"Sapienza" Università di Roma
Ingegneria dell'Informazione,
Informatica e Statistica
Dipartimento di Informatica

Automi: Calcolabilità e Complessità

Appunti integrati con il libro "Introduzione alla teoria della computazione",
Michael Sipser

Author
Alessio Bandiera

16 settembre 2023

Indice

Informazioni e Contatti **1**

1 Linguaggi ed espressioni regolari **2**
- 1.1 Automi finiti 2
  - 1.1.1 Linguaggi 2
  - 1.1.2 Determinismo 3
- 1.2 Non determinismo 6
  - 1.2.1 Definizioni 6
- 1.3 Operazioni regolari 9
  - 1.3.1 Definizioni 9
- 1.4 Espressioni regolari 14
  - 1.4.1 Definizioni 14
  - 1.4.2 Non determinismo generalizzato 16
- 1.5 Linguaggi non regolari 21
  - 1.5.1 Pumping lemma 21

2 Linguaggi e grammatiche context-free **23**
- 2.1 Grammatiche context-free 23
  - 2.1.1 Definizioni 23
- 2.2 Automi a pila 26
  - 2.2.1 Definizioni 26

Informazioni e Contatti

Prerequisiti consigliati:

- TODO: DA DECIDERE

Segnalazione errori ed eventuali migliorie:

Per segnalare eventuali errori e/o migliorie possibili, si prega di utilizzare il **sistema di Issues fornito da GitHub** all'interno della pagina della repository stessa contenente questi ed altri appunti (link fornito al di sotto), utilizzando uno dei template già forniti compilando direttamente i campi richiesti.

Gli appunti sono in continuo aggiornamento, pertanto, previa segnalazione, si prega di controllare se l'errore sia ancora presente nella versione più recente.

Licenza di distribuzione:

Contatti dell'autore e ulteriori link:

- Github: **https://github.com/ph04**
- Email: **alessio.bandiera02@gmail.com**
- LinkedIn: **Alessio Bandiera**

1

Linguaggi ed espressioni regolari

1.1 Automi finiti

1.1.1 Linguaggi

Definizione 1.1.1.1: Alfabeto

Si definisce **alfabeto** un qualsiasi insieme finito, non vuoto; i suoi elementi sono detti **simboli** o **caratteri**.

Esempio 1.1.1.1 (Alfabeto)**.** $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$ è un alfabeto, composto da 5 simboli.

Definizione 1.1.1.2: Stringa

Sia Σ un alfabeto; una **stringa su** Σ è una sequenza finita di simboli di Σ; la **stringa vuota** è denotata con ε.

- Data una stringa w di Σ, allora $|w|$ è la lunghezza di w.
- Se w ha lunghezza $n \in \mathbb{N}$, allora è possibile scrivere che $w = w_1 w_2 \cdots w_n$ con $w_i \in \Sigma$ e $i \in [1, n]$.

Esempio 1.1.1.2 (Stringa)**.** Sia $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{x}, \mathtt{y}, \mathtt{z}\}$ un alfabeto; allora una sua possibile stringa è $w = \mathtt{x1y0z}$.

Definizione 1.1.1.3: Stringa inversa

Sia Σ un alfabeto, e $w = w_1 w_2 \cdots w_n$ una sua stringa; allora si definisce l'**inversa** di w come segue:

$$w^{\mathcal{R}} := w_n w_{n-1} \cdots w_1$$

Definizione 1.1.1.4: Concatenazione

Sia Σ un alfabeto, e $x = x_1 x_2 \cdots x_n, y = y_1 y_2 \cdots y_n$ due sue stringhe; allora xy è la stringa ottenuta attraverso la **concatenazione** di x ed y.

Per indicare una stringa concatenata con se stessa k volte, si utilizza la notazione

$$x^k = \underbrace{xx \cdots x}_{k \text{ volte}}$$

Si noti che per ogni stringa x su Σ, si ha che $x^0 = \varepsilon$.

Definizione 1.1.1.5: Prefisso

Sia Σ un alfabeto, ed x, y due sue stringhe; allora x è detto essere un **prefisso** di y, se $\exists z \mid xz = y$, con z stringa in Σ.

Esempio 1.1.1.3 (Prefisso)**.** Sia $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ un alfabeto; allora la stringa $x = \mathtt{ab}$ è prefisso della stringa $y = \mathtt{abc}$, poiché esiste una stringa $z = \mathtt{c}$ tale per cui $xz = y$.

Definizione 1.1.1.6: Linguaggio

Sia Σ un alfabeto; si definisce **linguaggio** un insieme di stringhe di Σ. Un linguaggio è detto **prefisso**, se nessun suo elemento è prefisso di un altro. Il linguaggio vuoto si indica con $\emptyset$.

1.1.2 Determinismo

Definizione 1.1.2.1: DFA

Un **DFA** (*Deterministic Finite Automaton*) è una quintupla $(Q, \Sigma, \delta, q_0, F)$, dove

- Q è l'**insieme degli stati** dell'automa, un insieme *finito*
- Σ è l'**alfabeto** dell'automa, un insieme *finito*
- $\delta : Q \times \Sigma \to Q$ è la **funzione di transizione**, che definisce la relazione tra gli stati
- $q_0 \in Q$ è lo **stato iniziale**
- $F \subseteq Q$ è l'**insieme degli stati accettanti**, sui quali le stringhe possono terminare

Esempio 1.1.2.1 (DFA)**.** Un esempio di DFA è il seguente:

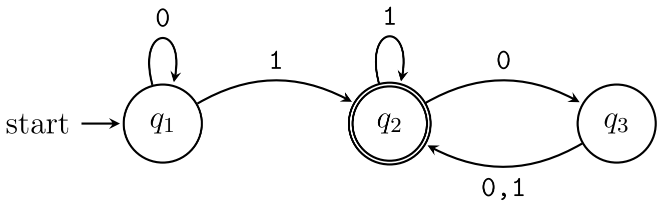


Figura 1.1: Un DFA.

esso può essere descritto secondo la quintupla $(Q, \Sigma, \delta, q_0, F)$ come segue:

- $Q = \{q_1, q_2, q_3\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- δ è la seguente:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

- q_1 è lo stato iniziale
- $F = \{q_2\} \subseteq Q$

Definizione 1.1.2.2: Linguaggio di un automa

Sia M un automa; allora il **linguaggio di** M è un insieme $L(M)$ contenente tutte le stringhe accettate da M; simmetricamente, si dice che M **riconosce** $L(M)$.

Esempio 1.1.2.2 (Linguaggio di un automa)**.** Si consideri il seguente automa M_1:

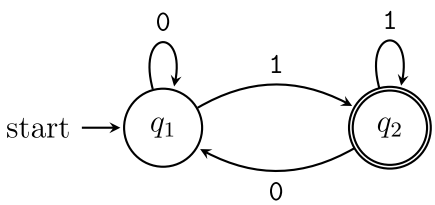


Figura 1.2: Un automa M_1.

sapendo che $\Sigma = \{\mathtt{0}, \mathtt{1}\}$, che q_1 è lo stato iniziale, e che $F = \{q_2\}$, ci si convince facilmente che

$$L(M_1) = \{w \mid w = w_1 w_2 \cdots w_{n-1}\mathtt{1}, n \in \mathbb{N}\}$$

ovvero, M_1 accetta tutte e sole le stringhe che terminano per `1`.

Definizione 1.1.2.3: Stringhe accettate (DFA)

Sia $M = (Q, \Sigma, \delta, q_0, F)$ un DFA, e sia $w = w_1 \cdots w_n$ una stringa tale per cui $\forall i \in [1, n] \quad w_i \in \Sigma$; allora, M **accetta** w se esiste una sequenza di stati $r_0, \ldots, r_n \in Q$ tali per cui

- $r_0 = q_0$
- $\forall i \in [1, n-1] \quad \delta(r_i, w_{i+1}) = r_{i+1}$
- $r_n \in F$

Dato un linguaggio A, si ha che M **riconosce** A se e solo se $A = \{w \mid M \text{ accetta } w\}$.

Definizione 1.1.2.4: Linguaggio regolare

Un linguaggio è detto **regolare** se e solo se esiste un DFA che lo riconosce.

