Elementi di calcolabilità e complessità

matteogiorgi.github.io

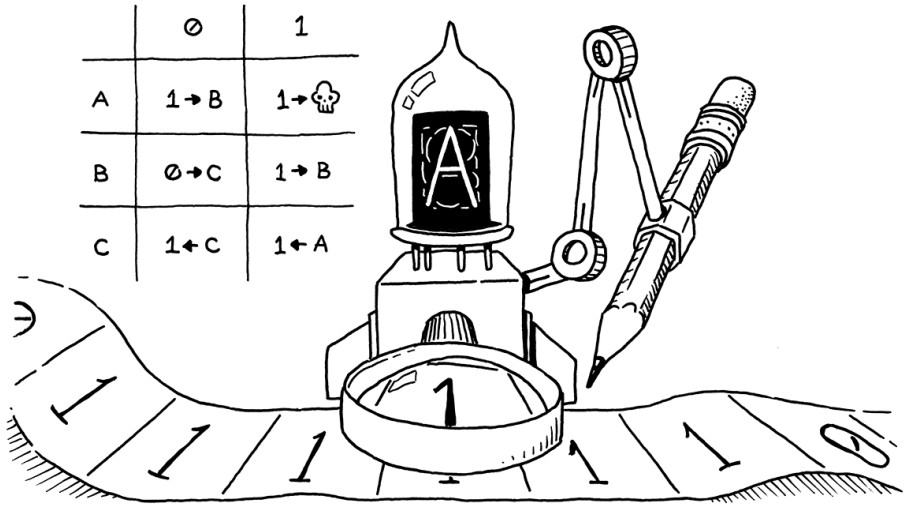
a.a. 2020

Beep-boop

Queste note sono state redatte dal sottoscritto durante le lezioni del corso di *Elementi di calcolabilità e complessità* tenuto dal prof. Pierpaolo Degano con l'ausilio del dott. Giulio Masetti per l'anno accademico 2019-2020.

Il materiale usato per la stesura, oltre alle note distribuite dal professore, comprende, in maniera più o meno estensiva, riferimanti ai seguenti testi:

- M. Sipser - *Introduction to the theory of computation*
- R.G. Taylor - *Models of computation and formal languages*
- N.J. Cutland - *Computability, introduction to recursive function theory*
- S.B. Cooper - *Computability theory*
- A. Bernasconi, B. Codenotti - *Introduzione alla complessità computazionale*
- G. Ausiello, F. D'Amore, G. Gambosi - *Linguaggi, modelli, complessità*



Algoritmi e macchine

Idea intuitiva di algoritmo

Il punto di partenza della teoria della calcolabilità è l'esigenza di formalizzare l'idea intuitiva di *funzione calcolabile* da un algoritmo, ovvero di *funzione algoritmica*[1].

Si può dire che un algoritmo sia un procedimento di calcolo che consente di pervenire alla soluzione di un problema, numerico o simbolico, mediante una sequenza finita di operazioni, completamente e univocamente determinate: una serie di istruzioni la cui esecuzione consente di trasformare l'insieme finito di dati simbolici che descrivono il problema, nella soluzione del problema stesso.

Caratteristiche distintive[2]

1. l'insieme delle istruzioni che definisce l'algoritmo è finito
2. l'insieme delle informazioni che rappresentano il problema
   - ha cardinalità finita
   - ha un effetto limitato su dati discreti
   - è descritta da dati finiti
3. il procedimento di calcolo (o *computazione*) è suddiviso in passi discreti e non fa uso di dispositivi analogici
4. ogni passo di computazione dipende solo dai precedenti e da una porzione finita dei dati in modo deterministico (è determinato senza ambiguità ovvero non soggetto ad alcuna distribuzione probabilistica non banale)
5. non c'è limite al numero di passi necessari all'esecuzione di un algoritmo, nè alla memoria richiesta per contenere i dati iniziali, intermedi e finali

