Ch. 2. Finite Automata & Regular Expressions

2.1. Introduction.

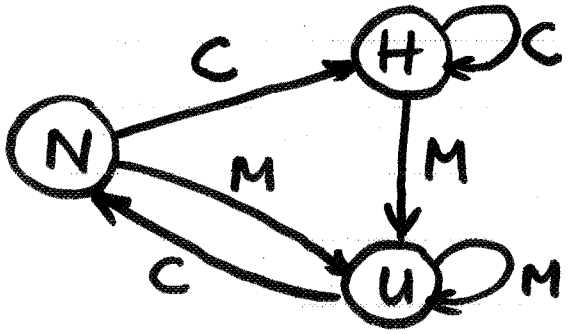
Finite automata are the simplest model of computation: comput. very limited, yet very useful in many diff. areas incl. compiler construction, switching theory, biology, comm. protocols ....

Q. How to model systems as finite-state machines?

Ex: How to describe behavior of a child?

There are 3 states: happy (H), neutral (N), unhappy (U).

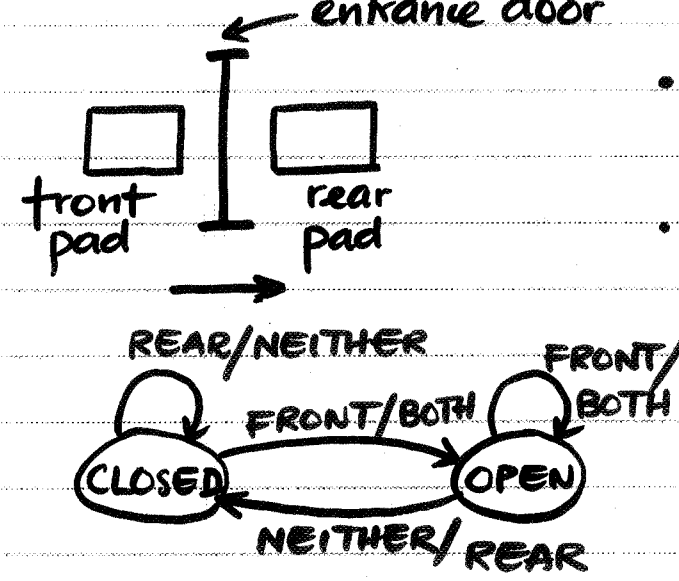
Two types of inputs: candy (C) and medicine (M)



↓ inputs	C	M
N	H	U
H	H	U
U	N	U

↑ states

Ex: Design of the controller for an autom. door



- door is in one of two states: OPEN, CLOSED
- controller senses:
  - front pad occupied : FRONT
  - rear _____ : REAR
  - both _____ : BOTH
  - none _____ : NEITHER

	FRONT	REAR	BOTH	NEITHER
CLOSED	OPEN	CLOSED	OPEN	CLOSED
OPEN	OPEN	CLOSED	OPEN	CLOSED

Q. What is an initial state?

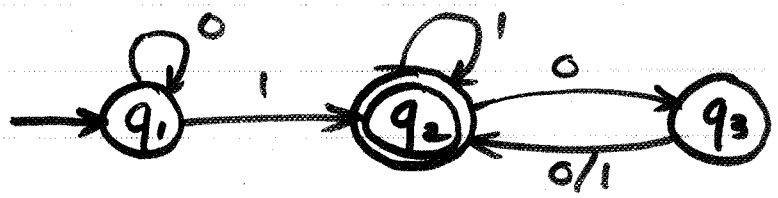
Q. What are accept/final states?

Q. What are input sequences leading from init. to an accept state?

2.2. Basic Definitions

Consider e.g. the following transition diagram:

M_1:



- states are q_1, q_2, q_3 : $\{q_1, q_2, q_3\} = Q$
- input symbols are 0, 1 : $\{0,1\} = \Sigma$
- labeled edges are transitions
- start / initial state is q_1
- accept / final state (s) is q_2

The set of transitions constitutes the transition function denoted by δ: if q is current state and a is current input, $\delta(q,a)$ is next state. That is

$$\delta: Q \times \Sigma \longrightarrow Q$$

Def. A finite automaton (FA) is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F) \quad \text{where}$$

(1) Σ is input alphabet

(2) Q is finite set of states

(3) $\delta : Q \times \Sigma \to Q$ is <u>transition fct</u>

(4) $q_0 \in Q$ is <u>start / initial state</u>

(5) $F \subseteq Q$ is set of <u>final states</u>

<u>Ex.</u> Consider FA M_1 :

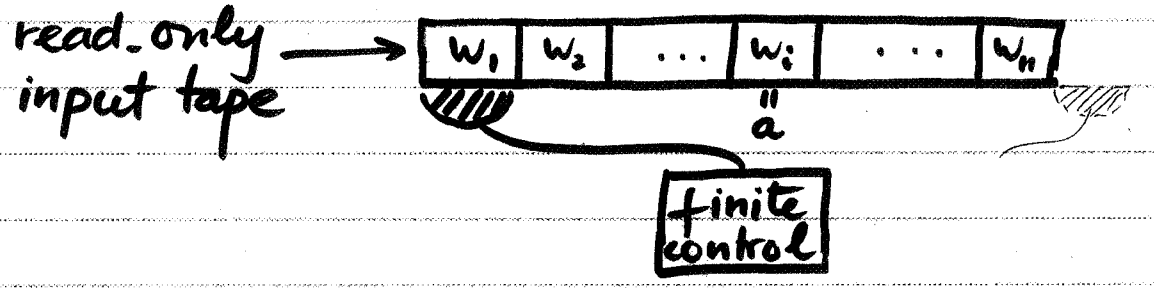
$Q = \{q_1, q_2, q_3\}$ $\quad \Sigma = \{0, 1\}$

q_1 is start state

$F = \{q_2\}$

δ	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

<u>Physical representation</u>



M scanning $w_i = a$ in state q moves head right and enters state q'

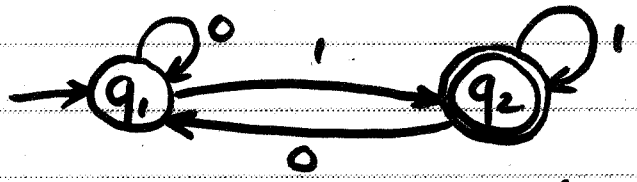
$\iff \delta(q, a) = q'$

L(M) denotes the language accepted by M.

$w \in \Sigma^*$ is accepted if M starting with q_0 enters some $q_f \in F$ after proc. w.

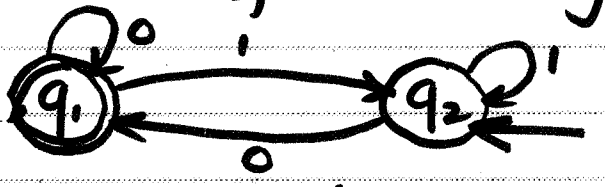
Ex. $L(M_1)$ = set of bin. strings s.t. the number of 0's following the last 1 is even.

Ex.



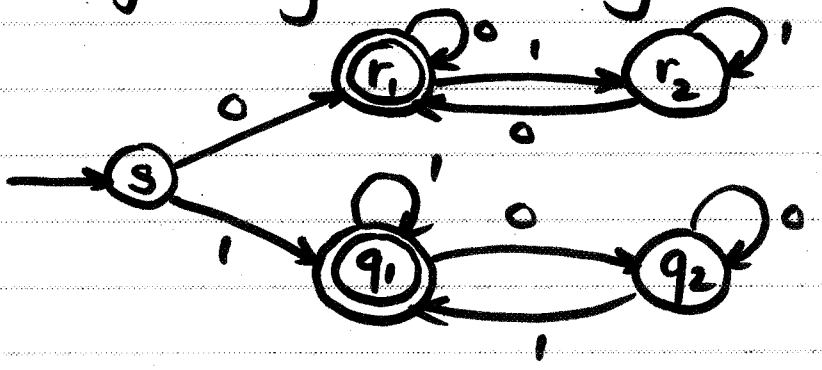
L(M) = set of bin. strings ending in 1

Ex.

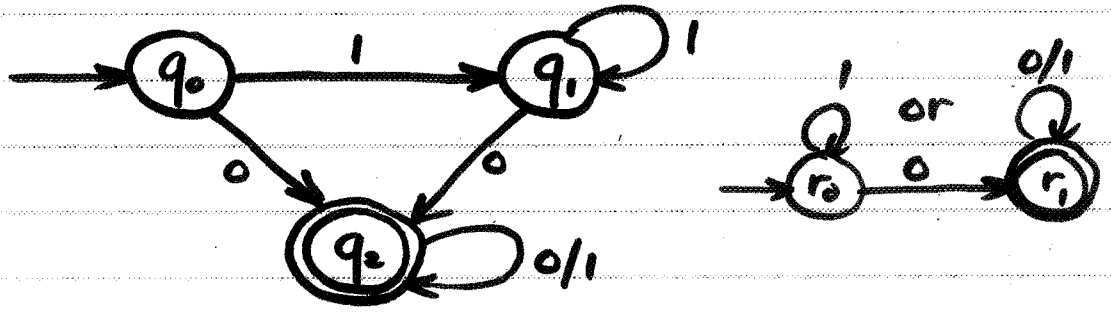


L(M) = set of bin. strings ending in 0

Ex An FA to accept bin. strings beginning & ending with same symb.

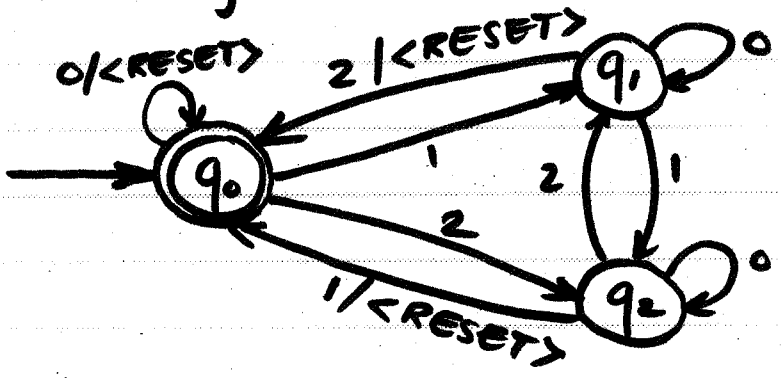


Ex



L(M) = set of bin. strings containing at least a 0
(or having 0 as substring)

Ex. M reads symbols from $\Sigma = \{0, 1, 2, \langle RESET \rangle\}$ and counts modulo 3. (addition)



Ex. Counting modulo k, $k \geq 1$ is fixed. (adding)

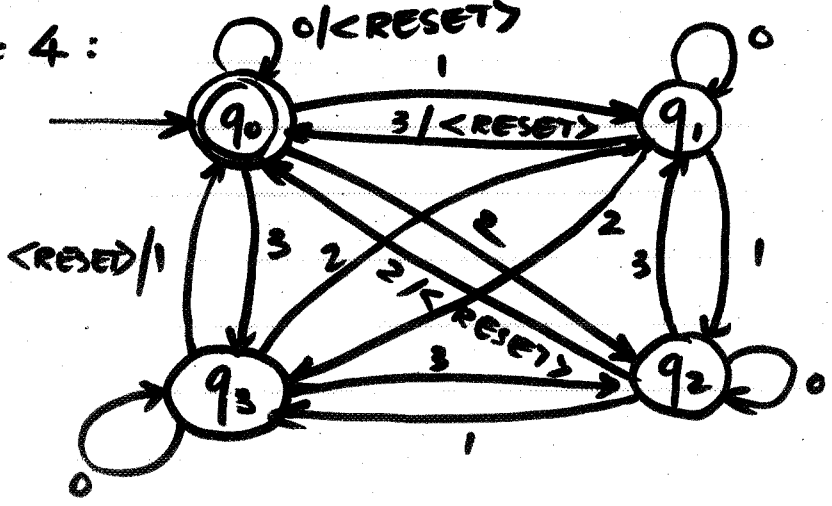
$$\Sigma = \{0, 1, 2, \ldots, k-1, \langle RESET \rangle\}$$

$$Q = \{q_0, q_1, \ldots, q_{k-1}\}$$

For $0 \leq j \leq k-1$:

$$\delta(q_j, 0) = q_j$$
$$\delta(q_j, 1) = q_{j+1 \bmod k}$$
$$\vdots$$
$$\delta(q_j, k-1) = q_{j+k-1 \bmod k}$$
$$\delta(q_j, \langle RESET \rangle) = q_0 .$$

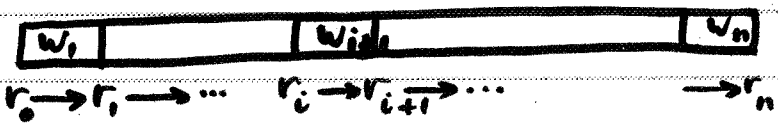
For $k = 4$:



Formal def. of computations.

$M = (Q, \Sigma, \delta, q_0, F)$

input string $w = w_1 \dots w_n$, $w_1, \dots, w_n \in \Sigma$



M accepts w if there exist states $r_0, r_1, \dots, r_n$ s.t.

(1) $r_0 = q_0$ (2) $r_n \in F$

(3) $\delta(r_i, w_{i+1}) = r_{i+1}$ $\forall i = 0, 1, \dots, n-1$

A language $L \subseteq \Sigma^*$ is regular if $L = L(M)$ for some FA M

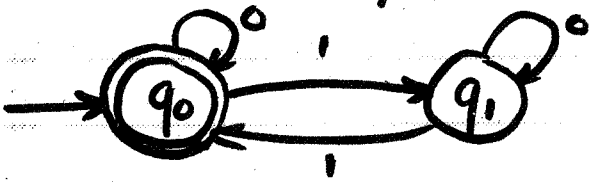
$L(M) = \{ w \in \Sigma^* \mid w \text{ is accepted by } M \}$.

Designing FAs

Assuming you have a finite amount of memory, try to come up with a strategy to accept the given language.

