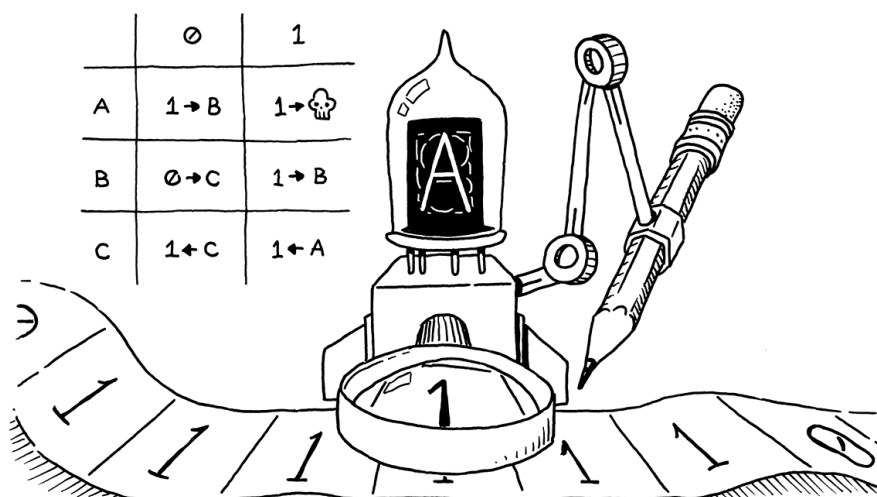


Note di calcolabilità

🌐 matteogiorgi.github.io



> UniPi 2020 ##
👉

Info

Queste note sono state redatte dal sottoscritto durante le lezioni del corso di *Elementi di calcolabilità e complessità* tenute dal prof. Pierpaolo Degano, con l'ausilio del dott. Giulio Masetti, per il Corso di Laurea in Informatica.

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

- Sipser - *Introduction to the theory of computation*
- Taylor - *Models of computation and formal languages*
- Cutland - *Computability, introduction to recursive function theory*
- Cooper - *Computability theory*
- Bernasconi, Codenotti - *Introduzione alla complessità computazionale*
- Ausiello, D'Amore, Gambosi - *Linguaggi, modelli, complessità*

Indice

1	Algoritmi e macchine	1
1.1	Idea intuitiva di algoritmo	1
1.1.1	Caratteristiche distintive	1
1.2	Macchina di Turing	2
1.3	Configurazione, transizione e computazione	3
1.3.1	δ come funzione	3
1.3.2	Una nuova definizione di MdT	4
1.3.3	Computazione e passo di computazione	4
1.3.4	Esempio di MdT	5
2	Automi e linguaggi	6
2.1	DFA e NFA	6
2.2	Linguaggio della macchina	7
2.2.1	Le operazioni regolari	8
2.3	Equivalenza tra DFA e NFA	8
2.3.1	Esempio di automi equivalenti	10
3	Decidibilità	11
3.1	Linguaggi T-riconoscibili e T-decidibili	11
3.1.1	Cosa posso decidere?	12

Algoritmi e macchine

1.1 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.

1.1.1 Caratteristiche distintive²

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

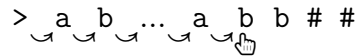
¹La nozione di funzione algoritmica sarà poi formalizzata dal concetto di *funzione ricorsiva*.

²Può 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.

1.2 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 $>$ che denotano rispettivamente l'assenza di informazione nella cella e la marca di inizio stringa.

> a b ... a b b # #


- 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
- 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 tale che lo stato di accettazione $h \notin Q$
- 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 non possa mai trovarsi a sinistra di $>$
- $q_0 \in Q$ è lo stato iniziale

³Nel 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.

1.3 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 *concatenazione*⁴ dei caratteri dell'alfabeto della macchina Σ , mentre Σ^F è più facile a scriversi che a dirsi:

$$\Sigma^F = \Sigma^* \circ (\Sigma \setminus \{\#\}) \cup \{\epsilon\}$$

In generale una configurazione di una MdT non è altro che la coppia formata dallo stato corrente della macchina più la stringa⁵ rappresentante la situazione sul nastro.

