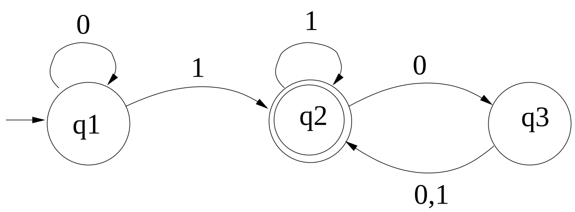
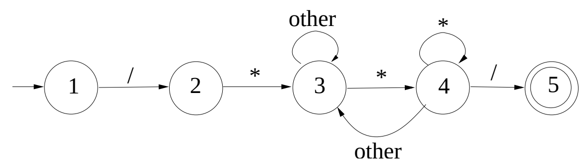
Chapter 1 from Sipser

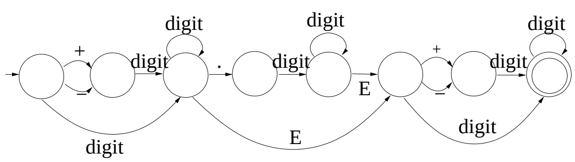
Examples of DFA's



Language recognized by this automaton:
$\{w | w$ contains at least one 1 and an even number of 0s follow the last 1$\}$



A finite automaton for C-style comments



A finite automaton for floating point numbers

Definition 1.1
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 a finite set called the **alphabet**
3. $\delta : Q \times \Sigma \to Q$ is the **transformation function**
4. $q_0 \in Q$ is the **start state**
5. $F \subseteq Q$ is the **set of accept states**.

Definition 1.10
Let A and B be languages. We define the regular operations **union**, **concatenation**, and **star**.

- **Union**: $A \cup B = \{x | x \in A \text{ or } x \in B\}$
- **Concatenation**: $A \circ B = \{xy | x \in A \text{ and } y \in B\}$
- **Star**: $A^* = \{x_1 x_2 \ldots x_k | k \geq 0 \text{ and each } x_i \in A\}$

Theorem 1.12
The class of regular languages is closed under the union operation, i.e. if A_1 and A_2 are regular languages, then $A_1 \cup A_2$ is also a regular language.
Proof:
Let M_1 recognize A_1 where
$A_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and M_2 recognize A_2 where
$A_2 = (Q_2, \Sigma, \delta_1, q_2, F_2)$. An automaton $M = (Q, \Sigma, \delta, q_0, F)$ that recognizes $A_1 \cup A_2$ is constructed as follows:

- $Q = Q_1 \times Q_2$
- $q_0 = (q_1, q_2)$
- $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $F = \{(r_1, r_2) | r_1 \in F_1 \text{ or } r_2 \in F_2\}$.

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

1. Q is a finite set of states
2. Σ is a finite alphabet
3. $\delta : Q \times \Sigma_\varepsilon \rightarrow \mathcal{P}(Q)$ is the transition function
4. $q_0 \in Q$ is the start state
5. $F \subseteq Q$ is the set of accept states

Theorem 1.19 Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

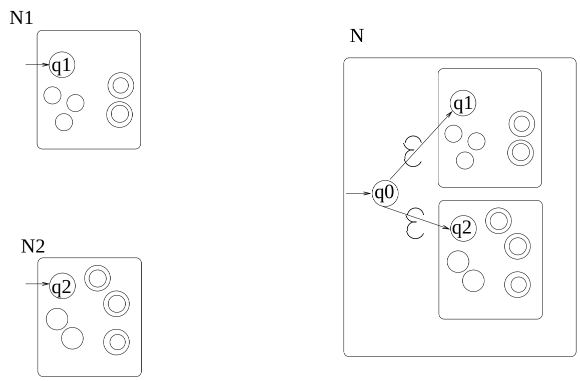
Proof Let $N = (Q, \Sigma, \delta, q_0, F)$ be a NFA recognizing some language A. We construct a DFA $M = (Q^{'}, \Sigma, \delta^{'}, q_0^{'}, F^{'})$ also recognizing A.

