

*Prof. Giovanni Pani*

*Dott.ssa Vita Santa Barletta*

JFLAP

# Macchine di TURING

# Macchina di Turing

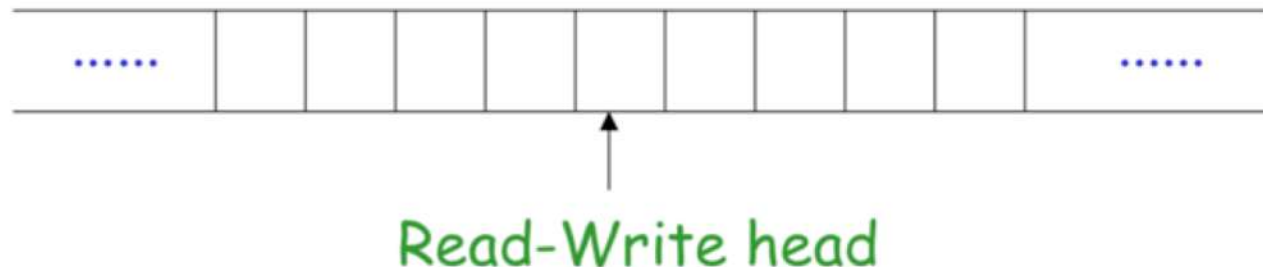
$M = (Q, \Sigma, \Gamma, \delta, q_0, \diamond, F)$

- $Q$ : insieme finito e non vuoto di stati
- $\Sigma$ : alfabeto di input
- $\Gamma$ : alfabeto dei simboli di nastro
- $\delta$ : funzione di transizione
- $q_0$ : stato iniziale
- $\diamond$ : simbolo di blank
- $F$ : insieme degli stati finali

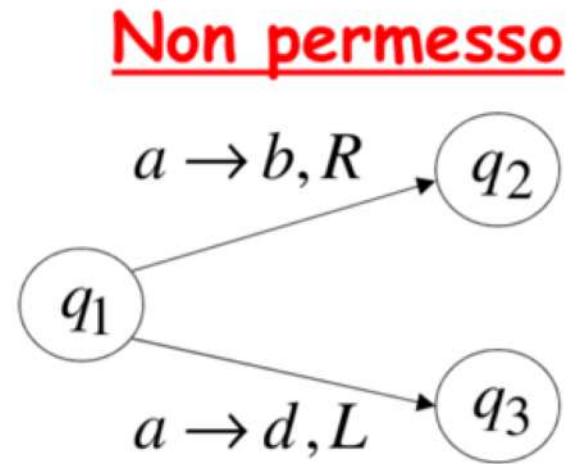
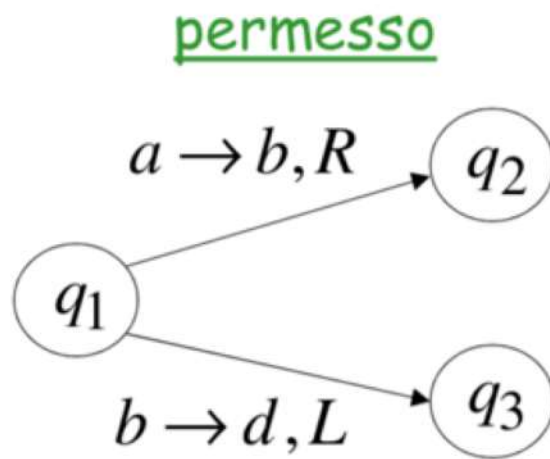
# Macchina di Turing

La head ad ogni transizione (time step):

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



# Macchina di Turing sono deterministiche



nessuna transizione lambda è permessa

# Stati di accettazione

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



**Non permesso**

# Linguaggi decidibili

Un linguaggio  $A$  è **decidibile**, se vi è una Turing Machine  $M$  (decisore) che accetta il linguaggio  $A$  e si ferma su ogni stringa di input

