Lezione 1 – Introduzione

Programmazione

Modulo 1 - Fondamenti di Programmazione

Unità didattica 1 – Premesse ed origini

Marco Anisetti

Università degli Studi di Milano - SSRI - CdL Online

Obiettivi del corso(1)

Corso introduttivo alla programmazione

Nella sua carriera professionale un informatico potrebbe non aver mai a che fare direttamente con la programmazione ma...

- La sua conoscenza è di fondamentale importanza dato che tutti gli strumenti con i quali interagirà sono di fatto dei software

Costituisce la differenza tra un utilizzatore del computer (seppur professionista) e un informatico

Obiettivi del corso(2)

Fornire gli strumenti di base per affrontare la programmazione

Fornire gli strumenti utili per apprendere qualsiasi linguaggio di programmazione

Approfondimenti sul il paradigma di programmazione imperativo e il linguaggio C

Approfondimenti sul paradigma di programmazione ad oggetti con elementi di programmazione Java

Materiale consigliato

D. E. Knuth. **The Art of Computer Programming**, vol. 1: Fundamental algorithms. Addison-Wesley, Reading (MA), 1997.

Brian W. Kernighan, Dennis M. Ritchie: **The C Programming Language** 1973

Ravi Sethi. **Programming Languages**. Addison-Wesley 1996.

G. Pighizzini M. Ferrari. **Dai fondamenti agli oggetti Corso di programmazione JAVA**. Terza Edizione Pearson Addison-Wesley Febbraio 2008

Altro materiale di dettaglio verrà indicato durante il corso

Prefazione

Due approcci possibili:

- **Idealistico**: Presentare la materia già organizzata alla luce dei moderni sviluppi (OO, pattern, ecc.)
- **Evoluzionistico**: Presentare la materia come evoluzione storica fino ad arrivare ai suoi recenti sviluppi

Agli albori dell'Informatica la programmazione poteva essere considerata un'arte

Verranno presentati argomenti che saranno approfonditi in corsi come Algoritmi e linguaggi di programmazione.

Calcolatori programmabili(1)

Macchine programmabili per fare calcoli

Primo approccio storico: un modello di calcolatore diverso per ciascuno dei calcoli che erano di volta in volta richiesti

Molto presto si intraprese a progettare dei calcolatori general purpose

Al crescere della complessità degli elaboratori, il compito di programmarli iniziò a superare in complessità addirittura il compito di progettarli

Programmare

Scrivere il programma che la macchina dovrà eseguire

- Sequenza ordinata di istruzioni da compiere

Il programma deve permettere di risolvere un problema

Chi scrive il programma sa risolvere un problema elencando la sequenza di cose da fare

Il programma è scritto per un esecutore che lo comprende

- oppure viene tradotto per l'esecutore

Il programmatore conosce bene il suo esecutore

Lezione 2 – Il ruolo della programmazione oggi

Programmazione

Modulo 1 - Fondamenti di Programmazione

Unità didattica 1 – Premesse ed origini

Marco Anisetti

Università degli Studi di Milano - SSRI - CdL Online

Il programmatore oggi(1)

Il programmatore degli albori

- si occupava di tutto, soluzione, programmazione, uso

Indosseremo i panni di questo programmatore

Ad oggi la programmazione è una parte del processo di sviluppo di un software

Un lavoro di team strutturato in vari modi

- Processo Waterfall
- Metodologia Agile

Il programmatore oggi(2)

Il programmatore non è più visto come un factotum, ma una parte di un team

Il programmatore è diventato uno specialista verticale

- L’esperto di codifica dell’algoritmo

Purtroppo questa visione non è completamente corretta

- Si dovrebbe ancora parlare di **Analista e Programmatore**

Il programmatore oggi(3)

La programmazione è ancora un’arte!

La componenti di inventiva: ricerca della soluzione, ricerca del miglior programma (implementazione)

SDK, tool di sviluppo ed **AI** (es. copilot) automatizzano gran parte del lavoro ma servono fortissime competenze di sviluppo

Le competenze di programmazione sono sempre fortemente necessarie

- Chi pensa alla soluzione non può ignorare come verrà tradotta in programma

Copilot

scrivimi un programma C che moltiplica due matrici

Copilot

Ecco un semplice programma in linguaggio C che moltiplica due matrici. Il codice assume che le matrici siano compatibili per la moltiplicazione (cioè il numero di colonne della prima matrice è uguale al numero di righe della seconda).

C

```c
#include <stdio.h>

#define ROW1 2  // Numero di righe della prima matrice
#define COL1 3  // Numero di colonne della prima matrice
#define ROW2 3  // Numero di righe della seconda matrice
#define COL2 2  // Numero di colonne della seconda matrice

int main() {
    int A[ROW1][COL1] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    int B[ROW2][COL2] = {
        {7, 8},
```

Mostra più linee

Adatta il programma per input da tastiera

Fai una versione con dimensioni dinamiche

Il programmatore domani

Le nuove tecniche di sviluppo software ridanno la centralità alla **trasversalità delle competenze**

Basta Silos di competenze del Waterfall ma processi DevOps

- La sicurezza ha la sua parte in questi processi (DevSecOps)

Automazione nel processo di sviluppo dove possibile e dove aiuta

AI a supporto della programmazione

- Test e analisi del codice
- Sviluppo di programmi

Lezione 1 – Formalizzazione di un Linguaggio

Programmazione

Modulo 1 - Fondamenti di Programmazione

Unità didattica 3 – Macchine a Stati

Marco Anisetti

Università degli Studi di Milano - SSRI - CdL Online

Linguaggio: definizione informale

Un repertorio di segni convenzionali e di regole per combinarli in enunciati più complessi

Un insieme di regole che permettano di associare un significato a ciascun enunciato.

Si distinguono 3 livelli

- **Sintattico**: regole che specificano in quali modi i segni possano essere combinati per formare enunciati;
- **Semantico**: regole che permettono di associare a ciascun segno e a ciascun enunciato il loro significato;
- **Pragmatico**: implicazioni pratiche e le conseguenze di un enunciato.

Alfabeto e stringhe(1)

Un alfabeto è un insieme (finito) di elementi chiamati lettere o simboli.

- Un esempio molto semplice di alfabeto è l'alfabeto binario $\Sigma = \{0, 1\}$

Una stringa *s* su un alfabeto è una sequenza finita di simboli appartenenti all' alfabeto

- Ad Esempio $s = 001101$ è una stringa sull'alfabeto binario $\Sigma = \{0, 1\}$

Alfabeto e stringhe(2)

Nozione di lunghezza di una stringa *s* definita come cardinalità della stessa |*s*|

- Es |001101| = 6

Nozione di stringa vuota ϵ ovvero la stringa con nessun simbolo

- $|\epsilon| = 0$

Operazioni sulle stringhe

Kleene Star: Dato alfabeto Σ la chiusura di Kleene Σ^* è l' insieme di tutte le possibili stringhe su Σ.

