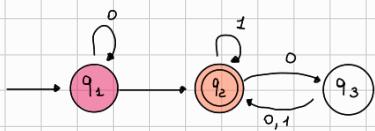


## 1.1 AUTOMI FINITI



La figura sopra è chiamata diagramma di stato di  $M_2$ . Esso ha 3 stati etichettati  $q_1, q_2, q_3$ :

- Lo stato iniziale,  $q_1$ , è indicato dall'arco entrante in esso e non uscente da uno stato.
- Lo stato accettante,  $q_2$ , è quello con doppio cerchio.
- Gli archi che vanno da uno stato a un altro sono chiamati transizioni.
- L'output è accetto o rifiuta.

### DEFINIZIONE FORMALE DI AUTOMA FINITO

La definizione formale di automa finito afferma che un automa finito è una lista di questi cinque oggetti: insieme degli stati, alfabeto di simboli di input, regole per i cambiamenti di stato, stato iniziale e stati accettanti.

Usiamo qualcosa il cui nome è funzione di transizione, denotata  $\delta$ , per definire le regole del cambiamento di stato. Se l'automa finito ha un arco da uno stato  $x$  a uno stato  $y$  etichettato con il simbolo di input  $z$ , questo significa che se l'automa è nello stato  $x$  quando legge un  $z$ , allora si sposta nello stato  $y$ . Possiamo indicare questa cosa con la funzione di transizione dicendo che  $\delta(x, z) = y$ .

#### DEFINIZIONE 1.5

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

1.  $Q$  è un insieme finito chiamato l'insieme degli stati;
2.  $\Sigma$  è un insieme finito chiamato l'alfabeto;
3.  $\delta : Q \rightarrow Q$  è la funzione di transizione;
4.  $q_0 \in Q$  è lo stato iniziale;
5.  $F \subseteq Q$  è l'insieme degli stati accettanti.

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

Una macchina può accettare diverse stringhe, ma riconosce sempre soltanto un linguaggio. Se la macchina non accetta alcuna stringa, essa riconosce ancora un linguaggio, ossia il linguaggio vuoto  $\emptyset$ .

### DEFINIZIONE FORMALE DI COMPUTAZIONE

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

1.  $r_0 = q_0$
2.  $\delta(r_i, w_{i+1}) = r_{i+1}$ , per  $i = 0, \dots, n-1$
3.  $r_n \in F$

La condizione 1 afferma che la macchina inizia nello stato iniziale. La condizione 2 afferma che la macchina passa da uno stato all'altro in base alla funzione di transizione. La condizione 3 afferma che la macchina accetta il suo input se termina la lettura in uno stato accettante. Diciamo che  $M$  riconosce il linguaggio  $A$  se  $A = \{w / M$  accetta  $w\}$ .

#### DEFINIZIONE 1.16

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

## LE OPERAZIONI REGOLARI

Definiamo 3 operazioni sui linguaggi regolari, 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**

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**:  $A^* = \{x_1 x_2 \dots x_k / k \geq 0 \text{ e } \text{ogni } x_i \in A\}$

L'operazione star è un'operazione unaria invece che un'operazione binaria. Essa opera concatenando un numero qualsiasi di stringhe in  $A$  insieme per ottenere una stringa di un nuovo linguaggio. Poiché "un numero qualsiasi" include 0 come possibilità, la stringa vuota è sempre un elemento di  $A^*$ , indipendentemente da chi sia  $A$ .

### Esempio 1.24

Sia  $\Sigma$  l'alfabeto standard a 26 lettere  $\{a, b, \dots, z\}$ . Se  $A = \{\text{good}, \text{bad}\}$  e  $B = \{\text{boy}, \text{girl}\}$ , allora:

- $A \cup B = \{\text{good}, \text{bad}, \text{boy}, \text{girl}\}$
- $A \circ B = \{\text{good boy}, \text{good girl}, \text{bad boy}, \text{bad girl}\}$
- $A^* = \{\epsilon, \text{good}, \text{bad}, \text{good good}, \text{good bad}, \text{bad good}, \text{bad bad}, \text{good good good}, \text{good good bad}, \text{good bad good}, \text{good bad bad} \dots\}$

### TEOREMA 1.25

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

In altre parole, se  $A_1$  e  $A_2$  sono linguaggi regolari, lo è anche  $A_1 \cup A_2$ .

