Theory of Languages and Automata

Chapter 1- Regular Languages

& Finite State Automaton

Sharif University of Technology

Automata Theory

- **Automata theory** is the study of abstract machines and automata, as well as the computational problems that can be solved using them.
- Automata play a major role in theory of computation, compiler construction, artificial intelligence, parsing and formal verification.

History of Automata

- The ***Book of Ingenious Devices*** (Arabic: كتاب الحيل *Kitab al-Hiyal*, Persian: كتاب ترفندها *Ketab tarfandha*, literally: "The Book of Tricks") was a large illustrated work on mechanical devices, including automata, published in 850 by the three brothers of Persian descent, known as the Banu Musa (Ahmad, Muhammad and Hasan bin Musa ibn Shakir) working at the House of Wisdom (*Bayt al-Hikma*) in Baghdad, Iraq, under the Abbasid Caliphate.

Finite State Automaton (FSA)

- We begin with the simplest model of Computation, called ***finite state machine*** or ***finite automaton***.
- are good models for computers with an extremely limited amount of memory.

  → Embedded Systems

- ***Markov Chains*** are the probabilistic counterpart of Finite Automata

History of FSA

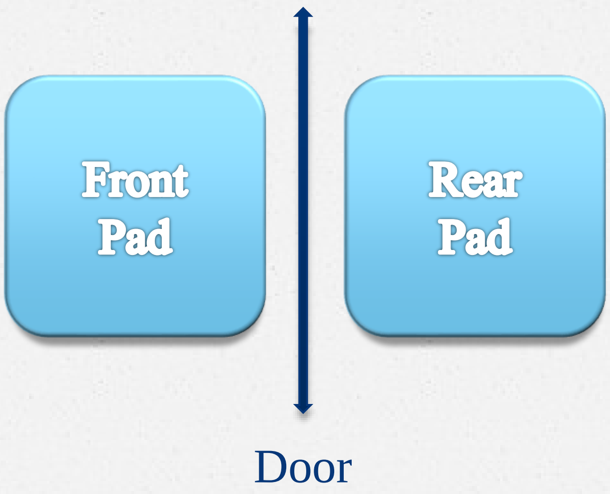
- Warren McCulloch and Walter Pitts, two neurophysiologists, were the first to present a description of finite automata in 1943.
- Their paper, entitled, "A Logical Calculus Immanent in Nervous Activity", made significant contributions to the study of neural network theory, theory of automata, the theory of computation and cybernetics.

History of FSA (con.)

- Later, two computer scientists, G.H. Mealy and E.F. Moore, generalized the theory to much more powerful machines in separate papers, published in 1955-56.
- The finite-state machines, the Mealy machine and the Moore machine, are named in recognition of their work.

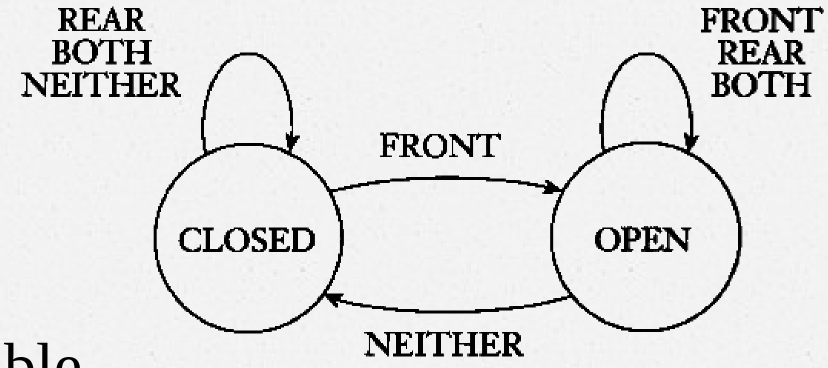
Simple Example

- Automatic door



Simple Example (cont.)

- State Diagram



- State Transition Table

	Neither	**Front**	**Rear**	**Both**
Closed	Closed	Open	Closed	Closed
Open	Closed	Open	Open	Open

Formal Definition

- A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where
  1. Q is a finite set called ***states***,
  2. Σ is a finite set called the ***alphabet***,
  3. $\delta : Q \times \Sigma \longrightarrow Q$ is the ***transition function***,
  4. $q_0 \in Q$ is the ***start state***, and
  5. $F \subseteq Q$ is the ***set of accept states***.

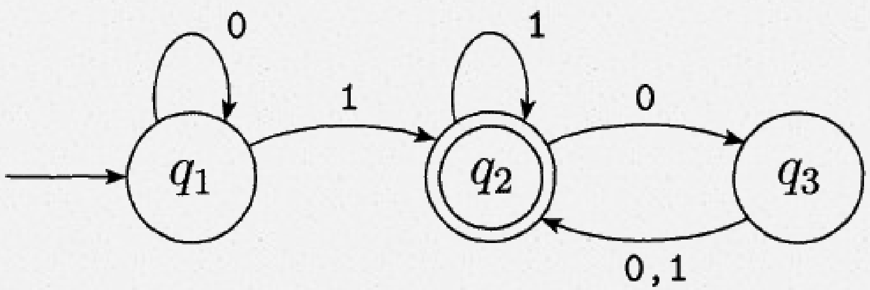
Example

- $M_1 = (Q, \Sigma, \delta, q_0, F)$, where

1. $Q = \{q_1, q_2, q_3\}$,
2. $\Sigma = \{0,1\}$,
3. δ is described as

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

1. q_1 is the start state, and
2. $F = \{q_2\}$.



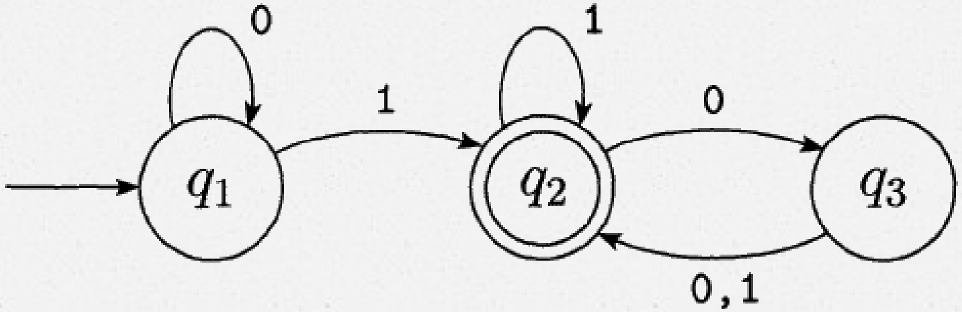
The finite automaton M_1

Language of a Finite machine

- If A is the set of all strings that machine M accepts, we say that A is the ***language of machine M*** and write: **L(M) = A**.
  - ✓ We say that ***M recognizes A***

    or that ***M accepts A***.

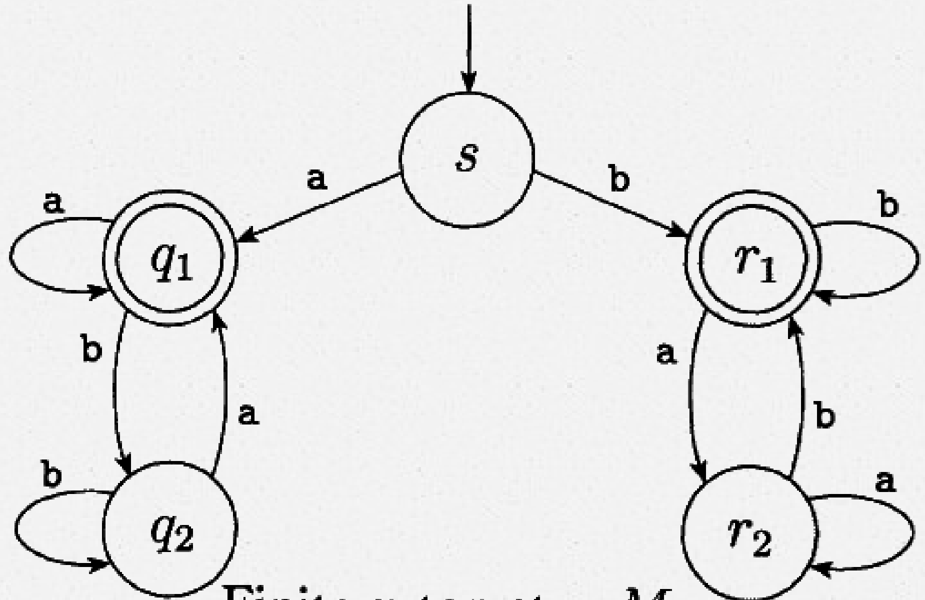
Example



The finite automaton M_1

- $L(M_1) = \{w \mid w$ contains at least one 1 and even number of 0s follow the last 1$\}$.

