Macchine di Turing

- Macchine di Turing
- Linguaggi Ricorsivi e Ricorsivamente Enumerabili

(basato su lucidi di Jeffrey Ullman disponibili su http://infolab.stanford.edu/~ullman/ialc.html)

Tesi di Turing-Church

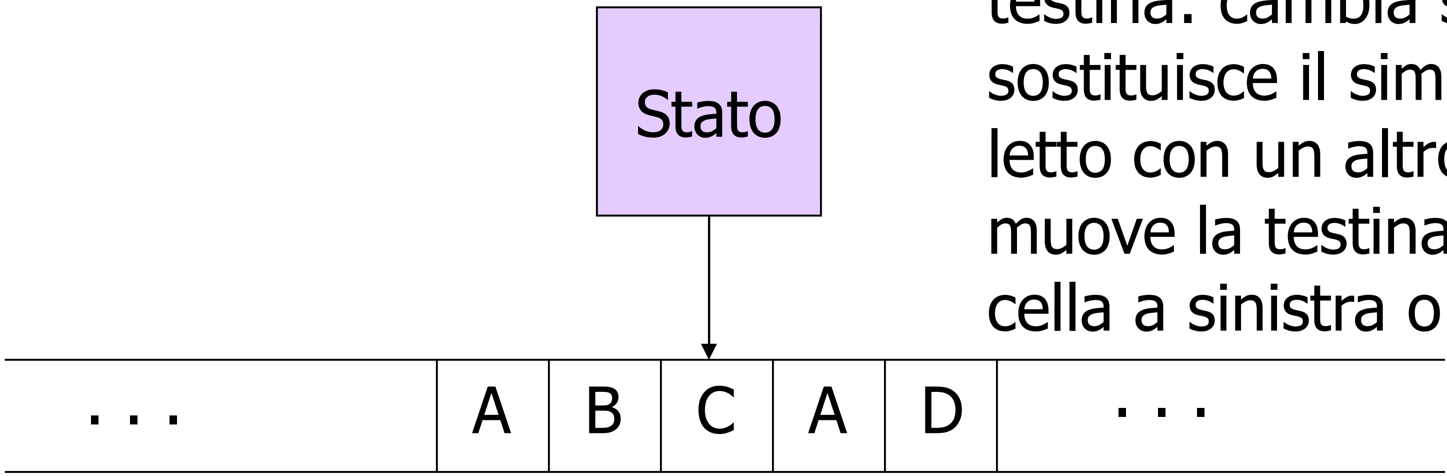
- Turing studiò i meccanismi basilari del calcolo automatico, elaborò un suo modello di calcolo (le Macchine di Turing) ed espresse il proprio pensiero nel seguente modo:
  - "SE UN PROBLEMA E' INTUITIVAMENTE CALCOLABILE (cioè risolvibile eseguendo un una procedura) ALLORA ESISTE UNA **TURING MACHINE** CHE LO RISOLVE"
- Vediamo cosa sono queste Turing Machine…

Definizione formale di Macchina di Turing deterministica

- Una TM è una 7-pla $(Q,\Sigma,\Gamma,\delta,q_0,B,F)$ con:
  - Un insieme **finito** di **stati** Q
  - Un **alfabeto di input** Σ
  - Un **alfabeto del nastro** Γ (contiene Σ)
  - Una funzione di **transizione** δ
  - Uno **stato iniziale** $q_0 \in Q$
  - Un simbolo **blank** $B \in \Gamma - \Sigma$
    - Tutto il nastro, meno l'input, all'inizio è blank
  - Un insieme di **stati finali** $F \subseteq Q$

Macchine di Turing

Azione:
funzione dello **stato** e del **simbolo** sotto la testina: cambia stato, sostituisce il simbolo letto con un altro, e muove la testina di una cella a sinistra o a destra



Nastro infinito con celle che contengono simboli appartenenti ad un **alfabeto finito** Γ

Convenzioni

- a, b, … sono simboli in input
- …, X, Y, Z sono simboli del nastro

- …, w, x, y, z sono stringhe di simboli in input
- α, β,… sono stringhe di simboli del nastro

La funzione di transizione

- Dati due argomenti:
  - Uno stato q in Q
  - Un simbolo Z del nastro appartenente a Γ
- δ(q, Z) è indefinito oppure è **una** tripla dalla forma (p, Y, D). funzione parziale deterministica
  - p è uno stato
  - Y è un nuovo simbolo del nastro
  - D è una **direzione**, L o R

Azioni della TM

- Se $\delta(q, Z) = (p, Y, D)$ allora, la TM nello stato q, che legge Z sul nastro:
  - Cambia lo stato in p
  - Rimpiazza Z con Y sul nastro
  - Muove la testina di una cella lungo la direzione D
    - Se D è L muove a sinistra
    - Se D è R muove a destra

Esempio: Macchina di Turing

- Si consideri una TM che legge la stringa in input in attesa di trovare un 1
  - la testina si trova inizialmente sul primo carattere dell'input
- Se ne trova uno, lo cambia in 0, entra in uno stato finale f, e si ferma
- Se raggiunge un blank, lo cambia in 1, e si sposta a sinistra

Esempio: Macchina di Turing (2)

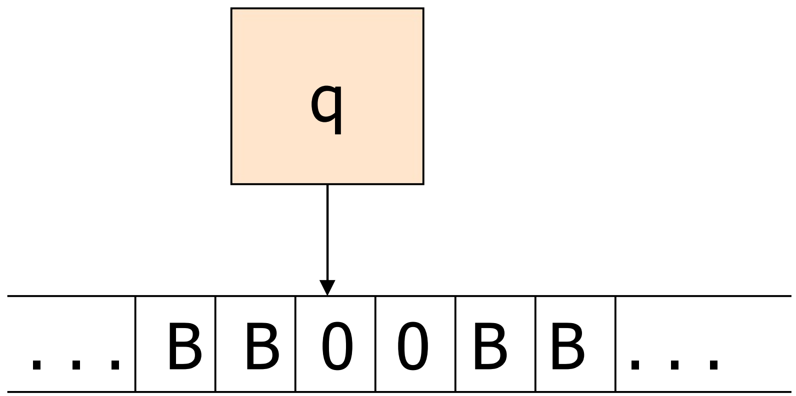
- Stati = {q (stato iniziale), f (finale)}
- Simboli in input = {0, 1}
- Simboli del nastro = {0, 1, B}
- $\delta(q, 0) = (q, 0, R)$
- $\delta(q, 1) = (f, 0, R)$
- $\delta(q, B) = (q, 1, L)$

Simulazione di TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

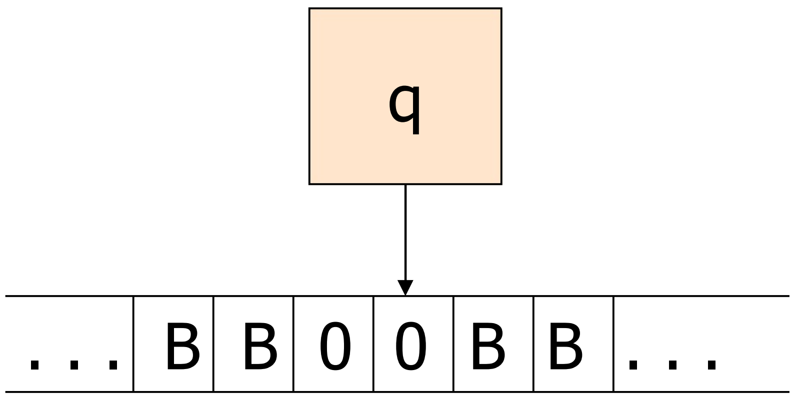


Simulazione di TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

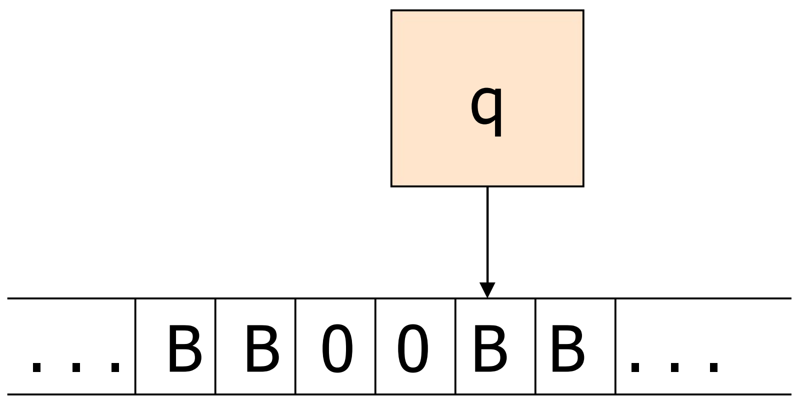


Simulazione di TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

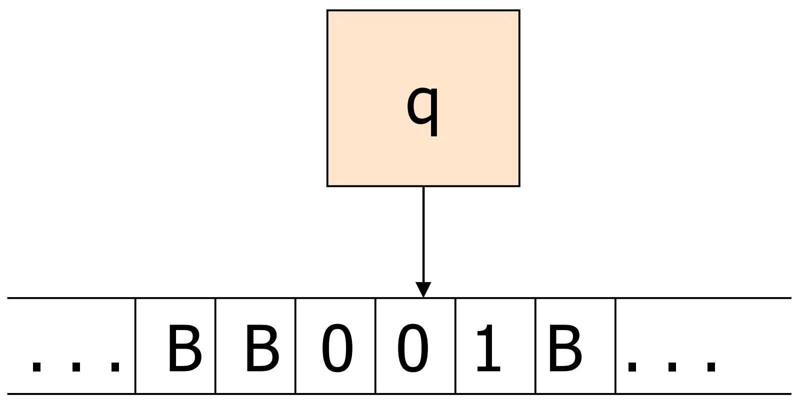


Simulazione di TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

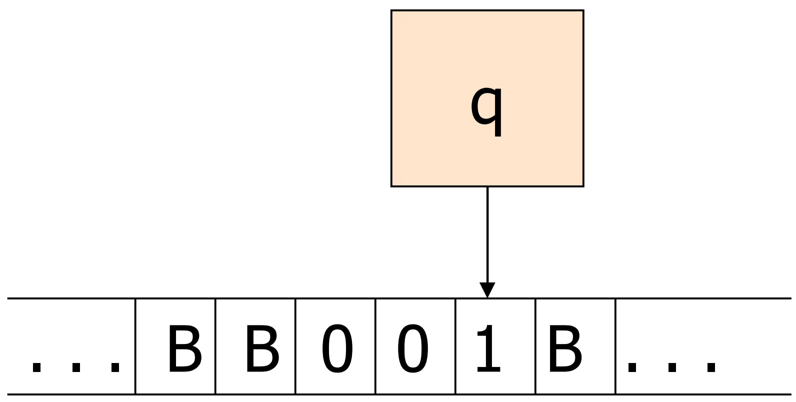


Simulazione di TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$

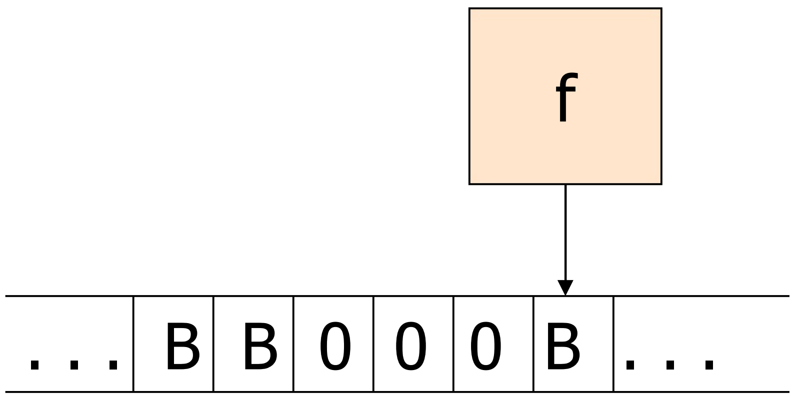


Simulazione di TM

$\delta(q, 0) = (q, 0, R)$

$\delta(q, 1) = (f, 0, R)$

$\delta(q, B) = (q, 1, L)$



La TM è in uno stato **finale**: **si ferma** e accetta

Descrizione Istantanea (ID) di una Turing Machine

- Inizialmente, una TM ha un nastro che consiste di una **stringa di simboli di input**, attorniata da una **infinità di blank** in entrambe le direzioni
- La TM è nello **stato iniziale**, e la testina è sul **simbolo in input più a sinistra**

ID di una TM – (2)

- In generale, una ID è rappresentata tramite una **stringa** $\alpha\mathbf{q}\beta$, dove:
  - $\alpha\beta$ è il contenuto del nastro
    - a sinistra di $\alpha\beta$ e a destra di $\alpha\beta$ si assume che il nastro contenga tutte B
  - **q** è lo stato della TM
  - la testina è posizionata sul primo carattere di β
    - se β è vuota sotto la testina c'è B

ID di una TM– (3)

- Come per i PDA usiamo i simboli $\vdash$ e $\vdash^*$ per rappresentare i **passaggi di ID**: rispettivamente, "diventa in una mossa" e "diventa in zero o più mosse"
- Esempio: le mosse dell'esempio precedente sono: $q00 \vdash 0q0 \vdash 00q \vdash 0q01 \vdash 00q1 \vdash 000f$

Definizione Formale di Mossa

1. Se $\delta(q, Z) = (p, Y, R)$, allora
   - $\alpha q Z \beta \vdash \alpha Y p \beta$
   - se Z è B, allora anche $\alpha q \vdash \alpha Y p$
2. Se $\delta(q, Z) = (p, Y, L)$, allora
   - $\alpha q Z \beta \vdash \gamma p X Y \beta$ con $\alpha=\gamma X$, oppure $\alpha=\gamma=\varepsilon$ e $X=B$
   - se Z è B, allora anche $\alpha q \vdash \gamma p X Y$ con stessa cond.

