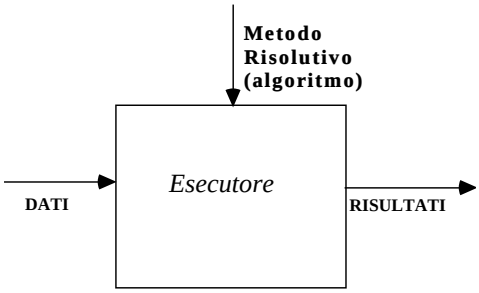
L'AUTOMA ESECUTORE

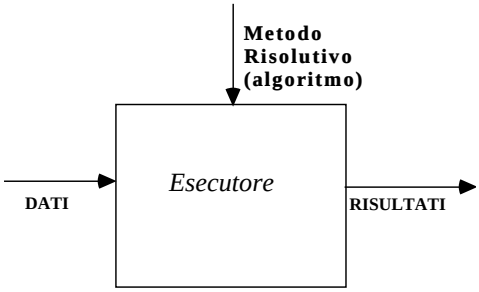


Un automa capace di **ricevere dall'esterno una descrizione dello algoritmo richiesto**

cioè

capace di **interpretare un linguaggio** (***linguaggio macchina***)

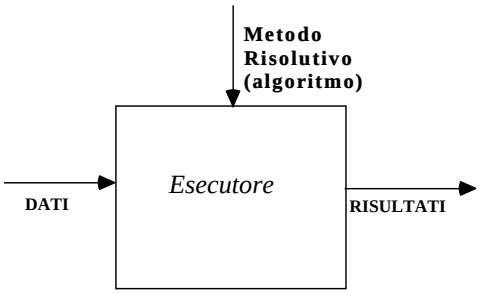
L'AUTOMA ESECUTORE



Vincolo di realizzabilità:

- se l'automa è fatto di parti, queste sono in numero finito
- ingresso e uscita devono essere denotabili attraverso un insieme finito di simboli.

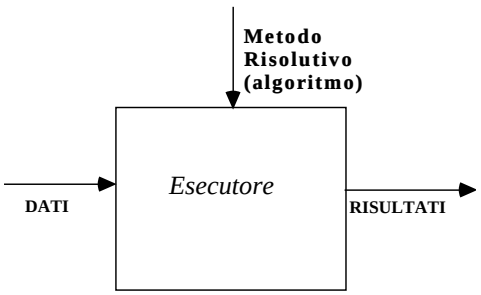
L'AUTOMA ESECUTORE



Realizzazione:

1) mediante **congegni meccanici**
   - macchina aritmetica (1649) di Blaise Pascal
   - macchina analitica di Charles Babbage (1792-1871)

L'AUTOMA ESECUTORE



Realizzazione:

2) mediante **un modello matematico**
   - funzionale (Hilbert, (1842-1943), Church, Kleene)
   - operazionale (Turing, 1912-1954)
   - sistemi di riscrittura (Post, Markov,..).

PERCHÉ I MODELLI MATEMATICI

Macchine diverse potrebbero avere **diversa capacità di risolvere problemi**

Una macchina potrebbe essere "più potente" di un'altra
È NECESSARIO SAPERLO

Se neanche la macchina "più potente" riesce a risolvere un problema, **potrebbero esserci PROBLEMI NON RISOLUBILI**

GERARCHIA DI MACCHINE

- macchine combinatorie
- macchine (automi) a stati finiti
- macchina a stack
- **macchina di Turing**

TESI DI CHURCH-TURING
Non esiste alcun formalismo capace di risolvere una classe di problemi più ampia della Macchina di Turing

LA MACCHINA DI TURING

Formalmente definita dalla quintupla:

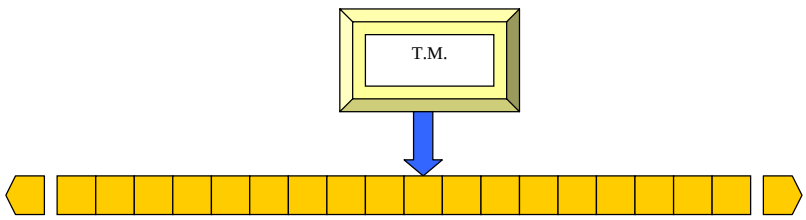
$$\langle \mathbf{A, S, mfn, sfn, dfn} \rangle$$

dove

- **A** = insieme finito dei **simboli di ingresso e uscita**
- **S** = insieme finito degli **stati** (di cui uno è *HALT*)
- **mfn: A × S → A** ***(funzione di macchina)***
- **sfn: A × S → S** ***(funzione di stato)***
- **dfn: A × S → D = {Left,Right,None}**
  (funzione di direzione)

LA MACCHINA DI TURING

Un *nastro* (illimitatamente espandibile)
rappresenta il deposito dei dati (***memoria***)



LA MACCHINA DI TURING

La TM è una macchina capace di:

- *leggere un simbolo* dal nastro
- *scrivere sul nastro il simbolo* specificato da `mfn()`
- transitare in un nuovo stato interno specificato da `sfn()`
- spostarsi sul nastro di una posizione nella direzione indicata da `dfn()`

Quando raggiunge lo stato HALT, la macchina si ferma

LA MACCHINA DI TURING E CPU REALI

In pratica:	corrisponde a:
• leggere / scrivere un simbolo dal / sul nastro	• lettura / scrittura dalla / sulla memoria RAM / ROM
• transitare in un nuovo stato interno	• nuova configurazione dei registri della CPU
• spostarsi sul nastro di una (o più) posizioni	• scelta della cella di memoria su cui operare (indirizzo contenuto nel RI)

RISOLVERE PROBLEMI CON LA T.M.

Risolvere un problema con la Turing Machine richiede quindi di:

- **definire** una opportuna ***rappresentazione dei dati iniziali sul nastro***
- **definire la *parte di controllo***, cioè le tre funzioni:

  `mfn()` `sfn()` `dfn()`

  in modo da **rendere disponibile sul nastro**, ***alla fine*****,** la ***rappresentazione della soluzione***

ESEMPIO: RICONOSCERE PALINDROMI

Problema:
riconoscere palindromi (di cifre binarie)

♦ 1 0 1 1 0 1 ♦

Soluzione:

a) definire un algoritmo

b) “programmarlo” sulla Turing Machine

ESEMPIO: RICONOSCERE PALINDROMI

Problema:
riconoscere palindromi (di cifre binarie)

♦ ♦ 0 1 1 0 1 ♦

Un possibile algoritmo
- leggere il primo simbolo a sinistra, ***ricordare se è 1 o 0*** e marcare tale casella con ♦
- spostarsi sull'ultimo simbolo e leggerlo
- se è diverso dal primo (*che qui era 1, ndr*), scrivere E (errore) e terminare (*non è il nostro caso*)

ESEMPIO: RICONOSCERE PALINDROMI

Problema:
riconoscere palindromi (di cifre binarie)

♦ ♦ 0 1 1 0 ♦ ♦

Un possibile algoritmo *(segue)*
- altrimenti, marcare tale casella con ♦ e ricominciare il controllo sulla stringa più corta rimasta (procedendo a ritroso)

Alla fine.. quando non ci sono più cifre fra ♦, scrivere T (true) e finire

ESEMPIO: RICONOSCERE PALINDROMI

Evoluzione (1)
Situazione iniziale

♦ 1 0 1 1 0 1 ♦

e poi via via…

♦ ♦ 0 1 1 0 1 ♦

♦ ♦ 0 1 1 0 1 ♦

♦ ♦ 0 1 1 0 ♦ ♦

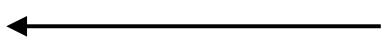
(segue)

ESEMPIO: RICONOSCERE PALINDROMI

Evoluzione (2)
Situazione

♦ ♦ 0 1 1 0 ♦ ♦

e poi:



♦ ♦ 0 1 1 ♦ ♦ ♦

♦ ♦ 0 1 1 ♦ ♦ ♦

♦ ♦ ♦ 1 1 ♦ ♦ ♦

(segue)

ESEMPIO: RICONOSCERE PALINDROMI

Evoluzione (3)

Situazione

♦ ♦ ♦ 1 1 ♦ ♦ ♦

e poi:

♦ ♦ ♦ ♦ 1 ♦ ♦ ♦

♦ ♦ ♦ ♦ 1 ♦ ♦ ♦

♦ ♦ ♦ ♦ T ♦ ♦ ♦

(FINE)

ESEMPIO: RICONOSCERE PALINDROMI

Problema:

riconoscere palindromi (di cifre binarie)

♦ 1 0 1 1 0 1 ♦

Soluzione:

a) definire un algoritmo

b) "programmarlo" sulla Turing Machine, cioè...

definire la *parte di controllo* data dalle tre funzioni `mfn()` `sfn()` `dfn()`

ESEMPIO: RICONOSCERE PALINDROMI

DEFINIRE LA PARTE DI CONTROLLO

Osservazione: le tre funzioni `mfn()` `sfn()` `dfn()` hanno il ***medesimo dominio A x S***

- A = insieme **finito** dei simboli di ingresso e uscita
- S = insieme **finito** degli stati (di cui uno è *HALT*)

dove **A** e **S** sono insiemi ***finiti***

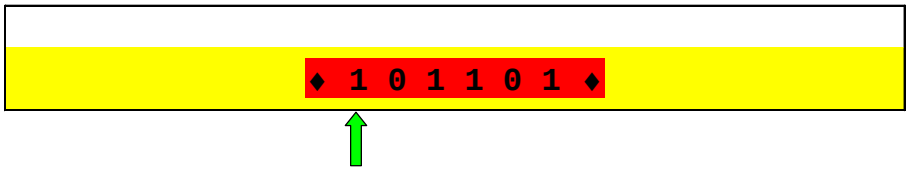
Quindi, **le tre funzioni possono essere definite *in forma estensionale tramite tabelle***, elencandone tutti i possibili valori

ESEMPIO: RICONOSCERE PALINDROMI

Nel nostro caso:

- **A = {0,1,♦,E,T}**
- **S = {HALT,s0,s1,s2,s3,s4,s5}**
  (stato iniziale: S1)

Configurazione iniziale del nastro:



ESEMPIO: RICONOSCERE PALINDROMI

mfn

	s0	s1	s2	s3	s4	s5
0	0	♦	0	0	♦	E
1	1	♦	1	1	E	♦
♦	♦	T	♦	♦	T	T

sfn

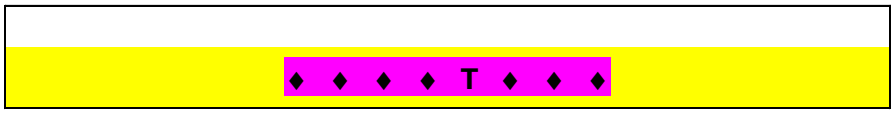
	s0	s1	s2	s3	s4	s5
0	s0	s2	s2	s3	s0	HALT
1	s0	s3	s2	s3	HALT	s0
♦	s1	HALT	s4	s5	HALT	HALT

dfn

	s0	s1	s2	s3	s4	s5
0	L	R	R	R	L	N
1	L	R	R	R	L	L
♦	R	N	L	L	N	N

ESEMPIO: RICONOSCERE PALINDROMI

Configurazione finale del nastro:



MACCHINE SPECIFICHE

- Una volta definita la parte di controllo, la Macchina di Turing è capace di risolvere un dato problema (risolubile)...
- **… ma così facendo, essa è *specifica di quel problema!***
- **Siamo circondati** da macchine specifiche:
  - calcolatrici
  - lavastoviglie,
  - videoregistratori, videogiochi
  - orologi, telecamere, ...

MACCHINE SPECIFICHE

- **Conviene fare macchine specifiche?**
  - sì, per usi particolari e mercati di massa…
  - **no, se vogliamo una macchina di uso generale con cui risolvere ogni problema (risolubile)**

MACCHINE UNIVERSALI

- **Una volta definita la parte di controllo**, cioè **l'algoritmo necessario**, la Macchina di Turing è capace di risolvere un dato problema (risolubile)
- **Finora**, l'algoritmo era ***cablato nella macchina***
- **..e se invece fosse *sul nastro*, e la macchina se lo andasse a prendere?**

