

# Linguaggi di Programmazione

## Docente: Cataldo Musto

### Capitolo 1 – Introduzione ai linguaggi di programmazione ed alla teoria dei linguaggi formali

Si ringraziano il Prof. Giovanni Semeraro, il Prof. Marco de Gemmis e il Prof. Pasquale Lops per la concessione del materiale didattico di base

# Apprendimento di un linguaggio di programmazione

- Perché un insegnamento “Linguaggi di Programmazione” oltre a quello di “Programmazione”?

# Apprendimento di un linguaggio di programmazione

- Una delle competenze fondamentali di un *buon informatico* è quella di **apprendere un nuovo linguaggio di programmazione** con naturalezza e velocità



# Apprendimento di un linguaggio di programmazione

## ■ Come si acquisisce questa competenza?

- Questa competenza non la si acquisisce soltanto imparando *ex novo* molti linguaggi diversi
- E' meglio conoscere tanti linguaggi di programmazione oppure **conoscere i fondamenti che sono alla base dei linguaggi di programmazione?**

# Apprendimento di un linguaggio di programmazione

## ■ Lingue “naturali”

- esistono analogie, somiglianze derivanti dalla genealogia
- **Es.: italiano – rumeno**
  - cioccolata = ciocolata
  - insalata = salata
  - treno = tren
- **Es.: italiano – francese**
  - vino = vin
  - sorella - soeur

## ■ Linguaggi di programmazione

- è difficile conoscerne un gran numero in modo approfondito
- è possibile conoscerne a fondo i meccanismi che ne ispirano il *progetto* e l'*implementazione*

**Lo studio degli aspetti generali dei linguaggi di programmazione** è un passaggio chiave della formazione universitaria e professionale di un informatico

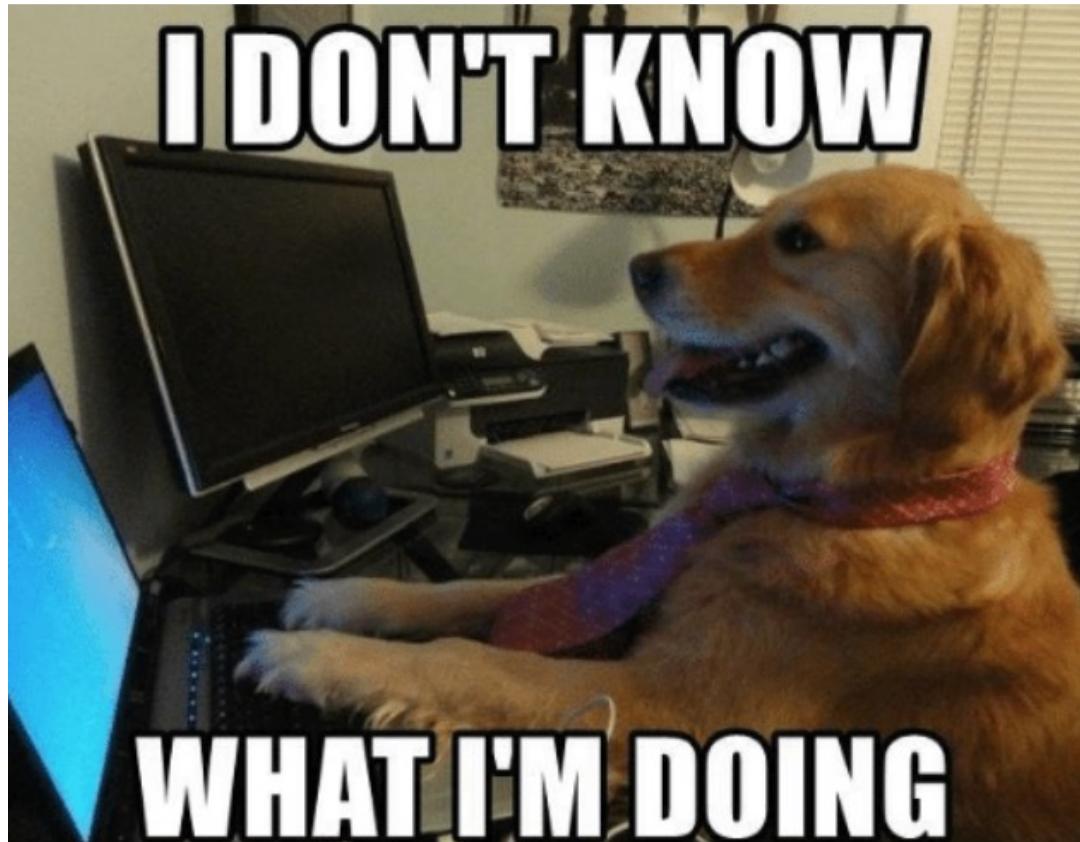


# Apprendimento di un linguaggio di programmazione: competenze fondamentali

- Come apprendere un linguaggio di programmazione?

# Apprendimento di un linguaggio di programmazione: competenze fondamentali

- Come apprendere un linguaggio di programmazione?



# Apprendimento di un linguaggio di programmazione: competenze fondamentali

- Dato un linguaggio di programmazione, **ogni informatico deve conoscere**
  - Gli aspetti prettamente linguistici (e.g., la sintassi dei linguaggi)
  - Come i costrutti possono essere implementati (ed il relativo costo)
  - Le tecniche di traduzione (e.g., compilazione)
  - Gli aspetti architetturali di un linguaggio

# Apprendimento di un linguaggio di programmazione: competenze fondamentali

- Dato un linguaggio di programmazione, **ogni informatico deve conoscere**
  - ✓ Gli aspetti prettamente linguistici (e.g., la sintassi dei linguaggi)
  - ✓ Come i costrutti possono essere implementati (ed il relativo costo)
  - ✓ Le tecniche di traduzione (e.g., compilazione)
  - Gli aspetti architetturali di un linguaggio
- **I primi tre aspetti sono trattati in questo corso!**

# Linguaggi di Programmazione

- Qual è il significato dell'espressione: “**linguaggio di programmazione**”?

# Linguaggi di Programmazione

- Cosa è un “**linguaggio di programmazione**”?
- Un linguaggio formale che specifica **un insieme di istruzioni** che possono essere usate per produrre dati in uscita: esso è utilizzabile per il controllo del comportamento di una macchina
  - [https://it.wikipedia.org/wiki/Linguaggio\\_di\\_programmazione](https://it.wikipedia.org/wiki/Linguaggio_di_programmazione)

# Linguaggi di Programmazione

- Cosa è un “**linguaggio di programmazione**”?
- **Un linguaggio formale** che specifica **un insieme di istruzioni** che possono essere usate per produrre dati in uscita: esso è utilizzabile per il controllo del comportamento di una macchina
  - [https://it.wikipedia.org/wiki/Linguaggio\\_di\\_programmazione](https://it.wikipedia.org/wiki/Linguaggio_di_programmazione)



**Linguaggi Formali** sono linguaggi progettati da persone per assolvere a compiti specifici (*Per esempio, le notazioni matematiche per denotare relazione tra simboli e numeri*)

# Linguaggi di Programmazione

- Cosa è un “**linguaggio di programmazione**”?
- Un linguaggio formale che specifica **un insieme di istruzioni** che possono essere usate per produrre dati in uscita: esso è utilizzabile per il controllo del comportamento di una macchina
  - [https://it.wikipedia.org/wiki/Linguaggio\\_di\\_programmazione](https://it.wikipedia.org/wiki/Linguaggio_di_programmazione)
- Come anticipato, i linguaggi servono **ad assolvere la nostra esigenza di comunicare con le macchine, impartendo comandi e istruzioni.**
  - Come si può comunicare con una macchina? Quale linguaggio la macchina è in grado di «capire» ?

# Linguaggi di Programmazione

- Il modo più immediato di comunicare con una macchina **è di impartire istruzioni in linguaggio macchina**
- Il linguaggio macchina è **l'unico linguaggio che è il calcolatore fisico** è grado di comprendere e decodificare
- Quali sono i limiti di questo formalismo di comunicazione?

# Linguaggi di Programmazione

- Il modo più immediato di comunicare con una macchina **è di impartire istruzioni in linguaggio macchina**
- Il linguaggio macchina è **l'unico linguaggio che è il calcolatore fisico** è grado di comprendere e decodificare
- **Quali sono i limiti** di questo formalismo di comunicazione?
  - Complessità di apprendimento
  - Complessità di esprimere programmi complessi
  - Scarsa «leggibilità» delle istruzioni

Abbiamo bisogno di formalismi più semplici ed espressivi per scrivere i nostri programmi. **Abbiamo bisogno di altri linguaggi!**

# Linguaggi di Programmazione

- Per facilitare il processo, è possibile scrivere ed eseguire algoritmi opportunamente formalizzati in un linguaggio  $\mathcal{L}$  diverso dal linguaggio macchina.
  - Un **programma** in  $\mathcal{L}$  è una sequenza di istruzioni del linguaggio  $\mathcal{L}$
  - la **sintassi** di  $\mathcal{L}$  permette di utilizzare determinati costrutti per comporre programmi in  $\mathcal{L}$
  - la **formalizzazione** consiste nella codifica degli algoritmi in un certo linguaggio  $\mathcal{L}$  definito da una specifica sintassi

# Macchina Astratta

- **Come si implementa un nuovo linguaggio di programmazione?**
- Per comprendere a pieno il significato dell'espressione, bisogna introdurre il concetto di **macchina astratta**.

# Il concetto di Astrazione

**Astrazione** - in filosofia è un metodo logico per ottenere concetti universali ricavandoli dalla conoscenza sensibile di oggetti particolari mettendo da parte ogni loro caratteristica spazio-temporale. **Astrazione** - nell'arte il termine designa la creazione di un segno astratto.

[Astrazione - Wikipedia](#)

<https://it.wikipedia.org/wiki/Astrazione>

## astratto

/a·stràt·to/

aggettivo

1. Ottenuto per astrazione, e quindi privo di corrispondenza con la realtà oggettiva e con i dati dell'esperienza sensibile: idee a.; un ragionamento a., che pecca di astrattezza.

# Il concetto di Astrazione

**Astrazione** - in filosofia è un metodo logico per ottenere concetti universali ricavandoli dalla conoscenza sensibile di oggetti particolari mettendo da parte ogni loro caratteristica spazio-temporale. **Astrazione** - nell'arte il termine designa la creazione di un segno astratto.

[Astrazione - Wikipedia](#)

<https://it.wikipedia.org/wiki/Astrazione>

## astratto

/a·stràt·to/

aggettivo

1. Ottenuto per astrazione, e quindi privo di corrispondenza con la realtà oggettiva e con i dati dell'esperienza sensibile: idee a., un ragionamento a., che pecca di astrattezza.

# Il concetto di Astrazione

**Astrazione** - in filosofia è un metodo logico per ottenere concetti universali ricavandoli dalla conoscenza sensibile di oggetti particolari mettendo da parte ogni loro caratteristica spazio-temporale. **Astrazione** - nell'arte il termine designa la creazione di un segno astratto.

[Astrazione - Wikipedia](#)

<https://it.wikipedia.org/wiki/Astrazione>

## astratto

/a·stràt·to/

aggettivo

1. Ottenuto per astrazione, e quindi privo di corrispondenza con la realtà oggettiva e con i dati dell'esperienza sensibile: idee a., un ragionamento a., che pecca di astrattezza.

L'astrazione è quindi un **meccanismo logico** con cui **acquisiamo conoscenza generale** a partire dall'osservazione del particolare (la realtà).

# Il concetto di Astrazione

**Astrazione** - in filosofia è un metodo logico per ottenere concetti universali ricavandoli dalla conoscenza sensibile di oggetti particolari mettendo da parte ogni loro caratteristica spazio-temporale. **Astrazione** - nell'arte il termine designa la creazione di un segno astratto.

[Astrazione - Wikipedia](#)

<https://it.wikipedia.org/wiki/Astrazione>

## astratto

/a·stràt·to/

aggettivo

1. Ottenuto per astrazione, e quindi privo di corrispondenza con la realtà oggettiva e con i dati dell'esperienza sensibile: idee a., un ragionamento a., che pecca di astrattezza.

Avete già visto dei modelli «astratti» di computazione? Delle macchine «generiche» con delle caratteristiche comuni a tutti gli elaboratori? ;)

# Macchine Astratte e Implementazione dei linguaggi di programmazione

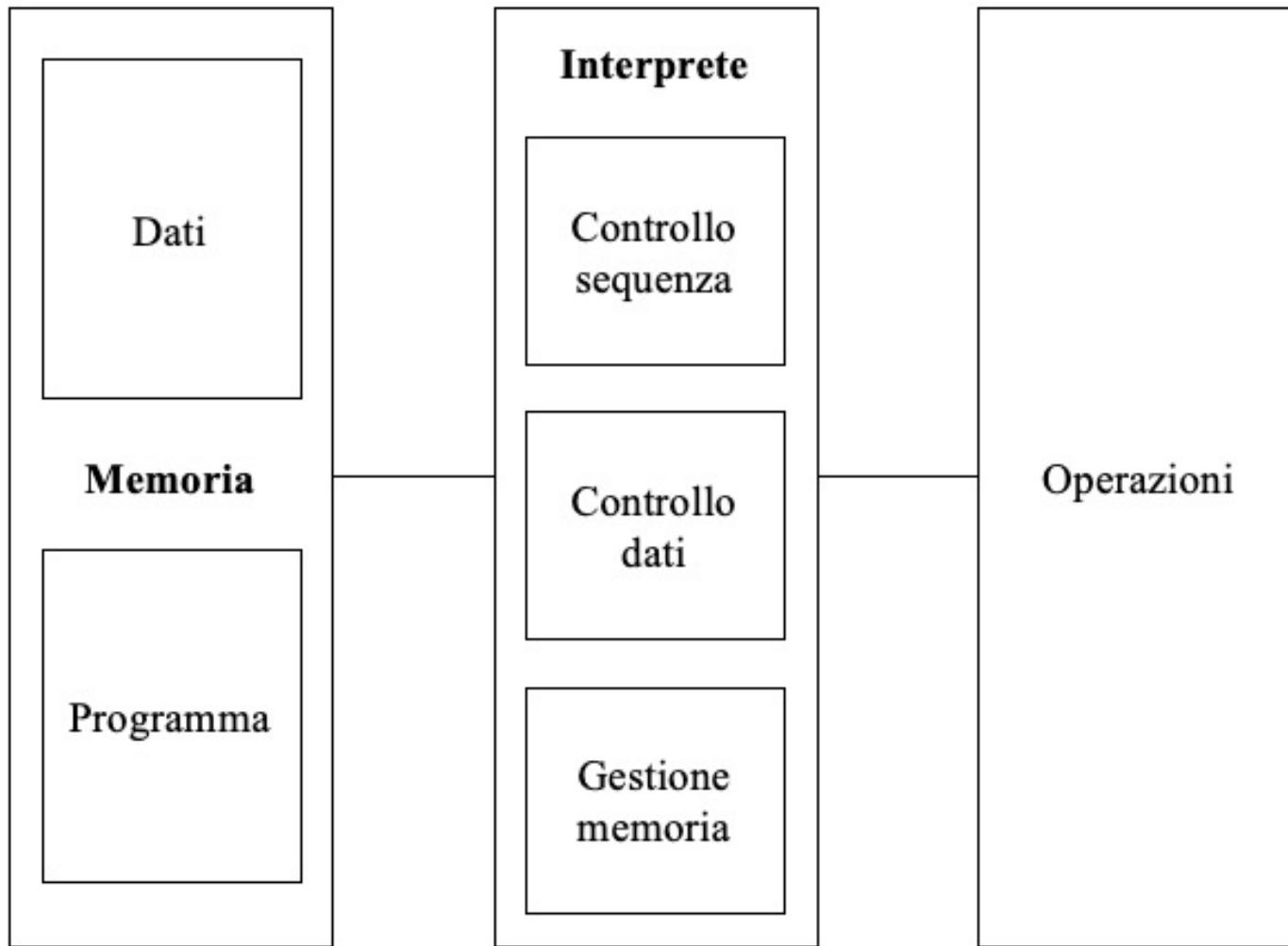
- La macchina astratta è la formalizzazione di un generico esecutore di algoritmi
- Concretamente, è un «intermediario» che si occupa di comprendere ed eseguire i programmi scritti in  $\mathcal{L}$ 
  - Diamo una definizione più formale e precisa.

# Definizioni di macchina astratta e linguaggio macchina

## ■ Macchina astratta per $\mathcal{L}$

- Si denota con  $\mathcal{M}_{\mathcal{L}}$
- *Un insieme di algoritmi e strutture dati che permettono di memorizzare ed eseguire programmi scritti in  $\mathcal{L}$*
- E' composta da:
  - una **memoria** per immagazzinare dati e programmi
  - un **interprete** che esegue le istruzioni contenute nei programmi

# Macchina Astratta

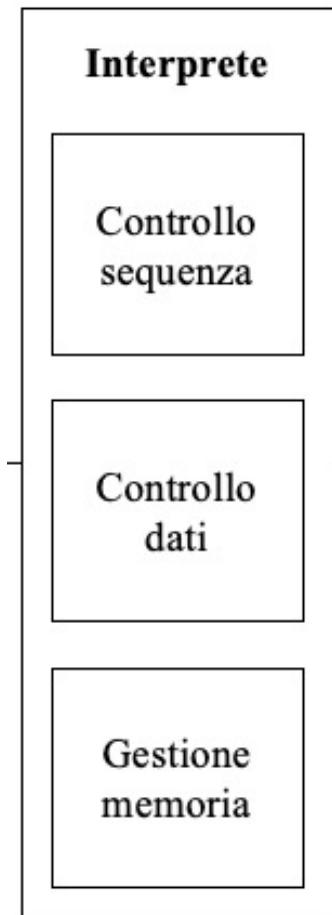


# Macchina Astratta

- La natura «astratta» della macchina evidenzia delle caratteristiche comuni a più tipologie di calcolatori e più linguaggi di programmazione
- Al netto della variabilità dei diversi linguaggi di programmazione, **esistono delle operazioni «standard» effettuate dagli interpreti**
- **Possiamo delineare delle componenti fisse, sempre presenti in una generica macchina astratta.**

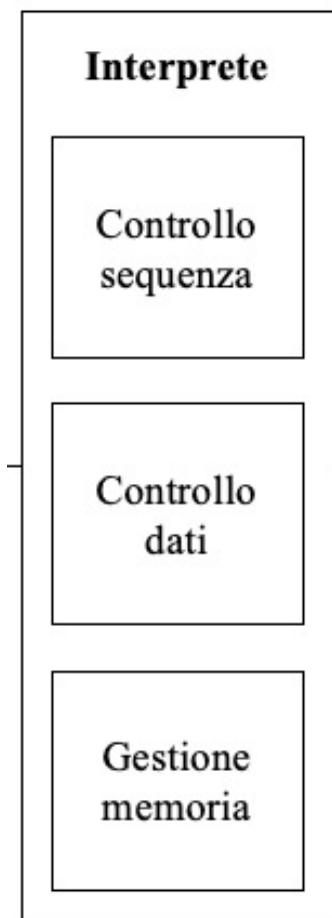
# Le operazioni dell'Interprete

- Al netto della variabilità dei diversi linguaggi di programmazione, esistono delle operazioni «standard» effettuate dagli interpreti
- Operazioni per **l'elaborazione dei dati primitivi**
  - operazioni aritmetiche su interi e reali
- Operazioni e strutture dati per il **controllo della sequenza di esecuzione**
  - servono per gestire il **flusso di controllo** delle istruzioni
  - strutture dati per memorizzare **l'indirizzo della prossima istruzione**
  - operazioni per manipolare le strutture dati
    - es.: calcolo dell'indirizzo della prossima istruzione

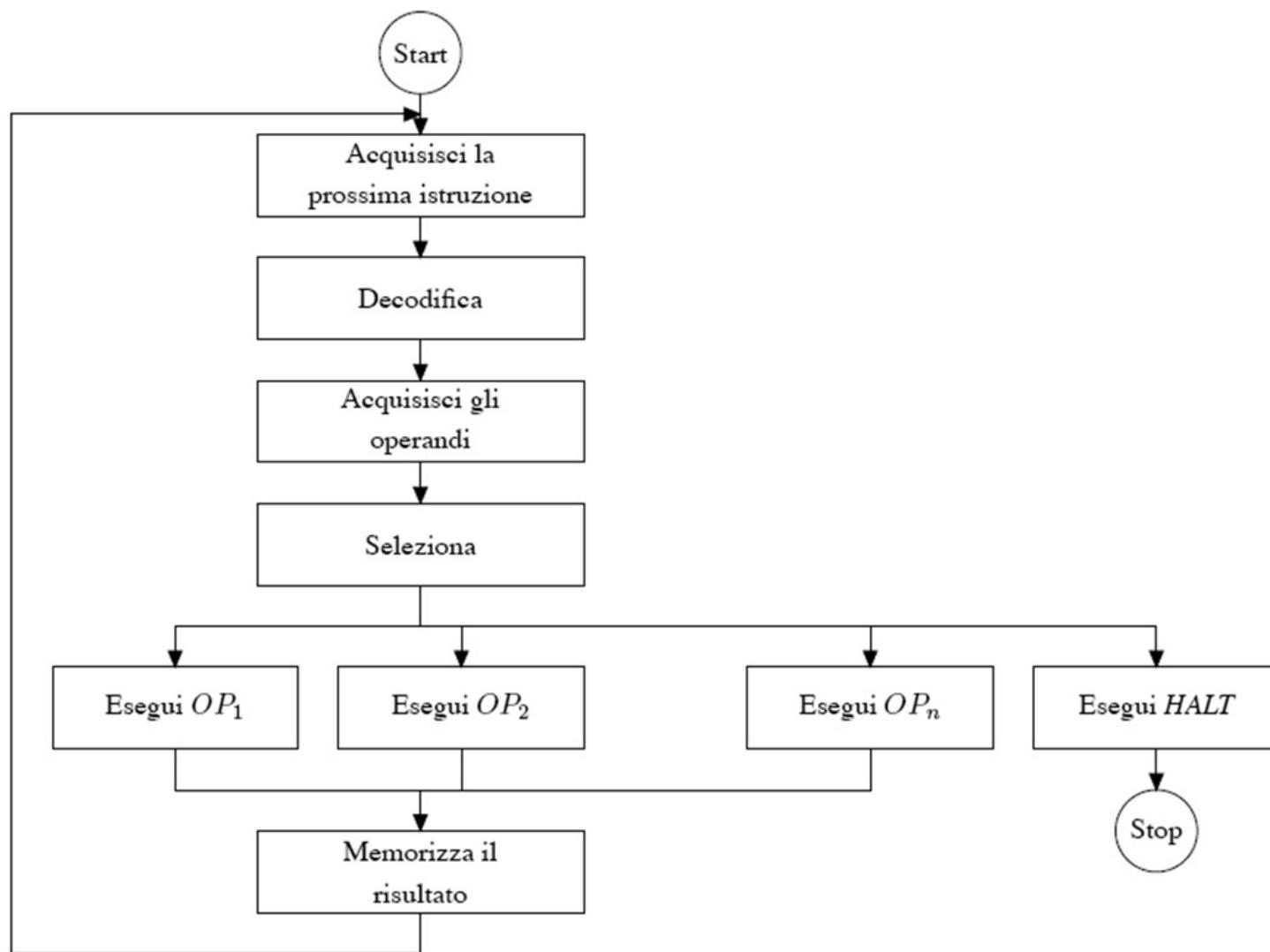


# Le operazioni dell'Interprete

- Operazioni e strutture dati per il **controllo del trasferimento dei dati**
  - gestiscono il trasferimento dei dati dalla memoria all'interprete e viceversa
    - es.: recupero degli operandi
    - possono far uso di strutture dati ausiliarie (es. pila)
- Operazioni e strutture dati per la **gestione della memoria**
  - relative all'allocazione di memoria per dati e programmi



# Ciclo di esecuzione del generico interprete



# Definizioni di macchina astratta e linguaggio macchina

## ■ Linguaggio Macchina

- Data una macchina astratta  $\mathcal{M}_{\mathcal{L}}$ , il linguaggio  $\mathcal{L}$  “compreso” dall’interprete di  $\mathcal{M}_{\mathcal{L}}$  è detto **linguaggio macchina** di  $\mathcal{M}_{\mathcal{L}}$

# Definizioni di macchina astratta e linguaggio macchina

## ■ Linguaggio Macchina

- Data una macchina astratta  $\mathcal{M}_{\mathcal{L}}$ , il linguaggio  $\mathcal{L}$  “compreso” dall’interprete di  $\mathcal{M}_{\mathcal{L}}$  è detto **linguaggio macchina** di  $\mathcal{M}_{\mathcal{L}}$
- Ad esempio, se  $\mathcal{M}_{\mathcal{L}}$ , è la macchina «fisica» il linguaggio  $\mathcal{L}$  “compreso” dal  $\mathcal{M}_{\mathcal{L}}$  è il **linguaggio binario**

# Definizioni di macchina astratta e linguaggio macchina

- Dunque posso esprimere le mie istruzioni in un linguaggio  $\mathcal{L}$  diverso dal linguaggio macchina se esiste una macchina astratta  $\mathcal{M}_{\mathcal{L}}$  in grado di comprendere ed eseguire programmi scritti in quel linguaggio
- **Implementare un linguaggio**
  - Significa implementare una macchina astratta  $\mathcal{M}_{\mathcal{L}}$  che abbia  $\mathcal{L}$  come linguaggio “compreso” dalla macchina

# Realizzazione di una macchina astratta

- Come si implementa una macchina astratta  $\mathcal{M}_{\mathcal{L}}$  ?