1.2 Non determinismo

1.2.1 Definizioni

Definizione 1.2.1.1: NFA

Un **NFA** (*Nondeterministic Finite Automaton*) è un automa in cui possono esistere varie scelte per lo stato successivo in ogni punto. Durante la computazione, ogni volta che viene incontrata una scelta, la macchina si *divide*, e ognuno dei vari automi risultanti computa le varie scelte indipendentemente.

Formalmente, un NFA è una quintupla $(Q, \Sigma, \delta, q_0, F)$, dove

- Q è l'**insieme degli stati**, un insieme *finito*
- Σ è l'**alfabeto** dell'automa, un insieme *finito*
- $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ è la **funzione di transizione**, che definisce la relazione tra gli stati
- $q_0 \in Q$ è lo **stato iniziale**
- $F \subseteq Q$ è l'**insieme degli stati accettanti**

dove $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$.

Se il simbolo di input successivo non compare su alcuno degli archi uscenti dallo stato occupato da una copia della macchina, quella copia cessa di proseguire; inoltre, se *una qualunque copia* della macchina è in uno stato accettante, l'NFA accetta la stringa di input. Si noti che questa divisione è descritta dall'insieme potenza $\mathcal{P}(Q)$, poiché da ogni stato si può arrivare ad un *insieme* di stati.

Si noti che il determinismo è un caso particolare di non determinismo, dunque un DFA è sempre anche un NFA.

Esempio 1.2.1.1 (NFA)**.** Un esempio di NFA è il seguente:

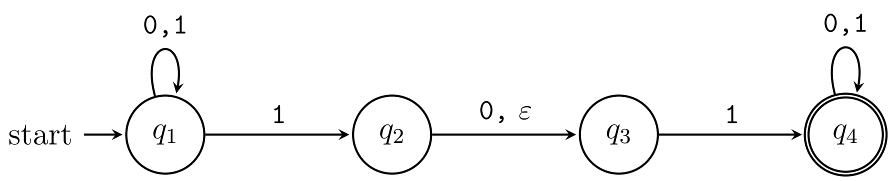


Figura 1.3: Un NFA.

esso può essere descritto secondo la quintupla $(Q, \Sigma, \delta, q_0, F)$ come segue:

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- δ è la seguente:

	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

- q_1 è lo stato iniziale
- $F = \{q_4\} \subseteq Q$

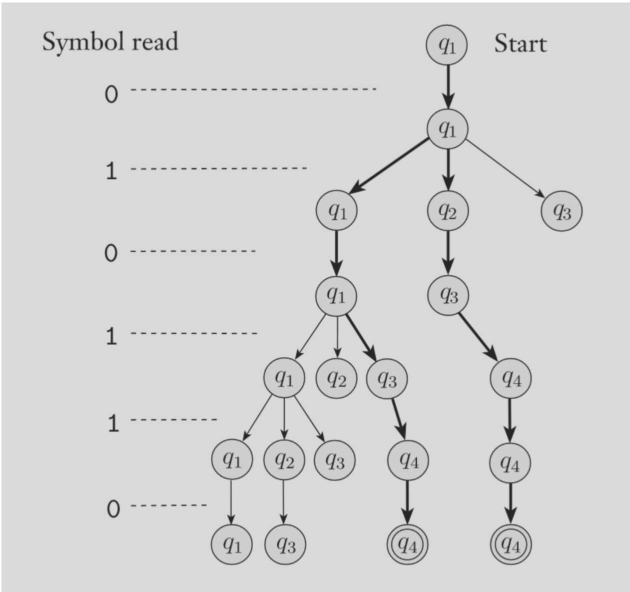


Figura 1.4: L'immagine rappresenta l'esecuzione dell'NFA descritto.

Si noti che nel momento in cui vengono incontrati ε-archi si giunge allo stato successivo nello stesso step dell'input incontrato, dunque gli ε-archi *non* producono uno step di esecuzione, ma vengono eseguiti non appena vengono incontrati, e vengono accorpati immediatamente.

Definizione 1.2.1.2: Stringhe accettate (NFA)

Sia $N = (Q, \Sigma, \delta, q_0, F)$ un NFA, e sia $w = w_1 \cdots w_n$ una stringa tale per cui $\forall i \in [1, n] \quad w_i \in \Sigma$; allora, M **accetta** w se esiste una sequenza di stati $r_0, \ldots, r_n \in Q$ tali per cui

- $r_0 = q_0$
- $\forall i \in [1, n-1] \quad r_{i+1} \in \delta(r_i, w_{i+1})$
- $r_n \in F$

Definizione 1.2.1.3: Equivalenza tra automi

Due macchine si dicono **equivalenti** se e solo se riconoscono lo stesso linguaggio.

Teorema 1.2.1.1: DFA equivalente ad NFA

Sia N un NFA; allora, esiste un DFA ad esso equivalente.

Dimostrazione. Sia $N = (Q, \Sigma_\varepsilon, \delta, q_0, F)$ l'NFA in ipotesi, tale da riconoscere un linguaggio A. Inoltre, si definisca

$$\forall k \geq 0 \quad E(R) := \bigcup_{r \in R} \delta^k(r, \varepsilon)$$

l'insieme degli stati raggiungibili da stati in R, applicando (anche ripetutamente) un numero arbitrario di ε-archi (si noti che per $k = 0$ si ha che $R \subseteq E(R)$).

Allora, sia $M = (Q', \Sigma, \delta', q_0', F)$ il DFA definito come segue:

- $Q' := \mathcal{P}(Q)$, scelto tale da rappresentare ogni possibile stato di N;
- $\forall R \in Q', a \in \Sigma \quad \delta'(R, a) := \bigcup_{r \in R} E(\delta(r, a))$, scelta tale in quanto, per un certo insieme di stati $R \in Q'$ di N, a $\delta'(R, a)$ viene assegnata l'unione degli stati che sarebbero stati raggiunti in N dagli $r \in R$ con a, calcolati dunque attraverso $\delta(r, a)$, aggiungendo infine i possibili ε-archi;
- $q_0' := E(\{q_0\})$, scelto tale da far iniziare M esattamente dove aveva inizio N, comprendendo anche i possibili ε-archi iniziali;
- $F' := \{R \in Q' \mid \exists r \in R : r \in F\}$, che corrisponde all'insieme degli insiemi di stati di N contenenti almeno uno stato accettante in N.

Allora M è in grado di riconoscere A per costruzione, poiché il DFA costruito emula l'NFA di partenza, tenendo anche in considerazione gli ε-archi. Dunque, sia N che M riconoscono A, e per definizione sono di conseguenza equivalenti. □

Corollario 1.2.1.1: Linguaggi regolari con NFA

Un linguaggio è regolare se e solo se esiste un NFA che lo riconosce.

Dimostrazione.

Prima implicazione. Sia A un linguaggio regolare, e dunque per definizione esiste un DFA che lo riconosce. Allora, poiché un DFA è sempre anche NFA, segue la tesi.

Seconda implicazione. Sia A un linguaggio tale da essere riconosciuto da un NFA; allora, per il Teorema 1.2.1.1, esiste un DFA equivalente all'NFA che riconosce A, e dunque per definizione A è regolare.

□

1.3 Operazioni regolari

1.3.1 Definizioni

Definizione 1.3.1.1: Unione

Siano A e B due linguaggi; allora, si definisce l'**unione** di A e B il seguente linguaggio:

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$

Esempio 1.3.1.1 (Unione)**.** Sia $\Sigma = \{\mathtt{a}, \ldots, \mathtt{z}\}$ l'alfabeto composto da 26 lettere, e siano $A = \{\mathtt{uno}, \mathtt{due}\}$ e $B = \{\mathtt{tre}, \mathtt{quattro}\}$ due linguaggi su Σ. Allora, si ha che

$$A \cup B = \{\mathtt{uno}, \mathtt{due}, \mathtt{tre}, \mathtt{quattro}\}$$

Proposizione 1.3.1.1: Chiusura dell'unione

Siano A e B due linguaggi regolari su un alfabeto Σ; allora $A \cup B$ è regolare.

