Marco Casu

Automi, Calcolabilità e Complessità

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Informatica

Questo documento è distribuito sotto la licenza GNU, è un resoconto degli appunti (eventualmente integrati con libri di testo) tratti dalle lezioni del corso di Automi, Calcolabilità e Complessità per la laurea triennale in Informatica. Se dovessi notare errori, ti prego di segnalarmeli.

INDICE

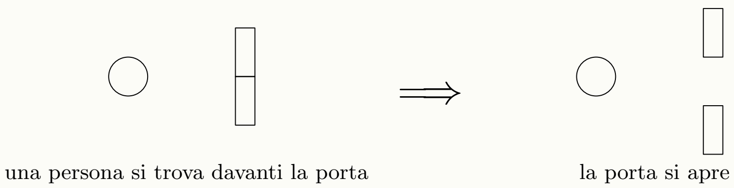
CAPITOLO

1 AUTOMI

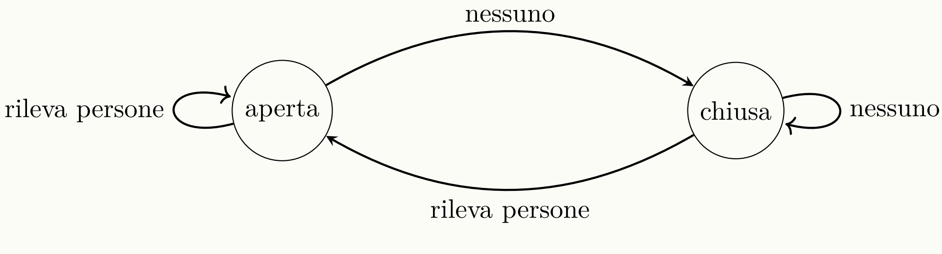
1.1 Linguaggi Regolari

Un *automa a stati finiti* è, seppure limitato nella memoria e nella gestione dell'input, il più semplice modello di computazione. Un automa può interagire con l'input esclusivamente "scorrendolo" in maniera sequenziale.

Esempio : Si vuole modellare una semplice porta con sensore, che si apre quando qualcuno si trova nelle vicinanze.



Un automa che modella il problema è il seguente :



Un automa ha alcuni stati speciali, come quello iniziale, indicato con un apposita freccia, e degli stati detti *di accettazione*, ossia stati in cui deve necessariamente terminare la computazione per essere definita valida, vengono rappresentati con un doppio cerchio.

Il modello di calcolo degli automi è riconducibile al concetto di *linguaggio regolare*, che verrà formalizzato in seguito, segue ora una definizione formale di automa.

Definizione (DFA) : : Un DFA (Deterministic Finite Automa) è una 5-tupla, $(Q, \Sigma, \delta, q_0, F)$ di cui

- Q è l'insieme degli stati possibili
- Σ è l'alfabeto che compone le stringhe in input

- δ è una mappa $Q \times \Sigma \to Q$ detta *funzione di transizione*.
- $q_0 \in Q$ è lo stato iniziale.
- $F \subseteq Q$ è l'insieme degli stati di accettazione.

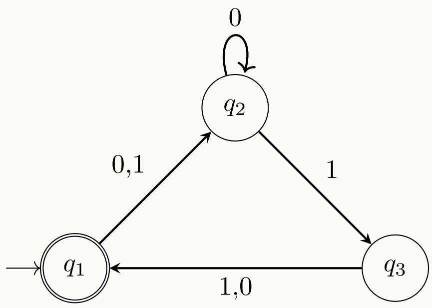


Figura 1.1: semplice automa

Nell'esempio in figura 1.1, si ha che

- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{0, 1\}$
- $F = \{q_1\}$
- $q_0 = q_1$
- $\delta =$

	0	1
q_1	q_2	q_2
q_2	q_2	q_3
q_3	q_1	q_1

Sia D un DFA, chiamiamo **linguaggio dell'automa**, e denotiamo $L(D)$, l'insieme delle stringhe che date in input a D fanno si che D termini su uno stato di accettazione. Per definire formalmente un linguaggio

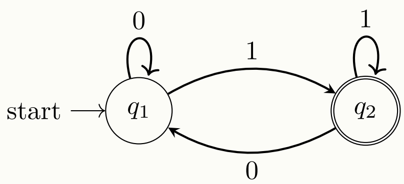


Figura 1.2: il linguaggio di tale automa risulta essere composto dalle stringhe che terminano con 1

di un automa, è necessario introdurre la **funzione di transizione estesa**:

$$\delta^*(q, \epsilon) = \delta(q, \epsilon)$$

$$\delta^*(q, ax) = \delta^*(\delta(q, a), x)$$

dove

$$a \in \Sigma, \quad x \in \Sigma^*, \quad \epsilon = \text{ stringa vuota}$$

Σ^* è l'insieme di tutte le stringhe formate dall'alfabeto Σ. Passiamo ora alla definizione di **configurazione**, essa rappresenta lo stato dell'automa ad un certo punto della computazione, essa è formata da una coppia

$$Q \times \Sigma^*$$

Rappresentante uno stato, ed una stringa di input rimanente da computare.

