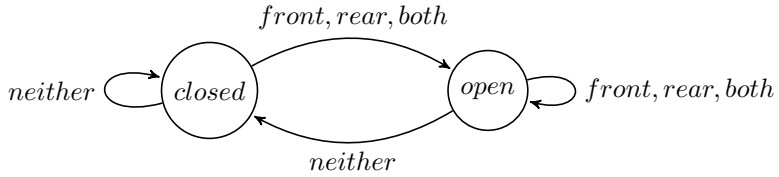
2 Automata and Language

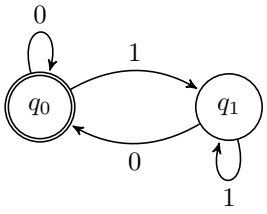
2.1 Finite Automaton

***e.g.* 2.1.** *Automatic Door*



	front	*rear*	*both*	*neither*
front	✓	×	✓	×
rear	×	✓	✓	×

***e.g.* 2.2.** $L = \{w \in \{0,1\}^* | w = w_1 w_2 \cdots w_n, w_n = 0\}$



remark. q_0 : *accepted state*

$Q = \{q_0, q_1\}, \Sigma = \{0,1\}, \delta : Q \times \Sigma \to Q$

	0	1
q_0	q_0	q_1
q_1	q_0	q_1

***def* 2.1.** *(finite automaton)*

A ***finite automaton*** *is a 5-tuple* $(Q, \Sigma, \delta, q_0, F)$*, where:*

1. Q *is a finite set called the states*
2. Σ *is the alphabet*

3. $\delta : Q \times \Sigma \to Q$ *is the transition function*

4. q_0 *is the start state*

5. $F \subseteq Q$ *is the set of accept states*

***def* 2.2.** *Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton, let $w = w_1 w_2 \cdots w_n$ be a string, where each $w_i \in \Sigma$.*

Then M accept w if there is a sequence of states $r_0, r_1, \cdots, r_n \in Q$, such that:

1. $r_0 = q_0$

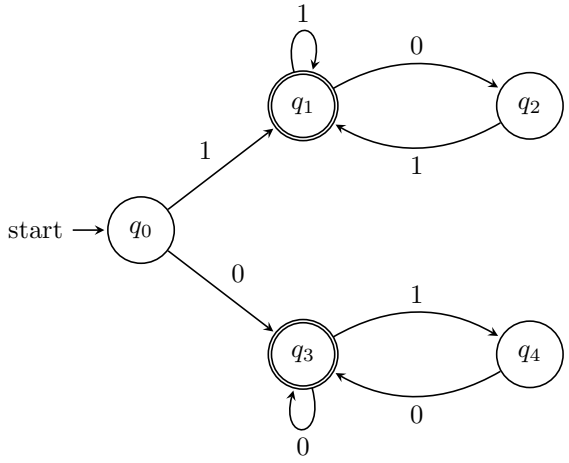
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, *for* $i = 0, 1, 2, \cdots, n-1$

3. $r_n \in F$

***def* 2.3.** *If L is the set of strings that M accepts, we say L is the language of M, and write L(M) = L, we say M recognizes/decides/accepts L.*

If M accepts no string, it recognizes one language namely, the empty language.

***e.g.* 2.3.** $L = \{w \in \{0,1\}^* | w = w_1 w_2 \cdots w_n, w_n = w_1\}$



2.2 Regular Language

***def* 2.4.** *(regular language) $L \subseteq \Sigma^*$ is a **regular language** if there is a finite automaton that accepts L*

Let $A, B \subseteq \Sigma^*$, define:

- *(union)* $A \cup B = \{x \in \Sigma^* | x \in A \text{ or } x \in B\}$

- *(concatenation)* $AB = \{xy | x \in A, y \in B\}$
- *(star)* $A^* = \{x_1 x_2 \cdots x_k | k \geq 0, x_1, x_2, \cdots, x_k \in A\}$

