



Linguaggi di Programmazione

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

Si ringrazia il Dott. Marco de Gemmis per la concessione di parte 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
- Questa competenza non la si acquisisce soltanto imparando *ex novo* molti linguaggi diversi
 - 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

■ 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

Macchine Astratte e Implementazione dei linguaggi di programmazione

- Qual è il significato dell'espressione:
“implementazione di un linguaggio di programmazione”?
- Si tratta di un concetto strettamente correlato a quello di:

MACCHINA ASTRATTA

Macchine Astratte e Implementazione dei linguaggi di programmazione

- Calcolatore = macchina fisica
 - consente di eseguire algoritmi opportunamente formalizzati perché siano “comprensibili” all’esecutore
 - la formalizzazione consiste nella codifica degli algoritmi in un certo linguaggio \mathcal{L} definito da una specifica sintassi
 - la sintassi di \mathcal{L} permette di utilizzare determinati costrutti per comporre programmi in \mathcal{L}
 - Un programma in \mathcal{L} è una sequenza di istruzioni del linguaggio \mathcal{L}
- Macchina astratta = astrazione del concetto di calcolatore fisico

Definizioni di macchina astratta e linguaggio macchina

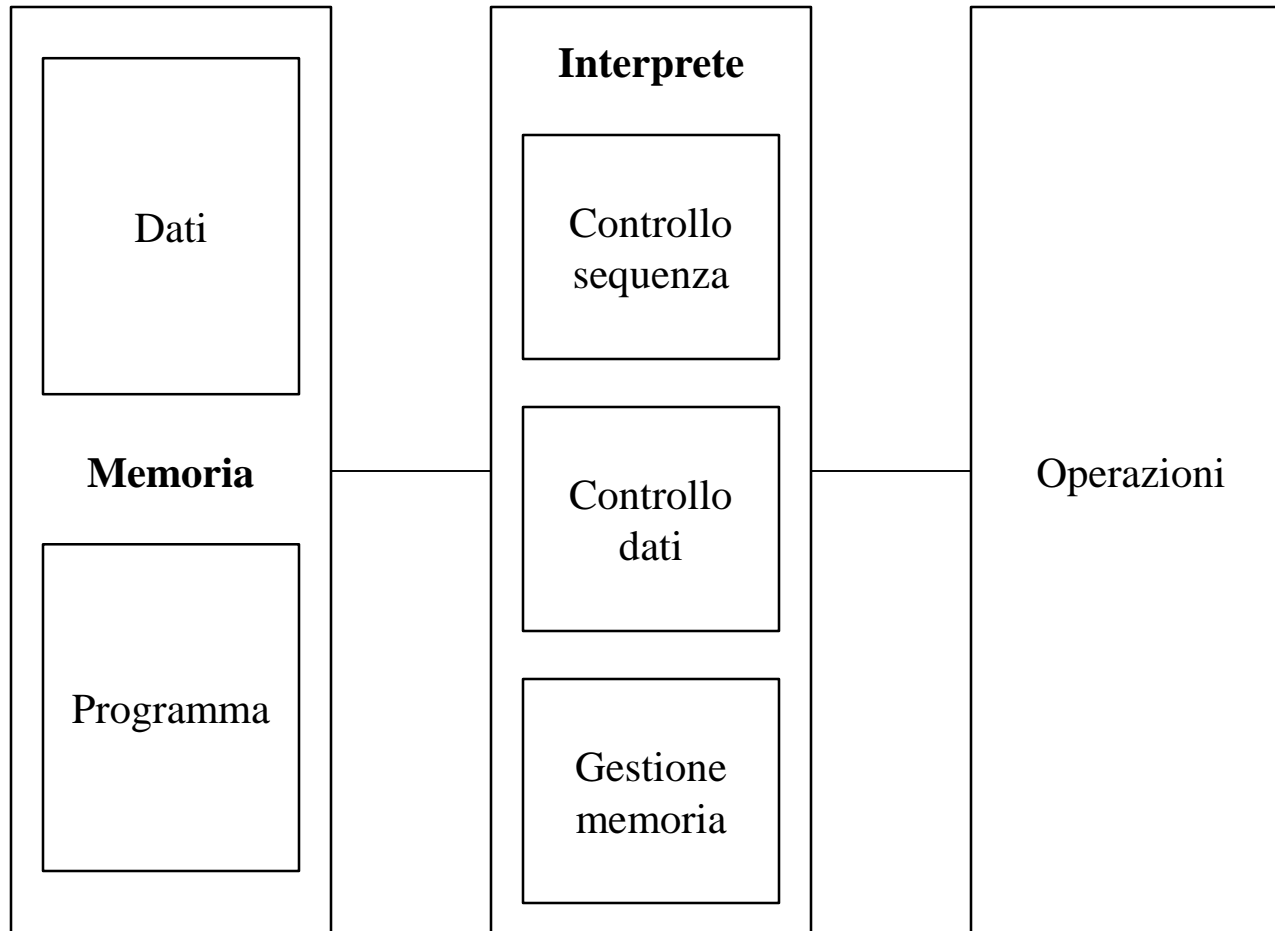
■ Macchina astratta per \mathcal{L}

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

■ 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 $\mathcal{M}_{\mathcal{L}}$

Macchina Astratta



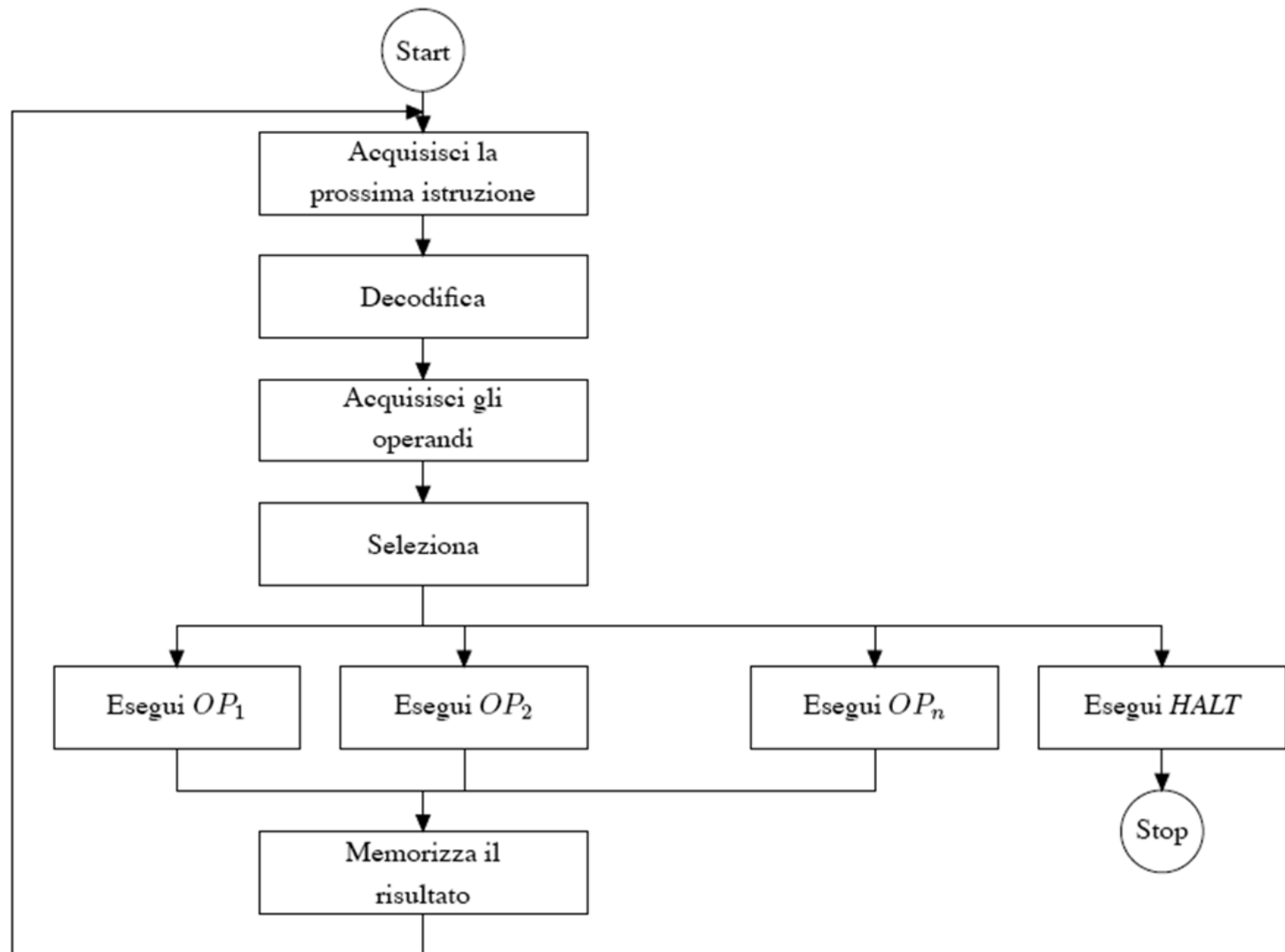
Le operazioni dell'Interprete

- Operazioni per l'elaborazione dei dati primitivi
 - numeri interi, reali
 - operazioni aritmetiche
- 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



Realizzazione di una macchina astratta

- Per essere effettivamente utilizzata, $\mathcal{M}_{\mathcal{L}}$ dovrà prima o poi utilizzare qualche dispositivo fisico
 - realizzazione “fisica” in hardware
 - algoritmi di $\mathcal{M}_{\mathcal{L}}$ realizzati direttamente mediante dispositivi fisici
- Possiamo pensare a realizzazioni che usino livelli intermedi tra $\mathcal{M}_{\mathcal{L}}$ ed il dispositivo fisico
 - **simulazione mediante software**
 - emulazione mediante firmware
 - microprogrammi in linguaggi di basso livello

Realizzazione mediante software

- Algoritmi e strutture dati di $\mathcal{M}_{\mathcal{L}}$ realizzati in un altro linguaggio \mathcal{L}' già implementato
- $\mathcal{M}_{\mathcal{L}}$ è realizzata mediante programmi in \mathcal{L}' che simulano le funzionalità di $\mathcal{M}_{\mathcal{L}}$
- $\mathcal{M}_{\mathcal{L}}$ è realizzata attraverso la macchina $\mathcal{M}'_{\mathcal{L}'}$
- $\mathcal{M}'_{\mathcal{L}'}$ è detta *macchina ospite* e si denota con $\mathcal{M}o_{\mathcal{L}o}$
- Intuitivamente l'implementazione di \mathcal{L} sulla macchina ospite avviene mediante una qualche “traduzione” di \mathcal{L} in $\mathcal{L}o$
- A seconda di come avvenga la traduzione di parla di:
 - implementazione interpretativa
 - implementazione compilativa

Funzioni Parziali

- Una funzione *parziale*

$$f: A \rightarrow B$$

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

- Può essere NON DEFINITA per qualche elemento di A
- Dato $a \in A$, **se** esiste un corrispondente in B, esso è denotato con $f(a)$

I programmi definiscono funzioni parziali

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

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

Definizione di interprete

In generale un programma scritto in \mathcal{L} si può vedere come una funzione parziale:

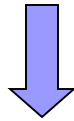
$$\mathcal{P}^{\mathcal{L}} : \mathcal{D} \rightarrow \mathcal{D} \quad \text{t.c.} \quad \mathcal{P}^{\mathcal{L}}(\text{Input}) = \text{Output}$$

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

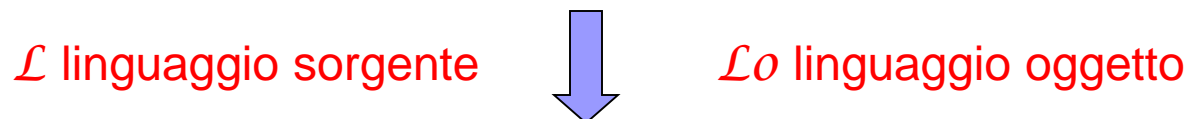
Definizione di compilatore

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

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

$$C_{\mathcal{L},\mathcal{L}o}(\mathcal{P}^{\mathcal{L}}) = \mathcal{P}^{\mathcal{L}o} \qquad \mathcal{P}^{\mathcal{L}}(Input) = \mathcal{P}^{\mathcal{L}o}(Input)$$

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



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

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

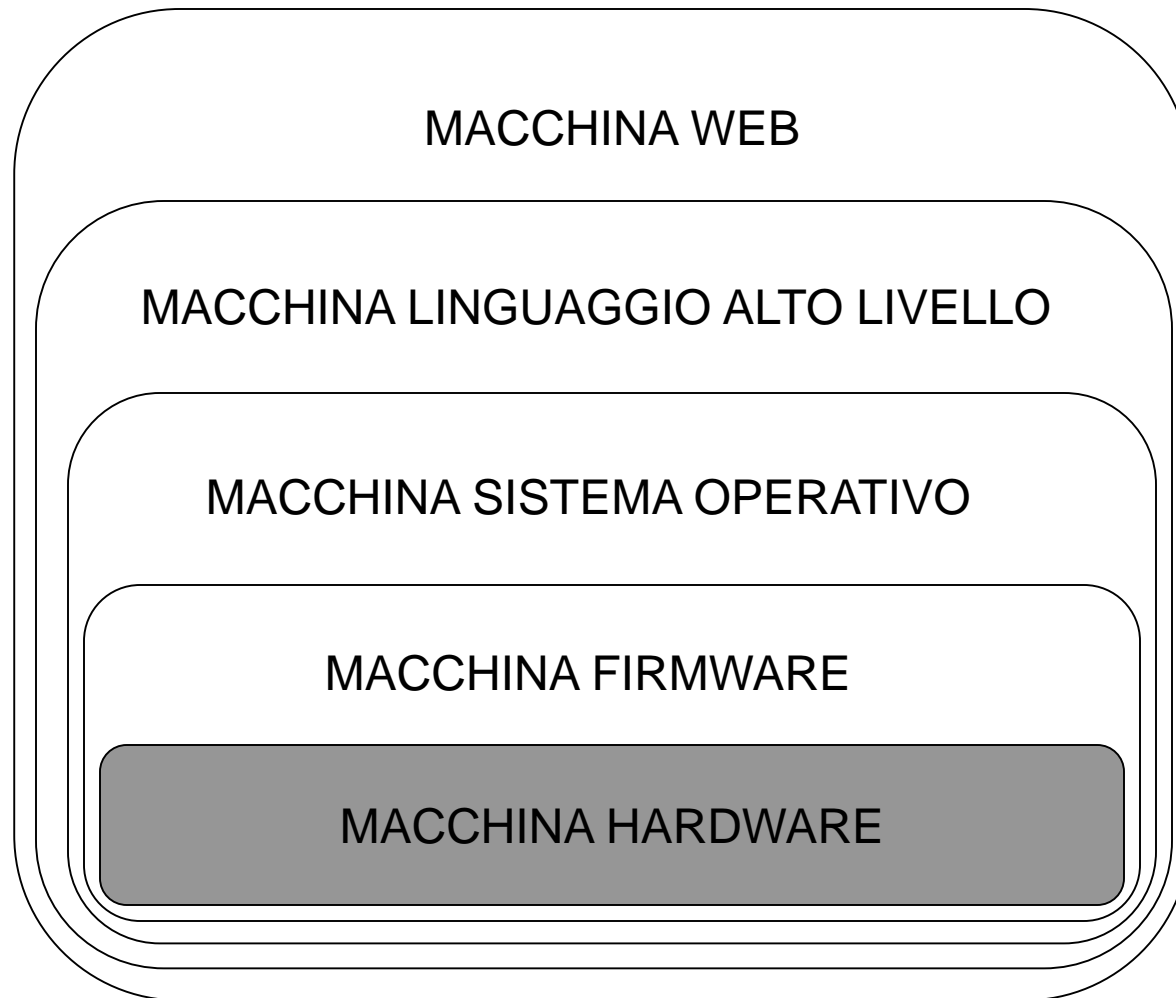
- perdita di informazioni rispetto al programma sorgente (difficile tracciare errori e dunque maggiori difficoltà nel debugging)

