Indice

Prefazione

Il principale intento di questo volume è di proporre, richiamandosi al pensiero di Harold Abelson, un approccio nuovo all'Informatica e al suo insegnamento universitario. Si intende incentivare, infatti, lo studio dell'Informatica secondo un approccio linguistico-formale, anziché fisico-elettronico. Ciò rappresenta un orientamento diverso in confronto alle prime esperienze, internazionali nonché italiane, concernenti la delineazione scientifico-didattica dell'Informatica.

Non a caso, la nascita di tale disciplina in Italia è strettamente legata allo sviluppo di quattro progetti, realizzati contemporaneamente, e in modo indipendente, a partire dal 1954:

- il Centro di Calcoli Numerici presso il Politecnico di Milano, dotato di una CRC 102A (la prima calcolatrice elettronica che entrò in funzione in Italia) per elaborare dati di applicazioni sia industriali che scientifiche;
- la Calcolatrice Elettronica Pisana (C.E.P.), promossa dallo stesso Enrico Fermi e progettata interamente dall'Università di Pisa;
- la FINAC (MARK I) dell'Istituto Nazionale per le Applicazioni del Calcolo del CNR di Roma;
- l'ELEA 9003 dell'Olivetti S.p.A., la prima calcolatrice elettronica commerciale.

I primi informatici italiani sono stati, pertanto, fisici e ingegneri, impegnati essenzialmente nella progettazione e realizzazione di macchine calcolatrici, per opera dei quali è nato in Italia il Corso di Laurea in Scienze dell'Informazione. La loro origine professionale è immediatamente rilevabile nell'impostazione di tipo fisico-elettronico adottata nei libri di testo da essi scritti, ad uso dei primi studenti universitari di questo novello Corso di Laurea. E nel resto del mondo la situazione non è stata tanto differente.

Alla fine degli anni '60, Abelson delinea un'altra concezione dell'Informatica, derivante dalla sua esperienza presso il Massachusetts Institute of Technology di Boston, in cui il *leit motiv* non è più costituito dalla progettazione fisica dei calcolatori, bensì dalla loro programmazione. In tale prospettiva, i programmi devono essere scritti per essere letti da altri, e solo *incidentalmente* per essere eseguiti da macchine.

Si tratta, seguendo Abelson, di riconoscere qualcosa di fondamentale e, per certi aspetti, sorprendente, cioè che l'essenza dell'Informatica "*ha poco a che fare con i computers*". Tali macchine hanno determinato, certamente, una rivoluzione nel

modo di pensare e nel modo di esprimere il pensiero, ma la dimensione delle macchine non esaurisce il significato dell'Informatica. Con l'avvento dei computers, si è avuto un fenomeno epocale e profondamente innovativo, caratterizzato dall'emergere di ciò che potrebbe essere meglio definito "*procedural epistemology*". Questa svolta epistemologica consiste nello studio della struttura della conoscenza da un punto di vista 'imperativo', in opposizione al punto di vista più 'dichiarativo' permeante la Matematica classica. Diversamente dall'Informatica, che fornisce uno strumento per dare in modo rigoroso indicazioni sulla definizione di 'come fare' (*how to*), la Matematica si pone da un angolo visuale non imperativo, in quanto è interessata, piuttosto, a fornire uno strumento per dare in modo rigoroso la definizione di 'che cosa è' (*what is*). Occorre restituire, tuttavia, all'Informatica la sua essenza matematica e linguistico-formale, non soltanto 'imperativa' né esclusivamente fisico-elettronica.

L'attenzione rivolta da Abelson alla programmazione dei calcolatori, più che alla loro progettazione fisica, implica ricercare "sia la perfezione della parte che l'adeguatezza del tutto"[1], nonché lo sviluppo di linguaggi formali in grado di fornire ai computers programmi aventi sempre più ampie potenzialità di *problem solving*, e di simulazione dei procedimenti logici della mente umana. Significativamente, nel citato libro di Abelson, l'uso della parola 'programma' è applicato alla creazione, esecuzione e studio di programmi scritti in linguaggio Lisp, da far eseguire su computers. Utilizzando il Lisp, egli afferma che si circoscrive il mezzo espressivo per la descrizione dei programmi, ma non si limita in alcun modo ciò che è possibile programmare. A tal proposito, qui va soltanto osservato che la scelta del Lisp come linguaggio di descrizione non possa essere apparsa se non rivoluzionaria all'epoca, in un contesto accademico dominato dalla programmazione 'imperativa'.

Il presente volume costituisce un estratto del materiale da noi redatto, e distribuito agli studenti del Corso di Laurea in *Produzione e Gestione di Servizi Informatici,* a supporto dell'insegnamento di *Programmazione*, e copre all'incirca un terzo del programma del corso. Il nostro sforzo è stato quello di compiere un'attività di seria divulgazione a fini scientifico-didattici, nella convinzione che sia sempre più importante realizzare un'adeguata attività divulgativa anche nell'ambito della Matematica e dell'Informatica. Lungo questa linea, può essere interessante ricordare le osservazioni di David Mumford sulla opportunità per tutti gli scienziati e, in particolare, per i matematici, di seguire l'esempio dei fisici, che si sono spesso impegnati nel diffondere e rendere accessibile il loro sapere[2]. Da parte nostra, abbiamo quindi raccolto l'invito di Mumford ad intraprendere il percorso

[1] Cfr. H. Abelson, G.J. Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, MA., 1985.

[2] Cfr. Intervista di P. Oddifreddi a David Mumford, *Matematici*, in "La Repubblica", 19-8-2002, p. 29.

divulgativo da lui stesso sperimentato in ambito matematico[3], e a realizzare opere di divulgazione in cui si cerchi di spiegare un po' di 'vera' scienza nel modo più semplice possibile, che peraltro è qualcosa di estremamente difficile svolgere in modo, al tempo stesso, rigoroso.

In definitiva, abbiamo cercato non solo di introdurre all'Informatica il lettore, sia questi studente o studioso, attraverso un approccio, nello spirito di Abelson, di tipo linguistico-formale, ma anche di volgarizzare, senza banalizzarli, concetti ritenuti spesso astratti e/o troppo avanzati (teoria dei linguaggi formali, classificazione delle grammatiche, ecc.).

É nostra opinione, probabilmente condivisa da molti e affiorante da quanto detto finora, che, tra le discipline scientifiche, l'Informatica sia 'sorella' della Matematica, poiché, al pari di essa, *non vi sono energie in gioco o masse in movimento.* Il calcolatore è visto qui, semplicemente, come uno strumento in grado di trasformare sequenze di simboli in ingresso in altre sequenze di simboli. Ma la lettura del testo non richiede alcuna conoscenza pregressa di matematica o di altre scienze.

Desideriamo esprimere i nostri più sentiti ringraziamenti al Magnifico Rettore dell'Università degli Studi del Molise, Prof. Giovanni Cannata, il cui generoso interessamento e impegno ha permesso la pubblicazione del nostro lavoro.

Vincenzo Acciaro
Michela Granatiero

[3] Cfr. D. Mumford, C. Series, D. Wright, *Indra's Pearls. The Vision of Felix Klein*, Cambridge University Press, New York, 2002: "this is a book about serious mathematics, but one, which we have written primarily for non-mathematicians" (from the *Introduction*).

We may hope that machines will eventually compete with men in all purely intellectual fields. But which are the best ones to start with? Even this is a difficult decision. Many people think that a very abstract activity, like the playing of chess, would be best. It can also be maintained that it is best to provide the machine with the best sense organs that money can buy, and then teach it to understand and speak English. This process could follow the normal teaching of a child. Things would be pointed out and named, etc. Again I do not know what the right answer is, but I think both approaches should be tried.

We can only see a short distance ahead, but we can see plenty there that needs to be done.

A.M. Turing (1950) Computing Machinery and Intelligence. *Mind 49:* 433-460.

1

Cos'è l'Informatica

L'Informatica nasce negli anni '40 e incorpora contributi che provengono da diverse discipline come la **cibernetica**, la **teoria dell'informazione**, la **psicologia cognitiva** e l'**intelligenza artificiale**.

Il termine ***Informatica* (*Informatique*)** é un neologismo, adottato per la prima volta nel 1962 dal francese Philippe Dreyfus e che deriva dalla concatenazione di due parole: ***Informazione*** (*Information*) e ***Automatica*** *(Automatique)*. Successivamente nel 1966 l'Accademia Francese ha accettato la seguente definizione: "La scienza del trattamento razionale dell'informazione, specialmente da parte di macchine automatiche, considerata come il supporto della conoscenza umana e della comunicazione nel campo tecnico, economico e sociale".

L'Informatica opera su due livelli:

- il livello **logico-concettuale**, che riguarda le teorie e i metodi dell'informatica intesa come **Scienza dell'Informazione;**
- il livello **tecnologico**, che riguarda le teorie e i metodi dell'informatica intesa come **Scienza dei Calcolatori.**

La **Scienza dell'Informazione:**

- studia le **strutture per la rappresentazione dei dati**;
- studia i **metodi per la soluzione dei problemi** con l'aiuto del calcolatore;
- studia i **linguaggi** per dialogare con le macchine;
- studia la messa a punto di **metodologie** per
  - la **produzione del software**,
  - **l'interazione tra processi** di calcolo,
  - **l'organizzazione di grandi basi di dati**,
  - la **rappresentazione** e il **trattamento** delle **immagini fisse o in movimento**,
  - l'**elaborazione automatica di documenti** di varia natura.

La **Scienza dei Calcolatori** studia i diversi tipi di:

- architetture delle macchine per elaborare informazioni;
- processori;
- memorie;

- dispositivi periferici per la comunicazione con l'ambiente circostante.

Anche se questi due piani sono distinti tra loro, spesso l'evoluzione tecnologica ha indotto innovazioni nel campo del software. Ad esempio, fino a pochi anni fa non era possibile trattare le informazioni di tipo multimediale (immagini, suoni, filmati), poiché non esistevano né dispositivi fisici per la loro acquisizione e riproduzione (webcam, microfoni, macchine fotografiche digitali) né memorie sufficientemente capaci di contenerle.

Il campo d'azione dell'Informatica comprende sia **la concezione e lo sviluppo di sistemi software**, che determinano il funzionamento e l'impiego degli elaboratori elettronici, sia **il progetto e la realizzazione di sistemi informatici complessi**, attraverso l'uso di architetture di calcolo sofisticate e reti telematiche.

La collaborazione tra industria delle telecomunicazioni e Informatica ha dato poi origine alla **Telematica**, che a sua volta ha influito non solo nell'evoluzione tecnologica della comunicazione tra calcolatori (**reti** di computer)**,** ma anche negli aspetti logico-concettuali per consentire lo scambio di informazioni, l'accesso a basi di dati distribuite ed il calcolo distribuito.

Tradizionalmente le attività connesse al settore dell'Informatica erano indicate con la sigla **EDP** (*Electronic Data Processing*), elaborazione elettronica dei dati. Attualmente, invece, si parla sempre più spesso di **ICT** (*Information and Communication Technology*), un insieme vario di tecnologie che serve a conservare, elaborare e trasferire informazioni.

Non dobbiamo dimenticare che l'Informatica é **la più umanistica delle scienze tecnologiche**: infatti, tenta di **riprodurre** attraverso modelli le nostre **capacità intellettuali** e, inoltre, tenta di offrire attraverso i suoi strumenti un'estensione dei **nostri strumenti cognitivi** (ovvero, della nostra mente).

2

Introduzione alla Teoria dell'Informazione

2.1 Il concetto di Informazione

Possiamo asserire senza timore di essere smentiti che la società in cui viviamo è *la Società dell'Informazione.* Infatti, l'*informazione* - oggi più che mai - condiziona in maniera decisiva ogni attività dell'uomo. Per di più l'informazione gioca un ruolo fondamentale nella trasmissione della vita stessa: infatti, il DNA porta informazione.

Eppure attualmente non esiste una definizione soddisfacente ed universalmente accettata di questo termine. Per un manager, ad esempio, informazione è tutto ciò che gli consente di prendere delle decisioni.

Ma qual è l'etimologia di questa parola? La parola informazione deriva dal latino *in formare*: "dare forma a qualcosa che non ha forma".

ESEMPIO. *Per un annunciatore televisivo informare equivale a dare una struttura ai propri pensieri sotto forma di sequenza di parole della lingua italiana.* ■

Partendo da esempi della vita quotidiana, osserviamo che *non tutte le notizie costituiscono informazione*. Infatti, affermiamo che qualcuno ci fornisce un'informazione se la notizia comunicata è *per noi eclatante*; in altre parole produce in noi una reazione di sorpresa in quanto tale notizia *è inattesa*. Invece, una persona noiosa non apporta informazione, poiché ripete continuamente gli stessi argomenti.

Possiamo, quindi, provare a definire l'**informazione** in modo indiretto:

- specificando la **natura** con cui essa si manifesta (scritta, orale, simbolica, …);
- evidenziando il suo elemento caratteristico di **sorpresa**;
- specificando gli **effetti** che essa produce nella persona o entità che la recepisce.

Quando si parla di informazione normalmente si presuppone l'esistenza di due entità (non necessariamente delle persone fisiche), che colloquiano tra loro: il **mittente**, che invia il messaggio, e il **ricevente**, al quale è destinato.

Il mittente, che vuole comunicare un concetto contenuto nella propria mente, deve *codificarlo* sotto forma di sequenza di simboli (ad esempio, sequenza di parole della lingua italiana), utilizzando un *formalismo* (ad esempio, la grammatica della lingua italiana). Il messaggio è quindi trasmesso dalla voce attraverso l'etere, per essere poi ricevuto dall'orecchio del ricevente, che lo decodifica e lo interpreta.

Fig. 2.1 – Esempio di comunicazione.

Nell'esempio della figura precedente la sintassi usata per la codifica del messaggio è quella della grammatica italiana, ed il messaggio consiste in un insieme di parole che si susseguono ("VOGLIO", "LA", "TORTA") seguendo delle regole. Infatti, la nostra frase è composta da un Soggetto, un Predicato e un Complemento Oggetto. Il Soggetto è sottinteso, il Predicato è costituito dalla parola "VOGLIO" e il Complemento Oggetto dalle parole "LA TORTA".

Una definizione formale di *informazione* può essere data definendo i tre possibili livelli con i quali l'informazione si manifesta:

- **Livello semantico:** il significato associato all'informazione (per il mittente);
- **Livello sintattico (o strutturale)**: la forma con cui l'informazione si manifesta;
- **Livello pragmatico**: l'impatto del messaggio nel ricevente (come reagisce all'informazione il ricevente).

Consideriamo, ad esempio, un politico che vuole comunicare le proprie idee agli elettori. Individuiamo l'aspetto semantico nel piano che il politico ha in mente, l'aspetto sintattico nel modo in cui il piano viene codificato utilizzando tutti i mezzi di comunicazione possibili e immaginabili (volantini, spot tv, ecc.), ed infine l'aspetto pragmatico nella reazione che l'elettore ha.

2.2 Il Contenuto Informativo

La Teoria dell'Informazione afferma che non è possibile parlare del **contenuto informativo** di un messaggio *ricevuto*, perché l'informazione rappresenta ciò che è inatteso.

Esempio – Estrazione di una pallina. *Supponiamo di avere un'urna contenete 100 palline numerate, per ognuna delle quali la probabilità di essere estratta è la stessa. Il mittente estrae una pallina e comunica il numero impresso su di essa al ricevente.* ■

Esempio - Lancio di una moneta. *Supponiamo che il mittente sia in possesso di una moneta perfettamente bilanciata e che comunichi, dopo averla lanciata, il messaggio "Testa" oppure "Croce".* ■

Cosa differenzia i due esempi appena visti? Nel primo caso vi è una probabilità su 100 che sia estratta una particolare pallina, mentre nel secondo vi è una probabilità su due che esca "*Croce*". Riferendoci ai due esempi, ci sembra naturale associare un contenuto informativo minore al messaggio "*Croce*" rispetto al messaggio "*É stata estratta la pallina 44*". In particolare, ai messaggi dell'esempio del lancio della moneta associamo un contenuto informativo che è il minimo possibile, poiché vi sono solo due scelte possibili ed equiprobabili. Questa idea sta alla base della definizione del bit, l'**unità di misura dell'informazione**.

Definizione. Un **BIT** (Binary digIT) è la quantità di informazione associata alla *comunicazione* di un evento, tra due equiprobabili. ■

Osserviamo, inoltre, che entrambi gli esempi precedenti, pur avendo contenuto informativo diverso, hanno simile effetto pragmatico:

- il ricevente sarà interessato al messaggio se ha degli interessi al suo contenuto, ad esempio, se egli scommette sull'eventualità che sia estratta la pallina 44;
- se viene estratta la pallina 44, il ricevente sarà profondamente soddisfatto, e sarà certamente insoddisfatto nel caso contrario.

Vediamo che una caratteristica dell'informazione è l'apporto di *novità*, che permette di accrescere la conoscenza del ricevente. Consideriamo il seguente esempio:

Esempio – Estrazione di un libro. *In un'urna ho un libro composto da una sola pagina contenente una poesia di Ungaretti e un libro che contiene l'intera Divina*

Commedia. Il mittente può estrarne uno con la stessa probabilità e leggerne il contenuto. ■

Ci domandiamo a quale dei due eventi è associato maggiore contenuto informativo:

- dal punto di vista *intuitivo* la quantità di informazione contenuta nella *Divina Commedia* è **maggiore** di quella di una sola poesia di *Ungaretti*;
- dal punto di vista della *Teoria dell'Informazione*, invece, il contenuto informativo è lo stesso, perché la probabilità dell'evento "estrazione della *Divina Commedia*" è **identica** a quella dell'evento "estrazione della poesia di *Ungaretti*". Si determina una situazione che per il senso comune è assurda: i due libri hanno lo stesso contenuto informativo, che è pari ad un **bit**.

2.3 Il Processo Comunicativo

Dal punto di vista storico, chi per primo ha provato a definire formalmente il concetto di informazione è stato **R.V.L. Hartley** (1888-1970), lo scienziato noto soprattutto per il suo *circuito oscillante*, in cui ha utilizzato per la prima volta una valvola per produrre corrente oscillante, sostituendola alla scintilla utilizzata sino a quel momento come mezzo per la produzione di onde elettromagnetiche, e rivoluzionando in tal modo l'intero mondo delle radiocomunicazioni.

Nel 1928, in uno storico articolo apparso sul *Bell System Technical Journal*, Hartley ha sviluppato per primo il concetto di informazione basato su "considerazioni fisiche, anziché psicologiche", finalizzato allo studio delle comunicazioni elettroniche. In particolare, nella prima parte dell'articolo suddetto, intitolata la *Misura dell'Informazione*, Hartley non definisce mai formalmente tale concetto, ma piuttosto tenta di caratterizzare l'informazione in termini di alcune sue proprietà. Egli asserisce che l'informazione esiste nel processo di trasmissione di simboli, in cui "*ciascun simbolo assume un certo significato per le parti comunicanti*".

Quando qualcuno riceve informazione, ogni simbolo ricevuto permette al ricevente di "*eliminare alternative*", escludendo altri simboli possibili ed il loro significato associato. Quindi, Hartley intende l'informazione come esclusione di eventi che non si possono verificare: si/no, bianco/nero, testa/croce, vero/falso, … Le sue idee gli valsero la medaglia d'oro dalla IEEE (Institute of Electrical and Electronics Engineers, Inc.) nel 1946.

Esempio- Elenco telefonico. *Consideriamo un elenco telefonico che contiene i nomi, gli indirizzi ed i numeri di telefono di persone residenti in una determinata città, ordinati per cognome. Se cerco Rossi Mario, il senso comune mi suggerisce di aprire l'elenco a metà, controllare i cognomi contenuti in quella pagina, e, se il cognome Rossi cercato non compare nella pagina, eseguire le successive ricerche nella parte precedente o in quella successiva dell'elenco, scartando la restante metà. Questo esempio suggerisce che l'informazione "cognomi presenti nella pagina" permette di scartare un certo numero di alternative, ovvero tutti i cognomi presenti nella prima (o seconda) parte dell'elenco. In questo senso l'informazione è un elemento che riduce l'incertezza di una scelta.* ■

Esempio – Analisi Cliniche. U*n medico, per poter effettuare una diagnosi precisa di una malattia, prescrive al paziente dei test clinici. Infatti, l'informazione "test positivo" gli consente di escludere eventuali altre malattie che presentano gli stessi sintomi.* ■

Gregory Bateson (1904-1980), uno dei più illustri antropologi e cibernetici di questo secolo, aveva una concezione diversa di informazione. Egli ha definito, infatti, l'informazione come "*that which changes us*", ciò che ci cambia.
Bateson ha contribuito ad elaborare la scienza della cibernetica con i colleghi Warren McCulloch e Norbert Wiener. Opponendosi fortemente a quegli scienziati che tentavano di "*ridurre*" ogni cosa a pura materia, si è dedicato a reintrodurre la "*mente*" nelle equazioni scientifiche. Dal suo punto di vista la mente è una parte essenziale della "realtà materiale" e, quindi, non ha senso cercare di scindere la mente dalla materia.

Claude E. Shannon (1916-2001), fondatore della **Teoria Matematica dell'Informazione,** è stato il primo ad introdurre la distinzione tra *forma* e *significato* nel processo comunicativo.

Egli, formalizzando l'idea di Hartley, ha affermato che l'informazione è: "*that which reduces uncertainty*", tutto ciò che può consentire di ridurre il nostro grado di incertezza su un evento che si può verificare. A tal fine egli ha introdotto per la prima volta la parola **bit** nel suo articolo "*A Mathematical Theory of Communication*", apparso sul *Bell System Technical Journal* nel 1948, riconoscendo il suo ruolo centrale nel processo della comunicazione, e dando così origine, forse inconsapevolmente, alla rivoluzione informatica a cui assistiamo oggi. Shannon ha introdotto il **bit** per rappresentare due situazioni possibili, ovvero "*elemento non presente in un insieme A*" ed "*elemento presente in un insieme A*".

Nell'esempio dell'elenco telefonico, l'insieme A è costituito dai cognomi presenti nella prima metà dell'elenco.

Secondo la teoria di Shannon un qualunque **processo comunicativo** consta essenzialmente di **cinque parti**:

- un **mittente**, o **sorgente** di informazioni, che produce un messaggio da comunicare ad un'altra entità;
- un **trasmettitore,** che viene utilizzato per codificare il messaggio in modo che possa viaggiare su un canale di comunicazione;
- un **canale di comunicazione**;
- un **ricevitore,** che riceve ciò che viaggia sul canale e lo decodifica per ricostruire il messaggio;
- il **destinatario**, al quale giunge il messaggio.

Consideriamo, ad esempio, l'attività comunicativa dell'uomo: i pensieri residenti nel cervello (la sorgente) di un uomo, utilizzando l'apparato vocale (il trasmettitore) sono codificati sotto forma di parole, che viaggiano lungo lo spazio circostante (il canale), per giungere finalmente all'apparato uditivo (il ricevitore) di un altro uomo e da qui essere convertiti nuovamente in concetti nel cervello (il destinatario) di un altro uomo.

Shannon introduce altre due importanti **ipotesi**:

- **la trasmissione dei simboli lungo il canale costituisce un fenomeno discreto**, ovvero l'invio di ciascun simbolo richiede una certa quantità di tempo, finita e non nulla;
- esiste una **sorgente di rumore,** che agisce sul canale modificando in modo imprevedibile il contenuto sintattico dell'informazione, la sua forma. In altre parole, la sorgente di rumore può trasformare un simbolo che viaggia lungo il canale in un altro simbolo, in modo assolutamente imprevedibile. Ad esempio, la parola "fiaschi" può diventare "fischia", o "giaschi" o …

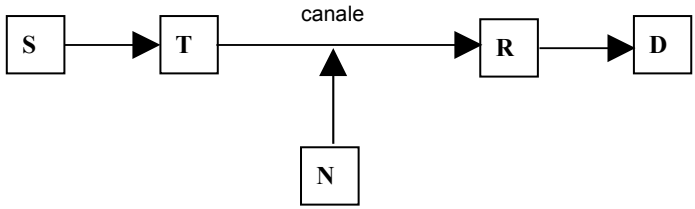


Fig. n. 2.2 – SCHEMA DEL PROCESSO COMUNICATIVO SECONDO SHANNON.

Nella figura è riportato lo schema di Shannon, dove S rappresenta la sorgente, T il trasmettitore, R il ricevitore, D il destinatario e N la sorgente di rumore.

Il suddetto processo comunicativo mette in luce diversi problemi:

- *come **misurare la quantità di informazione** che viaggia lungo il canale.*
  Ricordiamo che Shannon ha definito come unità di misura dell'informazione il **bit**;
- *come garantire **trasmissioni sicure.***
  Rendere le trasmissioni sicure significa porre l'invio dell'informazione al

riparo da occhi indiscreti, trasformandola. Dagli anni ’70 la teoria che si occupava di garantire trasmissioni sicure si è evoluta per conto proprio, diventando la **crittografia**, l’arte di nascondere i messaggi o renderli non intelligibili.

- *come garantire **trasmissioni affidabili**.*
  Shannon ha dimostrato che non è possibile garantire trasmissioni affidabili, se non introducendo ridondanza nel messaggio. Ad esempio, maggiore è il numero delle volte che lo stesso messaggio è replicato (ovvero rispedito), maggiore è la probabilità che almeno una volta il messaggio arrivi corretto.

Ricapitolando. Nella teoria di Shannon, *non è possibile* definire la quantità di informazione associata ad un messaggio *già ricevuto*, ma piuttosto la quantità di informazione associata ad un *papabile messaggio*. Inoltre, la quantità di informazione associata all’evento costituito dalla ricezione di un particolare messaggio è tanto più alta quanto più è bassa la probabilità che quell’evento si verifichi, ovvero tanto più alta quanto più inatteso è quel messaggio. Ad esempio, il messaggio “*Domani sorgerà il sole*” ha un basso contenuto informativo, perché scontata e attesa, mentre il messaggio “*Domani scoppierà la guerra*” ha un alto contenuto informativo.

2.4 Informazione ed Entropia

É interessante notare come sia possibile perdere "*qualcosa*" in ogni fase del processo comunicativo:

- nella codifica del messaggio della sorgente da parte del trasmettitore;
- lungo il canale, a causa del rumore;
- nella decodifica da parte del ricevitore per consegnare il messaggio al destinatario.

Ad esempio**:**

- ho in mente il ricordo di un bel tramonto, ma non riesco a comunicarlo verbalmente (ovvero *codificarlo* utilizzando il linguaggio *più naturale* per me, l'italiano);
- ho in mente il progetto di una bella casa, ma non so disegnare bene (ovvero *codificarlo* utilizzando carta e penna);
- ho in mente un motivo musicale, ma non so rappresentarlo su carta (ovvero *codificare il motivo sul pentagramma)*. In questo caso, simboli quali ♫ e ♪ rappresentano, in funzione della loro posizione all'interno del pentagramma, sia l'altezza del suono (ovvero la nota corrispondente: do, re, mi, …), sia la durata di ciascuna nota;
- chi è seduto in fondo alla classe non comprende bene la lezione a causa degli altri studenti che chiacchierano (ovvero vi è *rumore* sul *canale*);
- vedendo una foto a colori, un daltonico ha una rappresentazione dell'immagine diversa dall'originale (ovvero vi è un problema di *decodifica* del messaggio);
- leggo una lettera scritta in russo inviatami da un collega, ma non comprendo il suo contenuto (ovvero vi è un problema di *decodifica* del messaggio).

Nel processo di comunicazione dei nostri pensieri, quindi, siamo costretti a **strutturare** le nostre idee, che sono **libere** nella nostra mente, sotto forma di frasi della lingua italiana. In altre parole, cerchiamo di mettere **ordine** (**struttura**), laddove ordine non c'è. Inevitabilmente si perde "*qualcosa*".

I fisici definiscono *l'entropia* come la misura del disordine di un sistema: più ordinato, strutturato, è un sistema, minore è l'entropia e viceversa. Il processo di codifica dei nostri pensieri introduce inevitabilmente una perdita di entropia.

Tutto ciò è legato al problema di rappresentare i nostri pensieri in forma strutturata.

2.5 I Simboli

L'informazione viaggia lungo il canale come una successione di *simboli.* Tornando all'esempio della Figura n. 1, i simboli sono tratti dal lessico (ovvero, vocabolario) della lingua italiana.

Definizione. Un **simbolo** è un segno al quale è attribuito un valore convenzionale. ■

È evidente che un simbolo esiste indipendentemente dal supporto materiale che è utilizzato per rappresentarlo. Ad esempio, affermando che:

- *l'aquila è il simbolo degli USA*;
- *la colomba é il simbolo della pace*;
- *il verde é il simbolo della speranza*;
- *il simbolo" ®" indica un marchio depositato;*
- *il simbolo "€" indica la nostra valuta*

si stabilisce un'associazione tra il simbolo ed il significato corrispondente.

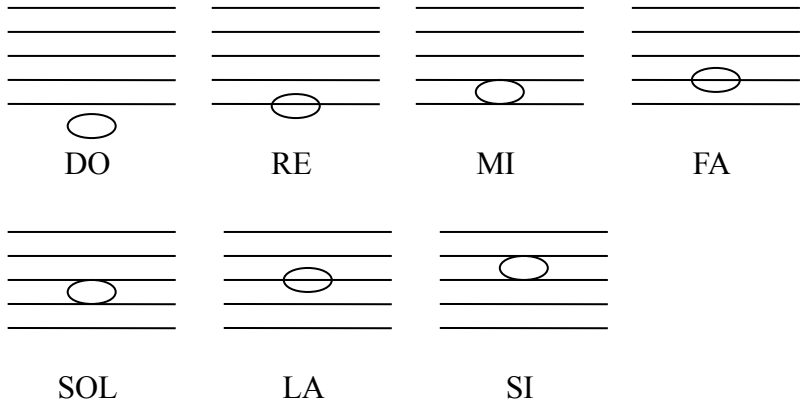
Occorre notare che il segno "4" rappresenta **per noi** la cifra araba 4 e costituisce, quindi, **per noi** un simbolo, mentre **per un bambino** di due anni è semplicemente uno scarabocchio! Allo stesso modo, ad un ideogramma dell'alfabeto cinese noi non attribuiamo alcun significato!

Definizione. Un **alfabeto** è un insieme finito di simboli. ■

Per poter comunicare occorre quindi definire un alfabeto ed il significato convenzionale di ciascuno dei simboli che lo compongono.

Nota bene. A livello hardware un elaboratore elettronico opera utilizzando due livelli di tensione, alto e basso, identificati convenzionalmente con le cifre arabe "1" e "0". Pertanto "1" e "0" costituiscono l'alfabeto utilizzato dall'hardware dell'elaboratore elettronico.

Esempio – Le Note Musicali. *Un esempio di alfabeto è dato dall'insieme delle note musicali rappresentate come simboli su un pentagramma.*



Tali
simboli
possono
essere
accostati
per comporre delle melodie. ■

3

Sistemi e Modelli

3.1 Il concetto di Sistema

In modo informale possiamo dire che un **sistema** é un insieme di entità reali o astratte, dette **componenti**, che interagiscono tra loro in modo tale che il loro funzionamento soddisfi determinate specifiche.

Quindi, di un sistema c'interessa conoscere:

- la *funzione* svolta (ovvero, a cosa serve?);
- l'insieme delle *componenti* di cui é costituito (ovvero, le parti);
- le *interazioni* con l'ambiente esterno;
- le *relazioni* che descrivono i rapporti esistenti tra le diverse componenti.

Esempio – Il sistema Ascensore. *Se analizziamo il sistema ascensore osserviamo che:*

- *la funzione svolta é quella di trasportare persone da un piano ad un altro;*
- *il sistema é composto da un motore, da una cabina, dalle pulsantiere e dai dispositivi di trasmissione del moto (carrucole, funi, ...);*
- *il sistema riceve in ingresso energia elettrica e persone e cede energia e le persone trasportate;*
- *i dispositivi fisici che compongono il sistema devono essere connessi tra loro, in modo da garantire una corretta trasmissione del moto.* ■

Fig. n. 3.1 – Il sistema Ascensore

Esempio – Il sistema Respiratorio. *Se analizziamo il sistema respiratorio, osserviamo che:*

- *la funzione svolta è quella di prelevare l'ossigeno dall'aria ed eliminare il diossido di carbonio dall'organismo;*
- *il sistema é composto dal naso, dalla faringe, dalla laringe, dalla trachea, dai bronchi, dai polmoni e dal diaframma;*
- *il sistema riceve in ingresso l'aria e cede il diossido di carbonio;*
- *mediante impulsi nervosi le varie componenti del sistema concorrono a trasportare ossigeno all'organismo ed ad espellere il diossido di carbonio.*■

Fig. n. 3.2 – Il sistema Respiratorio

Ad ogni istante un sistema é in una particolare condizione detta **stato**, che é determinata dalle *condizioni istantanee* delle *varie componenti*. Per determinare lo stato di un sistema non é necessario conoscere le condizioni di tutte le componenti, ma é necessario conoscere solo quelle rilevanti per lo studio.

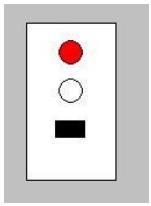
Esempio – Stati del sistema ascensore. *Riconsiderando l'esempio dell'ascensore, sono possibili due stati del sistema:*

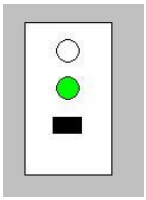
- ***ascensore fermo**, oppure*
- ***ascensore in movimento**.*

In entrambi i casi é sufficiente conoscere lo stato di una sola componente, il motore, per stabilire lo stato dell'intero sistema.

