Aniello Murano

Macchine di Turing

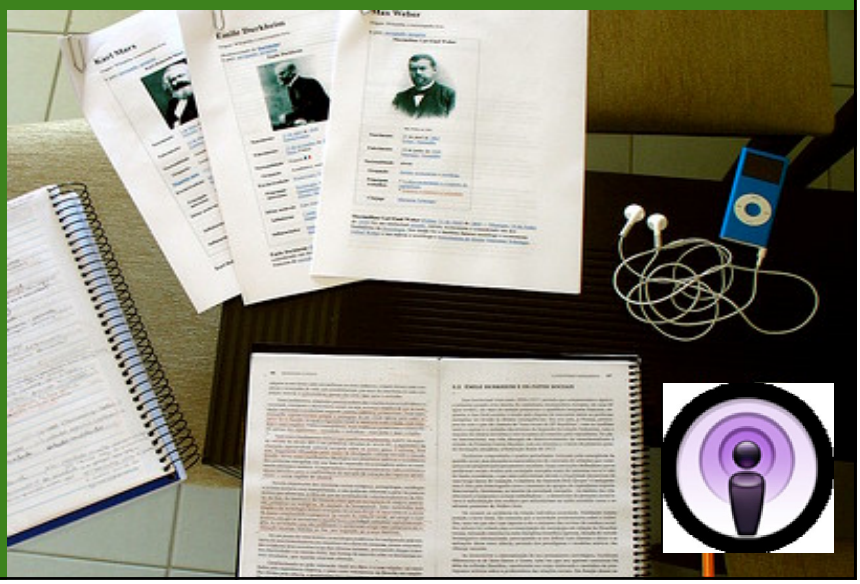
Lezione n.3
Parole chiave:
Turing machine

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Potere computazionale di MdT

- Tra le macchine computazionali, la **Macchina di Turing** (**MdT**) è senza dubbio la più potente
- In letteratura, è stata dimostrata l'equivalenza computazionale (ossia in grado di effettuare le stesse elaborazioni) con diversi altri modelli di calcolo di più ampia portata come:
  - le funzioni ricorsive di Jacques Herbrand e Kurt Gödel,
  - il lambda calcolo di Alonzo Church e Stephen Kleene,
  - la logica combinatoria di Moses Schönfinkel e Haskell Curry,
  - gli algoritmi di Markov,
  - i sistemi di Thue,
  - i sistemi di Post,
  - le macchine di Hao Wang
  - le macchine a registri elementari o RAM astratte di Marvin Minsky.
  - ....
  - ....
- **TESI DI CHURCH-TURING**:
  Per ogni problema calcolabile esista una MdT in grado di risolverlo

Intermezzo

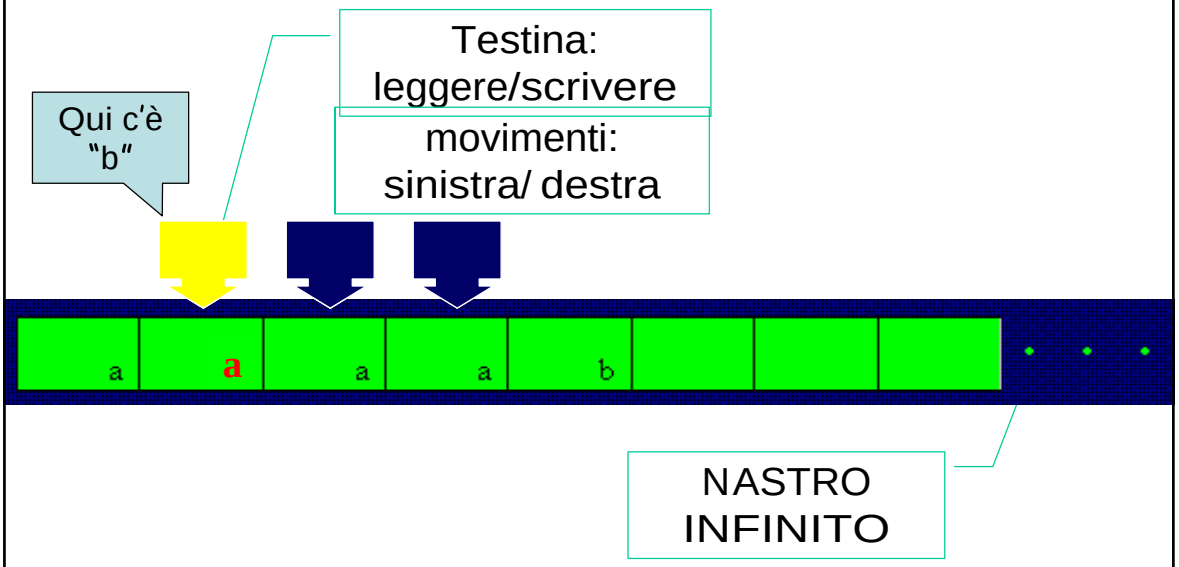
- La MdT come modello di calcolo è stato introdotta nel 1936 da Alan Turing.
- Nel 1936 venne pubblicato un articolo di Alan Turing intitolato "On computable numbers, with an application to the Entscheidungsproblem", in cui l'autore risolveva negativamente il problema della decidibilità (Entscheidungsproblem) lanciato nel 1900 da David Hilbert e Wilhelm Ackermann che recitava così:

  « 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?»
- L'utilità di un tale algoritmo sta nel fato che sarebbe in grado di risolvere tutti i problemi matematici e ancora più importante che ogni ragionamento umano poteva essere ridotto a mero calcolo meccanizzabile.
- E' da citare anche che una prima risposta negativa al problema della decidibilità la diede il matematico Gödel nel 1931 con il lavoro sull'incompletezza dei sistemi formali coerenti ("primo teorema di incompletezza");
- Gödel dimostrò che la semplice coerenza di un sistema formale non può garantire che ciò che in esso viene dimostrato sia vero oppure falso.

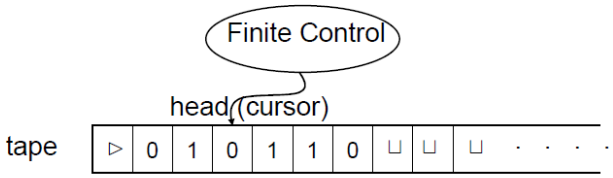
Descrizione della macchina di Turing

- Una macchina di Tuding opera su un nastro (potenzialmente infinito) che come per gli NFA si presenta come una sequenza di caselle nelle quali possono essere registrati simboli di un ben determinato alfabeto finito.
- A differenza degli NFA, la testina di una MdT può, oltre che leggere, anche scrivere sul nastro. Per questo motivo la testina viene chiamata di I/O. Inoltre, la testina può spostarsi a destra o a sinistra.
- Ogni passo dell'evoluzione viene determinato dallo stato attuale *s* nel quale la macchina si trova e dal carattere 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 e nell'eventuale cambiamento dello stato.
- Una **evoluzione** della macchina consiste in una sequenza di sue possibili "**configurazioni**".
- L'evoluzione di una MdT può sia arrestarsi in una certa configurazione (che può essere "utile" o "meno inutile" oppure può anche accadere che l'evoluzione non abbia mai fine.

Funzionamento schematico di una MdT



MACCHINA DI TURING: definizione formale

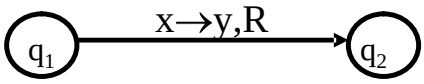


- Una MdT (deterministica) è formata da una 7-pla (Q, Σ, Γ, δ, q_0, accept, reject):
  - **Q** è un insieme finito di **stati**;
  - **STATI SPECIALI**: q_0 è lo **stato iniziale**
    "accept" è lo **stato accettante**;
    "reject" è lo **stato rifiutante**;
  - **Γ** è l'**alfabeto del nastro**. Include il simbolo speciale "-" oppure "␣" (simbolo di blank);
  - **Σ ⊆ Γ** \{-} è l'**alfabeto di input**;
  - **δ** : Q\{accept, recject} × Γ → Q× Γ ×{L,R} è la **funzione di transizione** (parziale).
- Nota: accept ≠ reject e sono degli **halting states,** cioè non esistono transizioni da questi stati.
- **Osservazione**: nella definizione classica, la MdT ha anche un simbolo speciale di starting sul nastro, uno stato speciale aggiuntivo "halt" di halting e la testina ha la possibilità di rimanere ferma in una transizione. Queste modifiche non alterano il potere computazionale della macchina. Talvolta, noi useremo queste varianti per semplicità.