Dimostrazione I. Per definizione, A e B sono linguaggi regolari, dunque esistono due automi finiti

$$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

tali da riconoscere rispettivamente A e B. Allora, sia $M = (Q, \Sigma, \delta, q_0, F)$ l'automa finito definito come segue:

- $Q := Q_1 \times Q_2 = \{(r_1, r_2) \mid r_1 \in Q_1 \wedge r_2 \in Q_2\}$, scelto tale in quanto permette di avere tutte le possibili combinazioni di stati dei due automi di partenza;
- $\forall (r_1, r_2) \in Q, a \in \Sigma \quad \delta((r_1, r_2), a) := (\delta_1(r_1, a), \delta_2(r_2, a))$, scelta tale in quanto permette di simulare entrambi gli automi di partenza contemporaneamente, mandando ogni stato di M_1 ed M_2 dove sarebbe andato nei rispettivi automi di appartenenza;
- $q_0 := (q_1, q_2)$, scelto tale in quanto deve essere lo stato in cui entrambe gli automi in ipotesi iniziavano;
- $F := (F_1 \times Q_2) \cup (Q_1 \times F_2) = \{(r_1, r_2) \mid r_1 \in F_1 \vee r_2 \in F_2\}$, scelto tale in quanto permette di simulare gli stati accettanti di entrambi gli automi, e vanno prese tutte le coppie che vedono almeno uno dei due stati come accettanti poiché altrimenti non si accetterebbero delle stringhe accettate da M_1 ed M_2 in partenza.

Allora, poiché M è in grado di simulare M_1 ed M_2 contemporaneamente, per costruzione accetterà ogni stringa di A e di B, dunque riconoscendo $A \cup B$, e di conseguenza $A \cup B$ è regolare per definizione. □

Dimostrazione II. Per definizione, A e B sono linguaggi regolari, dunque per il Teorema 1.2.1.1 esistono due NFA

$$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

tali da riconoscere rispettivamente A e B. Allora, sia $N = (Q, \Sigma, \delta, q_0, F)$ l'NFA costruito come segue:

- $Q := \{q_0\} \cup Q_1 \cup Q_2$, dove q_0 è un nuovo stato, e Q è scelto tale da includere sia N_1 che N_2;
- $\forall q \in Q, a \in \Sigma_\varepsilon \quad \delta(q, a) := \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \wedge a = \varepsilon \\ \emptyset & q = 0 \wedge a \neq \varepsilon \end{cases}$, scelta tale da poter eseguire contemporaneamente gli NFA N_1 ed N_2, definendo per casi la funzione di transizione; infatti, si noti che si è posta $\delta(q_0, \varepsilon) := \{q_1, q_2\}$ in modo da collegare il nuovo stato q_0 a q_1 e q_2, gli stati iniziali di N_1 ed N_2 rispettivamente;
- q_0 è il nuovo stato, che rappresenta lo stato iniziale di N;
- $F := F_1 \cup F_2$, scelto tale da costruire N in modo che accetti una stringa se e solo se la accetterebbero N_1 o N_2.

Allora, l'NFA risultante N è in grado di computare contemporaneamente N_1 ed N_2, ed è dunque in grado di riconoscere A e B contemporaneamente; di conseguenza, N riconosce $A \cup B$, che risulta dunque essere regolare per definizione.

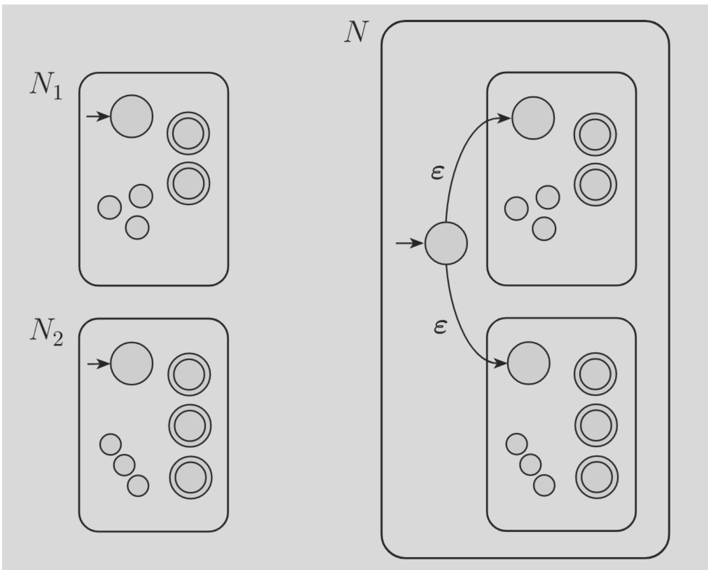


Figura 1.5: L'immagine raffigura la dimostrazione costruttiva descritta.

$\square$

Definizione 1.3.1.2: Concatenazione

Siano A e B due linguaggi; allora, si definisce la **concatenazione** di A e B il seguente linguaggio:

$$A \circ B = \{xy \mid x \in A \wedge y \in B\}$$

Esempio 1.3.1.2 (Concatenazione)**.** Sia $\Sigma = \{\texttt{a}, \ldots, \texttt{z}\}$ l'alfabeto composto da 26 lettere, e siano $A = \{\texttt{uno}, \texttt{due}\}$ e $B = \{\texttt{tre}, \texttt{quattro}\}$ due linguaggi su Σ. Allora, si ha che

$$A \circ B = \{\texttt{unotre}, \texttt{unoquattro}, \texttt{duetre}, \texttt{duequattro}\}$$

Proposizione 1.3.1.2: Chiusura della concatenazione

Siano A e B due linguaggi regolari su un alfabeto Σ; allora $A \circ B$ è regolare.

Dimostrazione. Per definizione, A e B sono linguaggi regolari, dunque per il Teorema 1.2.1.1 esistono due NFA

$$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$
$$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$$

tali da riconoscere rispettivamente A e B. Allora, sia $N = (Q, \Sigma, \delta, q_0, F)$ l'NFA costruito come segue:

- $Q := Q_1 \cup Q_2$, scelto tale da includere entrambe gli automi N_1 ed N_2 di partenza;
- $q_0 := q_1$, scelto tale da far iniziare l'esecuzione dell'automa su N_1;
- $F := F_2$, scelto tale da far terminare l'esecuzione dell'automa su N_2;
- $\forall q \in Q, a \in \Sigma_\varepsilon \quad \delta(q, a) := \begin{cases} \delta_1(q, a) & q \in Q_1 - F_1 \lor (q \in F_1 \land a \neq \varepsilon) \\ \delta_1(q, a) \cup \{q_2\} & q \in F_1 \land a = \varepsilon \\ \delta_2(q, a) & q \in Q_2 \end{cases}$,
  scelta tale da anteporre l'esecuzione di N_1 a quella di N_2; infatti, se $q \in Q_1 - F_1$ (è uno stato non accettante di N_1), oppure $q \in F_1$ ma $a \neq \varepsilon$, l'esecuzione di N_1 non viene alterata; diversamente, se invece $q \in F_1$ e $a = \varepsilon$, all'insieme di stati $\delta_1(q, \varepsilon)$ viene aggiunto q_2, ovvero lo stato iniziale di N_2, in modo da effettuare la concatenazione tra i due NFA non deterministicamente.

Allora, l'NFA N costruito computa inizialmente N_1, e se vengono raggiunti suoi stati accettanti, l'esecuzione prosegue attraverso N_2, al fine di realizzare la concatenazione tra le stringhe. Di conseguenza, l'automa è in grado di riconoscere $A \circ B$ per costruzione, e dunque $A \circ B$ è regolare per definizione.