Example



Finite automaton M_4

- M_4 accepts all strings that start and end with a or with b.

Formal Definition

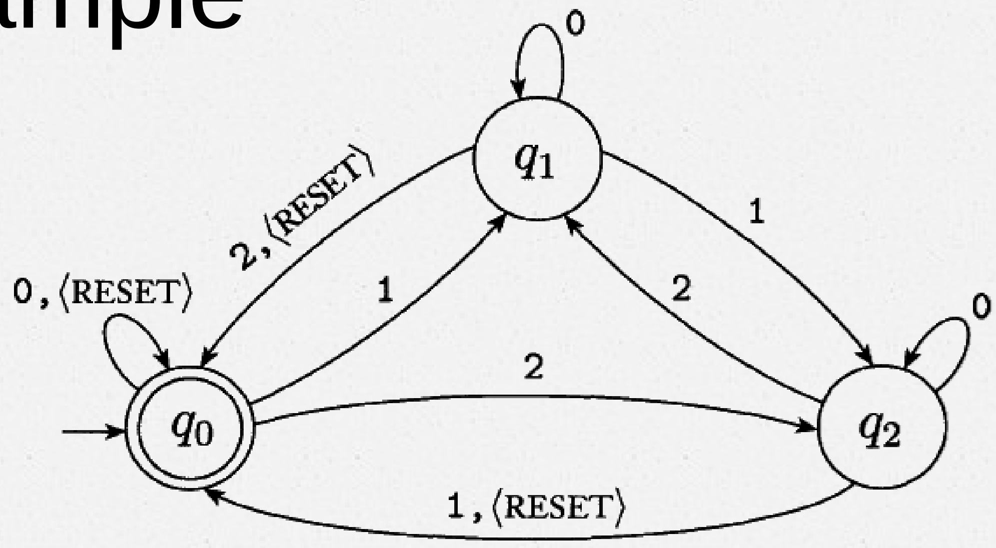
- $M = (Q, \Sigma, \delta, q_0, F)$
- $w = w_1 w_2 \ldots w_n$ $\quad \forall i,\ w_i \in \Sigma$

- M ***accepts*** $w \iff \exists\ r_0, r_1, \ldots, r_n \quad \forall i,\ r_i \in Q$
  1. $r_0 = q_0$,
  2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for i = 0, … , n-1,
  3. $r_n \in F$.

Regular Language

- A language is called a ***regular language*** if some finite automaton recognizes it.

Example



Finite automaton M_5

- $L(M_5) = \{w \mid$ the sum of the symbols in w is 0 modulo 3, except that <RESET> resets the count to 0$\}$.

✓ As M_5 recognizes this language, it is a regular language.

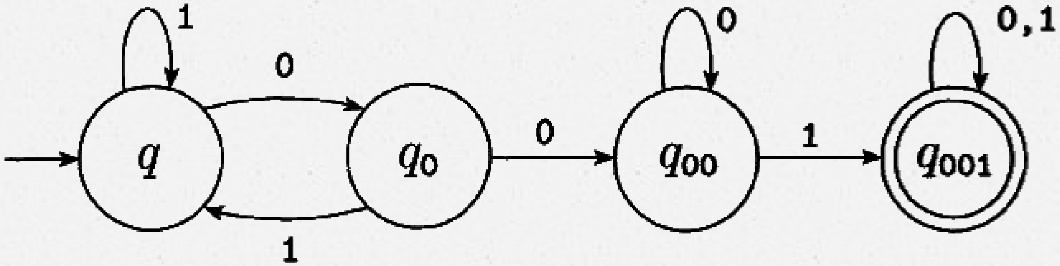
Designing Finite Automata

- Put *yourself* in the place of the machine and then see how you would go about performing the machine's task.

- Design a finite automaton to recognize the regular language of all strings that contain the string 001 as a substring.

Designing Finite Automata (cont.)

- There are four possibilities: You
  1. haven't just seen any symbols of the pattern,
  2. have just seen a 0,
  3. have just seen 00, or
  4. have seen the entire pattern 001.



Accepts strings containing 001

The Regular Operations

- Let A and B be languages. We define the regular operations ***union***, ***concatenation***, and ***star*** as follows.
  - **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
  - **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
  - **Star**: $A^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

Closure Under Union

- THEOREM

  The class of regular languages is closed under the union operation.

Proof

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, F_1)$ recognize A_1, and

  $M_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ recognize A_2.
- Construct $M = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.
  1. $Q = Q_1 \times Q_2$
  2. $\Sigma = \Sigma_1 \cup \Sigma_2$
  3. $\delta((r_1,r_2),a) = (\delta_1(r_1,a), \delta_2 (r_2,a))$.
  4. q_0 is the pair (q_1, q_2).
  5. F is the set of pair in which either members in an accept state of M_1 or M_2.

     $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$ $\qquad F \neq F_1 \times F_2$

Closure under Concatenation

- THEOREM

  The class of regular languages is closed under the concatenation operation.

- To prove this theorem we introduce a new technique called nondeterminism.

Nondeterminism

- In a ***nondeterministic*** machine, several choices may exit for the next state at any point.

- Nondeterminism is a generalization of determinism, so every deterministic finite automaton is automatically a nondeterministic finite automaton.

Differences between DFA & NFA

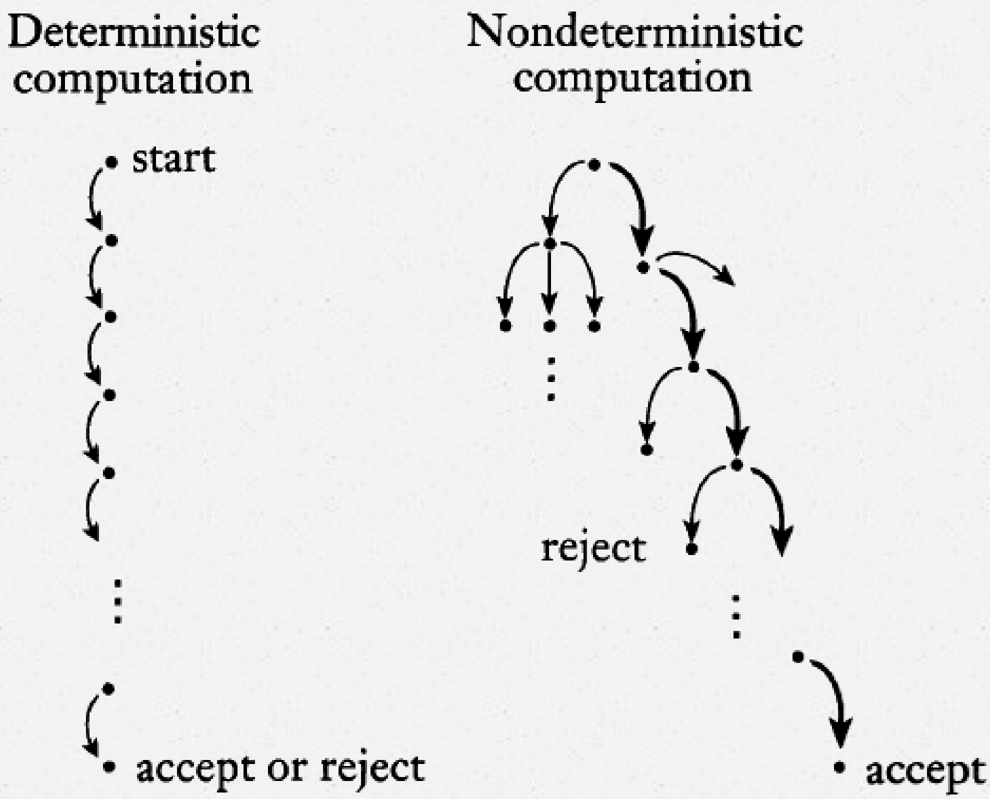
- First, very state of a DFA always has exactly one exiting transition arrow for each symbol in the alphabet.