Funzionamento di una MdT

- La funzione di transizione $\delta : Q\backslash\{accept, reject\} \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ permette alla MdT, a partire da un certo stato e leggendo un simbolo sul nastro, di transire in un nuovo stato, sostituire il simbolo letto sul nastro, e muovere la testina di lettura a sinistra o a destra.
- INIZIALMENTE: Stato=q_0 ; nastro in input ... - - $w_1 ... w_n$ - -... testina posizionata su w_1 (su ▷ se esiste, che si troverà subito a sinistra di w_1);
- AD OGNI PASSO: nuova configurazione conforme alla funzione di transizione δ;
- La testina non può muoversi a sinistra del primo simbolo della parola in input.
- Se si fa uso del marcatore di inizio stringa "▷", questo simbolo non viene mai sovrascritto e la testina non va mai oltre questo simbolo.
- Per semplicità, assumiamo che quando la testina si trova negli stati "accept" o "reject" (anche "halt" se previsto) non si muove.

Rappresentazione grafica



La rappresentazione grafica in figura rappresenta la situazione in cui:
- il simbolo corrente sul nastro è x,
- Lo stato corrente è q_1,
- La testina legge x, lo sostituisce con y, si sposta a destra e transisce nello stato q_2

Se x=y, allora l'etichettatura x→x,R si può semplicemente scrivere come x→R.
Se invece di R, ci fosse stato L, allora la testina si sposta a sinistra, a meno che si tratti della prima cella non blank, nel qual caso, la testina non si sposta.

CONFIGURAZIONE DI UNA MdT

- Una **configurazione** di una MdT è data dalle seguenti informazioni:
  1. Stato corrente;
  2. Contenuto del nastro (i simboli appartenenti a Σ)
  3. La posizione della testina.

- Una configurazione può essere rappresentata come una tupla **(u,q,v)** o con la stringa **uqv** dove:
  - q è stato corrente;
  - u ϵ Σ* è la stringa di simboli di Σ che si trova a sinistra della testina;
  - v ϵ Σ* è la stringa di simboli di Σ che si trova a destra della testina;

- Per esempio, $11q_7011$ rappresenta la configurazione dove il nastro è 11011, lo stato corrente è q_7 e la testina è sullo zero.

- Una configurazione u accept v è detta di **accettazione**; u reject v è invece di **rifiuto**

- La relazione di esecuzione tra configurazioni è (→ si legge "porta a") :
  - $C_1 \rightarrow C_2$: La MdT esegue un passo da C1 a C2;
  - $C_1 \rightarrow^* C_2$: chiusura transitiva - La MdT esegue in 0,1,2 o più passi uno spostamento da C_1 a C_2;

Accettazione di Linguaggi

- Una MdT M ***accetta*** una stringa in input se, prima o poi incontra lo stato "**accept**". Altrimenti la stringa è non accettata. L(M)={x ∈ Σ^* | M su input x raggiunge lo stato "accept"}

- Dunque, l'input è non accettato da M in due casi:

  1. M incontra ad un certo punto lo stato reject;

  2. M non si ferma mai (non incontra mai uno stato di halting)

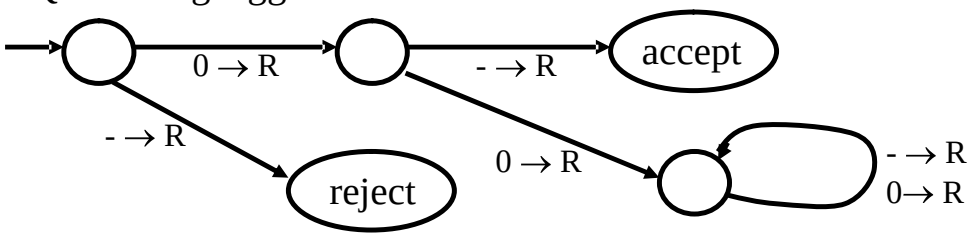
- Il linguaggio **riconosciuto** (accettato) da una MdT è l'insieme delle stringhe che esso accetta