In realtà i due approcci sono spesso combinati

Gerarchie di Macchine Astratte

- 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

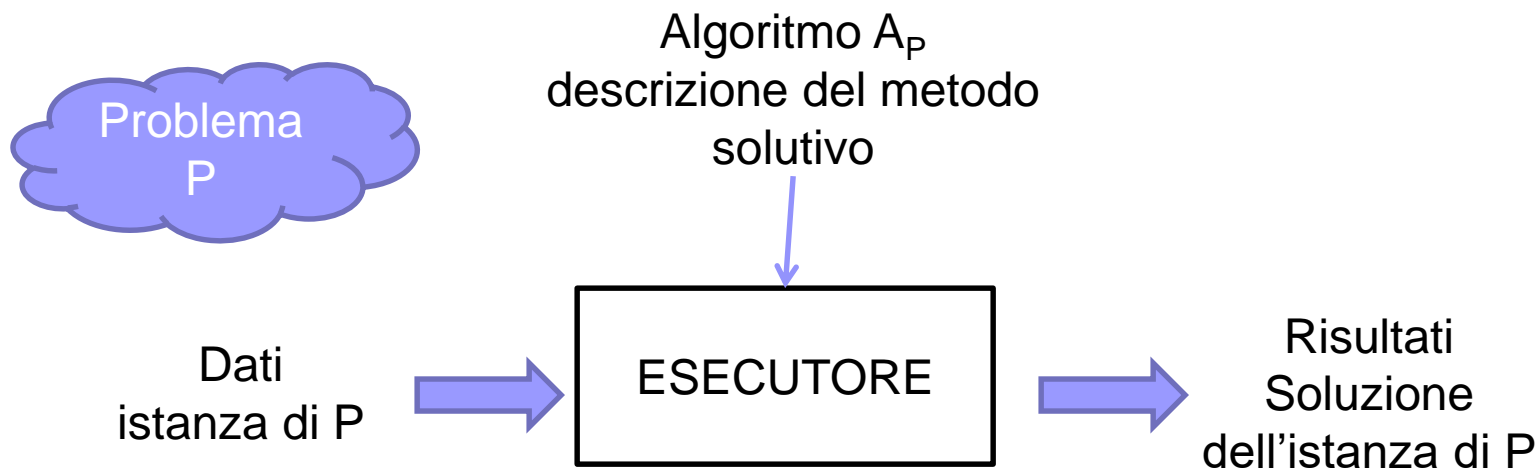
Una gerarchia di macchine astratte



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

IN GENERALE



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

Programmi e funzioni calcolabili

- Esistono dei problemi per i quali non esiste 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
- Esistono dei problemi per i quali non esiste un programma che li risolva? \rightarrow 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:

```
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

- 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

$K(P)$

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

$K(P)$

Stampa "LOOP" se $P(P)$ NON termina

va in loop se $P(P)$ termina

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

Il problema della fermata

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

$K(K)$ { Stampa "LOOP" se $K(K)$ NON termina
va in loop se $K(K)$ termina

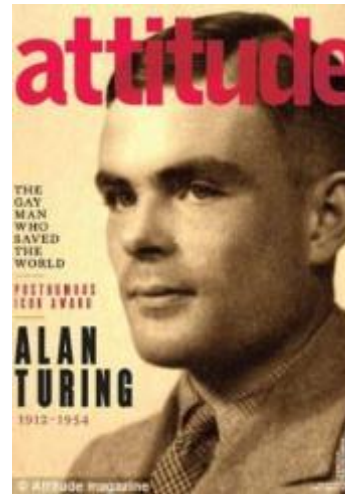
- Assurdo!
 - $K(K)$ termina con una stampa quando $K(K)$ non termina
 - $K(K)$ non termina quando $K(K)$ termina
 - Assurdo conseguente dall'aver supposto l'esistenza del programma H

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 → 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)

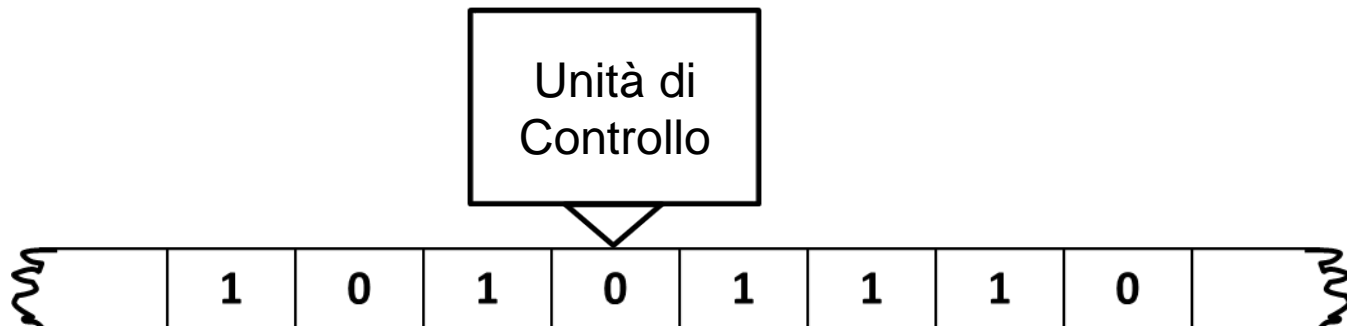
- Modello matematico di computazione introdotto negli anni '30
- Primo formalismo (=linguaggio) con il quale è stata dimostrata l'indecidibilità del problema della fermata



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 >
```

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

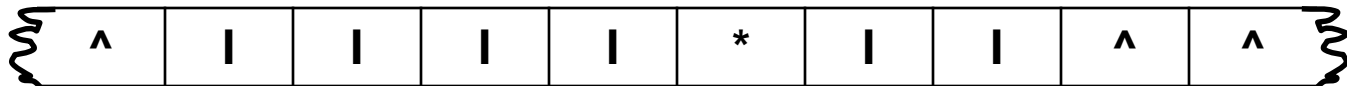
Una MdT è definita da una quintupla:

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

- 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

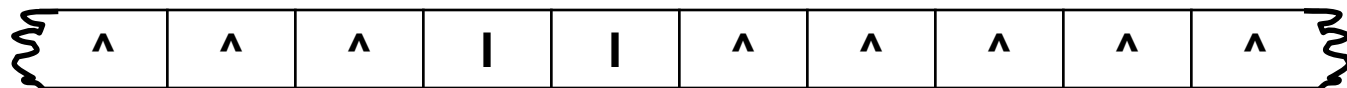


4 - 2

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

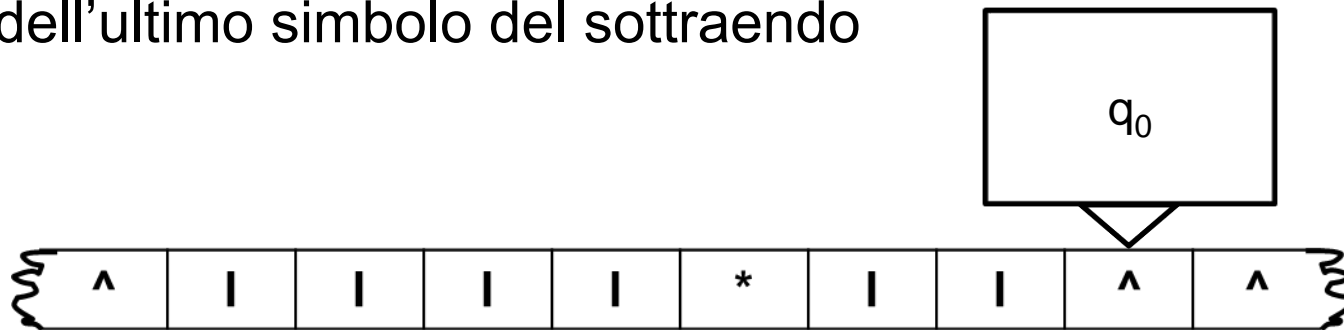
- Definire le funzioni fm , fd , δ
 - in modo da trasformare la configurazione iniziale in quella che rappresenta la soluzione
- In pratica il programma per una MdT è una sequenza di quintuple:

$$\langle x_i \in X, q_j \in Q, x_{ij} \in X, \{S, D, F\}, q_{ij} \in Q \rangle$$

- 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



Configurazione iniziale

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

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

1. Diminuisce 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
 - ☐ Se non ci sono più simboli da cancellare da m allora cancella il separatore \rightarrow HALT
 - ☐ In caso contrario torna al punto 1

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

$$\langle \wedge, q_0, \wedge, S, q_0 \rangle$$

$$\langle I, q_0, \wedge, S, q_1 \rangle$$

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

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

$$\langle I, q_1, I, S, q_1 \rangle$$

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

$$\langle I, q_2, \wedge, D, q_3 \rangle$$

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

$$\langle I, q_3, I, D, q_3 \rangle$$

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

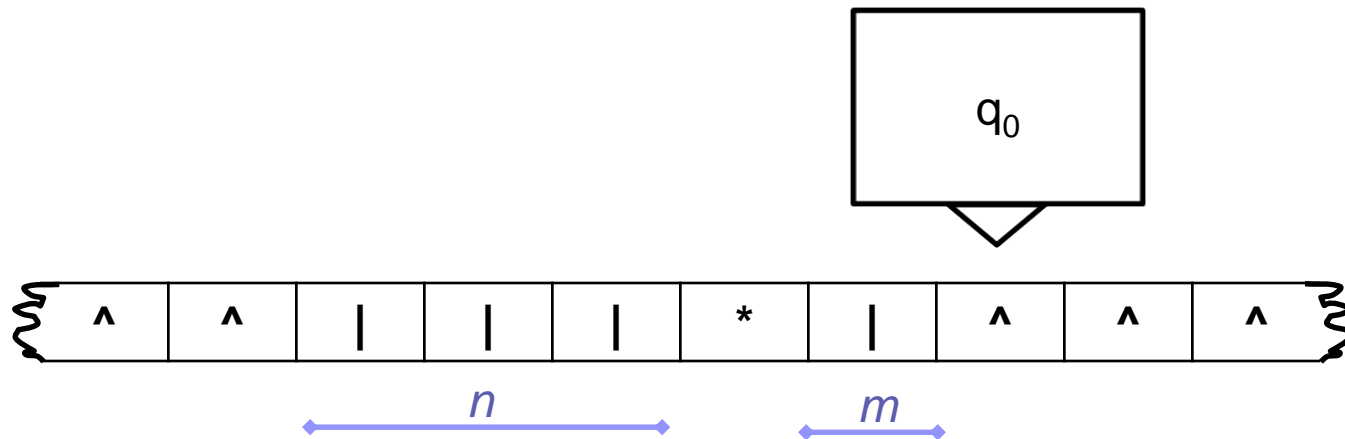
La Matrice Funzionale

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* S q_1$		$* D q_3$

Perché alcune transizioni non sono definite?

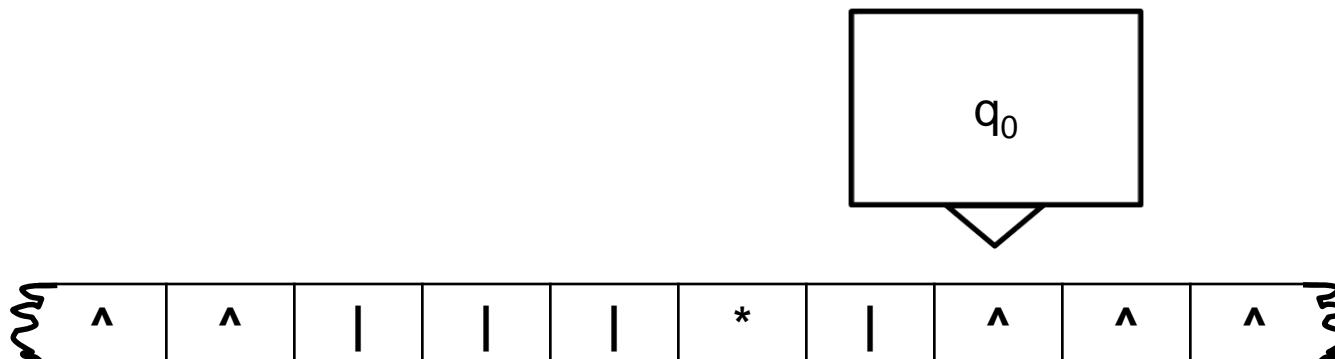
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



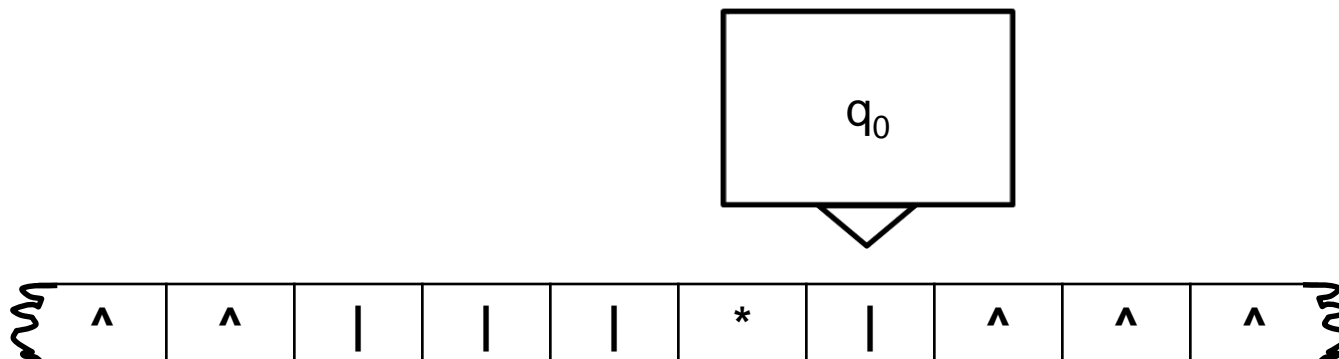
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* S q_1$		$* D q_3$



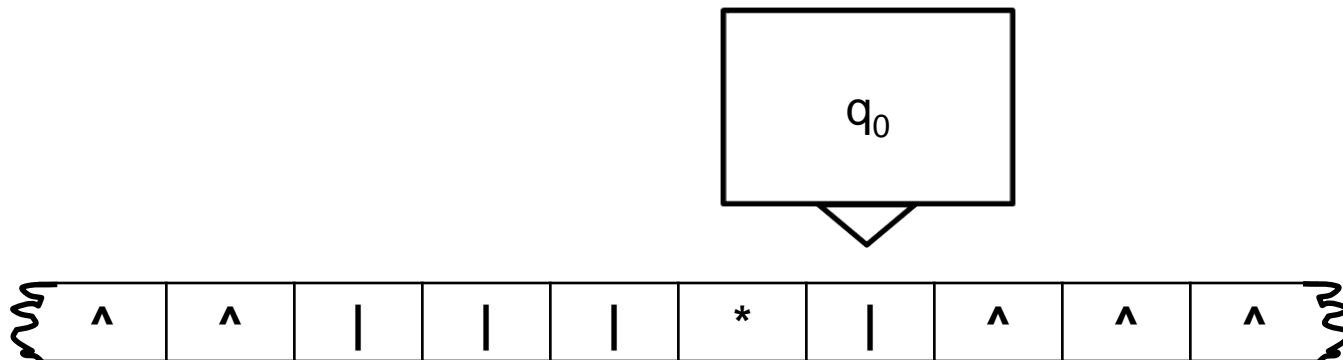
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



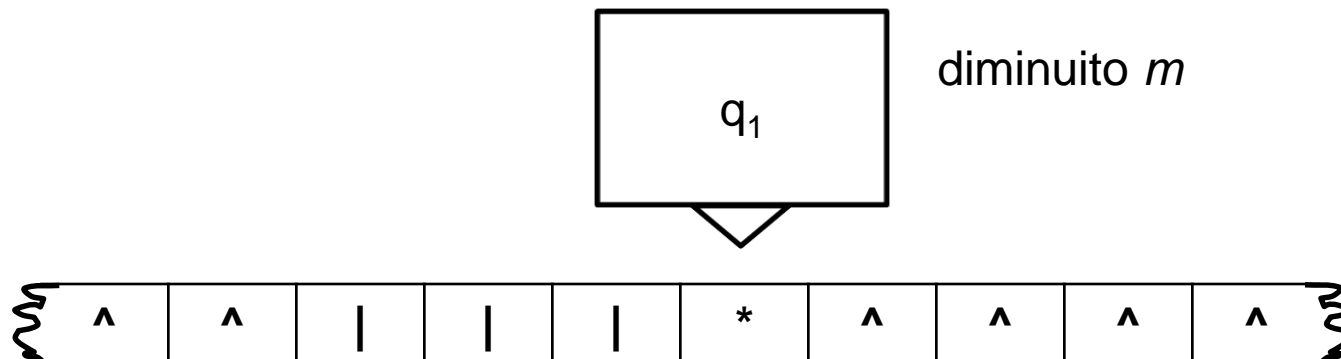
Computazione 3-1

$x \backslash Q$	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	$* Sq_1$		$* Dq_3$



