Automi

Autore
Alessandro Dori

Istituzione
Università "La Sapienza"

Data
12/10/2024

(email: dorialessandro99@gmail.com)

INDICE

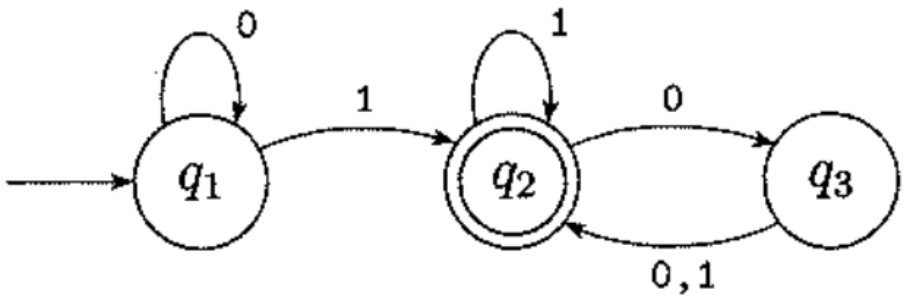
Linguaggi regolari

Automi finiti

Definizione formale di automa finito.

Un **automa finito** è una quintupla $(Q, \Sigma, \delta, q_0, F)$, dove:

- Q è un insieme finito chiamato l'insieme degli **stati**;
- Σ è l'insieme finito chiamato l'**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 accettanti**.



Possiamo descrivere M_1 formalmente ponendo $M_1 = (Q, \Sigma, \delta, q_0, F)$, dove
1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. δ è descritta come segue

	0	1
q1	q1	q2
q2	q3	q2
q3	q2	q2

5. 4. q_1 è lo stato iniziale
6. $F = \{q_2\}$.

> Se A è un insieme di tutte le stringhe che la macchina M accetta, diciamo che A è il linguaggio della macchina M e scriviamo $L(M) = A$.
> M riconosce A.

Nel nostro esempio, sia

$$A = \{w | w \text{ contiene almeno un 1 e un numero pari di 0 segue l'ultimo 1}\}.$$

Definizione formale di computazione

Sia $(Q, \Sigma, \delta, q_0, F)$ un automa finito e sia $w = w_1, w_2, \dots, w_n$ una stringa dove ogni w_i è un elemento dell'alfabeto Σ. Allora M accetta w se $\exists$ una sequenza di stati $r_0, r_1, \dots, r_n$ in Q con 3 condizioni:

1. $r_0 = q_0$ (la macchina inizia nello stato iniziale).
2. $\delta(r_i, w_{i+1}) = r_{i+1}, \text{ per } i = 0, \dots n-1$ afferma che la macchina passa da stato a stato in base alla funzione di transizione.
3. $r_n \in F$ La macchina accetta il suo input se termina la lettura in uno stato accettante.
   Diciamo che M **riconosce il linguaggio** A se

$$A = \{w | M \text{ accetta } w\}.$$

> Un linguaggio è chiamato un linguaggio regolare se un automa finito lo riconosce.

Le operazioni regolari

Definiamo tre sui linguaggi, chiamate operazioni regolari, e le usiamo per studiare le proprietà dei linguaggi regolari.

Definizione 1.23

Siano A e B linguaggi. Definiamo le operazioni regolari **unione, concatenazione e star** come segue:

- Unione: $A \cup B = \{x | x \in A \text{ o } x \in B\}$.
- Concatenazione: $A \circ B = \{xy | x \in A \text{ e } y \in B\}$.
- Star: $A^* = \{x_1, x_2, \dots x_k | k \geq 0 \text{ e ogni } x_i \in A\}$
  Unione: prende tutte le stringhe sia in A che in B e le raggruppa insieme in un linguaggio
  Concatenzione: antepone una stringa di A ad una stringa di B in tutti i modi possibili per ottenere le stringhe nel nuovo linguaggio.

L'operazione di star è un'operzione unaria.
Essa opera concatenando un numero qualsiasi di stringhe in a insieme per ottenere una stringa nel nuovo linguaggio.

> **Important**
>
> La stringa vuota ε è sempre un elemento di A^*, indipendentemente da chi sia A.

Una classe di oggetti è chiusa rispetto ad un'operazione se l'applicazione di questa operazione a elementi della classe restituisce un oggetto ancora nella classe.

Teorema 1.25

La classe dei linguaggi regolari è chiusa rispetto all'operazione di unione.
In alter parole se A_1 e A_2 sono linguaggi regolari, lo è anche $(A_1 \cup A_2)$.

- M_1 riconosce A_1.
- M_2 riconosce A_2.
- M riconosce $(A_1 \cup A_2)$.

> Gli stati accettanti in M sono coppie t.c. $M_1 \circ M_2$ è in uno stato accettante.

Teorema 1.26

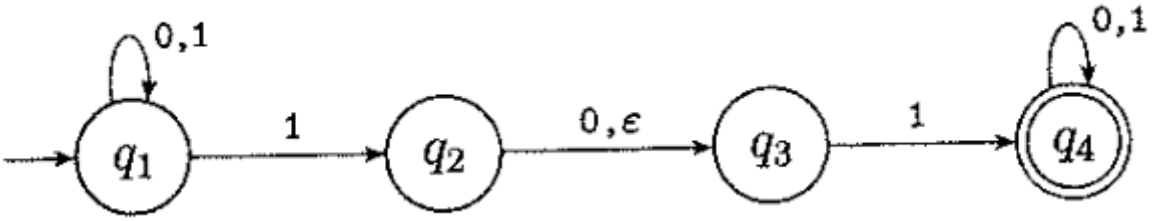
La classe dei linguaggi regolari è chiusa rispetto all'operazione di concatenazione.
Se A_1 e A_2 sono linguaggi regolari, allora lo è anche $(M_1 \circ M_2)$.

- M_1 riconosce A_1.
- M_2 riconosce A_2.
- M deve riconoscere $M_1 \circ M_2$, ma M non sa dove dividere il suo input.
  Per risolvere questo problema introduciamo una nuova tecnica chiamata non determinismo.

Non Determinismo.

In una macchina non deterministica, possono esistere diverse scelte per lo stato successivo in ogni punto.
Il non determinismo è una generalizzazione del determinismo, quindi ogni automa finito deterministico è autonomamente anche non deterministico.

esempio:



Note

DFA = Automa finito deterministico
NFA = Automa finito non deterministico

Ogni stato di un **DFA** ha sempre esattamente un arco di transizione uscente per ogni simbolo dell'alfabeto.
In un **NFA** uno stato può avere zero, uno, o più archi uscenti per ogni simbolo dell'alfabeto.
In un DFA le etichette sugli archi appartengono all'alfabeto, in un NFA troviamo anche ε; zero, uno o più archi possono uscire da ciascuno stato con l'etichetta ε.
In un NFA nel caso in cui abbiamo due strade diverse con lo stesso simbolo, la "computazione" si divide.
Se una di queste strade termina in uno stato non accettante, allora essa "muore", se una qualsiasi di queste strade (copie) accetta, allora l'NFA accetta la stringa di input.
Se incontriamo ε senza leggere alcun input, la macchina si divide in più copie multiple.

Important

Ogni NDA può essere trasformato in un DFAequivalente e costruire un NFA a volte è più semplice che costruire direttamente un DFA.

ESEMPIO 1.30

Sia A il linguaggio che consiste di tutte le stringhe su $\{0,1\}$ contenenti un 1 nella terza posizione dalla fine (ad esempio, 000100 è in A ma 0011 non lo è). Il seguente NFA N_2, con quattro stati, riconosce A.

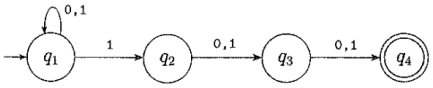
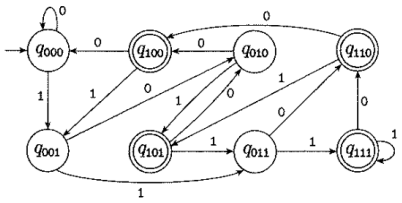


FIGURA 1.31
L'NFA N_2 che riconosce A

Un buon modo di vedere la computazione di questo NFA è dire che esso resta nello stato iniziale q_1 finché "ipotizza" che è a tre posizioni dalla fine. A questo punto, se il simbolo di input è un 1, passa nello stato q_2 e usa q_3 e q_4 per "controllare" se la sua ipotesi era corretta.

Come menzionato, ogni NFA può essere trasformato in un DFA equivalente; ma a volte questo DFA può avere molti più stati. Il più piccolo DFA per A contiene otto stati. Inoltre, capire il funzionamento dell'NFA è molto più facile, come si può vedere esaminando la figura seguente del DFA.



ESEMPIO 1.33

Il seguente NFA N_3 ha un alfabeto $\{0\}$ di simboli input che consiste di un solo simbolo. Un alfabeto contenente solo un simbolo è chiamato un ***alfabeto unario***.