# Linguaggi indecidibili

Linguaggi indecidibili

=

Linguaggi non decidibili



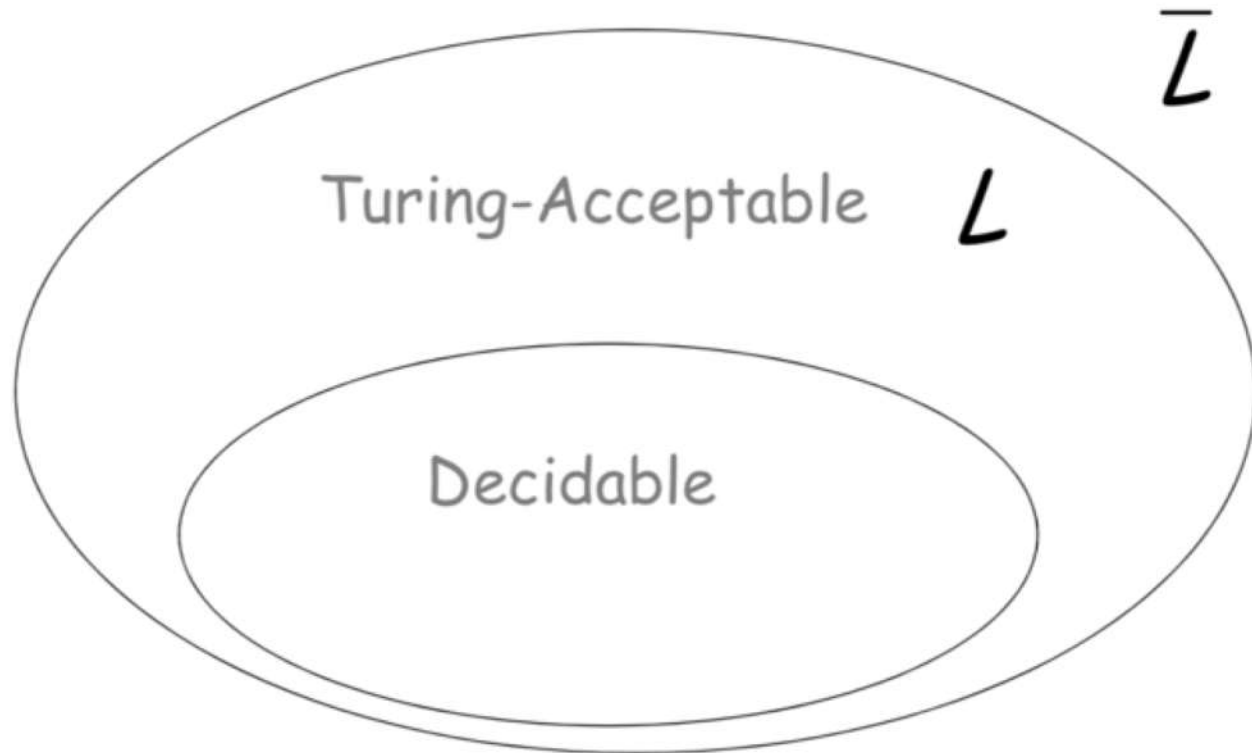
# Linguaggi indecidibili

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

# Linguaggio

## Turing-Acceptabile e decidibile

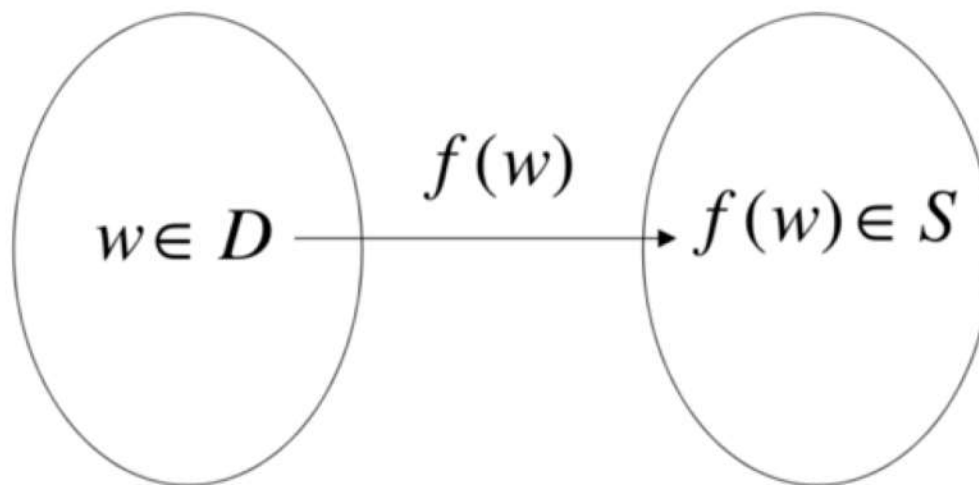


# Macchine di Turing: calcolare funzioni

Una funzione  $f(w)$  ha:

Dominio:  $D$

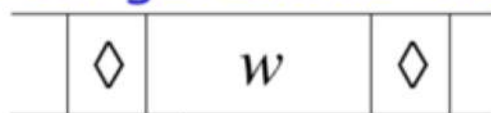
Regione dei risultati:  $S$



# Macchine di Turing: calcolare funzioni

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

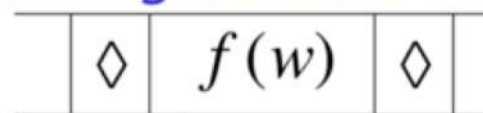
Configurazione iniziale



$q_0$

stato iniziale

Configurazione Finale

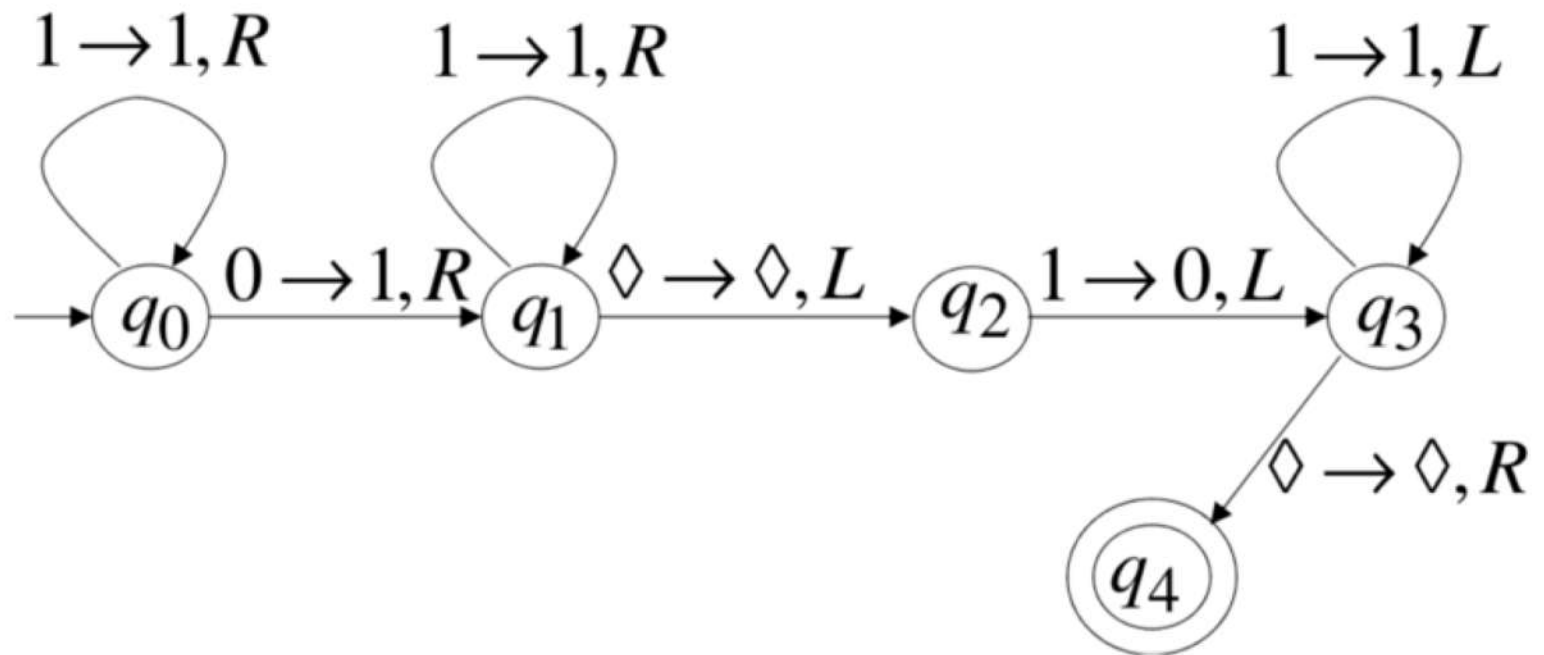


$q_f$

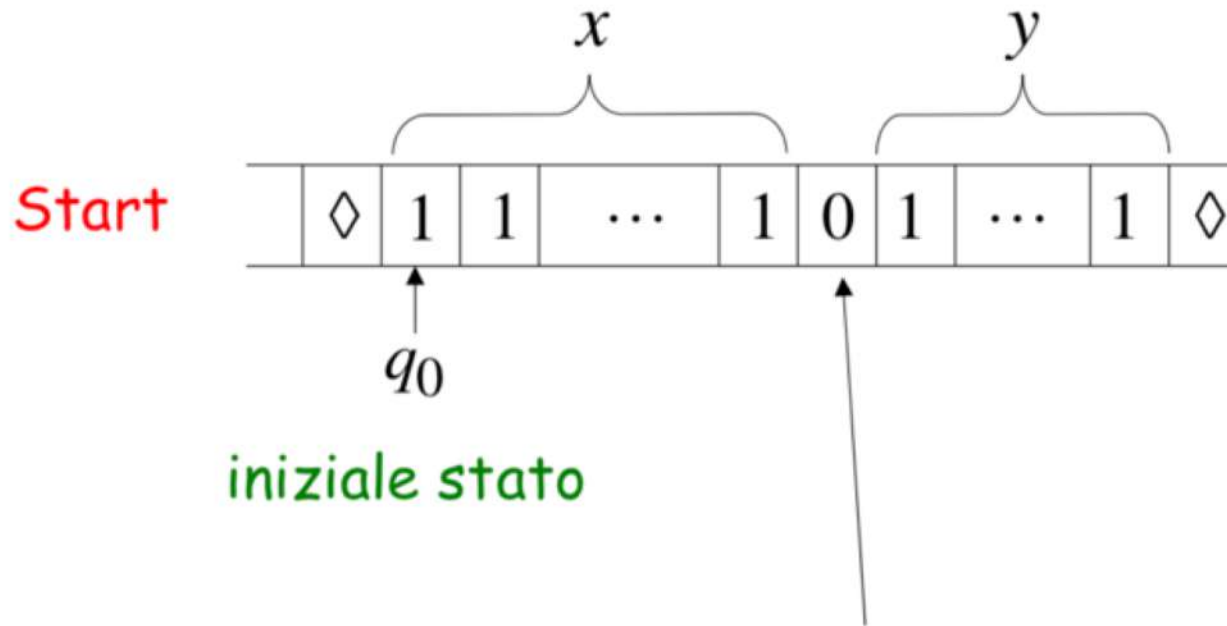
Stato di accettazione

Per tutti  $w \in D$  dominio

# Macchine di Turing: $f(x,y)=x+y$



# Macchine di Turing:

$$f(x, y) = x + y$$


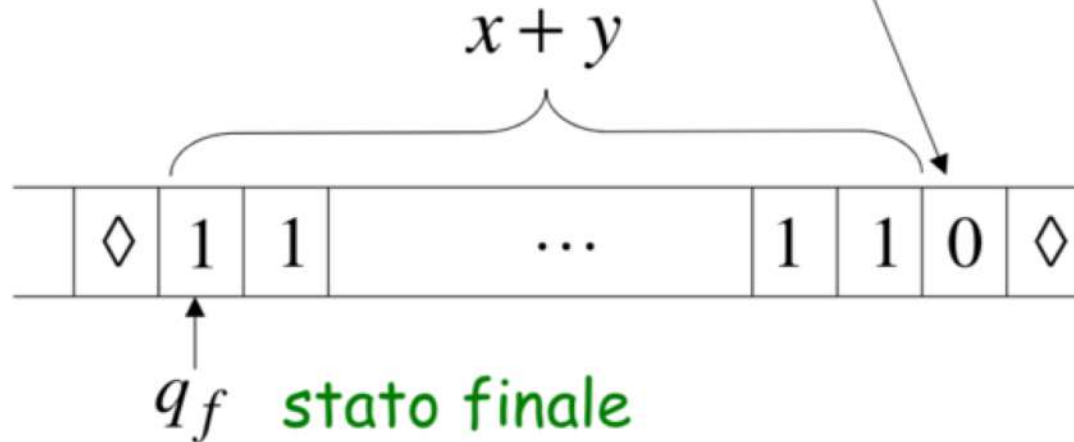
il 0 è il delimitatore che  
Separa I due numeri