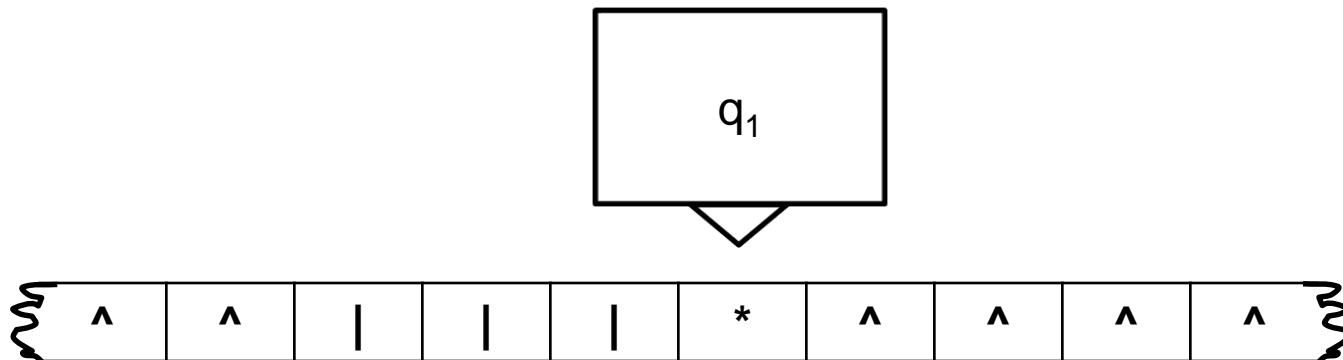
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



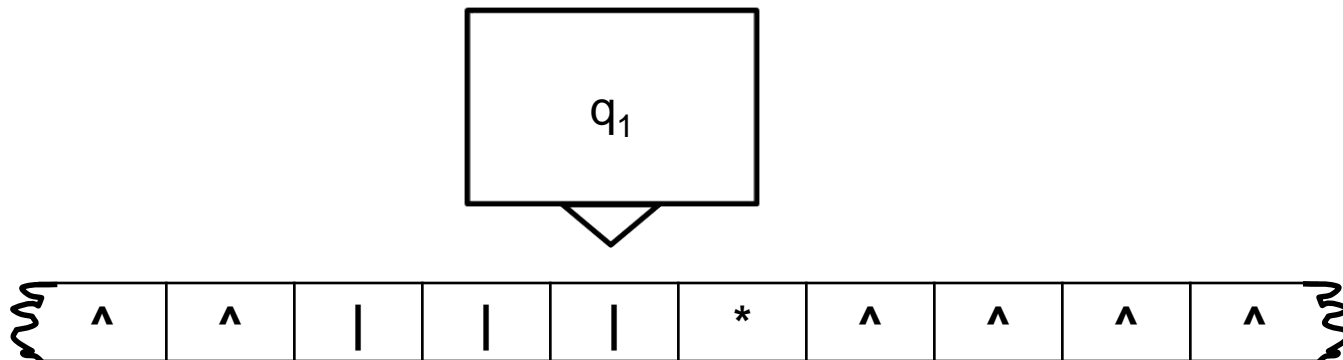
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



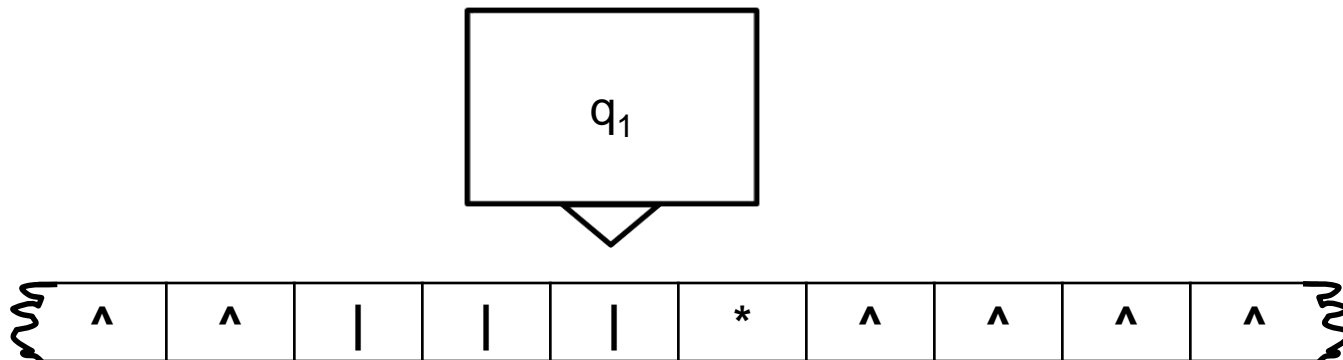
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



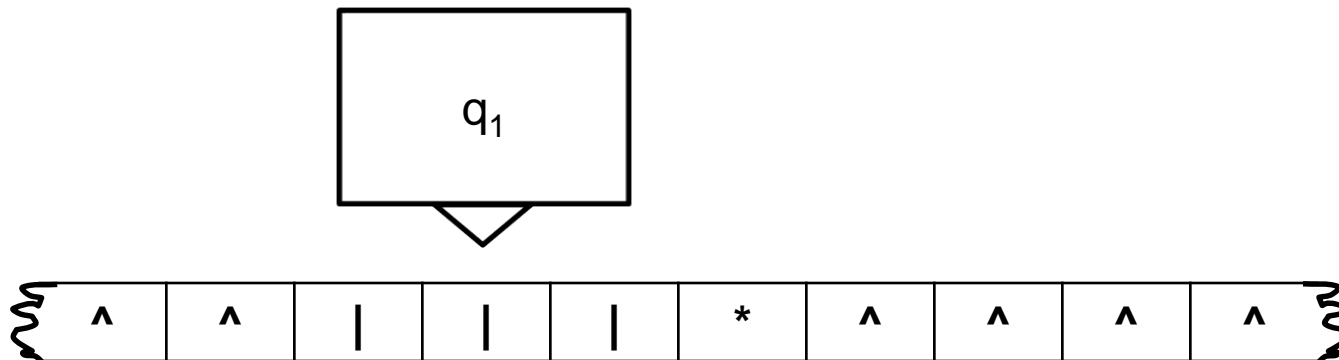
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



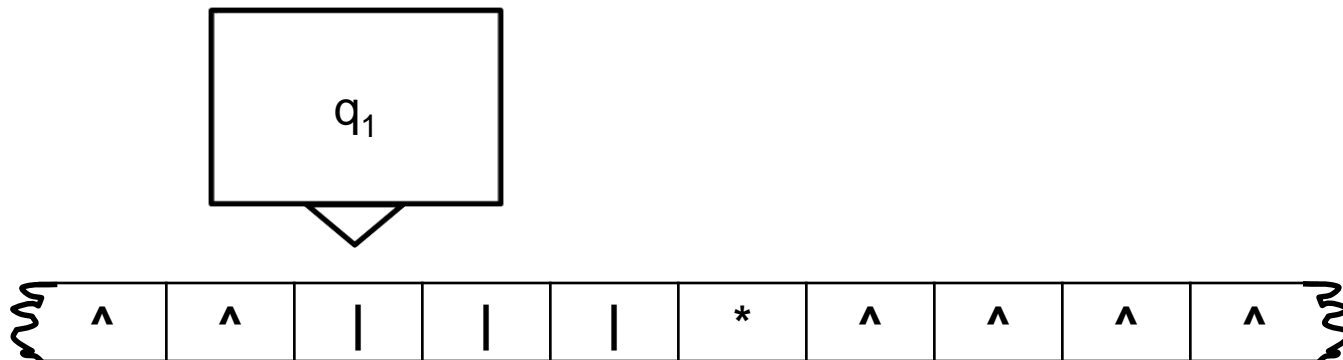
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



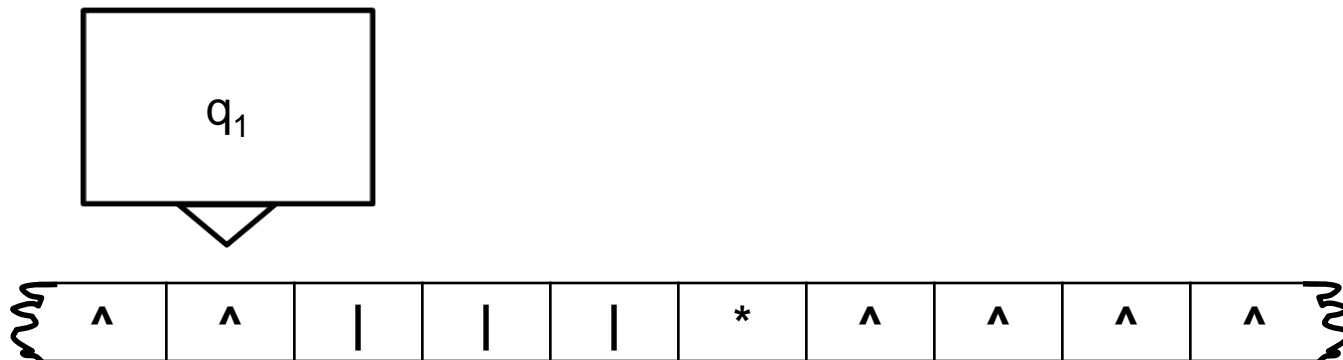
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



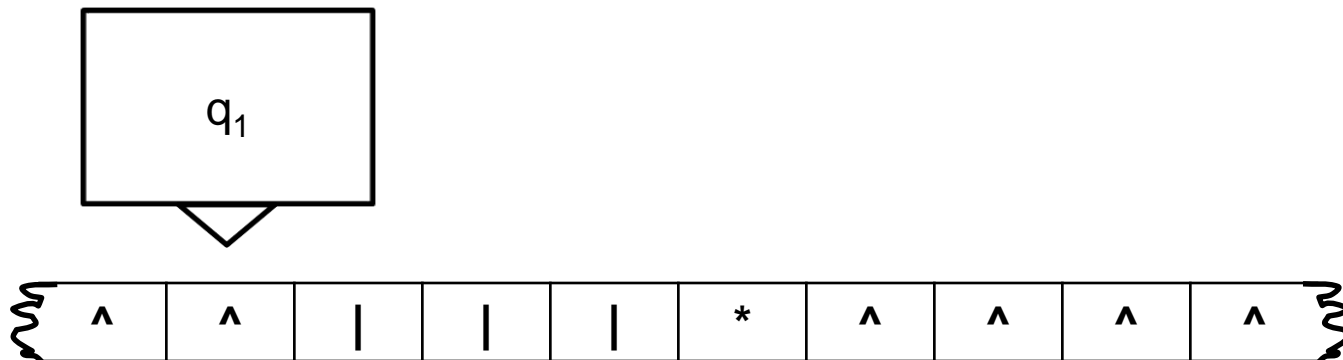
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



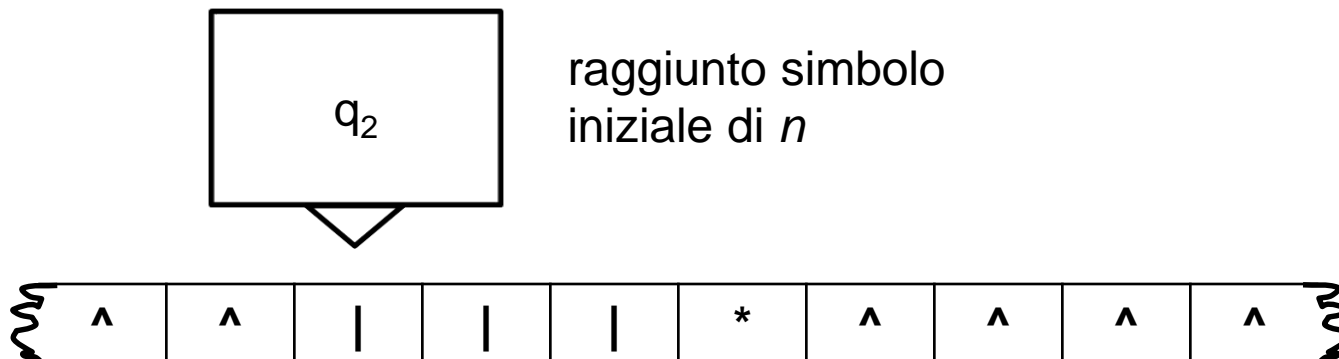
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* S q_1$		$* D q_3$



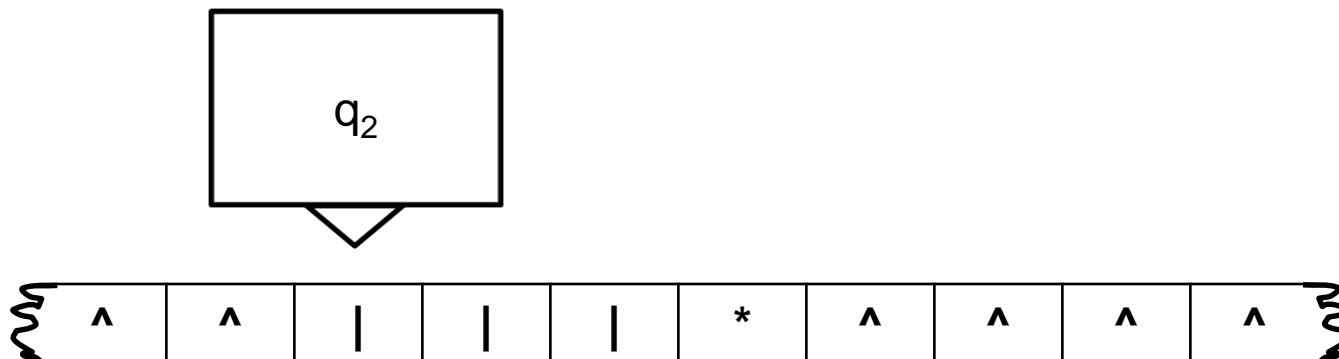
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



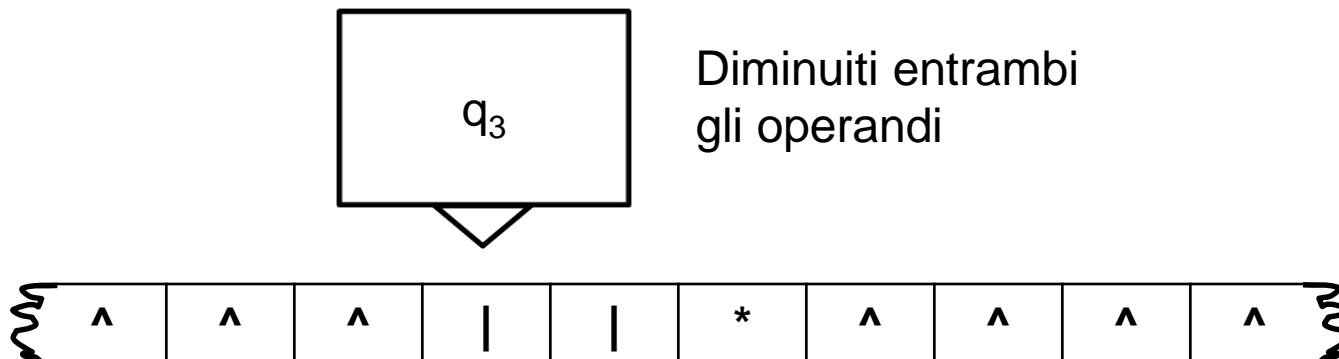
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



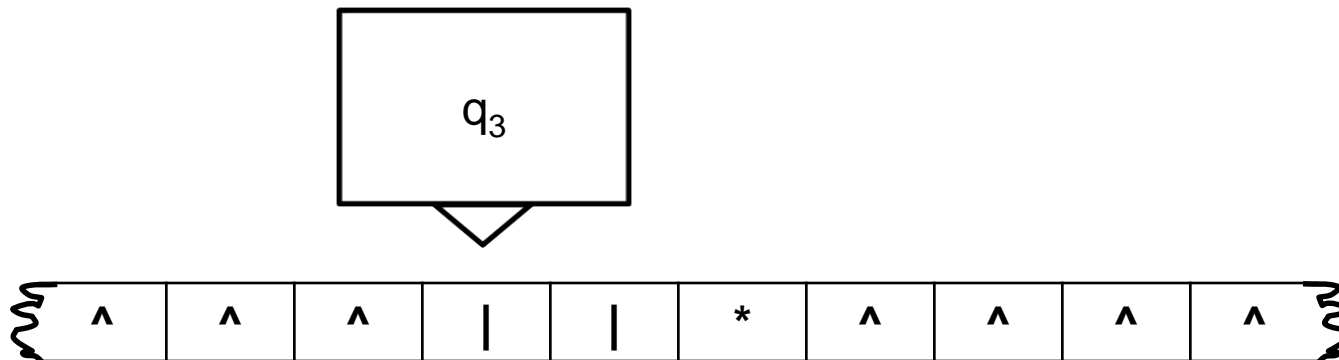
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