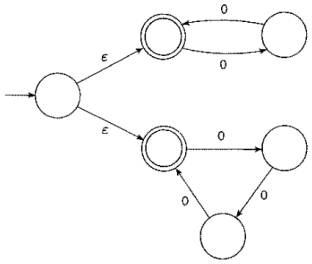


FIGURA 1.34
L'NFA N_3

Questa macchina mostra l'utilità di avere ε archi. Essa accetta tutte le stringhe della forma 0^k dove k è un multiplo di 2 o 3. (Ricorda che l'esponente denota ripetizione, non elevazione a potenza numerica.) Per esempio, N_3 accetta le stringhe ε, 00, 000, 0000, e 000000, ma non 0 o 00000.

Puoi vedere che la macchina opera inizialmente provando a indovinare se testare per un multiplo di 2 o un multiplo di 3, andando nel ciclo superiore o nel ciclo inferiore, e poi verificando se la propria ipotesi era corretta. Naturalmente, potremmo sostituire questa macchina con una che non ha ε-archi o perfino del tutto priva di non determinismo, ma la macchina mostrata è quella più facile da capire per questo linguaggio.

Definizione formale di automa finito non deterministico

Per un qualsiasi insieme Q denotiamo con $P(Q)$ la collezione di tutti i sottoinsiemi di Q.

- $P(Q)$ = insieme potenza di Q.
- Per ogni alfabeto Σ scriviamo Σ_ε per denotare $\Sigma \cup \{\varepsilon\}$.
  Ora possiamo descrivere formalmente il tipo di funzione di transizione in un NFA come:

$$\delta : Q \times \Sigma_\varepsilon \to P(Q)$$

Definizione 1.37

Un automa finito non deterministico è una quintupla $(Q, \Sigma, \delta, q_0, F)$, dove:

1. Q è un insieme finito di stati.
2. Σ è un alfabeto finito.
3. $\delta : Q \times \Sigma_\varepsilon \to P(Q)$ è la funzione di transizione.
4. $q_0 \in Q$ è lo stato iniziale.
5. $F \subseteq Q$ è l'insieme degli stati di accettazione.

Equivalenza tra gli NFA e i DFA

Teorema 1.39(orale)

1.39

Per ogni automa finito non deterministico esiste un automa finito deterministico equivalente.

IDEA.
Se un linguaggio è riconosciuto da un NFA, allora dobbiamo mostrare l'esistenza di un DFA che lo riconosce. L'idea è di trasformare l'NFA in un DFA equivalente che simula l'NFA.
Se k è il numero degli stati dell'NFA, esso ha 2^k sottoinsiemi di stati. Ogni sottoinsieme corrisponde a una delle possibilità che il DFA deve ricordare, quindi il DFA che simula l'NFA avrà 2^k stati. Ora noi dobbiamo quali saranno lo stato iniziale e gli stati accettanti del DFA, e quale sarà la sua funzione di transizione.

Dimostrazione.

Sia $N = (Q, \Sigma, \delta, q_0, F)$ l'NFA che riconosce un linguaggio A. Costruiamo un DFA $M = (Q', \Sigma, \delta', q_0', F')$ che riconosce A. Prima di tutto consideriamo il caso più semplice in cui N non ha ε-archi. In seguito considereremo gli ε-archi.

1. $Q' = P(Q)$.

- Ogni stato di M è un insieme di stati di N. Ricorda che $P(Q)$ è l'insieme dei sottoinsiemi di Q.

2. Per $R \in Q'$ e $a \in \Sigma$, sia $\delta'(R, a) = \{q \in Q | q \in \delta(r, a) \text{ per qualche } r \in R\}$.

- Se R è uno stato di M, esso è anche un insieme di stati di N. Quando M legge un simbolo a nello stato R, mostra dove a porta ogni stato in R. Poichè da ogni stato si può andare in un insieme di stati, prendiamo l'unione di tutti questi insiemi. Un

altro modo per scrivere questa espressione è

$$\delta'(R,a) = \bigcup_{r \in R} \delta(r,a).$$

ovvero l'unione di tutti gli insiemi $\delta(r,a)$ per ogni possibile $r \in R$.

3. $q_0' = \{q_0\}$.

- M inizia nello stato corrispondente alla collezione che contiene solo lo stato iniziale di N.

4. $F' = \{R \in Q' | R \text{ contiene uno stato accettante di } N\}$.

- La macchina M accetta se uno dei possibili stati in cui N potrebbe essere a quel punto è uno stato accettante.

Ora dobbiamo considerare gli ε-archi. Introduciamo qualche ulteriore notazione. Per ogni stato R di M, definiamo $E(R)$ come la collezione di stati che possono essere raggiunti dagli elementi di R proseguendo solo con ε-archi, includendo gli stessi elementi di R. Formalmente, per $R \subseteq Q$ sia

$$E(R) = \{q | q \text{ può essere raggiunto da } R \text{ attraverso 0 o più } \varepsilon\text{-archi}\}.$$

Poi modifichiamo la funzione di transizione di M ponendo dita supplementari su tutti gli stati che possono essere raggiunti proseguendo attraverso ε-archi dopo ogni passo. Sostituendo $\delta(r,a)$ con $E(\delta(r,a))$ realizziamo questo effetto. Quindi

$$\delta'(R,a) = \{q \in Q | q \in E(\delta(r,a)) \text{ per qualche } r \in R\}.$$

Inoltre, dobbiamo modificare lo stato iniziale di M per muovere inizialmente la dita su tutti i possibili stati che possono essere raggiunti dallo stato iniziale di N attraverso gli ε -archi. Cambiare q_0' in $E(\{q_0\})$ realizza questo risultato. Ora abbiamo completato la costruzione del DFA M che simula l'NFA N.
Ovviamente la costruzione di M funziona correttamente. A ogni passo nella computazione di M su un input, chiaramente entra in uno stato che corrisponde al sottoinsieme di stati in cui N potrebbe essere a quel punto. Quindi la nostra prova è completa.

Corollario 1.40

Corollario 1.40

Un linguaggio è regolare se e solo se qualche automa finito non deterministico lo riconosce.

ESEMPIO 1.41

Illustriamo la procedura che abbiamo dato nella prova del Teorema 1.39 per trasformare un NFA in un DFA usando la macchina N_4 che appare (presente) nell'Esempio 1.35. Per chiarezza, abbiamo rinominato gli stati di N_4 in $\{1,2,3\}$. Quindi nella descrizione formale di $N_4 = (Q, \{\mathtt{a},\mathtt{b}\}, \delta, 1, \{1\})$, l'insieme degli stati Q è $\{1,2,3\}$ come mostrato nella Figura 1.42.

Per costruire un DFA D equivalente a N_4, innanzitutto determiniamo gli stati di D. N_4 ha tre stati, $\{1,2,3\}$, quindi costruiamo D con otto stati, uno per ogni sottoinsieme dell'insieme degli stati di N_4. Etichettiamo ognuno degli stati di D con il corrispondente sottoinsieme. Quindi l'insieme degli stati di D è

$$\{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$

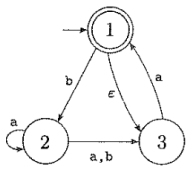


FIGURA 1.42
L'NFA N_4

Poi, determiniamo lo stato iniziale e gli stati accettanti di D. Lo stato iniziale è $E(\{1\})$, l'insieme degli stati che sono raggiungibili da 1 passando attraverso ε-archi, più 1 stesso. Un ε-arco va da 1 a 3, quindi $E(\{1\}) = \{1,3\}$. I nuovi stati accettanti sono quelli che contengono lo stato accettante di N_4; quindi $\{\{1\},\{1,2\},\{1,3\},\{1,2,3\}\}$.

Infine, determiniamo la funzione di transizione di D. Ciascuno degli stati di D va in una posizione sull'input a e in una posizione sull'input b. Illustriamo il processo di determinare la posizione degli archi di transizione di D con pochi esempi.

In D, lo stato $\{2\}$ va in $\{2,3\}$ sull'input a perché in N_4, lo stato 2 va sia in 2 che in 3 sull'input a e non possiamo andare più lontano da 2 o 3 attraverso ε-archi. Lo stato $\{2\}$ va nello stato $\{3\}$ sull'input b perché in N_4, lo stato 2 va solo nello stato 3 sull'input b e da 3 non possiamo andare più lontano attraverso ε archi.

Lo stato $\{1\}$ va in $\emptyset$ su a perché nessun arco etichettato con a esce da esso. Va in $\{2\}$ su b. Nota che la procedura nel Teorema 1.39 specifica che seguiamo gli ε archi *dopo* ogni simbolo di input che viene letto. Una procedura alternativa basata sul seguire gli ε archi prima di leggere ogni simbolo funziona ugualmente bene, ma questo metodo non è illustrato in questo esempio.

Lo stato $\{3\}$ va in $\{1,3\}$ su a perché in N_4, lo stato 3 va in 1 su a e 1 a sua volta va in 3 con un ε arco. Lo stato $\{3\}$ su b va in $\emptyset$.