In conclusione, una procedura algoritmica riceve in ingresso una descrizione finita dei dati del problema e restituisce, dopo un tempo finito, una descrizione finita del

[1] La nozione di funzione algoritmica sarà poi formalizzata dal concetto di *funzione ricorsiva.*

[2] Può essere utile aggiungere una ulteriore caratteristica a quelle elencate dal prof: *esiste un agente di calcolo in grado di eseguire le istruzioni (il calcolatore).*

risultato. La sua natura deterministica fa sì che l'algoritmo fornisca sempre lo stesso risultato ogni volta che riceve in ingresso gli stessi dati: così facendo stabilisce una relazione funzionale tra l'insieme dei dati e quello dei risultati.

Macchina di Turing

Esattamente come *funzioni ricorsive* e *λ-calcolo*, le Macchine di Turing sono un modello di calcolo. Nella sua versione più tradizionale una *MdT* si presenta come un dispositivo che accede ad un nastro potenzialmente illimitato diviso in celle, ciascuna contenente un simbolo appartenente ad un dato alfabeto Σ (alfabeto della macchina), comprendente i simboli # e $\triangleright$ che denotano rispettivamente l'assenza di informazione nella cella e la marca di inizio stringa.

- la MdT opera tramite un *cursore*, che può scorrere sul nastro in entrambe le direzioni e scrivere i caratteri σ_i appartenenti all'alfabeto Σ
- gli *stati* q_i della macchina appartengono all'insieme finito degli stati Q e identificano istante per istante le informazioni contenute nella computazione

  $\boxed{\triangleright\ a\ b\ \ldots\ a\ \underline{a}\ b\ \#\ \#}$
- il meccanismo che fa evolvere la computazione della macchina è detta *funzione di transizione*[3] δ e consente, partendo dallo stato q_x e dal carattere σ_x presente sulla cella puntata dalla testina, di portare la macchina in un altro stato q_y, scrivere un carattere σ_y su tale cella precedentemente occupata da σ_x ed eventualmente spostare la testina ($\Leftarrow$, $\Rightarrow$, $\equiv$)

Definizione. *Una Macchina di Turing è definita formalmente come la quadrupla*

$$M = (Q, \Sigma, \delta, q_0)$$

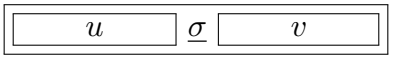
- Q *è l'insieme finito degli stati* q_i *che non contiene lo stato di accettazione* h
- Σ *è l'insieme finito dei simboli (alfabeto della macchina) al quale non appartengono i simboli di spostamento* $\Leftarrow$, $\Rightarrow$, $\equiv$
- $\delta \subseteq (Q \times \Sigma) \times (Q \cup h) \times \Sigma \times \{\Leftarrow, \Rightarrow, \equiv\}$ *è la relazione di transizione, tale che il carattere puntato dal cursore non possa mai trovarsi a sinistra di* $\triangleright$
- $q_0 \in Q$ *è lo stato iniziale*

[3] Nel trattare MdT deterministiche (quelle definite in questo capitolo) non è errato considerare δ come una funzione. La definizione formale fa riferimento a MdT nastro singolo, comprese quelle non-deterministiche dove δ restituisce un insieme di possibili configurazioni successive: ecco perchè sarà poi introdotta come relazione.

Configurazione, transizione e computazione

Definizione. *Una configurazione (istantanea) C di una MdT è definita dalla quadrupla*

$$(q, u, \sigma, v) \in (Q \cup \{h\}) \times \Sigma^* \times \Sigma \times \Sigma^F$$



- *q è lo stato corrente*
- *σ è il simbolo corrente*
- *u è la stringa di caratteri a sinistra del simbolo corrente*
- *v è la stringa di caratteri a destra del simbolo corrente*

Ne viene intuitivo dedurre che Σ^* rappresenti l'insieme di cardinalità infinita delle stringhe generabili dalla *giustapposizione* ($\circ$) dei caratteri dell'alfabeto della macchina Σ, mentre Σ^F è più facile a scriversi che a dirsi: $\Sigma^* \circ (\Sigma \setminus \{\#\}) \cup \{\epsilon\}$[4].