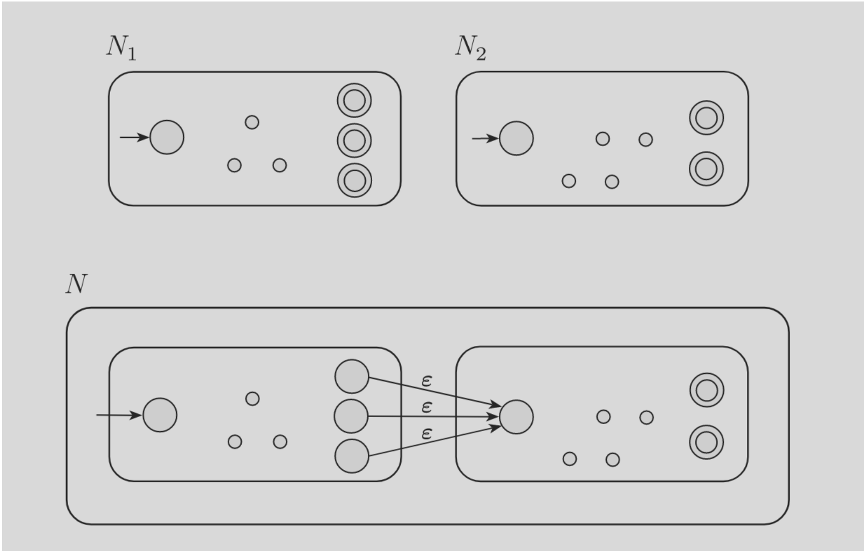


Figura 1.6: L'immagine raffigura la dimostrazione costruttiva descritta.

□

Definizione 1.3.1.3: Star

Sia A un linguaggio; allora, si definisce l'operazione unaria **star** che definisce il seguente linguaggio:

$$A^* = \{x_1 x_2 \cdots x_k \mid k \geq 0 \land \forall i \in [1, k] \quad x_i \in A\}$$

Si noti che $k = 0 \implies \varepsilon \in A^*$ per ogni linguaggio A.

Esempio 1.3.1.3 (Star)**.** Sia $\Sigma = \{\mathtt{a}, \ldots, \mathtt{z}\}$ l'alfabeto composto da 26 lettere, e siano $A = \{\mathtt{uno}, \mathtt{due}\}$ e $B = \{\mathtt{tre}, \mathtt{quattro}\}$ due linguaggi su Σ. Allora, si ha che

$$A^* = \{\varepsilon, \mathtt{uno}, \mathtt{due}, \mathtt{unouno}, \mathtt{unodue}, \mathtt{dueuno}, \mathtt{duedue}, \ldots\}$$

Proposizione 1.3.1.3: Chiusura dell'operazione star

Sia A un linguaggio regolare su un alfabeto Σ; allora A^* è regolare.

Dimostrazione. Per definizione, A è linguaggio regolare, dunque per il Teorema 1.2.1.1 esiste un NFA

$$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$$

tale da riconoscere A. Allora, sia $N = (Q, \Sigma, \delta, q_0, F)$ l'NFA costruito come segue:

- $Q := \{q_0\} \cup Q_1$, dove q_0 è un nuovo stato, posto prima di N_1;
- q_0 è il nuovo stato iniziale;
- $F := \{q_0\} \cup F_1$, poiché q_0 deve essere accettante, in modo tale da accettare ε (si noti che $\varepsilon \in A^*$ per ogni linguaggio A);

- $\forall q \in Q, a \in \Sigma_\varepsilon \quad \delta(q,a) := \begin{cases} \delta_1(q,a) & q \in Q_1 - F_1 \vee (q \in F_1 \wedge a \neq \varepsilon) \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \wedge a = \varepsilon \\ \{q_1\} & q = q_0 \wedge a = \varepsilon \\ \emptyset & q = q_0 \wedge a \neq \varepsilon \end{cases}$,

  scelta tale da ricominciare l'esecuzione dell'automa ogni volta che viene raggiunto uno stato accettante in N_1; infatti, se $q \in Q_1 - F_1$ (è uno stato non accettante di N_1), oppure q è accettante e $a \neq \varepsilon$, l'esecuzione procede normalmente con $\delta_1(q,a)$; differentemente, se $q \in F_1$ ma $a = \varepsilon$, allora l'esecuzione deve ricominciare da capo per poter effettuare la concatenazione multipla delle stringhe in A che caratterizzano l'operazione star, e dunque a $\delta_1(q,a)$ viene aggiunto q_1 (lo stato inziale di N_1); infine, ponendo $\delta(q_0,\varepsilon) := \{q_1\}$ si realizza l'ε-arco iniziale che collega q_0 (il nuovo stato) a q_1, al fine di rendere N in grado di accettare ε.

Allora, poichè l'NFA N è in grado di ricominciare l'esecuzione ogni volta che questa sarebbe terminata in N_1, è in grado di accettare molteplici copie concatenate delle stringhe in A, in maniera non deterministica, e dunque per definizione N riconosce A^*, il quale risulta allora essere regolare per definizione.

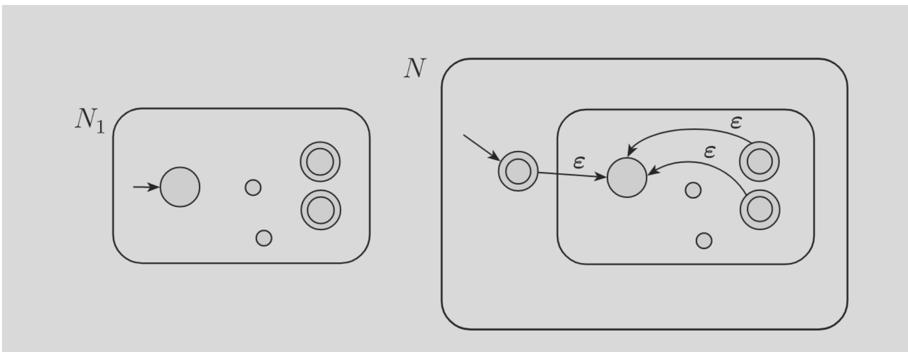


Figura 1.7: L'immagine raffigura la dimostrazione costruttiva descritta.

□

1.4 Espressioni regolari

1.4.1 Definizioni

Definizione 1.4.1.1: Espressione regolare

Sia Σ un alfabeto; allora, R si definisce **espressione regolare** se soddisfa una delle seguenti caratteristiche:

- $R = \emptyset$
- $R = \varepsilon$
- $R \in \Sigma$

Un'espressione regolare, dunque, è un modo compatto di definire un linguaggio. Si noti che le definizioni successive sono in grado di espandere la definizione appena fornita. L'insieme di tutte le espressioni su Σ è denotato con $\mathcal{R}$.

Data un'espressione regolare R, con $L(R)$ si intende il linguaggio che R descrive, ovvero l'insieme di stringhe che R rappresenta. Dunque, sono vere le seguenti:

- $L(\emptyset) = \emptyset$
- $L(\varepsilon) = \{\varepsilon\}$
- $\forall a \in \Sigma \quad L(a) = \{a\}$

Definizione 1.4.1.2: Unione

Siano R_1 ed R_2 due espressioni regolari; allora, si definisce l'**unione** di R_1 ed R_2 la seguente espressione regolare:

$$(R_1 \cup R_2)$$

e rappresenta uno qualsiasi dei caratteri di R_1 o di R_2.

Si noti che $R \cup \emptyset = \emptyset$ per qualsiasi espressione regolare R.

Esempio 1.4.1.1 (Unione)**.** Sia $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ un alfabeto; un esempio di espressione regolare di unione su Σ è il seguente:

$$R = (\mathtt{a} \cup \mathtt{c})$$

ed il valore di questa espressione equivale da $\mathtt{a}$ oppure $\mathtt{c}$, e dunque $L(R) = \{\mathtt{a}, \mathtt{c}\}$.

Esempio 1.4.1.2 (Espressioni regolari particolari)**.** Sia $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ un alfabeto; l'espressione regolare $(\mathtt{0} \cup \mathtt{1})$ rappresenta il linguaggio che consiste di tutte le stringhe di lunghezza 1 sull'alfabeto Σ, e dunque l'espressione regolare descritta si abbrevia generalmente con il simbolo Σ stesso.

Definizione 1.4.1.3: Concatenazione

Siano R_1 ed R_2 due espressioni regolari; allora, si definisce la **concatenazione** di R_1 ed R_2 la seguente espressione regolare:

$$(R_1 \circ R_2)$$

e rappresenta le stringhe che iniziano per R_1 e terminano con R_2.

Per indicare la concatenazione di R con sé stessa, si usa la seguente notazione

$$R^k := \underbrace{R \circ \cdots \circ R}_{k \text{ volte}}$$

Si noti che $R \circ \emptyset = \emptyset$ e $R \circ \varepsilon = R$, per qualsiasi espressione regolare R.