Se invece la pulsantiera dell'ascensore è dotata di un display per la visualizzazione del piano (0, 1, 2, 3), gli stati del sistema diventano:

- ***ascensore fermo a piano terra** (quando il display visualizza 0)*
- ***ascensore fermo al primo piano**(quando il display visualizza 1)*
- ***ascensore fermo al secondo piano**(quando il display visualizza 2)*
- ***ascensore fermo al terzo piano**(quando il display visualizza 3)*
- ***ascensore in movimento**(quando il display non visualizza nessun numero)* ■





Ascensore in movimento **Ascensore fermo**

Fig. n. 3.3 – GLI STATI DEL SISTEMA ASCENSORE

ESEMPIO – STATI DEL SISTEMA RESPIRATORIO. *Riconsiderando l'esempio del sistema respiratorio, sono possibili due stati del sistema:*

- ***fase d'inspirazione***
- ***fase di espirazione***.

In entrambi i casi é sufficiente conoscere lo stato del diaframma per stabilire lo stato dell'intero sistema: quando il diaframma è in contrazione avviene la fase d'inspirazione, quando invece è nello stato di rilassamento avviene la fase di espirazione. ■

Fig. n. 3.4 – GLI STATI DEL SISTEMA RESPIRATORIO

L'evoluzione nel tempo di un sistema é descritta dalla **storia degli stati**, una successione cronologica degli stati.

3.2 Variabili Endogene, Esogene e di Stato

Per *ambiente esterno* intendiamo l'insieme di tutti gli oggetti della realtà d'interesse, che non sono componenti del sistema ma che sono influenzati dalla sua attività o che possono influenzare il suo funzionamento.

Le variabili generate all'interno del sistema sono dette **endogene**, mentre quelle presenti nell'ambiente esterno e che afferiscono al sistema sono dette **esogene**.

In passato, le variabili endogene erano chiamate anche *variabili di uscita* del sistema, poiché esse dipendono dal sistema e sono generate dallo stesso, mentre le variabili esogene erano chiamate anche *variabili di ingresso* al sistema, perché prodotte dall'ambiente esterno e, quindi, indipendenti dal sistema.

Esempio – Il sistema Radio. *Consideriamo il sistema radio e studiamone il suo funzionamento. La variabile in ingresso é la frequenza da impostare, mentre la variabile in uscita é l'insieme dei suoni prodotti dalla radio.* ■

Fig. n. 3.5 – Il Sistema Radio.

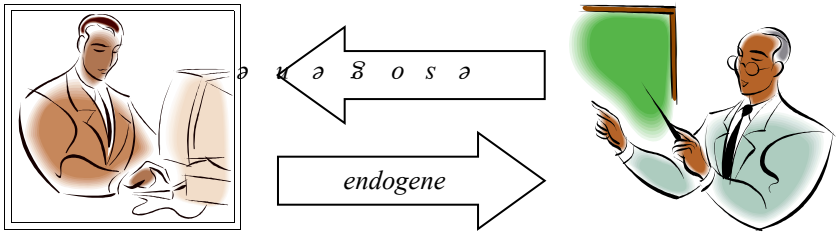
Gli stati del sistema sono descritti invece mediante le **variabili di stato**; esse interagiscono sia con le variabili endogene sia con quelle esogene secondo le relazioni funzionali del sistema.

Esempio – Le variabili di stato del sistema Radio. *Ritornando all'esempio precedente della radio, sue variabili di stato sono la condizione di acceso o spento, la frequenza impostata, ...* ■

Esempio – Il sistema Studente. *Analizziamo il sistema studente per valutarne il comportamento durante un esame. Le variabili da considerare sono:*

- ***esogene**: le domande del docente, i commenti dell'insegnante alle risposte date, i disturbi sonori dovuti al vocio degli altri studenti, ...*
- ***endogene**: le risposte dello studente;*

- ***di stato***: *il colore del viso, il battito cardiaco, la percentuale di risposte esatte, il livello di preparazione, ...* ■



Sistema Studente **Ambiente esterno**

Fig. n. 3.6 – IL SISTEMA STUDENTE

3.3 Sistemi Aperti e Sistemi Chiusi

Esistono diverse tipologie di classificazione dei sistemi. Una prima suddivisione dei sistemi può essere fatta in base alla presenza o meno delle variabili esogene e, quindi, in base all'interazione del sistema con l'ambiente esterno. Nel primo caso il sistema é **aperto**, altrimenti é **chiuso**.

Esempio – Il sistema Computer. *Un computer, studiato come sistema per la diffusione di informazioni, é un sistema aperto se é acceso, poiché comunica all'esterno informazioni visive e sonore e riceve informazioni attraverso l'uso della tastiera. Se é spento, é un sistema chiuso, perché non può né ricevere e né trasmettere informazioni.* ■

Fig. n. 3.7 – Il Sistema Computer

Esempio – Il sistema Termodinamico. *In Fisica un sistema si dice aperto se può scambiare con l'ambiente esterno sia materia che energia; quindi, una pentola piena d'acqua in ebollizione sul fuoco è un sistema aperto, perché riceve energia (calore) dal fuoco ed emette vapore acqueo all'esterno.*

Se invece lo scambio con l'ambiente esterno è solamente di tipo energetico, il sistema si dice chiuso; quindi, una pentola a pressione con il coperchio chiuso ermeticamente posta sul fuoco è un esempio di sistema chiuso, perché non viene ceduto all'ambiente esterno il vapore acqueo ■.

Aperto

Chiuso

Fig. n. 3.8 – Il Sistema Termodinamico

Esempio – La cellula. *In Biologia la cellula è un sistema aperto; infatti, attraverso la membrana cellulare avviene uno scambio di materia con l'ambiente esterno.* ■

Fig. n. 3.9 – La cellula

3.4 Sistemi Continui e Sistemi Discreti

Diciamo che una variabile di stato é **continua** se i valori (numerici) che essa assume costituiscono un intervallo dei numeri reali; in caso contrario tale variabile é detta **discreta**.

Ad esempio, la luminosità di una stanza dotata di finestre aperte é una variabile di stato continua. Al contrario, se l'illuminazione é artificiale, la luminosità della stanza costituisce una variabile di stato discreta (può assumere solo due valori).

In precedenza abbiamo detto che di un sistema studiamo la sua evoluzione nel tempo. *Il tempo é una grandezza astratta*. Tuttavia, nel sistema in esame, il tempo é rappresentato attraverso una variabile, che consente di riferirci agli istanti in cui il sistema é osservato. Se interessa osservare il comportamento del sistema solo in alcuni determinati istanti di tempo, il sistema é detto **discreto**. Se, al contrario, interessa osservare il comportamento dei sistemi in qualsiasi istante di tempo, il sistema é detto **continuo**.

3.5 Eventi ed Attività

Un **evento** rappresenta un cambiamento di stato *istantaneo* nel sistema, ovvero per il quale non é richiesto nessun tempo.

Un'**attività** invece rappresenta una condizione del sistema che richiede del tempo. In generale, un'attività può essere individuata da due eventi: quello con il quale inizia e quello con il quale termina.

Esempio – Attività del sistema Ascensore. *Ritornando all'esempio dell'ascensore, un'attività é costituita dall'andare da un piano ad un altro; tale attività é delimitata da due eventi, quello della partenza dell'ascensore e quello del suo arresto.* ■

3.6 Sistemi Deterministici e Sistemi Stocastici

Un'ulteriore distinzione dei sistemi può essere fatta in base alle relazioni che intercorrono tra le varie componenti del sistema. Se gli effetti prodotti dalle attività del sistema dipendono in modo univoco dalle variabili di ingresso il sistema é **deterministico**; se invece tali effetti variano in modo casuale il sistema é **stocastico**.

Esempio – Lancio di una moneta. *Consideriamo il lancio di una moneta. Siamo interessati all'esito del lancio, ovvero al verificarsi, in seguito al lancio, di un certo evento: testa oppure croce.*

- *Se (teoricamente!) conoscessimo tutte le variabili che descrivono la costituzione fisica ed il comportamento della moneta (peso, densità, ecc...), le variabili relative al lancio (posizione iniziale, forza, direzione, ...) e tutte le altre grandezze in gioco (velocità del vento, vibrazione del pavimento, ...), in base alle leggi della Fisica saremmo in grado di determinare a priori il risultato del lancio e, quindi, il sistema studiato sarebbe deterministico;*
- *Poiché questo problema é incredibilmente complesso ed il numero delle variabili in gioco estremamente elevato, risulta impossibile costruire un modello deterministico e, pertanto, si preferisce utilizzare un modello di tipo statistico, che ci permette di studiare il problema solo in termini di probabilità che un certo evento si verifichi.* ■

3.7 Modelli

Definiamo modello di un sistema una sua rappresentazione, che può essere sia una replica fisica sia una rappresentazione simbolica.

Definizione. Dato un *sistema reale* ed un *problema* da risolvere su *tale sistema reale*, un **modello** del sistema, relativo al problema in esame, é un sistema più semplice che contiene tutti e soli gli elementi del sistema reale che sono essenziali per la risoluzione del problema. ■

Un modello rappresenta quindi solo *alcune* delle caratteristiche di un oggetto reale. Il modello di una persona potrebbe essere ad esempio costituito da:

- una foto;
- un disegno;
- una statua;
- alcuni dei suoi attributi: peso, altezza, età.

Esistono, quindi, tanti tipi di rappresentazione dello stesso oggetto del mondo reale, ma non esiste la rappresentazione ideale.

Esempio – Il modello di una casa. *Supponiamo di voler misurare la* ***resistenza strutturale di una casa****. A tal fine potremmo provare a colpire la casa con un bulldozer, ma é fin troppo evidente che tale soluzione non é accettabile! Ricorriamo, quindi, ad un modello della casa:*

- *una prima soluzione consiste nel costruire una casa identica alla nostra e provare a colpirla con un bulldozer, ma tale soluzione sarebbe troppo costosa;*
- *una seconda soluzione potrebbe essere quella di costruire un modello in scala della casa, ma probabilmente non avrebbe le stesse caratteristiche di resistenza strutturale della casa originale;*
- *un'ulteriore soluzione potrebbe essere quella di definire un modello matematico della struttura della casa, su cui potremmo intervenire usando carta, penna e ragionamento logico-matematico: é evidente che tale modello sarebbe riutilizzabile ed economico.* ■

Nell'esempio suddetto, il modello matematico sembrerebbe il più preferibile, ma non é sempre così. Un architetto, infatti, non presenta al proprio cliente un modello matematico, ma un disegno in scala (il progetto).

3.8 Valutazione di un modello

Nella valutazione di un modello, secondo quanto detto precedentemente, occorre tener presente le seguenti caratteristiche:

- *adeguatezza* rispetto all'obiettivo prefissato:
  è, infatti, importante saper valutare se il modello é adeguato rispetto alle finalità che si prefigge. Per stabilire se un modello é adeguato si possono seguire alcuni criteri, quali:
  - **la completezza**: il modello deve contenere tutte le informazioni utili allo scopo;
  - **l'usabilità**: il modello deve consentire di ritrovare facilmente e velocemente le informazioni rappresentate;

- *livello di dettaglio o di astrazione:*
  i due termini esprimono concetti antitetici. Per **astrazione** si intende il processo con il quale si omettono dettagli irrilevanti al fine di conseguire l'obiettivo prefissato. Ovviamente, maggiore é il livello di dettaglio, minore é quello di astrazione, e viceversa. Perciò il modello deve consentire di ignorare certi dettagli in modo da potersi concentrare solo sulle caratteristiche per noi essenziali;

- *generalità*:
  il modello ha un ampio spettro di utilizzo, ovvero permette di rappresentare un vasto insieme d'entità del mondo reale.

3.9 Classificazione dei Modelli

Nell'esempio precedente relativo alla misura della resistenza strutturale di una casa abbiamo visto che per un sistema (la casa) e per un dato problema (la resistenza strutturale) esistono più modelli anche molto diversi tra loro, ma tutti aventi uno stretto legame con il sistema, che può essere individuato come mantenimento delle stesse proprietà di base o caratteristiche.

Perciò diamo una classificazione dei modelli secondo tali caratteristiche, suddividendoli in due classi in base al loro diverso grado di astrazione: modelli *reali o fisici* e modelli *astratti o simbolici*.

3.9.1 Modelli Fisici

Definizione. Un **modello fisico** é un modello la cui struttura é costituita da elementi fisici e pertanto é misurato con grandezze fisiche. ■

Esistono vari tipi di modelli fisici:

- **analogici:** si comportano analogamente allo stesso oggetto del mondo reale;
- **iconici:** assomigliano al sistema descritto;
- **in scala:** riproducono il sistema in dimensioni ridotte.

Esempio – Il modello di un serbatoio d'acqua. *Il comportamento di un serbatoio d'acqua, che si riempie e si svuota, può essere simulato utilizzando un componente elettronico, il condensatore.* ■

In questo caso é stato utilizzato un modello analogico.

Esempio – Il modello di una molecola. *Le strutture molecolari in genere sono rappresentate da sfere, che rappresentano gli atomi, e da barre, che rappresentano i legami tra gli atomi.* ■

È un tipico esempio di modello iconico.

Esempio – Il modello di una Ferrari. *Il modellino (giocattolo) di una Ferrari riproduce, in scala ridotta, la carrozzeria della Ferrari. Tuttavia esso non riproduce in scala ridotta le altre componenti (motore, telaio, ecc…).* ■

In questo caso é stato utilizzato un modello in scala.

3.9.2 Modelli Simbolici

Definizione. Un **modello simbolico** rappresenta un'entità del mondo reale mediante simboli che esprimono le grandezze in gioco. ■

Un tipico modello simbolico é costituito dalla segnaletica stradale, che riporta indicazioni di direzione, divieti, pericolo, …

A questa classe di modelli appartengono non solo i modelli verbali, come i linguaggi naturali (l'italiano, l'inglese, ...) e i linguaggi artificiali (i linguaggi di programmazione), ma anche i modelli matematici.

Definizione. Un **modello matematico** rappresenta un sistema utilizzando:

- *variabili,* per descrivere le componenti;
- *funzioni matematiche,* che pongono in relazione le variabili, per descrivere le attività del sistema. ■

Ad esempio, il moto di un proiettile può essere rappresentato utilizzando delle equazioni**;** lo scopo è, infatti, quello di prevedere dove andrà a finire il proiettile, note la resistenza dell'aria, la forma dell'oggetto e la velocità iniziale.

3.10 I Modelli per l'Informatica

I calcolatori elaborano informazioni: ricevono informazioni in forma simbolica e restituiscono informazioni in forma simbolica. Sono, quindi delle macchine automatiche che trasformano una sequenza di simboli in ingresso in una sequenza di simboli in uscita.

Diciamo quindi che i calcolatori elettronici operano a livello sintattico.

Ad esempio, un elaboratore che calcola la somma dei numeri 12 e 27 può essere visto come un esecutore astratto che riceve in ingresso la sequenza di simboli "1", "2", "+", "2", "7" e "=" e restituisce la sequenza "3", "9". I calcolatori non sono *macchine specializzate*: possono svolgere qualsiasi trasformazione, purché preventivamente definita (ovvero programmata).

Per poter usare il calcolatore dobbiamo fornirgli una *descrizione* della realtà d'interesse sotto forma di simboli. ***Ai fini informatici, quindi, il modello ideale é quello simbolico***.

Esempio – Un modello per una biblioteca. *Supponiamo di delegare a qualcuno la gestione di una biblioteca; per far ciò, occorre fornire una descrizione della realtà bibliotecaria che sia comprensibile da un esecutore arbitrario (umano o non):*

- *una prima soluzione potrebbe essere quella di usare un modello in scala, ad esempio dei libri in miniatura; ma non é economicamente conveniente, né efficace costruire e quindi gestire un milione di libri di miniatura! ;*
- *se il nostro fine fosse la gestione manuale del servizio dei prestiti bibliotecari, si potrebbe utilizzare come modello uno schedario contenente le informazioni relative ai singoli libri e ai prestiti in apposite cartelle;*
- *se, infine, la gestione dovesse essere delegata ad un esecutore non umano (automatizzata), dovrebbe essere necessariamente utilizzato un modello simbolico per rappresentare le informazioni contenute nelle cartelle: titolo del libro, posizione del libro, nome della persona che ha preso in prestito, data in cui é stato preso in prestito.* ■

3.11 Nota storica

L'origine della Programmazione ad Oggetti

La Simulazione é la scienza che si occupa di *simulare* o *imitare* il comportamento di sistemi reali, attraverso la definizione di particolari modelli, utili allo scopo. Tali modelli devono consentire di intervenire sul sistema simulato variando a piacere i parametri che lo caratterizzano, ed osservare quindi concretamente gli effetti di tali variazioni.

Esempio – I Tabelloni Elettronici. *I tabelloni elettronici (primo per diffusione su vasta scala, dal punto di vista storico, Lotus 123) costituiscono forse il più noto esempio di strumento di simulazione assistito dal calcolatore. Utilizzando un tabellone elettronico possiamo effettuare molto facilmente delle complesse simulazioni finanziarie, ad esempio.* ■

Anche se basata su tecniche matematiche e statistiche, spesso la Simulazione é considerata un'arte più che una scienza, in quanto hanno una notevole importanza l'intuito e l'esperienza di chi la utilizza.

La Simulazione di un sistema non richiede necessariamente un calcolatore elettronico: é possibile simulare il comportamento di un sistema reale utilizzando gli strumenti più vari.

Esempio – Pronto Soccorso. *Negli ambienti ospedalieri per simulare interventi di pronto soccorso sono utilizzati dei particolari manichini, con i quali è possibile rappresentare le possibili situazioni d'emergenza in cui medici e infermieri si possono trovare.* ■

Di interesse notevole al giorno d'oggi é la simulazione assistita dal calcolatore, in cui viene utilizzato un *programma* per simulare il comportamento di un sistema reale. In questo caso, possiamo modificare a piacere i parametri che caratterizzano il sistema, in quanto essi sono rappresentati utilizzando delle variabili all'interno del programma, e possiamo osservare *immediatamente* gli effetti di tali modifiche sul sistema stesso. Non a caso, gli anglosassoni usano l'espressione "*WHAT IF?"*, ovvero "*COSA SUCCEDEREBBE SE?"*, che incarna efficacemente lo spirito della simulazione!

Il termine *simulazione* é stato usato in senso scientifico per la prima volta negli anni '40, quando Von Neumann e Ulan definirono ed utilizzarono una tecnica nota come "Monte Carlo", per risolvere problemi matematici estremamente complessi

(soluzione di equazioni differenziali derivanti da problemi di fisica). Tali problemi si ritenevano impossibili da risolvere in modo tradizionale (ovvero, utilizzando i metodi dell'analisi matematica).

Esempio – Integrazione con il Metodo Monte Carlo. *Tale metodo ci permette di calcolare facilmente una buona approssimazione dell'area di una superficie, anche molto complessa, utilizzando tecniche probabilistiche. Il nome Monte Carlo è stato dato, infatti, proprio riferendosi al noto casinò del Principato di Monaco.*
*Per calcolare l'area di una superficie si individua un rettangolo [a, b] x [c, d] che la contiene, e si "sparano" **n** (dove n è un numero molto grande) punti a caso nel rettangolo. Infine, si contano il numero **k** dei punti che colpiscono l'area dell'integrale (bordo compreso). Allora sarà:*

$$\text{Area approssimata della superficie} = \left(\frac{k}{n}\right) \times \text{Area Rettangolo.}$$

Ovviamente l'approssimazione sarà tanto migliore quanto maggiore è il numero n di punti che vengono "sparati".

■

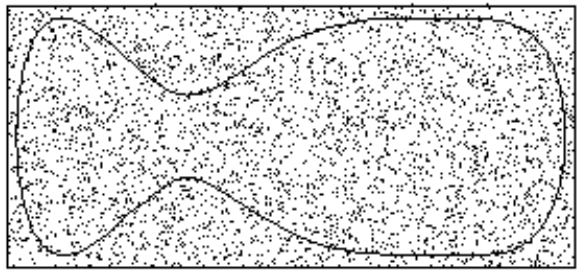


Fig. n. 3.10. – Un esempio di applicazione del metodo Monte Carlo

Negli anni '50, con l'avvento della Cibernetica e dell'Informatica, la simulazione conobbe un nuovo significato; infatti, gli studiosi di varie discipline compresero l'importanza degli elaboratori elettronici per la simulazione di un qualsiasi sistema rappresentabile con modelli di tipo simbolico o matematico.

Oggi, la simulazione al computer é parte integrante della nostra vita – basti pensare ai videogiochi, che simulano il comportamento dinamico (vale a dire, in movimento) di automobili di Formula 1, aeroplani, ecc.

Un modo estremamente semplice per rappresentare un sistema é quello di rappresentare ciascun componente mediante un *oggetto astratto*, il cui stato é

memorizzato *al proprio interno* utilizzando delle *variabili* (nell'accezione informatica del termine). Tale oggetto interagisce con gli altri oggetti presenti nel sistema inviando loro uno o più *messaggi* ogniqualvolta si verifica un determinato *evento* che lo riguarda (vale a dire, ogniqualvolta il proprio stato cambia).

Tale idea sta alla base della programmazione a oggetti. Il primo linguaggio di programmazione ad oggetti é stato, infatti, il SIMULA '67, un'estensione del linguaggio ALGOL '60, che introduceva per la prima volta i concetti di classi, istanze e metodi che caratterizzano i linguaggi ad oggetti moderni. Il nome stesso del linguaggio ricorda che il SIMULA '67 é nato per poter simulare sistemi di vario genere rappresentabili con modelli simbolici.

4

Dal Problema al Programma

4.1 Il Problema, le Azioni e i Processi

Il concetto di **problema** é un **concetto primitivo** in quanto é difficile fornire una definizione che non si riduca ad un sinonimo della parola stessa. Una situazione analoga si ha nel campo della Matematica con il concetto di *insieme*, che non può essere definito se non come una collezione, o una classe, o un'aggregazione di oggetti, e, quindi, con dei sinonimi.

Dal punto di vista storico l'uomo, fin dalla preistoria, si é trovato a dover risolvere dei problemi: sfamarsi, ripararsi dal freddo, difendersi dagli animali feroci. Per risolverli ha dovuto compiere delle *azioni*. Ad esempio, per risolvere il problema "*sfamarsi*" ha messo in atto l'azione di "*cacciare*", ottenendo come **soluzione** il "*cibo*".

Definizione. Un'**azione** é un avvenimento di cui conosciamo il *soggetto* (l'**esecutore**), l'*oggetto* su sui é realizzata l'azione e la *trasformazione* prodotta sull'oggetto dal soggetto. ■

Un'azione è, quindi, la trasformazione che un soggetto opera sul mondo che lo circonda. Ad esempio, se un uomo cammina compie un'azione. Infatti, l'uomo (*soggetto*) compie una trasformazione sul mondo in cui si trova, variando la posizione (*azione*) del suo corpo (*oggetto*) da un punto ad un altro.

Le azioni che ci interessano sono di un tipo ben definito, ovvero sono **azioni descrivibili**.

Definizione. Un'azione si dice **descrivibile** se ha una durata finita e di essa si conoscono tutti i suoi elementi caratterizzanti: il soggetto, l'oggetto e la trasformazione operata. ■

Occorre, quindi, poter riconoscere l'istante preciso in cui l'azione ha inizio, e l'istante preciso in cui termina. L'esempio classico di azione é quello del cucinare: il cuoco (*soggetto*) compiendo una trasformazione sugli ingredienti (*oggetto*) produce la pietanza. Normalmente sui libri di cucina é indicato il tempo occorrente

alla cottura ed alla preparazione del piatto e perciò tale esempio é sicuramente un'azione descrivibile.

Le azioni possono essere molto semplici o più complesse.

Definizione. Un'azione si chiama **azione elementare,** o **azione basica**, se non può essere suddivisa in azioni più semplici, altrimenti si chiama **azione composta**, o **processo**. ■

L'azione di premere un interruttore elettrico per accendere una lampada può essere considerata un'azione elementare. L'azione del costruire una casa, invece, comporta numerose fasi e sottofasi e, quindi, non é un'azione elementare ma un'azione composta o, detto con altre parole, un *processo*.

Non necessariamente le azioni devono essere svolte in sequenza. Un solo esecutore, però, non può eseguire due o più azioni contemporaneamente e perciò deve svolgerle una per volta e, quindi, una dopo l'altra in sequenza.

Definizione. Un processo si dice **sequenziale** se é composto da azioni semplici che devono essere eseguite una dopo l'altra. ■

Più esecutori che collaborano alla soluzione di uno stesso problema possono, però, svolgere le azioni in parallelo e, quindi, contemporaneamente. In tal caso il **processo** si chiama **non sequenziale**.

Esempio – Costruzione di un edificio. *Durante la costruzione di un edificio spesso sono eseguiti processi non sequenziali; ad esempio, mentre gli elettricisti realizzano l'impianto elettrico, gli idraulici realizzano gli impianti idrici e di riscaldamento.* ■

Nota bene. Nel mondo dell'Informatica (in particolare, quando si parla di sistemi operativi), intendiamo talvolta per processo l'insieme delle attività svolte dal **singolo** esecutore. In questo caso parliamo anche di **processi sequenziali concorrenti,** intendendo con questo l'insieme dei processi sequenziali svolti da più esecutori che concorrono alla soluzione del medesimo problema.

4.2 Il Processo di Delega

Quando l'uomo primitivo si é evoluto costituendo le società, non essendo più solo, ha avvertito la necessità di delegare la soluzione di una parte dei propri problemi ad altri uomini. É nato così il **processo di delega,** che tuttora é alla base dell'organizzazione della società.

Consideriamo, ad esempio, un'organizzazione strutturata in modo gerarchico quale l'Università. Al vertice di questa organizzazione troviamo il Magnifico Rettore, il quale delega ai docenti la risoluzione del problema di istruire gli studenti, al personale amministrativo la gestione contabile e legale dell'ente, al personale ausiliario il controllo degli accessi agli stabili dell'Università e così via…

Il processo di delega, però, non è usato solo nelle società di tipo gerarchico. Infatti, se un docente ha come problema da risolvere quello di procurarsi del pane, egli va dal fornaio e ne acquista un pezzo, ovvero il fornaio ha la delega a produrre del pane da parte del docente. Si noti che non esiste un rapporto di gerarchia tra il docente ed il fornaio; infatti, ad esempio, il docente ha la delega da parte del fornaio di istruire i figli.

È evidente che questo tipo di processo di delega non é dettato dal rapporto di gerarchia, ma deriva dalla tendenza della società alla **specializzazione**.

A questo proposito occorre menzionare il filosofo **Silvio Ceccato** (1914 - 1997), uno dei padri fondatori della cibernetica in Italia, che cercò di applicare la cibernetica all'attività linguistica dell'uomo. Ceccato affermava che "*l'uomo é un animale pigro*"; infatti, tutte le sue scelte sono dettate dalla pigrizia. Molte persone, ad esempio, preferiscono percorrere sempre la stessa strada per ritornare a casa per la forza dell'abitudine (pigrizia), anche se sono consapevoli che esistono percorsi più brevi. Probabilmente il **processo di delega** é dettato semplicemente dalla **pigrizia** dell'uomo!

Nel processo di **delega** sono sempre individuabili due figure: il **committente**, ovvero colui che ha un problema, e l'**esecutore**, ovvero colui a cui é delegata la risoluzione del problema. Tra i due é necessaria l'esistenza di un **canale** che consenta al committente di:

- comunicare **l'istanza del problema** da risolvere all'esecutore e,
- successivamente, all'esecutore, di fornire **la soluzione** al committente.

In termini giuridici un'istanza é una domanda scritta per chiedere il riconoscimento di un diritto, come ad esempio l'istanza di fallimento, l'istanza di divorzio, l'istanza di autocertificazione, ecc. In informatica il termine **istanza** indica semplicemente un caso particolare. Ha senso pertanto parlare di istanza di un problema, intendendo con questo un caso particolare del problema.

In un ristorante, ad esempio:

- il cliente (*committente*) ordina il pasto al cameriere (*canale di comunicazione*);
- il cameriere trasporta l'ordine (*istanza del problema*) al cuoco (*esecutore*);
- il cuoco cucina il pranzo (*risolve il problema*);
- infine il cuoco consegna al cameriere la pietanza (*soluzione*) perché la serva al cliente.

Pertanto il processo di delega può essere schematizzato come segue:

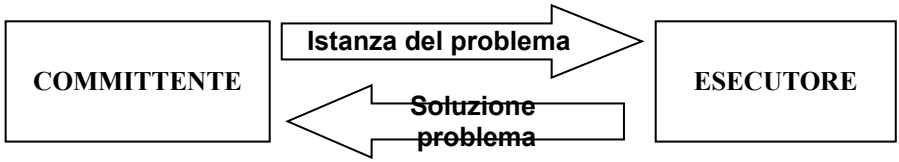


Fig. n. 4.1 – Schema del Processo di Delega

4.3 La Descrizione di un Processo

Osserviamo adesso il processo da un altro punto di vista, quello di un **osservatore esterno,** che osserva dettagliatamente l'esecutore al lavoro e annota i vari passaggi, ovvero tutte le attività che esso svolge per portare a termine il compito assegnato.

Consideriamo, ad esempio, un telespettatore che assiste ad un programma televisivo di cucina. Mentre il cuoco prepara in diretta la crema pasticciera, il telespettatore, senza avere alcuna influenza sul cuoco e sulla trasmissione, osserva le fasi dell'operazione prendendone nota e descrivendole attraverso l'uso del modo **indicativo:**

- *il cuoco prende tre uova,*
- *il cuoco rompe le tre uova,*
- *il cuoco mescola le tre uova a 150 gr di zucchero,*
- *ecc …*

Il testo scritto dal telespettatore rappresenta la **traccia del processo**.

Definizione. La **traccia del processo** é la descrizione di una istanza del processo.■

Ad ogni istanza del processo cambia la traccia del processo, perché cambia il suo effetto. Se, ad esempio, il cuoco cambia il tipo d'acqua (più dura, meno dura) per la cottura della pasta, o cambia il tipo di pasta utilizzato, cambierà sicuramente il tempo di cottura e, quindi, la traccia del nuovo processo non sarà più la stessa.

La traccia di un processo deve, quindi, necessariamente contenere:

- la lista dei dati;
- la sequenza delle azioni e dei controlli che scandiscono l'ordine di esecuzione delle azioni, presupponendo un tempo finito per ogni esecuzione. Ad esempio:
    - Il cuoco **adesso** verifica il grado di cottura della carne;
    - il cuoco **adesso** verifica se l'acqua bolle;

Nell'esempio precedente, l'istanza del problema non é

- *preparare la crema pasticciera utilizzando le uova, lo zucchero, la farina, il latte o la buccia di limone*

bensì

- *preparare la crema pasticciera utilizzando tre uova, 150 grammi di zucchero, 80 grammi di farina, mezzo litro di latte e la buccia di un limone.*

Perciò ogni singola istanza é caratterizzata da una **lista di dati,** specifici del problema. Nell'esempio precedente, tale lista é costituita da:

- *tre uova;*
- *150 grammi di zucchero;*
- *80 grammi di farina;*
- *mezzo litro di latte;*
- *la buccia di un limone.*

Esempio - Addizione di due numeri interi positivi. *Nel caso in cui si vogliono sommare i numeri "12" e "11", i dati di questa specifica istanza del problema sono i numeri interi "12" e "11", mentre il numero "23" costituisce la soluzione del problema.* ■

Esempio - Ordinamento di una sequenza di numeri interi. *Un'istanza del problema ORDINAMENTO è, ad esempio:*

ordina la sequenza "3, 8, 6, 0, 2".

In questo caso la lista dei dati di tale istanza é costituita dalla sequenza "3, 8, 6, 0, 2", mentre il risultato é costituito dalla sequenza "0, 2, 3, 6, 8". ■

Data la traccia di un processo non siamo in grado di replicare il processo, se non utilizzando gli stessi dati. Ritornando all'esempio della preparazione della crema pasticciera, il cuoco non riuscirà mai a replicare esattamente il processo partendo dalla traccia del processo (ottenuta dall'osservazione delle operazioni del cuoco nella trasmissione televisiva), poiché i dati non saranno mai esattamente identici: infatti, possono cambiare la temperatura dei fornelli, la qualità degli ingredienti, … La crema preparata sarà inevitabilmente diversa in qualche modo da quella prodotta durante la trasmissione.

Pertanto, se un committente deve delegare l'esecutore per la risoluzione di un problema, egli o qualcun altro per lui, dovrà "*istruire*" l'esecutore. Non é sufficiente fornire *la traccia* di un'istanza del processo, per esempio le operazioni da effettuare per addizionare i numeri "12" e "11", perché in tal modo l'esecutore riuscirebbe a risolvere il problema "*addizione*" solo per questi due numeri, ma occorre fornire un **metodo generale di risoluzione del problema**. Con questo intendiamo il **comportamento** che dovrà avere **l'esecutore del processo**, quando si presenta una qualsiasi istanza del problema da risolvere. L'esecutore deve essere in grado, infatti, di risolvere tutti i possibili casi del problema, e non un solo caso specifico.

Il modo verbale che caratterizza l'istruzione dell'esecutore é l'**imperativo.** Ad esempio, al cuoco, ovvero all'esecutore della preparazione della crema pasticciera, il comportamento sarà descritto con "*prendi le uova, rompi le uova, mescola le uova allo zucchero, ecc.*".

Per mettere l'esecutore in condizione di poter fare fronte a tutte le possibili situazioni che dovessero presentarsi durante la risoluzione delle diverse istanze di un problema, devo utilizzare il modo **condizionale:** se si verifica una certa condizione fai questa operazione. Ad esempio:

- se l'acqua bolle (*condizionale*),
  butta la pasta (*imperativo*);
- se la pasta é cotta (*condizionale*),
  togli la pasta dal fuoco (*imperativo*).

L'imperativo ed il condizionale sono i modi verbali che caratterizzano il metodo di risoluzione del problema:

- utilizzando l'imperativo specifico la sequenza delle operazioni che l'esecutore deve necessariamente effettuare,
- utilizzando il condizionale metto in condizione l'esecutore di prendere delle decisioni e di effettuare delle scelte quando si trova davanti a determinate condizioni.

