Macchine di Turing: prime varianti

12 aprile 2022

Varianti della MdT

Esistono diverse varianti della definizione di macchina di Turing deterministica.

Vedremo:

- la Macchina di Turing multinastro
- la Macchina di Turing non deterministica

Tali macchine di Turing hanno lo stesso potere computazionale (o espressivo) delle MdT deterministiche, cioè riconoscono la stessa classe di linguaggi.

Esistono anche altre varianti della macchina di Turing deterministica che hanno lo stesso potere espressivo.

Chiamiamo "robustezza" questa invarianza ad alcune variazioni nella definizione. Essa è una conferma che si tratta di un buon modello per la definizione di algoritmo.

Equivalenza di modelli

Siano $\mathcal{T}_1$ e $\mathcal{T}_2$ due famiglie di modelli computazionali. Per dimostrare che i modelli in $\mathcal{T}_1$ hanno lo stesso potere computazionale dei modelli in $\mathcal{T}_2$ occorre far vedere che per ogni macchina $M_1 \in \mathcal{T}_1$ esiste $M_2 \in \mathcal{T}_2$ equivalente ad M_1 e viceversa.

Abbiamo già dimostrato che la classe dei DFA ha lo stesso potere computazionale della classe degli NFA.

Come per le classi dei DFA e degli NFA, consiste spesso nel dimostrare che per ogni macchina di una classe ne esiste una dell'altra classe capace di simularla.

Come nel caso dei DFA e degli NFA, in genere una delle direzioni della prova è evidente.

Equivalenza di modelli

Per illustrare la robustezza del modello di macchina di Turing cerchiamo di variare il tipo di funzione di transizione consentito.

Nella nostra definizione, la funzione di transizione forza la testina a spostarsi verso sinistra o destra ad ogni passo; la testina non può restare ferma.

Supponiamo di aver permesso alla macchina di Turing la capacità di restare ferma.

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

Equivalenza di modelli

Questa caratteristica **non** permette alle macchine di Turing di riconoscere ulteriori linguaggi, cioè non aggiunge potere computazionale al modello scelto.

Per poter fare questa affermazione occorre trovare per ogni macchina di un tipo una equivalente dell'altro tipo.

La parte "non ovvia" è mostrare che possiamo trasformare qualsiasi macchina di Turing che ha la possibilità di "restar ferma" in una macchina di Turing equivalente che non ha tale capacità.

Lo facciamo costruendo una MdT in cui sostituiamo ogni transizione "resta ferma" con due transizioni, una che sposta la testina a destra e una che la riporta a sinistra.

Equivalenza di modelli

Formalizziamo questo discorso.

Chiamiamo $\mathcal{T}_{(L,R)}$ l'insieme delle macchine di Turing che abbiamo definito, cioè quelle con funzione di transizione

$$\delta : (Q \setminus \{q_{accept}, q_{reject}\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

$\mathcal{T}_{(L,R,S)}$ è l'insieme delle macchine M tali che
$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ è una settupla in cui
$Q, \Sigma, \Gamma, q_0, q_{accept}, q_{reject}$ sono definiti come in una MdT deterministica e la funzione di transizione δ è definita al modo seguente:

$$\delta : (Q \setminus \{q_{accept}, q_{reject}\}) \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$$

Equivalenza di modelli

Se $\delta(q, \gamma) = (q', \gamma', d)$ e se M si trova nello stato q con la testina posizionata su una cella contenente γ, alla fine della transizione:

- M si troverà nello stato q',
- $\gamma' \in \Gamma$ sarà il simbolo scritto sulla cella del nastro su cui la testina si trovava all'inizio della transizione (contenente γ),
- la testina si troverà sulla stessa cella cui si trovava all'inizio della computazione se $d = S$, si sarà spostata sulla cella di sinistra se (tale cella esiste e se) $d = L$, si sarà spostata sulla cella di destra se $d = R$.

Le nozioni di configurazione, di linguaggio deciso e di linguaggio riconosciuto da $M' \in \mathcal{T}_{(L,R)}$ sono estese in maniera ovvia alle macchine M in $\mathcal{T}_{(L,R,S)}$.

Equivalenza di MdT e Stayer

I modelli in $\mathcal{T}_{(L,R)}$ hanno lo stesso potere computazionale dei modelli in $\mathcal{T}_{(L,R,S)}$.

Prova

Ogni MdT classica, in particolare è uno stayer.

