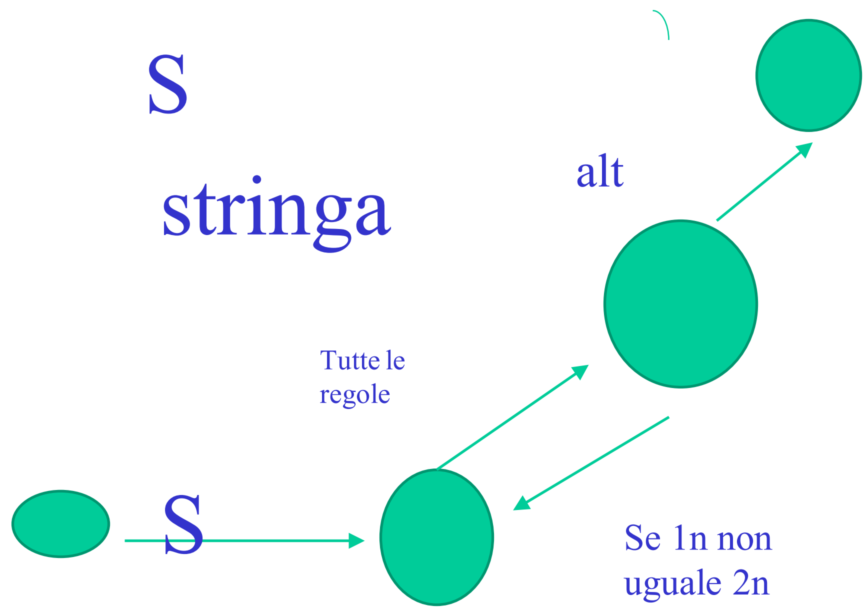
$$S \to abc \mid aAbc$$

$$Ab \to bA$$

$$Ac \to Bbcc$$

$$bB \to Bb$$

$$aB \to aa \mid aaA$$



Grammatiche senza limitazioni:

Produzioni

$$u \rightarrow v$$

Stringhe di variabili e terminali

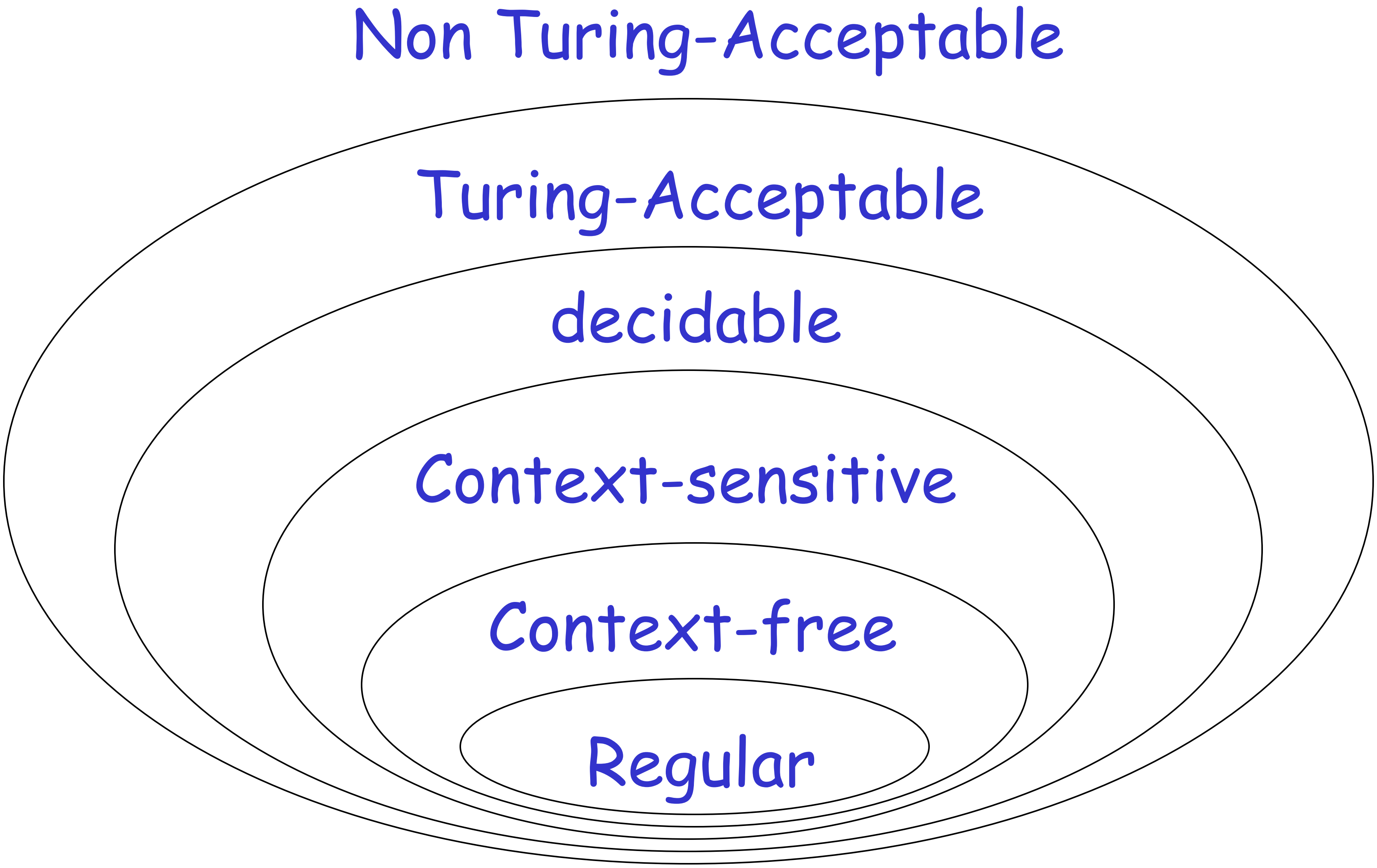
Stringhe di variabili e terminali

Teorema:

Un linguaggio L è Turing-Acceptable
Se L è generato da
una grammatica senza restrizione

Perchè? Come fare?

The Chomsky gerarchia



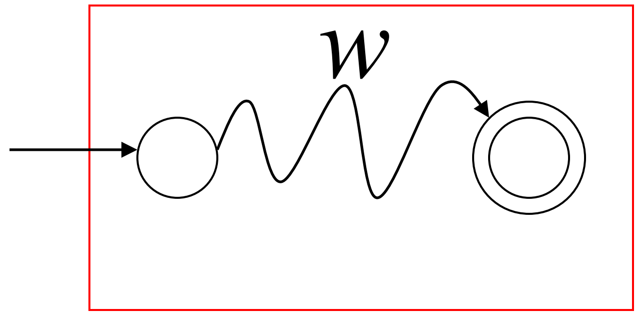
problemi decidibili (decidibile?) per linguaggi regolari

appartenenza

Domanda: dato un linguaggio regolare L
e una stringa w
Possiamo verificare se $w \in L$?

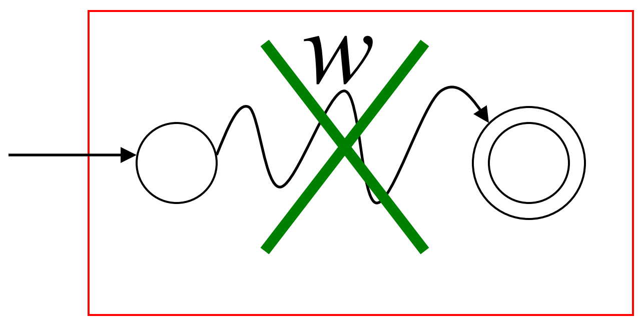
Risposta: Prendiamo un DFA che
accetta L
e verifichiamo se w è
accettato.

DFA



$$w \in L$$

DFA



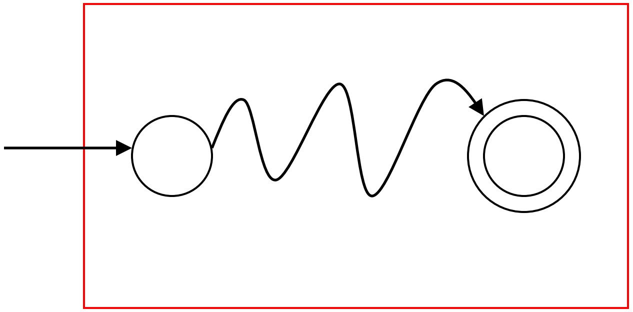
$$w \notin L$$

Domanda: dato un linguaggio regolare L
Come possiamo verificare
se L è vuoto: $(L = \emptyset)$?

Risposta: Prendiamo il DFA che accettà L

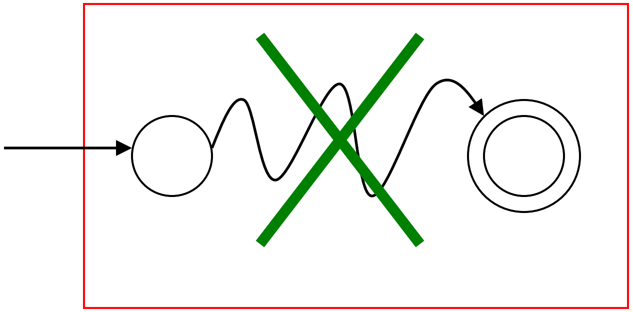
Verifichiamo se esiste almeno
un cammino da uno stato iniziale
ad uno stato di accettazione

DFA



$$L \neq \emptyset$$

DFA



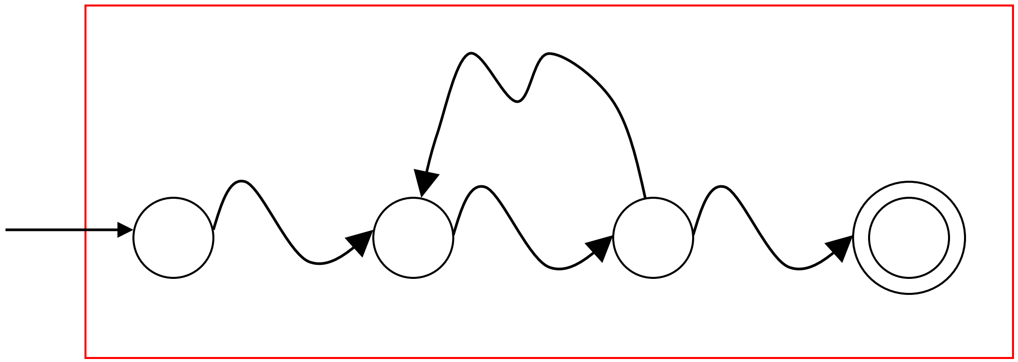
$$L = \emptyset$$

Domanda: Dato un linguaggio regolare L
Come possiamo verificare
Se L è finito?

Prendiamo il DFA che accetta L

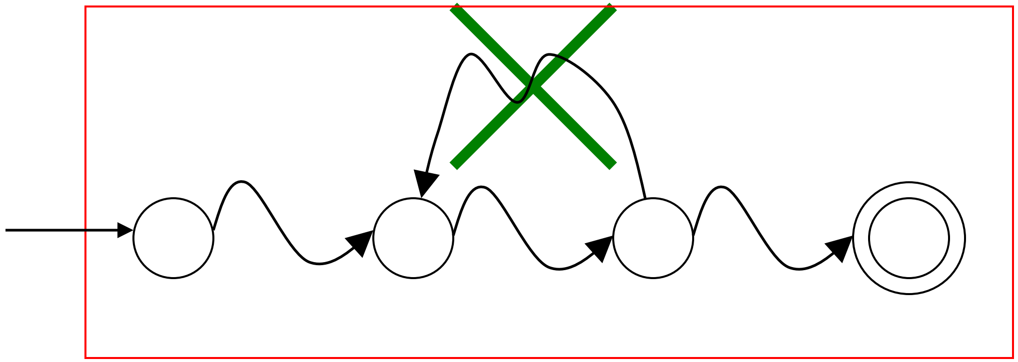
Risposta:
Verifichiamo se vi è un cammino con un loop.

