# Varianti Macchine di Turing

Esistono definizioni alternative di macchina di Turing.
Chiamate Varianti.

Tra queste vedremo: MdT a più nastri e MdT non deterministiche.

Mostriamo: tutte le varianti "ragionevoli" hanno la stessa capacità computazionale

# IIntroduzione e Motivazioni

Quando proviamo che un MdT ha una certa proprietà non ci poniamo domande del tipo:

- Quanto è grande la MdT ?
- Quanto è complesso programmarla?

Per il momento ci interessano solo proprietà esistenziali.

# Un esempio

Stayer: MdT in cui la testina può rimanere sulla stessa cella del nastro durante una transizione
Funzione di transizione: $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, S, R\}$
dove S serve ad indicare che la testina rimane ferma

# Potere computazionale

**Diciamo che il potere computazionale di due modelli è lo stesso se riconoscono la stessa classe di linguaggi**

MdT può essere facilmente simulata da stayer
(basta non usare la possibilità di stare)
$\Rightarrow$
Il potere computazionale di stayer è almeno pari al potere computazionale di MdT convenzionale

# Potere computazionale

Rimane da provare:

il potere computazionale di MdT è almeno pari al potere computazionale di Stayer
$\Leftrightarrow$
per ogni stayer esiste MdT che riconosce stesso linguaggio

Assumiamo M è stayer e mostriamo MdT M' che simula M

# Simulazione

M' definita come M tranne che
ogni transizione S è sostituita da due transizioni in M':

1. la prima porta M' in uno stato speciale (addizionale) e muove testina a destra (R)
2. la seconda ritorna allo stato originale muovendo la testina a sinistra (L) .

Quindi M e M' riconoscono lo stesso linguaggio

# Simulazione in generale

In generale:

Quando vogliamo provare che due varianti hanno lo stesso potere computazionale

mostriamo che possiamo simulare una con l'altra

# Implementazione della Memoria

In alcune occasioni avremo bisogno di memorizzare l'informazione letta sul nastro

Possiamo usare gli stati per memorizzare informazione

Es. Supponiamo MdT M deve leggere i primi 2 bit del nastro e scriverli alla fine dell'input (al posto dei 2 $\perp$ piú a sinistra)

# Implementazione della Memoria

Idea:

- Costruiamo una MdT
  $M_{00}$ che legge i primi 2 bit, cerca la fine dell'input e scrive 00 alla fine dell'input
- Replichiamo per tutte le possibili coppie: $M_{01}$, $M_{10}$, $M_{11}$
- M: dopo aver letto i primi 2 bit input, si sposta sulla replica corrispondente ai 2 bit letti (e quindi li scrive alla fine dell'input).

# Implementazione della Memoria

Nota: questa è una tecnica utilizzabile in generale

Se vogliamo che M memorizzi una sequenza di $k$ simboli possiamo

- avere una replica per ogni possibile sequenza di $k$ simboli
- M si sposta sulla replica che corrisponde alla sequenza mentre la legge

Necessari circa $|\Sigma|^k$ stati aggiuntivi

# Macchina di Turing multi–nastro

Una MdT a $k$ nastri è una normale macchina di Turing avente $k$ nastri

- Inizialmente l'input compare sul primo nastro e gli altri sono vuoti
- La funzione di transizione

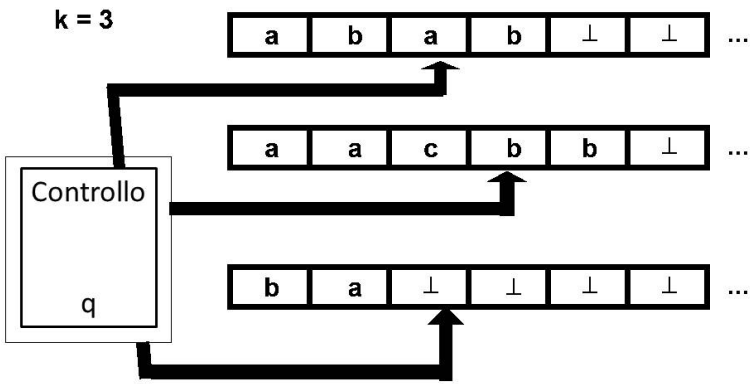
$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, S, R\}^k$$