  In an NFA a state may have zero, one, or more exiting arrows for each alphabet symbol.

- Second, in a DFA, labels on the transition arrows are symbols from the alphabet.

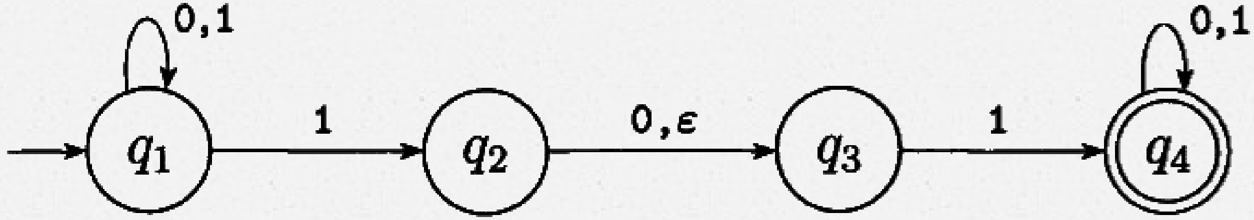
  An NFA may have arrows labeled with members of the alphabet or ε**.** Zero, one, or many arrows may exit from each state with the label ε.

Deterministic vs. Nondeterministic Computation



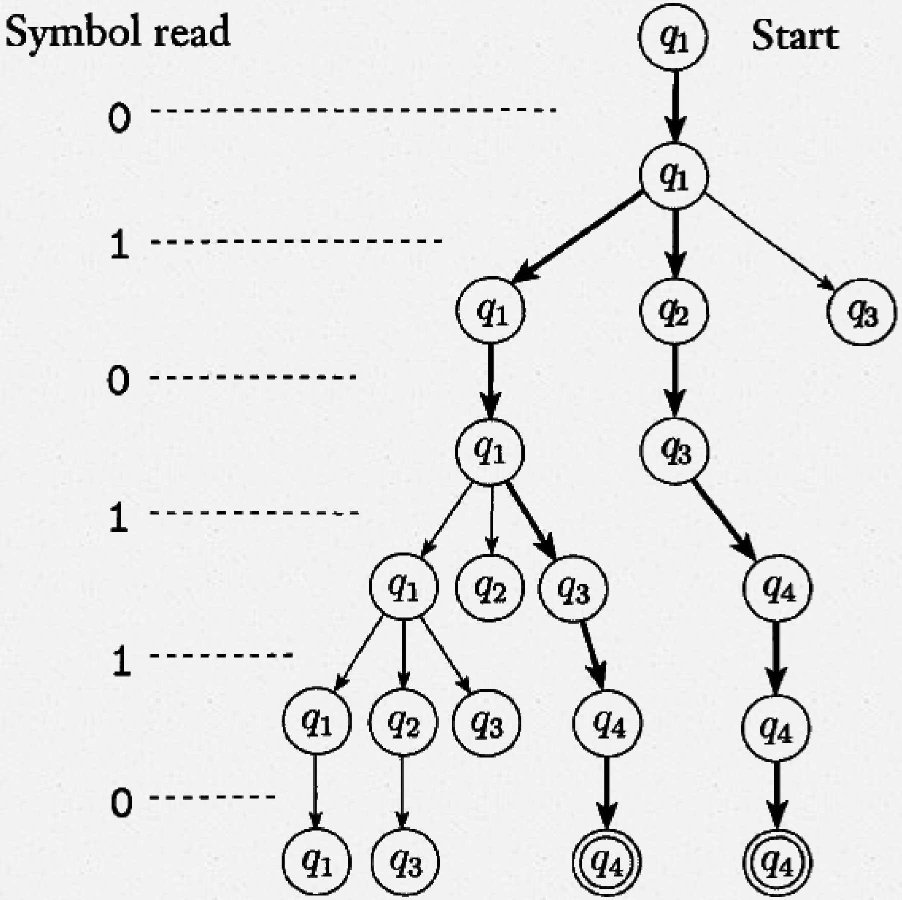
Example

- Consider the computation of N_1 on input 010110.



The nondeterministic finite automaton N_1

Example (cont.)



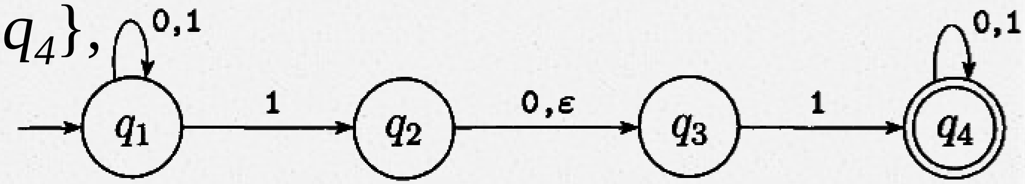
Formal Definition

- 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. $\delta : Q \times \Sigma_\varepsilon \longrightarrow P(\mathrm{Q})$ is the transition function,
  4. $q_0 \in Q$ is the start state, and
  5. $F \subseteq Q$ is the set of accept states.

Example

- $N_1 = (Q, \Sigma, \delta, q_0, F)$, where

1. $Q = \{q_1, q_2, q_3, q_4\}$,
2. $\Sigma = \{0,1\}$,
3. δ is given as



The nondeterministic finite automaton N_1

	0	1	ε
q_1	$\{q_1\}$	$\{q_1,q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_4\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

1. q_1 is the start state, and
2. $F = \{q_4\}$.

Equivalence of NFAs & DFAs

- THEOREM

  Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

- **PROOF IDEA** convert the NFA into an equivalent DFA that simulates the NFA.

  If k is the number of states of the NFA, so the DFA simulating the NFA will have 2^k states.

Proof

- Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing A. We construct a DFA $M = (Q', \Sigma', \delta', q_0', F')$ recognizing A.
- let's first consider the easier case wherein N has no ε arrows.

1. $Q' = P(Q)$.
2. $\delta'(R,a) = \bigcup_{r \in R} \delta(r,a)$.
3. $q_0' = \{q_0\}$.
4. $F' = \{R \in Q' \mid R$ contains an accept state of $N\}$.

Proof (cont.)

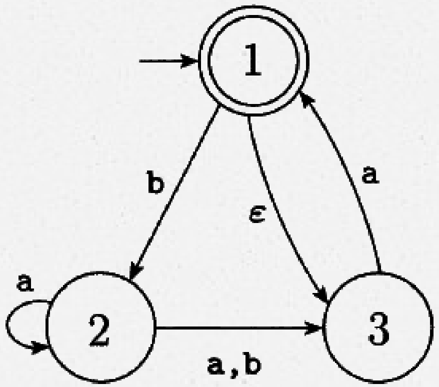
- Now we need to consider the ε arrows.
- for $R \subseteq Q$ let
- $E(R) = \{q \mid q$ can be reached from R by traveling along 0 or more ε arrows$\}$.

1. $Q' = P(Q)$.
2. $\delta'(R,a) = \{q \in Q \mid q \in E(\delta(r,a))$ for some $r \in R\}$.
3. $q_0' = E(\{q_0\})$.
4. $F' = \{R \in Q' \mid R$ contains an accept state of $N\}$.

