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
- $f : \Sigma \times Q \to \Sigma \times Q \times \{\leftarrow, \to\}$

La funzione di transizione può essere definita come una tabella

$$Q \left\{ \begin{array}{|c|c|c|c|} \hline & \sigma_0 & \cdots & \sigma_n \\ \hline q_0 & & & \\ \hline \vdots & & & \\ \hline q_n & & & \\ \hline \end{array} \right.$$

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 $f : \Sigma \times Q \to \Sigma \times Q \times \{\leftarrow, \to\}$
>
> 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.

	␣	0	1
LSB	$␣, carry, \leftarrow$	$0, LSB, \to$	$1, LSB, \to$
$carry$	$1, MSB, \leftarrow$	$1, MSB, \leftarrow$	$0, carry, \leftarrow$
MSB	$␣, halt, \to$	$0, MSB, \leftarrow$	$1, MSB, \leftarrow$

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 ␣ tra i due numeri

Una soluzione valida sarebbe questa, dove l'alfabeto è $\Sigma = \{\llcorner\!\lrcorner, 0, 1, \hat{0}, \hat{1}\}$ e la funzione di transizione è:

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

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).

	$\llcorner\!\lrcorner$	0	1
0	$\llcorner\!\lrcorner, 1, \leftarrow$	$0, 0, \rightarrow$	$1, 0, \rightarrow$
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

$$TM\left\{\begin{array}{c} \overbrace{\hphantom{\epsilon\quad 0\quad 1\quad 00\quad \cdots\quad}}^{\Sigma^*} \\ \begin{array}{c|ccccc} & \epsilon & 0 & 1 & 00 & \cdots \\ \hline \epsilon & & & & & \\ 0 & & & & & \\ 1 & & & & & \\ \vdots & & & & & \end{array} \end{array}\right.$$

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

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

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.
>
> $$\begin{aligned} UC \in \{f : \Sigma^* \to \{0,1\}\} &\quad \text{uncountable} \\ \{TM\} &\quad \text{countable} \end{aligned}$$

> **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.

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

> **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

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

Ipotiziamo per assurdo che H_ϵ sia ricorsiva.

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

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"

$$
\begin{aligned}
|\Sigma| &= n \\
|Q| &= m \\
halt &\notin Q
\end{aligned}
$$

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 *Riconosce* qualcosa se conferma qualcosa ("risponde si"). Ma non ha un comportamento stabilito in caso contrario
> Una macchina *Decide* qualcosa 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.

11 Random String

Qual'è la definizione di una stringa random? Potremmo dire che una stringa è casuale se ogni simbolo ha la stessa probabilità di apparire in ogni posizione. Ma data una determinata stringa, è possibile determinare se sia stata generata tramite un processo casuale o con intento?

Per esempio $S_0 = 000000$ non sembra casuale, ma potrebbe esserlo. $S_1 =$ 01101001010010 invece sembra più "casuale" ma potrebbe essere generato secondo una regola precisa.

S_0 è facilmente comprimibile. Possiamo dire "è composta da 9 0" (*Run-length encoding*). Arrivano quindi in considerazione i concetti di entropia e quantità di informazione.

Diciamo quindi che una stringa è *casuale* quando non può essere compressa (o non può essere compressa molto). Questo è ovviamente soggetto a eccezioni, come il caso che si tiri 100 volte testa con una moneta.
Questa definizione però si basa pesantemente sulla definizione di un "algoritmo di compressione".

> Useremo intercambiabilmente il termine *incomprimibile* e *casuale*.
> Assumeremo anche di trattare solo algoritmo lossless.

11.1 Compression Algorthim

Un algoritmo di compressione prende una stringa s e genera una descrizione di s più corta di s stesso. Questo implica ovviamente l'esistenza di un processo di *decompressione* simmetrico a quello di comressione.

$$\begin{aligned}
&S \in \Sigma^* \\
&|s| = l \\
&\exists t \in \Sigma^* \ s.t \ |t| < l \\
&\exists \underbrace{m}_{unzip} .m(\underbrace{t}_{zipped}) = \underbrace{s}_{original}
\end{aligned}$$

Ci stiamo affidando a una macchina di decompressione m, ma cosa sappiamo riguardo all'efficenza di m?

