Appunti Computability

Diego Oniarti

Anno 2024-2025

Contents

1 Macchina di Turing

Una macchina di Turing è rappresentata da:

- $\Sigma = \{\sigma_0, \dots, \sigma_{m-1}\}$

- $Q = \{q_0, \dots, q_{n-1}\}$
- $q_o \in Q$ stato iniziale
- $f : \Sigma \times Q \to \Sigma \times Q \times \{\leftarrow, \rightarrow\}$

La funzione di transizione può essere definita come una tabella

Q		σ_0	$\cdots$	σ_n
	q_0			
	$\vdots$			
	q_n			

Un altra rappresentazione per una macchina di Turing è quella della macchina a stati (come quelle viste in LFC). Gli stati corrispondono agli stati della macchina di Turnig, mentre le transizioni contengono il carattere letto, quello da scrivere, e la transizione.

NB. Come definiamo la funzione di transizione non è importante. Per la definizione di una macchina di Turnig basta che esista una funzione di transizione del tipo $f : \Sigma \times Q \to \Sigma \times Q \times \{\leftarrow, \rightarrow\}$

Non è necessario che la funzione di transizione sia totale.

Esempio Questa è la funzione di transizione per una macchina di Turing che inizia con un numero binario sul nastro e ci aggiunge 1.

	$\sqcup$	0	1
LSB	$\sqcup, carry, \leftarrow$	$0, LSB, \rightarrow$	$1, LSB, \rightarrow$
$carry$	$1, MSB, \leftarrow$	$1, MSB, \leftarrow$	$0, carry, \leftarrow$
MSB	$\sqcup, halt, \rightarrow$	$0, MSB, \leftarrow$	$1, MSB, \leftarrow$

Esempio Ideiamo una macchina di Turing che inizia con un numero sul nastro. La macchina deve:

- creare una copia del numero letto
- scrivere questa copia a destra del numero dato
- lasciare il numero intoccato
- lasciare un $\sqcup$ tra i due numeri

Una soluzione valida sarebbe questa, dove l'alfabeto è $\Sigma = \{\sqcup, 0, 1, \hat{0}, \hat{1}\}$ e la

funzione di transizione è:

	$\sqcup$	0	1	$\hat{0}$	$\hat{1}$
$next$	$\sqcup, halt, \rightarrow$	$\hat{0}, first0, \rightarrow$	$\hat{1}, first1, \rightarrow$		
$first1$	$\sqcup, scom1, \rightarrow$	$0, first1, \rightarrow$	$1, scom1, \rightarrow$		
$scom1$	$1, left, \leftarrow$	$0, scom1, \rightarrow$	$1, scom1, \rightarrow$		
$first0$	$\sqcup, scom0, \rightarrow$	$0, first0, \rightarrow$	$1, scom0, \rightarrow$		
$scom0$	$0, left, \leftarrow$	$0, scom0, \rightarrow$	$1, scom0, \rightarrow$		
$left$	$\sqcup, left, \leftarrow$	$0, left, \leftarrow$	$1, left, \leftarrow$	$0, next, \rightarrow$	$1, next, \rightarrow$

2 Multi-Tape Turing Machine

Per convenzione immaginiamo una macchina con un tape di input, uno di output, e gli altri sono per utilizza arbitrario.
Ogni tape ha un puntatore suo, e può decidere di muoverlo o lasciarlo intoccato.
La funzione di transizione prende questa forma

$$f : \Sigma^t \times Q \rightarrow \Sigma^t \times Q \times \{\leftarrow, \downarrow, \rightarrow\}^t$$

dove l'apice t (numero di tape) indica che l'elemento è ripetuto t volte all'interno di una tupla.
La funzione quindi prende l'input di tutti i tape e lo stato corrente. Questo decide cosa scrivere in ogni tape, lo stato a cui muoversi, e la direzione in cui muovere ogni tape.

Multi-Tape vs Single Tape Una macchina di Turing con più tape può svolgere le stesse computazioni di una macchina a tape singolo. È solo più veloce a farlo.

3 Turing vs Universal Machines

Una differenza che rimane tra la macchina di Turing descritta e un computer come lo vediamo oggi è la seguente. Una macchina di Turing è hard coded per svolgere un singolo compito.

Possiamo quindi formalizzare una macchina di Turing U che sia universale? si.
L'input di U deve essere un encoding di una macchina m.

