FINITE AUTOMATA

Dr. Nadeem Akhtar

Assistant Professor
Department of Computer Science & IT
The Islamia University of Bahawalpur

PhD – Laboratory IRISA-UBS – University of South Brittany
European University of Brittany, Bretagne - France
http://www.univ-ubs.fr

AUTOMATA THEORY (1)

Automata theory is the study of abstract computing devices, or “machines”.

(Computational model – idealized computer)

A manageable mathematical theory can be set up directly on a Computational model.

The simplest computational model is called the ***finite state machine*** or ***finite automaton***.

AUTOMATA THEORY (2)

In 1930's A. Turing studied an abstract machine that had all the capabilities of today's computers. Turing goal was to describe precisely the boundary between ***what a computing machine could do and what it could not do***; his conclusions apply not only to his abstract *Turing machines*, but to today's real machines.

AUTOMATA THEORY (3)

In the 1940's and 1950's, simpler kinds of machines, which we today call "finite automata" were studied by a number of researchers.

AUTOMATA THEORY (4)

In the late 1950's, N. Chomsky began the study of formal "**grammars**" These grammars have close relationships to abstract automata and serve today as the basis of some important software components, including parts of compilers.

AUTOMATA THEORY (5)

In 1969, S. Cook extended Turing's study of what could and what could not be computed. Cook was able ***to separate those problems that can be solved efficiently by computer from those problems that can in principle be solved, but in practice take so much time that computers are useless for all but very small instances of the problem***. These class of problems is called "**intractable**" or "**NP-hard**".

AUTOMATA THEORY (6)

It is highly unlikely that the exponential improvement in computing speed that computer hardware has been following ***("Moore's Law") will have significant impact on our ability to solve large instances of intractable problems***.

AUTOMATA THEORY (7)

What computer scientists do today depend on the theoretical developments.

Finite automata and certain kinds of formal grammars, are used in the design and construction of important kinds of software.

Turing machine help us understand what we can expect from our software.

AUTOMATA THEORY (8)

1- Software for designing and checking the behavior of digital circuits.

2- The "**lexical analyzer**" of a compiler, that is compiler component that breaks the input text into logical units, such as identifiers, keywords and punctuation.

3- Software for scanning large bodies of text, such as collection of Web pages, to find occurrences of words, phrases, or other patterns.

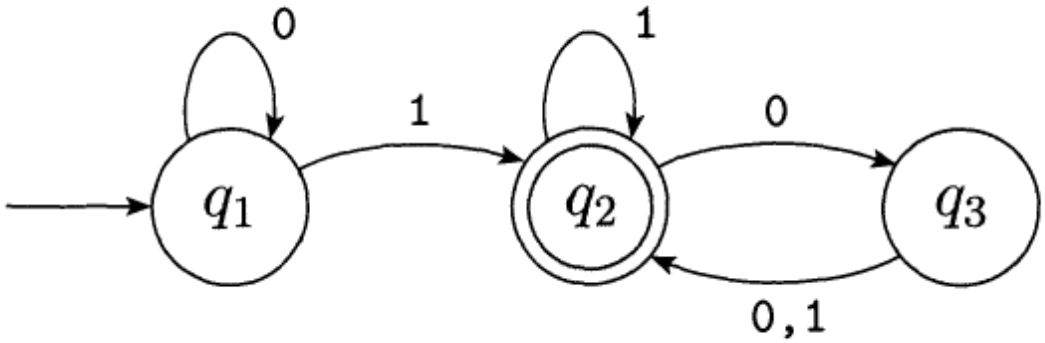
4- Software for verifying systems of all types that have **finite number of distinct states**, such as communications protocols or protocols for secure exchange of information.

FORMAL DEFINITION OF A FINITE AUTOMATON

A ***finite automaton*** is a 5-tuple (**Q, Σ, δ, q_o, F**) where

1. **Q** is a finite set called the ***states,***
2. **Σ** is a finite set called the ***alphabet,***
3. **δ: *Q x Σ* → *Q*** is the ***transition function,***
4. **q_o** ∈ Q is the ***start state***, and
5. **F ⊆ Q** is the ***set of accept states***.

