Corso di Laurea Magistrale in Ingegneria Informatica
A.A. 2011-2012

Linguaggi Formali e Compilatori

Automi e Macchine di Turing

Giacomo PISCITELLI

Automi

Generalità sugli Automi

In informatica teorica e in matematica discreta, un **automa** è un dispositivo di calcolo ideale, o un suo modello in forma di macchina sequenziale, creato per eseguire un particolare compito, di solito costituito da operazioni elementari semplici.

Il dispositivo automa può trovarsi in diverse **configurazioni** più o meno complesse caratterizzate primariamente da una variabile che appartiene ad un determinato insieme di **stati**, e che evolve in base agli stimoli od ordini ricevuti in ingresso schematizzati da simboli appartenenti ad un determinato alfabeto.

Quando l'automa si trova in un dato stato, esso può accettare solo un sottoinsieme dei simboli del suo alfabeto. L'evoluzione di un automa parte da un particolare stato detto **stato iniziale**. Un sottoinsieme privilegiato dei suoi stati è detto insieme degli **stati finali** o *marcati*.

Automi

In teoria dei sistemi un automa si definisce anche come un sistema dinamico (evolve nel tempo), discreto (nella scansione del tempo e nella descrizione del suo stato) e invariante (il sistema si comporta alla stessa maniera indipendentemente dall'istante di tempo in cui agisce).

Nel corso della storia della teoria dei linguaggi e degli automi, gli automi sono andati arricchendosi, come testimoniano la macchina di calcolo di Turing e gli automi di Chomsky per le varie costruzioni del linguaggio naturale.

In realtà molto pochi degli automi proposti sono impiegati nell'elaborazione dei linguaggi di programmazione. E nel seguito ci si limiterà a tali automi.

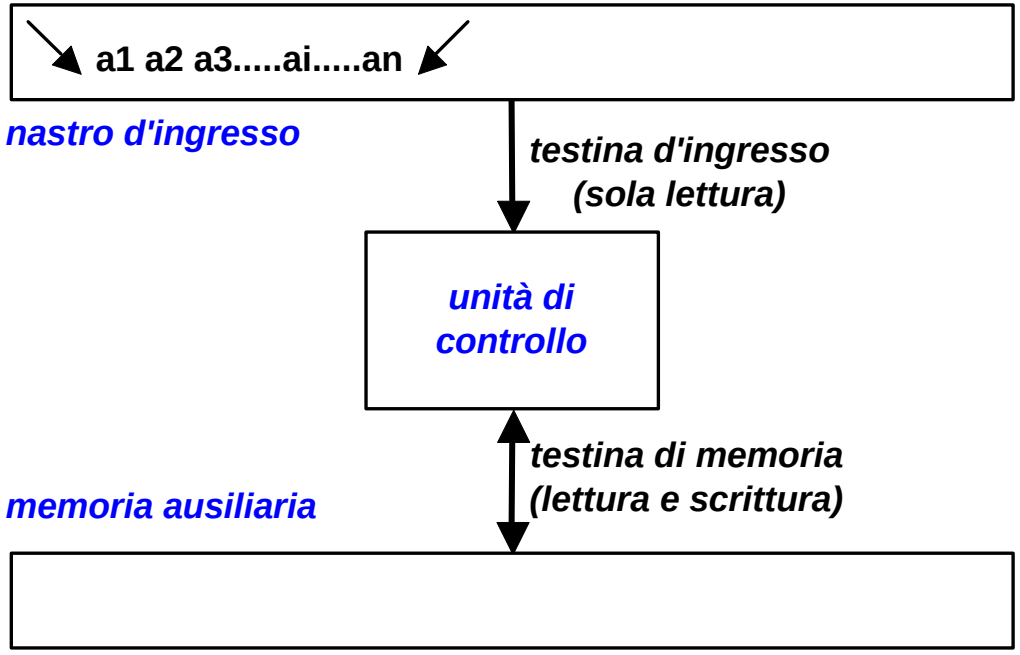
Automi

Automi e linguaggi

Gli **automi** sono spesso utilizzati per descrivere linguaggi formali, e per questo sono chiamati **accettori o riconoscitori di un linguaggio**.

Nella sua forma più generale un automa riconoscitore è costituito da 3 parti:

- il ***nastro d'ingresso***
- l'***unità di controllo***
- la ***memoria ausiliaria***.



Automi

Il **nastro d'ingresso** è suddiviso in caselle, ciascuna contenente un simbolo (o stimolo o carattere) dell'alfabeto d'ingresso Σ. A volte s'impone che la stringa d'ingresso sia delimitata a sinistra e a destra da due simboli riservati alla delimitazione, detti marca d'inizio (↘) e marca di fine (↙).

L'automa opera in istanti discreti: esso legge il simbolo sottostante la testina (o cursore) d'ingresso, che può muoversi, rispetto al nastro, di una posizione verso sinistra o destra.

L'**unità di controllo** è dotata, al suo interno, di una quantità finita di memoria (si dice che ha un *numero finito di stati*, nel qual caso l'automa si dice a stati finiti), mentre ……

… la **memoria ausiliaria** ha, in generale, una capacità illimitata; anche essa può essere immaginata come un nastro, contenente simboli di un *alfabeto di memoria*. A differenza del nastro d'ingresso, la memoria ausiliaria può essere letta e scritta e quindi può funzionare come deposito (o contenitore) di risultati intermedi determinati dal programma dell'unità di controllo.

La posizione di memoria a cui si accede è quella posta sotto la testina di memoria, che può scorrere avanti o indietro a secondo del controllo.

Automi: il funzionamento

Un automa esamina la stringa sul nastro d'ingresso compiendo una serie di *mosse*; *ogni mossa dipende dai simboli presenti sotto le testine e dallo stato dell'unità di controllo*.

Ogni mossa consiste in:

- lo spostamento della testina d'ingresso di una posizione (a destra o sinistra);
- la scrittura di un simbolo nella memoria;
- lo spostamento della testina di memoria di una posizione (a destra o sinistra);
- il cambiamento di stato dell'unità di controllo.

Alcune di queste azioni possono mancare.

Se la testina d'ingresso si può muovere soltanto verso destra, *il nastro è detto monodirezionale*, che è il caso che prenderemo in esame.

Lo spostamento della testina di memoria può essere automatico, se la memoria ausiliaria è organizzata secondo una particolare struttura, lo **stack** (o *pila*), che esamineremo in seguito.

Automi: la configurazione

Una configurazione dell'automa è definita da:

- la porzione di nastro d'ingresso alla destra della testina d'ingresso,
- lo stato dell'unità di controllo;
- la posizione della testina di memoria ed il contenuto di quest'ultima.

L'automa, all'origine, viene posto nella *configurazione iniziale*:

- la testina d'ingresso è sul simbolo che segue la marca d'inizio (↘);
- l'unità di controllo è in un particolare stato detto stato iniziale;
- la memoria contiene l'informazione iniziale.

Attraverso una serie di mosse (computazione) la configurazione dell'automa evolve.

Automi: la configurazione

Una **configurazione** è **finale** se l'unità di controllo è in uno stato finale e la testina d'ingresso è posta sulla marca di fine (↙).

A volte, in alternativa, la configurazione è finale se la memoria si trova in una particolare condizione (ad es. è vuota).

Una stringa (o parola) **x è accettata dall'automa** se esso, partendo dalla configurazione iniziale con ↘ **x** ↙ sul nastro d'ingresso, può eseguire almeno una sequenza di mosse che lo porta in una configurazione finale.

L'insieme delle stringhe accettate dall'automa costituisce il **linguaggio accettato** (o *riconosciuto* o *definito*) da esso.