# Realizzazione di una macchina astratta

- Come si implementa una macchina astratta  $\mathcal{M}_{\mathcal{L}}$  ?
- Per poter effettivamente eseguire programmi scritti in  $\mathcal{L}$ ,  $\mathcal{M}_{\mathcal{L}}$  può scegliere due strade
  - Interfacciarsi direttamente con la macchina fisica
    - Perché la macchina fisica è **l'unica che può materialmente eseguire le istruzioni del programma scritto in  $\mathcal{L}$**
  - Pensare a una ulteriore realizzazione che si «avvicini» alla macchina fisica
    - Una «traduzione»

# Realizzazione di una macchina astratta

- Generalmente, esistono tre metodi di interfacciamento:
  - realizzazione “fisica” in hardware
  - emulazione mediante firmware
  - simulazione mediante software

# Realizzazione di una macchina astratta

- Generalmente, esistono tre metodi di interfacciamento:
  - **realizzazione “fisica” in hardware** ←
  - emulazione mediante firmware
  - simulazione mediante software
- La realizzazione fisica in hardware consiste nel tradurre ogni istruzione di  $\mathcal{L}$  nel linguaggio macchina
  - Concretamente, consiste nell’interagire direttamente con bus, unità, reti, etc.
  - **rapporto costi-benefici troppo sbilanciato**, soprattutto per linguaggi complessi

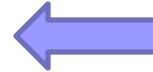
# Realizzazione di una macchina astratta

- Generalmente, esistono tre metodi di interfacciamento:
  - realizzazione “fisica” in hardware
  - **emulazione mediante firmware** 
  - simulazione mediante software
- L'emulazione mediante firmware consiste nella **scrittura di microprogrammi**. Si utilizza un linguaggio intermedio, con un set di istruzioni molto semplice, che va a comunicare direttamente con la macchina
  - Svantaggio: la programmazione resta ancora complessa, e richiede opportuni dispositivi di memorizzazione

# Realizzazione di una macchina astratta

- Generalmente, esistono tre metodi di interfacciamento:
  - realizzazione “fisica” in hardware
  - emulazione mediante firmware 
  - **simulazione mediante software**
- Questa metodologia suppone che esista un linguaggio  $\mathcal{L}'$  con la sua macchina astratta  $\mathcal{M}_{\mathcal{L}'}$ ,
- Si «simula» il nuovo linguaggio  $\mathcal{L}'$  **scrivendo il software in un altro linguaggio  $\mathcal{L}'$** , comprensibile alla macchina
  - $\mathcal{M}_{\mathcal{L}'}$  è realizzata mediante programmi in  $\mathcal{L}'$  che simulano le funzionalità di  $\mathcal{M}_{\mathcal{L}}$
  - Algoritmi e strutture dati di  $\mathcal{M}_{\mathcal{L}}$  realizzati in un altro linguaggio  $\mathcal{L}'$  già implementato
  - $\mathcal{M}'_{\mathcal{L}'}$  è detta *macchina ospite*

# Realizzazione di una macchina astratta

- Generalmente, esistono tre metodi di interfacciamento:
  - realizzazione “fisica” in hardware
  - emulazione mediante firmware
  - **simulazione mediante software** 
- **Spoiler:** questa è la metodologia più diffusa, utilizzata per i moderni linguaggi di programmazione

# Realizzazione mediante software

- L'implementazione di  $\mathcal{L}$  sulla macchina ospite avviene mediante **una qualche “traduzione” di  $\mathcal{L}$  in  $\mathcal{L}'$**



A seconda di come avvenga la traduzione di parla di:

- implementazione interpretativa (traduzione implicita)**
- implementazione compilativa (traduzione esplicita)**

# Recap

- Per poter impartire comandi alla macchina in un linguaggio  $\mathcal{L}$  diverso dal linguaggio macchina ho bisogno di una macchina astratta  $\mathcal{M}_{\mathcal{L}}$ 
  - $\mathcal{M}_{\mathcal{L}}$  è la formalizzazione di un generico esecutore di algoritmi, con delle macro-componenti standard
- Implementare un nuovo linguaggio  $\mathcal{L}$  significa implementare la sua macchina astratta  $\mathcal{M}_{\mathcal{L}}$ 
  - La macchina astratta si può implementare mediante realizzazione fisica, mediante emulazione firmware e simulazione software (la più diffusa)
  - Per implementare mediante simulazione software, abbiamo bisogno di una ulteriore macchina astratta (detta macchina ospite) e che ogni istruzione di  $\mathcal{L}$  sia tradotta nel linguaggio della macchina ospite

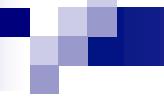
# Recap: Funzioni Parziali

- Per poter introdurre il processo di traduzione, è necessario prima fornire la definizione di **funzione parziale**
- Una funzione *parziale*

$$f: A \rightarrow B$$

è una corrispondenza tra elementi dell'insieme A e quelli dell'insieme B

- Se la funzione è definite per ogni elemento di A è detta **totale**.
- Se NON DEFINITA per qualche elemento di A, è detta **parziale**
- Dato  $a \in A$ , **se** esiste un corrispondente in B, esso è denotato con  $f(a)$



# I programmi definiscono funzioni parziali

# I programmi definiscono funzioni parziali

```
read(x);  
if x==1 then print(x) else  
    while (x<>1) do skip;
```

$$f(x) = \begin{cases} \text{?????} & \end{cases}$$

# I programmi definiscono funzioni parziali

```
read(x);  
if x==1 then print(x) else  
    while (x<>1) do skip;
```

$$f(x) = \begin{cases} 1 & \text{se } x=1 \\ \text{indefinita} & \text{altrimenti} \end{cases}$$

In generale un programma scritto in L si può vedere come una funzione parziale

$$\mathcal{P}^L : \mathcal{D} \rightarrow \mathcal{D} \quad \text{t.c. } \mathcal{P}^L(\text{Input}) = \text{Output}$$

dati

# Definizione di interprete

Possiamo dare la seguente definizione di interprete di  $\mathcal{L}$  in  $\mathcal{Lo}$ :

$$I_{\mathcal{L}}^{\mathcal{Lo}} : (\text{Prog}^{\mathcal{L}} \times \mathcal{D}) \rightarrow \mathcal{D} \quad \text{tale che}$$

$$I_{\mathcal{L}}^{\mathcal{Lo}}(\mathcal{P}^{\mathcal{L}}, \text{Input}) = \mathcal{P}^{\mathcal{L}}(\text{Input})$$

$\text{Prog}^{\mathcal{L}}$  = insieme dei possibili programmi del linguaggio L

$\mathcal{P}^{\mathcal{L}}$  = un generico programma

L'interprete per L è dunque un programma che prende in input un programma scritto in  $\mathcal{L}$  e i suoi dati, e produce in output un risultato (altri dati). **Tale risultato non è altro che il risultato dell'esecuzione del programma originario.**

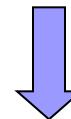
# Definizione di interprete

Possiamo dare la seguente definizione di interprete di  $\mathcal{L}$  in  $\mathcal{Lo}$ :

$$I_{\mathcal{L}}^{\mathcal{Lo}} : (\text{Prog}^{\mathcal{L}} \times \mathcal{D}) \rightarrow \mathcal{D} \quad \text{tale che}$$

$$I_{\mathcal{L}}^{\mathcal{Lo}}(\mathcal{P}^{\mathcal{L}}, \text{Input}) = \mathcal{P}^{\mathcal{L}}(\text{Input})$$

**Non vi è traduzione esplicita** dei programmi scritti in  $\mathcal{L}$ , ma solo un procedimento di decodifica



L'interprete, per eseguire un'istruzione  $i$  di  $\mathcal{L}$ , le fa corrispondere un insieme di istruzioni di  $\mathcal{Lo}$ . Tale decodifica non è una traduzione esplicita poiché il codice corrispondente a  $i$  è eseguito direttamente e non prodotto in uscita.

**Traduzione ed esecuzione non sono separate!**

# Definizione di compilatore

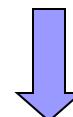
Un compilatore da  $\mathcal{L}$  a  $\mathcal{L}o$  è un programma che realizza la funzione:

$$C_{\mathcal{L}, \mathcal{L}o} : \text{Prog}^{\mathcal{L}} \rightarrow \text{Prog}^{\mathcal{L}o}$$

$$C_{\mathcal{L}, \mathcal{L}o}(\mathcal{P}^{\mathcal{L}}) = \mathcal{P}C^{\mathcal{L}o} \quad \mathcal{P}^{\mathcal{L}}(\text{Input}) = \mathcal{P}C^{\mathcal{L}o}(\text{Input})$$

**Traduzione esplicita** dei programmi scritti in  $\mathcal{L}$  in programmi scritti in  $\mathcal{L}o$

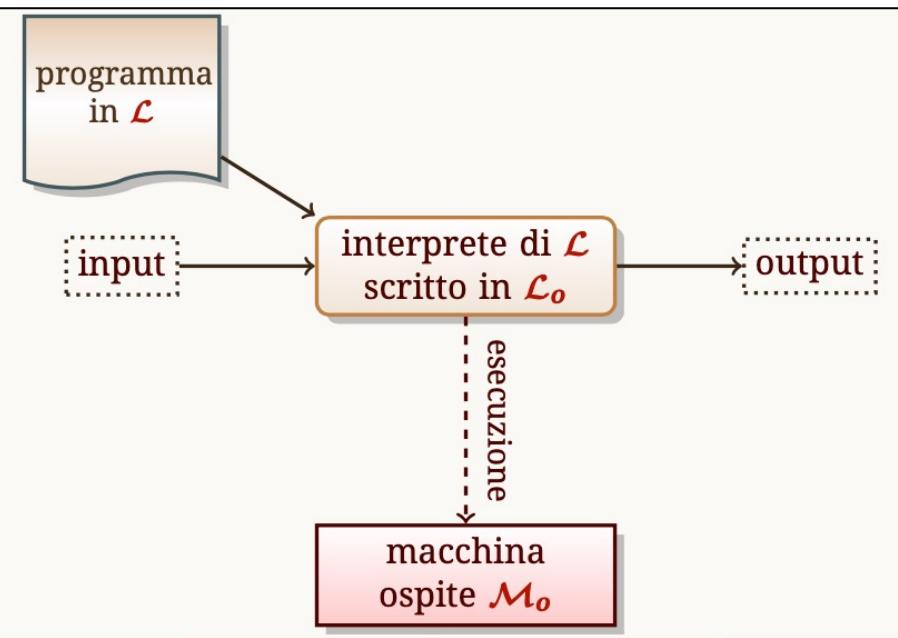
$\mathcal{L}$  linguaggio sorgente



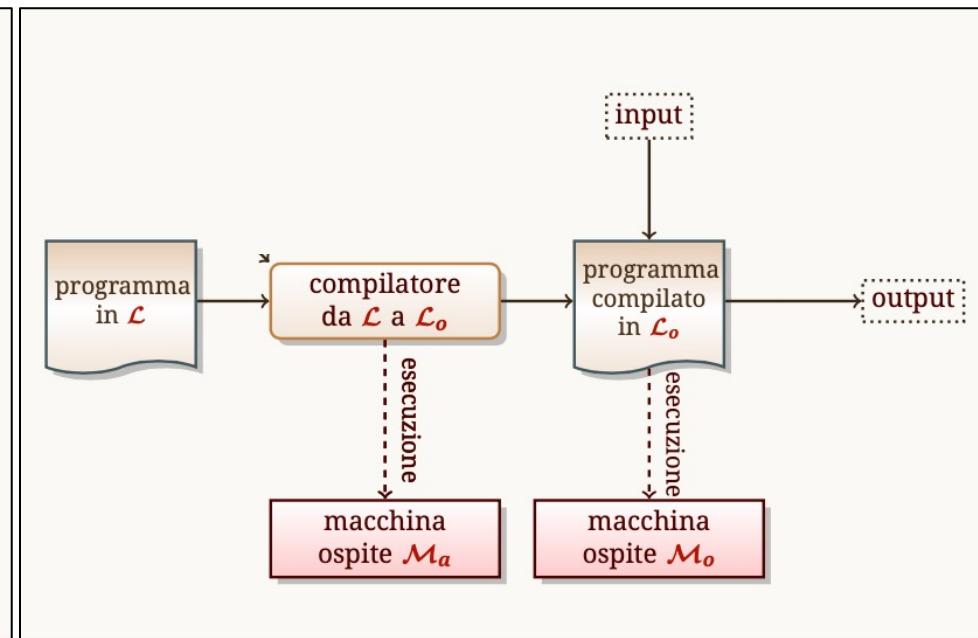
$\mathcal{L}o$  linguaggio oggetto

Per eseguire  $\mathcal{P}^{\mathcal{L}}$  su  $\text{Input}$ , bisogna eseguire  $C_{\mathcal{L}, \mathcal{L}o}$  con  $\mathcal{P}^{\mathcal{L}}$  come input. Si avrà come risultato un programma compilato  $\mathcal{P}C^{\mathcal{L}o}$  scritto in  $\mathcal{L}o$ , che sarà eseguito su  $\mathcal{M}_{\mathcal{L}o}$  con il dato in ingresso  $\text{Input}$

# Interpretazione vs. Compilazione



Traduzione  
**interpretativa**



Traduzione  
**compilativa**

# Interpretazione vs. Compilazione

## ■ Interpretazione

👉 scarsa efficienza

- decodifica costrutti a tempo di esecuzione
- decodifica di uno stesso comando ripetuto

👉 maggiore flessibilità

- debugging più semplice

## ■ Compilazione

👉 maggiore efficienza

- decodifica di un'istruzione fatta una sola volta indipendentemente da quante volte è eseguita

👉 minore flessibilità

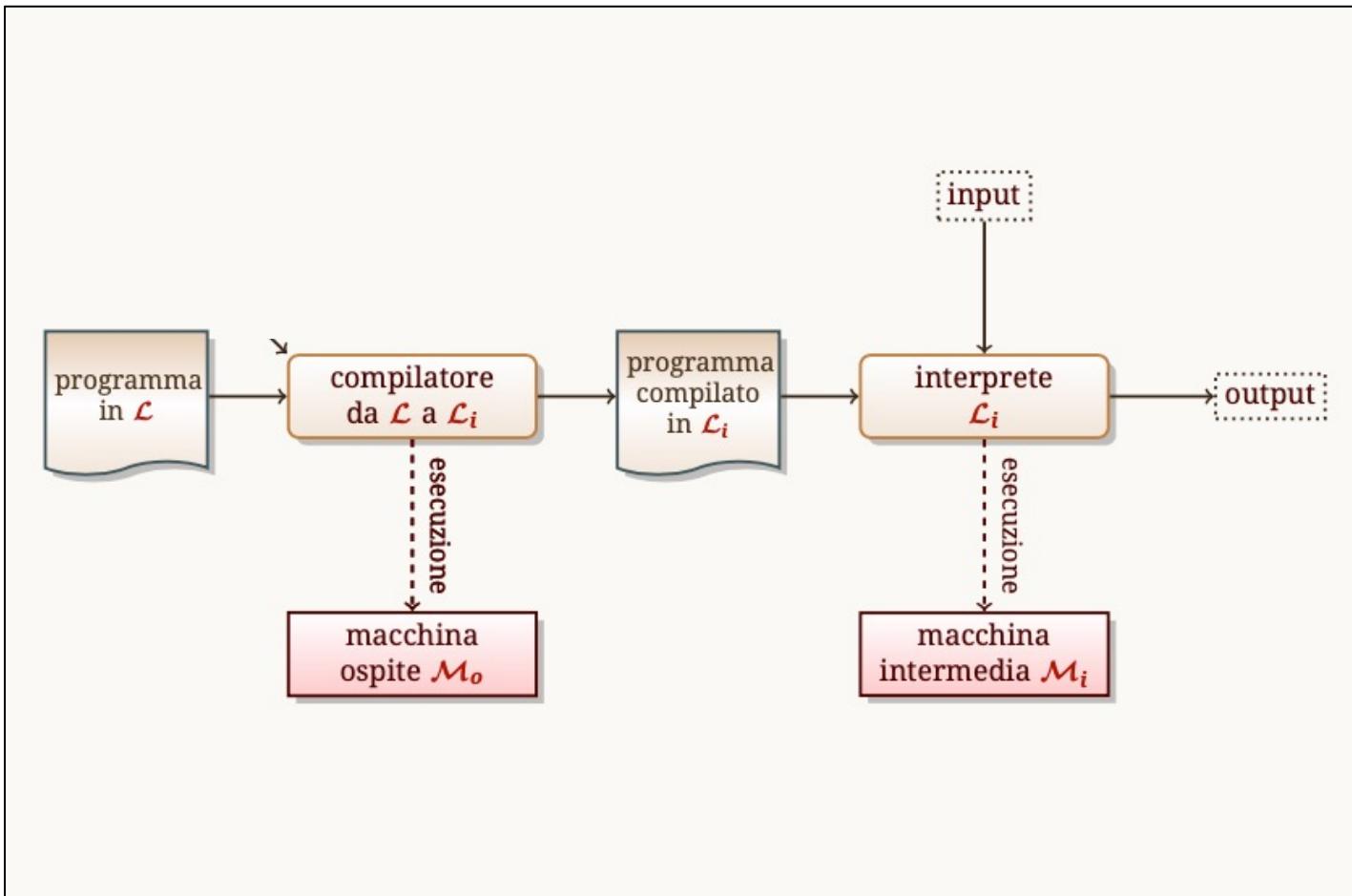
- perdita di informazioni rispetto al programma sorgente (difficile tracciare errori e dunque maggiori difficoltà nel debugging)

**In realtà i due approcci sono spesso combinati**

# Interpretazione vs. Compilazione

- **Interpretazione e compilazione sono spesso combinate nei moderni linguaggi di programmazione**
- Implementazioni puramente interpretative o puramente compilative **sono rarissime**
  - Traduzioni Interpretative: PROLOG
  - Traduzioni Compilative: C, C++, Pascal
- Molto spesso si effettua una **compilazione in un linguaggio intermedio** (es. Java Virtual Machine, ma anche Python), che poi viene interpretato in modo differente sulle diverse macchine
  - **Ogni linguaggio ha le sue scelte progettuali ed implementative differenti!**

# Interpretazione vs. Compilazione



Approccio  
misto

# Gerarchie di Macchine Astratte

- **Di quante macchine astratte abbiamo bisogno?**

# Gerarchie di Macchine Astratte

- **Di quante macchine astratte abbiamo bisogno?**
- Possiamo pensare ad una gerarchia di macchine astratte  $\mathcal{M}_{\mathcal{L}_0}, \mathcal{M}_{\mathcal{L}_1}, \dots \mathcal{M}_{\mathcal{L}_n}$
- $\mathcal{M}_{\mathcal{L}_i}$  è implementata sfruttando il linguaggio della macchina sottostante  $\mathcal{M}_{\mathcal{L}_{i-1}}$
- $\mathcal{M}_{\mathcal{L}_i}$  fornisce a sua volta il proprio linguaggio alla macchina sovrastante  $\mathcal{M}_{\mathcal{L}_{i+1}}$
- Indipendenza tra livelli
  - modifiche *interne* alle funzionalità di un livello non hanno influenza sugli altri livelli

# Una gerarchia di macchine astratte

MACCHINA LINGUAGGIO ALTO LIVELLO

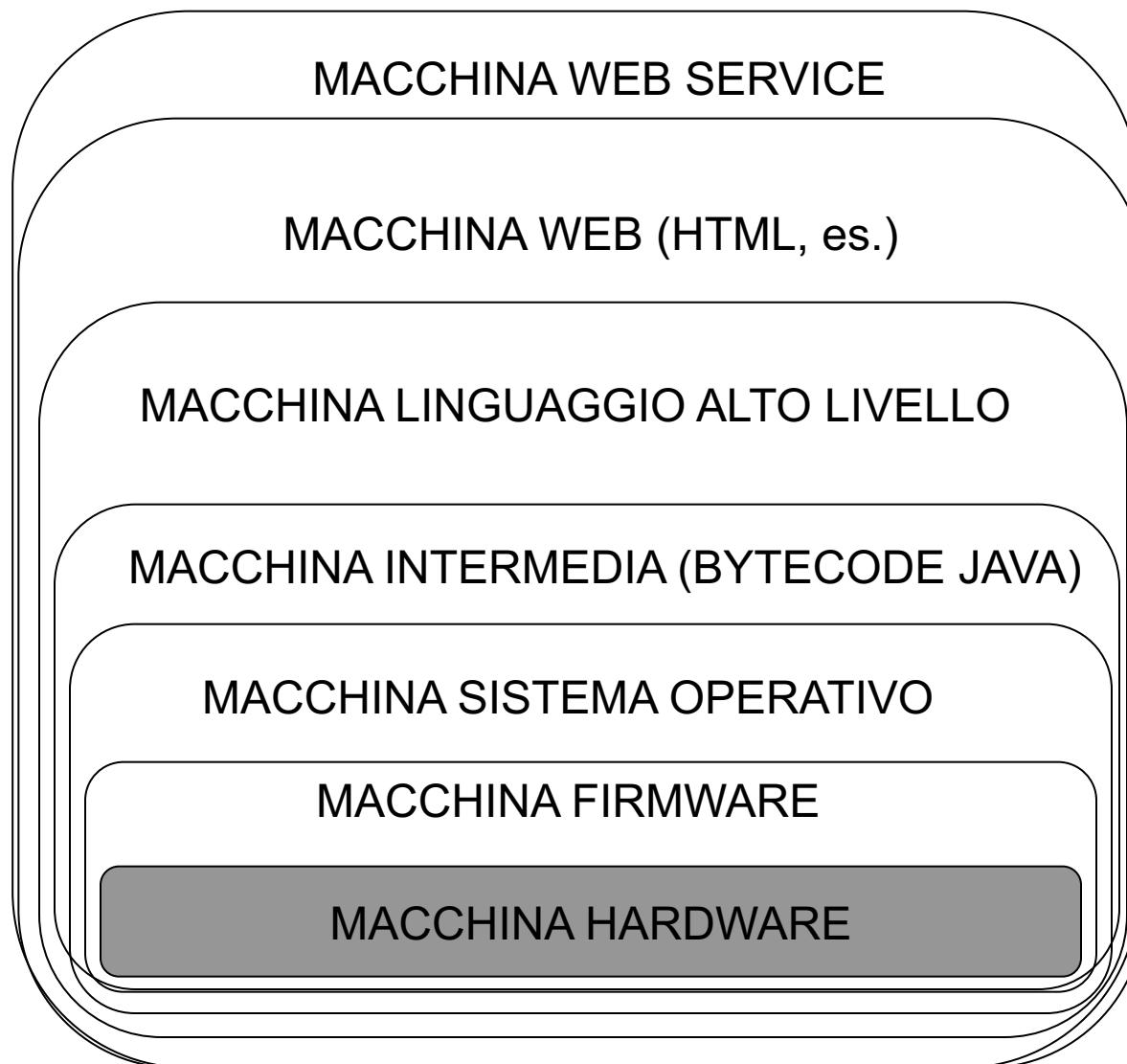
MACCHINA INTERMEDIA (BYTECODE JAVA)

MACCHINA SISTEMA OPERATIVO

MACCHINA FIRMWARE

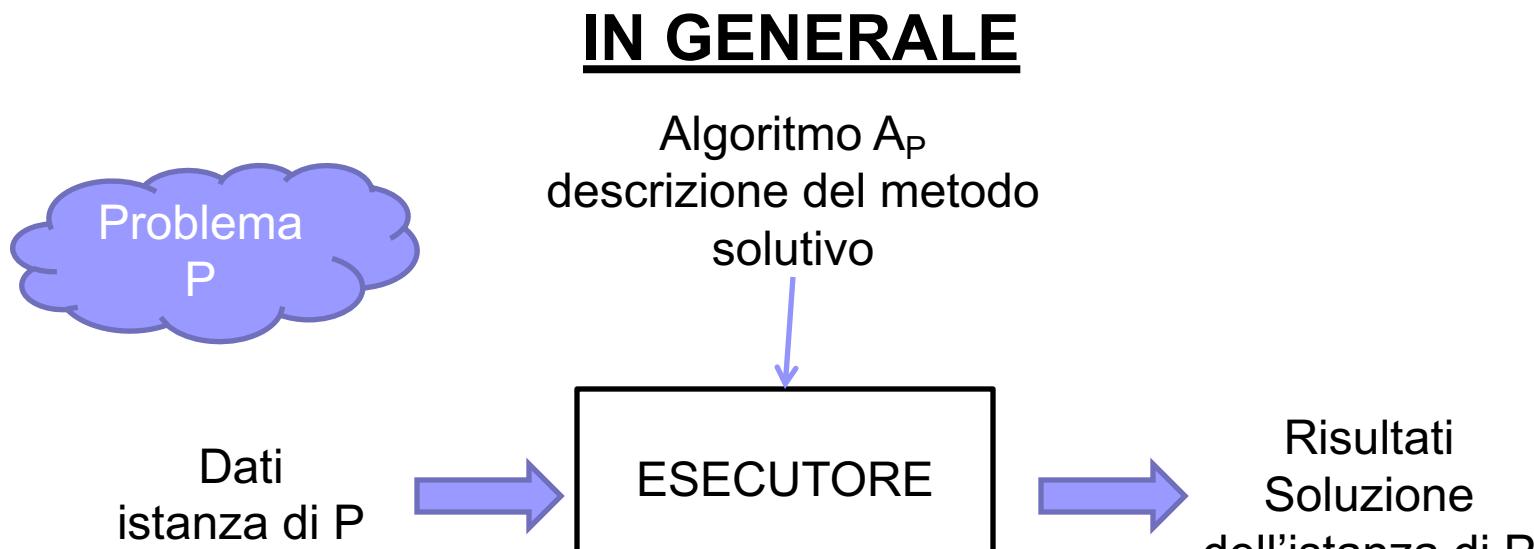
MACCHINA HARDWARE

# Una gerarchia (più moderna) di macchine astratte



# Linguaggi di Programmazione, Problemi, Interpreti

- **Linguaggio di programmazione** = formalismo per portare al livello di macchina fisica l'algoritmo solutivo  $A_p$  per un certo problema  $P$



Deve essere in grado di **interpretare** la descrizione del metodo solutivo

# Programmi e funzioni calcolabili

- Esistono dei problemi per i quali **non esista** un programma che li risolva?
- La risposta **dipende dal linguaggio** di programmazione?

# Funzioni (Parziali) Calcolabili

- Una **funzione parziale**  $f: A \rightarrow B$  è **calcolabile** nel linguaggio  $\mathcal{L}$  se esiste un programma  $P$  scritto in  $\mathcal{L}$  tale che:
  - Se  $f(a) = b$  allora  $P(a)$  termina e produce come output  $b$
  - Se  $f(a)$  non è definita allora  $P$  con input  $a$  va in ciclo
- Esempio: la funzione **somma** è calcolabile
  - **Possiamo scrivere un programma che, restituisce come output la somma dei due operandi.**
- Questo vale per tantissime tipologie di funzioni, dal calcolo del codice fiscale **al riconoscimento facciale sugli smartphone**

# Funzioni (Parziali) Calcolabili

- Una **funzione parziale**  $f: A \rightarrow B$  è **calcolabile** nel linguaggio  $\mathcal{L}$  se esiste un programma  $P$  scritto in  $\mathcal{L}$  tale che:
  - Se  $f(a) = b$  allora  $P(a)$  termina e produce come output  $b$
  - Se  $f(a)$  non è definita allora  $P$  con input  $a$  va in ciclo
- Esistono dei problemi per i quali non esiste un programma che li risolva?
- **Esistono funzioni non calcolabili?**

# Il problema della fermata

- Vogliamo stabilire se un programma termina su un dato input
  - Programma di *debugging* H
- H riceve in ingresso un qualsiasi programma P scritto nel linguaggio  $\mathcal{L}$  ed un generico input x per tale programma:

# Il problema della fermata

- Vogliamo stabilire se un programma termina su un dato input
  - Programma di *debugging* H
- H riceve in ingresso un qualsiasi programma P scritto nel linguaggio  $\mathcal{L}$  ed un generico input x per tale programma:

```
Boolean H(P,x)
    boolean term;
    if (P(x) termina) then term=true;
        else term=false;
    return term;
```

H( $P, x$ ) ritorna

  
true se  $P(x)$  termina  
false se  $P(x)$  va in loop

# Il problema della fermata

- Vogliamo stabilire se un programma termina su un dato input
  - Programma di *debugging* H
- H riceve in ingresso un qualsiasi programma P scritto nel linguaggio  $\mathcal{L}$  ed un generico input x per tale programma:

```
Boolean H(P,x)
    boolean term;
    if (P(x) termina) then term=true;
        else term=false;
    return term;
```

H( $P, x$ ) ritorna

{ **true** se  $P(x)$  termina  
**false** se  $P(x)$  va in loop



**esiste?**

# Il problema della fermata

- **Supponiamo che  $H$  esista.**
- Possiamo costruire un altro programma  $K$  scritto in  $\mathcal{L}$  che prenda in input un programma  $P$  (scritto sempre in  $\mathcal{L}$ )
  - Il programma  $K$  sfrutta  $H$  per decidere sulla terminazione di  $P$

$K(P)$  {

se $H(P,P) = \text{false}$	$\rightarrow$	Stampa "LOOP"
se $H(P,P) = \text{true}$	$\rightarrow$	va in loop

```
if (H(P,P)=false) then print("LOOP");  
else while (true) do print ("TERMINA");
```

- In pratica la terminazione di  $K$  è opposta rispetto a quella del suo input

# Il problema della fermata

- **Supponiamo che H esista.**
- Possiamo costruire un altro programma **K** scritto in  $\mathcal{L}$  che prenda in input un programma P (scritto sempre in  $\mathcal{L}$ )
  - Il programma K sfrutta H per decidere sulla terminazione di P

$K(P)$  {

- P(P) non termina  $\rightarrow$  Stampa "LOOP"
- P(P) termina  $\rightarrow$  va in loop

```
if (H(P,P)=false) then print("LOOP");  
else while (true) do print ("TERMINA");
```

- In pratica la terminazione di K è opposta rispetto a quella del suo input

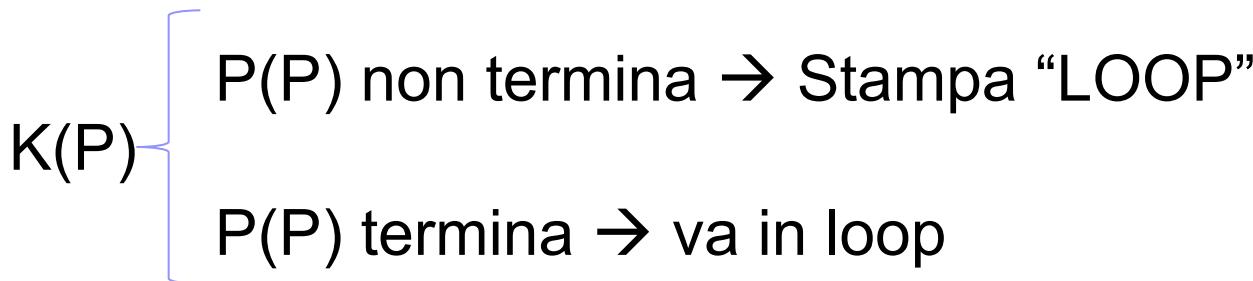
# Il problema della fermata

$K(P)$  {  
    P( $P$ ) non termina → Stampa “LOOP”  
    P( $P$ ) termina → va in loop

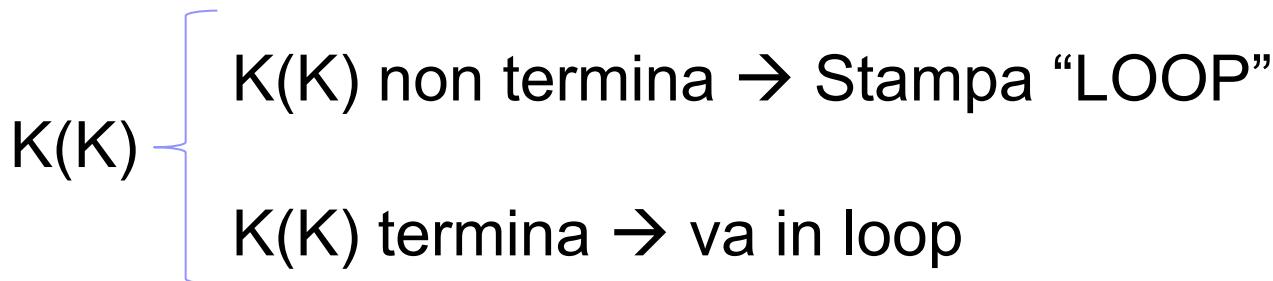
- Cosa accade se diamo in input a  $K$  il suo stesso testo?

