Argomenti trattati durante la lezione:

- Espressioni Regolari
- Equivalenza con gli automi finiti

Operazioni sui linguaggi

- **Unione:**
$$L \cup M = \{w : w \in L \text{ oppure } w \in M\}$$
- **Intersezione:**
$$L \cap M = \{w : w \in L \text{ e } w \in M\}$$
- **Complemento:**
$$\overline{L} = \{w : w \notin L\}$$
- **Concatenazione:**
$$L.M = \{uv : u \in L \text{ e } v \in M\}$$
- **Chiusura (o Star) di Kleene:**
$$L^* = \{w_1 w_2 \dots w_k : k \geq 0 \text{ e ogni } w_i \in L\}$$

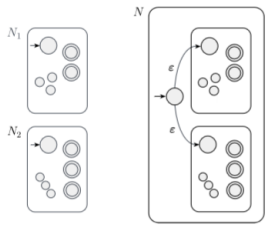
Come per le espressioni aritmetiche, anche per le espressioni regolari ci sono delle regole di precedenza degli operatori:

1. Chiusura di Kleene
2. Concatenazione (punto)
3. Unione (+)

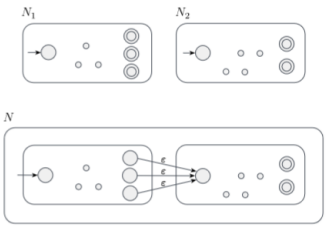
Da RE a ε-NFA

Caso Induttivo:

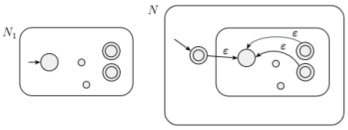
- automa per R + S



- automa per RS



- automa per R*



Esercizi (2)

Per ognuno dei seguenti linguaggi, costruire una ER sull'alfabeto $\{0, 1\}$ che li rappresenti:

4. Tutte le stringhe w che contengono la sottostringa 101
5. Tutte le stringhe w che non contengono la sottostringa 101

Sfida!

Costruire una ER sull'alfabeto $\{0, 1\}$ per il linguaggio di tutti i numeri binari multipli di 3.

Esercizio 4: Stringhe che contengono la sottostringa 101

```
(0|1)*101(0|1)*
```

Questa espressione rappresenta qualsiasi stringa che contiene "101" in una posizione qualsiasi:

- `(0|1)*` : qualsiasi prefisso (anche vuoto)
- `101` : la sottostringa richiesta
- `(0|1)*` : qualsiasi suffisso (anche vuoto)

Esercizio 5: Stringhe che NON contengono la sottostringa 101

Copia

```
(0|11|10(0|11|10)*)*
```

Questa espressione funziona perché garantisce che dopo aver letto "10", il carattere successivo non possa mai essere "1":

- `0` : possiamo inserire zero
- `11` : possiamo inserire due uno consecutivi
- `10(0|11|10)*` : dopo "10" possiamo avere solo "0", "11" o un altro "10", mai un singolo "1"

Sfida: Numeri binari multipli di 3

Co

```
(0|1(01*0)*1)*
```

Questa espressione deriva dall'automa a stati finiti che riconosce i multipli di 3 tramite i resti:

- Stato 0: resto 0 (accettazione)
- Stato 1: resto 1
- Stato 2: resto 2

Le transizioni seguono la logica: quando aggiungiamo una cifra a destra in un numero binario, il nuovo resto è determinato dal resto precedente e dalla cifra aggiunta secondo le regole:

- resto(2n) = (2 × resto(n)) mod 3
- resto(2n+1) = (2 × resto(n) + 1) mod 3

L'espressione cattura tutti i percorsi che, partendo dal resto 0, tornano al resto 0.