In generale una configurazione di una MdT non è altro che la coppia formata dallo stato corrente della macchina più la stringa[5] rappresentante la situazione sul nastro.

δ come funzione

Alla luce delle definizioni di MdT e configurazione, è importante analizzare meglio la relazione di transizione δ, definita precedentemente come la quadrupla

$$((q_x, \sigma_x), q_y, \sigma_y, D)$$

- (q_x, σ_x) è la coppia stato di partenza, carattere puntato dal cursore
- q_y è lo stato di arrivo
- σ_y è il carattere scritto nella cella precedentemente occupata da σ_x
- D è lo spostamento del cursore ($\Leftarrow$, $\Rightarrow$, $\equiv$)

Adesso è facilmente intuibile che, una macchina nella medesima configurazione di partenza che compie il medesimo spostamento del cursore, si troverà nella medesima configurazione di arrivo.

Questa constatazione permette di restringere la relazione δ in modo che sia una funzione rispetto al suo primo argomento (coppia stato corrente, carattere letto):

$$\delta(q_x, \sigma_x) = (q_y, \sigma_y, D)$$

[4]Quando definiremo meglio cos'è un linguaggio accettato da una MdT (o da un automa a stati finiti) sarà più chiaro come costruire un insieme di stringhe.

[5]Quando verranno trattate macchine a k nastri, si parlerà di una k-upla di stringhe, ciascuna delle quali riferita ad un nastro della macchina.

Una nuova definizione di MdT

Prima di procedere, è necessario utilizzare le considerazioni fatte fin'ora e ritoccare la nostra definizione di MdT estendendo la quadrupla ad una quintupla con l'introduzione di $\{q_A, q_R\}$; questo facilitarà il nostro percorso nell'utilizzo di macchine e linguagi regolari come strumento per trattare la classe dei *problemi decisionali*.

Definizione. *Una Macchina di Turing deterministica è definita come la quintupla*

$$M = (Q, \Sigma, \delta, q_0, \{q_A, q_R\})$$

- Q *è l'insieme finito degli stati* q_i *che non contiene gli stati terminali*
- $\Sigma \not\supseteq \{\Leftarrow, \Rightarrow, \equiv\}$ *è sempre l'insieme finito dei simboli (alfabeto della macchina)*
- $\delta{:}(Q\times\Sigma)\to(Q\times\Sigma\times\{\Leftarrow, \Rightarrow, \equiv\})$ *è la funzione di transizione*
- $q_0 \in Q$ *è lo stato iniziale*
- $\{q_A, q_R\} \not\subseteq Q$ *è l'insieme formato dagli stati terminali di accettazione e rifiuto*

Computazione e passo di computazione

Definizione. *Una computazione di una MdT è costituita da una sequenza di configurazioni, tale che* C_0 *rappresenti la configurazione iniziale e* C_{i+1} *il risultato dell'applicazione della funzione* δ *alla configurazione* C_i *(ovvero il passo di computazione* $C_i \to C_{i+1}$*). Si dirà inoltre che la computazione converge* $\downarrow$ *se, per* $n \geq 1$ *tale che*

$$C_0 \to C_1 \to ... \to C_n$$

C_n *è una configurazione finale, in caso contrario si dirà che diverge* $\uparrow$*.*