The finite automaton M_1 (1/2)



We can describe M_1 formally by writing $M_1 = (Q, \Sigma, \delta, q_1, F)$, 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

4. q_1 is the start state, and

5. $F = \{q_2\}$.

The finite automaton M_1 (2/2)

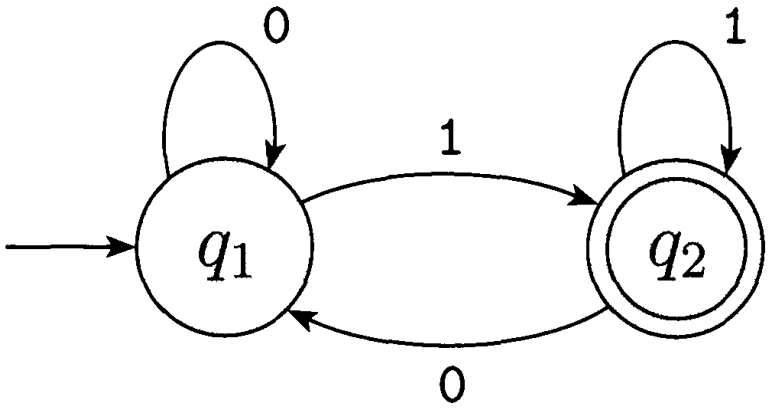
A = { w|w contains at least one 1 and

an even number of 0s follow the last 1}.

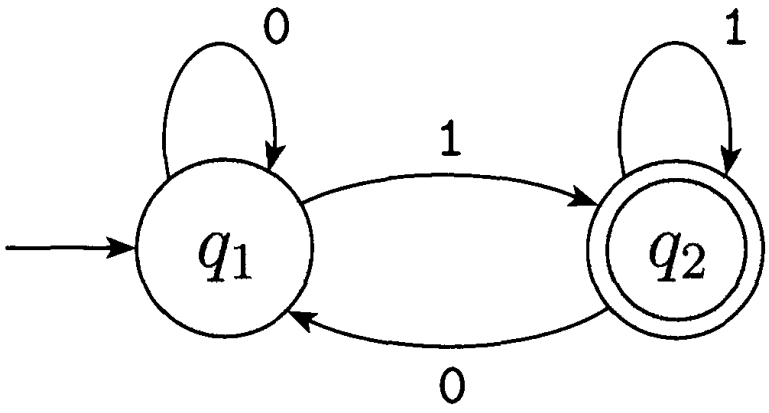
Then **$L(M_1) = A$**, or equivalently,

M_1 recognizes A.