Linguaggio di una TM

- Una TM M definisce un linguaggio per stato finale, come al solito:
  - $L(M) = \{w \mid q_0 w \vdash^* \alpha q \beta$, con q finale$\}$

Nota. Per **w non in L(M)** la TM M può:

- o **bloccarsi** prima o poi in uno stato non finale
- o **continuare per sempre** a computare senza mai raggiungere uno stato finale

Linguaggi Ricorsivamente Enumerabili

- Tali linguaggi vengono detti: **linguaggi ricorsivamente enumerabili**
  - Perché? Il termine venne inventato prima delle TM e fa riferimento ad una diversa nozione di calcolo basato su funzioni

Nota. La **computazione** della TM:

- **termina** per input **w in L(M)**
- **può non terminare** per input **w non in L(M)**

Problema di stabilire w in L(M): **semi-decidibile**

Linguaggi Ricorsivi

- Un **algoritmo** è una TM che **sicuramente termina** (cioè termina per ogni input)
- Se L = L(M) per **una qualche** TM M che è un algoritmo, diciamo che L è un **linguaggio ricorsivo**
  - Perché? Vedi la risposta del lucido precedente....

Esempio: Linguaggi Ricorsivi

- Ogni CFL è un linguaggio ricorsivo.
  - Basta usare l'algoritmo che, data w, prova tutti gli alberi sintattici con |w| foglie
- …in altri termini, ogni linguaggio per cui sia possibile in modo automatico **verificare se una data stringa vi appartiene oppure no** è ricorsivo
  - Problema di stabilire w in L(M): **decidibile**

Altro su Macchine di Turing

- "Trucchi" di programmazione
- Restrizioni
- Estensioni
- Proprietà di chiusura

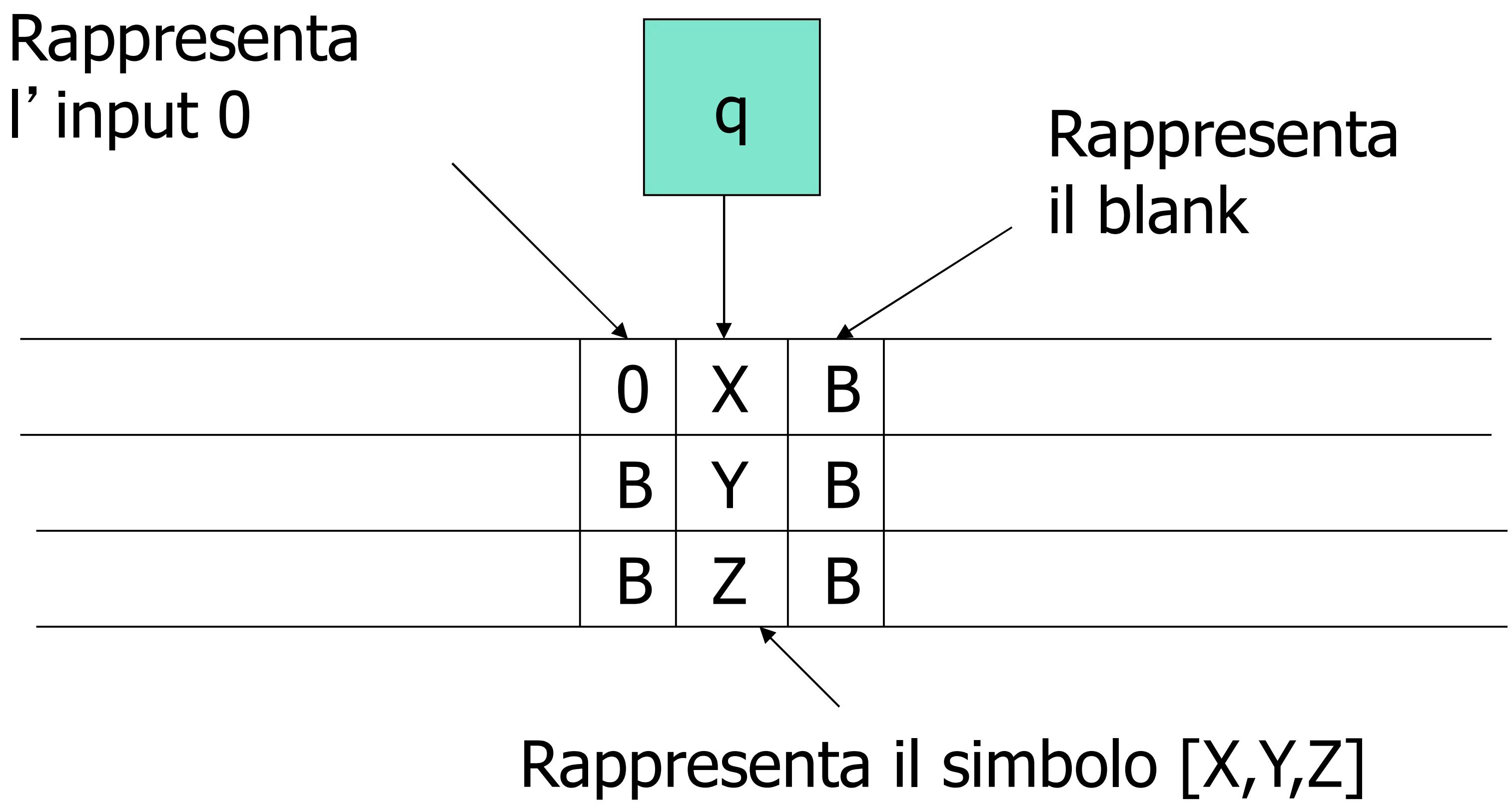
Osservazione

- All'inizio, le TM non sembrano molto potenti
  - Ma sono veramente in grado di calcolare tutto quello che i calcolatori calcolano?
- Vedremo "**trucchi**" **di programmazione** ed **estensioni di TM** per convincerci che queste possono simulare calcolatori reali

Trucco di programmazione: Tracce multiple

- Consideriamo simboli del nastro che sono **vettori di k elementi**
- Ogni elemento viene scelto all'interno di un alfabeto finito
- E' come se il nastro avesse k tracce
- Simboli in **input**: hanno **un solo elemento** del vettore diverso da blank

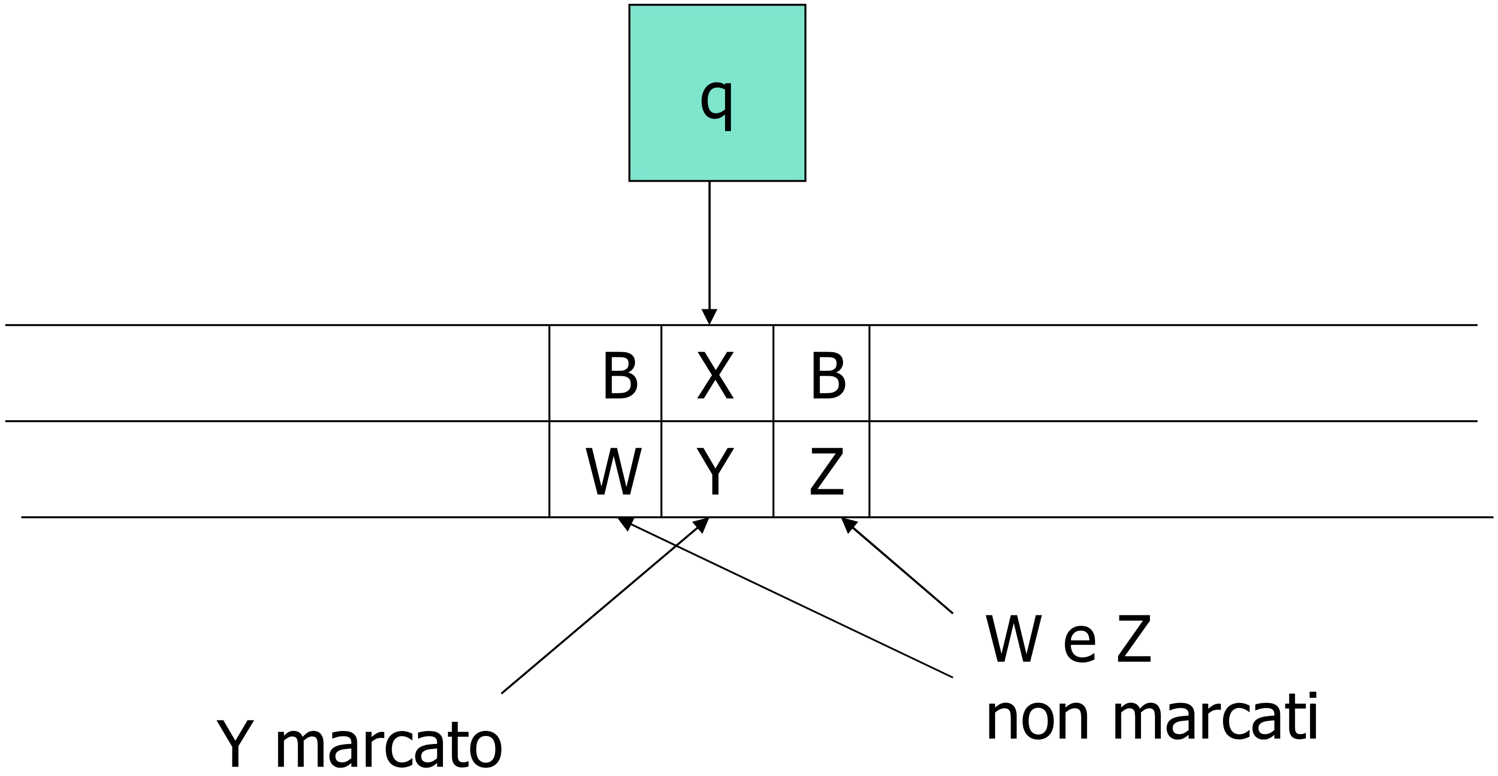
Esempio: tracce multiple



Trucco di programmazione: Marcature

- Una traccia potrebbe essere usata per **marcare** alcune posizioni
- Quasi tutte le celle contengono il blank in questa traccia, ma alcune contengono un **simbolo speciale**

Esempio: Marcatura



Trucco di programmazione: Stati con “piccole memorie”

- Anche lo **stato** può essere un **vettore**
- Il primo elemento è lo “**stato di controllo**”
- Altri elementi contengono simboli presi da un alfabeto finito

Esempio: uso di questi “trucchi”

- Questa TM semplicemente **copia l’input** (stringa binaria) **infinite volte**
- Stati di controllo:
  - q: marca la posizione e memorizza il simbolo letto
  - p: va a destra, continuando a ricordarsi il simbolo letto, finché non si trova un blank, dove depositerà il simbolo
  - r: va a sinistra fino alla marcatura

Esempio – (2)