Ex. An FA to accept bin. strings containing an even number of 1's

Strategy: Count modulo 2 the number of 1's read (0 = even, 1 = odd)



$q_0 \hat{=}$ 0 mod 2

$q_1 \hat{=}$ 1 mod 2

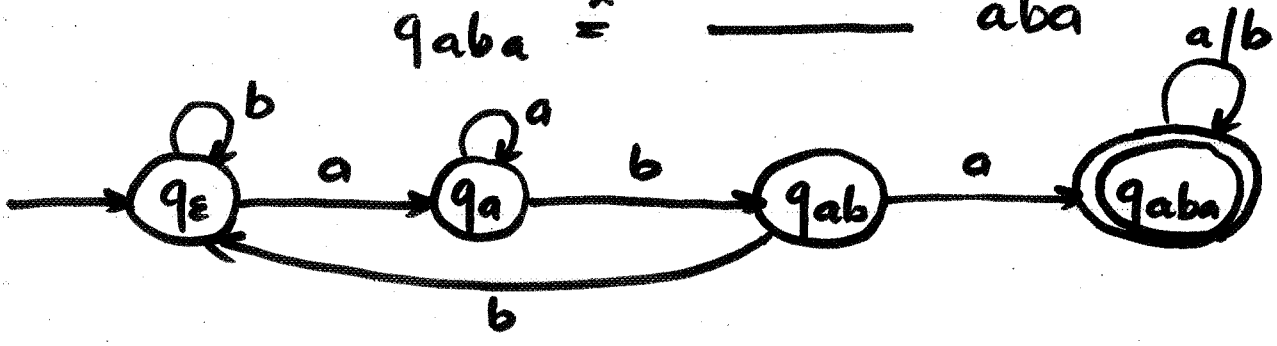
Ex An FA to accept strings over {a,b} containing aba as substring

Strategy: Keep searching for pattern aba:

$q_a \hat{=}$ just seen a

$q_{ab} \hat{=}$ ——— ab

$q_{aba} \hat{=}$ ——— aba



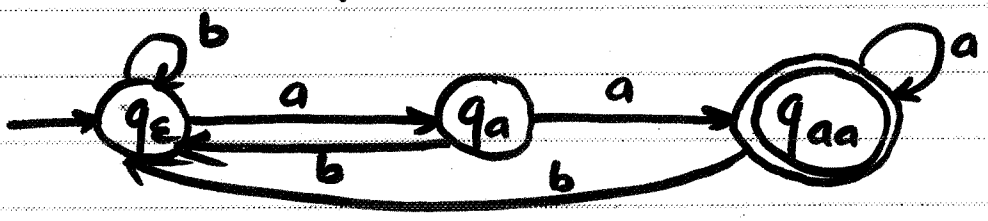
Ex. An FA to accept strings over $\{a,b\}$ ending in aa

Strategy. As before

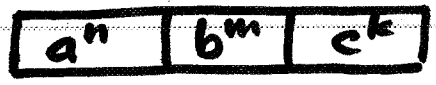
$q_a \triangleq$ just seen a

$q_{aa} \triangleq$ just seen aa

So at q_{aa} if we read a, it's ok; otherwise if we read b, start over.



Ex. An FA to accept $\{a^n b^m c^k \mid n,m,k \geq 0\}$

i.e., strings of form 

where a block could be empty.

Strategy. As before

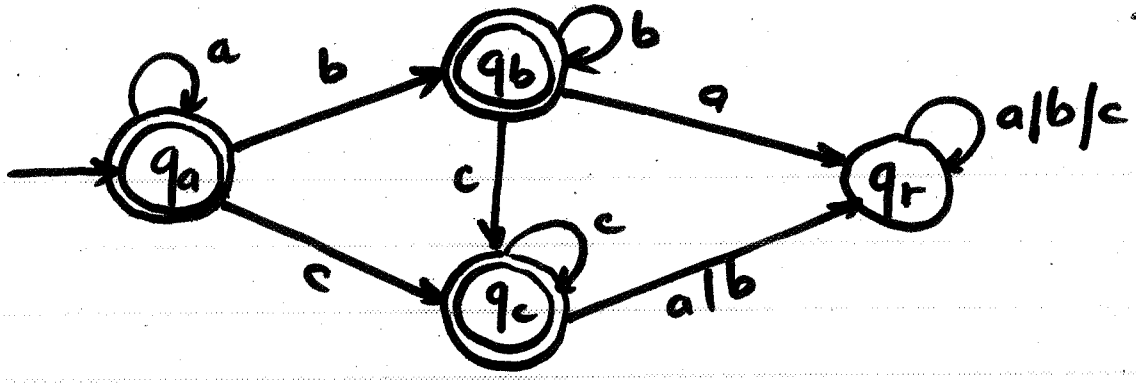
$q_a \triangleq$ we're within block of a's processing a's

$q_b \triangleq$ within block of b's

$q_c \triangleq$ within 3rd block of c's

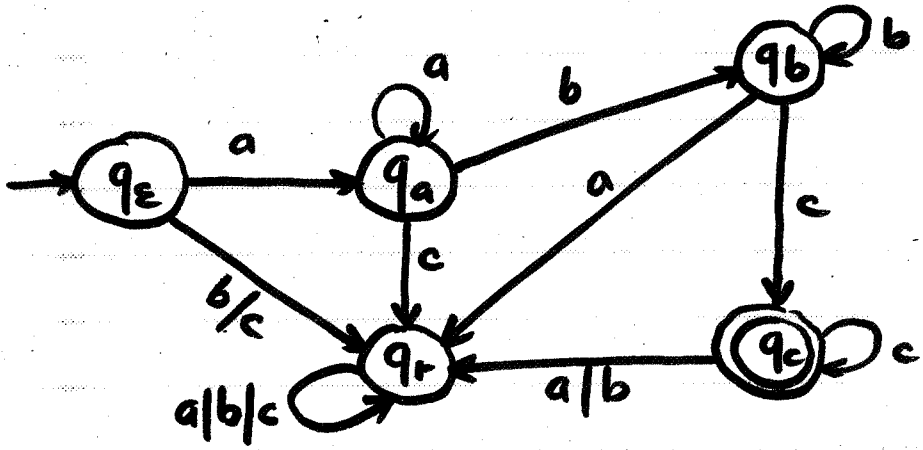
$q_r \triangleq$ rejecting state

If we're already in q_c and a is read, then reject.



Ex: What if $L = \{ a^n b^m c^k \mid n, m, k > 0 \}$?

Strategy: As before, but we have to make sure that the blocks are non-empty. So we accept only if we've read at least an a, a b and a c.



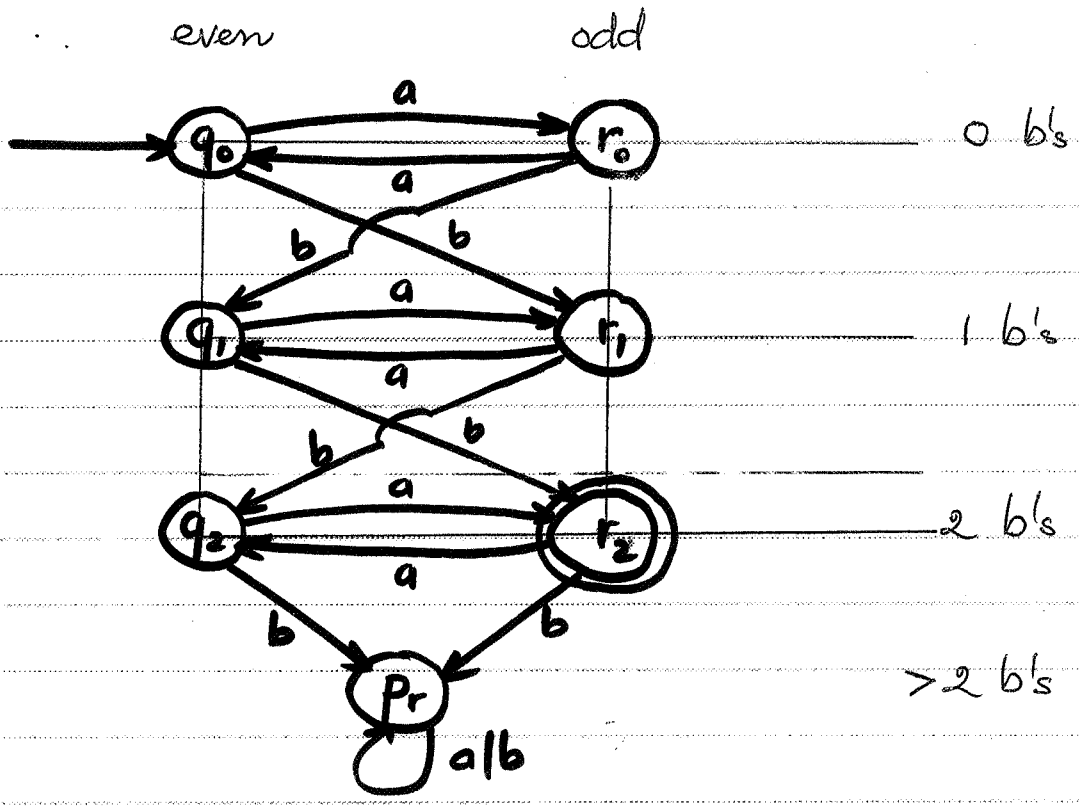
Note that q_a, q_b are now rejecting

Ex An FA to accept odd-length strings over {a,b} containing 2 b's

Strategy: We have to maintain 2 types of information

(1) Count modulo 2 the number of symbols read so far, and

(2) Count the number of b's read (0 or 1 or 2 or more)



$q \triangleq$ current length is even

$r \triangleq$ ——— odd

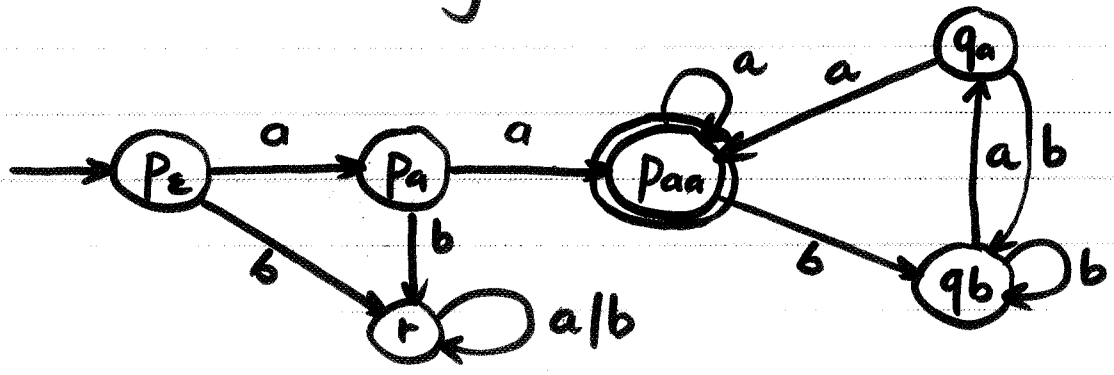
$o \triangleq$ number of b's is 0

$1 \triangleq$ ——— 1

$2 \triangleq$ ——— 2

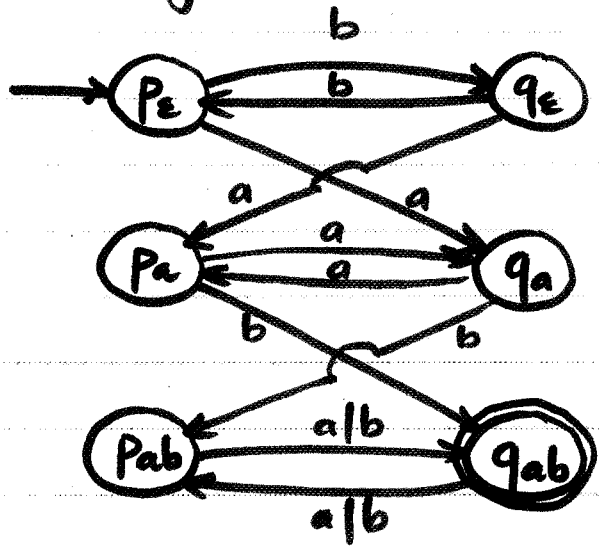
$P_r \triangleq$ ——— > 2.

Ex: An FA to accept strings beginning and ending with aa



Ex: An FA to accept odd_length strings over {a,b} containing ab as substring

Strategy: count length modulo 2 and at the same time search for substring ab



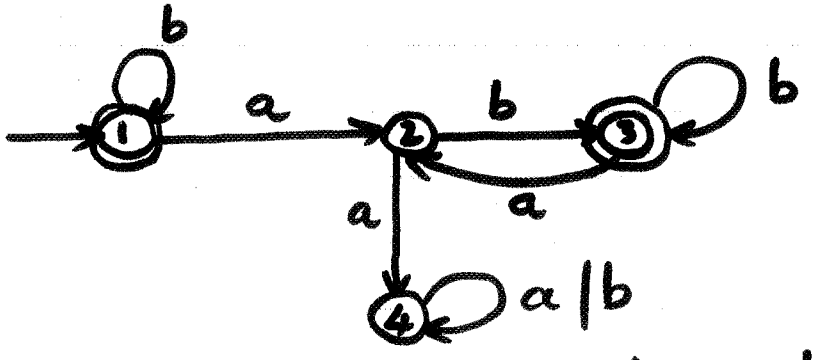
has not found anything

found a

found ab as substring

even_length ↑ odd_length ↑

Ex: An FA to accept strings over {a,b} s.t. every occurence of "a" is followed by a "b."



1 and 3 are equiv. and can be merged.

The Regular Operations.

Let $A, B \subseteq \Sigma^*$ be languages.

The regular operations on languages are:

- Union: $A \cup B = \{ x \in \Sigma^* \mid x \in A \vee x \in B \}$
- Concatenation: $AB = \{ xy \mid x \in A, y \in B \}$
- Kleene closure:

$$A^* = \bigcup_{n=0}^{\infty} A^n$$

$$= \{ x_1 \dots x_n \mid n \geq 0, x_1, \dots, x_n \in A \}$$

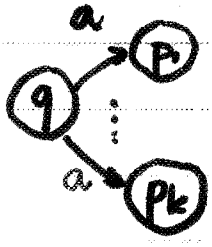
2.3. Nondeterministic Finite Automata

In an FA $M = (Q, \Sigma, \delta, q_0, F)$ given $q \in Q$, $a \in \Sigma$, the next state is uniquely determined : $\delta(q,a)$ since $\delta : Q \times \Sigma \to Q$ is a function.