***thm* 2.5.** *If A_1, A_2 are regular languages, so is $A_1 \cup A_2$*

Proof. Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_{10}, F_1)$ accepts A_1, $M_2 = (Q_2, \Sigma_2, \delta_2, q_{20}, F_2)$ accepts A_2, construct $M = (Q, \Sigma, \delta, q_0, F)$:

1. $Q = Q_1 \times Q_2 = \{(r_1, r_2) | r_1 \in Q_1, r_2 \in Q_2\}$
2. $\delta : Q \times \Sigma \to Q$ *is defined as for each* $(r_1, r_2) \in Q$, *and each* $a \in \Sigma$, *let* $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
3. $q_0 = (q_{10}, q_{20})$
4. $F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$

□

remark. *so is $A \cap B = \overline{\overline{A} \cup \overline{B}}$*[1]

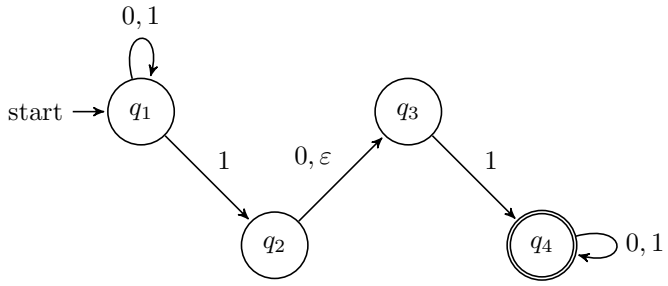
2.3 DFA and NFA

***thm* 2.6.** *If A_1, A_2 are regular languages, so is $A_1 A_2$*

- ***DFA****: deterministic finite automaton*
- ***NFA****: nondeterministic...*

If at least one of these processes accepts, then the entire computation accepts.

***e.g.* 2.4.** *NFA:*



input: 010110

[1] *proof of the closure under complement will be mentioned later*

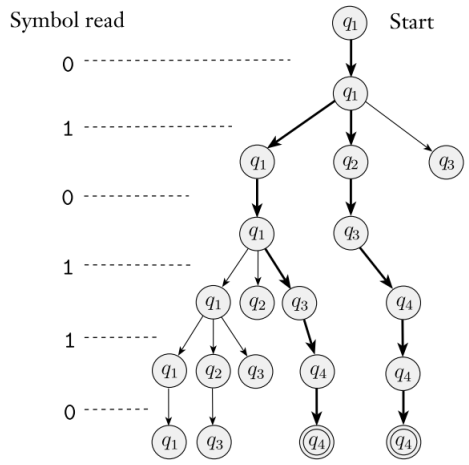


Figure 1: *The computation of NFA on input 010110*

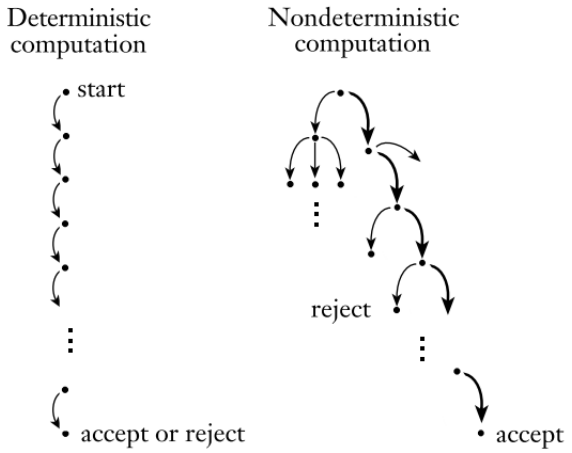
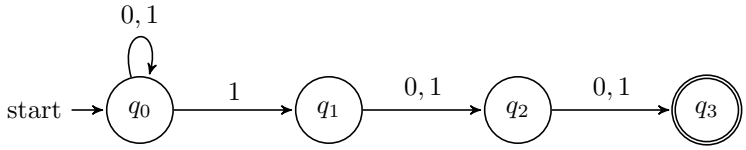