L'automa si ferma o perché ha raggiunto una configurazione finale o perché la mossa non è definita, nel qual caso la stringa non è accettata.

Allo schema prima illustrato di automa appartiene la macchina di Turing, che può assumere diverse varianti e che è accettata come formalizzazione di ogni procedura di calcolo.

Automi e indeterminismo

Il cambiamento che si produce con le mosse, e quindi **l'automa**, è **deterministico** se in ogni configurazione è ammessa non più di una mossa per ciascun simbolo in input, mentre è **non deterministico** se in almeno una configurazione sono possibili più mosse alternative in corrispondenza dello stesso simbolo o anche se l'alfabeto dei simboli prevede il simbolo vuoto **ε**.

Un automa non deterministico è un modo astratto di rappresentare un algoritmo che può *procedere per tentativi, esaminando due o più percorsi alternativi*.

Che tipo di computazione è quella di un automa non deterministico?

Una **computazione parallela**. Infatti, in corrispondenza di un simbolo a cui corrispondono più mosse alternative, la macchina "splitta" in più copie quante sono le alternative e ogni copia, in parallelo con le altre, procede separatamente. Se incontra un'altra alternativa, "splitta" ancora. Se il prossimo simbolo in input non è previsto in nessuna delle transizioni in uscita dallo stato corrente di una copia della macchina, quest'ultima muore assieme a tutta la computazione corrispondente.

Se una delle copie della macchina si trova in uno stato finale alla fine della stringa d'ingresso, l'automa non deterministico accetta la stringa.

In corrispondenza del simbolo vuoto **ε**, l'automa si suddivide in diverse copie, una per ciascun simbolo **ε**, ed una ferma allo stato corrente.

Approccio riconoscitivo dei linguaggi

Nell'**approccio riconoscitivo,** quindi**,** un linguaggio può essere definito mediante macchine astratte (automi) o algoritmi che accettano le stringhe che ne fanno parte e rifiutano quelle che non ne fanno parte.

Un Automa a stati finiti è il riconoscitore per il linguaggi regolari.

Un automa a stati finiti è il tipo più semplice di macchina per riconoscere linguaggi.

Un **automa a stati finiti** può essere **deterministico** (FSA) o **non deterministico** (NFSA).

In genere gli automi sono deterministici, ovvero dato uno stato ed un simbolo in ingresso è possibile una sola transizione.

Automi a stati finiti e *scanner*

Gli Automi a stati finiti sono largamente usati come **Analizzatori lessicali** (scanner).

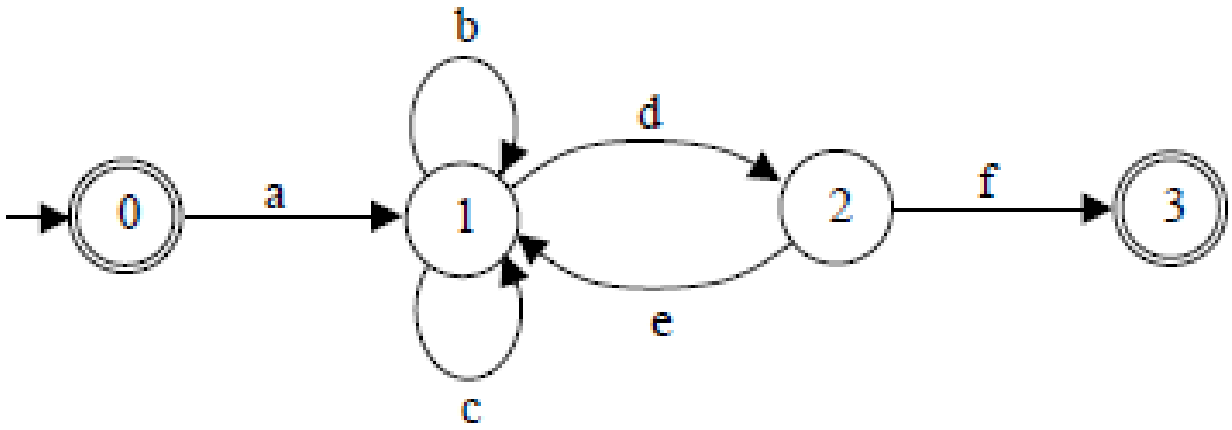
Nei 'compiler of compiler' e nei generatori di analizzatori lessicali (tipo **Lex** di UNIX) si utilizzano tutti i diversi formalismi di definizione dei linguaggi regolari.

Il generatore prende in input l'espressione regolare e:

- costruisce l'automa a stati finiti non deterministico corrispondente
- lo trasforma in automa a stati finiti deterministico
- fornisce come output il programma di analisi lessicale (scanner) per il linguaggio dato.

Esempio di automi a stati finiti e linguaggi

Esprimere la grammatica BNF del linguaggio regolare definito dal seguente automa:



A_0 ::= a A_1 | ε
A_1 ::= b A_1 | c A_1 | d A_2
A_2 ::= e A_1 | f A_3
A_3 ::= ε

Automi a stati finiti e *scanner*

Un **Deterministic Finite State Automata** (**FSA**) **M** = $\langle \Sigma, Q, \delta, q_0, F \rangle$ è, in generale, una quintupla costituita da:

- un alfabeto di input Σ
- un numero finito e non vuoto di stati $Q = \{q_0,\ldots,q_n\}$
- un insieme di transizioni di stato δ da uno stato ad un altro stato etichettato da caratteri di un alfabeto Σ $\delta : Q \times \Sigma \rightarrow Q$
- uno stato iniziale q_0
- un insieme di stati finali **F** $F \subseteq Q$ (stringa valida)

La funzione δ codifica le mosse dell'automa. $\delta(q_i, a) = q_j$ significa che **M**, trovandosi nello stato q_i e leggendo **a**, si porta nello stato q_j.

In particolare si conviene che, per ogni q_i, $\delta(q_i, \varepsilon) = q_i$.

Automi a stati finiti e *scanner*

La funzione di transizione di un FSA può essere rappresentata mediante:

- una matrice (tabella) di transizione
- un diagramma degli stati

Esempio.
Dato il linguaggio

$\{a^n b \mid n \geq 0\}$ generato da **$S \rightarrow aS \mid b$**

l'automa che lo riconosce è l'automa

$\langle \{a,b\}, \{q_0,q_1,q_2\}, \delta, q_0, \{q_1\} \rangle$

δ	a	b
q_0	q_0	q_1
q_1	q_2	q_2
q_2	q_2	q_2

matrice di transizione di stato
dell'automa riconoscitore del linguaggio
$\{a^n b \mid n \geq 0\}$