LA MACCHINA DI TURING COME UN RICONOSCITORE DI LINGUAGGIO

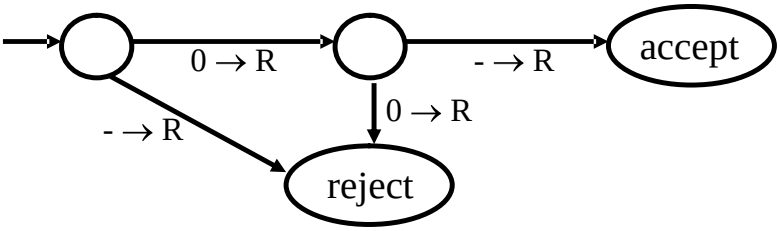
- Una MdT è detta **decisore** se si ferma su ogni input
- Se la macchina è un **decisore**, allora diremo che non solo accetta un linguaggio, ma lo decide anche. Formalmente, M decide un linguaggio $L \subseteq \Sigma^*$ se $L=L(M)$ e per ogni x in Σ^*-L, M termina in uno stato "reject";
- Un linguaggio è detto **Turing-riconoscibile** (o **ricorsivo-enumerabile**) se esiste una MdT che lo riconosce
- Un linguaggio è detto **Turing-decidibile** (o **ricorsivo**) se esiste una MdT che lo decide
- LINGUAGGI RICORSIVI ⊆ LINGUAGGI RICORSIVI ENUMERABILI;

RICONOSCITORE vs DECISORE

Quale linguaggio riconosce la seguente MdT?
Questo linguaggio è solo accettato o anche deciso?



Quale linguaggio riconosce la seguente MdT?
Questo linguaggio è solo accettato o anche deciso?



Esempio 1

- $L=\{x \in \{0,1\}^* \mid x \text{ ha un numero dispari di } 1\}$

Metodo: La testina si muove a destra e si mantiene nello stato la parità di 1 letti;

Stati $Q=\{q_0, q_1, \text{accept}, \text{reject}\}$

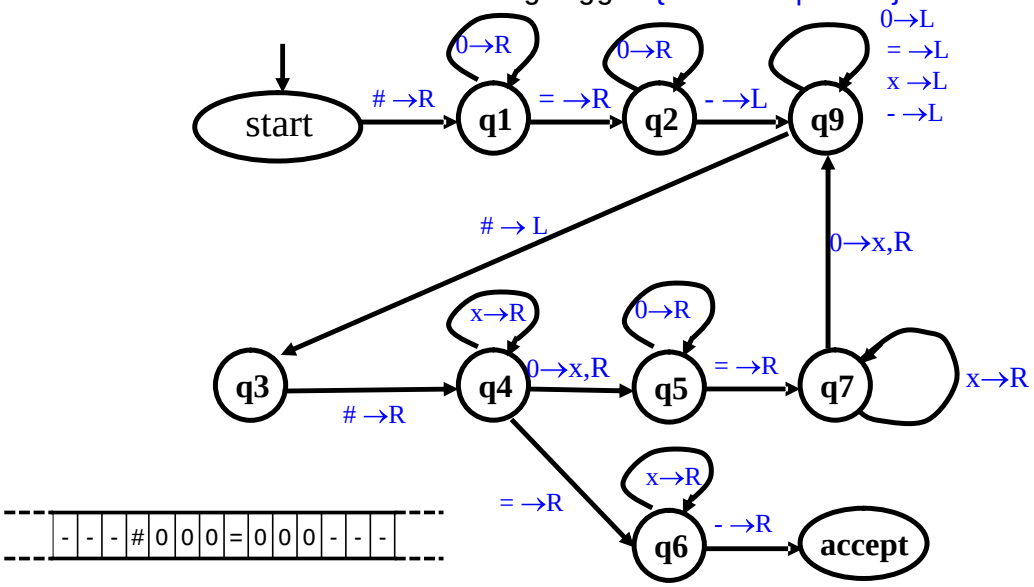
	0	1	-
q_0	$q_0 \rightarrow R$	$q_1 \rightarrow R$	reject $\rightarrow L$
q_1	$q_1 \rightarrow R$	$q_0 \rightarrow R$	accept $\rightarrow L$