- Gli **stati** hanno forma [x,Y], dove:
  - x è q, p, oppure r
  - Y è 0, 1, oppure B
    - 0 e 1 usati solo in caso x sia p
- I **simboli** del nastro hanno forma [U,V]
  - U è X (la "marcatura") oppure B
  - V è 0, 1 (i simboli di input) oppure B
    - [B,B] è il blank della TM; [B,0] e [B,1] sono gli input

La funzione di transizione

- Convenzione: *a* e *b* sono simboli dell'alfabeto di input (cioè 0 oppure 1)
- $\delta([q,B], [B,a]) = ([p,a], [X,a], R)$
  - Nello stato q, copia il simbolo in input sotto la testina (cioè *a*) nello stato
  - Marca la posizione letta
  - Va allo stato di controllo *p* e muove a destra

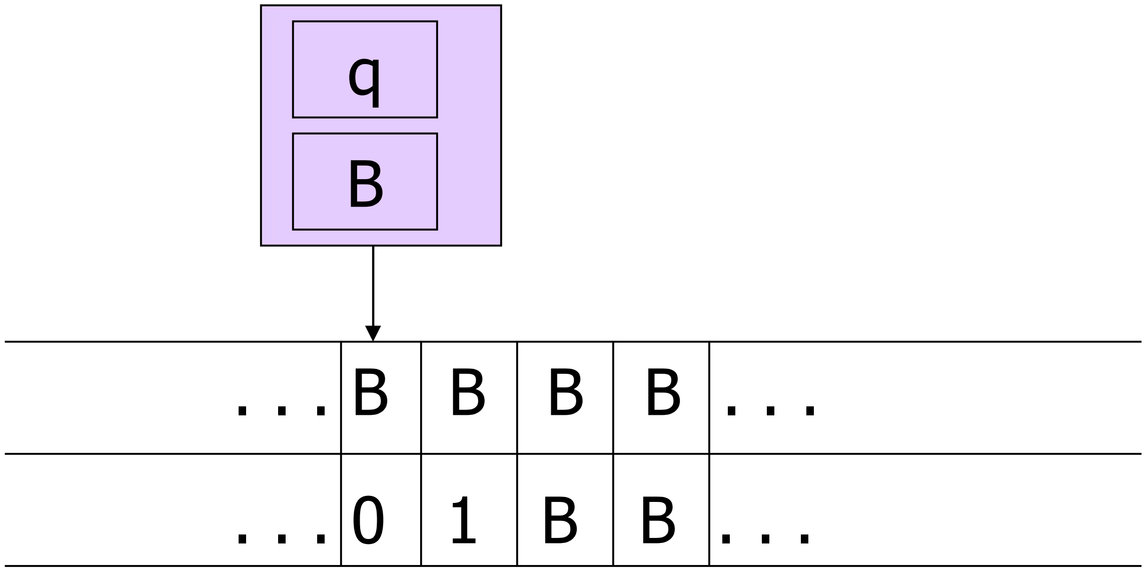
Funzione di transizione – (2)

- $\delta([p,a], [B,b]) = ([p,a], [B,b], R)$
  - Nello stato *p*, va a destra, in attesa di trovare un simbolo blank (non un blank nella sola traccia di marcatura)
- $\delta([p,a], [B,B]) = ([r,B], [B,a], L)$.
  - Quando trova B, lo rimpiazza con il simbolo memorizzato nello stato (cioè *a*)
  - Entra nello stato *r* e va a sinistra

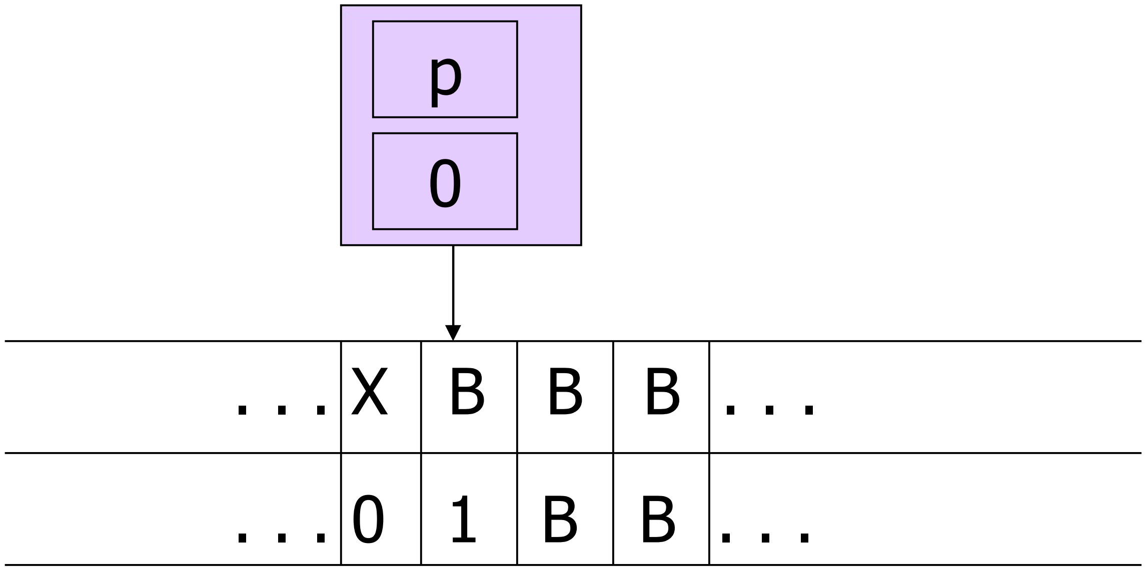
Funzione di transizione – (3)

- $\delta([r,B], [B,a]) = ([r,B], [B,a], L)$
  - Nello stato *r*, va a sinistra, alla ricerca della marcatura
- $\delta([r,B], [X,a]) = ([q,B], [B,a], R)$
  - Quando la marcatura viene trovata, entra nello stato *q* e va a destra
  - Rimuove la marcatura
  - Lo stato *q* inserirà una marcatura e si ripeterà il ciclo