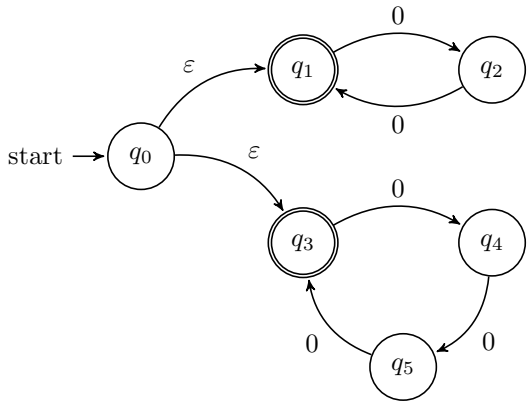
Figure 2: *Deterministic and nondeterministic computations with an accepting branch*

remark. *If a language can be accepted by DFA, then its time complexity is O(n), space complexity is O(1).*

***e.g.* 2.5.** *Let A be the language consisting of all strings over* $\{0,1\}$ *containing a* 1 *in the third position from the end (e.g.,* 000100 *is in A but* 0011 *is not). The following four-state NFA recognizes A.*



***e.g.* 2.6.** $L = \{0^k, 2 | k \text{ or } 3 | k\}$



***def* 2.7.** *(NFA)*

An ***NFA*** *is a 5-tuple* $(Q, \Sigma, \delta, q_0, F)$*,where:*

1. Q *is a finite set of states*
2. Σ *is the alphabet*
3. $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to P(Q)$ *is the transive function*
4. $q_0 \in Q$ *is the start state*
5. $F \subseteq Q$ *is the set of accept states*

remark. $P(Q)$ *is the collection of all the subsets of Q (power set)*

- *variant1:* $\delta : Q \times \Sigma^* \to Q$

  We guarantee there is at most one applicable transition.

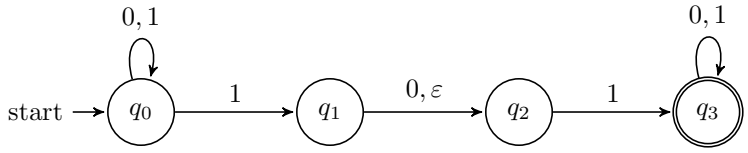
- *variant2: If there are multiple applicable transitions,non-deterministically choose one.*

***def* 2.8.**

Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA, and let $w \in \Sigma^*$. Say N accepts w if we can write $w = y_1 y_2 \cdots y_m$, where $y_i \in \Sigma \cup \{\varepsilon\}$, and there exist $r_0, r_1, \cdots, r_m \in Q$, such that:

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

***e.g.* 2.7.** $Q = \{q_0, q_1, q_2, q_3\}, \Sigma = \{0, 1\}, F = \{q_3\}, \delta : Q \times \Sigma \to Q$



$q \backslash \Sigma$	0	1	ε
q_0	$\{q_0\}$	$\{q_0, q_1\}$	$\emptyset$
q_1	$\{q_2\}$	$\emptyset$	$\{q_2\}$
q_2	$\emptyset$	$\{q_3\}$	$\emptyset$
q_3	$\{q_3\}$	$\{q_3\}$	$\emptyset$

***thm* 2.9.** *Every NFA has an equivalent DFA*

Proof. Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing A. Construct a DFA $M = (Q', \Sigma, \delta', q_0', F)$ recognizing A:

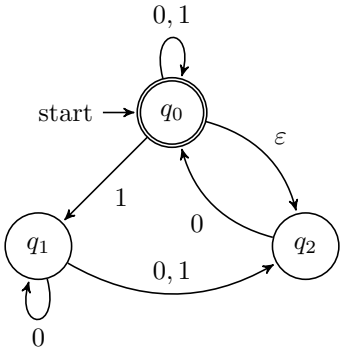
Let $E(R) = \{q \in Q |$ q can be reached from R by traveling along zero or more ε arrows$\}$

Define M as follows:

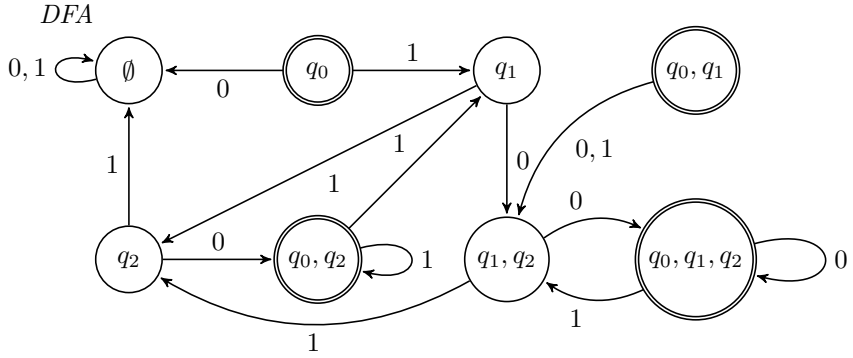
1. $Q' = P(Q)$
2. *For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q : q \in E(\delta(r, a))$ for some $r \in R\} = \bigcup_{r \in R} E(\delta(r, a))$*
3. $q_0' = E(\{q_0\})$
4. $F' = \{R \in Q', R \cap F \neq \emptyset\}$

□

***e.g.* 2.8.** *NFA*



DFA



***corollary* 2.10.** *A language is a regular language iff an NFA recognizes it.*

***thm* 2.5.** *The class of regular languages is closed under union.*

Second proof

Proof. See figure 3.

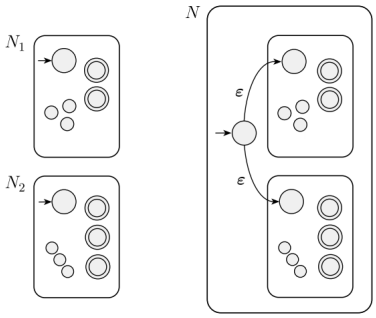


Figure 3: *Construction of an NFA N to recognize $A_1 \cup A_2$*

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ *recignize* A_1 *and* $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ *recognize* A_2
Construct $N = (Q, \Sigma, \delta, q, F)$ *as follows:*

1. $Q = Q_1 \cup Q_2 \cup \{q_0\}$
2. q_0 *is the start state*
3. $F = F_1 \cup F_2$
4. *For* $q \in Q, a \in \Sigma \cup \{\varepsilon\}$.

$$\textit{Let } \delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q \\ \delta_2(q,a) & q \in Q \\ \{q_1, q_2\} & q = q_0 \textit{ and } a = \varepsilon \\ \emptyset & q = q_0 \textit{ and } a \neq \varepsilon \end{cases}$$

□

***thm* 2.11.** *The class of regular languages is closed under concatenation.*

Proof. See figure 4. Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ *recignize* A_1 *and* $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ *recognize* A_2
Construct $N = (Q, \Sigma, \delta, q, F)$ *as follows:*

1. $Q = Q_1 \cup Q_2$
2. q_1 *is the start state*
3. $F = F_2$

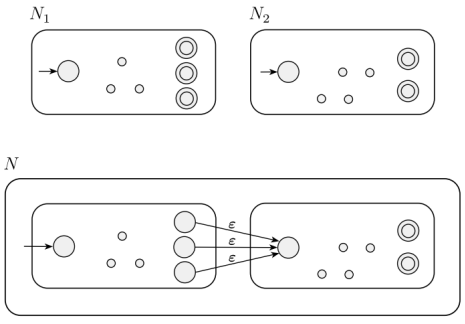


Figure 4: *Construction of an NFA* N *to recognize* A_1A_2

4. *For* $q \in Q, a \in \Sigma \cup \{\varepsilon\}$.

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

□

***thm* 2.12.** *The class of regular languages is closed under star.*

Proof. See figure 5.