Funzione di transizione rappresentata attraverso la tabella di transizione

- Ogni linguaggio regolare può essere analogamente deciso da una TM che muove la testina dall'inizio del nastro verso destra leggendo l'input;

Esempio 2

Si definica una MdT che decida il linguaggio $\{\#0^n=0^m \mid n=m\}$



Per semplicità assumiamo che tutte le transizioni mancanti vadano nello stato **reject**

Esercizi

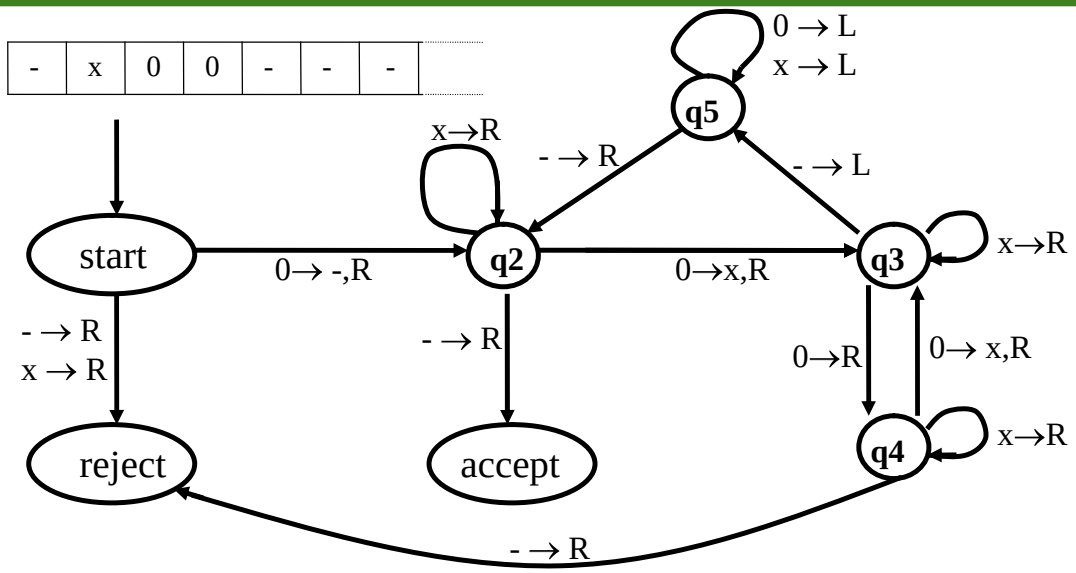
Si definica una MdT che decida il linguaggio $\{\#0^n<0^m \mid n<m\}$

	-	-	-	#	0	0	0	<	0	0	-	-	-

Si definica una MdT che decida il linguaggio $\{\#0^n+0^m=0^k \mid n+m=k\}$

	-	-	-	#	0	0	0	+	0	0	=	0	0	0	0	0	-	-	-

Esempio 2



Esempio di un MdT che decide il linguaggio $\{0^{2^n} \mid n > 0\}$ costituito da tutte le parole di 0 la cui lunghezza è una potenza di 2