Corollary

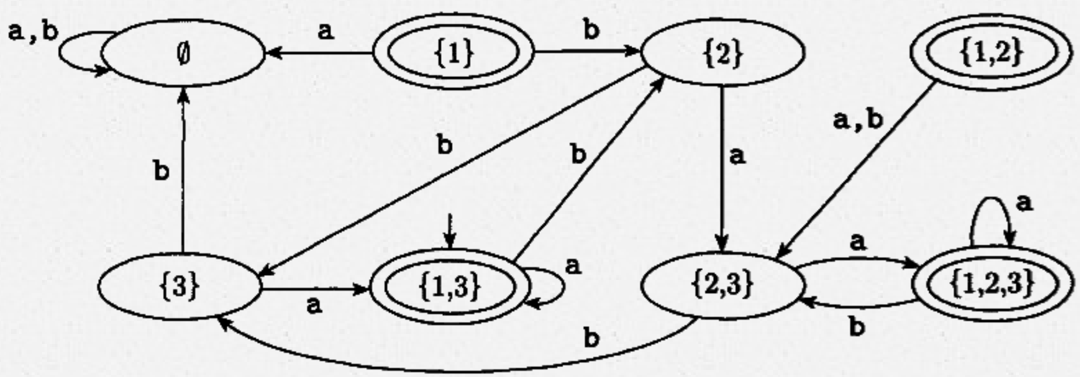
- A language is regular if and only if some nondeterministic finite automaton recognizes it.

Example



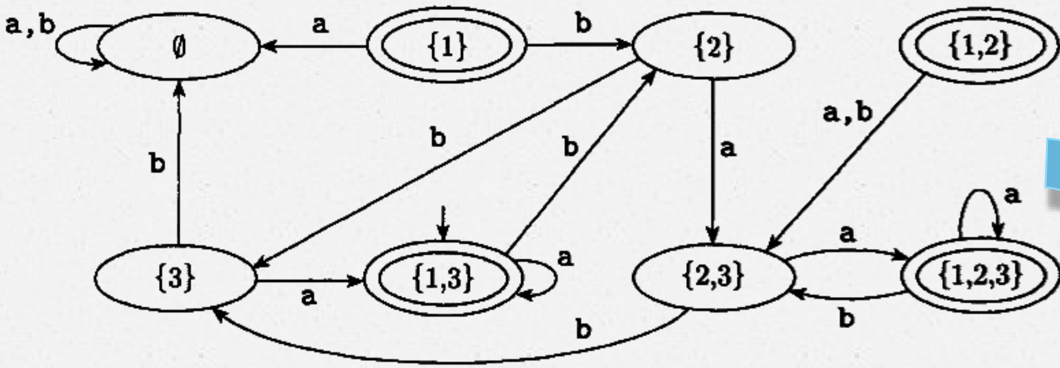
$N_4 = (Q, \{\mathrm{a,b}\}, \delta, 1, \{1\})$

- D's state set is
  {∅,{1},{2},{3},{1,2},{1,3},{2,3},{1,2,3}}.
- The start state is $E(\{1\}) = \{1,3\}$.
- The accept states are
  {{1},{1,2},{1,3},{1,2,3}}.

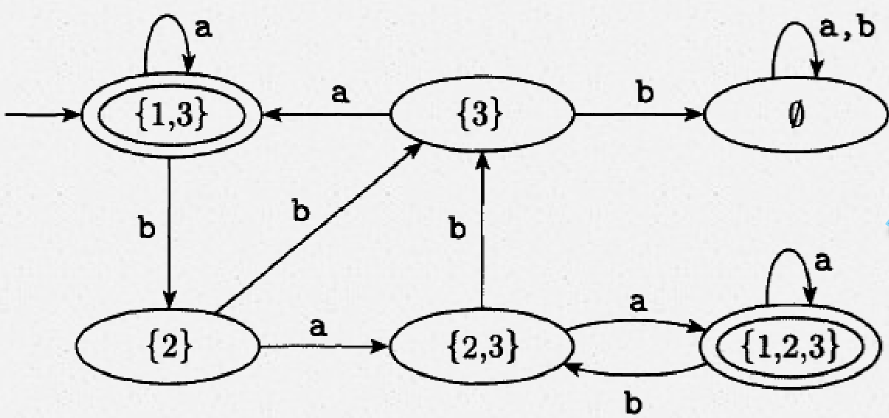


A DFA D that is equivalent to the NFA N_4

Example (cont.)



A DFA D that is equivalent to the NFA N_4



After removing unnecessary states

DFA minimization

- **DFA minimization** is the task of transforming a given deterministic finite automaton (DFA) into an equivalent DFA that has a minimum number of states.
- Here, two DFAs are called equivalent if they recognize the same regular languages.

Minimal DFA

- For each regular language, there also exists a **minimal automaton** that accepts it, that is, a DFA with a minimum number of states and this DFA is unique (except that states can be given different names).
- The minimal DFA ensures minimal computational cost for tasks such as pattern matching.

Minimal DFA (con.)

There are two classes of states that can be removed or merged from the original DFA without affecting the language it accepts to minimize it.

- **Unreachable states** are the states that are not reachable from the initial state of the DFA, for any input string.
- **Nondistinguishable states** are those that cannot be distinguished from one another for any input string.

Removing Unreachale States *for Automaton M*=(*Q*, Σ, δ, *q₀*, *F*)

```
let reachable_states := {q0};
let new_states := {q0};
do {
    temp := the empty set;
    for each q in new_states do
        for each c in Σ do
            temp := temp ∪ {p such that p = δ(q,c)};
        end;
    end;
    new_states := temp \ reachable_states;
    reachable_states := reachable_states ∪ new_states;
} while (new_states ≠ the empty set);
unreachable_states := Q \ reachable_states;
```

Nondistinguishable States

- One algorithm for merging the nondistinguishable states of a DFA, due to Hopcroft (1971), is based on partition refinement, partitioning the DFA states into groups by their behavior.
- These groups represent equivalent classes, whereby every two states of the same partition are equivalent if they have the same behavior for all the input sequences.