1.3.1 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):

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

⁴Antepone due stringhe di Σ in tutti i modi possibili ($\Sigma^* = \Sigma \circ \Sigma$). Vedi *operazioni regolari*.

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

1.3.2 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 faciliterà il nostro percorso nell'utilizzo di macchine e linguaggi 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
- $\delta: (Q \times \Sigma) \rightarrow (Q \times \Sigma \times \{\Leftarrow, \Rightarrow, \equiv\})$ è la funzione di transizione
- $q_0 \in Q$ è lo stato iniziale
- $\{q_A, q_R\} \not\subseteq Q$ è l'insieme degli stati terminali di accettazione e rifiuto

1.3.3 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 $C_i \rightarrow C_{i+1}$). Si dirà inoltre che la computazione converge \downarrow se, per $n \geq 1$ tale che

$$C_0 \rightarrow C_1 \rightarrow \dots \rightarrow C_n$$

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 \rightarrow_M tra C_i e C_{i+1} , una computazione di M è definibile come la *chiusura riflessiva e transitiva*⁶ di \rightarrow_M che, in accordo con la notazione usata precedentemente, può essere scritta come

$$C_0 \rightarrow_M^* C_n$$

⁶La chiusura riflessiva e transitiva \bar{R} di una relazione R è definita come $\bar{R} = \bigcap_{R \subseteq S} S$, dove S sia anch'essa relazione riflessiva e transitiva. Alternativamente la può pensare come la più piccola relazione definita per induzione attraverso le seguenti proposizioni

- $\forall a \in A. \exists \bar{R}(a, a)$
- $R(a, b) \Rightarrow \bar{R}(a, b)$
- $\bar{R}(a, b) \wedge \bar{R}(b, c) \Rightarrow \bar{R}(a, c)$

Prova a dimostrare l'equivalenza delle due definizioni \odot .

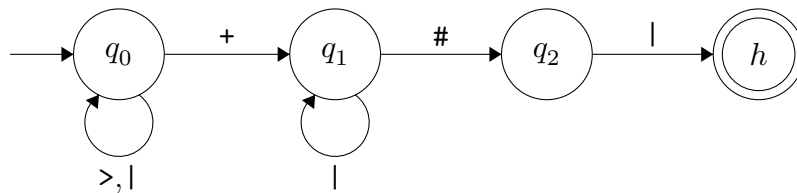
1.3.4 Esempio di MdT

Consideriamo una macchina M che calcoli la somma di due numeri naturali scritti in notazione unaria usando il simbolo $|$ e separati dal simbolo $+$. La seguente successione di configurazioni rappresenterà la computazione $(q_0, >|+||\#) \rightarrow_M^* (h, >||| \#)$

$$\begin{aligned}
 (q_0, >|+||\#) &\rightarrow (q_0, >|+||) \\
 &\rightarrow (q_0, >|+|) \\
 &\rightarrow (q_1, >|||) \\
 &\rightarrow (q_1, >|||) \\
 &\rightarrow (q_1, >||| \#) \\
 &\rightarrow (q_2, >|||) \\
 &\rightarrow (h, >||| \#)
 \end{aligned}$$

Dalla precedente posso facilmente tabulare la funzione δ , mentre la successione degli stati esplorati dalla macchina sono rappresentabili con un automa finito.

q	σ	δ
q_0	$>$	$q_0, >, \Rightarrow$
q_0	$ $	$q_0, , \Rightarrow$
q_0	$+$	$q_1, , \Rightarrow$
q_1	$ $	$q_1, , \Rightarrow$
q_1	$\#$	$q_2, \#, \Leftarrow$
q_2	$ $	$h, \#, \equiv$



Automi e linguaggi

2.1 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)
- $\delta: Q \times \Sigma \rightarrow Q$ 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 deterministico 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)
- $\Delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$ 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 tipo¹ 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 successivo².