$K(K) = ?$

# Il problema della fermata



- Cosa accade se diamo in input a  $K$  il suo stesso testo?



- K( $K$ ) termina con una stampa quando K( $K$ ) non termina
- K( $K$ ) non termina quando K( $K$ ) termina
- Assurdo conseguente dall'aver supposto l'esistenza del programma H
- H non può esistere! Assurdo!**

# Indecidibilità

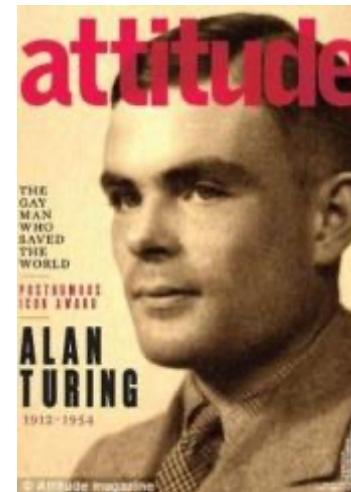
- Il problema della terminazione è *indecidibile*
  - Non possiamo costruire (e formalizzare) un algoritmo che lo risolva
- Esistono dei problemi per i quali non esiste un programma che li risolva? Esistono funzioni **non calcolabili**?
  - Sì, e il problema della terminazione è una di queste

# Indecidibilità

- Il problema della terminazione è *indecidibile*
  - Non possiamo costruire (e formalizzare) un algoritmo che lo risolva
- Esistono dei problemi per i quali non esiste un programma che li risolva? Esistono funzioni **non calcolabili**?
  - Sì, e il problema della terminazione è una di queste
  - Questo risultato dipende dal linguaggio  $\mathcal{L}$ ?
    - ovvero dipende dal formalismo usato per descrivere l'algoritmo?

# Macchine di Turing (MdT)

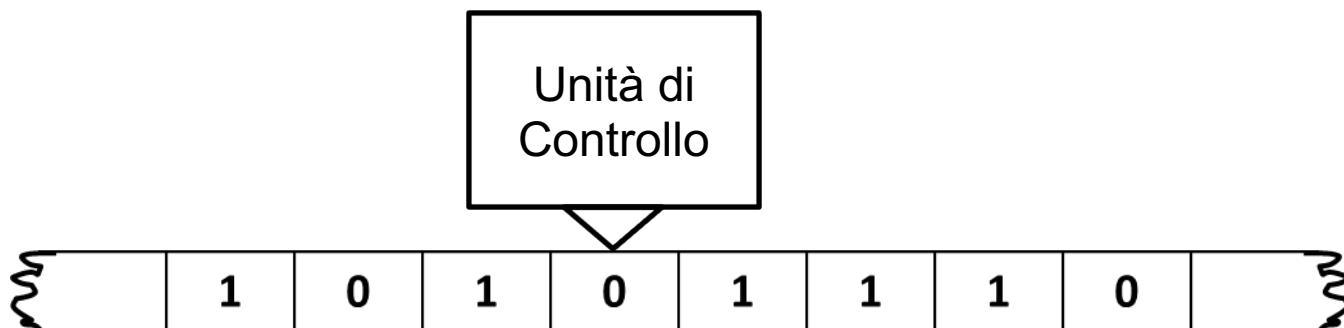
- Modello matematico di computazione introdotto negli anni '30
- Primo formalismo (=linguaggio) con il quale è stata **dimostrata l'indecidibilità del problema della fermata**



**Alan Mathison Turing**  
(Londra, 23 giugno 1912 – Wilmslow, 7 giugno 1954)

# Macchine di Turing (MdT)

- **Nastro potenzialmente infinito diviso in celle (memoria)**
  - *ogni cella contiene un simbolo preso da un alfabeto finito*
- **Testina di lettura/scrittura**
  - può leggere/scrivere in una cella per volta
  - può spostarsi a destra o a sinistra di una cella per volta
  - gestita da un controllo espresso da un numero finito di stati
- **Unità di controllo**
  - decodifica ed esegue comandi rivolti alla testina



# Macchine di Turing (MdT)

- L'unità di controllo esegue un programma  $P$  sui dati memorizzati sul nastro
- Le istruzioni di  $P$  sono del tipo:  
< simbolo\_letto,  
    stato\_corrente,  
    simbolo\_da\_scrivere,  
    sinistra/destra,  
    nuovo\_stato >
- Sostanzialmente le istruzioni **permettono di leggere e scrivere sul nastro e di spostare la testina.**

# Un parallelo tra MdT e moderni processori

- MdT
  - ① Legge / scrive su nastro
  - ② Transita in un nuovo stato
  - ③ Si sosta sul nastro di cella in cella
  - ④ Esegue un programma **specifico CABLATO** nella macchina → è specifica per un certo problema
- CPU
  - ① lettura / scrittura da / su memoria RAM o ROM
  - ② nuova configurazione dei registri della CPU
  - ③ scelta della cella di memoria su cui operare (indirizzo contenuto nel RI)
  - ④ È **generale**, nel senso che può eseguire programmi diversi

# MdT: il modello matematico

Rendiamo il tutto più formale.



# MdT: il modello matematico

Una MdT è definita da una quintupla:

$$M = (X, Q, fm, fd, \delta)$$

- $X$  = insieme finito di simboli
  - comprende il *blank* ovvero *cella vuota*
- $Q$  = insieme finito di stati
  - comprende *HALT* che definisce la terminazione

# MdT: il modello matematico

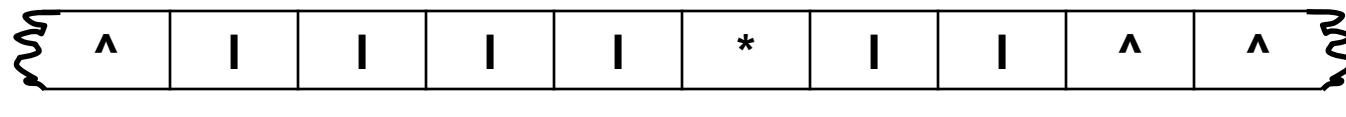
Una MdT è definita da una quintupla:

$$M = (X, Q, fm, fd, \delta)$$

- Funzione di macchina  $fm : Q \times X \rightarrow X$ 
  - Determina il simbolo da scrivere sul nastro
- Funzione di direzione  $fd : Q \times X \rightarrow \{S, D, F\}$ 
  - Determina lo spostamento della testina
  - S=sinistra, D=destra, F=ferma
- Funzione di transizione di stato  $\delta : Q \times X \rightarrow Q$ 
  - Definisce lo stato successivo della computazione

# Scrivere algoritmi per MdT: nastro

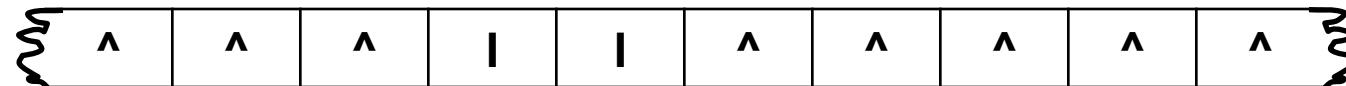
- Definire un'opportuna configurazione iniziale del nastro
  - Codificare i dati
  - Es.: nastro iniziale per problema della sottrazione tra interi



$$4 - 2$$

operandi codificati con 'I' e separati da \*  
blank=^

- Definire un'opportuna configurazione finale del nastro che rappresenti la soluzione



# Scrivere algoritmi per MdT: il controllo

- Definire le funzioni  $fm$ ,  $fd$ ,  $\delta$ 
  - in modo da trasformare la configurazione iniziale in quella che rappresenta la soluzione
- In pratica il programma per una MdT è una sequenza di quintuple:
$$\langle x_i \in X, q_j \in Q, x_{ij} \in X, \{S, D, F\}, q_{ij} \in Q \rangle$$

- In corrispondenza di un simbolo letto  $x_i$  e dello stato  $q_j$  in cui si trova l'unità di controllo, si determinano:
  - Il simbolo  $x_{ij} = fm(x_i, q_j)$  da scrivere nella cella corrente
  - Lo spostamento della testina  $fd(x_i, q_j)$
  - Il nuovo stato  $q_{ij} = \delta(x_i, q_j)$  in cui MdT continuerà la computazione

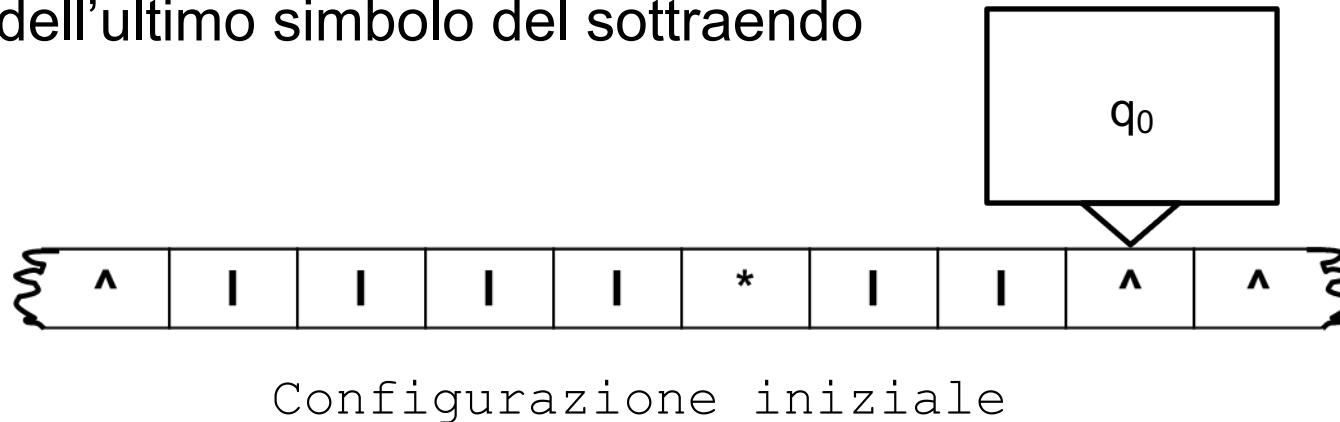
# Scrivere algoritmi per MdT: sottrazione tra interi

- Progettiamo un algoritmo per eseguire la sottrazione tra due numeri interi  $n$  e  $m$ ,  $n \geq 0$ ,  $m \geq 0$ 
  - Per semplicità assumiamo che  $n \geq m$
  - La testina è posizionata sulla prima cella vuota a destra dell'ultimo simbolo del sottraendo



# Scrivere algoritmi per MdT: sottrazione tra interi

- Progettiamo un algoritmo per eseguire la sottrazione tra due numeri interi  $n$  e  $m$ ,  $n \geq 0$ ,  $m \geq 0$ 
  - Per semplicità assumiamo che  $n \geq m$
  - La testina è posizionata sulla prima cella vuota a destra dell'ultimo simbolo del sottraendo

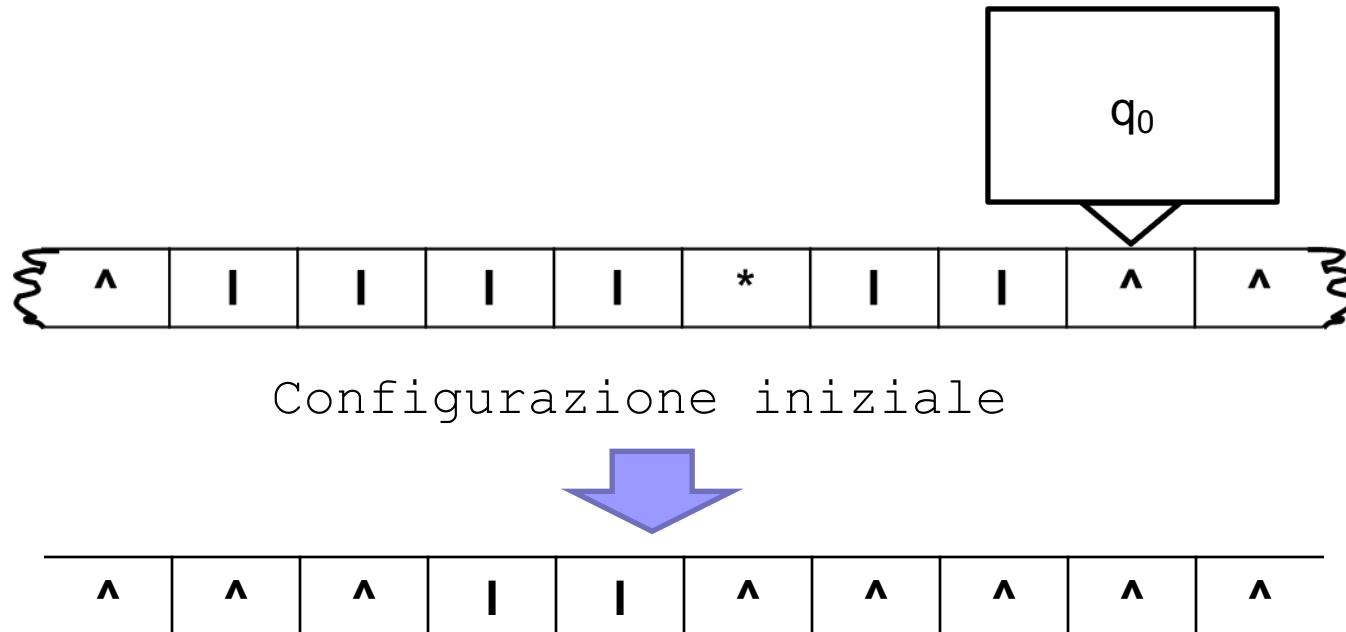


- Il modello di calcolo ci "obbliga" a pensare l'algoritmo in base alle operazioni possibili

# Un possibile algoritmo per calcolare $n-m$



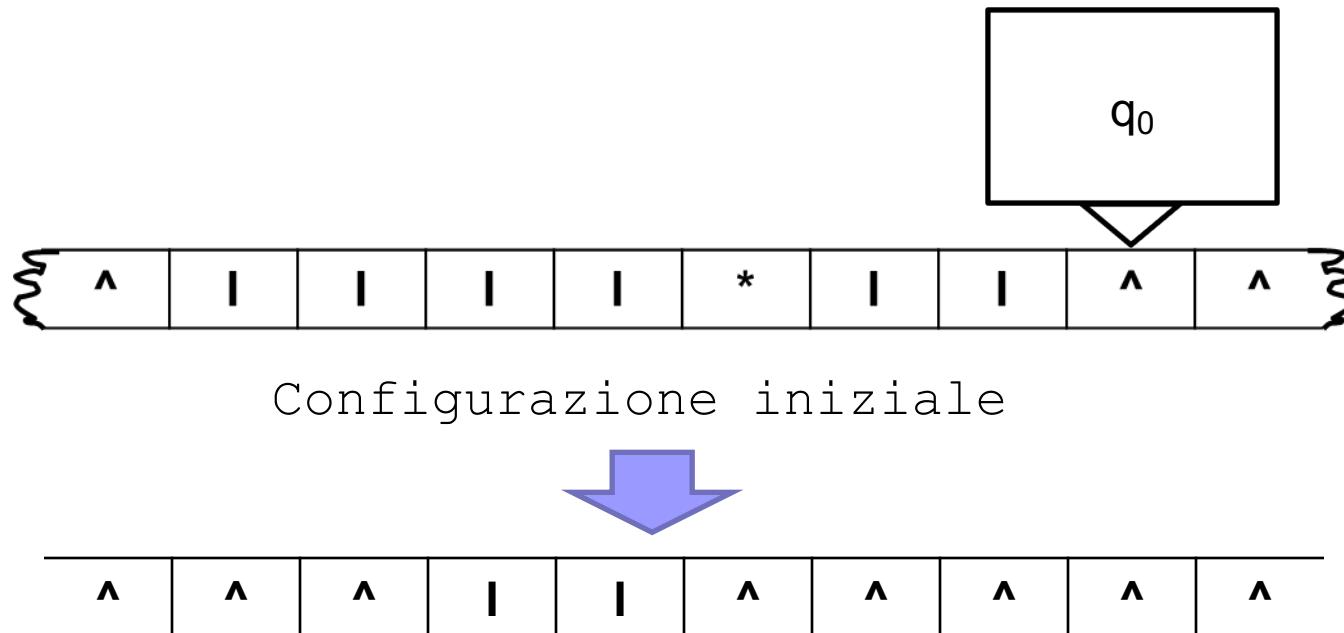
Quale può essere una possibile procedura solutiva?



# Un possibile algoritmo per calcolare $n-m$



Cancellare ugual numero di simboli dal minuendo  $n$  e dal sottraendo  $m$  in modo che sul nastro resti solo il risultato finale



# Un possibile algoritmo per calcolare $n-m$

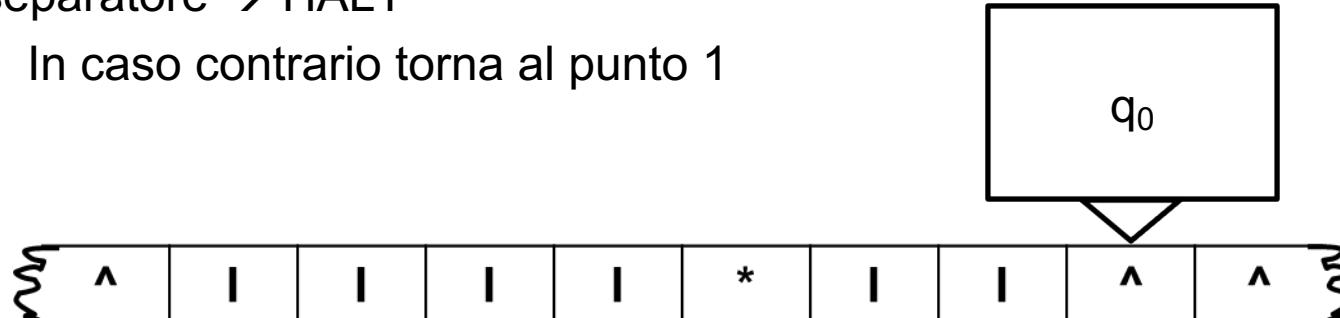
1. Diminuisci di una unità  $m$
2. Spostati a S in cerca del primo simbolo di  $n$
3. Cancellalo
4. Spostati a D in cerca dell'ultimo simbolo di  $m$
5. Se non ci sono più simboli da cancellare da  $m$  allora cancella il separatore → HALT
  - In caso contrario torna al punto 1

- Aggiungiamo le situazioni da «ricordare»
  - Gli stati

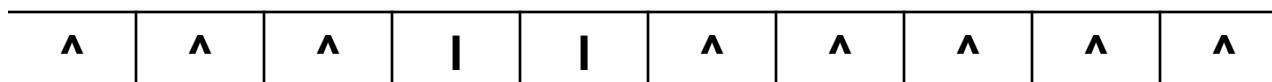


# Un possibile algoritmo per calcolare $n-m$

1. Diminuisci di una unità  $m$ 
  - ricorda di aver cancellato un simbolo da  $m$
2. Spostati a S in cerca del primo simbolo di  $n$
3. Cancellalo
  - Ricorda che ora entrambi gli operandi sono stati diminuiti di una unità
4. Spostati a D in cerca dell'ultimo simbolo di  $m$
5. Se non ci sono più simboli da cancellare da  $m$  allora cancella il separatore → HALT
  - In caso contrario torna al punto 1



Configurazione iniziale



# Le 5-ple di una MdT per la sottrazione

$$X = \{I, *, ^\wedge\}$$

$$Q = \{q_0, q_1, q_2, q_3, \text{HALT}\}$$

In quali «stati» mi posso trovare?

# Le 5-ple di una MdT per la sottrazione

$$X = \{I, *, \wedge\}$$

$$Q = \{q_0, q_1, q_2, q_3, \text{HALT}\}$$

$q_0 \equiv$  stato iniziale della computazione ovvero  
ricerca ultimo simbolo di  $m$

$q_1 \equiv$  diminuito  $m$

$q_2 \equiv$  raggiunto simbolo iniziale di  $n$

$q_3 \equiv$  diminuiti entrambi operandi

A partire da questi stati, posso definire il set di istruzioni che determina il comportamento della macchina

# Le 5-ple di una MdT per la sottrazione

$$X = \{ |, *, ^\}$$

$$Q = \{ q_0, q_1, q_2, q_3, \text{HALT} \}$$

$q_0 \equiv$  stato iniziale della computazione ovvero  
ricerca ultimo simbolo di  $m$

$q_1 \equiv$  diminuito  $m$

$q_2 \equiv$  raggiunto simbolo iniziale di  $n$

$q_3 \equiv$  diminuiti entrambi operandi

$\langle ^, q_0, ^, S, q_0 \rangle$

$\langle |, q_0, ^, S, q_1 \rangle$

$\langle *, q_0, ^, F, \text{HALT} \rangle$

$\langle ^, q_1, ^, D, q_2 \rangle$

$\langle |, q_1, |, S, q_1 \rangle$

$\langle *, q_1, *, S, q_1 \rangle$

$\langle |, q_2, ^, D, q_3 \rangle$

$\langle ^, q_3, ^, S, q_0 \rangle$

$\langle |, q_3, |, D, q_3 \rangle$

$\langle *, q_3, *, D, q_3 \rangle$

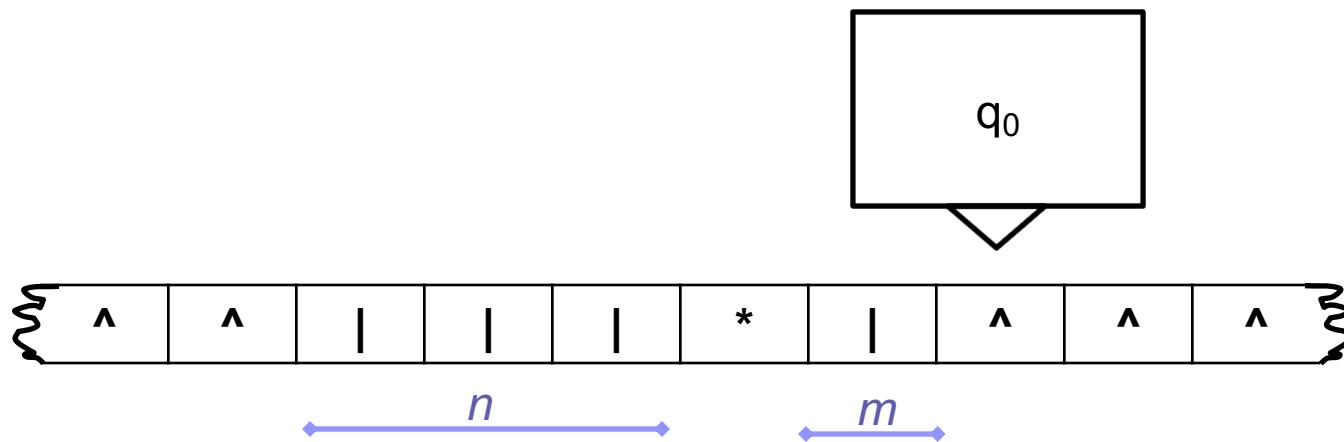
# La Matrice Funzionale

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$

Perché alcune transizioni non sono definite?

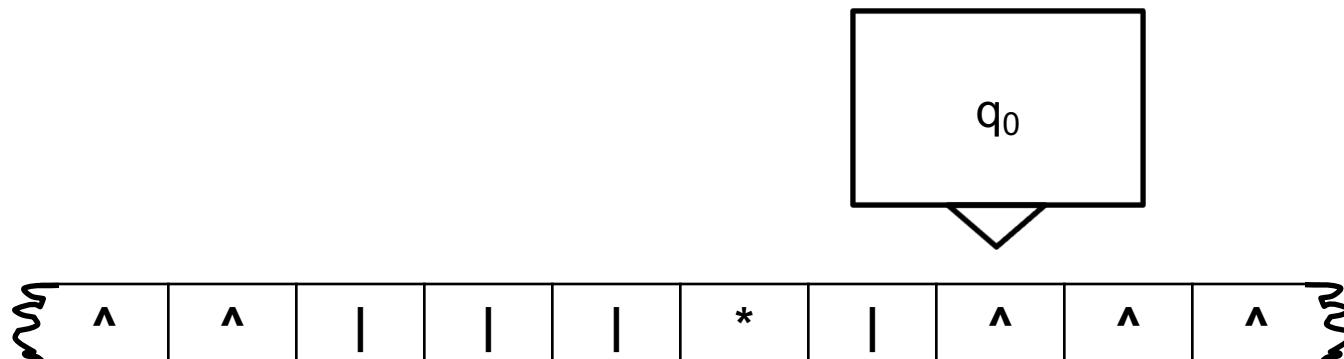
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



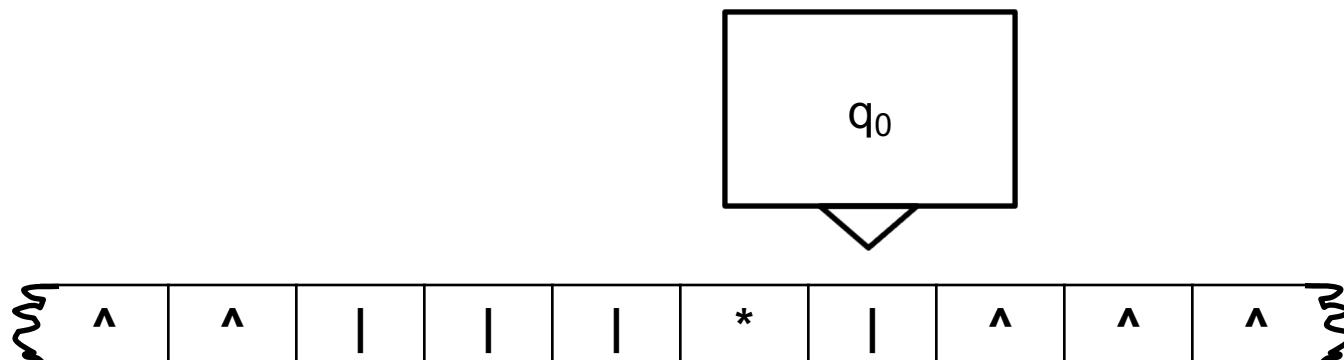
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



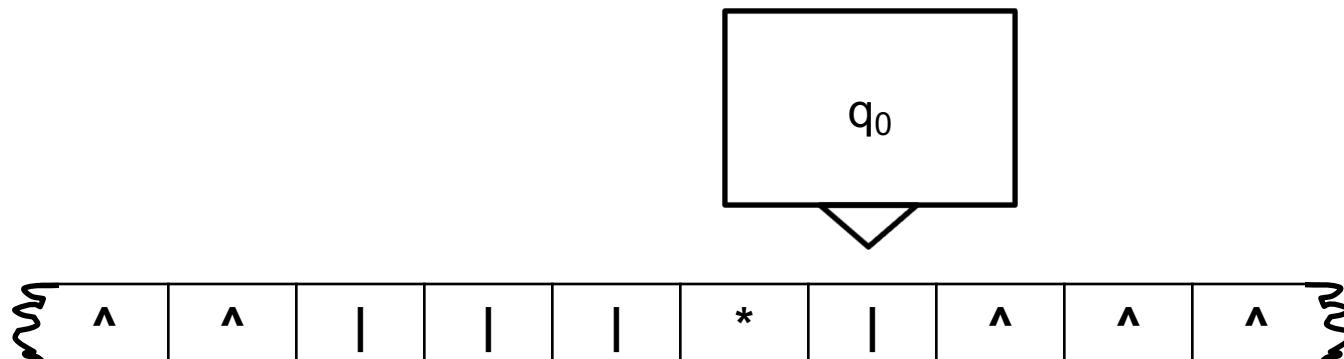
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