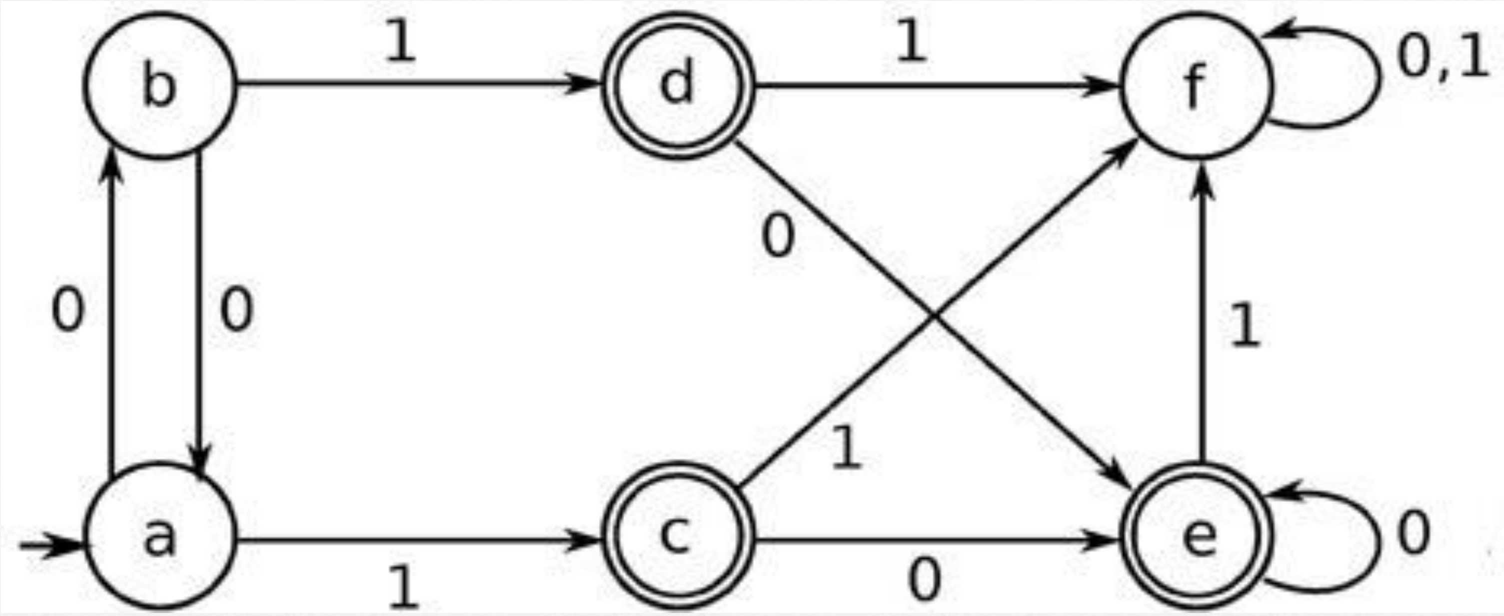
Hopcroft's algorithm

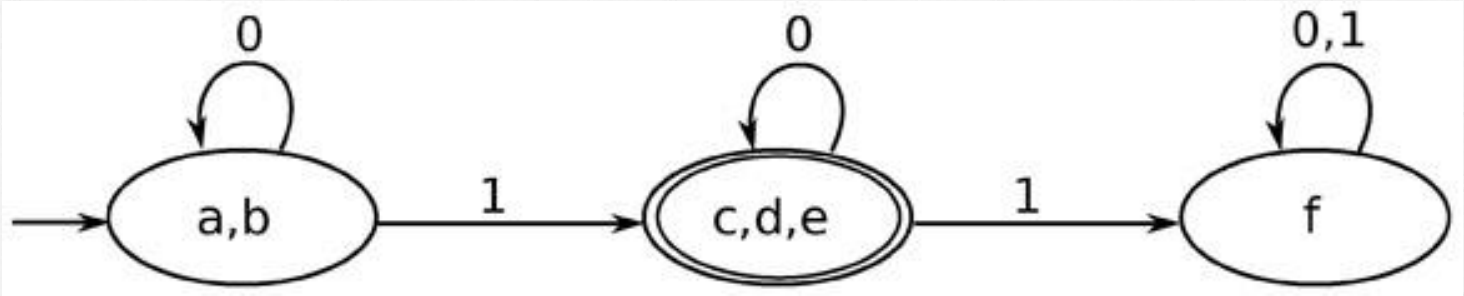
```
P := {F, Q \ F};
W := {F, Q \ F};
while (W is not empty) do
       choose and remove a set A from W
       for each c in Σ do
            let X be the set of states for which a transition on c
leads to a state in A
            for each set Y in P for which X ∩ Y is nonempty and Y \
X is nonempty do
            replace Y in P by the two sets X ∩ Y and Y \ X
            if Y is in W
                replace Y in W by the same two sets
            else
                if |X ∩ Y| <= |Y \ X|
                    add X ∩ Y to W
                else
                     add Y \ X to W
              end;
          end;
end;
```

Performance of Hopcroft's algorithm

- The worst-case running time of this algorithm is $O(ns \log n)$, where n is the number of states and s is the size of the alphabet.
- This remains the most efficient algorithm known for solving the problem, and for certain distributions of inputs its average-case complexity is even better, $O(n \log \log n)$.

Example



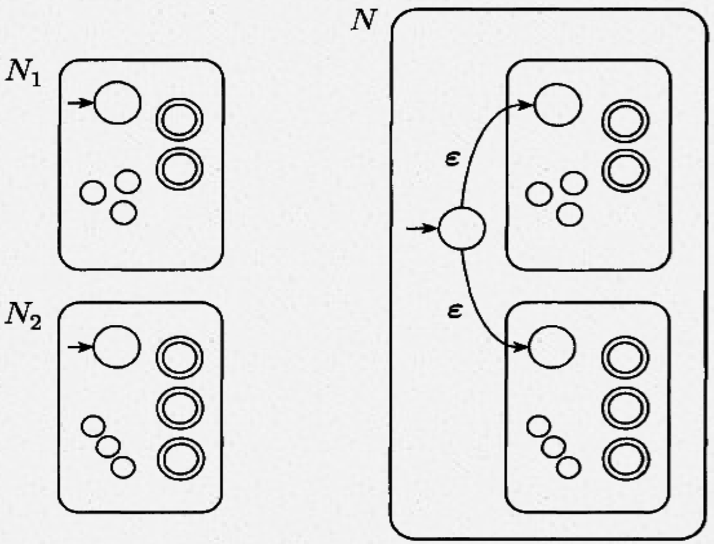


CLOSURE UNDER THE REGULAR OPERATIONS [Using NFA]

Closure Under Union

- The class of regular languages is closed under the Union operation.

Let NFA1 recognize A1 and NFA2 recognize A2.
Construct NFA3 to recognize A1 U A2.



Proof (cont.)

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

The state q_0 is the start state of N.

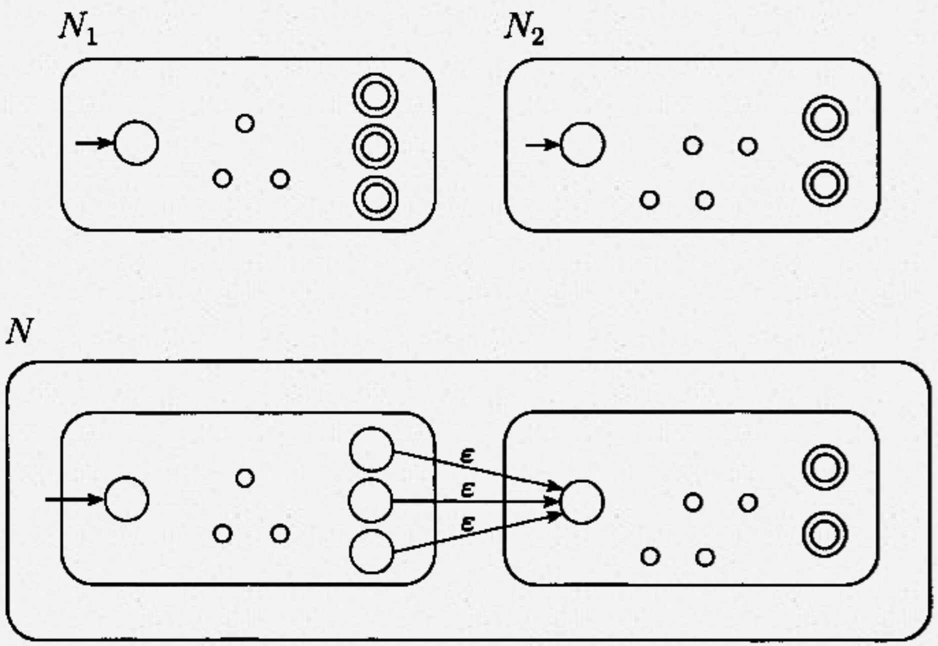
The accept states $F = F_1 \cup F_2$.

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

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

Closure Under Concatenation Operation

- The class of regular languages is closed under the concatenation operation.



Proof (cont.)

$Q = Q_1 \cup Q_2$.

The states of N are all the states of N_1 and N_2.

The state q_1 is the same as the start state of N_1.

The accept states F_2 are the same as the accept states of N_2.

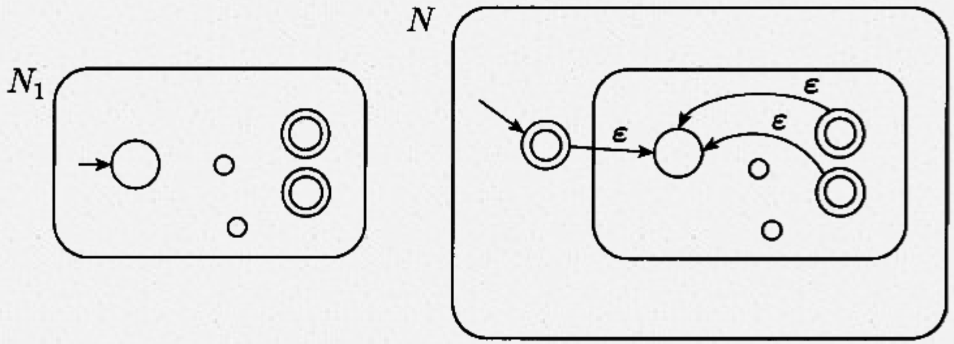
Define δ so that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$,

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

Closure Under Star operation

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

- We represent another NFA to recognize A*.



Proof (cont.)