# Macchine di Turing:

$$f(x, y) = x + y$$

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

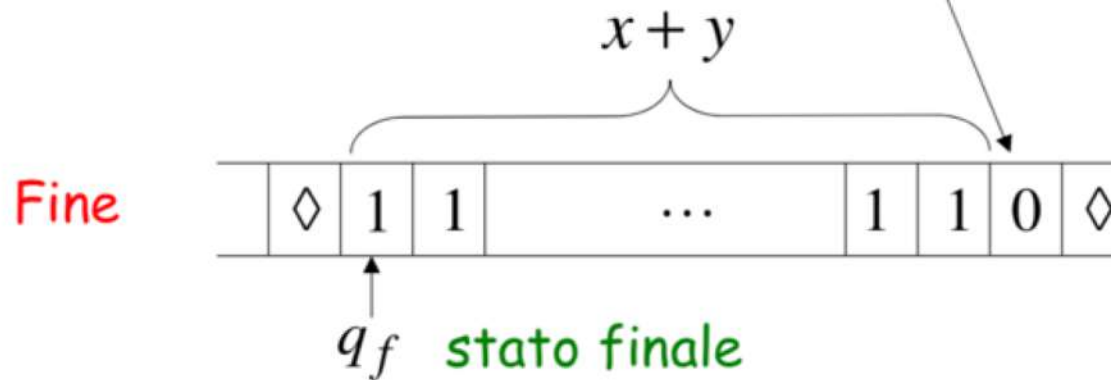
Fine



# Macchine di Turing:

$$f(x, y) = x + y$$

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





# Macchine di Turing:

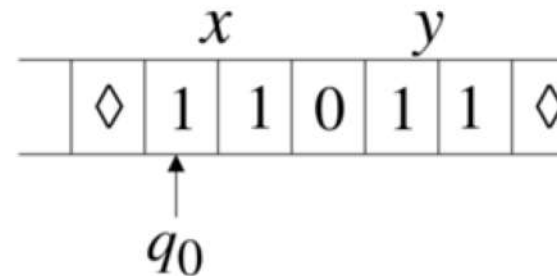
$$f(x, y) = x + y$$

esempio di esecuzione:

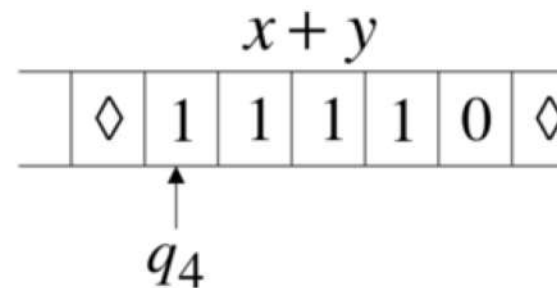
$x = 11$  (=2)

$y = 11$  (=2)

Time 0



Final Result



# Macchine di TURING UNIVERSALE

# Limitazione Macchina di Turing

Turing Machines eseguono  
**un solo programma per  
volta**

# Limitazione Macchina di Turing

Una TM per eseguire l'operazione  
tra due numeri seguirà  
**univocamente** ed  
**unicamente** quell'operazione

# Limitazione Macchina di Turing

# Soluzione

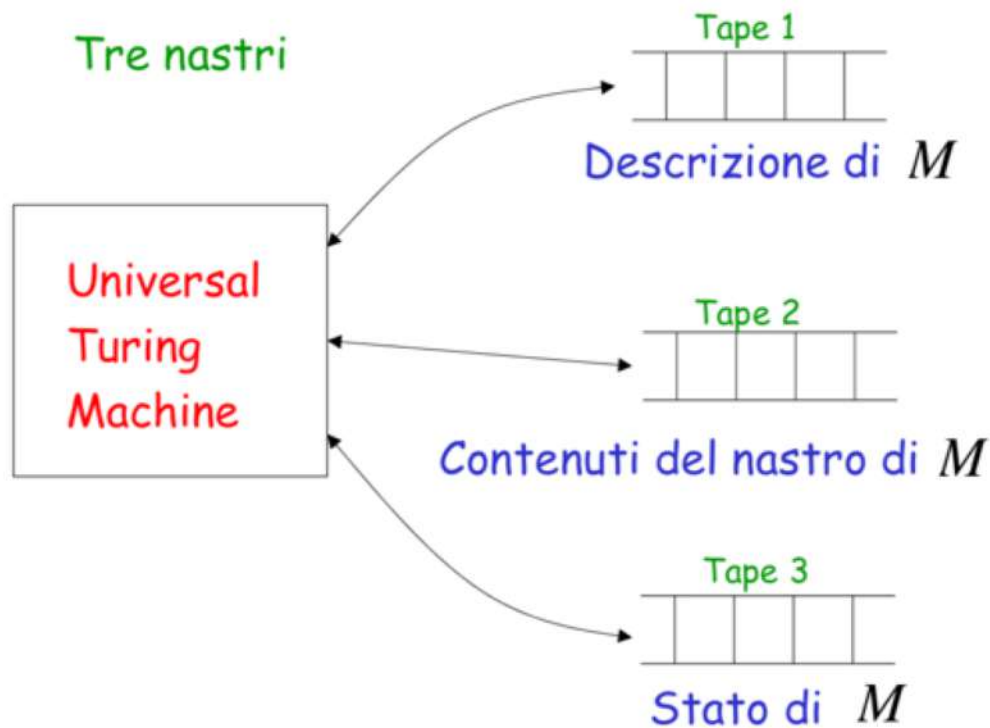
# Macchina di Turing Universale UTM

## Universal Turing Machine

Astrarre di un livello per simulare  
qualsiasi altra Turing Machine (TM)

# Universal Turing Machine

UTM sarà una MultiTape TM con 3 nastri



# Tape 1

Descriviamo la Turing Machine come una stringa di simboli:

- codifichiamo  $M$  come una stringa di simboli



# Tape 2

- Contiene l'input della macchina che stiamo simulando
- Lo stesso che avremmo inserito in una comune TM