Esempio 1.4.1.3 (Concatenazione)**.** Sia $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ un alfabeto; un esempio di espressione regolare di concatenazione su Σ è il seguente:

$$R = (\mathtt{a} \circ \mathtt{c})$$

che può essere scritto equivalentemente come $\mathtt{ac}$, e dunque $L(R) = \{\mathtt{ac}\}$.

Definizione 1.4.1.4: Star

Sia R un'espressione regolare; allora, si definisce l'operazione unaria **star** su R la seguente espressione regolare:

$$(R^*)$$

e tutte rappresenta le stringhe che si possono ottenere concatenando un qualsiasi numero di caratteri di R, e descrive dunque il linguaggio che consiste di tutte le stringhe dell'alfabeto di R.

Si noti che R^* comprende ε per qualsiasi espressione regolare R.

Spesso si usa la notazione $R^+ := RR^*$, ovvero le stringhe che si ottengono attraverso la concatenazione di 1 o più stringhe di R. Di conseguenza, si ha che $R^+ \cup \varepsilon = R^*$.

Esempio 1.4.1.4 (Star)**.** Sia $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ un alfabeto; un esempio di espressione regolare di star su Σ è il seguente:

$$R = (\Sigma^*)$$

che descrive il linguaggio che consiste di tutte le stringhe sull'alfabeto, e dunque $L(R) = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{aa}, \mathtt{bb}, \mathtt{cc}, \ldots\}$.

Esempio 1.4.1.5 (Espressioni regolari)**.** Sia $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ un alfabeto; i seguenti sono esempi di espressioni regolari su Σ:

- $L(\mathtt{0}^*\mathtt{1}\mathtt{0}^*) = \{w \mid w \text{ contiene un solo } \mathtt{1}\}$;
- $L(\Sigma^*\mathtt{001}\Sigma^*) = \{w \mid w \text{ contiene la stringa } \mathtt{001} \text{ come sottostringa}\}$;

- $L(\mathtt{0}\Sigma^*\mathtt{0} \cup \mathtt{1}\Sigma^*\mathtt{1} \cup \mathtt{0} \cup \mathtt{1}) = \{w \mid w \text{ inizia e termina con lo stesso carattere}\}$, poiché si noti che l'ordine delle operazioni, a meno di parentesi, è (i) star, (ii) concatenazione, ed infine (iii) unione;
- $L((\mathtt{0} \cup \varepsilon)(\mathtt{1} \cup \varepsilon)) = \{\varepsilon, \mathtt{0}, \mathtt{1}, \mathtt{01}\}$;
- $L(\emptyset^*) = \{\varepsilon\}$, poiché $\emptyset$ rappresenta il linguaggio vuoto, e dunque l'unica stringa che si può ottenere concatenando un qualsiasi numero di volte elementi del linguaggio vuoto, è ε.

1.4.2 Non determinismo generalizzato

Definizione 1.4.2.1: GNFA

Un **GNFA** (*Generalized Nondeterministic Finite Automaton*) è una versione generalizzata di un NFA, in cui gli archi delle transizioni sono espressioni regolari sull'alfabeto dato.

Un GNFA legge blocchi di simboli dall'input, e si muove lungo gli archi che sono etichettati da espressioni regolari che possono descrivere il blocco di simboli letto.

Inoltre, essendo una versione generalizzata di un NFA, un GNFA può avere diversi modi di elaborare la stessa stringa di input, e accetta quest'ultima se la sua elaborazione può far si che il GNFA si trovi in uno stato accettante al suo termine.

Nota: All'interno di questi appunti, a meno di specifica, si assume che ogni GNFA preso in considerazione abbia:

- un solo stato di inizio, privo di archi entranti, connesso con ogni altro stato, ma non con sé stesso;
- un solo stato accettante, privo di archi uscenti, non connesso con altri archi;
- ogni altro stato collegato con ogni altro stato, a meno di quello iniziale, anche con sé stessi.

Formalmente, dato un alfabeto Σ, un GNFA del tipo appena descritto è una quintupla $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ definita come segue:

- Q è l'**insieme degli stati** dell'automa, un insieme *finito*
- Σ è l'**alfabeto** dell'automa, un insieme *finito*
- $\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \to \mathcal{R}$ è la **funzione di transizione**, che definisce la relazione tra gli stati; si noti che δ ha come dominio il prodotto cartesiano tra gli stati, e come codominio $\mathcal{R}$, poiché a differenza di un normale DFA o NFA, un GNFA prende come input 2 stati (che non possono essere né quello iniziale né quello accettante) e restituisce un'espressione regolare
- $q_{\text{start}} \in Q$ è lo **stato iniziale**
- q_{accept} è lo **stato accettante**

Esempio 1.4.2.1 (GNFA)**.** Il seguente è il digramma di un GNFA sull'alfabeto $\Sigma = \{\texttt{a}, \texttt{b}\}$.

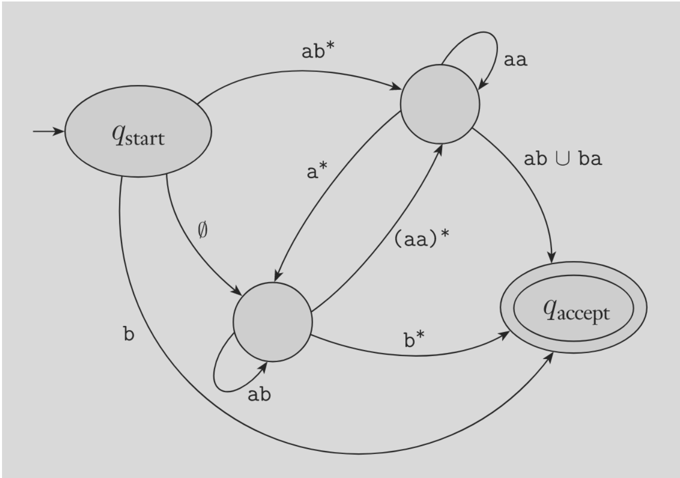


Figura 1.8: Un GNFA.

Definizione 1.4.2.2: Stringhe accettate (GNFA)

Sia $G = (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ un GNFA, e sia $w = w_1 \cdots w_n$ una stringa tale per cui $\forall i \in [1, n] \quad w_i \in \Sigma^*$; allora, G **accetta** w se esiste una sequenza di stati $q_0, \ldots, q_n \in Q$ tali per cui

- $q_0 = q_{\text{start}}$
- $\forall i \in [1, n-1] \quad w_i \in L(\delta(q_i, q_{i+1}))$, ovvero, w_i deve far parte del linguaggio rappresentato dall'espressione regolare sull'arco da q_i a q_{i+1}
- $q_n = q_{\text{accept}}$

Metodo 1.4.2.1: GNFA di un DFA

Sia M un DFA; allora per costruire un GNFA ad esso equivalente, è sufficiente:

- aggiungere un nuovo stato iniziale, con un ε-arco entrante sul vecchio stato iniziale;
- aggiungere un nuovo stato accettante, con ε-archi entranti provenienti dai vecchi stati accettanti;
- per gli archi con etichette multiple, sostituire questi ultimi con archi aventi come etichetta l'unione delle etichette precedenti;
- aggiungere archi etichettati con $\emptyset$ tra gli stati non collegati (si noti che questa operazione non varia l'automa di partenza, poiché un arco etichettato con $\emptyset$ non potrà mai essere utilizzato)

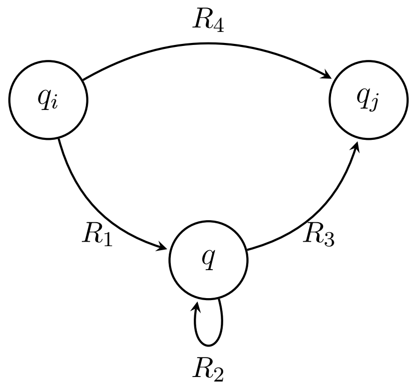
Algoritmo 1.4.2.1: Espressione regolare di un GNFA

Dato un GNFA $G = (Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, l'algoritmo restituisce un'espressione regolare equivalente a G.