1. $Q^{'} = \mathcal{P}(Q)$
2. For $R \subseteq Q$ let $E(R) = \{q | q$ can be reached from R by traveling along 0 or more ε arrows $\}$.
   Then
   $$\delta^{'}(R, a) = \cup_{r \in R} E(\delta(r, a))$$
3. $q_0^{'} = E(\{q_0\})$
4. $F^{'} = \{R \in Q^{'} | R$ contains an accept state of $N\}$

Corollary 1.20 p56 A language is regular if and only if some nondeterministic finite automaton recognizes it.

Theorem 1.22 (Alternative proof to Theorem 1.19) The class of regular languages is closed under union.

Proof:



Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and let $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2. Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$

2. The new state q_0 is the start state of N.

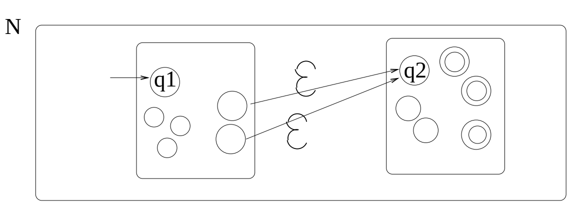
3. The accept states are $F = F_1 \cup F_2$

4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$

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

Theorem 1.23 The class of regular languages is closed under concatenation.

Proof:



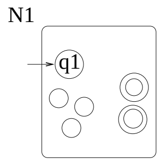
$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2. Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$

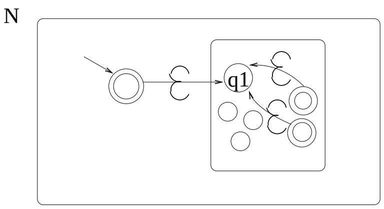
1. $Q = Q_1 \cup Q_2$
2. The start state q_1 is the same as the start state of N_1.
3. The accept states are the same as the accept states of N_2.
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$

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

Theorem 1.24 The class of regular languages is closed under the star operation.

Proof





Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1 and construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize A_1^*.

1. $Q = \{q_0\} \cup Q_1$ - i.e. the states of Q are the same as the states of Q_1 plus a new start state.
2. The new state q_0 is the start state of N.
3. $F = \{q_0\} \cup F_1$
4. Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$

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

Definition 1.26
R is a *regular expression* if R is

1. a for some a in the alphabet Σ
2. ε
3. $\emptyset$
4. $(R_1 \cup R_2)$ where R_1 and R_2 are regular expressions
5. $(R_1 \circ R_2)$ where R_1 and R_2 are regular expressions
6. (R_1^*) where R_1 is a regular expression

Example
A numerical constant that may inlude a fractional part:

$\{+, -, \varepsilon\}(DD^* \cup DD^*.D^* \cup D^*.DD^*)$
where $D = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Examples of generated strings:
$72, 3.14159, +7., -.01$

Example 1.27

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

7. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w|w$ starts and ends with the same symbol$\}$

8. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$

9. $(0 \cup \varepsilon)(1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$

10. $1^*\emptyset = \emptyset$

11. $\emptyset^* = \{\varepsilon\}$

Exercise 1.13 p 86
$\Sigma = \{0,1\}$
(a) $\{w|w$ begins with a 1 and ends with a 0$\}$
$1\Sigma^*0$

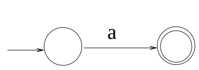
(b) $\{w|w$ contains at least three 1s$\}$
$\Sigma^*1\Sigma^*1\Sigma^*1\Sigma^*$

(e) $\{w|w$ starts with 0 and has odd length, or start with a 1 and has even length$\}$
$0(\Sigma\Sigma)^* \cup 1\Sigma(\Sigma\Sigma)^*$

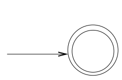
Lemma 1.29 If a language is described by a regular expression, then it is regular.