Per completezza, dato che un passo di computazione di una macchina M può essere visto come una relazione $\to_M$ tra C_i e C_{i+1}, una computazione di M è definibile come la *chiusura riflessiva e transitiva*[6] di $\to_M$ che, in accordo con la notazione usata precedentemente, può essere scritta come

$$C_0 \to_M^* C_n$$

[6] La chiusura riflessiva e transitiva $\bar{R}$ di una relazione R è la più piccola relazione definita per induzione con le proposizioni: $\boxed{\forall a \in A, \exists \bar{R}(a, a)}$, $\boxed{R(a, b) \Rightarrow \bar{R}(a, b)}$, $\boxed{\bar{R}(a, b), \bar{R}(b, c) \Rightarrow \bar{R}(a, c)}$. Alternativamente ed in modo del tutto equivalente, può essere definita come $\bar{R} = \cap_{R \subseteq S} S$, dove S è anch'essa una relazione riflessiva e transitiva (dimostrare l'equivalenza per esercizio).

Esempio di MdT

Consideriamo una macchina M che calcoli la somma di due numeri naturali rappresentati in notazione unaria con il simbolo | e separati dal simbolo +. La successione

$$(q_0, \underline{\rhd}| + ||\#) \to (q_0, \rhd\underline{|} + ||) \to (q_0, \rhd|\underline{+}||) \to (q_1, \rhd|||\underline{|}|) \to$$
$$\to (q_1, \rhd||||\underline{|}) \to (q_1, \rhd||||\underline{\#}) \to (q_2, \rhd||||\underline{|}) \to (h, \rhd|||\underline{\#})$$

rappresenterà la computazione $(q_0, \underline{\rhd}| + ||\#) \to^*_M (h, \rhd|||\underline{\#})$, mentre la funzione δ può essere tabellata come segue

q	σ	δ
q_0	$\rhd$	$q_0, \rhd, \Rrightarrow$
q_0	$\|$	$q_0, \|, \Rrightarrow$
q_0	$+$	$q_1, \|, \Rrightarrow$
q_1	$\|$	$q_1, \|, \Rrightarrow$
q_1	$\#$	$q_2, \#, \Lleftarrow$
q_2	$\|$	$h, \#, \equiv$

Automi e linguaggi

DFA e NFA

In questo capitolo verranno richiamate alcune nozioni utili per studiare *decidibilità*, *riducibilità* ed eventualmente l'insolubilità di problemi. Sarà utile avere ben chiaro cosa siano automi, linguaggi regolari e la loro relazione con le MdT.

Definizione. *Un automa finito deterministico (DFA) è la quintupla*

$$(Q, \Sigma, \delta, q_0, F)$$

- Q *è l'insieme finito degli stati*
- Σ *è l'insieme finito dei simboli (alfabeto)*
- $\delta{:}Q\times\Sigma\to Q$ *è la funzione di transizione*
- $q_0\in Q$ *è lo stato iniziale*
- $F\subseteq Q$ *è l'insieme degli stati di accettazione*

La funzione di transizione δ definisce le regole per il cambiamento di stato: specifica esattamente uno stato successivo per ogni possibile combinazione di stato e simbolo in input. Se ne deduce che *da ogni stato esce esattamente un arco di transizione per ogni possibile simbolo di input* (ecco perchè automa determinastico o DFA).

Definizione. *Un automa finito non deterministico (NFA) è la quintupla*

$$(Q, \Sigma, \Delta, q_0, F)$$

- Q *è l'insieme finito degli stati*
- Σ *è l'insieme finito dei simboli (alfabeto)*
- $\Delta{:}Q\times\Sigma_\epsilon\to\mathcal{P}(Q)$ *è la funzione di transizione*
- $q_0\in Q$ *è lo stato iniziale*
- $F\subseteq Q$ *è l'insieme degli stati di accettazione*

Le due definizioni di automa differiscono nel tipo[7] della funzione di transizione. Δ prende uno stato dall'insieme degli stati Q, un simbolo dall'alfabeto $\Sigma_\epsilon = \Sigma + \{\epsilon\}$ e produce l'insieme dei possibili stati successivi ($\subseteq \mathcal{P}(Q)$); δ invece, prende in input la coppia stato attuale, simbolo letto e produce un singolo stato successivo[8].