Prendiamo per esempio la macchina di Turing che aggiunge 1 ad un numero (vista in precedenza).

	$\sqcup$	0	1
0	$\sqcup, 1, \leftarrow$	$0, 0, \rightarrow$	$1, 0, \rightarrow$
1	$1, H$	$1, H$	$0, 1, \leftarrow$

Possiamo definire un'albeto che ci permetta di descrivere questa tabella sotto forma di stringa.

$$\Sigma = \{\text{␣}, 1, 0, ', ', ;\}$$
$$\text{tabella} = \text{␣}, 1, 0; 0, 0, 1; 1, 0, 1; 1, , ; 1, , ; 0, 1, 0$$

Prendiamo cone conenzione che ogni cella sia definita da 3 simboli separati da virgole. Un simbolo mancante è interpretato come l'*Halting State*.

La macchina universale U è composta da:

- Un tape $\lfloor m \rfloor$ che rappresenta la macchina m
- Un tape s che rappresenta lo stato di m
- Uno o più tape usati per l'esecuzione di m

nb. Come detto in Sec.2, questo piò essere svolto anche con una macchina a tape singolo.

L'esecuzione di m utilizzanto U richiede più *step* dell'esecuzione di m. Ma il numero di step di U scala linearmente con quello di m.

$$t(m, s) \leq 2|s| + 1$$
$$t(U, \lfloor m \rfloor s) \leq kt(m, s)$$

dove $t(a, b)$ è il tempo di esecuzione della macchina a sull'input b.

3.1 Random-access Machine

Un computer moderno può essere devinito come una "*random access machine*" in quanto accede agli indirizzi di memoria in tempo costante, a differenza della macchina di Turing che deve scorrere il tape.
Questa è l'unica differenza tra i due tipi di macchine. Quella di Turing è "lenta".

Ogni altro aspetto di una CPU odierna può essere creato analogamente in una macchina di Turing (Pc, registri, memoria, etc..).

Nota sull'alfabeto

Abbiamo usato un alfabeto Σ di 5 caratteri per descrivere la Turing machine, ma potremmo rappresentare ogni simbolo con un numero binario a tre cifre. Questo ci permette di descrivere un programma come una stringa binaria.

4 Uncomputabe machines

Possiamo rapprsentare ogni possibile macchina di Turing e ogni può output in una tabella

$$TM \begin{cases} \overbrace{\begin{array}{c|ccccc} & \epsilon & 0 & 1 & 00 & \cdots \\ \hline \epsilon & & & & & \\ 0 & & & & & \\ 1 & & & & & \\ \vdots & & & & & \end{array}}^{\Sigma^*} \end{cases}$$

$$UC : \Sigma^* \mapsto \{0,1\}$$

$$UC(\alpha) = \begin{cases} 0 & m_\alpha(\alpha) = 1 \\ 1 & \text{altrimenti} \end{cases}$$

m_α macchina descritta dalla stringa α

Ora possiamo usare l'argomento della diagonale per creare una macchina UC che non sia computabile.

Thesis

$$\forall m \in TM \exists s \in \{0,1\}^* : \ m(s) \neq UC(s)$$

Sapevamo già che esistessero problemi non calcolabili. Questa è sono un'altra prova.

$$\begin{array}{rl} UC \in \{f : \Sigma^* \to \{0,1\}\} & \text{uncountable} \\ \{TM\} & \text{countable} \end{array}$$

Congettura di Goldbach. Ogni numero maggiore di due può essere espresso come la somma di due numeri primi.

4.1 Halting Problem

Esiste una macchina $H(\lfloor M \rfloor, \epsilon)$ che so comporti così?

$$H(\lfloor M \rfloor, \epsilon) = \begin{cases} 1 & M(\epsilon) \text{ halts} \\ 0 & M(\epsilon) \text{ does not halt} \end{cases}$$

No. Se esistesse esisterebbe anche la macchina H' con quiesto comportamento.

$$H'(\lfloor M \rfloor, \epsilon) = \begin{cases} 0 & H(\lfloor M \rfloor, \epsilon) == 1 \\ \infty & H(\lfloor M \rfloor, \epsilon) == 0 \end{cases}$$

Il comportamento di $H(H'(\lfloor M \rfloor, \epsilon))$ non può poi essere definito.