The finite automaton M_2



The finite automaton M_2

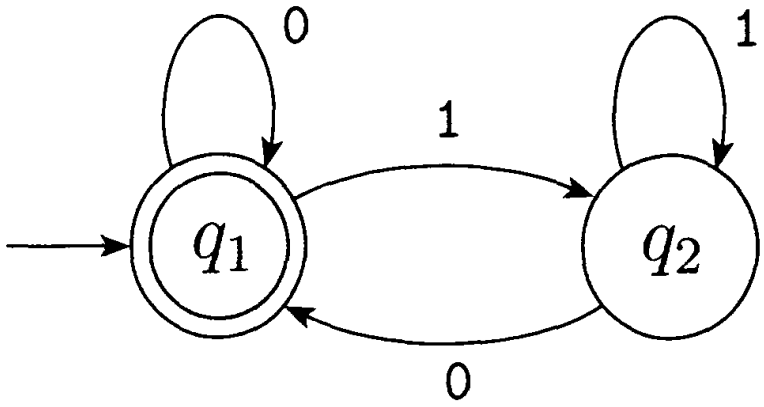


$$M_2 = \big(\{q_1, q_2\}, \{0,1\}, \delta, q_1, \{q_2\}\big).$$

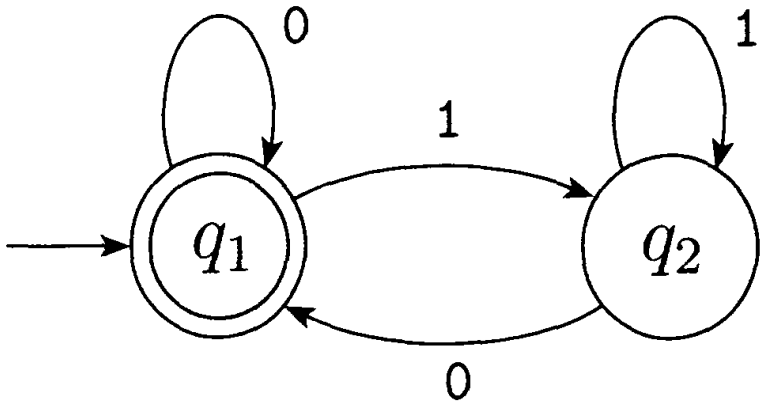
	0	1
q_1	q_1	q_2
q_2	q_1	q_2

$$L(M_2) = \{w | \ w \text{ ends in a } 1\}$$

The finite automaton M_3

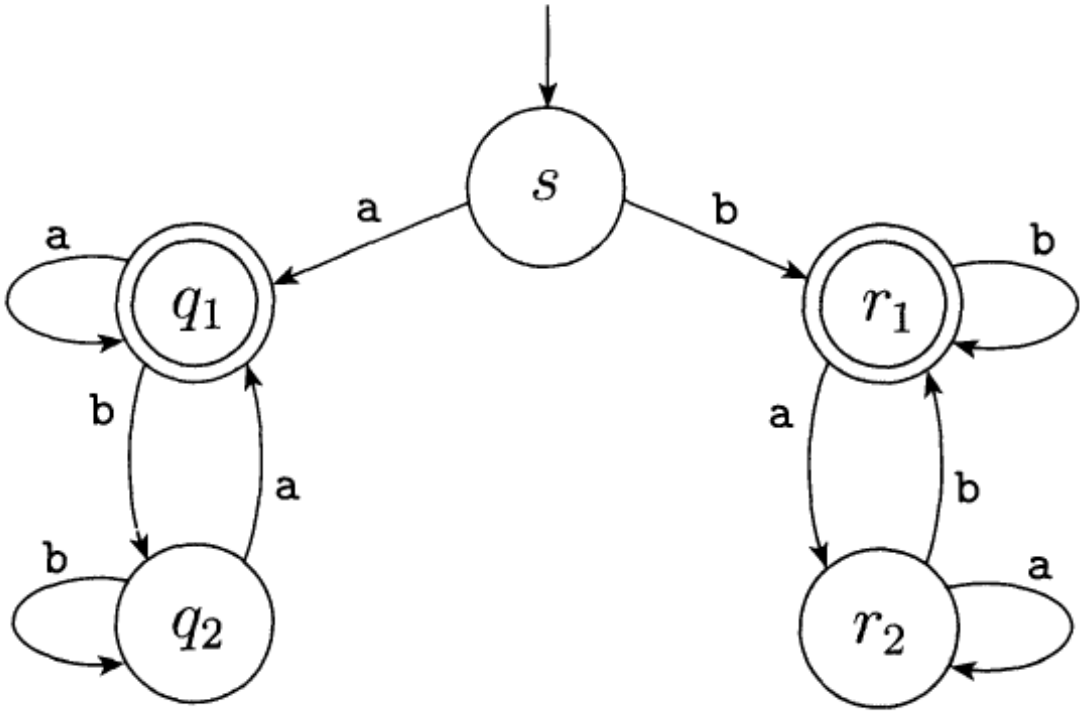


The finite automaton M_3

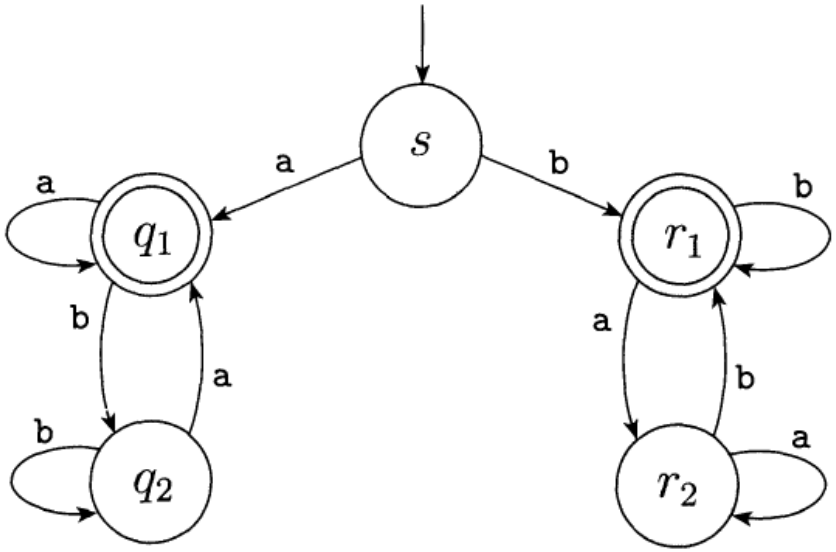


$L(M_3) = \{w |\ w$ is the empty string ε or ends in a $0\}$.

The finite automaton M_4

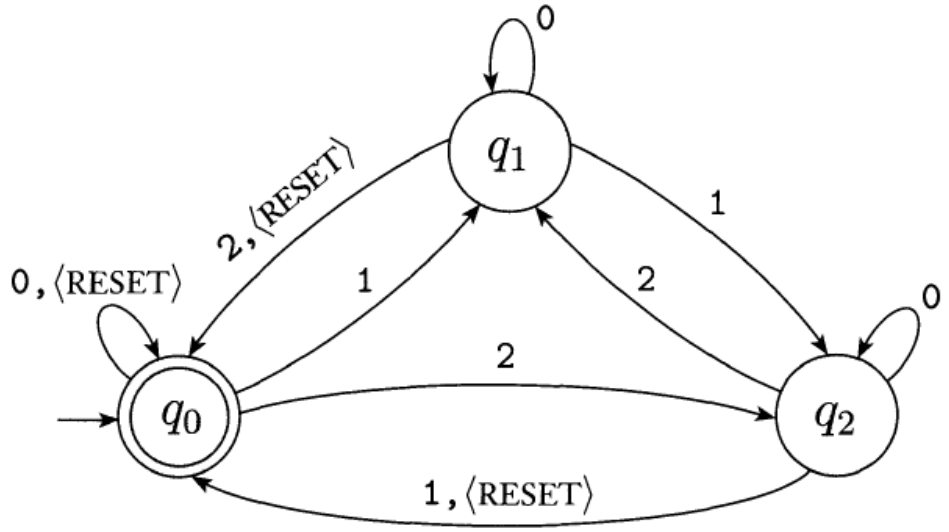


The finite automaton M_4



Machine **M_4** has two accept states q_1 and r_1.
Alphabet $\Sigma = \{a, b\}$
Accept strings are a, b, aa, bb, abs, bab
M_4 accepts all strings that start or end with a or that start or end with b.
M_4 accepts all strings that start or end with the same symbol.

The finite automaton M_5



Alphabet Σ = {0, 1, 2, <RESET>}

Machine **M_5** keeps a running count of the sum of the numerical input symbols it reads, modulo 3.

Every time it receives <RESET> symbol it resets the count to 0.

It accepts if the sum is 0, modulo 3, or in other words if the sum is a multiple of 3.

DESIGNING FINITE AUTOMATA (1)

"Reader as Automaton" method

- You are given some language and want to design a finite automaton that recognizes it. Pretending to be the automaton, you receive an input string and must determine whether it is a member of the language the automaton is supposed to recognize.

DESIGNING FINITE AUTOMATA (2)

- You get to see the symbols in the string one by one. After each symbol you must decide whether the string seen so far is in the language. The reason is that you, like the machine, don't know when the end of the string is coming so you must always be ready with the answer.

DESIGNING FINITE AUTOMATA (3)

- Finite automaton is a type of machine having only finite number of states, which means a finite memory.

DESIGNING FINITE AUTOMATA (4)

Example:

- Alphabet is {0, 1} and
- The language consists of all strings with an odd number of 1s.