1. $Q = \{q_0\} \cup Q_1$ The states of N are the states of N1 plus a new start state.
2. The state q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$
4. The accept states are the old accept states plus the new start state.
5. Define δ so that for any $q \in Q$ and $a \in \Sigma_\varepsilon$:

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

Regular Expression

- Say that R is a regular expression if R is:
  1. a for some a in the alphabet Σ
  2. ε
  3. $\emptyset$
  4. $(R_1 U R_2)$, where R_1 and R_2 are regular exp.
  5. $(R_1 o\, R_2)$, where R_1 and R_2 are regular exp.
  6. (R_1^*), where R_1 and R_2 are regular exp.

Recursive Definition?

Regular Expression Language

- Let R be a regular expression. L(R) is the language that is defined by R:
  1. if $R = a$ for $a \in \Sigma$ then $L(R)=\{a\}$
  2. if $R = \varepsilon$ then $L(R) = \{\varepsilon\}$
  3. if $R = \emptyset$ then $L(R) = \emptyset$
  4. if $R = R_1 U R_2$ then $L(R) = L(R_1)\ U\ L(R_2)$
  5. if $R = R_1\ o\ R_2$ then $L(R) = L(R_1)\ o\ L(R_2)$
  6. if $R = R_1^*$ then $L(R) = \left(L(R_1)\right)^*$

Examples(cont.)

1. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$
2. $\Sigma^* 1 \Sigma^* = \{w|w \text{ contains at least one } 1\}$
3. $0^* 10^* = \{w|w \text{ contains a single } 1\}$
4. $\Sigma^* 001 \Sigma^* = \{w|w \text{ contains } 001 \text{ as a substring}\}$
5. $01U10 = \{01,10\}$
6. $(\Sigma\Sigma)^* = \{w|w \text{ is a string of even length}\}$
7. $(\Sigma\Sigma\Sigma)^* = \{w|\text{the lentgh of w is a multiple of } 3\}$

8$(\Sigma\Sigma\Sigma\Sigma)^* = \{w|\text{the lentgh of w is a multiple } of\ 4\}$

9. $(0 \cup \varepsilon)1^* = 01^* \cup 1^*$
10. $(0 \cup \varepsilon)(1 \cup \varepsilon) \quad = \{\varepsilon, 0,1,01\}$
11. $1^* \emptyset = \emptyset$
12. $\emptyset^* = \{\varepsilon\}$

Equivalence of DFA and Regular Expression

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

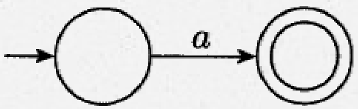
Lemma:

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

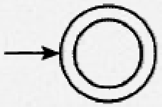
Building an NFA from the Regular Expression

- We consider the six cases in the formal definition of regular expressions

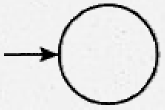
1. $R = a$ for some a in Σ. Then $L(R) = \{a\}$, and the following NFA recognizes $L(R)$.



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 $L(R)$.

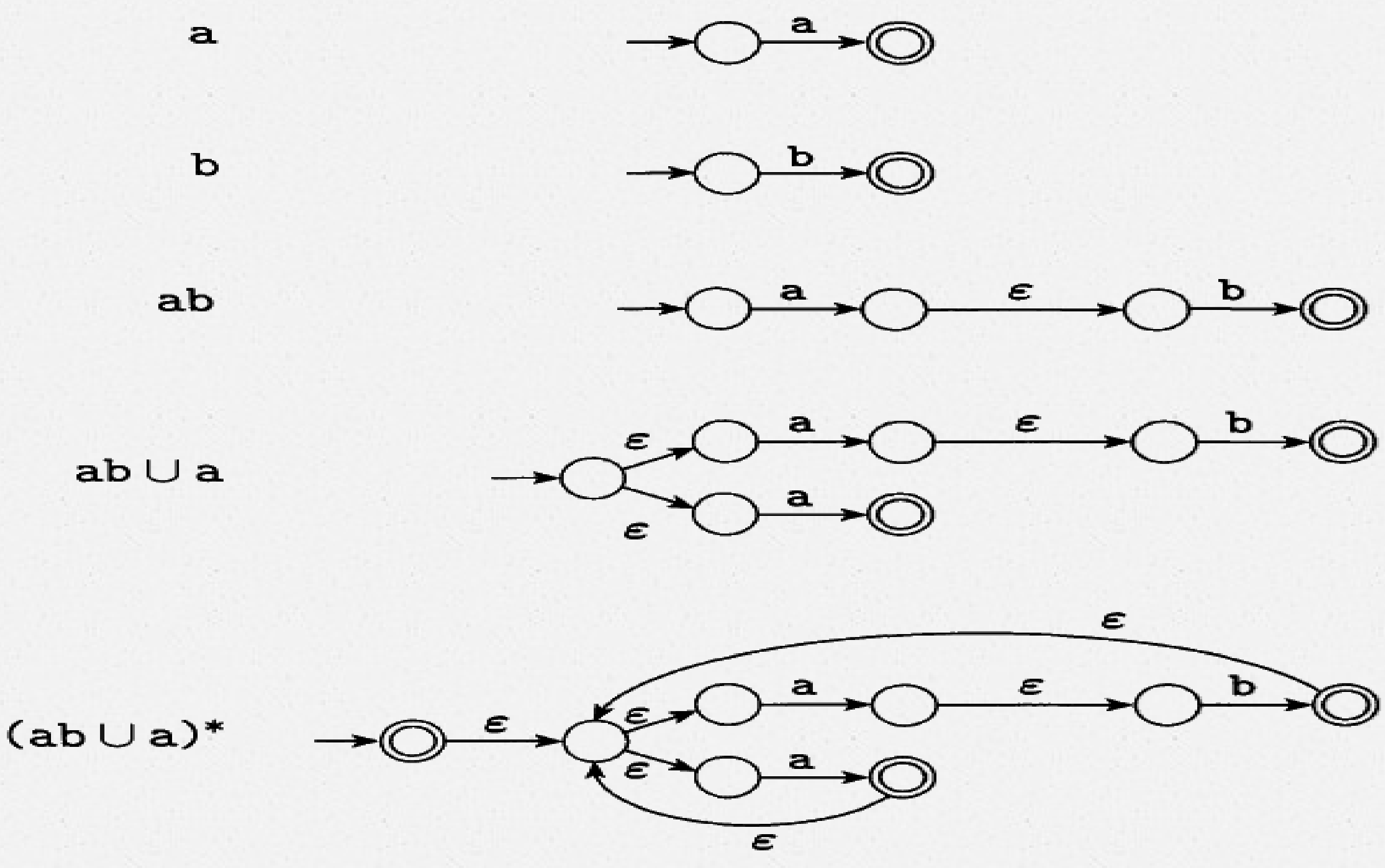


4. $R = R_1 \cup R_2$.

5. $R = R_1 \circ R_2$.

6. $R = R_1^*$.

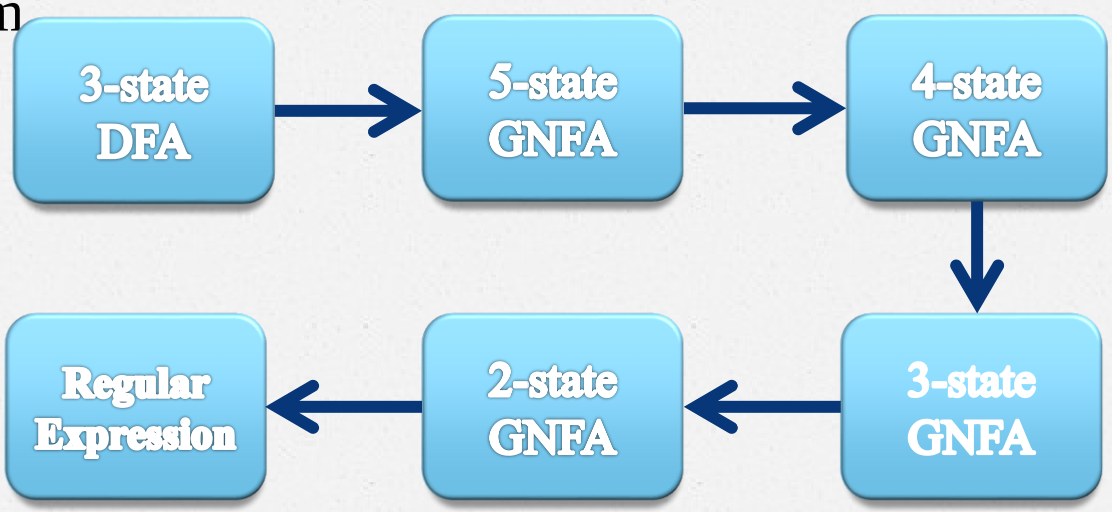
Examples



Building an NFA from the regular expression (ab ∪ a)*

Other direction of the proof

- We need to show that, if a language A is regular, a regular expression describes it!
- First we show how to convert DFAs into GNFAs, and then GNFAs into regular expressions.
- We can easily convert a DFA into a GNFA in the special form



Formal Definition

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

1. Q is a finite set called ***states***,
2. Σ is a the input ***alphabet***,
3. $\delta : (Q - \{q_{accept}\}) \times (\mathrm{Q} - \{Q_{start}\} \longrightarrow \boldsymbol{R}$ is the ***transition function***,
4. q_{start} is the ***start state***, and
5. q_{accept} is the ***accept state***.

