

La Macchina di Turing (MdT)

Alberto Marchetti Spaccamela

A.A. 2021-2022

Definizione di Macchina di Turing

La macchina di Turing (MdT) è stata proposta nel 1936 come modello di calcolo

La memoria della MdT è organizzata linearmente come le cellule di un nastro infinito; ogni cella del nastro può contenere esattamente un simbolo di un alfabeto con un numero finito di simboli.

La macchina è dotata di una testina di lettura e scrittura del nastro che, ad ogni istante, è posizionata su una cella del nastro.

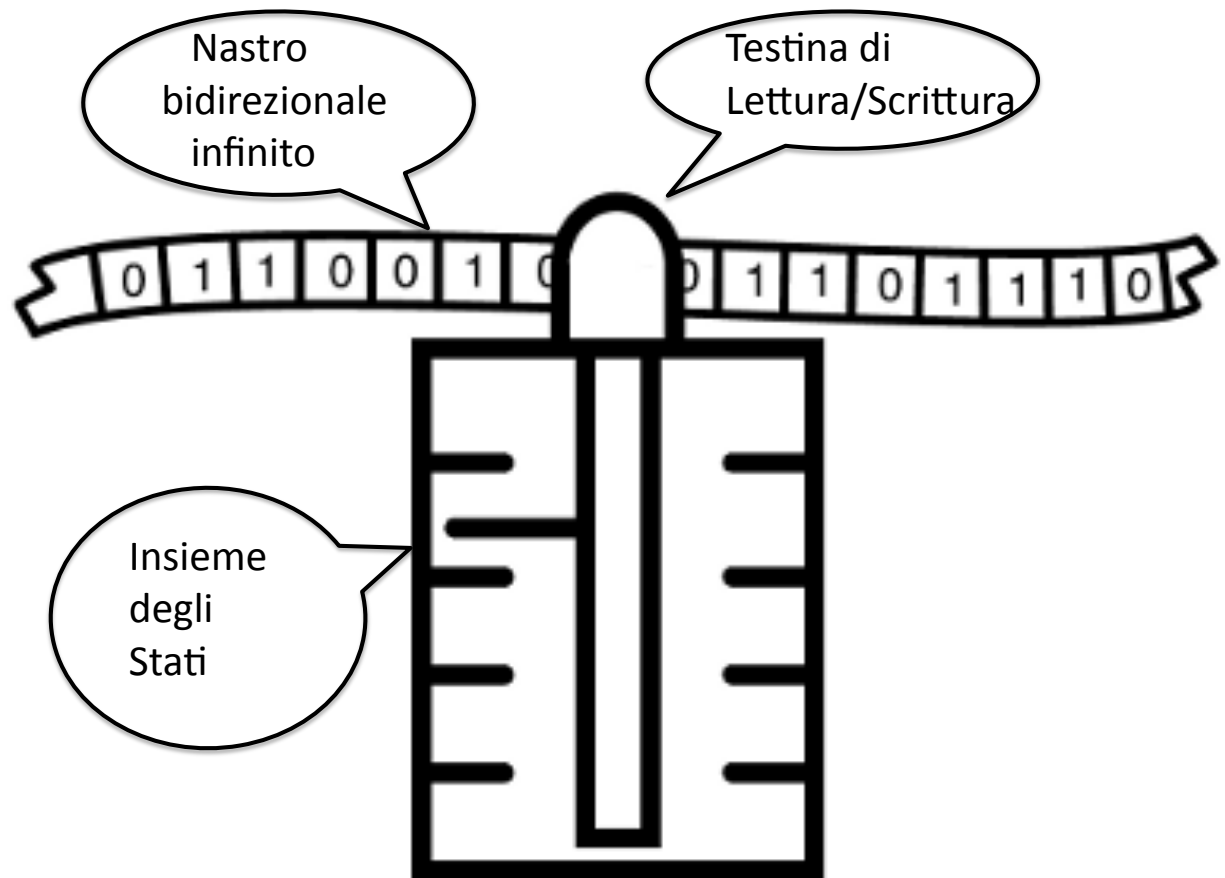
All'inizio, i dati di ingresso sono posti sul nastro che per la parte rimanente è vuoto e la testina di lettura/scrittura è posizionata sul primo carattere dell'input.

La parte di controllo può essere immaginata come una scatola del cambio che assume un numero prefissato e finito di posizioni. Ogni possibile posizione è un possibile stato della macchina. Il numero di stati varia da macchina a macchina.

Definizione di Macchina di Turing

Possiamo immaginare una Macchina di Turing come un automa a stati finiti che legge **e scrive** in memoria ed è **in grado di muoversi sul nastro nelle due direzioni**. (Nota: un automa a stati finiti non scrive sul nastro e si muove sempre nella stessa direzione - scorre input)

una macchina di Turing con un alfabeto binario e con dieci possibili posizioni del 'cambio' (stati).



Definizione di Macchina di Turing

La parte di controllo è così specificata.

- Ad ogni istante la macchina di Turing si trova in uno stato e la testina di lettura legge la cella del nastro su cui è posizionata.
- Poiché sia l'alfabeto sul nastro che il numero di stati della macchina sono finiti abbiamo un numero finito di possibili coppie <stato, carattere letto>;
- Per ciascuna coppia di valori sono specificate tre informazioni:
Carattere da stampare, Movimento della testina, Prossimo stato
- Il controllo, sulla base della coppia <stato, carattere letto> che rappresenta lo stato corrente e il simbolo letto dalla testina, esegue i seguenti passi:
 1. stampa un nuovo simbolo sul nastro (o lascia sul nastro lo stesso simbolo letto)
 2. sposta la testina di lettura/scrittura di una posizione a destra o a sinistra sul nastro
 3. passa allo stato successivo (che può anche essere lo stesso stato)

Definizione di Macchina di Turing

Stati: Sia V l'insieme degli stati; in V distinguiamo **uno stato iniziale q_0** e un **insieme di stati finali F**

Passo di calcolo

Il controllo, sulla base della coppia $\langle \text{stato}, \text{carattere letto} \rangle$ che rappresenta lo stato corrente e il simbolo letto dalla testina:

1. stampa un nuovo simbolo sul nastro (o lascia sul nastro lo stesso simbolo letto)
2. sposta la testina di lettura/scrittura di una posizione a destra o a sinistra sul nastro
3. passa allo stato successivo (che può anche essere lo stesso stato)

Calcolo

All'inizio la MdT è nello stato q_0 e la testina si trova sul primo carattere dell'input (più a sinistra)

Successivamente esegue una sequenza di passi di calcolo

La MdT si ferma quando giunge in uno degli stati finali; il risultato del calcolo è sul nastro a partire dalla posizione finale della testina

Definizione di Macchina di Turing

Definizione 1.1: macchina di Turing

Una *macchina di Turing* è costituita da un **alfabeto di lavoro** Σ contenente il simbolo \square e da un **grafo delle transizioni**, ovvero un grafo etichettato $G = (V, E)$ tale che:

- $V = \{q_0\} \cup F \cup Q$ è l'insieme degli **stati** (q_0 è lo stato **iniziale**, F è l'insieme degli stati **finali** e Q è l'insieme degli stati che non sono iniziali né finali);
- E è l'insieme delle **transizioni** e, a ogni transizione, è associata un'etichetta formata da una lista l di triple (σ, τ, m) , in cui σ e τ sono simboli appartenenti a Σ e $m \in \{R, L, S\}$, tale che non esistono due triple in l con lo stesso primo elemento.

I simboli σ e τ che appaiono all'interno di una tripla dell'etichetta di una transizione indicano, rispettivamente, il simbolo attualmente letto dalla testina e il simbolo da scrivere, mentre il valore m specifica il movimento della testina: in particolare, R corrisponde allo spostamento a destra, L allo spostamento a sinistra e S a nessun spostamento. Nel seguito di questa dispensa, per evitare di dover specificare ogni volta

Definizione di Macchina di Turing

Stati: Sia V l'insieme degli stati; in V distinguiamo **uno stato iniziale q_0** e un **insieme di stati finali F**

Calcolo

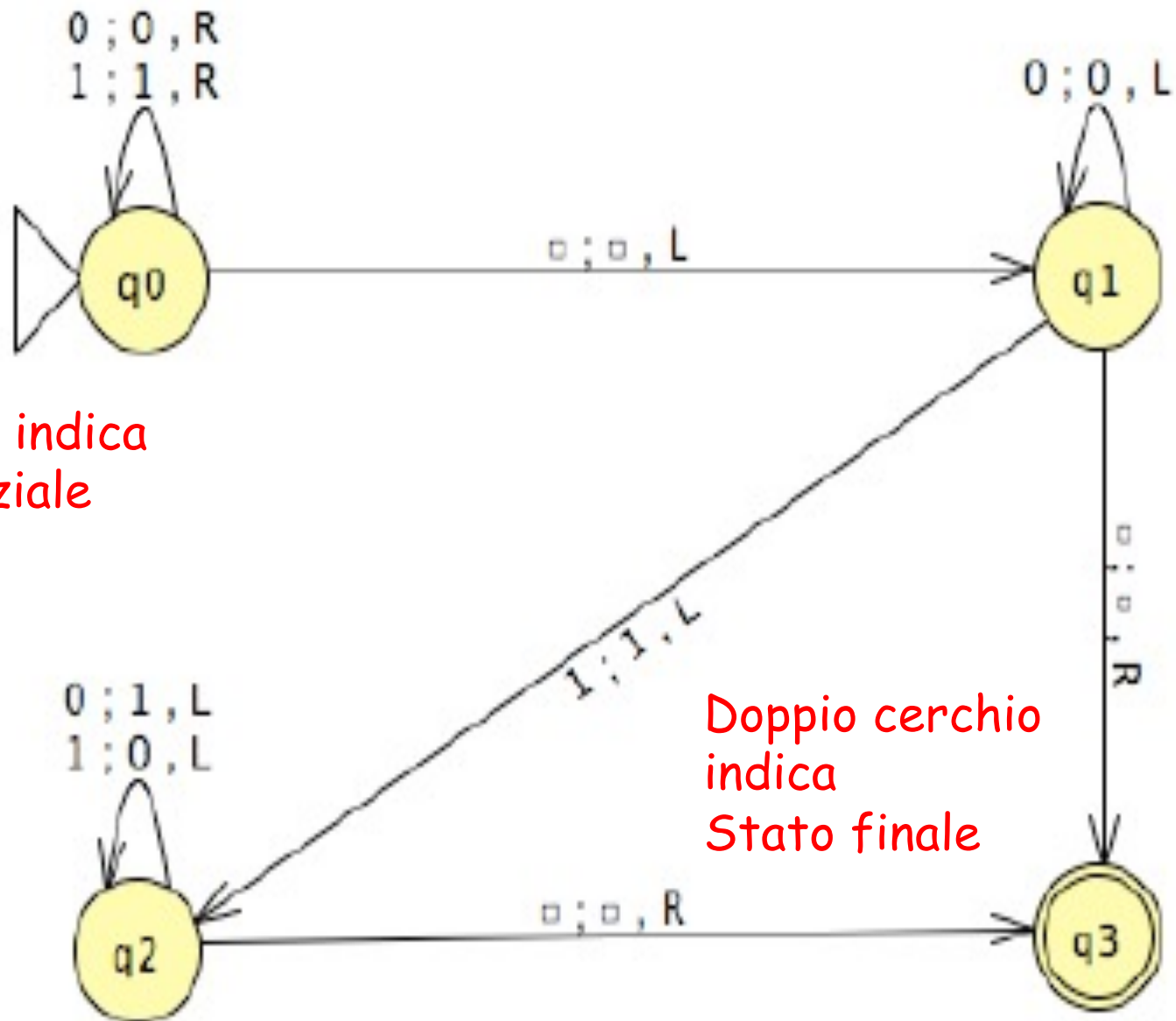
All'inizio la MdT è nello stato q_0 e la testina si trova sul primo carattere dell'input (più a sinistra)

Successivamente esegue una sequenza di passi di calcolo

La MdT si ferma quando

- giunge in uno degli stati finali; il risultato del calcolo è sul nastro a partire dalla posizione finale della testina
- Si trova in uno stato q non finale legge il carattere c sul nastro e non è definita la corrispondenza $\langle q, c \rangle$. In questo caso possiamo dire che la MdT abortisce.

Rappresentazione grafica (L sinistra, R destra)



Esempio: MdT per il complemento bit a bit

calcolare la stringa binaria ottenuta eseguendo il complemento bit a bit della stringa binaria ricevuta in input:

- tre possibili stati q_0 , q_1 e q_2 , di cui il primo q_0 , stato iniziale, e q_2 , l'unico stato finale.

Intuizione sull'uso e degli stati

- q_0 rappresenta lo stato in cui modifichiamo la stringa di input: per fare questo abbiamo una sequenza di passi; in ciascuno di questi complementiamo un bit e ci spostiamo sul bit successivo (a destra)
- Quando nello stato q_0 il carattere letto è \square allora tutta la stringa è stata complementata bit a bit abbiamo finito ma dobbiamo portare la testina sul primo carattere; usiamo lo stato q_1
- q_1 rappresenta lo stato in cui ci spostiamo a sinistra lasciando immutato il nastro; questa fase termina quando incontriamo nuovamente il carattere \square ; a questo punto la MdT si sposta a destra e passa nello stato q_2 e si ferma
- q_2 rappresenta lo stato finale; nota che a questo punto la MdT è sul primo carattere del nastro

Esempio 1: MdT per il complemento bit a bit

calcolare la stringa binaria ottenuta eseguendo il complemento bit a bit della stringa binaria ricevuta in input:

- tre possibili stati q_0 (iniziale), q_1 e q_2 (finale)

Programma

La prima istruzione fa sì che la macchina scorra l'intera stringa di input, complementando ciascun bit incontrato

1. Se lo stato è q_0 e il simbolo letto non è \square , complementa il simbolo letto e sposta la testina a destra. (rimani nello stesso stato)

Le successive tre istruzioni permettono di riportare la testina all'inizio della stringa modificata.

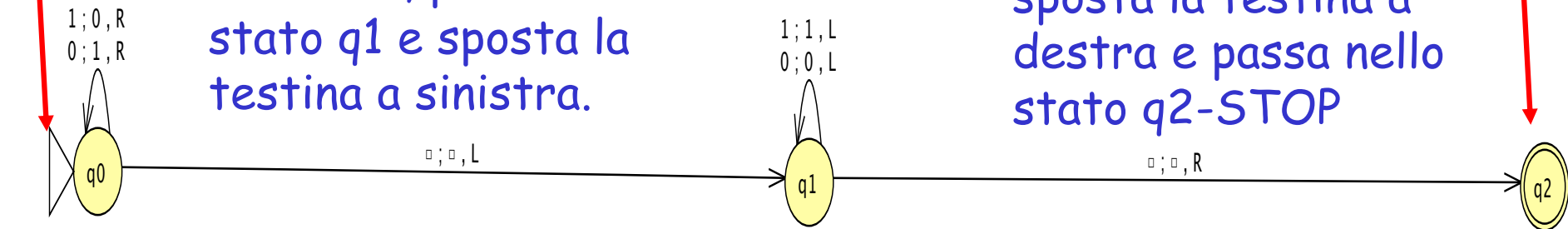
2. Se lo stato è q_0 e il simbolo letto è \square , passa allo stato q_1 e sposta la testina a sinistra. (la testina torna sull'ultimo carattere di input)
3. Se lo stato è q_1 e il simbolo letto non è \square , sposta la testina a sinistra (rimani nello stato q_1) (la testina torna indietro)
4. Se lo stato è q_1 e il simbolo letto è \square , sposta la testina a destra e passa nello stato q_2 (siamo arrivati alla fine del nastro la testina torna indietro)

Triangolo indica
Stato iniziale

Doppio cerchio
indica
Stato finale

stato è q_0 e il simbolo
letto è \square , passa allo
stato q_1 e sposta la
testina a sinistra.

stato è q_1 e il
simbolo letto è \square ,
sposta la testina a
destra e passa nello
stato q_2 -STOP



stato è q_0 ,
simbolo letto
non è \square ,
complementa il
simbolo letto
e sposta la
testina a destra.
(rimani nello
stato q_0)

stato è q_1 e il
simbolo letto
non è \square , sposta
la testina a
sinistra (rimani
nello stato q_1)
(la testina
torna indietro)

ricorda \square
rappresenta il
carattere vuoto

Rappresentazione tabellare

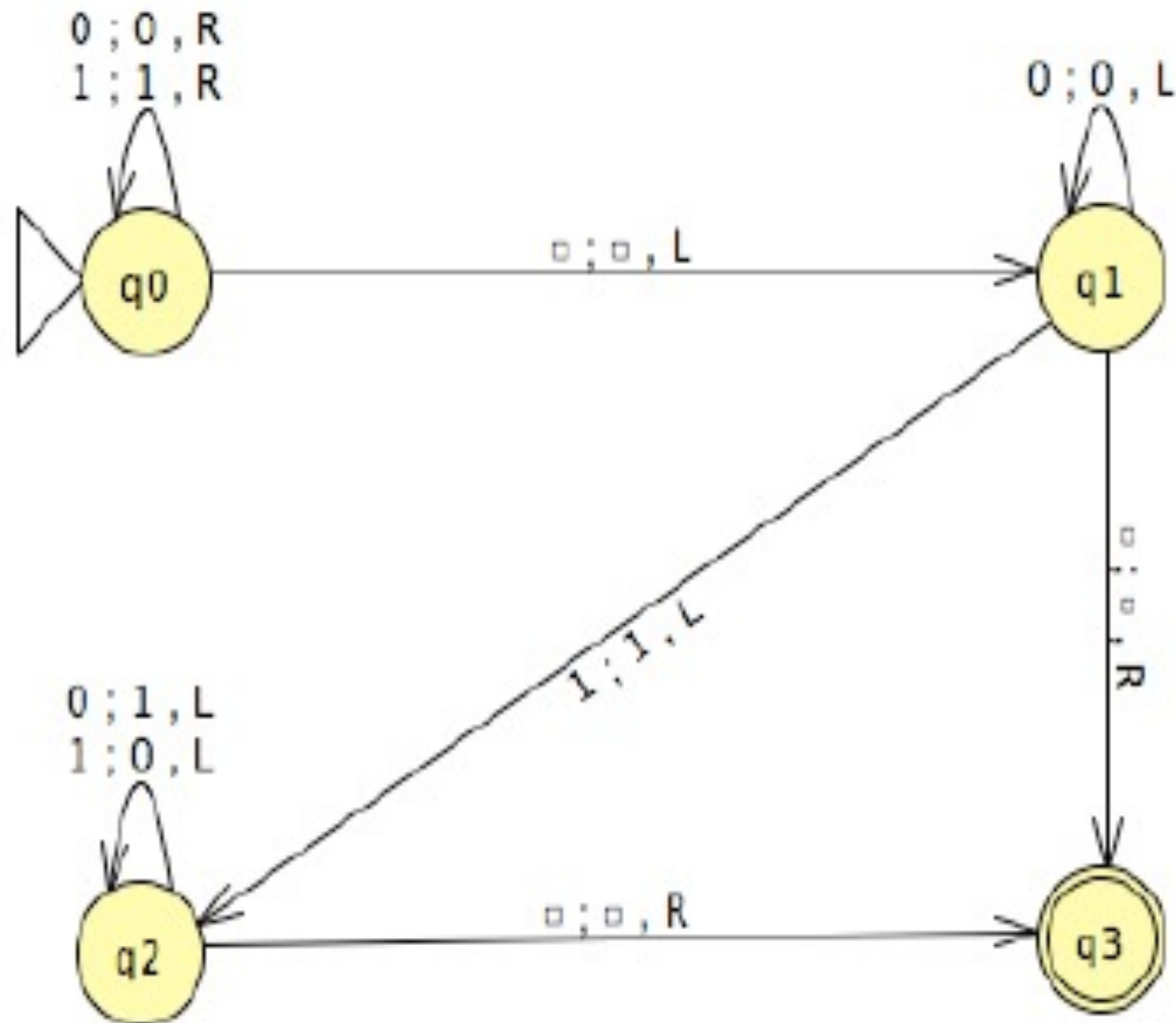
.3: rappresentazione tabellare della macchina per il complemento bit a bit

ferimento alla macchina di Turing introdotta nell'Esempio 1.1 e mostr
, la corrispondente rappresentazione tabellare di tale macchina è la segu

stato	simbolo	stato	simbolo	movimento
q0	0	q0	1	R
q0	1	q0	0	R
q0	□	q1	□	L
q1	0	q1	0	L
q1	1	q1	1	L
q1	□	q2	□	R

Nota □ rappresenta il carattere vuoto (nessun
carattere, spazio) sul nastro

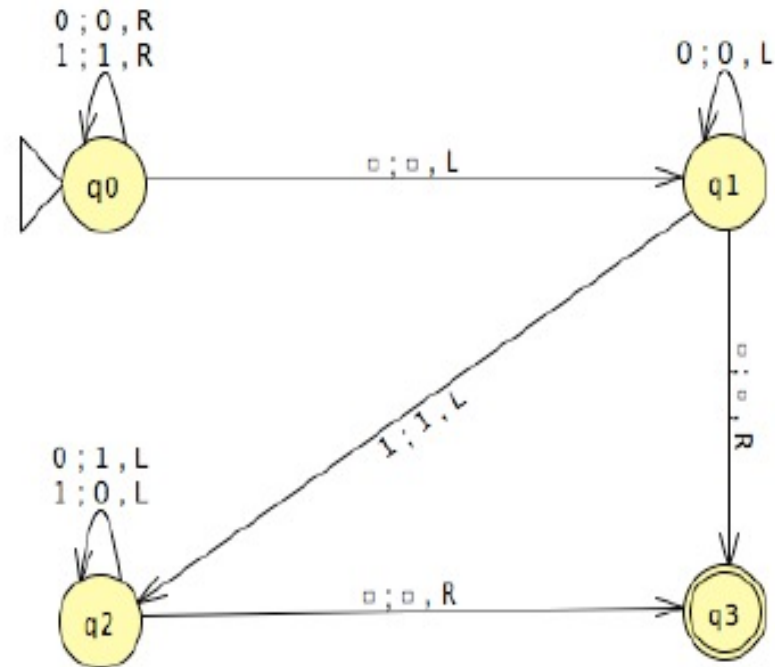
Esempio 2: cosa calcola questa MdT? (Alfabeto binario)



Esempio 2: cosa calcola questa MdT? (Alfabeto binario)

stato

- q_0 : scorriamo input (senza modificarlo) fino alle fine; alla fine vado in stato q_1
- q_1 : fino a che trovo 0 riscrivo e mi muovo a sinistra; quando trovo 1 o \square vado in q_2 o q_3
- q_2 : se leggi 1 scrivi 0 se leggi 0 scrivi 1 e rimani in q_2 ; se leggi \square vai in q_3
- q_3 : termino



La MdT calcola il complemento a 2
di una stringa binaria

Esempio 3: una MdT per riconoscere le stringhe 0^n1^n stringhe che iniziano con 0 seguite da ugual numero di 1

Definiamo una MdT che, data in input una sequenza di simboli 0 seguita da una sequenza di simboli 1, termina

- in uno stato finale se il numero di simboli 0 è uguale a quello dei simboli 1,
- altrimenti termina in uno stato non finale non avendo istruzioni da poter eseguire.

Problema

- Il numero di zeri e uno non è fissato
- Le MdT hanno un numero finito di stati e quindi non possono contare

Esempio 3: una MdT per riconoscere le stringhe 0^n1^n stringhe che iniziano con 0 seguite da ugual numero di 1

Problema

- Il numero di zeri e uno non è fissato
- Le MdT hanno un numero finito di stati e quindi non possono contare

Idea

- La stringa deve iniziare con 0 e terminare con 1
- Eseguo un ciclo in cui ad ogni iterazione cancello il primo 0 e l'ultimo 1 (cancellare vuol dire scrivere \square sul nastro)
- Se il numero di 0 iniziali è pari al numero di 1 che seguono allora in questo modo ottengo....

Esempio 3: una MdT per riconoscere le stringhe 0^n1^n stringhe che iniziano con 0 seguite da ugual numero di 1

Problema

- Il numero di zeri e uno non è fissato
- Le MdT hanno un numero finito di stati e quindi non possono contare

Idea

- La stringa deve iniziare con 0 e terminare con 1
- Eseguo un ciclo in cui ad ogni iterazione cancello il primo 0 e l'ultimo 1 (cancellare vuol dire scrivere \square sul nastro)
- Se il numero di 0 iniziali è pari al numero di 1 che seguono allora in questo modo ottengo....

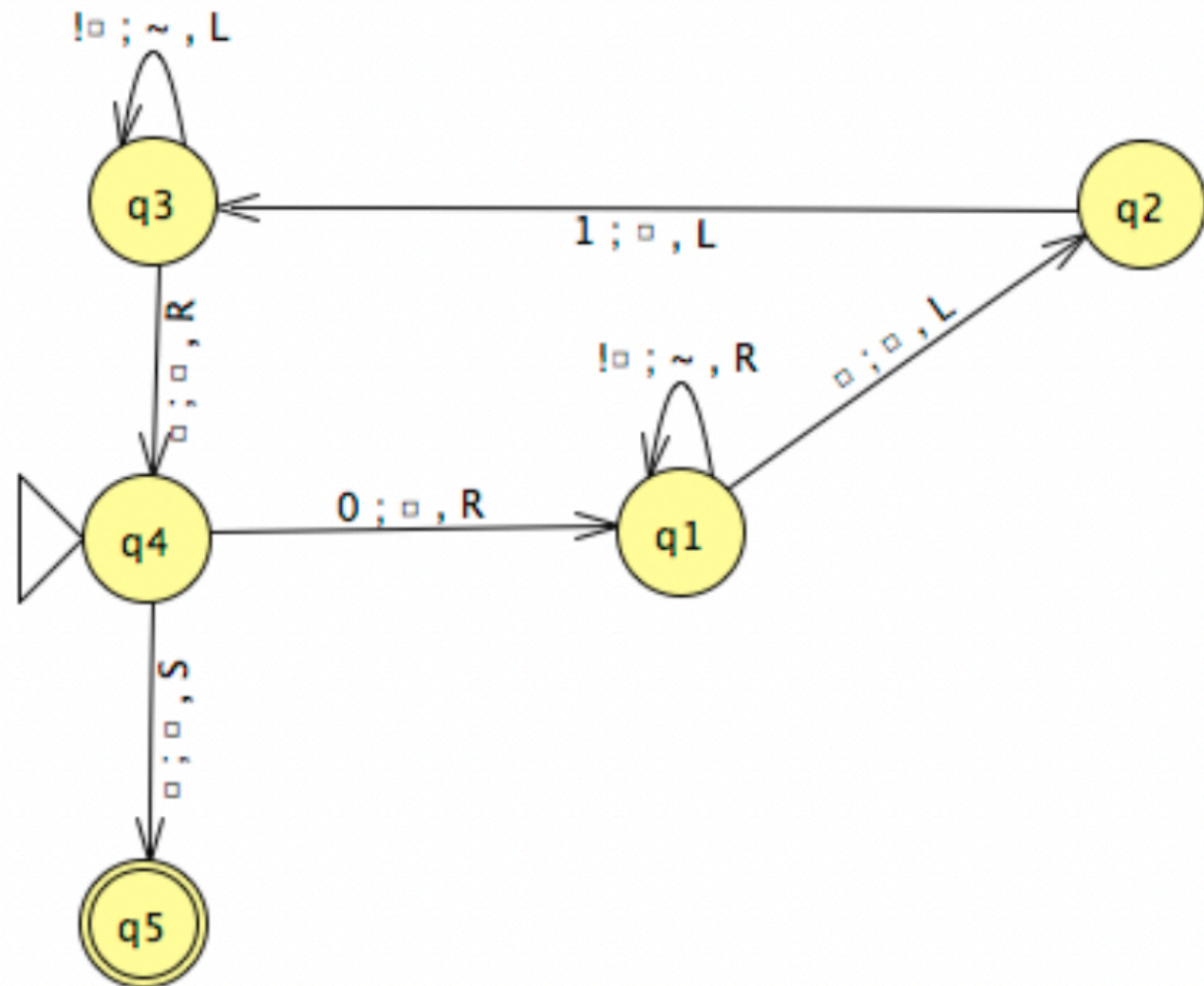
stato

- q_0 :
- q_1 :
- q_2 :
- q_3 :

Esempio 3: una MdT per riconoscere le stringhe $0^n 1^n$

Per semplificare la figura

- $!\square$ equivale a diverso da \square
- \sim riscrivi il carattere letto



Macchina di Turing con k nastri

Ci sono k nastri invece di uno; in rosso le modifiche rispetto a un solo nastro

La parte di controllo è così specificata.

- Ad ogni istante la macchina di Turing si trova in uno stato e ciascuna testina di lettura legge la cella nastro su cui è posizionata.
- Poiché sia l'alfabeto sul nastro che il numero di stati della macchina sono finiti abbiamo un numero finito di $(k+1)$ sequenze

$\langle \text{stato}, \text{carattere nastro 1}, \text{nastro 2}, \dots \text{nastro } k \rangle$;

Per ciascuna coppia di valori sono specificate $2k+1$ informazioni:

- Il prossimo stato e, per ciascun nastro, 2 informazioni: Carattere da stampare e movimento della testina,
- Il controllo, esegue i seguenti passi:
 1. su ciascun nastro stampa un nuovo simbolo (o lascia sul nastro lo stesso simbolo letto) e sposta la testina di lettura/scrittura di una posizione a destra o a sinistra sul nastro
 2. passa allo stato successivo (che può anche essere lo stesso stato)

Macchina di Turing con k nastri

Per ciascuna coppia di valori sono specificate $2k+1$ informazioni:

- Il prossimo stato e, per ciascun nastro, 2 informazioni: Carattere da stampare e movimento della testina,
- Il controllo, esegue i seguenti passi:
 1. su ciascun nastro stampa un nuovo simbolo (o lascia sul nastro lo stesso simbolo letto) e sposta la testina di lettura/scrittura di una posizione a destra o a sinistra sul nastro
 2. passa allo stato successivo (che può anche essere lo stesso stato)

Assumiamo che all'inizio l'input sia su un nastro e gli altri nastri siano vuoti

Macchina di Turing con k nastri

Per ciascuna coppia di valori sono specificate $2k+1$ informazioni:

- Il prossimo stato e, per ciascun nastro, 2 informazioni: Carattere da stampare e movimento della testina,
- Il controllo, esegue i seguenti passi:
 1. su ciascun nastro stampa un nuovo simbolo (o lascia sul nastro lo stesso simbolo letto) e sposta la testina di lettura/scrittura di una posizione a destra o a sinistra sul nastro
 2. passa allo stato successivo (che può anche essere lo stesso stato)

Assumiamo che all'inizio l'input sia su un nastro e gli altri nastri siano vuoti

Esempio MdT con k nastri

Consideriamo il linguaggio $L = \{0^n 1^n : n > 0\}$


definiamo una macchina di Turing T con 2 nastri che, data in input una sequenza di simboli 0 seguita da una sequenza di simboli 1, termina in uno stato finale se il numero di simboli 0 è uguale a quello dei simboli 1, altrimenti termina in uno stato non finale

La macchina opera nel modo seguente.

1. Scorre il primo nastro verso destra fino al primo simbolo 1:

per ogni simbolo 0 letto viene scritto un simbolo 1 sul secondo nastro, spostando la testina a destra dopo aver eseguito la scrittura.

2. Scorre il primo nastro da sinistra verso destra e il secondo nastro da destra verso sinistra, controllando che i simboli letti sui due nastri siano uguali: se così non fosse, termina senza avere alcuna istruzione da poter eseguire (la stringa NON appartiene al linguaggio).

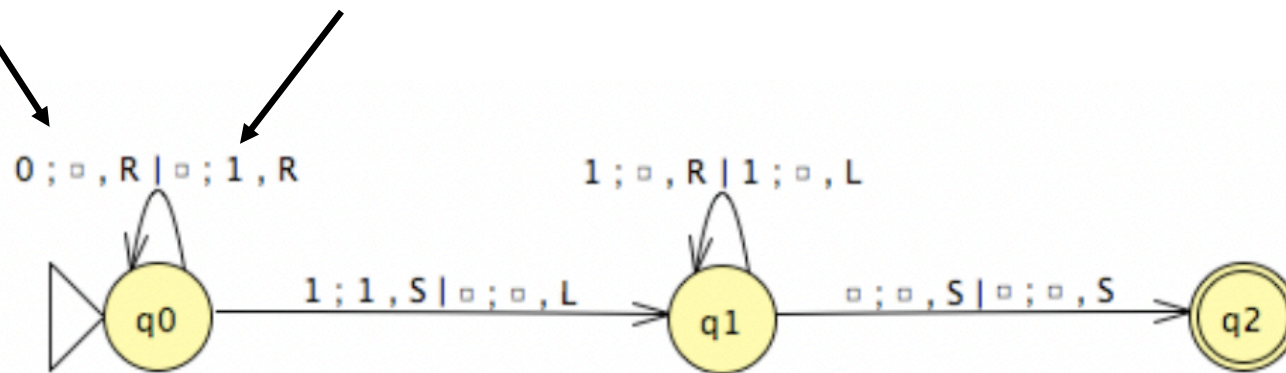
3. Una volta incontrato il simbolo  su entrambi i nastri, termina nello stato finale.

Esempio MdT con k nastri

Consideriamo il linguaggio $L = \{0^n 1^n : n > 0\}$

definiamo una macchina di Turing T con 2 nastri che, data in input una sequenza di simboli 0 seguita da una sequenza di simboli 1, termina in uno stato finale se il numero di simboli 0 è uguale a quello dei simboli 1, altrimenti termina in uno stato non finale

Nastro 1. Nastro 2



Stato q0: scorri nastro e per ogni 0 scrivi 1 su nastro 2

Stato q1: scorri nastro 1 verso destra e nastro 2 verso sinistra;

- Se leggi 1 e 1 sui due nastri rimani in stato q1
- Se leggi \square e \square vai in stato q2 e termina
- Altrimenti rifiuta (non c'è transizione)

MdT con 1 e con k nastri

Chiaramente, tutto ciò che può essere fatto da una macchina di Turing ordinaria risulta essere realizzabile da una macchina di Turing multi-nastro.

Infatti una macchina di Turing ordinaria T corrisponde al caso particolare in cui il numero di nastri k sia uguale a 1: pertanto, una macchina di Turing con $k > 1$ nastri può simulare T sul primo nastro non modificando mai il contenuto dei rimanenti $k-1$ nastri.

A prima vista, potrebbe invece sembrare che le macchine multi-nastro siano computazionalmente più potenti di quelle ordinarie.

In realtà tutto quello che possiamo calcolare con le macchine con k nastri lo possiamo calcolare su una macchina con un solo nastro

L'idea della dimostrazione consiste nel concatenare uno dopo l'altro il contenuto dei k nastri, separati da un simbolo speciale, che non faccia parte dell'alfabeto di lavoro della macchina multi-nastro: indichiamo con $\#$ tale simbolo.

MdT con 1 e con k nastri

In realtà tutto quello che possiamo calcolare con le macchine con k nastri lo possiamo calcolare su una macchina con un solo nastro

L'idea della dimostrazione consiste nel concatenare uno dopo l'altro il contenuto dei k nastri, separati da un simbolo speciale, che non faccia parte dell'alfabeto di lavoro della macchina multi-nastro: indichiamo con # tale simbolo.

Intuizione: possiamo usare un unico quaderno con anelli per inserire gli appunti di ciascun corso utilizzando un foglio divisorio fra una materia ed un'altra

Per simulare una macchina con k nastri usando una macchina ad un 1 nastro inseriamo un simbolo aggiuntivo # nell'alfabeto (separatore)

MdT con 1 e con k nastri

Simulare una MdT T con k nastri con una MdT T' con un solo nastro


La macchina T' simulerà ogni istruzione di T scorrendo i k nastri e "raccogliendo" le informazioni relative ai k simboli letti dalle k testine.

Terminata questa prima scansione, T' sarà in grado di decidere quale transizione applicare, per cui scorrerà nuovamente i k nastri applicando su ciascuno di essi la corretta operazione di scrittura e di spostamento della testina.

Al termine di questa seconda scansione, T' riposizionerà la testina all'inizio del primo nastro e sarà così pronta a simulare la successiva istruzione.

Un limite delle Macchine di Turing (MdT):

Le MdT sono "hardwired"



Le MdT eseguono un solo programma (codificato nella funzione di transizione)

I computer che usiamo sono programmabili (eseguono diversi programmi)

Soluzione: **La macchina di Turing universale**

Attributi:

- Riprogrammabile
- Simula ogni altra Macchina di Turing

Nota: Turing ha ideato la macchina universale nel 1936 (ben prima dell'introduzione dei computer) aveva all'epoca 24 anni!

La macchina di Turing universale U

Simula ogni altra Macchina di Turing

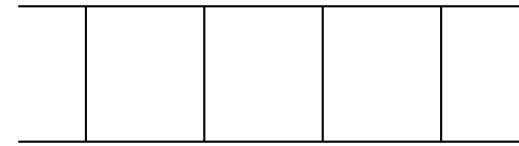
Input della macchina di Turing universale

1. DM - descrizione di M -
una macchina di Turing (ad un nastro)
2. I - stringa di input di M

Output calcolo di M con input I

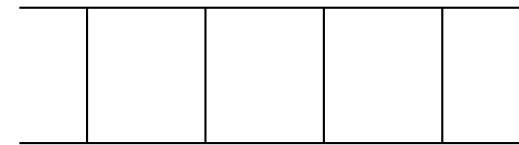
tre nastri: all'inizio

Nastro 1



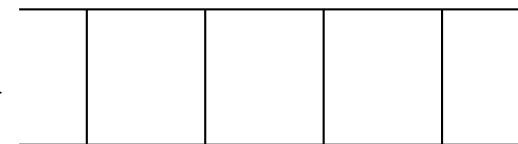
Descrizione di M
e dell'Input I

Nastro 2



vuoto

Nastro 3



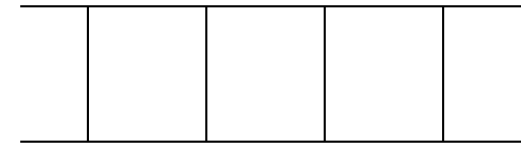
vuoto

Macchina
Universale



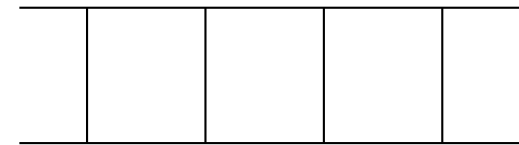
Prima di vedere come opera la
macchina universale
Dobbiamo vedere come
codifichiamo le informazioni
presenti sui tre nastri

Nastro 1



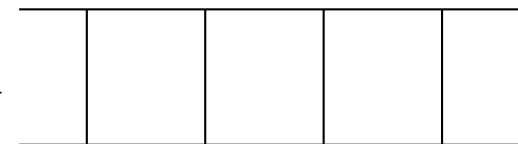
Descrizione di M

Nastro 2



Stato attuale di M

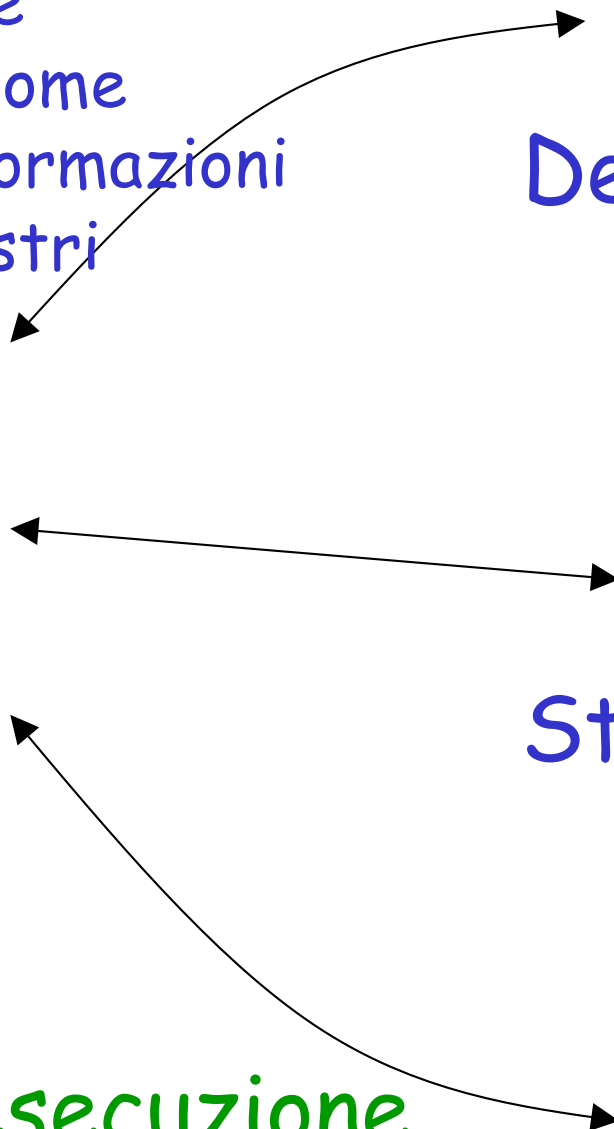
Nastro 3



Nastro di M

U -
Macchina
Universale

tre nastri: in esecuzione



Descrizione nastro M
come una stringa di simboli



Descriviamo (codifichiamo) il nastro di M
come una stringa di simboli :

Assumi alfabeto di M sia $\{0,1, \square\}$

Codifichiamo

0 con Z 1 con U \square con B

Quindi se nastro M contiene $\square 01100\square$

Terzo nastro di macchina universale è BZUZZB

Stato di M

Nastro 2

--	--	--	--	--

Descriviamo (codifichiamo) gli stati in cui si trova M come una stringa di simboli :

Assumiamo M abbia stati $q_0, q_1, q_2, \dots, q_n$
e che q_0 sia lo stato iniziale e q_n lo stato finale

Codifichiamo gli stati $q_0, q_1, q_2, \dots, q_n$
con rappresentazione binaria

q_0 con 0, q_1 con 1, q_2 con 10, q_3 con 11,

Descrizione di M

Nastro 1

--	--	--	--	--

Descriviamo (codifichiamo) M
come una stringa di simboli:

- I movimenti della testina sono così codificati

Sinistra L Destra R Nessun movimento S

- Gli stati di M sono codificati in binario
- I caratteri del nastro di M sono codificati con
Z, U, B

In questo modo quando leggiamo sui nastri della
macchina universale sappiamo a cosa ci si riferisce

Nastro 1

--	--	--	--	--

Descrizione di M

Descriviamo (codifichiamo) M

come una stringa di simboli : riassunto

Alfabeto di macchina universale

- Z, U, B (per codificare alfabeto di M)
- $0, 1$ (per codificare gli stati di M)
- R, L, S (per codificare gli spostamenti)

Inoltre abbiamo bisogno di un simbolo aggiuntivo

- $;$ (utilizzato per separare sul primo nastro la descrizione di M dall'input)

Descrizione di M

Nastro 1

--	--	--	--	--

Alfabeto di macchina universale

Z, U, B (per codificare alfabeto di M)

0,1 (per codificare gli stati di M)

R, L, S (per codificare gli spostamenti)

stato	simbolo	stato	simbolo	movimento	codifica
q0	0	q0	1	R	0Z0UR
q0	1	q0	0	R	0U0ZR
q0	□	q2	□	R	0B10BR
q2	0	q2	0	L	10Z10ZL
q2	1	q2	1	L	10U10UL
q2	□	q1	□	R	10B1BR

Codifica COMPLETA macchina è

0Z0UR0U0ZR0B10BR10Z10ZL10U10UL10B1BR

stato	simbolo	stato	simbolo	movimento	codifica
q0	0	q0	1	R	0Z0UR
q0	1	q0	0	R	0U0ZR
q0	□	q2	□	R	0B10BR
q2	0	q2	0	L	10Z10ZL
q2	1	q2	1	L	10U10UL
q2	□	q1	□	R	10B1BR

Codifica: 0Z0UR0U0ZR0B10BR10Z10ZL10U10UL10B1BR

Stessa macchina codifica diversa

0B11BL0U0ZROZO....


stato	simbolo	stato	simbolo	movimento	codifica
q0	□	q2	□	R	0B10BR
q0	1	q0	0	R	0U0ZR
q0	0	q0	1	R	0Z0UR
q2	□	q1	□	R	10B1BR
q2	1	q2	1	L	10U10UL
q2	0	q2	0	L	10Z10ZL

Passi Macchina Universale

1. Copia sul terzo nastro l'input x codificato mediante i simboli U e Z .
2. Inizializza il contenuto del secondo nastro con la codifica dello stato iniziale di M .
3. In base allo stato (contenuto nel secondo nastro) e al simbolo letto (terzo nastro), cerca sul primo nastro una transizione che possa essere applicata. Se tale transizione non viene trovata, termina in una configurazione di rigetto.
4. Altrimenti, applica la transizione modificando il contenuto del terzo nastro e aggiornando il secondo nastro in base al nuovo stato di M .
5. Se il nuovo stato è uno stato finale di M , termina nell'unico stato finale di U . Altrimenti, torna al Passo 3.

passo 1: Copia sul terzo nastro l'input x codificato mediante i simboli U e Z .

1. Posiziona la testina sul primo simbolo alla destra del simbolo " ; " il quale viene cancellato.

2. Scorre il primo nastro verso destra e, per ogni simbolo diverso da , copia U o Z sul terzo nastro (spostando la testina di questo nastro a destra) e lo cancella.

3. Posiziona la testina del primo nastro e quella del terzo nastro sul simbolo diverso da più a sinistra.

Passo 2: Inizializza il contenuto del secondo nastro con la codifica dello stato iniziale di T.

il secondo passo inizializza il contenuto del secondo nastro con la codifica dello stato iniziale, **deve semplicemente scrivere su tale nastro il simbolo 0 e posizionare la testina su di esso**

Ricordiamo che in base alle assunzioni fatte nel paragrafo precedente, lo stato iniziale di una qualunque macchina di Turing è quello di indice 0.

La macchina è pronta per iniziare la simulazione

Passo 3: In base allo stato (nastro 2) e al simbolo letto (nastro 3), cerca sul primo nastro una transizione da applicare

Cerca sul primo nastro la prima occorrenza del contenuto del secondo nastro (stato).

Una volta trovata tale occorrenza la ricerca prosegue verificando se il simbolo immediatamente successivo è uguale al simbolo attualmente scandito sul terzo nastro: in tal caso, posiziona la testina sulla seconda parte della transizione appena identificata.

Altrimenti, prosegui cercando sul primo nastro un'altra occorrenza del contenuto del secondo nastro.

Se si scorre tutto il primo nastro e non si trova la transizione ERRORE

Passo 4: Copia sul terzo nastro l'input x codificato mediante i simboli U e Z .

1. Cancella l'intero contenuto del secondo nastro e copia su di esso il nuovo stato (sequenza di simboli 0 e 1 contenuta sul primo nastro e il cui primo simbolo è attualmente scandito)
2. Sostituisci il simbolo attualmente scandito sul terzo nastro con quello presente sul primo nastro e sposta la testina di quest'ultimo nastro di una posizione a destra.
3. In base al simbolo letto sul primo nastro, sposta la testina del terzo nastro (se il nuovo simbolo letto su tale nastro è \square sostituiscilo con il simbolo B)
4. Posiziona la testina del primo nastro sul primo simbolo a sinistra diverso da \square (preparandosi per prossimo passo)

Passo 5: Se il nuovo stato è uno stato finale di M , termina nell'unico stato finale di U .
Altrimenti, torna al Passo 3.

Ricordiamo che siamo in stato finale di M se il contenuto del secondo nastro è uguale alla stringa 1.

1. Se siamo in stato finale cancella il contenuto del primo nastro e copia l'output della macchina simulata, decodificato facendo uso di simboli 0 e 1.

2. Se non è così la macchina di Turing universale torna a eseguire il terzo passo

Sulle dispense la macchina U è data nei dettagli

Osservazioni

1. U è una Macchina di Turing in grado di simulare ogni altra macchina di Turing

2. U ha 28 stati (!) ed è in grado di simulare una qualunque macchina di Turing (anche una con un milione di stati)

Assurdo?

1. Sappiamo che esistono interpreti C scritti in C

2. Un interprete C ha dimensione finita ed è in grado di eseguire un programma C di lunghezza qualunque

Sulle dispense la macchina U è data nei dettagli

Osservazioni

1. U è una Macchina di Turing in grado di simulare ogni altra macchina di Turing
2. U ha 28 stati (!) ed è in grado di simulare una qualunque macchina di Turing (anche una con un milione di stati)

Assurdo?

1. Sappiamo che esistono interpreti C scritti in C
2. Un interprete C ha dimensione finita ed è in grado di eseguire un programma C di lunghezza qualunque

Il problema della fermata vale per ogni linguaggio di programmazione o modello di calcolo noto. Quindi il risultato di indecidibilità ha valore oltre l'informatica: si riferisce alla nostra capacità di calcolare

Domanda: il problema è importante in pratica?

In fin dei conti i programmatori hanno risolto il problema per i programmi che usiamo in pratica. Possiamo dire che il problema della fermata è risolubile in tutti i casi pratici? (e allora chi se ne frega)

Due osservazioni

- Un programma che risolve il problema della fermata farebbe risparmiare tempo quando i nostri programmi ciclano (magari indicando il punto in cui cicla) COMODO
- Più in generale esistono altri problemi indecidibili che pongono limiti a quello che possiamo programmare

Congettura di Goldbach

uno dei più vecchi problemi irrisolti nella teoria dei numeri (dal 1742). Essa afferma che ogni numero pari maggiore di 2 può essere scritto come somma di due numeri primi (anche uguali).

Python any (all)

any(iterable)

Ritorna True se un qualunque elemento di *iterable* è true. Se *iterable* è vuoto ritorna False

all(iterable)

Ritorna True se ciascun elemento di *iterable* è True. Se *iterable* è vuoto ritorna True

Programma seguente si ferma se e solo se la congettura di Goldbach è falsa.

```
def isprime(p):  
    return all(p % i for i in range(2,p-1))  
def Goldbach(n):  
    return any( (isprime(p) and isprime(n-p))  
               for p in range(2,n-1))  
  
n = 4  
while True:  
    if not Goldbach(n): break  
    n+= 2
```

se sapessi risolvere problema fermata avrei
risolto la congettura di Goldbach

Linguaggi decidibili e non decidibili

Un linguaggio L è decidibile se la sua funzione caratteristica X_L è calcolabile.

$X_L(y) = 1$ se y appartiene a L

$X_L(y) = 0$ altrimenti

Se X_L non è calcolabile L è indecidibile

Esempio Linguaggio derivato da problema fermata MdT (M si ferma con input x)?

$X_{\text{stop}}(c, x) = 1$ se MdT codificata da c si ferma con input x

$X_{\text{stop}}(c, x) = 0$ altrimenti

Linguaggi decidibili e non decidibili

Un linguaggio L è **decidibile** se la sua funzione caratteristica X_L è calcolabile.

$X_L(y) = 1$ se y appartiene a L

$X_L(y) = 0$ altrimenti

Esistono linguaggi non decidibili

X_{stop} (linguaggio derivato dal problema della fermata) è indecidibile

Linguaggi decidibili e non decidibili

Un linguaggio L è **semi-decidibile** se esiste una macchina di Turing T tale che, per ogni stringa binaria y , se y appartiene a L , allora $T(y)$ termina in una configurazione finale, altrimenti $T(y)$ non sappiamo se termina o meno

Teorema L_{Stop} è semidecidibile

Prova: esiste Macchina di Turing universale

Riduzione fra da problema Fermata a
problema Fermata_input_0

Problema Fermata_input_0

Input: una stringa che descrive una MdT M

Output: 1 se M si ferma con input 0

Apparentemente Fermata_input_0
sembra più semplice di Fermata
(dobbiamo risolvere solo per un input)

Riduzione da problema Fermata a
problema Fermata_input_0

Algoritmo: (obiettivo risolvere: Fermata
assumendo che esista alg. $A(M)$ che risolve
Fermata_input_0 per ogni M)

Input: M, x ; Output Fermata(M, x)

1. Sia $N_{M,x}$ la MdT che risolve il seguente: "su
input $z \in \{0,1\}^*$ valuta M su input x e ritorna il
risultato" (Nota si ignora z e x è nella codifica
della macchina di Turing)
2. Ritorna $y = A(N_{M,x})$ su input 0.

Riduzione da problema Fermata a
problema Fermata_input_0

Algoritmo: (obiettivo risolvere: Fermata
assumendo che esista alg. $A(M)$ che risolve
Fermata_input_0 per ogni M)

Una riduzione da problema A a problema B
mostra che se sapessi risolvere B allora risolvo
anche problema A . Quindi B non è più facile di A

Per fare questo mostriamo che

per ogni istanza P di A esiste una istanza Q di B
la cui soluzione mi fornisce la risposta a P

Riduzione da problema Fermata a problema Fermata_input_0

Prova formale: idea (difficoltà principale) è pensare che in $N_{M,x}$ x non è input di M ma è una nuova costante inserita nel programma che definisce una macchina di Turing diversa da M .

L'algoritmo NON esegue $N_{M,x}$ ma semplicemente scrive la descrizione di $N_{M,x}$ come una stringa e fornisce questa stringa in input a A (che risolve il prob. Fermata_input_0)

Lemma: data la stringa M,x,z la macchina $N_{M,x}$ si ferma su input z se e solo se M si ferma con input x

Riduzione da problema Fermata a problema Fermata_input_0

Lemma: data la stringa M, x, z la macchina $N_{M,x}$ si ferma su input z se e solo se M si ferma con input x

Prova: $N_{M,x}$ ignora input z (fa la stessa cosa qualunque sia z); $N_{M,x}$ si ferma se e solo se M si ferma su input x

In altre parole

$$\text{Fermata}(M, x) = \text{Fermata_input_0}(N_{M,x})$$

NB: la descrizione di M e del suo input sono stringhe di bit; x in questo caso NON e' input di una MdT ma parte della descrizione di $N_{M,x}$

IMPORTANTE: per dimostrare
indecidibilità la riduzione va da problema A
che so indecidibile a problema B che voglio
dimostrare indecidibile

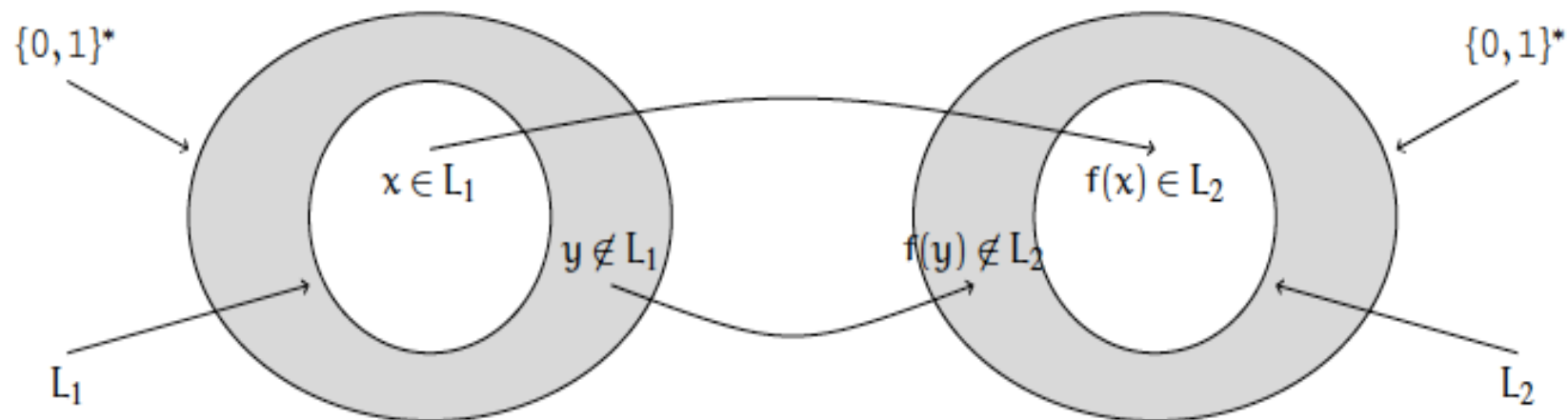
La riduzione mostra che se sapessi risolvere
 B allora risolvo anche problema A

Quindi B non è più facile di A

La riduzione nella direzione opposta non
fornisce nulla (mi dice che A non è più facile
di B ma non dice nulla su quanto sia difficile
 B)

Un linguaggio L_1 è riducibile a un linguaggio L_2 se esiste una funzione totale calcolabile $f : \{0,1\}^* \rightarrow \{0,1\}^*$, detta riduzione, tale che, per ogni stringa binaria x , $x \in L_1$ se e solo se $f(x) \in L_2$ (x in L_1 se e solo se $f(x)$ in L_2)

Figura 3.4: riducibilità tra linguaggi



Siano $L1$ e $L2$ due linguaggi tali che $L1$ è riducibile a $L2$.

1- Se $L2$ è decidibile, allora $L1$ è decidibile.

2- Se $L1$ non è decidibile, allora $L2$ non è decidibile.

Intuizione se $L1$ riducibile a $L2$ allora

- $L1$ non è più difficile di $L2$ (vedi 1 sopra)
- $L2$ è almeno tanto difficile quanto $L1$ (2 sopra)

Risultato_sempre_0: data MdT M output 1 se e solo se M da in output 0 per ogni input

Linguaggio associato:

$L_{\text{sempre}_0} = \{\text{codifica di } M \text{ tale che output di } M \text{ è } 0 \text{ per ogni input}\}$

Teorema: Risultato_sempre_0 è indecidibile

Prova: Riduzione da Fermata_input_0 a problema Risultato_sempre_0

NB: la riduzione va da problema che so indecidibile a problema che voglio dimostrare indecidibile

Esempi di riduzione

Esempio 3.6: problema della terminazione con input fissato

Consideriamo il seguente linguaggio: $L_{\text{stop}-0} = \{c_T : c_T \in \mathcal{C} \wedge T(0) \text{ termina}\}$. Dimostriamo ora che L_{stop} è riducibile a $L_{\text{stop}-0}$: dal Teorema 3.5 e dal Corollario 3.2, segue che $L_{\text{stop}-0}$ non è decidibile. Data una stringa binaria y , la riduzione f per prima cosa verifica se $y = \langle c_T, x \rangle$ con $c_T \in \mathcal{C}$: in caso contrario (per cui $y \notin L_{\text{stop}}$), $f(y)$ produce la codifica di una qualunque macchina di Turing che con input 0 non termina (per cui $f(y) \notin L_{\text{stop}-0}$). Altrimenti, $f(\langle c_T, x \rangle)$ produce la codifica $c_{T'}$ di una macchina di Turing T' che, con input una stringa binaria z , per prima cosa cancella z e lo sostituisce con x e, quindi, esegue T con input x . Chiaramente, $\langle c_T, x \rangle \in L_{\text{stop}}$ se e solo se $c_{T'} = f(\langle c_T, x \rangle) \in L_{\text{stop}-0}$.

Esempi di riduzione

Esempio 3.7: problema dell'accettazione

Consideriamo il seguente linguaggio: $L_{acc} = \{\langle c_T, x \rangle : c_T \in \mathcal{C} \wedge T(x) \text{ termina in una configurazione finale}\}$. Dimostriamo ora che L_{stop} è riducibile a L_{acc} : dal Teorema 3.5 e dal Corollario 3.2, segue che L_{acc} non è decidibile. Date due stringhe binarie $c_T \in \mathcal{C}$ e x , $f(\langle c_T, x \rangle)$ produce la coppia $\langle c_{T'}, x \rangle$ dove $c_{T'}$ è la codifica di una macchina di Turing T' che, con input una stringa binaria z , esegue T con input z : se $T(z)$ termina, allora T' termina in una configurazione finale. Chiaramente, $\langle c_T, x \rangle \in L_{stop}$ se e solo se $T'(x)$ termina in una configurazione finale se e solo se $\langle c_{T'}, x \rangle = f(\langle c_T, x \rangle) \in L_{acc}$.

Esempi di riduzione

Esempio 3.8: problema del linguaggio vuoto

Consideriamo il seguente linguaggio: $L_{\text{empty}} = \{c_T : c_T \in \mathcal{C} \wedge \forall x \in \{0, 1\}^* [T(x) \text{ non termina in una configurazione finale}]\}$. Dimostriamo ora che L_{acc} è riducibile a L_{empty}^c : dall'esempio precedente, dal Corollario 3.2 e dal Teorema 3.1, segue che L_{empty}^c e L_{empty} non sono decidibili. Date due stringhe binarie $c_T \in \mathcal{C}$ e x , $f(\langle c_T, x \rangle)$ produce la codifica $c_{T'}$ di una macchina di Turing T' che, con input una stringa binaria y , per prima cosa cancella y e lo sostituisce con x e, quindi, esegue T con input x . Abbiamo che $\langle c_T, x \rangle \in L_{\text{acc}}$ se e solo se $T(x)$ termina in una configurazione finale se e solo se, per ogni stringa binaria y , $T'(y)$ termina in una configurazione finale se e solo se $c_{T'} \in L_{\text{empty}}^c$ (in quanto T' si comporta allo stesso modo indipendentemente dalla stringa di input y).

Esempi di riduzione

Esempio 3.9: problema dell'equivalenza tra linguaggi

Consideriamo il seguente linguaggio.

$$L_{eq} = \{ \langle c_{T_1}, c_{T_2} \rangle \quad : \quad c_{T_1} \in \mathcal{C} \wedge c_{T_2} \in \mathcal{C} \wedge \forall x \in \{0, 1\}^* [T_1(x) \text{ termina in una configurazione finale se e solo se } T_2(x) \text{ termina in una configurazione finale}] \}$$

(in altre parole, L_{eq} include tutte le coppie di codifiche di macchine di Turing che decidono lo stesso linguaggio). Riduciamo ora L_{empty} a L_{eq} : dall'esempio precedente e dal Corollario 3.2, segue che L_{eq} non è decidibile. Data una stringa binaria $c_T \in \mathcal{C}$, $f(c_T)$ produce la coppia $\langle c_T, c_R \rangle$ dove c_R è la codifica di una macchina di Turing R che non accetta alcuna stringa. Chiaramente, $c_T \in L_{empty}$ se e solo $\langle c_T, c_R \rangle \in L_{eq}$.

RAM (Random Access Machines, anni '60)
Un modello di calcolo ispirato ai calcolatori

Poche istruzioni:

- INCrementa di 1 il contenuto del registro r
- DECrementa di 1 il contenuto del registro r
- IF register $r = 0$ THEN Salta all'istruzione I
ELSE continua con la prossima istruzione

RAM equivalenti a MdT, Java

Un po' di storia: Hilbert

Hilbert (fine '800) era interessato ai fondamenti della matematica e si pose due domande

- *E' possibile che ogni proposizione matematica possa essere dimostrata vera o falsa?"*
- *E' possibile identificare un metodo automatico per provare la verità o la falsità di una proposizione matematica*

Un po' di storia: Hilbert, Gödel

Hilbert (fine '800) era interessato ai fondamenti della matematica e si pose due domande

E' possibile che ogni proposizione matematica possa essere dimostrata vera o falsa?"

- Gödel dimostrò che esistono proposizioni matematiche (dell'aritmetica) che sono indecidibili, cioè non possono essere dimostrate che siano vere o che siano false!!
- La prova dell'ind decidibilità del problema della fermata di Turing è ispirata alla prova di Gödel

Un po' di storia: Hilbert, Turing

“E' possibile identificare un metodo automatico per provare la verità o la falsità di una proposizione matematica”

- Alla luce del risultato di Gödel la domanda deve essere riformulata cercando di stabilire un metodo algoritmo universale per i problemi che possiamo risolvere
- In altre parole (realizzare la nostra intuizione di calcolo)
- In questo contesto Turing propose il suo modello di calcolo

Un po' di storia, Church, Markov ed altri

- Contemporaneamente a Turing Church ha introdotto un concetto di calcolabilità basato su un sistema algoritmico (λ -calcolo)
- Tale sistema è stato dimostrato equivalente alle macchine di Turing, dallo stesso Turing.
- Ciò ha permesso di formulare la Tesi di Church-Turing:
Ogni funzione che sia dimostrata calcolabile con un qualunque modello di calcolo formale è calcolabile secondo Turing (o il λ -calcolo)

La tesi di Church Turing non si può dimostrare

- ***in teoria è sempre possibile trovare un modello di calcolo più potente delle Macchine di Turing***

Un po' di storia, Church, Markov ed altri

- Ciò ha permesso di formulare la Tesi di Church-Turing:
Ogni funzione che sia dimostrata calcolabile con un qualunque modello di calcolo formale è calcolabile secondo Turing
- ***La tesi di Church Turing non si può dimostrare MA***
- sono stati proposti molti modelli di calcolo (Markov, Kleene, Post, e altri) basati su approcci diversi.
- Tutti i modelli proposti sono equivalenti alle MdT

Sulla base di questi risultati oggi è convinzione comune

- ***che la tesi di Church-Turing sia vera e***
- ***che non esista un modello di calcolo più potente delle MdT***

Random Access Machine:

modello di calcolo simile a computer

Ha un numero potenzialmente infinito di *registri* e un *contatore di programma*

- Ogni registro è individuato da un indirizzo intero
- Ogni registro può contenere un numero intero arbitrariamente grande: n denota un intero, (n) denota il contenuto del registro il cui indirizzo è n ; $[n]$ denota il contenuto del registro il cui indirizzo è contenuto nel registro il cui indirizzo è n

Programma RAM: Sequenza di istruzioni (eventualmente etichettate); le istruzioni sono:

- $(n):=o1 \text{ operatore } o2$ oppure $[n]:=o1 \text{ operatore } o2$ dove operatore è $\{+, -, *, /\}$ e $o1$ e $o2$ possono essere m , (m) o $[m]$
- $\text{if } o1 \text{ operatore } o2 \text{ goto etichetta}$ dove operatore è $\{=, <>, <=, <\}$ e $o1$ e $o2$ possono essere m , (m) o $[m]$
- **end**

Esecuzione di un programma RAM

- Inizialmente i primi n registri contengono n valori interi, tutti altri registri contengono 0 e il contatore di programma è uguale a 0
- A ogni passo l'istruzione indicata dal contatore di programma viene eseguita
 - Se l'istruzione non è if, il contatore di programma aumenta di 1
 - Altrimenti, se la condizione non è verificata, il contatore di programma aumenta di 1
 - Altrimenti, il contatore di programma viene posto uguale all'indice dell'istruzione corrispondente all'etichetta
- Istruzione end termina esecuzione
 - Output contenuto nel registro 0

Esempio

Decidere se una sequenza di 10 numeri interi è palindroma (es. 1 0 0 1 è palindromo, 1 0 3 0 no)

- i 10 numeri sono nelle celle 0,1,2,...9
- La locazione di memoria 10 contiene all'inizio 0
- Output 1: sequenza palindroma, 0 altrimenti

```
      (11) := 9 + 0
loop:  if (11) <= (10) goto yes
      if [10] <> [11] goto no
      (10) := (10) + 1
      (11) := (11) - 1
      if 0 = 0 goto loop
yes:   (0) := 1 + 0
      end
no:    (0) := 0 + 0
end
```

RAM (Random Access Machines, anni '60)
Un modello di calcolo ispirato ai calcolatori

Poche istruzioni:

- INCrementa di 1 il contenuto del registro r
- DECrementa di 1 il contenuto del registro r
- IF register $r = 0$ THEN Salta all'istruzione I
ELSE continua con la prossima istruzione

RAM equivalenti a MdT, Java

Tesi di Church Turing

- Si può dimostrare che RAM e Macchine di Turing sono modelli di calcolo equivalenti (per ogni MdT esiste un programma RAM equivalente e viceversa)
- Modello di calcolo almeno potente quanto un linguaggio macchina

Linguaggi di programmazione

- Ogni programma scritto in Java può essere tradotto in linguaggio macchina (Compito dei compilatori)
- Se dimostriamo che esistono funzioni non calcolabili da un linguaggio macchina, allora abbiamo che esistono funzioni non calcolabili da un programma Java (Python, C,...)

Esercizi

Dimostrare che se L oppure L^c è un linguaggio finito, allora L e L^c sono due linguaggi decidibili.

Dimostrare che se $L1$ e $L2$ sono due linguaggi decidibili, allora anche $L1$ unione $L2$ e $L1$ intersezione $L2$ sono decidibili.

Disegnare una MdT con due nastri che verifichi se una sequenza di parentesi (e) sono ben bilanciate (es. $((()))$ SI; $((()))($ NO)