# Tape 3

- Come nastro di lavoro che ospiterà ad ogni “passaggio” la  $\delta$  (transizione) da analizzare ed eseguire





# Codifica

- Necessità di codificare la Turing Machine come una stringa di simboli

# Codifica 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 della transizione

Transizione:	$\delta(q_1, a) = (q_2, b, L)$
	↓ ↓ ↓ ↓ ↓
codifica:	10101101101
	↑
	separatore

# Codifica Turing Machine

Transizione:

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

Codifica:

10101101101 00 1101101110111011

↑  
separatore

# Codifica

- Componente essenziale nella progettazione della Macchina di Turing Universale
- Per una UTM è una stringa  $w_{in} \in \Sigma^*$  su alfabeto  $\Sigma$
- Dobbiamo quindi codificare una TM , rappresentando la sua descrizione in forma "simbolica"

# Codifica

- Possiamo distinguere 3 differenti livelli logici di codifica:
  1. Codifica elemento atomico
  2. Codifica transizione
  3. Codifica macchina



# Codifica

- Scegliamo di codificare l'insieme  $N$  in alfabeto unario
- $\Sigma_{\text{encode}} = \{1\}$ 
  - es:  $0 = \langle 1 \rangle$ ,  $1 = \langle 11 \rangle$ ,  $2 = \langle 111 \rangle$ , ...;
  - definiamo quindi la funzione di codifica:
    - $\pi : \Sigma \cup Q \cup \{R, L\} \rightarrow \Sigma_{\text{encode}}^*$

# Codifica

**N.B.!!!**

- Nelle TM da simulare il dominio delle operazioni di spostamento rimane  $\{R, L\}$ ,
- Invece nell'implementazione della UTM usiamo, per comodità, anche l'identità  $S$  (non atomica) composta dalla concatenazione di  $R$  ed  $L$

# Codifica elemento atomico

- In maniera informale, preso un oggetto  $O$  ne indichiamo la sua codifica con  $\langle O \rangle$
- $\forall \sigma_i \in \Sigma: \pi(\sigma_i) = 1, 1, \dots, 1 = \langle \sigma_i \rangle$  con  $0 \leq i \leq |\Sigma|$
- $\forall q_i \in Q: \pi(q_i) = 1, 1, \dots, 1 = \langle q_i \rangle$  con  $0 \leq i \leq |Q|$
- $\pi(R) = 1 = \langle R \rangle, \quad \pi(L) = 11 = \langle L \rangle$

# Codifica transizione

Possiamo identificare una singola transizione con una 5-upla

$$(q_{in}, \sigma_r, q_{out}, \sigma_w, op.code)$$

- $q_{in}$  : stato corrente (diagramma a stati finiti)
- $\sigma_r$  : simbolo letto
- $q_{out}$  : nuovo stato
- $\sigma_w$  : simbolo da scrivere
- $op.code$ : operazione di spostamento (right, left)

# Codifica transizione

## Introduzione di un nuovo simbolo

- Lo zero(0) in modo che esso funga da separatore tra elementi della transizione e transizioni stesse (00)
- Di conseguenza estendiamo il nostro albeta di codifica

$$\Sigma^*_{\text{encode}} = \{1\} \cup \{0\}$$

# Codifica transizione

Utilizzando la funzione  $\pi$  possiamo codificare una generica transizione

$$\langle (q_{in}, \sigma_r, q_{out}, \sigma_w, op.code) \rangle = \pi(q_{in})0\pi(\sigma_r)0\pi(q_{out})0\pi(\sigma_w)0\pi(op.code)00$$

# Codifica transizione

- Durante l'implementazione, la struttura della transizione può essere modificata per ottimizzarne l'analisi
- A livello semantico, per convenzione, dividiamo la transizione in 2 sezioni:

$$(q_{in}, \sigma_r, q_{out}, \sigma_w, op.code)$$

in-section                      out-section

# Codifica transizione

- Per la sezione *in*, eliminiamo la componente  $q_{in}$  poiché definiamo un ordinamento totale sull'insieme delle transizioni
- Riguardo la sezione *out* modifichiamo l'ordine delle componenti utilizzando quella che definiamo "notazione polacca modificata" (es:  $0\#1 \rightarrow \#10$ )
- La struttura dell'istruzione si riduce quindi alla seguente 4-upla:

$$(\sigma_r, \sigma_w, op.code, q_{out})$$



# Codifica macchina

- Codificare l'intera  $TM$
- Essenzialmente la codifica  $\langle M \rangle$  di una  $TM$   $M$ , si riduce a descrivere tutte le singole transizioni
- Definiamo quindi l'insieme di tutte le  $\delta$  (transizioni) di  $M$ :

$$\Delta = \{ \text{tutte le 4-uple } (\sigma_r, \sigma_w, op.code, q_{out}) \text{ di } M \}$$

- Quindi la codifica di  $M$  risulta:

$$\langle M \rangle = \langle \Delta \rangle = \pi(\sigma_r)0\pi(\sigma_w)0\pi(op.code)0\pi(q_{out})00$$

.....

$$\pi(\sigma_r)0\pi(\sigma_w)0\pi(op.code)0\pi(q_{out})00$$

# Codifica macchina

A livello implementativo scaturiscono delle considerazioni

- Definire un ordine totale implicito tra le transizioni, ovvero, nell'insieme  $\Delta$  tutte le transizioni sono ordinate in base al valore numerico dello stato corrente (e poi in base al loro carattere di input), evitando così di dover codificare la componente  $q_{in}$  dell'istruzione e quindi la ricerca (randomica) della stessa
  - Si tratterà solo di "saltare" le  $n-1$  istruzioni per trovare l' $n$ -esima desiderata