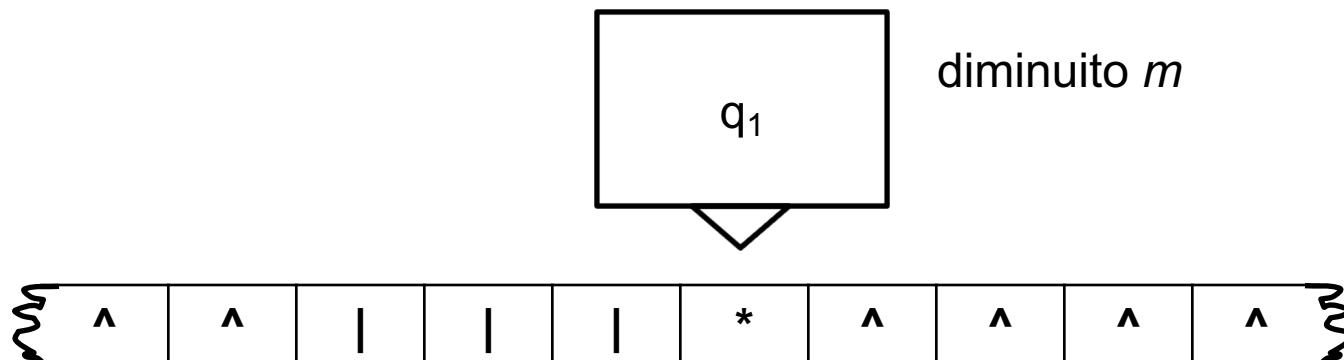
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ HALT	$* S q_1$		$* D q_3$



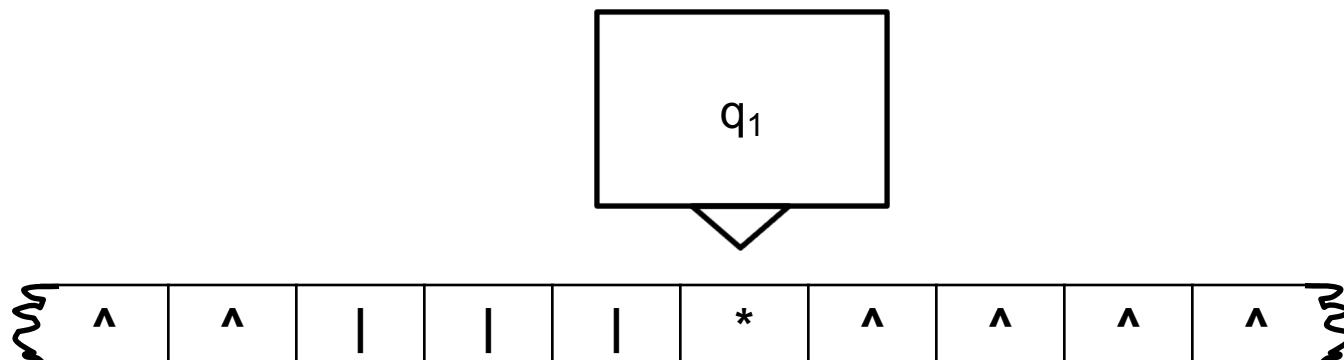
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



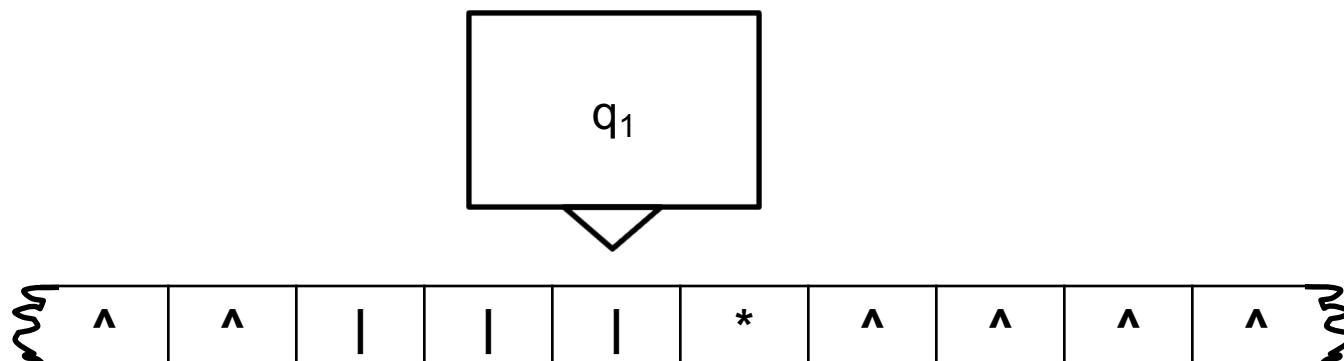
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ HALT	$* S q_1$		$* D q_3$



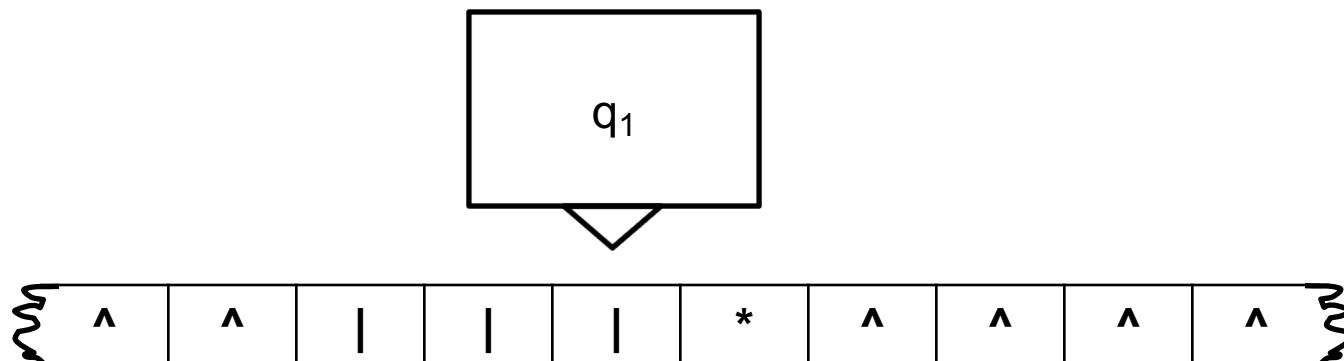
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



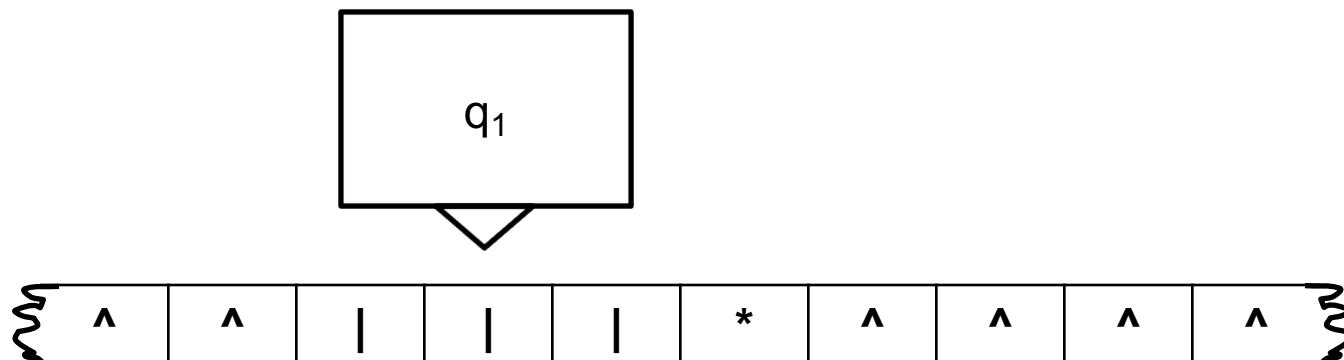
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



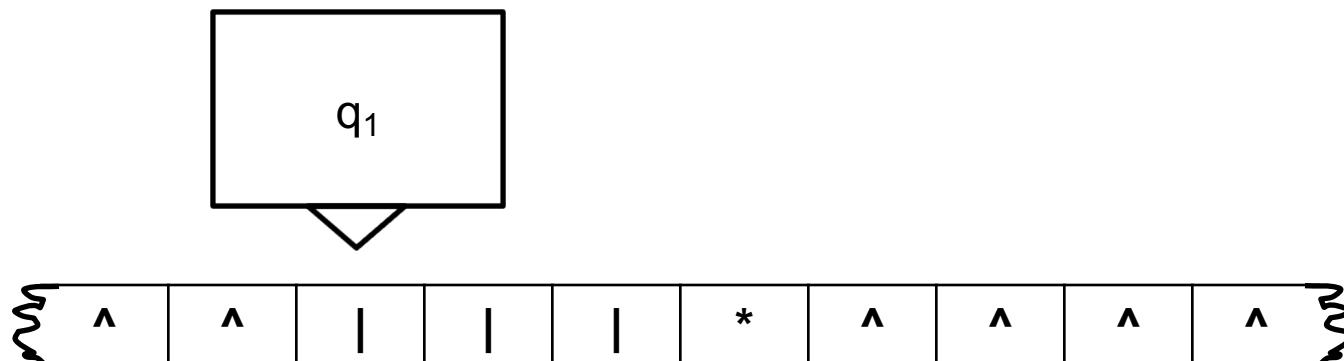
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



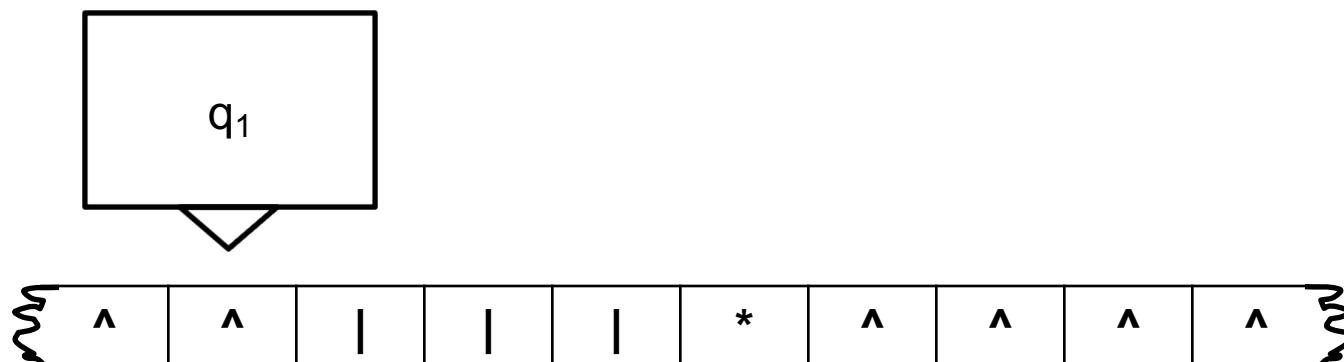
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



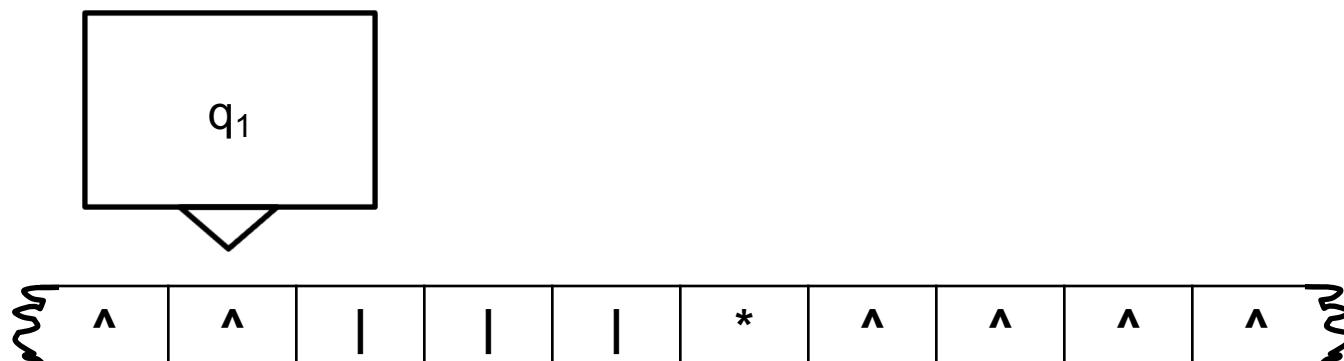
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



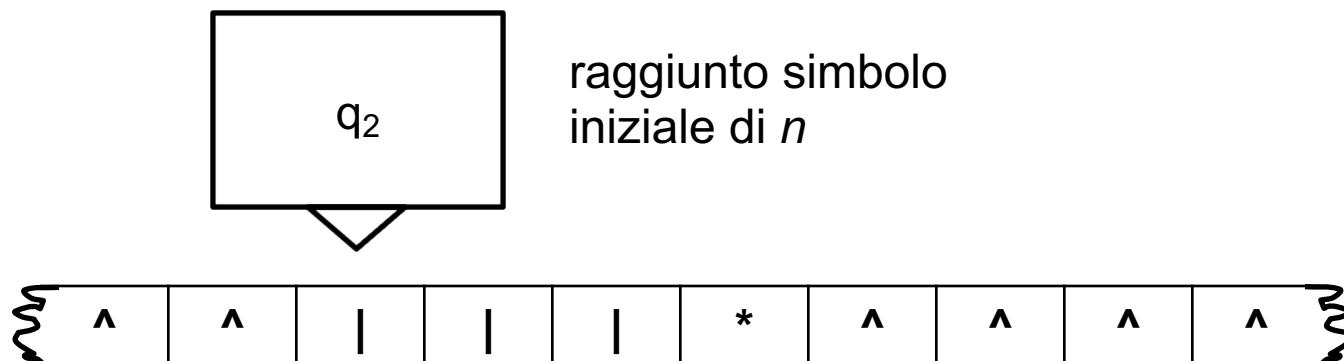
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



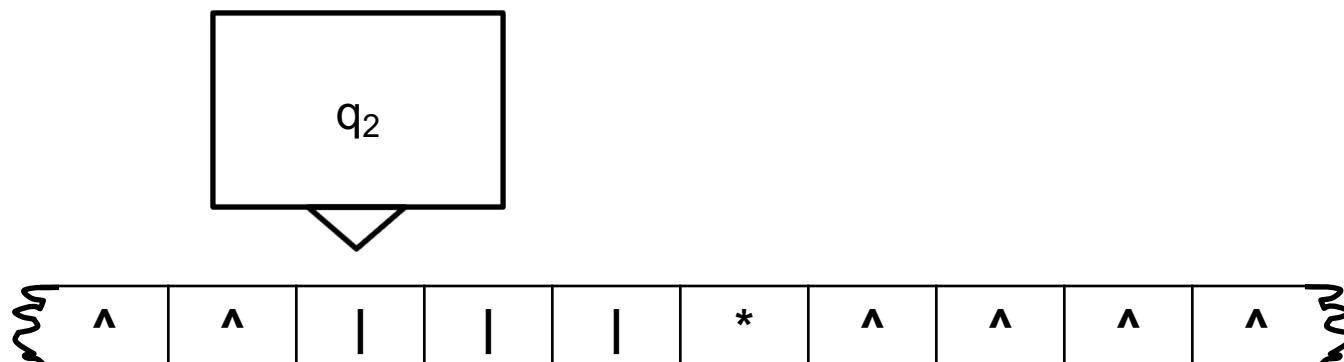
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



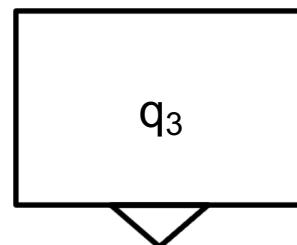
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$

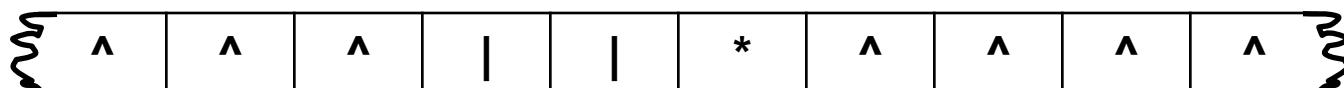


# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
$ $	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$

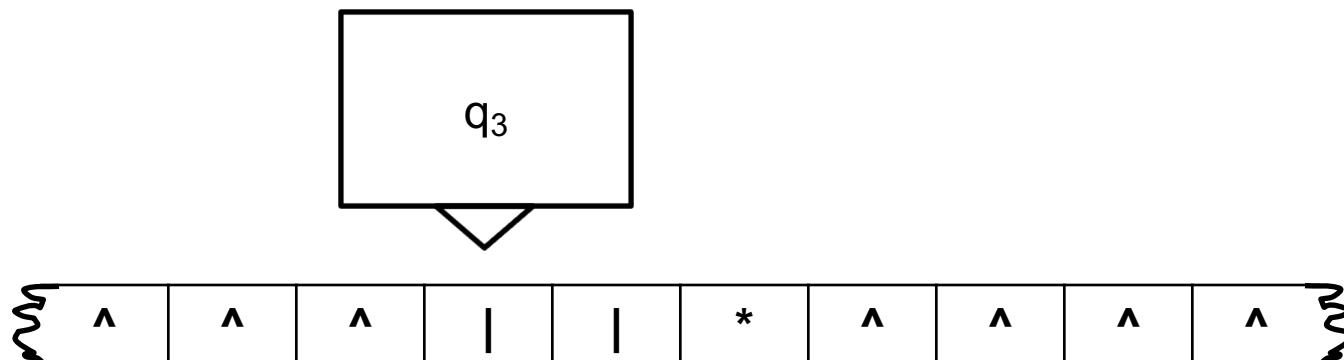


Diminuiti entrambi  
gli operandi



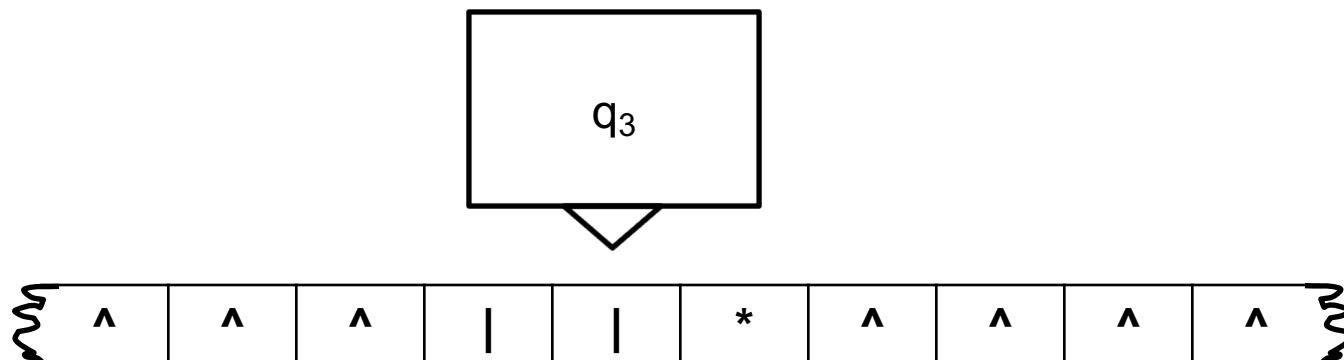
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



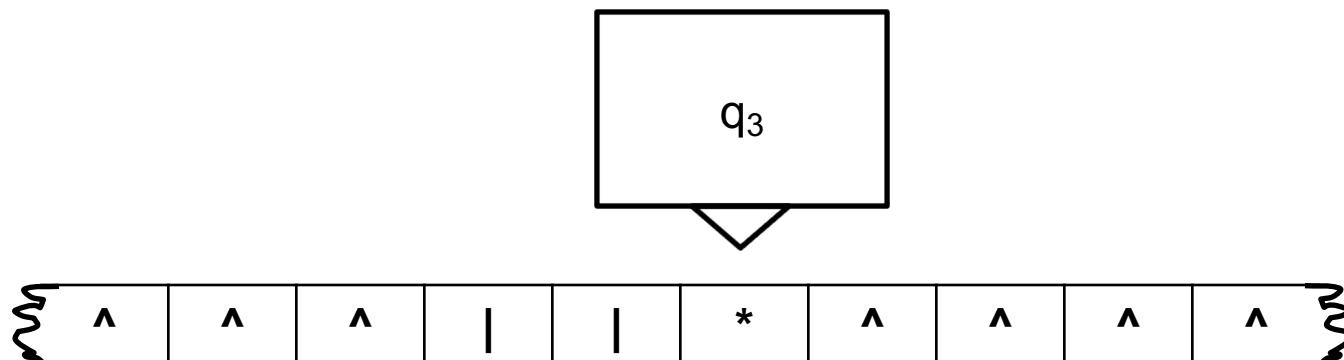
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



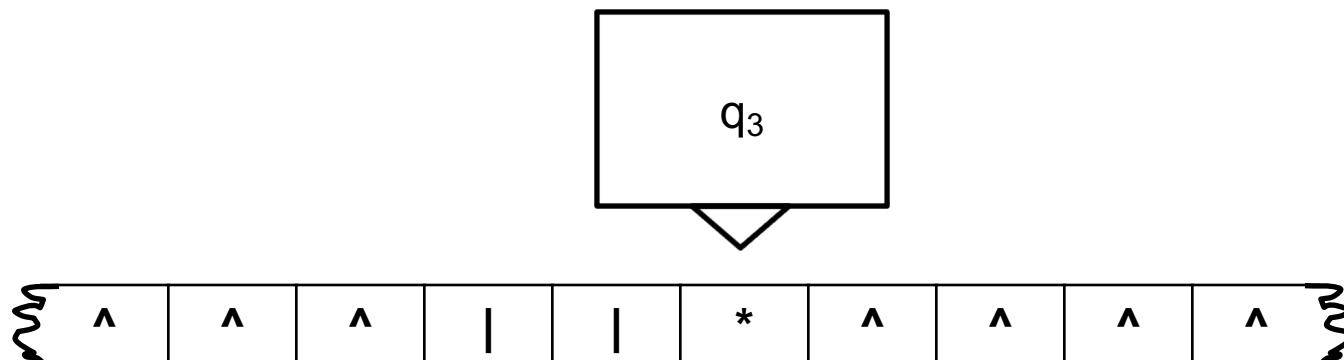
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



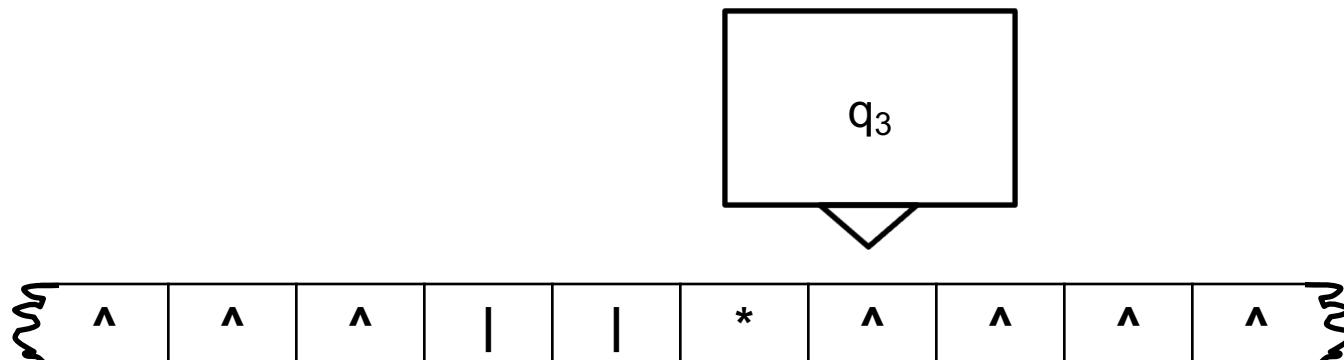
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ HALT	$* S q_1$		$* D q_3$



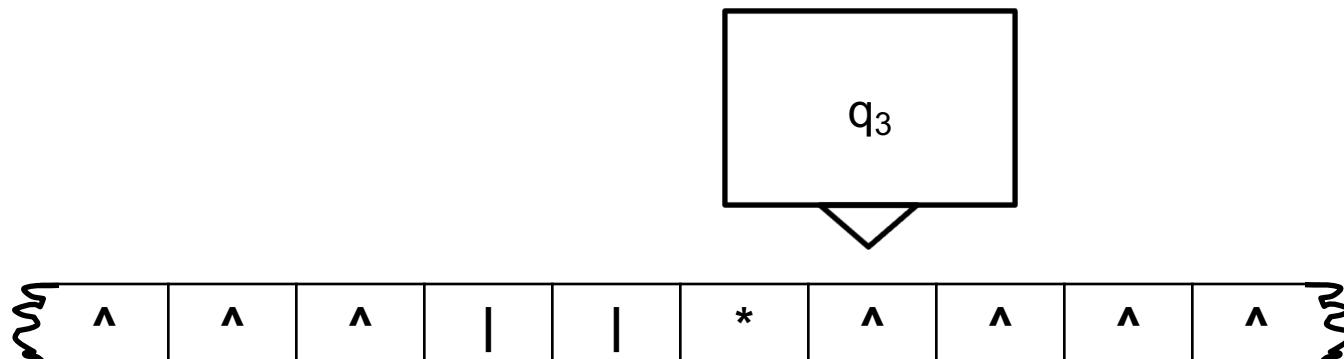
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



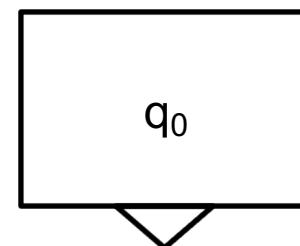
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$

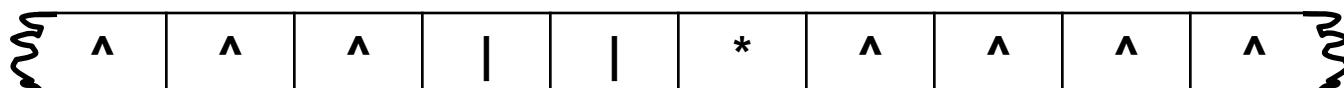


# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$^S q_0$	$^D q_2$		$^S q_0$
	$^S q_1$	$  S q_1$	$^D q_3$	$  D q_3$
*	$^F \text{ HALT}$	$* S q_1$		$* D q_3$



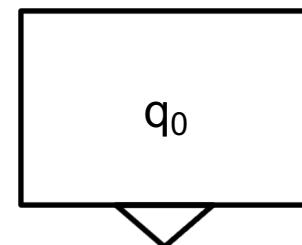
Stato iniziale della  
computazione



# Computazione 3-1

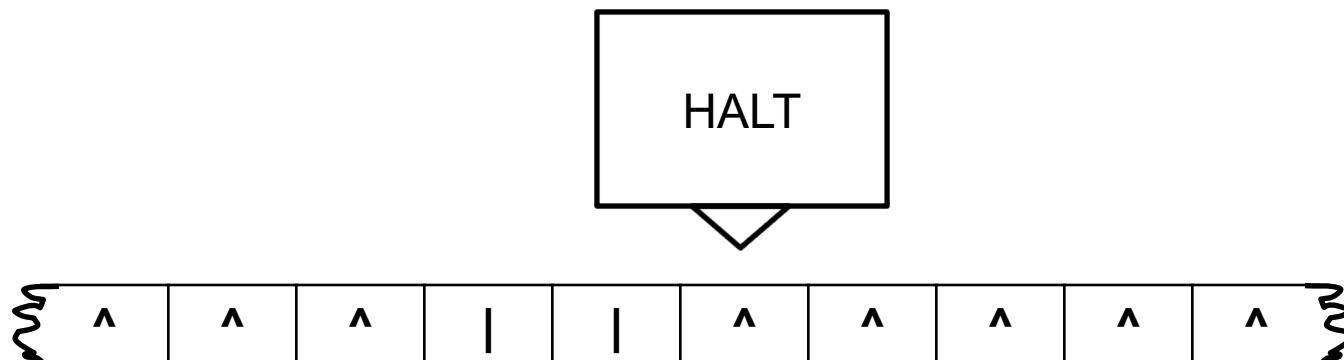
$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$^S q_0$	$^D q_2$		$^S q_0$
	$^S q_1$	$  S q_1$	$^D q_3$	$  D q_3$
*	$^F \text{ HALT}$	$* S q_1$		$* D q_3$

trovare un \* nello stato iniziale  
della computazione è segno  
del fatto che non ci sono più  
simboli da processare in  $m$