4.4 L'Algoritmo

Nota Storica

La parola algoritmo ha origine nel Medio Oriente. Essa proviene dall'ultima parte del nome dello studioso persiano **Abu Jàfar Mohammed Ibn Musa Al-Khowarizmi (**780 - 850**)**, il cui testo di Algebra *Hisab al-jabr w'al-muqabala* (825 d.C. circa) ha esercitato una profonda influenza nei secoli successivi. Il termine algebra deriva proprio dalla parola araba "*al-jabr*".

Riportiamo di seguito alcune osservazioni sugli algoritmi dovute a D.E. Knuth (1938 - vivente), uno dei padri fondatori dell'Informatica.

> *"Tradizionalmente gli algoritmi erano impiegati esclusivamente per risolvere problemi numerici. Tuttavia, l'esperienza con i calcolatori ha dimostrato che i dati possono virtualmente rappresentare qualunque cosa.*
>
> *Di conseguenza, l'attenzione della scienza dei calcolatori si é trasferita allo studio delle diverse strutture con cui si possono rappresentare le informazioni, e all'aspetto ramificato o decisionale degli algoritmi, che permette di eseguire differenti sequenze di operazioni in dipendenza dello stato delle cose in un particolare istante.*
>
> *É questa la caratteristica che rende talvolta preferibili, per la rappresentazione e l'organizzazione delle informazioni, i modelli algoritmici a quelli matematici tradizionali".*

4.4.1 La definizione di Algoritmo

Un **algoritmo** é un metodo di risoluzione di un problema godente delle seguenti proprietà:

- **finitezza della descrizione**, ovvero, la sua descrizione deve essere finita;
- **non ambiguità**, ovvero, non devono essere ammesse più interpretazioni da parte dell'esecutore;
- **finitezza del tempo di esecuzione**, ovvero, l'esecutore deve effettuare un numero di passi (azioni basiche e controlli) finito;
- **generalità**, ovvero, l'algoritmo deve permettere di risolvere qualsiasi istanza del problema.

Consideriamo alcuni esempi di metodi di risoluzione di problemi, che non costituiscono degli algoritmi.

- Per risolvere il problema "*scrivere tutti i numeri naturali*" possiamo descrivere la soluzione nel seguente modo:

  "*scrivi 1*"
  "*scrivi 2*"
  "*scrivi 3*"
  …..

  Un tale metodo non costituisce però un algoritmo, perché la sua descrizione non è finita;
- Nella ricetta della crema pasticciera, quando il cuoco deve aggiungere la buccia di limone, non sa se la buccia deve essere grattugiata o semplicemente tagliata in striscioline. La ricetta non é, infatti, chiara su questo punto e pertanto l'algoritmo stesso é ambiguo;
- Un qualunque metodo di risoluzione del problema "*elencare tutti i numeri interi più grandi di un numero dato*" non potrà mai essere un algoritmo; poiché tali numeri sono infiniti, l'esecuzione di tale metodo richiederà un tempo infinito;
- Consideriamo il problema di "*cuocere la pasta al dente*". Se l'algoritmo per risolverlo non prevede l'azione "*leggi il tempo di cottura sulla confezione*", non sarà un algoritmo generale; infatti, ogni formato di pasta ha un proprio tempo di cottura e nel caso in cui l'algoritmo usasse lo stesso tempo di cottura per tutti ci sarebbero casi in cui la pasta è cotta al dente, ma ci sarebbero casi in cui la pasta sarebbe scotta o addirittura cruda.

All'inizio dell'esecuzione, l'esecutore riceve dal committente un insieme di **dati,** che rappresentano la descrizione della particolare istanza del problema che si intende risolvere: a tal fine deve essere definito un meccanismo con il quale

l'esecutore riceve dal committente l'**istanza** del problema. I dati appartengono ad un **insieme finito di simboli**.
Al termine dell'esecuzione, l'esecutore restituisce la soluzione al committente: a tal fine deve essere definito un meccanismo con il quale l'esecutore comunica al committente la **soluzione.**

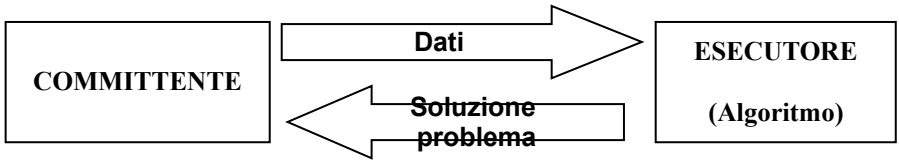


Fig. n. 4.2 – ESECUZIONE DI UN ALGORITMO PER UNA ISTANZA DEL PROBLEMA.

È importante osservare che il tempo di esecuzione di un algoritmo è finito se e solo se è finita la traccia del processo.

4.4.2 Descrizione formale del problema

Una descrizione informale del problema che si intende risolvere può essere molto utile in un primo momento, ad esempio, per chiarirsi le idee. Molto spesso sappiamo che un certo problema esiste, tuttavia non siamo in grado di formalizzarlo esattamente.

Questo é il classico caso che si presenta ai medici, ai quali molti pazienti chiedono direttamente la prescrizione di un dato medicinale (collirio), invece di spiegare i sintomi del malessere (bruciore agli occhi).

Fig. n. 4.3 – Il Medico ed il Paziente

Questo é purtroppo anche un problema cruciale dell'Ingegneria del Software: spesso, infatti, il committente di un sistema informativo tenta di comunicare all'informatico la soluzione del problema che ha, anziché comunicarne la sua descrizione!

Una descrizione informale del problema può essere sufficiente nel caso in cui l'esecutore sia un essere dotato di capacità cognitiva simile alla nostra e che condivida con noi:

- *l'esperienza* (ad esempio possiamo dire ad un nostro collaboratore "sbrigami la pratica del 3 dicembre", ed il nostro collaboratore saprà esattamente cosa fare!);

- *il senso comune* (ad esempio posso dire ad uno sconosciuto: "*mi scusi, che ora è?*" e lo sconosciuto sarà in grado di rispondermi, senza chiedermi: "*in quale parte del mondo?*").

Nell'attesa di costruire degli oggetti artificiali dotati della nostra esperienza o del nostro senso comune (questo é uno degli obiettivi dell'**Intelligenza Artificiale**), occorre specificare esattamente, in una prima fase, la natura del problema da risolvere.

Occorre far attenzione a non confondere la natura del problema da risolvere con la descrizione del metodo di risoluzione del problema.

ESEMPIO - ORDINAMENTO DI UNA SEQUENZA DI NUMERI INTERI. *In precedenza il problema é stato presentato in modo informale. Una sua* ***definizione formale*** *può essere data invece nel seguente modo:*

- ***Dati:*** *Una sequenza di n numeri interi positivi* $a_1, \ldots a_n$ *più piccoli di un limite prefissato (ad esempio 1000).*
- ***Risultato:*** *Una sequenza* $b_1, \ldots b_n$, *di numeri interi positivi, che costituisce una permutazione (ovvero un riarrangiamento) della sequenza iniziale, ma che soddisfa la relazione* $b_i < b_{i+1}$ *per ogni* i. ■

4.4.3 Descrizione del comportamento dell'Esecutore

Poiché l'esecutore in genere non dispone di intelligenza propria, occorre *insegnargli, comunicargli, trasmettergli* un *metodo di risoluzione del problema*, quindi descrivere il comportamento che dovrà assumere in tutte le situazioni in cui si potrà trovare.

É lo stesso metodo utilizzato da una maestra che insegna ad un bambino come svolgere un'addizione, quando il bambino non sa neppure cos'è un numero. Il metodo di risoluzione del problema é quindi un **metodo costruttivo,** perché descrive esattamente le operazioni da compiere quando "il bambino" (*esecutore*) incontra un'istanza del problema (*due interi da sommare*).

Nota bene. Si deve fare attenzione a non confondere il problema da risolvere con la descrizione del metodo di risoluzione del problema.

Nel campo dell'Informatica ci riferiamo all'esecutore che indica un'entità che svolge un compito al posto nostro (*il calcolatore elettronico*, ad esempio) utilizzando il termine **automa** *(parola di origine russa, che significa **schiavo**)*.

Esempio - Addizione di due numeri interi positivi. *Vediamo ora come sia possibile insegnare a svolgere un'addizione semplice (senza alcun riporto) ad un bambino (automa) senza che questi conosca il concetto di numero.*

*Innanzitutto, la maestra mostra ad un bambino della sua classe i simboli: "0", "1", "2", "3", "4", "5", "6", "7", "8" e "9". Se la maestra chiede al bambino di riprodurre uno di questi simboli, egli sarà attento a riprodurre esattamente la forma del simbolo. Per lui, infatti, i simboli "*6*", "6*" e "**6***" sono diversi tra loro, perché non conosce la semantica (significato) associata al simbolo: il simbolo "6" é solo un segno al quale il bambino di 3 anni non associa alcun significato.*

Per consentire al bambino (automa) di svolgere un'addizione semplice, senza alcun riporto, quale ad esempio:

ESERCIZIO

$$\begin{array}{cccc} & 1 & 3 & 3 & + \\ 3 & 2 & 2 & 1 & = \\ \hline \end{array}$$

la maestra gli fornisce la seguente tabella:

Tabella di riferimento per l'addizione

+	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

Sarà, quindi, sufficiente insegnare al bambino che ogni volta in cui gli viene richiesto di svolgere una certa addizione, dovrà seguire le seguenti operazioni:

1) *trascrivi le due sequenze di segni che ti sono forniti (primo addendo e secondo addendo) in colonne allineate a destra;*
2) *partendo dalla colonna di destra confronta il segno superiore con i segni che compaiono nella prima riga in alto della tabella di riferimento, poi confronta il segno inferiore con i segni della prima colonna di sinistra della tabella di riferimento, quindi identifica il segno posto in corrispondenza riga-colonna;*
3) *trascrivi il segno individuato in colonna rispetto ai segni considerati;*
4) *passa alla colonna a sinistra e ripeti le operazioni descritte per la prima colonna;*
5) *se in una delle due righe dell'operazione non é presente alcun segno, assumi che è presente il segno "0";*
6) *se non trovi alcun segno nella colonna, arrestati.*

ESERCIZIO

$$\begin{array}{cccc} & 1 & 3 & 3 & + \\ 3 & 2 & 2 & 1 & = \\ \hline \end{array}$$

3 3 5 4

Il bambino sarà in grado così di risolvere il problema, restituendo una soluzione che per lui non é altro che una sequenza di segni, mentre la maestra, che associa un significato a quei simboli, interpreterà la sequenza "3", "3", "5", "4" come il numero intero "3354". ■

4.4.4 Azioni e Controlli

All'automa **non é richiesta intelligenza**, perché **lavora a livello sintattico** e non semantico: occorre esclusivamente che esso sia in grado di compiere delle azioni elementari e dei semplici controlli.

Le azioni di un automa devono avere un tempo di esecuzione finito. Inoltre, se un automa reale (un bambino, un calcolatore elettronico, ecc.) deve sommare due cifre, é scontato che il tempo dell'operazione deve essere non solo finito, ma possibilmente anche breve.

Nota bene. Le azioni elementari ed i controlli svolti dall'automa sono **attività discrete.**

Ricordiamo che una **attività discreta** é individuata da due **eventi**, quello con il quale inizia e quello con il quale termina. Un **evento** rappresenta un cambiamento di stato che avviene istantaneamente e, quindi, non richiede nessun tempo di esecuzione. Ad esempio, l'attività della "*preparazione della crema pasticciera*" é discreta, in quanto inizia con l'evento "*il cuoco prende le uova*" e finisce con l'evento "*il cuoco termina la cottura della crema*". Al contrario, il sorgere del sole con il graduale amento della luminosità e poi il tramonto con la graduale diminuzione di luminosità, sono esempi di **attività continue**, poiché dalla condizione di oscurità a quella di illuminazione ci sono innumerevoli situazioni intermedie.

Ritornando all'esempio dell'addizione di due numeri, il bambino svolge delle azioni discrete, quali:

- controlla un segno presente nella riga superiore;
- controlla il segno corrispondente nella riga sottostante;
- individua nella tabella dell'addizione il segno corrispondente;
- trascrive il risultato dell'operazione;
- ecc.

Anche i controlli (ad esempio, quando il bambino controlla se sono presenti altri segni da trattare), ricordiamo, devono avvenire in tempo finito.

Con un automa pensiamo di risolvere i problemi di natura simbolica, cioè problemi che consistono nella manipolazione di simboli. **L'automa, infatti, accetta sequenze di simboli e restituisce sequenze di simboli.**

4.4.5 Il modello dell'Automa

Per spiegare le trasformazioni simboliche che un automa é in grado di effettuare, occorre definire il **modello dell'automa**. Con questo intendiamo, informalmente, una descrizione dell'insieme delle azioni basiche e dei controlli che l'automa é in grado di svolgere.

Ad esempio, affermare che un certo automa (il *bambino*) é in grado di effettuare un certo numero di azioni elementari e di controlli costituisce un modello dell'automa bambino.

Definizione. Un **modello di calcolo** é semplicemente un'astrazione di un esecutore reale, in cui si omettono dettagli irrilevanti allo studio di un algoritmo per risolvere un problema. ■

Esistono tanti differenti modelli di calcolo (che vedremo prossimamente, quali la *Macchina di Turing, gli Automi a Stati Finiti,* ecc.).

L'adozione di un particolare modello di calcolo, piuttosto che di un altro, al fine di studiare un algoritmo, dipende da vari fattori:

- *capacità espressiva del modello in relazione al problema assegnato*:
  sta ad indicare che un certo modello é preferibile ad un altro per esprimere la soluzione algoritmica di un determinato problema;
- *livello di astrazione*:
  è tanto maggiore quanto maggiore é la quantità di dettagli che sono omessi;
- *generalità*:
  esistono modelli più generali di altri. Diremo che un modello é **più generale** di un altro (che equivale a dire **più potente** di un altro) se tutti i problemi risolubili con il secondo sono risolubili anche con il primo.

Uno dei risultati più sorprendenti della Teoria della Computabilità riguarda *l'esistenza di modelli di calcolo assolutamente generali.* Uno di questi é la **Macchina di Turing** che vedremo prossimamente.

4.5 Il Concetto Matematico di Problema e di Algoritmo

In precedenza abbiamo detto che in Informatica il concetto di problema non può definirsi se non con dei sinonimi. In Matematica, invece, il problema è definito utilizzando il concetto di **funzione**.

Ricordiamo che una funzione non é altro che una *regola di corrispondenza* che associa ad ogni elemento di un insieme detto insieme di partenza (detto *dominio*), un elemento di un altro insieme detto insieme di arrivo (o *codominio*). Se indichiamo con *x* l'elemento dell'insieme di partenza e con *f* la funzione, allora l'elemento associato ad *x* sarà denotato *f(x)*.

Esempio – La funzione Compleanno. *Supponiamo che l'insieme di partenza sia costituito da tutte le persone e l'insieme di arrivo dai giorni dell'anno. Definiamo allora la funzione f in modo da associare ad una persona il proprio compleanno: se x é il mio vicino di casa Mario Rossi, allora f(x) sarà uguale a "24 gennaio".* ■

Definizione. Un **problema é** una funzione **P** che associa ad ogni elemento di un insieme di partenza **I,** che chiamiamo insieme delle **istanze**, un elemento di un insieme di arrivo **S,** che chiamiamo insieme delle **soluzioni**. ■

La funzione é, quindi, una *regola di corrispondenza* che associa ad ogni istanza di un problema la relativa soluzione. Ad esempio, considerando il problema posto precedentemente "*addizione di due numeri interi*", ad ogni istanza del problema somma corrisponde la relativa soluzione.

Un "*metodo di risoluzione del problema*" indica all'esecutore come passare effettivamente dall'insieme delle istanze all'insieme delle soluzioni.

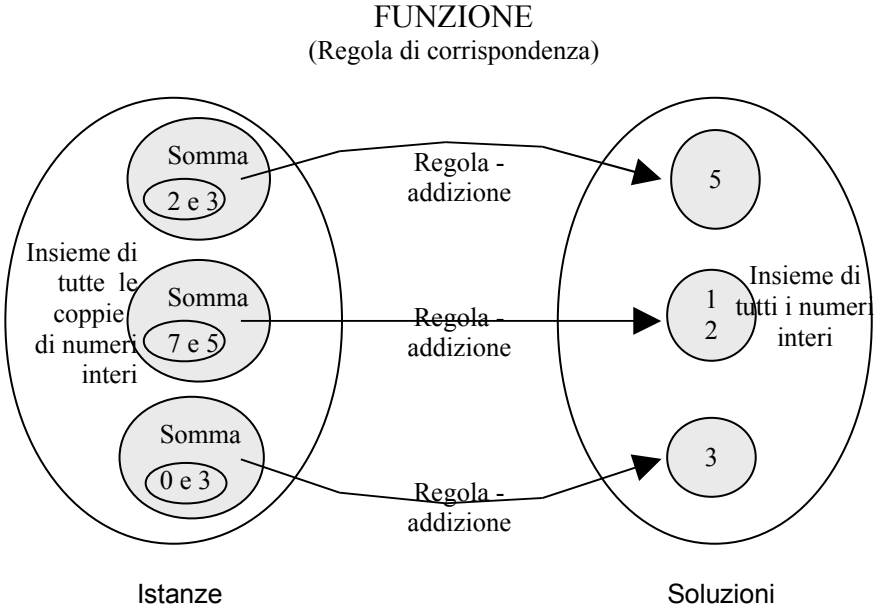


Fig. n. 4.2 – IL PROBLEMA ADDIZIONE DAL PUNTO DI VISTA MATEMATICO

Un algoritmo ***A*** quindi:

- definisce una funzione che associa ad ogni valore in ingresso x il corrispondente valore in uscita ***A(x)***, ovvero una regola di corrispondenza, dall'insieme delle istanze del problema all'insieme delle soluzioni;
- **ma**, attenzione, **indica al tempo stesso in maniera costruttiva come determinare, data un'istanza del problema (dati di input), la corrispondente soluzione (output).**

Definizione. Un **algoritmo** ***A*** risolve un problema ***P*** se ***P(x)*** = ***A(x)***, per ogni istanza ***x*** assegnata del problema **P**. ■

4.6 Il Programma

L'algoritmo é definito informalmente come la descrizione delle operazioni che occorre compiere per conseguire la soluzione di un problema. È, quindi, un qualcosa che esiste esclusivamente nella nostra mente. Nel momento in cui l'algoritmo è messo "*nero su bianco*" non é più un algoritmo, ma una sua rappresentazione.

Se quindi ho un algoritmo in mente e voglio trasmetterlo ad un automa per fornirgli un metodo di risoluzione di un problema, sono costretto a **codificarlo** (ovvero, a **rappresentarlo**) utilizzando un linguaggio comprensibile da entrambi. Riconosciamo dunque un caso particolare del processo comunicativo definito da Shannon!

Nel caso in cui ho a che fare con un calcolatore elettronico (*automa*) il linguaggio si chiama **linguaggio di programmazione**, e l'attività di **codifica** (o **rappresentazione**) che svolgo si chiama **programmazione**, o più precisamente **implementazione dell'algoritmo**.

Definizione. Un **programma** é la **codifica** (o **rappresentazione**) di un algoritmo, utilizzando un **linguaggio di programmazione** opportunamente specificato. ■

A proposito dei programmi, D.E. Knuth afferma che:

> *Un programma é l'esposizione di un algoritmo in un linguaggio accuratamente definito. Quindi, il programma di un calcolatore rappresenta un algoritmo, per quanto l'algoritmo stesso sia un costrutto intellettuale che esiste indipendentemente da qualsiasi rappresentazione. Allo stesso modo, il concetto di numero 2 esiste nella nostra mente anche quando non sia espresso graficamente.*

Programmi per risolvere problemi numerici sono stati scritti sin dal 1800 A.C., quando i matematici babilonesi del tempo di Hammurabi hanno stabilito delle regole per la risoluzione di alcuni tipi di operazioni. Le regole erano determinate come procedure passo-passo, applicate sistematicamente ad esempi numerici particolari.

4.6.1 I costi

Ad ogni programma di calcolo sono associati dei costi:

- L'*Ingegneria del Software* si occupa, tra l'altro, di minimizzare i costi relativi allo sviluppo (analisi del problema, progettazione, implementazione) dei programmi ed alla loro successiva manutenzione.
- La *Teoria della Complessità Computazionale*, invece, é la branca dell'Informatica che si occupa di minimizzare i costi di esecuzione.

I costi relativi allo sviluppo ed alla manutenzione dei programmi sono misurati in anni-uomo.

Definizione. Un **anno-uomo** indica quanto un dipendente a tempo pieno è pagato per tutto l'anno per il suo lavoro. ■

Per poter parlare di costi di esecuzione, occorre prima di tutto definire il concetto di risorse di calcolo.

Definizione. Le **risorse di calcolo** a disposizione del programma sono:

1. Il **Tempo** utilizzato per eseguire l'algoritmo;
2. Lo **Spazio di lavoro** utilizzato per memorizzare i risultati intermedi;
3. Il **Numero degli esecutori**, se più esecutori collaborano per risolvere lo stesso problema. ■

Tale classificazione delle risorse é totalmente indipendentemente dalla sua interpretazione informatica.

Esempio. *In un ufficio lo spazio di lavoro è rappresentato dai moduli cartacei, gli esecutori dagli impiegati mentre il tempo dalle giornate lavorative.* ■

Qualora ci si riferisca al **campo specifico dell'informatica**:

- lo spazio di lavoro diventa la **memoria** del calcolatore;
- il numero degli esecutori diventa il **numero dei processori** a nostra disposizione, in un sistema dotato di più processori.
- il tempo di esecuzione non é misurato in minuti o secondi, ma contando il **numero di passi**, ovvero il numero delle azioni basiche più il numero di controlli, necessari a svolgere l'operazione.

Definizione. Il **costo relativo all'esecuzione di un programma** è definito come la quantità di risorse di calcolo che il programma utilizza durante l'esecuzione. ■

In particolare, il numero di passi svolti per risolvere una determinata istanza del problema costituisce il **costo temporale dell'algoritmo**.

Definizione. Un algoritmo é **efficiente** se fa un uso contenuto (ovvero parsimonioso) delle risorse a sua disposizione. ■

È molto importante saper valutare la quantità di risorse consumate, poiché un consumo eccessivo di risorse può pregiudicare la possibilità di utilizzo di un algoritmo.

Dati due algoritmi per risolvere lo stesso problema P, può accadere che il primo faccia un uso più efficiente della risorsa spazio, mentre il secondo privilegi la risorsa tempo. Quale algoritmo è preferibile scegliere?

> *Tra le due risorse spazio e tempo, il tempo va quasi sempre privilegiato. Infatti, lo spazio è una risorsa riutilizzabile, mentre il tempo non lo è.*

4.6.2 Irrisolubilità algoritmica

Può sembrare strano, nell'epoca in cui viviamo, parlare di problemi di natura simbolica che non possano essere risolti con l'ausilio di un calcolatore elettronico! Purtroppo ciò avviene, e molto più spesso di quanto si pensi.

Definizione. Diciamo che un problema é algoritmicamente non risolubile se non può essere risolto, **né ora né mai,** ricorrendo alla nozione attuale di algoritmo. ■

Esempio - Il Problema del Tassellamento o "Tiling Problem", definito da **Martin Gardner** nel gennaio 1977 in un articolo sulla rivista *Scientific American*. *Dato un numero infinito di mattonelle di forma esagonale identiche, é sempre possibile ricoprire un piano senza lasciare spazi vuoti. Lo stesso si può dire per mattonelle di forma triangolare o rettangolare, ma non per quelle di forma di pentagono regolare. Nel caso più generale, ci domandiamo se, data una forma geometrica, sia possibile ricoprire il piano utilizzando mattonelle aventi tale forma.* ■

Nel 1966 **R. Berger** ha dimostrato che tale problema é algoritmicamente non risolubile, ovvero, non potrà esistere né ora né mai una soluzione algoritmica a tale problema utilizzando i modelli di macchine calcolatrici esistenti.

L'affermazione *"né ora, né mai"* nella definizione precedente potrebbe sembrare piuttosto forte, o addirittura senza senso: chi ci assicura, infatti, che un domani, un problema giudicato oggi algoritmicamente irresolubile, non possa essere risolto algoritmicamente, ricorrendo alla nozione attuale d'algoritmo? Ce lo assicura la Logica Matematica, attraverso una **dimostrazione** della **irrisolubilità algoritmica** di un determinato problema.

4.6.3 Intrattabilità algoritmica

Definizione. Un problema è intrattabile se qualunque algoritmo che lo risolva richiede una quantità molto elevata di risorse. ■

La logica matematica fornisce alla teoria degli algoritmi gli strumenti per riconoscere e classificare i problemi intrattabili.

Problemi intrattabili sorgono, ad esempio, in giochi quali la dama o gli scacchi.

4.6.4 Conclusioni

Abbiamo parlato di problemi ed algoritmi, ma anche di irrisolubilità algoritmica e di intrattabilità. Può sembrare strano, ma la maggior parte dei problemi di tipo simbolico a cui siamo posti di fronte ogni giorno sono intrattabili o irrisolubili. Per questo motivo, riteniamo che le grandi corporazioni dell'informatica ci propongono sempre gli stessi pacchetti, atti a risolvere i medesimi problemi (archiviazione di grandi quantità di dati, ecc.). Sono gli unici problemi che al giorno d'oggi siamo in grado di risolvere, efficientemente!

5

La Rappresentazione dell'Algoritmo

Nel capitolo precedente si è parlato di **algoritmo** come metodo di risoluzione dei problemi. Ricordiamo che l'algoritmo è un concetto astratto e, per comunicarlo, abbiamo bisogno di codificarlo, di rappresentarlo.

Ritornando al problema dell'addizione di due numeri interi, la maestra non fa altro che codificare l'algoritmo utilizzando frasi della lingua italiana. Le azioni che il bambino deve compiere potrebbero essere rappresentate in forma testuale oppure attraverso dei disegni. Sono tutte rappresentazioni dello stesso algoritmo.

Rappresentare un algoritmo utilizzando un linguaggio testuale equivale ad utilizzare una **rappresentazione lineare**, perché la rappresentazione sarà costituita da una sequenza (da cui il termine *lineare*) di simboli.

Esempio – La Rappresentazione della Divina Commedia. *Potremmo rappresentare la Divina Commedia utilizzando come supporto una strisciolina di carta (un coriandolo), magari lunga alcune centinaia di chilometri, trascrivendo un carattere dopo l'altro. Avremmo bisogno di un simbolo per rappresentare lo spazio, che separa parole consecutive, e di un simbolo che ci indichi la terminazione di un periodo (quando scriviamo utilizzando un foglio di carta e la penna, noi andiamo a capo per indicare che un periodo termina e inizia un altro, e non per soddisfare il nostro senso estetico!).* ■

Si noti che questo sistema è proprio quello utilizzato dai telegrafi, che codificano l'informazione su un nastro di carta utilizzando come alfabeto l'alfabeto Morse, in cui ciascun simbolo è rappresentato da punti e linee consecutivi.

Si noti anche che quando esprimiamo (rappresentiamo) dei concetti parlando, utilizziamo una rappresentazione lineare costituita da una sequenza di parole (suoni) che si susseguono nel tempo.

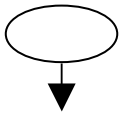
Nelle prossime pagine vedremo una **rappresentazione bidimensionale degli algoritmi**, una delle prime forme di rappresentazione degli algoritmi storicamente utilizzata. Con il termine bidimensionale intendiamo dire che occorre uno spazio a due dimensioni (il piano, ovvero, in pratica, un foglio di carta) per tale rappresentazione.

5.1 I Diagrammi di Flusso

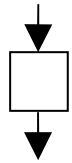
I diagrammi di flusso, o flowchart, sono una rappresentazione bidimensionale del flusso di controllo degli algoritmi, utilizzando un linguaggio grafico (ovvero *visuale*) definito da regole ben precise. Informalmente, con la parola flusso di controllo indichiamo la relazione temporale che deve esistere tra le azioni e le condizioni utilizzate per descrivere l'algoritmo (*relazione temporale* sta per "*dopo aver fatto questo, cosa mi tocca fare?*").

I diagrammi di flusso sono stati introdotti dagli informatici negli anni '60 e sono stati poi applicati in tanti altri contesti, come ad esempio in campo medico per i protocolli diagnostici: un medico prescrive delle analisi (controllo) e, a seconda dell'esito, prescrive una certa cura (azione) oppure un'altra analisi, e così via.

I simboli basilari di tale linguaggio sono:

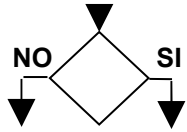


Il simbolo iniziale, che indica l'inizio di un processo. L'unica freccia presente è in uscita, che indica la strada da intraprendere all'inizio del processo.

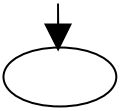


Il simbolo dell'azione, utilizzando un rettangolo, che rappresenta l'azione da compiere, con due frecce, una entrante ed una uscente. Le frecce rappresentano il punto di ingresso ed il punto di uscita dall'azione, ovvero la strada che porta all'azione e la strada da intraprendere una volta terminata l'azione. All'interno del rettangolo vi è la *specifica dell'azione*, ovvero la sua descrizione.

Il simbolo del controllo, dove il rombo rappresenta il controllo da effettuare, la freccia in alto rappresenta il punto di ingresso (la strada che porta al controllo), le frecce laterali rappresentano le strade da intraprendere quando si verificano due possibili alternative: l'esito del controllo può essere, solo ed esclusivamente, positivo o negativo.



All'interno del rombo deve essere definita la *specifica della condizione*, ovvero la sua descrizione.



Il simbolo della terminazione, che indica la fine del processo.
L'unica freccia presente è in entrata.

Questi quattro simboli costituiscono il lessico (ovvero il *vocabolario*) del nostro linguaggio, così come le parole della lingua italiana costituiscono il lessico della lingua italiana. Con essi è possibile rappresentare il flusso di controllo di un qualsiasi algoritmo, cioè è possibile rappresentare la successione delle operazioni da svolgere.

Quando formiamo le frasi della lingua italiana, dobbiamo rispettare alcune regole grammaticali. Allo stesso modo per formare le frasi del nostro linguaggio (ovvero, *i diagrammi di flusso*) dobbiamo seguire alcune **regole**, che costituiscono la grammatica dei diagrammi di flusso:

1. ogni freccia entra in un blocco o si innesta su un'altra freccia;
2. deve esistere un solo simbolo ovale di ingresso ed un solo simbolo ovale di terminazione;
3. da qualunque simbolo deve essere possibile raggiungere il simbolo della terminazione;
4. dal simbolo iniziale deve essere possibile raggiungere qualsiasi simbolo.

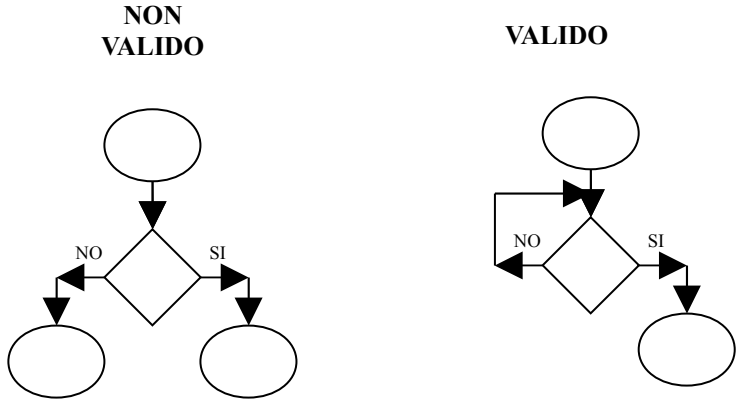


Fig. n. 5.1 – Esempi di Diagrammi di Flusso corretti e non

Il primo diagramma rappresenta l'inizio seguito da un controllo, dal quale si giunge a due terminazioni. Non è valido perché non rispetta la seconda regola.

Il secondo diagramma, invece, rappresenta l'inizio seguito da un controllo; al mancato verificarsi della condizione si torna, attraverso l'uscita negativa innestata sulla freccia di ingresso del controllo, a ripetere il controllo; mentre a condizione verificata si giunge all'unica terminazione attraverso l'uscita positiva. Pertanto non risulta violata nessuna delle regole sopra enunciate e, quindi, il diagramma è valido.

ESEMPIO - IMBOTTIGLIAMENTO DEL VINO. *L'algoritmo per la risoluzione del problema è rappresentato nella seguente figura.* ■

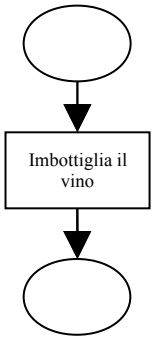


Fig. n. 5.2 – PRIMO DIAGRAMMA DI FLUSSO PER IL PROBLEMA *IMBOTTIGLIAMENTO DEL VINO*"

Nel diagramma di flusso della figura precedente dal simbolo iniziale passiamo direttamente all'azione "*Imbottiglia il vino*", quindi al simbolo di terminazione; ma un'azione di questo genere è un'azione complessa ovvero un processo, perché può essere suddivisa in più azioni elementari. L'algoritmo deve essere scomposto in azioni più semplici così come vediamo nella figura n. 5.3 (a).