Viceversa se $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject}) \in \mathcal{T}_{(L,R,S)}$ definiamo $M' = (Q \cup Q^c, \Sigma, \Gamma, \delta', q_0, q_{accept}, q_{reject}) \in \mathcal{T}_{(L,R)}$ dove:

- se $\delta(q_i, \gamma) = (q_j, \gamma', d)$ e $d \in \{L, R\}$ allora $\delta'(q_i, \gamma) = \delta(q_i, \gamma)$
- se $\delta(q_i, \gamma) = (q_j, \gamma', S)$ allora $\delta'(q_i, \gamma) = (q_j^c, \gamma', R)$ e $\delta'(q_j^c, \eta) = (q_j, \eta, L)$ per ogni $\eta \in \Gamma$.
- $Q^c = \{q^c \mid (q, \gamma, S) \in \delta((Q \setminus \{q_{accept}, q_{reject}\}) \times \Gamma)\}$.

Anche in questo caso $L(M) = L(M')$.

Altra variante di MdT

E una MdT con «S = resta ferma» al posto di «L = muovi a sinistra»?

Questa variante NON è equivalente.

L'accesso alla memoria/nastro è restrittivo

Altre varianti equivalenti di MdT

- MdT a sola scrittura (Ex. 3.17)
- MdT a nastro doppiamente infinito (ex. 3.18)
- MdT con Reset a sinistra (Ex. 3.19)

Sono tutte equivalenti alla MdT.

Sono stati proposti molti altri modelli di computazione (+ o – simili alle MdT). Sorprendentemente:

Tutti i modelli «ragionevoli» con un accesso non restrittivo ad una memoria illimitata risultano essere equivalenti.

Equivalenza di modelli di computazione

Tutti i modelli «ragionevoli» con un accesso non restrittivo ad una memoria illimitata risultano essere equivalenti.

I modelli di computazione immaginabili sono tanti, ma la classe di «algoritmi» che essi descrivono rimane la stessa!

Analogia con i linguaggi di programmazione: non esiste u «algoritmo» che può essere programmato in C e non in Java, o viceversa.

I linguaggi di programmazione immaginabili sono tanti, ma la classe di «algoritmi» che essi descrivono rimane la stessa!

Livelli di descrizione di MdT

La MdT è nata come modello preciso di «algoritmo», «procedura effettiva di calcolo».

Tre livelli di descrizione:

1. Descrizione ad alto livello
2. Descrizione implementativa (come muove la testina, memorizza i dati, …)
3. Descrizione formale, precisa (come settupla)

TM – example revisited

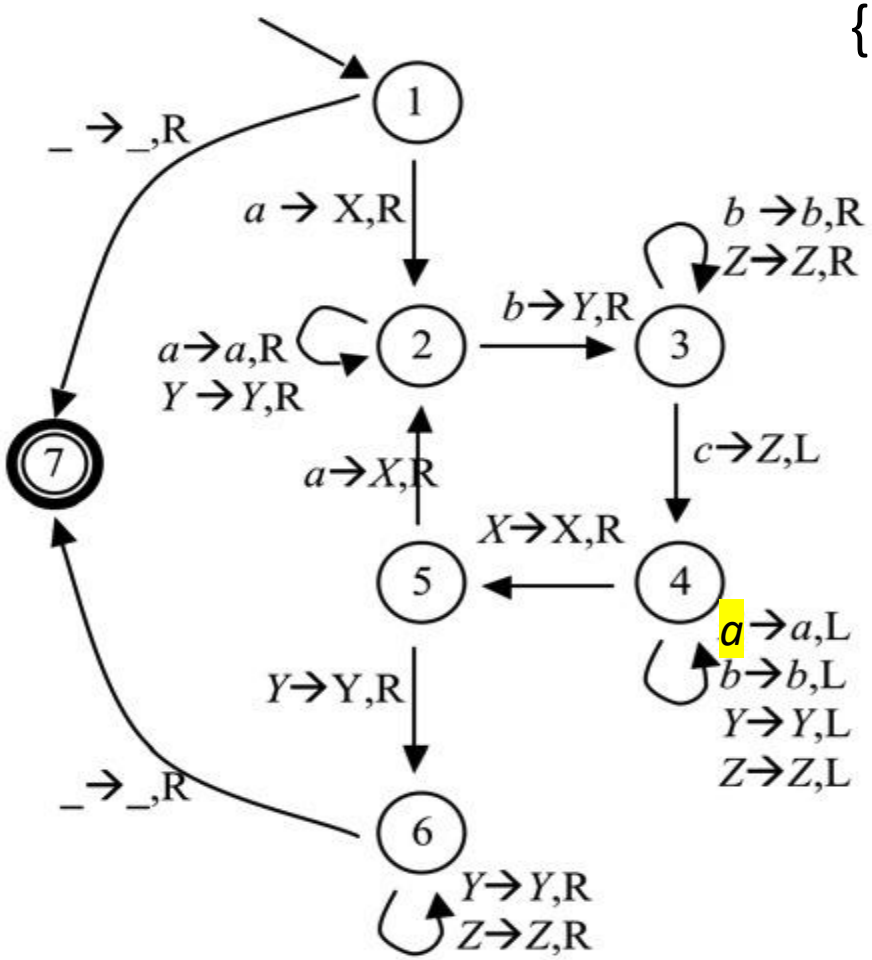
TM M recognizing $B = \{a^k b^k c^k \mid k \geq 0\}$