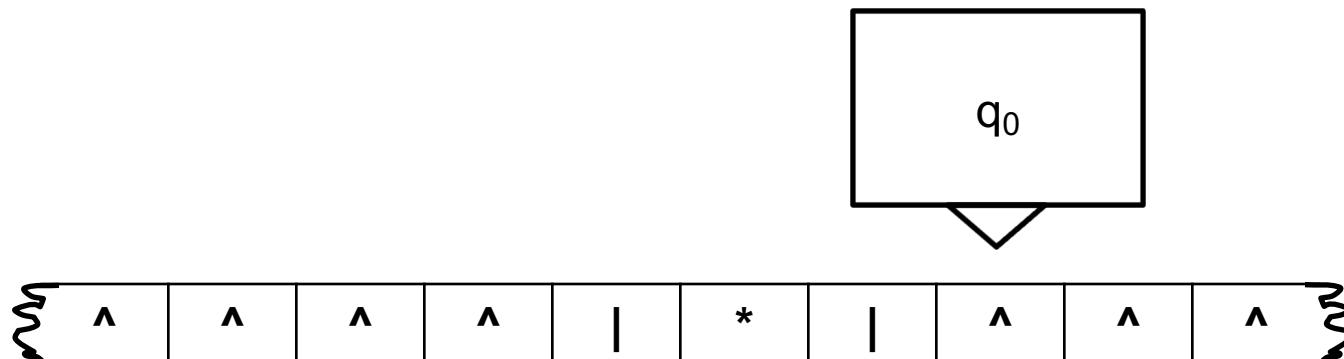
# Computazione 3-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ HALT	$* S q_1$		$* D q_3$



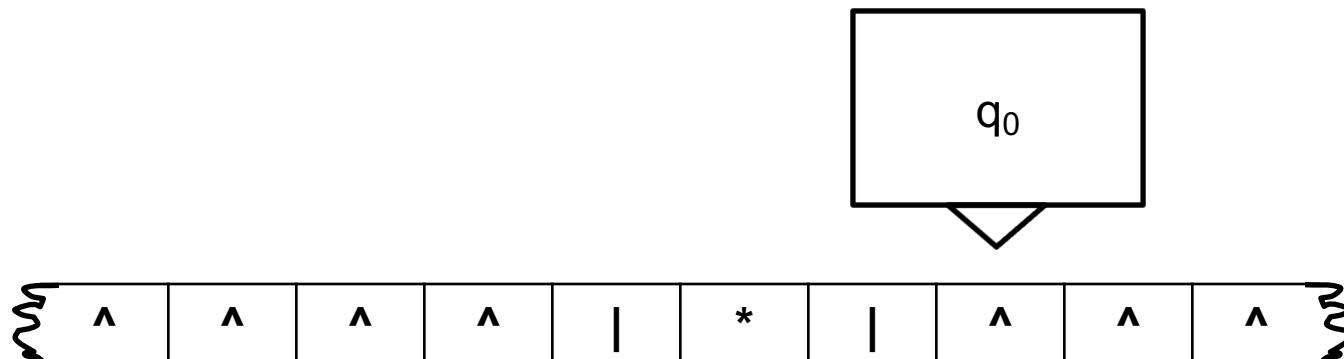
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



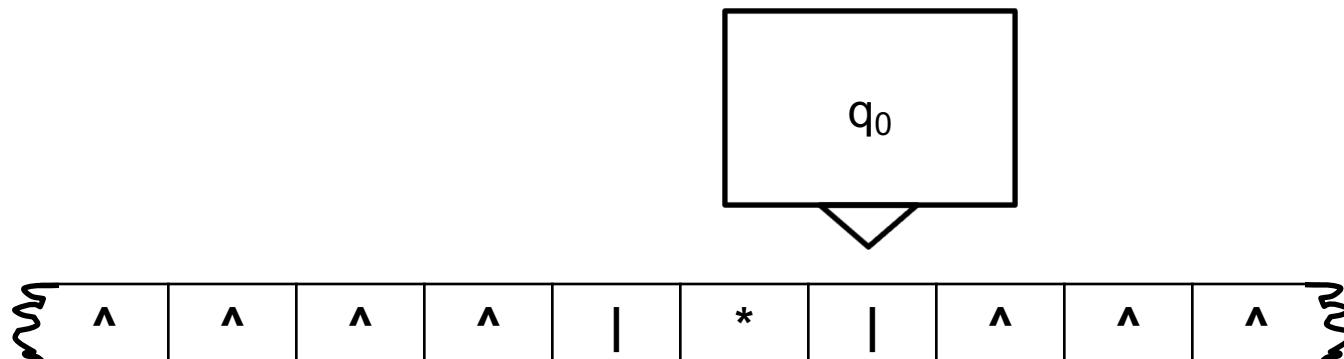
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



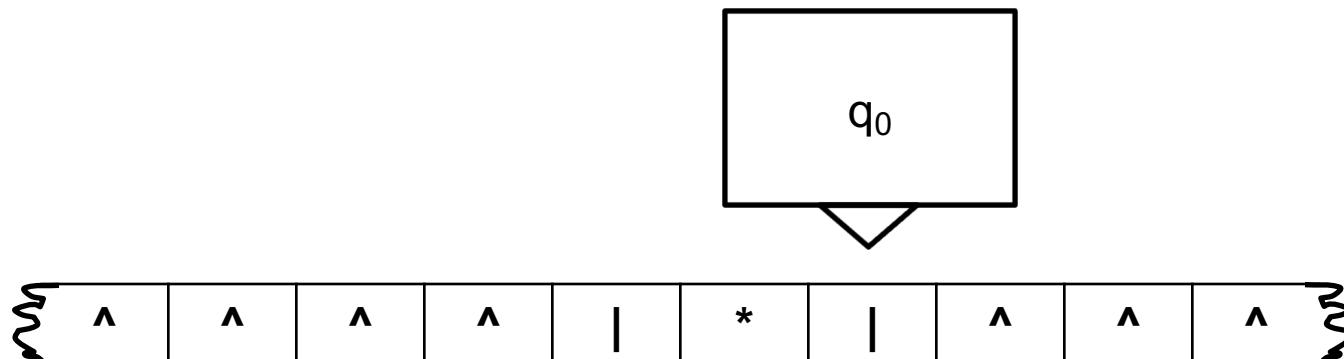
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



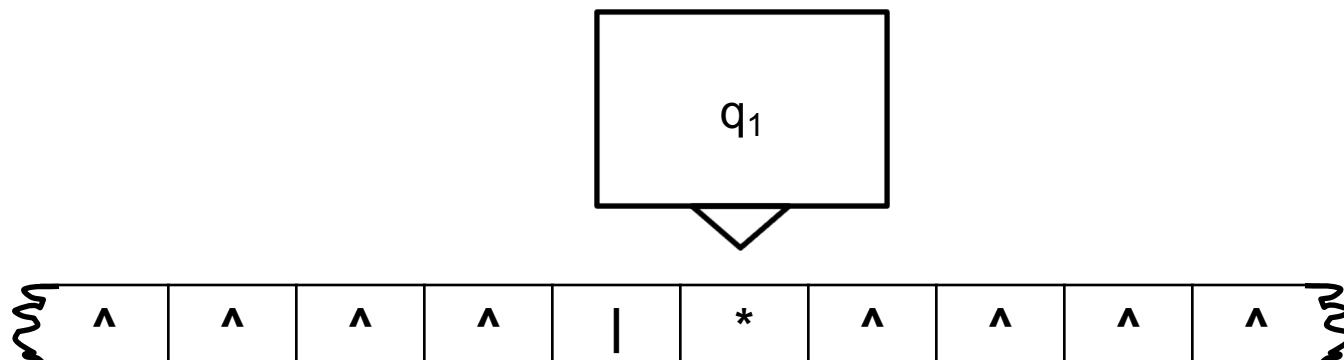
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



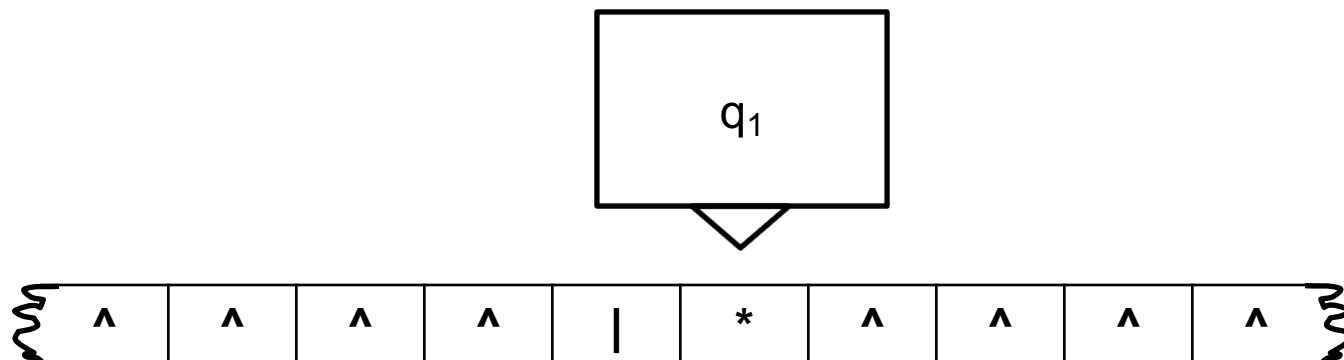
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ HALT	$* S q_1$		$* D q_3$



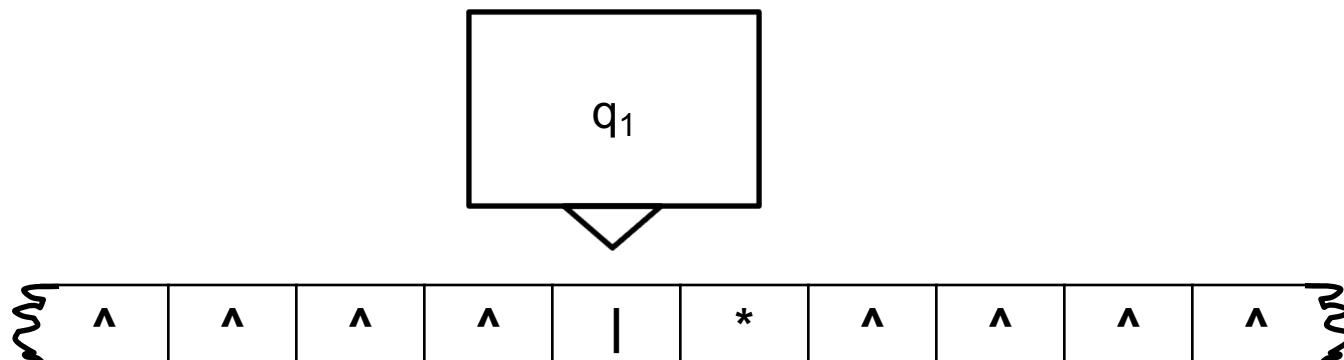
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ HALT	$* S q_1$		$* D q_3$



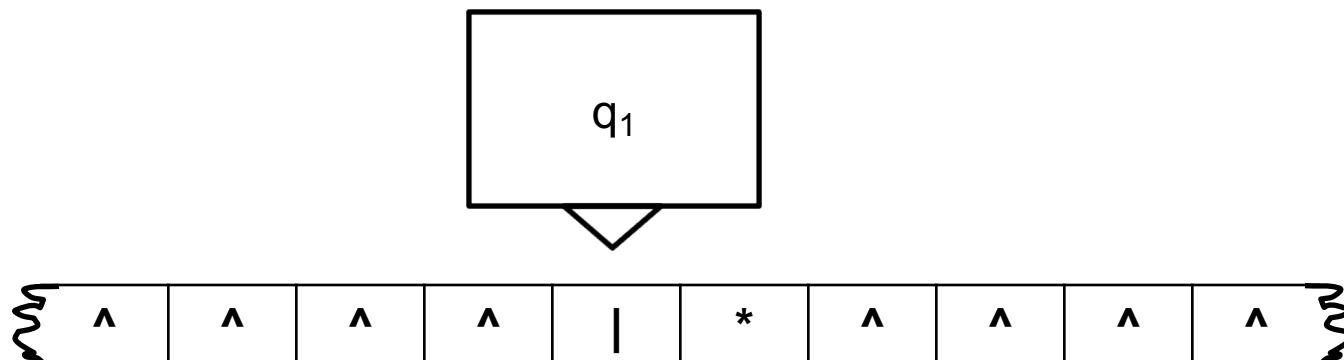
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



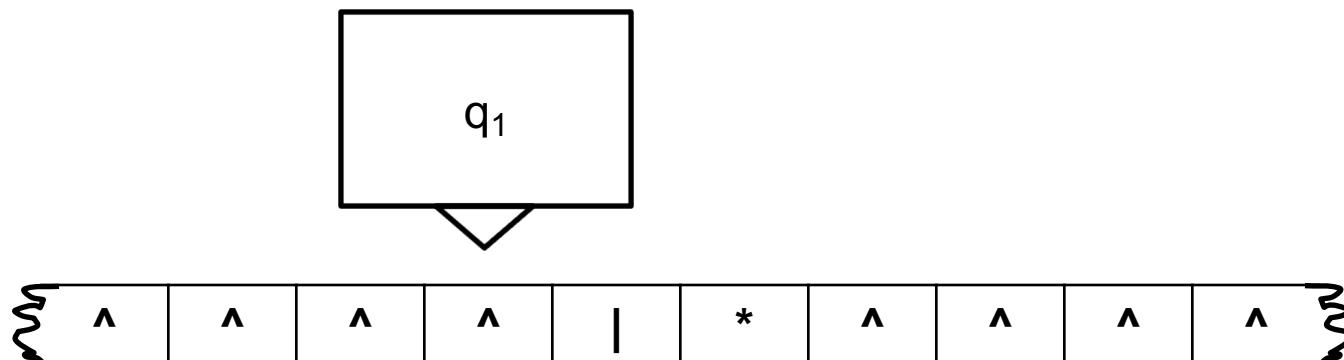
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



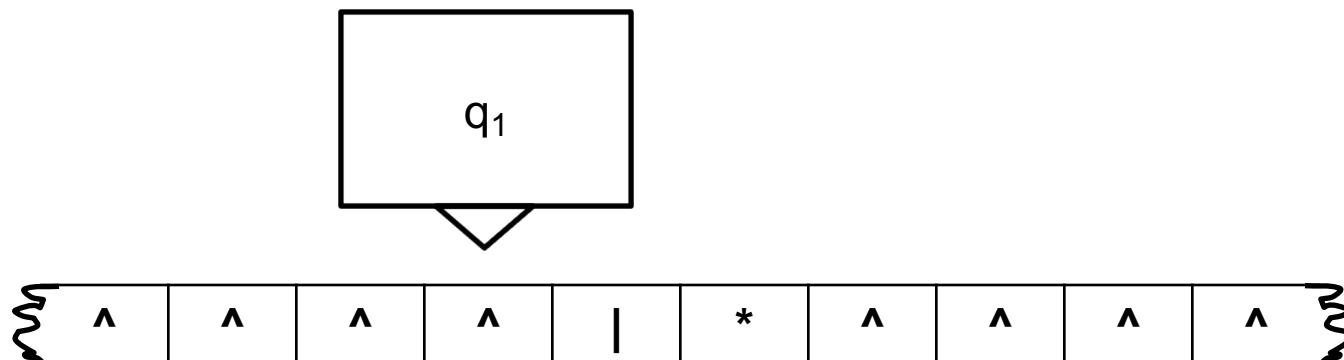
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



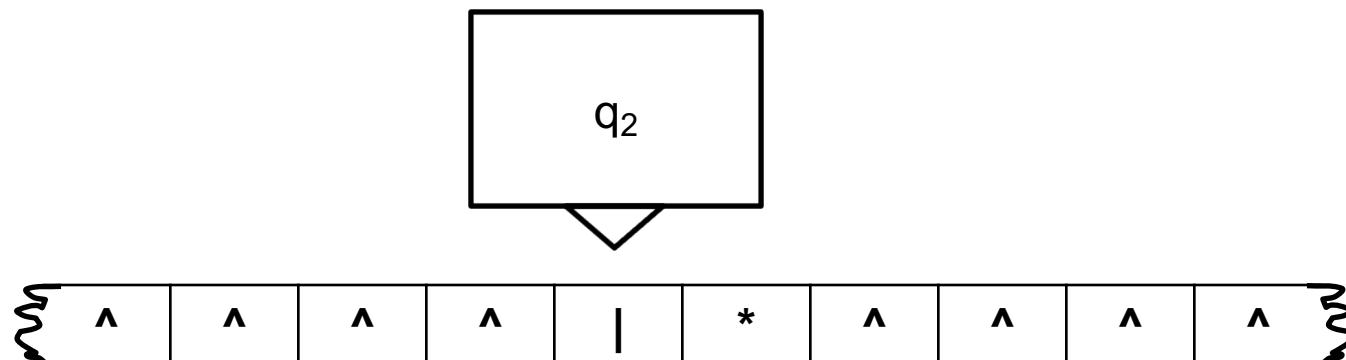
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



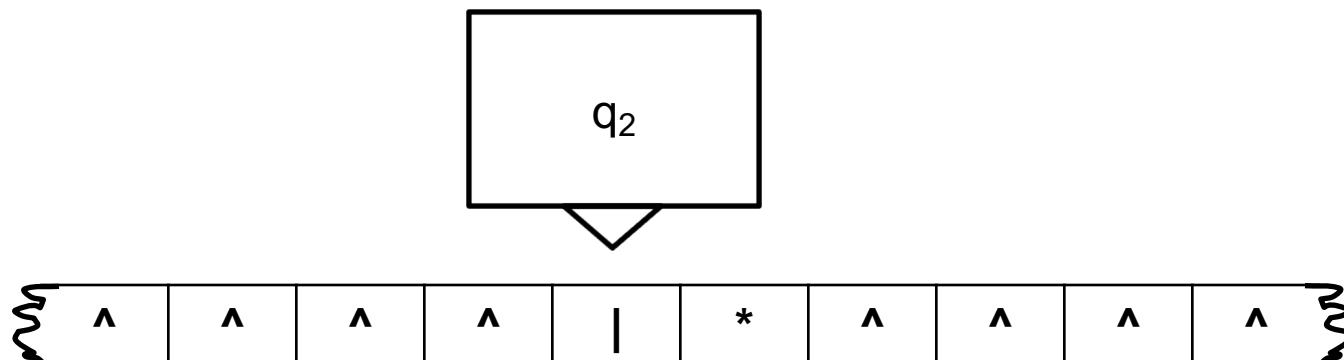
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



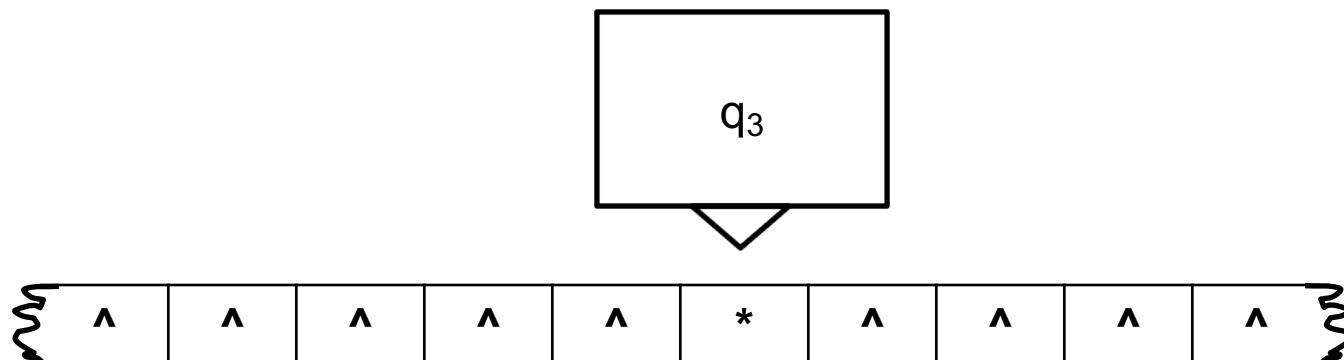
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



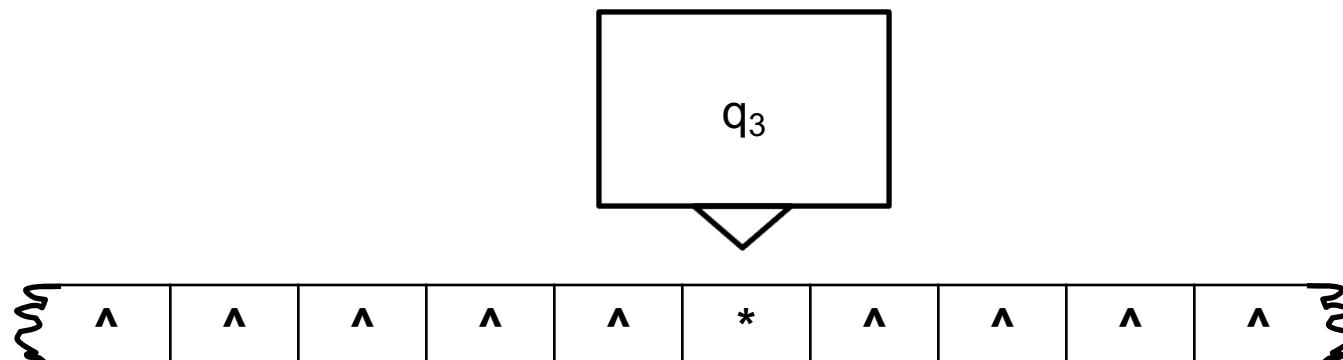
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



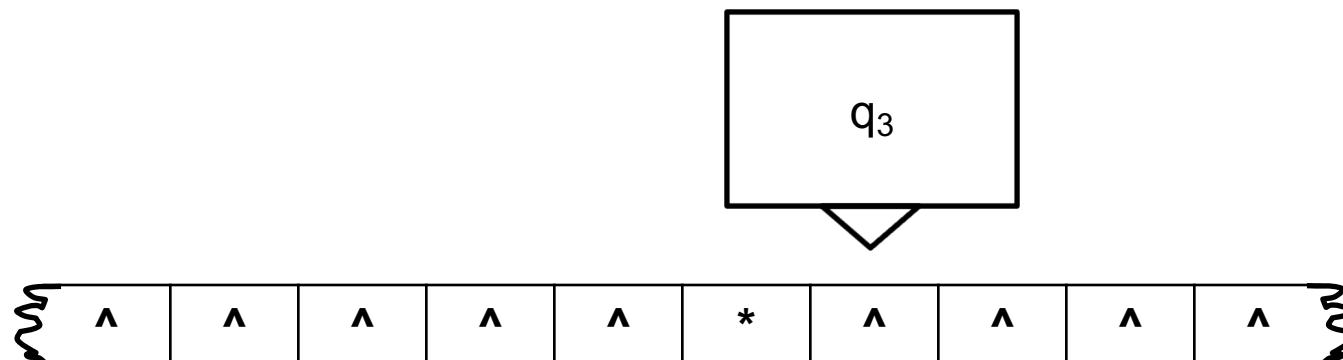
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ HALT	$* S q_1$		$* D q_3$



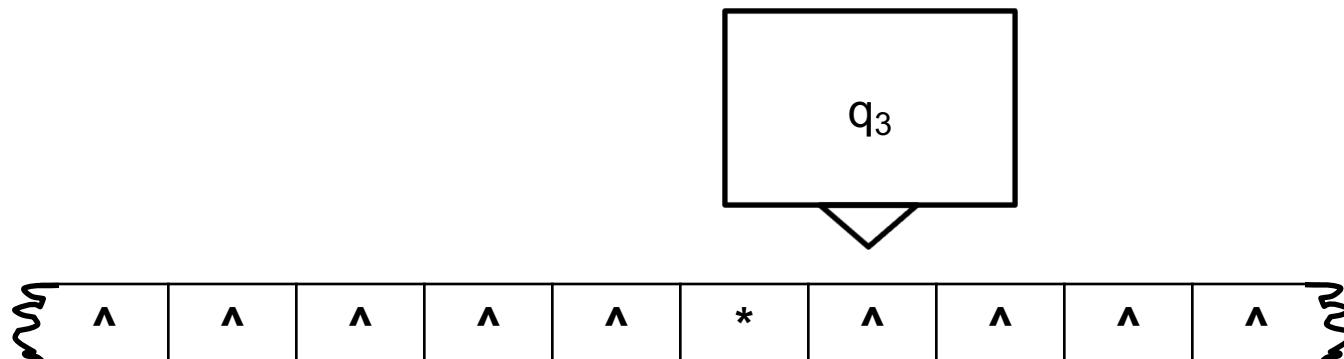
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



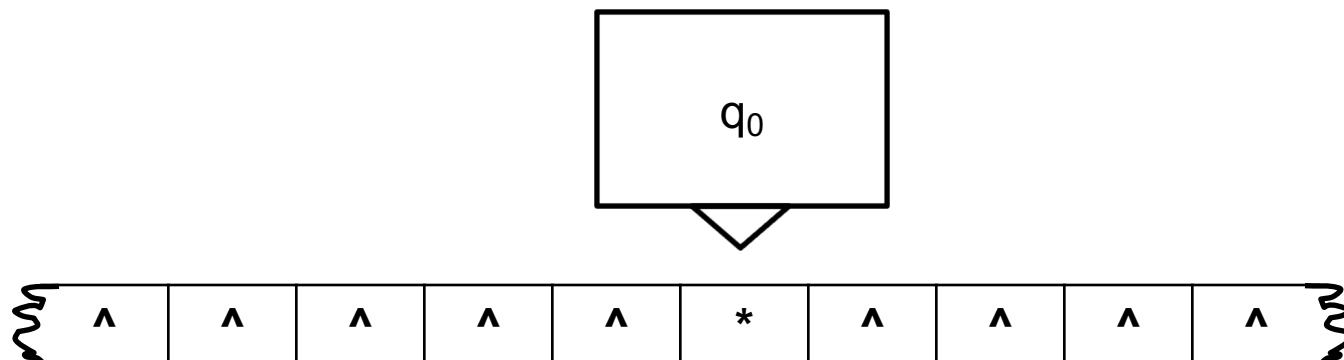
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



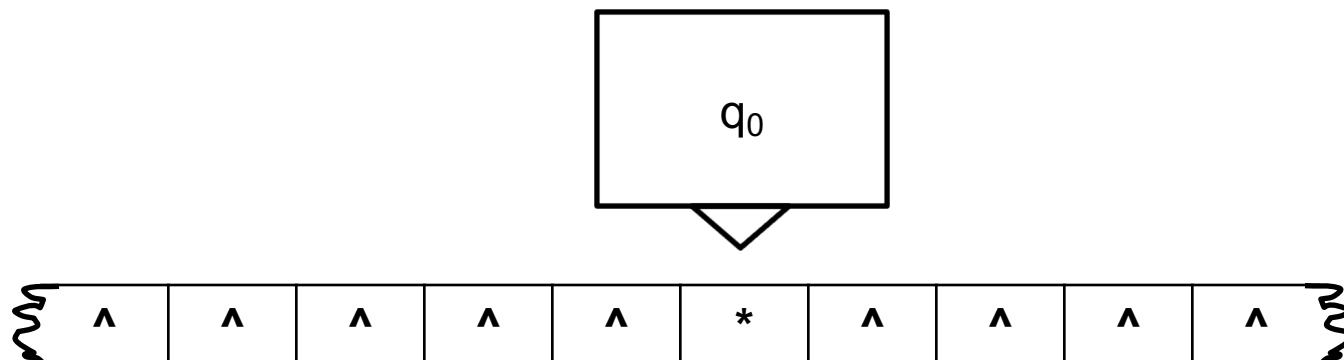
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



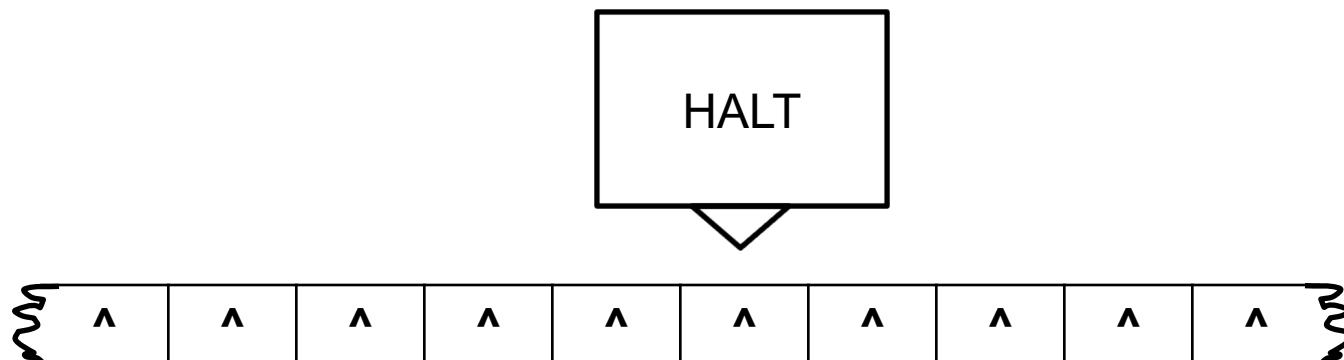
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ <u>HALT</u>	$* S q_1$		$* D q_3$