Assumptions

For convenience we require that GNFAs always have a special form that meets the following conditions:

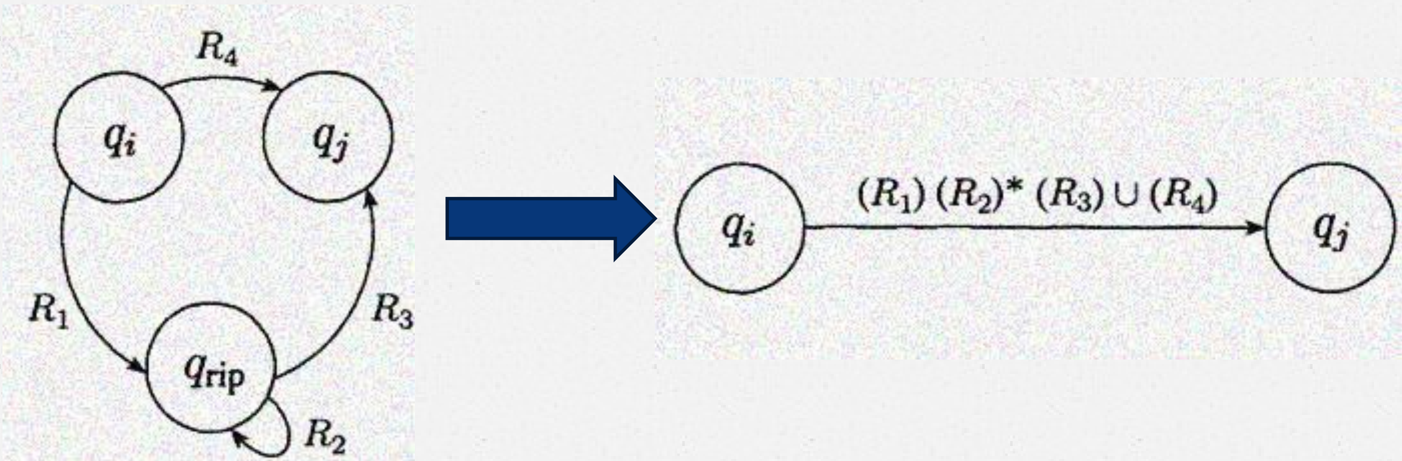
1. The start state has transition arrows going to **every** other state but **no** arrows coming in from any other state.
2. There is only a **single accept** state, and it has arrows coming in from every other state but no arrows going to any other state. Furthermore, the accept state is **not** the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

Acceptance of Languages for GNFA

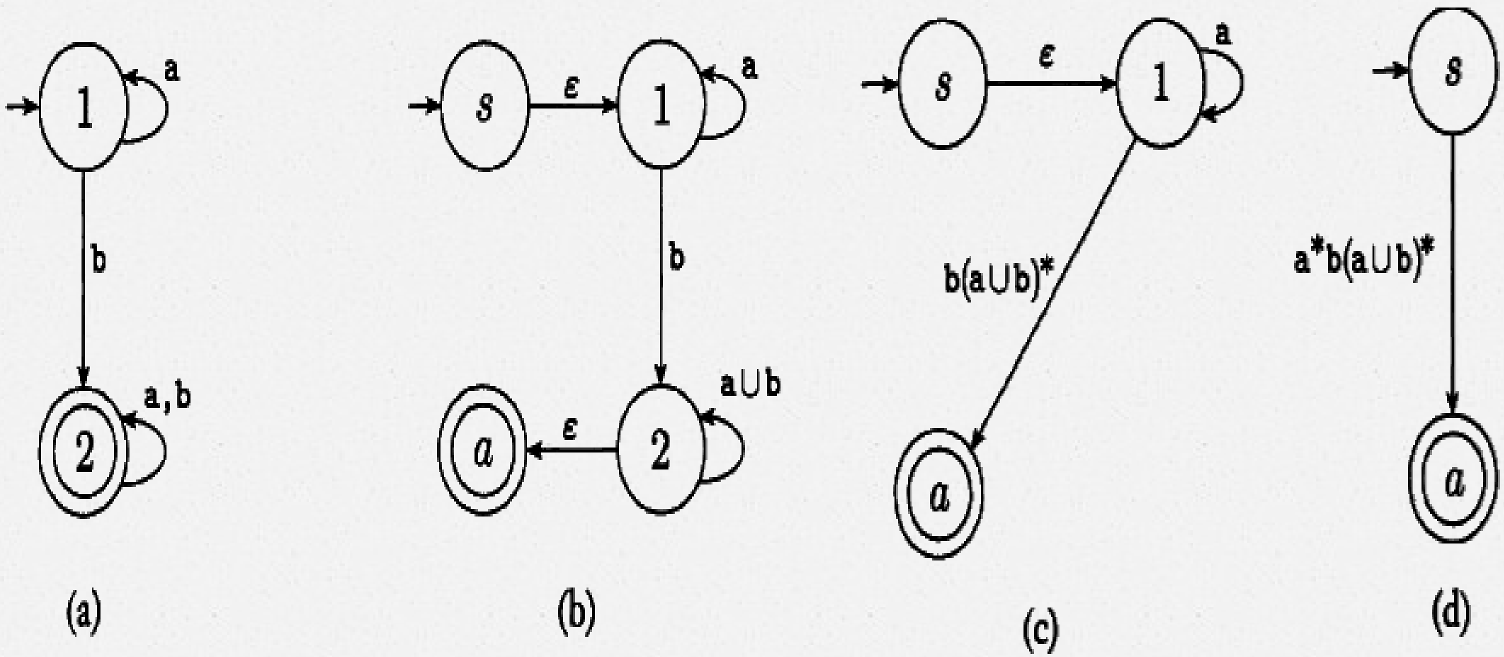
- A GNFA accepts a string w in Σ^* if $w = w_1\ w_2\ \ldots\ w_k$, where each w_i is in Σ^* is in Σ^* and a sequence of q_0, $q_1, \ldots, q_k$ exists such that

1. $q_0 = q_{start}$ is the start state,
2. $q_k = q_{accept}$ is the accept state, and
3. For each i, we have $w_i \in L(R_i)$ where $R_i = \delta(q_{i-1}, q_i)$; in other words R_i is the expression on the arrow from q_{i-1} to q_i.

How to Eliminate a State?