LA MACCHINA DI TURING COME FUNZIONE COMPUTABILE

- Consideriamo la variante della MdT con lo stato aggiuntivo "halt"
- FUNZIONE COMPUTABILE con M: su input x, se M termina in uno stato di "accept" allora M(x)="accept", altrimenti se termina in uno stato di "no" M(x)="reject", se invece termina in uno stato "halt" allora M(x)= contenuto del nastro che si trova tra il primo simbolo "blank" (o il simbolo speciale "▷") e l'insieme infinito di simboli blank "-";
- Su altri input x (es. quando M gira all'infinito): M(x)= indefinito;
- Per le funzioni possiamo avere per input e output alfabeti differenti;
- Una funzione è ricorsiva se è computabile da qualche Macchina di Turing;

Trasduttori

- In pratica, una MdT che calcola una funzione (anche parziale), può essere vista come un **Trasduttore**
- A partire da un input sul nastro, se la macchina ad un certo punto si ferma, l'output è la stringa presente sul nastro in quel momento.
- Questo sarà ancora più evidente quando vedremo le macchine di Turing a più nastri, dove un nastro può essere usato per l'input, uno per l'output e altri nastri per l'esecuzione della macchina
- Esercizio: Definire una macchina trasduttrice che calcoli la funzione prodotto di due interi positivi in notazione unaria, seconde le seguenti convenzioni:
  - Sul nastro memorizzata la stringa $1^n\#1^m$, dove le due sequenze non vuote di 1 rappresentano i numeri da moltiplicare in notazione unaria.
  - Quando la macchina si ferma, sul nastro verrà memorizzata la stringa 1^{nm}.

ESEMPIO

- INCREMENTO DI UN NUMERO BINARIO: $f(x)=x+1$ dove $x \in \{0,1\}^+$

Metodo: Muovi la testina verso destra fino alla fine(al primo simbolo di Ц)
e vedi se $x=1^n$(tutti i bit=1) :

- se $x=1^n$ allora cambia in 10^n;
- altrimenti sostituisci tutti gli 1 con gli 0 e l'ultimo 0 a 1;

Stati: Q={s,p,q,r,h} (h è accept)

	▷	0	1	⊔
s	s,▷,R	p,0,R	s,1,R	q,0,L
p	p,▷,R	p,0,R	p,1,R	q,⊔,L
q	r,▷,R	h,1,S	q,0,L	q,⊔,L
r	r,▷,R	h,1,S	h,1,S	h,⊔,S

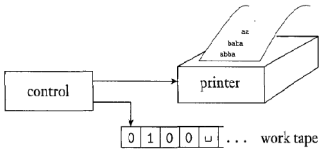
ESEMPIO(3) RICONOSCIMENTO DI UNA PALINDROMA CON UNA TM AD UN NASTRO

- LINGUAGGIO DI PALINDROME: tutte le stringhe x tali che x è uguale a x letta in senso contrario;
  - Esempio: amanama,010010 ;
- Macchina di Turing per l'accettazione del linguaggio di palindrome: Muovi la testina avanti e indietro in modo che:
  - verifica il primo simbolo con l'ultimo simbolo e marcali, ad esempio sostituendoli il primo con ▷ e l'ultimo con Ц ;
  - verifica il secondo simbolo con il penultimo, e cosi via…

	▷	0	1	⊔
s	s,▷,R	q0,▷,R	q1,▷,R	yes,⊔,S
q0	q0,▷,R	q0,0,R	q0,1,R	p0,⊔,L
q1	q1,▷,R	q1,0,R	q1,1,R	p1,⊔,L
p0	yes,▷,S	r,⊔,L	no,1,S	r,⊔,S
p1	yes,▷,S	no,0,S	r,⊔,L	r,⊔,S
r	s,▷,R	r,0,L	r,1,L	r,⊔,S

Perché "recursively enumerable"