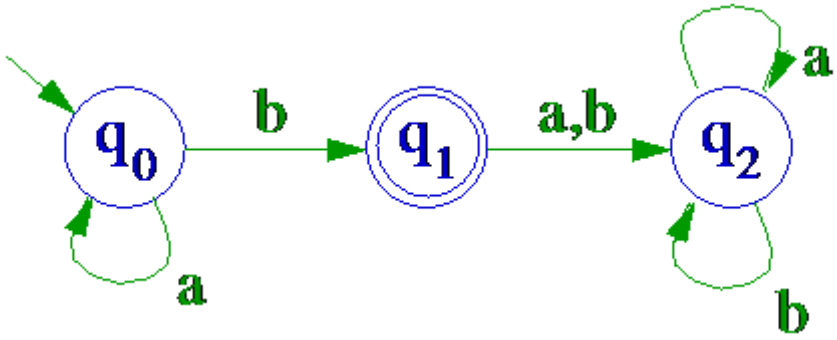
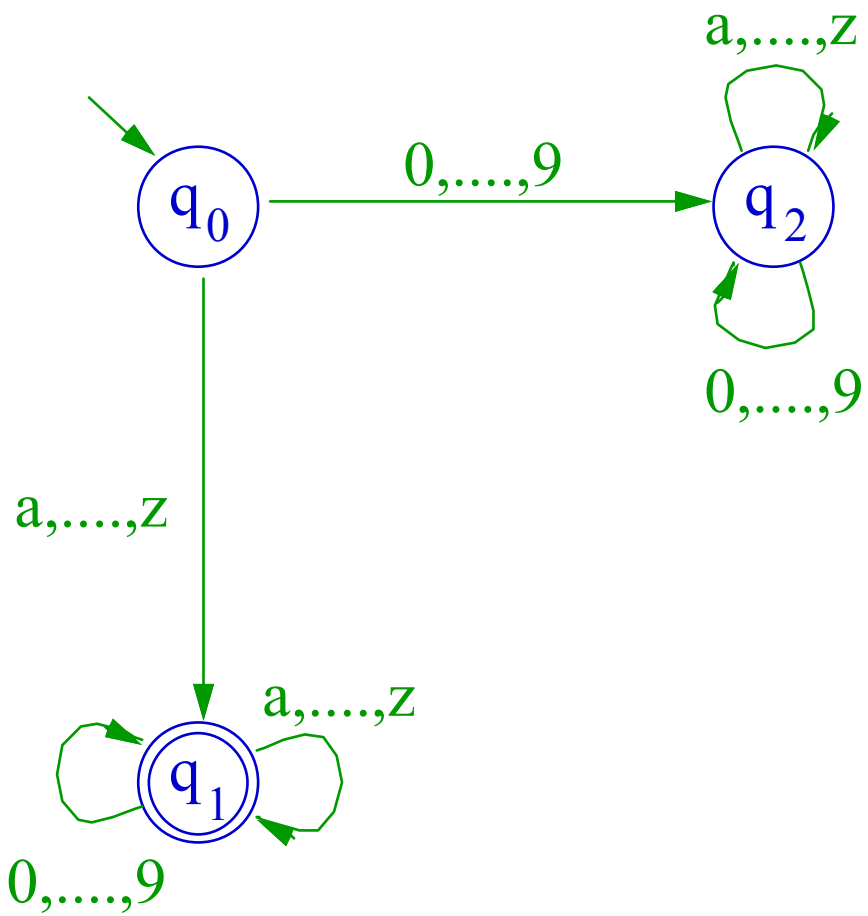


diagramma degli stati dell'automa

Automi a stati finiti e *scanner*

Esempio.

Automa che riconosce gli identificatori.



Automi a stati finiti e *scanner*

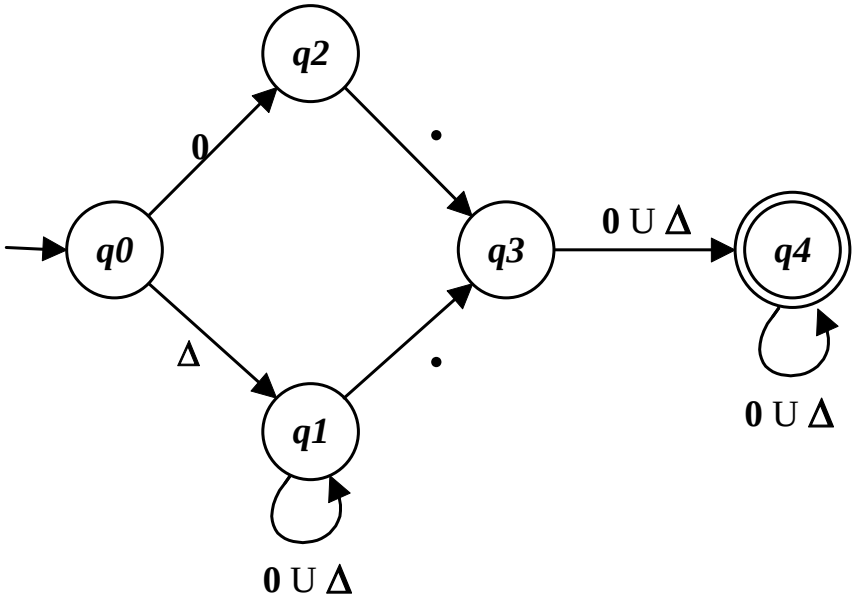
Esempio.

Automa che riconosce le costanti reali.

$\Delta = \{1,2,3,4,5,6,7,8,9\}$

$\Sigma = \Delta \cup \{0,\bullet\}$

$L_1 = (0 \cup \Delta(0 \cup \Delta)^*) \bullet \Delta(0 \cup \Delta)^+$



stato presente	carattere letto				
	0	1	. . . . .	9	•
q0	**q2**	**q1**	. . . . .	**q1**	----
q1	**q1**	**q1**	. . . . .	**q1**	**q3**
q2	----	----	. . . . .	----	**q3**
q3	**q4**	**q4**	. . . . .	**q4**	----
q4	**q4**	**q4**	. . . . .	**q4**	----

Automa minimo

In generale esistono infiniti automi equivalenti (che, cioè, accettano lo stesso linguaggio), differenti per il numero degli stati.

Un risultato teorico utile è che, a meno di un isomorfismo (cioè a meno di una ridenominazione degli stati), è unico il riconoscitore minimo o **automa minimo** (rispetto alla cardinalità |Q| dell'insieme degli stati) di un linguaggio **L**.

Algoritmo di minimizzazione da un automa generico **M** *all'automa minimo* **M'**.

Definite 2 generiche relazioni sull'insieme **Q** degli stati di **M**, si dice che:

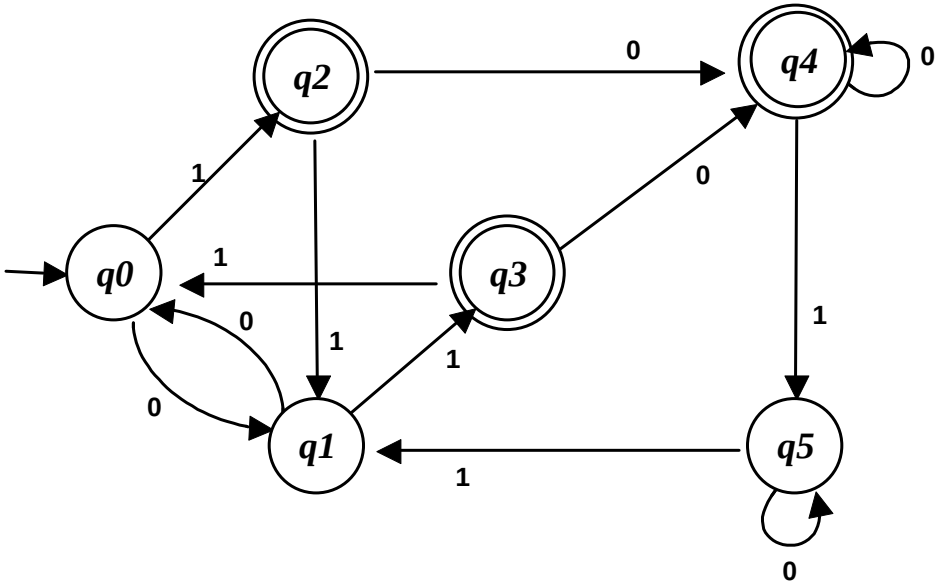
- la relazione p è equivalente a q (e si scrive $p \equiv q$) se e solo se, per ogni stringa x, o entrambi gli stati $\delta(p, x)$ e $\delta(q, x)$ sono finali, o nessuno dei 2 lo è; ovviamente si ha $p \equiv p$.
- la relazione r è distinguibile da q (e si scrive $r \neq q$) se esiste una stringa x per cui $\delta(r, x) \in \mathbf{F}$ e $\delta(q, x) \notin \mathbf{F}$, o viceversa.