Lo stato $\{1,2\}$ su a va in $\{2,3\}$ perché 1 non punta ad alcuno stato con a archi, 2 punta sia a 2 che a 3 con a archi, e nessuno dei due punta altrove con ε archi. Lo stato $\{1,2\}$ su b va in $\{2,3\}$. Continuando in questo modo, otteniamo il diagramma per D nella Figura 1.43.

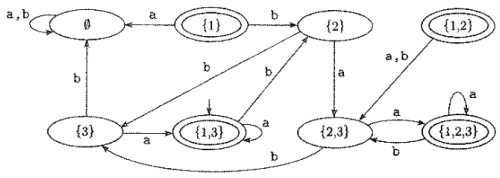


FIGURA 1.43
Un DFA D che è equivalente all'NFA N_4

Possiamo semplificare questa macchina osservando che nessun arco punta agli stati $\{1\}$ e $\{1,2\}$, quindi essi possono essere tolti senza ripercussioni sulla prestazione della macchina. Facendo così si ottiene la figura seguente.

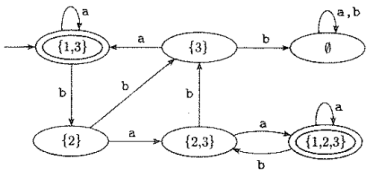


FIGURA 1.44
Il DFA D dopo aver tolto gli stati non necessari

Chiusura rispetto alle operazioni regolari

Il nostro scopo è provare che l'unione, la concatenazione e lo star di linguaggi regolari sono ancora regolari. L'uso del non determinismo rende semplici le prove molto più semplici. Innanzitutto consideriamo di nuovo la chiusura rispetto all'unione.

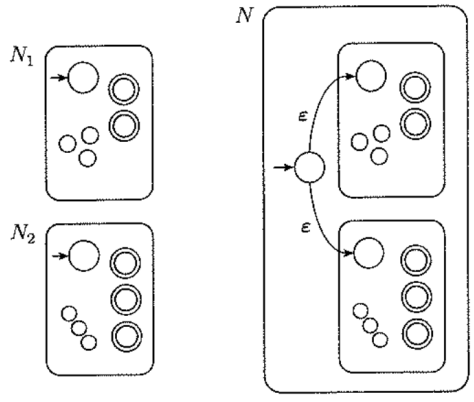
Teorema 1.45(orale)

1.45

La classe dei linguaggi regolari è chiusa rispetto all'operazione di unione.

IDEA.
Abbiamo i linguaggi regolari A_1 e A_2 e vogliamo provare che $A_1 \cup A_2$ è regolare.
L'idea è prendere due NFA, N_1 ed N_2 per A_1 e A_2 e comporli in un nuovo NFA, N.
La macchina N deve accettare il suo inputo se $N_1 \circ N_2$ accetta questo input. La nuova macchina ho un nuovo stato iniziale che si dirama negli stati iniziali delle vecchie macchine con ε archi.
In questo modo se una delle due macchine accetta l'input, anche N lo accetterà.

Rappresentiamo questa costruzione nella figura seguente. Sulla sinistra, indichiamo lo stato iniziale e gli stati accettanti delle macchine N_1 ed N_2 con cerchi grandi e alcuni ulteriori stati con cerchi piccoli. Sulla destra, mostriamo come comporre N_1 ed N_2 in N aggiungendo archi di transizione supplementari.



Dimostrazione

Sia $N = (Q_1, \Sigma, \delta_1, q_1, F_1)$ che riconosce$ A_1 ed
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ che riconosce A_2 .
Costruiamo $N = (Q, \Sigma, \delta, q_0, F)$ per riconoscere $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
   - Gli stati di N sono tutti gli stati di N_1 ed N_2, con l'aggiunta di un nuovo stato iniziale q_0.
2. Lo stato q_0 è lo stato iniziale di N.
3. L'insieme degli stati accettanti $F = F_1 \cup F_2$.
   - Gli stati accettanti di N sono tutti gli stati accettanti di N_1 ed N_2. In questo modo, N accetta se N_1 accetta o N_2 accetta.
4. Definiamo δ in modo che per ogni $q \in Q$ e per ogni $a \in \Sigma_\varepsilon$,

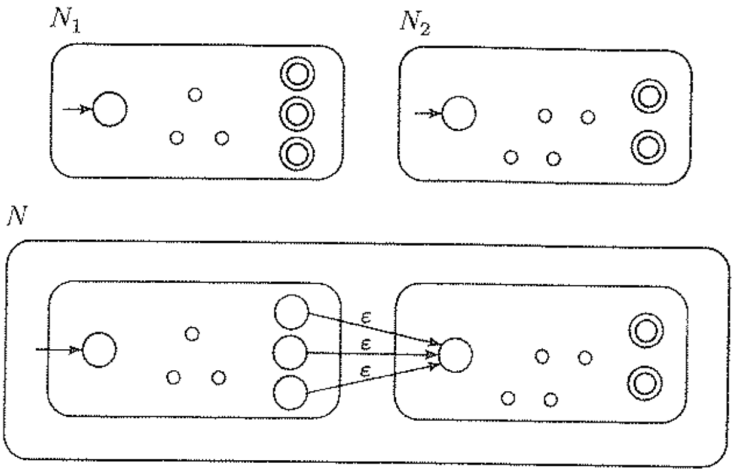
$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \\ \delta_2(q,a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ e } a = \varepsilon \\ 0 & q = q_0 \text{ e } a \neq \varepsilon. \end{cases}$$

Teorema 1.47(orale)

1.47

La classe dei linguaggi regolari è chiusa rispetto all'operazione di concatenazione.

IDEA.
Abbiamo i linguaggi regolari A_1 e A_2 e vogliamo provare che $A_1 \circ A_2$ è regolare. L'idea è sempre quella di prendere due NFA e combinarli in uno solo, ma questa volta in modo diverso. Poniamo come stato iniziale di N lo stato iniziale di N_1. Gli stati accettanti di N_1 hanno ulteriori ε-archi che non deterministicamente permettono di diramarsi in N_2 ogni volta che N_1 è in uno stato accettante, indicando che ha trovato un pezzo iniziale dell'input che costituisce una stringa in A_1. Gli stati accettanti di N sono solo gli stati accettanti di N_2. Quindi, esso accetta quando l'input può essere diviso in due parti, la prima accettata da N_1 e la seconda da N_2. !



Dimostrazione

Sia $N = (Q_1, \Sigma, \delta_1, q_1, F_1)$ che riconosce$ A_1 ed
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ che riconosce A_2 .
Costruiamo $N = (Q, \Sigma, \delta, q_1, F_2), F)$ per riconoscere $A_1 \circ A_2$.

1. $Q = Q_1 \cup Q_2$.
    - Gli stati di N sono tutti gli stati di N_1 e N_2.
2. Lo stato q_1 è uguale allo stato iniziale di N_1.

3. Gli stati accettanti F_2 sono uguali agli stati accettanti di N_2.
4. Definiamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\varepsilon$,

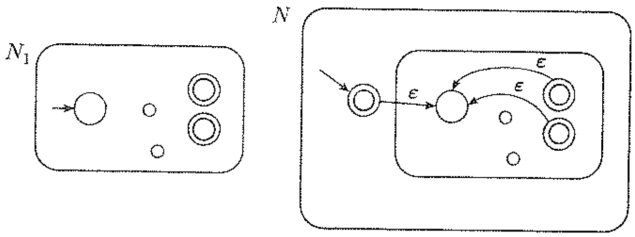
$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ e } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ e } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & q \in F_1 \text{ e } a = \varepsilon \\ \delta_2(q,a) & q \in Q_2. \end{cases}$$

Teorema 1.49(orale)

1.49

La classe dei linguaggi regolari è chiusa rispetto all'operazione star.

IDEA.
Abbiamo un linguaggio regolare A_1 e vogliamo provare che anche A_1^* è regolare. Prendiamo un NFA N_1 per A_1 e lo modifichiamo per riconoscere A_1^*. L'NFA N risultante accetterà il suo input quando esso può essere diviso in varie parti ed N_1 accetta ogni parte.
Possiamo costruire N come N_1 con ε-archi supplementari che dagli stati accettanti ritornano allo stato iniziale. In questo modo, quando l'elaborazione giunge alla fine di una parte che N_1 accetta, la macchina N ha a scelta di tornare indietro allo stato iniziale per provare a leggere un'altra perte che N_1 accetta. Inoltre, dobbiamo modificare N in modo che accetti ε, che è sempre un elemento di A_1^*. L'idea è di aggiungere un nuovo stato iniziale, che è anche uno stato accettante, e che ha un ε-arco entrante nel vecchio stato iniziale.



Dimostrazione

Sia $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ che riconosce A_1.
Costruiamo $N = (Q, \Sigma, \delta, q_0, F)$ per riconoscere A_1^*.

1. $Q = \{q_0\} \cup Q_1$.
   - Gli stati di N sono gli stati di N_1 più un nuovo stato iniziale.
2. Lo stato q_0 è il nuovo stato iniziale.