Linguaggio della macchina[9]

Automa o MdT, incomincia a essere chiaro che la nostra macchina sia qualcosa che mangia stringhe e decide se appartengono a un dato insieme.

- Esistono stringhe che una macchina M non digerisce?
- Che succede quando una macchina M prende in input una stringa w?
- Come distinguo gli insiemi di stringhe che la macchina elabora?

Per rispondere devo prima piazzare qualche paletto e senza perdere tempo in pesanti formalismi, ecco qua sotto alcune definizioni fondamentali.

Definizione. *Sia $M=\{Q, \Sigma, \delta, q_0, F\}$ un automa finito e $w=w_0w_1...w_n$ una stringa composta da caratteri $w_i \in \Sigma$. Si dice che la macchina M accetta la stringa w se esiste una sequenza di stati $[r_0, r_1, ..., r_{n+1}]$ in Q, con tre condizioni*

1. $r_0 = q_0$ — *La macchina è partita*
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, $0 \leq i \leq n-1$ — *La macchina cambia stato con δ*
3. $r_n \in F$ — *La macchina accetta l'input*

Si noti l'assonanza con la definizione di computazione: i tre passi precedenti sono infatti da considerarsi i tre stadi della computazione per un automa finito. La macchina accetta l'input (quindi parte), salta di stato in stato con una legge predefinita (funzione di transizione) e dopo un numero indefinito (ma finito) di salti, casca in uno degli stati di terminazione.

Definizione. *Il linguaggio A della macchina M è l'insieme di tutte le stringhe accettate dalla macchina: $A=L(M)$.*

Definizione. *La macchina M riconosce il linguaggio A se $A=\{w \mid w \in A,\ M \text{ accetta } w\}$.*

Definizione. *Un linguaggio A è chiamato regolare sse esiste un automa finito M che lo riconosce.*

[7] L'*arrow type* di una funzione è generato dal *type constructor* che giustappone i tipi primitivi di argomenti e risultato ottenendo il tipo composto della funzione (`f :: a -> b`).

[8] Come nella più comune bibliografia $\mathcal{P}$ indica l'insieme delle parti e ϵ la stringa vuota, anche se in questo caso è più corretto riferirsi al carattere nullo.

[9] Abuso di notazione: qui (ed in futuro) con il termine *macchina* mi riferisco ad un automa ma queste proprietà sono valide anche per una MdT.

Dalle precedenti definizioni, posso già facilmente intuire qualche semplice implicazione che però non dimostro per mancanza di spazio (prova per esercizio) 😐.

$$\boxed{A \text{ è regolare}} \Leftrightarrow \boxed{\exists\ M \text{ che riconosce } A} \Leftrightarrow \boxed{M \text{ accetta tutte le stringhe di } A}$$

Definizione. *Due macchine M_1 e M_2 si dicono equivalenti se riconoscono lo stesso linguaggio $A{=}L(M_1){=}L(M_2)$.*

Le operazioni regolari

Trattare i linguaggi regolari non è lo scopo di queste note ma, visto che li ho introdotti, è giusto spendere due parole su come manipolarne le stringhe.

Definizione. *Dati i linguaggi A e B, sono definite operazioni regolari*

1. $A{\cup}B{=}\{w \,|\, w{\in}A \vee w{\in}B\}$ *Unione*
2. $A{\circ}B{=}\{wv \,|\, w{\in}A \wedge v{\in}B\}$ *Concatenazione*
3. $A^*{=}\{w_0w_1...w_k \,|\, w_i{\in}A \wedge 0{\leq}i{\leq}k\}$ *Star*

L'unione fa esattamente quello che ci si aspetti faccia l'unione tra insiemi: raggruppa tutte le stringhe di A e B; la concatenazione antepone una stringa di A ad una di B in tutti i modi possibili; star concatena un numero qualsiasi di stringhe di A.