- Esempio consideriamo $\Sigma = \{0, 1\}$ allora
- $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 000, \cdots\}$

Concatenazione: Consideriamo due stringhe *a* e *b* la loro concatenazione è la stringa *ab* (esistono anche notazioni esponenziali per concatenazioni multiple tipo a^n)

Sottostringa: Consideriamo la stringa s, una sottostringa di s è una qualsiasi parte di simboli consecutivi della stringa s

L'insieme di tutte le stringhe che si possono ottenere da Σ tranne la stringhe vuota ϵ viene indicato come Σ^+

Linguaggio formale

Un linguaggio formale è un insieme di parole su un alfabeto (non sono generalmente degli insiemi finiti)

- Linguaggio L è un sottoinsieme delle parole costruibili su un alfabeto Σ, $L \subseteq \Sigma^*$.
- **Simboli** o **token**: sono gli atomi di cui è costituito un enunciato in un linguaggio
- **Parola** di un alfabeto è una stringa di quello stesso alfabeto
- Σ^* denota l'insieme di tutte le parole composte da elementi di Σ, compresa ϵ

Il linguaggio non contenente alcuna parola viene detto linguaggio vuoto

Esempi di linguaggi

Italiano: parole sull'alfabeto {*a, b, . . . , z*} elencate nel vocabolario. Linguaggio finito {*a, abaco, . . . , zuppo*}

Espressioni aritmetiche: parole sull'alfabeto {0, 1, . . . , 9, (,), +, ∗} secondo delle regole ben precise

- ((10 + 4) ∗ 3) è una parola in *L*,
- (10 + 4(non è una parola in *L*

Operazioni sui linguaggi

Cardinalità: numero di stringhe che lo compongono

Concatenazione: linguaggio $L_c = L_1 L_2$

Potenza n-esima L^n :insieme di stringhe ottenute concatenando *n* stringhe di *L*, con possibili ripetizioni.

- Se $L = \{a, bb\}$
- $S^0 = \{\epsilon\}$ $S^1 = \{a, bb\}$ $S^2 = \{aa, abb, bba, bbbb\}$
- $S^3 = \{aaa, aabb, abba, abbbb, bbaa, bbabb, bbbba, bbbbbb\}$

Chiusura L^*: linguaggio costituito dall'unione di tutte le n-esime potenze di *L*

Chiusura positiva L^+: Chiusura con $n > 0$

Codice

Codice: Un linguaggio *L* in cui ogni parola in L^+ è decomponibile in un unico modo come prodotto di parole di *L*.

- **Proprietà:** univocamente decifrabile
- Data una parola in L^+ esiste un solo modo di ottenerla come prodotto di parole di *L*.
- $C = \{1, 10\}$ è un codice
- $C' = \{bab, aba, ab\}$ non è un codice
- Il codice ASCII.

Un linguaggio *L* in cui ogni parola non è prefisso di nessun altra parola

Linguaggio formato da parole di ugual lunghezza

Codice

Codice: Un linguaggio *L* in cui ogni parola in L^+ è decomponibile in un unico modo come prodotto di parole di *L*.

- **Proprietà:** univocamente decifrabile
- Data una parola in L^+ esiste un solo modo di ottenerla come prodotto di parole di *L*.
- $C = \{1, 10\}$ è un codice
- $C' = \{bab, aba, ab\}$ non è un codice (ababab -> aba+bab -> ab+ab+ab)
- Il codice ASCII.

Un linguaggio *L* in cui ogni parola non è prefisso di nessun altra parola

Linguaggio formato da parole di ugual lunghezza

Appartenenza ad un linguaggio

Data una parola (stringa o frase) w ∈ Σ*, ci sono due possibilità:

- w appartiene al linguaggio (w ∈ L) cioè rappresenta un enunciato di L
- w non appartiene al linguaggio (w ∉ L) cioè non rappresenta un enunciato valido di L.

***L* contiene un numero infinito di parole cioè di possibili enunciati**

Appartenenza ad un linguaggio

Data una parola (stringa o frase) w ∈ Σ*, ci sono due possibilità:

- w appartiene al linguaggio (w ∈ L) cioè rappresenta un enunciato di L
- w non appartiene al linguaggio (w ∉ L) cioè non rappresenta un enunciato valido di L.

***L* contiene un numero infinito di parole cioè di possibili enunciati**

Esempio: dire se un indirizzo email appartiene al linguaggio degli indirizzi email validi

Appartenenza ad un linguaggio

Data una parola (stringa o frase) w ∈ Σ*, ci sono due possibilità:

- w appartiene al linguaggio (w ∈ L) cioè rappresenta un enunciato di L
- w non appartiene al linguaggio (w ∉ L) cioè non rappresenta un enunciato valido di L.

***L* contiene un numero infinito di parole cioè di possibili enunciati**

Esempio: dire se un indirizzo email appartiene al linguaggio degli indirizzi email validi (validi non esistenti!)

Riconoscitori e generatori(1)

Approccio riconoscitivo: rappresentare L attraverso la definizione di un algoritmo che, per ogni parola w ∈ Σ*, termina con il risultato "si" se w ∈ L e "no" altrimenti.

- Non tutti i linguaggi ammettono un riconoscitore
- Quelli che lo ammettono sono detti ricorsivi o decidibili

Automi a stati finiti: famiglia di riconoscitori (riconoscitori di linguaggi regolari)

Riconoscitori e generatori(2)

Approccio generativo: consiste nel definire una procedura in grado di generare sistematicamente tutte le parole *w* di *L*, una dopo l'altra.

- Quindi se riesco a generare w' da L allora w' appartiene a L

Grammatiche: Un'importantissima classe di sistemi generativi, che sono di fatto impiegati per descrivere in modo finito linguaggi infiniti.