Per minimizzare l'automa si determinano le classi di equivalenza[1] della relazione $\equiv$.

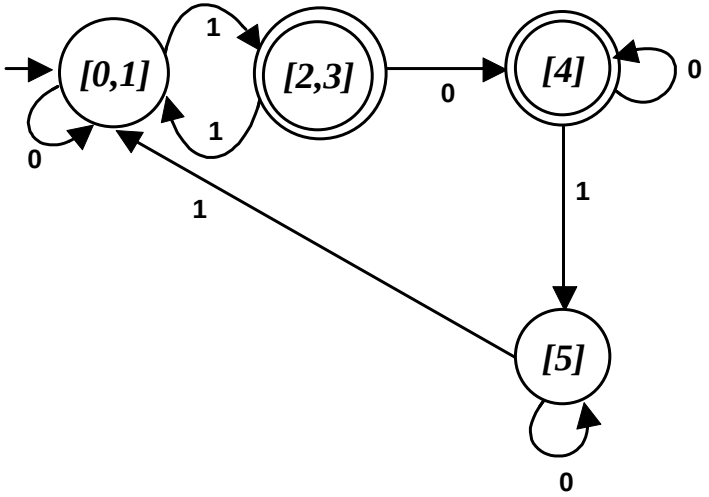
[1] Una classe di equivalenza è l'insieme di tutti gli stati equivalenti fra di loro.

Automa minimo

Esempio



a) automa originale



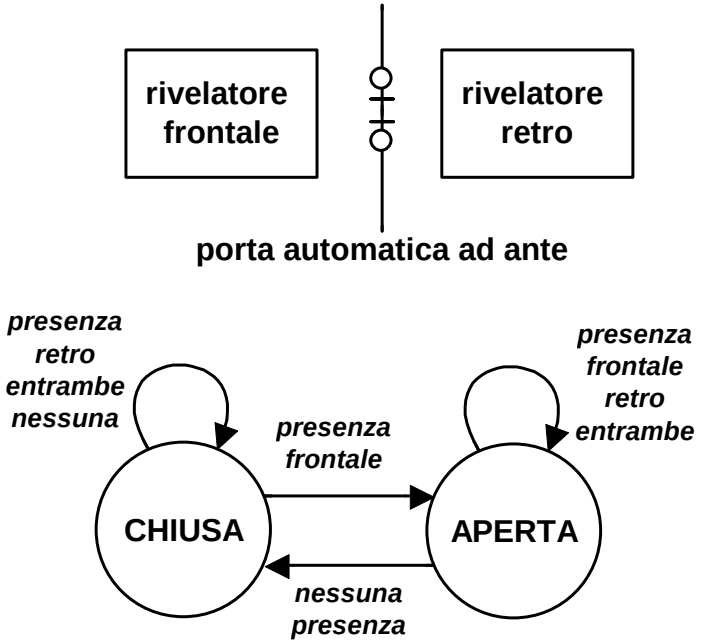
b) automa minimo

Automi come sistemi di transizione

Gli automi consentono in generale di rappresentare sistemi di transizione in un insieme finito di stati:

- un ascensore
- un distributore di biglietti dell'autobus
- il sistema elettrico di un'automobile
- un dispositivo elettronico digitale
- .....

Esempio: Il controllore di una porta automatica a 2 ante



Automi come sistemi di transizione

Esercizio

Realizzare un automa che riconosce tutte le sequenze di monete da 0.10 €, 0.20 €, 0.50 €, 1.00 €, che consentono di raggiungere il costo di un biglietto dell'autobus (1.50 €). Nello stato iniziale la somma disponibile e' 0. La macchina non da' resto.

Automi non deterministici

Un **Non-deterministic Finite State Automata** (NFSA) è un modelllo matematico che consiste di:

S – un insieme finito di stati
Σ - un insieme finito di simboli di input (alfabeto)
mosse – una funzione (di transizione) che, data una coppia stato-simbolo, restituisce uno stato, cioè

$$\text{Mossa} : S \times (\Sigma \cup \varepsilon) \rightarrow S$$

s_0 - uno stato iniziale, $s_0 \in S$
F – un insieme di stati finali $F \in S$

- In un NFSA sono consentite **ε-** mosse. In altre parole, esistono transizioni che non consumano simboli.
- Un NFSA accetta una stringa **x**, se e solo se esiste un cammino dallo stato iniziale ad uno stato finale tale che ogni arco sia etichettato con i simboli in **x**

Automi non deterministici

Eliminazione dell'indeterminismo

Un automa non deterministico è difficile da realizzare mediante un programma, poiché i linguaggi di programmazione non consentono (salvo eccezioni) scelte non deterministiche. Un NFSA è, perciò, una rappresentazione astratta di un algoritmo per riconoscere le stringhe di un certo linguaggio, mentre un FSA è un semplice, concreto algoritmo di riconoscimento di stringhe.

D'altra parte, spesso l'algoritmo di riconoscimento più immediato per talune applicazioni è di tipo non deterministico.

Ne deriva che risulta molto utile la trasformazione di un automa da non deterministico a deterministico. Tale trasformazione, sempre possibile per gli automi finiti, si svolge in 2 fasi:

1) eliminazione degli archi corrispondenti a **ε-** mosse
2) sostituzione delle transizioni non deterministiche con altre deterministiche che portano in nuovi stati

Automi e classi di linguaggi

A diverse classi di automi corrispondono diverse classi di linguaggi, caratterizzate da diversi livelli di complessità.

Due *automi* sono *equivalenti* se hanno la caratteristica di accettare lo stesso linguaggio.

Due automi equivalenti non è detto che siano dello stesso tipo, né che abbiano la stessa complessità. In particolare si dimostra che:

TEOREMA: **Ogni automa finito non deterministico ha un equivalente automa finito deterministico**.

Un **FSA** e l'equivalente **NFSA** riconoscono la stessa classe di linguaggi.

Ciò è sorprendente ed utile.

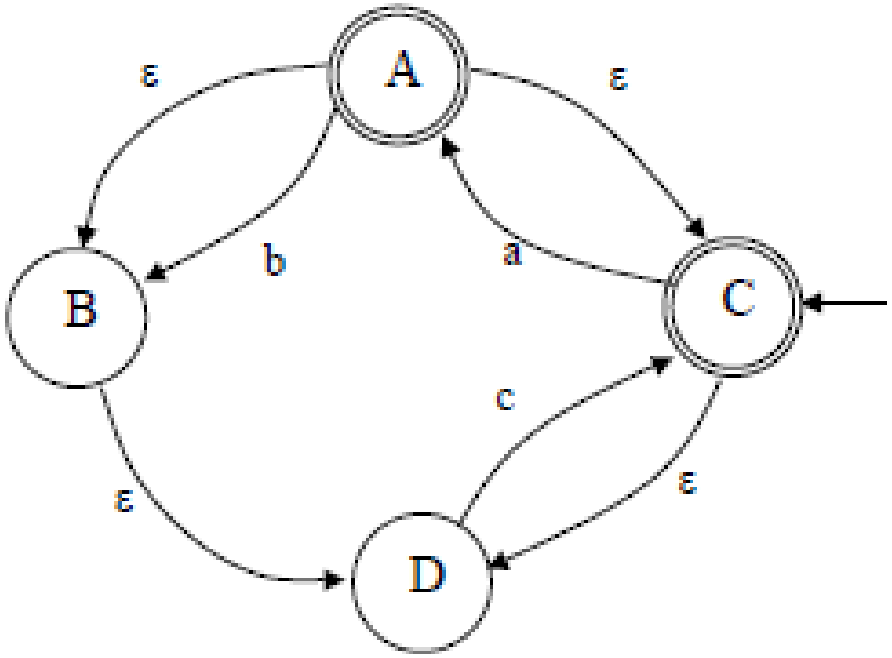
È sorprendente perché ci si aspetterebbe che un **NFSA** sia più potente di un **FSA** e che sia perciò capace di riconoscere più linguaggi.