ER che mostra che tutte le stringhe contengano ciascun simbolo almeno una volta

$(a+b+c)^*a(a+b+c)^*b(a+b+c)^*c$

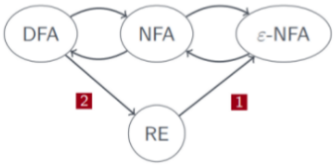
ER per stringhe binarie che contengono almeno tre 1:
(0+1)*1(0+1)*1(0+1)*1(0+1)*

ER per stringhe di testo che descriva le date in formato GG/MM/AAAA
0(1+2...+9)+(1+2)/(1+2+...9)+3(0+1)/(0+1)(1+2)/(0+1+...9)(0+1+...9)(0+1+...9)(0+1+...9)

5) Tutte le stringhe che contengono 4k + 1 occorrenze di "b" per "k" >= 0
Soluzione:
((a|c)*b(a|c)*b(a|c)*b(a|c)*b(a|c)*)* (a|c)*b(a|c)*

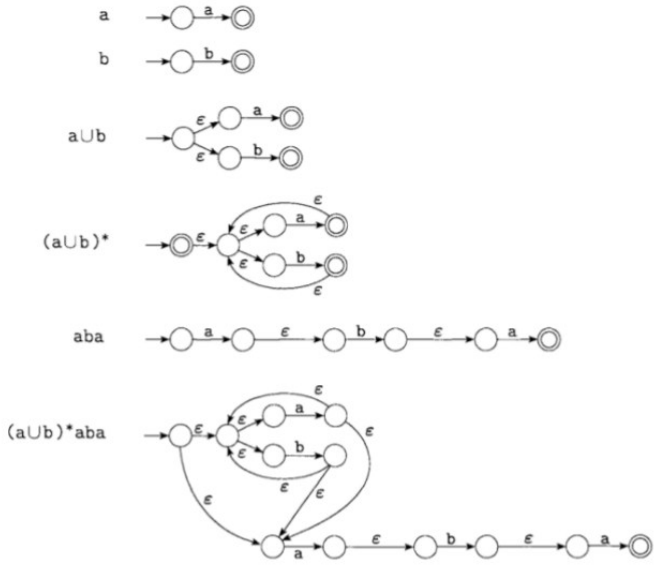
Equivalenza tra FA e RE

Sappiamo già che DFA, NFA, e ε-NFA sono tutti equivalenti.

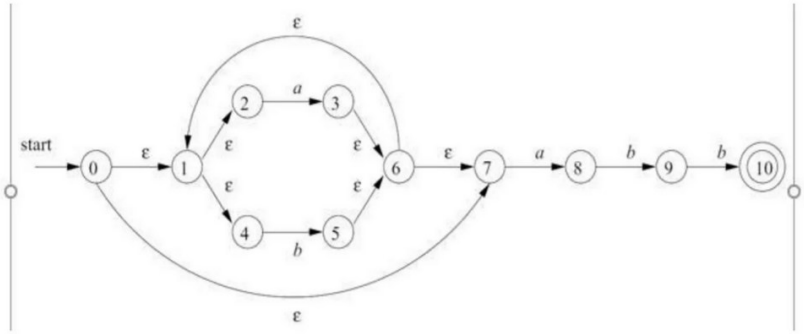


Gli FA sono equivalenti alle espressioni regolari:

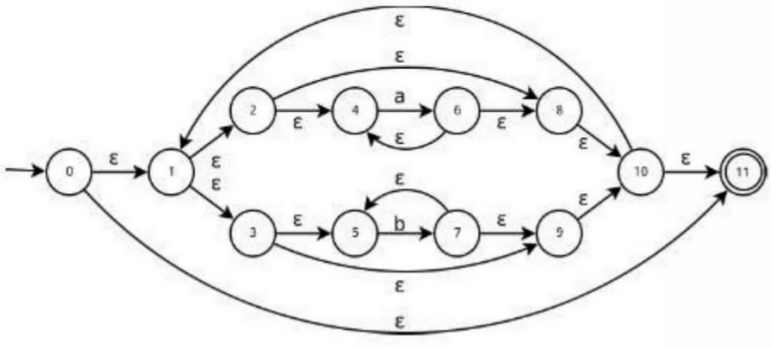
1. Per ogni espressione regolare R esiste un ε-NFA A, tale che $L(A) = L(R)$
2. Per ogni DFA A possiamo costruire un'espressione regolare R, tale che $L(R) = L(A)$