Importanza dei linguaggi formali

Ci interessiamo a linguaggi infiniti le cui parole sono:

- Definite su un particolare alfabeto accettato dall'elaboratore
  - Es. {a, b,..., z, A, B,..., Z, 0, 1,..., 9, >, <, =, +, ?, /, (,),...}
- Direttamente o indirettamente comprensibili all'elaboratore
  - Es. direttamente se fossero in linguaggio macchina (alfabeto di 0 e 1), indirettamente se fossero in forma mnemonica

Studio delle proprietà sintattiche dei programmi

- Definizione della sintassi
- Verifica delle proprietà sintattiche
- Traduzione da un linguaggio ad un altro

Importanza dei linguaggi formali

Ci interessiamo a linguaggi infiniti le cui parole sono:

- Definite su un particolare alfabeto accettato dall'elaboratore
  - Es. {a, b,..., z, A, B,..., Z, 0, 1,..., 9, >, <, =, +, ?, /, (,),...}
- Direttamente o indirettamente comprensibili all'elaboratore
  - Es. direttamente se fossero in linguaggio macchina (alfabeto di 0 e 1), indirettamente se fossero in forma mnemonica

Studio delle proprietà sintattiche dei programmi

- Definizione della sintassi
- Verifica delle proprietà sintattiche
- Traduzione da un linguaggio ad un altro (per far eseguire un programma lo si traduce)

Lezione 2 – Automi a Stati Finiti

Programmazione

Modulo 1 - Fondamenti di Programmazione

Unità didattica 3 – Macchine a Stati

Marco Anisetti

Università degli Studi di Milano - SSRI - CdL Online

Automi a stati finiti

Un automa a stati finiti è un modello matematico di un sistema con ingressi e uscite discreti

In un certo istante il sistema può trovarsi in uno stato scelto tra un numero finito di stati possibili

Lo stato corrente del sistema riassume l'informazione circa la sequenza di ingresso fornita in passato e serve per determinare il comportamento del sistema al successivo ingresso (stato futuro)

Il passaggio tra stati è detto transizione

Automa Completamente Specificato: una transizione per ogni stato e per ogni input

Definizione formale

Formalmente, un automa finito *M* su un alfabeto Σ è una quintupla $\langle K, \Sigma, \delta, q_0, F \rangle$

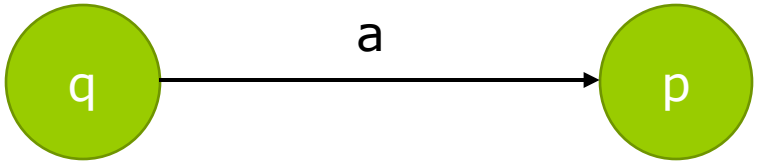
- K è un insieme finito e non vuoto di stati in cui si può trovare M
- Σ è un alfabeto finito di simboli di ingresso,
- $\delta : K \times \Sigma \rightarrow K$ è la funzione di transizione di stato
- $q_0 \in K$ è lo stato iniziale
- $F \subseteq K$ è l'insieme degli stati finali (1 o più di uno)

Rappresentazione grafica

Un automa può essere convenientemente rappresentato in forma tabulare, mediante la sua funzione δ, o in forma grafica

La forma grafica utilizza un grafo orientato detto grafo delle transizioni

Se c'è una transizione dallo stato *q* allo stato *p* ricevuto l'ingresso *a*, allora c'è un arco da *q* a *p* etichettato con *a*



Riconoscitore di un linguaggio

Riconoscitore di un linguaggio come una black box:

- *ingresso:* parole, costruite su un alfabeto finito $S = \{s_1, s_2, ..., s_n\}$
- *uscita:* un valore in $\{0, 1\}$

Il funzionamento consiste nella sua esecuzione:

- Riceve in ingresso una serie di parole $w \in S^*$
- Ritorna in uscita 1 se la parola è accettata, 0 se respinta

Riconosce un linguaggio definito su S^*

- Accetta le parole appartenenti al linguaggio

Sistema a Stati

Il comportamento della black box è descritto dall'insieme di parole accettate

- Per le quali l'output è 1

Si modella attraverso un automa a stati

Esempio man mano che leggo una lettera della parola data in ingresso potrei cambiare di stato

Un esempio intuitivo

Esempio macchina per l'erogazione di bibite/snack da 50 centesimi

- Accetta monete da 10c e da 20c,

Un esempio intuitivo

Esempio macchina per l'erogazione di bibite/snack da 50 centesimi

- Accetta monete da 10c e da 20c, ovvero alfabeto finito
  S = {10c, 20c}

Un esempio intuitivo

Esempio macchina per l'erogazione di bibite/snack da 50 centesimi

- Accetta monete da 10c e da 20c, ovvero alfabeto finito
  S = {10c, 20c}
- Non da resto
- Rifiuta monete che portano a superare i 50 centesimi

Un esempio intuitivo

Esempio macchina per l'erogazione di bibite/snack da 50 centesimi

- Accetta monete da 10c e da 20c, ovvero alfabeto finito
  S = {10c, 20c}
- Non da resto
- Rifiuta monete che portano a superare i 50 centesimi

Si tratta di una macchina che riconosce un linguaggio fatto da parole su S per cui si arriva alla erogazione di una bibita

- Eroga la bibita quando la parola è accettata

Un esempio intuitivo

Esempio macchina per l'erogazione di bibite/snack da 50 centesimi

- Accetta monete da 10c e da 20c, ovvero alfabeto finito
  S = {10c, 20c}
- Non da resto
- Rifiuta monete che portano a superare i 50 centesimi

Si tratta di una macchina che riconosce un linguaggio fatto da parole su S per cui si arriva alla erogazione di una bibita

- Eroga la bibita quando la parola è accettata
- Esempio:
  - 20c 20c 10c è una parola accettata
  - 10c 20c non è accettata

Un esempio intuitivo

Esempio macchina per l'erogazione di bibite/snack da 50 centesimi

- Accetta monete da 10c e da 20c, ovvero alfabeto finito
  S = {10c, 20c}
- Non da resto
- Rifiuta monete che portano a superare i 50 centesimi

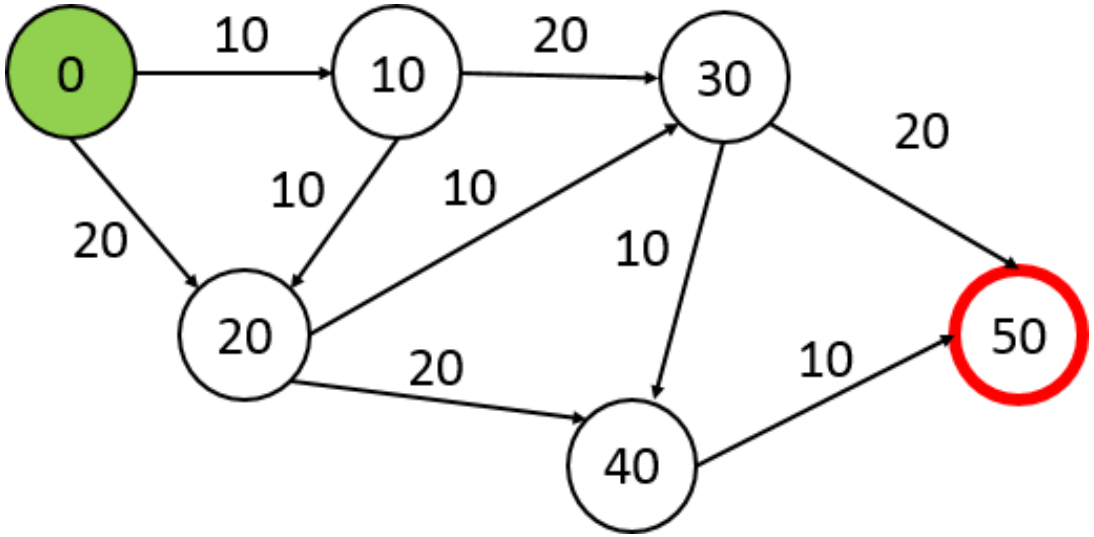
Per intuito potremmo pensare di

- Elencare tutte le parole accettate
  - Quindi cercare tra quelle accettate
- Descriverlo come un automa a stati