**IDEA:** Abbiamo due linguaggi regolari  $A_1$  e  $A_2$ , e vogliamo mostrare che anche  $A_1 \cup A_2$  è regolare. Poiché  $A_1$  e  $A_2$  sono regolari, sappiamo che un automa finito  $M_1$  riconosce  $A_1$ , e un automa finito  $M_2$  riconosce  $A_2$ . Per provare che  $A_1 \cup A_2$  è regolare, esibiamo un automa finito  $M$ , che riconosce  $A_1 \cup A_2$ . Costruiamo  $M$  da  $M_1$  ed  $M_2$ . Per riconoscere il linguaggio unione, la macchina  $M$  deve accettare il suo input esattamente quando  $M_1$  o  $M_2$  lo accetterebbero.

Immagina che tu sia  $M$ . Come ricevi i simboli input uno a uno, tu simuli sia  $M_1$  che  $M_2$  contemporaneamente. In questo modo è necessario scorrere l'input solo una volta. Ma come puoi tenere traccia di entrambe le simulazioni con una memoria finita? Tutto quello che hai bisogno di ricordare è lo stato in cui ciascuna macchina sarebbe se avesse letto l'input fino a quel punto. Quindi, devi ricordare una coppia di stati. Se  $M_1$  ha  $K_1$  stati ed  $M_2$  ha  $K_2$  stati, il numero delle coppie di stati è il prodotto  $K_1 \times K_2$ .

**DIM:** Supponiamo che  $M_1$  riconosca  $A_1$ , dove  $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ , ed  $M_2$  riconosca  $A_2$ , dove  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ .

Costruiamo  $M$  che riconosca  $A_1 \cup A_2$ , dove  $M = (Q, \Sigma, \delta, q_0, F)$ .

1.  $Q = \{(r_1, r_2) / r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$

Questo insieme è il prodotto cartesiano degli insiemii  $Q_1$  e  $Q_2$ , ed è denotato con  $Q_1 \times Q_2$ .

2.  $\Sigma$ , l'alfabeto, è lo stesso di  $M_1$  e  $M_2$ .

3.  $\delta$  è definita come segue. Per ogni  $(r_1, r_2) \in Q$  e ogni  $a \in \Sigma$ , sia:  $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$ .

4.  $q_0$  è la coppia  $(q_1, q_2)$ .

5.  $F$  è l'insieme delle coppie in cui l'uno o l'altro elemento è uno stato accettante di  $M_1$  o  $M_2$ . Possiamo scriverlo come:

$$F = \{(r_1, r_2) / r_1 \in F_1 \text{ o } r_2 \in F_2\}$$

Questo conclude la costruzione di  $M$ .

## TEOREMA 1.26

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

Per provare questo teorema, proviamo qualcosa di simile alla prova del caso dell'unione. Come prima, possiamo iniziare con gli automi finiti  $M_1$  ed  $M_2$  che riconoscono i linguaggi regolari  $A_1$  e  $A_2$ . Ma ora, invece di costruire l'automa  $M$  che accetta il suo input se  $M_1$  o  $M_2$  lo accetta, esso deve accettare il suo input se può essere diviso in due parti, tali che  $M_1$  accetta la prima parte ed  $M_2$  accetta la seconda parte. Il problema è che  $M$  non sa dove dividere il suo input (cioè, dove finisce la prima parte e inizia la seconda). Per risolvere questo problema, introduciamo una nuova tecnica.

## 1.2 NON DETERMINISMO

Quando la macchina è in un dato stato e legge il simbolo di input successivo, sappiamo qual è lo stato successivo - esso è univocamente determinato. In questo caso si parla di **computazione deterministica**. 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 deterministico è automaticamente un automa finito deterministico.

La differenza tra un automa finito deterministico (DFA) e un automa finito non deterministico (NFA) è subito evidente:

- Ogni stato di un DFA ha sempre esattamente un arco di transizione uscente per ogni simbolo nell'alfabeto. L'NFA viola questa regola.  
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 di transizione sono simboli dell'alfabeto. In generale, un NFA può avere archi etichettati con elementi dell'alfabeto o  $\epsilon$ . Zero, uno, o più archi possono essere etichettati con  $\epsilon$ .

## DEFINIZIONE FORMALE DI AUTOMA FINITO NON DETERMINISTICO

La definizione formale di automa finito non deterministico è simile a quella di automa finito deterministico. Essi differiscono in un aspetto essenziale: nel tipo di funzione di transizione. In un DFA, la funzione di transizione prende uno stato e un simbolo di input e produce lo stato successivo. In un NFA, la funzione di transizione prende uno stato e un simbolo di input o la stringa vuota e produce l'insieme dei possibili stati successivi.