11.2 Kolmogorov

Consideriamo la dimensione non solo di t ma anche della macchina m che decomprime t. Questo equivale a avere un "self expanding executable file".

> Immagina di scaricare un file compresso in un formato sconosciuto e di dover anche scaricare un programma per la decompressione del suddetto file.
> La dimensione del programma di estrazione dovrebbe essre inclusa nella dimensione totale del download.

Per definire la dimensione di m fixiamo una macchina universale U.

$$\begin{aligned}
&S \in \Sigma^* \\
&|s| = l \\
&\underbrace{(m,t)}_{description\ of\ s} \qquad s.t\ U(m,t) = s
\end{aligned}$$

Ottimalità ovviamente a noi interessa la coppia (m,t) più piccola possibile (quella "ottimale") che riproduca s alla fine.
Un caso triviale sarebbe la macchina che copia un input e l'input originale. Questo ovviamente non è ottimale.

11.3 "Most strings are uncompressable"

$$\Sigma = \{0,1\}$$
$$S = \Sigma^{100}$$
$$|S| = |\Sigma^{100}| = 2^{100}$$

Quante stringhe $s \in \Sigma^{100}$ sono comprimibili almeno del 10%? $(C(s) = t \quad |t| \leq 90)$

Perché esista un algoritmo di compressione deve esistere alche un algoritmo di decompressione.

$$t \in \bigcup_{i=0}^{90} \Sigma^i = T$$
$$|T| = \left| \bigcup_{i=0}^{90} \Sigma^i \right| = \sum_{i=0}^{90} 2^i = 2^{91} - 1$$
$$\frac{|T|}{|S|} = \frac{1}{512}$$

Quindi una stringa su 512 di tutte quelle di lunghezza 100 o meno è comprimibile di almeno il 10%. Questo a prescindere dall'algoritmo di compressione.
Quindi "most strings are random".
Fortunatamente, le stringhe che ci interessano di solito non sono random, quindi sono comprimibili.

11.4 Kolmogorov complexity

$$D_{s,u} = \{(m,t) \in \Sigma^* \ s.t \ U(m,t) = s\}$$
$$K_U(s) = \min_{(m,t)\in D_{s,u}} (|m| + |t|)$$

Chiamiamo la *Complessità di Kolmogorov* di una stringa s (con rispetto a una macchina universale U) la lunghezza minima di m, t tali che $U(m,t) = s$.
Questa definizione di *complessità* ovviamente dipende dalla macchina universale U. Se volessimo prendere anche U in considerazione?
Supponiamo di avere due macchine universali U e V. Possiamo prendere una descrizione in U e usarla in V aggiungendo dell'overhead.

$$V(u,(m,t)) = U(m,t) = s$$

Una caratteristica particolare è che l'overhead è sempre lo stesso indipendentemente da m e t. È sempre la stessa u.

$$\forall U, V \in UTM \exists C_{U,V} = |u_v|^*$$
$$K_V(s) \leq K_U(s) + C_{U,V}$$
$$K_V = O(K_U)$$

Le due rappresentazioni sono asintoticamente equivalenti.

11.5 La Kolmogorov Complexity È Unconmputable

Prendiamo una macchina universale U. K_U **non** è computable.
Prendiamo la frase

"The smallest number that cannot be defined with less than thirteen words"

Questa frase è un paradosso, perché se trovassimo questo numero la frase gli si applicherebbe (ed essendo una frase di 12 parole lo renderebbe definibile con meno di 13 parole).
Ora dobbiamo trovare una maniera formale di dimostrare la con computabilità della complessità di Kolmogorov utilizzando questo paradosso.

- number *to* string

*rappresentation of u in v

- defined *to* Kologorov complexity
- words *to* symbold

$$K_U(s) \text{The minimum \# of symbols that define } s \text{ (wrt } U\text{)}$$

Prova per assurdo Supopniamo di avere una macchina $m_{K,U} \in TM$ tale che $\forall s \in \Sigma^* m_{K,U}(s) = K_U(s)$

Algorithm 2: m

```
forall s ∈ Σ* do
    if K_U(s) > |⌊m_{k,u}⌋| + 1000000* then
        output s and HALT;
    end
end
```

$$|\lfloor m \rfloor| \leq |\lfloor m_{ku} \rfloor| + 1000000$$

*Costante di overhead per l'esecuzione dell'algoritmo stesso