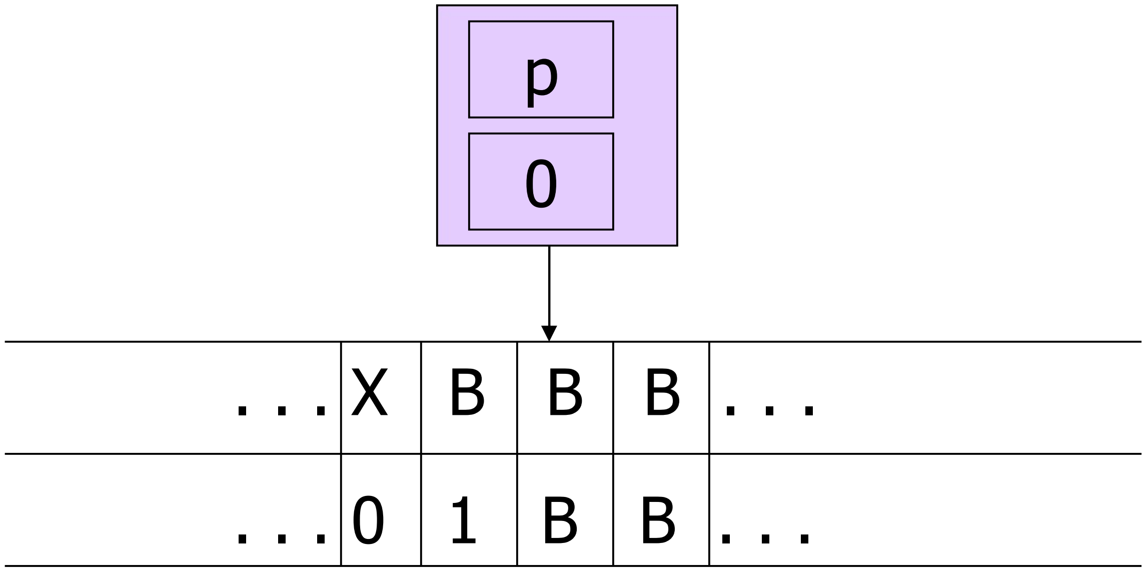
Simulazione della TM



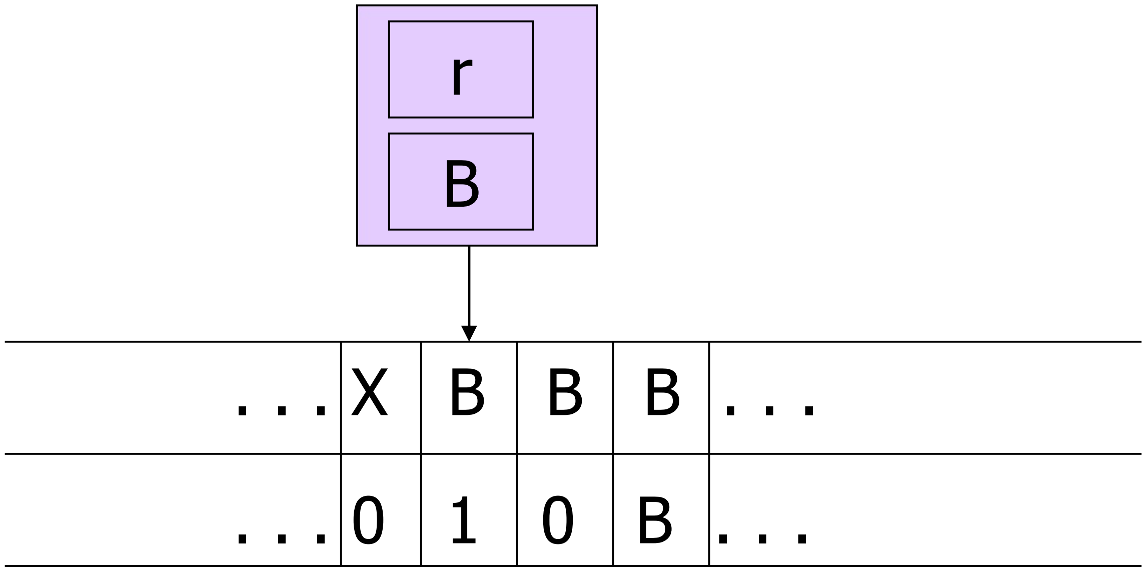
Simulazione della TM



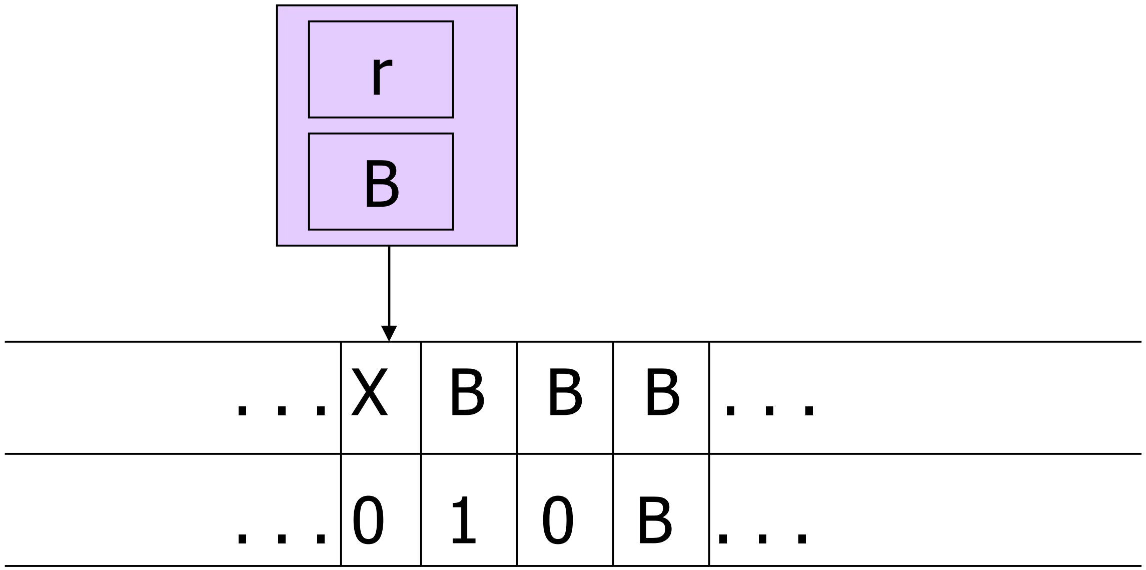
Simulazione della TM



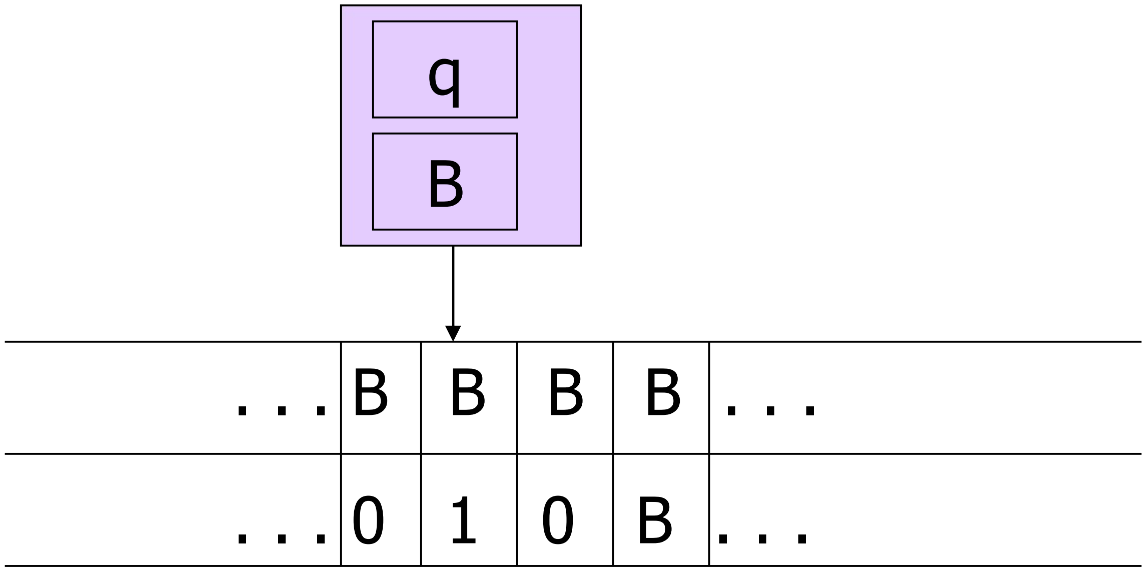
Simulazione della TM



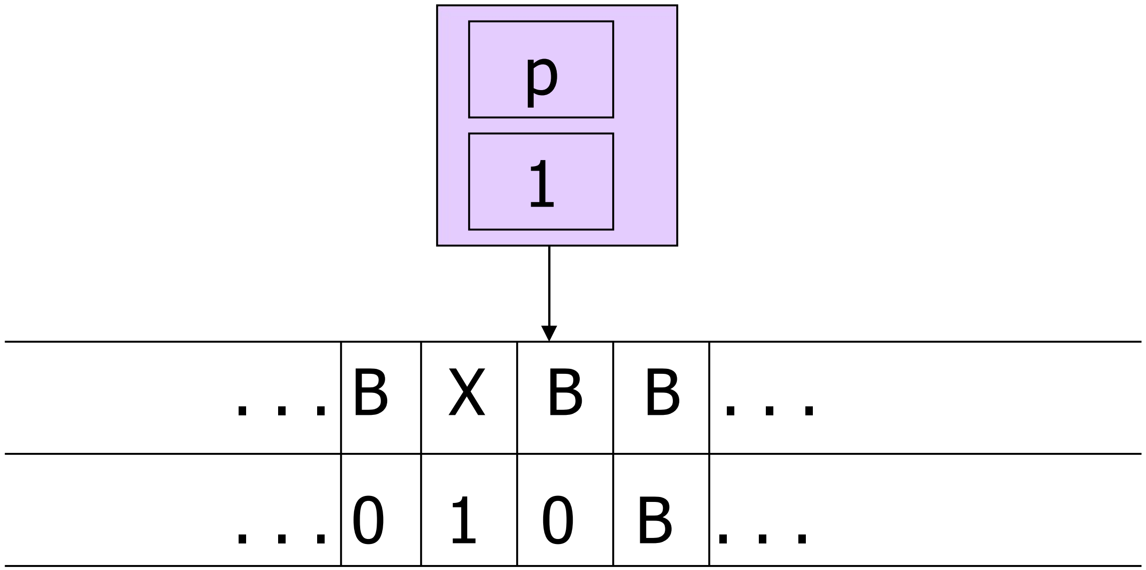
Simulazione della TM



Simulazione della TM



Simulazione della TM

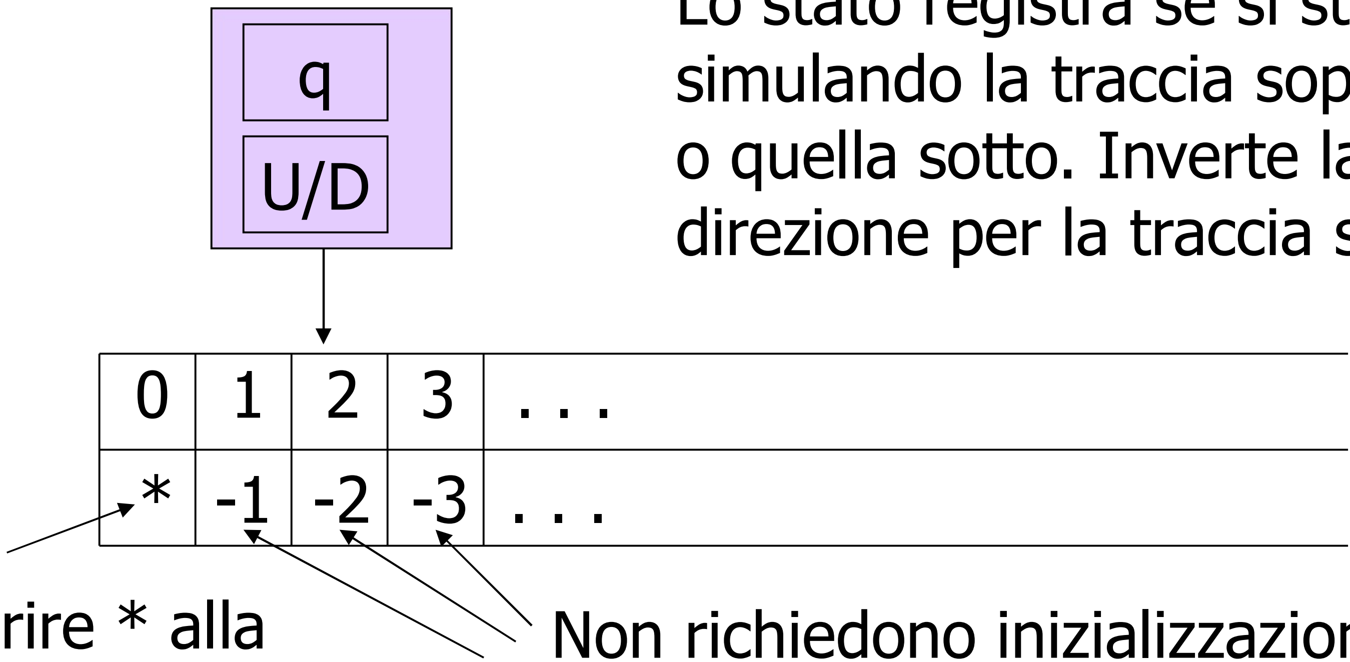


Nastro semi-infinito

- Possiamo assumere che la TM non si muova **mai a sinistra della posizione iniziale**
- Indichiamo con 0 tale posizione;
  con 1,2,… le posizioni a destra;
  con -1,-2,… le posizioni a sinistra
- La nuova TM usa **due tracce**:
  - Sopra contiene i simboli in posizione 0,1,2, …
  - Sotto i simboli in posizione –1,–2,…

Simulazione del nastro infinito con quello semi-infinito

Lo stato registra se si sta simulando la traccia sopra o quella sotto. Inverte la direzione per la traccia sotto



Inserire * alla prima mossa

Non richiedono inizializzazione: sono inizialmente B

Ulteriori restrizioni

- **Due stack** possono simulare un nastro
  - Uno contiene le **posizioni alla sinistra della testina**; l'altro le **posizioni alla destra**
- **Conclusione**: in sostanza, la differenza fra PDA e TM, è che **la TM ha due pile** invece che una sola!

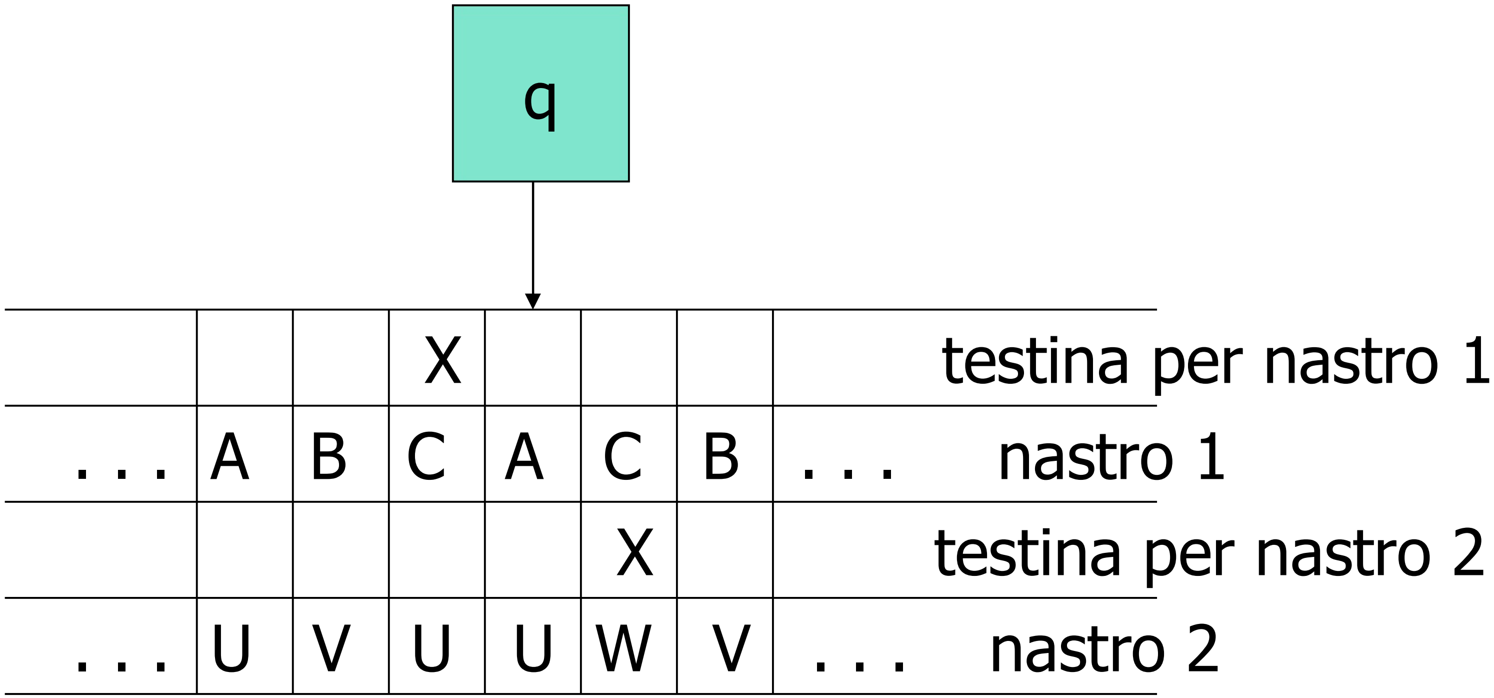
Turing Machine Multinastro

- Permettiamo alle TM di avere **k nastri** (per un qualsiasi k) e **k testine**
- Le mosse della TM dipendono dallo stato e dal simbolo sotto la testina di ogni nastro
- In una mossa, la TM può cambiare stato, scrivere simboli sotto le testine, e muovere **ogni testina indipendentemente**

Simulare k nastri con uno solo

- Usare **2k tracce**
- Ogni **nastro** è rappresentato da una **traccia**
- La posizione della corrispondente testina è **marcata** in una **traccia aggiuntiva**
- Ci si muove avanti/indietro simulando i passi di ogni singola testina
  - Una **passata** simula una **mossa**

Immagine della simulazione



TM Nondeterministiche

- Permettere alle TM di avere una scelta fra varie possibili mosse
  - Ogni **scelta** è una **possibile tripla**
    nuovo stato - simbolo da scrivere - direzione
    (le TM deterministiche hanno al più una tripla)
- La TM accetta l'input se esiste una **possibile sequenza di scelte** che porta ad uno stato accettante

Simulare una NTM con una DTM

- La DTM mantiene sul proprio nastro una **coda di ID** della NTM
- Una **seconda traccia** è usata per
  - **Marcare** la **ID in testa alla coda**
  - Generare, una alla volta, le **successive ID di quella in testa alla coda**, per poi copiarle in fondo alla coda

Immagine del nastro della DTM

Testa della coda

Posizione in cui inserirere le nuove ID, successive a ID_k

X Nuove ID

ID_0 # ID_1 # ... # ID_k # ID_{k+1} ... # ID_n # Nuove ID

Fine della coda

Operazioni della DTM

- La DTM cerca la ID in testa alla coda
- Cerca lo stato della ID così da poter determinare le possibili mosse
- Se ci sono *m* possibili mosse, **crea *m* nuove ID**, una per ogni mossa, e le copia in fondo alla coda

Operazioni della DTM – (2)

- Dopo che tutte le ID sono state messe in fondo alla coda, si **sposta in avanti la marcatura in testa alla coda**
- Tuttavia, se si crea una **ID con stato accettante**, la DTM si ferma in uno stato accettante

Quali vantaggi dalle estensioni?