Per scrivere la definizione formale, dobbiamo introdurre alcune nozioni aggiuntive. Per un qualsiasi insieme  $Q$  denotiamo  $P(Q)$  la collezione di tutti i sottoinsiemi di  $Q$ . In questo caso  $P(Q)$  è chiamato **l'insieme potenza** di  $Q$ . Per ogni alfabeto  $\Sigma$  scriviamo  $\Sigma^*$  per denotare  $\Sigma \cup \{\epsilon\}$ . Ora possiamo descrivere la funzione di transizione come  $\delta: Q \times \Sigma^* \rightarrow P(Q)$ .

### 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.  $\Sigma$  è un alfabeto finito
3.  $\delta: Q \times \Sigma^* \rightarrow P(Q)$  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.

La definizione formale di computazione per un NFA è simile a quella per un DFA. Sia  $N = (Q, \Sigma, \delta, q_0, F)$  un NFA e sia  $w$  una stringa sull'alfabeto  $\Sigma$ . Diciamo che  $N$  accetta  $w$  se possiamo scrivere  $w$  come  $w = y_1 y_2 \dots y_m$ , dove ciascun  $y_i$  è un elemento di  $\Sigma^*$  ed esiste una sequenza di stati  $r_0, r_1, \dots, r_m$  in  $Q$  con 3 condizioni:

1.  $r_0 = q_0$
2.  $r_{i+1} \in \delta(r_i, y_{i+1})$ , per  $i = 0, \dots, m-1$
3.  $r_m \in F$

La condizione 1 dice che la macchina inizia nello stato iniziale. La condizione 2 dice che lo stato  $r_{i+1}$  è uno dei possibili stati successivi quando  $N$  è nello stato  $r_i$  e sta leggendo  $y_{i+1}$ . Infine, la condizione 3 dice che la macchina accetta il suo input se l'ultimo stato è uno stato accettante.

## EQUIVALENZA TRA GLI NFA E I DFA

Gli automi finiti deterministici e non deterministici riconoscono la stessa classe di linguaggi. Per questo motivo vengono dette equivalenti.

### TEOREMA 1.39

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

**IDEA:** L'idea è 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.

**DIM:** 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 fare la costruzione completa, consideriamo inizialmente il caso più semplice in cui l'NFA non ha  $\epsilon$ -archi.

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

Ogni stato di  $M$  è un insieme di stati di  $N$ .

$$2. \delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$$

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  $R$ . Poiché da ogni stato si va in un insieme di stati, prendiamo l'unione di tutti questi insiemi.

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

$M$  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  $\epsilon$ -archi. Per farlo introduciamo un'ulteriore notazione.

Per ogni stato  $R$  di  $M$ , definiamo  $E(R) = \{q/q \text{ può essere raggiunto da } R \text{ attraverso } 0 \text{ o più } \epsilon\text{-archi}\}$ .

Poi modifichiamo la funzione di transizione di  $M$ . Sostituendo  $\delta(r, a)$  con  $E(\delta(r, a))$  realizziamo questo effetto.

Inoltre dobbiamo modificare lo stato iniziale di  $M$ . Cambiare  $q_0'$  con  $E(\{q_0\})$  realizza questo risultato.

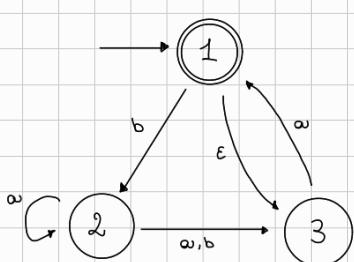
### COROLLARIO 1.40

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

Il teorema 1.39 mostra che ogni NFA può essere trasformato in un DFA equivalente. Conseguentemente, se un NFA riconosce un linguaggio, lo fa anche qualche DFA, e quindi il linguaggio è regolare. L'altra direzione del "se e solo se" afferma che un linguaggio è regolare solo se qualche NFA lo riconosce. Cioè, se un linguaggio è regolare, qualche NFA deve riconoscerlo. Ormai questa condizione è vera perché un linguaggio regolare ha un DFA che lo riconosce e un DFA è anche un NFA.