3. $F = \{q_0\} \cup F1$.
    - Gli stati accettanti sono i vecchi stati accettanti più il nuovo stato iniziale.
4. Definziamo δ in modo che per ogni $q \in Q$ e ogni $a \in \Sigma_\varepsilon$,

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ e } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ e } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ e } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ e } a = \varepsilon \\ 0 & q = q_0 \text{ e } a \neq \varepsilon \end{cases}$$

Espressioni regolari

In aritmetica, possiamo usare le operazioni $+, \times$ per costruire espressioni come

$$(5 + 3) \times 4.$$

Analogamente, possiamo usare le operazioni regolari per costruire espressioni che descrivono linguaggi, che sono chiamate **espressioni regolari**.
Un esempio è:

$$(0 \cup 1)0^*.$$

Il valore dell'espressione aritmetica è il numero 32, mentre il valore di un'espressione regolare è un linguaggio. In questo caso, il valore è il linguaggio che consiste di tutte le stringhe che iniziano con uno 0 o un 1 seguito da un qualsiasi numero di simboli uguali a 0. Le espressioni regolari hanno un ruolo importante nelle applicazioni dell'informatica. Nelle applicazioni che coinvolgono testo, gli utenti possono voler cercare stringhe che soddisfano alcuni schemi.

Esempio 1.51

Un altro esempio di espressione regolare è

$$(0 \cup 1)^*.$$

Essa inizia con il linguaggio $(0 \cup 1)$ a cui applica l'operatore $*$. Il valore di questa espressione è il linguaggio che consiste di tutte le possibili stringhe di simboli 0 e 1. Se $\Sigma = \{0, 1\}$, possiamo usare Σ come abbreviazione per l'espressione regolare $(0 \cup 1)$. Più in generale, se Σ è un qualsiasi alfabeto, l'espressione regolare Σ descrive il linguaggio che consiste di tutte le stringhe di lunghezza 1 su questo alfabeto e Σ^* descrive il linguaggio che consiste di tutte le stringhe su quell'alfabeto. Analogamente, Σ^*1 è il linguaggio che contiene tutte le stringhe che termiinano con 1. Il linguaggio $(0\Sigma^*) \cup (\Sigma^*1)$ consiste di tutte le stringhe che iniziano con uno 0 o terminano con un 1.

Important

Nelle espressioni regolari, l'operazione star è eseguita per prima, seguita dalla concatenazione e infine dall'unione, a meno che delle parentesi non cambino l'ordine usuale.

Definizione formale di espressione regolare

Definizione

Diciamo che R è un'espressione regolare se R è

1. a per qualche a nell'alfabeto Σ,
2. ε,
3. 0,
4. $(R_1 \cup R_2)$, doveR_1 ed R_2 sono espressioni regolari,
5. $(R_1 \circ R_2)$, doveR_1 ed R_2 sono espressioni regolari, o
6. (R_1^*), dove R_1 è un'espressione regolare.

Nei punti 1 e 2, le espressioni regolari a e ε rappresentano i linguaggi $\{a\}$ e $\{\varepsilon\}$, rispettivamente. Nel punto 3, l'espressione regolare 0 rappresenta il linguaggio vuoto. Nei punti 4, 5, e 6, le espressioni rappresentano i linguaggi ottenuti prendendo l'unione o la concatenazione >dei linguaggi R_1 ed R_2, o lo star del linguaggio R_1, rispettivamente.

Warning

Non confondere le espressioni regolari ε e 0. L'espressione ε rappresenta il linguaggio che contiene una sola stringa - ossia, la stringa vuota - mentre 0 rappresenta il linguaggio che non contiene alcuna stringa.

Apparentemente, sembra che stiamo definendo la nozione di espressione regolare in termini di sé stessa. Se fosse vero, avremmo una definizione circolare, che non sarebbe valida. Comunque, R_1, ed R_2 sono sempre più piccole di R. Quindi in realtà stiamo definendo le espressioni regolari in termini di espressioni regolari più piccole, evitando in tal modo la circolarità. Una definizione di questo tipo è chiamata una **definizione induttiva.**

Le parentesi in un'espressione possono essere omesse. Se lo sono, la valutazione è fatta nell'ordine della precedenza: star, poi concatenazione, poi unione.
Per comodità, usiamo R^+ come abbreviazione per RR^*. In altre parole, mentre R^* contiene tutte le stringhe che sono concatenazione di 0 o più stringhe di R, il linguaggio R^+ contiene tutte le stringhe che sono concatenazione di 1 o più stringhe di R. Quindi $R^+ \cup \varepsilon = R^*$.
Inoltre, denotiamo con R^k l'abbreviazione della concatenazione di k copie di R insieme.

Quando vogliamo distinguere tra un'espressione regolare R e il linguaggio che descrive, denotiamo con $L(R)$ il linguaggio di R.

ESEMPIO 1.53

Negli esempi seguenti, assumiamo che l'alfabeto Σ sia $\{0,1\}$.

1. $0^*10^* = \{w | \ w \text{ contiene un solo } 1\}$.
2. $\Sigma^*1\Sigma^* = \{w | \ w \text{ ha almeno un } 1\}$.
3. $\Sigma^*001\Sigma^* = \{w | \ w \text{ contiene la stringa } 001 \text{ come sottostringa}\}$.
4. $1^*(01^+)^* = \{w | \text{ ogni } 0 \text{ in } w \text{ è seguito da almeno un } 1\}$.
5. $(\Sigma\Sigma)^* = \{w | \ w \text{ è una stringa di lunghezza pari}\}$.[5]
6. $(\Sigma\Sigma\Sigma)^* = \{w | \text{ la lunghezza di } w \text{ è un multiplo di } 3\}$.
7. $01 \cup 10 = \{01, 10\}$.
8. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w | \ w \text{ inizia e termina con lo stesso simbolo}\}$.
9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$.
   L'espressione $0 \cup \varepsilon$ descrive il linguaggio $\{0, \varepsilon\}$, quindi l'operazione di concatenazione aggiunge 0 o ε prima di ogni stringa in 1^*.
10. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$.
11. $1^*\emptyset = \emptyset$.
    Concatenando l'insieme vuoto a un qualsiasi insieme si ottiene l'insieme vuoto.
12. $\emptyset^* = \{\varepsilon\}$.
    L'operazione star concatena un numero qualsiasi di stringhe del linguaggio per ottenere una stringa nel risultato. Se il linguaggio è vuoto, l'operazione star può concatenare 0 stringhe, dando solo la stringa vuota.

Se R è un'espressione regolare qualsiasi, abbiamo le seguenti identità. Esse sono un buon test per controllare se hai capito la definizione.

[5] La ***lunghezza*** di una stringa è il numero di simboli che essa contiene.

$R \cup 0 = R$.
Aggiungere il linguaggio vuoto a un qualsiasi altro linguaggio non lo cambia.

$R \circ \varepsilon = R$.
Concatenare la stringa vuota a una qualsiasi stringa non la cambia.

Tuttavia, scambiare 0 e ε nelle precedenti identità può far si che le uguaglianze non siano vere.

- $R \cup \varepsilon$ può non essere uguale a R.
- Per esempio, se $R = 0$, allora $L(R) = \{0\}$, ma $L(R \cup \varepsilon) = \{0, \varepsilon\}$.

- $R \circ 0$ può non essere uguale a R.
- Per esempio, se $R = 0$, allora $L(R) = \{0\}$ ma $L(R \circ 0) = 0$.

Gli oggetti elemenetari in un linguaggio di programmazione, chiamati ***$token$, come i nomi delle variabili e le costanti, possono essere descritti con espressioni regolari. Per esempio, una costante numerica che può avere una parte decimale e/o un segno può essere descritta come un elemento del linguaggio

$$(+ \cup - \cup \varepsilon)(D^+ \cup D^+.D^* \cup D^*.D^+)$$

dove $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ è l'alfabeto delle cifre decimali. Esempi di stringhe generate sono: $72,\ 3.14159,\ +7.$, e $-.01$.

Equivalenza con gli automi finiti

Le espressioni regolari e gli automi finiti sono equivalenti rispetto alla loro potenza descrittiva. Ogni espressione regolare può essere trasformata in un automa finito che riconosce il linguaggio che essa descrive, e viceversa. Ricorda che in linguaggio regolare è un linguaggio che è riconosciuto da qualche automa finito.

Teorema 1.54

Un linguaggio è regolare se e solo se qualche espressione regolare lo descrive.

Questo teorema deve essere dimostrato in entrambe le direzioni. Noi enunciamo e proviamo ciascuna direzione in un lemma separato.

Lemma 1.55(orale)

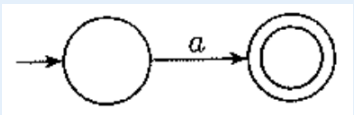
1.55

Se un linguaggio è descritto da un'espressione regolare, allora esso è regolare.

IDEA.
Supponiamo di avere un'espressione regolare R che descrive un linguaggio A.
Mostriamo come trasformare R in un NFA che riconosce A.
Per il Corollario 1.40, se un NFA riconosce A allora A è regolare.