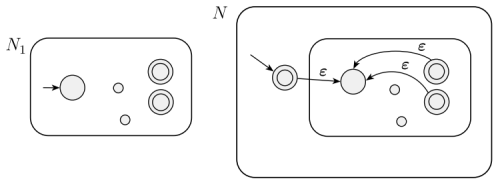


Figure 5: *Construction of an NFA N to recognize* A_1^*

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ *recognize* A_1
Construct $N = (Q, \Sigma, \delta, q, F)$ *as follows:*

1. $Q = Q_1 \cup \{q_0\}$
2. q_0 *is the start state*
3. $F = \{q_0\} \cup F_1$
4. *For* $q \in Q, a \in \Sigma \cup \{\varepsilon\}$.

$$\textit{Let } \delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \textit{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \textit{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \textit{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \textit{ and } a = \varepsilon \\ \emptyset & q = q_0 \textit{ and } a \neq \varepsilon \end{cases}$$

□

***thm* 2.13.** *The class of regular languages is closed under complement.*

Proof. *Let DFA* $M = (Q, \Sigma, \delta, q_0, Q_{accept})$ *recognizing* A*, then* $\overline{A} = \Sigma^* - A$*,*
construct $M' = (Q, \Sigma, \delta, q_0, Q'_{accept}), Q'_{accept} = Q - Q_{accept}$*, it's easy to prove* $L(Q') = \overline{A}$*.* □

2.4 Regular Expression

***e.g.* 2.9.** *Consider students' ID number, with boys' ending with an odd number, while girls' ending with even number.*

$boys : (0 \cup 1 \cup \cdots \cup 9)^*(1 \cup 3 \cup 5 \cup 7 \cup 9)$

***def* 2.14.** *(regular expression)*

R is a ***regular expression*** *if R is:*

1. a *for some* $a \in \Sigma$
2. ε
3. $\emptyset$
4. $R_1 \cup R_2$*, where* R_1, R_2 *are regular expressions*
5. $R_1 R_2$*, where* R_1, R_2 *are regular expressions*
6. R^**, where* R *is a regular expression*

***e.g.* 2.10.**

1. 0^*10^*
2. $\Sigma^*1\Sigma^* = \{w | w = \cdots 1 \cdots\}$
3. $\Sigma^*001\Sigma^* = \{w | w = \cdots 001 \cdots\}$
4. $1^*(01^+)^* = \{w \in \{0,1\}$*, every 0 in w is followed by at least 1*$\}$
5. $(\Sigma\Sigma)^* = \{w |$*the length of w is even*$\}$
6. $1^*\emptyset = \emptyset$

***exercises* 2.1.**

1. *w contains substring 110:* $\{\Sigma^*110\Sigma^*\}$
2. *w doesn't contain 00 as substring:* $(0 \cup \varepsilon)(1 \cup 10)^*$

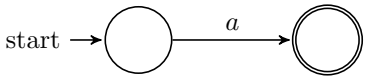
3. *the number of 1s is a multiple of 3:* $\{(0^*10^*10^*10^*)^*\}$
4. *w contains at least two 1s and one 0:* $(\Sigma^*1\Sigma^*1\Sigma^*0\Sigma^*) \cup (\Sigma^*1\Sigma^*0\Sigma^*1\Sigma^*) \cup (\Sigma^*0\Sigma^*1\Sigma^*1\Sigma^*)$

***thm* 2.15.** *A language is regular iff some regular expression described it*

***lemma* 2.16.** ($\Leftarrow$) *If a language is decribed by a regular expression, then it's regular*

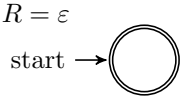
Proof. Let's convert R into an NFA N

1. $R = a, a \in \Sigma$

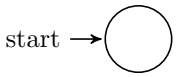
   Note that this machine fits the definition of an NFA but not that of a DFA because it has some states with no exiting arrow for each possible input symbol. Of course, we could have presented an equivalent DFA here; but an NFA is all we need for now, and it is easier to describe. Formally, $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$*, where we describe* δ *by saying that* $\delta(q_1, a) = \{q_2\}$ *and that* $\delta(r, b) = \emptyset$ *for* $r \neq q_1$ *or* $b \neq a$*.*