# Codifica macchina

- Come conseguenza abbiamo definito totalmente la funzione  $\delta$ , e quindi previsto uno stato di errore  $q_e$

$$\forall q_i \in Q - F - \{q_e\}, \quad \forall \sigma_i \in \Sigma: (q_i, \sigma_i) = \emptyset \Rightarrow (q_i, \sigma_i) = (q_e, E, S)$$

e aggiunto poi delle transizioni "dummy" le quali da uno stato  $q_i$  corrente fanno transitare la macchina nello stato di errore  $q_e$  scrivendo sul tape 2 il carattere simbolico E per segnalare l'avvenuto errore di computazione

In generale, dato l'alfabeto di nastro  $\Gamma$ , per trovare la k-esima istruzione dovremmo saltare  $p$  istruzioni, con:

$$p = |\Gamma| \cdot (k - 1)$$

# Codifica macchina

- In ultimo, assegnando un numero progressivo ad ogni stato, si perderebbe la semantica di  $q_e$  e  $q_f$  che verrebbero identificati cos' come tutti gli altri, pur essendo degli stati "speciali"

Introduciamo, pertanto, un nuovo simbolo, il dollaro (\$), per indicare durante l'analisi dell'istruzione se  $q_{out}$  è uno stato finale (\$) o uno stato di errore (\$\$)

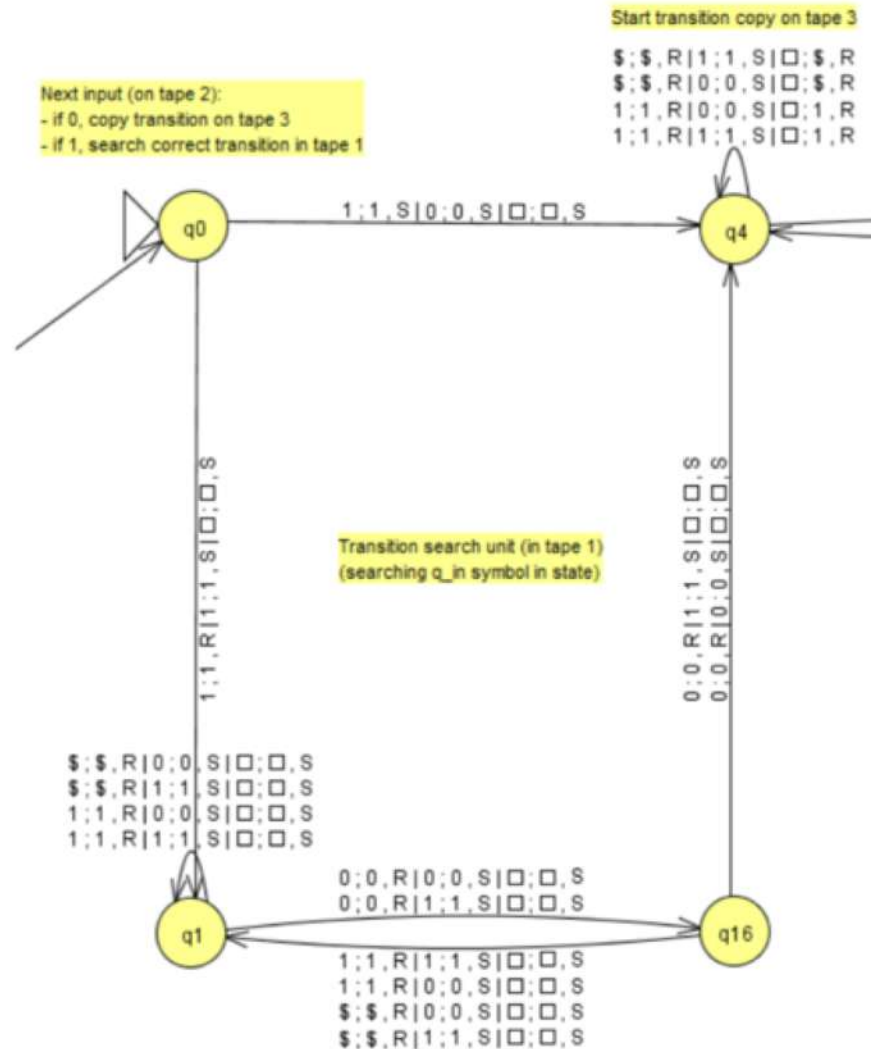
- Codificando:
  - stato finale:  $\langle q_f \rangle = \pi(q_f) = \$$
  - stato di errore:  $\langle q_e \rangle = \pi(q_e) = \$\$$
- Evitiamo in questo modo di introdurre un nuovo simbolo per lo stato di errore che avrebbe aumentato esponenzialmente le transizioni per l'UTM
- Aggiorniamo l'alfabeto di codifica:

$$\Sigma_{\text{encode}} = \{1\} \cup \{0\} \cup \{\$, \$\}$$

# Transition Search Unit (in Tape 1)

- Se sul tape 2 il carattere osservato è 0 (prima  $\delta$  utile) passa direttamente all'unità di copia
- Se è 1 questa macchina esegue la ricerca della transizione giusta, facendo riferimento a  $\sigma_r$  puntato sul tape 2
- La testina del tape 1 si muove di una  $\delta$  alla volta tra le transizioni di un prestabilito stato  $q_{in}$