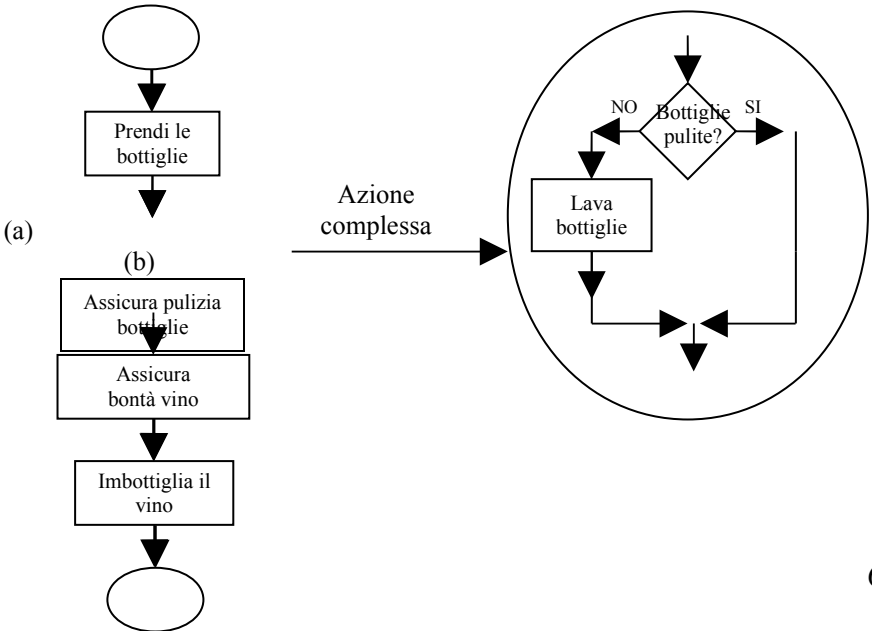


Fig. n. 5.3 - RAFFINAMENTO DEL DIAGRAMMA DI FLUSSO PER IL PROBLEMA "*IMBOTTIGLIAMENTO DEL VINO*"

L'azione "*Assicura che le bottiglie siano pulite*" non è ancora un'azione elementare, ma è composta da più azioni elementari così come possiamo vedere nella figura n. 5.3 (b). Al blocco di sinistra possiamo sostituire questo schema.

Occorre ripetere tale procedimento per tutte le altre azioni complesse, finché non si ottengono solo azioni elementari, ovvero azioni non suddivisibili in azioni più semplici.

Con i simboli e le regole precedentemente illustrati, è possibile produrre diagrammi complicati a piacere ed è proprio ciò che è stato fatto all'inizio, ottenendo diagrammi tanto intricati quanto illeggibili: gli americani parlavano di diagrammi di flusso "***spaghetti-like***" per indicare diagrammi di flusso ingarbugliati come un piatto di spaghetti.

Abbiamo già definito il lessico (i *simboli*) e la grammatica (le *regole*) del nostro linguaggio; come possiamo evitare di produrre diagrammi di flusso ingarbugliati e difficilmente interpretabili? Attraverso la definizione di uno **stile**, una sorta di semplice galateo della rappresentazione dei diagrammi di flusso.

Nella lingua italiana è possibile scrivere frasi contorte e macchinose, sebbene grammaticalmente corrette, per comunicare un concetto; ma è altrettanto possibile trasmettere il medesimo concetto attraverso una frase elegante, fluida e più facilmente comprensibile. Allo stesso modo nella realizzazione dei diagrammi di flusso si può adottare un metodo che ne permetta una rappresentazione stilisticamente ordinata ed elegante: parleremo di **diagrammi di flusso strutturati**, per indicare diagrammi di flusso che aderiscono a queste convenzioni stilistiche.

É importante notare che qualsiasi algoritmo può essere rappresentato con un diagramma di flusso strutturato, come asserisce il seguente **teorema**, dimostrato nel 1966 da due matematici italiani, **Corrado Böhm** (nato a Milano nel 1923, vivente) e **Giuseppe Jacopini** (scomparso recentemente, nel 2001), apparso nello storico articolo *Flow diagrams, Turing machines, and languages with only two formation rules,* pubblicato nel numero di maggio (1966) della rivista **Communications of the ACM**.

Teorema (Böhm e Jacopini). *Sia dato un algoritmo P e un diagramma di flusso che lo descrive. È sempre possibile determinare un algoritmo Q equivalente a P, che sia descrivibile con un diagramma di flusso strutturato.* ■

Informalmente, diciamo che due **algoritmi** sono **equivalenti** se producono lo **stesso effetto** a partire dallo **stesso insieme di dati** (ricordiamo che l'insieme dei dati descrive l'*istanza del problema* da risolvere).

La dimostrazione del precedente teorema non è semplice e, pertanto, è omessa in questo corso introduttivo. L'aspetto importante di tale dimostrazione consiste nell'aver definito un metodo costruttivo per rappresentare l'algoritmo in modo strutturato; infatti, i due studiosi italiani, **Böhm** e **Jacopini**, hanno illustrato e dimostrato com'è possibile rappresentare una qualunque azione complessa utilizzando solo tre schemi base (o schemi di composizione fondamentali), che prendono il nome di "**sequenza**", "**selezione**" e "**iterazione**" (o ripetizione).

5.2 Gli Schemi di Composizione Fondamentali

5.2.1 La Sequenza

Lo schema di sequenza è composto da più azioni (non necessariamente elementari), che devono essere eseguite una dopo l'altra, rispettando l'ordine indicato dalle frecce. La linea tratteggiata sta per "ci possono essere ulteriori blocchi nella sequenza".

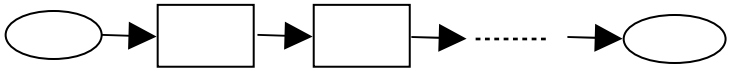
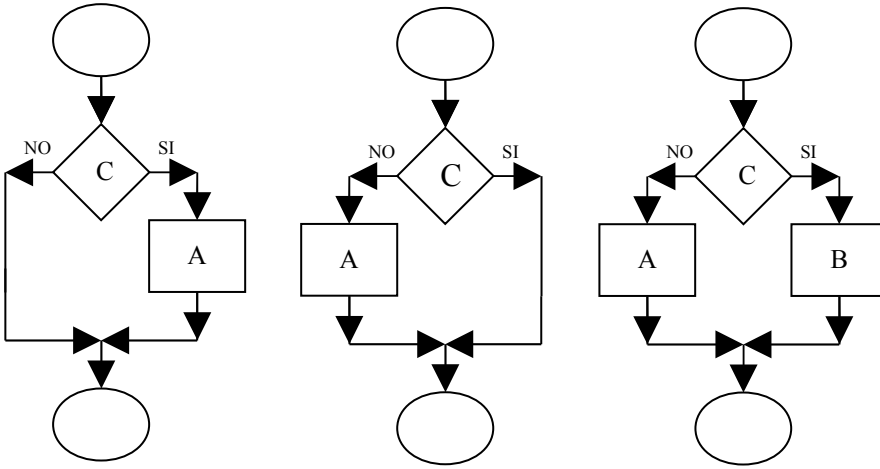


Fig. n. 5.4 - LA SEQUENZA

5.2.2 La Selezione

Lo schema di selezione permette invece di selezionare un'azione da compiere in relazione al fatto che una specifica condizione si verifichi o no. Lo schema di selezione ha tre possibili varianti:

- nel caso (a) se la condizione C è verificata è eseguita un'azione A (non necessariamente elementare), altrimenti non viene svolta nessuna azione;
- nel caso (b) se la condizione C non è verificata è eseguita un'azione A (non necessariamente elementare), altrimenti non viene svolta nessuna azione;
- nell'ultimo caso (c) se la condizione C è verificata è eseguita l'azione A, altrimenti l'azione B.



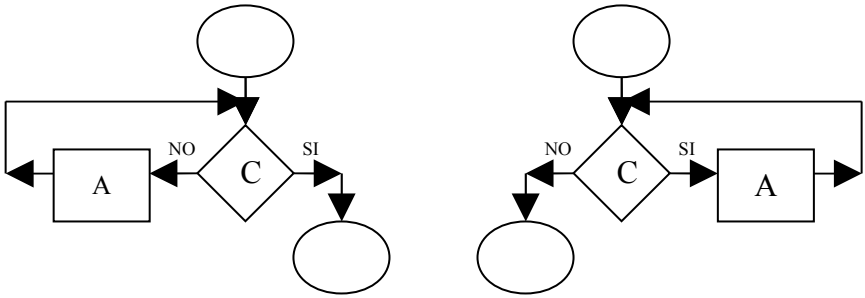
(a) (b) (c)

Fig. n. 5.5 – LA SELEZIONE

5.2.3 La Ripetizione

Nello schema di ripetizione, una certa azione A vene ripetuta fino a che non si verifica una certa condizione C. Lo schema di iterazione può presentarsi in quattro modi: nei primi due l'azione A può essere eseguita zero o più volte, mentre negli altri e due l'azione A è eseguita almeno una volta.

(a) (b)



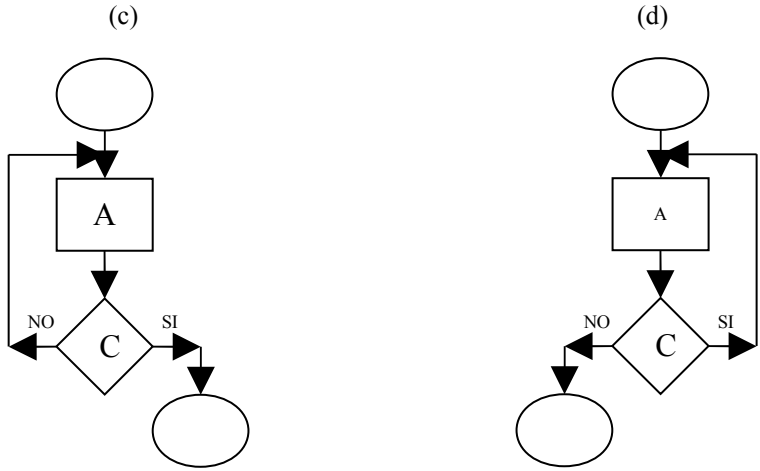
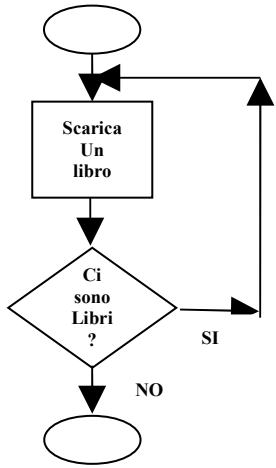


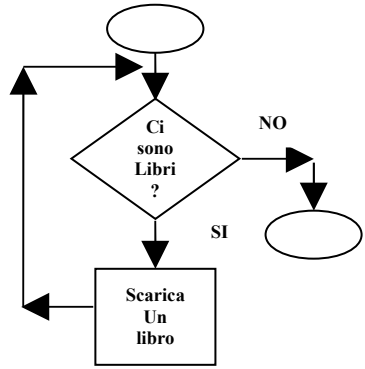
Fig. n. 5.6 – L'Iterazione o Ripetizione

5.2.4 Esempi

ESEMPIO - SCARICARE DEI LIBRI DA UN FURGONCINO.

Consideriamo i due possibili diagrammi di flusso:





(1) (2)

Nel caso (1) è stato utilizzato lo schema di ripetizione (d). È importante osservare che in questo caso l'azione è svolta almeno una volta; quindi, si dà per scontato che ci sia almeno un libro nel furgoncino e, successivamente, finché il controllo registrerà la presenza di libri da scaricare, si tornerà ad eseguire di nuovo l'azione "*scarica un libro*".

Nel caso (2) è stato utilizzato lo schema di ripetizione (b); prima di svolgere qualunque azione, si esegue il controllo, che potrebbe portare direttamente alla terminazione, senza che l'azione sia stata svolta. In altre parole si tiene presente l'eventualità che il furgoncino possa essere vuoto.

Esempio - Viaggio verso New York.

Abbiamo detto che con i tre schemi di base (sequenza, selezione, ripetizione) è possibile rappresentare qualunque azione complessa. Vediamo come ciò può essere fatto considerando il seguente caso per risolvere il problema "*Andare a New York*".

La tecnica è sempre la stessa: partiamo da uno schema generale composto da:

Fig. n. 5.7 – Primo passo.

Se l'azione "*Andare a New York*" fosse elementare non ci sarebbe altro da fare, ma poiché andare a New York comporta una moltitudine di altre azioni più semplici, cerchiamo di scindere le varie fasi e cerchiamo di capire quale schema di base applicare. Potremmo, ad esempio, individuare innanzitutto una serie di azioni da svolgere consecutivamente:

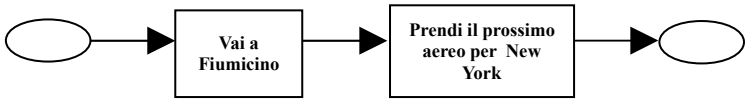


Fig. n. 5.8 – Secondo passo.

Siamo passati così da un unico blocco contenente tutto il metodo di risoluzione del problema, a due blocchi posti in sequenza contenenti due azioni che possono essere di tipo elementare oppure ulteriormente semplificabili. Notiamo che sostituendo al blocco "*Andare a New York*" i due nuovi blocchi posti in sequenza abbiamo ottenuto schemi tra loro equivalenti. Nella rappresentazione così ottenuta possiamo facilmente riconoscere uno schema di sequenza.

Vediamo ora se i due blocchi ottenuti possono essere semplificati. Il blocco "*Vai a Fiumicino*", per esempio, è costituito da diverse fasi, come scegliere un mezzo di trasporto ecc. Quindi, "*esplodiamo*" questo blocco facendo alcune considerazioni: come ci vado a Fiumicino? Con l'auto, ma possiedo un'auto? Se possiedo un'auto dovrò prima fare benzina, e se invece non ho l'auto dovrò andare in treno, ma prima devo fare il biglietto, ecc. Rappresentiamo tutte queste azioni e scelte attraverso lo schema mostrato di seguito.

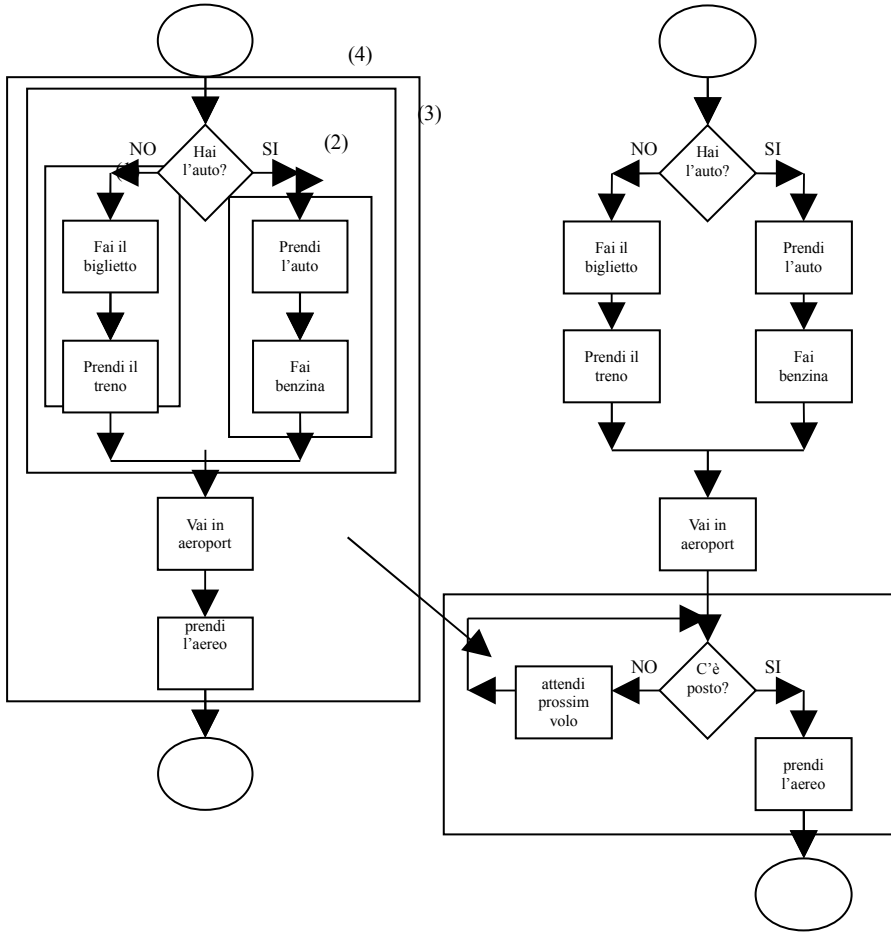


Fig. n. 5.9 – SCHEMA FINALE

Nello schema ottenuto riconosciamo subito una sequenza (indicata con 1), una sequenza (indicata con 2), quindi una selezione (indicata con 3), infine un'altra sequenza (indicata con 4). "Esplodendo" il blocco "*Prendi l'aereo*" otteniamo invece uno schema che riconosciamo essere una ripetizione.

Il procedimento ora mostrato con un esempio prende il nome di **programmazione strutturata**, che consiste in una tecnica di costruzione dei diagrammi di flusso per raffinamenti successivi: partendo da un unico blocco che rappresenta l'intero problema da risolvere, andiamo a sostituirlo con uno schema di composizione

appropriato alle esigenze (sequenza, selezione, iterazione), e così via fino ad ottenere esclusivamente azioni elementari.

Notiamo che dall' "*esplosione*" di un blocco rappresentante un'azione complessa, otteniamo uno schema che ha sempre almeno un altro blocco che rappresenta l'azione. Inoltre, ogni schema di composizione ha sempre e solo un punto di entrata ed un punto di uscita.

Con un diagramma strutturato deve essere anche possibile effettuare l'operazione inversa: partendo da uno schema esploso, si deve poter tornare ad ottenere un singolo blocco rappresentante l'intero problema. Si deve procedere semplicemente riconoscendo i vari schemi di composizione e comprimendoli in blocchi più complessi, fino a giungere al blocco iniziale.

Quella appena descritta costituisce l'**analisi** del diagramma di flusso ed è assimilabile all'analisi logica della lingua parlata. L'insieme di regole per la combinazione dei simboli (che siano parole o, come nel nostro caso, simboli grafici) di un linguaggio prende il nome di **grammatica**. La grammatica è **generativa,** nel senso che permette di generare tutte ed esclusivamente le frasi valide del linguaggio (nel nostro caso diagrammi validi, o strutturati).

Esempio. *Vediamo ora il Teorema di Böhm-Jacopini in azione: a sinistra abbiamo uno schema non strutturato ed a destra uno schema strutturato equivalente.*

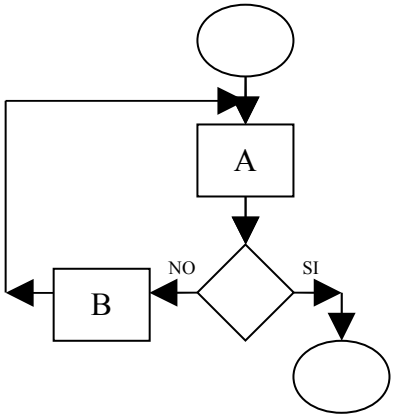


Fig. n. 5.10 – Uno schema non strutturato.

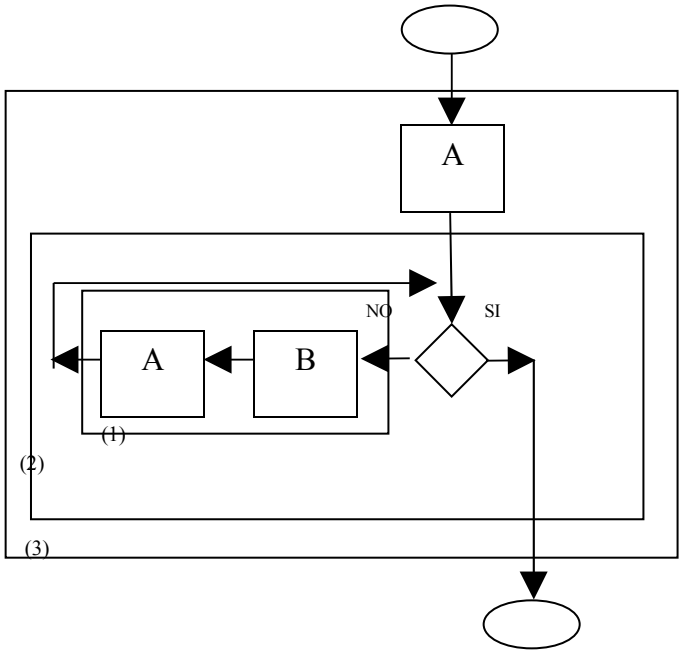


Fig. n. 5.11 – UNO SCHEMA STRUTTURATO EQUIVALENTE.

I due schemi si equivalgono sotto il profilo della funzionalità, ovvero rappresentano **graficamente** il **flusso di controllo** di algoritmi **equivalenti**.

Il primo non è però strutturato, perché non è riconoscibile nessuno degli schemi fondamentali di composizione.

Nel secondo, invece, possiamo riconoscere:

- uno schema di sequenza (1);
- uno di iterazione (2);
- ed infine uno di sequenza (3).

5.2.5 Convenzioni

Entriamo nel dettaglio della rappresentazione dei diagrammi di flusso, specificando alcune convenzioni. Se disponiamo, per la rappresentazione di un determinato diagramma di flusso, di un certo numero di azioni elementari diverse (es.: "*prendi un libro*", "*accendi la lampada*"), le indicheremo con A_1, A_2, A_3, mentre i diversi controlli li indicheremo con C_1, C_2. Quindi, nel seguente diagramma di flusso

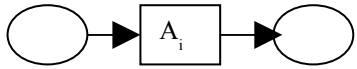


Fig. n. 5.12 – UN SEMPLICE ALGORITMO COMPOSTO DA UNA SOLA AZIONE ELEMENTARE.

A_i è una qualsiasi azione elementare di cui disponiamo. Tale diagramma è sicuramente strutturato perché abbiamo un inizio, un'azione elementare ed una fine.

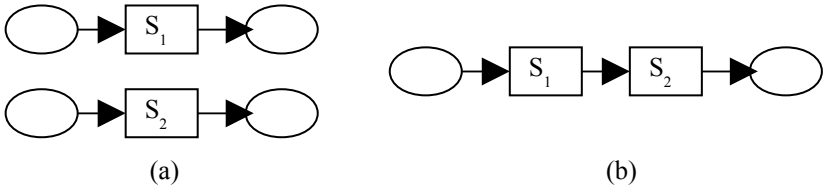


Fig. n. 5.13 – UN ALGORITMO COMPOSTO DA UNA SEQUENZA DI BLOCCHI STRUTTURATI.

Nella figura n. 13 (a) S_1 e S_2 sono blocchi strutturati, che possono contenere azioni anche molto complesse. Se combino i due blocchi strutturati S_1 e S_2, in sequenza, ottengo un diagramma anch'esso strutturato come nella figura n. 13 (b).

Se S_1 e S_2 sono blocchi strutturati, il diagramma di flusso della figura n. 14 è strutturato.

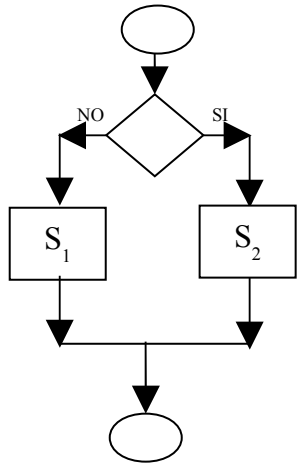


Fig. n. 5.14 – UN ALGORITMO CON UN CONTROLLO DI SELEZIONE.

Infine, se S_1 è strutturato, allora il seguente diagramma nel suo insieme rimane strutturato.

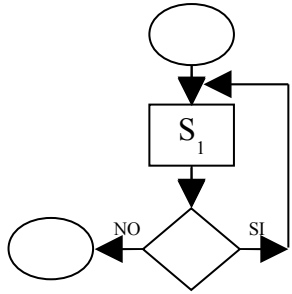
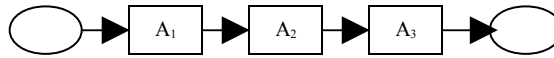
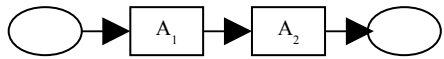


Fig. n. 5.15 – UN ALGORITMO CON UNA STRUTTURA CICLICA

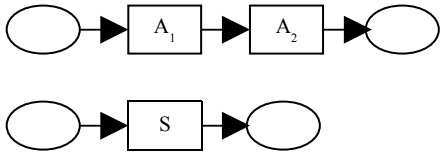
Quindi, ad esempio,



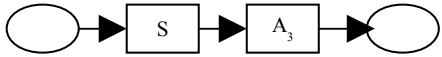
è strutturato, poiché



è strutturato:



ed infine



è strutturato.

5.3 La Pseudocodifica

Poniamo il caso di dover comunicare un algoritmo a qualcuno in contatto con noi telefonicamente. Come possiamo fare? Se abbiamo a disposizione il diagramma di flusso, che rappresenta il **flusso di controllo** dell'algoritmo, potremmo pensare ad esempio di descrivere tale diagramma verbalmente al nostro interlocutore. Ma com'è possibile comunicare qualcosa che ha una rappresentazione bidimensionale (il diagramma di flusso) utilizzando le parole, che rappresentano in modo lineare una sequenza di suoni che si susseguono nel tempo? Per risolvere tali problemi abbiamo bisogno di un linguaggio che ci permetta di descrivere in modo **lineare e strutturato** gli **algoritmi**: chiameremo tale linguaggio uno **pseudolinguaggio**.

Occorre notare che uno pseudolinguaggio non è che un misto di espressioni del linguaggio naturale (*l'italiano*) e costrutti fondamentali di un linguaggio di programmazione (*nel nostro caso il linguaggio **C***). La sua utilità risiede nella semplicità di scrittura, nella facile e rapida comprensione (poiché utilizziamo costrutti di un *linguaggio di programmazione* ben *formalizzato*, non vi può essere ambiguità nell'interpretazione delle frasi) e nella sinteticità (poiché utilizziamo espressioni in *linguaggio naturale*).

Si **noti bene** che uno pseudolinguaggio è utilizzato fondamentalmente per i seguenti scopi:

- comunicare ad un nostro simile (*umano*) un algoritmo al fine di:
    - farlo eseguire (l'umano è il nostro *automa*);
    - avere una sua opinione (se lavoro all'interno di un *team di sviluppo software,* ad esempio);
- chiarir**ci** le idee sull'algoritmo (ovvero *come devo agire* per risolvere il problema?).

L'attività di rappresentazione degli algoritmi utilizzando uno **pseudolinguaggio** prende il nome di **pseudocodifica**. Il risultato ottenuto si chiama **pseudocodice**.

Il linguaggio di programmazione che utilizzeremo come base per la definizione nel nostro pseudolinguaggio è il linguaggio **C**, sviluppato nel 1972 nei ***Laboratori Bell*** da **Dennis Ritchie** per il sistema operativo **UNIX**, e formalizzato nel **1978** da **Brian Kernigham** e **Dennis Ritchie** nel documento **The C Programming Language**, divenuto poi il manuale di riferimento del linguaggio.

5.3.1 Il nostro pseudolinguaggio

Di seguito è definito un meccanismo di pseudocodifica per rappresentare in modo lineare (*testuale*) e strutturato degli algoritmi**.**

Innanzitutto dobbiamo essere capaci di distinguere le parole che costituiscono il lessico della lingua italiana ("*Il", "cane*", "*mangia*", "*del*", "*formaggio*") dalle parole che stanno ad indicare qualcos'altro ("*azione*", "*condizione*", …). Ad esempio, la parola "*azione*" può indicare: "*esci a giocare*", "*vai a dormire*", "*mangia la pappa*", ecc. Per distinguere queste parole, che stanno per qualcos'altro, dalle omonime parole del lessico italiano, le racchiudiamo tra parentesi acute, come ad esempio:

<*azione*>

e le chiamiamo **simboli non terminali** oppure **categorie sintattiche**.

Passiamo ora ad illustrare il nostro pseudolinguaggio. Si noti innanzitutto che:

> *uno schema di flusso strutturato **S** rappresenta un'azione, e pertanto indicheremo lo pseudocodice che rappresenta **S** con la categoria sintattica* <***azione***>.

In particolare, l'algoritmo più semplice, quello raffigurato in fig. 5.12, composto da una sola azione elementare <azione> sarà codificato semplicemente come:

<azione>

ESEMPIO - AZIONE. *Se il nostro schema di flusso* **S** *è composto dalla sola azione* "*Prendi l'ombrello*"*, la sua pseudocodifica sarà semplicemente:*

"*Prendi l'ombrello*" ■

Sequenza. Supponiamo di avere due diagrammi di flusso strutturati $\mathbf{S_1}$ e $\mathbf{S_2}$, che rappresentano graficamente due azioni <**azione**$_1$> e <**azione**$_2$>. Allora la sequenza illustrata in fig. 11 (b) può essere rappresentata come segue:

{<azione$_1$> <azione$_2$>}

ovvero, con una parentesi graffa aperta, seguita dalla pseudocodifica del diagramma di flusso strutturato S_1, da uno spazio, dalla pseudocodifica del diagramma di flusso strutturato S_2, ed infine da una parentesi graffa chiusa.

Esempio - Sequenza. *Se* <azione$_1$> *sta per* "*Prendi l'ombrello*"*, e* <azione$_2$> *sta per "Esci", allora scriveremo:*

{"*Prendi l'ombrello*" "*Esci*"} ■

Per rendere lo pseudocodice più leggibile, scriveremo:

{"*Prendi l'ombrello*"
"*Esci*" }

ovvero trascriveremo le azioni da compiere una di seguito all'altra, su più righe.

Selezione. Consideriamo adesso lo schema fondamentale selezione. Il diagramma di flusso della fig. 5.5 (a) è rappresentato come segue:

if (<condizione>)
<azione>

dove <condizione> indica ora lo pseudocodice associato alla condizione C della fig. 5.5 (a).

Esempio – Selezione. A.

if ("*Piove*")
"*Apri l'ombrello*" ■

In tal caso l'azione *"Apri l'ombrello*" è eseguita se la condizione "*Piove*" è verificata.

Analogo è il caso della fig. 5.5 (b): in tal caso faremo precedere alla categoria sintattica <condizione> il simbolo **!** che indica la **negazione** di ciò che segue.

Esempio – Selezione B.

if (! "*Piove*")
"*Esci a giocare*" ■

In tal caso l'azione *"Esci a giocare*" è eseguita se la condizione "*Piove*" non è verificata.

Consideriamo adesso il caso della fig. 5.5 (c) che viene rappresentato come segue:

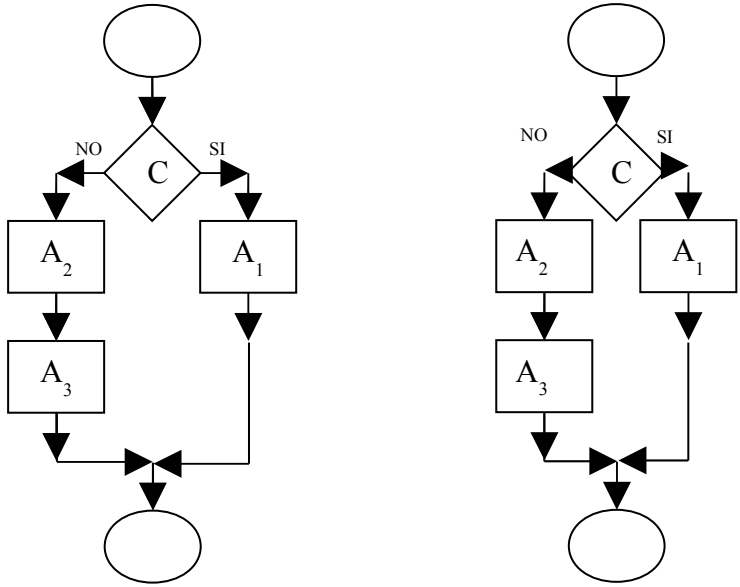
if (<condizione>)
    <azione$_1$>
else
    <azione$_2$>

ESEMPIO – SELEZIONE C.

if ("*Piove*")
    "Apri l'ombrello"
else
    "Vai al mare"

In tal caso, se la condizione "*Piove*" è verificata viene eseguita l'azione "*Apri l'ombrello*", altrimenti viene eseguita l'azione"*Vai al mare*".

Esempi più complessi. Possiamo così iniziare a comporre i vari schemi per rappresentare algoritmi più complessi. Gli algoritmi rappresentati nella seguente figura in forma bidimensionale:



possono essere rappresentati in forma lineare rispettivamente come:

```
if ( <condizione> )
  <azione1>
else
  { <azione2>  <azione3> }
```

```
if ( ! <condizione> )
  { <azione2>  <azione3> }
else
  <azione1>
```

Potremmo allora ottenere pseudocodice del tipo:

```
if ( "Piove" )
       "Apri l'ombrello"
else
  {    "Vai al mare"
       "Fai il bagno" }
```

```
if ( ! "Hai sonno" )
            "Esci"
else
     {      "Spegni la tv "
            "Vai a letto" }
```

Iterazione. Consideriamo l'ultimo schema fondamentale di composizione, l'iterazione (o ripetizione). Il diagramma di flusso della fig. 5.6 (b) è rappresentato con:

```
while ( <condizione> )
        <azione>
```

Esempio – Iterazione B.

```
while ( "C'è almeno un libro" )
   "Scarica un libro"
```
■

Invece il diagramma di flusso della fig. 5.6 (d) è rappresentato con:

```
do <azione>
  while ( condizione )
```

Esempio – Iterazione C.

```
do "Scarica un libro"
   while ( "C'è almeno un libro" );
```
■

6

I Linguaggi e le Grammatiche

In questo capitolo sono introdotti i concetti fondamentali di grammatica e di linguaggio generato da una grammatica.
Mostreremo poi che è possibile descrivere i linguaggi di programmazione attraverso particolari grammatiche, dette libere dal contesto (context-free), utilizzando una notazione molto compatta: la Backus Naur Form.

6.1 I Linguaggi

Nei capitoli precedenti abbiamo già definito il concetto di simbolo (*un simbolo è un segno a cui si attribuisce un valore convenzionale*) e di alfabeto (*un alfabeto è un insieme finito e non vuoto di simboli*).

L'alfabeto che è utilizzato per costruire le frasi di un linguaggio è detto **alfabeto dei simboli terminali**. Indichiamo tale insieme con la lettera T.