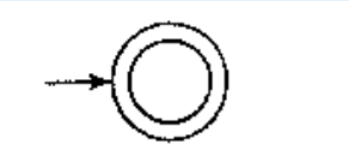
DIMOSTRAZIONE
Trasformiamo R in un NFA N. Consideriamo i sei casi nella definizione di espressione regolare.

1. $R = a$ per qualche $a \in \Sigma$. Allora $L(R) = \{a\}$ e il seguente NFA riconosce $L(R)$.

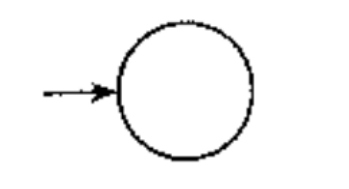


   Nota che questa macchina soddisfa la definizione di NFA ma non quella di DFA perché ha qualche stato con nessun arco uscente per ogni possibile simbolo di input. Naturalmente, qui avremmo potuto presentare un DFA equivalente; ma un NFA è tutto quello di cui abbiamo bisogno per ora, ed è più facile da descrivere. Formalmente, $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, dove descriviamo δ dicendo che $\delta(q_1, a) = \{q2\}$ e $\delta(r, b) = 0$ per $r \neq q_1$ o $b \neq a$.

2. $R = \varepsilon$. Allora $L(R) = \{\varepsilon\}$ e il seguente NFA riconosce $L(R)$.



   Formalmente, $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$ dove $\delta(r, b) = 0$ per ogni r e b.
3. $R = 0$. Allora $L(R) = 0$, e il seguente NFA riconosce $L(R)$.

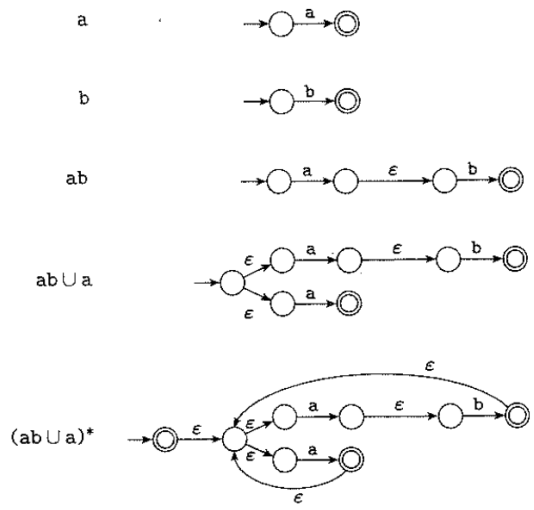


   Formalmente, $N = (\{q\}, \sigma, \delta, q, 0)$, dove $\delta(r, b) = 0 \text{ per ogni } r \text{ e } b$.
4. $R = R_1 \cup R_2$.
5. $R = R_1 \circ R_2$.
6. $R = R_1^*$.

Per gli ultimi tre casi, usiamo le costruzioni date nelle prove che la classe dei linguaggi regolari è chiusa rispetto alle operazioni regolari. In altre parole, costruiamo l'NFA per R dagli NFA per R_1 ed R_2 (o solo R_1 nel caso 6) e mediante l'appropriata costruzione della chiusura.

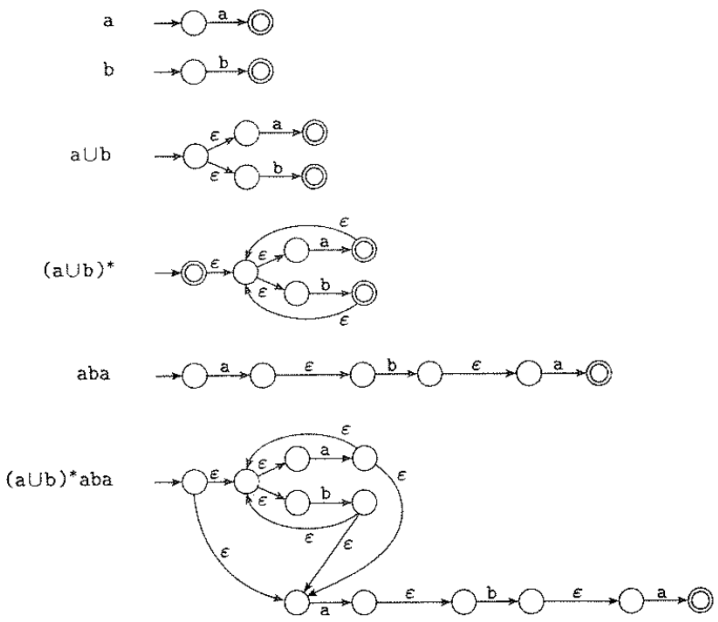
ESEMPIO 1.56

Trasformiamo l'espressione regolare $(ab \cup a)^*$ in un NFA in una sequenza di passi. Costruiamo l'automa dalle sottoespressioni più piccole alle sottoespressioni più grandi finché abbiamo un NFA per l'espressione iniziale, come mostrato nel diagramma seguente. Nota che questa procedura generalmente non dà l'NFA con il minor numero di stati. In questo esempio, la procedura dà un NFA con otto stati, ma il più piccolo NFA equivalente ha solo due stati. Riesci a trovarlo?



ESEMPIO 1.58

Nella Figura 1.59, trasformiamo l'espressione regolare $(a \cup b)^*aba$ in un NFA. Alcuni dei passi meno importanti non sono mostrati.



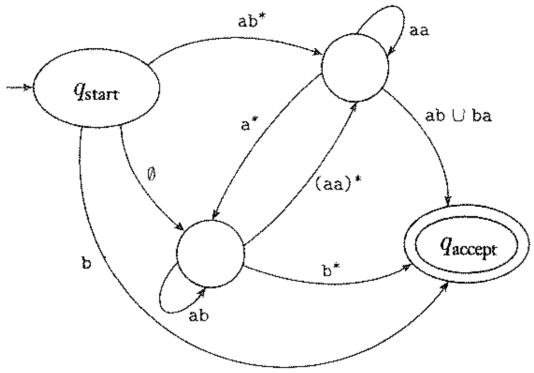
Lemma 1.60

1.60

Se un linguaggio è regolare, allora è descritto da un'espressione regolare.

Automa finito non deterministico generalizzato GNFA

Gli automi finiti non deterministici generalizzati sono semplicemente automi finiti non deterministici nei quali gli archi delle transizioni possono avere espressioni regolari come etichette, invece che solo elementi dell'alfabeto o ε. Il GNFA legge blocchi di simboli dall'input, non necessariamente solo un simbolo alla volta come in un comune NFA. Il GNFA si muove lungo un arco di transizione che collega due stati leggendo un blocco di simboli dall'input che formano una stringa descritta dall'espressione regolare su quell'arco. UN GNFA è non deterministico e quindi può avere diversi modi di elaborare la stessa stringa di input. La figura seguente presenta un esempio di un GNFA.



Per comodità, richiediamo che i GNFA abbiano sempre una forma speciale che soddisfi le seguenti condizioni.

- Lo stato iniziale ha archi di transizione uscenti verso un qualsiasi altro stato ma nessun arco entrante proveniente da un qualsiasi altro stato.
- Esiste un solo stato accettante, ed esso ha archi entranti provenienti da un qualsiasi altro stato ma nessun arco uscente verso un qualsiasi altro stato. Inoltre, lo stato accettante non è uguale allo stato iniziale.
- Eccetto che per lo stato iniziale e lo stato accettante, un arco va da ogni stato ad ogni altro stato e anche da ogni stato in se stesso

Possiamo facilmente trasformare un DFA in un GNFA nella forma speciale. Aggiungiamo semplicemente un nuovo stato iniziale con un ε-arco che entra nel vecchio stato iniziale e un nuovo stato accettante con ε-archi entranti, provenienti dai vecchi stati accettanti. Se alcuni archi hanno più etichette (o se ci sono più archi che collegano gli stessi due stati nella stessa direzione), sostituiamo ognuno di essi con un solo arco la cui etichetta è l'unione delle precedenti etichette. Infine, aggiungiamo archi con etichetta 0 tra stati che non hanno archi. Questo ultimo passo non cambierebbe il linguaggio riconosciuto perché una transizione etichettata con 0 non può mai essere usata.

Definizione 1.64

Un automa finito non deterministico generalizzato è una quintupla, $(Q, \Sigma, \delta, q_{start}, q_{accept})$, dove

1. Q è l'insieme finito degli stati,

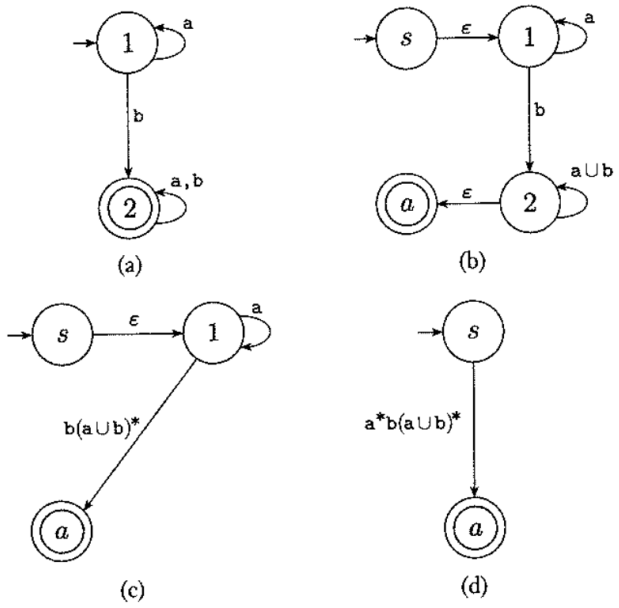
2. Σ è l'alfabeto di input,
3. $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \to R$ è la funzione di transizione,
4. q_{start} è lo stato iniziale e
5. q_{accept} è lo stato accettante.