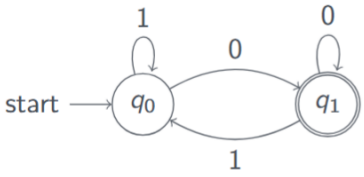
Construct NFA for RE: (a+b)*abb

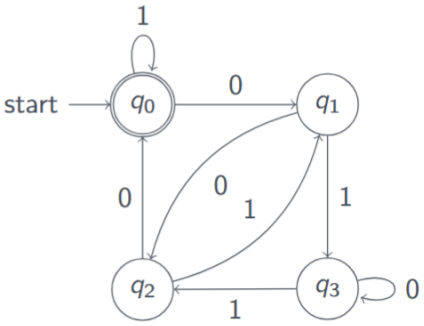


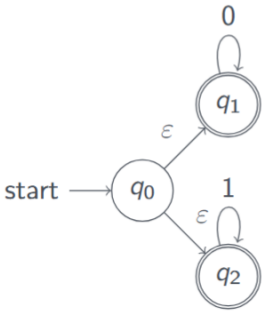
Construct NFA epsilon for RE : (a*|b*)*



Costruite una Espressione Regolare equivalente ai seguenti automi:







Automa 1 (in alto a sinistra)

```
1*0(0|11*0)*
```

Questa espressione riflette il comportamento dell'automa dove:

- Si può iniziare con zero o più '1' (rimanendo in q0)
- Si deve poi leggere uno '0' per passare a q1 (stato di accettazione)
- Una volta in q1, possiamo:
  - Leggere un altro '0' e rimanere in q1
  - Leggere un '1', tornare a q0, leggere zero o più '1', e poi un '0' per tornare a q1

Automa 2 (in alto a destra)

Copia

```
(1|000|01(0|11)*10)*
```

Questa espressione rappresenta tutte le stringhe accettate dall'automa con stato iniziale e finale q0:

- Si può rimanere in q0 leggendo '1'
- Si può seguire il percorso q0→q1→q2→q0 con la sequenza '000'
- Si può seguire il percorso q0→q1→q3→...→q2→q0 con la sequenza '01' seguita da zero o più '0' o coppie '11', e terminando con '10'

Automa 3 (in basso a sinistra)

```
0*|1*
```

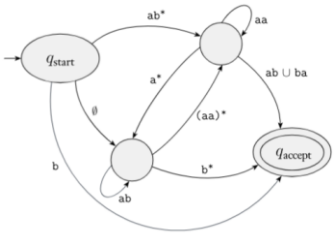
Questa espressione rappresenta il comportamento più semplice dell'automa con ε-transizioni:

- Da q0 possiamo andare con ε in q1 (accettazione) e leggere zero o più '0'
- Oppure possiamo andare con ε in q2 (accettazione) e leggere zero o più '1'
- Quindi il linguaggio accetta stringhe composte solo da '0' o solo da '1'

Conversione per eliminazione di stati

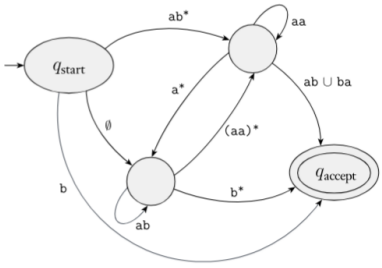
- La procedura che vedremo è in grado di convertire un qualsiasi automa (DFA o NFA) in una espressione regolare equivalente
- Si procede per eliminazione di stati
- Quando uno stato q viene eliminato, i cammini che passano per q scompaiono
- si aggiungono nuove transizioni etichettate con espressioni regolari che rappresentano i cammini eliminati
- alla fine otteniamo un'espressione regolare che rappresenta tutti i cammini dallo stato iniziale ad uno stato finale
  $\Rightarrow$ cioè il linguaggio riconosciuto dall'automa