Un **passo della computazione** in un automa rappresenta una transizione da una configurazione ad un altra, è una relazione binaria $\vdash_D: Q \times \Sigma^*$ tale che

$$(p, ax) \vdash_D (q, x) \iff \delta(p, a) = q \quad \text{dove} \quad p, q \in Q, \quad a \in \Sigma, \quad x \in \Sigma^*$$

Si può estendere la definizione di passo di computazione, considerando la sua *chiusura transitiva* $\vdash_D^*$. Essa si ottiene aggiungendo a $\vdash_D$ tutte le coppie in $Q \times \Sigma^*$ che rendono la relazione chiusa rispetto la riflessività e rispetto la transitività.

$$(q, aby) \vdash_D (p, by) \ \wedge \ (p, by) \vdash_D (ry) \implies (q, aby) \vdash_D^* (r, y)$$

Ad esempio, nell'automa in figura 1.2, risulta chiaro che

$$\begin{cases} (q_1, 011) \vdash_D (q_1, 11) \\ (q_1, 11) \vdash_D (q_2, 1) \\ (q_2, 1) \vdash_D (q_2, \epsilon) \end{cases} \implies (q_1, 011) \vdash_D^* (q_2, \epsilon)$$

Inoltre

$$\begin{gathered} \delta^*(q_1, 011) = \\ \delta^*(q_1, 11) = \\ \delta^*(q_2, 1) = \\ \delta^*(q_2, \epsilon) = q_2 \end{gathered}$$

Se non specificato diversamente, con ϵ verrà indicata la stringa vuota. Utilizzando le precedenti definizioni, è possibile definire formalmente quali sono gli input accettati da un DFA.

Definizione : : Sia $D = (Q, \Sigma, \delta, q_0, F)$ un DFA, e sia $x \in \Sigma^*$ una stringa, essa è **accettata** da D se

$$\delta^*(q_0, x) \in F$$

Il **linguaggio riconosciuto** da D è

$$☞ \quad L(D) = \{x \in \Sigma^* | \delta^*(q_0, x) \in F\} \quad ☜$$

Definizione (Linguaggi Regolari) : : L'insieme dei linguaggi regolari, denotato REG, contiene tutti i linguaggi, tali che esiste un DFA che li ha come linguaggi riconosciuti.

$$☞ \quad REG = \{L \ | \ \exists D = (Q, \Sigma, \delta, q_0, F) \ \texttt{t.c.} \quad L \in \Sigma^* \wedge L(D) = L\} \quad ☜$$

Uno fra gli scopi di questo corso riguarda il capire come progettare automi, e capire se, ogni linguaggio è regolare, o ce ne sono alcuni che non possono essere riconosciuti da alcun possibile DFA.

1.1.1 Esempi di DFA

Vediamo in questa sezione alcuni semplici esempi di DFA.

Esempio 1) Si vuole progettare un DFA che accetti il seguente linguaggio

$$\{x \in \{0, 1\}^* \mid w_h(x) \geq 3\}$$

Si ricordi come

$$w_h(x) = \text{ occorrenze di 1 in } x$$

Una volta progettato il DFA, è anche importante dimostrarne la correttezza, ossia dare una prova matematico che l'automa in questione accetti il linguaggio.

- Se $x \in L(D)$ allora D accetta x
- Se D accetta x allora $w_h(x) \geq 3$

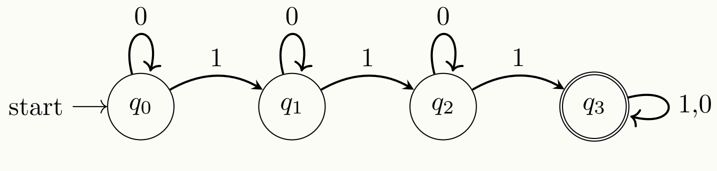


Figura 1.3: Esempio (1) di DFA

In questo, e nei seguenti casi, essendo i DFA estremamente semplici, risulta ovvio che accettino il dato linguaggio, in casi più avanzati, sarà necessario fornire una dimostrazione rigorosa.

Esempio 2) Si vuole progettare un DFA che accetti il seguente linguaggio

$$\{x \in \{0,1\}^* \mid x = 1y \land y \in \{0,1\}^*\}$$

Appunto sulla notazione : Se $a \in \Sigma^*$ e $b \in \Sigma^*$, allora con ab si denota la concatenazione di stringhe.

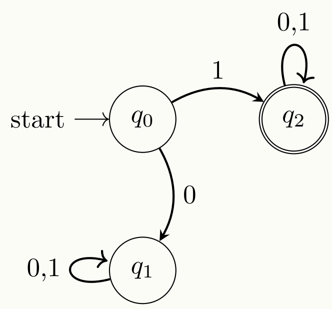


Figura 1.4: Esempio (2) di DFA

Nell'esempio (2), quando dallo stato q_0 il DFA riceve in input 0, la computazione cade su uno stato "buco nero", dalla quale non si può uscire a prescindere dall'input, l'operazione che fa cadere in questo stato è da considerarsi "non definita" in quanto non porterà mai la computazione a terminare su uno stato accettabile, è quindi comodo rimuovere tale stato dal diagramma.