Esempio.
L'insieme $T_1 = \{a, b, c\}$ *è un alfabeto composto dai tre simboli a, b e c.* ■

Esempio. *L'insieme* $T_2 = \{il, gatto, topo, mangia\}$ *è un alfabeto composto da quattro vocaboli della lingua italiana. In questo caso*

gatto

non è una sequenza di cinque simboli, ma è un unico simbolo terminale. ■

Esempio – Il Lessico della Lingua Italiana. Se d*efiniamo* T_3 *come il* ***lessico*** *della lingua italiana, allora* T_3 *è costituito da tutti i vocaboli che sono utilizzati per costruire le frasi della lingua italiana.* ■

Esempio – Il Sistema Numerico Decimale. *Un altro alfabeto terminale è l'insieme delle cifre del sistema numerico decimale* $T_4 = \{ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$*, che è composto da dieci simboli* ■.

È possibile prendere i simboli di un alfabeto terminale e concatenarli (ovvero, *metterli in sequenza*).

Considerando l'ultimo esempio, otteniamo sequenze come 00120 o 125.

Considerando, invece, l'esempio che lo precede, otteniamo sequenze come

il topo mangia

oppure semplicemente

il

ma anche

gatto topo il

Definizione. Una successione finita di simboli di un alfabeto prende il nome di **stringa**. La **lunghezza** di una stringa è data dal numero di simboli di cui è composta. ■

Esempio. *La stringa "gatto topo il" dell'alfabeto* T_2 *ha lunghezza 3.* ■

Partendo da un alfabeto H, ovvero da un insieme di simboli, posso definire un nuovo insieme che contiene tutte le possibili stringhe di simboli di questo alfabeto. Questo nuovo insieme si chiama **chiusura transitiva** dell'alfabeto H, e si indica con H^+**.**

Un esempio di notevole importanza si ha quando l'alfabeto in questione è il nostro alfabeto T dei simboli terminali. In tal caso, ricordiamo, la **chiusura transitiva** di T, che indico con T^+, è l'insieme costituito da tutte le stringhe di lunghezza finita e non nulla di elementi di T.

Ad esempio, la chiusura transitiva dell'alfabeto T del primo esempio è composta da tutte:

- le stringhe di lunghezza 1: *a*, *b*, *c*;
- le stringhe di lunghezza 2: *aa*, *ab*, *ac*, *ba*, *bb*, *bc*, *ca*, *cb*, *cc*;
- le stringhe di lunghezza 3: *aaa*, *aab*, *aac*, *aba*, *aca*, …;
- le stringhe di lunghezza 4; *aaaa*, *aaab*, *aaac*, …;
- ecc.

Quindi, T^+ = { *a*, *b*, *c*, *aa*, *ab*, *ac*, *ba*, …, *aaa*, *aab*, *aac*, …, *aaaa*, *aaab*, … }

La chiusura transitiva è sempre un insieme infinito, anche se l'alfabeto terminale contiene un solo simbolo.

ESEMPIO. *La chiusura transitiva dell'alfabeto terminale* $\boldsymbol{T} = \{ 0 \}$ *è data da*

$$\boldsymbol{T}^{+} = \{ 0,\ \ 00,\ \ 000,\ \ 0000, \ldots.\} \qquad \blacksquare$$

Definizione. Un **linguaggio** $\boldsymbol{L}$ costruito a partire dall'alfabeto $\boldsymbol{T}$ è un qualunque sottoinsieme di $\boldsymbol{T}^{+}$. ■

Le stringhe di simboli terminali che appartengono a $\boldsymbol{L}$ si chiamano **frasi** del linguaggio $\boldsymbol{L}$.

Un linguaggio definito sull'alfabeto del primo esempio potrebbe essere $\boldsymbol{L} = \{ a,\ aa,\ \ ca,\ \ bbaca \}$. In tal caso, *ca* è una frase di questo linguaggio.

Mentre un alfabeto $\boldsymbol{T}$ è un insieme finito e la sua chiusura transitiva $\boldsymbol{T}^{+}$ è sempre un insieme infinito, un linguaggio $\boldsymbol{L}$ definito su $\boldsymbol{T}$ invece può essere composto da un unico elemento, ma potrebbe anche essere un insieme infinito. Inoltre, dall'insieme $\boldsymbol{T}^{+}$ posso definire infiniti sottoinsiemi di simboli terminali, che costituiranno un numero infinito di linguaggi.

Per l'alfabeto del primo esempio posso definire il linguaggio

$$\boldsymbol{L_1} = \{ a,\ \ aa,\ \ ca,\ \ bbaca \}$$

ma anche

$$\boldsymbol{L_2} = \{ ab,\ \ ba \}.$$

ESEMPIO. *Consideriamo l'alfabeto* $\boldsymbol{T} = \{ +,\ \ -,\ \ 0,\ \ 1,\ \ 2,\ \ 3,\ \ 4,\ \ 5,\ \ 6,\ \ 7,\ \ 8,\ \ 9 \}$*, che può essere definito come l'insieme dei simboli utilizzati per rappresentare i numeri interi. In tal caso un numero intero non è altro che una frase del linguaggio dei numeri interi. La chiusura transitiva* $\boldsymbol{T}^{+}$ *sarà costituita da infinite sequenze di questi simboli, come ad esempio*

$$254,\ \ +123,\ \ 0,\ \ \ 512,\ +11+-0---$$

ma, attenzione, la stringa +11+-0--- *non appartiene al linguaggio dei numeri interi.* ■

Per evitare di generare sequenze che non appartengono al linguaggio occorre stabilire delle regole di produzione, che permettano di formare (ovvero, produrre) le sole sequenze valide.

Nel caso precedente occorre stabilire che le stringhe del **linguaggio dei numeri interi** devono essere:

- sequenze di cifre numeriche, che possono essere precedute dal segno + o dal segno – ;
- fa eccezione il numero zero, che non deve essere preceduto da alcun segno.

Esistono due metodi per descrivere un linguaggio:

- Il primo è quello **estensionale**, che prevede di specificare direttamente tutte le sequenze valide del linguaggio elencandole. Ad esempio, il linguaggio composto dai numeri pari è {0, 2, 4, 6, …}. Questo è l'approccio che abbiamo utilizzato negli esempi precedentemente esposti. È evidente che se le sequenze sono numerose o infinite, come nel caso dei numeri interi, questa modalità si rivela impraticabile;
- Il secondo è quello **intensionale**, che consiste nel definire le caratteristiche che accomunano gli elementi del linguaggio. Il linguaggio dei numeri pari può essere definito intensionalmente come l'insieme dei numeri che dividendoli per due danno come resto zero. Tipi di definizioni intensionali sono:
    - *generativo*: consiste in un sistema compatto che permette di rappresentare tutte e sole le stringhe valide di un linguaggio mediante la definizione di una **grammatica;**
    - *algebrico*: il linguaggio è rappresentato da un'**espressione algebrica** o dalla soluzione di un sistema di relazioni algebriche;
    - *riconoscitivo*: si costruisce un **automa** per riconoscere o accettare le stringhe del linguaggio. Vedremo questo approccio nelle seguenti lezioni.

In questa lezione descriveremo il primo approccio *intensionale*, ovvero quello *generativo*.

Nota bene. In questa sezione abbiamo definito la chiusura transitiva di un insieme **H** di simboli, e la abbiamo indicata con $\mathbf{H}^{+}$. In molti libri di testo si incontra la **chiusura riflessiva e transitiva** di un insieme **H**, indicata con il simbolo $\mathbf{H}^{*}$. Questa si ottiene dalla chiusura transitiva aggiungendo una stringa di lunghezza zero, ovvero la stringa nulla, composta da zero caratteri. La stringa nulla si indica talvolta con il simbolo **ε**. La chiusura riflessiva transitiva di un insieme **H** si chiama anche **chiusura di Kleene** di **H**, dal nome di un famoso logico. In questa lezione abbiamo utilizzato la chiusura transitiva e non quella riflessiva transitiva, per non appesantire inutilmente la trattazione.

L'attività di ricerca di Stephen Kleene (1909 - 1994), come professore presso la University of Wisconsin a Madison dal 1948 al 1979, era rivolta essenzialmente allo studio della teoria degli algoritmi ed alle funzioni ricorsive. Ha collaborato infatti con ***Church***, ***Gödel*** ***e*** ***Turing*** allo sviluppo della teoria della ricorsione.

6.2 Le Grammatiche

Prima di definire le regole della grammatica occorre introdurre un altro tipo di simboli, i **simboli non terminali** o **categorie sintattiche.** Essi sono simboli che possono essere sostituiti da altri simboli, sia terminali che non terminali.

Ad esempio, nell'analisi logica, il simbolo non terminale <*verbo*> può essere sostituito dai verbi *avere*, *cantare* *andare*, ecc. Allo stesso modo il simbolo non terminale <*soggetto*> può essere sostituito da *lo studente*, *il cane*, ecc.

Per convenzione i simboli non terminali sono indicati con le lettere maiuscole: A, B, C, ecc.

Consideriamo l'alfabeto terminale $\boldsymbol{T_2}$ dell'esempio precedente e indichiamo con $\boldsymbol{N}$ l'insieme dei simboli non terminali $\boldsymbol{N}$ = {A, B, C}. L'insieme $(\boldsymbol{N} \cup \boldsymbol{T})^+$ rappresenta la chiusura transitiva di $(\boldsymbol{N} \cup \boldsymbol{T})$, ovvero l'insieme di tutte le stringhe di simboli terminali e non terminali. Stringhe appartenenti a $(\boldsymbol{N} \cup \boldsymbol{T})^+$ saranno, ad esempio: A, AA, *il*, *gatto il* B, *il il*, C *mangia* .

Definizione. Dato un simbolo non terminale W ed una stringa $\alpha \in (N \cup T)^*$, ovvero composta da simboli terminali e non, diciamo che una **regola grammaticale** (o **regola di produzione**), avente parte sinistra W e parte destra α, è una regola che permette di sostituire il simbolo W con la stringa α. Indichiamo tale regola con la notazione:

$$W \rightarrow \alpha$$ ■

ESEMPIO. *Le seguenti sono tre regole grammaticali, costruite utilizzando l'alfabeto dei simboli terminali* $\boldsymbol{T}$ = {*il, gatto, topo*}*, e l'alfabeto dei simboli non terminali* $\boldsymbol{N}$ = *{A, B}:*

$A \rightarrow$ *il gatto* A A B *il*
$A \rightarrow B$
$C \rightarrow A\ B$ *topo* ■

Una regola di produzione indica che, quando incontro un certo simbolo non terminale posto a sinistra della freccia, posso (ma attenzione, non devo) sostituire tale simbolo con ciò che compare a destra.

Ad esempio, la regola di produzione

$$\langle soggetto \rangle \rightarrow Giovanni$$

indica che posso sostituire il simbolo non terminale <soggetto> con il simbolo *Giovanni*, mentre la regola

$$\langle soggetto \rangle \rightarrow lo \quad studente$$

indica che posso sostituire il simbolo non terminale <*soggetto*> con la stringa *lo studente* composta da due simboli terminali.

Definizione. Una **grammatica G** è specificata attraverso **quattro** elementi:

1. un alfabeto di simboli terminali ***T***;
2. un alfabeto di simboli non terminali ***N***;
3. un simbolo non terminale **S** detto **seme** o **assioma**;
4. un insieme finito e non vuoto di regole di produzione **P**, di cui almeno una deve avere l'assioma S nella parte sinistra. ■

Le grammatiche così definite sono chiamate **non contestuali** o **libere da contesto** o **di tipo 2**, perché ad un simbolo non terminale posso sostituire una qualsiasi stringa di terminali e non terminali, indipendentemente dal contesto, ovvero da ciò che circonda il simbolo da sostituire.

Vediamo come si formano le frasi di un linguaggio a partire dai concetti appena esposti.

Esempio. *Sia* $\boldsymbol{T} = \{ 0, \ 1 \}$ *l'alfabeto dei simboli terminali e* $\boldsymbol{N} = \{S, A\}$ *l'alfabeto dei simboli non terminali. Sono definite le seguenti regole grammaticali:*

$S \rightarrow A0$	*(regola 1)*
$A \rightarrow 0$	*(regola 2)*
$A \rightarrow 1$	*(regola 3)*
$A \rightarrow AA$	*(regola 4)*

Per dimostrare che la stringa 010 *appartiene al linguaggio generato dalla grammatica appena definita, occorre partire dall'assioma S e applicare le regole grammaticali fino ad ottenere la stringa voluta. Quindi, compiremo i seguenti passi:*

$S \Rightarrow A0$	*in base alla regola 1*
$A0 \Rightarrow AA0$	*in base alla regola 4*
$AA0 \Rightarrow A10$	*in base alla regola 3*

A10 ⇒ 010 *in base alla regola 2* ■

Definizione. Data una grammatica G e due stringhe β e γ composte da simboli terminali e non, si dice che **γ deriva direttamente da β** in G e si scrive $\beta \Rightarrow \gamma$ se

- esiste una regola grammaticale $A \rightarrow \alpha$ in G;
- la stringa β contiene almeno un simbolo A;
- la stringa γ è ottenuta dalla stringa β sostituendo **un solo** simbolo A che essa contiene con α ■

Quindi, la stringa A10 dell'esempio precedente deriva direttamente dalla stringa AA0, sostituendo alla seconda A il simbolo 1.

Definizione. Data una grammatica G e due stringhe β e $\gamma \in (N \cup T)^*$, si dice che **γ deriva da β** in G e si scrive $\beta \Rightarrow^* \gamma$ se esiste un numero finito *n* di derivazioni dirette, tali che:

$$\beta_0 \Rightarrow \beta_1 \Rightarrow \ldots \Rightarrow \beta_n \text{ dove } \beta_0 = \beta \text{ e } \beta_n = \gamma. \quad \blacksquare$$

Quindi, la successione di derivazioni

$$S \Rightarrow A0 \Rightarrow AA0 \Rightarrow A10 \Rightarrow 010$$

dell'esempio precedente può essere scritta in maniera compatta con l'espressione

$$S \Rightarrow^* 010$$

che leggiamo

da S derivo 010.

Definizione. Il **linguaggio** L_G generato da una grammatica **G** è dato dall'insieme delle stringhe α di simboli terminali che si possono derivare a partire dall'assioma S, in base alle regole grammaticali di G:

$$L_G = \{ \alpha \in T^* \mid S \Rightarrow^* \alpha \} \quad \blacksquare$$

Esempio. *Sia data una grammatica* **G** *con alfabeto dei simboli terminali* $T = \{ 0, 1 \}$*, con alfabeto dei simboli non terminali* $N = \{S\}$*, con assioma S e con le seguenti regole grammaticali:*

S→01	*(Regola 1)*
S→0S1	*(Regola 2)*

Il linguaggio generato da **G** *è dato da* $L_G = \{01, 0011, 000111, 00001111, 0000011111, \ldots\ldots\}$. ■

Ad esempio, dimostriamo che la stringa 00001111 appartiene a tale linguaggio, indicando in corrispondenza di ciascuna freccia la regola di produzione utilizzata:

$$S \Rightarrow^{2} 0S1 \Rightarrow^{2} 00S11 \Rightarrow^{2} 000S111 \Rightarrow^{1} 00001111$$

e, quindi,

$$S \Rightarrow^{*} 00001111$$

6.3 Equivalenza di grammatiche

Si noti che uno stesso linguaggio può essere generato da più grammatiche.

ESEMPIO. *Consideriamo il linguaggio* $\boldsymbol{L}$ *composto dalle stringhe del tipo* $a^n b^m$ *con n e m maggiori di zero.*

Con la notazione $a^n b^m$ *intendiamo rappresentare qualunque stringa costituita da un certo numero di a seguito da un certo numero di b, come ad esempio la stringa aaabbbbbb. Abbiamo adottato una* ***espressione algebrica*** *per descrivere il linguaggio e, quindi, un metodo* ***intensionale****.*

Definiamo la grammatica $\mathbf{G_1}$ *che ha* $\boldsymbol{T}$ = { a, b } *come alfabeto dei simboli terminali,* $\boldsymbol{N_1}$ = { S } *come alfabeto dei simboli non terminali,* **S** *come assioma e le seguenti regole grammaticali:*

$$S \rightarrow ab$$
$$S \rightarrow aS$$
$$S \rightarrow Sb$$

È facile verificare che $\boldsymbol{L_{G1}} = \boldsymbol{L}$.

Consideriamo adesso la grammatica $\mathbf{G_2}$ *che ha sempre* $\boldsymbol{T}$ = { a, b } *come alfabeto dei simboli terminali,* $\boldsymbol{N_2}$ = { A, B } *come alfabeto dei simboli non terminali,* **A** *come assioma e le seguenti regole grammaticali:*

$$A \rightarrow a$$
$$A \rightarrow aA$$
$$A \rightarrow aB$$
$$B \rightarrow b$$
$$B \rightarrow bB$$

È facile verificare che anche $\boldsymbol{L_{G2}} = \boldsymbol{L}$.

Quindi, le due grammatiche $\mathbf{G_1}$ *e* $\mathbf{G_2}$ *generano lo stesso linguaggio* $\boldsymbol{L}$. ■

Definizione. Due grammatiche si dicono **<u>equivalenti</u>** se generano lo stesso linguaggio. ■

6.4 La Backus Naur Form

Il simbolismo utilizzato nella sezione precedente, se pur corretto formalmente, può generare talvolta confusione. Infatti, le lettere maiuscole dell'alfabeto italiano possono far parte di vocaboli del linguaggio descritto, e quindi generare confusione con le corrispondenti categorie sintattiche. In altre parole, se l'alfabeto terminale utilizzato è composto dalle lettere maiuscole e minuscole dell'alfabeto inglese, e l'alfabeto non terminale dai simboli **A** e **B**, come distinguere il non terminale **A** dal terminale **A**?

Per risolvere questa ambiguità negli anni '50 John Backus (1924 - vivente) e Peter Naur (1928 - vivente) introdussero un **metalinguaggio** (ovvero un linguaggio per descrivere un altro linguaggio) chiamato in loro onore **BNF** (*Backus Naur Form*). Tale metalinguaggio oggi trova largo uso come strumento di specifica per le grammatiche dei linguaggi di programmazione.

Riassumiamo qui di seguito le convenzioni adottate in questo **metalinguaggio**:

- la freccia utilizzata nelle regole di produzione è sostituita da ::=
- ogni simbolo non terminale é racchiuso fra parentesi acute, ad esempio <soggetto>, superando così i limiti della rappresentazione dovuti all'uso delle lettere maiuscole (che sono solo 26 nell'alfabeto inglese!);
- le regole grammaticali che hanno lo stesso simbolo non terminale a sinistra sono raggruppate in un'unica regola, che ha nella parte sinistra il simbolo non terminale e nella parte destra le parti destre delle varie regole separate da una barra verticale "|", che si legge *oppure*.

6.4.1 Esempi di BNF

Esempio. *Sia* ***T*** = { 0, 1 } *l'alfabeto dei simboli terminali,* ***N*** = {<Seme>} *l'alfabeto dei simboli non terminali, <Seme> l'assioma e come unica regola*

<Seme> ::= 01 | 0<Seme>1

Si noti che tale regola corrisponde alle due regole

<Seme> ::= 01
<Seme> ::= 0<Seme>1 ■

ESEMPIO. *Sia dato l'alfabeto dei simboli terminali*

$$\boldsymbol{T} = \{ +,\ -,\ 0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9 \}$$

l'alfabeto dei simboli non terminali

$\boldsymbol{N}$ = {<numero intero>, <numero positivo>, <numero negativo>,
<sequenza di cifre>, <cifra>, <cifra≠da zero>}

e l'assioma <numero intero>. *Le regole grammaticali sono:*

<cifra≠da zero> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.
<cifra> ::= 0 | <cifra≠da zero>
<sequenza di cifre> ::= <cifra> | <cifra> <sequenza di cifre>
<numero positivo> ::= 0 | + <cifra≠da zero> |
+ <cifra≠da zero> <sequenza di cifre>
<numero negativo> ::= - <cifra≠da zero> | - <cifra≠da zero><sequenza di cifre>
<numero intero> ::= <numero positivo> | <numero negativo>

Ad esempio, verifichiamo che +385 è una stringa del linguaggio ***L****, illustrando la sequenza di derivazioni a partire dall'assioma:*

<numero intero> ⇒ <numero positivo>
⇒ + <cifra≠da zero> <sequenza di cifre>
⇒ + 3 <sequenza di cifre>
⇒ + 3 <cifra> <sequenza di cifre>
⇒ + 3 <cifra≠da zero> <sequenza di cifre>
⇒ + 3 8 <sequenza di cifre>
⇒ + 3 8 <cifra>
⇒ + 3 8 <cifra≠da zero>
⇒ + 3 8 5 ■

Il procedimento con il quale determiniamo se una stringa deriva dall'assioma si chiama **analisi grammaticale** o **analisi sintattica**.

6.5 Extended BNF

La **Extended BNF** (EBNF) migliora la leggibilità e la concisione della BNF introducendo dei nuovi simboli.
La *croce di Kleene* + rappresenta una sequenza di **uno o più** elementi del simbolo non terminale o terminale.

Esempio. *Nell'esempio precedente la regola grammaticale che definisce una* <sequenza di cifre> *può essere sostituita dalla seguente:*

$$\text{<sequenza di cifre>} ::= \text{<cifra>}^{+}$$

■

La *Stella di Kleene* * rappresenta una sequenza di **zero o più** elementi del simbolo non terminale o terminale.

Esempio. *Per il linguaggio L delle stringhe costituite da uno o più simboli* ***a*** *possiamo definire la grammatica che lo genera con la seguente BNF:*

$$\text{<A>} ::= a^{+}$$

■

Le parentesi graffe possono essere usate per **raggruppare** elementi, che ci permettono di evitare di definire simboli non terminali intermedi.

Esempio. *Ritornando all'esempio dei numeri interi, la regola grammaticale che definisce* <numero intero> *può essere riscritta nel seguente modo:*

<numero intero> ::= 0 | { + | - } <cifra≠da zero> |
{ + | - } <cifra≠da zero> <sequenza di cifre>

■

Nota: Molti autori usano le parentesi graffe con il significato di zero o più ripetizioni degli elementi racchiusi; in altre parole si assume lo stesso significato della stella di Kleene.

Le parentesi quadre possono essere usate per indicare elementi **opzionali** e, quindi, zero o una occorrenza degli elementi racchiusi.

Esempio. *Per il linguaggio L costituito da tutte le stringhe di uno o più simboli* ***b*** *precedute eventualmente dal simbolo* ***a*** *possiamo definire la seguente grammatica che lo genera in EBNF:*

$$\langle S \rangle ::= [a]\ b^{+}$$

■

La croce e la stella di Kleene non si usano con le parentesi quadre.

Le alternative possono essere **impilate**, ovvero messe una sull'altra, anziché essere separate utilizzando la barretta verticale, per aumentare la leggibilità della sintassi.

Esempio. *Di seguito ridefiniamo la grammatica dei numeri interi in EBNF:*

<cifra≠da zero> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9.
<cifra> ::= 0 | <cifra≠da zero>
<numero intero> ::= 0 | $\left\{\begin{matrix}+\\ \\ -\end{matrix}\right.$ <cifra≠da zero> | $\left\{\begin{matrix}+\\ \\ -\end{matrix}\right.$ <cifra≠da zero><cifra>$^{+}$

■

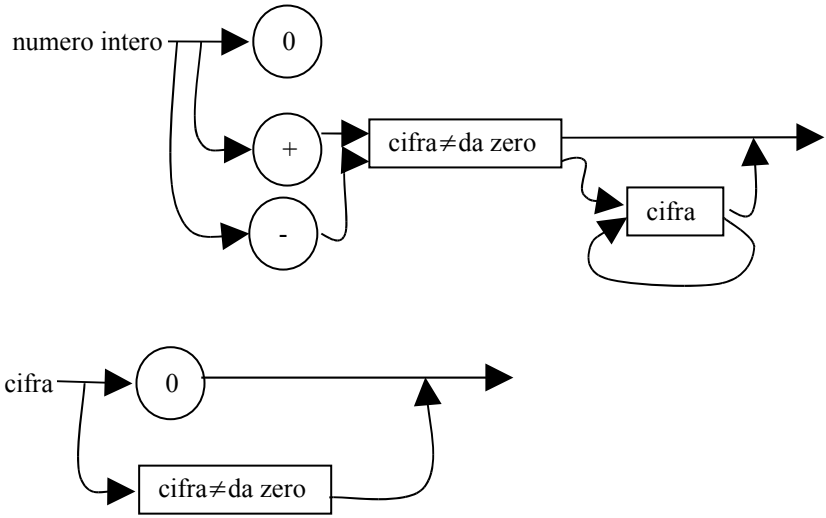
6.6 I Diagrammi Sintattici

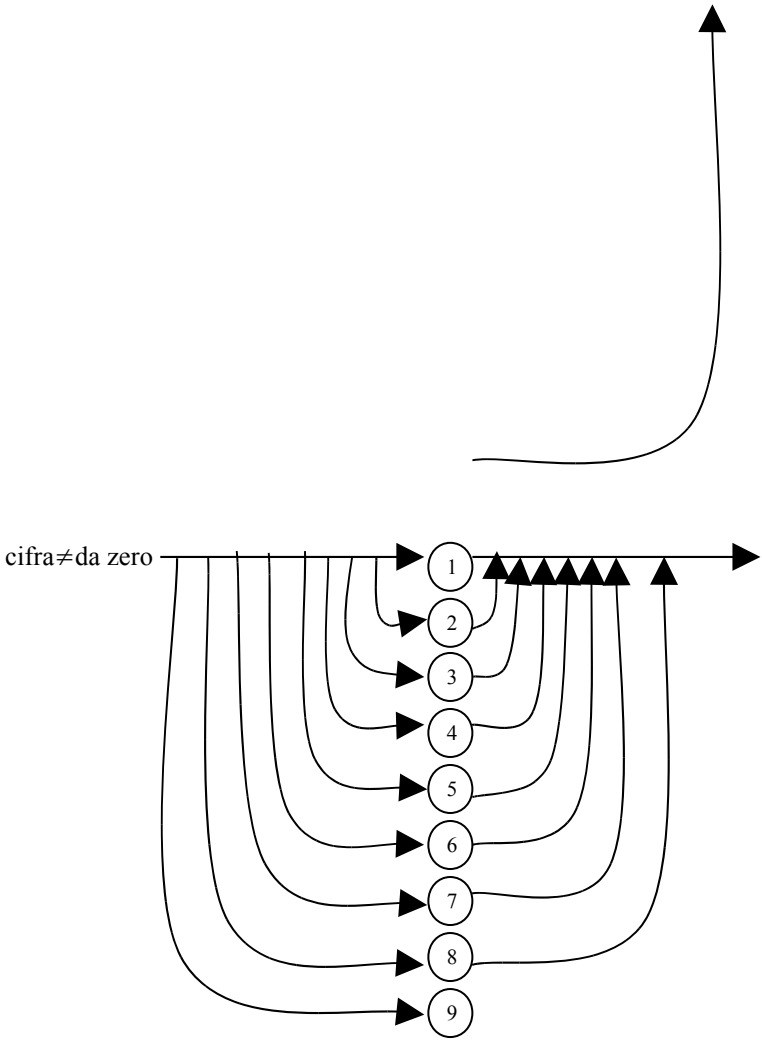
I diagrammi sintattici (o anche *carte sintattiche*) sono stati usati per la prima volta nel "*Pascal User Manual and Report*" di Jensen e Wirth. Essi costituiscono un efficace strumento grafico per rappresentare le regole di produzione di una grammatica, dove

- i *rettangoli* stanno ad indicare i simboli non terminali;
- gli *ovali* stanno ad indicare i simboli terminali.

Queste figurine sono collegate tra loro utilizzando delle frecce che indicano l'ordine con cui compaiono questi simboli nella regola grammaticale. Inoltre, il simbolo non terminale che compare nella parte sinistra di una regola di produzione é scritto così com'é, e da esso parte la prima freccia.

Per chiarire il concetto, mostriamo qui di seguito come rappresentare l'esempio precedente costituito dalla grammatica dei numeri interi:




cifra≠da zero
1
2
3
4
5
6
7
8
9

6.7 Linguaggi di Programmazione

La seguente definizione formalizza il concetto di linguaggio di programmazione, caratterizzandone solo l'aspetto sintattico.

Per definire formalmente un **linguaggio di programmazione *L*** occorre fornire una grammatica ***G*** tale che $L = L_G$, ovvero una grammatica che generi ***L***.

Una frase di questo linguaggio è chiamata **programma**.

ESEMPIO. *Data una grammatica per il linguaggio di programmazione **C**, posso costruire **tutti i programmi sintatticamente corretti** attraverso una serie di derivazioni ottenute, a partire dall'assioma, applicando le regole di produzione della grammatica.* ■

6.8 Alberi sintattici

In precedenza abbiamo visto come le derivazioni possono essere rappresentate in forma lineare, ovvero attraverso una sequenza di derivazioni dirette. Per rendere tale rappresentazione più chiara, conviene indicare su ogni freccia che rappresenta una derivazione:

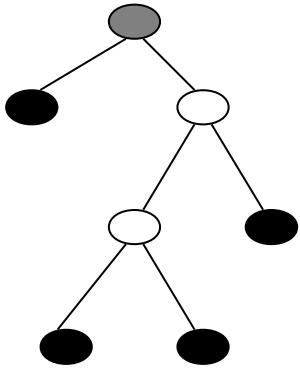
$$\Rightarrow$$

la regola di produzione che è stata applicata, ad esempio:

$$\overset{3}{\Rightarrow}$$

Esiste, però, un modo più efficace per rappresentare la catena di derivazioni, chiamato *albero sintattico*.

In informatica, gli alberi sono tradizionalmente rappresentati in modo invertito rispetto a come accade in natura: la radice è posta in alto, mentre le foglie sono poste in basso.



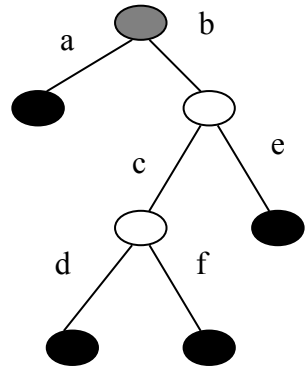
Nella figura a lato è rappresentato un albero: la **radice** è evidenziata con il colore grigio, le **foglie** con il colore nero, mentre gli elementi intermedi, detti **nodi intermedi**, sono in bianco.

Le linee che collegano i vari nodi intermedi, la radice e le foglie si chiamano **rami (**o **archi)**.

La radice, i nodi intermedi e le foglie costituiscono i **nodi** (o **vertici**) dell'albero. Quindi, l'albero raffigurato ha 7 nodi.

Quando assegniamo un nome ad un componente dell'albero (ovvero ad un nodo, o ad un ramo), diciamo che lo **etichettiamo** (è come se ponessimo al componente un'etichetta con il nome).

Nell'esempio a lato, gli archi sono etichettati con le lettere minuscole **a**, **b**, **c**, **d**, **e**, **f**.



Nella figura seguente utilizziamo un albero per rappresentare graficamente una classificazione di tipo botanico. Si noti che, in questo esempio, tutti i nodi sono etichettati, ma i rami non lo sono.

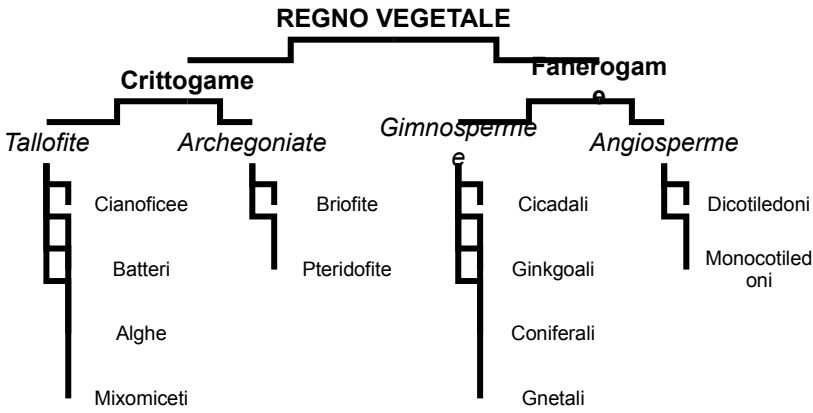


Fig. n. 6.1 – Classificazione del Regno Vegetale

Nella figura seguente utilizziamo un albero per rappresentare graficamente la dinastia dei re Merovingi. In tal caso parliamo di **albero genealogico**. Si noti che, in questo esempio, tutti i nodi sono etichettati, ma i rami non lo sono.