M = "On input w

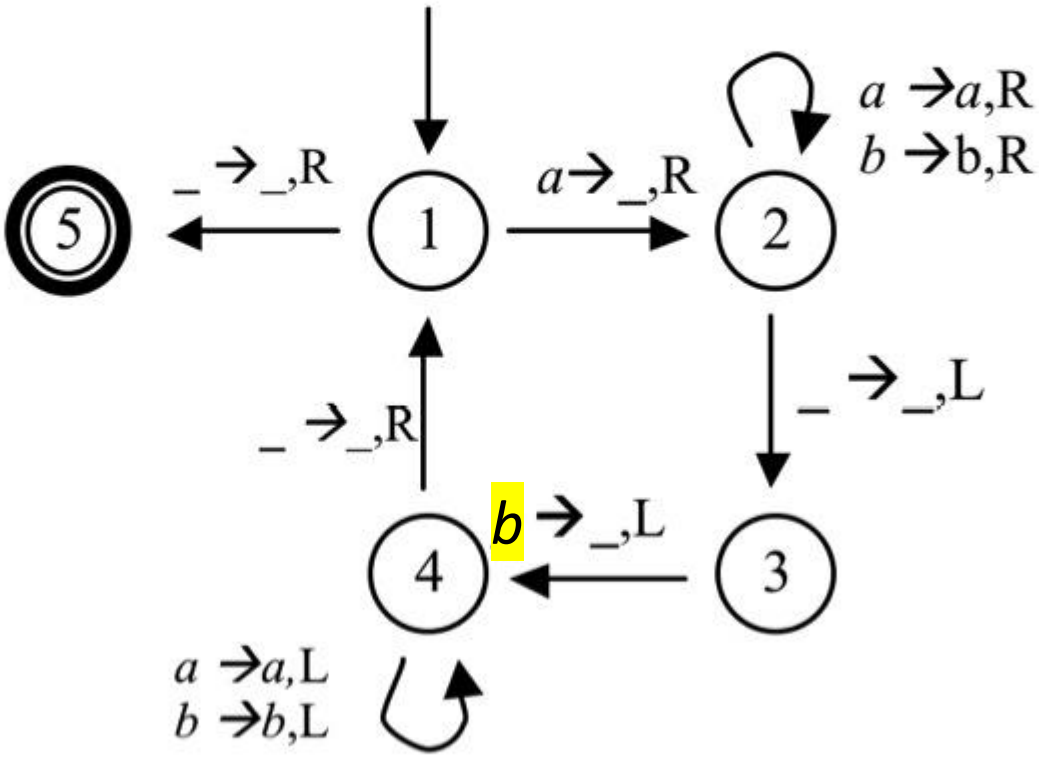
1. Check if $w \in a^* b^* c^*$, *reject* if not.
2. Count the number of a's, b's, and c's in w.
3. *Accept* if all counts are equal; *reject* if not."