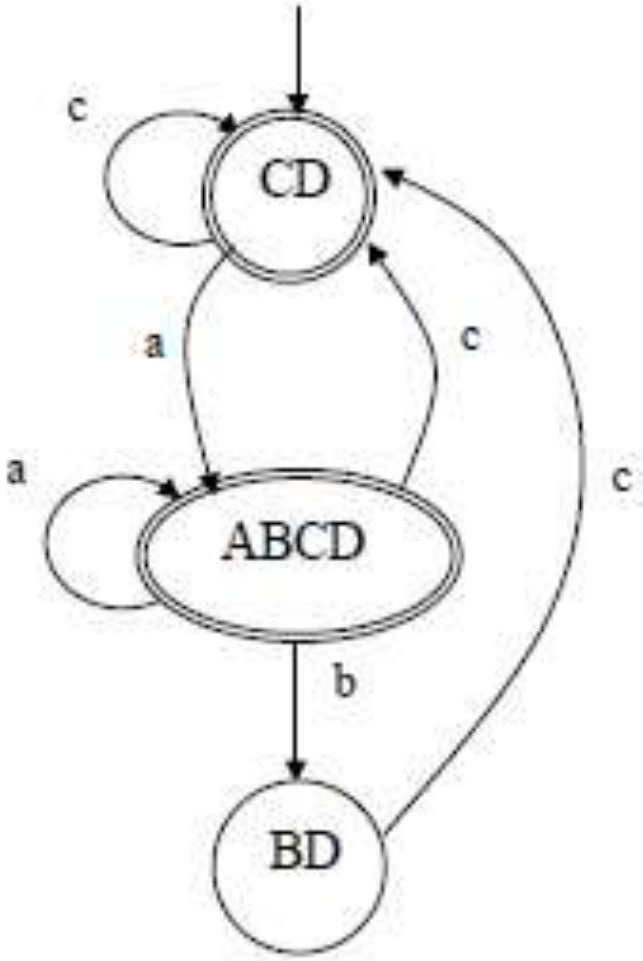
È utile perché descrivere un **NFSA** per un dato linguaggio è a volte più facile di descrivere lo stesso linguaggio con l'equivalente **FSA**.

- ✓ deterministic – faster recognizer, but it may take more space
- ✓ non-deterministic – slower, but it may take less space

NFSA e FSA equivalente

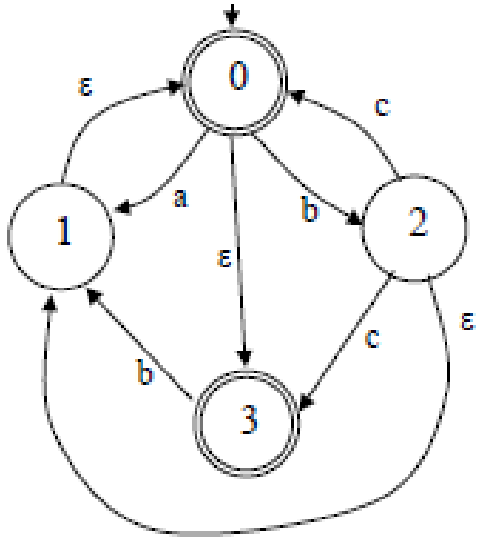
Dato il seguente automa nondeterministico, generare l'automa deterministico equivalente.



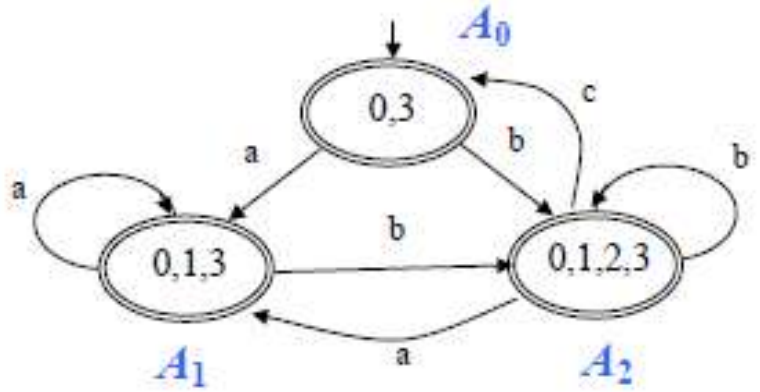


NFSA e FSA equivalente

Dopo aver rappresentato l'automa deterministico equivalente al seguente automa,



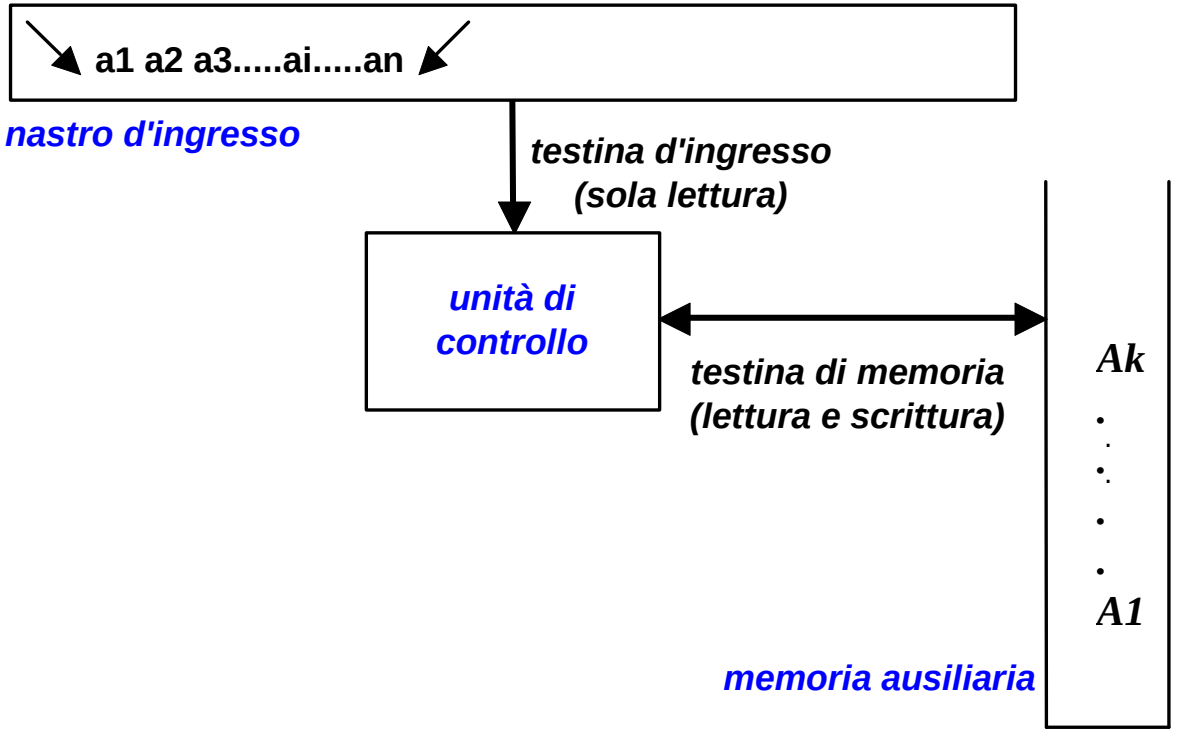
specificare la BNF relativa al linguaggio regolare riconosciuto da tale automa determinato.