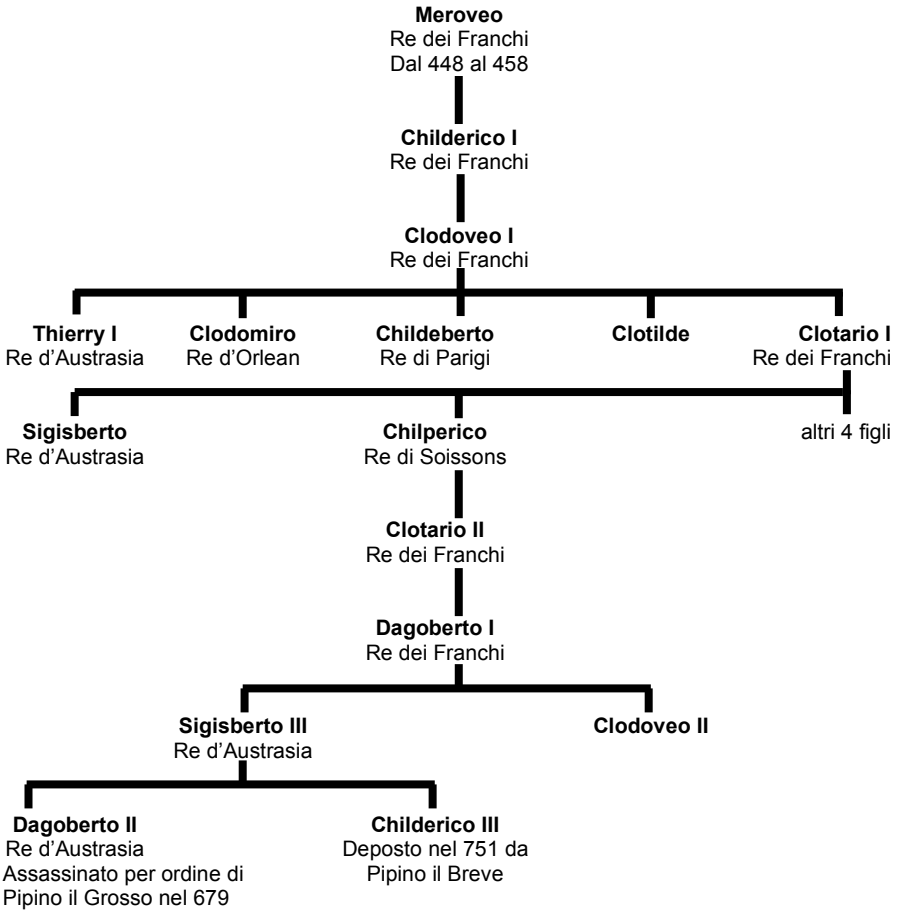


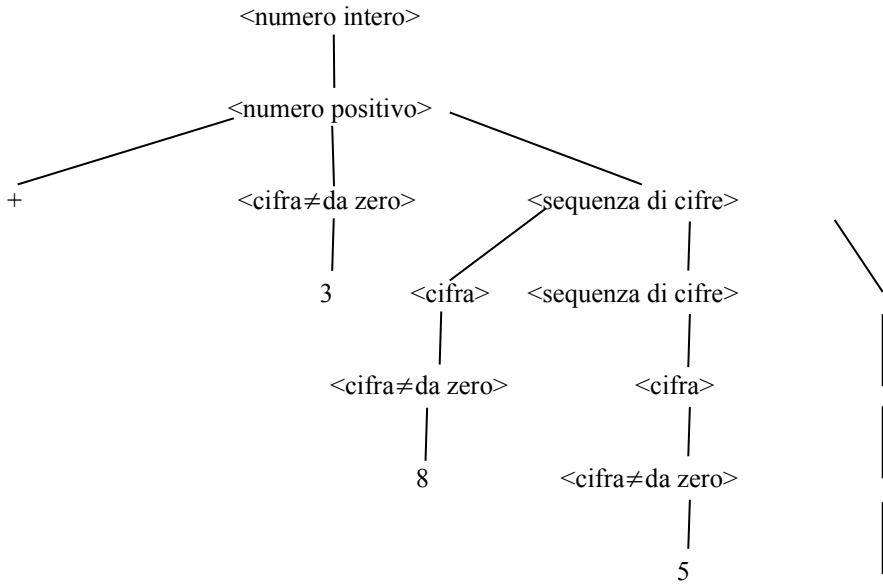
Fig. n. 6.2 - La dinastia dei re Merovingi

Gli alberi sono stati tradizionalmente utilizzati per rappresentare strutture gerarchiche, quale ad esempio l'organigramma di un'istituzione, che potrebbe essere l'Università degli Studi del Molise. In questo caso la radice rappresenta il Magnifico Rettore.

L'albero sintattico è un particolare albero che ha le seguenti caratteristiche:

- la radice è etichettata con l'assioma;
- i nodi sono etichettati dai simboli non terminali;
- le foglie sono etichettate da simboli terminali;
- i figli di ciascun nodo sono ottenuti applicando al padre una regola grammaticale.

Quindi, l'albero sintattico della stringa +385 dell'esempio sui numeri interi può essere così rappresentato.



7

La Macchina di Turing

In questo capitolo sarà esaminato il modello di calcolo denominato **Macchina di Turing** (MdT). Tale modello di calcolo è stato introdotto nel 1937 da **Alan Turing** (1912 - 1954), uno dei matematici più eclettici di questo secolo (a tal proposito, si veda l'appendice A, contenente un sunto della sua biografia). Il merito di Turing consiste essenzialmente nel fatto di aver descritto un calcolatore moderno prima che la tecnologia avesse raggiunto il punto in cui la sua costruzione fosse una proposta realistica.

Ricordiamo, inoltre, che durante la seconda guerra mondiale si è occupato anche della decifrazione del codice Enigma usato dai tedeschi e che successivamente durante la guerra fredda ha continuato a lavorare nell'ambiente militare.

La **MdT** è **importantissima** per **l'informatica**, poiché:

- costituisce **un modello di calcolo** assolutamente **generale** (*tesi di Church-Turing*): in altre parole, essa è in grado di risolvere qualsiasi problema algoritmicamente risolubile. Potremmo dire, in modo più accattivante, che essa è in grado di simulare ogni altro linguaggio di programmazione ed ogni altro modello di calcolo;
- ci permette di **definire** esattamente la nozione di **algoritmo**;
- ci permette di **definire** in modo semplice ed inequivocabile la nozione di risorse utilizzate da un algoritmo (**spazio e tempo**);
- ci permette di **riconoscere**, almeno parzialmente, i **linguaggi di tipo 0.**

Grazie all'esistenza di un modello di calcolo assolutamente generale, quale la MdT, la nozione d'irrisolubilità algoritmica introdotta precedentemente assume quindi un significato ben preciso.

Al fine di agevolare la comprensione dell'argomento, in questo capitolo saranno presentate alcune MdT molto semplici.

7.1 Definizione di Macchina di Turing

Una Macchina di Turing consiste di:

- un **nastro infinito**, suddiviso in celle. Ogni cella può contenere un solo simbolo, tratto da un insieme finito *S* detto alfabeto esterno;
- una **testina** capace di leggere un simbolo da una cella, scrivere un simbolo in una cella, e muoversi di una posizione sul nastro, in entrambe le direzioni;
- un insieme finito *Q* di **stati**, tali che la macchina si trovi esattamente in uno di essi in ciascun istante;
- un **programma**, che specifica esattamente cosa fare per risolvere un problema specifico.

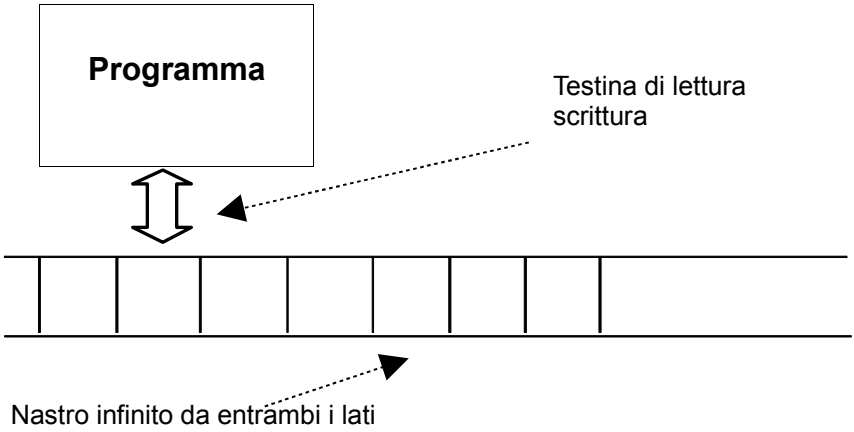


Fig. n. 7.1 – Lo Schema della Macchina di Turing

7.1.1 L'alfabeto esterno della MdT

L'alfabeto esterno $S = \{s_1, \ldots, s_k\}$ è utilizzato per codificare l'informazione in input e quella che la MdT produce nel corso della computazione.

Assumiamo che *S* contenga sempre un simbolo speciale detto lettera vuota o blank, indicato con *b*. Diciamo che una cella è vuota se contiene *b*. Scrivendo la lettera vuota b in una cella viene cancellato il contenuto di quella cella.

7.1.2 Gli stati della MdT

I possibili stati di una macchina di Turing sono denotati $q_1, q_2, ..., q_n, q_0, q_{f1}, q_{f2}, q_{fm}$

- Gli stati $q_1, ... q_n$ sono detti **stati ordinari**.
- Lo stato q_0 è detto **stato iniziale**.
- Gli stati $q_{f1}, q_{f2}, ..., q_{fm}$ sono detti **stati finali**.

7.1.3 Configurazione iniziale della MdT

La macchina di Turing si trova inizialmente nello stato q_o.

L'informazione da elaborare è contenuta in celle contigue del nastro, ed è codificata utilizzando i simboli dell'alfabeto esterno *S*.

Tutte le altre celle del nastro contengono inizialmente la lettera vuota.

La testina di lettura/scrittura è posizionata in corrispondenza del primo simbolo valido (quello che si trova più a sinistra).

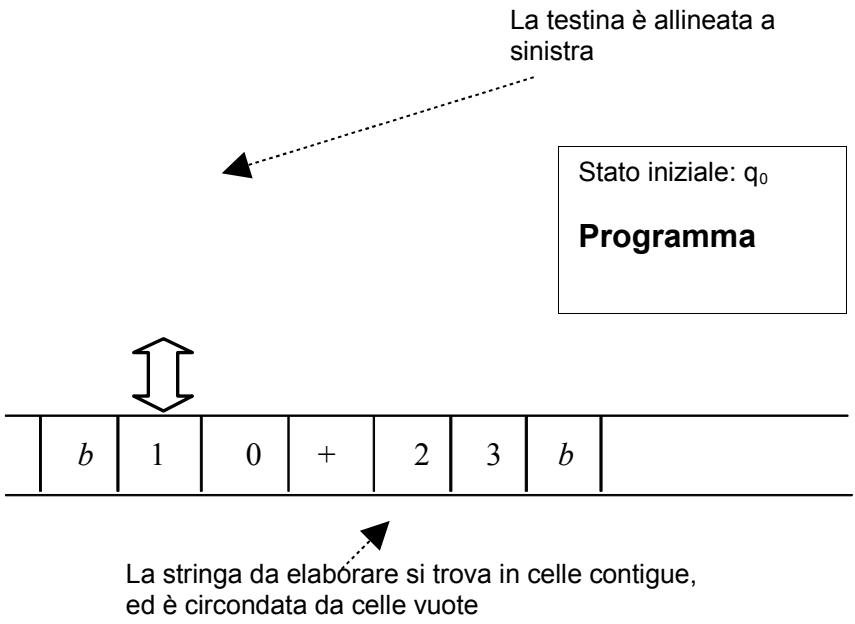


Fig. n. 7.2 – La configurazione iniziale della MdT

7.1.4 Il Programma della MdT

Se indichiamo con *q* lo stato in cui la MdT si trova ad un certo istante, e con *s* il simbolo che si trova sul nastro in corrispondenza della testina, il programma dovrà specificare, per ogni possibile coppia *(q, s)*:

- in quale nuovo stato la MdT dovrà portarsi;
- il simbolo da scrivere sul nastro nella posizione corrente;
- se la testina di lettura debba rimanere ferma, spostarsi di una posizione a sinistra, o spostarsi di una posizione a destra.

Ogni qualvolta consideriamo il carattere presente sotto la testina e lo stato in cui ci troviamo, ed effettuiamo le azioni corrispondenti, parliamo di una **transizione** (o *passo*).

Nota bene. In seguito ad una transizione:

- la testina potrebbe rimanere ferma;
- lo stato potrebbe rimanere inalterato;
- il carattere scritto potrebbe essere lo stesso presente sotto la testina.

Molto spesso si confonde la transizione (o passo) con il movimento della testina: non commettete questo errore!

7.1.5 Il Programma come Funzione

Sia T l'insieme *{ferma, sinistra, destra}*. Possiamo vedere il programma eseguito da una MdT come una funzione

$$f: Q \times S \rightarrow Q \times S \times T$$

È possibile specificare un programma utilizzando una matrice, detta **matrice funzionale** o matrice di transizione, le cui righe sono indicizzate utilizzando l'alfabeto esterno, e le cui colonne sono indicizzate utilizzando l'insieme degli stati.

Il generico elemento della matrice di indice *(q, s)* conterrà *f(q, s)*.

ESEMPIO.

	q_o	q_1	q_f
1	*(q_1,1, DESTRA)*	*(q_0, 1, DESTRA)*	
b	*(q_f, P, FERMO)*	*(q_f, D, FERMO)*	
P			
D			

Riferendoci alla matrice funzionale qui raffigurata, $f(q_o\, b)$ sarà uguale a (q_f, P, FERMO).
Le cellette lasciate vuote stanno ad indicare delle combinazioni (q, s), ovvero delle combinazioni (stato, carattere) che non si non possono mai verificare. ■

7.1.6 Terminazione della computazione

La computazione termina in uno dei seguenti casi:

- *quando la MdT raggiunge uno stato finale:* diremo in questo caso che la MdT si è portata in una configurazione finale;
- *quando la MdT si porta in una configurazione in cui nessuna mossa è definita:* diremo in questo caso che la MdT si è portata in una configurazione di *alt*.

Per semplificare la trattazione, possiamo sempre assumere che quando la MdT si trova in uno stato finale nessuna mossa sia definita, e pertanto la MdT giungerà in entrambi i casi ad una configurazione di *alt*.

Qual è allora la differenza tra le due condizioni di terminazione? Se la MdT giunge ad uno stato finale, questo sta ad indicare che essa ha risolto il problema da noi posto, e sul nastro avremo (da qualche parte) la soluzione del problema.

Per semplificare la trattazione, in tutti gli esempi presentati in questa lezione supporremo:

- di avere un solo stato finale, denotato q_f
- che la MdT sia in grado di risolvere sempre il problema da noi posto.

Nota bene. Non è detto che la computazione necessariamente termini, ovvero che la macchina si arresti, prima o poi. Può accadere infatti che la MdT entri in un ciclo infinito, ed in questo caso la computazione non terminerà mai.

Definizione. Diciamo che una MdT **converge** se essa si arresta (*prima o poi*), **diverge** in caso contrario. ■

7.1.7 Tempo di esecuzione

Definiamo il tempo di esecuzione come il numero di transizioni (o passi) effettuate, ovvero il numero di volte in cui dobbiamo andare a controllare sulla tabellina (la matrice funzionale) cosa fare.

Nota bene. Il tempo di esecuzione non ha niente a che fare con il numero di spostamenti che effettua la testina di lettura scrittura, che rappresenta un parametro assolutamente irrilevante!

7.1.8 Occupazione istantanea di memoria

Durante l'esecuzione di un programma, la MdT utilizzerà un certo numero di cellette del nastro. L'occupazione di memoria ad un certo istante è definita come il numero di cellette compreso tra la prima cella non vuota e l'ultima cella non vuota, incluse.

Nota bene. In base alla definizione appena data, nell'esempio raffigurato qui sotto, l'occupazione di memoria è di 6 celle, e non di 4, anche se vi sono 2 celle vuote comprese tra la prima e l'ultima non vuote!

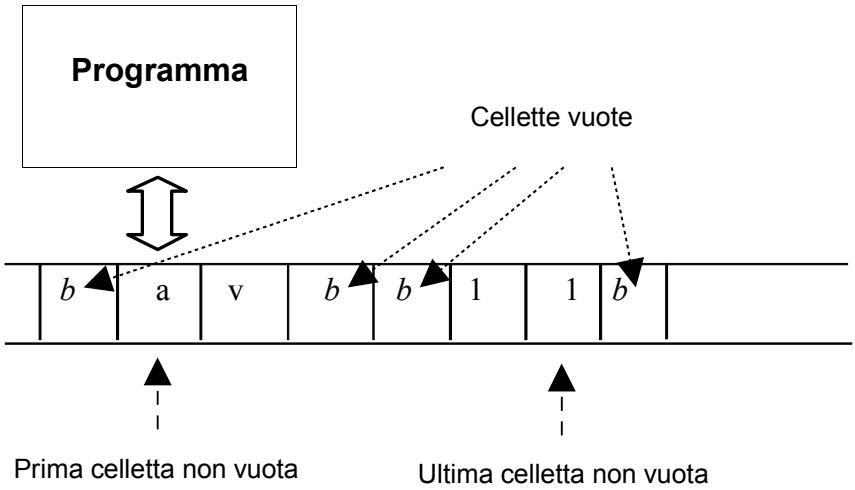


Fig. n. 7.3 - Configurazione della MdT ad un certo istante

Si noti che lo spazio utilizzato durante l'esecuzione di un programma può variare da una configurazione all'altra, a causa delle operazioni di scrittura.

Definiamo quindi lo spazio di memoria utilizzato durante l'esecuzione di un certo programma (che termina, chiaramente) come la più grande tra le occupazioni istantanee di memoria.

7.2 Ipotesi fondamentale della Teoria degli Algoritmi

Tesi di Church-Turing. *Qualunque algoritmo può essere espresso sotto forma di matrice funzionale ed essere eseguito dalla corrispondente Macchina di Turing.* ■

La **Tesi di Church-Turing** in effetti è una **congettura**, ovvero non esiste una dimostrazione formale di tale asserzione, pur essendo universalmente accettata.

Ricordiamo a tale proposito che una **tesi** è un'asserzione che si può dimostrare formalmente, a partire da alcuni **assiomi** ed utilizzando delle regole di ragionamento logico-deduttivo dette **regole di inferenza**. Ad esempio, utilizzando gli assiomi dell'algebra ed alcune semplici regolette, possiamo dimostrare che il quadrato di un qualsiasi numero intero è positivo.

La Tesi di Church-Turing ci dice semplicemente che la MdT è **il modello di calcolo più generale** che conosciamo, ovvero può essere utilizzato per risolvere qualsiasi problema risolubile algoritmicamente.

In alternativa, possiamo dire che la MdT è in grado di emulare (ovvero, imitare) qualsiasi altro modello di calcolo. In particolare un linguaggio di programmazione (ad esempio, il **C** o il **Pascal**), è un modello di calcolo. Quindi, una MdT è in grado di emulare qualsiasi linguaggio di programmazione, ovvero comportarsi come si comporterebbe l'interprete di un qualsiasi linguaggio di programmazione, se opportunamente programmata (per programmazione di una MdT si intende la specifica della corrispondente matrice funzionale).

7.3 Irrisolubità

Alla luce della Tesi di Church-Turing, possiamo riformulare la nozione di irrisolubilità data in precedenza come segue:

> *Un problema è non risolubile algoritmicamente se nessuna Macchina di Turing è in grado di fornire la soluzione al problema in tempo finito.*

Abbiamo visto in precedenza che esistono problemi irrisolubili algoritmicamente, e che la logica matematica si occupa (tra l'altro) dei limiti della computabilità, ovvero dello studio e della classificazione di tali problemi.

7.4 Esempi di MdT

7.4.1 Una MdT per il Controllo di Parità

Definiamo una MdT in grado di determinare se una stringa assegnata di 1 contenga un numero pari o dispari di 1. La MdT dovrà scrivere al termine della stringa assegnata il simbolo D o il simbolo P, dove D sta per dispari e P sta per pari. Ad esempio, data la configurazione iniziale:

b	b	b	1	1	1	b	b	b	b

al termine della computazione il nastro dovrà contenere:

b	b	b	1	1	1	D	b	b	b

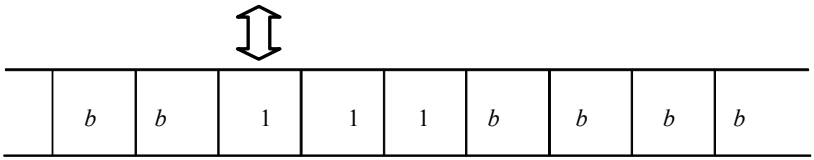
Qui di seguito é descritta una MdT adatta al nostro scopo:

$S = \{1, b, P, D\}$, $Q = \{q_o, q_1, q_f\}$

e la matrice di transizione della MdT è di seguito mostrata:

	q_o	q_1	q_f
1	*(q_1,1, DESTRA)*	*(q_0, 1, DESTRA)*	
b	*(q_f, P, FERMO)*	*(q_f, D, FERMO)*	
P			
D			

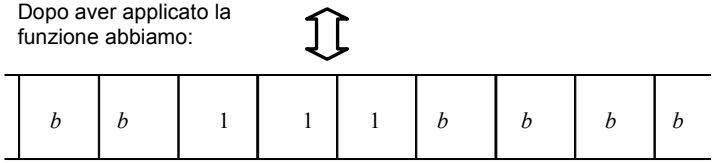
Riportiamo la traccia dell'esecuzione a partire dalla configurazione iniziale:



Primo passo:

$(q_0, 1) \rightarrow (q_1, 1, \texttt{DESTRA})$

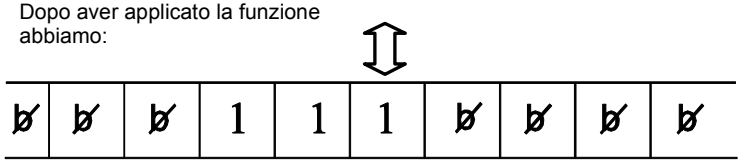
Dopo aver applicato la funzione abbiamo:



Secondo passo:

$(q_1, 1) \rightarrow (q_0, 1, \texttt{DESTRA})$

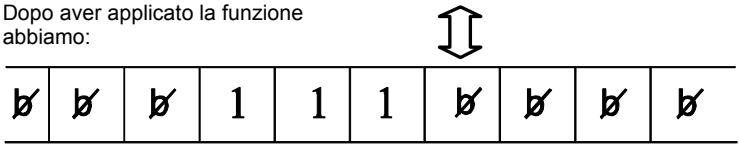
Dopo aver applicato la funzione abbiamo:



Terzo passo:

$(q_0, 1) \rightarrow (q_1, 1, \texttt{DESTRA})$

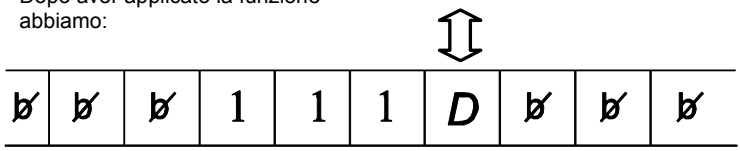
Dopo aver applicato la funzione abbiamo:



Quarto passo:

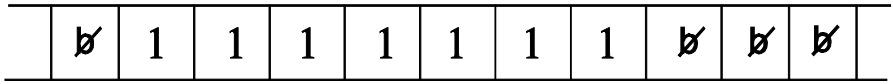
$(q_1, ø) \rightarrow (q_f, D, \text{FERMO})$

Dopo aver applicato la funzione abbiamo:

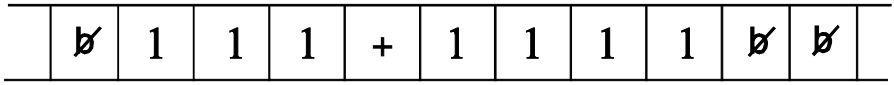


7.4.2 Una MdT per addizionare due numeri espressi in notazione unaria

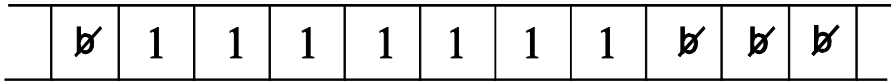
Rappresentiamo in unario un numero n come una sequenza di n simboli identici. Ad esempio, il numero 7 è rappresentato come segue:



Rappresentiamo l'addizione dei numeri 3 e 4 espressi in notazione unaria come segue:



Voglio ottenere la seguente configurazione sul nastro:



Qui di seguito é descritta una Macchina di Turing adatta al nostro scopo.
L'alfabeto esterno e l'insieme degli stati sono rispettivamente:

$S = \{1, b, +\}$ $Q = \{q_0, q_1, q_2, q_f\}$.

La funzione di transizione può essere descritta come segue in forma matriciale:

	q_o	q_1	q_2	q_f
1	*(q_o, 1, DESTRA)*	*(q_1, 1, DESTRA)*	*(q_f, b, FERMO)*	
+	*(q_1, 1, DESTRA)*			
b		*(q_2, b, SINISTRA)*		

7.4.3 Una MdT per il Riconoscimento delle Stringhe Palindrome

Definiamo una MdT in grado di verificare se una stringa sia *palindroma*: una stringa si dice palindroma se la sequenza di caratteri da cui è composta è la stessa sia se la si legge da sinistra verso destra che da destra verso sinistra. La MdT dovrà scrivere al termine della stringa assegnata il simbolo S se la stringa è palindroma o il simbolo N altrimenti. Ad esempio, data la configurazione iniziale:

b	a	c	c	a	*b*

al termine della computazione il nastro dovrà contenere:

b	*b*	*b*	S	*b*	*b*

Qui di seguito é descritta una MdT adatta al nostro scopo:

S = { a , c , b , S , N }, Q = { q_0 , q_1 , q_2 , q_3 , q_4 , q_5 , q_f }

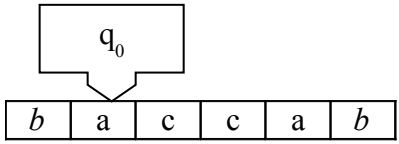
e la matrice di transizione della MdT è di seguito mostrata:

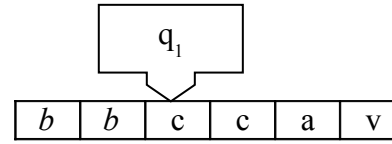
	a	c	*b*	S	N
q_0	(q_1, *b*,*DESTRA*)	(q_2, *b*, *DESTRA*)	(q_f, S, *FERMO*)		
q_1	(q_1, a, *DESTRA*)	(q_1, c, *DESTRA*)	(q_1, *b*, *SINISTRA*)		
q_2	(q_2, a, *DESTRA*)	(q_2, c, *DESTRA*)	(q_1, *b*, *SINISTRA*)		
q_3	(q_5, *b*, *SINISTRA*)	(q_f, N, *FERMO*)	(q_f, S, *FERMO*)		
q_4	(q_f, N, *FERMO*)	(q_5, *b*, *SINISTRA*)	(q_f, S, *FERMO*)		
q_5	(q_5, a, *SINISTRA*)	(q_5, c, *SINISTRA*)	(q_0, *b*, *DESTRA*)		
q_f					

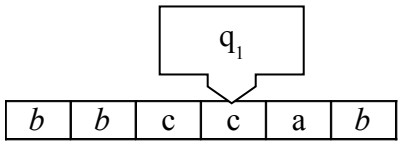
Diamo una breve descrizione degli stati della MdT. Nello stato iniziale q_0 la MdT legge il primo carattere della stringa di input, *a* oppure *c*, e da qui si porta nello stato q_1 se il primo carattere letto é *a*, q_2 se il primo carattere letto é *c*. Gli stati q_1 e q_2 occorrono per scorrere tutta la stringa, ed arrestarsi non appena si incontri il carattere blank. Giunti in corrispondenza del primo blank, la MdT si porta nello

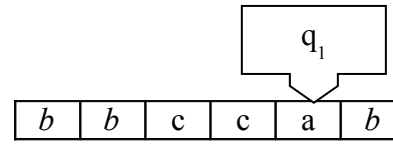
stato q_3 se si trova nello stato q_1, oppure nello stato q_4 se si trova nello stato q_2, e quindi la testina viene fatta spostare di una posizione a sinistra. A questo punto l'ultimo carattere presente viene confrontato con il primo letto, e, se i due sono uguali, esso viene cancellato, e la MdT si porta nello stato q_5 (stato di *riavvolgimento*), per portare la testina in corrispondenza del primo carattere a sinistra non blank presente sul nastro. Giunti qui, si provvede a confrontare nuovamente il primo carattere e l'ultimo, e così via… Si noti che a questo punto, se non ci sono più caratteri da confrontare, la MdT si porta nello stato finale q_f e quindi scrive sul nastro il simbolo S per comunicare che la stringa data è palindroma.

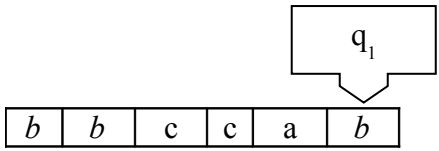
Riportiamo la traccia dell'esecuzione a partire dalla configurazione iniziale:

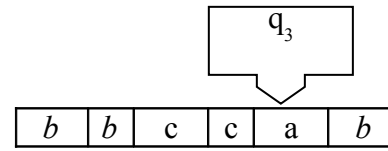

q0
b a c c a b

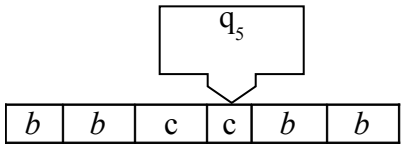

q1
b b c c a v

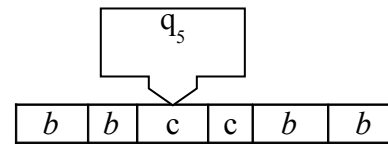

q1
b b c c a b

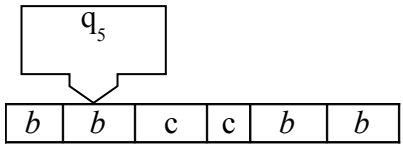

q1
b b c c a b

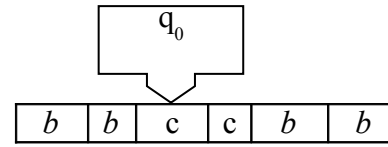

q1
b b c c a b

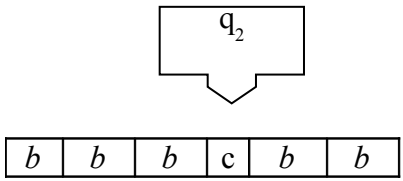

q3
b b c c a b

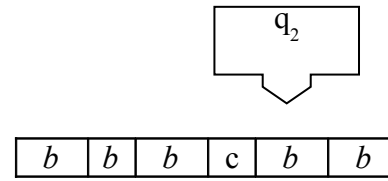

q5
b b c c b b

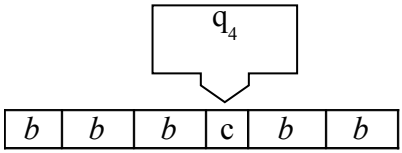

q5
b b c c b b

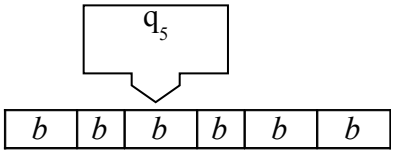

q5
b b c c b b

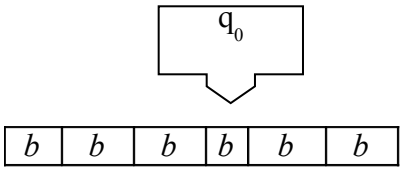

q0
b b c c b b

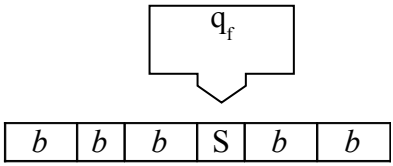

q2
b b b c b b


q2
b b b c b b


q4
b b b c b b


q5
b b b b b b


q0
b b b b b b


qf
b b b S b b

7.5 La Macchina Universale di Turing

Abbiamo visto che la MdT è il modello di calcolo più generale esistente. In altre parole, essa può essere utilizzata per risolvere qualsiasi problema algoritmicamente risolubile.

Abbiamo anche visto come sia possibile **descrivere completamente** una MdT utilizzando una tabella, contenente tre elementi in ogni casellina, chiamata **matrice funzionale**. Senza entrare nei dettagli (peraltro, abbastanza semplici) è possibile codificare tale matrice sotto forma di una stringa β.

A questo punto possiamo costruire una MdT A che accetta in input la stringa β ed una stringa x di simboli dell'alfabeto esterno di B e produce lo stesso risultato che B produrrebbe con x in input: **chiamiamo tale Macchina di Turing A una Macchina di Turing Universale**.

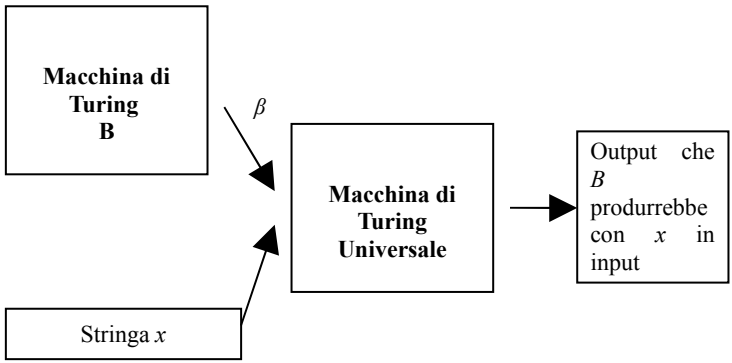


Fig. n. 7. 4–Schema della MdT Universale

7.6 Il Problema dell'Alt

Come abbiamo visto in precedenza, una Macchina di Turing può arrestarsi (*convergere*) se:

- si porta in uno stato finale, oppure
- si porta in una configurazione in cui nessuna mossa è definita

o, in alternativa, può entrare in un ciclo infinito (*divergere*), ovvero non arrestarsi mai.

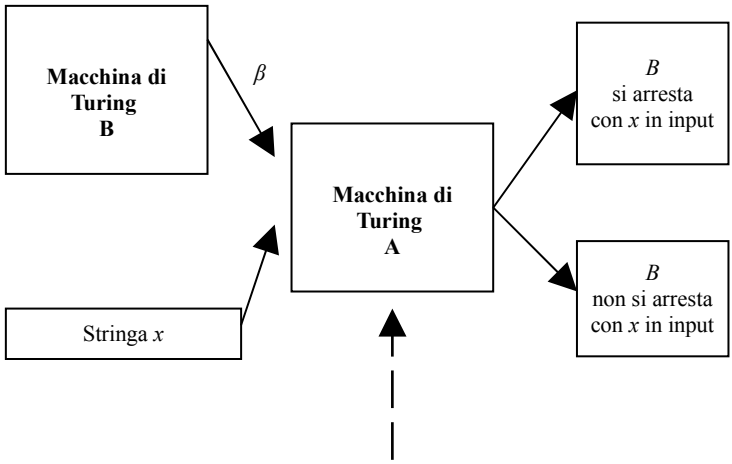
Il Problema dell'Alt è il seguente: è possibile costruire una Macchina di Turing *A* che accetti in input:

- la descrizione β di una seconda Macchina di Turing *B* (vedi sezione riguardante MdT Universale)*;*
- una stringa *x* di simboli dell'alfabeto esterno di *B*

e restituisca:

- **Converge,** se *B* termina con *x* in input;
- **Diverge,** altrimenti.