- Ecco i vantaggi…
- Quando discutiamo di **costruzioni** che lavorano a partire da una TM, possiamo assumere che tale TM sia semplice
  - Cioè una TM deterministica con un nastro semi-infinito
- Ma la TM che costruiamo può avere più nastri, essere nondeterministica, …

Proprietà di chiusura per linguaggi Ricorsivi e RE

- Entrambi chiusi rispetto a unione, intersezione, concatenazione, stella di Kleene e reverse
- Ricorsivi chiusi rispetto a complemento e differenza

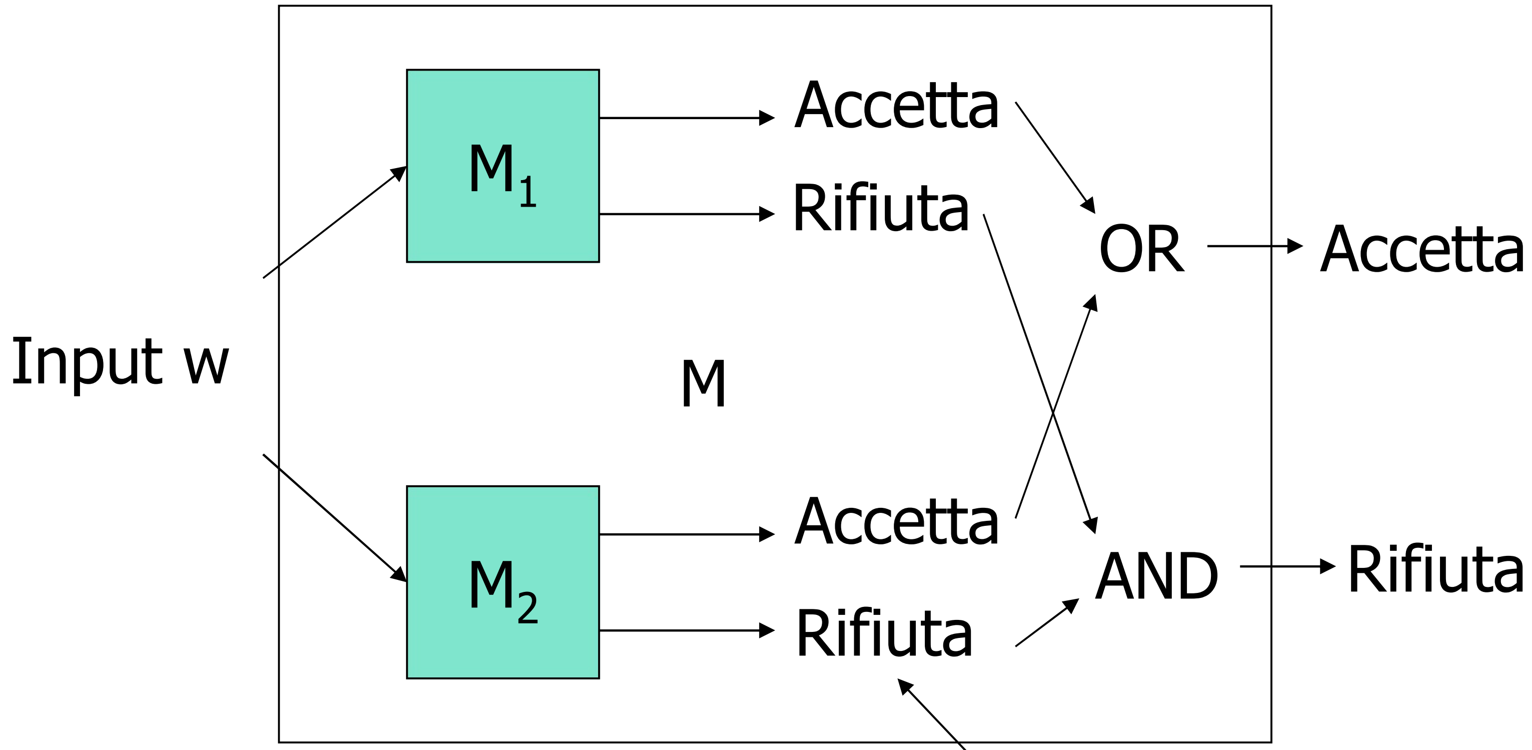
Unione

- Sia $L_1 = L(M_1)$ e $L_2 = L(M_2)$
- Assumiamo che M_1 e M_2 siano TM con un solo nastro semi-infinito
- Costruiamo una TM con 2 nastri che prima copia l'input sul secondo nastro e poi simula "in parallelo" le due TM (M_1 sul primo nastro, M_2 sul secondo nastro)

Unione – (2)

- Linguaggi **Ricorsivi**: se M_1 e M_2 sono entrambi algoritmi, allora M sicuramente termina entrambe le simulazioni (è quindi un algoritmo)
  - Accetta se almeno una accetta
- Linguaggi **RE**: accetta se almeno una accetta, ma potrebbe succedere che una o entrambe le TM non terminino mai (né si bloccano, né accettano)

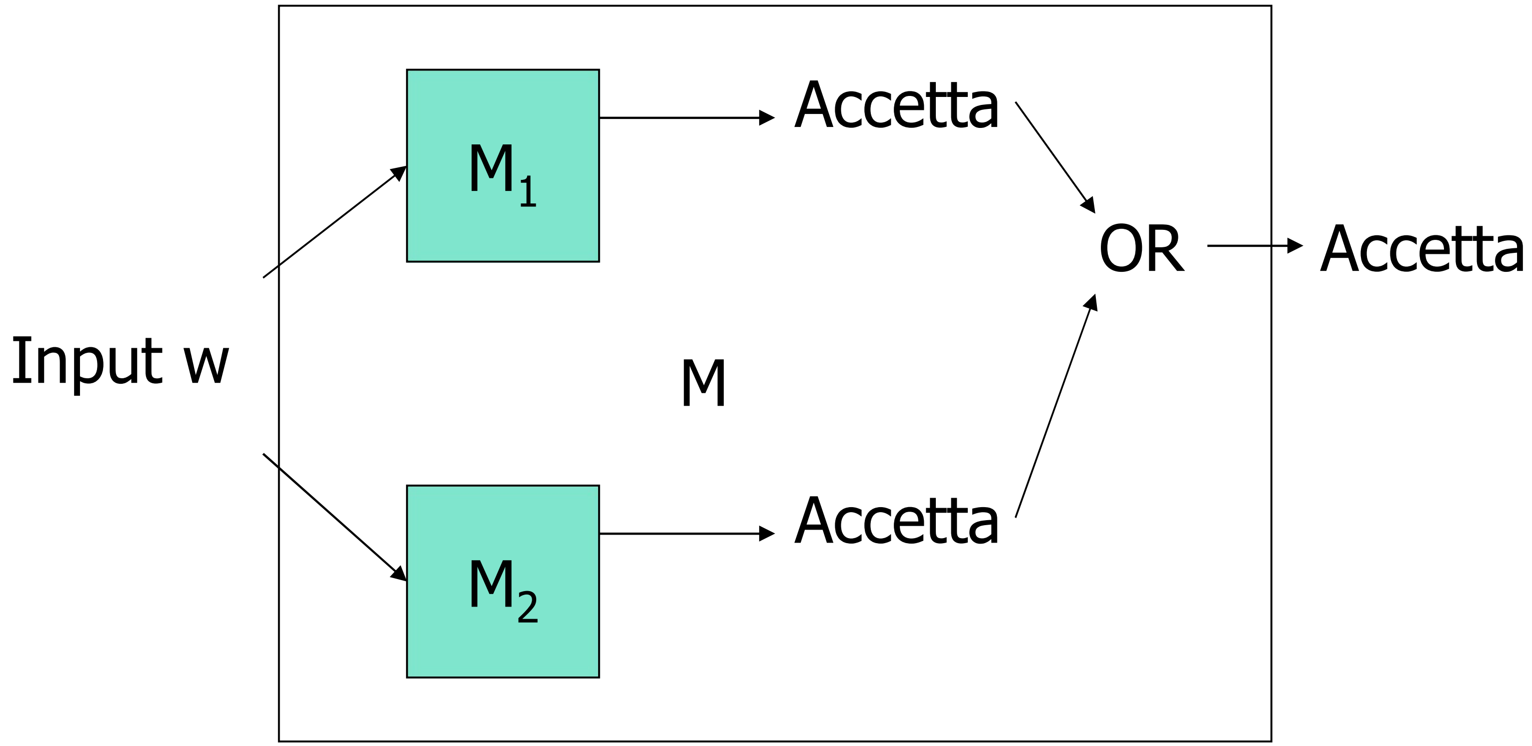
Immagine per Unione / Ricorsivi



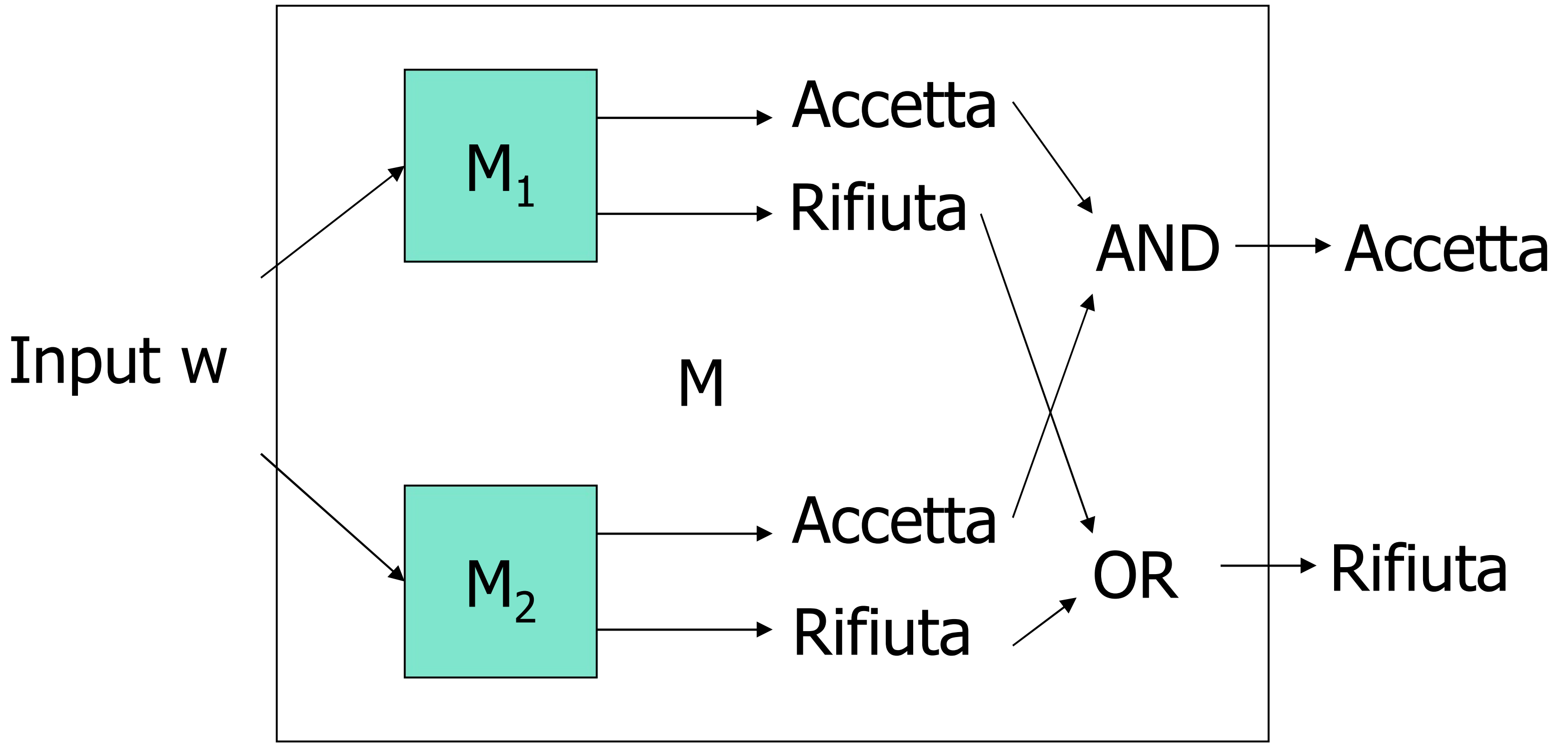
AND: devo attendere entrambi gli input
OR: me ne basta uno