Un GNFA accetta una stringa w in Σ^* se $w = w_1 w_2 \dots w_k$, dove ogni w_i è in Σ^* ed esiste una sequenza di stati $q_0, q_1, \dots, q_k$ tale che

1. $q_0 = q_{start}$ è lo stato iniziale,
2. $q_k = q_{accept}$ è lo stato accettante e
3. per ogni i, risulta $w_i \in L(R_i)$, dove $R_i = \delta(q_{i-1}, q_i)$; in altre parole, R_i è l'espressione sull'arco da q_{i-1} a q_i.

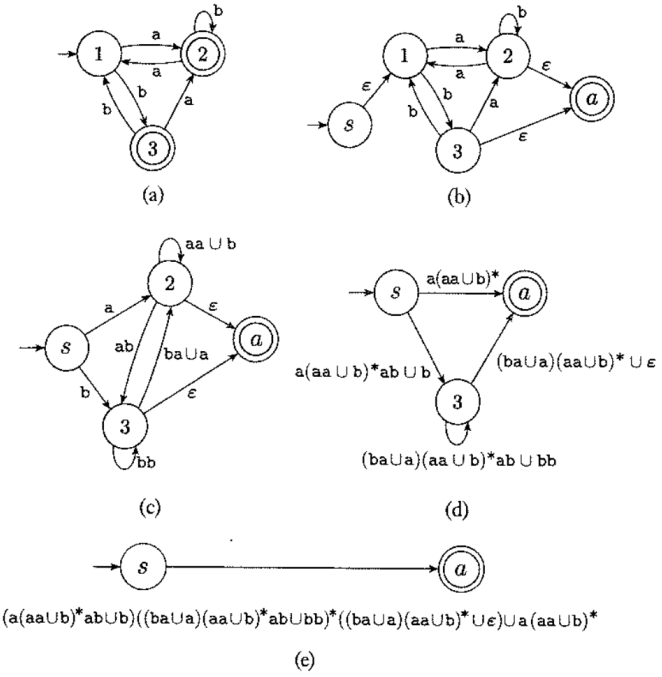
Esempio 1.66

In questo esempio, usiamo il precedente algoritmo per trasformare un DFA in un'espressione regolare. Iniziamo con il DFA con due stati nella Figura 1.67(a).
Nella Figura 1.67(b), creiamo un GNFA con quattro stati aggiungendo un nuovo stato iniziale e un nuovo stato accettante, chiamati s e a invece di q_{start} e q_{accept} in modo da poterli disegnare in modo conveniente. Per evitare di ingombrare la figura, non disegniamo gli archi etichettati 0, sebbene essi vi siano. Nota che sostituiamo le etichette a, b sul ciclo nello stato 2 del DFA con l'etichetta $a \cup b$ nel corrispondente punto del GNFA. Facciamo in questo modo perché le etichette del DFA rappresentano due transizioni, una per a e l'altra per b, mentre il GNFA può avere solo una singola transizione che va da 2 a sé stesso.
Nella Figura 1.67(c), eliminiamo lo stato 2 e aggiorniamo le etichette degli archi restanti. In questo caso, la sola etichetta che cambia è quella da 1 ad a. Nella parte (b) era 0, ma nella parte (c) essa è $b(a \cup b)^*$. Otteniamo questo risultato seguendo il passo 3 della procedura $CONVERT$. Lo stato q_i è lo stato 1, lo stato q_j è a, e q_{rip} è 2, quindi $R_1 = b, R_2 = a \cup b, R_3 = \varepsilon$ ed $R_4 = 0$. Pertanto, la nuova etichetta sull'arco da 1 ad a è $(b)(a \cup b)^*(\varepsilon) \cup 0$. Semplifichiamo quest'espressione regolare in $b(a \cup b)^*$. Nella Figura 1.67(d), eliminiamo lo stato 1 dalla parte (c) e seguiamo la stessa procedura. Poiché restano solo lo stato iniziale e lo stato accettante, l'etichetta sull'arco che li collega è l'espressione regolare equivalente al DFA iniziale.



ESEMPIO 1.68

In questo esempio, iniziamo con un DFA con tre stati. I passi nella trasformazione sono mostrati nella figura seguente.



Linguaggi non regolari

In questa sezione, mostriamo come provare che alcuni linguaggi non possono essere riconosciuti da alcun automa finito.

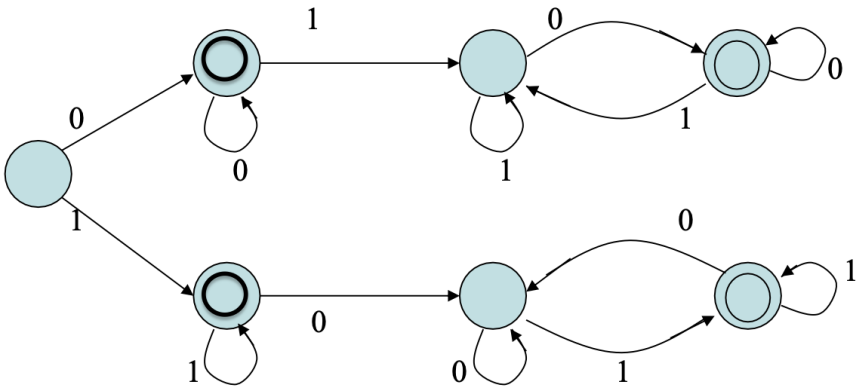
Consideriamo il linguaggio $B = \{0^n1^n | n \geq 0\} = \{\varepsilon, 01, 0011, 000111, \dots\}$. Intuitivamente un DFA che riconosce B dovrebbe ricordare quanti 0 ha visto fin quando legge l'input, ma non è possibile con un numero finito di stati.

Di seguito presentiamo un metodo per provare che linguaggi come B non sono regolari.

Solo perchè il linguaggio sembra richiedere memoria non limitata, non significa che non sia regolare.

Per esempio, consideriamo due linguaggi sull'alfabeto $\Sigma = \{0, 1\}$:

- $C = \{w | w$ ha lo stesso numero di simboli uguali a 0 e simboli uguali $1\}$ non è regolare;
- $D = \{w | w$ha un numero uguale di occorrenze di 01 e 10 come sottostringhe$\}$.
  Sorprendentemente D è regolare ed è accettato dal seguente DFA:



Il pumping lemma

Questo teorema afferma che tutti i linguaggi regolari hanno una proprietà speciale. Se noi possiamo mostrare che un linguaggio non ha questa proprietà, siamo sicuri che esso non è regolare. La proprietà afferma che tutte le stringhe nel linguaggio possono essere "replicate" se la loro lunghezza raggiunge almeno uno specifico valore speciale, chiamato la **lunghezza del pumping**. Questo significa che ogni tale stringa contiene una parte che può essere ripetuta un numero qualsiasi di volte ottenendo una stringa che appartiene ancora al linguaggio.

Teorema 1.70(orale)

Teorema 1.70 (orale)

Se A è un linguaggio regolare, allora esiste un numero p (la lunghezza del pumping) tale che se s è una qualsiasi stringa in A di lunghezza almeno p, allora s può essere divisa in tre parti, $s = xyz$, soddisfacenti le seguenti condizioni:

1. per ogni $i \geq 0, xy^iz \in A$,
2. $|y| > 0$, e
3. $|xy| \leq p$.

Note

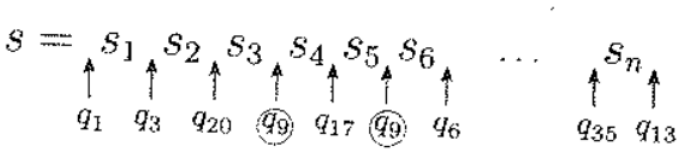
Ricordiamo che $|s|$ rappresenta la lunghezza della stringa s, y^i indica i copie di y concatenate insieme, e y^0 è uguale a ε.