2. $R = \varepsilon$

   Formally, $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$*, where we describe* δ *by saying that* $\delta(q_1, a) = \emptyset$ *for* $\forall r, b$*.*

3. $R = \emptyset$

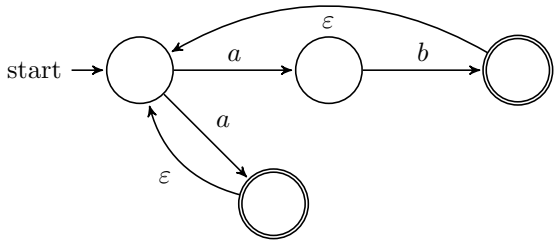
   Formally, $N = (\{q_1\}, \Sigma, \delta, q_1, \emptyset)$*, where we describe* δ *by saying that* $\delta(q_1, a) = \emptyset$ *for* $\forall r, b$*.*

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

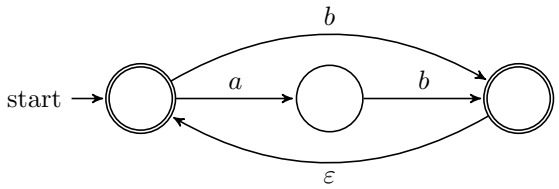
   For the last three cases, we use the constructions given in the proofs that the class of regular languages is closed under the regular operations. In other words, we construct the NFA for R from the NFAs for R_1 *and* R_2 *(or just* R_1 *in case 6) and the appropriate closure construction.*

□

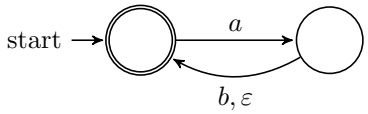
***e.g.* 2.11.** $(ab \cup a)^*$



It can be simplified as below:



Or as this:



Now let's turn to the other direction of the proof of Theorem 2.15.

***lemma* 2.17.** ($\Rightarrow$) *If a language is regular, then it is described by a regular expression.*

PROOF IDEA *We need to show that if a language A is regular, a regular expression describes it. Because A is regular, it is accepted by a DFA. We describe a procedure for converting DFAs into equivalent regular expressions.*

Proof. We break this procedure into two parts, using a new type of finite automaton called a ***generalized nondeterministic finite automaton***, ***GNFA***.

DFA $\rightarrow$ *GNFA* $\rightarrow$ *regular expressions* □

***def* 2.18.** *A generalized nondeterministic finite automaton is a 5-tuple,* $(Q, \Sigma, \delta, q_{start}, q_{accept})$*, where:*

1. Q *is a finite set of states*
2. Σ *is the input alphabet*
3. $\delta : (Q - \{q_{accept}\}) \times (Q - \{q_{start}\}) \rightarrow R$ *is the transive function*

4. q_{start} *is the start state*

5. q_{accept} *is the accept state*

remark. GNFA *requires:*

1. *The start state has transition arrows going to every other state but no arrows coming in from any other state.*

2. *There is only a single accept state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is not the same as the start state.*

3. *Except for the start and accept states, one arrow goes from every state to every other state and also from each state to itself.*

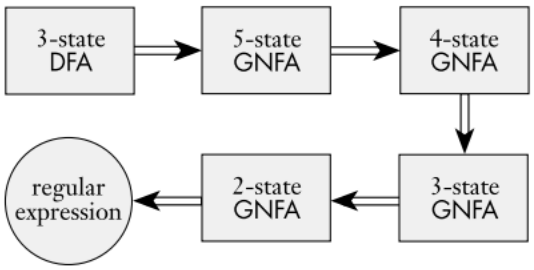


Figure 6: *Typical stages in converting a DFA to a regular expression*

Let's finish the proof for Lemma 2.17

Proof.