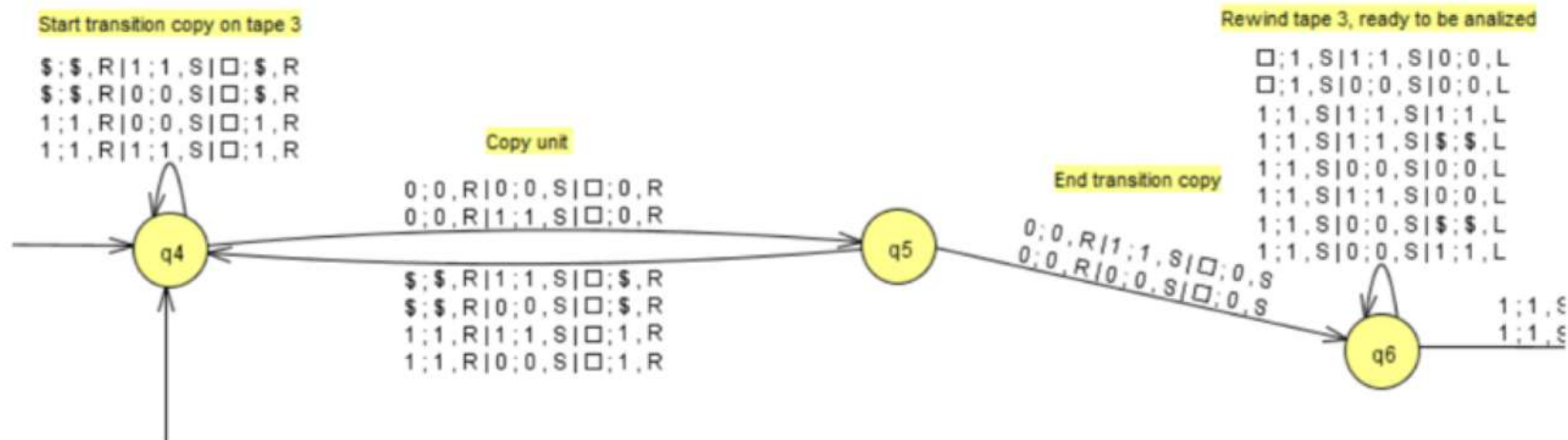
# Transition Search Unit (in Tape 1)



# Copy Unit

- Copia la transizione, sulla quale si è fermata la testina del tape 1 nella sottomacchina precedente, sul tape 3 (di lavoro)
- Alla fine della copia esegue un rewind sul tape 3, pronto per essere scandito

# Copy Unit





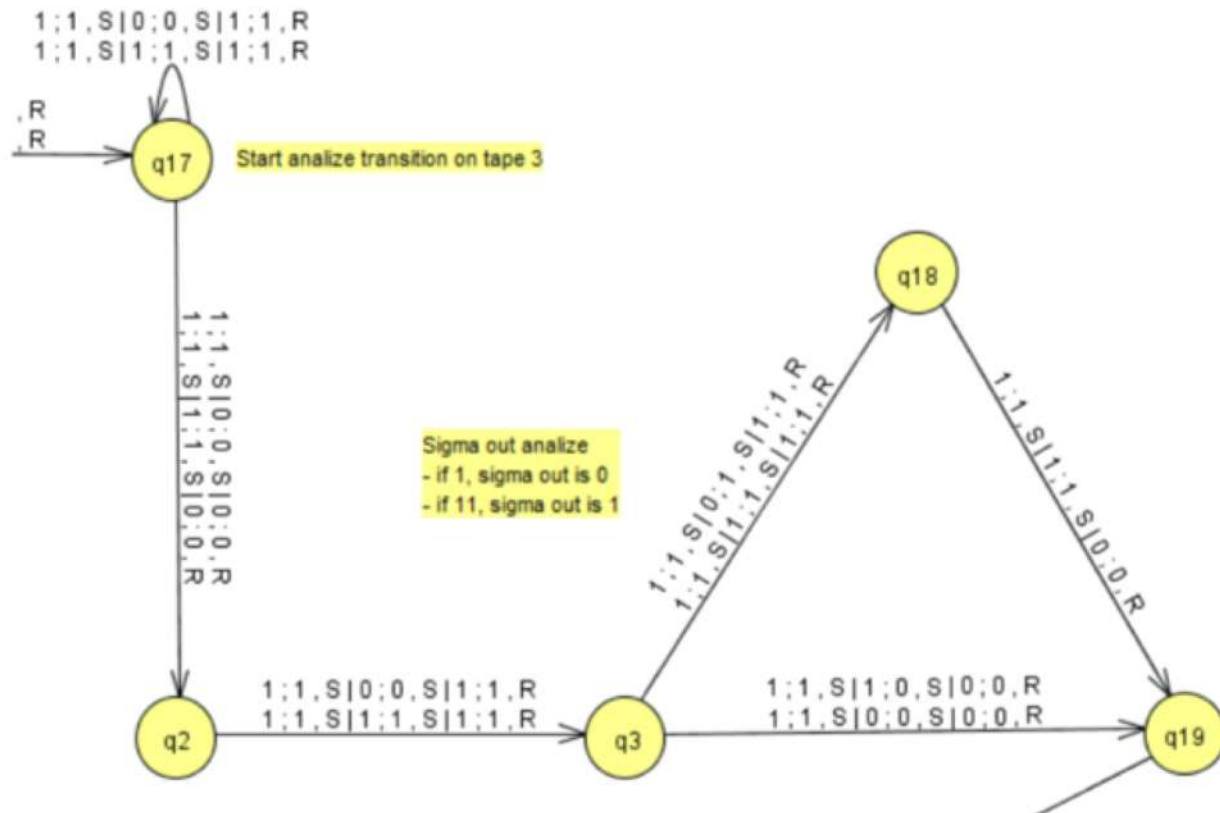
# Transition Analyzer

- Questa sottomacchina, cuore della UTM , analizza l'intera parte output della transizione presente sul tape 3
- Per farlo ha bisogno di 3 unità

# Transition Analyzer: Sigma out ( $\sigma_{\text{out}}$ ) Analyzer

- Sigma out ( $\sigma_{\text{out}}$ ) Analyzer: determina il carattere  $\sigma_w$  che sovrascriverà la testina sul tape 2