Ricorda: blocca senza accettare

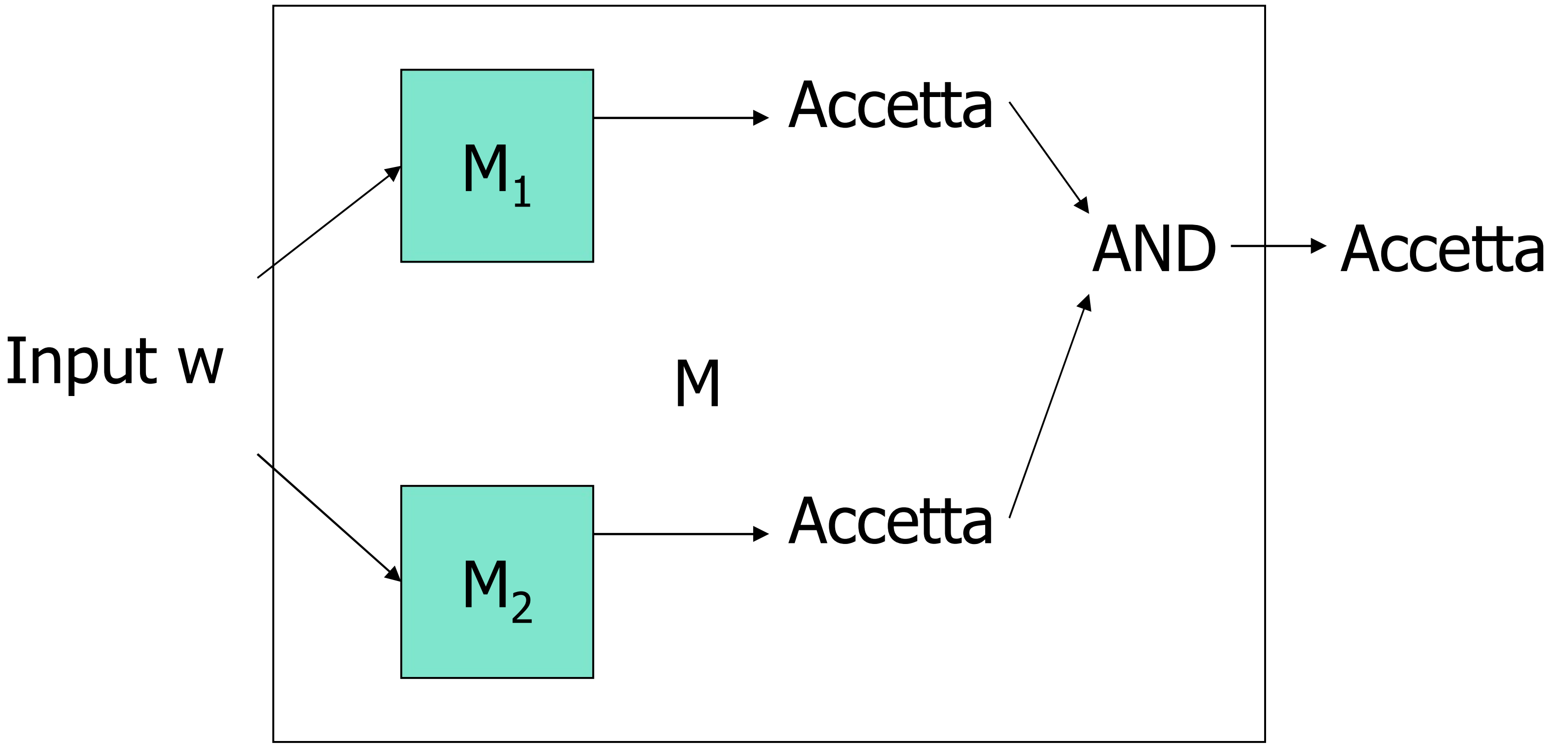
Immagine per Unione / RE



Intersezione/Ricorsivi – Stessa Idea



Intersezione RE



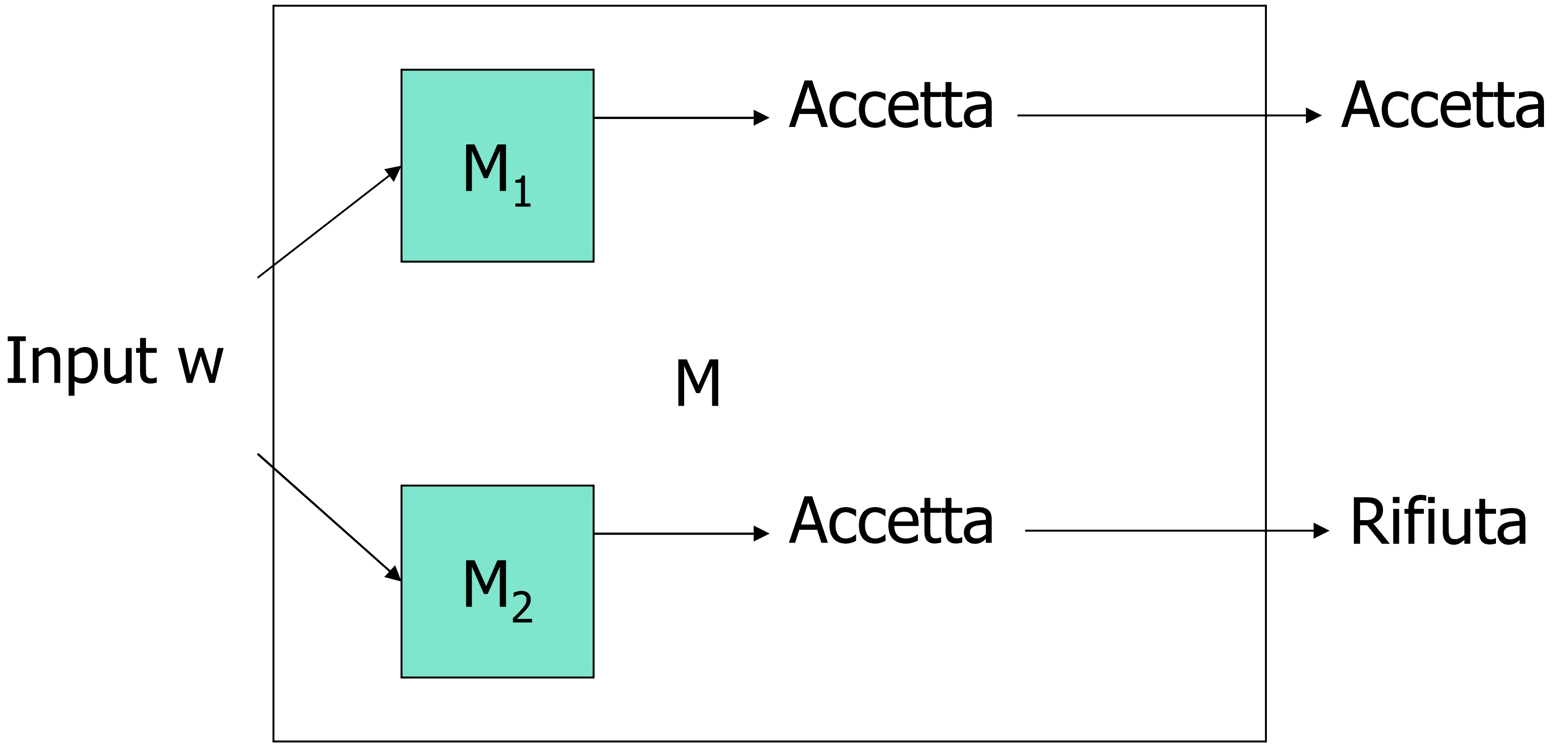
Complemento

- Linguaggi **Ricorsivi**: la TM M si fermerà prima o poi
  - Accetta se M non accetta
- Linguaggi **RE**: non si può fare la medesima costruzione!
  - per una stringa che non è in L(M), M potrebbe non terminare mai (non si riesce quindi a farla andare in uno stato finale)

Complemento di RE

- In generale il **complemento** di un linguaggio L **RE** può essere **non RE**
- **Teorema di Post**
  Se L è **RE** ed il complemento di L è **RE** allora si ha che L è **Ricorsivo**
  - cioè, se **L** riconosciuto da una TM $\mathbf{M_1}$ e **complemento di L** riconosciuto da una TM $\mathbf{M_2}$ allora **L** è **Ricorsivo**

Linguaggio e suo complemento RE



Differenza

- Differenza tra L_1 ed L_2 è **intersezione** di L_1 con **complemento** di L_2
- Linguaggi **Ricorsivi**: chiusi rispetto a entrambe le operazioni
- Linguaggi **RE**: se fosse chiuso rispetto a differenza lo sarebbe anche rispetto a complemento (basta prendere come L_1 l'insieme di tutte le stringhe)

Concatenazione / RE

- Siano $L_1=L(M_1)$ e $L_2=L(M_2)$
- Assumiamo che M_1 e M_2 siano TM con un solo nastro semi-infinito
- Costruiamo una TM a **due nastri nondeterministica** M:
  - Prova una suddivisione dell'input w = xy
  - Muove y sul secondo nastro
  - Simula M_1 su x, M_2 su y
  - Accetta se entrambe accettano

Concatenazione / Ricorsivi

- Si **prova** sistematicamente **ogni suddi-visione** (sono finite) dell'input w = xy
- M_1 e M_2 termineranno per ogni x e y
- Si accetta se entrambe accettano per una qualche suddivisione
- Rifiuta se tutte le suddivisioni falliscono

Nota: **non** è sufficiente usare **nondeterminismo** per mostrare che **termino sempre**!

Stella di Kleene

- Le stesse idee continuano a funzionare
- **RE**: scommettere (usando il nondeterminismo) su una qualche suddivisione, accettare se M_1 accetta tutti i singoli pezzi
- **Ricorsivi**: sistematicamente si provano tutte le possibili suddivisioni (sono in numero finito) per cercarne uno in cui tutti i pezzi vengono riconosciuti

Reverse

- All'inizio si inverte l'input (come?)
- Poi si simula una TM per L sull'input invertito
- Funziona sia per **Ricorsivi** sia per **RE**

Decidibilità

- Macchine di Turing codificate come numeri
- Diagonalizzazione
- Problemi come linguaggi
- Problemi indecidibili

Enumerazione delle stringhe

- E' possibile definire un **ordinamento** di tutte le stringhe (finite) su un alfabeto:
  - Per esempio, ordine lessicografico
- Ci sono però infinite stringhe su un alfabeto
- Esiste quindi una **funzione biunivoca da numeri naturali a stringhe**:
  - ha senso parlare di **j-esima stringa** (detta stringa di **indice j**)

Enumerazione delle TM

- In modo simile è possibile definire un **ordinamento** di tutte le TM:
  - Per esempio, ordine lessicografico della loro descrizione (che è finita)
- Ci sono però infinite TM differenti (come sono infiniti i diversi FA e PDA)
- Esiste quindi **una funzione biunivoca da numeri naturali a TM**:
  - ha senso parlare di **i-esima TM** (detta TM di **indice i**)

Tabella di Accettazione

Stringa j →

1 2 3 4 5 6 . . .