- Quando s è divisa in xyz, x o z potrebbe essere ε, ma la condizione 2 dice che $y \neq \varepsilon$ (senza la condizione 2 il teorema sarebbe banalmente vero).
- La condizione 3afferma che le parti x e y insieme hanno lunghezza al più p; questa è una condizione tecnica supplementare che ogni tanto troviamo utile quando dimostriamo che alcuni linguaggi non sono regolari.
- Scritto più precisamente il pumping lemma è:

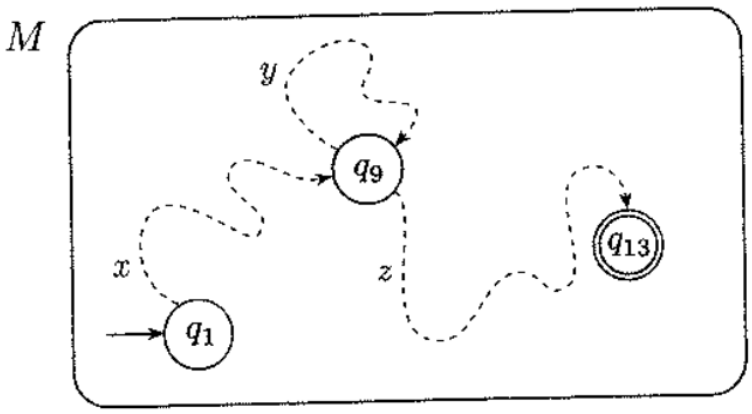
$$A \text{ regolare } \Rightarrow \exists\, p \in N\, \forall s \in A(\ |s| \geq p \Rightarrow \exists\, x, y, z \quad t.c.(\ s = xyz \wedge |y| > 0 \wedge |xy| \leq p \wedge \forall i \in N.$$

IDEA.

Sia $M = (Q, \Sigma, \delta, q_1, F)$ un DFA che riconosce A. Assegnamo alla lunghezza del pumping p il numero degli stati di M. Mostriamo che ogni stringa s in A di lunghezza almeno p può essere divisa nelle tre parti xyz, soddisfacenti le nostre tre condizioni. Cosa accade se nessuna stringa in A è di lunghezza almeno p? Allora il nostro compito è perfino più facile perchè il teorema diventa *banalmente* vero: ovviamente le tre condizioni valgono per tutte le stringhe di lunghezza almeno p se non vi è alcuna di tali stringhe.
Se s in A ha lunghezza almeno p, consideriamo la sequenza di stati che M attraversa nella computazione con input s. Inizia con lo stato iniziale q_1, poi va in q_3, poi per esempio in q_{13}. Se s è in A, sappiamo che M accetta s, quindi q_{13} è uno stato accettante.
Se supponiamo che n sia la lunghezza di s, la sequenza di stati $q_1, q_3, q_{20}, \ldots, q_{13}$ ha lunghezza $n+1$. Poichè n è almeno p , sappiamo che $n+1$ è più grande di p, il numero di stati di M. Quindi, la sequenza deve contenere uno stato che si ripete (**principio della piccionaia**).
La figura seguente mostra la stringa s e la sequenza di stati che M attraversa quando elabora s. Lo stato q_9 è quello che si ripete.



Ora dividiamo s nelle tre componenti x, y e 2. La componente x è la parte di s che compare prima di q_9, la componente y è la parte tra le due occorrenze di q_9 e la componente z è la parte restante di s, che viene dopo la seconda occorrenza di q_9. Quindi x porta M dallo stato q_1 a q_9, y riporta M da q_9 a q_9 e z porta M da q_9 allo stato accettante q_{13}, come mostrato nella figura seguente.



Vediamo perchè questa divisione di s soddisfa le tre condizioni. Supponiamo di eseguire M sull'input $xyyz$. Sappiamo che x porta q_1 a q_9, e poi il primo y lo riporta da q_9 a q_9, come fa il secondo y e poi z lo porta in q_{13}. Quindi essendo q_{13} uno stato accettante, M accetta l'input $xyyz$. Il discorso è analogo per xy^iz per ogni $i > 0$. Nel caso $i = 0$, $xy^iz = xz$, anch'essa accettata. Questo prova la condizione 1.
Per verificare la condizione 2, vediamo che $|y| > 0$, poichè era la parte di s tra due diverse occorrenze dello stato q_9.
Per ottenere la condizione 3, ci assicuriamo che q_9 sia la prima ripetizione nella sequenza. Per il principio della piccionaia, i primi $p + 1$ stati nella sequenza devono contenere una ripetizione. Quindi $|xy| \leq p$.

Dimostrazione

Sia $M = (Q, \Sigma, \delta, q_1, F)$ un DFA che riconosce A e sia p il numero di stati di M.
Sia $s = s_1 s_2 \dots s_n$ una stringa in A di lunghezza n, dove $n \geq p$. Sia $r_1, \dots nr_{n+1}$ la sequenza di stati attraversati da M mentre elabora s, quindi $r_{i+1} = \delta(r_i, s_i)$ per $1 \leq i \leq n$. Questa sequenza ha lunghezza $n + 1$, che è almeno $p + 1$. Due tra i primi $p + 1$ stati devono essere lo stesso stato, per il principio della piccionaia. Chiamiamo il primo di questi r_j e il secondo r_l. Poichè r_l si presenta tra le prime $p + 1$ posizioni in una sequenza che inizia in r_1, abbiamo $l \leq p + 1$. Ora sia $x = s_1 \dots s_{j-1}$, $y = s_j \dots s_{j-1}$ e $z = s_l \dots s_n$. Poichè x porta M da r_1 a r_j, y porta M da r_j a r_j e z porta M da r_j a r_{n+1}, che è uno stato accettante, M deve accettare xy^iz per $i \geq 0$. Sappiamo che $j \neq l$, perciò $|y| > 0$; e $l \leq p + 1$, perciò $|xy| \leq p$. Quindi tutte le condizioni del pumping lemma sono rispettate.

Per usare il pumping lemma per provare che un linguaggio B non è regolare, in primo luogo si assuma che B sia regolare per ottenere una contraddizione. Poi si usi il pumping lemma per assicurare l'esistenza di una lunghezza del pumping p tale che tutte le stringhe di lunghezza maggiore o uguale a p in B possano essere iterate. In seguito, si trovi una stringa s in B che ha lunghezza maggiore o uguale a p, ma che non può essere iterata. Infine, si dimostri che s non può essere iterata considerando tutti i modi di dividere s in x, y e z (prendendo in considerazione la condizione 3 del pumping lemma

se è utile) e, per ogni tale divisione, trovando un valore i tale che $xy^iz \notin B$. Questo passo finale spesso comporta il dover raggruppare i vari modi di dividere s in diversi casi e l'analizzarli individualmente. L'esistenza di s contraddirebbe il pumping lemma se B fosse regolare. Quindi B non può essere regolare. Trovare s a volte richiede un po di ragionamento creativo. Potresti dover cercare tra diversi candidati per s prima di scoprirne uno che funzioni. Prova con elementi di B che sembrano esibire l'"essenza" della non regolarità di B.

Usage of the Pumping Lemma

P.L.: A regular $\Rightarrow \exists p \in \mathbb{N}\ \forall s \in A\ (|s| \geq p \Rightarrow \exists x,y,z$ s.t. $(s = xyz \wedge |y| > 0 \wedge |xy| \leq p \wedge \forall i \in \mathbb{N}.\ xy^iz \in A))$

Hence, the contrapositive of this statement is

$\forall p \in \mathbb{N}\ \exists s \in A\ (|s| \geq p \wedge \forall x,y,z\ (s \neq xyz \vee |y| = 0 \vee |xy| > p \vee \exists i \in \mathbb{N}.\ xy^iz \notin A)$
$\Rightarrow A$ is not regular

Equivalently:

$\forall p \in \mathbb{N}\ \exists s \in A\ (|s| \geq p \wedge \forall x,y,z\ ((s = xyz \wedge |y| > 0 \wedge |xy| \leq p) \Rightarrow \exists i \in \mathbb{N}.\ xy^iz \notin A)$
$\Rightarrow A$ is not regular

Practical use (for proving that A is not regular):

- Consider a generic p
- Find a string $s \in A$ long at least p and decompose it in all possible xyz, with $|y|>0$ and $|xy| \leq p$
- For each such decomposition, find an i such that $xy^iz \notin A$
- Then, A is not regular

6

ESEMPIO 1.73

Sia B il linguaggio $\{0^n1^n \mid n \geq 0\}$. Usiamo il pumping lemma per provare che B non è regolare. La prova è per assurdo.

Assumiamo al contrario che B sia regolare. Sia p la lunghezza del pumping data dal pumping lemma. Scegliamo s uguale alla stringa 0^p1^p. Poiché s è un elemento di B ed s ha lunghezza maggiore di p, il pumping lemma assicura che s può essere divisa in tre parti, $s = xyz$, tali che per ogni $i \geq 0$ la stringa xy^iz è in B. Consideriamo tre casi per mostrare che questo risultato è impossibile.

1. La stringa y consiste solo di simboli uguali a 0. In questo caso, la stringa $xyyz$ ha più simboli uguali a 0 che simboli uguali a 1 e quindi non è un elemento di B, violando la condizione 1 del pumping lemma. Questo caso porta a una contraddizione.
2. La stringa y consiste solo di simboli uguali a 1. Anche questo caso porta a una contraddizione.
3. La stringa y consiste sia di simboli uguali a 0 che di simboli uguali a 1. In questo caso, la stringa $xyyz$ può avere lo stesso numero di simboli

ESEMPIO 1.73

Sia B il linguaggio $\{0^n1^n | n \geq 0\}$. Usiamo il pumping lemma per provare che B non è regolare. La prova è per assurdo.