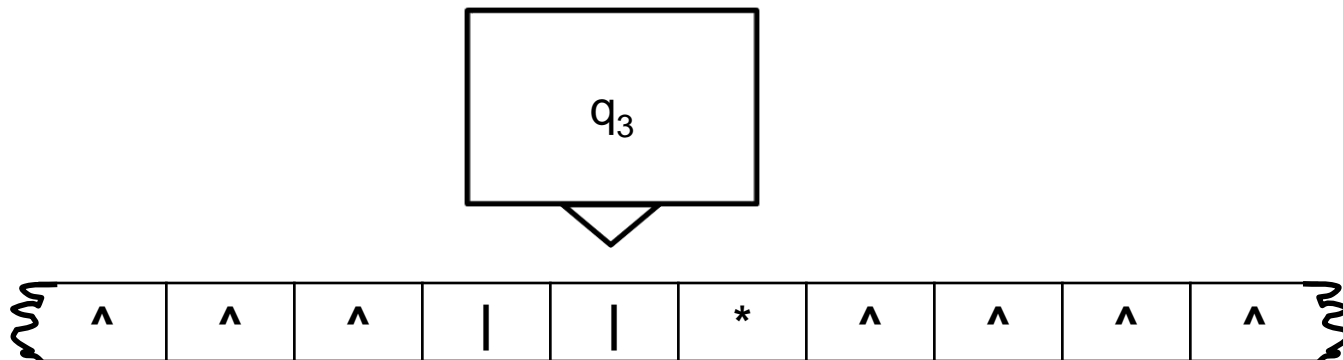
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



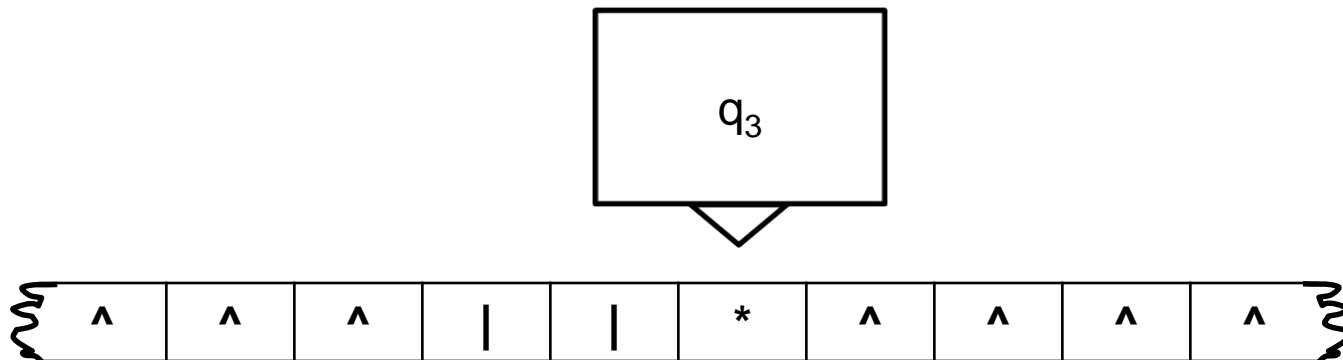
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



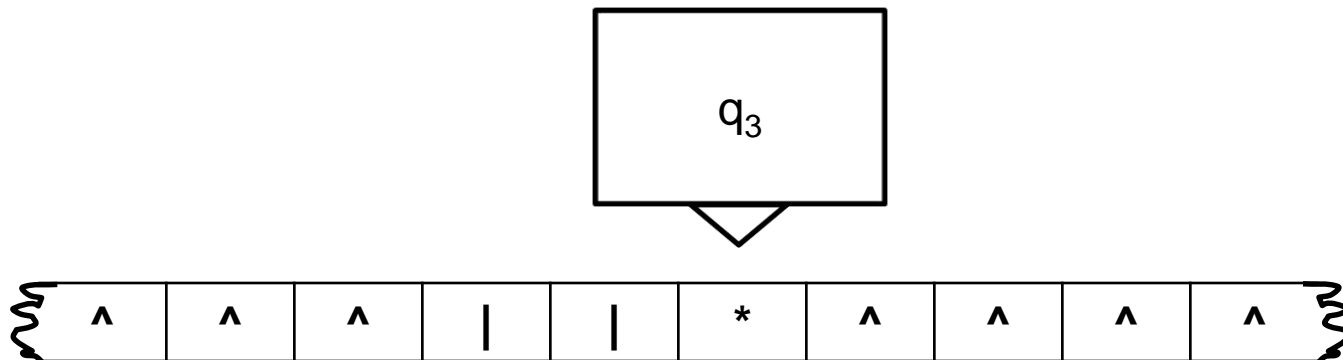
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



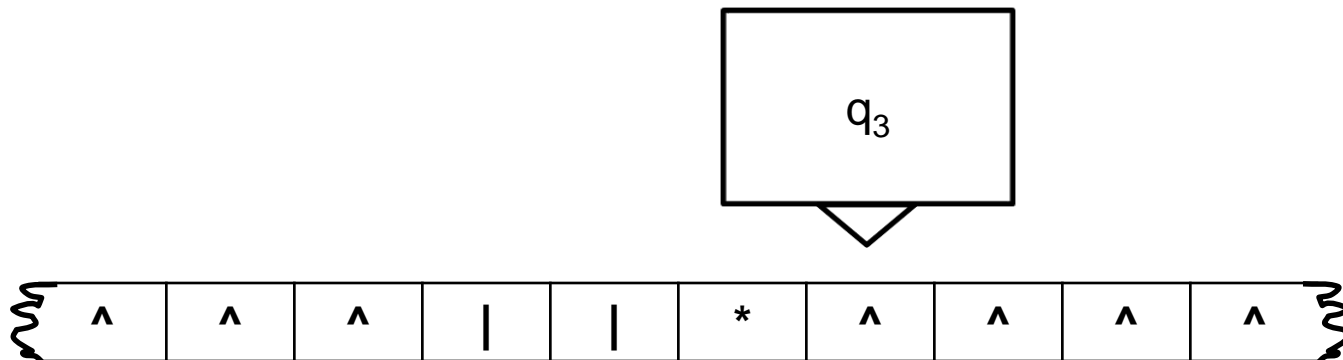
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



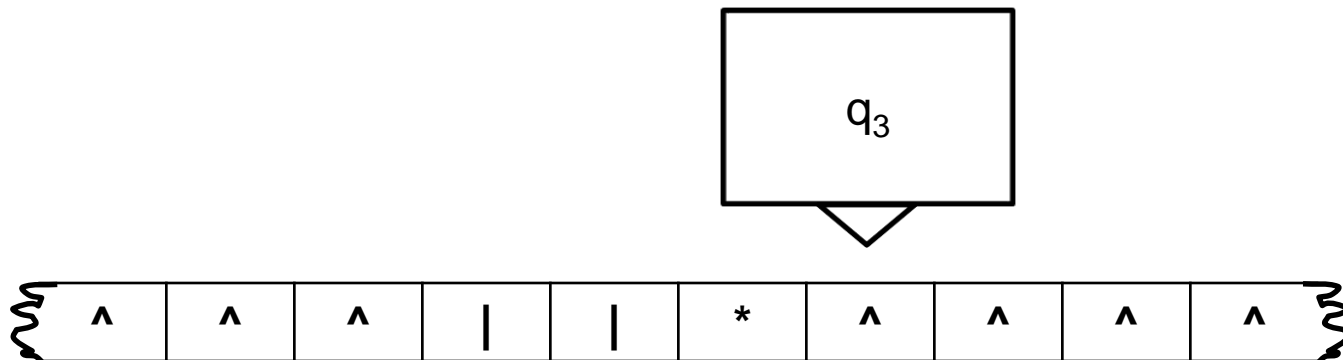
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



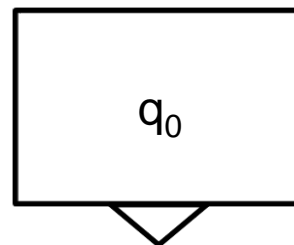
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$

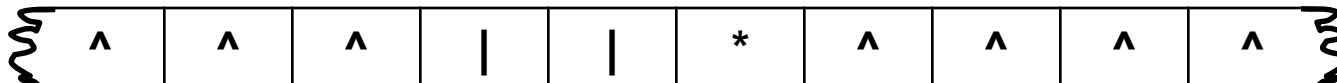


Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



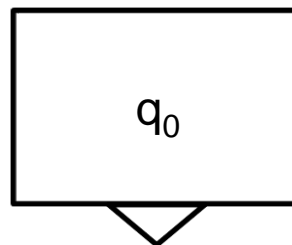
Stato iniziale della
computazione



Computazione 3-1

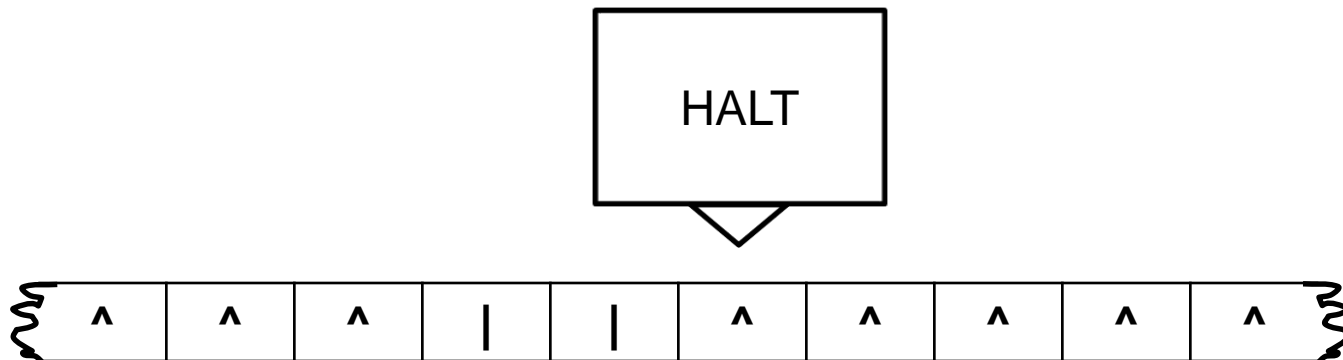
$x \backslash Q$	q_0	q_1	q_2	q_3
\wedge	$\wedge Sq_0$	$\wedge Dq_2$		$\wedge Sq_0$
$ $	$\wedge Sq_1$	$ Sq_1$	$\wedge Dq_3$	$ Dq_3$
$*$	$\wedge F$ <u>HALT</u>	$* Sq_1$		$* Dq_3$

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



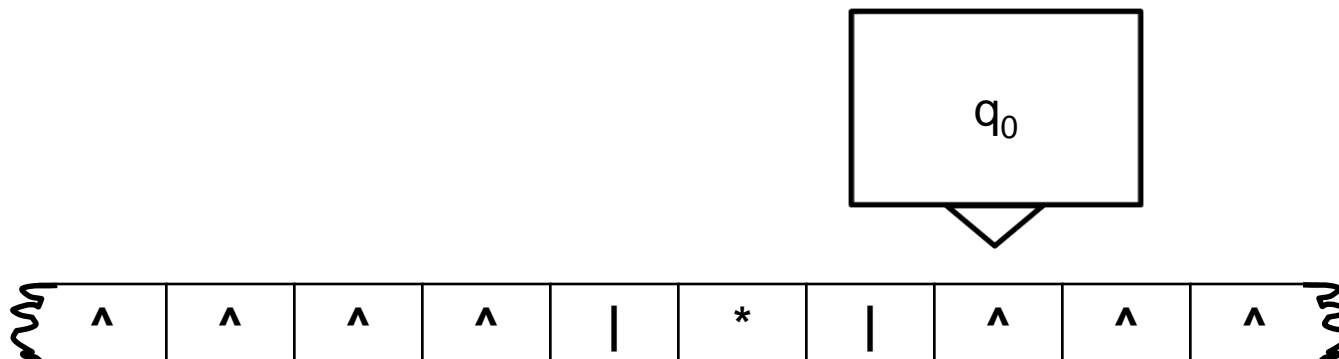
Computazione 3-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



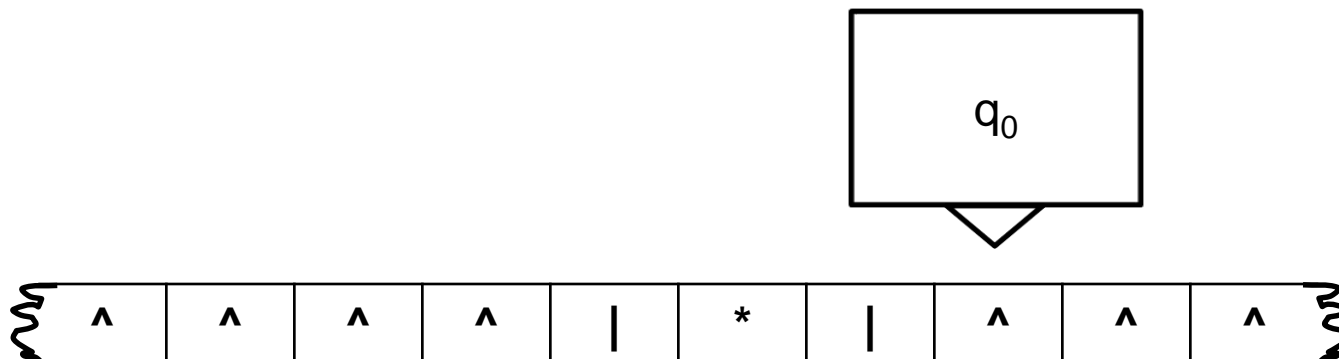
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



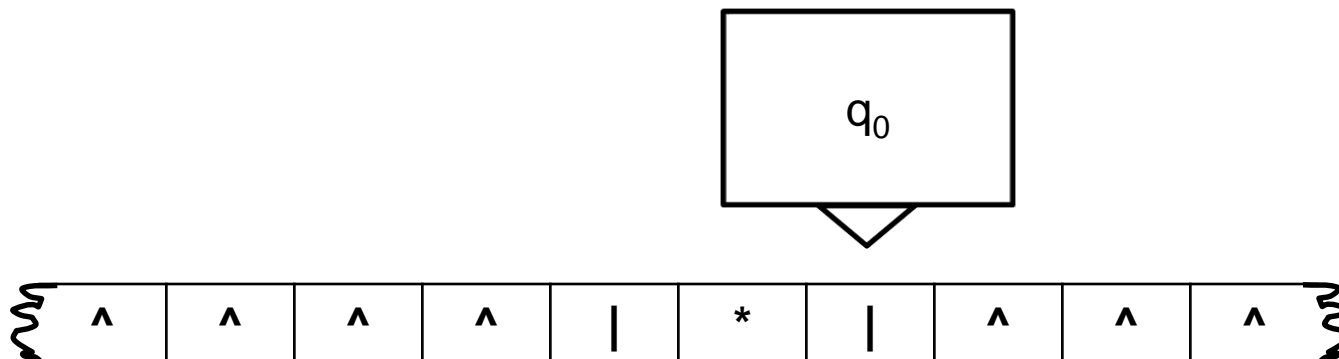
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



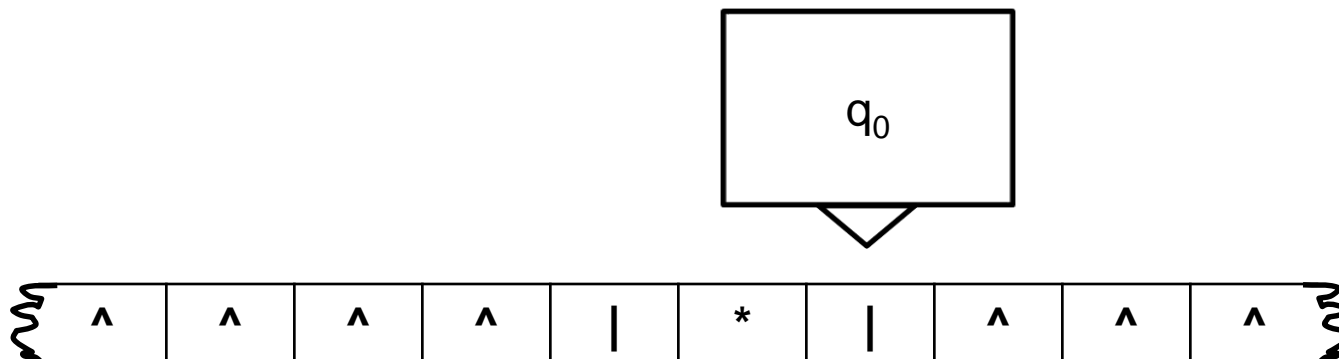
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