### Esempio 1.41 (Passaggio dal NFA al DFA)

Consideriamo l'NFA  $N$  seguente:



La descrizione formale di  $N = (Q, \{\alpha, b\}, \delta, 1, \{1\})$ , con  $Q = \{1, 2, 3\}$   
Rappresentiamo la funzione di transizione:

$\delta$	$a$	$b$
1	$\emptyset$	$\{2\}$
2	$\{2, 3\}$	$\{3\}$
3	$\{1\}$	$\emptyset$

Per costruire un DFA  $D$  equivalente a  $N$ , innanzitutto determinizziamo gli stati di  $D$ .  $N$  ha tre stati  $\{1, 2, 3\}$ , quindi  $D$  avrà  $2^3 = 8$  stati. Etichettiamo ciascuno degli stati di  $D$  con il corrispondente sottoinsieme. Quindi l'insieme degli stati di  $D$  è:

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

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

Infine andiamo a determinare la funzione di transizione di  $D$ . Dalla teoria sappiamo che, in generale, formalmente:

$$\delta_D(R, \omega) = \bigcup_{r \in R} \delta(r, \omega).$$

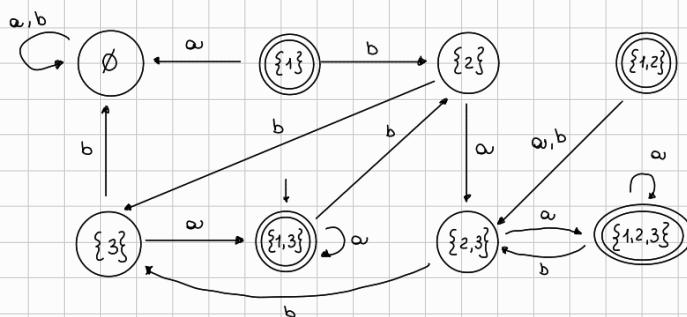
Ad esempio, se  $R = \{q_1, q_2\}$ , e se la funzione di transizione dell'NFA stabilisce che  $\delta(q_1, \omega) = \{p_1, p_2\}$  e  $\delta(q_2, \omega) = \{p_3\}$ , allora:

$$\delta_D(S, \omega) = \bigcup_{r \in S, r \in R} \delta(r, \omega) = \delta(q_1, \omega) \cup \delta(q_2, \omega) = \{p_1, p_2\} \cup \{p_3\} = \{p_1, p_2, p_3\}.$$

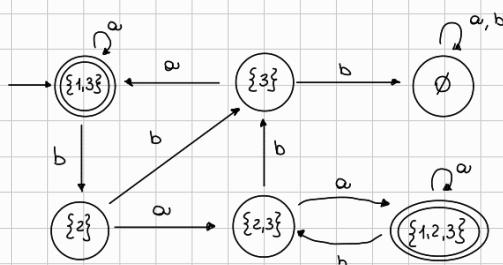
Applichiamo dunque al nostro esempio:

$\delta_D$	$a$	$b$
$\emptyset$	$\emptyset$	$\emptyset$
$\{\{1\}\}$	$\emptyset$	$\{\{2\}\}$
$\{\{2\}\}$	$\{\{2,3\}\}$	$\{\{3\}\}$
$\{\{3\}\}$	$\{\{1,3\}\}$	$\emptyset$
$\{\{1,2\}\}$	$\{\{2,3\}\}$	$\{\{2,3\}\}$
$\{\{1,3\}\}$	$\{\{1,3\}\}$	$\{\{2\}\}$
$\{\{2,3\}\}$	$\{\{1,2,3\}\}$	$\{\{3\}\}$
$\{\{1,2,3\}\}$	$\{\{1,2,3\}\}$	$\{\{2,3\}\}$

Avremo dunque il seguente automa:



Possiamo semplificare questa macchina osservando che nessun arco punta agli stati  $\{1\}$  e  $\{1,2\}$ , quindi possono essere tolti senza ripercussioni sulla prestazione della macchina.



### 1.3 ESPRESSIONI REGOLARI

In aritmetica, possiamo usare le operazioni  $+$  e  $\times$  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 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.

#### DEFINIZIONE FORMALE DI ESPRESSIONE REGOLARE

##### Definizione 1.52

- Diciamo che  $R$  è un'espressione regolare se  $R$  è
1.  $a$  per qualche  $a \in \Sigma$
  2.  $\epsilon$
  3.  $\emptyset$
  4.  $(R_1 \cup R_2)$ , dove  $R_1$  ed  $R_2$  sono espressioni regolari
  5.  $(R_1 \circ R_2)$ , dove  $R_1$  ed  $R_2$  sono espressioni regolari
  6.  $(R_1^*)$ , dove  $R_1$  è un'espressione regolare

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

#### EQUIVALENZA CON AUTOMI FINITI

##### 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. Verrà enunciata e provata ciascuna direzione in lemmi separati.