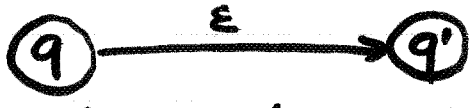
Thus, M is deterministic.

We call it a deterministic finite automaton (DFA for short)

In a nondeterministic finite automaton (NFA) N, given $q \in Q$ and $a \in \Sigma$ there are several choices for the next state, i.e., $\delta(q,a)$ is a finite subset of Q :

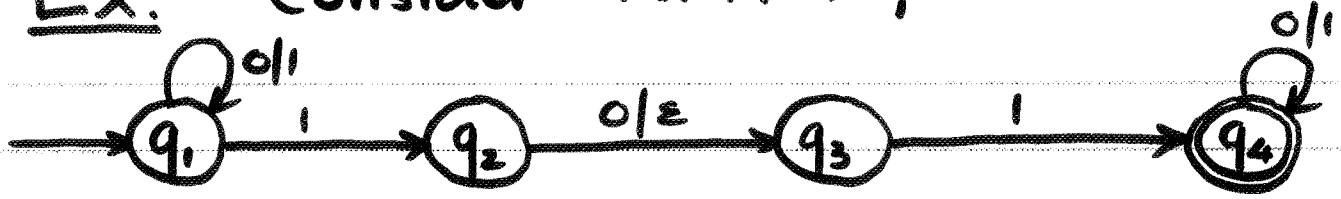


$$\delta(q,a) = \{ p_1, \ldots, p_k \} \subseteq Q$$

In an NFA we also allow ε-transitions : 

i.e., N can make a transition from q to q' without consuming any input symbol.

Ex. Consider NFA N_1:



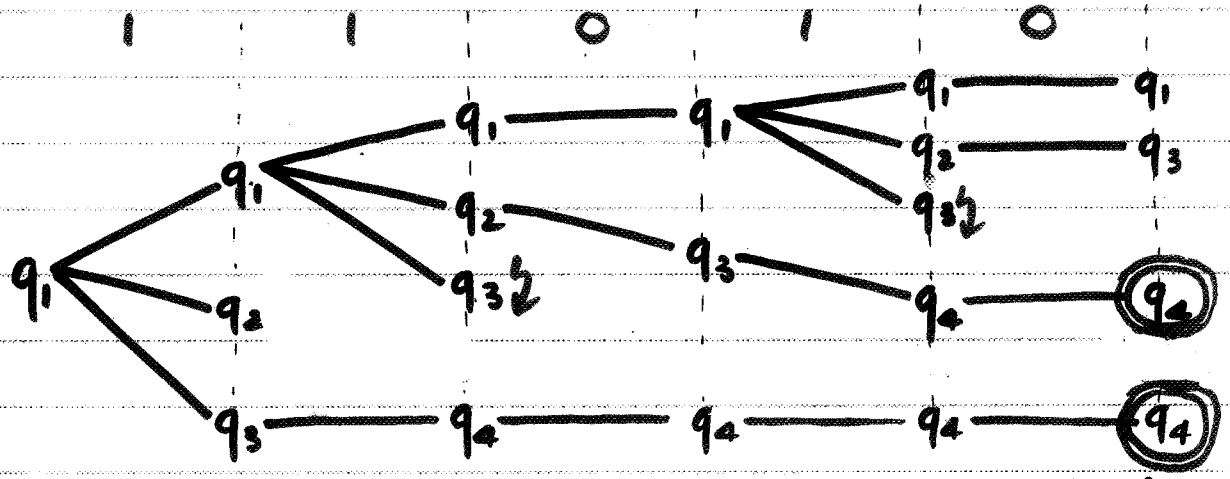
$\delta(q_1, 0) = \{q_1\}$ $\quad \delta(q_2, \varepsilon) = \{q_3\}$

$\delta(q_1, 1) = \{q_1, q_2\}$ $\quad \delta(q_3, 0) = \emptyset$

Q. What is a computation on an NFA?

Consider e.g. input string $w = 11010$

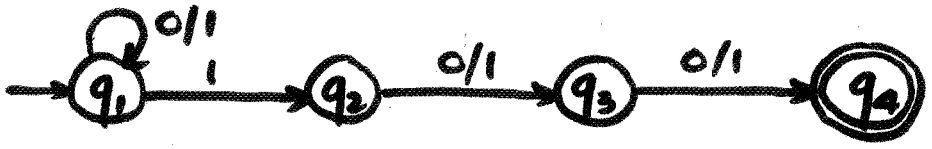
Possible computations are:



There are 3 accepting computations.

If there is an accepting computation of NFA N on input w, w is declared accepted by N.

Thus N_1 accepts bin. strings containing 11 or 101 as substring

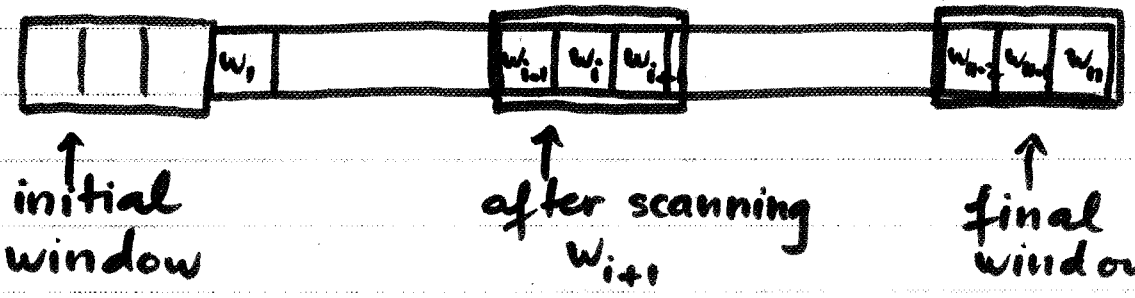
Ex: N_2 

Q. What does N_2 accept?

A. Bin. strings in which the 3rd symb. from the right is 1.

Q. Can we construct an equiv. DFA (to accept the same language)?

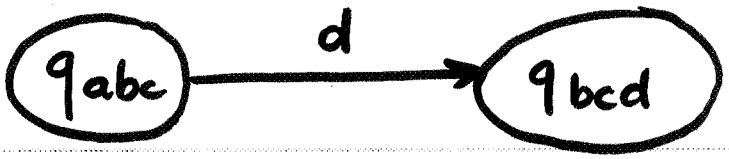
Strategy: While processing input string maintain a window of size 3 that contains the 3 most recently read input symbols:



Thus, initial window contains ε and final window contains $w_{n-2}w_{n-1}w_n$.

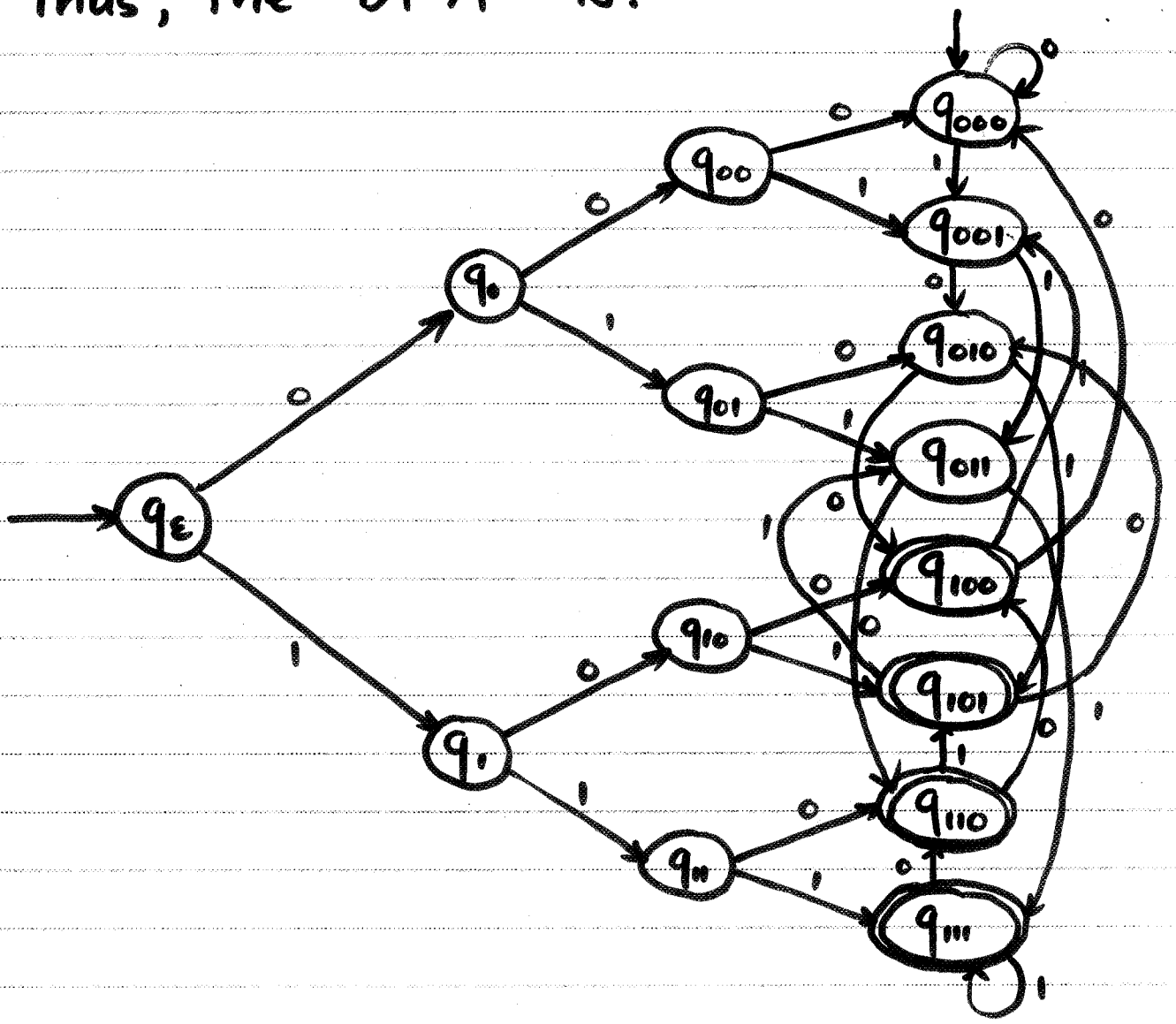
Construction of DFA:

States are q_v where v is a string of length ≤ 3 corresponding to window content: q_ε is init. state, q_{1ab}, $a,b \in \{0,1\}$ are final states.



, a, b, c, d $\in \{0,1\}$

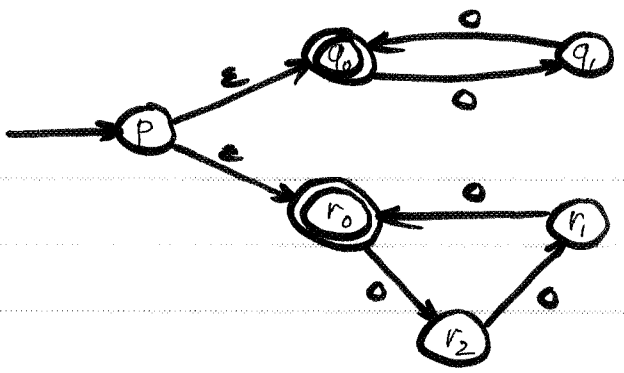
Thus, the DFA is:



Q. Can we construct a smaller DFA?

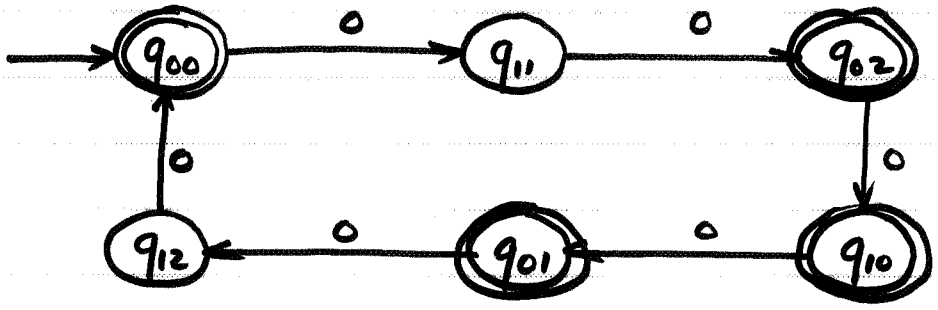
A. Yes. We can make q_{000} initial state and delete q_v, where $|v| \leq 2$. The resulting DFA accept the same language.

Ex.



$L(M) = \{ 0^k \mid k \text{ is multiple of 2 or 3} \}$

Equiv. DFA is:

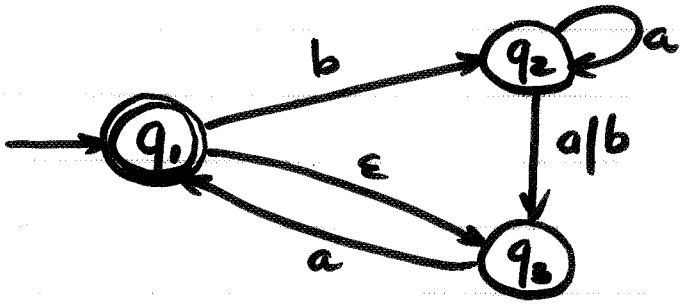


States are q_{ij} where:

- i is used to count modulo 2
- j ——————————— 3

Ex.

M:



Observe: $\delta(q_2, a) = \{q_2, q_3\}$

$\delta(q_1, a) = \phi$; $\delta(q_1, \varepsilon) = \{q_3\}$

Letting $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$,

$\delta: Q \times \Sigma_\varepsilon \longrightarrow 2^Q$ (or $\mathcal{P}(Q)$)

i.e., $\delta(q, a) = \{p_1, \ldots, p_k\} \subseteq Q$

for $q \in Q$, $a \in \Sigma_\varepsilon$

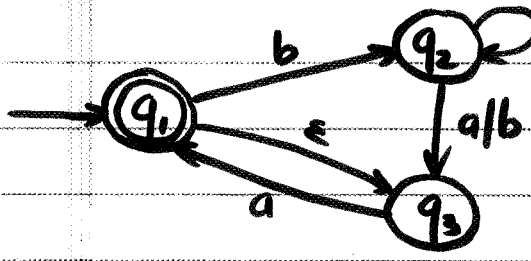
Def. An NFA is a 5-tuple

$$M = (Q, \Sigma, \delta, q_0, F)$$

where:

- Q is finite set of states
- Σ is input alphabet
- $q_0 \in Q$ is initial state
- $F \subseteq Q$ is set of final states
- $\delta : Q \times \Sigma_\varepsilon \to 2^Q$ is transition fctn.