TM i ↓

1
2
3 x
4
5
6
.
.
.

x=**0** significa che la i-esima TM **non accetta** la j-esima stringa;
x=**1** significa che la **accetta**

Diagonalizzazione

Stringhe

TM

	1	2	3	4	5	…
1	1	0	0	1	0	…
2		1				
3			0			
4				0		
5					1	
…						…

Diagonalizzazione

Stringhe

Inverti i valori nella diagonale

TM

	1	2	3	4	5	…
1	0	0	0	1	0	…
2		0				
3			1			
4				1		
5					0	
…						…

Diagonalizzazione

Inverti i valori nella diagonale

TM

Stringhe

	1	2	3	4	5	…
1	0	0	0	1	0	…
2		0				
3			1			
4				1		
5					0	
…						…
	0	0	1	1	0	…

?

Non lo si può trovare in una riga! – è **diversa da tutte le righe** almeno per un valore

Dettagli del ragionamento per Diagonalizzazione

- Ogni volta che abbiamo una tabella come la precedente, possiamo **diagonalizzarla**
  - Cioè, costruire una sequenza D complementando ogni bit lungo la diagonale principale
- Formalmente, $D = a_1a_2\ldots$, con:
  - $a_i=0$ se la casella (i,i) contiene 1,
  - $a_i=1$ altrimenti

La Dimostrazione per Diagonalizzazione

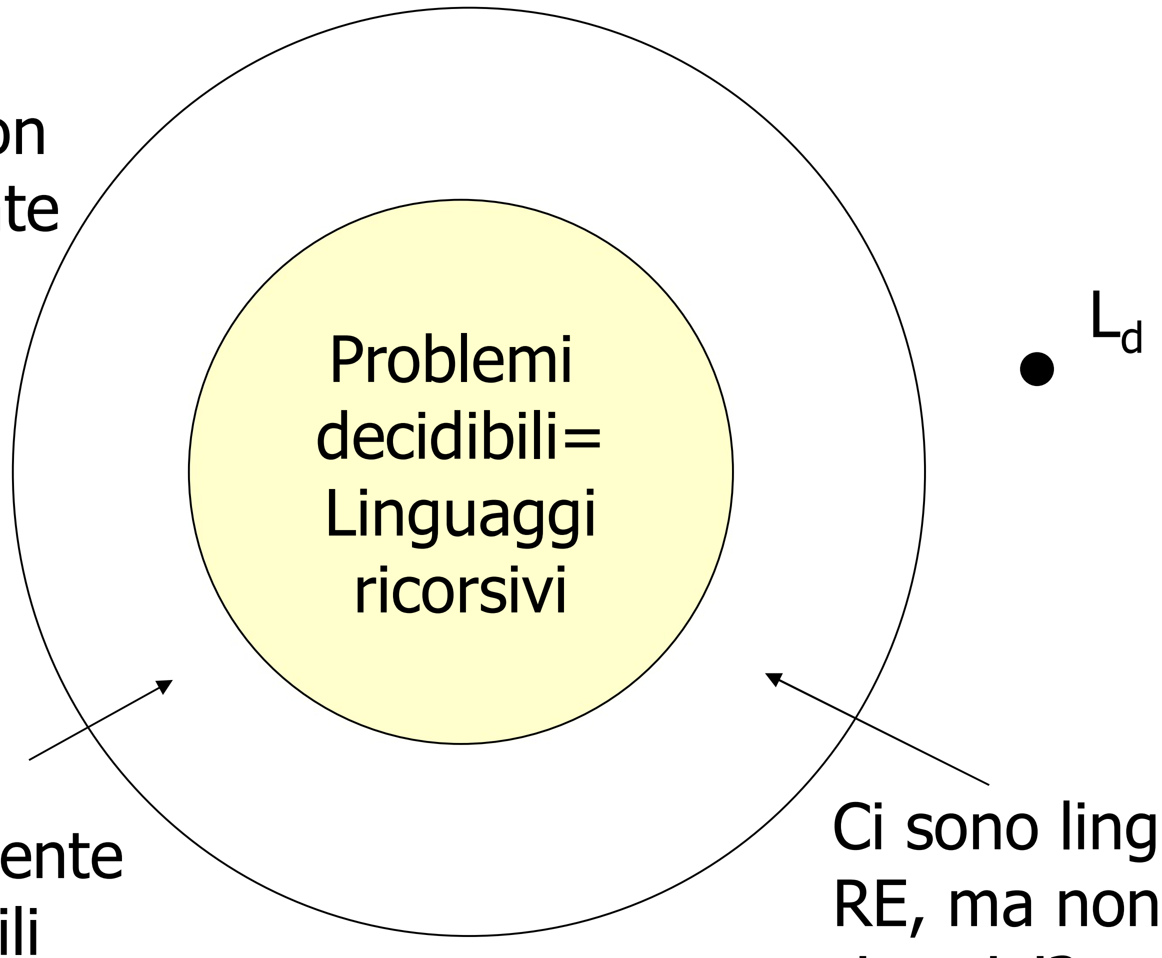
- Può D essere una riga della tabella?
  - cioè essere il linguaggio accettato da una TM?
- Supponiamo che sia la j-esima riga
  - ma D contiene nella j-esima colonna un valore diverso rispetto a quello della j-esima riga!
- Questo vale per ogni j
- Quindi D non è una riga (rappresenta un linguaggio che **nessuna TM riconosce**)

Linguaggio di Diagonalizzazione

- Linguaggio di diagonalizzazione è quindi
  L_d = {w | w è la stringa i-esima,
  e la i-esima TM **non** riconosce w}
- Abbiamo mostrato che L_d **non è** un linguaggio **ricorsivamente enumerabile**
- cioè non esiste una TM M tale che L(M)= L_d

Graficamente

Linguaggi non ricorsivamente enumerabili



Linguaggi ricorsivamente enumerabili

Ci sono linguaggi RE, ma non ricorsivi?

Dalla teoria alla pratica

- Il fatto che L_d sia indecidibile è interessante dal punto di vista **teorico**, ma non ha impatto sul mondo reale
- Considereremo altri problemi legati alle TM, per mostrare che alcuni **problemi reali sono indecidibili**

Esempio: Problemi Indecidibili

- Può una certa linea di codice essere eseguita durante l'esecuzione di un programma?
- Può una variabile contenere un certo valore durante l'esecuzione di un programma?

Il Linguaggio Universale

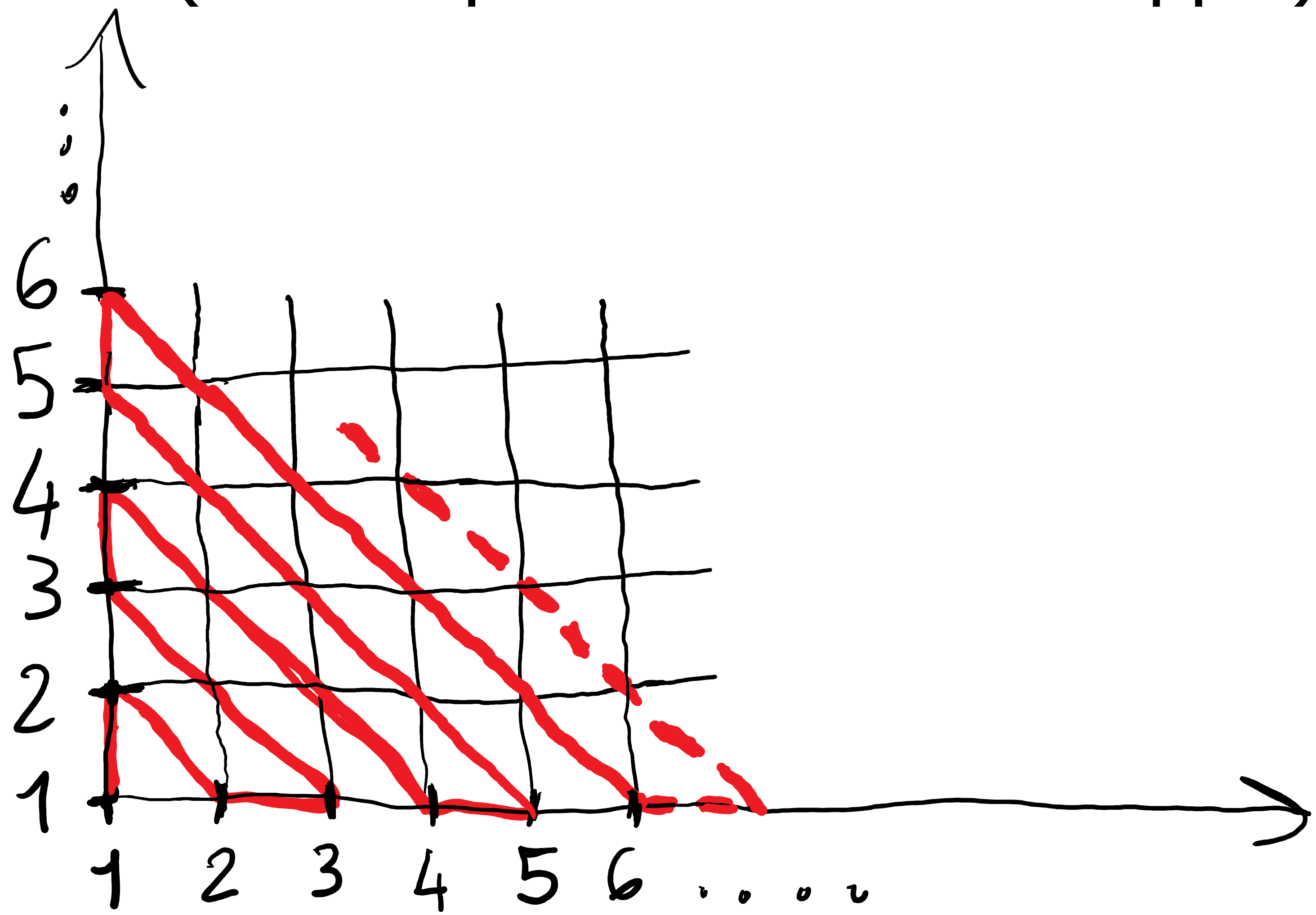
- Un esempio di un linguaggio ricorsivamente enumerabile, ma non ricorsivo, è il linguaggio L_u della **Macchina di Turing Universale**
- La **UTM** ha come input l'**indice i** di una certa TM M e l'**indice j** di una stringa w, e accetta se e solo se $\mathbf{M_i}$ **accetta** $\mathbf{w_j}$
- UTM è una semplice **simulazione** (realizzabile con più nastri)

L_u : linguaggio universale (definizione formale)

- Supponiamo di **enumerare le coppie** <i,j> e le rappresentiamo con corrispondenti stringhe $\mathbf{w}_{\mathbf{<i,j>}}$
- $L_u = \{\mathbf{w}_{\mathbf{<i,j>}} \mid M_i \text{ accetta } w_j\}$
- UTM garantisce che L_u è sicuramente RE
- Mostreremo che $\mathbf{L_u}$ **non** può essere **ricorsivo**

Dovetailing

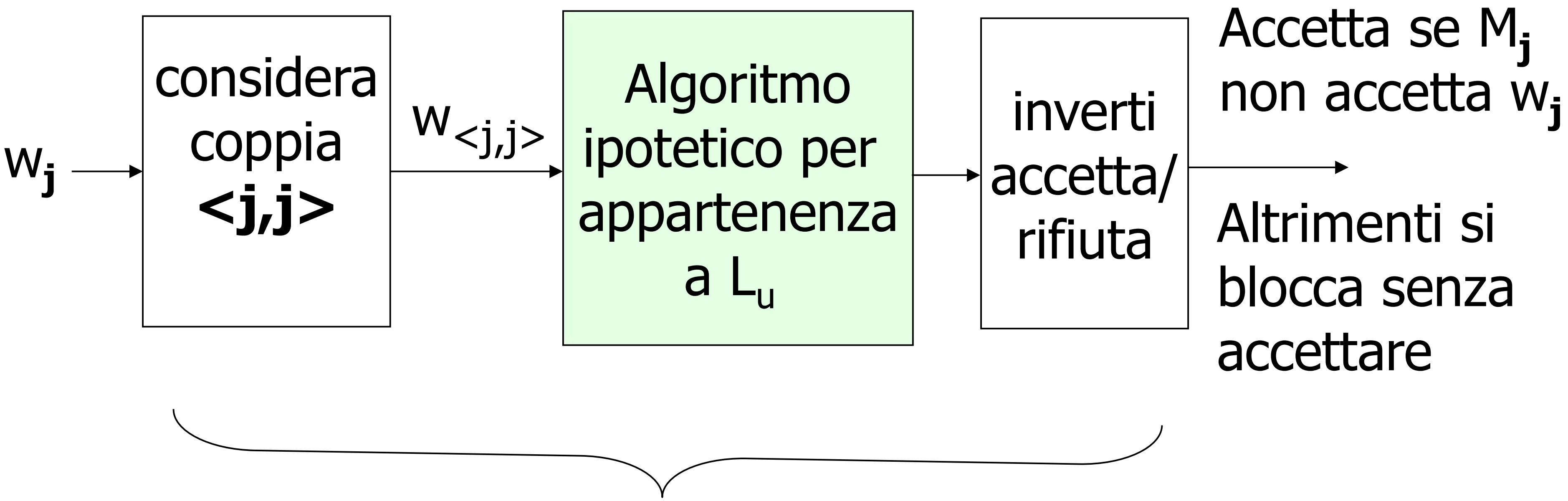
(tecnica per enumerare le coppie)