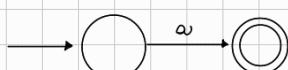
##### Lemme 1.55

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.

**DIM:** 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. Avremmo potuto presentare un DFA equivalente, ma in questo caso ci basta l'NFA.

2.  $R = \epsilon$ . Allora  $L(R) = \epsilon$  e il seguente NFA riconosce  $L(R)$ :



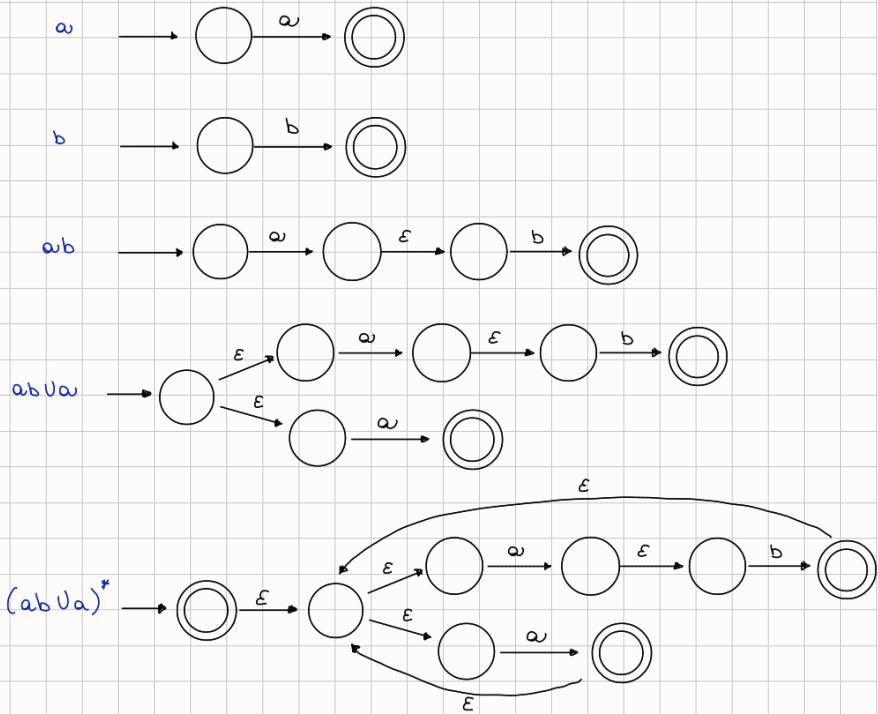
3.  $R = \emptyset$ . Allora  $L(R) = \emptyset$  e il seguente NFA riconosce  $L(R)$ :



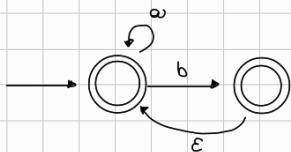
4.  $R = R_1 \cup R_2$
5.  $R = R_1 \circ R_2$
6.  $R = R_1^*$

**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. In questo esempio, la procedura dà un NFA con otto stati, ma il più piccolo NFA equivalente ha solo due stati.



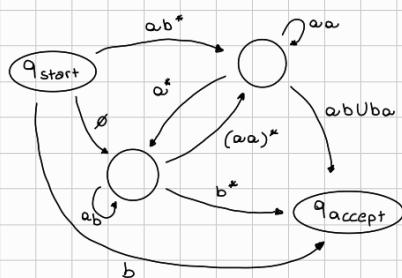
Il più piccolo NFA dovrebbe essere:

**LEMMA 1.60**

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

**IDEA:** Dobbiamo mostrare che se un linguaggio  $A$  è regolare, allora un'espressione regolare lo descrive. Poiché  $A$  è regolare, esso è accettato da un DFA. Descriviamo una procedura per trasformare i DFA in espressioni regolari equivalenti. Dividiamo questa procedura in due parti, usando un nuovo tipo di automa finito chiamato **automa finito non deterministico generalizzato (GNFA)**. Prima mostriamo come trasformare un DFA in un GNFA e poi come trasformare un GNFA in un'espressione regolare.

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  $\epsilon$ .



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

- lo stato iniziale ha archi di transizione uscenti ma nessun arco entrante.
- Esiste un solo stato accettante, ed esso ha archi entranti ma non uscenti. 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 su se stesso.