# Transition Analyzer: Sigma out ( $\sigma_{out}$ ) Analyzer



# Transition Analyzer: Move Analyzer

- Move Analyzer: sceglie la mossa *op.code* che deve eseguire la testina sul tape 2, tra R, L, S

Alla fine esegue un rewind del tape 1 pronto per essere scandito

Rewind tape 1, ready to be choosed by q\_out (in tape 3)

1;1,L|0;0,S|1;1,S  
 0;0,L|1;1,S|1;1,S  
 1;1,L|1;1,S|1;1,S  
 \$;\$,L|1;1,S|1;1,S  
 0;0,L|0;0,S|1;1,S  
 \$;\$,L|0;0,S|1;1,S

q21

1;1,S|1;1,R  
 0;0,S|1;1,R

1;1,S|1;1,R|0;0,R  
 1;1,S|0;0,R|0;0,R

q20

1;1,S|1;1,R  
 1;1,S|0;0,L|0;0,R

q22

1;1,S|1;1,S|1;1,R  
 1;1,S|0;0,S|1;1,R

Move analyze  
 - if 1, next move is Right  
 - if 11, next move is Left

# Transition Analyzer: Transition Search Unit

- Transition Search Unit (in Tape 2):  
esegue la ricerca dello stato  $q_{out}$  nel  
quale trovare la prossima  
transizione che verrà analizzata  
La testina del tape 1 si muove di

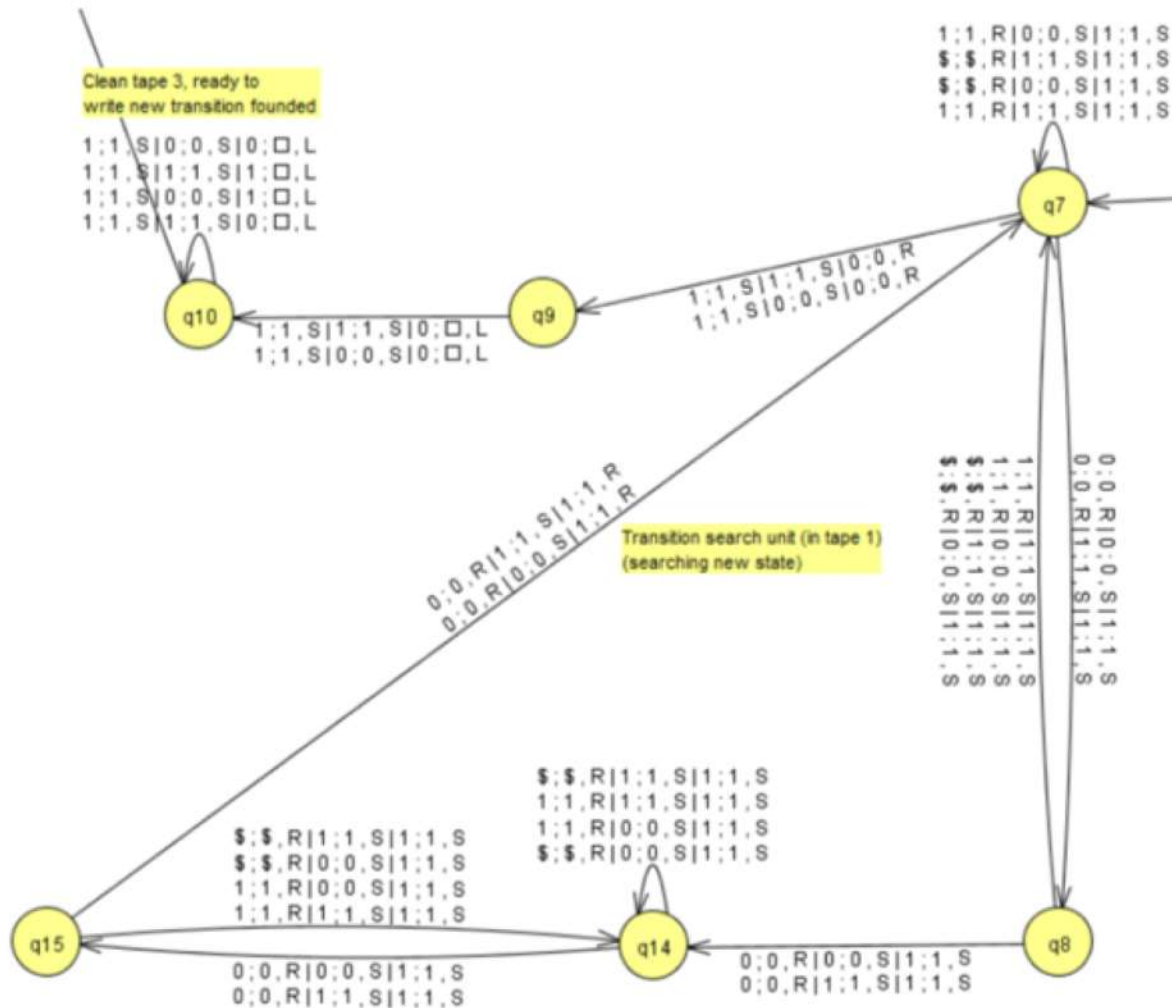
$2n-1$   $\delta$  alla volta

con  $n$  pari al numero di simboli  
dell'alfabeto di input

# Transition Analyzer: Clean Tape 3 Unit

- Clean Tape 3 Unit: pulisce il tape 3 che ospiterà la prossima transizione ricercata dalla TSU (in Tape 1)

# Transition Analyzer: Transition Search Unit





# Error/Final Control

- Gestisce gli stati "speciali" quali il finale (nel caso nella transizione ci fosse \$) e quello di errore (nel caso si trovasse \$\$)

# Error/Final Control