$A_0 \rightarrow$ a A_1 |b A_2 | ε
$A_1 \rightarrow$ a A_1 |b A_2 | ε
$A_2 \rightarrow$ a A_1 |b A_2 |c A_0 | ε

AUTOMI A PILA
(*PUSH DOWN AUTOMATA o* PDA).

Gli automi possono anche essere dotati di memoria ausiliaria organizzata secondo una particolare struttura, lo **stack** (o *pila*).



Un automa a pila può essere pensato come un automa finito, esteso con un nastro illimitato usato come pila.

Automi a pila

Il movimento della testina è unidirezionale verso destra. Ad ogni mossa l'automa può avanzare con la testina, cambiare stato e fare un'operazione **push** o **pop**.

L'automa inizia il calcolo (configurazione iniziale) nello **stato q_0**, con il solo **simbolo Z_0 in fondo alla pila** e la testina d'ingresso posizionata sul **primo simbolo della stringa x** sul nastro d'ingresso.

Le mosse possibili sono:

- ✓ se, nello stato q_0, legge un simbolo (ad es. una ***a*** o una ***b***) esegue una **push(A)** (oppure **push(B)**);
- ✓ se, nello stato q_0, legge un particolare simbolo (ad es. una ***c***) passa allo stato q_1;
- ✓ se, nello stato q_1, legge un simbolo ***a*** (oppure ***b***) e in cima alla pila trova **A** (oppure **B**), esegue una **pop**;
- ✓ in ogni altro caso si ferma rigettando la stringa.

Perché l'automa termini di leggere, il nastro d'ingresso deve essere delimitato dalla marca di fine ↙.

La stringa è accettata se, dopo la lettura di ↙, lo stato è q_2 o, in alternativa, se la pila è vuota o è presente solo Z_0, sul quale non si può eseguire né **push** né **pop**.

NOTA BENE. La testina sul nastro di ingresso può anche non spostarsi.

Questa situazione e' espressa dicendo che la testina legge **ε** sul nastro di ingresso.

Automi a pila: il funzionamento

Si consideri la stringa ***a b c b a*** ↙ e si osservi come cambia il contenuto della pila e lo stato dell'unità di controllo (cioè la configurazione).

Pila	Nastro						Stato controllo
Z_0	***a***	***b***	***c***	***b***	***a***	↙	q_0
Z_0	A	***b***	***c***	***b***	***a***	↙	q_0
Z_0	A	B	***c***	***b***	***a***	↙	q_0
Z_0	A	B		***b***	***a***	↙	q_1
Z_0	A				***a***	↙	q_1
Z_0						↙	q_1
Z_0							q_2

Si noti come l'effetto di una mossa dipende, oltre che dal simbolo in ingresso, dal contenuto del top dello stack e dallo stato dell'unità di controllo.

Si noti ancora che il carattere Z_0 di fondo della pila non viene toccato durante la computazione, quasi che esso fosse inamovibile sul fondo.

Automi a pila e parser (o analizzatore sintattico)

Gli automi a pila sono in grado di riconoscere una classe più ampia di linguaggi rispetto agli automi a stati finiti.

Mentre gli automi a stati finiti sono in grado, infatti, di riconoscere solo linguaggi regolari, gli automi a pila possono riconoscere anche linguaggi liberi dal contesto.

Gli automi a pila sono, cioè, un sovrainsieme di quelli a stati finiti.

Alcune loro sottoclassi sono sufficientemente ampie da consentire di descrivere linguaggi di programmazione e al tempo stesso sufficientemente ristrette da consentire la costruzione di analizzatori **Analizzatori sintattici** (parser) che operano in tempo lineare, se, come vedremo in seguito, l'automa a pila riconoscitore è di tipo deterministico.

Automi a pila e parser (o analizzatore sintattico)

Un Push Down Automata **PDA** = **$<\Sigma, \Gamma, Z_0, Q, q_0, F, \delta>$** è una eptupla costituita da:

- **Σ** un alfabeto di input
- Γ alfabeto dei simboli della pila
- **$Z_0 \in \Gamma$** simbolo di pila iniziale
- **Q** insieme finito di stati dell'unità di controllo
- **$q_0 \in Q$** stato iniziale
- **$F \subseteq Q$** insieme di stati finali
- **$\delta: Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$** funzione di transizione

Se abbiamo una regola di transizione $\delta(q_i,a,A)=\{(q_j,BA),(q_h,\varepsilon)\}$ essa significa che se nello stato interno q_i, leggiamo a sul nastro ed A è il simbolo affiorante sulla pila, si realizzano, non deterministicamente, due transizioni; la prima sostituisce il simbolo affiorante sulla pila con la stringa di caratteri BA e si porta nel nuovo stato interno q_j, la seconda sostituisce il simbolo affiorante sulla pila con la stringa vuota ε, in altre parole cancella A dalla pila, e si porta nel nuovo stato interno q_h.

Convenzione**:** se metto BA in pila, B e' il nuovo simbolo affiorante.

Automi a pila e parser (o analizzatore sintattico)

L'automa così come formalmente definito è, in generale, non deterministico, sia perché prevede che, in corrispondenza di un simbolo sul nastro d'ingresso e di un elemento affiorante sulla pila, possano essere svolte due attività diverse con transizioni di stato diverse, sia perché vi possono essere delle ε-mosse.

L'uso di un PDA non deterministico come riconoscitore di stringhe ha una complessità di calcolo eccessiva, essendo il relativo algoritmo della classe $\mathbf{O}(n^3)$ oppure $\mathbf{O}(n^2)$ se la grammatica non è ambigua.

Poiché per applicazioni pratiche si cerca di impiegare riconoscitori di complessità lineare, si capisce bene come siano preferibili automi a pila di tipo deterministico.

Macchine di Turing: un po' di storia


Alan Turing (1912 - 1954)

La **macchina di Turing** (**MdT**) come modello di calcolo è stato introdotta nel 1936 da Alan Turing in un famoso articolo dal titolo "Sui numeri computabili, con una applicazione al problema della decisione".

Questo articolo era la risposta al problema della decidibilità (**Entscheidungs-problem**), uno dei 3 problemi (*coerenza*, *completezza*, *decidibilità*), proposti da Hilbert nel 1928 come sfida ai colleghi matematici.

Una MdT è una macchina formale (spesso citata come "di carta"), cioè un sistema formale che può descriversi come un meccanismo ideale, ma in linea di principio realizzabile concretamente, che può trovarsi in stati ben determinati, opera su stringhe in base a regole ben precise e costituisce un modello di calcolo.

Essa, come descritto al termine di queste note, ha la particolarità di essere retta da regole di natura molto semplice, ovvero di potersi descrivere come costituita da meccanismi elementari; inoltre è possibile presentare a livello sintetico le sue evoluzioni mediante descrizioni meccanicistiche piuttosto intuitive.

Macchine di Turing: un po' di storia

David Hilbert (1862-1943)

Il problema della ***decidibilità*** si basava su una domanda all'apparenza innocua: esiste sempre una maniera rigorosa di stabilire, 'decidere', se un certo enunciato matematico sia vero o falso? Che 1+1=2 sia una verità, o che l'equazione 2x5=0 sia falsa, è immediatamente visibile a tutti. Ma è sempre così? È sempre possibile capire, decidere se una proposizione matematica è vera o meno?

Altrettanto spinoso era il problema della ***completezza***. Interrogarsi sulla completezza della matematica significava per Hilbert porsi il quesito: siamo certi che non esistano in matematica enunciati veri ma non dimostrabili?