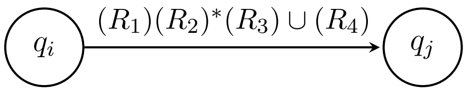
1: **function** CONVERTGNFATOREGEX(G)
2:     **if** $|Q| == 2$ **then**
3:         **return** $\delta(q_{\text{start}}, q_{\text{accept}})$
4:     **else if** $|Q| > 2$ **then**
5:         $q \in Q - \{q_{\text{start}}, q_{\text{accept}}\}$
6:         $Q' := Q - \{q\}$
7:         $\forall q_i \in Q' - \{q_{\text{accept}}\}, q_j \in Q' - \{q_{\text{start}}\} \quad \delta'(q_i, q_j) := \delta(q_i, q)\delta(q, q)^*\delta(q, q_j) \cup \delta(q_i, q_j)$
8:         $G' := (Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$
9:         **return** `convertGNFAtoRegEx`(G')
10:     **end if**
11: **end function**

Idea. L'algoritmo inizia prendendo in input un GNFA G, ed inizialmente viene controllato il numero di stati di G:

- se $|Q|$ è 2, allora sicuramente $Q = \{q_{\text{start}}, q_{\text{accept}}\}$, e poiché si vuole resituire l'espressione regolare equivalente a G, di fatto quest'ultimo è costituito esclusivamente dall'espressione regolare posta sull'arco tra q_{start} e q_{accept}, dunque è sufficiente restituirla in output (riga 3);
- se $|Q|$ è maggiore di 2, allora viene costruito un GNFA G', avente uno stato in meno, ovvero q (scelto alla riga 5), naturalmente diverso da q_{start} e da q_{accept}; successivamente, per ogni coppia di stati (q_i, q_j), viene definita $\delta'(q_i, q_j)$ in modo tale da accorpare tutte le possibili configurazioni di stati; ad esempio, prendendo in esame il seguente GNFA



dove R_1, R_2, R_3 ed R_4 sono espressioni regolari, è possibile vedere che il seguente GNFA è ad esso equivalente



poiché l'arco che q ha su sé stesso è stato descritto attraverso $(R_2)^*$, gli archi (q_i, q) e (q, q_j) sono stati inseriti per concatenazione, ed infine è stato unito l'altro possibile cammino verso q_j tramite unione; si noti che tale espressione regolare tiene in considerazione tutte le possibili configurazioni di archi tra stati di un GNFA;

- si noti che, sia per la Definizione 1.4.2.1, sia per come la riga 5 dell'algoritmo opera, si ha che $|Q| \geq 2$, dunque non è necessario gestire ulteriori casi.

Allora, l'algoritmo è in grado di restituire l'espressione regolare equivalente a G.

Dimostrazione. La dimostrazione procede per induzione su k, il numero di stati del GNFA.

- *caso base*: $k = 2$; se G ha solo 2 stati, allora necessariamente $Q = \{q_{\text{start}}, q_{\text{accept}}\}$, e dunque $\texttt{convertGNFAtoRegEx}(G) = \delta(q_{\text{start}}, q_{\text{accept}})$, alla riga 3.
- *ipotesi induttiva*: se G è un GNFA con $k - 1$ stati, allora $\texttt{convertGNFAtoRegEx}(G)$ è un'espressione regolare equivalente a G.
- *passo induttivo*: è necessario dimostrare che, se G è un GNFA con k stati, allora $\texttt{convertGNFAtoRegEx}(G)$ è un'espressione regolare equivalente a G. In primo luogo, è necessario dimostrare che G e G' sono equivalenti, e dunque sia w una stringa accettata da G; allora, in un ramo accettante della computazione, G entra in una sequenza di stati
$$q_{\text{start}}, q_1, q_2, \ldots, q_{\text{accept}}$$
dunque, se il ramo non contiene lo stato rimosso q, allora sicuramente G' accetta w; viceversa, se q è contenuto all'interno di tale ramo, allora per quanto discusso all'interno dell'idea di dimostrazione, l'espressione regolare inserita al posto dello stato q rimosso è in grado di tenere in considerazione ogni possibile configurazione di archi, e dunque G' accetta sicuramente w; infine, è possibile applicare la stessa osservazione per mostrare che una stringa accettata da G' deve essere necessariamente accettata anche da G. Allora, poiché G e G' sono equivalenti, e G ha k stati, allora necessariamente al termine del k-esimo passo dell'algoritmo, G' avrà $k - 1$ stati, e su di esso è possibile applicare l'ipotesi induttiva.

$\square$

Teorema 1.4.2.1: Linguaggi ed espressioni regolari

Un linguaggio è regolare se e solo se esiste un'espressione regolare che lo descrive.

Dimostrazione.

Prima implicazione. Sia A un linguaggio regolare; allora, per definizione, esiste un DFA che lo riconosce, e sia questo M. Utilizzando il Metodo 1.4.2.1, è possibile costruire un GNFA che sia equivalente ad M; sia quest'ultimo G. Allora, è sufficiente applicare l'Algoritmo 1.4.2.1 su G per ottenere l'espressione regolare ad esso equivalente; dunque, segue la tesi.

Seconda implicazione. Sia A un linguaggio su un alfabeto Σ, descritto da un'espressione regolare R. Allora, la dimostrazione procede costruendo degli NFA, per casi, come segue:

- $R = \emptyset \implies L(R) = \emptyset$; allora, il seguente NFA è in grado di riconoscere $L(R)$:

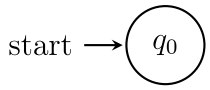
  

  Figura 1.9: Un NFA in grado di riconoscere $L(\emptyset)$.

  L'automa mostrato è descritto dalla quintupla $N = (\{q_0\}, \Sigma, \delta, q_0, \varnothing)$, dove $\forall a \in \Sigma \quad \delta(q_0, a) = \emptyset$.

- $R = \varepsilon \implies L(R) = \{\varepsilon\}$; allora, il seguente NFA è in grado di riconoscere $L(R)$:

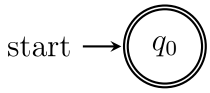
  

  Figura 1.10: Un NFA in grado di riconoscere $L(\varepsilon)$.

  L'automa mostrato è descritto dalla quintupla $N = (\{q_0\}, \Sigma, \delta, q_0, \{q_0\})$, dove $\forall a \in \Sigma \quad \delta(q_0, a) = \emptyset$.

- $R \in \Sigma \implies L(R) = \{a\}$ per qualche $a \in \Sigma$; allora, il seguente NFA è in grado di riconoscere $L(R)$:

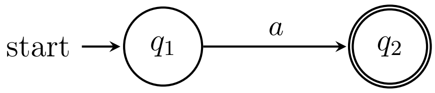
  

  Figura 1.11: Un NFA in grado di riconoscere $L(a)$.

  L'automa mostrato è descritto dalla quintupla $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, dove $\forall q \in \{q_1, q_2\}, x \in \Sigma \quad \delta(q, x) = \begin{cases} \{q_2\} & q = q_1 \land x = a \\ \emptyset & q \neq q_1 \lor x \neq a \end{cases}$.

- si noti che se esistono due espressioni regolari R_1 ed R_2 tali che $R = R_1 \cup R_2$, oppure $R = R_1 \circ R_2$, o ancora $R = R_1^*$, è sufficiente costruire gli NFA che sono stati costruiti nelle dimostrazioni della Proposizione 1.3.1.1, della Proposizione 1.3.1.2 e della Proposizione 1.3.1.3.

Allora, per qualsiasi espressione regolare R, tale da descrivere un certo linguaggio A, è possibile costruire un NFA che riconosce il linguaggio che R descrive. Allora, per il Corollario 1.2.1.1, A è regolare.

□

1.5 Linguaggi non regolari

1.5.1 Pumping lemma

Principio 1.5.1.1: Principio della piccionaia

Siano A e B due insiemi finiti, tali che $|B| < |A|$. Allora, non esiste alcuna funzione iniettiva $f : A \to B$.

In altri termini, avendo una piccionaia con m caselle, non è possibile inserire più di m piccioni al suo interno: due volatili dovranno necessariamente condividere la casella.