1. *Add a new start state with* ε *arrow to the old start state*

2. *Add a new accept state with* ε *arrow(s) from old accept states*

3. *Replace multiple lables or arrows between two states with a single arrow whose label is the union of the previous lables*

4. *Add arrows with* $'\emptyset'$ *between states that had no arrows*

Now consider convert ***GNFA*** *to* ***regular expression***

We use the procedure ***CONVERT(G)***

1. *Let k be the number of states in G*
2. *If $k = 2$, return $\delta(q_{start}, q_{accept})$*
3. *If $k > 2$, choose $q_{rip} \in Q \backslash \{q_{start}, q_{accept}\}$*

   Let G' be the GNFA $(Q', \Sigma, \delta', q_{start}, q_{accept})$, where:

   (a) $Q' = Q - q_{rip}$

   (b) $\forall q_i \in Q' - \{q_{qccept}\}, \forall q_j \in Q' - \{q_{start}\}$

   Then $\delta'(q_i, q_j) = \delta(q_i, q_j) \cup (\delta(q_i, q_{rip})\delta(q_{rip}, q_{rip})^*\delta(q_{rip}, q_j))$

   (see Figure7)

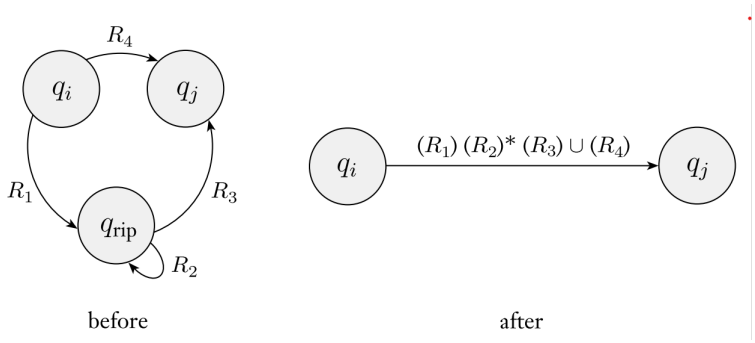


Figure 7: *Constructing an equivalent GNFA with one fewer state*

remark. *$q_i = q_j$ is possible*

4. *compute convert(G') and return*

claim* 2.19.** *For any GNFA G,* ***CONVERT(G) *is equivalent to G.*

Proof. We prove this claim by induction on k, the number of states of the GNFA.

Basis:*Prove the claim true for $k = 2$ states. If G has only two states, it can have only a single arrow, which goes from the start state to the accept state. The regular expression label on this arrow describes all the strings that allow G to get to the accept state. Hence this expression is equivalent to G.*

Induction step: *Assume that the claim is true for k - 1 states and use this assumption to prove that the claim is true for k states. First we show that G and G′ recognize the same language. Suppose that G accepts an input w. Then in an accepting branch of the computation, G enters a sequence of states:* $q_{start}, q_1, q_2, \cdots, q_{accept}$

If none of them is the removed state q_{rip}*, clearly G′ also accepts w. The reason is that each of the new regular expressions labeling the arrows of G′ contains the old regular expression as part of a union.*

If q_{rip} *does appear, removing each run of consecutive* q_{rip} *states forms an accepting computation for G′. The states* q_i *and* q_j *bracketing a run have a new regular expression on the arrow between them that describes all strings taking qi to* q_j *via* q_{rip} *on G. So G′ accepts w.*

Conversely omitted.

The induction hypothesis states that when the algorithm calls itself recursively on input G′ , the result is a regular expression that is equivalent to G′ because G′ has k - 1 states. Hence this regular expression also is equivalent to G, and the algorithm is proved correct □