MACCHINA DI TURING UNIVERSALE

- **Una Macchina di Turing** la cui parte di controllo (cioè il cui algoritmo "cablato") consiste nel **leggere *dal nastro* una *descrizione dell'algoritmo* richiesto**
- **È una macchina UNIVERSALE: *senza modifiche alla sua struttura*, può essere *istruita per risolvere un qualunque problema (risolubile)***
- **Una macchina *programmabile***

MACCHINA DI TURING UNIVERSALE

- **Ma leggere *dal nastro* una *descrizione dell'algoritmo* richiesto richiede di:**
  - **saper descrivere tale algoritmo**
  - **il che richiede un qualche *linguaggio***
  - **e una macchina che lo *interpreti***

Dunque...

- **la Universal Turing Machine (UTM) è *l'interprete di un linguaggio!***

MACCHINA DI TURING UNIVERSALE

- **Una Universal Turing Machine (UTM) *modella il concetto di elaboratore di uso generale* ("general purpose")**

- Una macchina che *va a cercare le "istruzioni" da compiere***… (fetch)**
- ... le interpreta**… (decode)**
- ... e le esegue (**execute)**

MACCHINA DI TURING UNIVERSALE E MACCHINA DI VON NEUMAN

- **Una Universal Turing Machine (UTM) è in grado di *elaborare***
  - prendendo *dati* e *algoritmo* **dal nastro**
  - e scrivendo **sul nastro** i *risultati*
- Dunque, **una UTM opera solo da/verso il nastro** (astrazione della memoria)**: non esiste il concetto di "*mondo esterno*"**
- **Ingresso/Uscita**

MACCHINA DI TURING UNIVERSALE E MACCHINA DI VON NEUMAN

Dunque,

- **la macchina di Von Neumann è modellata dalla UTM per ciò che attiene alla *computazione...***
- **ma prevede anche la dimensione dell' *interazione***

PROBLEMI RISOLUBILI E COMPUTABILITÀ

Secondo la Tesi di Church-Turing, **non esiste un formalismo "più potente" di TM**, ossia capace di risolvere una classe più ampia di problemi

Dunque...

se neanche la macchina di Turing riesce a risolvere un problema, **quel problema non è risolubile**

PROBLEMA RISOLUBILE

Un problema la cui soluzione **può essere espressa da una Macchina di Turing** o formalismo equivalente

QUALCHE DEFINIZIONE

FUNZIONE CARATTERISTICA DI UN PROBLEMA

- Dato un **problema P**,
  l'insieme X dei suoi dati di ingresso,
  l'insieme Y delle risposte corrette,
  si dice ***funzione caratteristica* del problema P** la funzione:

$$f_P: X \rightarrow Y$$

che associa a ogni dato d'ingresso la corrispondente risposta corretta

QUALCHE DEFINIZIONE

FUNZIONE CARATTERISTICA DI UN PROBLEMA

- **Perché questo artificio?**
- Perché grazie a questa funzione, **decidere *se un problema è risolubile*** **equivale a chiedersi** ***se la funzione f_P è computabile***

D'ora in poi parleremo quindi solo di ***funzioni computabili***, sapendo che ciò equivale a parlare di ***problemi risolubili***

QUALCHE DEFINIZIONE

FUNZIONE COMPUTABILE

- **Una funzione f: A→B per la quale esiste una Macchina di Turing che**
  - data sul nastro una rappresentazione di **x∈ A**

  dopo un numero finito di passi
  - produce sul nastro una rappresentazione del risultato **f(x)∈ B**

FUNZIONI NON COMPUTABILI

È facile dimostrare che **esistono funzioni *definibili ma non computabili***

e, quindi, **problemi *non risolubili***

ESEMPIO:

Problema dell' *halt* della macchina di Turing

Dire se una data macchina di Turing T, con un generico ingresso X, si ferma oppure no

FUNZIONI NON COMPUTABILI

La funzione caratteristica f_H di questo problema può essere così definita:

$$f_H(m,x) = \begin{cases} 1, \text{ se m con ingresso x si ferma} \\ 0, \text{ se m con ingresso x non si ferma} \end{cases}$$

Si può dimostrare che **questa funzione è *definita* ma *non computabile***, in quanto tentare di calcolarla conduce a un assurdo