muove (in modo indipendente) le testine dei vari nastri

# Macchina di Turing multi–nastro

- Simile a MdT: usa $k$ nastri con $k \geq 1$
- Ciascun nastro con propria testina
- Istruzione specifica (oltre nuovo stato)
  - $k$ simboli letti
  - $k$ simboli da scrivere
  - $k$ movimenti delle $k$ testine
- Configurazione iniziale: input su primo nastro, rimanenti nastri vuoti

# Es.: MdT multi-nastro



# Macchina di Turing multi-nastro

Espressione

$$\delta(q_i, a_1, \ldots, a_k) = (q_j, b_1, \ldots, b_k, D_1, \ldots, D_k)$$

indica che se la MdT a $k$ nastri M

- si trova nello stato $q_i$
- la testina $i$ legge $a_i$, per $i = 1 \ldots, k$

allora

- M va nello stato $q_j$
- la testina $i$ scrive $b_i$ e si muove nella direzione $D_i \in \{L, S, R\}$, per $i = 1 \ldots, k$

# Esempio: MdT a 2 nastri per $0^n1^n$

1. Scorre primo nastro verso destra fino a primo `1`: per ogni `0`, scrive un `1` sul secondo nastro
2. Scorre primo nastro verso destra e secondo nastro verso sinistra: se simboli letti non uguali, termina in stato non finale
3. Se legge $\perp$ su entrambi i nastri, termina in stato finale

# MdT e MdT multi-nastro

MdT multinastro sembrerebbero più potenti delle MdT ordinarie

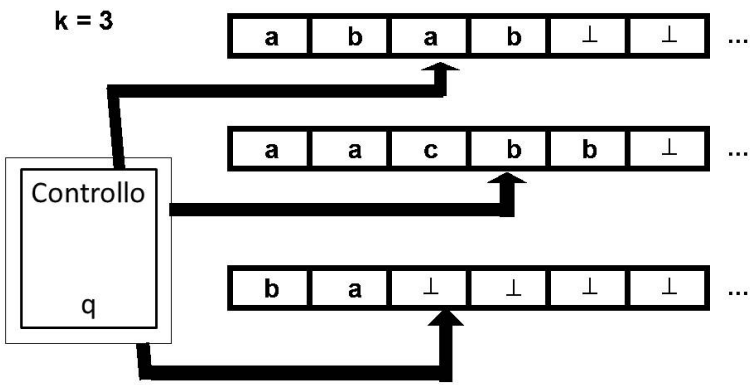
In realtà possiamo mostrare che le due varianti sono equivalenti

**Teorema**

MdT e MdT multi-nastro sono modelli equivalenti

In un verso: ovvio (MdT è anche MdT multi-nastro)

# Da MdT multi–nastro a MdT




Immaginiamo i $k$ nastri affiancati, ognuno con indicata la posizione della testina

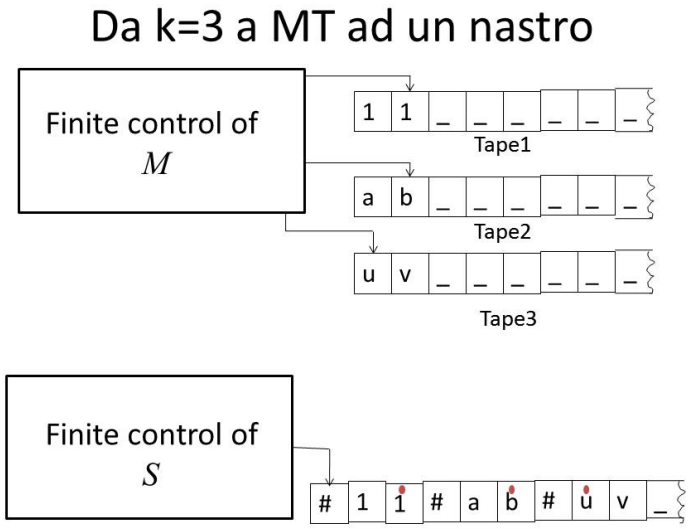
Dobbiamo codificare questa informazione su un solo nastro