Un esempio intuitivo

Esempio macchina per l'erogazione di bibite/snack da 50 centesimi

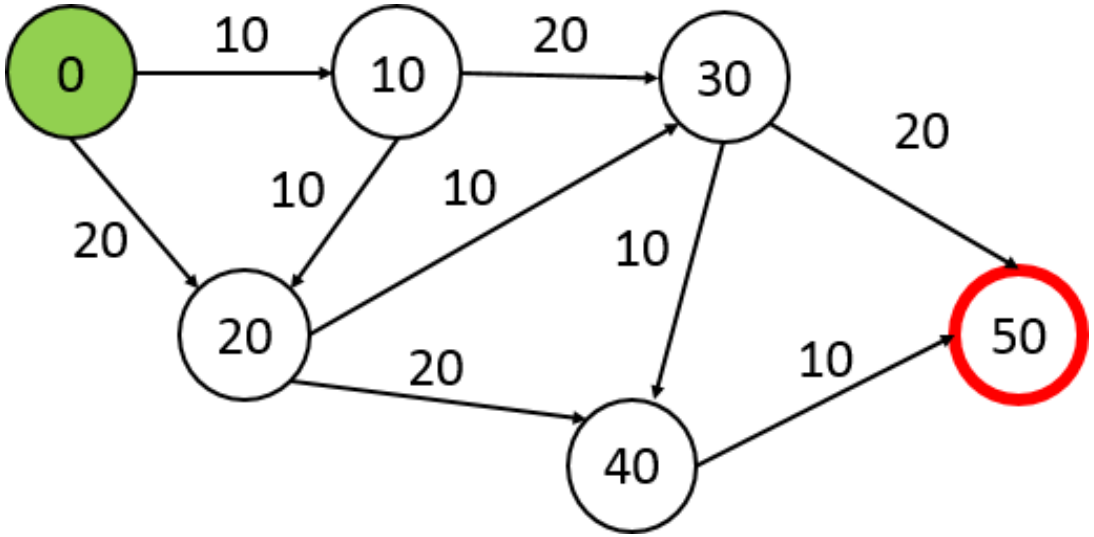
- Accetta monete da 10c e da 20c, ovvero alfabeto finito S = {10c, 20c}
- Non da resto
- Rifiuta monete che portano a superare i 50 centesimi



Un esempio intuitivo

Esempio macchina per l'erogazione di bibite/snack da 50 centesimi

- Accetta monete da 10c e da 20c, ovvero alfabeto finito
  S = {10c, 20c}
- Non da resto
- Rifiuta monete che portano a superare i 50 centesimi



E' completamente Specificato?

Enigma del barcarolo

Un barcarolo doveva trasportare sull' altra riva del fiume un lupo, una pecora e un cavolo, ma non trovò che una barca capace di portarne due alla volta (egli stesso più un altro).

Gli era stato ordinato di non procurare alcun danno né agli animali né alla pianta

- Non può lasciare soli cavolo e pecora o pecora e lupo

(Tratto dalle "Propositiones ad acuendos juvenes" di Alcuino)

Soluzione intuitiva

Prima trasporta la pecora, lasciando a terra il lupo e il
Cavolfiore; poi torno indietro e porta sull’altra riva il lupo.

Giunto sull’altra riva, fa scendere il lupo, carica in barca la pecora e la riporto indietro.

Fa scendere dalla barca la pecora, prende il cavolfiore e lo porta sull’altra riva, quindi ritorna con la barca vuota ed infine trasporto la pecora.

Soluzione intuitiva

Prima trasporta la pecora, lasciando a terra il lupo e il Cavolfiore; poi torno indietro e porta sull'altra riva il lupo.

Giunto sull'altra riva, fa scendere il lupo, carica in barca la pecora e la riporto indietro.

Fa scendere dalla barca la pecora, prende il cavolfiore e lo porta sull'altra riva, quindi ritorna con la barca vuota ed infine trasporto la pecora.

Una soluzione più formale?

Codifica come parola di un linguaggio

Consideriamo le seguenti mosse con la relativa codifica in caratteri

- Attraversa con lupo (l)
- Attraversa con pecora (p)
- Attraversa con cavolofiore(c)
- Attraversa solo (b)

La parola che identifica una soluzione?

Codifica come parola di un linguaggio

Consideriamo le seguenti mosse con la relativa codifica in caratteri

- Attraversa con lupo (l)
- Attraversa con pecora (p)
- Attraversa con cavolofiore(c)
- Attraversa solo (b)

La parola che identifica una soluzione: pblpcbp

In gergo informatico questa parola è una stringa (sequenza ordinata di caratteri)

Modellazione della soluzione(1)

Come possiamo modellare questa soluzione in termini di riconoscimento di una parola con un automa?