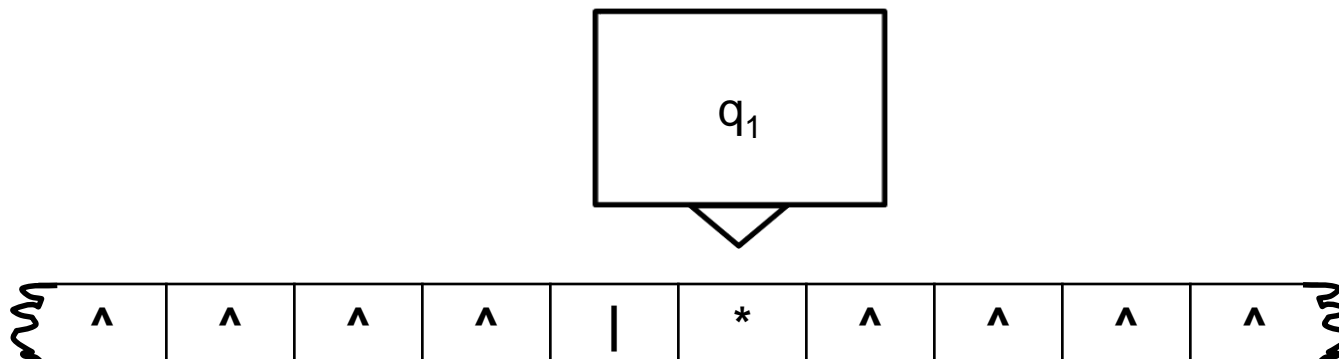
Test casi particolari: computazione 1-1

$x \backslash Q$	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	$* Sq_1$		$* Dq_3$



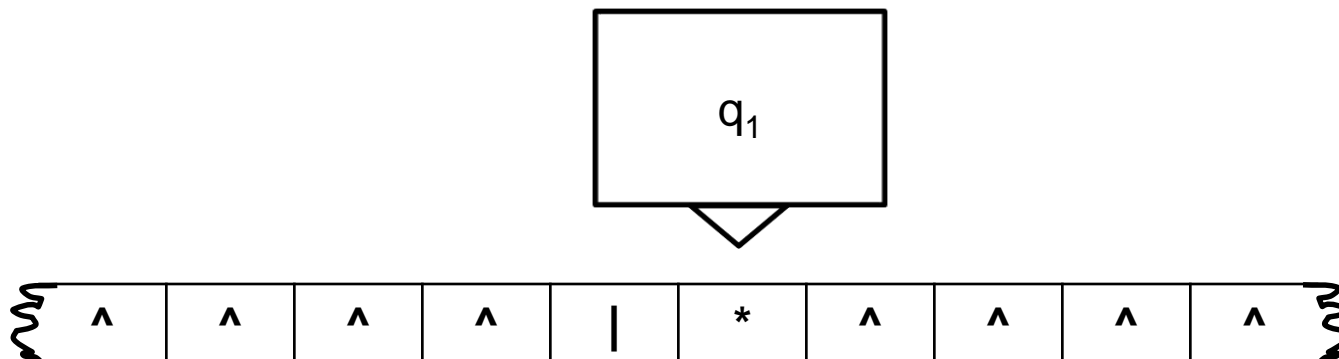
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



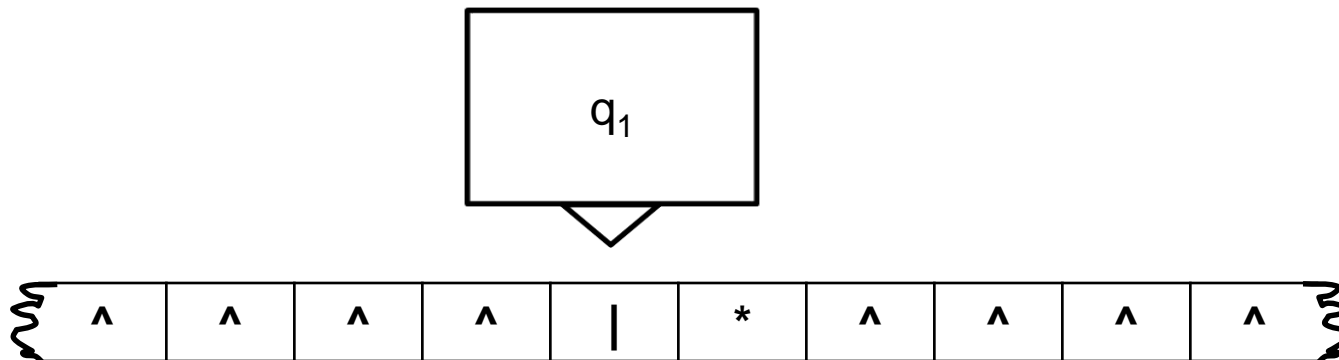
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



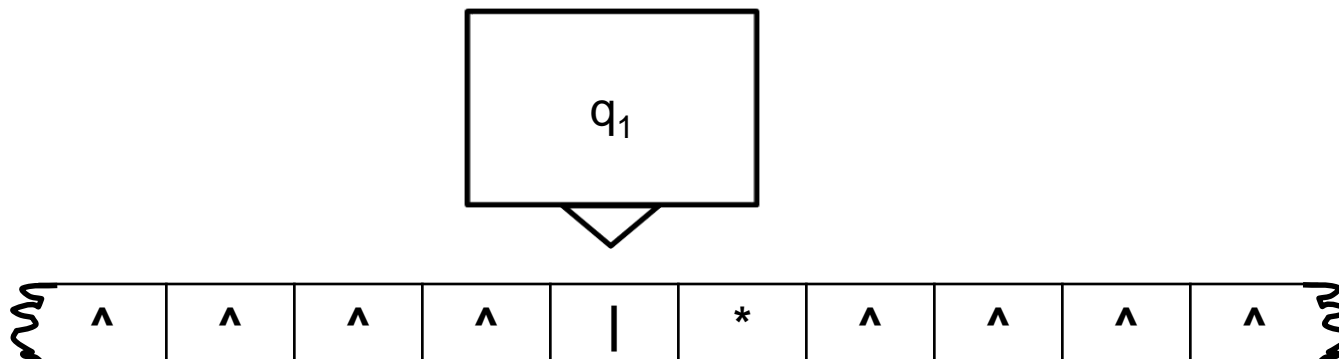
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



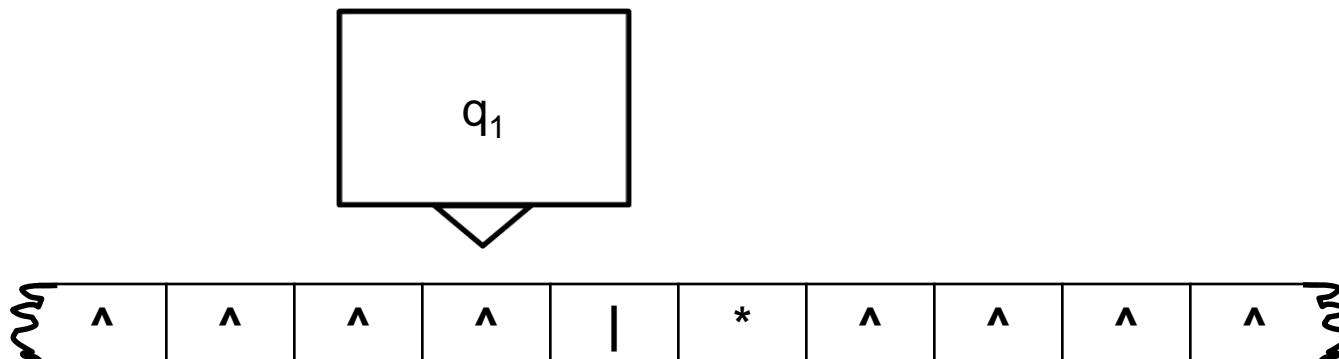
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



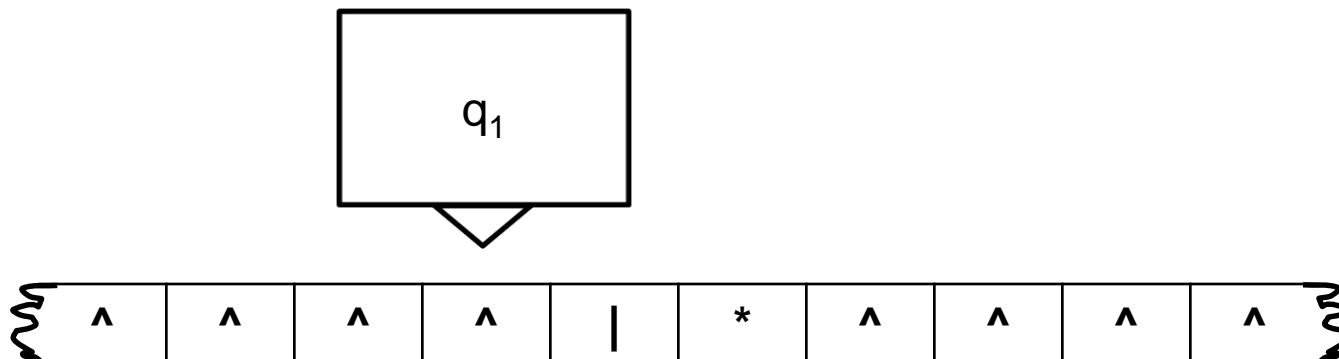
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



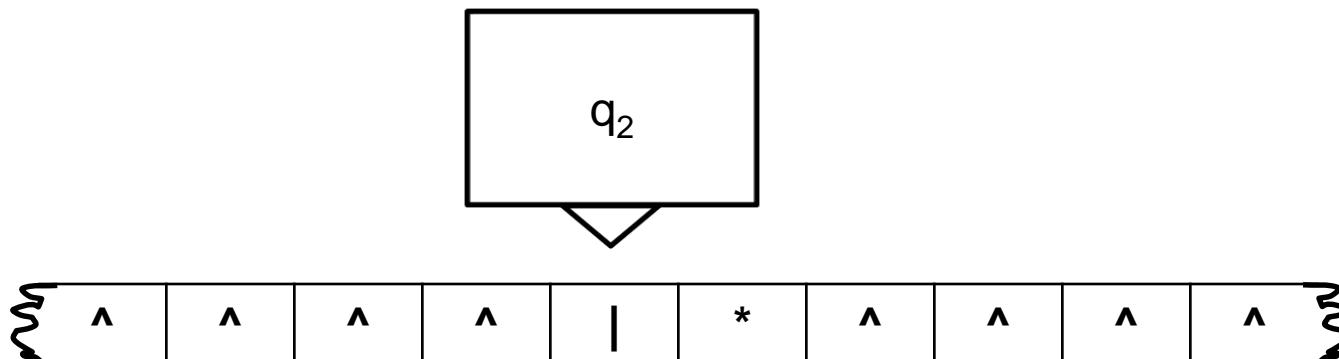
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



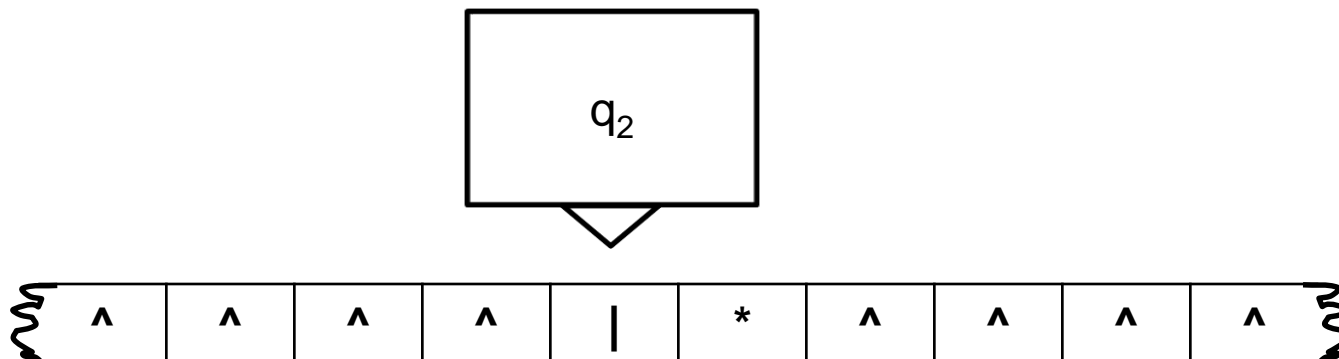
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



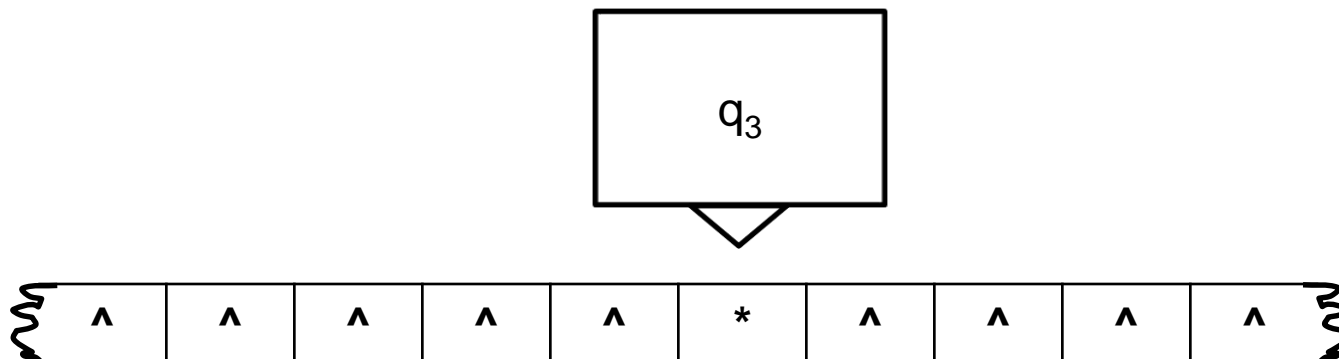
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



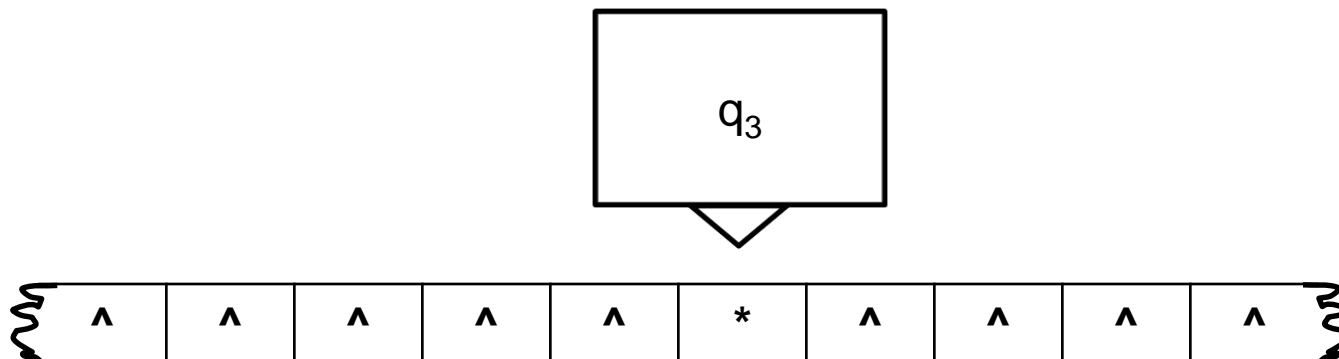
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



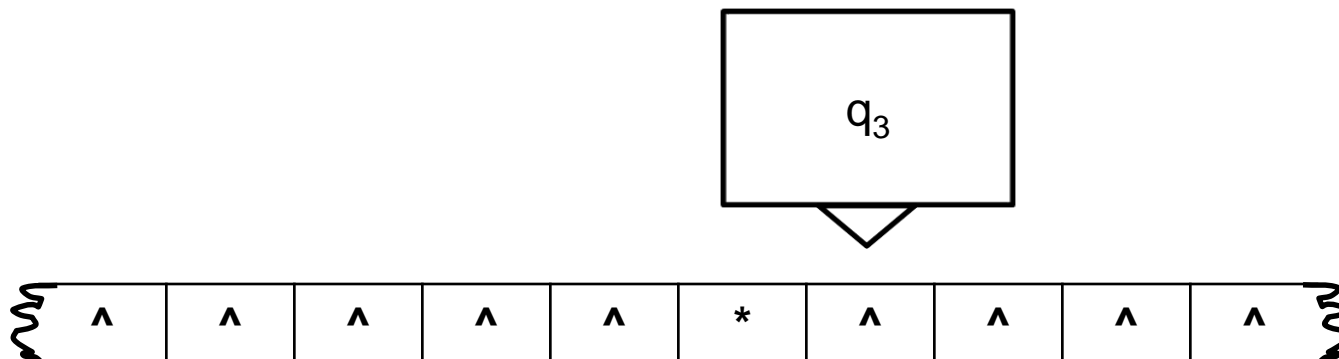
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