# Da MdT multi-nastro a MdT

- Concatenare contenuto dei $k$ nastri, su k blocchi consecutivi separati da #
  - ogni blocco ha lunghezza variabile che dipende dal contenuto del nastro corrispondente
  - un elemento marcato nel blocco i-mo indica la posizione della testina i-ma

    es. testina S punta ad elemento del primo blocco e legge $\dot{\gamma}$: la testina del primo nastro è in questa posizione e legge $\gamma$;
  - Alfabeto esteso $\Gamma_2$, t.c. $\dot{a} \in \Gamma_2$ per ogni $a \in \Gamma$

# Da MdT multi-nastro a MdT



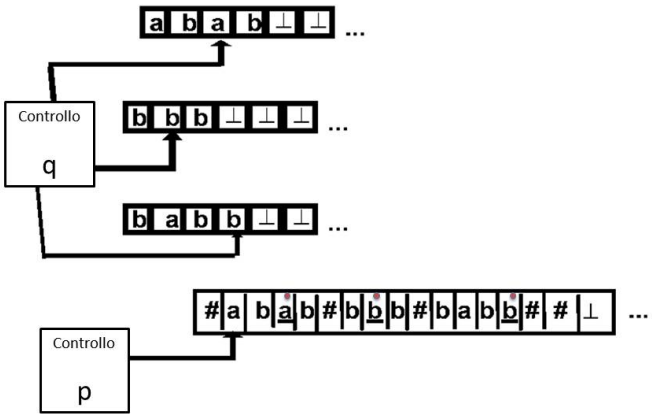

Configurazione iniziale di $S$: $\#\dot{a}_1 \ldots a_n \# \dot{\perp} \# \ldots \dot{\perp} \#$,

# Da MdT multi-nastro a MdT

- Per ogni istruzione di MdT multi-nastro $M$, la MdT $S$
  - Scorre i $k$ nastri e "raccoglie" informazioni sui $k$ simboli letti
  - Applica transizione, scorrendo i $k$ nastri e applicando su ciascuno scrittura e spostamento

# Es.

Da k=3 a MT ad un nastro



# Es.

# Es.

# Da MdT multi-nastro a MdT

Per ogni mossa del tipo $\delta(q, a_1, ..., a_k) = (s, b_1, ..., b_k, D_1, ..., D_k)$

- S scorre il nastro verso destra, fino al primo $\perp$
  memorizzando nello stato i simboli marcati sui singoli nastri;
  Memorizzazione: stato aggiuntivo per ogni possibile sequenza di $\leq k$ simboli di $\Gamma$
- la testina si riposiziona all'inizio del nastro,
- poi il nastro viene scorso di nuovo
  eseguendo su ogni sezione (cioé un nastro di M) le azioni che simulano quelle delle testine di M: scrittura e spostamento.

# Da MdT multi-nastro a MdT

- Durante il secondo passaggio S scrive su tutti i simboli "marcati" e
- sposta il marcatore alla nuova posizione della testina corrispondente di M

# Da MdT multi-nastro a MdT

Nota.
Se una testina di M muove su $\perp$ più a destra del suo nastro allora la testina virtuale (puntino) sul segmento corrispondente del nastro di S si sposta sul delimitatore $\#$

In questo caso, S deve spostare di una posizione tutto il suffisso del nastro a partire da $\dot{\#}$ fino all'ultimo $\#$ a destra.