Ex. Consider NFA M above



	a	b	ε
q_1	$\emptyset$	$\{q_2\}$	$\{q_3\}$
q_2	$\{q_2, q_3\}$	$\{q_3\}$	$\emptyset$
q_3	$\{q_1\}$	$\emptyset$	$\emptyset$

□

An input string $w \in \Sigma^*$ is accepted by M if w can be written as

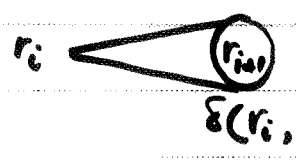
$$w = y_1 y_2 \cdots y_m \,, \quad y_i \in \Sigma_\varepsilon$$

s.t. there exist states $r_0, r_1, \ldots, r_m \in Q$ with

(1) $r_0 = q_0$ (2) $r_m \in F$

(3) $r_{i+1} \in \delta(r_i, y_{i+1})$ $\forall i = 0, \ldots, m-1$

$$r_0 \xrightarrow{y_1} r_1 \xrightarrow{y_2} \cdots r_i \xrightarrow{y_{i+1}} r_{i+1} \cdots \xrightarrow{y_m} r_m$$



$L(M)$ = language accepted by M

Equivalence of NFAs and DFAs

Theorem. $L = L(M)$ for some DFA M $\iff$ $L = L(N)$ for some NFA N.

Pf. "$\Rightarrow$": obvious since every DFA is an NFA (that has a unique choice for next state and no ε-transitions)

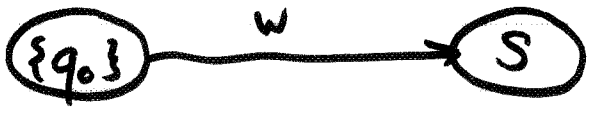
"$\Leftarrow$": Given $N = (Q, \Sigma, \delta, q_0, F)$ we want to construct an equiv. DFA.

Idea. Assume for the time being that N has no ε-transitions. We construct an equiv. DFA

$$M' = (Q', \Sigma, \delta', q_0', F') \quad \text{s.t.}$$

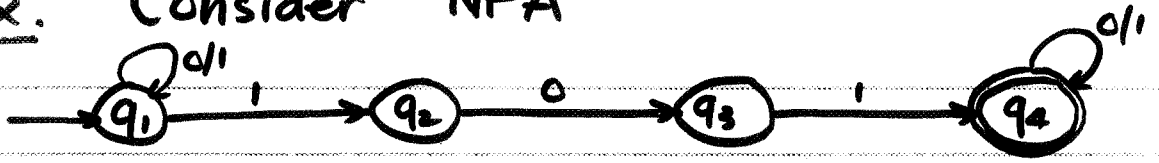
on any input $w \in \Sigma^*$, M' stores in its finite control all states reached in N after processing w, i.e.,

Then in M':

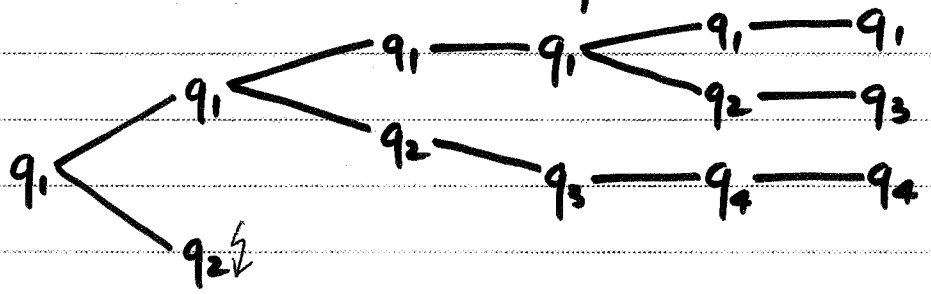


i.e., S is a state in M'.

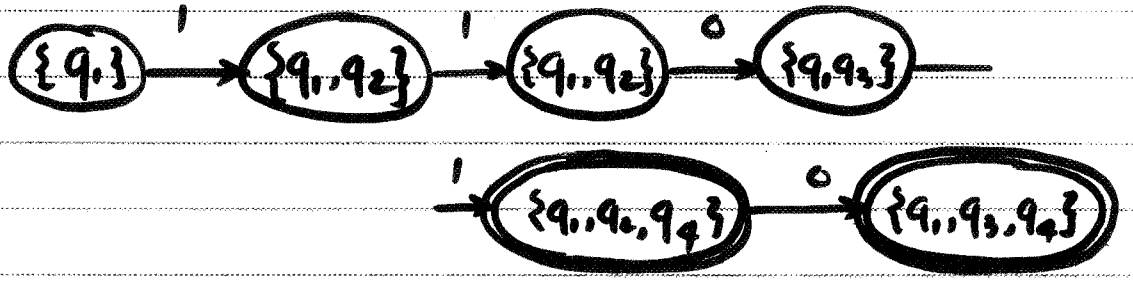
Ex. Consider NFA



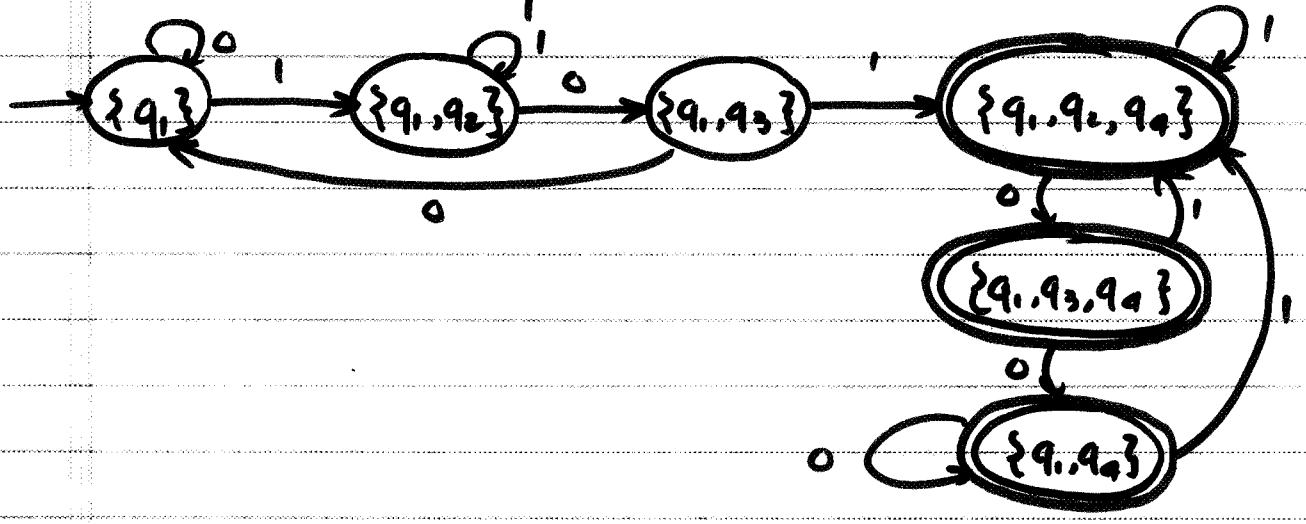
and input 11010. The computations are:



So the computation of equiv. DFA on 11010 is



Thus the equiv. DFA is



for example $\delta'(\{q_1\}, 1) = \{q_1, q_2\}$

$\delta'(\{q_1, q_2\}, 0) = \{q_1, q_3\}$

$\delta'(\{q_1, q_3\}, 1) = \{q_1, q_2, q_4\}$

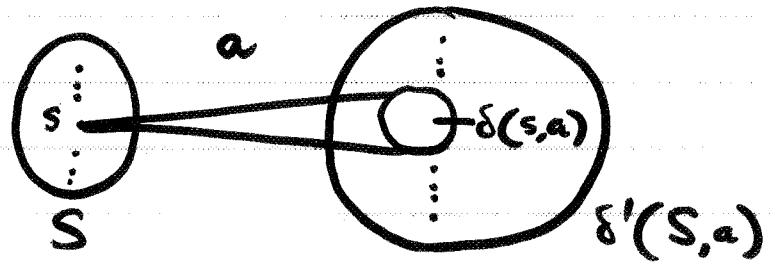
Construction of $M' = (Q', \Sigma, \delta', q_0', F')$

(1) $Q' = 2^Q$

(2) $q_0' = \{q_0\}$

(3) $F' = \{ S \in Q' \mid S \cap F \neq \emptyset \}$

(4) $\delta'(S, a) := \bigcup_{s \in S} \delta(s, a)$



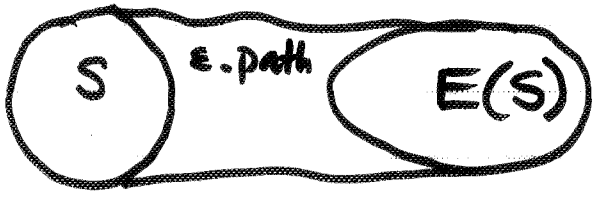
Taking ε-transitions into account

For $S \in Q'$ define the ε-closure of S by:

$E(S) := \{ q \in Q \mid q$ can be reached from some $s \in S$ by ≥ 0 ε-transitions$\}$

$= \{ q \in Q \mid q$ can be reached from some $s \in S$ without processing any input symbol $\}$

$= \{ q \in Q \mid$ there is a path labeled ε from some $s \in S$ to $q \}$



Define

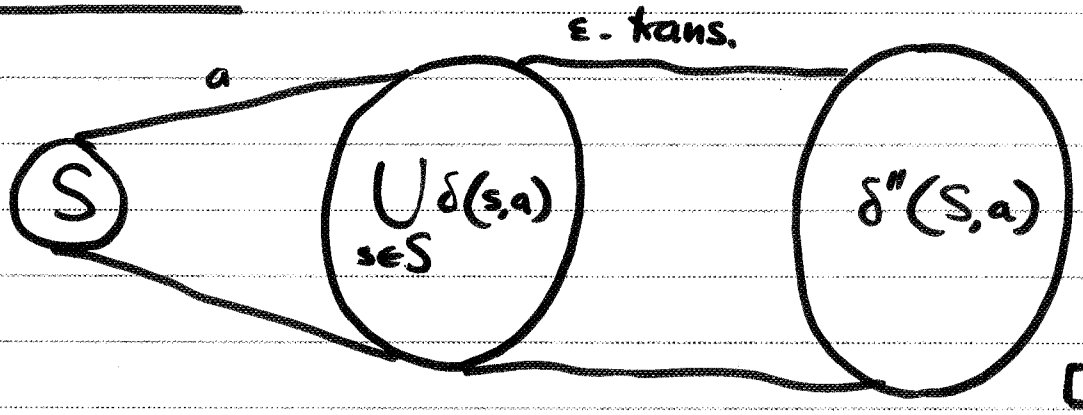
- $q_0'' := E(\{q_0\})$
- $\delta''(S,a) := E(\delta'(S,a))$

$$= \bigcup_{s \in S} E(\delta(s,a))$$

Then the equiv. DFA M is

$$M = (Q', \Sigma, \delta'', q_0'', F')$$

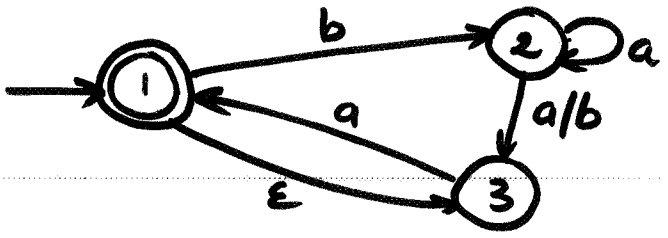
Illustration:



□

This construction is called the subset construction

Ex. N

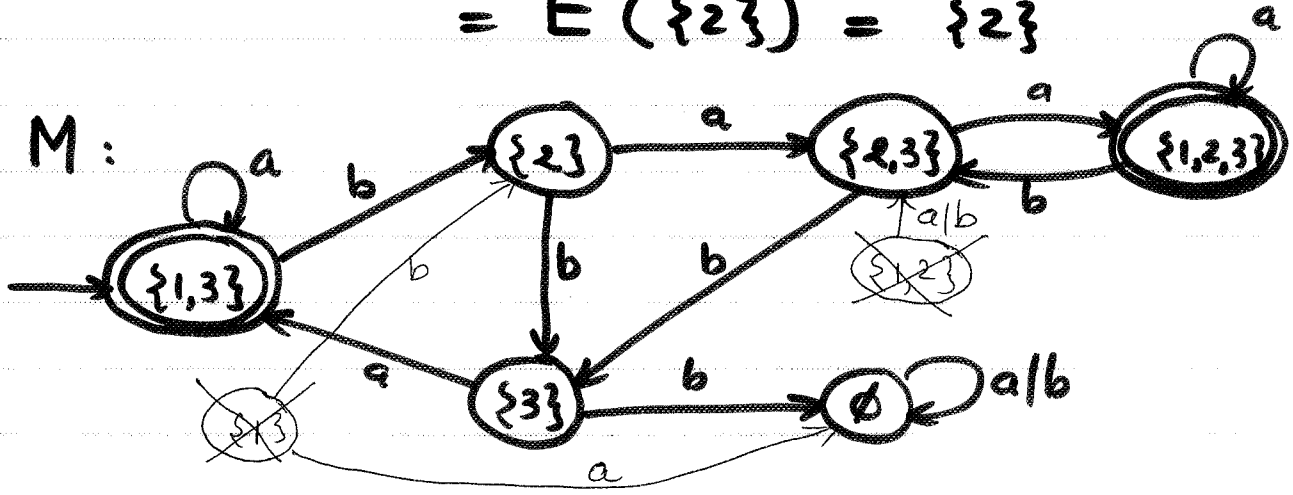


$q_0'' = E(\{1\}) = \{1,3\}$

$\delta''(\{1,3\}, a) = E(\delta(1,a) \cup \delta(3,a))$
$= E(\{1\}) = \{1,3\}$

$\delta''(\{1,3\}, b) = E(\delta(1,b) \cup \delta(3,b))$
$= E(\{2\}) = \{2\}$

M:



$\delta''(\{2\}, a) = E(\delta(2,a)) = E(\{2,3\}) = \{2,3\}$

$\delta''(\{2\}, b) = E(\delta(2,b)) = E(\{3\}) = \{3\}$

$\delta''(\{3\}, a) = E(\delta(3,a)) = E(\{1\}) = \{1,3\}$

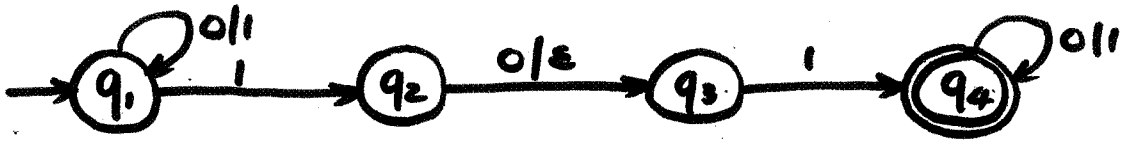
$\delta''(\{3\}, b) = E(\delta(3,b)) = E(\phi) = \phi$

$\delta''(\{2,3\}, a) = E(\delta(2,a) \cup \delta(3,a))$
$= E(\{1,2,3\}) = \{1,2,3\}$

$\vdots$

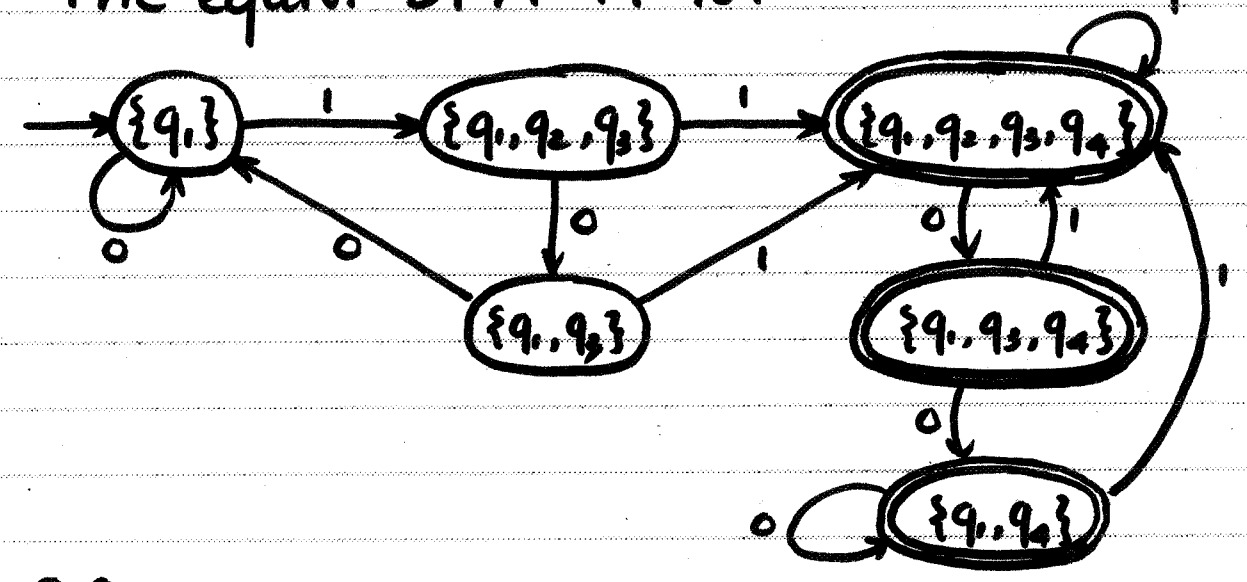
Note that $\{1\}$, $\{1,2\}$ cannot be reached from $\{1,3\}$, and hence don't appear in M.

Ex:



N:

The equiv. DFA M is:



e.g.

$$\delta''(\{q_1\},0) = E(\delta(q_1,0)) = E(\{q_1\}) = \{q_1\}$$

$$\delta''(\{q_1\},1) = E(\delta(q_1,1)) = E(\{q_1,q_2\}) = \{q_1,q_2,q_3\}$$

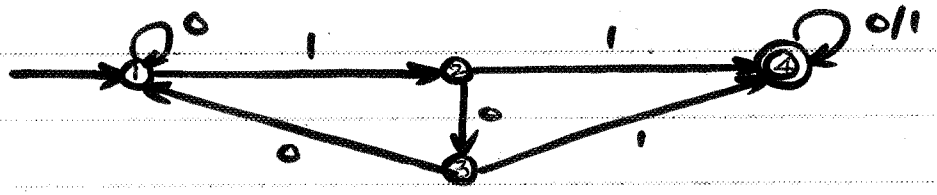
$$\delta''(\{q_1,q_2,q_3\},0) = E(\delta(q_1,0) \cup \delta(q_2,0) \cup \delta(q_3,0))$$

$$= E(\{q_1,q_3\}) = \{q_1,q_3\}$$

$$\delta''(\{q_1,q_2,q_3\},1) = E(\delta(q_1,1) \cup \delta(q_2,1) \cup \delta(q_3,1))$$

$$= E(\{q_1,q_2,q_4\}) = \{q_1,q_2,q_3,q_4\}$$

A smaller DFA:



(The 3 final states of M are equiv. & can be merged)

Cor: L is regular $\iff$ L = L(N) for some NFA N

2.4. Regular Expressions.

Def. Let Σ be an alphabet. The regular expressions over Σ and the sets they denote are def. recursively as follows:

(1) **Basis.** ε is a reg. expr. denoting $L(\varepsilon) = \{\varepsilon\}$
ϕ ——— $L(\phi) = \phi$
$\forall a \in \Sigma$: a ——— $L(a) = \{a\}$

(2) **Rec. Step.** If α, β are reg. expr. denoting $L(\alpha)$ and $L(\beta)$, then
$(\alpha + \beta)$ is a reg. expr. denoting $L(\alpha) \cup L(\beta)$
$(\alpha\beta)$ ——— $L(\alpha)\,L(\beta)$
(α^*) ——— $(L(\alpha))^*$

(3) Nothing else is a reg. expr. over Σ.

Note. Priority of operators in decreasing order

$$( ) \quad * \quad \cdot \quad + \Longrightarrow$$

Ex. $\Sigma = \{0, 1\}$

(1) $0^*10^* = (((0^*)1)(0^*))$
denotes set of bin. strings containing exactly one 1

(2) $(0+1)^*1(0+1)^*$ denotes set of bin. strings containing at least a 1 (or containing 1 as a substring).

(3) $(0+1)^*010(0+1)^*$ denotes set of bin. strings containing 010 as substring

(4) $((0+1)(0+1))^*$ or $((0+1)^2)^*$
= set of even-length bin. strings

(5) $((0+1)^3)^*$ = set of bin. strings whose length is a multiple of 3.

(6) $(0+1)((0+1)^2)^*$ = set of odd-length bin. strings

(7) $0(0+1)^*1$ = set of bin. strings beginning with 0, ending with 1

(8) $0(0+1)^*0+0$ = set of bin. strings beginning & ending with 0

(9) $(0(0+1)^*0+0)+(1(0+1)^*1+1)$ = set of bin. strings beginning and ending with same symbol.

(10) $(0+1)(0+1)^*(0+1)$ = set of bin. strings of length ≥ 2 $= (0+1)^2(0+1)^*$

(11) $\phi^* = \varepsilon$

Some Regular Expressions Identities.

The following identities are useful in simplifying reg. expr.

(1) $\alpha + \phi = \phi + \alpha = \alpha$

(2) $\alpha . \varepsilon = \varepsilon . \alpha = \alpha$

(3) $\alpha . \phi = \phi . \alpha = \phi$

(4) $\alpha(\beta + \gamma) = \alpha\beta + \alpha\gamma$

(5) $(\alpha\beta)^*\alpha = \alpha(\beta\alpha)^*$

(6) $(\alpha+\beta)^* = (\alpha^*\beta^*)^*$

Proof of (6): We make use of $(\alpha^*)^* = \alpha^*$

"$\subseteq$":
$$\left.\begin{array}{l} \alpha \subseteq \alpha^*\beta^* \\ \beta \subseteq \alpha^*\beta^* \end{array}\right\} \Rightarrow \alpha+\beta \subseteq \alpha^*\beta^*$$

$$\Rightarrow (\alpha+\beta)^* \subseteq (\alpha^*\beta^*)^*$$

"$\supseteq$":
$$\left.\begin{array}{l} \alpha^* \subseteq (\alpha+\beta)^* \\ \beta^* \subseteq (\alpha+\beta)^* \end{array}\right\} \Rightarrow \alpha^*\beta^* \subseteq (\alpha+\beta)^*$$

$$\Rightarrow (\alpha^*\beta^*)^* \subseteq ((\alpha+\beta)^*)^* = (\alpha+\beta)^*$$

Equivalence of NFAs and REs

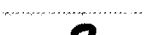
Goal: $L = L(N)$ for some NFA N
$\iff$ $L = L(\alpha)$ for some RE α.
(Kleene Theorem)

Thm. $L = L(\alpha)$ for some RE α
$\Rightarrow$ $L = L(N)$ for some NFA N.

Pf. Let α be an RE. Our goal is to construct for α an equiv. NFA N. N is constructed recursively from α.

(1) **Basis.**

$\alpha = \phi$: $N =$

$\alpha = \varepsilon$: $N =$ 

$\alpha = a, a \in \Sigma$: $N =$

(2) **Recursive Step.** $\alpha = \alpha_1\alpha_2$ or $\alpha_1 + \alpha_2$ or α_1^*
Assume inductively that we have constructed for α_1, α_2 equiv. NFAs

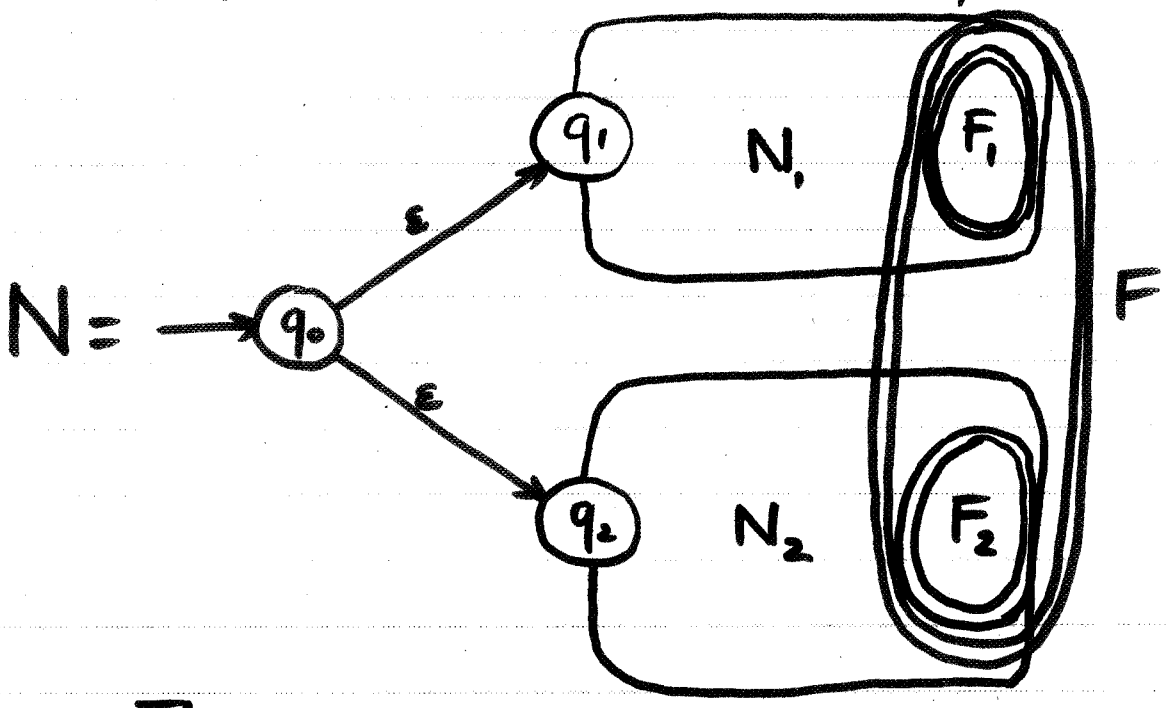
$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and

$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ resp.

w.l.o.g. $Q_1 \cap Q_2 = \phi$ and ϕ is not in α_1, α_2

Case 1. $\alpha = \alpha_1 + \alpha_2$

Then N is constructed as follows:



Thus,

$$N = (Q_1 \cup Q_2 \cup \{q_0\}, \Sigma, \delta, q_0, F)$$

where

- $F = F_1 \cup F_2$
- $$\delta(q,a) := \begin{cases} \{q_1, q_2\} & \text{if } q = q_0, a = \varepsilon \\ \delta_1(q,a) & \text{if } q \in Q_1 \\ \delta_2(q,a) & \text{if } q \in Q_2 \end{cases}$$

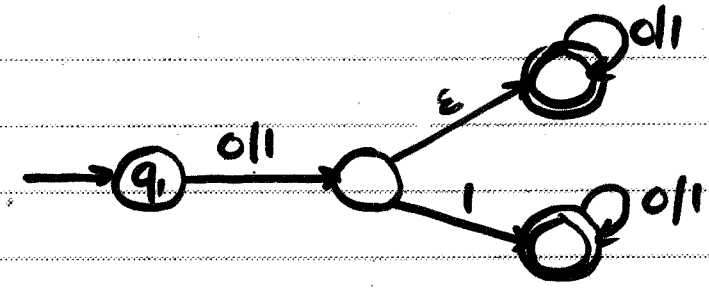
Then, $L(\alpha) = L(\alpha_1) \cup L(\alpha_2)$