E col terzo problema fondante, il problema della ***coerenza***, che con gli altri 2 toccava le basi logiche di tutta la matematica, si chiedeva: come possiamo esser sicuri che la matematica non nasconda contraddizioni fra i suoi assiomi? Come possiamo dimostrare che non accadrà mai di ritrovarsi, partendo da un enunciato vero e compiendo solo passaggi matematici corretti, a un'assurdità del tipo 2+2=0?

Macchine di Turing: un po' di storia

Non era la prima volta che Hilbert, preoccupato per i fondamenti della matematica, logorati dalla nascita delle geometrie non euclidee di Riemann e Lobacevskij, lanciava una sfida ai colleghi di tutto il mondo sollecitandoli a studiare la matematica come un linguaggio formalizzato (costituito da un insieme finito di segni e regole). Al primo congresso internazionale dei matematici, svoltosi a Parigi nel 1900, tenne una conferenza in cui tentava di prevedere quali sarebbero stati gli sviluppi della matematica nel ventesimo secolo. Li individuò in 23 problemi, insoluti da anni o da secoli, e da lui giudicati di importanza cruciale. Nei decenni successivi le migliori menti matematiche gareggiarono per risolvere i quesiti di Hilbert.

Fino a poco tempo fa uno dei più celebri era il ***teorema di Fermat***.

$$x^n + y^n = z^n$$

valido solo per n=1,2 e dimostrato nel 1995 dall'inglese Andrew Wiles, con un lavoro di cento pagine.

Un altro caso famoso, la ***congettura di Goldbach***:

ogni numero intero pari maggiore di 2 può essere espresso come somma di due numeri primi (es. 4 = 3 + 1; 6 = 5 + 1; 8 = 5 + 3; ….)

è stato parzialmente dimostrato nel 2003 fino al limite di $1x10^{16}$.

Macchine di Turing: un po' di storia


Kurt Gödel (1906-1978)

Già nel 1931, a 3 anni dalla formulazione dei 3 problemi, il logico boemo Kurt Gödel aveva dato risposta ai primi due (***completezza*** e ***coerenza***), con i suoi ***teoremi di incompletezza***.

Per ogni sistema formale coerente F che si proponga di decidere, cioè dimostrare o rifiutare, tutte le asserzioni dell'aritmetica, *esiste una proposizione aritmetica che non può essere né dimostrata né rifiutata all'interno del sistema stesso*. Dunque, il sistema formale F è incompleto.

Fondamentalmente, la dimostrazione del primo teorema consiste nella costruzione, all'interno di un sistema assiomatico formale, di **una affermazione p** a cui si può dare la seguente interpretazione meta-matematica:

p = "Questa affermazione non può essere dimostrata"

una moderna variante del paradosso del mentitore del cretese Epimenide (VI sec. A.C.) che afferma "tutti i cretesi sono mentitori", più opportunamente formulata nel IV sec. A.C. da Eubulide di Megara nella forma «*quello che sto dicendo in questo momento è una menzogna*».

Macchine di Turing: un po' di storia

Il contributo di Gödel

Godel mostrò, in sostanza, come all'interno di qualsiasi sistema formale vi è una proposizione vera, ma che all'interno del sistema non può essere dimostrata. La proposizione è la proposizione che dice di se stessa che non è dimostrabile nel sistema. Ciò significava che vi era una nozione di verità distinta da quella di dimostrabilità e che la pura coerenza di un sistema non garantiva ciò che in esso era dimostrato.

Il lavoro di Gödel dimostrò di fatto l'impossibilità del programma di Hilbert. Con il suo teorema di incompletezza, Gödel dimostrava l'**esistenza di una proposizione indecidibile all'interno di un sistema logico formale**.

È importante sottolineare che questa indecidibilità è soltanto interna al sistema. Da un punto di vista esterno, la proposizione è chiaramente vera. Gödel dimostra quindi che è impossibile provare la coerenza di sistemi formali al loro interno.

Il problema della decidibilità

Nel 1928, come riferito nella precedente "storia", David **Hilbert** e Wilhelm **Ackermann** avevano lanciato il **problema della decidibilità** o Entscheidungsproblem.

La questione era stata posta da Hilbert nei seguenti termini:

esiste sempre, almeno in linea di principio, un metodo meccanico (cioè una maniera rigorosa) attraverso cui, dato un qualsiasi enunciato matematico, si possa stabilire se esso sia vero o falso?

I vantaggi derivanti dall'eventuale esistenza di un tale metodo sarebbero enormi: un tale algoritmo sarebbe in grado di risolvere tutti i problemi matematici e, molto di più, sarebbe possibile ridurre ogni ragionamento umano a mero calcolo meccanizzabile.

Il problema della decidibilità

Cinque anni dopo un giovane sconosciuto di nome Alan Turing risolse quasi per gioco il problema della ***decidibilità*** proposto da Hilbert. Con la sua "macchina immaginaria".

Partendo dalla **congettura o tesi di Church-Turing** (1936), secondo cui tutto ciò che è calcolabile mediante un processo algoritmico è computabile da una macchina di Turing, ne deriva che se un determinato compito non può essere eseguito da una macchina di Turing, non sussiste un algoritmo per eseguirlo. In questo modo Turing dimostrò che non esiste un algoritmo per *l'entschiendungs-problem*. Vediamo meglio come.

Turing chiamò D l'insieme dei numeri (o stati) che non sono codici di fermata (o stati marcati) di nessuna macchina di Turing. Allora egli pensò alla creazione di un algoritmo che, dato un numero naturale, stabilisse se appartiene o non all'insieme D. Se tale algoritmo esistesse, ci dovrebbe essere un macchina di Turing che, dato il numero, se appartiene a D si fermi dando 1, in caso contrario si fermi dando 0.

Turing mostrò però che, tramite l'introduzione di nuove quintuple, è permessa la costruzione di una nuova macchina M', la quale si fermi se il numero appartiene a D dando un 1, in caso inverso continui a spostarsi a destra sul nastro all'infinito. Questo però è impossibile giacché D è stato costruito per essere diverso dall'insieme di fermata di qualsiasi macchina di Turing. Perciò, l'ipotesi che vi sia un algoritmo per distinguere i membri di D dai non membri deve essere errata. **Il problema dell'arresto della macchina di Turing è indecidibile**.

Il teorema di Gödel e l'intelligenza umana

Il teorema di Gödel segna la fine della "certezza" nella consistenza e completezza della matematica e delle relazioni fra la matematica e il mondo reale?

La dimostrazione di proposizioni indecidibili all'interno di un sistema assiomatico sufficientemente ricco pare convalidare il punto di vista che la matematica rifletta qualche aspetto profondo della mente umana. Se i numeri e le loro relazioni matematiche "fossero" componenti del mondo reale, indipendentemente dalla mente umana, allora ogni proposizione a loro riguardo dovrebbe asserire se "è" o "non è" cosí. Il teorema di Gödel mostra perciò che, anche se i numeri fossero «reali», la nostra mente non potrebbe afferrare in modo adeguato le definizioni e gli assiomi che rifletterebbero la loro «vera» natura.

L'argomentazione che è irrilevante chi abbia creato i numeri, dato che essi a ogni modo esistono e ci sono ben noti, appare assai ragionevole, non soltanto per quanto riguarda le applicazioni quotidiane in economia o in ingegneria, ma anche per procedere abbastanza avanti nei domini dell'aritmetica e dell'analisi superiore.

Tuttavia, quando tentiamo di estendere l'uso dei numeri a campi che trascendono la nostra esperienza, oppure tentiamo di seguire le loro implicazioni troppo lontano, fino a imbatterci in paradossi e contraddizioni, allora non è irrilevante domandarsi se è la mente oppure il mondo a non essere consistente.