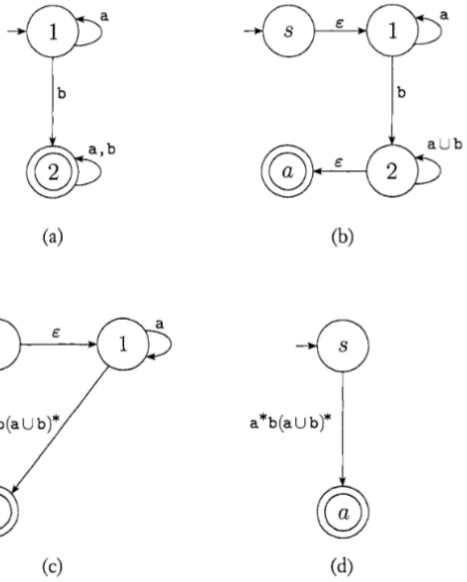
- Sono NFA dove le transizioni sono etichettate con espressioni regolari
- Ogni transizione consuma un blocco di simboli dall'input che appartiene al linguaggio dell'espressione regolare



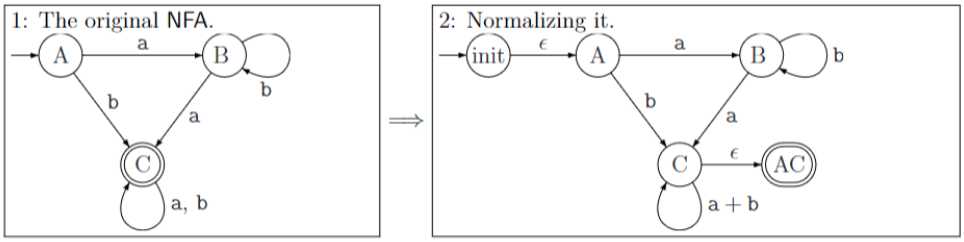
Primo passo: da NFA a GNFA

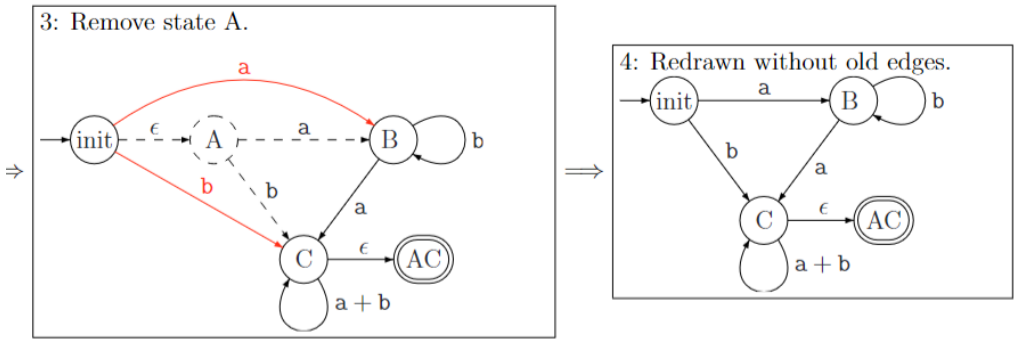
1. Nuovo stato iniziale q_{start} con transizione ε verso il vecchio q_0
2. Nuovo stato finale q_{accept} con transizione ε da tutti i vecchi stati finali $q \in F$
3. Rimpiazzo transizioni multiple tra due stati con l'unione delle etichette
4. Aggiungo transizioni etichettate con $\emptyset$ tra stati non collegati da transizioni