Lemma 1.5.1.1: Pumping lemma

Sia A un linguaggio regolare; allora, esiste un $p \in \mathbb{N}$, detta **lunghezza del pumping**, tale che per ogni stringa $s \in A$ tale per cui $|s| \geq p$, esistono 3 stringhe $x, y, z \in A \mid s = xyz$, soddisfacenti le seguenti condizioni:

- $\forall i \geq 0 \quad xy^iz \in A$
- $|y| > 0$ (o, equivalentemente, $y \neq \varepsilon$)
- $|xy| \leq p$

Dimostrazione. Poiché A è un linguaggio regolare in ipotesi, per definizione esiste un DFA $M = (Q, \Sigma, \delta, q_1, F)$ in grado di riconoscerlo. Allora, sia $p := |Q|$, e sia $s \in A \mid s = s_1 s_2 \cdots s_n$ tale che $\forall i \in [1, n] \quad s_i \in \Sigma$, e $n \geq p$. Inoltre, sia

$$\forall i \in [1, n] \quad r_{i+1} := \delta(r_i, s_i)$$

la sequenza di stati attraversati da M mentre elabora s; dunque, si ha che $r_{n+1} \in F$. Si noti che la sequenza di stati ha dunque cardinalità

$$|\{r_1, \ldots, r_{n+1}\}| = n + 1$$

in quanto devono essere attraversati n archi, e dunque $n + 1$ stati. Inoltre, $n \geq p \iff n + 1 \geq p + 1$, e dunque per il Principio 1.5.1.1, due dei primi $p + 1$ stati della sequenza devono necessariamente essere lo stesso stato; siano r_j il primo ed r_l il secondo, con $j \neq l$. Allora, sicuramente $l \leq p + 1$, poiché r_l è uno stato tra i primi $p + 1$.

Si pongano dunque

$$\begin{cases} x := s_1 \cdots s_{j-1} \\ y := s_j \cdots s_{l-1} \\ z := s_l \cdots s_n \end{cases}$$

Allora, si ha che:

- x porta M da r_1 ad r_j, y porta M da r_j ad r_j stesso, ed infine z porta M da r_j ad r_{n+1}, e poiché $r_{n+1} \in F$, M accetta sicuramente xy^iz per ogni $i \geq 0$;
- $j \neq l \implies |y| > 0$;
- $l \leq p+1 \implies |xy| \leq p$, poiché r_l è lo stato che viene dopo aver letto $s_{l-1} \in y$, e dunque nel caso limite si ha che $l = p+1 \implies |xy| = p$.

Allora, sono soddisfatte tutte le condizioni della tesi.

La seguente rappresentazione raffigura un automa definito come segue:

$$M = (\{q_1, \ldots, q_9, \ldots, q_{13}\}, \Sigma, \delta, q_1, \{q_{13}\})$$

dove $|Q| = 13$, e $r_j = r_l = q_9$ è lo stato che si ripete all'interno dei primi $p+1$ stati della sequenza presi in esame nella dimostrazione.

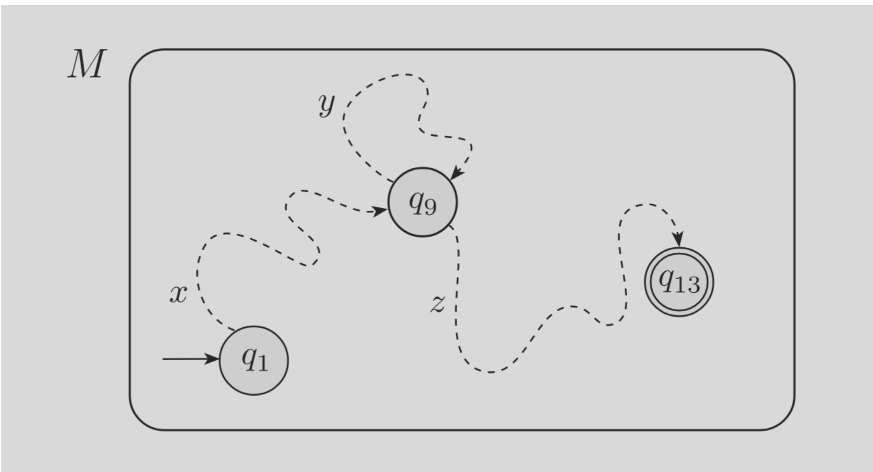


Figura 1.12: L'automa M descritto.

□

Esempio 1.5.1.1 (Pumping lemma). Sia $B = \{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$ un linguaggio. TODO NON HO CAPITO

2

Linguaggi e grammatiche context-free

2.1 Grammatiche context-free

2.1.1 Definizioni

Definizione 2.1.1.1: Grammatiche context-free

Una **grammatica context-free**, o CFG (*Context-Free Grammar*) è una quadrupla (V, Σ, R, S), dove

- V è l'insieme delle **variabili**, un insieme *finito*
- Σ è l'insieme dei **terminali**, un insieme *finito*, dove $\Sigma \cap V = \varnothing$
- R è l'insieme delle **regole** o **produzioni**, un insieme *finito*
- $S \in V$ è la **variabile iniziale**, ed è generalmente il primo simbolo della prima regola della grammatica

Le CFG sono della forma

$$\begin{gathered} X \to Y \\ \vdots \end{gathered}$$

dove $X, Y \in V$, e $X \to Y \in R$ è una regola della CFG.

Una grammatica è un insieme di regole di sostituzione di stringhe, in grado di produrre quest'ultime a partire da una variabile iniziale, mediante una sequenza di scambi. In particolare, partendo dalla variabile iniziale, vengono sostituiti gli elementi alla sinistra del simbolo $\to$ di una certa regola in R, con quelli alla sua destra.

Due regole $X \to Y, X \to Z \in R$ possono essere accorpate con $X \to Y|Z$. Siano u, v, w stringhe di variabili in V, e $A \to w \in R$; si dice che uAv **produce** uwv, denotato con $uAv \implies uwv$. Se $u = v$, oppure esistono stringhe $u_1, \ldots, u_k$ con $k \geq 0$ tali che

$$u \implies u_1 \implies \ldots \implies u_k \implies v$$

si dice che u **deriva** v, denotato con $u \overset{*}{\Longrightarrow} v$.

Esempio 2.1.1.1 (CFG)**.** Un esempio di CFG è il seguente:

$$A \to \mathtt{0}A\mathtt{1}$$
$$A \to B$$
$$B \to \mathtt{\#}$$

In essa, si hanno:

$$V := \{A, B\}$$
$$\Sigma := \{\mathtt{0}, \mathtt{1}, \mathtt{\#}\}$$
$$S := A \in V$$

Da essa, è possibile ottenere la stringa `000#111` attraverso le seguenti sostituzioni:

$$A \implies \mathtt{0}A\mathtt{1} \implies \mathtt{000}A\mathtt{111} \implies \mathtt{000}B\mathtt{111} \implies \mathtt{000\#111}$$

Definizione 2.1.1.2: Linguaggio di una grammatica

Data una grammatica G, il **linguaggio di** G è l'insieme delle stringhe che la grammatica G è in grado di generare, ed è denotato con $L(G)$. In simboli, data una grammatica G, si ha che

$$L(G) = \{w \in \Sigma^* \mid S \overset{*}{\Longrightarrow} w\}$$

Se G è una CFG, allora $L(G)$ è detto **linguaggio context-free**, o CFL (*Context-Free Language*).

Esempio 2.1.1.2 (CFL)**.** Si prenda in esame la CFG dell'Esempio 2.1.1.1, e sia essa G. Allora, in tale grammatica il linguaggio risulta essere

$$L(G) = \{\mathtt{0}^n\mathtt{\#}\mathtt{1}^n \mid n \geq 0\}$$

Definizione 2.1.1.3: Derivazione a sinistra

Sia G una grammatica; una stringa si dice essere **derivata a sinistra** se è stata ottenuta applicando regole di G sulle variabili più a sinistra disponibili.

Esempio 2.1.1.3 (Derivazione a sinistra)**.** TODO

Definizione 2.1.1.4: Ambiguità

Sia G una grammatica; se esistono due stringhe u, v con $u \neq v$, tali che esiste una terza stringa z derivata a sinistra sia da u che da v, si dice che G genera z **ambiguamente**; inoltre, G si dice essere **ambigua** se genera stringhe ambiguamente.