DFA



L è infinito

DFA



L è finito

Domanda: dato linguaggi regolari L_1 e L_2
Come possiamo verificare che

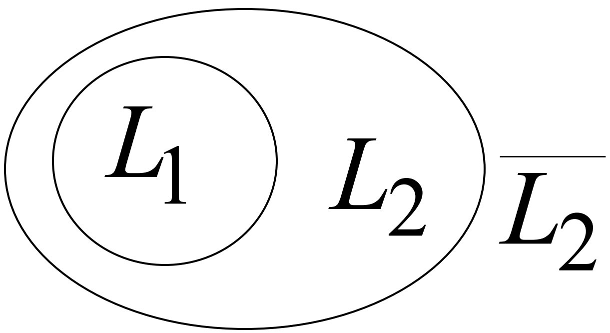
$$L_1 = L_2$$

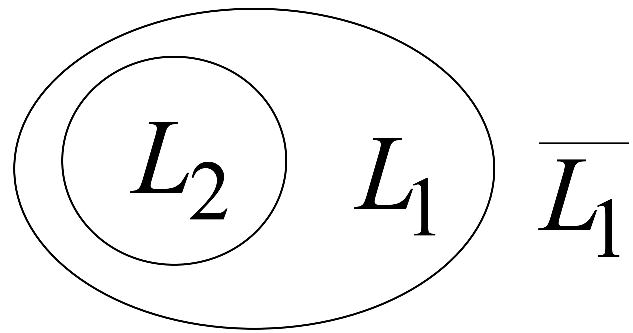
Risposta: Trova se

$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$$

$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) = \emptyset$$

$L_1 \cap \overline{L_2} = \emptyset$ and $\overline{L_1} \cap L_2 = \emptyset$



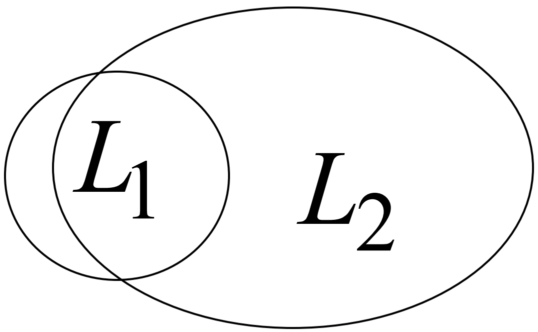


$L_1 \subseteq L_2$ $L_2 \subseteq L_1$

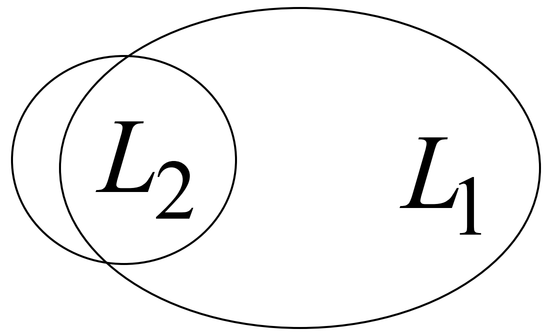
$$L_1 = L_2$$

$$(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2) \neq \emptyset$$

$$L_1 \cap \overline{L_2} \neq \emptyset \quad \text{or} \quad \overline{L_1} \cap L_2 \neq \emptyset$$



$$L_1 \not\subset L_2$$



$$L_2 \not\subset L_1$$

$$L_1 \neq L_2$$

problemi decidibili per linguaggi Context-Free

appartenenza:

per grammatiche context-free G

Se la stringa $w \in L(G)$

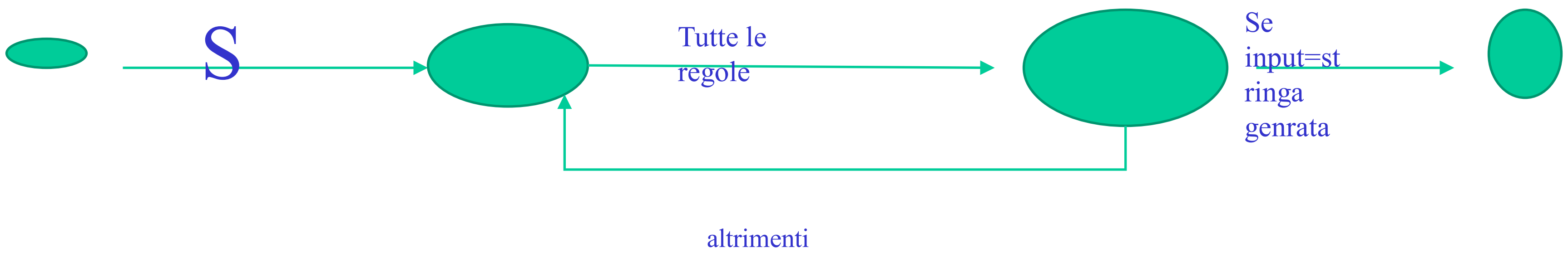
appartenenza : Parsers

- "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   Q c= c Q

Delta(Q, c)=(Q, c, L)   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

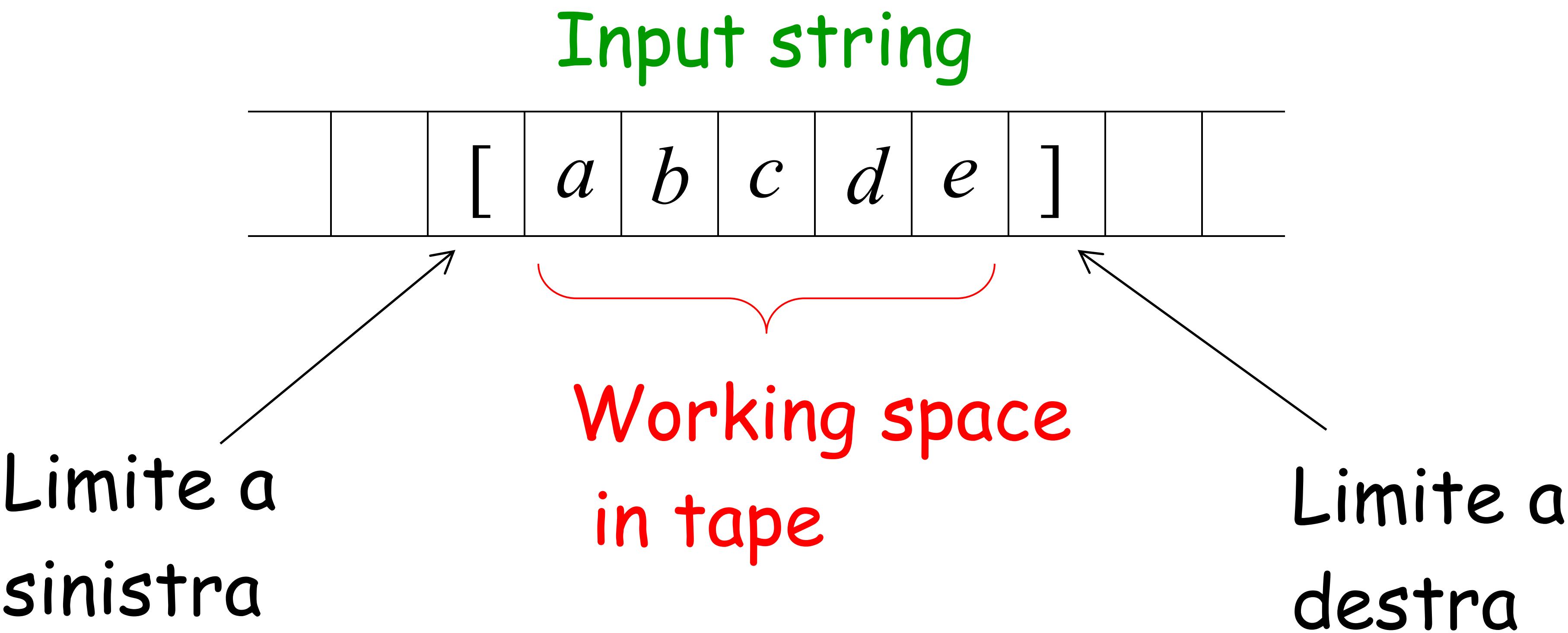
La gerarchia di Chomsky

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)



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:

$$S \to aBc$$

$$aB \to cA$$

$$Ac \to d$$

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

$$u \to v$$

Stringhe di variabili
E terminali

Stringhe di variabil
E terminali

e: $$|u| \leq |v|$$

Il linguaggio $\{a^n b^n c^n\}$

è context-sensitive:

$$S \to abc \mid aAbc$$

$$Ab \to bA$$

$$Ac \to Bbcc$$

$$bB \to Bb$$

$$aB \to aa \mid aaA$$

Theorem:

Un linguaggio L è context sensistive

se e solo se

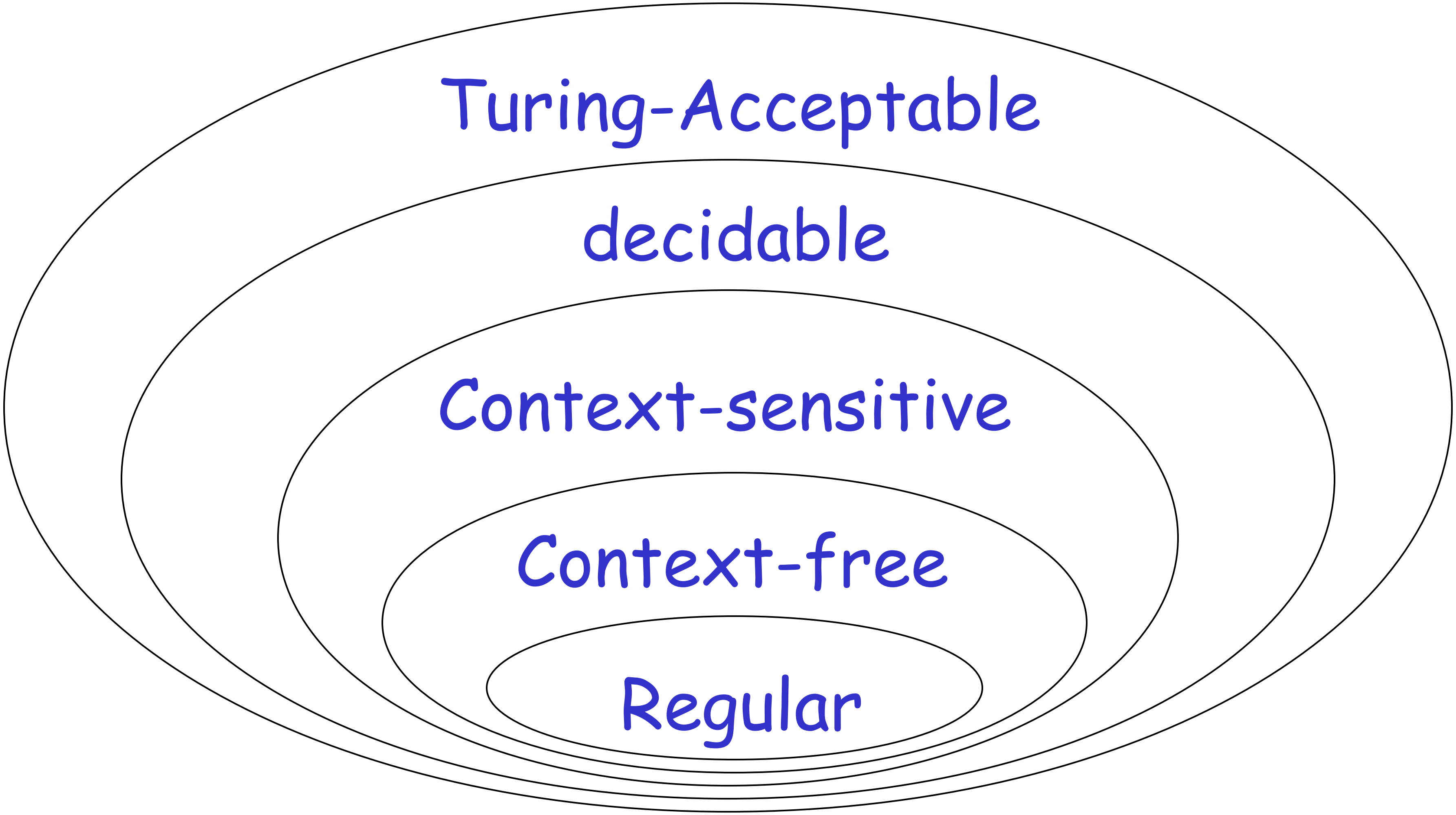
È accettato da Linear-Bounded automa

osservazione:

Vi è un linguaggio che è context sensitive
e non è decidibile

The Chomsky gerarchia

Non Turing-Acceptable

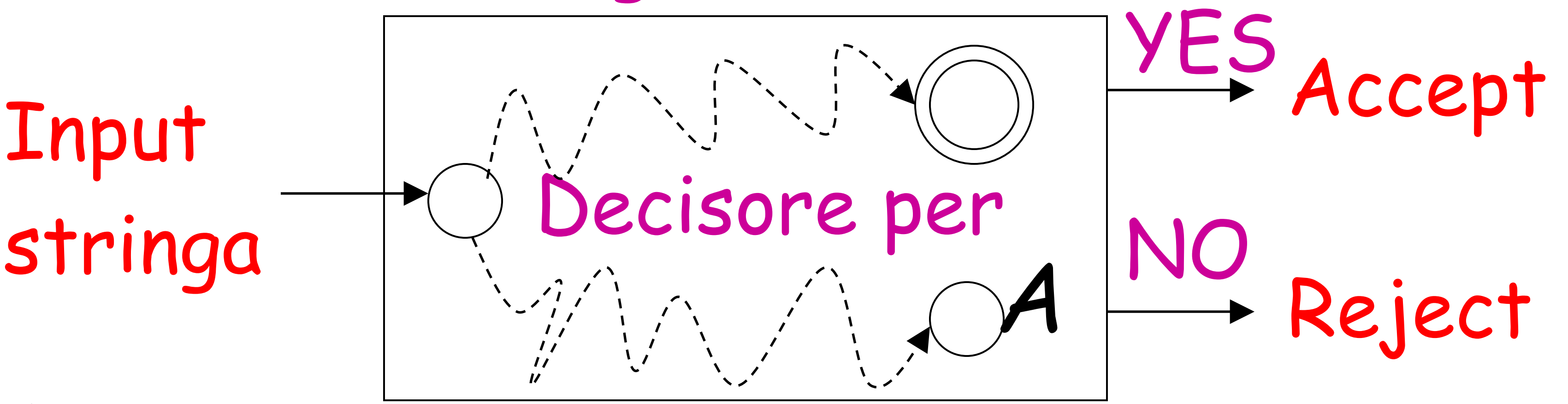


Problemi indecidibili (unsolvable problems)

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)

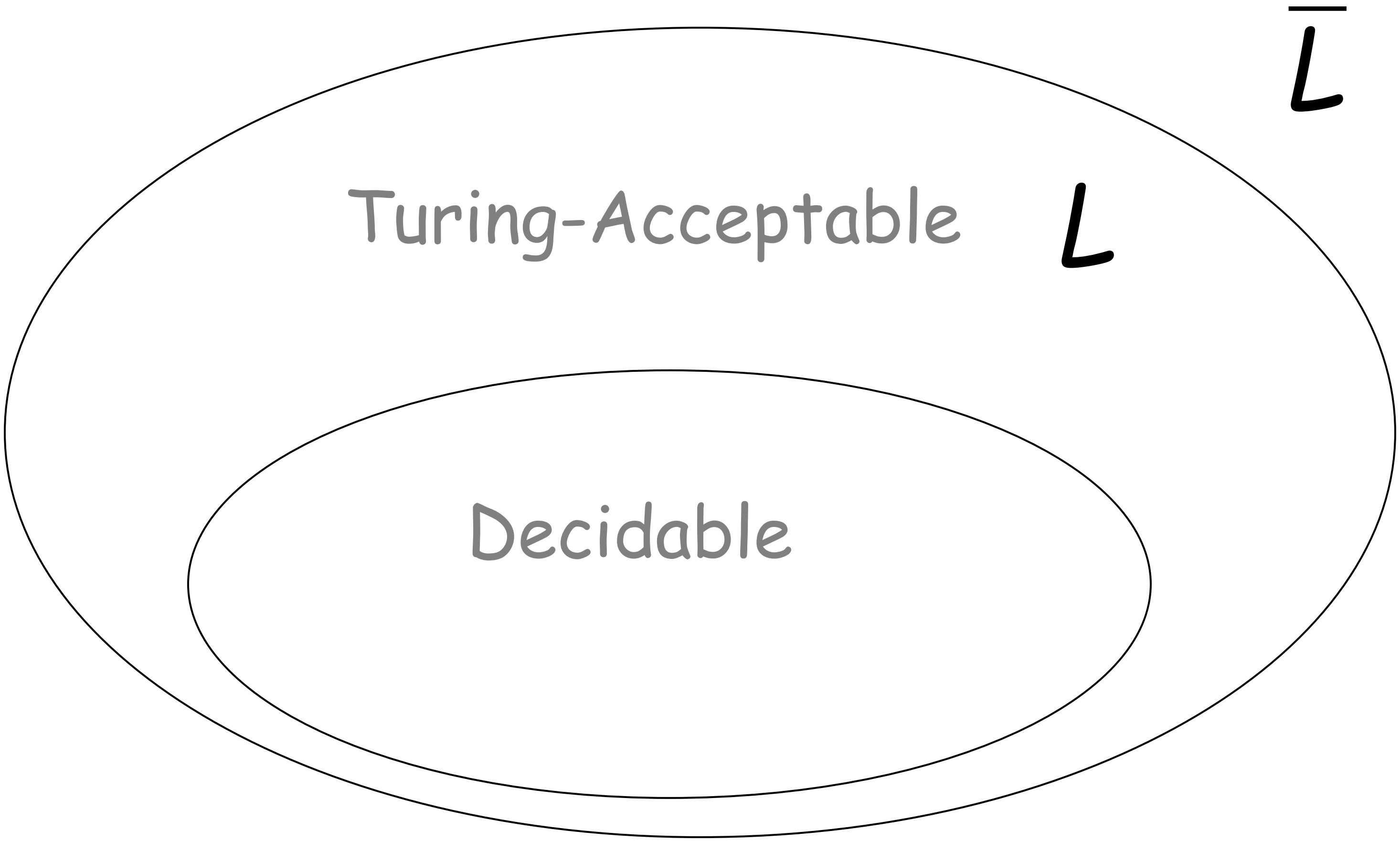
Definizione. Turing accettabile

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 gia 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

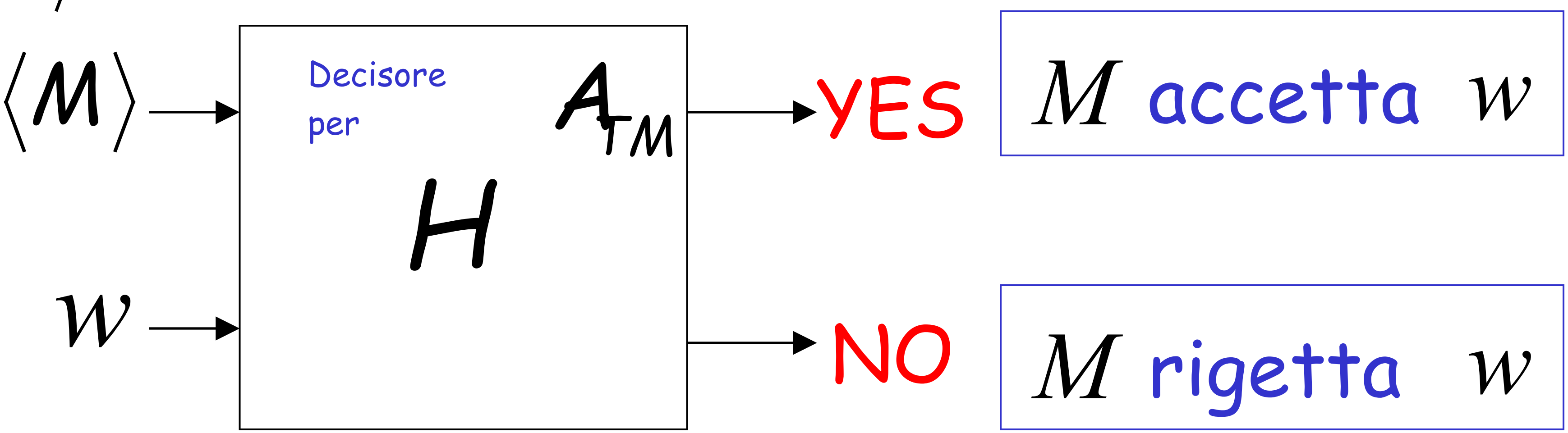
(The membership problem non è decidible)

Supponiamo che A_{TM} sia decidibile

Supponiamo che A_{TM} è decidibile

Esiste una macchina H:

Input string
$\langle M, w \rangle$

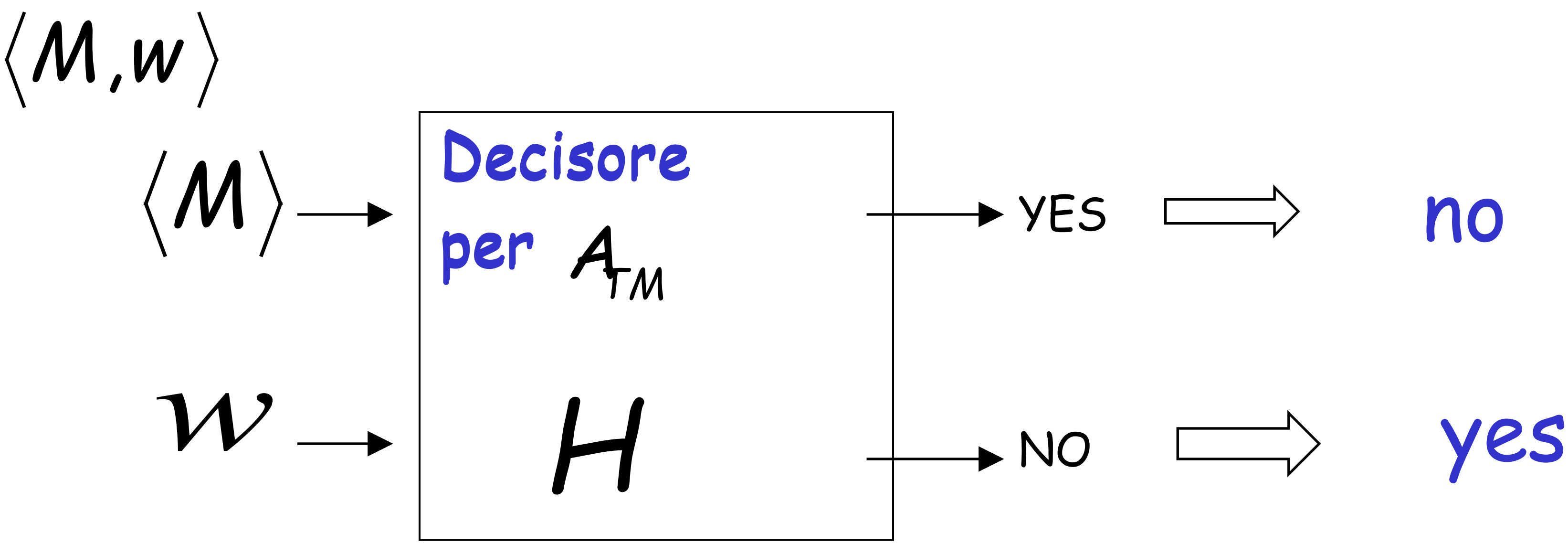


Cambiamo, diagonalizziamo, definiamo la macchina Diag:

Diag accetta (yes) se H dice no; ovvero se M(w) =no

Diag rigetta (no) se H dice si; ovvero se M(w) =s

Diag:



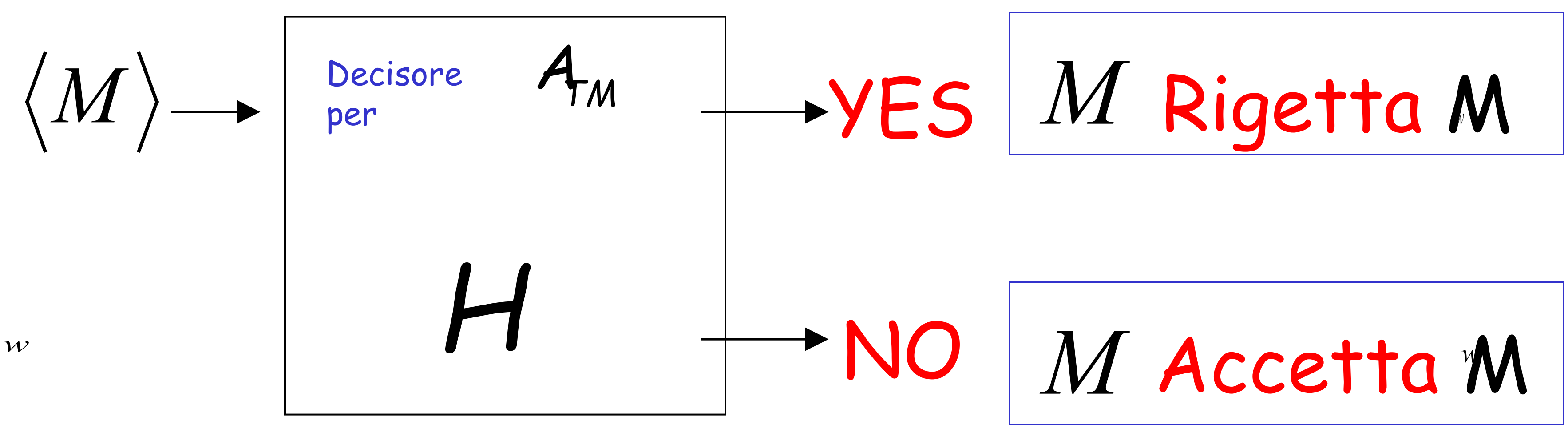
Semplifichiamo Diag .

Definiamo Diag

Descrizione di Diag:

Diag accetta M se M rigetta M

Diag rigetta M se M accetta M

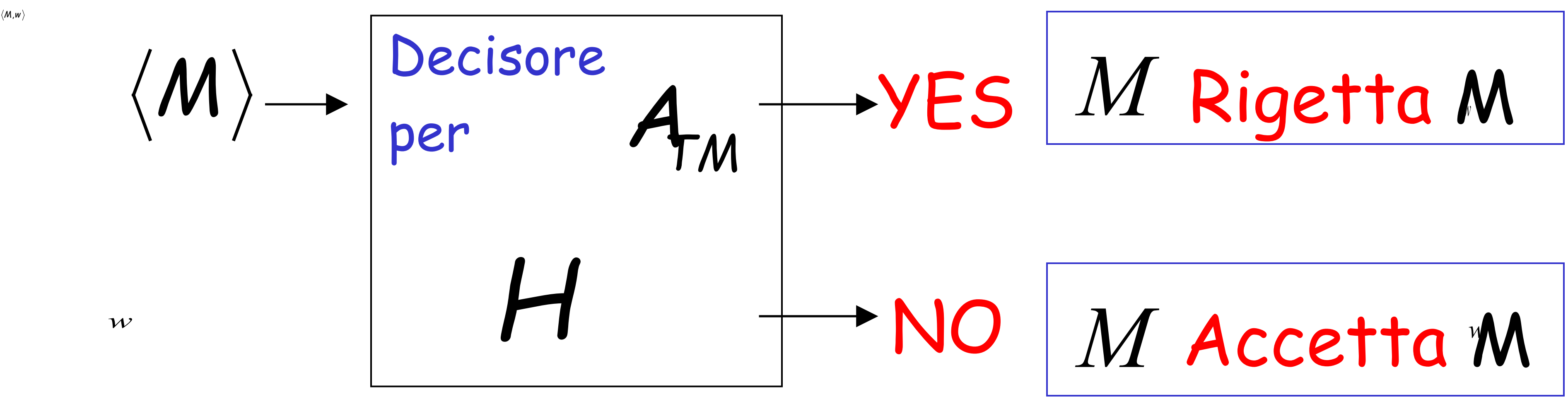


A_{TM}

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

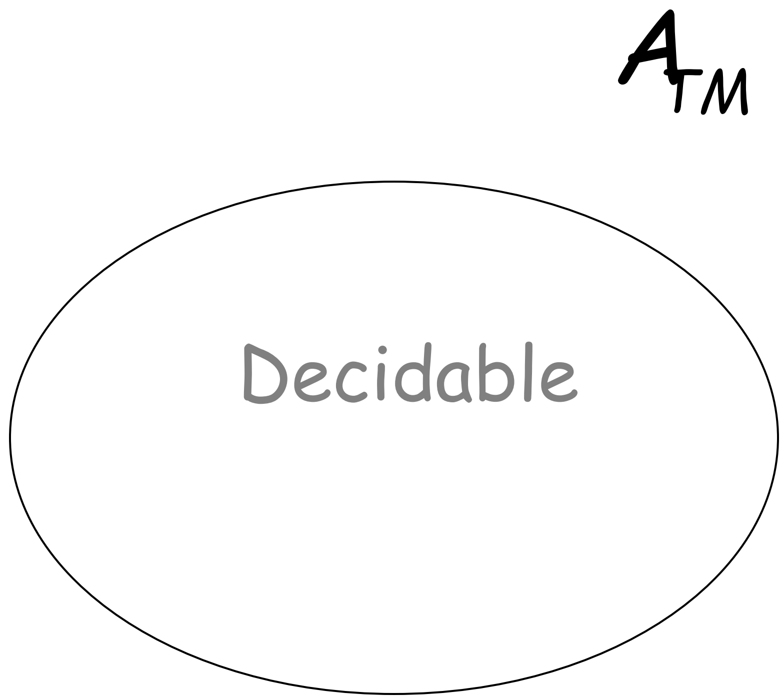
Al posto di M sostituiamo Diag

Cosa accade?:

Diag accetta Diag se Diag rigetta Diag (!!)

Diag rigetta Diag se Diag accetta Diag (!!)

Abbiamo mostrato:



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}:

$\langle M,w \rangle \longrightarrow$

1. Run M on input w
2. If M accepts w then accept $\langle M,w \rangle$

Halting Problem

Input: • Turing Machine M
• String w

domanda: M si ferma nel processo di calcolo con stringa di input w?

linguaggio corrispondente:

$$HALT_{TM} = \{\langle M, w\rangle : M \text{ è una Turing machine che si ferma sull'input w } \}$$

$$A_{TM} = \{\langle M, w\rangle : M \text{ è una Turing machine che accetta la stringa } w\}$$

$$HALT_{TM} = \{\langle M, w\rangle : M \text{ è una Turing machine che si ferma sull'input w }\}$$

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

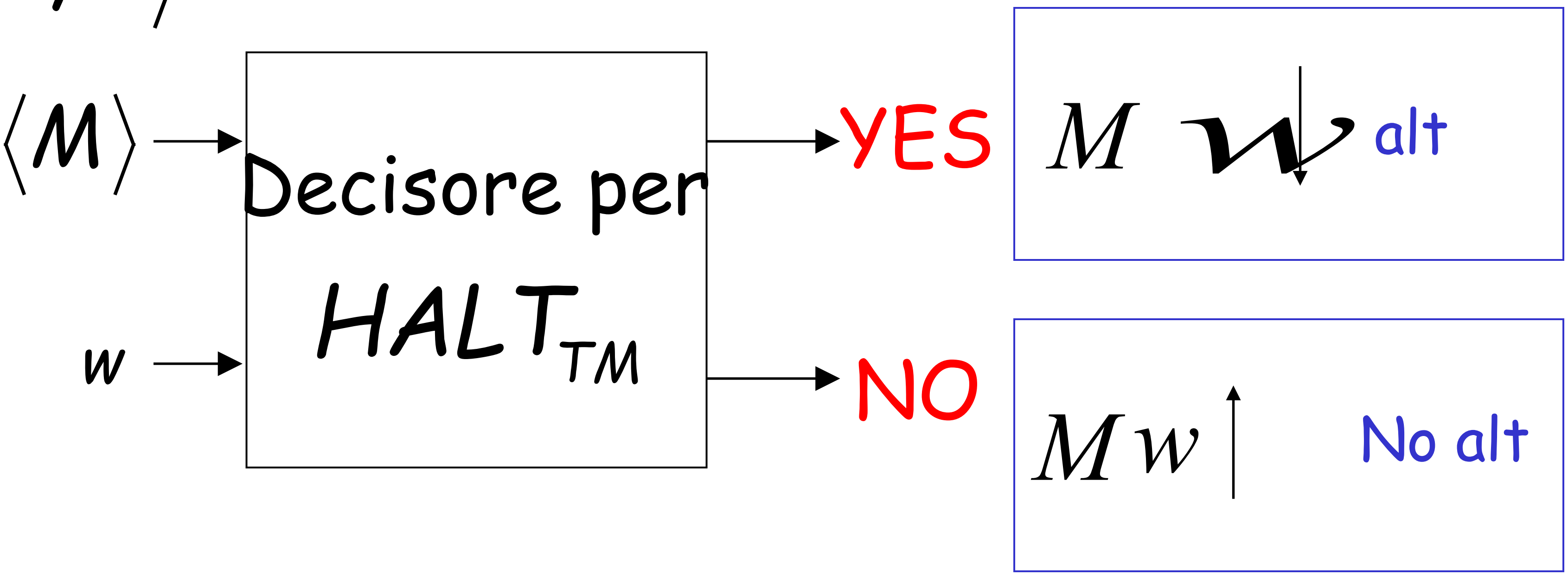
è decidibile

contradizione!