Proof: We consider the six cases in the formal definition of regular expressions.

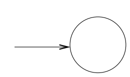
1. $R = a$ for some $a \in \Sigma$. Then $L(R) = \{a\}$ and the following NFA recognizes $L(R)$:



2. $R = \varepsilon$. Then $L(R) = \{\varepsilon\}$ and the following NFA recognizes $L(R)$:



3. $R = \emptyset$. Then $L(R) = \emptyset$ and the following NFA recognizes $L(R)$:



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

For 4,5 and 6 we convert R_1 and R_2 to NFA's and then use the appropriate construction to combine these outomata or to obtain an automata recognizing $L(R_1^*)$.

Theorem 1.28 A language is regular if and only if some regular expression describes it.

Last time:
regular expression $\rightarrow$ NFA
This time:
NFA $\rightarrow$ regular expression.

We will use a new type of automaton, a **generalized nondeterministic finite automaton** (GNFA).

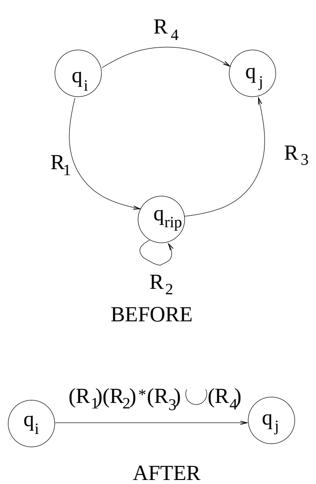
A GNFA is a NFA where we use regular expressions as labels on the arrows.

A summary of the conversion process:

- Add new begin state with an ε arrow from the new begin state to the old begin state.

- Add a new accept state and ε arrow(s) from the old accept state(s) to the new accept state.

- Select a state
  (not the new start or accept state)
  to be ripped (call it q_{rip}).
  For each pair of states q_i and q_j
  (may be the same state) with arrows from
  q_i to q_{rip} and q_{rip} to q_j do the following:



- Stop ripping states if only the begin state and end state are left.

Section 1.4 The pumping Lemma

We use the pumping lemma to show that a language is **not** regular.

Theorem 1.37 - The 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 three conditions:

1. for each $i \geq 0$, $xy^iz \in A$

2. $|y| > 0$ and

3. $|xy| \leq p$.

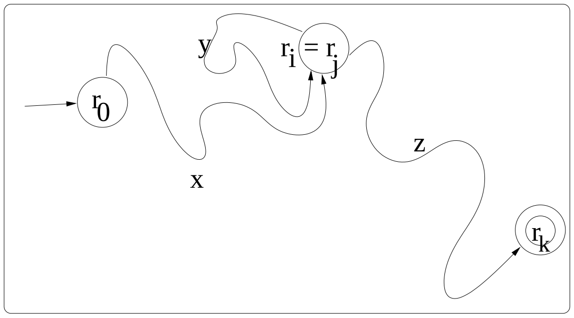
Proof: Let M be a DFA recognizing A. Let p be the number of states in M.

Let $s = s_1 s_2 ... s_k$ be any string in A of length $k \geq p$. Let $r_0, r_1 ..., r_k$ be the states visited in the DFA under the input s.

Note that r_0 is the start state and r_k is an accept state.

Since M has only p states, we have that $r_i = r_j$ for some $i < j$. Pick the smallest i and j for which this is true.

Let $x = s_1 ... s_i$, $y = s_{i+1} ... s_j$, and $z = s_{j+1} ... s_k$. Notice that x, y and z will have the required properties.



M

How do we use the pumping lemma?
Let A be a language. We want to show that A is **not** regular.

1. Assume that A regular. Thus we give a proof by contradiction that A is not regular.
2. Let p be the pumping length.
3. Pick a string s in A of length at least p of a given form. This is the hard part.
4. Show that for **any possible way** of writing s as xyz, where x, y and z satisfy requirents 2 and 3 of the pumping lemma, it is always possible to find an i such that xy^iz is not in A.
5. We conclude by contradiction that A is not regular.