NB! Questa non è la dimostrazione usata dal prof. Per quella chiedi in giro.

4.1.1 rambling

Halt non è ricorsiva. Halt è ricorsivamente enumerabile? Si. Basta usare il metodo "parallelo" diagonale visto in precedenza. (avanzare tutti i casi di uno step alla volta) faccio un backup

5 Recap

5.1 Insiemi

Abbiamo due modi di definire un subset ti tutte le stringhe.

$$s \subseteq \Sigma^*$$
$$f : \Sigma^* \to \{0, 1\}$$

5.2 recursive

Un set è *recursive* se e solo se

$$s \in R \iff \exists m\ TM\ s.t. \forall x \in \Sigma^*\ m(x) = \begin{cases} 0 & x \notin s \\ 1 & x \in s \end{cases}$$

5.3 ricursively enumerable

Ci sono tre modi di definire RE.

- Un set è ricorsivamente enumerabile se:

  $$s \in RE \iff \exists m\ TM\ s.t. \forall x \in \Sigma^*\ m(x) = \begin{cases} 1 & x \in s \\ \text{anything else} & x \notin s \end{cases}$$

  Anything Else include anche il non haltare mai.

- $$\forall x \in \Sigma^* m(x) = \begin{cases} 1 & x \in s \\ \infty & x \notin s \end{cases}$$

- m scrive su un tape tutti e soli gli elementi di s.

Possiamo dimostrare che le 3 definizioni sono equivalenti.

- $2 \implies 1$: Triviale. $\infty \in$ Anything Else
- $1 \implies 2$: Assumendo di avere una macchina m_1, possiamo costruire una macchina m_2.

$$m_2(x) = \begin{cases} 1 & m_1(x) = 1 \\ \infty & otherwise \end{cases}$$

  Questa tecnica di prendere una macchina e modificarla per crearne un'altra è chiamata **riduzione**.

- $2 \implies 3$: Assumiamo di avere m_2.

$queue \leftarrow empty$
$\forall x \in \Sigma^* :$
  $queue.push\ (x, \text{init configuration of } m_2(x))$
  $\forall (y, \text{configuration of } m_2(y)) \in q :$
    if configuration is halted :
      output y
      remove from queue$(y, config)$
    $else$:
      advance configuration by one step

Diagonale. Questo è a tutti gli effetti un ennesimo utilizza del metodo diagonale.

5.4 coRE

Un set è Co recursively enumerable (coRE) se il suo complementare è ricorsivamente enumerabile.

$$s \in RE \qquad m(x) = \begin{cases} 1 & x \in S \\ \text{anything else} & x \notin S \end{cases}$$

$$s \in coRE \qquad m(x) = \begin{cases} 0 & x \notin S \\ \text{anything else} & x \in S \end{cases}$$

$$s \in coRE \qquad \overline{m(x)} = \begin{cases} 0 & x \notin S \\ \infty & x \in S \end{cases}$$

Set. Dato il powerset di Σ^* (tutte le stringhe), RE e $coRE$ sono due sottoinsiemi di $P(\Sigma^*)$. R (linguaggi ricorsivi) **è** l'intersezione di RE e

coRE.

6 Ordering

Sia dato un linguaggio $L \subseteq \Sigma^*$ e un ordinamento $<$. Assumiamo che L sia ricorsivamente enumerabile ma non ricorsivo $L \in RE \setminus R$. Essendo L in RE, esiste una macchina m che produce tutti gli elementi di L. Possiamo provare che non esiste una macchina che li produce in ordine.

7 boh

$$HALT = \{(t, s) : m_t(s) \neq \infty\} \in RE \setminus R$$
$$HALT_\epsilon = \{t : m_t(\epsilon) \neq \infty\} \notin R$$

Ipotiziamo per assurdo che H_ϵ sia ricorsiva.

$$H_\epsilon : \Sigma^* \to \{0, 1\}$$
$$t \mapsto \begin{cases} 1 & m_t(s) \text{ halts} \\ 0 & \text{otherwise} \end{cases}$$

AO! mi sono distratto e non ho seguito. Però la prova funziona per riduzione e contraddizione. Crea una macchina H che scrive un input e chiama H_ϵ mi pare

8 Busy beaver "game"

$$|\Sigma| = n$$
$$|Q| = m$$
$$halt \notin Q$$

Il numero di macchine possibili con questi parametri è $2n^2m(m+1)$. Si può vedere questo costruendo la tabella che definisce le transizioni della macchina.