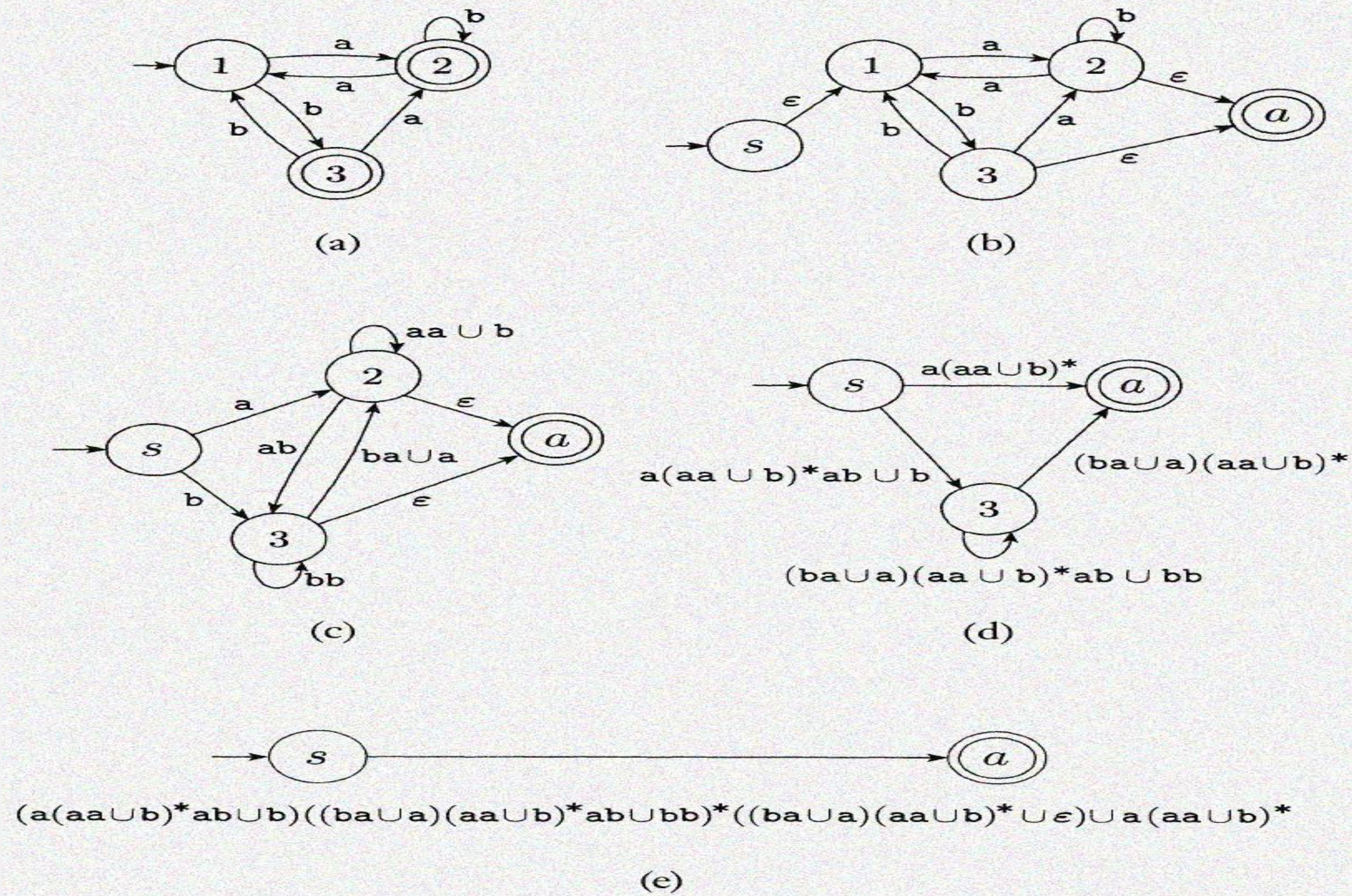


Example



Converting a two-state DFA to an equivalent regular expression

Example



1
2
3
a
b
a
b
b
a
(a)
s
1
2
3
a
ε
a
a
b
b
a
b
ε
ε
(b)
s
2
3
a
aa ∪ b
a
ab
ba∪a
b
ε
ε
bb
(c)
s
a
3
a(aa∪b)*
a(aa ∪ b)*ab ∪ b
(ba∪a)(aa∪b)*
(ba∪a)(aa ∪ b)*ab ∪ bb
(d)
s
a
(a(aa∪b)*ab∪b)((ba∪a)(aa∪b)*ab∪bb)*((ba∪a)(aa∪b)*∪ε)∪a(aa∪b)*
(e)

Grammar

- A *grammar* G is a 4-tuple

$$G = (V, \Sigma, R, S)$$

where:

1. V is a finite set of ***variables***,
2. Σ is a finite, disjoint from V, of ***terminals***,
3. R is a finite set of ***rules***,
4. S is the ***start*** variable.

Rule

A rule is of the form

$$x \rightarrow y$$

where $x \in (V \cup \Sigma)^{+}$ and $y \in (V \cup \Sigma)^{*}$

The rules are applied in the following manner: given a string w of the form

$$w = uxv,$$

We say that the rule $x \rightarrow y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv,$$

This is written as

$$\boxed{w \Rightarrow z.}$$

Derivation

- If

$$W_1 \Rightarrow \; W_2 \Rightarrow \; \ldots \Rightarrow W_n$$

we say that W_1 derives W_n and write

$$W_1 \Rightarrow^* W_n$$

Thus, we always have

$$W \Rightarrow^* W$$

Language of a Grammar

Let G = (V, Σ, R, S) be a grammar. Then, the set

$$\boxed{\boldsymbol{L(G) = \{W \in \Sigma^*:\ S \Rightarrow^* \ W\}}}$$

is the *language generated* by G.

Example

Consider the grammar

$$G = (\{S\}, \{a,b\}, P, S\}$$

with P given by

$$S \rightarrow aSb$$

$$S \rightarrow \varepsilon$$

Then

$$S \rightarrow aSb \rightarrow aaSbb \rightarrow aabb$$

So we can write

$$S \Rightarrow^{*} aabb$$

Then,

$$L(G) = \{a^n b^n : n \geq 0\}$$

A Notation for Grammars

Consider the grammar

$$G = (\{S\}, \{a,b\}, P, S\}$$

with P given by

$$S \rightarrow aSb$$

$$S \rightarrow \varepsilon$$

The above grammar is usually written as:

$$\boxed{\mathbf{G:\ \ S \rightarrow aSb \mid \varepsilon}}$$

Regular Grammar

A grammar G = (V, Σ, R, S) is said to be **right-linear** if all rules are of the form

$$A \rightarrow xB$$

$$A \rightarrow x$$

Where A, B ∈ V, and X∈ Σ*. A grammar is said to be **left-linear** if all rules are of the form

$$A \rightarrow Bx$$

$$A \rightarrow x$$

A **regular grammar** is one that is **either** right-linear **or** left-linear.

Theorem

Let $G = (V, \Sigma, R, S)$ be a right-linear grammar. Then:

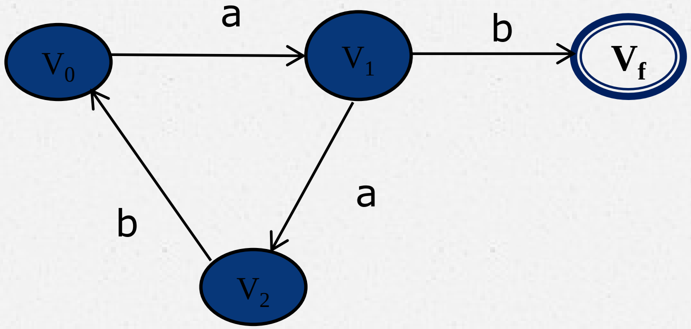
$\boxed{\textbf{L(G) is a regular language.}}$

Example

Construct a NFA that accepts the language generated by the grammar

$$V_0 \rightarrow aV_1$$

$$V_1 \rightarrow abV_0 \mid b$$



Theorem

Let **L** be a regular language on the alphabet Σ.

Then:

There exists a right-linear grammar G = (V, Σ, R, S)

$\boxed{\text{Such that } \mathbf{L = L(G).}}$

Theorem

Theorem A language is regular if and only if there exists a left-linear grammar G such that L = L(G).

Outline of the proof:

Given any left-linear grammar with rules of the form

$$A \to Bx$$

$$A \to x$$

We can construct a right-linear $\hat{G}$ by replacing every such rule of G with

$$A \to x^R B$$

$$A \to x^R$$

We have $L(G) = L(\hat{G})^R$.

Theorem

A language L is regular ***if and only*** if there exists a regular grammar G such that L = L(G).

$$L \text{ is } \boldsymbol{regular} \leftrightarrow \exists\, G : L = L(G); G \text{ is } \boldsymbol{regular}$$