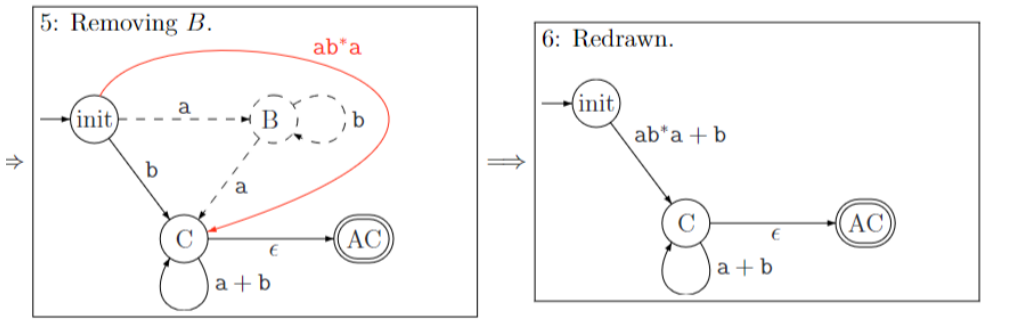


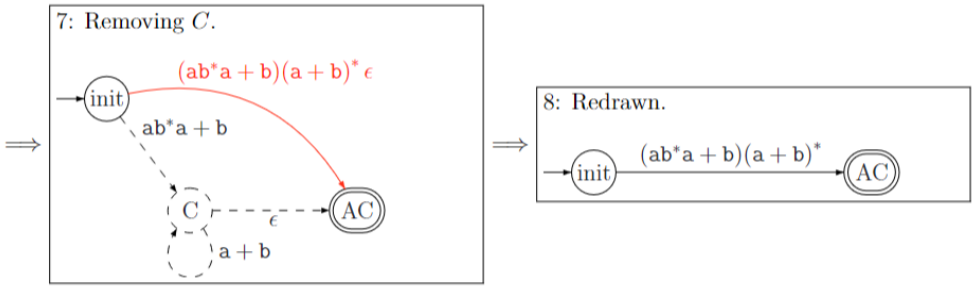


2.1 Example: From NFA to regex in 8 easy figures



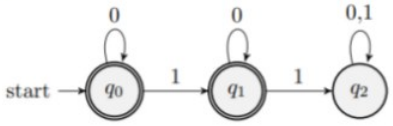




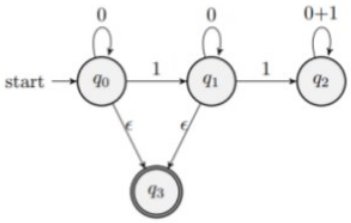


Thus, this automata is equivalent to the regular expression $(ab^*a + b)(a + b)^*$.

2. Stringhe binarie che non comprendono la stringa 101

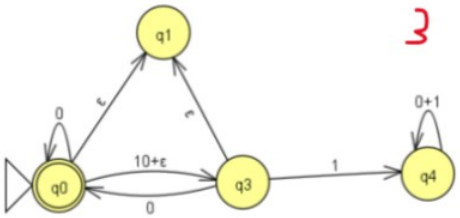


1



2

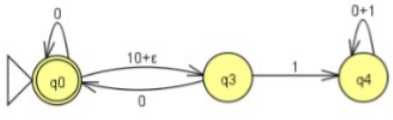
Passaggi:
Elimino q2 q0-q3 10+ε



3

4

Elimino q1: q0-q3-q4 (10+ε)