Supponiamo che $HALT_{TM}$ è decidibile

Input string

$\langle M, w \rangle$



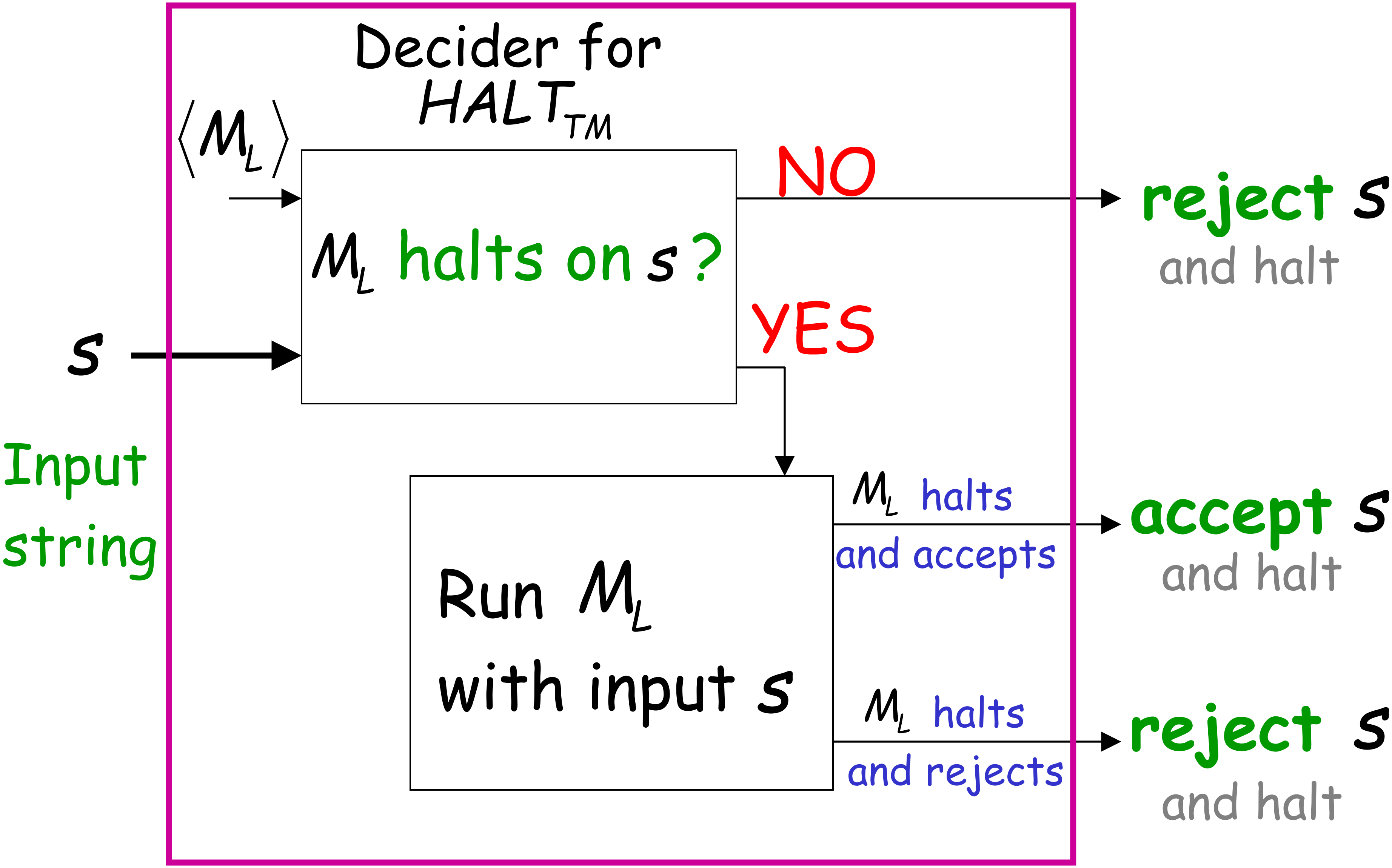
sia L un linguaggio Turing-Accettabile
Turing-Semidecidibile

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:

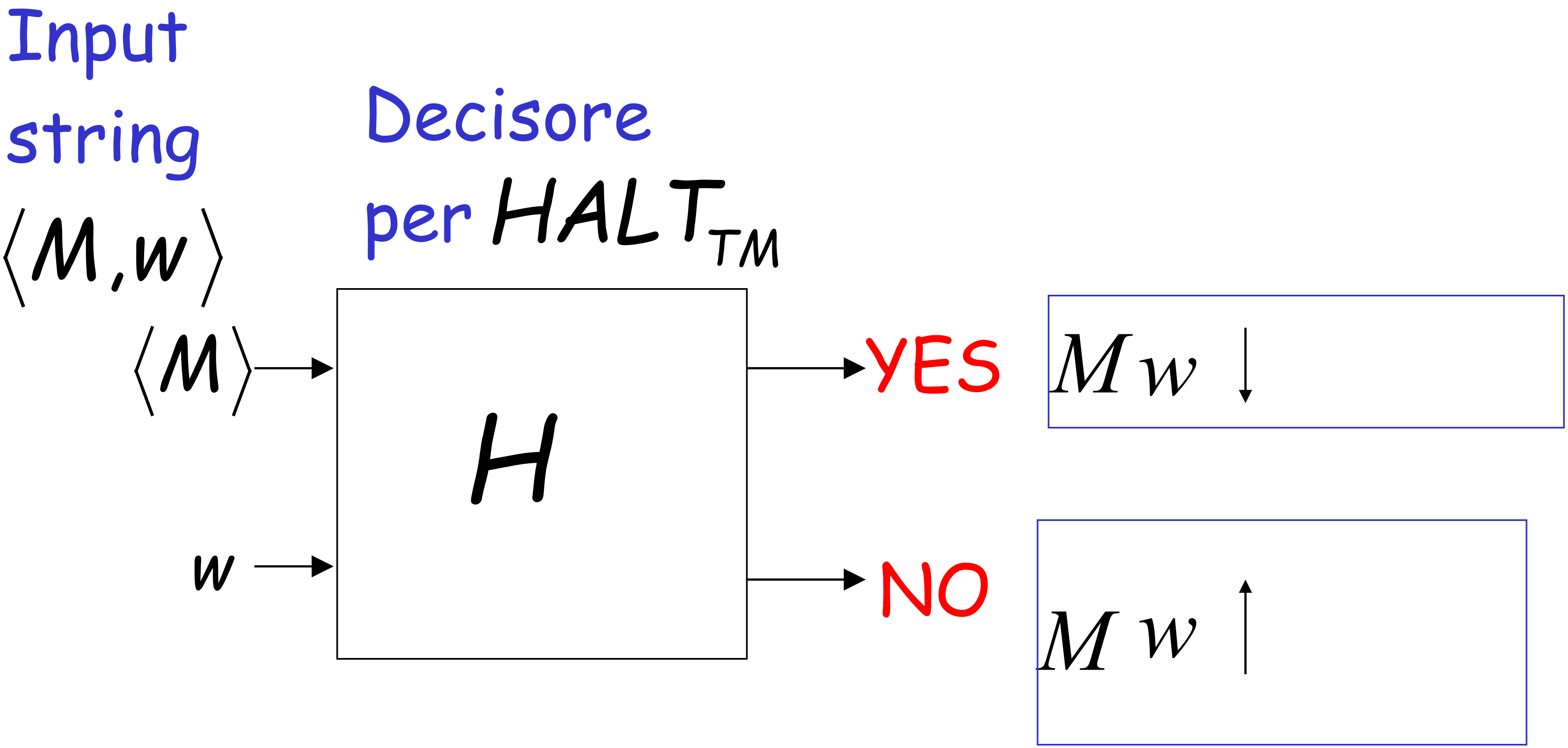
Idea di base:

Per assurdo: assumiamo che

l' halting problem decidibile;

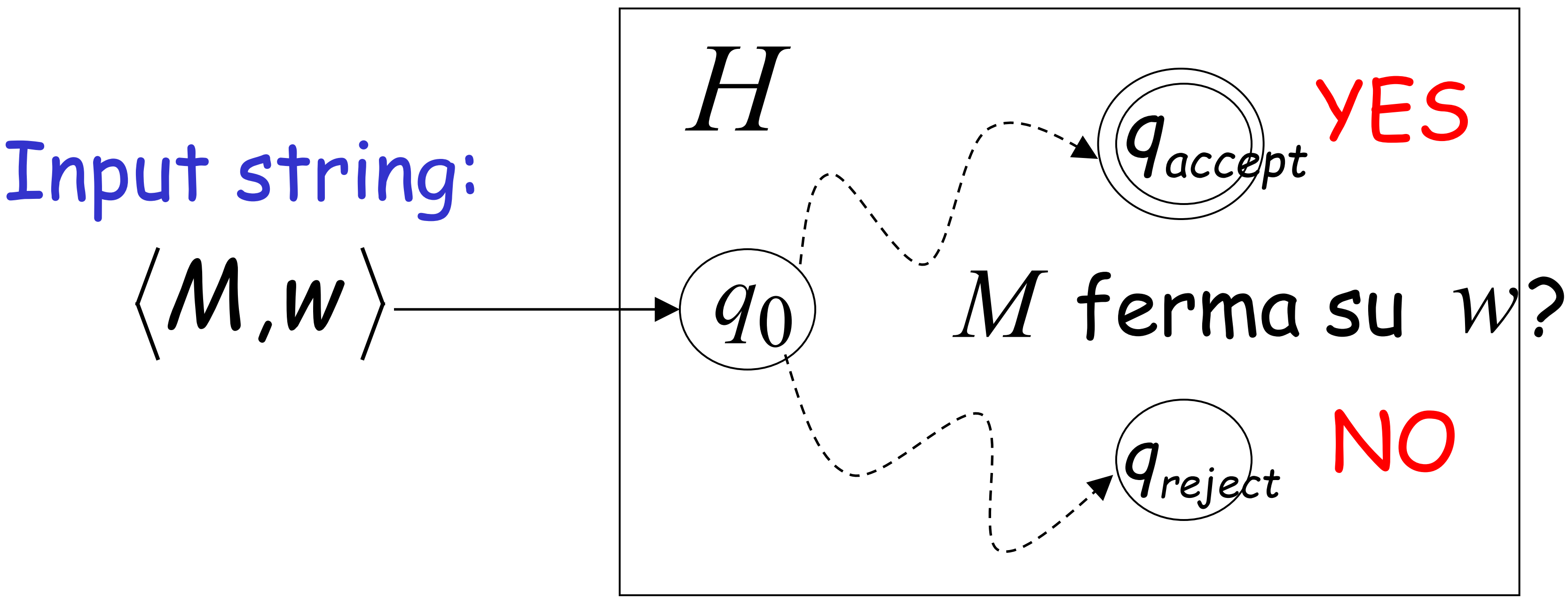
Cercheremo di ottenere una contradizione via diagonalizzazione