**DIM:** In primo luogo definiamo formalmente il GNFA. Questo è simile a un automa finito non deterministico tranne che per la funzione di transizione, che ha la forma:

$$\delta: (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow R$$

Il simbolo  $R$  è la collezione di tutte le espressioni regolari sull'alfabeto  $\Sigma$ .

#### Definizione 1.64

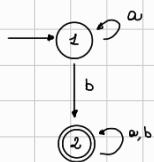
Un automa finito non deterministico generalizzato è una quintupla  $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$ , dove:

1.  $Q$  è l'insieme finito degli stati
2.  $\Sigma$  è l'alfabeto di input
3.  $\delta: (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow R$  è la funzione di transizione.
4.  $q_{\text{start}}$  è lo stato iniziale
5.  $q_{\text{accept}}$  è lo stato accettante.

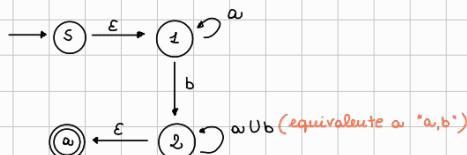
Per capire come effettuare la trasformazione da DFA a GNFA vediamo un esempio.

#### Esempio 1.66

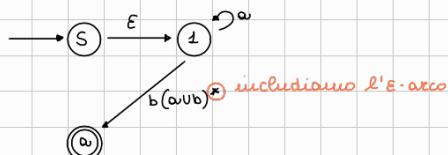
Iniziamo con il DFA con due stati =>



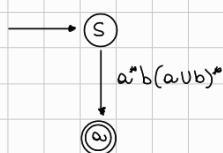
Creiamo ora un GNFA con quattro stati aggiungendo un nuovo stato iniziale e un nuovo stato accettante:



Eliminiamo lo stato 2 e aggiorniamo le etichette degli archi restanti:



Eliminiamo lo stato 1 ed eseguiamo la stessa procedura:



## 1.4 LINGUAGGI NON REGOLARI

Per comprendere il potere computazionale degli automi finiti dobbiamo anche comprenderne i limiti. In questa sezione, mostriamo come provare che alcuni linguaggi non possono essere riconosciuti da alcun automa finito.

### IL PUMPING LEMMA PER I LINGUAGGI REGOLARI

La nostra tecnica per provare la non regolarità deriva da un teorema sui linguaggi regolari, tradizionalmente chiamato **pumping lemma**.

#### TEOREMA 1.70

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,  $s = xyz$ , soddisfacendo le seguenti condizioni:

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

DIM: Sia  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA che riconosce il linguaggio  $A$ , e sia  $p$  (lunghezza del pumping) il numero di stati di  $M$ . Sia  $s = s_1 s_2 \dots s_n$  una stringa in  $A$  di lunghezza  $n$ , dove  $n \geq p$  (perché la stringa deve essere lunga almeno  $p$ ).

Sia infine  $r_1, r_2, \dots, r_{n+1}$  la sequenza di stati che  $M$  assume durante la computazione di  $s$ , tali che  $\delta(r_i, s_i) = r_{i+1}$  per  $1 \leq i \leq n$ .

Il numero di stati di  $M$  è  $p$  e dato che  $n \geq p$ , per il **principio della piccionaia**, sappiamo che ci deve essere almeno uno stato che si ripete. Pertanto esistono  $i$  e  $j$  tali che  $r_i = r_j$ .

Se chiamiamo il primo degli stati che si ripetono  $r_j$  e il secondo  $r_i$ , possiamo suddividere la stringa in ingresso in questo modo:

$$\begin{aligned}x &= s_1 \dots s_{j-1} \\y &= s_j \dots s_{i-1} \\z &= s_i \dots s_n\end{aligned}$$

Dato che la sottostringa  $x$  tiene occupato  $M$  dallo stato  $r_1$  a  $r_j$ , la sottostringa  $y$  lo tiene occupato da  $r_j$  ad  $r_j$ , e la sottostringa  $z$  da  $r_j$  a  $r_{n+1}$ , che è lo stato accettante, allora  $M$  deve accettare  $xy^i z$  per qualsiasi  $i \geq 0$  (prima condizione del teorema).

Verifichiamo invece la seconda condizione del teorema osservando che poiché  $j \neq i$ , allora per forza di cose  $|y| > 0$ , perché nel peggior caso  $|y| = 1$ .

In fine, dato che tra i primi  $p+1$  elementi della sequenza di stati deve esserci almeno una ripetizione, allora  $|xy| \leq p$  (terza condizione del teorema).