Definizione 2.1.1.5: Linguaggi inerentemente ambigui

Un linguaggio si dice essere **inerentemente ambiguo** se non esistono grammatiche non ambigue che lo possano generare.

Esempio 2.1.1.4 (Linguaggi inerentemente ambigui)**.** TODO $\{\mathtt{a}^i\mathtt{b}^j\mathtt{c}^k \mid i = j \lor j = k\}$

Definizione 2.1.1.6: Forma normale di Chomsky

Una CFG (V, Σ, R, S) è in **forma normale di Chomsky**, o CNF (*Chomsky Normal Form*), se ogni regola è della forma

$$A \to BC$$
$$A \to a$$

dove

$$V := \{A, B, C\}$$
$$\Sigma := \{a\}$$
$$S := A$$

e la regola $S \to \varepsilon \in R$ è sempre permessa, chiamata **ε-regola**.

Dunque, la CNF non permette di

i) avere **regole unitarie**, ovvero regole della forma $A \to B$;

ii) avere regole della forma $X \to S \in R$ per qualche $X \in V$, ovvero avere S alla destra del simbolo $\to$ di una regola.

Teorema 2.1.1.1: CFL generati da CFG in CNF

Ogni CFL è generato da una CFG in forma normale di Chomsky.

Dimostrazione. Sia un CFL generato da una CFG $G = (V, \Sigma, R, S)$. Allora, è possibile rendere G in CNF attraverso i seguenti passaggi:

- aggiungere $S_0 \in V$, e $S_0 \to S \in R$, in modo da non avere$S$ alla destra di nessuna regola in R;
- data una variabile $A \in V - \{S\}$ tale che esiste una regola $A \to \varepsilon \in R$, tale regola viene rimossa da R, e per ogni regola della forma $X \to uAv \in R$ per qualche $X \in V$ e u, v stringhe di variabili e terminali, viene aggiunta $X \to uv \in R$; se $X \to A \in R$, allora viene aggiunta $X \to \varepsilon \in R$, solo se quest'ultima non era stata precedentemente rimossa
- per ogni regola unitaria $A \to B \in R$, quest'ultima viene rimossa, ed ogni regola $B \to u \in R$ per qualche stringa u, viene aggiunta $A \to u \in R$, solo se questa non era una regola unitaria precedentemente rimossa;

- infine, ogni regola restante $A \to u_1 \dots u_k$ con $k \geq 3$ (e $u_1, \dots, u_k$ variabili o terminali) viene rimpiazzata con

$$A \to u_1 A_1, \dots A_1 \to u_2 A_2 \to A_2 \to u_3 A_3, \dots, A_{k-2} \to u_{k-1} u_k$$

e $A_1, \dots, A_{k-2} \in V$; ogni terminale TODO

$\square$

2.2 Automi a pila

2.2.1 Definizioni

Definizione 2.2.1.1: PDA

Un **PDA** (*Pushdown Automaton*) è una sestupla $(Q, \Sigma, \Gamma, \delta, q_0, F)$, dove

- Q è l'**insieme degli stati**, un insieme *finito*
- Σ è l'**alfabeto** dell'automa, un insieme *finito*
- Γ è l'**alfabeto dello stack** (o *pila*)
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ è la **funzione di transizione**, che definisce la relazione tra gli stati
- $q_0 \in Q$ è lo **stato iniziale**
- $F \subseteq Q$ è l'**insieme degli stati accettanti**

dove $\Gamma_\varepsilon := \Gamma \cup \{\varepsilon\}$.

Un PDA è un NFA dotato di uno **stack** illimitato, che gli consente di riconoscere alcuni linguaggi non regolari, poiché in esso è in grado di porre i simboli che legge dalla stringa di input, di fatto implementando un sistema di *memoria*.

Si noti che la macchina può usare differenti alfabeti per il suo input e la sua pila, infatti la definizione formale vede due alfabeti distinti, Σ e Γ rispettivamente. Inoltre, Γ_ε compare nel prodotto cartesiano del dominio di δ, poiché il simbolo in cima allo stack del PDA è in grado di determinare anch'esso la mossa seguente dell'automa; a tal proposito, $\varepsilon \in \Gamma$ permette di ignorare il primo elemento della pila. In aggiunta, Γ_ε compare anche all'interno dell'insieme potenza (si noti che un PDA è non deterministico) del codominio di δ, al fine di poter decidere se si vuole salvare il simbolo letto all'interno dello stack o meno (tramite $\varepsilon \in \Gamma$).

Esempio 2.2.1.1 (PDA)**.** Un esempio di PDA $(Q, \Sigma, \Gamma, \delta, q_1, F)$ è il seguente:

- $Q = \{q_1, q_2, q_3, q_4\}$
- $\Sigma = \{\mathtt{0}, \mathtt{1}\}$
- $\Gamma = \{\mathtt{0}, \mathtt{\$}\}$

- $F = \{q_1, q_4\}$

e δ è data dalla seguente tabella di transizione:

Input:	0			1			ε		
Stack:	0	\$	ε	0	\$	ε	0	\$	ε
q_1									$\{(q_2, \$)\}$
q_2			$\{(q_2, 0)\}$	$\{(q_3, \varepsilon)\}$					
q_3				$\{(q_3, \varepsilon)\}$				$\{(q_4, \varepsilon)\}$	
q_4									

Figura 2.1: L'immagine rappresenta δ in forma tabellare.

Definizione 2.2.1.2: Stringhe accettate (PDA)

Sia $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ un PDA, e sia $w = w_1 \cdots w_n$ una stringa tale per cui $\forall i \in [1, n] \quad w_i \in \Sigma_\varepsilon$; allora, M **accetta** w se esistono una sequenza di stati $r_0, \ldots, r_n \in Q$ e una sequenza di stringhe $s_0, \ldots, s_n \in \Gamma^*$ tali per cui

- $r_0 = q_0$
- $s_0 = \varepsilon$, ovvero, lo stack è inizialmente vuoto
- $\forall i \in [1, n-1] \quad \exists a, b \in \Gamma_\varepsilon, t \in \Gamma^* \mid (r_{i+1}, b) \in \delta(r_i, w_{i+1}, a) \land \begin{cases} s_i = at \\ s_{i+1} = bt \end{cases}$, ovvero, M si muove correttamente in base allo stato, al simbolo nelo stack, ed al prossimo simbolo di input; si noti che le stringhe $s_0, \ldots, s_n$ rappresentano la sequenza del contenuto dello stack che M ha su un ramo accettante della computazione, infatti $s_i = at$ diventa $s_{i+1} = bt$ con l'iterazione successiva, dunque a è stato sostituito con b in cima alla pila
- $r_n \in F$

Esempio 2.2.1.2 (Linguaggi riconosciuti da un PDA)**.** Si prenda in considerazione il PDA descritto nell'Esempio 2.2.1.1, e sia questo M. Il seguente è il suo diagramma di stato:

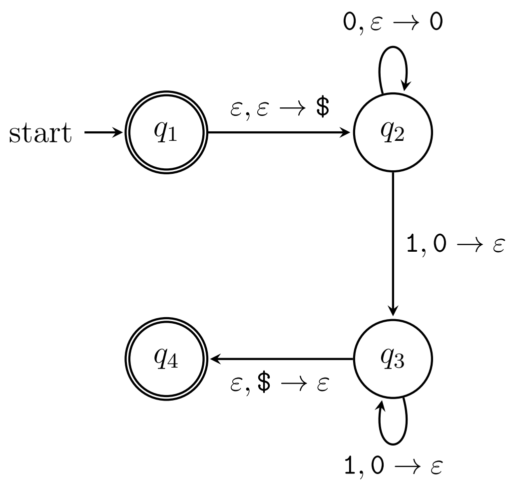


Figura 2.2: Il PDA M.

Teorema 2.2.1.1: CFL e PDA

Un linguaggio è context-free se e solo se esiste un PDA che lo riconosce.

Dimostrazione.

Prima implicazione. TODO

Seconda implicazione. TODO

$\square$