- Come abbiamo detto in precedenza, la classe dei linguaggi Turing-riconoscibile (Turing-recognizable) è anche chiamata *ricorsiva enumerabile* (*recursively enumerable*).
- Questo termine ha origine da una variante delle macchine di Turing: l'***enumeratore.***
- Un enumeratore è una macchina di Turing con stampante usata per stampare stringhe.
- Ogni qualvolta la macchina di Turing vuole aggiungere una stringa alla lista, semplicemente manda la stringa alla stampante.
- Nella figura è mostrato lo schema dell'enumeratore.
- Un enumeratore E inizia a lavorare con un nastro vuoto.
- Se l'enumeratore lavora per sempre, può produrre una lista infinita di stringhe
- Il linguaggio enumerato da E è l'insieme di tutte le possibili stringhe che E riesce a stampare.
- In particolare, è utile notare che E può generare le stringhe in qualsiasi ordine, anche con ripetizione.
- **Teorema:** Un linguaggio è Turing-riconoscibile (Turing-recognizable) se e solo se esiste un enumeratore che lo enumera.



Macchine di Turing e gli algoritmi

- Un algoritmo si può definire come una collezione di istruzioni semplici per la realizzazione di un determinato compito.
- Gli algoritmi sono anche conosciuti in ambiti non informatici come procedure o ricette.
- Gli algoritmi giocano un ruolo importante in varie discipline (ingegneria, matematica, medicina, ecc.)
- Anche la matematica classica mostra una varietà di algoritmi per la soluzione dei problemi più disparati, come per esempio trovare numeri primi, il massimo comune divisore ecc.)
- Anche se gli algoritmi hanno una lunga storia nella letteratura, la loro effettiva formalizzazione risale al ventesimo secolo.
- La storia delle prossime slide mostra come la precisa definizione di algoritmo è stata fondamentale per la soluzione di un importante problema matematico.

I problemi di Hilbert

- Nel 1900, il matematico David Hilbert, in un importante congresso matematico tenutosi a Parigi, identificò 23 problemi matematici difficili e li consegnò alla comunità scientifica come "problemi competitivi" per il secolo che stava per iniziare.
- Alcuni problemi sono stati risolti in pochi anni, altri hanno visto la soluzione solo pochi anni fa, ed altri ancora non hanno avuto tuttora risposta .
- Il decimo problema della lista riguardava gli algoritmi ed in particolare chiedeva di definire un algoritmo che "dato in input un polinomio, fosse in grado di testare se quel polinomio ammettesse o meno radice intera".
- Una radice di un polinomio è un assegnamento di valori alle sua variabili in modo tale che il valore del polinomio diventi 0.
- Un polinomio è la somma di termini, dove ciascun termine è il prodotto di una costante (chiamato coefficiente) e di alcune variabili.
- Per esempio, $6x^3 yz^2 + 3xy^2 - x^3 - 10$ è un polinomio di 4 termini con variabili x, y e z. La usa radice che $x = 5$, $y = 3$, and $z = 0$. Inoltre, questa radice è intera perché tutte gli assegnamenti alle variabili sono numeri interi. Si osservi che esistono polinomi con radici non intere.

L'indecidibilità del decimo problema di Hilbert

- Una osservazione sul decimo problema di Hilbert:
- Hilbert non usò il termine algoritmo per questo problema, ma quello di "processo" che, per definizione, è determinato da un numero finito di operazioni.
- Dunque, Hilbert non chiedeva di controllare l'esistenza di un algoritmo ma solo di implementarlo. Questo perché era convinzione comune che ogni problema dovesse ammettere una procedura risolutiva.
- Oggi sappiamo che il decimo problema di Hilbert è indecidibile! (questo risultato è stato formalmente provato nel 1970 da Yuri Matijasevic che estese una idea di Martin Davis, Hilary Putnam, and Julia Robinson)
- Ai tempi di Hilbert, gli strumenti a disposizione dei matematici erano certamente adeguati per mostrare procedure per alcuni problemi, ma erano completamente inadeguati per provare l'inesistenza di una procedura per quelli che oggi sappiamo essere indecidibili.
- I primi risultati sul decimo problemi di Hilbert arrivarono negli anni '30 e passarono per una definizione rigorosa del concetto di algoritmo.
- Tale definizione arrivò ad opera di Alonzo Church and Alan Turing:
  - Church usò un sistema notazionale chiamato λ-calculus.
  - Turing usò un modello di macchina computazionale (la macchina di Turing)
- Queste due definizioni furono poi mostrate essere equivalenti. Questa equivalenza diede origine alla seguente tesi chiamata tesi di ***Church-Turing.***