Supponi che $HALT_{TM}$ è decidibile

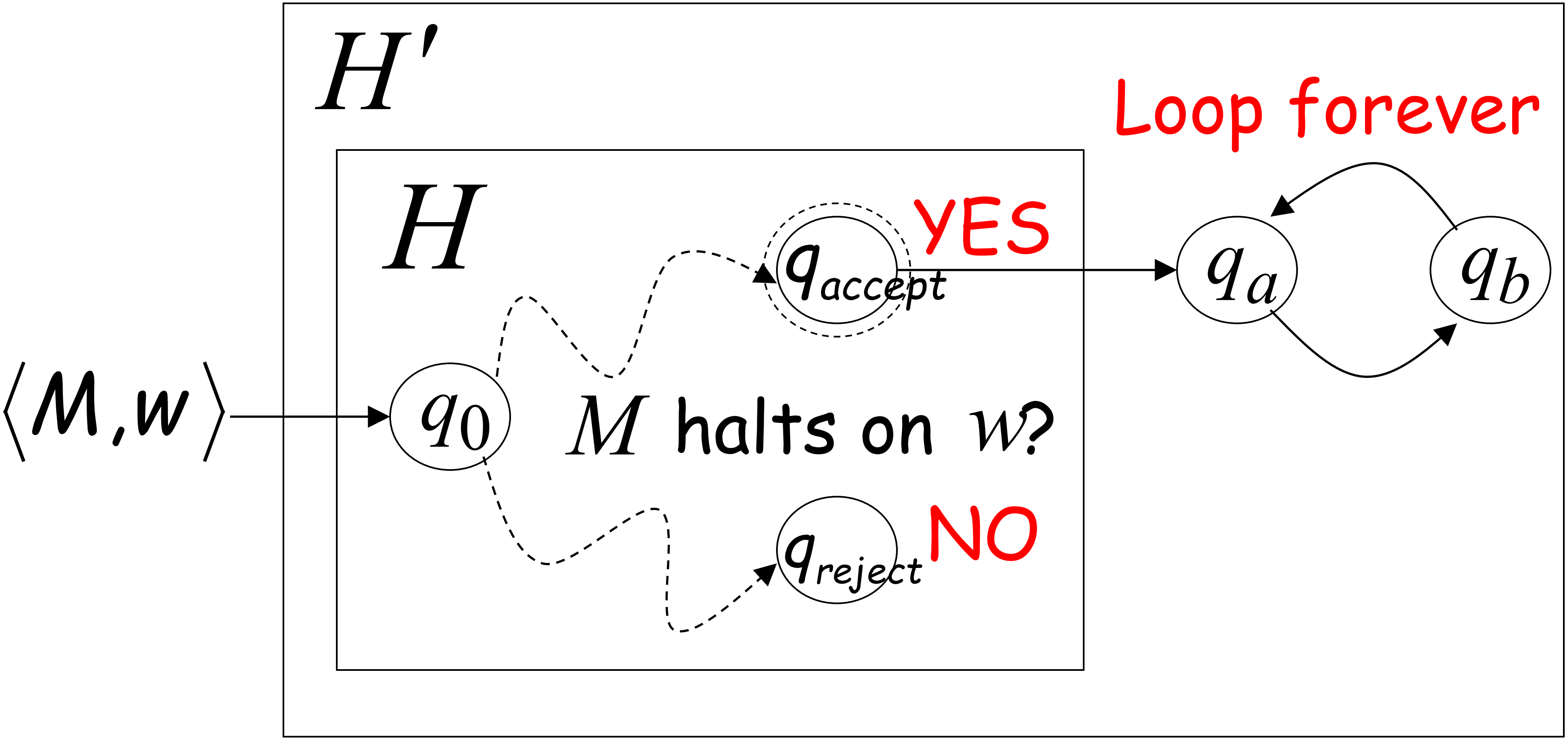


Guardiamo dentro H

Decider per $HALT_{TM}$



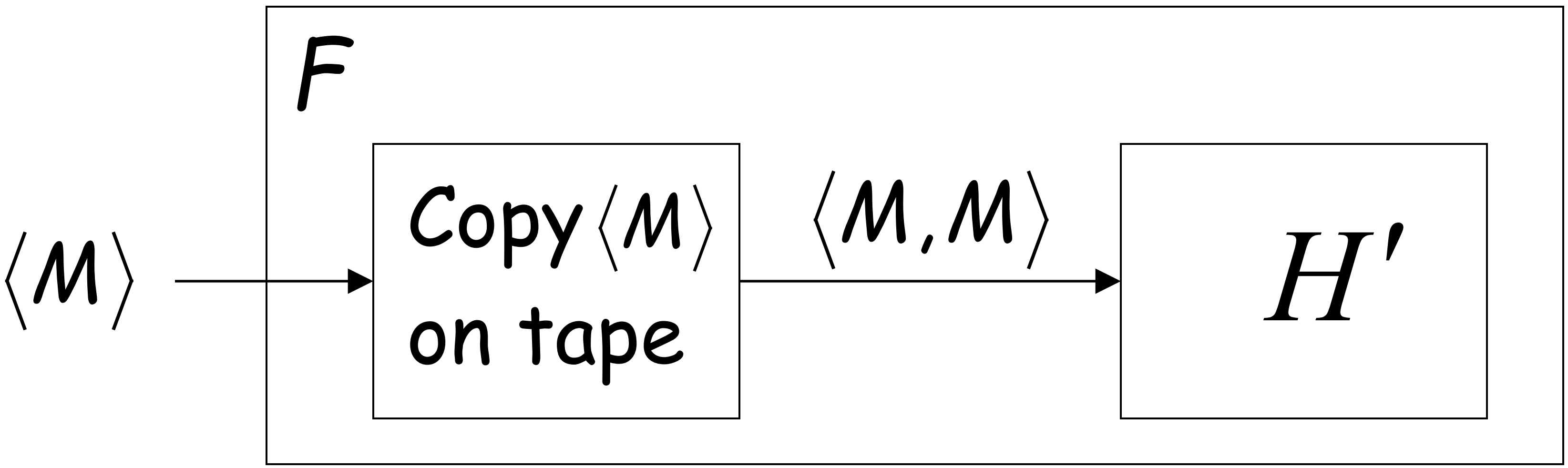
Costruiamo la macchina H' :



If M halts on input w **Then** Loop Forever

Else Halt

Costruiamo la macchina F:

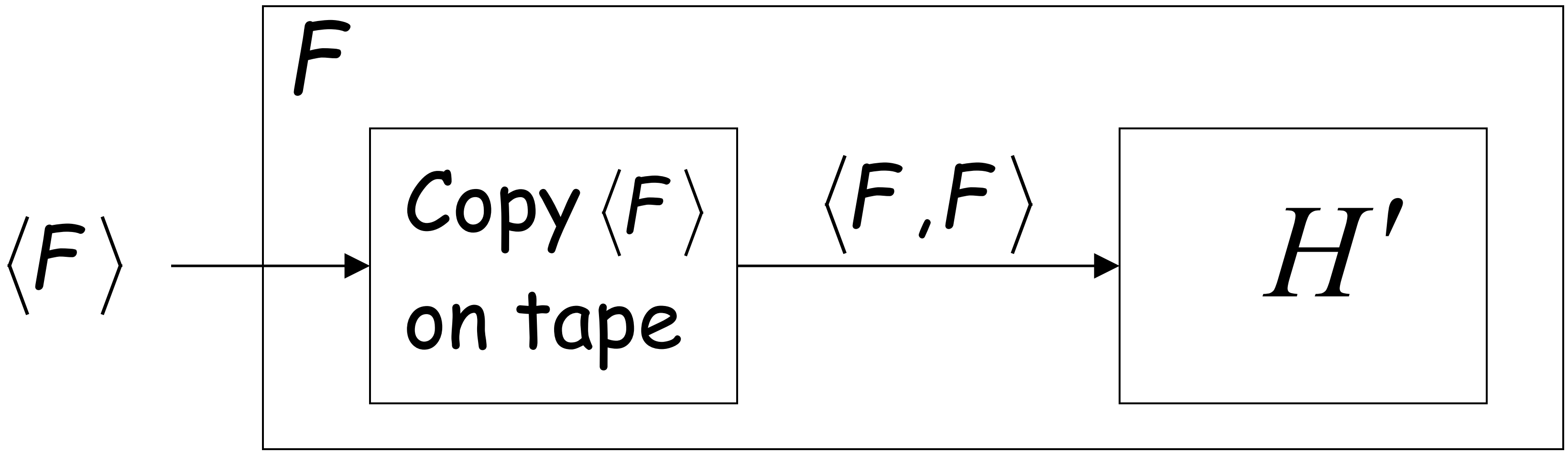


If M halts on input $\langle M \rangle$

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 terminazione*[8]*] Siano dati un alfabeto Γ ed una codificazione che associa ad ogni macchina di Turing $\mathcal{M} = \langle \Gamma, \flat, Q, \delta, q_0, F \rangle$ una sua codifica $c_{\mathcal{M}} \in \Gamma^*$. La funzione*

$$h(c_{\mathcal{M}}, x) = \begin{cases} 1 & \textit{se } \mathcal{M} \textit{ termina su input } x \\ 0 & \textit{se } \mathcal{M} \textit{ non termina su input } x \end{cases}$$

non è T-calcolabile.

90

h (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'(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:

$$h''(codice_M) \quad = 0 \text{ se } h'(codice_M) = 0$$
$$= \text{indefinito altrimenti}$$

h'(codice$_M$) = 1 se M (codice$_M$) termina

= 0 se M(codice$_M$) non termina

h"(codice$_M$) = 0 se h'(codice$_M$) = 0 **(se M(codice$_M$) non termina)**

= **indefinito altrimenti (se M (codice$_M$) termina)**

)

termina con 0 se h' si è fermata con 0 e

si mette a ciclare, se h' si è fermata con 1 calcoliamo $h''(\text{codice}_{h''})$:

$$h''(\text{codice}_{h''}) = \text{indefinito} \qquad \text{se } h''(\text{codice}_{h''}) \text{ è definita}$$
$$= 0 \qquad \text{se } h''(\text{codice}_{h''}) \text{ è indefinita}$$

In ogni caso abbiamo una contraddizione. Quindi non può esistere la macchina H.

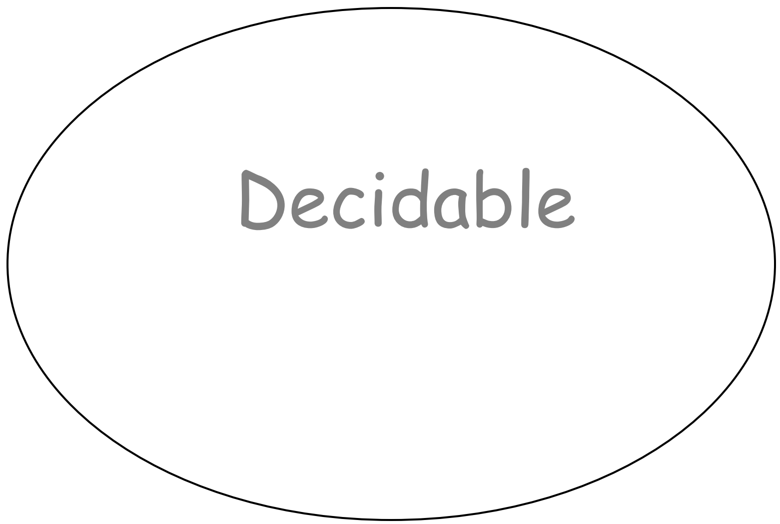
$$h''(D_M) = 0 \qquad \text{se } h'(D_M) = 0$$
$$= \text{indefinito} \qquad \text{altrimenti}$$