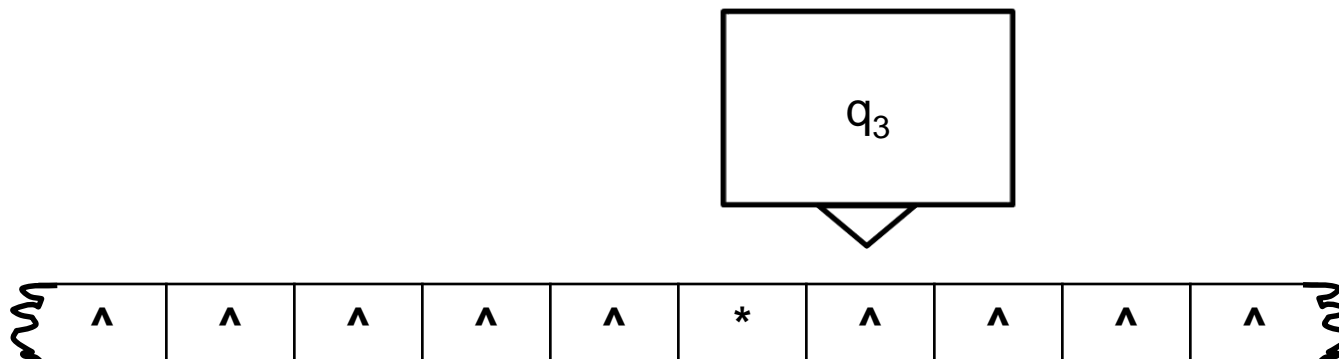
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



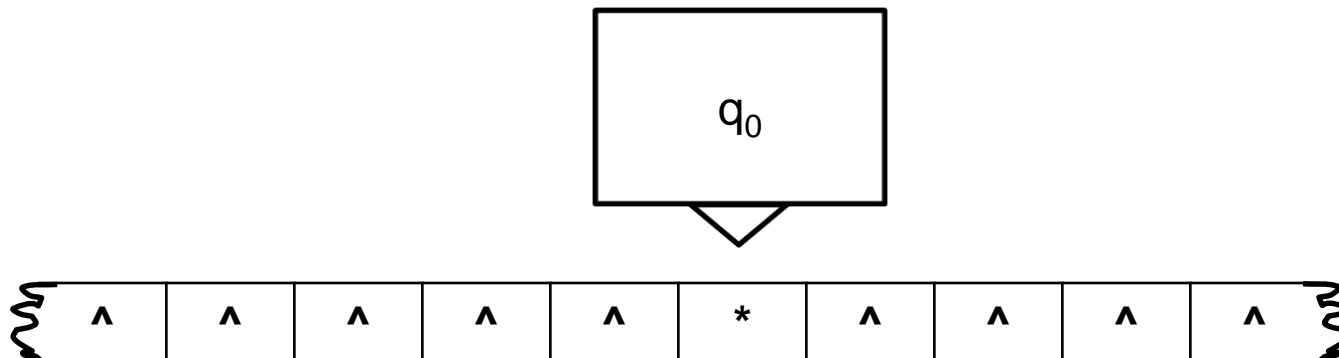
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



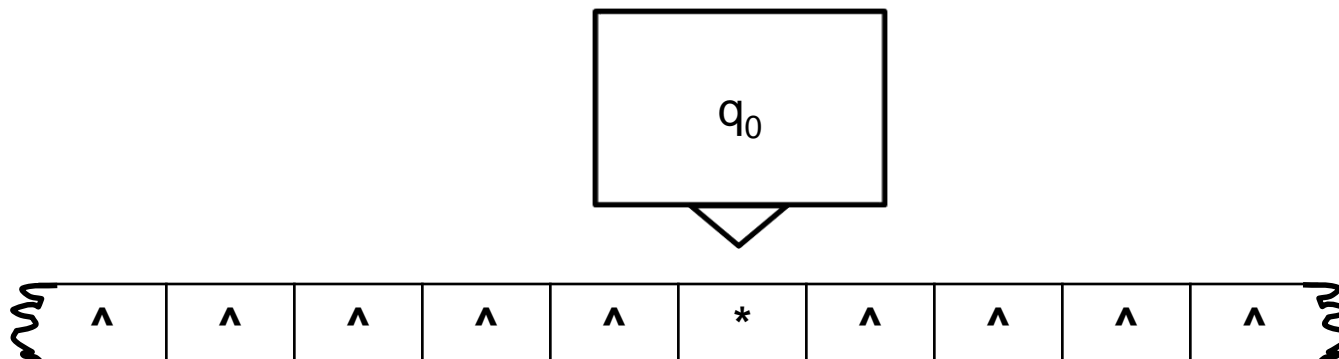
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



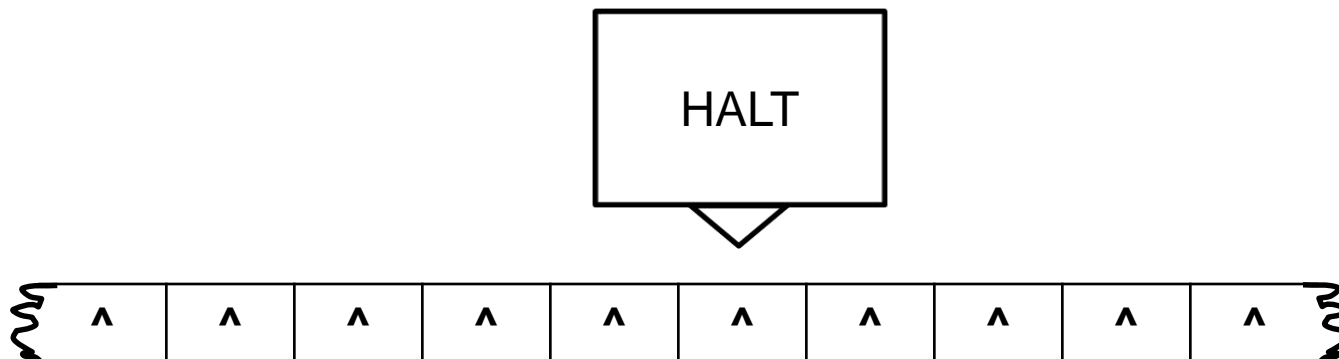
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F$ <u>HALT</u>	$* Sq_1$		$* Dq_3$



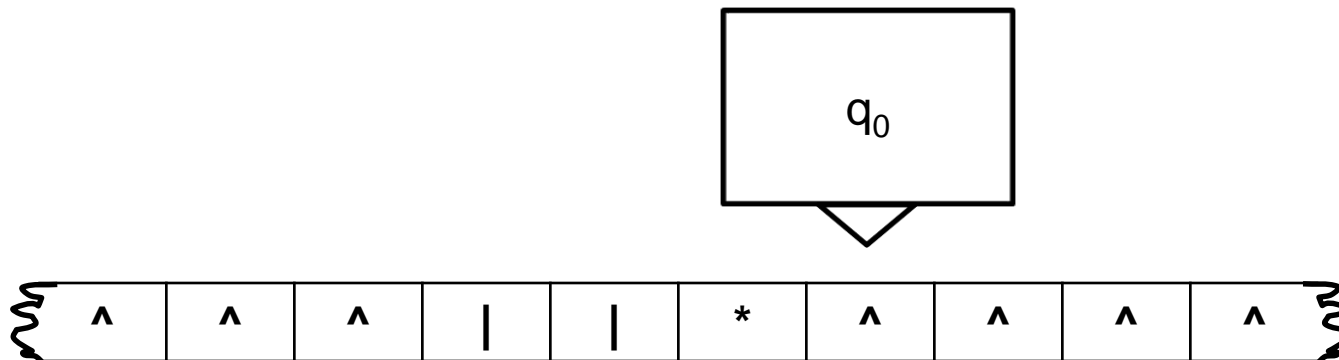
Test casi particolari: computazione 1-1

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



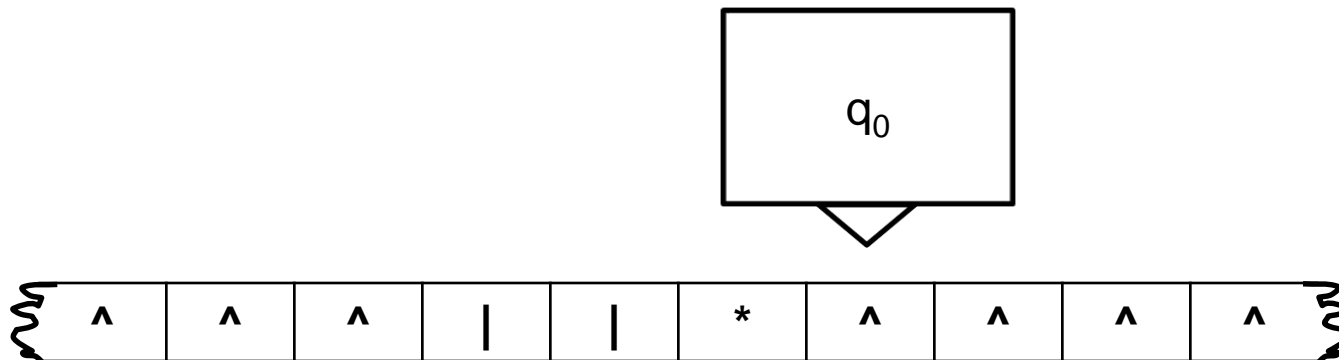
Test casi particolari: computazione 2-0

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



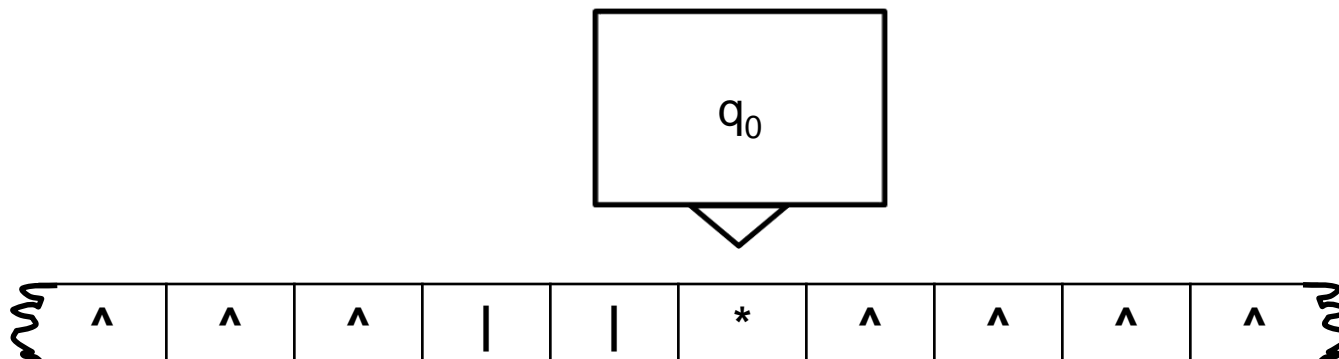
Test casi particolari: computazione 2-0

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



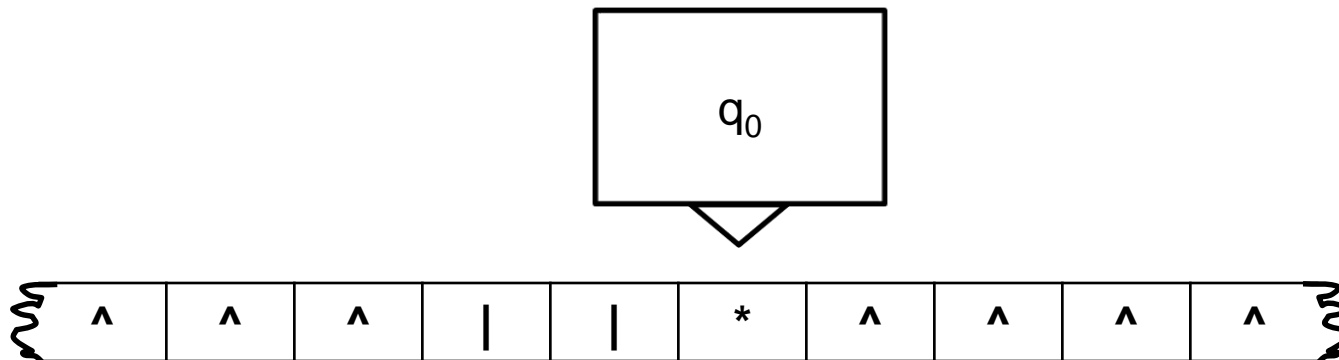
Test casi particolari: computazione 2-0

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



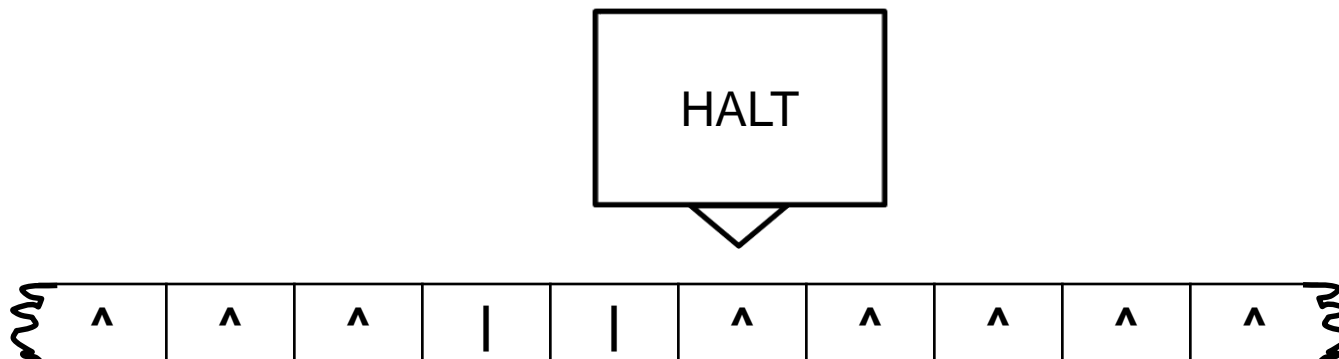
Test casi particolari: computazione 2-0

$x \backslash Q$	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$
$*$	$\wedge F$ <u>HALT</u>	$* Sq_1$		$* Dq_3$