$= L(N_1) \cup L(N_2)$

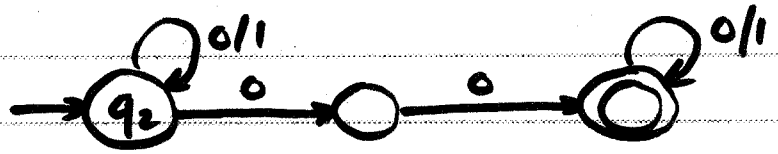
$= L(N)$

Ex:

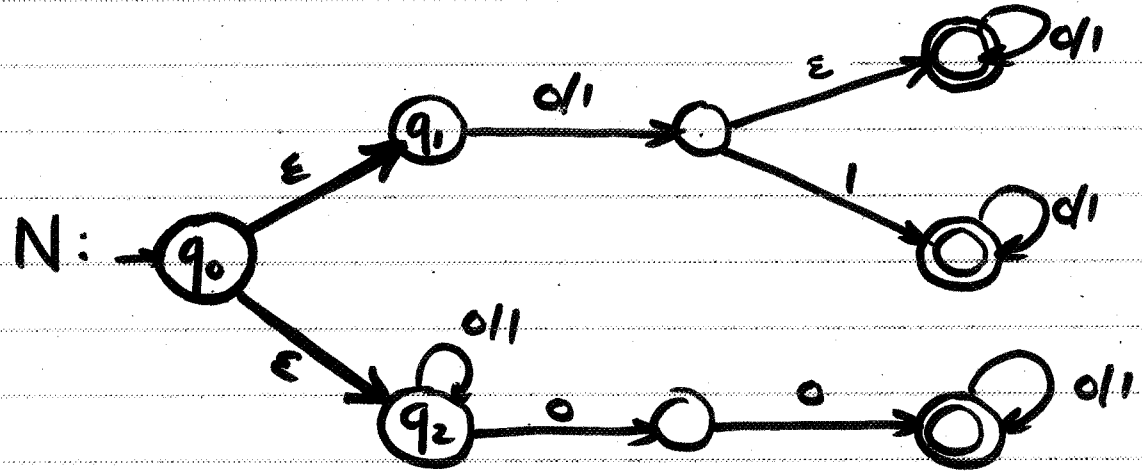
N_1 :



N_2 :

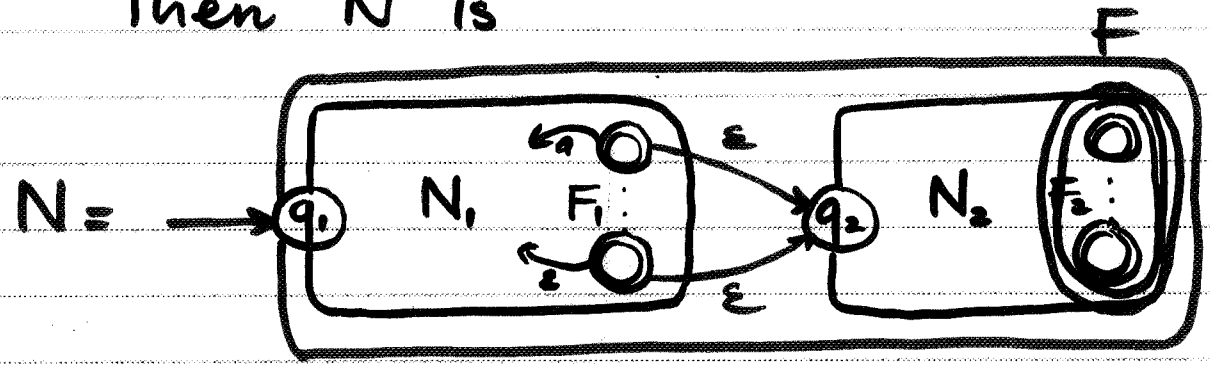


N:



Case 2. $\alpha = \alpha_1 \alpha_2$

Then N is



Thus,

$$N = (Q_1 \cup Q_2, \Sigma, \delta, q_1, F_2)$$

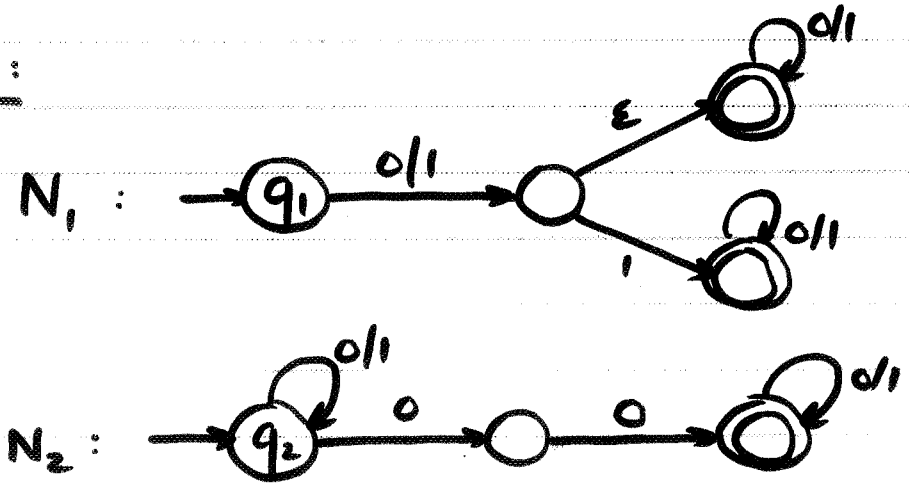
where

$$\delta(q,a) := \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 - F_1 \\ \delta_1(q,a) & \text{if } q \in F_1 \wedge a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_2\} & \text{if } q \in F_1 \wedge a = \varepsilon \\ \delta_2(q,a) & \text{if } q \in Q_2 \end{cases}$$

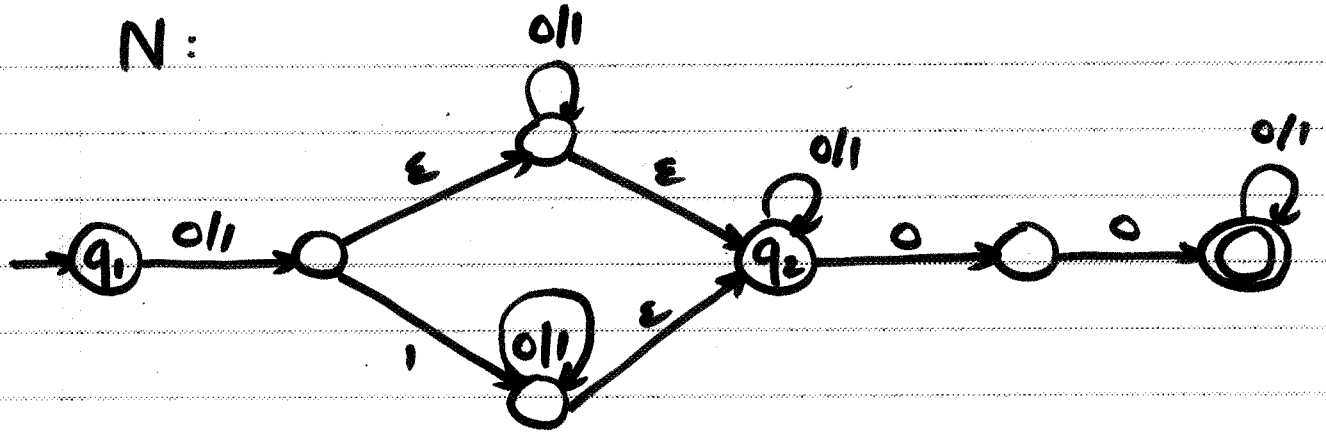
Then,

$$\begin{aligned} L(N) &= L(N_1)\, L(N_2) \\ &= L(\alpha_1)\, L(\alpha_2) \\ &= L(\alpha) \end{aligned}$$

Ex:

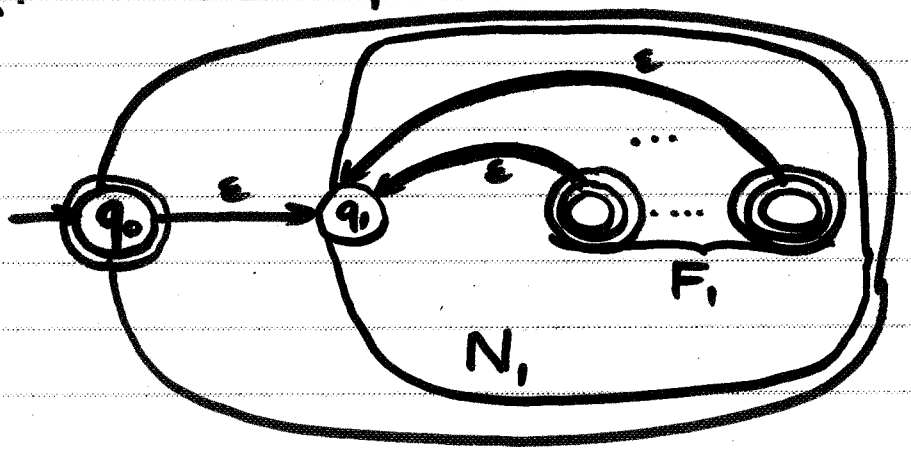


N:



Case 3. $\alpha = \alpha_1^*$

N:

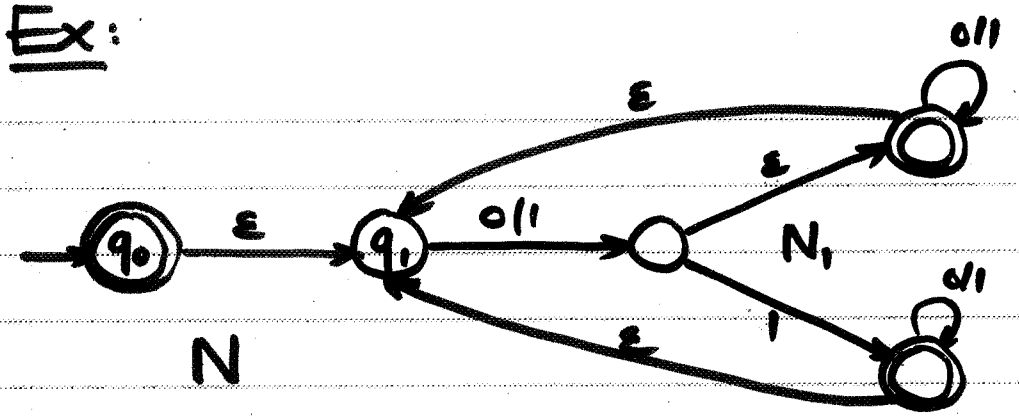


$$N = (Q_1 \cup \{q_0\}, \Sigma, \delta, q_0, F_1 \cup \{q_0\})$$

where

$$\delta(q,a) := \begin{cases} \delta_1(q,a) & \text{if } q \notin F_1 \\ \delta_1(q,a) & \text{if } q \in F_1 \wedge a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & \text{if } q \in F_1 \wedge a = \varepsilon \\ \{q_1\} & \text{if } q = q_0 \wedge a = \varepsilon \\ \emptyset & \text{if } q = q_0 \wedge a \neq \varepsilon \end{cases}$$

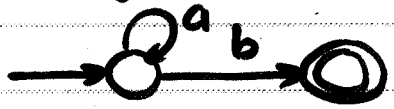
Ex:



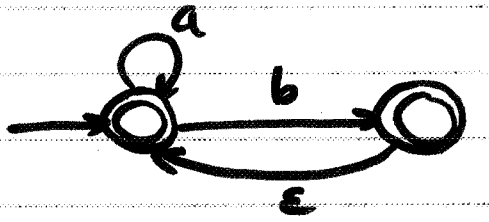
□

Remark.

In Case (3) the new initial state q_0 is needed in certain cases as the following example shows:

N_1:  $L(N_1) = a^*b$

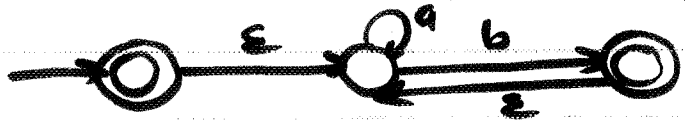
Suppose we do not add a new init. state. Then we would obtain

$\tilde{N}$: 

Clearly,

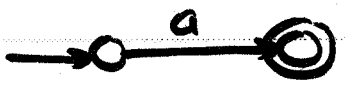
$L(\tilde{N}) \neq (a^*b)^*$ since $a \in L(\tilde{N})$, but $a \notin (a^*b)^*$.