Dopo, S scrive $\dot{\perp}$ nella prima posizione soggetta a shift (dove c'era $\dot{\#}$).

$\#a\dot{b}\#xyz\# \quad \rightarrow \#ab\dot{\#}xyz\# \quad \rightarrow \quad \#ab\dot{\perp}\#xyz\#$

# Da MdT multi-nastro a MdT

Per ogni istruzione di MdT multi-nastro $M$, la MdT $S$

1. Scorre i $k$ nastri e "raccoglie" informazioni sui $k$ simboli letti
2. Applica transizione, scorrendo i $k$ nastri e applicando su ciascuno scrittura e spostamento
3. Infine la MdT entra nello stato che ricorda il nuovo stato di S e riposiziona la testina all'inizio del nastro.

**Molti più stati e mosse, ma S simula M!**

# Macchina di Turing non deterministica

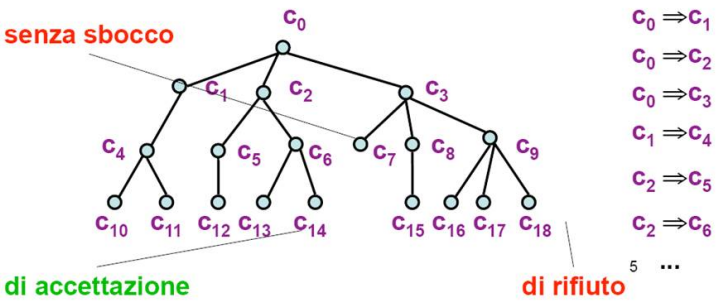
Una MdT non deterministica (NMdT ) è una macchina di Turing avente funzione di transizione

$$\delta : Q \times \Gamma \to P(Q \times \Gamma \times \{L, R\})$$

invece di $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

Analogamente alla differenza tra DFA e NFA

# Macchina di Turing non deterministica




Computazione di una NMdT: ben descritta da albero contenente ogni configurazione raggiungibile dalla iniziale ($c_0$) su dato input:
Se un cammino (da $c_0$) raggiunge lo stato accetta allora la NMdT

# MdT e NMdT

NMdT sembrerebbero più potenti delle MdT ordinarie

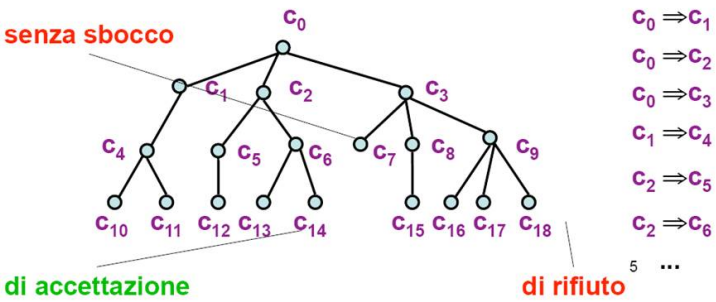
In realtà possiamo mostrare che le due varianti sono equivalenti

**Teorema**

MdT e NMdT sono modelli equivalenti

In un verso: ovvio MdT è anche NMdT

# da NMdT a MdT




Guardiamo alla computazione di una NMdT N come ad delle configurazioni raggiungibili da quella iniziale $c_0$ su input $w$

# da NMdT a MdT

Idea: per ogni input w,

- M deve eseguire tutte le possibili computazioni di N su $w$ e
- accettare sse almeno una raggiunge stato accetta