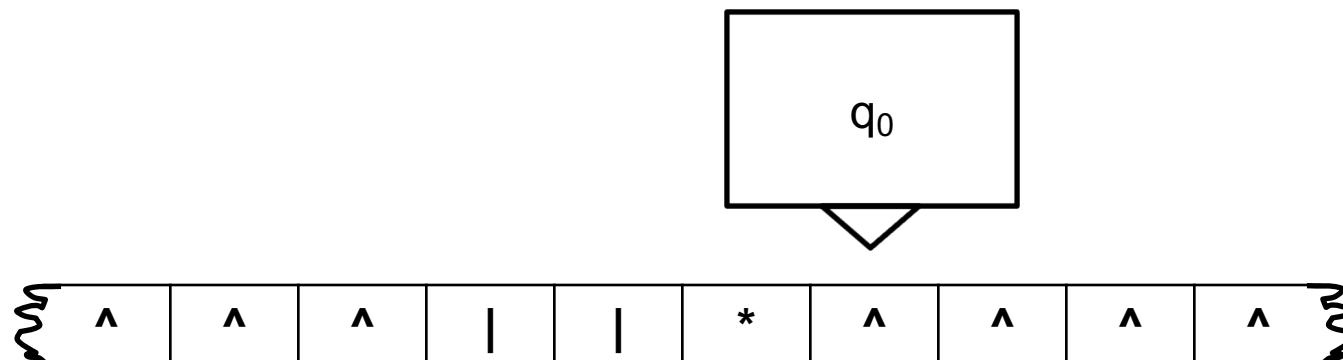
# Test casi particolari: computazione 1-1

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$^S q_0$	$^D q_2$		$^S q_0$
	$^S q_1$	$  S q_1$	$^D q_3$	$  D q_3$
*	$^F$ HALT	$* S q_1$		$* D q_3$



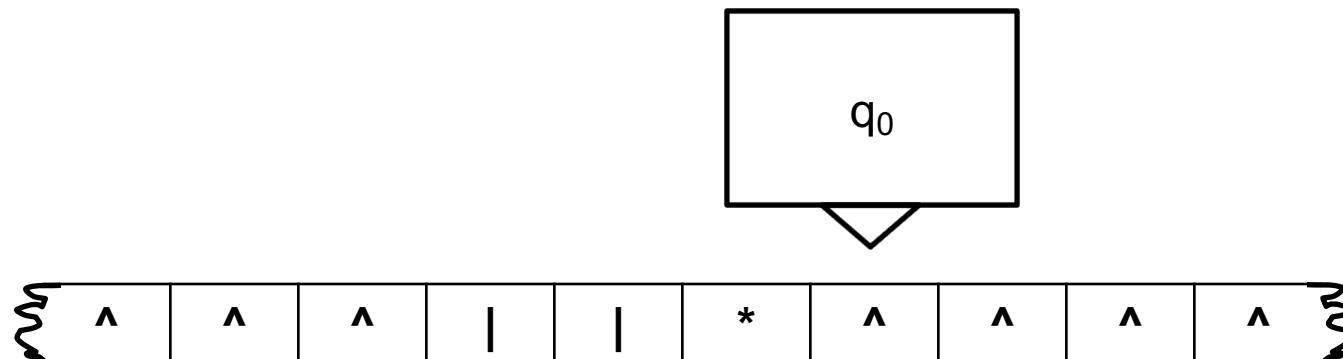
# Test casi particolari: computazione 2-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



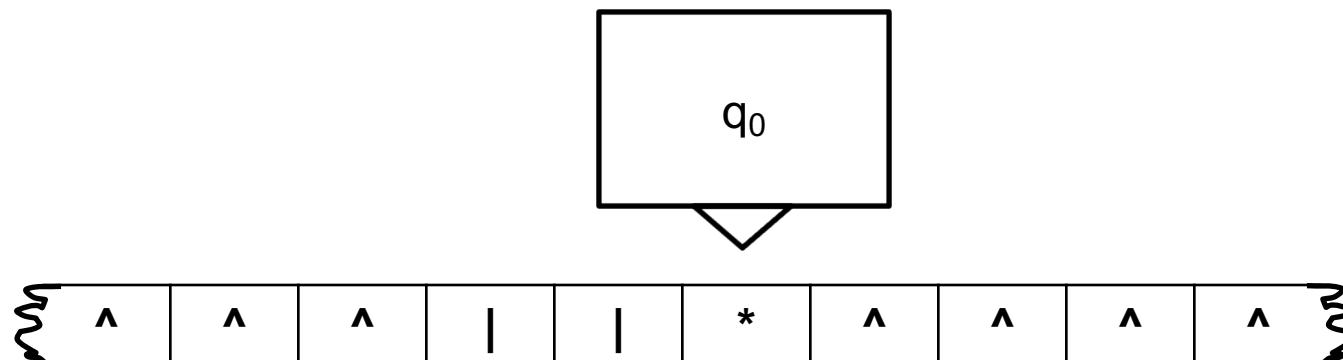
# Test casi particolari: computazione 2-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



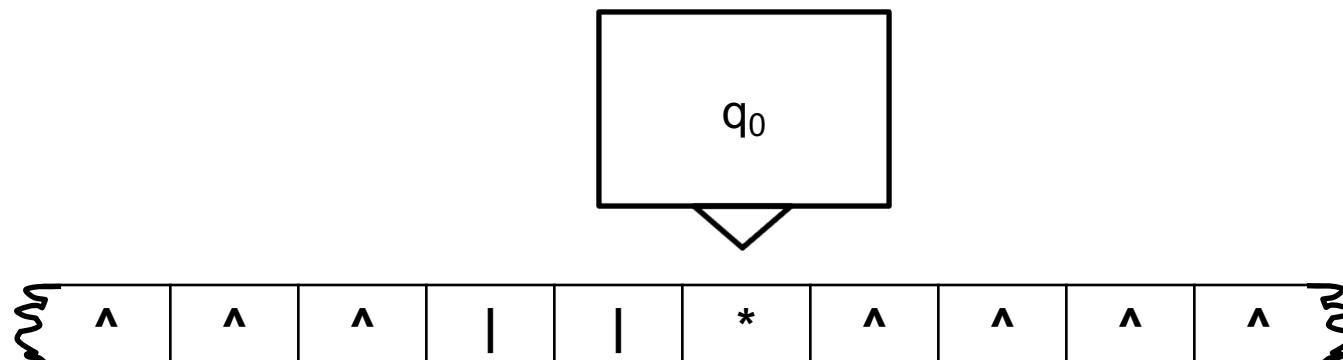
# Test casi particolari: computazione 2-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



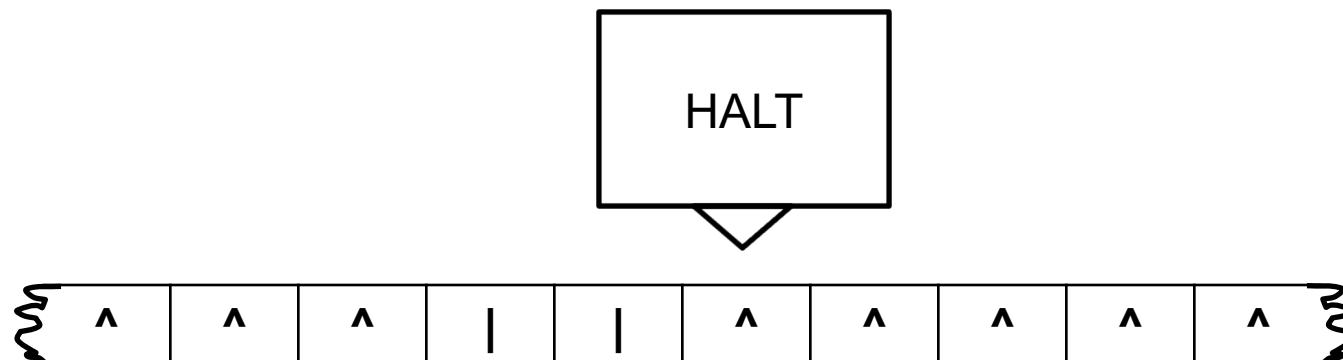
# Test casi particolari: computazione 2-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ <u>HALT</u>	$* S q_1$		$* D q_3$



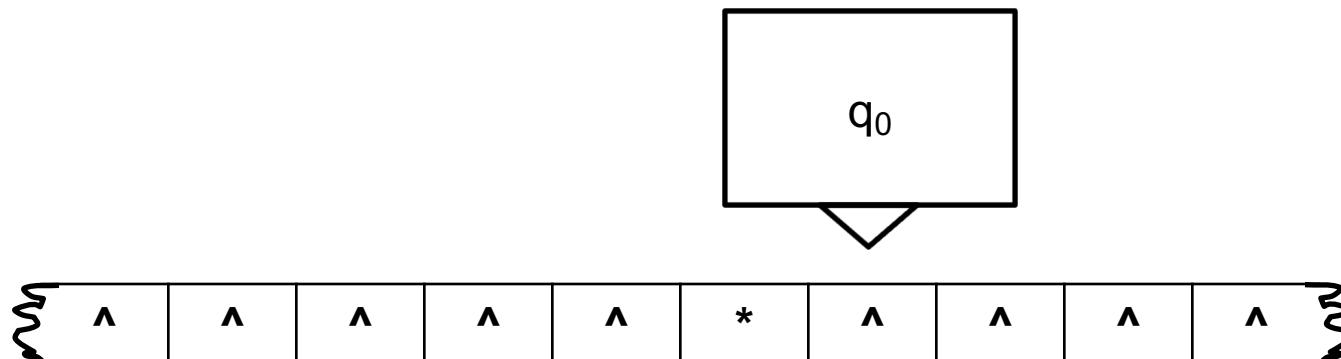
# Test casi particolari: computazione 2-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$^S q_0$	$^D q_2$		$^S q_0$
	$^S q_1$	$  S q_1$	$^D q_3$	$  D q_3$
*	$^F$ HALT	$* S q_1$		$* D q_3$



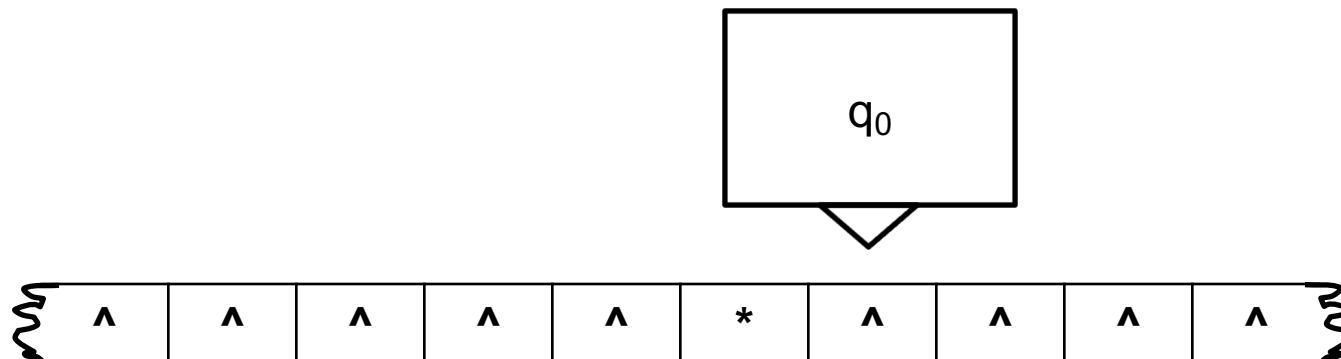
# Test casi particolari: computazione 0-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



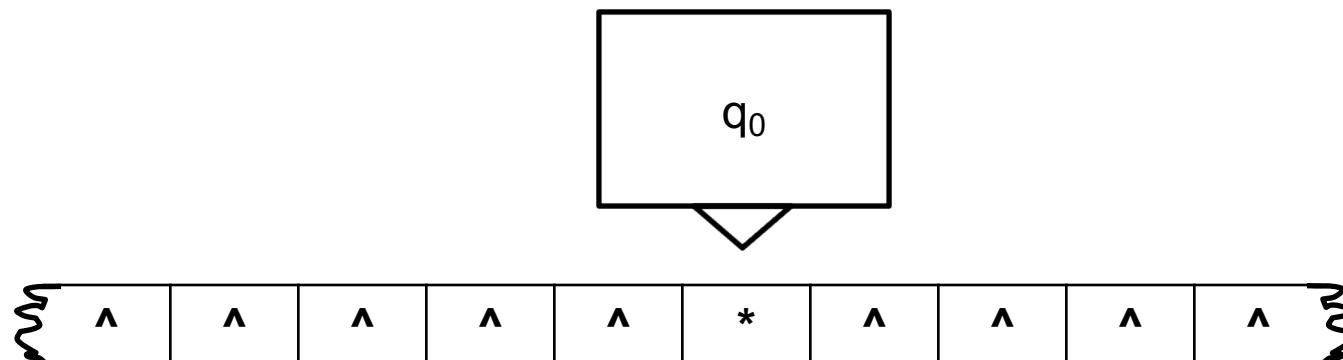
# Test casi particolari: computazione 0-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



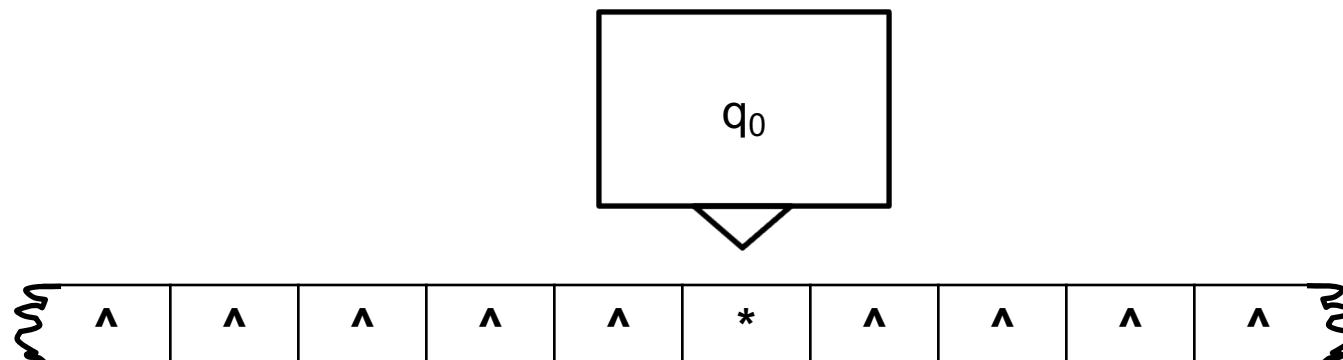
# Test casi particolari: computazione 0-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
	$\wedge Sq_1$	$  Sq_1$	$\wedge Dq_3$	$  Dq_3$
*	$\wedge F$ HALT	$* Sq_1$		$* Dq_3$



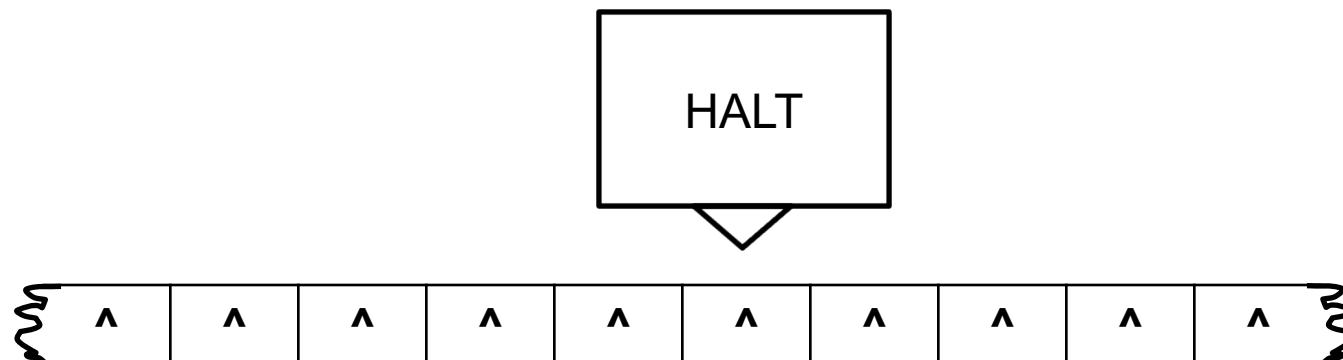
# Test casi particolari: computazione 0-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$\wedge S q_0$	$\wedge D q_2$		$\wedge S q_0$
	$\wedge S q_1$	$  S q_1$	$\wedge D q_3$	$  D q_3$
*	$\wedge F$ <u>HALT</u>	$* S q_1$		$* D q_3$



# Test casi particolari: computazione 0-0

$x \setminus Q$	$q_0$	$q_1$	$q_2$	$q_3$
$\wedge$	$^S q_0$	$^D q_2$		$^S q_0$
	$^S q_1$	$  S q_1$	$^D q_3$	$  D q_3$
*	$^F$ HALT	$* S q_1$		$* D q_3$



# Have Fun with MdT



- Stabilire una stringa binaria contiene lo stesso numero di '0' e '1'
- Stabilire se una stringa binaria è palindroma (ovvero si legge indifferentemente da S a D, es.: 01000010)
- Stabilire se un numero rappresentato con 'l' è pari oppure dispari

# Tesi di Church-Turing

- La classe delle funzioni calcolabili coincide con la classe delle funzioni calcolabili da MdT
  - *ogni funzione calcolabile è calcolata da una MdT*
  - *Non esiste alcun formalismo capace di risolvere una classe di problemi più ampia di quella che si può risolvere con MdT*
- Le funzioni calcolabili con C o Java sono di più di quelle calcolabili con MdT?

NO

# La MdT Universale (MdTU)

- Legge dal nastro DATI e PROGRAMMA
  - *Il programma non è più cablato nell'unità di controllo*
  - *Codificato sul nastro come i dati*
  - *In pratica sono rappresentate sul nastro anche le 5-ple che definiscono l'algoritmo solutivo*
- E' una macchina programmabile
  - prende le 5-ple (istruzioni) dal nastro → FETCH
  - le decodifica → DECODE
  - le esegue scrivendo sul nastro → EXECUTE
- Cosa vi ricorda?

# La MdT Universale (MdTU)

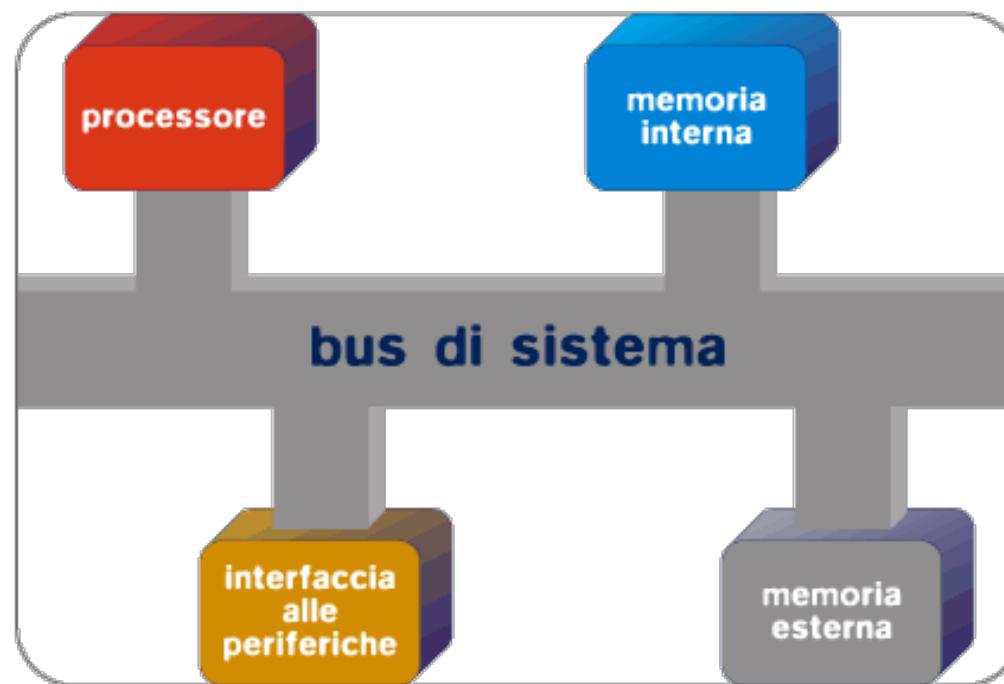
- Legge dal nastro DATI e PROGRAMMA
  - *Il programma non è più cablato nell'unità di controllo*
  - *Codificato sul nastro come i dati*
  - *In pratica sono rappresentate sul nastro anche le 5-ple che definiscono l'algoritmo solutivo*
- E' una macchina programmabile
  - prende le 5-ple (istruzioni) dal nastro → DECODA
  - le decodifica → DECODE
  - le esegue scrivendo su
- E' un interprete!



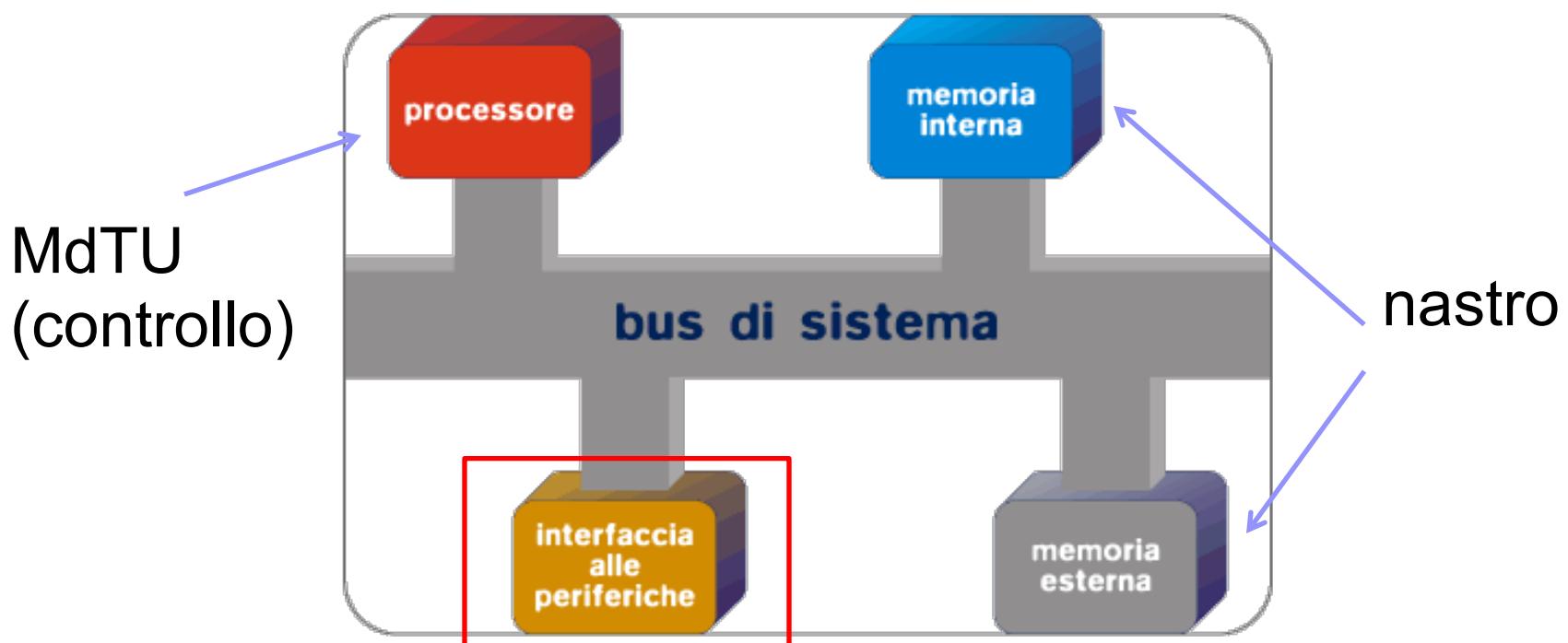
# La MdT Universale (MdTU)

- Legge dal nastro DATI e PROGRAMMA
  - *Il programma non è più cablato nell'unità di controllo*
  - *Codificato sul nastro come i dati*
  - *In pratica sono rappresentate sul nastro anche le 5-ple che definiscono l'algoritmo solutivo*
- E' una macchina programmabile
  - prende le 5-ple (istruzioni) dal nastro → FETCH
  - le decodifica → DECODE
  - le esegue scrivendo sul nastro → EXECUTE
- **Cosa altro vi ricorda?**

# MdTU e Macchina Von Neumann



# MdTU e Macchina Von Neumann

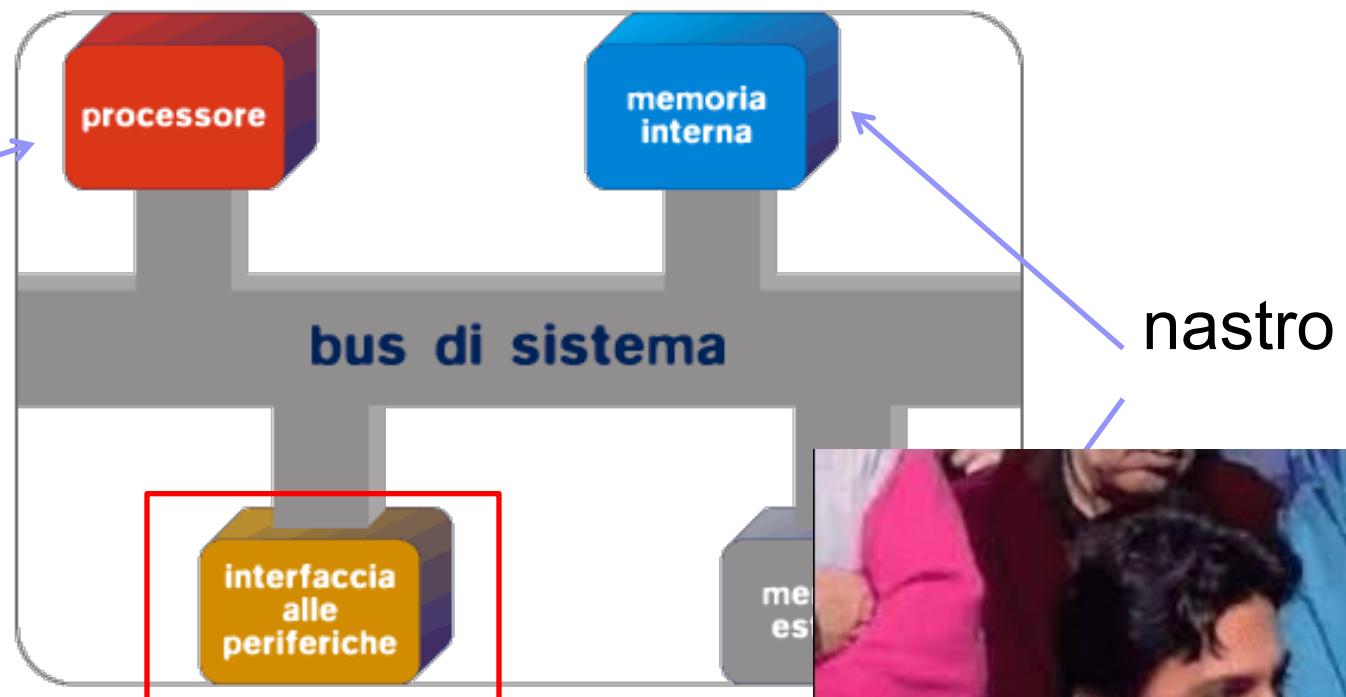


MdTU è un modello della macchina di Von Neumann ovvero un  
modello degli attuali calcolatori!!!

(manca solo la parte di I/O)

# MdTU e Macchina Von Neumann

MdTU  
(controllo)



MdTU è un modello della macchina di Von Neumann.  
modello degli attuali calcolatori

(manca solo la parte di I/O)



# Recap

- Un linguaggio di programmazione  $\mathcal{L}$  è un **formalismo per portare al livello di macchina fisica gli algoritmi**
  - *Implementare  $\mathcal{L}$  significa realizzarne l'interprete ovvero il programma che **esegue la traduzione** di  $\mathcal{L}$  per la macchina ospite*
- La possibilità di risolvere un problema non dipende da  $\mathcal{L}$ 
  - Tutti i linguaggi di programmazione calcolano esattamente le stesse funzioni calcolate dalle MdT
  - Tutti i linguaggi di programmazione sono Turing-completi

# Riferimenti

- Maurizio Gabbrielli, Simone Martini. *Linguaggi di Programmazione: Principi e paradigmi*. Seconda edizione. McGraw-Hill, 2011.
  - Capitoli 1 e 5

# Domande?



**Linguaggi di Programmazione**  
docente: Cataldo Musto  
[cataldo.musto@uniba.it](mailto:cataldo.musto@uniba.it)

# Introduzione: studio dei linguaggi

- Quando iniziamo a studiare un linguaggio tutti quanti godiamo di un grande vantaggio: **siamo esperti in un linguaggio**, quello con cui comunichiamo con gli altri.
- Oltre ad essere competenti in uno o più linguaggi naturali, lo studente in informatica ha familiarità con diversi linguaggi di programmazione, come il FORTRAN, il PASCAL, il C, il PROLOG, ...