Fine
idem

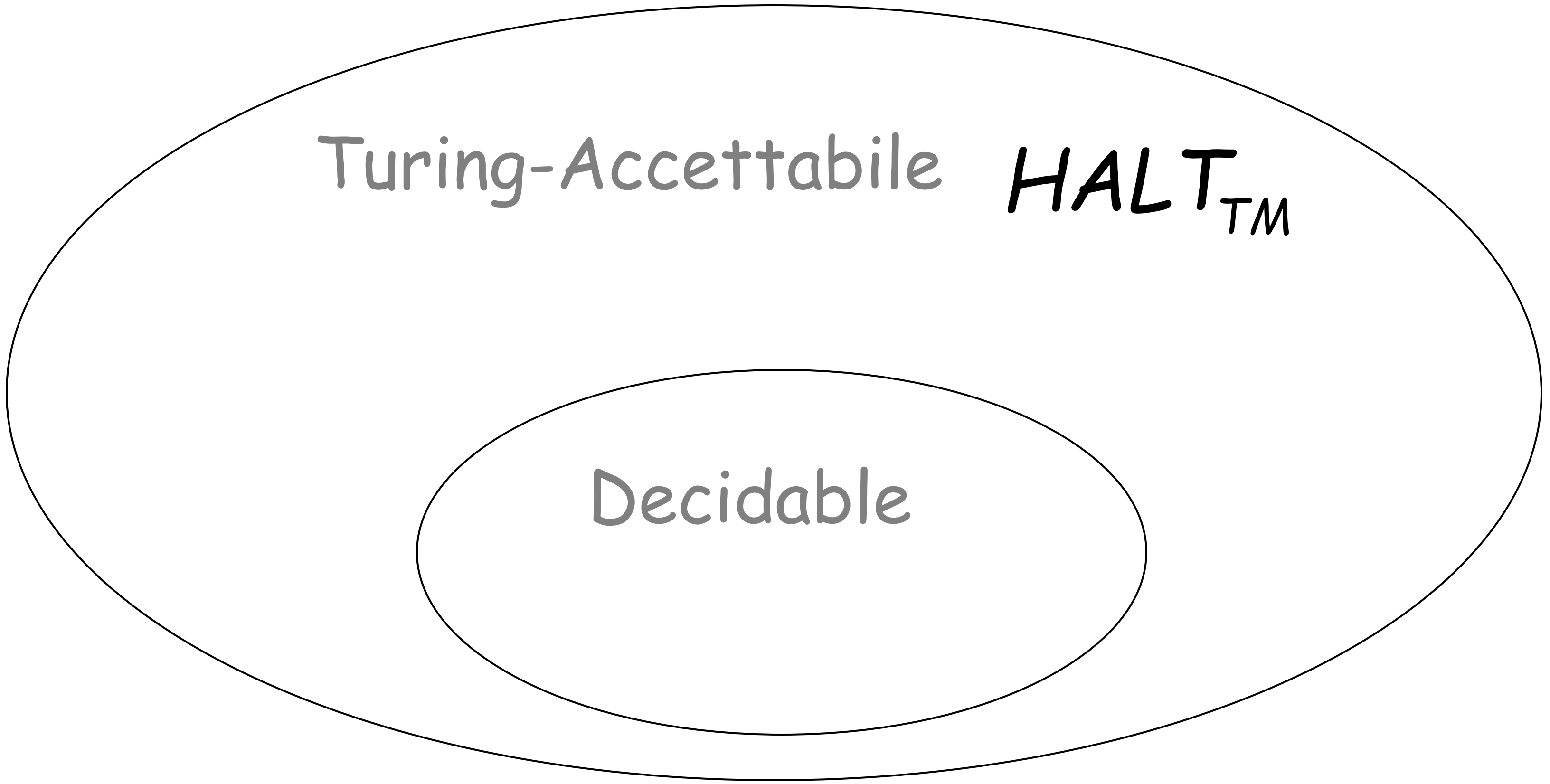
$$h'(D_M) = 1 \qquad \text{se } M\,(D_M) \text{ termina}$$
$$= 0 \qquad \text{se } M(D_M) \text{ non termina}$$

Abbiamo mostrato

indecidibile $HALT_{TM}$



Adesso proviamo che:



$HALT_{TM}$ è Turing-Acceptable (semidecidibile)

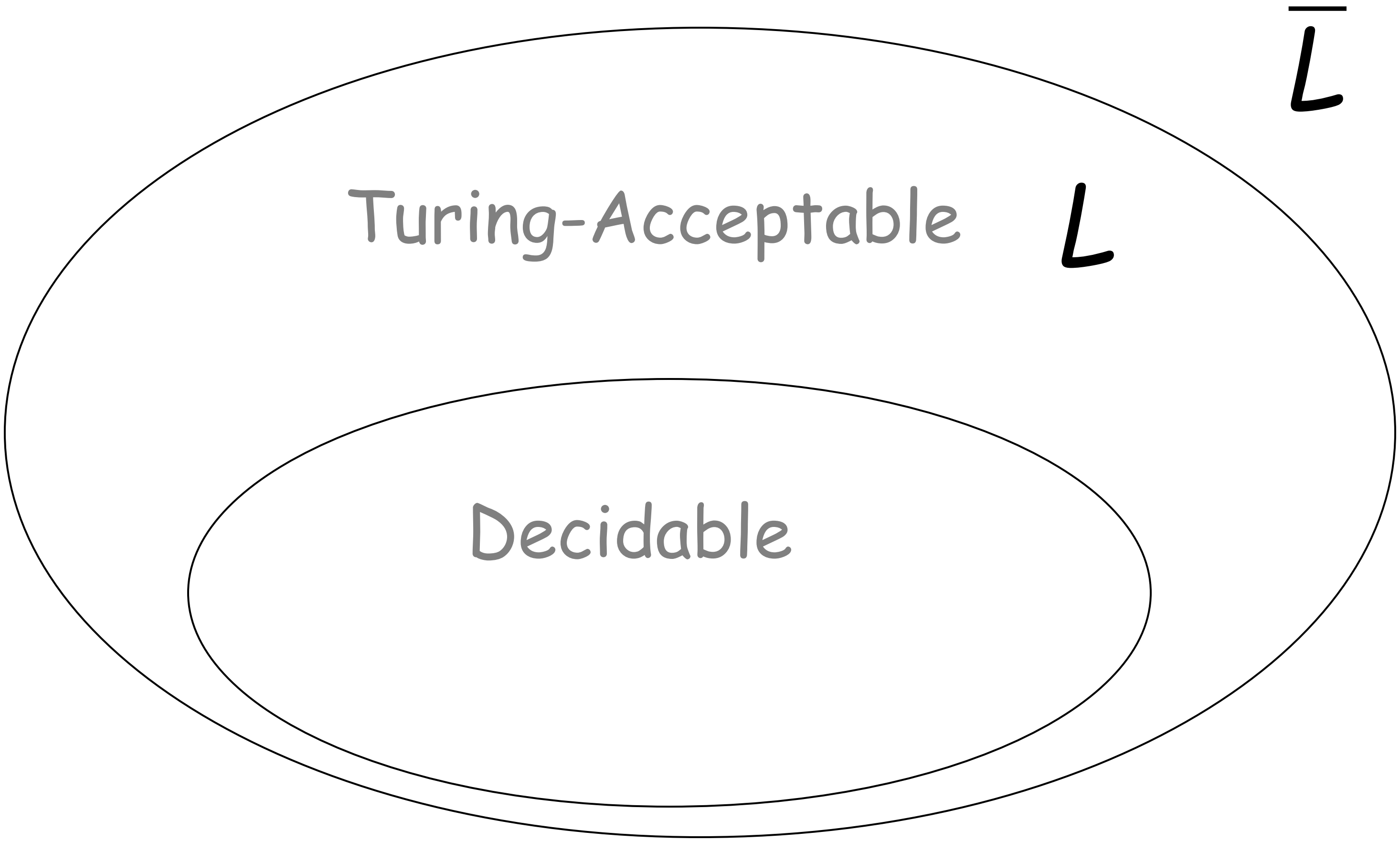
Turing machine che accetta $HALT_{TM}$:

$\langle M,w \rangle \longrightarrow$

1. Run M on input w

2. If M halts on w
   then accept $\langle M,w \rangle$

Abbiamo gia mostrato che esistono linguaggi indecidibili:



L

Tesi di Church Turing

Fine indecidibilità.

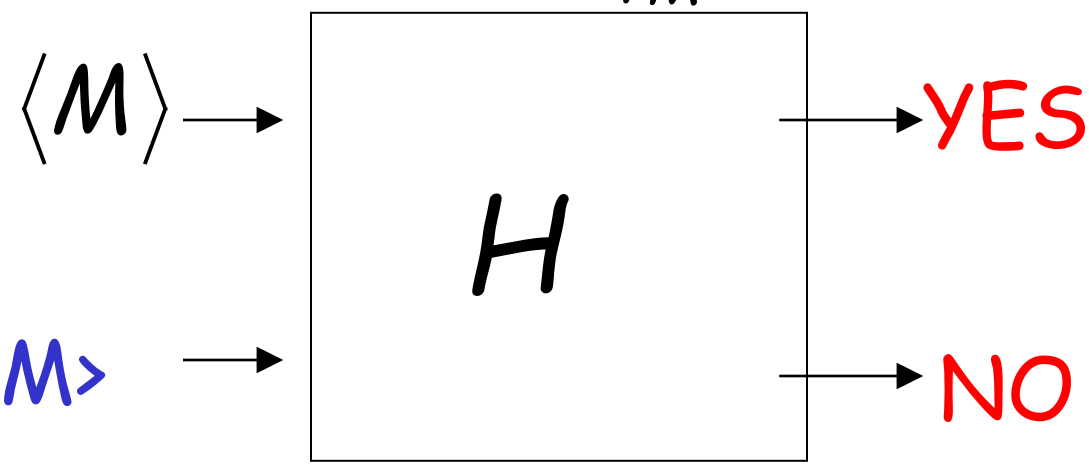
A_{TM}

Supponiamo w=<M> e
calcoliamo D(<M> , <M>)

Input string

Decisore per A_{TM}

$\langle M,w \rangle$



accetta

M rigetta <M>

M accetta <M>

rigetta

Teorema: A_{TM} è indecidibile

(The membership problem non è risolvibile)

Proof:

Idea di base:

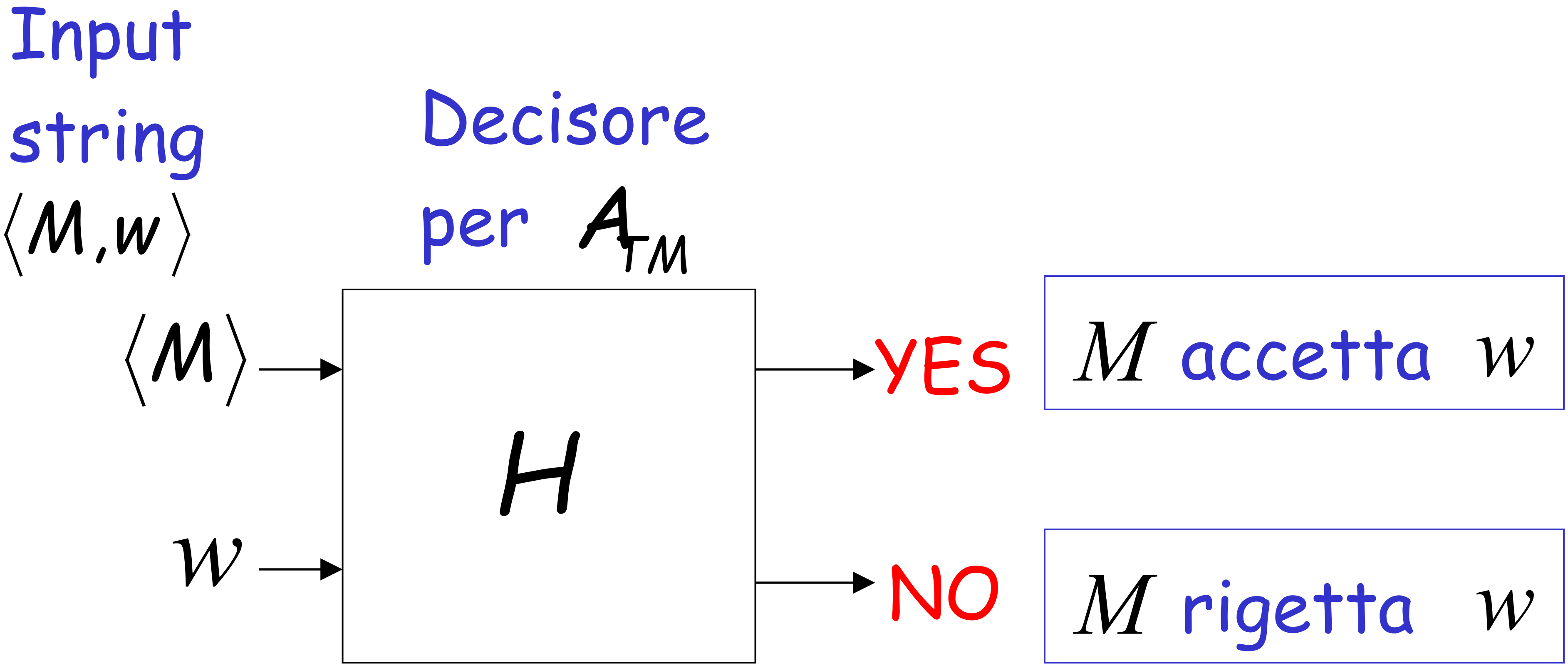
Assumiamo che A_{TM} è decidibile;