- Alfabeto Σ={l, p, c, b}

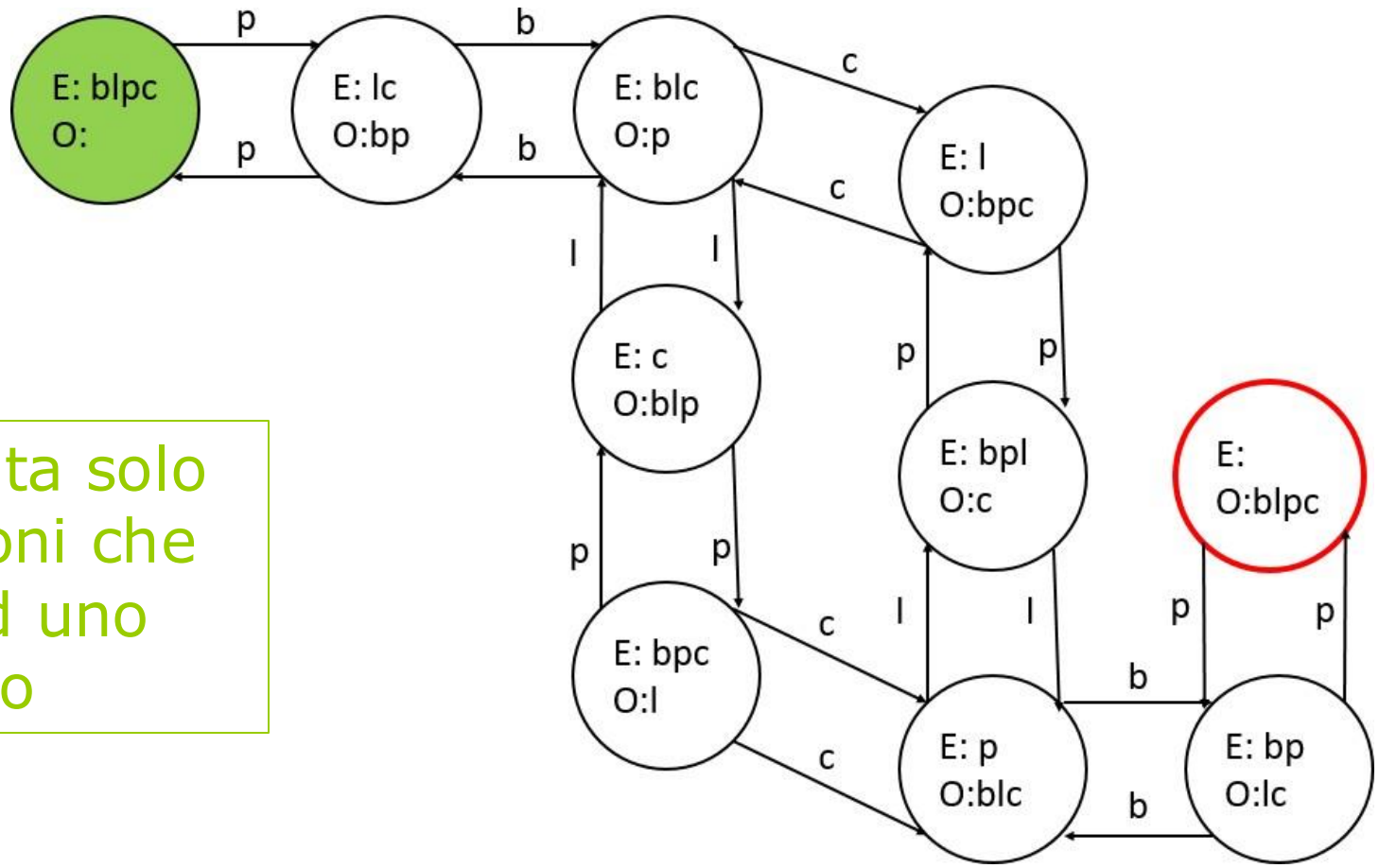
Consideriamo le presenze sulle due sponde Est (E) e Ovest (O) come lo stato del sistema

- Dobbiamo considerare anche la presenza del barcarolo sulle sponde
- Es.

E:bplc O: è lo stato iniziale tutti su sponda est (E)

Le transizioni da uno stato all'altro sono gli spostamenti (lettere dell'alfabeto)

Modellazione della soluzione(2)



Rappresenta solo le transizioni che portano ad uno stato valido

Linguaggio delle soluzioni

Il grafo dell'automa definisce il linguaggio delle soluzioni

- $\{x \in \{l, p, c, b\}^* |$ *iniziando in uno stato iniziale e seguendo gli archi definiti da x si arriva nello stato finale*$\}$

Il linguaggio è infinito

Procedura computazionale per decidere se una stringa è soluzione.

- Genera un percorso dallo stato iniziale allo stato finale.

Automi a Stati Finiti (ASF): proprietà

Dinamicità: evoluzione attraverso diversi stati

Discretezza: le variabili d'ingresso e gli stati del sistema possono essere espressi con valori discreti

Simboli finiti: il numero di simboli di ingresso e di stati è rappresentabile da un numero finito

Automi a Stati Finiti (ASF): tipologie

Automi a stati finiti deterministici (ASFD)

- per qualsiasi input, esisterà una ed una sola transizione

Automi a stati finiti non deterministici (ASFND)

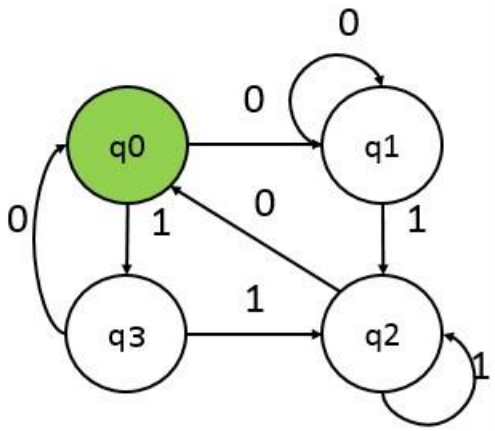
- almeno uno stato presenta più di una possibile computazione per determinati valori in ingresso (più stati raggiungibili con un input).

Il determinismo è un caso particolare di non determinismo

ASF: esempio

Un esempio di automa deterministico con la relativa tabella di transizione

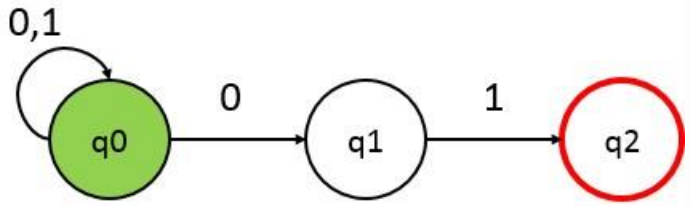
Stato	Input	Stato prossimo
q0	0	q1
q0	1	q3
q1	0	q1
q1	1	q2
q2	0	q0
q2	1	q2
q3	0	q0
q3	1	q2



ASFND: esempio

Un esempio di automa non deterministico con la relativa tabella di transizione

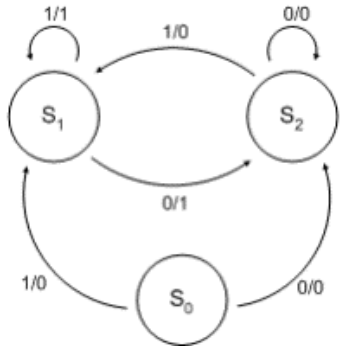
Stato	Input	Stato prossimo
q0	0	{q0, q1}
q0	1	q0
q1	0	
q1	1	q2
q2	0	
q2	1	



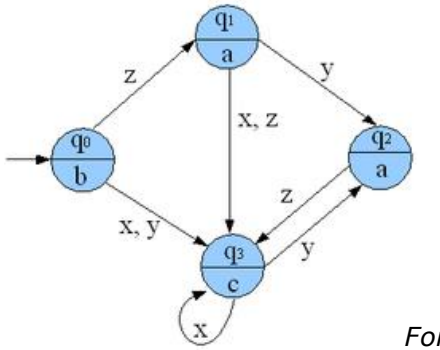
ASF con output

Macchine a stati che generano output

- **macchina di Mealy:** automa a stati finiti i cui valori di uscita sono determinati dallo stato attuale e dall'ingresso corrente