5

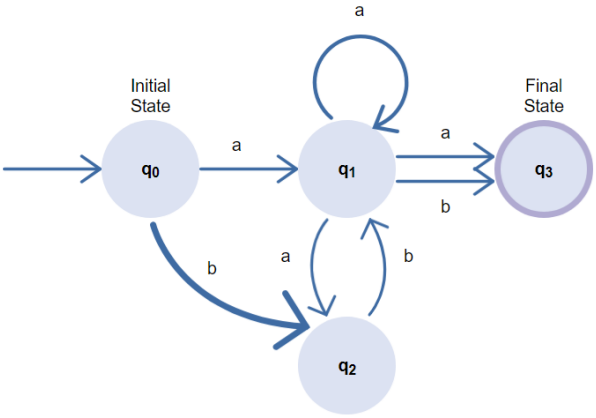
Elimino q3: q0-q4 (0*(10+ε))*1

Elimino q4: (0*(10+ε))*1+(0+1)*

ER: 0*(ε+(10+ε))1(0+1)*

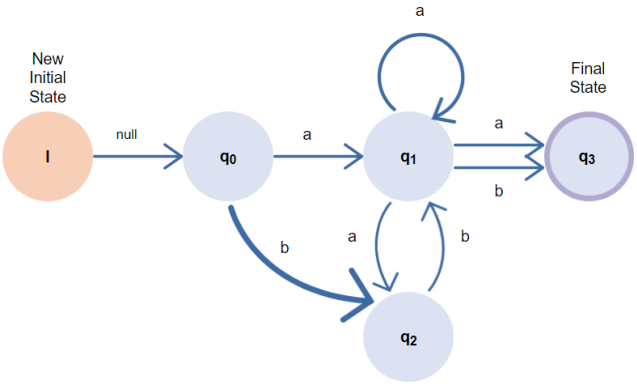
6

Let's consider the following finite automaton:



Finite automaton

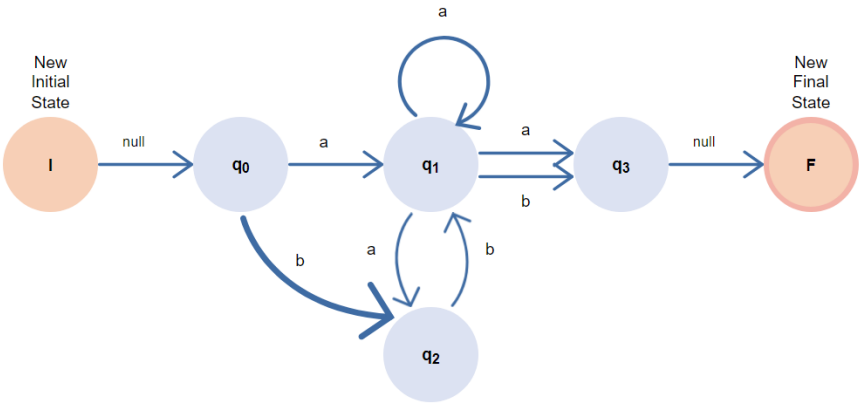
Add a new initial state, I. Make a null transition from state I to state q_0.



Adding state I

Step 2: Add a final state

Add a new final state, F. Make a null transition from state q_3 to state F.



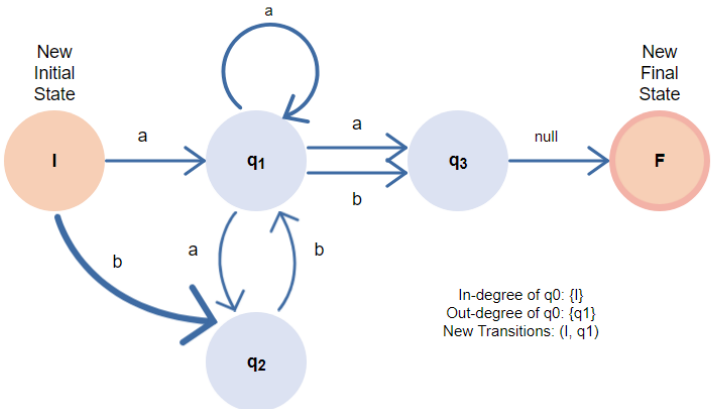
Adding state F

Step 3: State elimination

Perform the elimination of states other than I and F.