# da NMdT a MdT

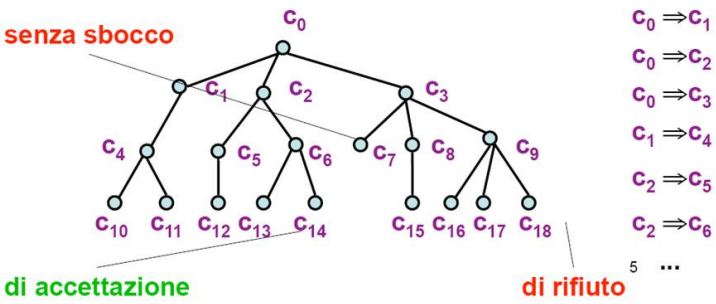
Alcune delle computazioni di N possono essere infinite
$\Rightarrow$ alcuni cammini sono infiniti
$\Rightarrow$ Se M esegue la simulazione e segue un cammino infinito, va in loop (anche se su altro cammino raggiungerebbe lo stato accetta)

# da NMdT a MdT

Alcune delle computazioni di N possono essere infinite
$\Rightarrow$ alcuni cammini sono infiniti
$\Rightarrow$ Se M esegue la simulazione e segue un cammino infinito, va in loop (anche se su altro cammino raggiungerebbe lo stato accetta)

Idea: eseguiamo tutte le simulazioni contemporaneamente!
Usando una BFS dell'albero delle computazioni

# da NMdT a MdT



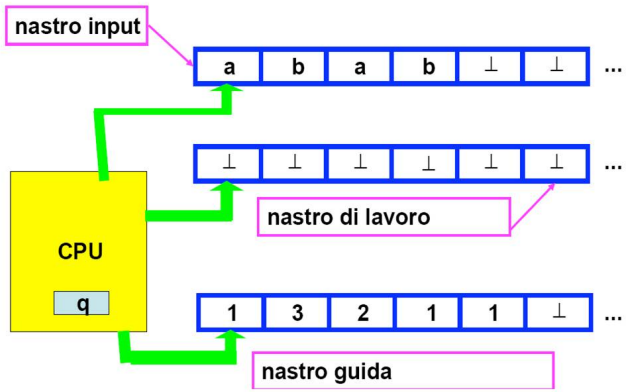

1. Esegui il primo passo di ogni computazione. Se almeno una accetta allora accetta.
2. Esegui il secondo passo di ogni computazione. Se almeno una accetta allora accetta.

   ...

i. Esegui il passo i-mo di ogni computazione. Se almeno una accetta allora accetta.

   ...

# da NMdT a MdT

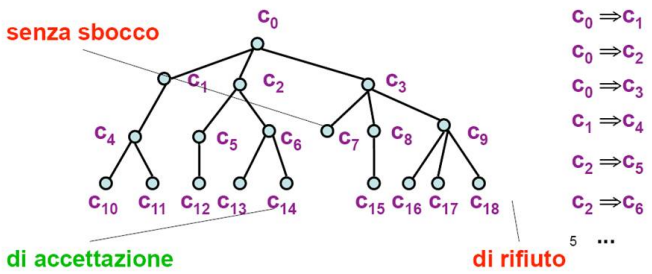



Simulazione fatta mediante MdT D a 3 nastri.
Inizialmente:
Nastro 1 contiene input di N, Nastri 2 e 3 vuoti

# da NMdT a MdT

- Il nastro 1 contiene sempre la stringa input e non viene mai modificato.
- Il nastro 2 contiene una copia del nastro di $N$ corrispondente ad una diramazione della sua computazione non deterministica.
- Il nastro 3 serve a tener traccia della posizione di $D$ nell'albero della computazione non deterministica di $N$.

# da NMdT a MdT




Se il nastro 3 contiene la stringa 221 significa che $D$ sta simulando la configurazione $C_{14}$ di $N$.

# da NMdT a MdT : Come è mantenuta la linea di computazione?

N non deterministica $\Rightarrow$ più possibili transizioni da stessa config.