Following the above constr. we obtain the correct NFA for $(a^*b)^*$:

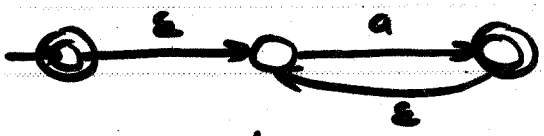


Ex: Constr. an equiv. NFA for $(a^*b)^*$

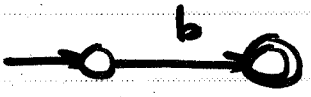
For a:



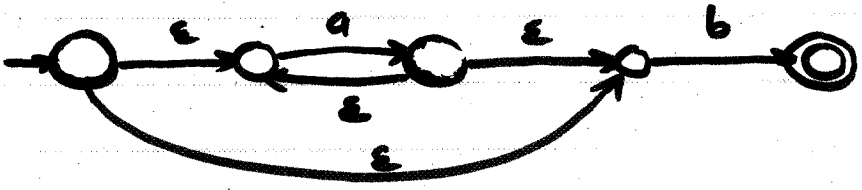
For a^*:



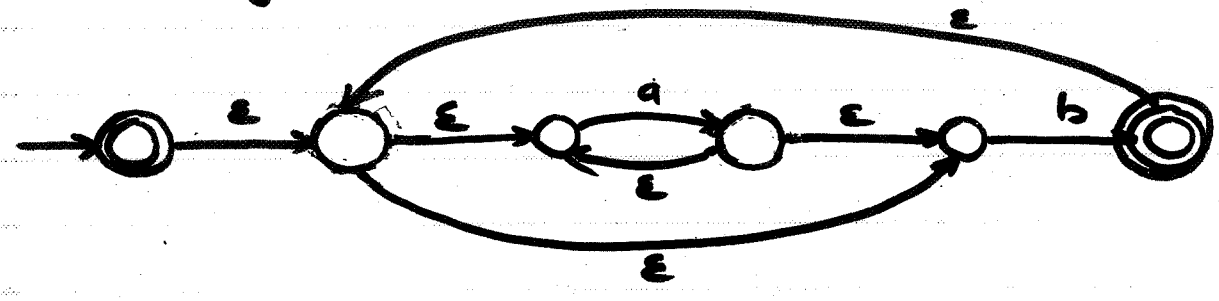
For b:



For a^*b:

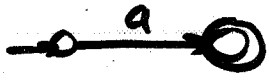


Finally for $(a^*b)^*$:

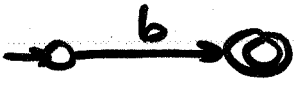


Ex: Constr. an equiv. NFA for $(ab+a)^*$

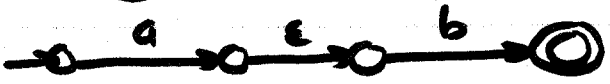
For a:



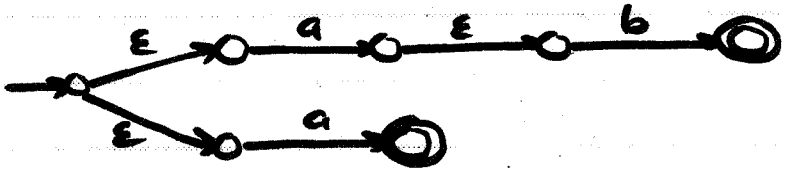
For b:



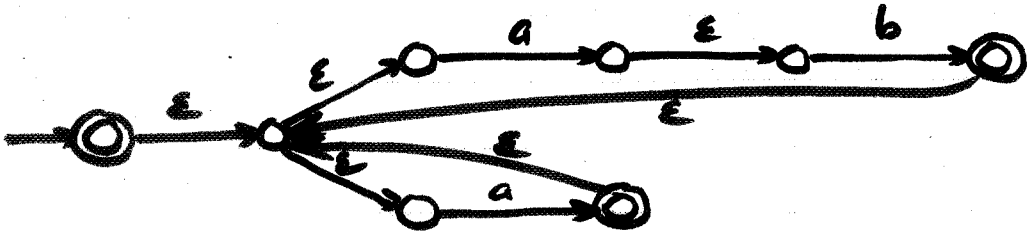
For ab:



For ab + a:

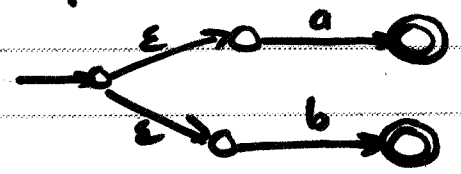


For $(ab+a)^*$:

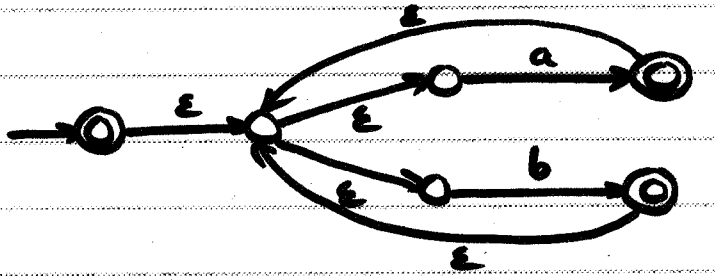


Ex: Constr. an equiv. NFA for $(a+b)^*aba$

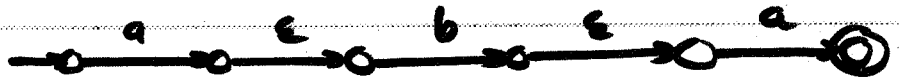
For $a+b$:



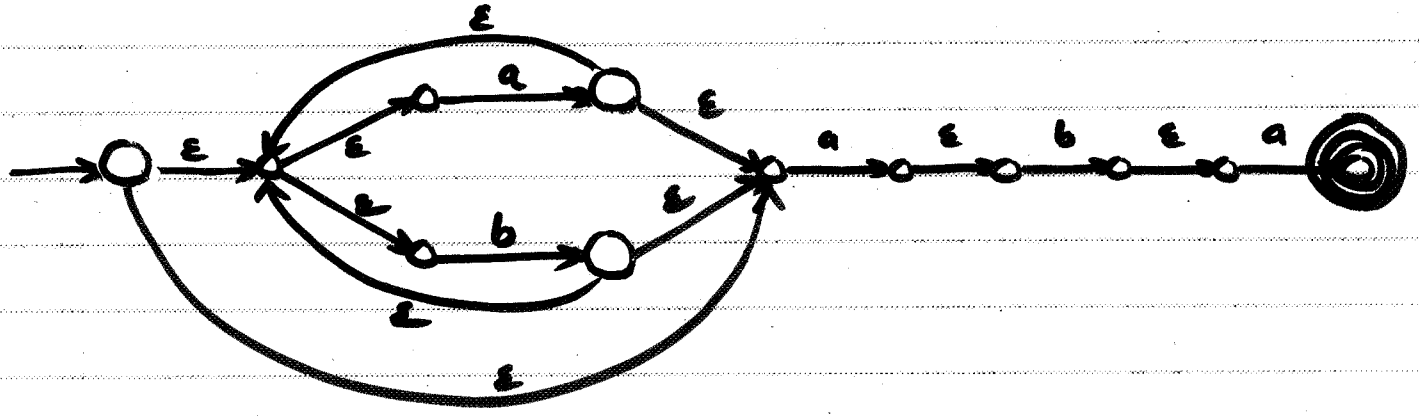
For $(a+b)^*$:



For aba:



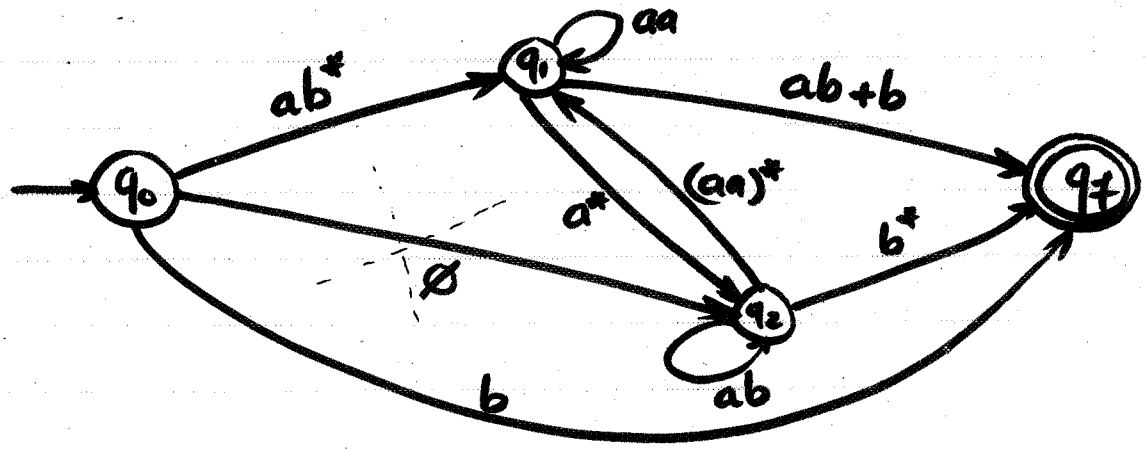
For $(a+b)^*aba$



Next: Convert a given DFA (NFA) to an equiv. RE.

First: we introduce the notion of a generalized NFA (GNFA)

Ex: (transitions of a GNFA are labeled by REs)



For example, ab^3a^4b is accepted by computation $q_0 \xrightarrow{ab^3} q_1 \xrightarrow{a^2} q_1 \xrightarrow{a^2} q_1 \xrightarrow{b} q_f$

or computation $q_0 \xrightarrow{ab^3} q_1 \xrightarrow{a^4} q_2 \xrightarrow{b} q_f$

Note that a trans. labeled $\emptyset$ means it does not exist.

Thus, a GNFA is an NFA in which transitions are labeled by REs.

Furthermore, a GNFA satisfies:

(1) There is a distinguished initial state & a distinguished final state q_0 and q_f s.t.
- no trans. entering q_0, and
- no trans. originating from q_f

(2) Transitions are labeled by REs in $\mathcal{R}$ (= set of REs over Σ)

Def. A GNFA is a 5-tuple $N = (Q, \Sigma, \delta, q_0, q_f)$ s.t.

(1) q_0 and q_f are init./final states

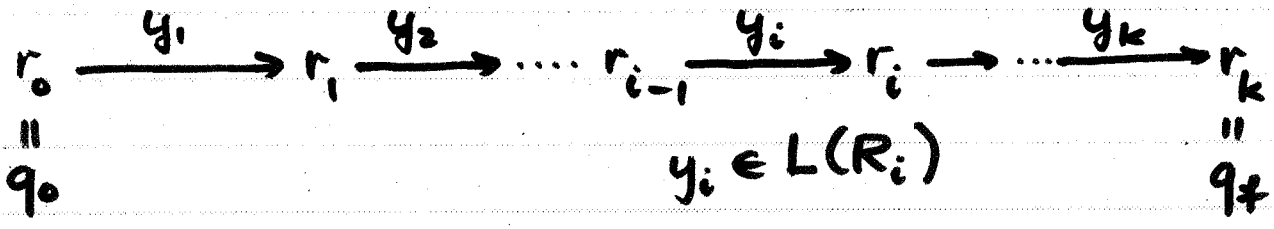
(2) $\delta : (Q - \{q_f\}) \times (Q - \{q_0\}) \longrightarrow \mathcal{R}$ is the trans. fct. □

Consider input string $w \in \Sigma^*$. It is accepted by N if it can be written as $w = y_1 y_2 \cdots y_k$, $y_i \in \Sigma^*$ s.t. $\exists$ states $r_0, r_1, \ldots, r_k \in Q$ satisfying

(1) $r_0 = q_0$, $r_k = q_f$, and

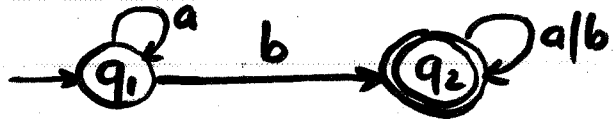
(2) $\forall i = 1, \ldots, k : \delta(r_{i-1}, r_i) = R_i \wedge y_i \in L(R_i)$

Illustration:

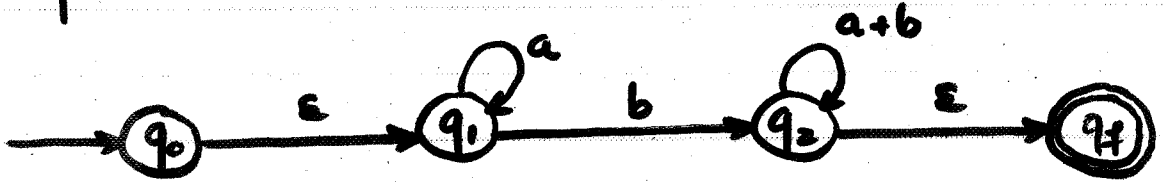


Transforming an NFA to an equiv. GNFA

Ex: N :



equiv. GNFA N':



Input. An NFA $N = (Q, \Sigma, \delta, q_1, F)$

Output. An equiv. GNFA

$$N' = (Q \cup \{q_0, q_f\}, \Sigma, \delta', q_0, q_f)$$

Method.

(1) Add q_0 and ε-trans. from q_0 to original initial state q_1

(2) Add final state q_f and ε-trans from each original final state $q \in F$ to q_f

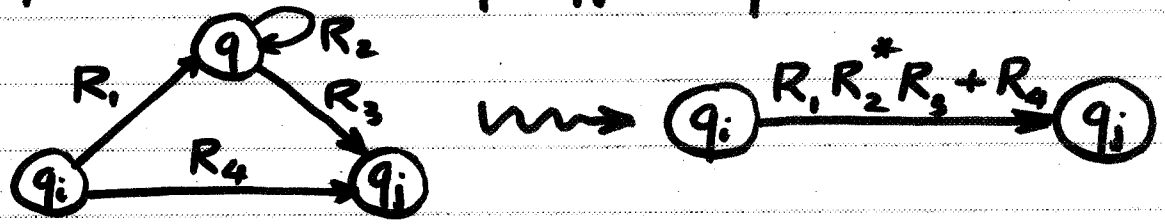
(3) Replace transitions between any pair of states by a single trans. whose label is + of labels of original transitions. □

Goal: To shrink a given GNFA to an equiv. GNFA with only two states, namely q_0 and q_f.

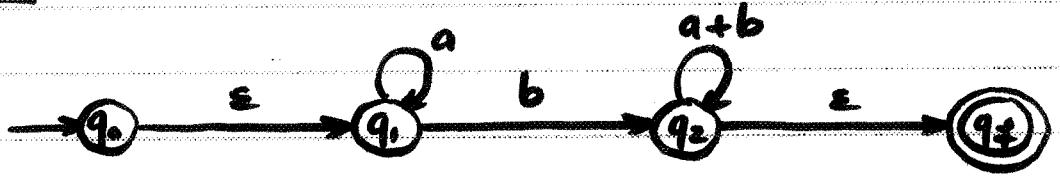
$\Rightarrow$ The label of the trans. from q_0 to q_f is the desired RE.