- **macchina di Moore**: automa a stati finiti le cui uscite sono solo in funzione dello stato corrente



Fonti wikipedia

Automi a stati finiti: esempi software

Applicazioni:

- Firmware: dispenser automatici, biglietterie , driver
- Compilatori: analizzatore lessicale, parser (riconosce linguaggi)
- Progettazione circuiti digitali
- Protocolli, sistemi biologici
- Elaboratori di testo
- Ricerche web o su file
- Videogiochi

Lezione 3 – Macchina di Turing

Programmazione

Modulo 1 - Fondamenti di Programmazione

Unità didattica 3 – Macchine a Stati

Marco Anisetti

Università degli Studi di Milano - SSRI - CdL Online

Alan Mathison Turing

Alan Mathison Turing (1912 - 1954) matematico, logico e crittanalista britannico, considerato uno dei padri dell'informatica e uno dei più grandi matematici del Novecento.

Fu uno dei più brillanti decrittatori che operavano in Inghilterra, durante la seconda guerra mondiale.

Decifrò Enigma, il codice usato dai sottomarini tedeschi. Morì mangiando una mela al cianuro, in seguito ad una persecuzione omofobica condotta nei suoi confronti.

Macchina di Turing (MdT)

Modello fondamentale per la teoria dell'informatica

- Si basa sugli automi a stati finiti

Permette di analizzare e definire il concetto di algoritmo

Ha permesso di ottenere risultati negli ambiti di

- Calcolabilità
- Complessità
- Equivalenza di algoritmi

MdT: Definizione informale

La Macchina di Turing (MdT) è una macchina (basata su un automa a stati finiti) avente nastro potenzialmente infinito

- L'automa definisce quello che si chiama **dispositivo di controllo**
- Il nastro è formato da una sequenza di celle

Sul nastro agisce una testina

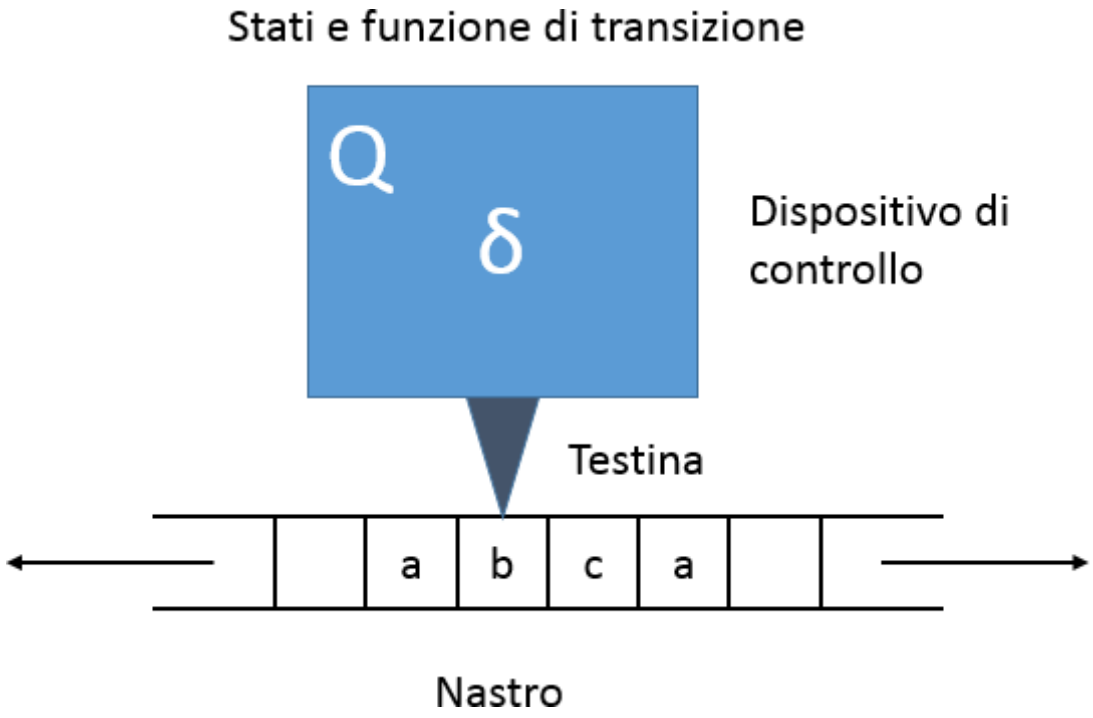
La testina può muoversi in entrambe le direzioni sul nastro.

Può leggere e scrivere in ogni cella del nastro

- un alfabeto predefinito

Le celle possono essere vuote

MdT: Definizione informale



Esistono delle varianti della MdT: multinastro, non deterministiche ecc.

MdT: Operazioni

La testina di lettura-scrittura:

- Spostamento di una posizione (a sinistra o a destra).
- Scrittura o lettura di un simbolo nella cella sotto la testina (sostituendo il contenuto precedente).

Il dispositivo di controllo in ogni istante si trova in uno stato tra un insieme finito di stati

A seconda dello stato e del simbolo letto

- Cambia stato
- Esegue una azione

MdT: determinazione

Contenuto iniziale del nastro

Posizione iniziale della testina

Stato iniziale del dispositivo di controllo

- Automa → Tabella delle transizioni di stato

MdT: Elaborazione

Elaborazione $y = F(x)$

- x nastro allo stato iniziale
- y nastro nello stato finale
- F=d(MdT): posizione iniziale, stato iniziale, tabella di transizione

MdT: Definizione Formale

Tabella delle transizione di stato

Una quintupla di elementi:

- *s*: lo stato della macchina all'istante presente
- *i*: il simbolo letto all'istante presente
- *S(s,i)*: lo stato della macchina all'istante successivo
- *I(s,i)*: il simbolo scritto dalla macchina all'istante successivo
- *V(s,i)*: il verso del movimento della macchina (destra o sinistra)

MdT: Definizione Formale

Tabella delle transizione di stato

Una quintupla di elementi:

- *s*: lo stato della macchina all'istante presente
- *i*: il simbolo letto all'istante presente
- *S(s,i)*: lo stato della macchina all'istante successivo
- *I(s,i)*: il simbolo scritto dalla macchina all'istante successivo
- *V(s,i)*: il verso del movimento della macchina (destra o sinistra)

Ci viene da pensare ad un automa con output dato che scrive sul nastro

MdT: esempio

Costruiamo una macchina che valuti se il numero di occorrenze del simbolo 1 in una sequenza di 0 e 1 è pari.