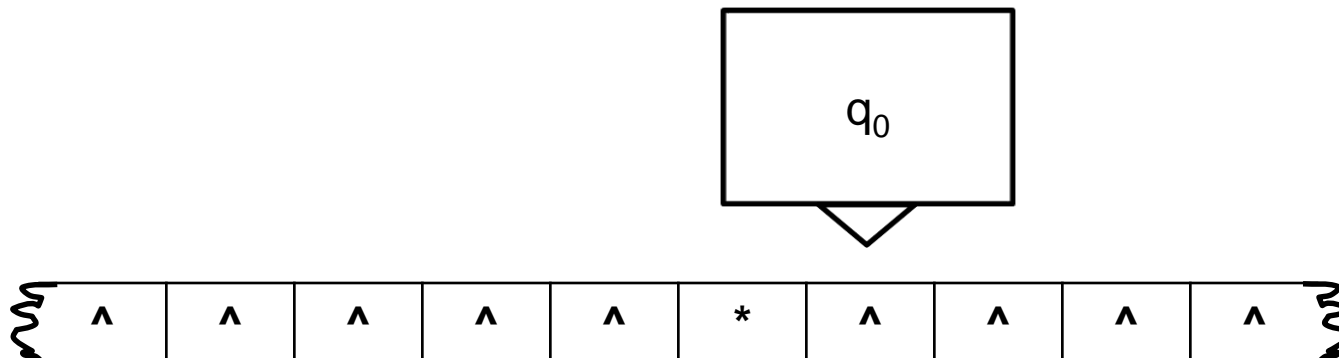
Test casi particolari: computazione 2-0

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



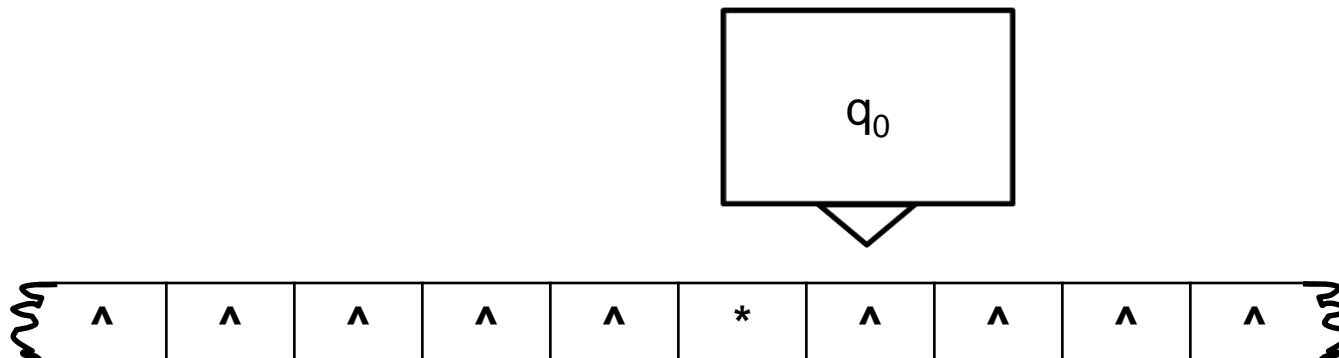
Test casi particolari: computazione 0-0

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



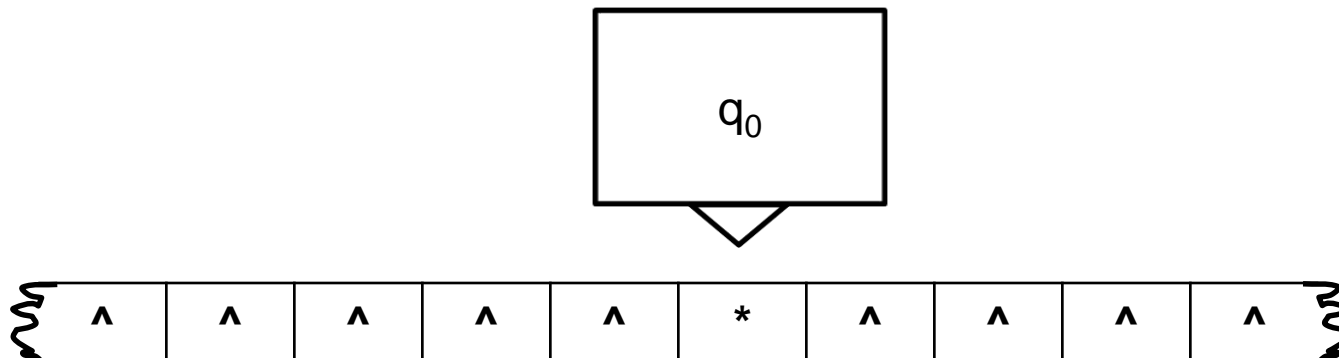
Test casi particolari: computazione 0-0

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



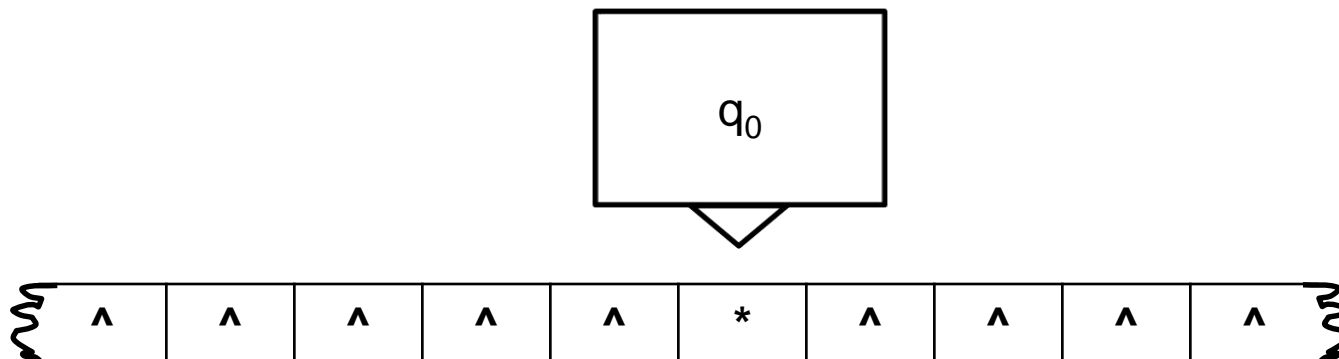
Test casi particolari: computazione 0-0

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



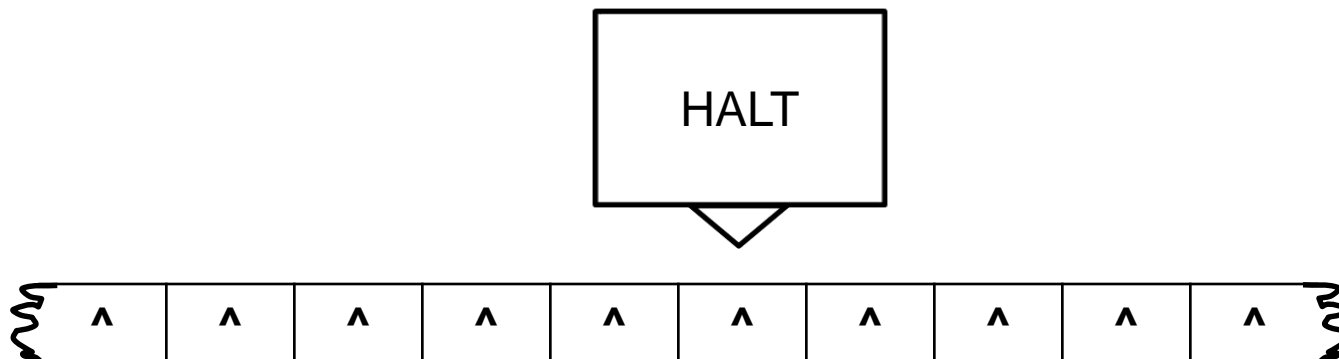
Test casi particolari: computazione 0-0

$x \backslash Q$	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$
$*$	$\wedge F$ <u>HALT</u>	$* Sq_1$		$* Dq_3$



Test casi particolari: computazione 0-0

$x \backslash Q$	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$
$*$	$\wedge F \text{ HALT}$	$* Sq_1$		$* Dq_3$



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 '|' 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

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

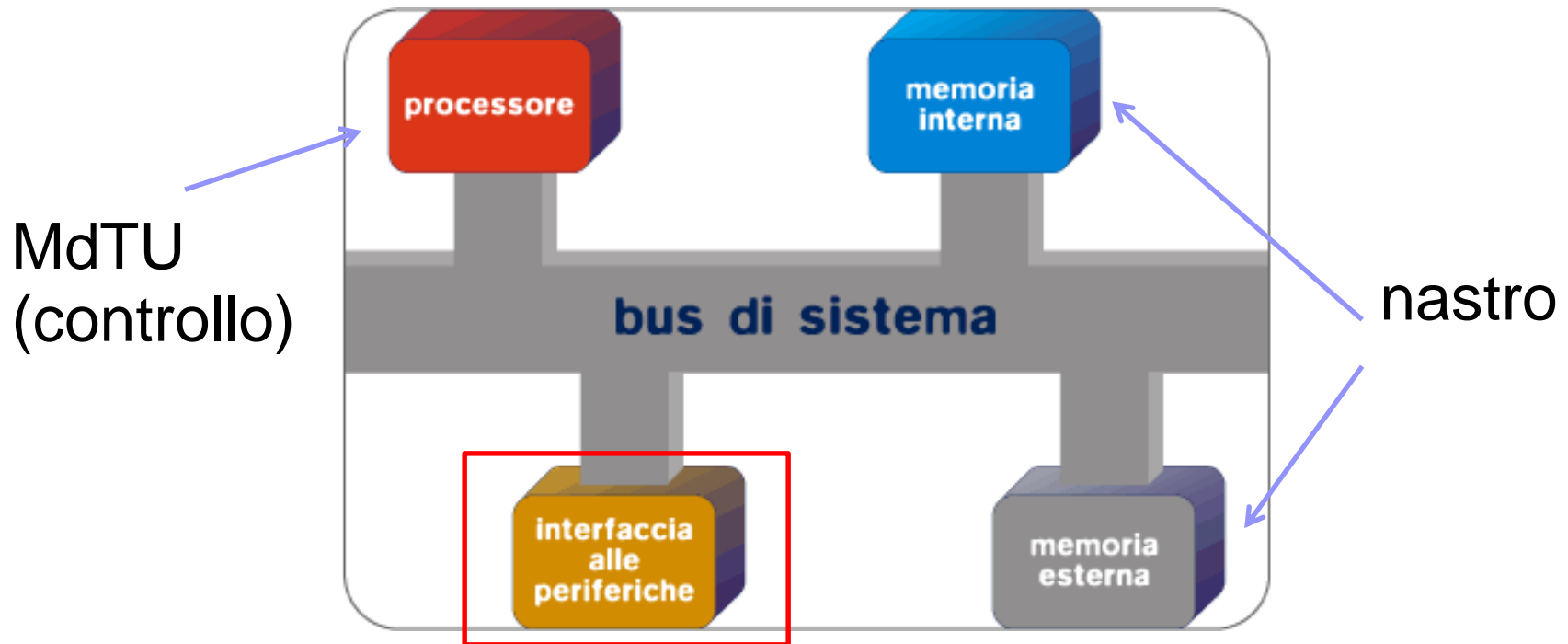
■ CPU

- 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

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
- E' un interprete!!!!

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)

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

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.
- 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

- Livelli di descrizione di un linguaggio
 - grammatica
 - quali frasi sono corrette?
 - semantica
 - cosa significa 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

- 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 per descrivere la sintassi di un linguaggio di programmazione**

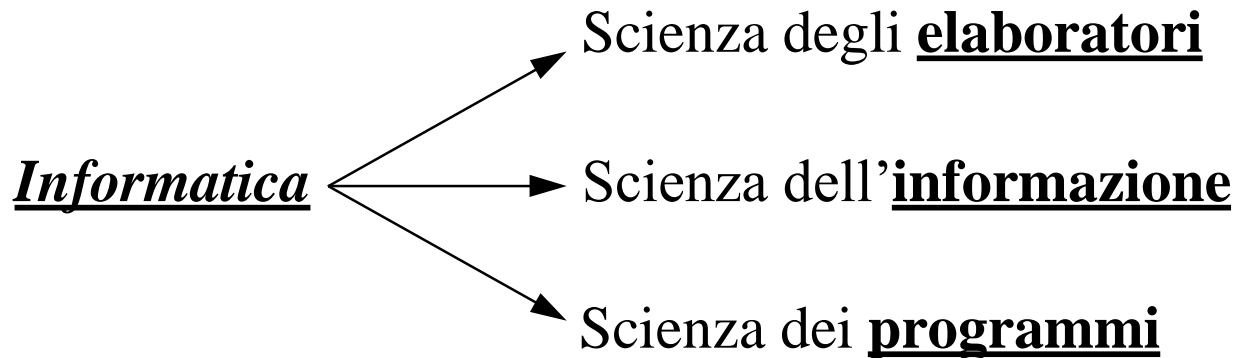
Introduzione alla teoria dei linguaggi formali

- Questa prima parte del programma ha l'obiettivo di fornire una **base** sufficiente alla comprensione delle **tecniche di compilazione**, oggetto della seconda parte del corso.
- E' per questa ragione che 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

- L'informatica teorica è la **scienza degli algoritmi**, in tutti i loro aspetti, poiché è il concetto di algoritmo che è in qualche modo comune a tutti i settori dell'informatica.