2.2 Linguaggio della macchina

Automa o MdT, incomincia a essere chiaro che la nostra macchina³ 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 $w = w_0 w_1 \dots w_n$ 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 $[r_0, r_1, \dots, r_n]$ in Q , con tre condizioni

- | | |
|--|---------------------------|
| 1. $r_0 = q_0$ | $\exists r_0$ inizio |
| 2. $\delta(r_i, w_{i+1}) = r_{i+1}, 0 \leq i \leq n-1$ | $r_i \rightarrow r_{i+1}$ |
| 3. $r_n \in F$ | $\exists r_n$ accett. |

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. L'insieme (potenzialmente ∞) di tutte le stringhe accettate dalla macchina è chiamato linguaggio della macchina ($A = L(M)$).

Definizione. Si dirà che M riconosce il linguaggio A se ne accetta tutte le stringhe

$$A = \{w \mid w \in A \wedge M \text{ accetta } w\}$$

¹L'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 ($\mathbf{f} :: \mathbf{a} \rightarrow \mathbf{b}$).

²Come 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.

³Ho volutamente abusato del termine *macchina*. Qui mi riferisco ad un automa ma queste definizioni sono utilizzabili anche con una MdT. Tra poco sarò più preciso.

Definizione. Un linguaggio è detto regolare sse \exists un automa finito che lo riconosce.

Definizione. Due macchine si dicono equivalenti se riconoscono lo stesso linguaggio.

2.2.1 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. $A \cup B = \{w \mid w \in A \vee w \in B\}$ | Unione |
| 2. $A \circ B = \{wv \mid w \in A \wedge v \in B\}$ | Concatenazione |
| 3. $A^* = \{w_0 w_1 \dots w_k \mid w_i \in A \wedge 0 \leq i \leq k\}$ | 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 .

Teorema. La classe dei linguaggi regolari è chiusa rispetto alle operazioni regolari.

Dimostrazione. Troppo lunga, vedi Sipser ☺.

□

2.3 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 (incluso 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 iniziale 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. \square

Corollario. *Le seguenti proposizioni sono semanticamente equivalenti*

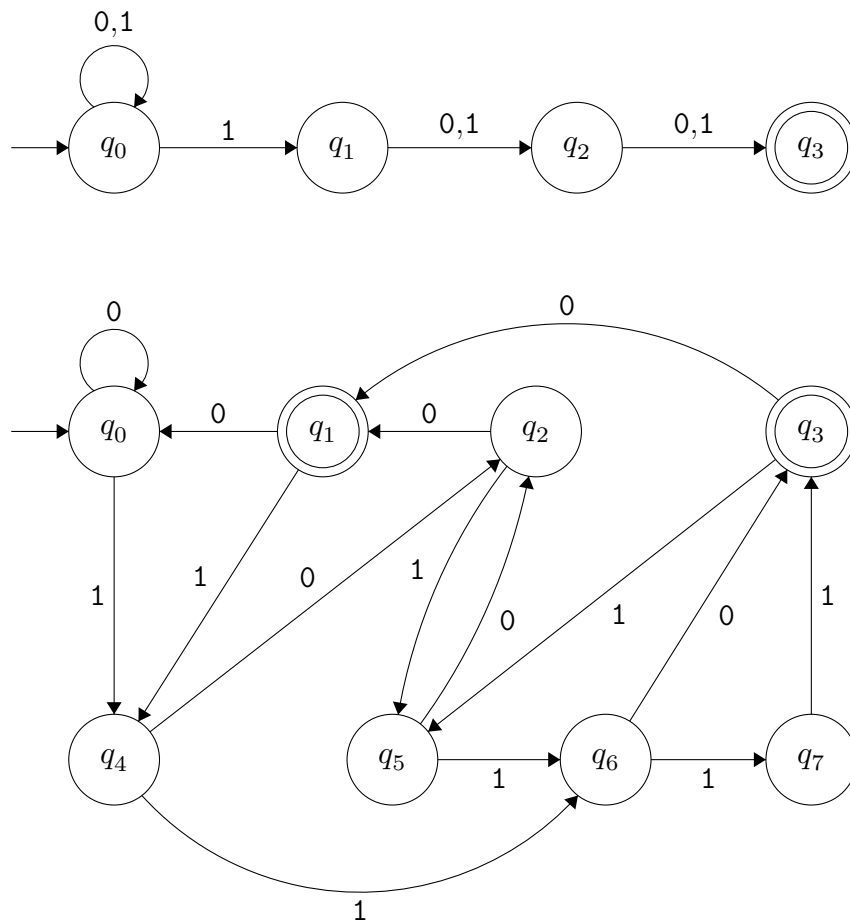
- \exists un NFA che riconosce il linguaggio L
- \exists un DFA che riconosce il linguaggio L
- L è regolare