Si dimostra che il Problema dell'Alt è irrisolubile, ovvero una siffatta Macchina di Turing A **non può esistere**! La dimostrazione è omessa.



7.6.1 Conseguenze del Teorema dell'Alt sull'attività di programmazione

Poiché la MdT rappresenta il modello di calcolo più generale conosciuto, nessun linguaggio di programmazione esistente o futuro potrà mai offrire una capacità di calcolo superiore ad essa.

Quindi, il Teorema dell'Alt si può applicare tranquillamente ai linguaggi di programmazione.

Il teorema asserisce che non è possibile costruire un programma *A* che:

- accetti in input:
    - un altro programma *B*;
    - una stringa *x*, che rappresenta (ovviamente) l'istanza di un certo problema;
- sia in grado di determinare se il programma *B* termini o no sul particolare input *x*.

Nota bene. Il Teorema dell'Alt non asserisce assolutamente che non è possibile determinare se un certo programma *B* termini avendo in input una determinata stringa *x*! Molto spesso possiamo garantire la terminazione di un *particolare* programma (molto semplice) da noi costruito, per ogni istanza *x* dell'input. Potremmo, in teoria, persino costruire un programma *A* che accetti in input un programma *B* ed una stringa *x* e sia in grado di stabilire la terminazione del programma *B,* con *x* in input, la maggior parte delle volte! Ciò che il teorema realmente asserisce è che non è possibile costruire un tale strumento di verifica che funzioni per qualsiasi *B* e qualsiasi *x*!

7.7 Il Test di Turing

Il Test di Turing è stato introdotto da A.M. Turing come "gioco dell'imitazione" nel suo articolo "Computing Machinery and Intelligence", apparso sulla rivista Mind nel 1950 (Vol. 59, No. 236, pp. 433-460).

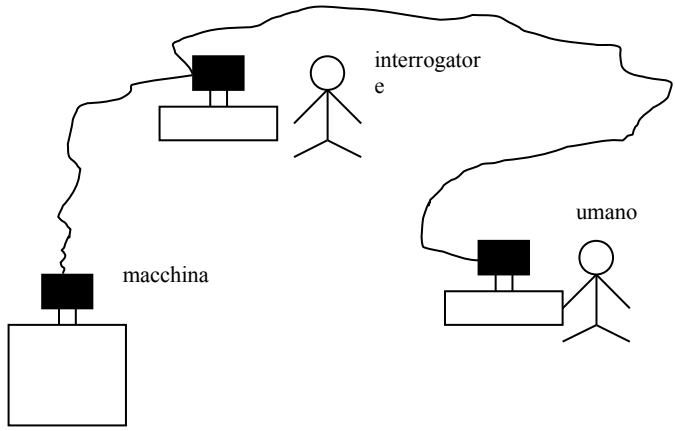
Questo storico articolo inizia così:

> *Desidero considerare il seguente problema "Può una macchina pensare?". A tal fine occorre definire innanzitutto il significato dei termini "macchina" e "pensare"...*

Al giorno d'oggi possiamo formulare il Test di Turing come segue: un **umano**, detto **l'interrogatore**, è collegato attraverso un **terminale** ad un altro umano e ad una **macchina**. Il suo obiettivo è di **capire chi sia la macchina e chi l'umano**, semplicemente ponendo delle domande. Se la macchina riesce ad ingannare l'interrogatore, la macchina è intelligente.

Il Test di Turing, com'era ovvio attendersi, ha scatenato negli ultimi 50 anni accese discussioni nel campo dell'Intelligenza Artificiale, della filosofia e delle scienze cognitive.

Ad oggi, nessuna macchina ha superato il Test di Turing.



Esercizi

1. Costruire una MdT capace di riconoscere se una stringa assegnata è palindroma. Una stringa si dice *palindroma* se è uguale quando è letta da sinistra verso destra o da destra verso sinistra. Ad esempio, la stringa

   anna

   e la stringa

   madam I m adam

   sono palindrome.

2. Costruire una MdT capace di riflettere (capovolgere) una stringa data in input.

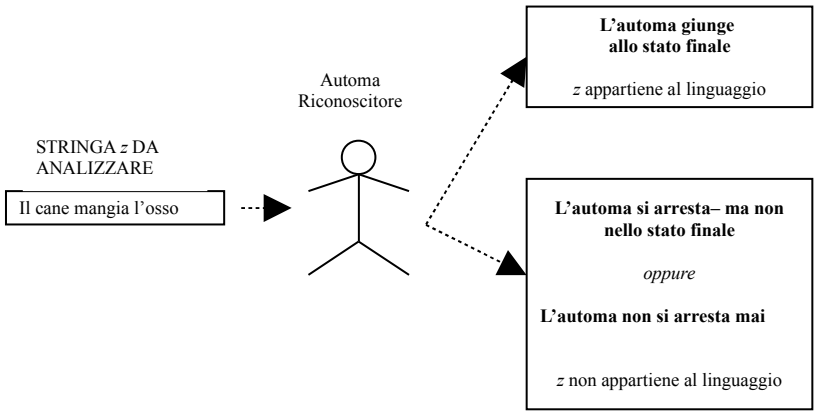
8

La Classificazione delle Grammatiche secondo Chomsky

8.1 Premessa

Prima di procedere con la classificazione richiamiamo il concetto di automa riconoscitore di un linguaggio.

Diciamo che una stringa *z* è accettata da un automa se, partendo dalla configurazione iniziale ed avendo *z* sul nastro di ingresso, l'automa può eseguire una sequenza finita di mosse che lo portano ad uno *stato finale*.



Possiamo allora definire un linguaggio ***L*** come l'insieme delle stringhe di simboli terminali accettate da un automa. Questo approccio è chiamato **riconoscitivo**, in contrasto all'approccio **generativo** che utilizza una grammatica per generare tutte ed esclusivamente le frasi di un certo linguaggio ***L***.

I due approcci, riconoscitivo e generativo, sono totalmente equivalenti. L'approccio **generativo** è utilizzato ad esempio dal **programmatore** per scrivere programmi sintatticamente corretti, mentre l'approccio **riconoscitivo** è utilizzato dal compilatore (in particolare, **dall'analizzatore sintattico** presente all'interno del

compilatore, che non è altro che l'automa raffigurato qui sopra) per decidere se un certo programma è sintatticamente corretto.

Ricordiamo che per descrivere una grammatica occorre fornire:

- l'insieme dei simboli terminali *T*;
- l'insieme dei simboli non terminali *N*;
- un insieme *P* di *regole di riscrittura* (o *regole di produzione*);
- l'assioma *S*.

Una regola di riscrittura si presenta come

$$v \rightarrow w$$

dove *v* e *w* sono stringhe arbitrarie (anche nulle) di simboli terminali e non terminali, ma *v* deve contenere almeno un simbolo non terminale.

Nel 1956, il linguista americano **Noam Chomsky** (1928-vivente) classificò le grammatiche in quattro classi, denotate rispettivamente 0, 1, 2, 3. Ciascuna classe comprende tutte le successive. Questa classificazione si riflette ovviamente sugli automi corrispondenti, per cui parleremo di:

- automi di tipo 0, in grado di riconoscere linguaggi di tipo 0;
- automi di tipo 1, in grado di riconoscere linguaggi di tipo 1;
- automi di tipo 2, in grado di riconoscere linguaggi di tipo 2;
- automi di tipo 3, in grado di riconoscere linguaggi di tipo 3.

Quando diciamo che un automa è **più potente** (o **più generale**) di un altro, intendiamo dire che esso è in grado di riconoscere una classe più vasta di linguaggi. Ad esempio, un automa di tipo 1 è più potente di un automa di tipo 3, poiché esso è in grado di riconoscere tutti i linguaggi di tipo 1, che comprendono quelli di tipo 3.

Attualmente Noam Chomsky è professore di *Linguistica, Sintassi, Semantica e Filosofia dei Linguaggi* presso il MIT.

8.2 Grammatiche di tipo 0

In questo caso, sia *v* che *w* non sono soggette ad alcuna limitazione.

Le grammatiche di tipo 0 sono anche dette *generali*. I linguaggi generati da grammatiche di tipo 0 si chiamano anche:

- *semidecidibili;*
- *ricorsivamente enumerabili*.

AUTOMA RICONOSCITORE CORRISPONDENTE

Si dimostra che un linguaggio è di tipo 0 se e solo se è accettato da una Macchina di Turing provvista di memoria ausiliaria costituita da un **nastro di lunghezza illimitata**.

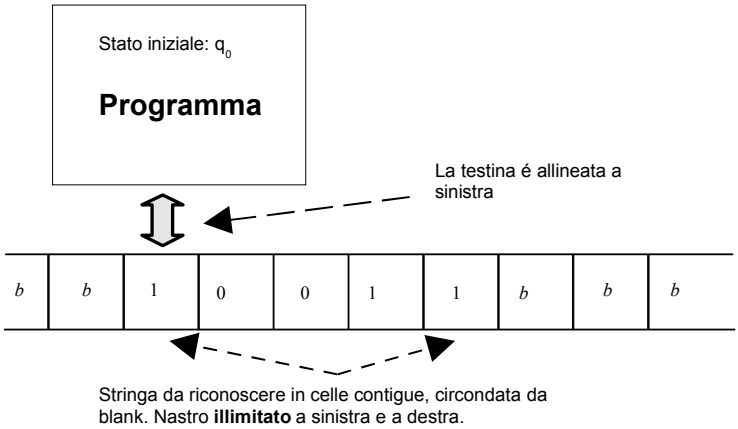


Fig. n. 8.1 – CONFIGURAZIONE INIZIALE DELLA MDT

Diciamo che i linguaggi di tipo 0 sono **semidecidibili**, poiché l'automa riconoscitore (ovvero, la Macchina di Turing a nastro illimitato):

- **si arresta in uno stato finale** se la stringa *z* appartiene al linguaggio;
- può **arrestarsi in uno stato non finale**, o **non arrestarsi affatto**, in caso contrario.

8.3 Grammatiche di tipo 1

Le grammatiche di tipo 1 sono dette anche *dipendenti dal contesto* e le regole di riscrittura hanno la forma:

$$yAz \rightarrow ywz$$

dove A è un non terminale, y e z sono stringhe arbitrarie (anche nulle) di terminali e non terminali, e per di più w deve contenere almeno un simbolo.

Le stringhe (di terminali e non terminali) y e z che circondano il non terminale A prendono il nome di contesto: la regola $yAz \rightarrow ywz$ sta ad indicare che la sostituzione di A con w può avvenire solo nel contesto specificato, da cui il nome di grammatiche contestuali.

Nota bene. É ammessa la regola

$$S \rightarrow \varepsilon$$

(dove ε rappresenta la stringa nulla) se e solo se S non compare nella parte destra di alcuna regola di produzione.

I linguaggi generati da grammatiche di tipo 1 si chiamano anche:

- *decidibili;*
- *ricorsivi.*

Un'altra Caratterizzazione dei Linguaggi di Tipo 1

Poiché le regole di riscrittura hanno la forma:

$$yAz \rightarrow ywz$$

dove A è un non terminale e w deve contenere almeno un simbolo, vediamo immediatamente che in ogni regola di produzione, la lunghezza (ovvero, il numero di simboli) della parte sinistra è minore o uguale della lunghezza della parte destra.

Si può anche dimostrare che se tutte le regole di riscrittura si presentano come:

$$v \rightarrow w \qquad |v| \leq |w|$$

ovvero:

- *v* e *w* sono stringhe arbitrarie (anche nulle) di simboli terminali e non terminali;
- *v* contiene almeno un simbolo non terminale;
- la lunghezza di *v* è minore o uguale della lunghezza di *w*

allora il linguaggio generato è di tipo 1.

Quindi, otteniamo la seguente caratterizzazione alternativa dei linguaggi di tipo 1:

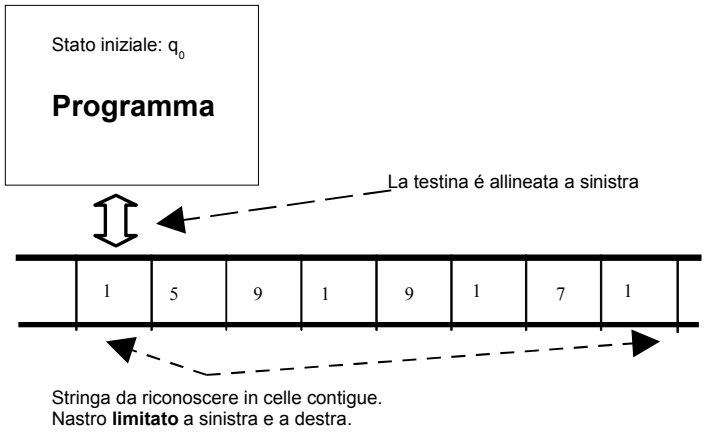
Definizione alternativa. Un **linguaggio** è **di tipo 1** se può essere generato da una grammatica in cui le regole di riscrittura si presentano come:

$$v \rightarrow w \qquad |v| \leq |w|$$

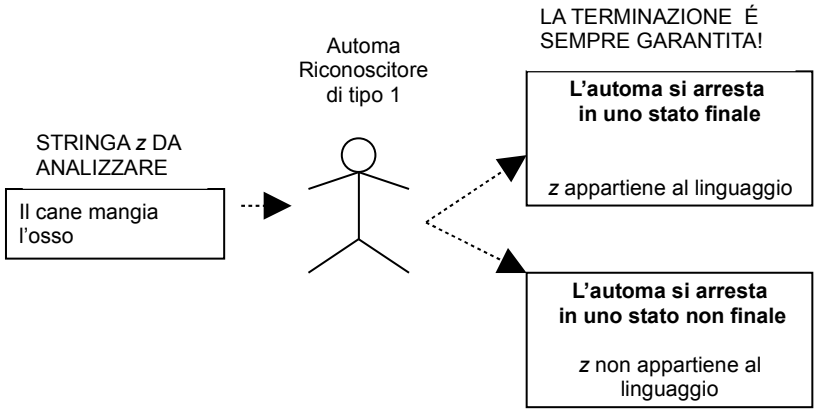
dove *v* e *w* sono stringhe arbitrarie (anche nulle) di simboli terminali e non terminali, e *v* contiene almeno un simbolo non terminale. ■

AUTOMA RICONOSCITORE CORRISPONDENTE

Si dimostra che un linguaggio è di tipo 1 se e solo se è accettato da una Macchina di Turing provvista di memoria ausiliaria costituita da un **nastro di lunghezza pari** a quella della stringa da analizzare.



Diciamo che i linguaggi di tipo 1 sono decidibili, poiché in ogni caso l'automa riconoscitore (ovvero, la Macchina di Turing a nastro limitato) si **arresta sempre**, sia che la stringa *z* appartenga al linguaggio, sia che essa non vi appartenga.


LA TERMINAZIONE É SEMPRE GARANTITA!
Automa Riconoscitore di tipo 1
STRINGA *z* DA ANALIZZARE
Il cane mangia l'osso
L'automa si arresta in uno stato finale
z appartiene al linguaggio
L'automa si arresta in uno stato non finale
z non appartiene al linguaggio

8.4 Grammatiche di tipo 2

Le grammatiche di tipo 2 sono dette anche *libere dal contesto* e le regole di riscrittura hanno la forma:

$$A \rightarrow w$$

dove A è un non terminale, w è una stringa arbitraria (ma non nulla, ovvero w deve contenere almeno un simbolo) di simboli terminali e non terminali.

I linguaggi generati da grammatiche di tipo 2 si chiamano anche *non contestuali o context free.*

Automa Riconoscitore Corrispondente

Di dimostra che un linguaggio è di tipo 2 se e solo se è accettato da un **automa a pila**, dotato di memoria ausiliaria organizzata come una pila.

Nota bene. I linguaggi di programmazione appartengono ad un sottoinsieme dei linguaggi di tipo 2, detto dei **linguaggi non contestuali deterministici**, ottenuti imponendo ulteriori restrizioni sulle grammatiche di tipo 2. Pur non entrando nel merito, desideriamo aggiungere che queste restrizioni sulle grammatiche dei linguaggi di programmazione sono motivate dal desiderio che l'analisi sintattica dei programmi sia efficiente (ovvero, non richieda troppo tempo).

8.5 Grammatiche di tipo 3

Le grammatiche di tipo 3 sono dette anche *lineari destre* e le regole di riscrittura hanno la forma:

- $A \rightarrow bC$
- $A \rightarrow d$

dove

- A e C sono simboli non terminali;
- b è un simbolo terminale;
- d è un simbolo terminale, oppure la stringa nulla ε.

I linguaggi generati da grammatiche di tipo 3 si chiamano anche *regolari*.

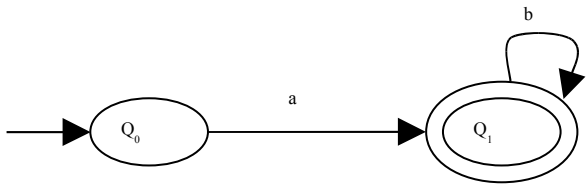
AUTOMA RICONOSCITORE CORRISPONDENTE

Si dimostra che un linguaggio è di tipo 3 se e solo se è accettato da un automa a stati finiti, ovvero sprovvisto di memoria ausiliaria.

ESEMPIO. *Il seguente automa riconosce il linguaggio regolare generata dalla grammatica avente come produzioni:*

S → a
S → aB
B → b
B → bB

dove S è l'assioma



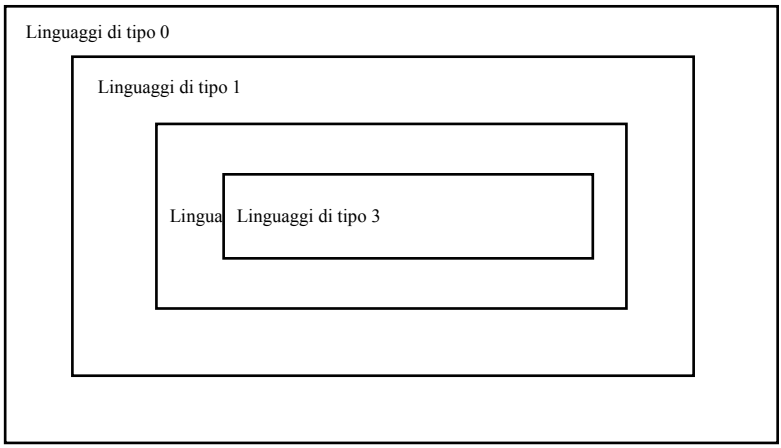
■

8.6 Teorema della Gerarchia

Si dimostra che:

- i linguaggi di tipo 3 sono contenuti in quelli di tipo 2;
- i linguaggi di tipo 2 sono contenuti in quelli di tipo 1;
- i linguaggi di tipo 1 sono contenuti in quelli di tipo 0.

Ovvero, graficamente:



La dimostrazione è omessa.

9

Gli Automi a Stati Finiti

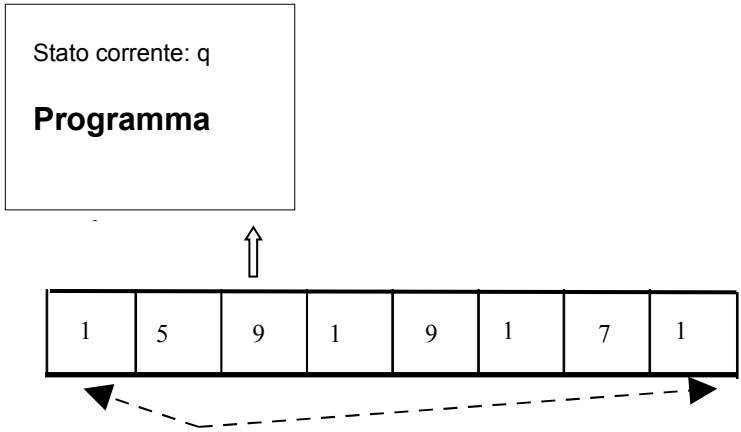
Gli automi a stati finiti costituiscono l'esempio più elementare di riconoscitori di un linguaggio. Tali automi sono dispositivi non dotati di memoria, ovvero non hanno la capacità di ricordare ciò che è successo in passato. A causa di questa profonda limitazione, la loro capacità computazionale è molto bassa. Tuttavia essi trovano applicazioni in innumerevoli contesti. Basti pensare, ad esempio, che:

- l'analizzatore lessicale di un linguaggio di programmazione (*che,* ricordiamolo*, ha il compito di riconoscere i token del linguaggio, ovvero le costanti, le variabili, le parole chiave, ecc.*) è generalmente un automa a stati finiti;
- il sistema di controllo di piccoli elettrodomestici è basato su un automa a stati finiti;
- il sistema di controllo dei moderno telefonini (*cellulari*) è basato anch'esso su un automa a stati finiti.

9.1 Definizione di Automa a Stati Finiti

Un automa a stati finiti **M** consiste di:

- un **nastro di ingresso finito**, suddiviso in celle. Ogni cella può contenere un solo simbolo, tratto da un insieme finito *S* detto alfabeto di ingresso;
- una **testina** capace di leggere un simbolo da una cella, e muoversi di una posizione sul nastro di ingresso, da sinistra verso destra;
- un insieme finito *Q* di **stati**, tali che l'automa si trovi esattamente in uno di essi in ciascun istante;
- un **programma**, che specifica esattamente cosa fare.



Stringa *x* da riconoscere in celle contigue.

9.1.1 L'alfabeto di ingresso

L'alfabeto di ingresso $S = \{s_1, \ldots, s_k\}$ è utilizzato esclusivamente per codificare l'informazione in input.

ESEMPIO. *Se il nostro automa dovrà riconoscere i **token** (ovvero, i vocaboli) di un linguaggio di programmazione, costituiti da:*

- *parole chiave (o riservate), ad esempio **if** o **while** in C;*
- *variabili, ad esempio **saldo** o **saldo_giornaliero** in C;*
- *costanti, ad esempio **ALIQUOTA** in C;*
- *tipi di dato definiti dall'utente, ad esempio **MioVettore** in C;*
- *ecc.*

allora, l'alfabeto di ingresso sarà costituito da tutti i caratteri alfabetici presenti sulla tastiera (a, b, c, ...), da tutti i caratteri numerici (0, 1, 2, ...), e dai caratteri di interpunzione (. , ; : ...). ■

9.1.2 Gli stati dell'automa

I possibili stati di un automa a stati finiti sono denotati $q_1, q_2, \ldots, q_n, q_0, q_{f1}, q_{f2}, \ldots, q_{fm}$

- lo stato q_0 è detto **stato iniziale**;
- gli stati $q_1, \ldots q_n$ sono detti **stati ordinari**;
- gli stati $q_{f1}, q_{f2}, \ldots, q_{fm}$ sono detti **stati finali**.

Lo stato indica una particolare situazione (o condizione) in cui l'automa si trova.

Riferendoci a situazioni note, possiamo trovarci nello stato di *allegria* o *tristezza*, ma non entrambe. Una barzelletta o un evento triste può modificare il nostro stato.

Oppure possiamo trovarci in uno ed uno solo dei possibili stati di *sano*, *ammalato* o *convalescente*. In tal caso, degli eventi esterni (virus, cure del medico, ecc.) causano il passaggio da uno stato ad un altro.

O ancora, possiamo trovarci in uno ed uno solo dei possibili stati di *dormiente* e *sveglio*, ad un certo istante. Un evento esterno, ad esempio il suono della sveglia, modifica il nostro stato nel caso in cui stiamo dormendo.

9.1.3 Il programma eseguito dall'automa

Se indichiamo con *q* lo stato in cui l'automa si trova ad un certo istante, con *s* il simbolo che si trova sul nastro di ingresso in corrispondenza della testina di lettura, il programma dovrà specificare, per ogni possibile coppia *(q, s)* in quale nuovo stato l'automa dovrà portarsi.

9.1.4 Il programma come funzione

Possiamo vedere, quindi, il programma eseguito da un automa a stati finiti come una funzione:

$$f: Q \times S \rightarrow Q$$

dal prodotto cartesiano dell'insieme Q degli stati per l'insieme S dell'alfabeto di ingresso all'insieme Q degli stati. Quindi, se abbiamo m stati ed un alfabeto di n simboli, dovremo considerare tutte le m x n possibili coppie.

9.2 Rappresentazione matriciale dell'automa

Possiamo rappresentare il nostro automa attraverso una tabellina (ovvero, una matrice) di *m* righe e *n* colonne, in cui le righe indicano i possibili stati e le colonne i possibili simboli dell'alfabeto.

Nella casellina contenuta in una particolare riga e colonna indicheremo lo stato in cui l'automa dovrà portarsi in corrispondenza della particolare combinazione stato-simbolo.

Tale tabella prende il nome di **matrice di transizione**.

Esempio. *Supponiamo che il nostro alfabeto sia costituito dai 3 simboli a*, *b* e *c*, *e di avere, oltre allo stato iniziale* q_0, *due stati ordinari* q_1 e q_2 *ed uno stato finale* q_f.

	A	b	c
q_0	q_1	q_f	q_0
q_1	q_0	q_1	q_f
q_2	q_f	q_0	q_0
q_f	q_2	q_f	q_2

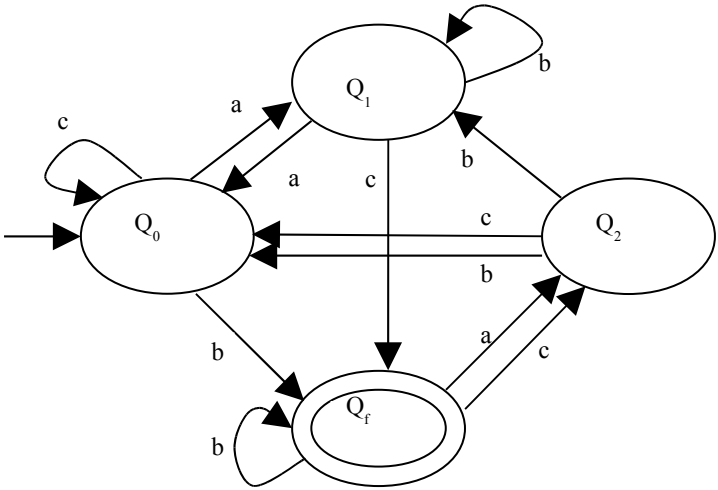
Se ci troviamo nello stato $\boldsymbol{q_2}$ *ed incontriamo il simbolo b, la tabellina ci indica che dovremo portarci nello stato* $\boldsymbol{q_0}$. ■

9.3 Rappresentazione grafica dell'automa

Possiamo rappresentare il nostro automa attraverso un grafo orientato, costruito come segue:

- il grafo è dotato di tanti vertici (*ovvero, tanti cerchietti*) quanti sono gli stati. Ogni vertice è etichettato con il nome dello stato (*ovvero, all'interno del cerchietto scriviamo il nome dello stato che esso rappresenta*);
- se dallo stato q_i incontrando il simbolo s (*dell'alfabeto di ingresso*) andiamo nello stato q_j , allora avremo un arco (*cioè una freccetta*) che va dallo stato q_i allo stato q_j etichettato con il simbolo s ;
- gli stati finali saranno rappresentati utilizzando due cerchietti concentrici;
- per distinguere lo stato iniziale dagli altri, utilizzeremo un arco che entra in tale stato e non parte da alcun altro stato.

Tale grafo prende il nome di **diagramma di transizione** o **diagramma degli stati** o **grafo degli stati.** Per l'automa rappresentato in forma matriciale nella pagina precedente, avremo il diagramma seguente:

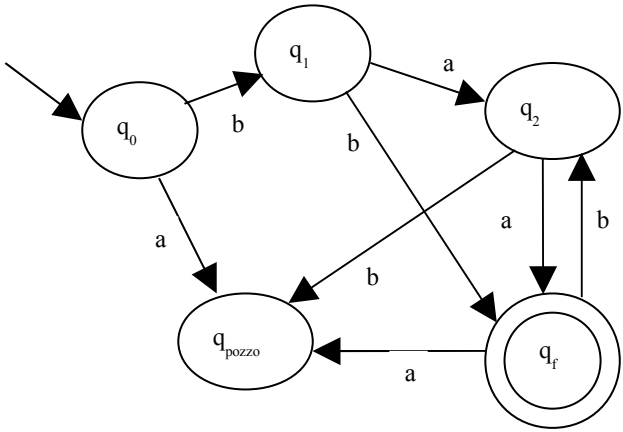


9.4 Stati Pozzo

Può accadere che, giunti ad una configurazione costituita da un certo carattere *s* presente sotto la testina e da certo uno stato *q*, possiamo già escludere dal linguaggio la stringa *x* da riconoscere, ovvero asserire che la stringa *x* in input non appartenga al linguaggio.

Per gestire tali situazioni, definiamo un (*uno solo basta*) nuovo stato ordinario q_{pozzo} in cui portarsi in questi casi; da tale stato pozzo nessuna transizione è possibile.

ESEMPIO.



Nell'automa qui illustrato, nello stato pozzo si va a finire in uno dei seguenti casi:

- *ci si trova nello stato q_0 e si incontra un carattere a;*
- *ci si trova nello stato q_2 e si incontra il carattere b;*
- *ci si trova nello stato q_f e si incontra il carattere a.*

■

L'introduzione dello stato pozzo può rendere la rappresentazione grafica dell'automa molto ingarbugliata! Per ovviare a questo inconveniente, lo stato pozzo si omette, e si omettono anche tutti gli archi (*le freccette*) che portano ad esso ottenendo, ad esempio, dall'automa precedente il seguente automa, costituito da un minor numero di stati:

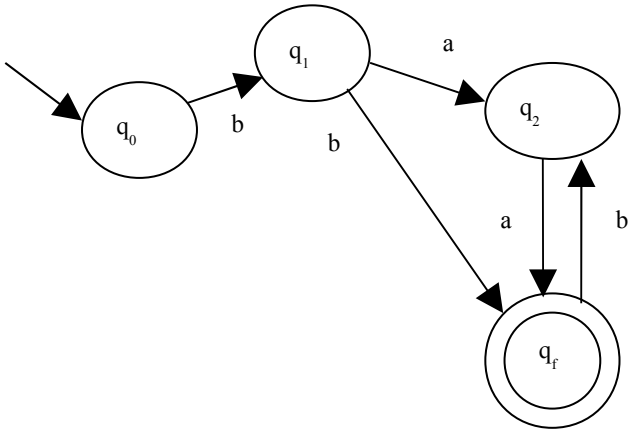


Fig. n. 9.1 - AUTOMA RIDOTTO

In tal caso, se da un certo stato non è presente la freccetta uscente etichettata con un particolare carattere in ingresso (*ad esempio dallo stato* ***q₂*** *non è presente alcuna freccetta etichettata con* ***b***), diremo che **il comportamento dell'automa non è definito** per quella particolare coppia stato-carattere.

9.5 Il programma come funzione parziale

Abbiamo visto che il programma eseguito da un automa a stati finiti può essere considerato una funzione:

$$f: Q \times S \rightarrow Q$$

dal prodotto cartesiano dell'insieme Q degli stati per l'insieme S dell'alfabeto di ingresso all'insieme Q degli stati. Se abbiamo m stati, ed un alfabeto composto da n simboli, **dobbiamo considerare** tutte le m x n possibili coppie.

Tuttavia, come abbiamo visto nelle precedenti pagine, per semplificare la progettazione dell'automa e rendere la sua rappresentazione grafica più chiara, conviene omettere l'eventuale stato pozzo in esso presente. In tal caso, alle coppie (q, s) che portano ad uno stato pozzo, non associamo assolutamente nulla.

Abbiamo ora un problema formale: in matematica, una funzione definita su un insieme **deve prevedere** un valore per ogni elemento dell'insieme! Che senso ha parlare di insieme di definizione di una funzione, se poi la funzione non è definita in un certo punto?

Per risolvere questo problema, di natura formale, si introducono le funzioni parziali:

Definizione. Una **funzione parziale**, definita su un insieme A, ed a valori in un insieme B, è una regola di corrispondenza che:

- ad ogni elemento di un certo sottoinsieme di A associa un elemento di B (*e quindi è una funzione, nell'accezione classica del termine, definita su un sottoinsieme di A*);
- ai restanti elementi di A, la associa un particolare valore, che indichiamo con il simbolo "*non definito*". ■

Ci rendiamo conto immediatamente che la nozione di funzione parziale non è altro che un trucchetto utile per semplificarci la vita. Se ci riferiamo alla rappresentazione matriciale dell'automa ridotto, illustrato in precedenza, per indicare che per una particolare coppia stato-simbolo non è definito il comportamento, lasceremo in bianco la casellina corrispondente.

9.6 Configurazione dell'automa

Per configurazione dell'automa a stati finiti ad un certo istante si intende la coppia (q, δ) così definita:

q è lo stato attuale in cui si trova l'automa,

δ rappresenta la porzione di stringa di ingresso che rimane da esaminare (sul cui primo carattere si trova la testina di lettura).

Chiaramente una configurazione dell'automa non è altro che una fotografia che scattiamo all'automa ad un certo istante.

Si noti che, essendo possibile sul nastro di ingresso la sola operazione di lettura, anziché specificare la porzione δ di stringa di ingresso che rimane da esaminare, è sufficiente indicare la posizione della testina all'interno del nastro.

9.6.1 Configurazione iniziale dell'automa

L'automa si trova inizialmente nello stato q_o.

La stringa *x* da riconoscere è contenuta in celle contigue del nastro di ingresso, ed è codificata utilizzando i simboli dell'alfabeto esterno *S*. Il nastro non contiene altre celle.

La testina di lettura è posizionata in corrispondenza del simbolo che si trova più a sinistra.