- Per ogni configurazione di N, MdT D codifica tutte le possibili transizioni e le enumera
- Sia $COMP_i$ = prefisso lungo $i$ di una computazione di N
  Codifichiamo $COMP_i$ con stringa $b_1 \dots b_i$ di $i$ digit $b$-ari:
  $b$ = max numero transizioni possibili da qualsiasi config.
  - D parte con la configurazione iniziale
    $b_1$ = numero della transizione al primo passo di $COMP_i$ .
  - Per $2 \leq j \leq i$, $b_j$ è il numero della transazione attuale al passo $j$–mo di $COMP_i$

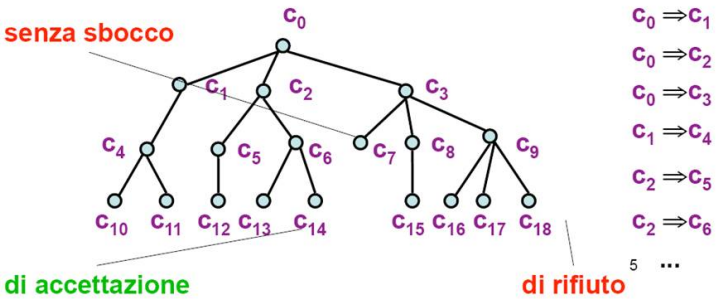
# da NMdT a MdT

Es. La stringa 421 codifica un prefisso di lunghezza 3 in cui

- al primo passo N esegue la **quarta** transizione nella lista delle transizioni possibili (da configurazione iniziale)
- al secondo passo N esegue la **seconda** transizione nella lista delle transizioni possibili (da config. raggiunta a passo 1)
- al terzo passo N esegue la **prima** transizione nella lista delle transizioni possibili (da conf. passo 2)

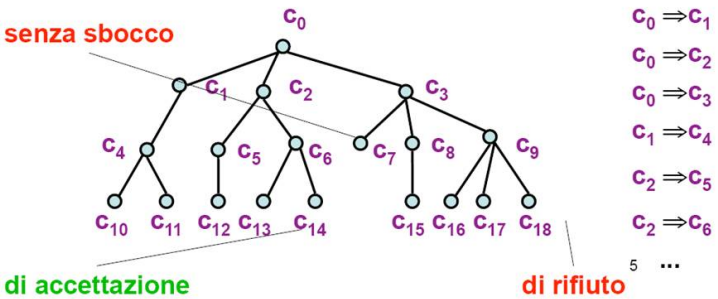
Nota: alcune configurazioni possono ammettere meno di b scelte, quindi non tutte le sequenze b-arie rappresentano un prefisso di computazione.

# da NMdT a MdT




- $b = 3$: sul nastro 3 ci saranno stringhe su $\{1, 2, 3\}$.
- Successione di stringhe sul nastro 3:
  $1, 2, 3, 11, 12, 13, 21, 22, 23, 31, 32, 33, 111, 112, 113, \ldots$

# da NMdT a MdT




- $b = 3$: sul nastro 3 ci saranno stringhe su $\{1, 2, 3\}$.
- Successione di stringhe sul nastro 3:
  $1, 2, 3, 11, 12, 13, 21, 22, 23, 31, 32, 33, 111, 112, 113, \ldots$

Le stringhe in rosso non corrispondono a un prefisso valido di computazione: la macchina $D$ le ignora.

# da NMdT a MdT

La simulazione di N con D risulta:

1. Scrivi 1 sul nastro 3
2. Copia input sul nastro 2

# da NMdT a MdT

La simulazione di N con D risulta:

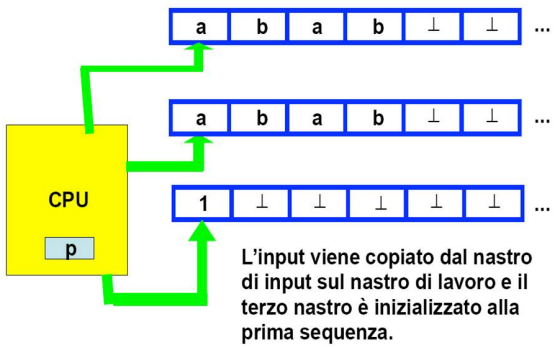
1. Scrivi 1 sul nastro 3
2. Copia input sul nastro 2
3. Se il nastro 3 contiene la codifica di un prefisso di una computazione di N, allora esegui la transizione corrispondente; se la computazione accetta, allora accetta