High-level description is ok.
You do not need to manage tapes, states, etc…

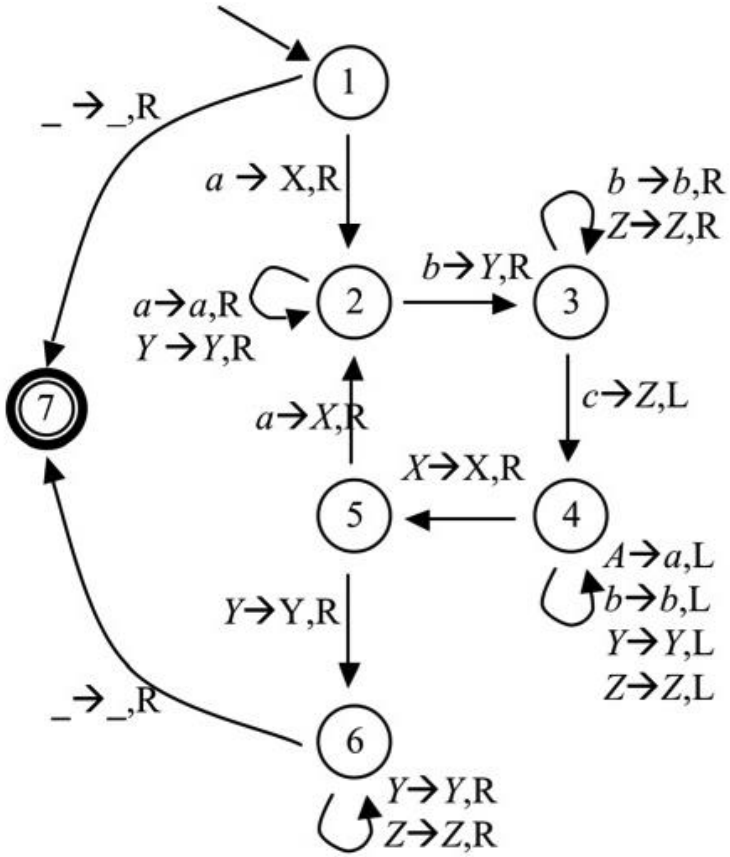
$\{ a^n b^n c^n \}$


1
2
3
4
5
6
7
_ → _,R
a → X,R
b → b,R
Z → Z,R
b → Y,R
a → a,R
Y → Y,R
c → Z,L
a → X,R
X → X,R
a → a,L
b → b,L
Y → Y,L
Z → Z,L
Y → Y,R
_ → _,R
Y → Y,R
Z → Z,R

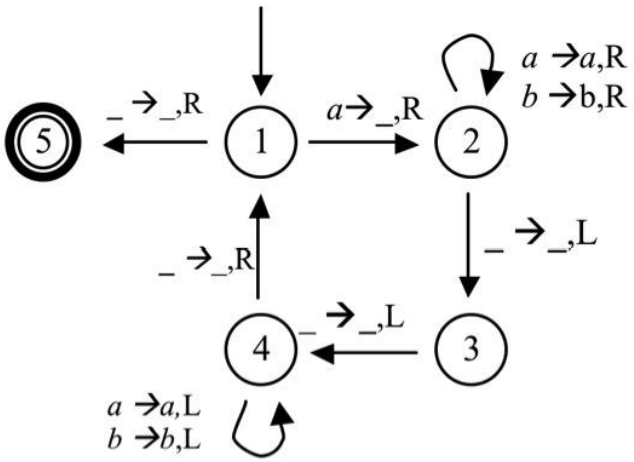
$\{a^n b^n\}$



$\{ a^n b^n c^n \}$



$\{a^n b^n\}$



Strategie analoghe?

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)

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

Esercizio

Progettare una MdT che decida il linguaggio

$$L = \{ 0^{2^n} \mid n \geq 0 \}$$

e che sia **concettualmente diversa** da quella vista

in una precedente lezione.

Discussa a lezione un'altra strategia (diagramma non del tutto corretto).
Esiste pure una terza strategia/algoritmo: sapreste trovarla?

3.8 Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet $\{0,1\}$.

[A] **a.** $\{w \mid w$ contains an equal number of 0s and 1s$\}$

b. $\{w \mid w$ contains twice as many 0s as 1s$\}$

c. $\{w \mid w$ does not contain twice as many 0s as 1s$\}$

Esercizio

Progettare una MdT che calcoli la funzione f(x,y), differenza intera di due interi positivi x e y

$$f(x, y) = x - y \qquad \text{se } x \geq y$$
$$f(x, y) = 0 \qquad \text{altrimenti.}$$

Si supponga che l'input sia $< x > 0 < y >$, dove $< n > = 1^n$, è la rappresentazione unaria dell'intero positivo n.