Example 1.38
Show that $A = \{0^n 1^n | n \geq 0\}$ is not regular.

Assume that A is regular and let p be the pumping length.

Let $s = 0^p 1^p$. Note that $|s| = 2p \geq p$.

From the pumping lemma we can write s as xyz such that

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

Note that y can not contain 0's and 1's, otherwise $xy^2z \notin A$, since the order of the 0's and 1's are not correct in xy^2z. (In this case we pumped up - i.e. we selected $i \geq 2$. Choosing $i = 0$ is refered to as pumping down.)

Also note that if y contains only 0's, $xy^2z \notin A$, since xy^2z has more 0's than 1's. We get a similar contradiction if y contains only 1's.

We notice for any possible way of dividing s into xyz according to rules 2 and 3 of the pumping lemma, we can always pump up and obtain a string not in A.

Thus by contradiction we conclude that A is not regular.

Example 1.41

Show that $A = \{1^{n^2} | n \geq 0\}$ is not regular.

Assume that A is regular and let p be the pumping length.

Let $s = 1^{p^2}$. Note that $|s| = p^2 \geq p$.

From the pumping lemma we can write s as xyz such that

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

Let $|y| = s > 0$. Then $xy^iz \in A$ for all $i \geq 0$, i.e. $(p^2 - is)$ is a square for all $i \geq 0$. Thus $p^2 - s, p^2, p^2+s, p^2+2s, ...$ are all squares. This is not possible, since the distance between squares gets bigger and bigger ($(a+1)^2 - a = 2a+1 \to \infty$ as $a \to \infty$).

We notice that for any possible way of dividing s into xyz according to rules 2 and 3 of the pumping lemma, we can always pump up and obtain a string not in A.

Thus by contradiction we conclude that A is not regular.

Exercise 1.17a Show that $A = \{0^n 1^n 2^n | n \geq 0\}$ is not regular.

Assume that A is regular and let p be the pumping length.

Let $s = 0^p 1^p 2^p$. Note that $|s| = 3p \geq p$.

From the pumping lemma we can write s as xyz such that

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

Note that y can not contain more than one type of symbol (for example 0's and 1's), otherwise $xy^2z \notin A$, since the order of the symbols will not be correct in xy^2z. (In this case we pumped up - i.e. we selected $i \geq 2$. Choosing $i = 0$ is refered to as pumping down.)

Also note that if y contains only 0's, $xy^0z \notin A$, since xy^0z has less 0's than 1's (I pumped down just for the fun of it - pumping up also works). We get a similar contradiction if y contains only 1's or only 2's.

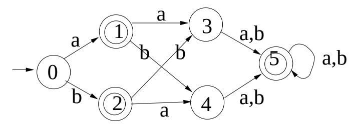
We notice for any possible way of dividing s into xyz according to rules 2 and 3 of the pumping lemma, we can always pump up or down and obtain a string not in A.

Thus by contradiction we conclude that A is not regular.

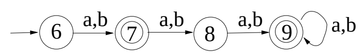
DFA STATE MINIMIZATION

This material is not from Sipser.

EXAMPLE 1



DFA



DFA MINIMIZED

Language recognized by both automata:
$\{a, b\} \cup \{$strings of length 3 or greater$\}$

In the first automaton, states 3 and 4 are equivalent, since they both go to state 5 under both input symbols.

Once we collapse them, we can collapse states 1 and 2 for the same reason, giving the second automaton.

State 0, becomes state 6;
states 1 and 2 collapse to become state 7;
states 3 and 4 collapse to become state 8;
state 5 becomes state 9.

Here is the collapse algorithm that works in general.
If state p and q are equivalent, denoted by $p \equiv q$, they are collapsed to the same state.