# da NMdT a MdT

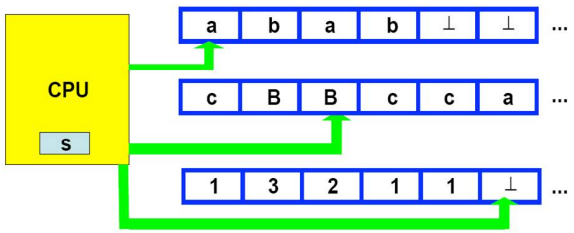
La simulazione di N con D risulta:

1. Scrivi 1 sul nastro 3
2. Copia input sul nastro 2
3. Se il nastro 3 contiene la codifica di un prefisso di una computazione di N, allora esegui la transizione corrispondente; se la computazione accetta, allora accetta
4. incrementa di 1 il numero b-ario sul nastro 3
5. Vai al passo 2.

# da NMdT a MdT


a b a b ⊥ ⊥ ...
a b a b ⊥ ⊥ ...
1 ⊥ ⊥ ⊥ ⊥ ⊥ ...
CPU
p
L'input viene copiato dal nastro di input sul nastro di lavoro e il terzo nastro è inizializzato alla prima sequenza.

# da NMdT a MdT



# Dettagli costruttivi?

Le costruzioni sono state delineate, non abbiamo dato dettagli per
conversione da MdT a k nastri a MdT (a 1 nastro)
conversione da NMdT a MdT
Perché non dare maggiori dettagli?

# Dettagli costruttivi?

Capire cosa fa una MdT data risulta difficile, specialmete se grande
Convincersi che una data MdT riconosce un linguaggio é difficile solo guardando la macchina

Se sappiamo che una MdT puó essere costruita, non è necessario costruirla in dettaglio
Diamo meno dettagli, ma diciamo come dovrebbe funzionare!

# Dettagli costruttivi?

Capire cosa fa una MdT data risulta difficile, specialmete se grande
Convincersi che una data MdT riconosce un linguaggio é difficile solo guardando la macchina

Se sappiamo che una MdT puó essere costruita, non è necessario costruirla in dettaglio
Diamo meno dettagli, ma diciamo come dovrebbe funzionare!
Tranne che per divertimento!

# Nozione di Computabilità

Abbiamo visto alcune variazioni di M e mostrato equivalenza
Esistono varie altre variazioni – tutte equivalenti

Anche altri modelli computazionali sono stati mostrati equivalenti a MdT

# Tesi di Church–Turing

Formalizzazioni di nozione di computabilità:

- Alonzo Church (matematico americano) creò un metodo per definire le funzioni computabili, $\lambda$*–calcolo*
- Alan Turing creò la *Macchina di Turing*
- Church, Kleene e Rosser diedero la definizione formale di una classe di funzioni il cui valore può essere ottenuto mediante *ricorsione*.

Si è dimostrtato che sono equivalenti

# Tesi di Church–Turing

Formalizzazioni di nozione di computabilità:

- Alonzo Church (matematico americano) creò un metodo per definire le funzioni computabili, $\lambda$–*calcolo*
- Alan Turing creò la *Macchina di Turing*
- Church, Kleene e Rosser diedero la definizione formale di una classe di funzioni il cui valore può essere ottenuto mediante *ricorsione*.

Si è dimostrtato che sono equivalenti

Tesi di Church–Turing: *"Se esiste un algoritmo per eseguire un calcolo allora questo calcolo può essere eseguito da una MdT (o variante equivalente)"*