- Leggendola dal nastro

Consideriamo che la sequenza debba iniziare e terminare con il simbolo «?»

Numero di 1 pari: la macchina scrive 1 nella prima cella a destra della stringa di ingresso

Numero di 1 dispari: la macchina scrive 0 nella prima cella a destra della stringa di ingresso

MdT: esempio

Pensiamo ad una macchina a stati che svolga il compito richiesto

Alfabeto: {1,0,?}

Stati:

- Stato iniziale (S)
- Stato Pari (P)
- Stato Dispari (D)
- Stato di Terminazione (T)

MdT: esempio

Passa da pari a dispari leggendo 1

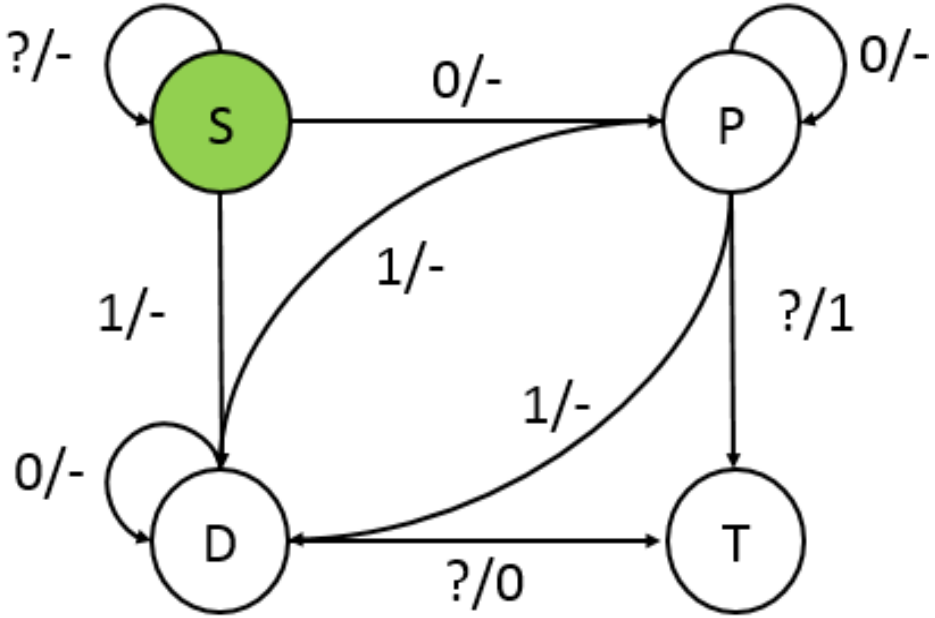
Leggendo 0 tranne la prima volta resta nello stato

- Se leggo 0 la prima volta ho letto zero numeri 1 e quindi è pari

MdT: esempio

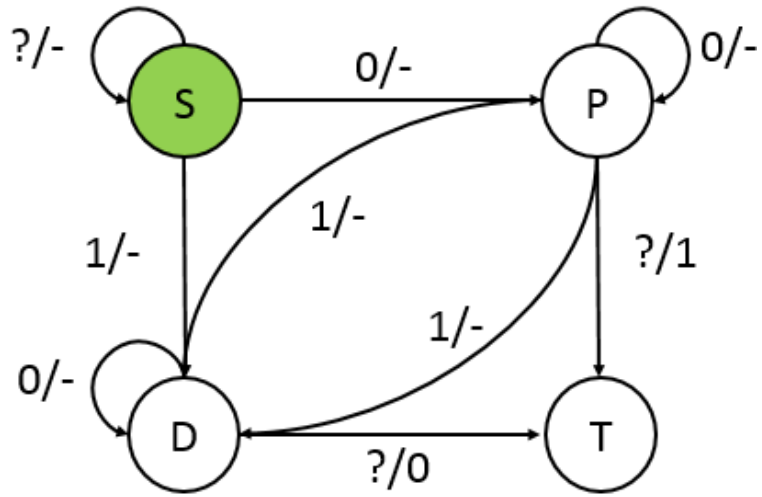
Passa da pari a dispari leggendo 1

Leggendo 0 tranne la prima volta resta nello stato



MdT: esempio

Quintuple MdT				
s	**i**	**I(s, i)**	**V(s, i)**	**S(s, i)**
S	?	-	>	S
S	1	-	>	D
S	0	-	>	P
P	?	1	Termina	T
P	1	-	>	D
P	0	-	>	P
D	?	0	Termina	T
D	0	-	>	D
D	1	-	>	P
T	-	-	Termina	E



Discussione(1)

Una prima esperienza di programmazione.

Linguaggio direttamente comprensibile all'esecutore (le quintuple)

La macchina che esegue il programma non è una macchina reale ma una macchina teorica

- **Per ogni particolare istanza del problema abbiamo una macchina di Turing differente**
- **Il programma è codificato nella automa**

Da notare che il linguaggio che usiamo per descrivere la soluzione è nella pratica un linguaggio formale (in particolare un **codice**)

Discussione(2)

Una prima intuitiva nozione di Algoritmo

Algoritmo: procedura per risolvere «meccanicamente» una classe di problemi predefiniti

La MdT è un esecutore di algoritmi, anzi è una macchina teorica in grado di eseguire qualsiasi algoritmo

Utilizzi

Algoritmo: una MdT che si arresti trasformando il nastro da t a t' rappresenta l'elaborazione per $Y = F(X)$

Calcolabilità: secondo la tesi di Church non esiste un formalismo né una macchina concreta che possa calcolare una funzione F non calcolabile secondo Turing

- Modello teorico di macchina potente
- Si può usare per verificare la potenza di altri modelli teorici
- Esistono problemi **non Turing risolubili**
  - es. il famoso **problema dell'arresto**

MdT Universale

MdTU: MdT in grado di imitare una qualsiasi particolare macchina di Turing

Primo modello di calcolatore programmabile

- MdT normali eseguono un solo programma, che è ”scritto” nella tabella delle quintuple

Gli odierni computer sono in un certo senso delle MdTU

Turing ha dimostrato che la MdT Universale non è in grado di decidere il problema dell’arresto (Turing-indecidibile)

- Questo identifica un limite per tutti i meccanismi computazionali

MdT Universale: funzionamento(1)

Un MdTU è una MdT che:

- Riceve la codifica di una MdT M e una stringa I $\in \Sigma$
- Simula i passi che la macchina M compirebbe se ricevesse la stringa I
- Restituisce la stessa risposta (e lo stesso contenuto sul nastro)

MdTU assume in input il programma che deve eseguire

- La codifica di M

MdT Universale: funzionamento(2)

La MdTU tratta i programmi (M) e i dati (I) in maniera sostanzialmente analoga

- Memorizzati sullo stesso supporto
- Rappresentati utilizzando lo stesso alfabeto di simboli
- Elaborati in modo simile