Dimostrazione. Banale, segue dal teorema ☺. \square

2.3.1 Esempio di automi equivalenti

Sia A il linguaggio che consiste di tutte le stringhe su $\{0,1\}$ contenenti un 1 nella terza posizione dalla fine (dove per esempio $000100 \in A$, mentre $0011 \notin A$).

Ecco qua sotto due automi finiti equivalenti che riconoscono A : il primo non deterministico, il secondo deterministico. Controlla il DFA costruito usando il teorema di equivalenza precedente ☺.



Decidibilità

3.1 Linguaggi T-riconoscibili e T-decidibili

Dopo la parentesi fatta sugli automi, sono perfettamente in grado di puntualizzare quanto definito in precedenza riguardo le stringhe accettate da una macchina.

Definizione. Sia $M=(Q, \Sigma, \delta, q_0, \{q_A, q_R\})$ una MdT e $w=w_0w_1...w_n$ una stringa composta da caratteri $w_i \in \Sigma$. Si dice che M accetta la stringa w se esiste una sequenza di configurazioni $[C_0, C_1, ..., C_n]$ tale che

- | | |
|--|---------------------------|
| 1. C_0 è la configurazione iniziale su input w | $\exists C_0$ inizio |
| 2. Ogni C_i produce C_{i+1} | $C_i \rightarrow C_{i+1}$ |
| 3. C_n è una configurazione di accettazione | $\exists C_n$ accett. |

Esattamente come detto prima M riconosce un linguaggio A se ne accetta tutte le sue stringhe. Si scriverà $A=L(M)$.

Appare chiaro che sono tre i risultati possibili per una MdT: *accettare*, *rifiutare* oppure andare in *loop*¹. Quindi fai attenzione perchè una macchina M può non accettare un input sia raggiungendo lo stato q_R che andando in loop.

A volte distinguere una macchina che è entrata in un loop da una che sta semplicemente impiegando molto tempo risulta difficile, per questo sono preferibili MdT che si fermano su ogni singolo input. Una tale macchina è detta **decisore** e diremo che **decide** un certo linguaggio se lo riconosce.

Definizione. Un linguaggio si dice Turing-riconoscibile sse esiste una MdT che lo riconosce (ovvero, per ogni stringa del linguaggio la macchina termina nello stato di accettazione).

Definizione. Un linguaggio si dice Turing-decidibile sse esiste una MdT che lo decide (ovvero, per ogni stringa del linguaggio la macchina termina nello stato di accettazione, per tutte le altre termina nello stato di rifiuto).

¹Si intende una sequenza ∞ di configurazioni.

Teorema. *Ogni linguaggio Turing-decidibile è Turing-riconoscibile.*

Dimostrazione. Segue direttamente dalle due definizioni: se è decidibile significa che raggiunge lo stato di accettazione per tutte le stringhe del linguaggio, quindi è Turing-riconoscibile. \square

3.1.1 Cosa posso decidere?

Di seguito darò alcuni esempi di linguaggi Turing-decidibili; si renderanno utili le precedenti pagine sugli automi finiti e le definizioni di riconoscibilità e decidibilità.

I problemi computazionali proposti, a partire da quello dell'*accettazione* qua sotto, verranno rappresentati mediante linguaggi; si tratterà quindi di scrivere il problema in questione come un insieme di stringhe e chiedersi se esiste una MdT che decide tale linguaggio². Mostrare dunque che il linguaggio è decidibile equivale a mostrare che il problema computazionale è decidibile.