Assumiamo al contrario che B sia regolare. Sia p la lunghezza del pumping data dal pumping lemma. Scegliamo s uguale alla stringa 0^p1^p. Poiché s è un elemento di B ed s ha lunghezza maggiore di p, il pumping lemma assicura che s può essere divisa in tre parti, $s = xyz$, tali che per ogni $i \geq 0$ la stringa xy^iz è in B. Consideriamo tre casi per mostrare che questo risultato è impossibile.

1. La stringa y consiste solo di simboli uguali a 0. In questo caso, la stringa $xyyz$ ha più simboli uguali a 0 che simboli uguali a 1 e quindi non è un elemento di B, violando la condizione 1 del pumping lemma. Questo caso porta a una contraddizione.
2. La stringa y consiste solo di simboli uguali a 1. Anche questo caso porta a una contraddizione.
3. La stringa y consiste sia di simboli uguali a 0 che di simboli uguali a 1. In questo caso, la stringa $xyyz$ può avere lo stesso numero di simboli

uguali a 0 e simboli uguali a 1, ma essi non saranno nell'ordine corretto, con qualche 1 prima di qualche 0. Quindi non è un elemento di B, il che è una contraddizione.

Pertanto una contraddizione è inevitabile se assumiamo che B sia regolare, quindi B non è regolare. Nota che possiamo semplificare questo ragionamento applicando la condizione 3 del pumping lemma per eliminare i casi 2 e 3.

In questo esempio, trovare la stringa s era facile perché una qualsiasi stringa in B di lunghezza p o maggiore avrebbe funzionato. Nei successivi due esempi, alcune scelte per s non funzionano quindi è richiesta un'attenzione supplementare.

ESEMPIO 1.74

Sia $C = \{w | \ w$ ha lo stesso numero di simboli uguali a 0 e simboli uguali a $1\}$. Usiamo il pumping lemma per provare che C non è regolare. La prova è per contraddizione.

Assumiamo al contrario che C sia regolare. Sia p la lunghezza del pumping data dal pumping lemma. Come nell'Esempio 1.73, sia s la stringa 0^p1^p. Poiché s è un elemento di C che ha lunghezza maggiore di p, il pumping lemma assicura che s può essere divisa in tre parti, $s = xyz$, tali che per ogni $i \geq 0$ la stringa xy^iz è in C. Ci piacerebbe mostrare che questo risultato è impossibile. Ma aspetta, esso *è* possibile! Se prendiamo x e z uguali alla stringa vuota e y uguale alla stringa 0^p1^p, allora xy^iz ha sempre lo stesso numero di simboli uguali a 0 e simboli uguali a 1 e quindi è in C. Quindi *sembra* che s possa essere iterata.

Qui la condizione 3 nel pumping lemma è utile. Essa stabilisce che quando iteriamo s, essa deve essere divisa in modo che $|xy| \leq p$. Questa restrizione sul modo in cui s può essere divisa rende più facile mostrare che la stringa $s = 0^p1^p$ che abbiamo scelto non può essere iterata. Se $|xy| \leq p$, allora y deve consistere solo di simboli uguali a 0, perciò $xyyz \notin C$. Quindi, s non può essere iterata. Questo ci fornisce la contraddizione desiderata.

Scegliere la stringa s in questo esempio richiede più attenzione che nell'Esempio 1.73. Se invece avessimo scelto $s = (01)^p$, avremmo avuto dei problemi perché abbiamo bisogno di una stringa che *non può* essere iterata e quella stringa *può* essere iterata, perfino prendendo in considerazione la condizione 3. Riesci a vedere come iterarla? Un modo per farlo è porre $x = \varepsilon$, $y = 01$ e $z = (01)^{p-1}$. Allora $xy^iz \in C$ per ogni valore di i. Se non riesci a trovare una stringa che non può essere iterata al primo tentativo, non disperare. Provane un'altra!

Un metodo alternativo per provare che C non è regolare segue dal fatto che sappiamo che B non è regolare. Se C fosse regolare, anche $C \cap 0^*1^*$ sarebbe regolare. Questo perché il linguaggio 0^*1^* è regolare e la classe dei linguaggi regolari è chiusa rispetto all'intersezione, come provammo nella

nota a fondo pagina 3 (pagina 49). Ma $C \cap 0^*1^*$ è uguale a B, e noi sappiamo che B non è regolare dall'Esempio 1.73.

ESEMPIO 1.75

Sia $F = \{ww | w \in \{0,1\}^*\}$. Mostriamo che F non è regolare, usando il pumping lemma.

Assumiamo al contrario che F sia regolare. Sia p la lunghezza del pumping data dal pumping lemma. Sia s la stringa 0^p10^p1. Poiché s è un elemento di F ed s ha lunghezza maggiore di p, il pumping lemma assicura che s può essere divisa in tre parti, $s = xyz$, che verificano le tre condizioni del lemma. Mostriamo che questo risultato è impossibile.

La condizione 3 è ancora una volta cruciale perché senza di essa potremmo iterare s se poniamo x e z uguali alla stringa vuota. Con la condizione 3 la prova segue poiché y deve consistere solo di simboli uguali a 0, quindi $xyyz \notin F$.

Osserva che abbiamo scelto $s = 0^p10^p1$ come stringa che esibisce l'"essenza" della non regolarità di F, invece per esempio della stringa 0^p0^p. Sebbene 0^p0^p sia un elemento di F, essa non riesce a dar luogo a una contraddizione poiché può essere iterata.

ESEMPIO 1.76

Mostriamo un linguaggio non regolare unario. Sia $D = \{1^{n^2} | n \geq 0\}$. In altre parole, D contiene tutte le stringhe di simboli uguali a 1 la cui lunghezza è un quadrato perfetto. Usiamo il pumping lemma per provare che D non è regolare. La prova è per contraddizione.

Assumiamo al contrario che D sia regolare. Sia p la lunghezza del pumping data dal pumping lemma. Sia s la stringa 1^{p^2}. Poiché s è un elemento di D ed s ha lunghezza almeno p, il pumping lemma assicura che s può essere divisa in tre parti, $s = xyz$, tali che per ogni $i \geq 0$ la stringa xy^iz è in D. Come negli esempi precedenti, mostriamo che questo risultato è impossibile. Farlo in questo caso richiede una piccola riflessione sulla successione dei quadrati perfetti:

$$0, 1, 4, 9, 16, 25, 36, 49, \ldots$$

Nota il divario crescente tra gli elementi successivi di questa sequenza. Elementi grandi di questa sequenza non possono essere vicini l'un l'altro.

Ora consideriamo le due stringhe xyz e xy^2z. Queste stringhe differiscono l'una dall'altra per una sola ripetizione di y, e conseguentemente le loro lunghezze differiscono di una quantità uguale alla lunghezza di y. Per la condizione 3 del pumping lemma, $|xy| \leq p$ e quindi $|y| \leq p$. Abbiamo $|xyz| = p^2$ e allora $|xy^2z| \leq p^2+p$. Ma $p^2+p < p^2+2p+1 = (p+1)^2$. Inoltre, la condizione 2 implica che y non è la stringa vuota e perciò $|xy^2z| > p^2$.

Quindi, la lunghezza di xy^2z è compresa strettamente tra i quadrati perfetti consecutivi p^2 e $(p+1)^2$. Perciò questa lunghezza non può essere essa stessa un quadrato perfetto. Dunque arriviamo alla contraddizione che $xy^2z \notin D$ e concludiamo che D non è regolare.

ESEMPIO 1.77

Talvolta "cancellare" ("pumping down") è utile quando applichiamo il pumping lemma. Usiamo il pumping lemma per mostrare che $E = \{0^i1^j | i > j\}$ non è regolare. La prova è per contraddizione.

Assumiamo che E sia regolare. Sia p la lunghezza del pumping per E data dal pumping lemma. Sia $s = 0^{p+1}1^p$. Allora s può essere divisa in xyz, dove le tre parti soddisfano le condizioni del pumping lemma. Per la condizione 3, y consiste solo di simboli uguali a 0. Esaminiamo la stringa $xyyz$ per vedere se essa può essere in E. Aggiungere una copia supplementare di y aumenta il numero di simboli uguali a 0. Ma E contiene tutte le stringhe in 0^*1^* che hanno più simboli uguali a 0 che simboli uguali a 1, quindi aumentando il numero di simboli uguali a 0 otterremo ancora una parola in E. Non vi è contraddizione. Dobbiamo provare qualcos'altro.

Il pumping lemma afferma che $xy^iz \in E$ anche quando $i = 0$, quindi consideriamo la stringa $xy^0z = xz$. Eliminare la stringa y fa diminuire il numero di simboli uguali a 0 in s. Ricorda che s ha solo uno 0 in più dei simboli uguali a 1. Pertanto, xz non può avere più simboli uguali a 0 di quelli uguali a 1, di conseguenza non può essere un elemento di E. Quindi otteniamo una contraddizione.

Linguaggi context-free