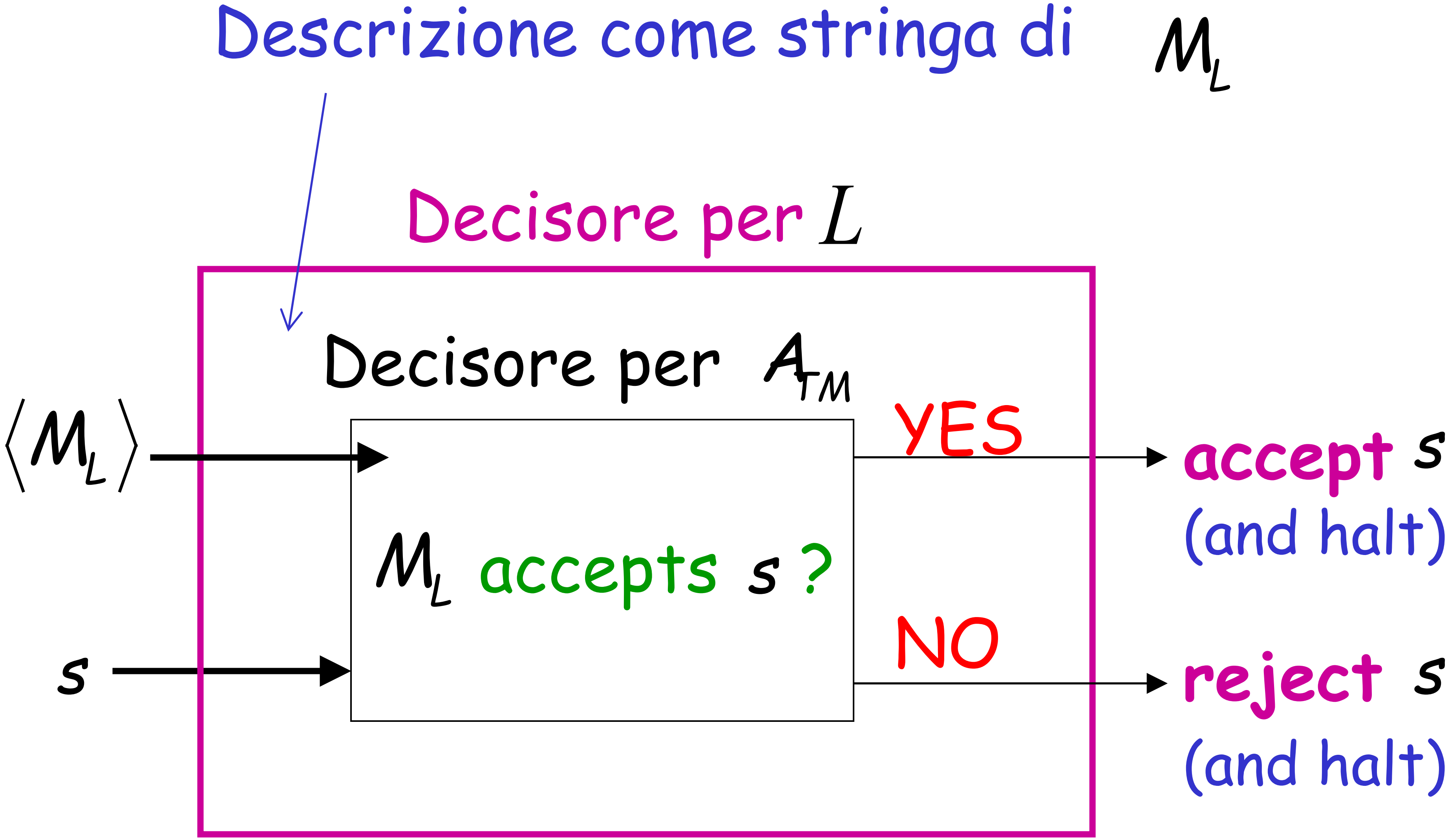
Proveremo che ogni linguaggio

è Turing-Acceptable

assurdo!

Allora avremo un decisore cosi definito:





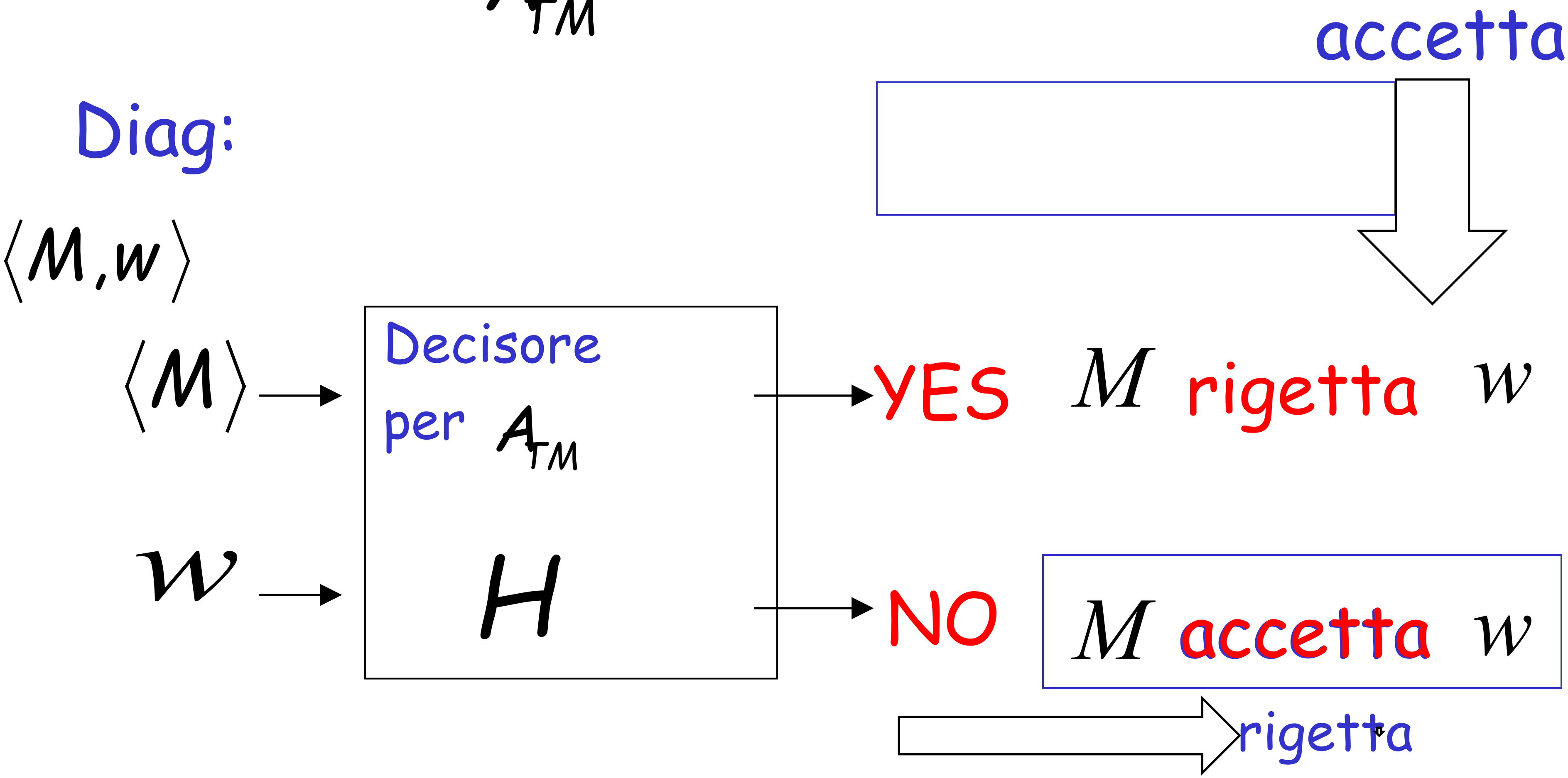
Sia M_L il decisore per il linguaggio L

Cambiamo, diagonalizziamo, chiamiamo la macchina Diag.

Diag accetta se A_{TM} dice yes ovvero se M(w) =no

Diag rigetta se A_{TM} dice no ovvero se M(w) =s

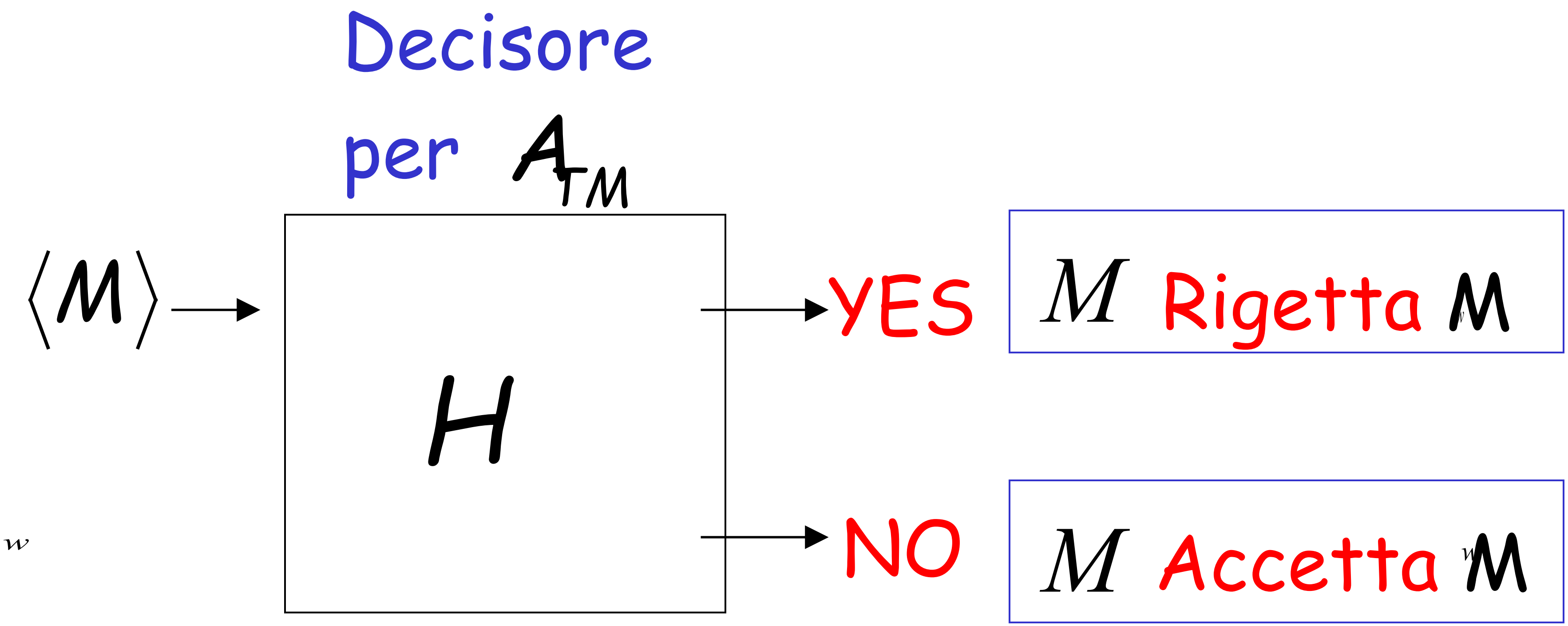
Diag:



Descrizione di Diag:

Diag accetta M se M rigetta M

Diag rigetta M se M accetta M



Mostriamo che: