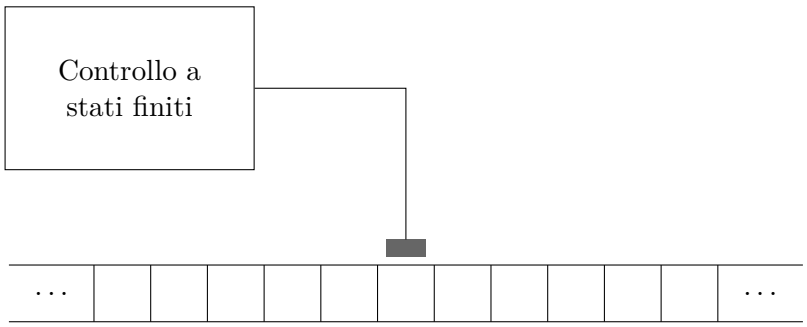
Azzolini Riccardo 2020-12-11

Macchine di Turing

1 Macchine di Turing deterministiche

Una macchina di Turing è sostanzialmente un automa a stati finiti arricchito di una memoria esterna che — a differenza della memoria dei PDA — ha una politica di accesso estremamente liberale (equivalente all'accesso casuale). Più nel dettaglio, una macchina di Turing può essere vista come un *controllo a stati finiti* che comanda una *testina* di lettura e scrittura, la quale scorre su un *nastro* bi-infinito (infinito sia verso sinistra che verso destra). Tale nastro è diviso in *celle*, ciascuna delle quali contiene un simbolo, e il simbolo presente sotto alla testina contribuisce a determinare le mosse della macchina, insieme allo stato interno del controllo a stati finiti.



Formalmente, una **macchina di Turing** (MdT) **deterministica**[1] è una settupla

$$M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$$

in cui:

- Q è l'insieme finito e non vuoto degli **stati**;
- Σ è l'**alfabeto di input**;
- Γ è l'**alfabeto di lavoro**, l'insieme dei simboli che possono comparire nel nastro, che deve includere propriamente l'alfabeto di input, $\Sigma \subset \Gamma$ (perché l'input verrà dato alla macchina scrivendolo sul nastro);
- $q_0 \in Q$ è lo **stato iniziale** della macchina;

[1] A differenza di quanto fatto per gli automi a stati finiti e gli automi a pila, per le MdT verrà presentato solo il modello deterministico.

- $B \in \Gamma \setminus \Sigma$ è il simbolo **blank**, un simbolo appartenente all'alfabeto di lavoro ma non a quello di input (ragion per cui si ha $\Sigma \subset \Gamma$ e non $\Sigma \subseteq \Gamma$) che verrà messo nelle celle del nastro in cui non sono presenti simboli significativi (le celle "vuote");
- $F \subseteq Q$ è l'insieme degli **stati finali** (o **accettanti**);
- δ è la **funzione di transizione**, che:
  - è definita su un *sottoinsieme* di $Q \times \Gamma$, ovvero è una *funzione parziale* (questo semplifica la descrizione delle macchine);
  - laddove è definita, associa a un elemento di $Q \times \Gamma$ un elemento di $Q \times \Gamma \times \{\mathtt{L}, \mathtt{R}\}$.

Per semplificare il modello, si assume che *la funzione di transizione sia indefinita sugli stati accettanti*. Come si vedrà dopo, ciò farà sì che la computazione della macchina si arresti quando questa arriva in uno stato finale.

1.1 Interpretazione della funzione di transizione

La funzione di transizione δ determina l'evoluzione della macchina in base allo stato interno attuale $q \in Q$ e al simbolo $X \in \Gamma$ letto dalla testina (quello presente nella cella del nastro attualmente visitata dalla testina):

- Se $\delta(q, X)$ è indefinita, allora la macchina è **bloccata**, cioè la computazione della macchina si arresta.
- Se invece $\delta(q, X)$ è definita, e in particolare $\delta(q, X) = (p, Y, M)$, allora la macchina esegue la seguente mossa:
  - passa dallo stato interno q a p (si osservi che può essere $p = q$, dunque lo stato può non cambiare);
  - scrive il simbolo Y nella cella attualmente visitata, al posto di X (ma può essere $Y = X$);
  - sposta la testina a sinistra se $M = \mathtt{L}$, o a destra se $M = \mathtt{R}$ (in questa formalizzazione, la testina non può restare ferma).

Osservazione: Esistono tante diverse varianti delle macchine di Turing, che non comportano un cambiamento nella classe di linguaggi riconosciuti. Ad esempio, ci sono formalizzazioni in cui la testina può rimanere ferma, o può spostarsi di più di una posizione alla volta, o ancora ci possono essere più nastri di lavoro, ecc.

1.2 Esempio di descrizione

La macchina di Turing

$$M = \langle \{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{0, 1, X, Y, B\}, \delta, q_0, B, \{q_4\} \rangle$$

ha:

- cinque stati, indicati con $q_0, \ldots, q_4$;
- l'alfabeto di input $\{0, 1\}$;
- l'alfabeto di lavoro $\{0, 1, X, Y, B\}$, che estende l'alfabeto di input, e in cui B è il simbolo di blank;
- lo stato iniziale q_0;
- un unico stato finale, q_4.

Siccome ha un dominio finito, la funzione di transizione δ può essere descritta mediante una tabella, nella quale le righe corrispondono agli stati in Q, le colonne corrispondono ai simboli dell'alfabeto di lavoro Γ, e le celle contengono i valori della funzione di transizione (qui si usa $-$ per indicare i casi in cui essa è indefinita):

	0	1	X	Y	B
q_0	$(q_1, X, \mathtt{R})$	$-$	$-$	$(q_3, Y, \mathtt{R})$	$-$
q_1	$(q_1, 0, \mathtt{R})$	$(q_2, Y, \mathtt{L})$	$-$	$(q_1, Y, \mathtt{R})$	$-$
q_2	$(q_2, 0, \mathtt{L})$	$-$	$(q_0, X, \mathtt{R})$	$(q_2, Y, \mathtt{L})$	$-$
q_3	$-$	$-$	$-$	$(q_3, Y, \mathtt{R})$	$(q_4, B, \mathtt{R})$
q_4	$-$	$-$	$-$	$-$	$-$

Questa tabella costituisce di fatto il *programma* della macchina.

2 Descrizioni istantanee

Per poter definire le computazioni delle MdT, bisogna capire quali siano le informazioni rilevanti al fine di determinare l'evoluzione futura di una macchina. Intuitivamente, queste sono lo stato interno, la posizione della testina e il contenuto del nastro, ma c'è un problema: il nastro è infinito, quindi non si riesce a elencare i simboli contenuti in tutte le sue celle. La soluzione è considerare rilevante solo la frazione del nastro che contiene simboli significativi, cioè che non è solo una sequenza infinita di simboli di blank: siccome

- all'inizio della computazione, sul nastro ci saranno blank in tutte le celle tranne quelle in cui viene inserito l'input, che è di dimensione finita,

- dopo un numero finito di mosse, la macchina potrà aver visitato solo un numero finito di celle,

questa frazione del nastro rimarrà sempre finita. Tutte queste informazioni vengono rappresentate formalmente mediante le descrizioni istantanee.

Una **descrizione istantanea** (**ID**) di una MdT è una tripla $(\alpha, q, \beta) \in \Gamma^* \times Q \times \Gamma^*$, che si rappresenta con la notazione

$$\alpha q \beta = X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n$$

(dove $\alpha = X_1 X_2 \dots X_{i-1}$ e $\beta = X_i X_{i+1} \dots X_n$, quindi $X_k \in \Gamma$ per ogni k). Essa rappresenta la configurazione della MdT in cui:

- lo stato interno è q;
- $X_1 \dots X_n$ è il **contenuto significativo del nastro**;
- il simbolo X_i scritto immediatamente alla destra dello stato q è quello situato attualmente sotto la testina.

Il contenuto significativo del nastro è la stringa $X_1 \dots X_n$ formata dai simboli contenuti nella parte di nastro "non definitivamente vuota", cioè:

- X_1 è il simbolo più a sinistra diverso da B e preceduto esclusivamente da simboli B;
- X_n è il simbolo più a destra diverso da B e seguito esclusivamente da simboli B.

Questa definizione serve a eliminare dalla descrizione solo gli infiniti blank ai due lati del nastro, e non eventuali blank presenti tra altri simboli, che devono invece essere considerati significativi. Ad esempio:

$$\dots BBBB \underbrace{01B01B10}_{\text{contenuto significativo}} BBBB \dots$$

3 Passi di computazione

Un **passo di computazione** è una relazione tra due descrizioni istantanee I e J: si indica con la notazione $I \Rightarrow_M J$ il fatto che J è ottenuta da I in un passo di computazione della macchina M (per semplificare la notazione, si può scrivere $I \Rightarrow J$, omettendo il pedice M, se la macchina a cui ci si riferisce è chiara dal contesto).

Data una configurazione $I = X_1 X_2 \dots X_{i-1} q X_i X_{i+1} \dots X_n$, una passo di computazione della macchina consiste nel modificare I secondo il valore della funzione di transizione sulla coppia (q, X_i), formata dallo stato attuale e dal simbolo sotto la testina. Se $\delta(q, X_i)$ è indefinita, la macchina è bloccata: non è possibile eseguire un passo di computazione. Se invece $\delta(q, X_i) = (p, Y, M)$, allora l'ID J è ottenuta da I modificando lo stato in

p, scrivendo Y al posto di X_i e muovendo la testina nella direzione indicata da M. Formalmente, J è definita per casi, che dipendono dalla direzione M, dalla posizione attuale i della testina e dal valore del simbolo Y da scrivere sul nastro al posto di X_i.

Nei casi in cui la mossa definita dalla funzione di transizione prevede uno spostamento della testina a sinistra, cioè quando $\delta(q, X_i) = (p, Y, \mathtt{L})$:

- Se $i = 1$, ovvero se la testina è posizionata sul primo simbolo significativo del nastro,
$$I = qX_1X_2 \dots X_n$$
il passo di computazione è:
$$qX_1X_2 \dots X_n \Rightarrow pBYX_2 \dots X_n$$
In particolare, sotto la testina rimane un simbolo B, che in I era già presente a sinistra della testina, ma non era indicato in quanto non significativo.

- Se $i = n$ e $Y = B$, cioè se la testina è sull'ultimo simbolo significativo
$$I = X_1 \dots X_{n-2}X_{n-1}pX_n$$
e la mossa prevede di sostituire X_n con il simbolo blank, il passo di computazione è:
$$X_1 \dots X_{n-2}X_{n-1}pX_n \Rightarrow X_1 \dots X_{n-2}pX_{n-1}$$
Infatti, il simbolo blank appena scritto "rimane fuori" dalla parte significativa del nastro (avendo solo altri blank alla sua destra), e rimane X_{n-1} come ultimo simbolo significativo.

- Negli altri casi ($1 < i < n$, oppure $i = n$ e $Y \neq B$), il passo di computazione è:
$$X_1 \dots X_{i-2}X_{i-1}qX_iX_{i+1} \dots X_n \Rightarrow X_1 \dots X_{i-2}pX_{i-1}YX_{i+1} \dots X_n$$
si sostituiscono q con p e X_i con Y, poi si sposta la testina su X_{i-1} (scrivendo p a sinistra di tale simbolo).

Quando invece $\delta(q, X_i) = (p, Y, \mathtt{R})$, cioè la mossa prevede uno spostamento a destra, le definizioni del passo di computazione sono sostanzialmente simmetriche:

- Se $i = n$, ovvero se la testina è sull'ultimo simbolo significativo,
$$I = X_1 \dots X_{n-1}qX_n$$
il passo di computazione è:
$$X_1 \dots X_{n-1}qX_n \Rightarrow X_1 \dots X_{n-1}Yp$$
Si osservi che la testina rimane sopra a un simbolo B, che non viene indicato perché non è compreso nella parte significativa del nastro.

- Se $i = 1$ e $Y = B$, cioè se la testina è sul primo simbolo significativo

  $$I = qX_1X_2 \dots X_n$$

  e la mossa prevede di sostituire X_1 con il simbolo blank, il passo di computazione è:

  $$qX_1X_2 \dots X_n \Rightarrow qX_2 \dots X_n$$

  Il simbolo B che sostituisce X_1 si unisce alla sequenza infinita di blank non significativi a sinistra, quindi non compare nell'ID risultante, lasciando X_2 come primo simbolo significativo.

- Negli altri casi ($1 < i < n$, oppure $i = 1$ e $Y \neq B$), il passo di computazione è:

  $$X_1 \dots X_{i-1}qX_iX_{i+1} \dots X_n \Rightarrow X_1 \dots X_{i-1}YpX_{i+1} \dots X_n$$

  si sostituiscono q con p e X_i con Y, poi si sposta la testina su X_{i+1} (scrivendo p a sinistra di tale simbolo).

Osservazione: Siccome la testina non può stare ferma, in un passo di computazione $I \Rightarrow J$ si ha sempre $I \neq J$.

4 Computazioni

Una **computazione** (**in zero o più passi**) di una MdT è una sequenza di passi di computazione della macchina. Formalmente, date una MdT $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$ e due descrizioni istantanee I e J, si scrive $I \overset{*}{\Rightarrow}_M J$ (anche qui si può omettere il pedice M) se e solo se esiste una sequenza di ID $K_1, K_2, \dots, K_n$ (con $n \geq 1$) tale che

- $I = K_1$,
- $J = K_n$,
- per ogni $i = 1, 2, \dots, n-1$ si ha $K_i \Rightarrow_M K_{i+1}$.

Come caso particolare, se $n = 1$ si ha $I = K_1 = K_n = J$, ovvero $I \overset{*}{\Rightarrow} I$: la macchina non effettua alcun passo di computazione.

5 Linguaggio accettato

Per definire il linguaggi accettato da una MdT, è prima necessario chiarire come l'input sia fornito alla macchina. Nei modelli visti in precedenza (gli automi a stati finiti e a pila), l'input era in un certo senso gestito "esternamente", presentando all'automa solo un simbolo alla volta. Invece, nel caso delle MdT, i simboli di input vengono scritti sul

nastro prima dell'inizio della computazione, dopodiché la macchina è libera di leggere (e modificare) "quelli che vuole".

Formalmente, data una MdT $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$, l'**ID iniziale** (**configurazione iniziale**) di M su input $w = X_1 \dots X_n \in \Sigma^*$ è:

$$q_0 w = q_0 X_1 \dots X_n$$

- lo stato è quello iniziale della macchina;
- il contenuto significativo del nastro è la stringa di input;
- la testina è posizionata sul primo simbolo dell'input.

Il **linguaggio accettato** (**riconosciuto**) dalla MdT M è l'insieme di stringhe w sull'alfabeto di input di M per cui esiste una computazione che, partendo dall'ID iniziale di M su input w, arriva in un'ID in cui lo stato è finale:

$$L(M) = \{w \in \Sigma^* \mid q_0 w \Rightarrow \alpha p \beta \text{ con } p \in F\}$$

Su questa nozione di accettazione vanno fatte alcune importanti osservazioni:

- Avendo assunto che la funzione di transizione sia indefinita sugli stati finali, se la MdT arriva in uno stato finale allora si arresta. Di conseguenza, la computazione si arresta sicuramente per tutte le stringhe accettate.
- Al contrario, una computazione non accettante potrebbe *non arrestarsi* (non terminare, continuare all'infinito).
- La condizione di accettazione non richiede che la macchina abbia letto tutto l'input.

Queste ultime due osservazioni, in particolare, differenziano fortemente le macchine di Turing rispetto ai modelli visti in precedenza, che consumavano tutto l'input e si arrestavano sempre.

6 Linguaggi ricorsivamente enumerabili e linguaggi decidibili

Le macchine di Turing portano alla definizione di *due* classi di linguaggi:

- Un linguaggio L è **ricorsivamente enumerabile** (**r.e.**, *Recursively Enumerable*) se e solo se esiste una MdT M che lo riconosce, ovvero tale che $L(M) = L$.
- Un linguaggio L è **decidibile** se e solo se esiste una MdT M che lo riconosce, $L(M) = L$, e che *si arresta per ogni possibile input.*

La differenza tra i linguaggi ricorsivamente enumerabili e quelli decidibili è che una MdT che riconosce un linguaggio r.e. potrebbe non arrestarsi per (alcune o tutte) le stringhe non appartenenti al linguaggio, mentre una che riconosce un linguaggio decidibile si deve arrestare anche per ogni stringa non appartenente al linguaggio.

7 Tesi di Church-Turing

A partire dall'inizio del '900, sorse in ambito matematico la necessità di capire i limiti della nozione di calcolo, per trovare la risposta a domande come "esiste un algoritmo in grado di stabilire la verità o la falsità di qualsiasi formula della logica del primo ordine?". Divenne allora necessario studiare formalmente i concetti di calcolabilità e algoritmo, e in particolare individuare un modello in grado di calcolare tutto ciò che è calcolabile: solo così sarebbe stato possibile dimostrare che certe cose non sono invece calcolabili, indipendentemente dal modello scelto.

Negli anni '30 vennero proposti tre principali modelli di calcolo, completamente diversi tra di loro:

- le *funzioni generali ricorsive* (Kurt Gödel e Stephen Kleene, 1934/1936);
- il *lambda calcolo* (Alonzo Church, 1936);
- le *macchine di Turing* (Alan Turing, 1936).

Incredibilmente, venne poi dimostrato che queste tre formalizzazioni sono equivalenti: una funzione è calcolabile secondo uno di questi modelli se e solo se lo è anche secondo gli altri due. È allora ragionevole supporre che tali modelli catturino una nozione "naturale" di ciò che è calcolabile. Quest'assunzione è stata formalizzata nella **tesi**[2] **di Church-Turing**, che afferma che una funzione sui numeri naturali[3] è *intuitivamente calcolabile* se e solo se è calcolabile da una macchina di Turing (e quindi anche in un qualunque altro formalismo equivalente).

Da allora, sono stati definiti molti altri modelli di calcolo, tra cui

- i sistemi canonici (Emil Post, 1943),
- la logica combinatoria (Haskell Curry, 1960 circa, ma già formulata negli anni '30),
- gli algoritmi di Markov (Andrey Markov, 1960),
- le macchine RAM,
- i linguaggi WHILE,
- i linguaggi di programmazione general purpose (Java, C, LISP, ecc.),

[2] Si parla di tesi, e non di teorema, perché questa è un'affermazione "metafisica", impossibile da dimostrare, dato che non esiste una definizione formale di cosa sia intuitivamente calcolabile.

[3] Le funzioni sui numeri naturali sono sufficienti a esprimere tutti i problemi trattati dall'informatica.

ma tutti questi sono risultati equivalenti ai tre elencati prima, continuando a suggerire la validità della tesi di Church-Turing.

Oggi si dice che una funzione è **Turing calcolabile** se è calcolata da una macchina di Turing, e che un linguaggio di programmazione è **Turing completo** se calcola tutte (e sole[4]) le funzioni Turing calcolabili.

[4]Ormai si tende a "dare per buona" la tesi di Church-Turing, quindi in genere non si dimostra neanche più che un linguaggio calcola *solo* le funzioni Turing calcolabili. Invece, dimostrare che le calcola tutte è necessario, perché un linguaggio/modello potrebbe tranquillamente essere meno potente delle macchine di Turing (si pensi ad esempio agli automi a stati finiti e a pila).