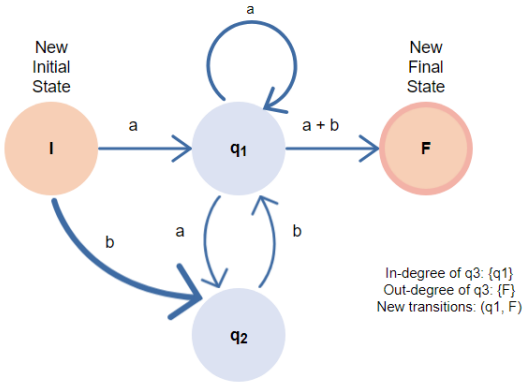
Step 3.1

Eliminate state q_0.



Step 3.2

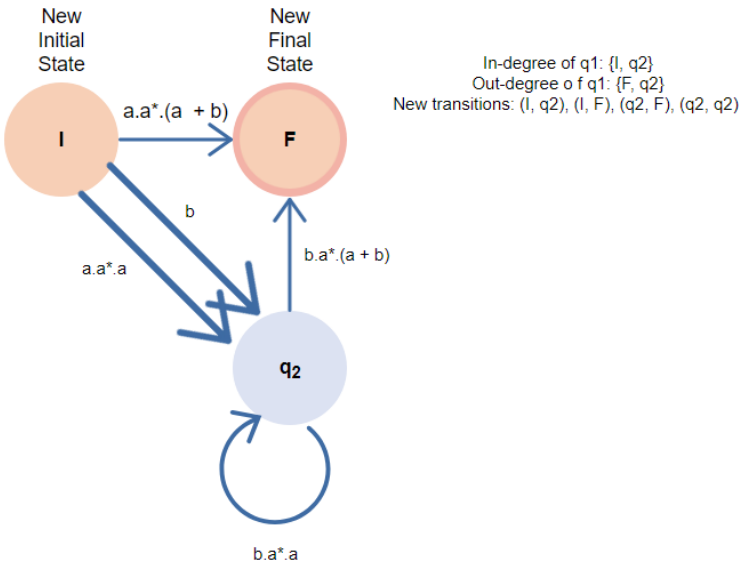
Eliminate state q_3. Concatenate transitions from state q_3 to state F as per the basic rules of writing regular expressions.



Eliminating q3

Step 3.3

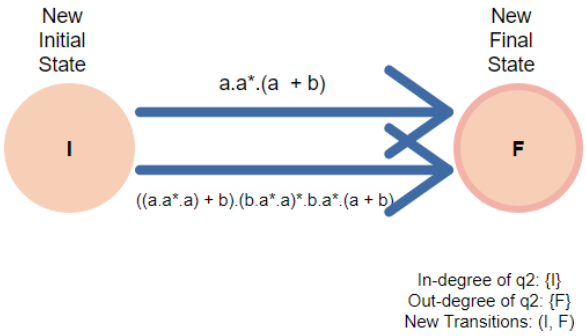
Eliminate q_1. Check for the in-degree and out-degree of state q_1. Write the regular expressions for the new transitions acquired after removing state q_1.



Eliminating q1

Step 3.4

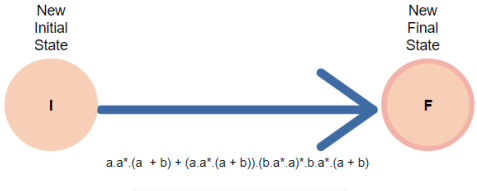
Eliminate state q_2.



Eliminating q2

Step 3.5

Put it all together.



The final regular expression

Result

The resultant regular expression for the given finite automaton is as follows:

$$a.a*.(a+b)+((a.a*.a)+b).(b.a*.a)*.b.a*.(a+b).$$