Thus ***Lemma 2.17*** *and* ***Theorem 2.15*** *are proved* □

e.g. **2.12.**

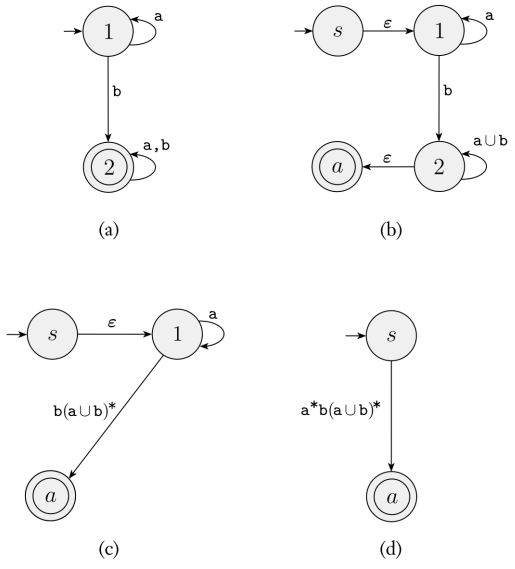


Figure 8: *Converting a two-state DFA to an equivalent regular expression*

2.5 Nonregular Languages

***e.g.* 2.13.** *Consider a language* $B = \{0^n 1^n | n \geq 0\}$

If we attempt to find a DFA that recognizes B, we discover that the machine seems to need to remember how many 0s have been seen so far as it reads the input. Because the number of 0s isn't limited, the machine will have to keep track of an unlimited number of possibilities. But it cannot do so with any finite number of states.

"Almost all languages are non-regular"

lemma* 2.20.** ***(pumping lemma)

If A is a regular language, then there is a number p (the pumping length) where if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$*, satisfying the following conditions:*

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

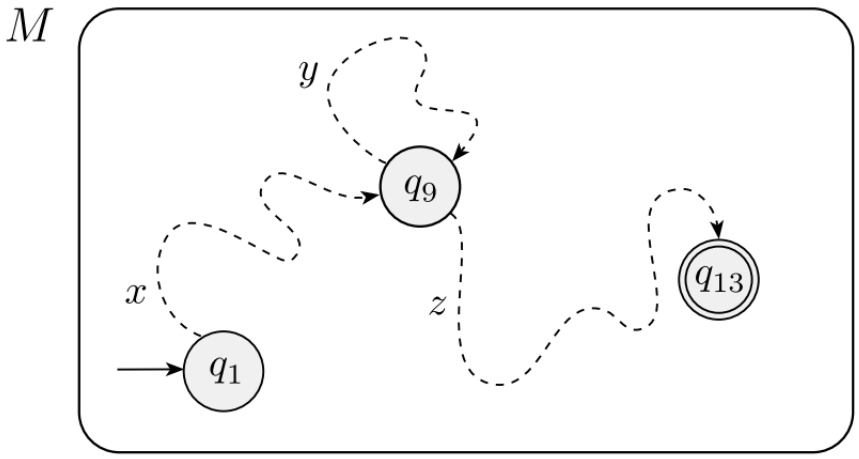


Figure 9: *Example showing how the strings x, y, and z affect M*

Proof. *Let $M = (Q, \Sigma, \delta, q_1, F)$ recognizing A and p be the number of states of M.*

Let $s = s_1 s_2 \cdots s_n$ be a string in A of length n, where $n \geq p$. Let $r_1, r_2, \cdots, r_{n+1}$ be the sequence of states that M enters while processing s, so $r_{i+1} = \delta(r_i, s_i)$ for $1 \leq i \leq n$. This sequence has length $n + 1$, which is at least $p + 1$. Among the first $p + 1$ elements in the sequence, two must be the same state, by the ***pigeonhole principle****.We call the first of these r_j and the second r_l. Because r_l occurs among the first $p + 1$ places in a sequence starting at r1, we have $l \leq p + 1$.*
Now let

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

it's easy to prove this satisfy 3 conditions □

***exercises* 2.2.**

1. *$L = \{0^n 1^n | n \geq 0\}$ is non-regular*

   Proof. *Let $n = p$,consider $s = 0^n 1^n$. By* ***pumping lemma****, write:*
   $s = xyz, |xy| \leq p = n$, then $x, y = 0^*, xy^i z \notin L, \forall i \neq 1$, contradiction! □