Lezione 4 – Architettura di un elaboratore

Programmazione

Modulo 1 - Fondamenti di Programmazione

Unità didattica 3 – Macchine a Stati

Marco Anisetti

Università degli Studi di Milano - SSRI - CdL Online

La Macchina di Von Neumann (1946)

La macchina di Von Neumann costituisce l'architettura della maggior parte dei Calcolatori

John Von Neumann realizzò ENIAC (Electronic Numerical Integrator and Computer)

ENIAC era in grado di effettuare 300 moltiplicazioni al secondo, ed occupava una stanza lunga più di 30 metri

Fino al 1952, l'ENIAC fu il calcolatore più potente del mondo. Poi entrarono in servizio due nuovi calcolatori, EDVAC e ORDVAC, e nel 1955 ENIAC fu spento definitivamente

Componenti fondamentali

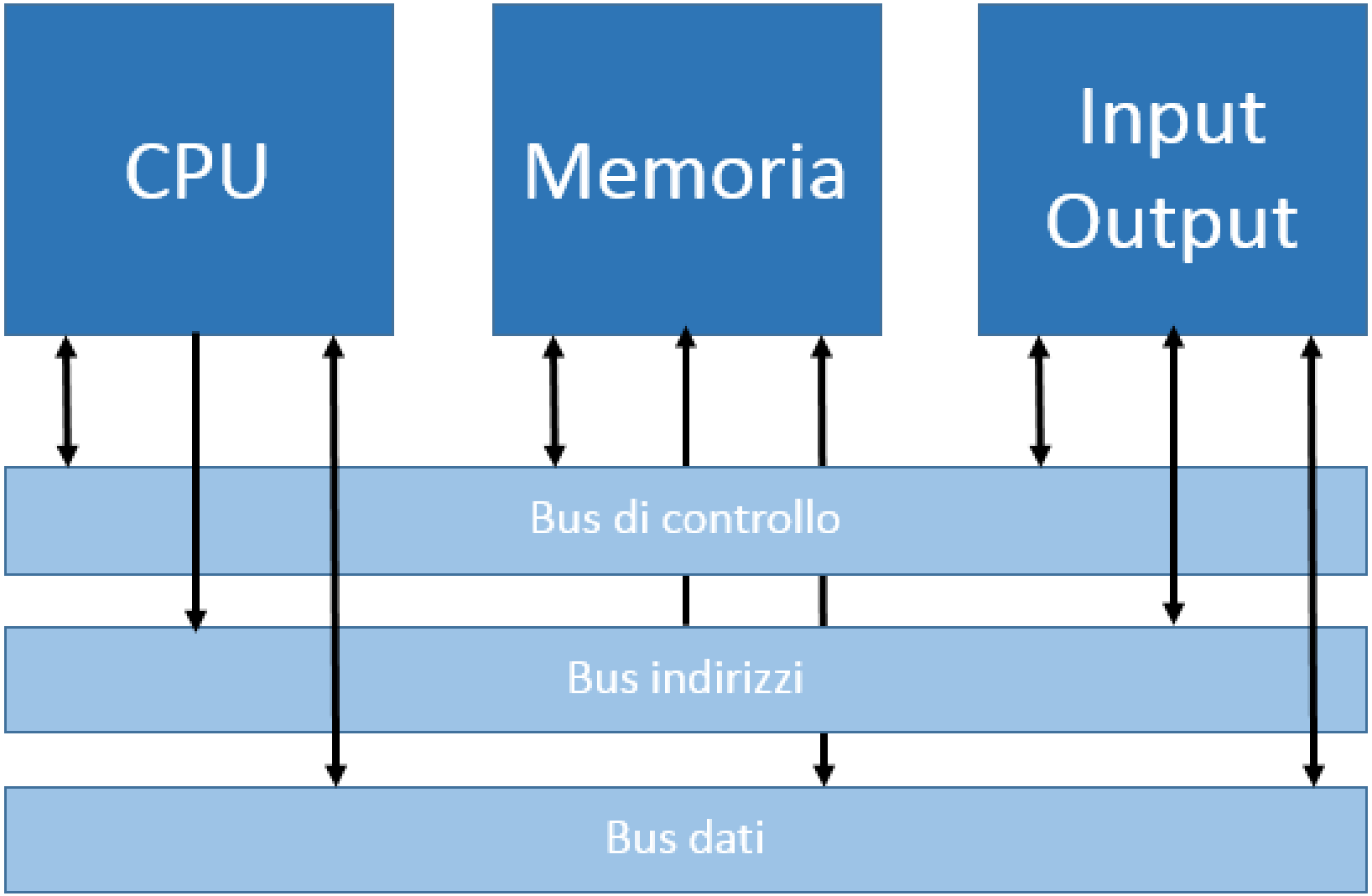
CPU (Central Processing Unit):è in grado di acquisire, interpretare ed eseguire le istruzioni di un Programma

La Memoria Centrale: è il dispositivo dove si trovano le informazioni necessarie all'esecuzione di un programma, ossia istruzioni e dati

Dispositivi di Input/Output o Periferiche: permettono di trasferire informazioni tra memoria centrale e/o CPU e l'ambiente esterno

Bus di sistema: collega tra loro i vari componenti

Architettura



MdT e Architettura di un elaboratore

Un calcolatore secondo l'architettura di von Neumann è in un certo senso la realizzazione concreta di una MdTU

- La memoria dati/programmi può essere considerata l'equivalente del nastro

La potenza computazionale è la medesima

Una macchina di von Neumann è un calcolatore universale

- MdTU è un modello astratto degli attuali elaboratori

Linguaggio e automa a stati

Linguaggio L è un sottoinsieme delle parole costruibili su un alfabeto Σ, $L \subseteq \Sigma^*$

Il sottoinsieme di parole di un linguaggio può essere descritto attraverso un riconoscitore o un generatore

Il riconoscitore è implementato generalmente come automa a stati finiti

- Lo stato dell'automa rappresenta a pieno la sua evoluzione a seguito di una sequenza di input

Automa a stati e MdT

La MdT può essere vista come un modello matematico di algoritmo e si basa su un automa a stati finiti (**dispositivo di controllo**)

Tesi di Church-Turing: **qualsiasi** algoritmo è modellabile con una macchina di Turing

Da cui il corollario che nessuna macchina potrà mai risolvere problemi che una macchina di Turing non possa risolvere (computabilità)

MdT e Architettura di un elaboratore

Una MdT è specifica per un problema

Una MdTU è generica per ogni problema:

- Esegue una MdT specifica indicata come programma

Come l'architettura di Von Neumann

- Programmi e dati memorizzati nello stesso modo
- Carica la descrizione della MdT da eseguire in memoria
  - Crea la sua macchina stati e la esegue