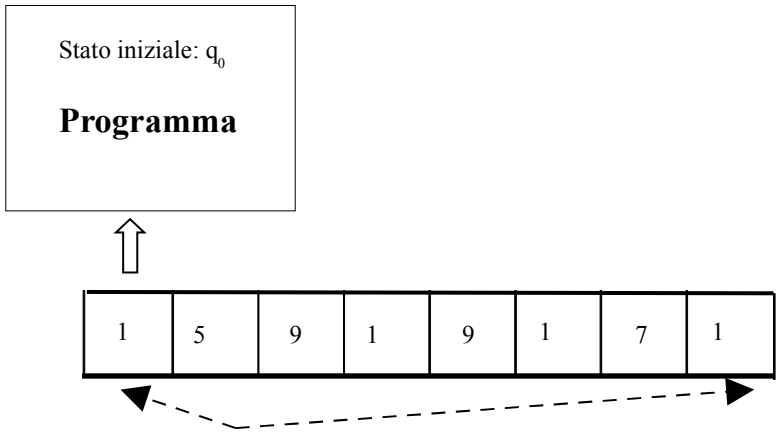


Fig. n. 9.2 – CONFIGURAZIONE INIZIALE DELL'AUTOMA.

9.6.2 Transizione dell'Automa a Stati Finiti

Per **transizione** (o *passo*) di un automa a stati finiti si intende la successione delle seguenti due azioni:

considerando il carattere presente sotto la testina di lettura e lo stato in cui ci l'automa si trova, l'automa si porta nello stato specificato dal programma;
la testina di lettura viene fatta avanzare a destra di una posizione.

Nota bene. In seguito ad una transizione lo stato potrebbe rimanere inalterato.

9.6.3 Terminazione della computazione

L'automa, partendo dalla configurazione iniziale, effettua una o più transizioni, fin quando si verifica una delle seguenti condizioni:

- *l'automa si porta in una configurazione in cui nessuna mossa è definita;*
- *non ci sono altri caratteri da leggere sul nastro di ingresso.*

Si noti che, nel primo caso, ciò equivale al fatto che alla coppia (*q, s*) corrente, costituita dallo stato attuale e dal carattere presente sotto la testina di lettura, non è associato alcuno stato.

9.7 Riconoscimento di linguaggi

Diremo che una stringa *x* è accettata dall'automa a stati finiti se:

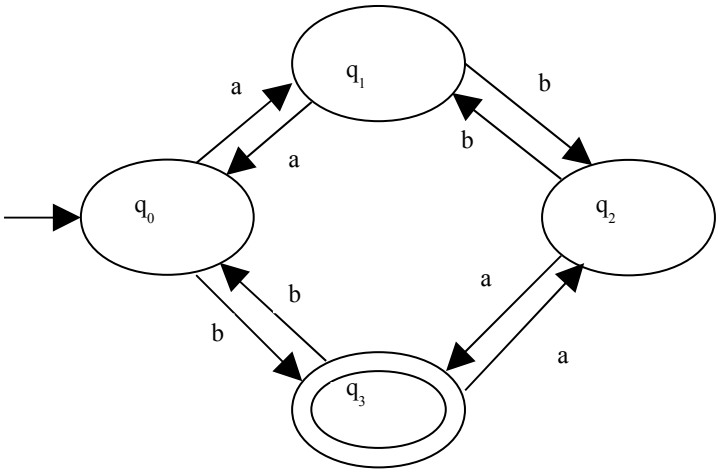
- partendo dalla configurazione iniziale, l'automa giunge dopo un certo numero di mosse ad una configurazione finale, *ed inoltre*
- non ci sono altri caratteri da esaminare sul nastro di ingresso.

Esempio di Automa a Stati Finiti.

Vogliamo riconoscere il linguaggio costituito da tutte le stringhe di a e di b, in cui compaiono un numero pari di a ed un numero dispari di b. A tal fine occorrono solo quattro stati, con il seguente significato associato:

- q_0 *indica che sono stati incontrati (letti) finora un numero* ***pari*** *di a ed un numero* ***pari*** *di b;*
- q_3 *indica che sono stati letti finora un numero* ***dispari*** *di a ed un numero* ***pari*** *di b;*
- q_1 *indica che sono stati letti finora un numero* ***pari*** *di a ed un numero* ***dispari*** *di b;*
- q_2 *indica che sono stati letti finora un numero* ***dispari*** *di a ed un numero* ***dispari*** *di b*

convenendo, ovviamente, che il numero 0 sia pari.



Si noti che, modificando lo stato finale, si riconosce un differente linguaggio. Ad esempio, se l'unico stato finale è q_2 *il linguaggio riconosciuto è costituito dalle stringhe aventi un numero dispari di a ed un numero dispari di b.*

Se gli stati finali sono q_2 *e* q_3 *allora il linguaggio è l'unione dei precedenti due, ovvero è costituito dalle stringhe aventi un numero dispari di a ed un numero dispari di b e dalle stringhe aventi un numero pari di a ed un numero dispari di b.*

■

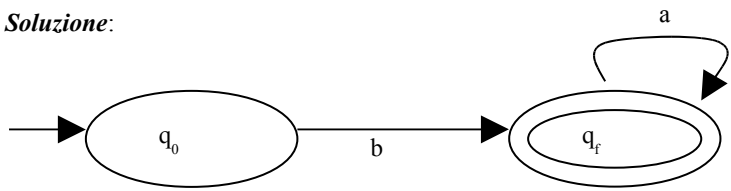
9.8 Linguaggi riconosciuti da un Automa a Stati Finiti

La classe dei linguaggi riconosciuti dagli automi a stati finiti coincide con la classe dei linguaggi regolari, o di tipo 3.

ESEMPIO – COSTRUZIONE DI UN AUTOMA A PARTIRE DA UNA GRAMMATICA DI TIPO 3. *Costruire un automa a stati finiti in grado di riconoscere il linguaggio generato dalla seguente grammatica:*

$S \rightarrow bX$
$X \rightarrow aX$
$X \rightarrow a$

Soluzione:



■

9.9 Risultato fondamentale (senza dimostrazione)

Dato un automa a stati finiti è sempre possibile costruire una grammatica di tipo 3 che genera il linguaggio riconosciuto dall'automa.

E viceversa:

Data una grammatica di tipo 3 è sempre possibile costruire un automa a stati finiti che riconosce il linguaggio generato dalla grammatica.

10

Gli Automi a Pila Deterministici

Esempio 1. *Consideriamo il linguaggio costituito dalle stringhe* $\beta\$\beta^R$, *dove* β *è una qualunque successione di simboli* 0 *e* 1, \$ *indica un altro simbolo dell'alfabeto terminale T, e* β^R *indica la stringa* β *riflessa (cioè capovolta*).

Frasi valide di questo linguaggio sono ad esempio:

\$ 00011\$11000 01\$10 00\$00 111\$111

Tale linguaggio è generato dalla seguente grammatica:

$S \rightarrow \$$
$S \rightarrow 0S0$
$S \rightarrow 1S1$

che è chiaramente di tipo 2.

Per riconoscere se una stringa assegnata, composta da simboli dell'alfabeto terminale T = {0, 1, \$}, sia una frase di questo linguaggio, un automa a stati finiti dovrebbe ricordare la sottostringa che precede il simbolo \$, *e verificare se la sottostringa che segue il simbolo \$ la riproduce specularmente.*

Poiché il carattere \$ *può essere preceduto da una qualunque stringa di* 0 *e di* 1, *sarebbe necessario un automa capace dì ricordare un numero illimitato di situazioni diverse, corrispondenti alle infinite stringhe di* 0 *e di* 1. *Un automa a stati finiti non può essere dunque utilizzato, perché esso è in grado di ricordare solo un numero finito di situazioni diverse, tante quanti i suoi stati.*

■

Esempio 2. *Consideriamo il linguaggio formato dalle stringhe costituite da un certo numero di* a *seguito dallo stesso numero di* b.

Frasi valide di questo linguaggio sono, ad esempio:

ab aabb aaabbb aaaabbbb

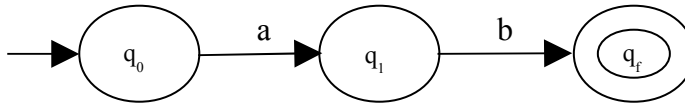
Tale linguaggio è generato dalla seguente grammatica:

$S \rightarrow ab$
$S \rightarrow aSb$

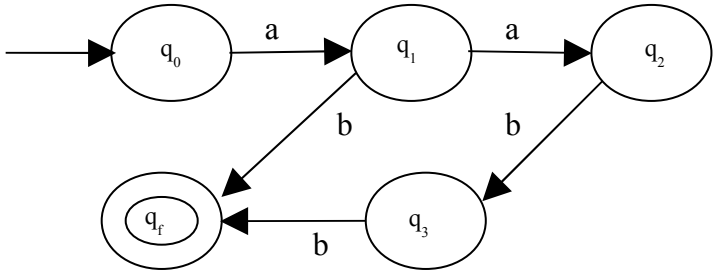
che è chiaramente di tipo 2.

Per riconoscere se una stringa assegnata, composta da simboli dell'alfabeto terminale T = {a, b}, sia una frase di questo linguaggio, un automa a stati finiti (che non sa contare!) dovrebbe disporre di uno stato diverso per ogni possibile numero di a che precede il primo carattere b.

Per riconoscere il linguaggio {ab} occorre un automa costituito da almeno tre stati, ovvero:



Per riconoscere il linguaggio {ab, aabb} abbiamo bisogno di altri due stati:



Procedendo in questo modo (i matematici direbbero: "procedendo per induzione matematica"), osserviamo che, per riconoscere il linguaggio costituito da ***al più*** *n caratteri a seguiti da n caratteri b, occorrono esattamente 2n+1 stati.*

Un automa a stati finiti non può essere dunque utilizzato per rappresentare il nostro linguaggio, perché il numero n di caratteri a *(che precede gli n caratteri* b*) può assumere un qualsiasi valore!* ■

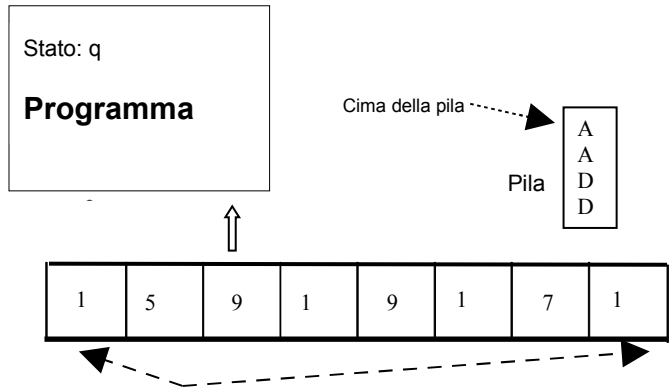
Gli esempi appena fornito mostrano che un automa a stati finiti non è in grado di riconoscere i linguaggi non contestuali. Per riconoscere tali linguaggi abbiamo dunque bisogno di automi più potenti, i cosiddetti automi a pila, che costituiscono l'argomento del presente capitolo.

La maggior potenza degli automi a pila é data dal fatto che essi, a differenza degli automi a stati finiti, sono dotati di memoria, organizzata come una pila (si pensi ad una pila di piatti).

10.1 Definizione di Automa a Pila Deterministico

Un automa a pila deterministico M consiste di:

- un **nastro di ingresso finito**, suddiviso in celle. Ogni cella può contenere un solo simbolo, tratto da un insieme finito *S* detto alfabeto di ingresso;
- una **testina** capace di leggere un simbolo da una cella, e muoversi di una posizione sul nastro di ingresso, da sinistra verso destra;
- un insieme finito *Q* di **stati**, tali che l'automa si trovi esattamente in uno di essi in ciascun istante;
- una **memoria ausiliaria**, organizzata come una **pila**. Ogni celletta della pila può contenere un solo simbolo, tratto da un insieme finito *P*, detto alfabeto di pila;
- un **programma**, che specifica esattamente cosa fare per risolvere un problema specifico.



Stringa *x* da riconoscere in celle contigue.

10.1.1 *L'alfabeto di ingresso*

L'alfabeto di ingresso $S = \{s_1, \ldots, s_k\}$ è utilizzato esclusivamente per codificare l'informazione in input.

10.1.2 L'alfabeto di pila

L'alfabeto di pila $P = \{p_1, \ldots, p_j\}$ è utilizzato per rappresentare i risultati intermedi della computazione nella memoria, che ricordiamolo, é gestita come una pila.

Assumiamo che P contenga sempre un simbolo speciale, indicato con Z. Inizialmente, la pila contiene il solo simbolo Z.

10.1.3 Gli stati dell'Automa a Pila

I possibili stati di un automa a pila sono denotati $q_1, q_2, \ldots, q_n, q_0, q_{f1}, q_{f2}, \ldots, q_{fm}$

- Gli stati $q_1, \ldots q_n$ sono detti **stati ordinari**.
- Lo stato q_0 è detto **stato iniziale**.
- Gli stati $q_{f1}, q_{f2}, \ldots, q_{fm}$ sono detti **stati finali**.

10.1.4 Scrittura di una stringa sulla pila

La scrittura di una stringa di simboli (dell'alfabeto di pila) sulla pila è definita nel modo seguente:

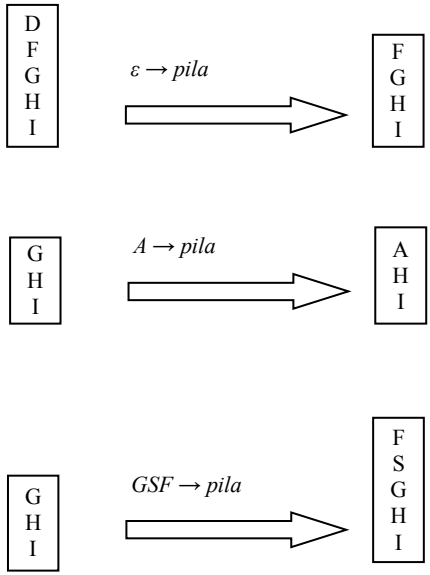
se la stringa è vuota, ovvero non contiene alcun carattere (indichiamo tale stringa con il simbolo ε), allora il carattere presente sulla cima della pila viene rimosso (e quindi la dimensione della pila decresce di una unità);

se la stringa non è vuota, allora:

il primo carattere della stringa da scrivere sostituisce il carattere in cima alla pila;

se sono presenti altri caratteri nella stringa da scrivere, essi vengono inseriti man man nella pila, uno sull'altro.

Esempio - Scrittura di Stringhe sulla Pila. *Indichiamo la scrittura di una stringa x sulla pila con la notazione $x \rightarrow$ pila. I seguenti disegni mostrano il contenuto della pila, rispettivamente prima e dopo l'operazione di scrittura specificata.*


D
F
G
H
I
ε → pila
F
G
H
I
G
H
I
A → pila
A
H
I
G
H
I
GSF → pila
F
S
G
H
I

■

10.1.5 Il programma dell'Automa a Pila

Se indichiamo con *q* lo stato in cui l'automa si trova ad un certo istante, con *s* il simbolo che si trova sul nastro di ingresso in corrispondenza della testina di lettura, e con *p* il simbolo presente sulla cima della pila, il programma dovrà specificare, per ogni possibile terna *(q, s, p)*:

- in quale nuovo stato l'automa dovrà portarsi;
- la stringa di simboli, appartenenti all'alfabeto di pila da scrivere sulla pila.

In aggiunta, é possibile specificare il comportamento dell'automa quando ci si trovi alla presenza di una particolare coppia *(q, p)*, ovvero di una particolare coppia (stato, carattere sulla cima della pila), ***senza leggere alcun carattere presente sul nastro di ingresso***. In altre parole, se l'automa si trova in un certo stato ed al tempo stesso é presente un particolare carattere sulla pila, possiamo immediatamente specificare il nuovo stato in cui portarsi e la stringa da scrivere sulla pila, senza considerare il carattere presente sotto la testina di lettura.

10.1.6 Il programma come funzione

Ricordiamo che P^* indica la chiusura riflessiva e transitiva dell'insieme P, detta anche chiusura di Kleene di P. Possiamo vedere il programma eseguito da un automa a pila deterministico come una funzione:

$$f: Q \times S \times P \rightarrow Q \times P^*$$

Ma, attenzione! Abbiamo detto che in alcune situazioni particolari é possibile specificare il comportamento dell'automa senza leggere alcun carattere presente sul nastro di ingresso; ciò equivale a non leggere niente, ovvero a leggere la stringa vuota ε. Quindi, per essere più precisi, occorre dire che il programma eseguito da un automa a pila deterministico é una funzione:

$$f: Q \times (S \cup \varepsilon) \times P \rightarrow Q \times P^*$$

dove, con $S \cup \varepsilon$ si intende l'unione insiemistica dell'alfabeto S e della stringa vuota.

10.2 Configurazione dell'Automa a Pila

Per configurazione dell'automa a pila ad un certo istante si intende una terna (q, δ, ζ) così definita:

q è lo stato attuale in cui si trova l'automa;

δ rappresenta la porzione di stringa di ingresso che rimane da esaminare (sul cui primo carattere si trova la testina di lettura);

ζ rappresenta il contenuto della pila.

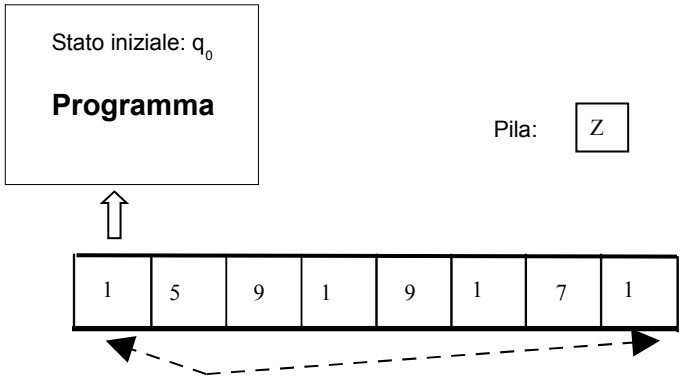
Chiaramente una configurazione dell'automa a pila non è altro che una fotografia che scattiamo all'automa ad un certo istante.

10.2.1 Configurazione iniziale dell'Automa a Pila

L'automa a pila si trova inizialmente nello stato q_o.

La stringa x da riconoscere è contenuta in celle contigue del nastro di ingresso, ed è codificata utilizzando i simboli dell'alfabeto esterno S. Il nastro non contiene altre celle.

La testina di lettura è posizionata in corrispondenza del simbolo che si trova più a sinistra.



10.2.2 Transizione dell'Automa a Pila

Per **transizione** (o *passo*) di un automa a pila, si intende la successione delle seguenti due azioni:

considerando il carattere presente sotto la testina di lettura, lo stato in cui ci l'automa si trova, ed il simbolo posto sulla cima della pila, sono effettuate le azioni corrispondenti specificate nel programma;
la testina di lettura viene fatta avanzare a destra di una posizione,

Nota bene. In seguito ad una transizione:

- lo stato potrebbe rimanere inalterato;
- il carattere scritto sulla cima della pila potrebbe essere lo stesso presente, nel qual caso la pila rimarrebbe inalterata;
- in ogni caso, la testina si sposta a destra di una posizione.

10.2.3 Terminazione della computazione

L'automa, partendo dalla configurazione iniziale, effettua una o più transizioni, fin quando si verifica uno delle seguenti condizioni:

- *l'automa raggiunge uno stato finale;*
- *l'automa si porta in una configurazione in cui nessuna mossa è definita.*

10.3 Riconoscimento di linguaggi

Diremo che una stringa x è accettata dall'automa se, partendo dalla **configurazione iniziale**, l'automa giunge dopo un certo numero di mosse in uno **stato finale**, e non ci sono altri caratteri da esaminare sul nastro di ingresso.

Per semplificare la trattazione, in tutti gli esempi presentati in questa lezione supporremo di avere un solo stato finale, denotato q_f

Definizione alternativa. Alcuni autori utilizzano la seguente definizione alternativa di accettazione:

> *Una stringa x è accettata dall'automa se, partendo dalla configurazione iniziale, l'automa giunge dopo un certo numero di mosse ad una configurazione in cui non ci sono altri caratteri da esaminare sul nastro di ingresso, e la pila è vuota.*

Tale definizione è, fortunatamente, del tutto equivalente alla prima: si dimostra infatti che, i linguaggi riconosciuti da un automa a pila nella condizione di stato finale coincidono con i linguaggi riconosciuti nella condizione di pila vuota.

Si noti che, nel caso in cui sia utilizzata la condizione di pila vuota per riconoscere le frasi di un linguaggio, lo stato finale potrebbe non essere presente. Si studi a tal riguardo l'esercizio svolto riportato in fondo al capitolo.

10.3.1 Tempo di esecuzione

Definiamo il tempo di esecuzione come il numero di transizioni effettuate, ovvero il numero di volte in cui dobbiamo andare a controllare sulla tabellina (la matrice funzionale) cosa fare.

É dunque ovvio che il tempo di esecuzione è pari al più alla lunghezza della stringa da esaminare.

10.3.2 Occupazione istantanea di memoria

Durante l'esecuzione di un programma, l'automa a pila deterministico utilizzerà un certo numero di cellette della pila. L'occupazione di memoria ad un certo istante è definita come la dimensione corrente della pila.

Esempio - L'Automa a Pila Deterministico che riconosce il linguaggio $b\$b^R$

- *L'alfabeto esterno è costituito dai simboli* 0, 1 e $.
- *Le stringhe* β *sono costituite da* 0 *e da* 1.
- *L'alfabeto di pila è costituito dai simboli* 0, 1 *e* Z.
- *Lo stato iniziale è* q_0.
- *L'unico stato ordinario è* q_1.
- *Lo stato finale è* q_f.

La seguente tabellina descrive il programma che l'automa esegue:

Carattere sulla pila	**Stato**	**Carattere in ingresso: 0**	**Carattere in ingresso: 1**	**Carattere in ingresso: $**
Z	q_0	Z0 → pila Resta in q_0	Z1 → pila Resta in q_0	Z → pila Va in q_1
Z	q_1	Va nello stato finale q_f senza leggere alcun carattere	Va nello stato finale q_f senza leggere alcun carattere	Va nello stato finale q_f senza leggere alcun carattere
0	q_0	00 → pila Resta in q_0	01 → pila Resta in q_0	0 → pila Va in q_1
0	q_1	ε→ pila Resta in q_0	non definito	non definito
1	q_0	10 → pila Resta in q_0	11 → pila Resta in q_0	1 → pila Va in q_1
1	q_1	non definito	ε→ pila Resta in q_1	non definito

■

Per verificare la comprensione dell'algoritmo, si consiglia di svolgere il seguente esercizio prima di andare avanti.

ESERCIZIO.
Simulate il comportamento dell'automa precedente fornendo ad esso le seguenti stringhe in input:

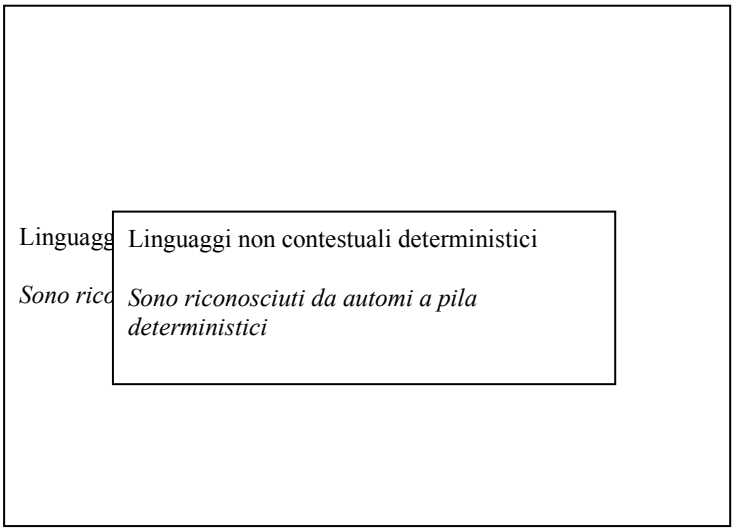
0$0
10$01
10$11
In che stato l'automa si arresta? Cosa resta sulla pila al termine dell'esecuzione del programma? Commentare.
10$011
In che stato l'automa si arresta? Cosa resta sulla pila al termine dell'esecuzione del programma? Commentare.
10$0
In che stato l'automa si arresta? Cosa resta sulla pila al termine dell'esecuzione del programma? Commentare.

Notate che negli esempi *c, d* ed *e* quando l'esecuzione termina - senza riconoscere la stringa - il contenuto della pila, lo stato dell'automa e la posizione della testina possono essere utilizzati per comprendere perché la stringa assegnata non appartenga al linguaggio.

Ciò è esattamente quanto avviene nei compilatori moderni quando essi rilevano un errore sintattico: non solo essi ci avvertono del fatto, ma ci suggeriscono persino la probabile causa (tipico esempio: dove dovrebbe trovarsi una parentesi mancante!).

10.4 Linguaggi riconosciuti da un Automa a Pila Deterministico

La classe dei linguaggi riconosciuti dagli automi a pila deterministici costituisce una sottoclasse propria dei linguaggi non contestuali, chiamata classe dei linguaggi non contestuali deterministici.



Per poter riconoscere l'intera classe dei linguaggi non contestuali occorrono degli automi più potenti di quelli descritti sinora, chiamati automi a pila non deterministici, la cui trattazione esula dagli scopi introduttivi di questo corso.

Esempio – Un automa per il linguaggio $\beta\beta^R$
Consideriamo il linguaggio costituito dalle stringhe $\beta\beta^R$, dove β è una qualunque successione di simboli dell'alfabeto terminale T = {0, 1}, e β^R indica la stringa β riflessa (cioè capovolta).
Frasi valide di questo linguaggio sono ad esempio:

0001111000 **0110** **0000** **111111**

■

Tale linguaggio non può essere riconosciuto da alcun automa a pila deterministico.

Infatti, tornando all'esempio precedente costituito dal linguaggio $\beta\$\beta^R$, intuitivamente potremmo dire che il carattere **$** è utilizzato dall'automa a pila per stabilire quando iniziare ad effettuare il confronto sulla seconda metà della stringa, in altre parole per stabilire quando iniziare a svuotare la pila. Nel caso del linguaggio $\beta\beta^R$, l'automa a pila deterministico non sa quando iniziare ad effettuare il confronto, ovvero quando iniziare a svuotare la pila, poiché il carattere che funge da separatore è assente!

Il riconoscimento del linguaggio $\beta\beta^R$ è comunque possibile utilizzando un automa a pila non deterministico. Infatti, tale linguaggio è generato dalla seguente grammatica:

$S \rightarrow 00$
$S \rightarrow 11$
$S \rightarrow 0S0$
$S \rightarrow 1S1$

che è chiaramente di tipo 2.

ESERCIZIO SVOLTO.

Si vuole costruire un automa a pila in grado di riconoscere il linguaggio delle parentesi ben disposte.

Immaginate di avere un'espressione aritmetica, composta da numeri, parentesi tonde e dai quattro operatori aritmetici (i simboli +, -, /, x). Togliete da essa tutti gli operatori e tutti gli operandi – ciò che resta è una sequenza di parentesi tonde aperte e chiuse, ben disposte. Il linguaggio delle parentesi ben disposte è costituito da tutte queste sequenze.

Ad esempio:

(())

(()())

()()(())

((((()))))

sono frasi valide di questo linguaggio.

Si noti che tale linguaggio è generato dalla seguente grammatica, di tipo 2:

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

(verificate questa affermazione costruendo gli alberi di derivazione delle frasi riportate sopra).

L'automa che ci accingiamo a costruire riconosce il linguaggio delle parentesi ben disposte **nella condizione di pila vuota**. In altre parole, una frase è riconosciuta se entrambe le condizioni che seguono sono soddisfatte:

- non ci sono altri caratteri da leggere sul nastro di ingresso;
- la pila non contiene alcun simbolo.

L'automa a pila è specificato qui di seguito:

- l'alfabeto esterno è costituito dai simboli (e) ;
- l'alfabeto di pila è costituito dai simboli £ e Z;
- lo stato iniziale è q_0, che è anche l'unico stato ordinario;
- non ci sono stati finali;
- il comportamento dell'automa è descritto dalla tabellina che segue:

Carattere sulla pila	**Stato**	**Carattere in ingresso: (**	**Carattere in ingresso:)**
Z	q_0	£ → pila Resta in q_0	non definito
£	q_0	£ £ → pila Resta in q_0	ε → pila Resta in q_0

Simulate il funzionamento dell'automa utilizzando le frasi del linguaggio delle parentesi ben disposte riportate precedentemente.

Appendice A

Breve biografia di Alan Turing

1912	Nasce a Paddington, nella contea di Londra, il 23 giugno.
1926-31	Studia presso la Sherborne School.
1930	La morte del suo amico Christopher Morcom lo getta in uno stato di profonda depressione. Si fa strada in Turing l'idea che lo spirito sopravviva al corpo.
1931-34	Frequenta l'università presso il King's College, a Cambridge.
1932-35	Si occupa di meccanica quantistica, probabilità e logica.
1935	É eletto "*fellow of King's College*".
1936	Concepisce la Macchina di Turing e la macchina universale, formalizza la nozione di computabilità.
1936-38	Consegue il Ph.D. in logica, algebra e teoria dei numeri presso la Princeton University.
1938-39	Ritorna a Cambridge. Viene a conoscenza della macchina di cifratura Enigma, utilizzata dai tedeschi durante la guerra.
1939.40	Concepisce una macchina per decodificare il codice Enigma.
1939-42	La decodifica del codice Enigma permette di vincere la battaglia dell'Atlantico.
1943-45	É nominato consulente capo del gruppo anglo-americano di crittografia.
1945	Lavora presso il National Physical Laboratory, a Londra.
1946	I suoi studi nel campo dei sistemi hardware e software producono risultati di rilevanza mondiale.
1947-48	Si occupa di programmazione, reti neurali ed intelligenza artificiale.
1948	Lavora presso la Manchester University.
1949	Per la prima volta un computer viene utilizzato per risolvere problemi matematici non banali.
1950	Concepisce il Test di Turing per definire la nozione di intelligenza di una macchina.
1951	Si occupa di modelli non lineari per studiare la crescita di una popolazione.
1952	É arrestato per omosessualità. Perde il vaglio (del controspionaggio) di non rappresentare un rischio per la sicurezza dello Stato.
1953-54	Scrive lavori (incompiuti) in biologia e fisica.
1954	Muore suicida ingerendo cianuro, a Wilmslow, nel Cheshire, il 7 giugno.

Appendice B

Il Premio Turing

Il riconoscimento più ambito nel campo dell'informatica è il Premio Turing, conferito ogni anno dall'ACM (Association of Computing Machinery), in onore di A.M. Turing, ad individui che si sono distinti per il loro contributo di natura tecnica allo sviluppo dell'informatica.

Il contributo si richiede che sia duraturo nel tempo e di estrema rilevanza scientifica.

Il riconoscimento è accompagnato da un premio di 100.000 dollari, offerti dalla Intel Corporation.

I vincitori nel 2003 sono stati Ronald L. Rivest, Adi Shamir e Leonard M. Adleman.

Negli anni passati i vincitori sono stati i seguenti:

1966 A.J. Perlis
1967 Maurice V. Wilkes
1968 Richard Hamming
1969 Marvin Minsky
1970 J.H. Wilkinson
1971 John McCarthy
1972 E.W. Dijkstra
1973 Charles W. Bachman
1974 Donald E. Knuth
1975 Allen Newell
1975 Herbert A. Simon
1976 Michael O. Rabin
1976 Dana S. Scott
1977 John Backus
1978 Robert W. Floyd
1979 Kenneth E. Iverson
1980 C. Antony R. Hoare
1981 Edgar F. Codd
1982 Stephen A. Cook
1983 Ken Thompson
Dennis M. Ritchie
1984 Niklaus Wirth
1985 Richard M. Karp
1986 John Hopcroft
1986 Robert Tarjan
1987 John Cocke
1988 Ivan Sutherland
1989 William (Velvel) Kahan
1990 Fernando J. Corbato'
1991 Robin Milner
1992 Butler W. Lampson
1993 Juris Hartmanis
1993 Richard E. Stearns
1994 Edward Feigenbaum
1994 Raj Reddy
1995 Manuel Blum
1996 Amir Pnueli
1997 Douglas Engelbart
1998 James Gray
1999 Frederick P. Brooks, Jr.
2000 Andrew Chi-Chih Yao
2001 Ole-Johan Dahl
2001 Kristen Nygaard

Bibliografia

[1] *A.V.Aho, J.E.Hopcroft, J.D.Ullman,* **Data structures and algorithms**, Addison-Wesley, 1983.

[2] *A.V.Aho, J.E.Hopcroft, J.D.Ullman,* **The Design and Analysis of Computer Algorithms**, Addison-Wesley, 1974.

[3] *G.Bateson,* **Mind and Nature – A Necessary Unity**, Bantam Books, 1980.

[4] *S.Ceccato,* **Cibernetica per tutti**, Feltrinelli, 1968-1974.

[5] *T.H.Cormen, C.E.Leiserson, R.L.Rivest,* **Introduzione agli algoritmi**, Vol. 1, Jackson Libri, 1995.

[6] *T.H.Cormen, C.E.Leiserson, R.L.Rivest,* **Introduzione agli algoritmi**, Vol. 2, Jackson Libri, 1995.

[7] *J.E.Hopcroft, J.D.Ullmann,* **Formal Languages and Their Relation to Automata**, Addison-Wesley, 1969.

[8] *B.Kernigham, D.Ritchie,* **The C Programming Language**,

[9] *D.E.Knuth,* **Fundamental Algorithms**, vol. 1, **The Art of Computer Programming**, Addison-Wesley, 1968. Second edition, 1973.

[10] *D.E.Knuth,* **Seminumerical Algorithms**, vol. 2, **The Art of Computer Programming**, Addison-Wesley, 1968. Second edition, 1973.

[11] *C.E.Shannon,* **A Mathematical Theory of Communication**, Bell System Technical Journal, 1948.

[12] *A.Turing,* **Computing Machinery and Intelligence**, vol. 59, n. 236, Mind, 1950, pp. 433-460.