Tesi di Church-Turing

- Un problema (o una funzione) è **calcolabile (o decidibile)** in modo algoritmico (o calcolabile in modo effettivo, o effettivamente calcolabile) se esiste un algoritmo che consente di calcolarne i valori per tutti gli argomenti.
- Nel 1936 il logico americano Alonzo Church, in seguito alle sue ricerche sulla computabilità effettiva, propose di identificare la classe delle funzioni calcolabili mediante un algoritmo (o funzioni effettivamente calcolabili) con una particolare classe di funzioni aritmetiche (funzioni ricorsive). Tale identificazione è oggi nota col nome di **Tesi di Church**.
- È possibile dimostrare l'equivalenza tra la classe delle funzioni ricorsive e la classe delle funzioni **Turing-computabili** (decidibili), in quanto ogni funzione Turing-computabile è ricorsiva, e viceversa (Turing 1937).
- La Tesi di Church può quindi essere formulata come segue:

"una funzione è effettivamente calcolabile
se solo se
è Turing-computabile"

Tesi di Church Turing(2)

- Tesi di Church - Turing:

  "La classe delle funzioni calcolate da una MdT è la classe delle funzioni che informalmente sono considerate effettivamente calcolabili, o equivalentemente la classe dei linguaggi decisi da una MdT corrisponde alla classe dei problemi intuitivamente effettivamente risolvibili."

- Quindi possiamo identificare l'idea di algoritmo con quella di una MdT che si ferma sempre.

- Mentre un semialgoritmo corrisponde a una MdT che non sempre si ferma.

- Per approfondimenti vedere anche:

  http://www-ictserv.poliba.it/piscitelli/doc/lucidi%20TL/7_Automi%20e%20Macchine%20di%20Turing.pdf

  http://www.dif.unige.it/epi/hp/frixione/appunti_MT.pdf

Il decimo problema di Hilbert è Turing-recognizable

- Mostriamo che il decimo problema di Hilbert è Turing-riconoscibile (Turing-recognizable).
- Dapprima consideriamo il problema per il caso semplificato di polinomi ad una sola varabile, tali come $4x^3 - 2x^2 + x - 7$.
- In questo caso, valutiamo se l'insieme D = {p | p è un polinomio su x con radice intera su x} è Turing-riconoscibile (Turing-recognizable) o meno.
- Una macchina di Turing M in grado di accettare D è la seguente:
  1. M prende input un polinomio p sulla variabile x.
  2. Poi M valuta p assegnando a x successivamente i valori 0, 1, -1, 2, -2, 3, 3....
  3. Se ad un certo punto il polinomio si valuta 0′ allora M va in accettazione.
- Se p ha una radice intera, allora M prima o poi la troverà e dunque accetterà p. Se invece p non ha una radice intera allora M continuerà a computare all'infinito.
- Per polinomi a n variabili, si può utilizzare una macchina di Turing M′ equivalente ad M, dove il test è fatto su insiemi ordinati di valori interi differenti.
- Sia M che M′ così come sono state definite sono riconoscitori ma non decisori di linguaggi.
- Tuttavia, è possibile convertire M in un decisore utilizzando il seguente risultato:

  "La radice intera di un polinomio a singola varabile se esiste è compresa tra $\pm k(c_{max}/c_1)$ dove k è il numero dei termini, c_{max} è il coefficiente con valore assoluto più grade e c_1 è il coefficiente del termine di grado massimo.
- Matijasevic ha mostrato invece che è impossibile calcolare questo limite per polinomi con più variabili.

Proprietà delle Turing Machine

- **Teorema:** Se L e il suo complemento sono Turing riconoscibili allora L è decidibile.
- **Teorema:** Le MdT sono chiuse rispetto a unione e intersezione.
- **Teorema:** La classe dei linguaggi decidibili è chiusa rispetto al complemento.

- La dimostrazione dei precedenti teoremi è lasciata per esercizio agli studenti