1. Write down a table of all pairs, initially all unmarked.

2. Mark $\{p, q\}$ if $p \in F$ and $q \notin F$ and vice versa.

3. Repeat the following untill no more changes occur:
   if there exists an unmarked pair $\{p, q\}$ such that $\{\delta(p, a), \delta(q, a)\}$ is marked for some $a \in \Sigma$, then mark $\{p, q\}$.

4. When done, $p \equiv q$ if and only if $\{p, q\}$ is not marked.

In the first step, all pairs are unmarked.

	a	b
→ 0	1	2
1F	3	4
2F	4	3
3	5	5
4	5	5
5F	5	5

0					
—	1				
—	—	2			
—	—	—	3		
—	—	—	—	4	
—	—	—	—	—	5

After step 2, all pairs consisting of one accept state and one nonaccept have been marked.

0					
✓	1				
✓	—	2			
—	✓	✓	3		
—	✓	✓	—	4	
✓	—	—	✓	✓	5

Now look at an unmarked pair, say $\{0, 3\}$.

Under input a, 0 and 3 go to 1 and 5 respectively ($\{0, 3\} \to \{1, 5\}$).

The pair $\{1, 5\}$ is not marked, so we don't mark $\{0, 3\}$, at least not yet.

Under input b, $\{0, 3\} \to \{2, 5\}$, which is not marked, so we still don't mark $\{0, 3\}$.

We then look at unmarked pairs $\{0, 4\}$ and $\{1, 2\}$ and find that we cannot mark them yet for the same reasons.

But for $\{1, 5\}$ under input a, $\{1, 5\} \to \{3, 5\}$, and $\{3, 5\}$ is marked. So we mark $\{1, 5\}$.

Similarly, under input a, $\{2, 5\} \to \{4, 5\}$, which is marked, so we mark $\{2, 5\}$.

Under both input a and b, $\{3, 4\} \to \{5, 5\}$, which is never marked (it is not even in the table), so we don't mark $\{3, 4\}$.

After the first pass of step 3, the table looks like:

0					
✓	1				
✓	—	2			
—	✓	✓	3		
—	✓	✓	—	4	
✓	✓	✓	✓	✓	5

Now we make another pass through the table.

As before, $\{0,3\} \rightarrow \{1,5\}$ under input a, but this time $\{1,5\}$ is marked, so we mark $\{0,3\}$.

Similarly, $\{0,4\} \rightarrow \{2,5\}$ under input b, and $\{2,5\}$ is marked, so we mark $\{0,4\}$.

This gives:

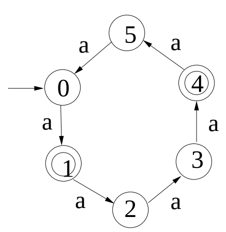
0					
✓	1				
✓	—	2			
✓	✓	✓	3		
✓	✓	✓	—	4	
✓	✓	✓	✓	✓	5

Now we check the remaining unmarked pairs and find out that $\{1,2\} \rightarrow \{3,4\}$ and $\{3,4\} \rightarrow \{5,5\}$ under both a and b, and neither $\{3,4\}$ nor $\{5,5\}$ is marked, so there are no new marks.

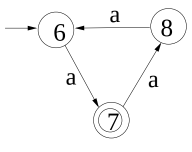
We are left with unmarked pairs $\{1,2\}$ and $\{3,4\}$, indicating that $1 \equiv 2$ and $3 \equiv 4$, thus 1 and 2 are collapsed to one state, and similarly 3 and 4. This is the same result as before.

EXAMPLE 2

Repeat the previous procedure for the following example:



DFA



DFA MINIMIZED

	a
→ 0	1
1F	2
2	3
3	4
4F	5
5	0

Make sure you end up with:

0					
✓	1				
✓	✓	2			
—	✓	✓	3		
✓	—	✓	✓	4	
✓	✓	—	✓	✓	5

Thus $0 \equiv 3$, $1 \equiv 4$, $2 \equiv 5$. So there are 3 states in the minimized DFA.