Lemma. *Sia A_{DFA} il linguaggio contenente le coppie formate dalle codifiche di tutti i DFA con delle particolari stringhe che essi accettano*

$$A_{DFA} = \{\langle B, w \rangle \mid B \text{ è un DFA che accetta la stringa di input } w\}$$

A_{DFA} è un linguaggio decidibile.

Dimostrazione. L'idea è quella di presentare una MdT M che decide A_{DFA} . M deve prendere come input la coppia $\langle B, w \rangle$, simulare B su w e accettare se la simulazione termina in uno stato di accettazione, viceversa rifiutare.

Quindi, dopo aver controllato la validità della coppia in input, M esegue la simulazione: tiene traccia dello stato corrente di B e della sua posizione corrente nell'input w scrivendo queste informazioni sul nastro, usa la funzione di transizione per aggiornare stati e posizioni fin quando non termina l'elaborazione dell'ultimo simbolo di w . M accetta l'input se B si trova in uno stato di accettazione, rifiuta altrimenti. \square

Lemma. *Sia A_{NFA} il linguaggio contenente le coppie formate dalle codifiche di tutti gli NFA con delle particolari stringhe che essi accettano*

$$A_{NFA} = \{\langle B, w \rangle \mid B \text{ è un NFA che accetta la stringa di input } w\}$$

A_{NFA} è un linguaggio decidibile.

²Nella formalizzazione del linguaggio salta subito all'occhio che automi e MdT sono rappresentati come stringhe. Per fare questo è necessaria una corrispondenza biunivoca tra macchina e codifica della macchina di cui non ho ancora fatto menzione (sarà argomento dei prossimi capitoli). Per adesso è sufficiente sapere che ogni macchina può essere codificata con una stringa e viceversa.

Dimostrazione. Presentiamo una MdT N che decide A_{NFA} . N converte l'NFA B in un DFA C equivalente, esegue M come una sottoprocedura che decide A_{DFA} su input $\langle C, w \rangle$ e accetta se M accetta, altrimenti rifiuta. \square

Con prossimo problema, chiamato *test del vuoto* per il linguaggio di un automa finito, si vuole determinare se un DFA accetta una qualche stringa.

Lemma. *Sia E_{DFA} il linguaggio contenente le codifiche di tutti i DFA che non accettano alcuna stringa*

$$E_{DFA} = \{ \langle A \rangle \mid A \text{ è un DFA e } L(A) = \emptyset \}$$

E_{DFA} è un linguaggio decidibile.

Dimostrazione. Ciascun DFA accetta almeno una stringa se dallo stato iniziale è possibile raggiungere uno stato di accettazione percorrendo il verso delle frecce; posso quindi trattare i DFA come grafi connessi orientati e progettare una MdT T che esegua la marcatura degli stati per implementare tale controllo.

T marca lo stato iniziale e, successivamente, marca tutti gli stati che hanno una transizione proveniente da uno stato già marcato finché non si esauriscono gli stati inesplorati. Se nessuno stato di accettazione è marcato accetta, altrimenti rifiuta. \square

Lemma. *Sia EQ_{DFA} il linguaggio contenente tutte le coppie formate dalle codifiche di due DFA che riconoscono lo stesso linguaggio*

$$EQ_{DFA} = \{ \langle A, B \rangle \mid A \text{ e } B \text{ sono DFA tali che } L(A) = L(B) \}$$

EQ_{DFA} è un linguaggio decidibile.

Dimostrazione. A partire da A e B costruisco un nuovo DFA C che accetta solo quella stringhe che sono accettate da A o da B ma non da entrambi. In particolare, se A e B riconoscono lo stesso linguaggio, C non accetterà nulla

$$L(C) = (L(A) \cap \overline{L(B)}) \cup (\overline{L(A)} \cap L(B))$$

Quindi $L(C) = \emptyset \Leftrightarrow L(A) = L(B)$. \square