Idea: (Equiv.-preserving state elimination)

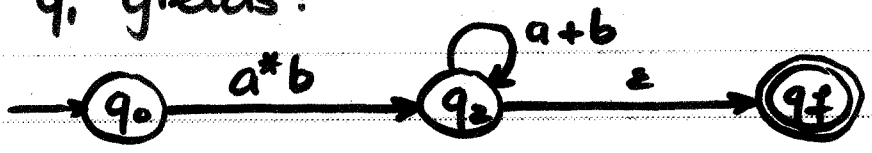
- Select an arbitrary state $q \notin \{q_0, q_f\}$
- Modify transitions in GNFA as follows: $\forall\, q_i, q_j \neq q$:



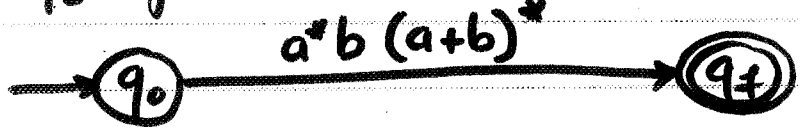
Ex. GNFA



Elim. q_1 yields:



Elim. q_2 yields:



Thus, the desired RE is $a^*b(a+b)^*$

Shrinking a GNFA to an equiv. GNFA with only two states

Input. GNFA $N = (Q, \Sigma, \delta, q_0, q_f)$

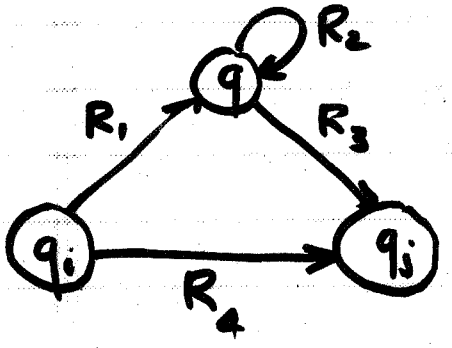
Output. An equiv. GNFA N' with only two states q_0, q_f

Method.

repeat . select a state $q \in Q - \{q_0, q_f\}$

. **for** all $q_i \in Q - \{q, q_f\}$ and $q_j \in Q - \{q, q_0\}$ **do**

label transition $q_i \longrightarrow q_j$ by $R_1 R_2^* R_3 + R_4$, where



R_1 is label of $q_i \longrightarrow q$

R_2 ——— $q \longrightarrow q$

R_3 ——— $q \longrightarrow q_j$

R_4 ——— $q_i \longrightarrow q_j$

until N' has only 2 states q_0, q_f ∎

Thm. There is an algorithm that constructs for an NFA an equiv. RE.

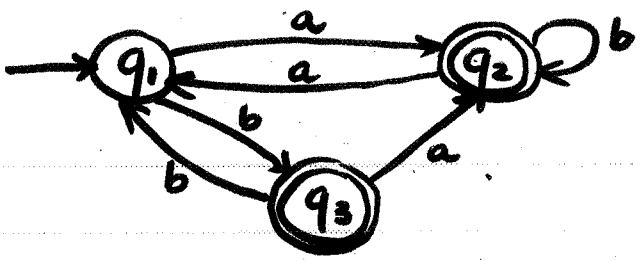
Pf. Given an NFA N:

. Transform N to an equiv. GNFA

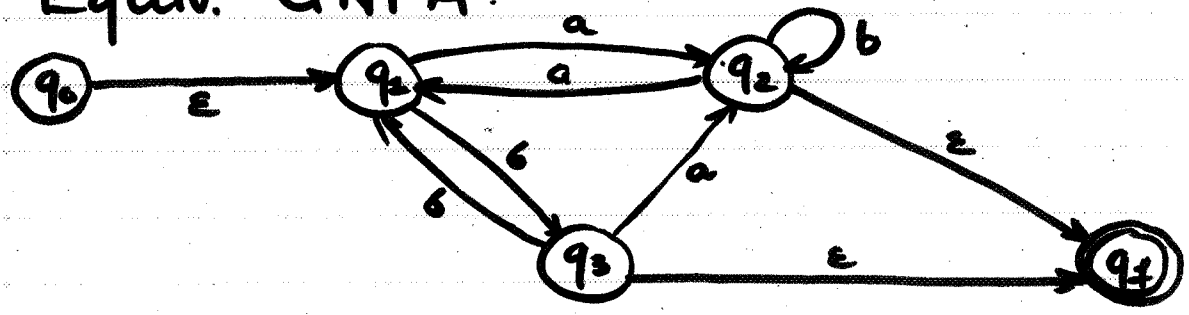
. Shrink GNFA to an equiv. GNFA N' with only two states q_0, q_f

. Label of $q_0 \longrightarrow q_f$ is desired RE ⊠

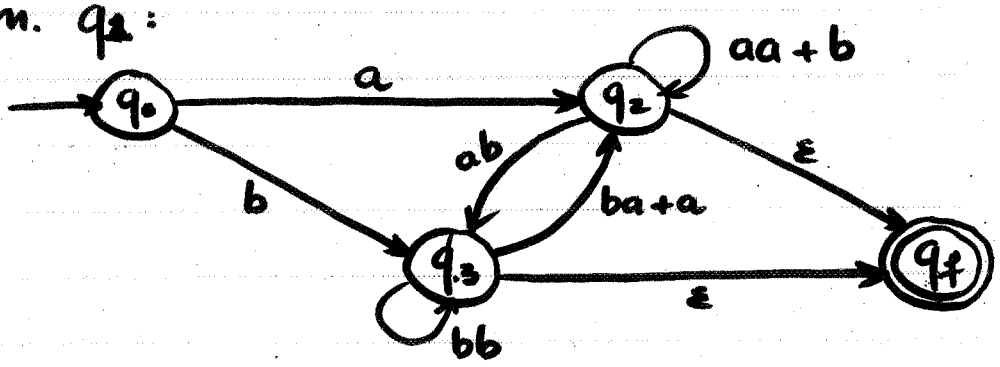
Ex: NFA N :



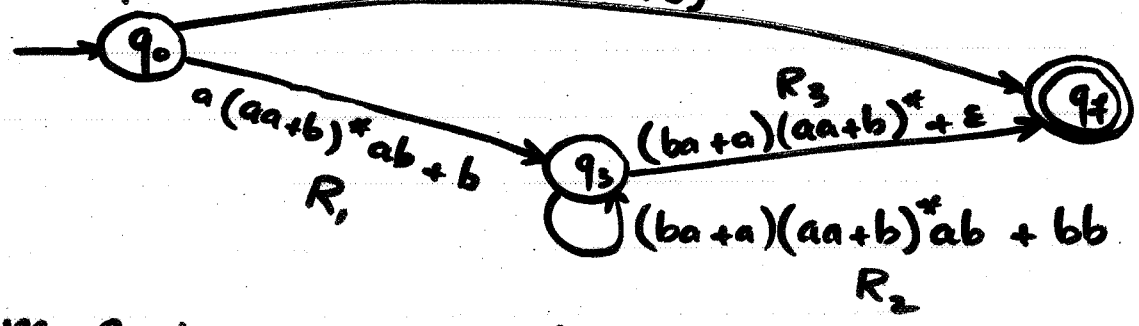
Equiv. GNFA:



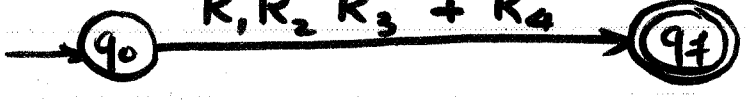
Elim. q_1:



Elim. q_2:



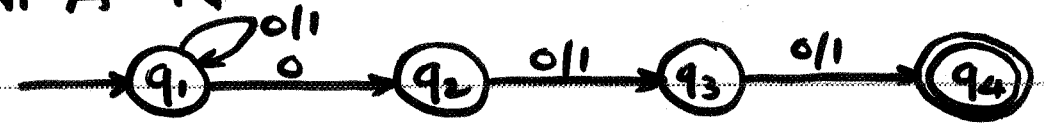
Elim. q_3:



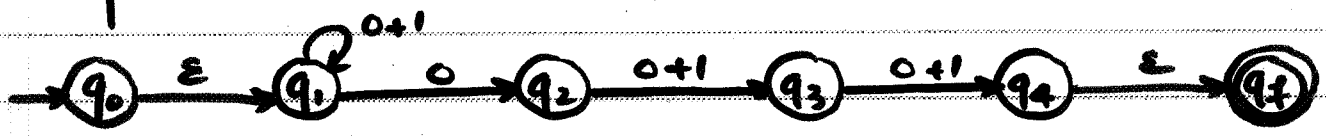
Thus, desired RE is:

$$[a(aa+b)^*ab+b][(ba+a)(aa+b)^*ab+bb]^*[(ba+a)(aa+b)^*+\varepsilon] + a(aa+b)^*$$

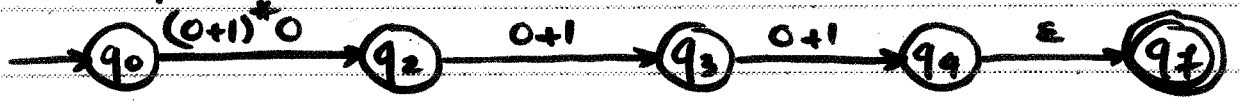
Ex: NFA N:



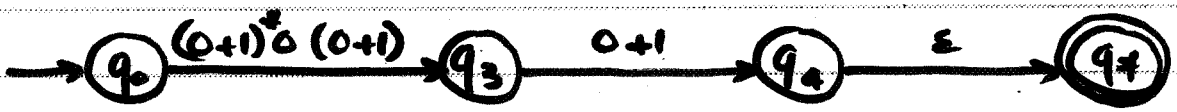
Equiv. GNFA:



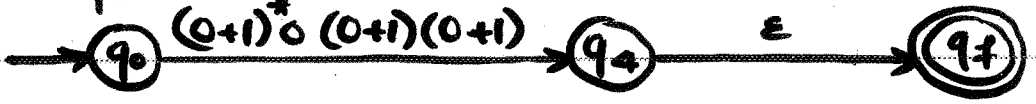
Elim. q_1:



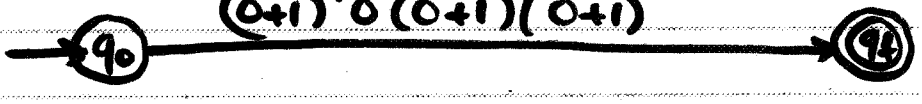
Elim. q_2:



Elim. q_3:



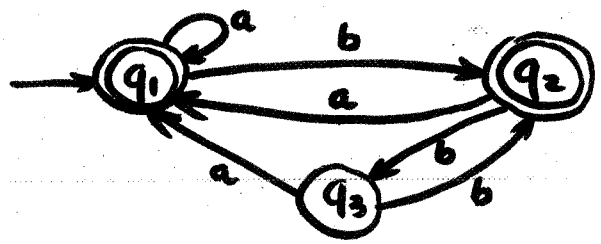
Elim. q_4:



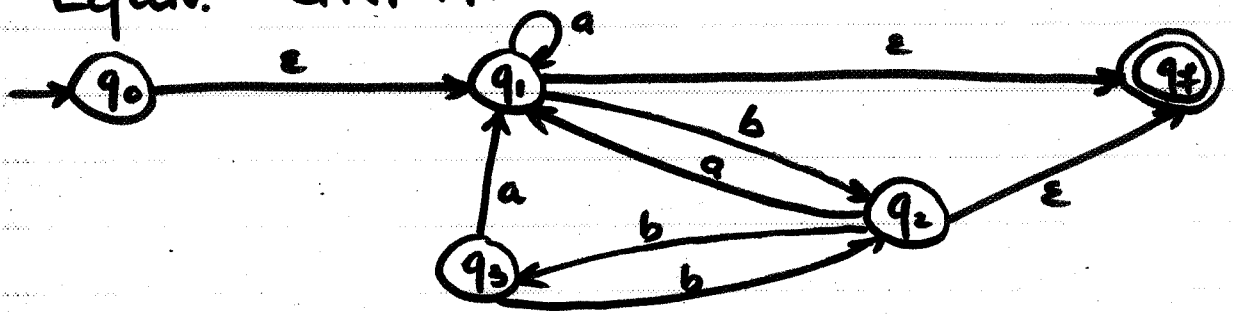
Thus the equiv. RE is

$$(0+1)^*\,0\,(0+1)(0+1)$$

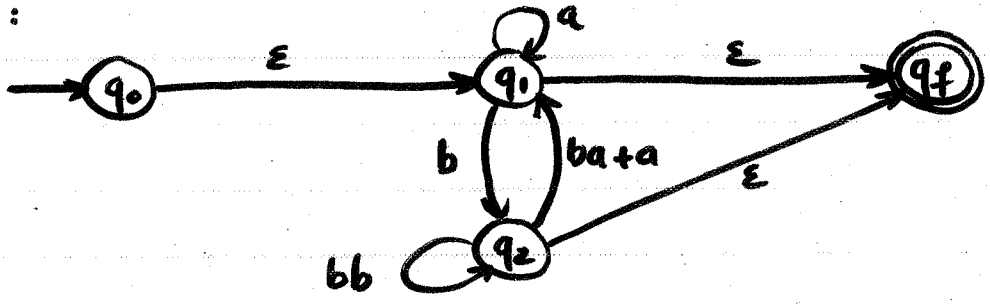
Ex: NFA N:



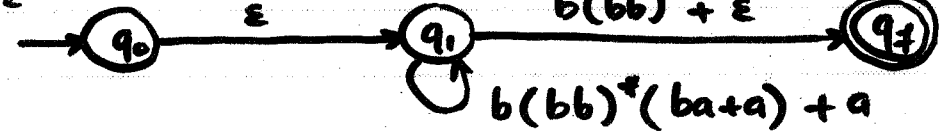
Equiv. GNFA:



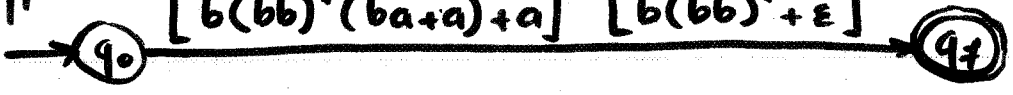
Elim. q_3:



Elim. q_2:



Elim. q_1:



Thus, RE is:

$$[b(bb)^*(ba+a)+a]^*[b(bb)^*+\varepsilon]$$

Chapter Summary

We introduced

DFAs $(Q, \Sigma, \delta, q_0, F)$

$\delta: Q \times \Sigma \to Q$

NFAs $(Q, \Sigma, \delta, q_0, F)$

$\delta: Q \times \Sigma_\varepsilon \to 2^Q$

REs over Σ using $\cdot, +, ^*$

We proved:

DFAs $\Leftrightarrow$ NFAs : **subset constr.**

NFAs $\Leftrightarrow$ REs : **Kleene Theor**

In particular:

RE $\xrightarrow[\text{rec. constr.}]{}$ NFA $\xrightarrow[\text{subset constr.}]{}$ DFA

DFA $\longrightarrow$ NFA $\xrightarrow[\text{state elimination}]{}$ RE