# Introduzione: studio dei linguaggi

- Questi linguaggi sono usati per scrivere programmi e quindi per comunicare con il computer.
- Chiunque utilizzi un computer potrà notare che esso **è particolarmente limitato nel comprendere il significato desiderato** dei programmi.
- In linguaggio naturale possiamo di solito comunicare anche attraverso **frasi mal strutturate e incomplete**
- Quando dobbiamo comunicare con un computer la situazione cambia.

# Introduzione: studio dei linguaggi

- I nostri programmi devono **aderire rigorosamente a regole stringenti** e anche differenze minori vengono rigettate come errate.
  - Si chiamano appunto **linguaggi formali**
- Idealmente, ogni frase in un linguaggio dovrebbe essere corretta sia **semanticamente** (avere cioè il corretto significato) sia **sintatticamente** (avere la corretta struttura grammaticale).

# Introduzione: studio dei linguaggi

- Nell'italiano parlato, le frasi sono spesso sintatticamente sbagliate, ciononostante convogliano la semantica desiderata.
- Nella programmazione questo non è possibile.
  - E' essenziale che la **sintassi** sia corretta al fine di comunicare una qualsivoglia semantica.
  - Cosa è la semantica di una frase (o di un programma)?

# Sintassi e semantica

- Quando studiamo la **semantica** di una frase, ne stiamo studiando il significato.
- Tutte le frasi che seguono hanno la **stessa interpretazione** semantica, cioè lo stesso significato, sebbene siano **sintatticamente differenti**:
  - The man hits the dog
  - The dog is hit by the man
  - L'homme frappe le chien

- Concentriamoci per ora sulla **sintassi**

# Sintassi e semantica

- Lo studio della sintassi è lo studio della grammatica, cioè della struttura delle frasi. **Cosa significa analizzare sintatticamente una frase?**

# Sintassi e semantica

- Lo studio della sintassi è lo studio della grammatica, cioè della struttura delle frasi. **Cosa significa analizzare sintatticamente una frase?**
- L'analisi sintattica consiste nell'individuazione delle parti grammaticali componenti e la verifica della loro validità. **Ad esempio:**

*The man hits the dog*

- E' strutturata come:

# Sintassi e semantica

- Lo studio della sintassi è lo studio della grammatica, cioè della struttura delle frasi. **Cosa significa analizzare sintatticamente una frase?**
- L'analisi sintattica consiste nell'individuazione delle parti grammaticali componenti e la verifica della loro validità. **Ad esempio:**

*The man hits the dog*

- E' strutturata come:



Ogni frase in questa forma è sintatticamente valida in inglese.

# Sintassi e semantica

- A partire dall'analisi sintattica precedente, possiamo cominciare a **formalizzare la struttura astratta** delle frasi (corrette)

## The man

< parte nominale >

hits

### < parte verbale >

the dog

< parte nominale >

# Sintassi e semantica

- A partire dall'analisi sintattica precedente, possiamo cominciare a **formalizzare la struttura astratta** delle frasi (corrette)

## < parte nominale > :: ?

The man



< parte nominale >

hits

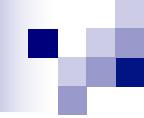


< parte verbale >

the dog



### < parte nominale >



# Sintassi e semantica

- A partire dall'analisi sintattica precedente, possiamo cominciare a **formalizzare la struttura astratta** delle frasi (corrette)

< parte nominale > :: = < articolo > < nome >

The man



< parte nominale >

hits



### < parte verbale >

the dog



### < parte nominale >

# Sintassi e semantica

- A partire dall'analisi sintattica precedente, possiamo cominciare a **formalizzare la struttura astratta** delle frasi (corrette)

< parte nominale > :: = < articolo > < nome >

< articolo > :: = *The* | a

< nome > ::= ***car*** | ***man*** | ***dog***

< parte verbale > :: = *hits* | *eats*

The man



< parte nominale >

hits

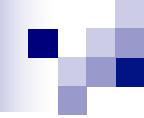


< parte verbale >

the dog



### < parte nominale >



# Sintassi e semantica

- A partire dall'analisi sintattica precedente, possiamo cominciare a **formalizzare la struttura astratta** delle frasi (corrette)

< parte nominale > :: = < articolo > < nome >

< nome > ::= ***car*** | ***man*** | ***dog***

< articolo > :: = *The* | a

< parte verbale > :: = *hits* | *eats*

***Spoiler: questa è una semplicissima grammatica***



# Sintassi e semantica

- A partire dall'analisi sintattica precedente, possiamo cominciare a **formalizzare la struttura astratta** delle frasi (corrette)

< parte nominale > :: = < articolo > < nome >

< nome > ::= ***car*** | ***man*** | ***dog***

< articolo > :: = *The* | a

< parte verbale > :: = ***hits*** | ***eats***

- Queste regole sono scritte in **BNF**, una notazione usata comunemente per descrivere la sintassi dei linguaggi di programmazione.

# Sintassi e semantica

```
< frase semplice > ::= < parte nominale > < parte verbale >
                         < parte nominale >
< parte nominale > ::= < articolo > < nome >
< nome > ::= car | man | dog
< articolo > ::= The | a
< parte verbale > ::= hits | eats
```

- Nell'esempio, *<frase semplice>* è definita come una *<parte nominale>* seguita da una *<parte verbale>* seguita, a sua volta, da un'altra *<parte nominale>*.
- Ciascuna delle due occorrenze di *<parte nominale>* deve essere espansa in *<articolo>* seguito da un *<nome>*.
- Espandendo le parti destre delle regole posso costruire **frasi corrette del linguaggio**

# Sintassi e semantica

```
< frase semplice > ::= < parte nominale > < parte verbale >
                         < parte nominale >
< parte nominale > ::= < articolo > < nome >
< nome > ::= car | man | dog
< articolo > ::= The | a
< parte verbale > ::= hits | eats
```

- Ad esempio posso espandere
  - La parte nominale in <articolo> e <nome>
  - la prima occorrenza di <articolo> in **The**
  - La prima occorrenza di <nome> in **man**
  - La <parte verbale> in **hits**
  - Il secondo <articolo> in **The** ed infine l'ultimo <nome> in **dog**,
  - **Risultato: The man hits the dog**
    - Questo dimostra che la frase è una delle nostre frasi semplici.

# Sintassi e semantica

```
< frase semplice > :: = < parte nominale > < parte verbale >
                           < parte nominale >
< parte nominale > :: = < articolo > < nome >
< nome > :: = car | man | dog
< articolo > :: = The | a
< parte verbale > :: = hits | eats
```

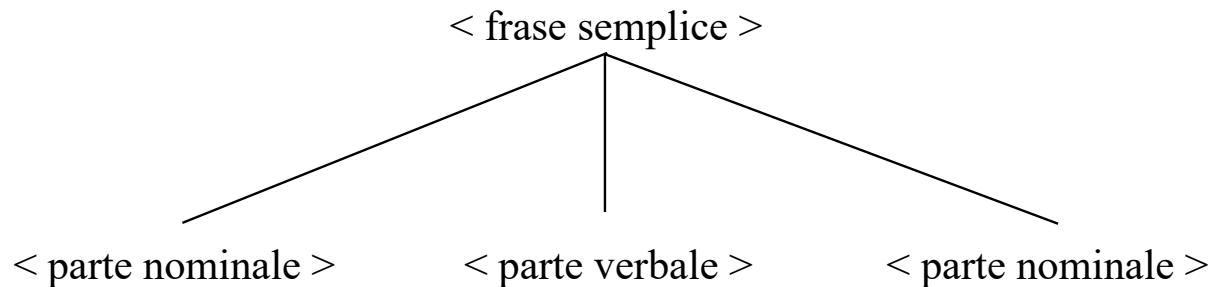
- Tutto ciò si può riassumere nel seguente  
**Albero di derivazione**

< frase semplice >  
      ^

# Sintassi e semantica

```
< frase semplice > :: = < parte nominale > < parte verbale >
                           < parte nominale >
< parte nominale > :: = < articolo > < nome >
< nome > :: = car | man | dog
< articolo > :: = The | a
< parte verbale > :: = hits | eats
```

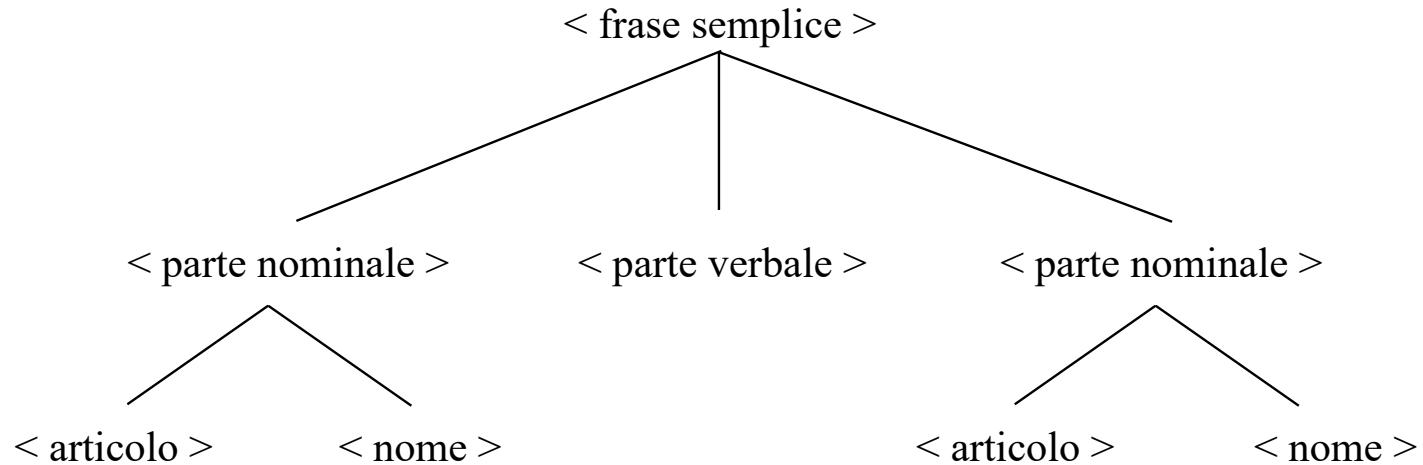
- Tutto ciò si può riassumere nel seguente  
**Albero di derivazione**



# Sintassi e semantica

```
< frase semplice > :: = < parte nominale > < parte verbale >
    < parte nominale >
    < parte nominale > :: = < articolo > < nome >
    < nome > :: = car | man | dog
    < articolo > :: = The | a
    < parte verbale > :: = hits | eats
```

- Tutto ciò si può riassumere nel seguente  
**Albero di derivazione**

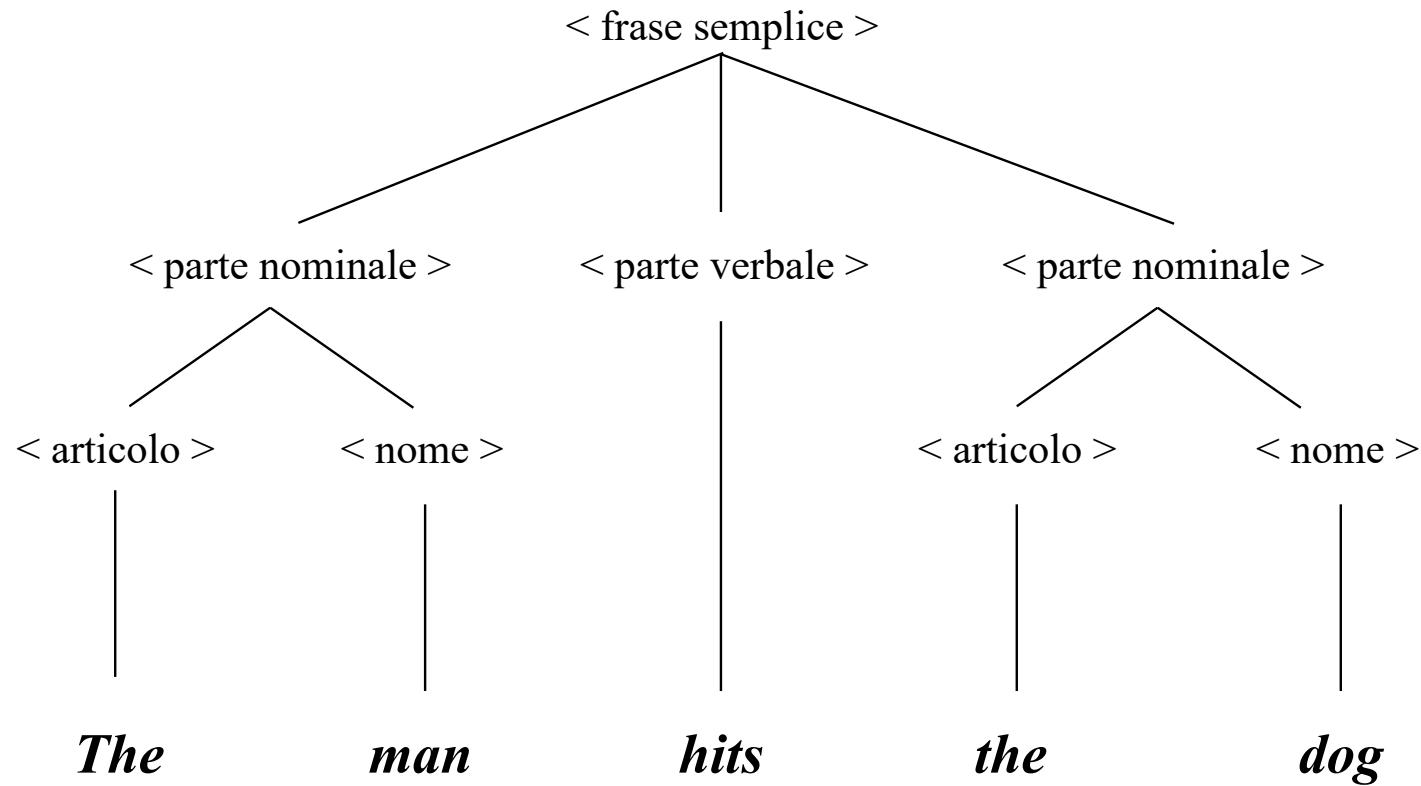


# Sintassi e semantica

```
< frase semplice > ::=< parte nominale > < parte verbale >
                           < parte nominale >
< parte nominale > ::=< articolo > < nome >
< nome > :: = car | man | dog
< articolo > :: = The | a
< parte verbale > :: = hits | eats
```

- Tutto ciò si può riassumere nel seguente

## Albero di derivazione

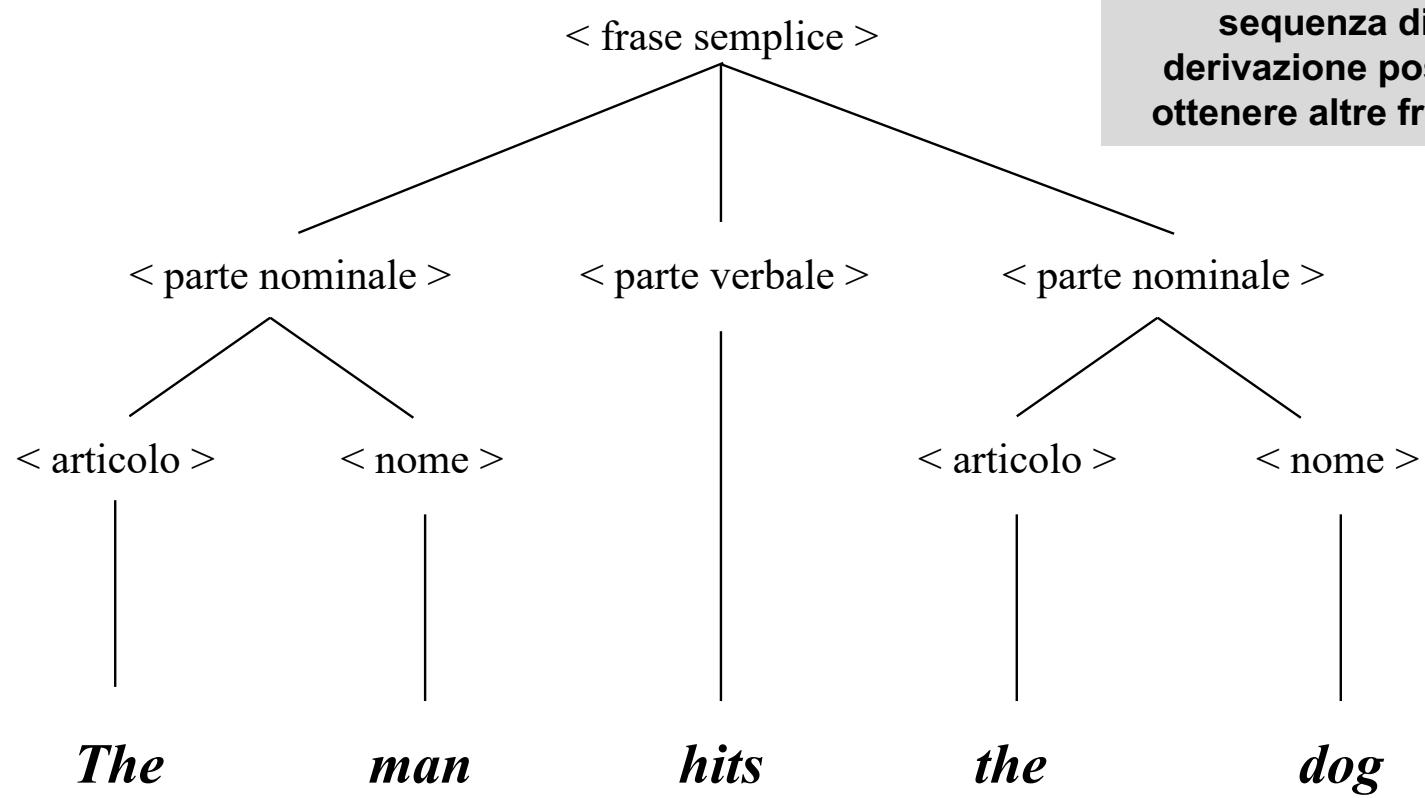


# Sintassi e semantica

```
< frase semplice > ::=< parte nominale > < parte verbale >
                           < parte nominale >
< parte nominale > ::=< articolo > < nome >
< nome > :: = car | man | dog
< articolo > :: = The | a
< parte verbale > :: = hits | eats
```

- Tutto ciò si può riassumere nel seguente

## Albero di derivazione



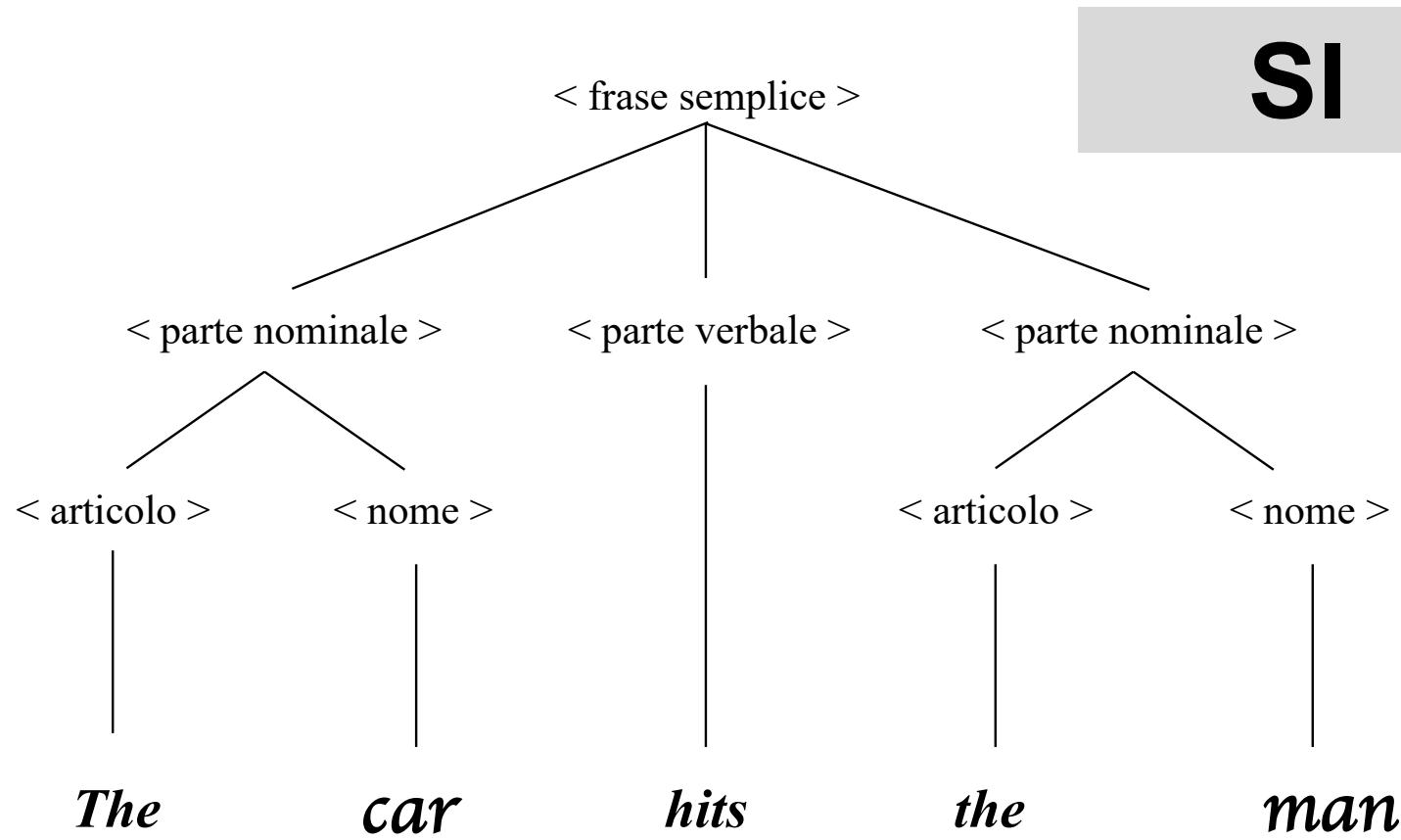
Seguendo la stessa  
sequenza di  
derivazione posso  
ottenere altre frasi?

# Sintassi e semantica

```
< frase semplice > ::=< parte nominale > < parte verbale >
< parte nominale > ::=< articolo > < nome >
< nome > :: = car | man | dog
< articolo > :: = The | a
< parte verbale > :: = hits | eats
```

- Tutto ciò si può riassumere nel seguente

## Albero di derivazione



SI

# Sintassi e semantica

- La definizione data di <frase semplice> dà luogo a **72 diverse frasi derivabili.**
  - Sebbene tutte siano sintatticamente corrette, in base alla nostra definizione, alcune non hanno un'interpretazione semantica sensata (non hanno senso compiuto).
  - Una di queste è la frase: ***The car eats the man***
  - Non necessariamente una frase corretta ha senso.
    - Sintassi diversa da Semantica!

# Sintassi e semantica

- Consideriamo la seguente frase (non derivabile da <frase semplice> utilizzando le regole date in precedenza):

***They are flying planes***

- Quale è l'analisi sintattica di questa frase?



# Sintassi e semantica

- Consideriamo la seguente frase (non derivabile da <frase semplice> utilizzando le regole date in precedenza):

***They are flying planes***

- Un'analisi sintattica di questa frase è:



Quest'analisi suggerisce che **they** si riferisce a **planes** (*aerei nel cielo che volano*). **E' l'unica possibile?**

# Sintassi e semantica

- Un'analisi alternativa sarebbe:

They  
  
<pronomo>

are flying  
  
<verbo>

planes  
  
<nome>

e questa implicherebbe un'interpretazione semantica completamente differente, in cui **they** si riferisce a chiunque sia attualmente al controllo dei **planes** (*essi stanno pilotando gli aerei*).

# Sintassi e semantica

- Nell'esempio, l'analisi sintattica è di aiuto all'interpretazione semantica.
- Per quanto sia scelto "ad hoc" e mostri un fenomeno poco comune nel linguaggio naturale, l'esempio è illustrativo di un concetto importante in computazione.
- **Questo processo avviene anche nei linguaggi di programmazione**
  - Una fase cruciale nel processo di compilazione di un programma è l'**analisi sintattica**, come passo essenziale per la sua **interpretazione semantica**.

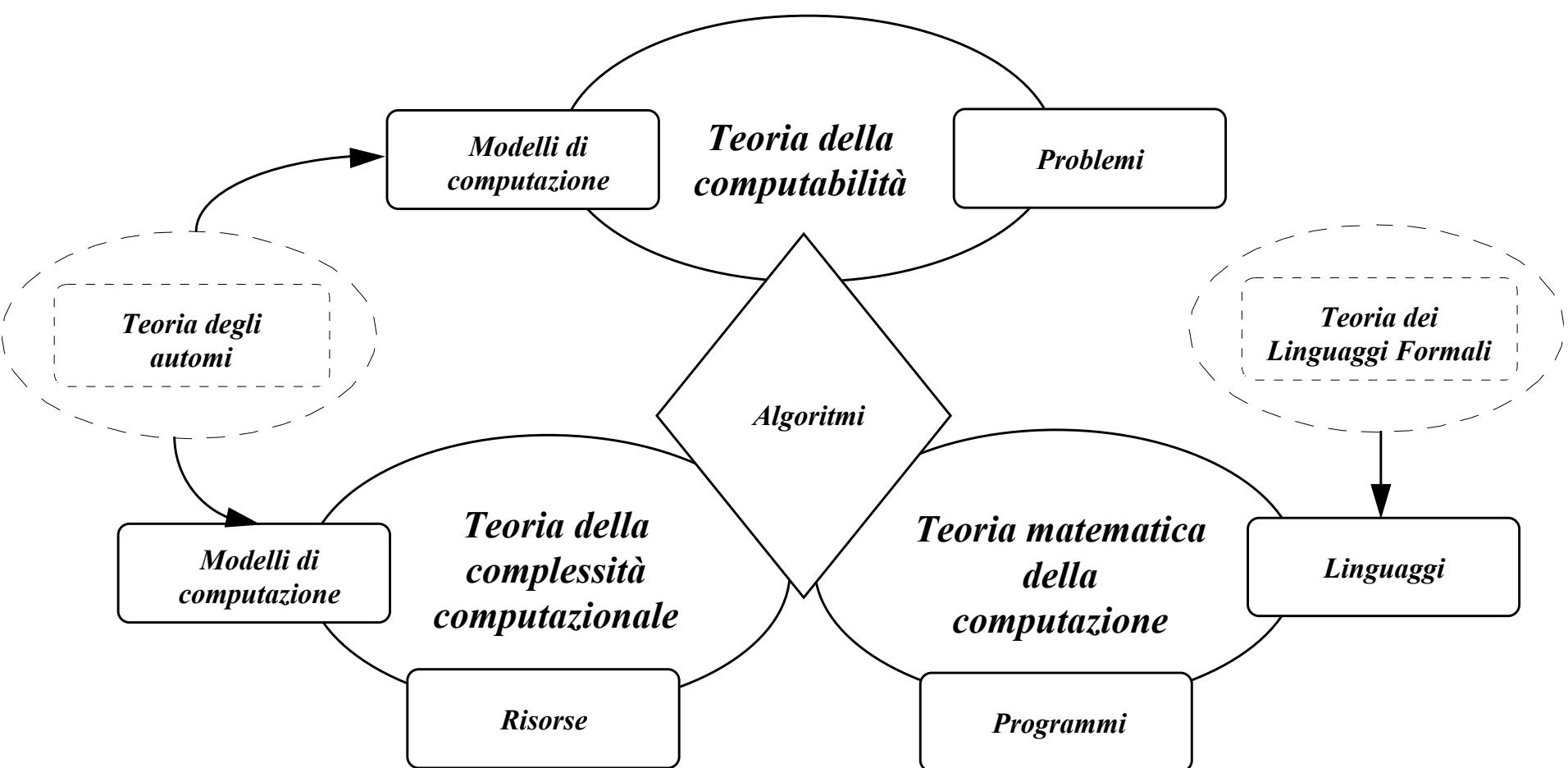
# Perché tutto questo ci interessa?