In ultimo è possibile dimostrare che la classe dei linguaggi regolari è chiusa rispetto alle tre operazioni regolari (vedi Sipser cap1).

Equivalenza tra DFA e NFA

Erroneamente si potrebbe pensare che gli NFA abbiano maggior potere computazionale dei DFA, e che quindi riconoscano più linguaggi; non è così perchè automi finiti deterministici e non deterministici riconoscono la stessa classe di linguaggi. Vediamo di formalizzare l'idea.

Teorema. *Per ogni automa finito non deterministico, esiste un automa finito deterministico equivalente.*

Dimostrazione. L'idea è quella di procedere per costruzione e trasformare l'automa non deterministico in uno deterministico con 2^k stati, pari cioè alla cardinalità di $\mathcal{P}(Q_{\text{NFA}})$. Ricorda infatti che, se chiamiamo k la cardinalità di Q_{NFA}, il numero dei sottoinsiemi corrispondenti ad una delle possibilità che il nuovo automa deve ricordare è 2^k.

Sia $N{=}(Q, \Sigma, \delta, q_0, F)$ l'NFA di partenza che riconosce un linguaggio A, voglio costruire un DFA $M{=}(Q', \Sigma', \delta', q_0', F')$ equivalente. Inizialmente considero il caso più semplice in cui N non ha ϵ-archi.

1. Ogni stato di M è un insieme di stati di N
$$Q' = \mathcal{P}(Q)$$
2. Se R è uno stato di M, esso è anche un insieme di stati di N. Quando M legge un simbolo w nello stato R, ci mostra gli stati successivi a quelli in R e, poichè da ogni stato si può andare in un insieme di stati, ne prendiamo l'unione
$$\begin{aligned} R{\in}Q,\, w{\in}\Sigma \;\Rightarrow\; \delta'(R, w) &= \{q{\in}Q \mid \exists\, r{\in}R.\, q{\in}\delta(r, w)\} \\ &= \bigcup_{r \in R} \delta(r, w) \end{aligned}$$
3. Lo stato iniziale di M è quello corrispondente alla collezione che contiene solamente lo stato iniziale di N
$$q_0' = q_0$$
4. M accetta se N è in uno stato accettante in quel momento
$$F' = \{R{\in}Q' \mid R \text{ contiene uno stato accettante di } N\}$$

Adesso considero anche gli ϵ-archi. Per ogni stato R di M, definisco $E(R)$ come la collezione di stati che possono essere raggiunti dagli elementi di R proseguendo solo con ϵ-archi (includendo gli stessi elementi di R). Per $R{\subseteq}Q$, sarà
$$E(R) = \{q \mid q \text{ può essere raggiunto da } R \text{ attraverso 0 o più } \epsilon\text{-archi}\}$$

Poi modifico δ' in modo che possa raggiungere tutti quegli stati esplorabili con ϵ-archi dopo ogni passo. Sostituisco $\delta(r, w)$ con $E(\delta(r, w))$ e ottengo
$$\delta'(R, w) = \{q{\in}Q \mid \exists\, r{\in}R.\, q{\in}E(\delta(r, w))\}$$

In ultimo devo modificare lo stato iniziale di M per potermi muovere inizialmente su tutti i possibili stati che possono essere raggiunti dallo stato inizile di N attraverso gli ϵ-archi. Per fare questo cambio q_0' in $E(q_0)$. Ho completato la costruzione di M.

Da notare che M funziona correttamente perchè ad ogni passo della computazione, entra in uno stato che corrisponde al sottoinsieme di stati nei quali N potrebbe trovarsi. □

Corollario. *(Dimostra per esercizio 😐)*

$$\boxed{\exists \text{ un NFA che riconosce } L} \Leftrightarrow \boxed{\exists \text{ un DFA che riconosce } L} \Leftrightarrow \boxed{L \text{ è regolare}}$$