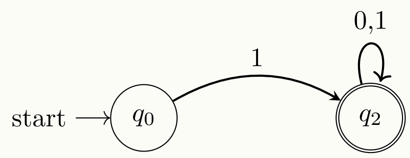


Figura 1.5: Esempio (2.1) di DFA

Anche in questo caso la dimostrazione della correttezza risulta banale.

Esempio 3) Si vuole progettare un DFA che accetti il seguente linguaggio

$$\{x \in \{0,1\}^* \mid x = 0^n 1, \ \ n \in \mathbb{N}\}$$

Con $0^n 1$ si intende una stringa che abbia un numero naturale di di 0 (quindi, almeno uno), e che termini con 1.

Dimostrazione di correttezza : Sia x una stringa data in input al DFA

- se x ha un numero naturale di 0, sicuramente ne ha almeno 1, quindi ad un certo punto della computazione passerà dallo stato q_0 allo stato q_1

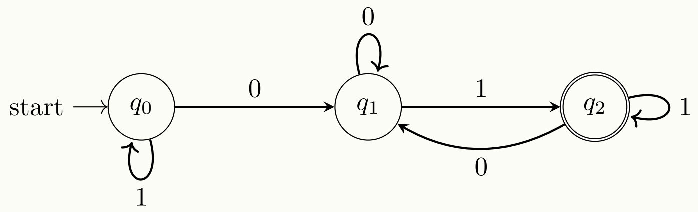


Figura 1.6: Esempio (3) di DFA

- la computazione, una volta passata per lo stato q_1, non tornerà più nello stato q_0, quindi le configurazioni successive si troveranno solamente in uno stato fra q_1 e q_2.
- supponiamo x termini con 1 :
  - Se la configurazione si trova su q_1, si ha $\delta(q_1, 1) = q_2$, la computazione termina su uno stato accettabile.
  - Se la configurazione si trova su q_2, si ha $\delta(q_2, 1) = q_2$, la computazione termina su uno stato accettabile.

  In entrambi i casi, essendo che l'automa termina su uno stato accettabile, esso riconosce il linguaggio, che risulta quindi, regolare. ■

1.2 Operazioni sui Linguaggi

Lo studio delle proprietà dei linguaggi regolari può fornire opportune accortezze utili nella progettazione di automi, siccome i linguaggi sono insiemi di stringhe costruiti su un alfabeto Σ, essi godono delle operazioni insiemistiche.

Risulta utile definire formalmente la concatenazione fra stringhe, siano

$$x = a_1, a_2 \dots, a_n \qquad y = b_1, b_2 \dots, b_n$$

due stringhe, esse possono essere concatenate

$$xy = a_1, a_2 \dots, a_n, b_1, b_2 \dots, b_n$$

L'operazione di concatenazione non è commutativa, può essere definita ricorsivamente in tal modo :

$$x(ya) = (xy)a$$

dove

$$x, y \in \Sigma^* \qquad a \in \Sigma$$

Siano L_1, L_2 due linguaggi regolari in *REG* (per semplicità, definiti su uno stesso alfabeto Σ), e sia n un numero naturale, sono definite su di essi le seguenti operazioni :

- **unione** : $L_1 \cup L_2 = \{x \in \Sigma^* \mid x \in L_1 \lor x \in L_2\}$
- **intersezione** : $L_1 \cap L_2 = \{x \in \Sigma^* \mid x \in L_1 \land x \in L_2\}$
- **complemento** : $\neg L_1 = \{x \in \Sigma^* \mid x \notin L_1\}$
- **concatenazione** : $L_1 \circ L_2 = \{xy \mid x \in L_1 \land y \in L_2\}$
- **potenza** : $L_1^n = \underbrace{L_1 \circ L_1 \circ L_1, \cdots \circ L_1}_{n \text{ volte}}$

- **star** : $L_1^* = \{x_1, x_2 \dots, x_k \mid k \in \mathbb{Z}^+ \land x_i \in L_1\}$
  Si può definire anche diversamente
$$L_1^* = \bigcup_{k=0}^{\infty} L_1^k$$

Una particolarità dei linguaggi regolari *REG*, è che sono chiusi rispetto a tutte le operazioni appena elencate.

$$L_1 \cup L_2 \in REG \qquad L_1 \circ L_2 \in REG \qquad L_1 \cap L_2 \in REG$$

$$L_1^n \in REG \qquad \neg L_1 \in REG \qquad L_1^* \in REG$$

Esempio di concatenazione e potenza :

$$\Sigma = \{a, b\} \quad L_1 = \{a, ab, ba\} \quad L_2 = \{ab, b\} \quad L = \{a, ab, ba\}$$

$$L_1 \circ L_2 = \{aab, ab, abab, abb, baab, bab\}$$

$$L^2 = \{aa, aab, aba, abab, abba, baa, baba\}$$