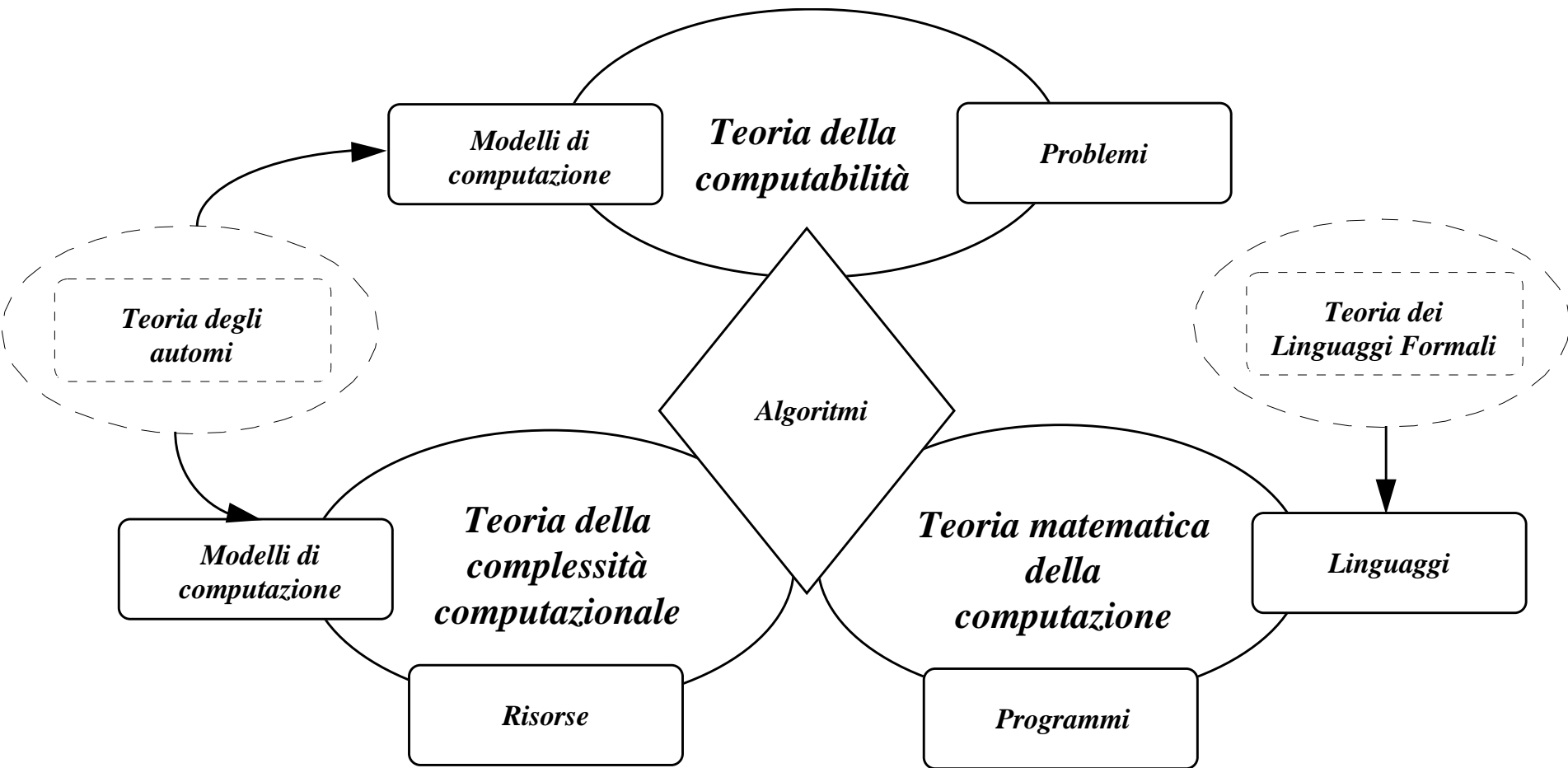


elaboratori ≡ macchine che eseguono gli algoritmi

informazione ≡ materia su cui lavorano gli algoritmi

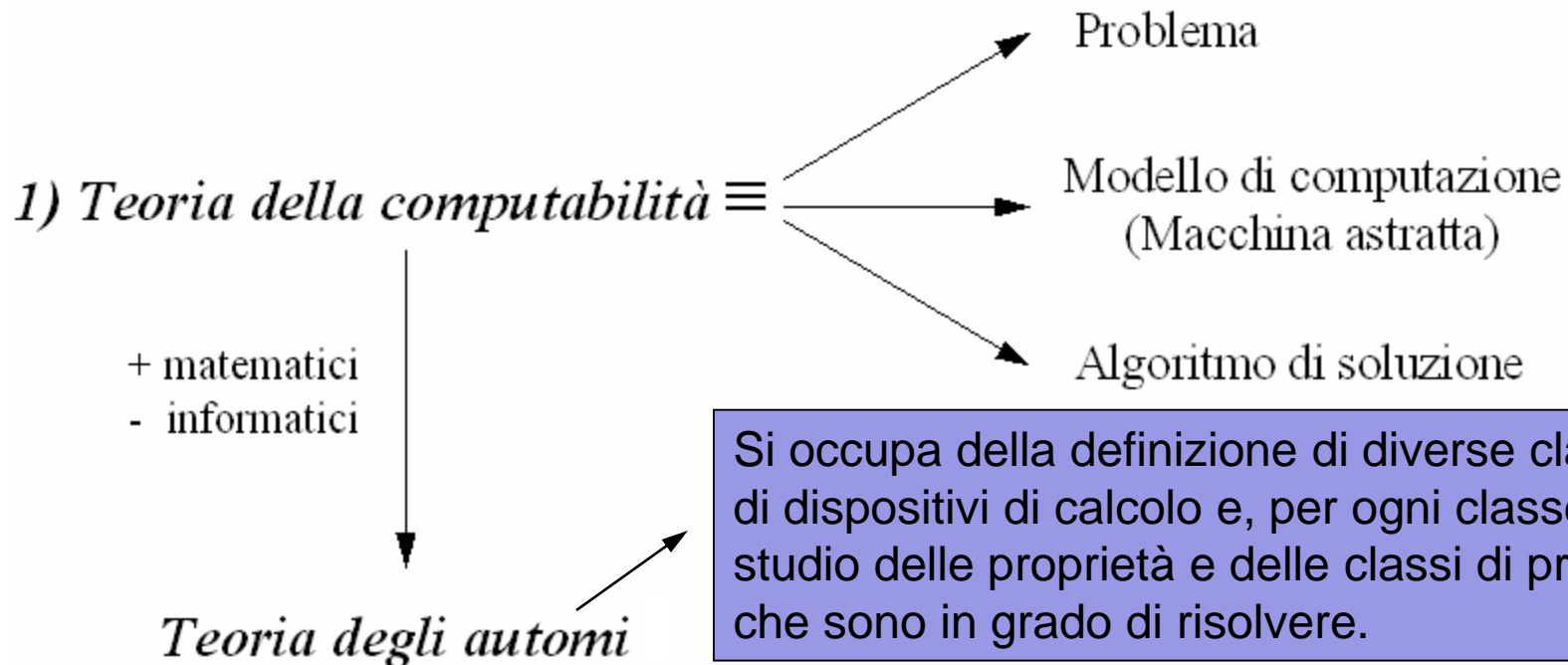
programmi ≡ algoritmi descritti in un particolare linguaggio

Aree di ricerca dell'informatica teorica



Aree di ricerca dell'informatica teorica

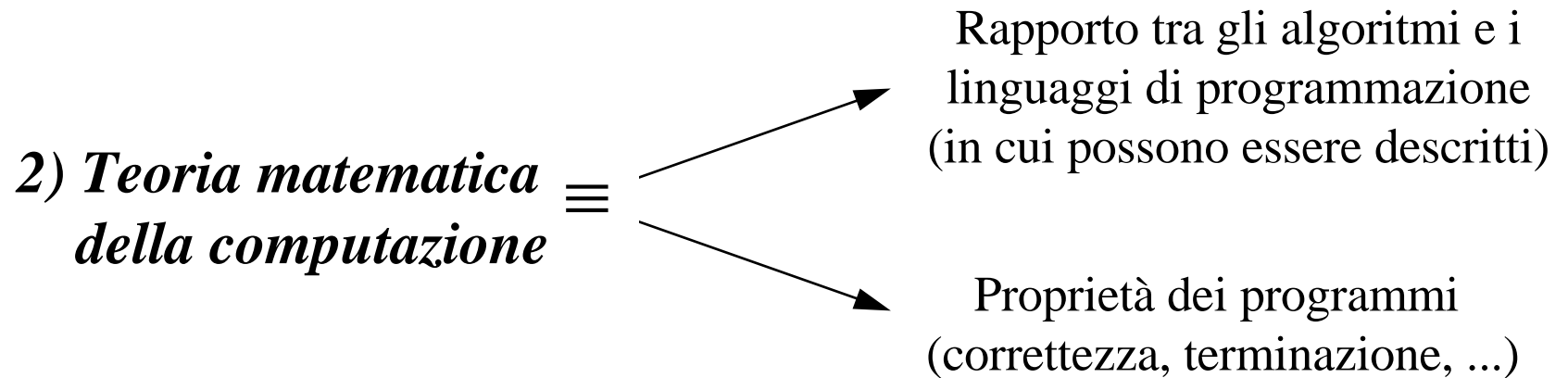
- Lo schema riportato in figura fornisce una panoramica delle aree di ricerca che ricadono nell'ambito più generale dell'informatica teorica. Tali aree sono:



Aree di ricerca dell'informatica teorica

- Particolarmente feconda si è rivelata l'interazione tra la ***Teoria degli Automi*** e la ***Teoria dei Linguaggi Formali***.
- Quest'ultima, nata in ambito linguistico per caratterizzare i linguaggi naturali, è stata sviluppata ed utilizzata dagli informatici teorici, che avevano l'esigenza di descrivere gli algoritmi con linguaggi di programmazione comprensibili dalla macchina, ma non troppo difficili per l'uomo e soprattutto **non ambigui**.

Aree di ricerca dell'informatica teorica



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

Studio dei linguaggi

- Quando iniziamo a studiare un linguaggio, formale o meno, 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, ...

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.

Sintassi e semantica

- I nostri programmi devono aderire rigorosamente a regole stringenti e anche differenze minori vengono rigettate come errate.
- Idealmente, ogni frase in un linguaggio dovrebbe essere corretta sia **semanticamente** (avere cioè il corretto significato) sia **sintatticamente** (avere la corretta struttura grammaticale).
- Nell'italiano parlato, le frasi sono spesso sintatticamente sbagliate, ciononostante convogliano la semantica desiderata.

Sintassi e semantica

- In programmazione, comunque, è essenziale che la **sintassi** sia corretta al fine di comunicare una qualsivoglia semantica.
- Quando studiamo la semantica di una frase, ne stiamo studiando il significato.

Sintassi e semantica

- 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

Sintassi e semantica

- Lo studio della sintassi è lo studio della grammatica, cioè della struttura delle frasi. La frase:

The man hits the dog

può essere analizzata sintatticamente, cioè risolta nelle parti grammaticali componenti, come segue:



ed ogni frase in questa forma è sintatticamente valida in inglese.

- Possiamo descrivere un particolare insieme di tali frasi semplici in inglese usando le seguenti regole:

- 115/134**

Sintassi e semantica

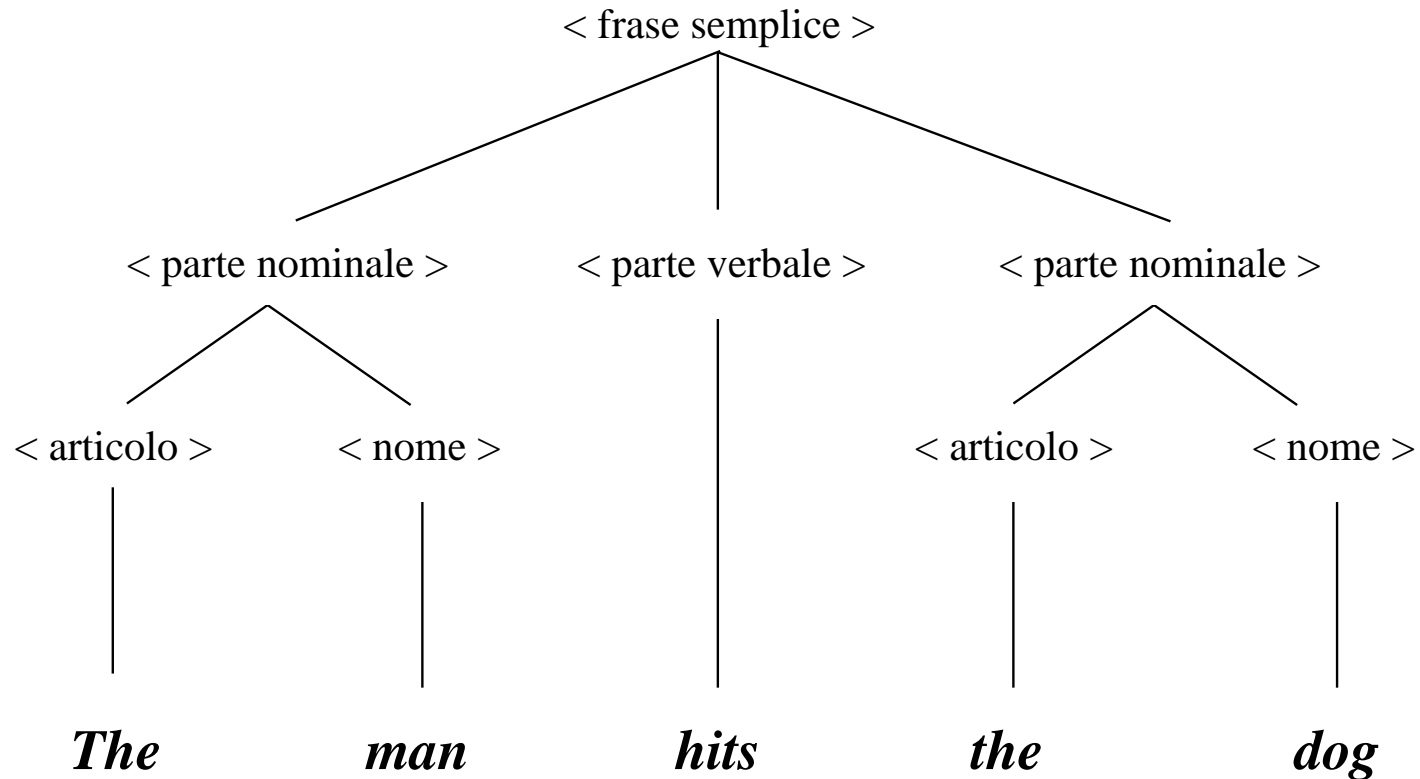
- Queste regole sono scritte in **BNF**, una notazione usata comunemente per descrivere la sintassi dei linguaggi di programmazione.
- BNF (Backus Naur Form) è un metalinguaggio.
- Nell'esempio, *<frase semplice>* è definita come una *<parte nominale>* seguita da una *<parte verbale>* seguita, a sua volta, da un'altra *<parte nominale>*.
- Ciascuna delle due occorrenze di *<parte nominale>* deve essere espansa in *<articolo>* seguito da un *<nome>*.

Sintassi e semantica

- Scegliendo di espandere 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**, si dimostra che la frase **The man hits the dog** è una delle nostre frasi semplici.

Sintassi e semantica

- Tutto ciò si può riassumere nel seguente
Albero di derivazione



Sintassi e semantica

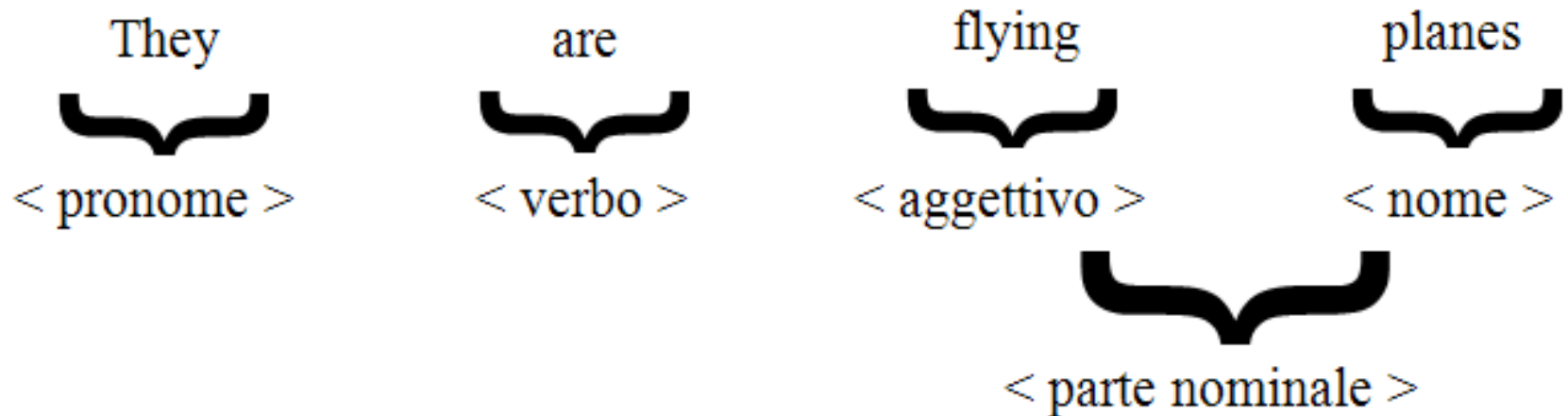
- La definizione data di *<frase semplice>* dà luogo a 72 diverse frasi derivabili.
 - Sebbene tutte siano sintatticamente corrette, in base alla nostra definizione, alcune non hanno un'interpretazione semantica sensata (non hanno senso compiuto).
 - Una di queste è la frase: ***The car eats the man***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

Sintassi e semantica

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

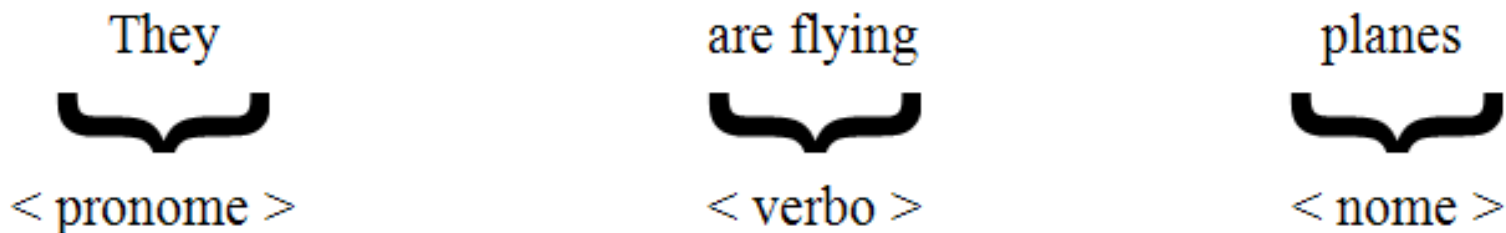
They are flying planes

- Un'analisi sintattica di questa frase è:



Sintassi e semantica

- Quest'analisi suggerisce che **they** si riferisce a **planes** (*aerei nel cielo che volano*). Un'analisi alternativa sarebbe:



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.
- Una fase cruciale nel processo di compilazione di un programma è l'**analisi sintattica**, come passo essenziale per la sua **interpretazione semantica**.

Teoria dei linguaggi formali

- Negli anni '50, N. Chomsky, linguista americano cerca di descrivere la sintassi del linguaggio naturale secondo semplici **regole di riscrittura e trasformazione**.
- Chomsky considera alcune **restrizioni** sulle regole sintattiche e classifica i linguaggi in base alle restrizioni imposte alle regole che generano tali linguaggi.
- 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

- 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.

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

- ☐ i) La stringa $()$ è ben formata;
- ☐ ii) se la stringa di simboli A è ben formata, allora lo è anche (A) ;
- ☐ iii) se le stringhe A e B sono ben formate, allora lo è anche la stringa AB .

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:

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

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

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

- 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 (\quad) o con (S) o con SS ”.

Linguaggio delle parentesi ben formate

- 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. 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.

Linguaggio delle parentesi ben formate

■ Esempio: Generazione di $(())(())(())$

□ Primo modo:

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

□ La corrispondente sequenza di applicazione delle produzioni è:

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

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:

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

■ Una sequenza di applicazioni di regole di produzione prende il nome di *derivazione*.

MdT per linguaggio delle parentesi ben formate

- Possiamo costruire una MdT per riconoscere il linguaggio delle parentesi ben formate?
- Stampa 'Y' se la stringa è ben formata, 'N' altrimenti

Notazione

- Nei due esempi precedenti abbiamo fatto uso della notazione $\alpha \Rightarrow_{(n)} \beta$

- L'interpretazione è la seguente: “da α si produce direttamente β per effetto dell'applicazione della regola di riscrittura n ”. Ad esempio

$$SS \Rightarrow_{(2)} (S)S$$

- 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:

$$S \Rightarrow^* (())(())(())$$

che leggiamo come “ 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:

$$\begin{array}{ccc} S & & \langle S \rangle \\ \rightarrow & & ::= \end{array}$$

Riferimenti

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