Chiamiamo $\Sigma(m)$ Il numero massimo di 1 che una macchina con m stati può mettere sul tape.
Poi chiamiamo $S(m)$ il numero massimo di step che una macchina con m esegue prima di haltare.
Per entrambi consideriamo solo macchine che ricevono ϵ come input e hantano.

$S(m)$ non è computabile.

9 2024-10-07

$$(L, <) \subseteq (\Sigma^*, >)$$
$$L \in RE \setminus R$$
$$m_L$$

10 Proprietà di una TM

Una *Proprietà* di una Turing machine è una qualsiasi funzione binaria (decision function) sulla macchina.

$$HALT_\epsilon : TM \mapsto \{0, 1\}$$

Un esempio è Halt. Altri sono:

1. m has 10 states (Computable)
2. m decides prime numbers (Specification)
3. m recognizes haltime TMs (Semantica)
4. m deecides halting TMs (Computable perché è sempre *False*. Triviale)

Riconoscere vs Decidere. Una macchina di Turing *Decide* qualcosa se conferma qualcosa ("risponde si"). Ma non ha un comportamento stabilito in caso contrario
Una macchina *Riconosce* qualcoss se risponde "si" o "no" in maniera definitiva

Quindi una proprietà "P" **decide** un set di Turing Machines.

$$P : TM \mapsto \{0, 1\}$$

Triviale Una proprietà P è *triviale* se $P = \emptyset$ o $P = TM$

Specification Boh ˚¬˚

Semantic Ogni macchina di Turing può essere vista come

$$m(s) = \begin{cases} 1 \\ \text{anything else} \\ \infty \end{cases}$$

Quindi possiamo dire che ogni macchina di Turing riconosce un linguaggio $L(m) = \{s \in \Sigma^* : m(s) = 1\}$

Una proprietà è *Semantica* se.

$$\forall m_1, m_2 \in TM. L(m_1) = L(m_2) \implies P(m_1) = P(m_2)$$

Se le due macchine compiono lo stesso lavoro (riconoscono lo stesso linguaggio): O entrambe hanno la proprietà, o nessuna delle due la ha.

$$P(m) = \text{"All strings recognized by } m \text{ have an even length"}$$

La macchina che riconosce solo la stringa vuota (ϵ) ha questa proprietà. Questo perché tutte le stringhe che vengono riconosciute da questa macchina (solo 1) hanno lunghezza 0, che è pari.

10.1 Teorema di Rice

Se una proprietà è sia *semantica* che *non triviale* allora è **undecidable**

Prova per assurdo Sia P semantica e non triviale. Deve esserci almeno una macchina m_p per cui la proprietà sia vera (altrimenti sarebbe triviale)

$$m_p \in TM \ s.t. \ P(m_p) = 1$$

Without loss of generality: $L(m) = \emptyset \implies P(m) = 0$ Le machcine che riconoscono l'insieme vuoto non hanno la proprietà P.

Supponiamo per assurdo che P sia decidibile. Quindi

$$\exists \mathcal{P} \in TM \ s.t. \ \forall m : \mathcal{P}(m) = P(m)$$

Esiste una macchina $\mathcal{P}$ che decide la proprietà P.

Abbiamo poi la macchina $HALT : \ TM \times \Sigma^* \mapsto \{0, 1\}$ che decide se una certa macchina halta con un certo input.

Prendiamo poi una macchina qualsiasi $n \in TM$. Ovviamente possiamo ottenere $\mathcal{P}(\lfloor n \rfloor)$. La macchina prende un input t e:

Algorithm 1: n

save t on a separate tape;
Put s on the input tape;
run $m(s)$;
restore original input t;
run $m_p(t)$

$$P(m_{ms}) = \begin{cases} 0 & m(s) = \infty \\ m_p(t) & m(s) \neq \infty \end{cases}$$

$$m(s) = \infty \implies L(n_{ms}) = \emptyset$$
$$m(s) \neq \infty \implies L(n_{ms}) = L(m_p) \implies P(n_{ms} = P(m_p) = 1$$

Quindi questa macchina risolverebbe l'halting problem. Questo è ovviamente assurdo e prova la tesi.