- Perché il linguaggio naturale e i linguaggi di programmazione **non sono così diversi tra loro**
- Analogie e differenze tra i linguaggi, vengono studiate da una teoria che prende il nome di **«Teoria dei Linguaggi Formali»**, fondamentale per tutta la ricerca in Informatica Teorica

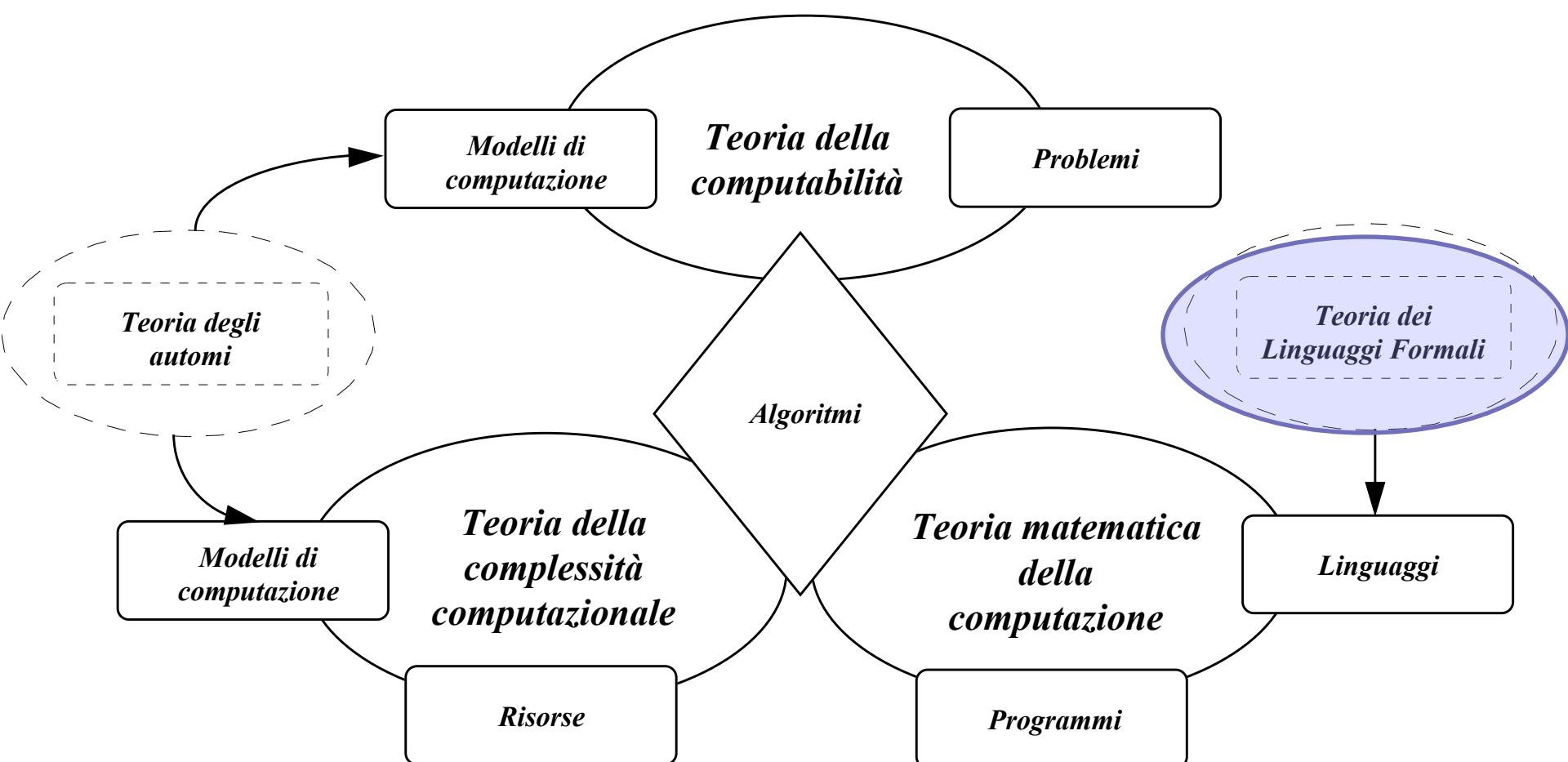
# Perché tutto questo ci interessa?

- Perché il linguaggio naturale e i linguaggi di programmazione **non sono così diversi tra loro**
- Analogie e differenze tra i linguaggi, vengono studiate da una teoria che prende il nome di **«Teoria dei Linguaggi Formali»**, fondamentale per tutta la ricerca in Informatica Teorica
  - Informatica Teorica?**

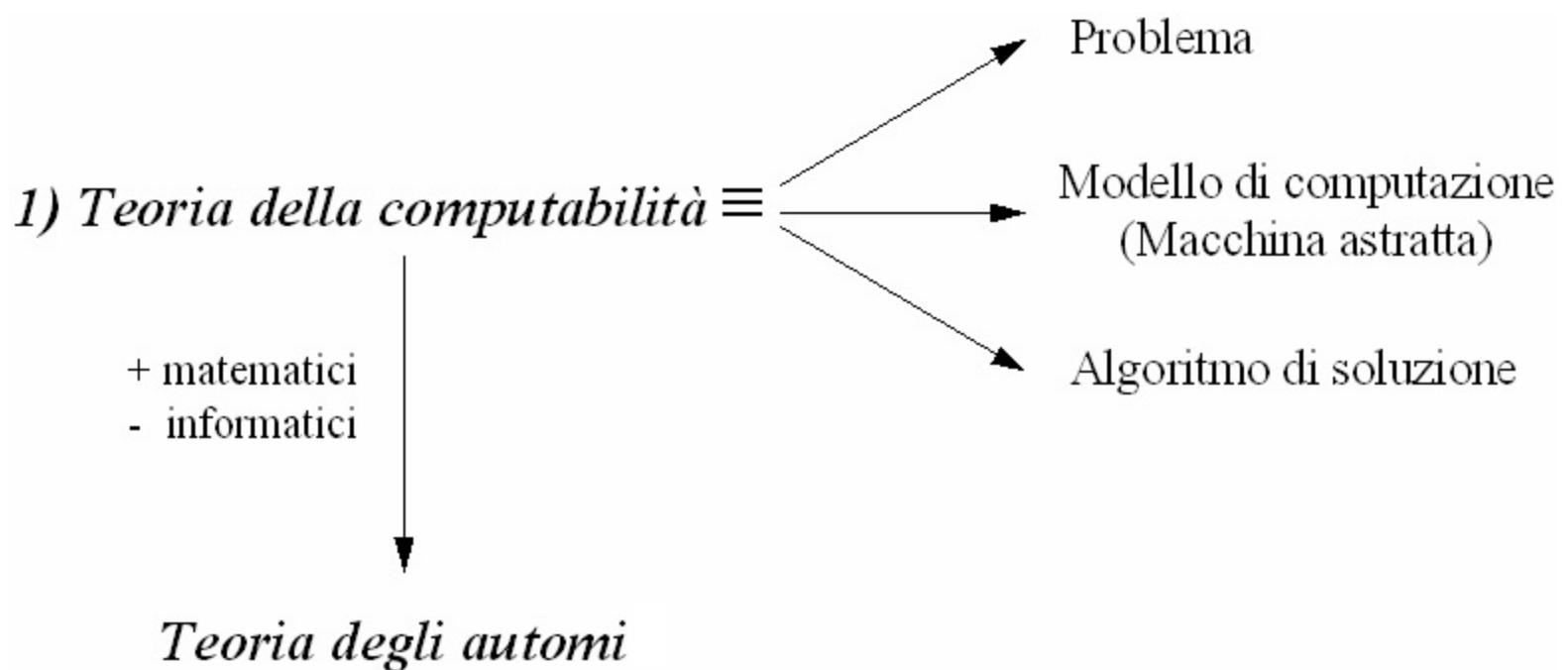
# Aree di ricerca dell'informatica teorica



# Aree di ricerca dell'informatica teorica

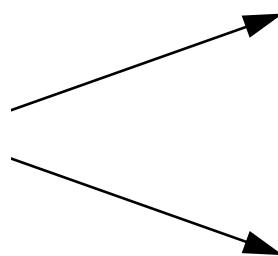


# Aree di ricerca dell'informatica teorica



# Aree di ricerca dell'informatica teorica

2) *Teoria matematica  
della computazione* ≡

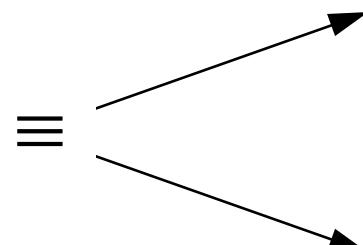


Rapporto tra gli algoritmi e i linguaggi di programmazione  
(in cui possono essere descritti)

Proprietà dei programmi  
(correttezza, terminazione, ...)

# Aree di ricerca dell'informatica teorica

## 3) *Teoria della complessità computazionale*



Rapporto tra gli algoritmi e le risorse necessarie per eseguirli

Aspetti quantitativi del procedimento di soluzione di un problema

# Introduzione alla teoria dei linguaggi formali

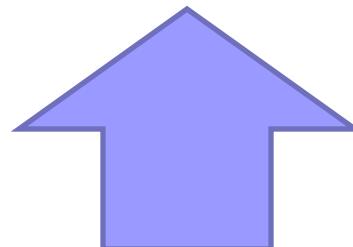
- La **teoria dei linguaggi formali** rappresenta spesso uno scoglio per gli studenti di informatica a causa della sua **forte dipendenza dalla notazione**.
  - **Preparatevi a essere molto «formali» nello svolgimento e nella risoluzione degli esercizi**

# Introduzione alla teoria dei linguaggi formali

- La **teoria dei linguaggi formali** rappresenta spesso uno scoglio per gli studenti di informatica a causa della sua **forte dipendenza dalla notazione**.
  - **Preparatevi a essere molto «formali» nello svolgimento e nella risoluzione degli esercizi**
- L'argomento, d'altra parte, non può essere evitato perché ogni laureato in informatica deve possedere una **buona comprensione dell'operazione di compilazione** e questa, a sua volta, si fonda pesantemente sulla **teoria dei linguaggi formali**.

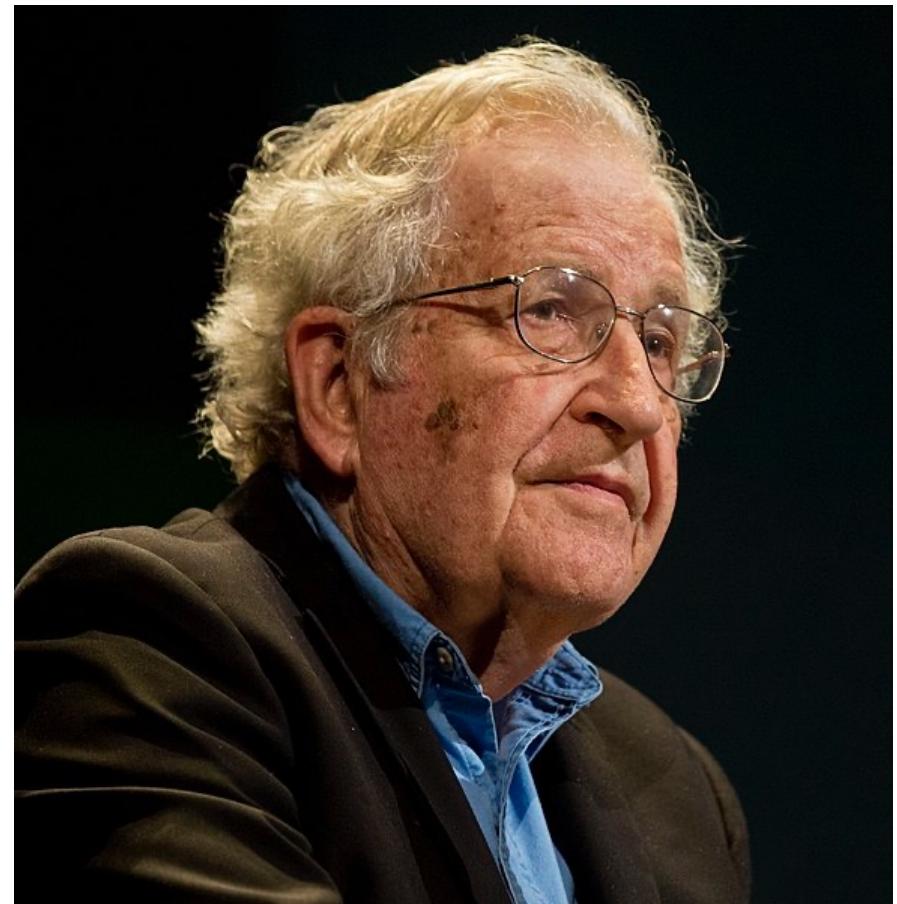
# Introduzione alla teoria dei linguaggi formali

- La **teoria dei linguaggi formali** rappresenta spesso uno scoglio per gli studenti di informatica a causa della sua **forte dipendenza dalla notazione**.
  - **Preparatevi a essere molto «formali» nello svolgimento e nella risoluzione degli esercizi**
- L'argomento, d'altra parte, non può essere evitato perché ogni laureato in informatica deve possedere una **buona comprensione dell'operazione di compilazione** e questa, a sua volta, si fonda pesantemente sulla **teoria dei linguaggi formali**.



# Introduzione alla teoria dei linguaggi formali

- La teoria nasce grazie al lavoro di N. Chomsky, linguista americano, che negli anni '50 cerca di descrivere **la sintassi del linguaggio naturale** secondo semplici **regole di riscrittura e trasformazione**.
- Chomsky classifica i linguaggi in base alle **restrizioni** imposte alle regole che generano tali linguaggi.



# Teoria dei linguaggi formali

- Una classe importante di regole che generano linguaggi formali va sotto il nome di **grammatiche libere da contesto** (prive di contesto) o **Context-free grammars (C.F.)**.

# Teoria dei linguaggi formali

- Una classe importante di regole che generano linguaggi formali va sotto il nome di **grammatiche libere da contesto** (prive di contesto) o **Context-free grammars (C.F.)**.
- L'importanza di tale classe (e dei linguaggi da essa generati) risiede nel fatto che lo sviluppo dei primi linguaggi di programmazione di alto livello (ALGOL 60), che segue di pochi anni il lavoro di Chomsky, **dimostra che le grammatiche C.F. sono strumenti adeguati a descrivere la sintassi di base di molti linguaggi di programmazione.**
- Questa prima parte del programma ha l'obiettivo di fornire **concetti di teoria dei linguaggi formali**, necessari comprensione delle **tecniche di compilazione**, oggetto della parte finale del corso.

# Introduzione alla teoria dei linguaggi formali

- Livelli di descrizione di un linguaggio
  - **Come possiamo descrivere un linguaggio?**

# Introduzione alla teoria dei linguaggi formali

## ■ Livelli di descrizione di un linguaggio

- **grammatica**

- 

- **semantica**

- 

- **pragmatica**

- 

- **implementazione (per i linguaggi di programmazione)**

-

# Introduzione alla teoria dei linguaggi formali

## ■ Livelli di descrizione di un linguaggio

### **grammatica**

- quali frasi sono corrette?

### **semantica**

- Quale è il significato di una frase corretta?

### **pragmatica**

- come usare una frase corretta e sensata?

### **implementazione (per i linguaggi di programmazione)**

- come **eseguire** una frase corretta in modo da rispettarne il significato?

# Concetto intuitivo di grammatica

- In questo corso, ci focalizzeremo soprattutto **sul livello grammaticale**
- **Come si definisce un grammatica?**
  - **complesso di regole** necessarie alla costruzione di frasi, sintagmi e parole di una determinata lingua
  - **Da quali elementi è costituita?**

# Concetto intuitivo di grammatica

- In questo corso, ci focalizzeremo soprattutto **sul livello grammaticale**
- **Come si definisce un grammatica?**
  - Alfabeto del linguaggio
    - simboli con cui “costruire” le parole del linguaggio
    - es.: alfabeto latino su 22 o 26 lettere
  - Lessico
    - parole del linguaggio
    - es.: informatica
  - Sintassi
    - determina quali sequenze di parole costituiscono frasi legali (corrette)

Le grammatiche sono uno strumento utile poiché la sintassi di un linguaggio di programmazione, come dimostrato da Chomsky, si può descrivere **attraverso una grammatica**.

# Introduzione alla teoria dei linguaggi formali

- Tra i vari gruppi della classificazione di Chomsky, ci concentreremo quasi esclusivamente su due tipi di grammatiche:
  - **regolari**
  - **libere da contesto**

poiché i linguaggi formali utilizzati in informatica sono per lo più di questi tipi.

# Introduzione alla teoria dei linguaggi formali

- Tra i vari gruppi della classificazione di Chomsky, ci concentreremo quasi esclusivamente su due tipi di grammatiche:
  - **regolari**
  - **libere da contesto**poiché i linguaggi formali utilizzati in informatica sono per lo più di questi tipi.
- **Guardiamo un primo esempio di linguaggio formale!**

# Esempio di linguaggio C.F.

- Un linguaggio C.F. è il linguaggio delle parentesi ben formate che comprende tutte le stringhe di parentesi aperte e chiuse bilanciate correttamente.

Esempio:

( ) è ben formata;

( ( ) ( ) ) è ben formata;

( ( ) ( ) non è ben formata.

- Questo linguaggio è fondamentale in informatica come notazione per contrassegnare il “raggio d’azione” nelle espressioni matematiche e nei linguaggi di programmazione

# Linguaggio delle parentesi ben formate

## ■ Definizione

- Quando una coppia di parentesi è «ben formata»?
- Partiamo da alcuni esempi e proviamo a ricavare delle regole generali
- Esempi di parentesi ben formate:
  - ()
  - (( ))
  - ((( )))
  - ()()
  - ()()(( ))
- **Quali caratteristiche hanno le parentesi ben formate?**

# Linguaggio delle parentesi ben formate

## ■ Definizione

- Quando una coppia di parentesi è «ben formata»?
- Partiamo da alcuni esempi e proviamo a ricavare delle regole generali
- Esempi di parentesi ben formate:

■ ( )	←	parentesi semplice
■ ( ( ) )	←	ricorsione
■ ( ( ( ) ) )	←	ricorsione
■ ( ) ( )	←	concatenazione
■ ( ) ( ) ( ( ) )	←	ricorsione + concatenazione

- **Quali caratteristiche hanno le parentesi ben formate?**

# Linguaggio delle parentesi ben formate

## ■ Definizione

- Quando una coppia di parentesi è «ben formata»?
- Partiamo da alcuni esempi e proviamo a ricavare delle regole generali
- Esempi di parentesi ben formate:

■ ( )	←	parentesi semplice
■ ( ( ) )	←	ricorsione
■ ( ( ( ) ) )	←	ricorsione
■ ( ) ( )	←	concatenazione
■ ( ) ( ) ( ( ) )	←	ricorsione + concatenazione

- **Proponiamo delle regole!**

# Linguaggio delle parentesi ben formate

## ■ Esempi di parentesi ben formate:

- ( ) parentesi semplice
- ( ( ) ) ricorsione
- ( ( ( ) ) ) ricorsione
- ( ) ( ) concatenazione
- ( ) ( ) ( ) ricorsione + concatenazione

## ■ Definizione

1. La stringa ( ) è ben formata;
2. se la stringa di simboli  $A$  è ben formata, allora lo è anche  $(A)$ ;
3. se le stringhe  $A$  e  $B$  sono ben formate, allora lo è anche la stringa  $AB$ .

# Linguaggio delle parentesi ben formate

- **Definizione**
  1. La stringa  $( )$  è ben formata;
  2. se la stringa di simboli  $A$  è ben formata, allora lo è anche  $(A)$ ;
  3. se le stringhe  $A$  e  $B$  sono ben formate, allora lo è anche la stringa  $AB$ .
- A partire da questa definizione, possiamo generare **un sistema di regole di riscrittura attraverso cui costruire** l'insieme delle stringhe lecite di parentesi ben formate:

# Linguaggio delle parentesi ben formate

- In corrispondenza di questa definizione induttiva, possiamo considerare **un sistema di riscrittura che genera esattamente l'insieme delle stringhe lecite di parentesi ben formate**:

$$(1) \quad S \rightarrow ()$$

$$(2) \quad S \rightarrow (S)$$

$$(3) \quad S \rightarrow SS$$

- Si legge «**S produce ( )»**

# Notazione

- Nelle regole di produzione si è fatto uso di due tipi di simboli: caratteri che possono apparire nelle derivazioni, ma non nelle stringhe finali, detti *simboli nonterminali* o *variabili* (nell'esempio,  $S$  è il solo nonterminale) e caratteri che possono apparire nelle stringhe finali, detti *simboli terminali* (nell'esempio,  $e$  sono i simboli terminali).
- In BNF si usa la seguente notazione:

$S$	$\langle S \rangle$
$\rightarrow$	$::=$

# Linguaggio delle parentesi ben formate

- In corrispondenza di questa definizione induttiva, possiamo considerare **un sistema di riscrittura che genera esattamente l'insieme delle stringhe lecite** di parentesi ben formate:
  - (1)  $S \rightarrow ()$
  - (2)  $S \rightarrow (S)$
  - (3)  $S \rightarrow SS$
- Le regole di riscrittura (1), (2) e (3) sono dette *produzioni* o *regole di produzione*.
  - Stabiliscono che “data una stringa, si può formare una nuova stringa sostituendo una  $S$  o con  $()$  o con  $(S)$  o con  $SS$ ”.

# Linguaggio delle parentesi ben formate

- Grammatica delle parentesi ben formate:
  - (1)  $S \rightarrow ()$
  - (2)  $S \rightarrow (S)$
  - (3)  $S \rightarrow SS$
- Confrontando la definizione di parentesi ben formate e le tre produzioni, si osserva una corrispondenza diretta tra le parti i) e ii) della definizione e le produzioni (1) e (2). La produzione (3) è apparentemente diversa dalla parte (iii) della definizione.

# Linguaggio delle parentesi ben formate

- Grammatica delle parentesi ben formate:
  - (1)  $S \rightarrow ()$
  - (2)  $S \rightarrow (S)$
  - (3)  $S \rightarrow SS$
- Abbiamo utilizzato simboli distinti A e B nella definizione induttiva per evidenziare il fatto che le due stringhe di parentesi ben formate che consideriamo non sono necessariamente uguali. Nella produzione corrispondente non c'è necessità di usare simboli distinti perché in  $S \rightarrow SS$  le due S che compaiono a destra possono essere sostituite indipendentemente.
- Possiamo cioè applicare una produzione ad una delle due S (alla prima o alla seconda) indifferentemente. Il risultato non cambia.
- **Proviamo a generare delle stringhe valide a partire da quelle regole**

# Linguaggio delle parentesi ben formate

- Esempio: Generazione di  $((\ ))(\ )( \ ))$

$$(1) \ S \rightarrow (\ )$$

$$(2) \ S \rightarrow (S)$$

$$(3) \ S \rightarrow SS$$

# Linguaggio delle parentesi ben formate

- Esempio: Generazione di  $((\ ))(\ )( \ ))$ 
  - La corrispondente sequenza di applicazione delle produzioni è:  
 $(3) \rightarrow$

$$S \xrightarrow{(3)} SS$$

- (1)  $S \rightarrow()$
- (2)  $S \rightarrow(S)$
- (3)  $S \rightarrow SS$

# Linguaggio delle parentesi ben formate

- Esempio: Generazione di  $((\ ))(\ )( \ ))$ 
  - La corrispondente sequenza di applicazione delle produzioni è:  
 $(3) \rightarrow (2)$

$$S \xrightarrow{(3)} SS \xrightarrow{(2)} (S)S$$

- (1)  $S \rightarrow (\ )$
- (2)  $S \rightarrow (S)$
- (3)  $S \rightarrow SS$

# Linguaggio delle parentesi ben formate

- Esempio: Generazione di  $((\ ))(\ )( \ ))$ 
  - La corrispondente sequenza di applicazione delle produzioni è:  
 $(3) \rightarrow (2) \rightarrow (1)$

$$S \xrightarrow{(3)} SS \xrightarrow{(2)} (S)S \xrightarrow{(1)} ((\ ))$$

- (1)  $S \rightarrow (\ )$
- (2)  $S \rightarrow (S)$
- (3)  $S \rightarrow SS$

# Linguaggio delle parentesi ben formate

- Esempio: Generazione di  $((\ ))(\ )( \ ))$ 
  - La corrispondente sequenza di applicazione delle produzioni è:  
 $(3) \rightarrow (2) \rightarrow (1) \rightarrow (2)$

$$S \xrightarrow{(3)} SS \xrightarrow{(2)} (S)S \xrightarrow{(1)} ((\ ))S \xrightarrow{(2)} ((\ ))(S)$$

(1)  $S \rightarrow (\ )$

(2)  $S \rightarrow (S)$

(3)  $S \rightarrow SS$

# Linguaggio delle parentesi ben formate

- Esempio: Generazione di  $((\ ))(\ )( \ ))$ 
  - La corrispondente sequenza di applicazione delle produzioni è:  
 $(3) \rightarrow (2) \rightarrow (1) \rightarrow (2) \rightarrow (3) \rightarrow (1) \rightarrow (1)$

$$S \xrightarrow{(3)} SS \xrightarrow{(2)} (S)S \xrightarrow{(1)} ((\ ))S \xrightarrow{(2)} ((\ ))(S) \xrightarrow{(3)} ((\ ))(SS) \xrightarrow{(1)} ((\ ))((\ ))S \xrightarrow{(1)} ((\ ))((\ ))((\ ))$$

(1)  $S \rightarrow (\ )$

(2)  $S \rightarrow (S)$

(3)  $S \rightarrow SS$

# Linguaggio delle parentesi ben formate

- Esempio: Generazione di  $((\ ))(\ )( \ ))$ 
  - La corrispondente sequenza di applicazione delle produzioni è:
$$(3) \rightarrow (2) \rightarrow (1) \rightarrow (2) \rightarrow (3) \rightarrow (1) \rightarrow (1)$$
- Una sequenza di applicazioni di regole di produzione prende il nome di **derivazione**.

$$S \xrightarrow{(3)} SS \xrightarrow{(2)} (S)S \xrightarrow{(1)} ((\ ))S \xrightarrow{(2)} ((\ ))(S) \xrightarrow{(3)} ((\ ))(SS) \xrightarrow{(1)} ((\ ))((\ ))S \xrightarrow{(1)} ((\ ))((\ ))(\ ) \xrightarrow{(1)} ((\ ))((\ ))( \ ))$$

(1)  $S \rightarrow (\ )$

(2)  $S \rightarrow (S)$

(3)  $S \rightarrow SS$

# Linguaggio delle parentesi ben formate

- Esempio: Generazione di  $((\ ))( (\ )( \ ))$ 
  - Secondo modo:

$$S \xrightarrow{(3)} SS \xrightarrow{(2)} S(S) \xrightarrow{(3)} S(SS) \xrightarrow{(1)} S((\ )S) \xrightarrow{(1)} S((\ )( )) \xrightarrow{(2)} (S)((\ )( )) \xrightarrow{(1)} ((\ ))( (\ )( ))$$

- La corrispondente sequenza di applicazione delle produzioni è:

$$(3) \rightarrow (2) \rightarrow (3) \rightarrow (1) \rightarrow (1) \rightarrow (2) \rightarrow (1)$$

# Notazione

- Nei due esempi precedenti abbiamo fatto uso della notazione  $\alpha \Rightarrow \beta$ 
  - L'interpretazione è la seguente: "da  $\alpha$  si produce direttamente  $\beta$  per effetto dell'applicazione della regola di riscrittura  $n$ ". Ad esempio
$$SS \xrightarrow{(n)} (S)S$$
  - si legge: "SS produce direttamente  $(S)S$  per effetto dell'applicazione della regola di riscrittura (2)".

# Notazione

- Nei due esempi precedenti abbiamo fatto uso della notazione  $\alpha \Rightarrow \beta$ 
  - L'interpretazione è la seguente: "da  $\alpha$  si produce direttamente  $\beta$  per effetto dell'applicazione della regola di riscrittura  $n$ ". Ad esempio
  - si legge: "SS produce direttamente  $(S)S$  per effetto dell'applicazione della regola di riscrittura (2)".
- Le derivazioni precedenti (modi 1 e 2) vengono riassunte attraverso la notazione:

$$S \xrightarrow{*} (( ))(( )( ))$$

che leggiamo come "S produce  $(( ))(( )( ))$ "

# Riferimenti

- Semeraro, G., Elementi di Teoria dei Linguaggi Formali, [ilmolibro.it](http://ilmiolibro.kataweb.it/libro/informatica-e-internet/317883/elementi-di-teoria-dei-linguaggi-formali/), 2017  
[- Capitolo 1](http://ilmolibro.kataweb.it/libro/informatica-e-internet/317883/elementi-di-teoria-dei-linguaggi-formali/)

# Domande?



**Linguaggi di Programmazione**

docente: Cataldo Musto

[cataldo.musto@uniba.it](mailto:cataldo.musto@uniba.it)