Prova

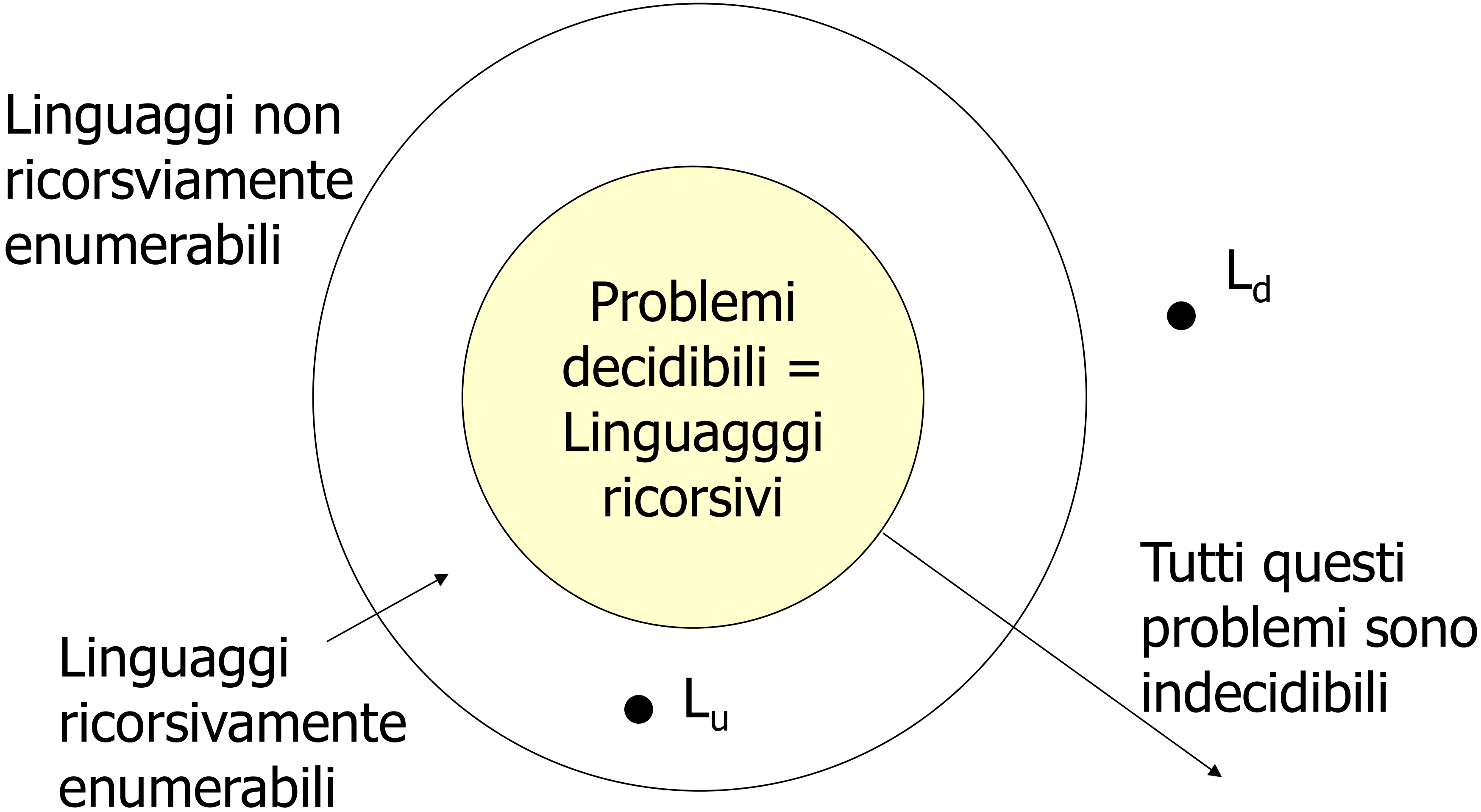
- Ipotizziamo per assurdo che L_u sia ricorsivo
- Data una **stringa w**, possiamo decidere se è in $\mathbf{L_d}$ nel seguente modo
  - Calcoliamo a quale indice corrisponde w: cioè consideriamo **j tale che** $\mathbf{w=w_j}$
  - Verifichiamo se $\mathbf{w_{<j,j>}}$ **non appartiene a** $\mathbf{L_u}$ (possibile per ipotesi) e in tal caso concludiamo che **w è in** $\mathbf{L_d}$
- Ma questo non è possibile!
- **Conclusione**: $\mathbf{L_u}$ è **RE**, ma **non** è **ricorsivo**

Immagine della Riduzione da L_d a L_u



Questo sarebbe un algoritmo per L_d , che non può esistere!

Graficamente



Problemi reali

- Indecidibilità di L_u (problema della fermata):
  - es. è indecidibile il problema di stabilire se, dato il codice di una funzione (es in C) ed il valore dei parametri, l'esecuzione di tale codice terminerà
  - è indecidibile anche il problema di stabilire se, dato un programma, il nome x di una variabile e un valore v, x può valere v durante l'esecuzione
    - es. nel main uso una variabile booleana **x inizialmente false**, chiamo la funzione sopra ed assegno, al ritorno da tale funzione, **x=true**

Problemi reali

- È indecidibile il problema di stabilire se, data una **formula della logica dei predicati** essa è, o meno, **valida**
  - Programmazione logica (es. PROLOG) è **Turing** equivalente (può codificare qualsiasi procedura)
  - un **programma logico** è una formula della logica dei predicati
  - la **terminazione** con successo dell'esecuzione, per un goal, si ha quando formula totale è **valida**
    - riduco problema fermata a problema validità

E' possibile liberarsi delle computazioni non terminanti?

- Consideriamo un ipotetico **linguaggio di programmazione** con il quale è possibile scrivere **soli algoritmi**
  - cioè un linguaggio di programmazione in cui un programma è **garantito terminare sempre** (**accettando o rifiutando**)
  - è possibile che esista un tale linguaggio di programmazione e che consenta di esprimere **tutti i linguaggi ricorsivi?**
    - cioè tutti i problemi risolubili algoritmicamente?

Diagonalizzazione

Stringhe

tutti i programmi di un ipotetico linguaggio di programmazione che produce soli algoritmi

	1	2	3	4	5	…
1	1	0	0	1	0	…
2		1				
3			0			
4				0		
5					1	
…						…

Diagonalizzazione

Inverti i valori nella diagonale

Stringhe

tutti i programmi di un ipotetico linguaggio di programmazione che produce soli algoritmi

	1	2	3	4	5	…
1	0	0	0	1	0	…
2		0				
3			1			
4				1		
5					0	
…						…

Diagonalizzazione

Inverti i valori nella diagonale

tutti i programmi di un ipotetico linguaggio di programmazione che produce soli algoritmi

Stringhe

	1	2	3	4	5	…
1	0	0	0	1	0	…
2		0				
3			1			
4				1		
5					0	
…						…

Assurdo! E' un linguaggio **ricorsivo** non esprimibile nel linguaggio di programmazione

(data $w_{\mathbf{j}}$ basta darla in input all'algoritmo **j** e invertire la risposta)

Altri Problemi Indecidibili

Proprietà di linguaggi RE (riconosciuti da TM)

- Decidibilità di problemi su linguaggi RE (e quindi riconosciuti da una TM M):
  - E' L(M) un linguaggio regolare?
  - E' L(M) un CFL?
  - L(M) include le stringhe palindrome?
  - E' L(M) vuoto?
  - L(M) contiene più di 1000 stringhe?
  - L(M) = L(M')?
  - Ecc., ecc.

Proprietà di linguaggi RE (riconosciuti da TM)

- È decidibile se, data una MDT M, L(M) soddisfa o meno una proprietà **P**?
  - esempio: P = essere vuoto
- Ciò è come chiedersi:
  - considerato $\mathbf{L_P}$ = { w_i | L(M_i) soddisfa P },
  - $\mathbf{L_P}$ è **ricorsivo** oppure no?

Teorema di Rice

- Ci sono due casi di proprietà P **banali** per cui **L_P è ricorsivo**
  - **proprietà P mai soddisfatta** ($L_P = \emptyset$)
  - **proprietà P sempre soddisfatta** (L_P = tutte le stringhe)
- **Teorema di Rice**: per **tutte le altre proprietà P** (non banali) **L_P è non ricorsivo**

Applicazioni del Teorema di Rice

- Tutti i problemi interessanti su linguaggi RE (e quindi di TM) sono **indecidibili**:
  - E' L(M) un linguaggio regolare?
  - E' L(M) un CFL?
  - L(M) include le stringhe palindrome?
  - E' L(M) vuoto?
  - L(M) contiene più di 1000 stringhe?
  - L(M) = L(M')?
  - Ecc., ecc.

Applicazioni del Teorema di Rice

- Solo **proprietà banali** sono **decidibili**:
  - E’ L(M) un linguaggio RE?
    - Dato M è **sempre vero**
      (quindi algoritmo banale che risponde sempre sì)
  - E’ L(M) non RE?
    - Dato M è **sempre falso**
      (quindi algoritmo banale che risponde sempre no)

Esempi a cui il Teorema di Rice non si applica

- Problemi che riguardano caratteristiche di una TM **non esclusivamente riguardanti il linguaggio che riconosce**:
  - M riconosce w in 10 passi?
  - M è la i-esima TM (cioè $M=M_i$)?
  - M_i riconosce w_i?

Da decidibilità problemi su linguaggi a calcolabilità funzioni

- In altre trattazioni **decibilità del problema se**, dato un i, $\mathbf{w_i \in L}$ espressa come:
  calcolabilità della funzione f:$\mathbb{N}\rightarrow$**{0,1}**
  in cui **f(i)=1** corrisponde al fatto che $\mathbf{w_i \in L}$
  - **f** si dice **calcolabile** quando il problema è **decidibile** (calcolato da TM che termina sempre)
  - **f** si dice **parzialmente calcolabile** quando il problema è **semi-decidibile** (calcolato da TM)

Da decidibilità problemi su linguaggi a calcolabilità funzioni

- Concetto calcolabilità di funzioni esteso a: **calcolabilità di funzioni f:**$\mathrm{N}\rightarrow\mathrm{N}$
  - **f** si dice **calcolabile** quando esiste una TM che termina sempre e che, su input i, dà **risultato f(i)**
  - **invece** che TM che **accettano o meno**: TM che danno un risultato (sul nastro quando terminano)
- Quando si ha a che fare con una funzione o un predicato (es. in modello logica predicati) bisogna sempre **chiedersi** se sia **calcolabile**

Esercizi

- Si consideri il seguente linguaggio

$L = \{ w_{<x,y>} \mid x$ e y sono tali che $L(M_x)$ e $L(M_y)$ hanno almeno una stringa in comune $\}$

dove M_x e M_y sono la x-esima e la y-esima macchina di Turing.

Dire se L è ricorsivo, ricorsivamente enumerabile, o nemmeno ricorsivamente enumerabile. Giustificare la risposta.

Esercizi

- Si consideri il seguente linguaggio

  $L = \{ w_{<x,y>} \mid$ x e y sono tali che $L(M_x)$ e $L(M_y)$ **non** hanno stringhe in comune $\}$

  dove M_x e M_y sono la x-esima e la y-esima macchina di Turing.

  Dire se L è ricorsivo, ricorsivamente enumerabile, o nemmeno ricorsivamente enumerabile. Giustificare la risposta.