1. Progettare una macchina di Turing che sposta l'input a destra di una casella.

2. Progettare una macchina di Turing che calcola il successore in binario.

3. Esercizi_ETC_lez2_definizioni.pdf

Un altro esempio

La funzione $f(x) = 2x$ e' calcolabile

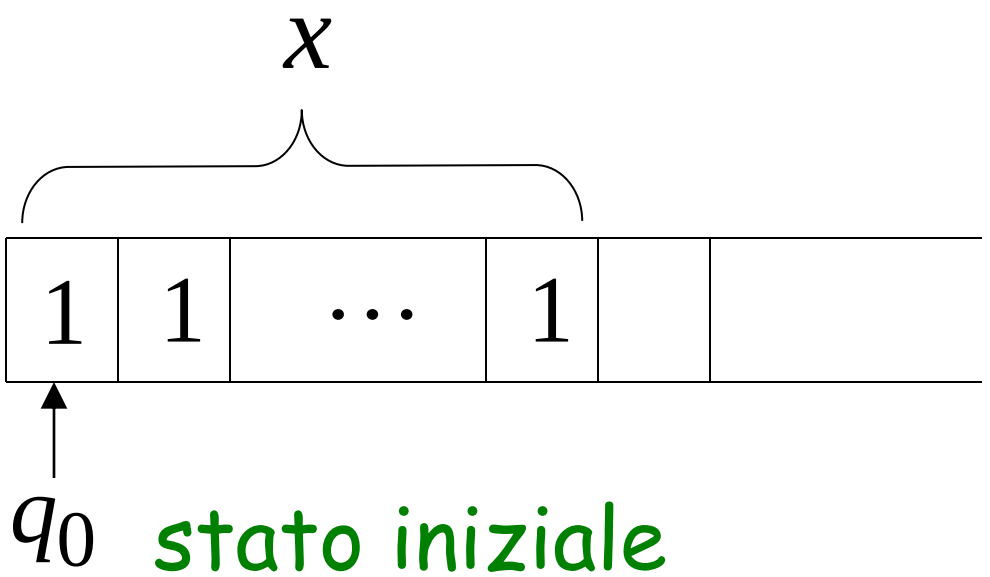
x e' un intero

Macchina di Turing:

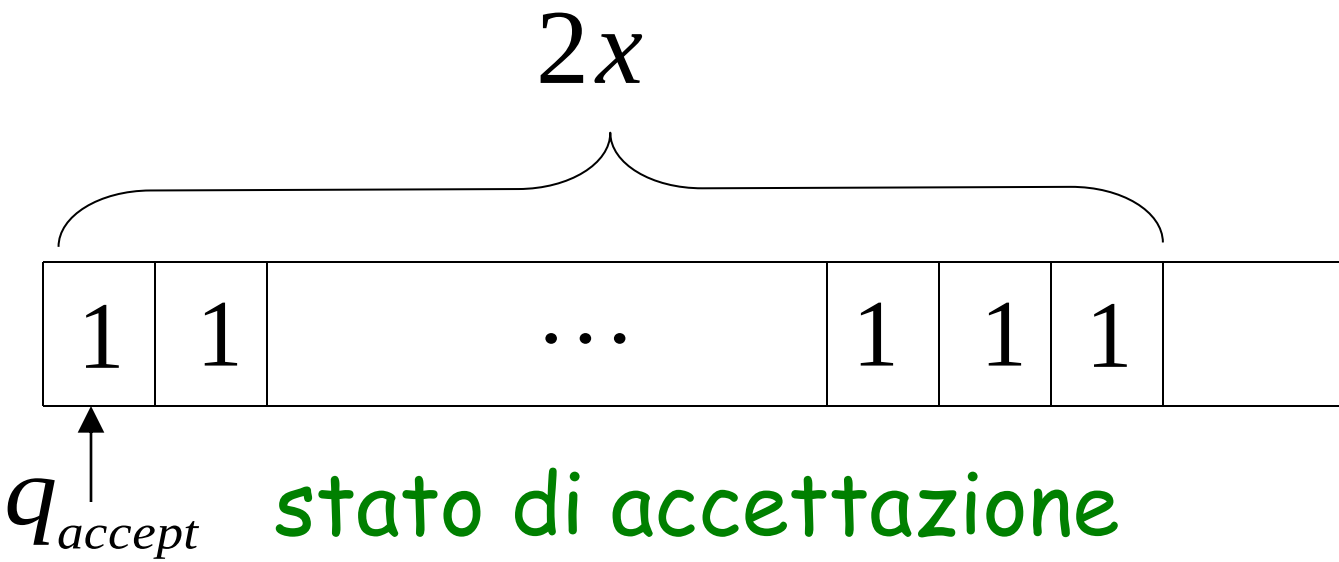
Stringa input: x in notazione unaria

Stringa output: xx in notazione unaria

Inizio


x
1
1
…
1
q_0
stato iniziale

Fine


2x
1
1
…
1
1
1
q_{accept}
stato di accettazione