Il teorema di Gödel e l'intelligenza umana

Dopo il lavoro di Gödel era difficile pensare che potesse esistere un algoritmo che fornisse regole esplicite per dimostrare, all'interno di un sistema formale e in tutti i casi, che da un insieme di premesse si potesse derivare un'ipotetica conclusione tramite la logica del primo ordine.

Alan Turing, riducendo il concetto di calcolabilità effettiva a quello di procedura meccanica, con la sua Macchina di Turing, mostrò che un simile algoritmo in effetti non esisteva.

Si può dunque asserire che l'intelligenza umana non ha una natura algoritmica?

Marvin Minsky ha sostenuto che l'intelligenza umana può commettere errori e può comprendere affermazioni che sono in realtà incoerenti o false. Ciò nonostante, egli racconta che Kurt Gödel gli disse della sua convinzione che gli esseri umani possiedono una modalità intuitiva, non solo computazionale, per arrivare alla verità e che quindi il suo teorema non pone limiti a ciò che può essere riconosciuto come vero dall'uomo.

Macchina di Turing

Il massimo livello di complessità di un automa è raggiunto dalla macchina di Turing, modello che generalizza gli automi a pila (e *a fortiori* gli automi a stati finiti).

Ha una capacità computazionale che si presume essere la massima: si dimostra infatti che essa è equivalente, ossia in grado di effettuare le stesse elaborazioni di tutti gli altri modelli di calcolo di più ampia portata.

Di conseguenza si è consolidata la convinzione che per ogni problema calcolabile esista una MdT in grado di risolverlo: questa è la cosiddetta **congettura di Church-Turing**, la quale postula in sostanza che per ogni funzione calcolabile esista una macchina di Turing equivalente, ossia che l'insieme delle funzioni calcolabili coincida con quello delle funzioni ricorsive.

Per le sue caratteristiche, il modello della MdT è un efficace strumento teorico che viene largamente usato nella teoria della calcolabilità e nello studio della complessità degli algoritmi.

Macchina di Turing

Per definire in modo formalmente preciso la nozione di algoritmo preferenzialmente oggi si sceglie di ricondurlo alle elaborazioni effettuabili con macchine di Turing.

Della MdT vengono considerate molteplici varianti. Noi consideriamo una variante piuttosto semplice che possiamo chiamare **macchina di Turing deterministica a un nastro e con istruzioni a cinque campi**.

Esistono numerose altre varianti della MdT.

Definizione informale di MdT (1/3)

La macchina può agire sopra un nastro che si presenta come una **sequenza di caselle** nelle quali possono essere registrati **simboli** di un ben determinato **alfabeto finito**; essa è dotata di una **testina di lettura e scrittura** (I/O) con cui è in grado di effettuare operazioni di lettura e scrittura su una casella del nastro.

La macchina evolve nel tempo e ad ogni istante si può trovare in uno **stato interno** ben determinato facente parte di un **insieme finito di stati**. Inizialmente sul nastro viene posta una stringa che rappresenta i dati che caratterizzano il problema che viene sottoposto alla macchina.

La macchina è dotata anche di un **insieme finito di istruzioni** che determinano la sua evoluzione in conseguenza dei dati iniziali. L'evoluzione si sviluppa per passi successivi che corrispondono a una sequenza discreta di istanti successivi.

Le proprietà precedenti sono comuni a molte macchine formali (automa a stati finiti, automa a pila, ...). Caratteristica delle MdT è quella di disporre di un nastro potenzialmente infinito, cioè estendibile quanto si vuole qualora questo si renda necessario.

Definizione informale di MdT (2/3)

Ogni passo dell'evoluzione viene determinato dallo **stato attuale *s*** nel quale la macchina si trova e dal **carattere *c*** che la testina di I/O trova sulla casella del nastro su cui è posizionata e si concretizza nell'eventuale modifica del contenuto della casella, nell'eventuale spostamento della testina di una posizione verso destra o verso sinistra e nell'eventuale cambiamento dello stato.

Quali azioni vengono effettuate ad ogni passo viene determinato dalla **istruzione**, che supponiamo unica, che ha come prime due componenti *s* e *c*; le altre tre componenti dell'istruzione forniscono nell'ordine il **nuovo stato**, il **nuovo carattere** e una **richiesta di spostamento** verso sinistra, nullo o verso destra.

Definizione informale di MdT (3/3)

Una evoluzione della macchina consiste in una sequenza di sue possibili **configurazioni**, ogni configurazione essendo costituita dallo stato interno attuale, dal contenuto del nastro (una stringa di lunghezza finita) e dalla posizione sul nastro della testina di I/O.

Nei casi più semplici l'evoluzione ad un certo punto si arresta in quanto non si trova nessuna istruzione in grado di farla proseguire.

Si può avere un arresto in una configurazione "utile" dal punto di vista del problema che si vuole risolvere; in tal caso quello che si trova registrato sul nastro all'atto dell'arresto rappresenta il risultato dell'elaborazione.

Si può avere però anche un arresto "inutile" che va considerato come una conclusione erronea dell'elaborazione.

Va subito detto che può anche accadere che un'evoluzione non abbia mai fine (**problema dell'arresto**).

Definizione formale di MdT (1/2)

Si definisce **macchina di Turing deterministica a un nastro e istruzioni a cinque campi**, termine che abbreviamo con MdT1n5i, una macchina formale della seguente forma:

S è un insieme finito detto **insieme degli stati** della macchina;

s_0 è un elemento di *S* detto **stato iniziale** della macchina;

F è un sottoinsieme di *S* detto **insieme degli stati finali** della macchina;

A è un alfabeto finito detto **alfabeto del nastro** della macchina;

β è un carattere dell'alfabeto *A* detto **segno di casella vuota del nastro** della macchina

$\delta : S \times A \mapsto S \times A \times \{-1, 0, +1\}$ è detta **funzione di transizione** della macchina.

Definizione formale di MdT (2/2)

Se $\delta(s,a) = \langle b,t,m \rangle$, la corrispondente quintupla $\langle s,a,b,t,m \rangle$ può considerarsi come l'**istruzione** che viene eseguita quando la macchina si trova nello stato ***s*** e la testina di I/O legge ***a*** sulla casella sulla quale è posizionata; essa comporta la scrittura del carattere ***b***, la transizione allo stato ***t*** e:

quando m = -1 lo spostamento della testina di una posizione a sinistra,
quando m = 0 nessuno spostamento della testina,
quando m = +1 lo spostamento della testina di una posizione a destra.

Rilevanza della MdT e congettura di Church-Turing

Tra le macchine formali che in linea di principio si possono considerare come varianti estremamente articolate della macchina di Turing introdotta inizialmente vanno considerati anche gli attuali elaboratori elettronici programmabili con linguaggi di programmazione dei vari livelli, fino a quelli in grado di effettuare calcoli simbolici ed elaborazioni parallele.

Si deve anche aggiungere che della MdT risulta opportuno anche considerare varianti non deterministiche, macchine formali che sono in grado di portare avanti contemporaneamente diverse elaborazioni, in numero illimitato. Anche queste macchine formali, a prima vista lontane da modelli di meccanismi concretamente realizzabili, possono considerarsi idealizzazioni di sistemi di computer che operano in parallelo, sistemi che la odierna tecnologia consente di realizzare abbastanza comunemente (i cosiddetti cluster) fino a quelli per il grid computing.

Tutte queste considerazioni rendono ragionevole sostenere la **congettura o tesi di Church-Turing** (1936): **Ogni algoritmo che puo' essere eseguito da un umano o da un computer puo' essere eseguito da una macchina di Turing**.

Per dimostrare che una funzione è calcolabile (o un linguaggio è decidibile), basta scrivere un algoritmo che la calcoli: la tesi di Church-Turing ci assicura che può essere eseguito da una macchina di Turing.