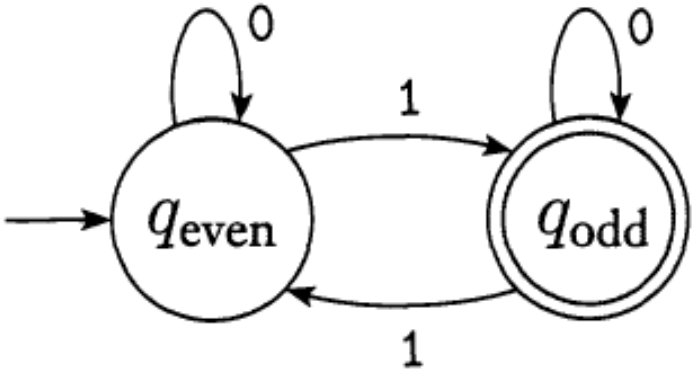
Construct a finite automaton E_1 to recognize this language

DESIGNING FINITE AUTOMATA (5)

Example:

- Alphabet is {0, 1} and
- The language consists of all strings with an odd number of 1s.

Construct a finite automaton E_1 to recognize this language



DESIGNING FINITE AUTOMATA (6)

Example:

Design a finite automaton E_2 to recognize the regular language of all strings that contain the string 001 as a substring.

For example, 0010, 1001, 001, and 11111110011111 are all language, but 11 and 0000 are not.

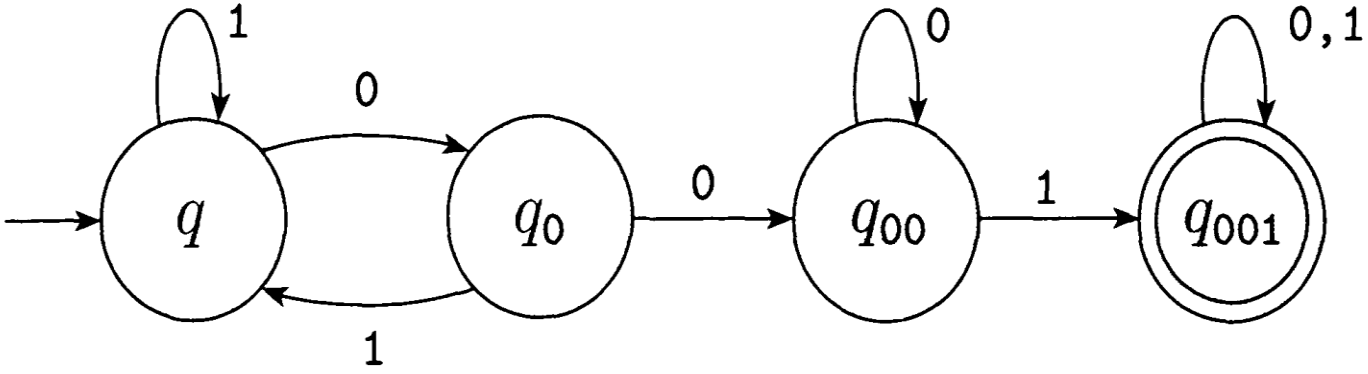
So there are four possibilities:

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.

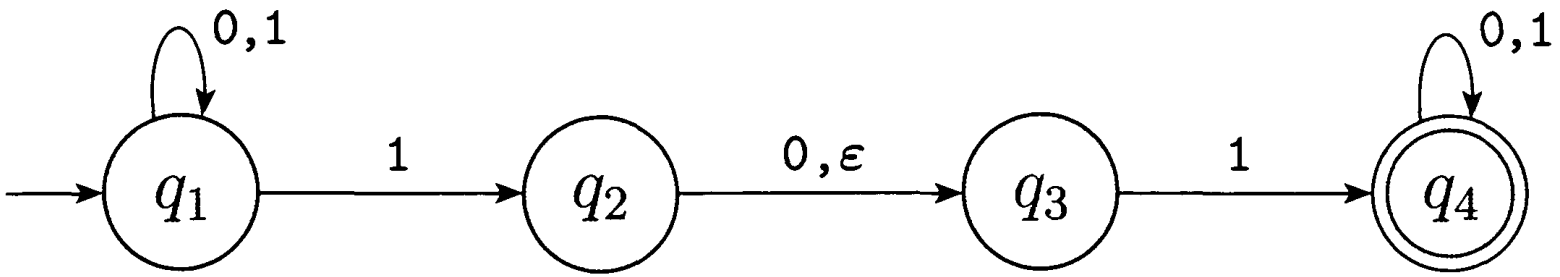
Assign the states q, q_0, q_{00}, and q_{001} to these possibilities.

DESIGNING FINITE AUTOMATA (7)

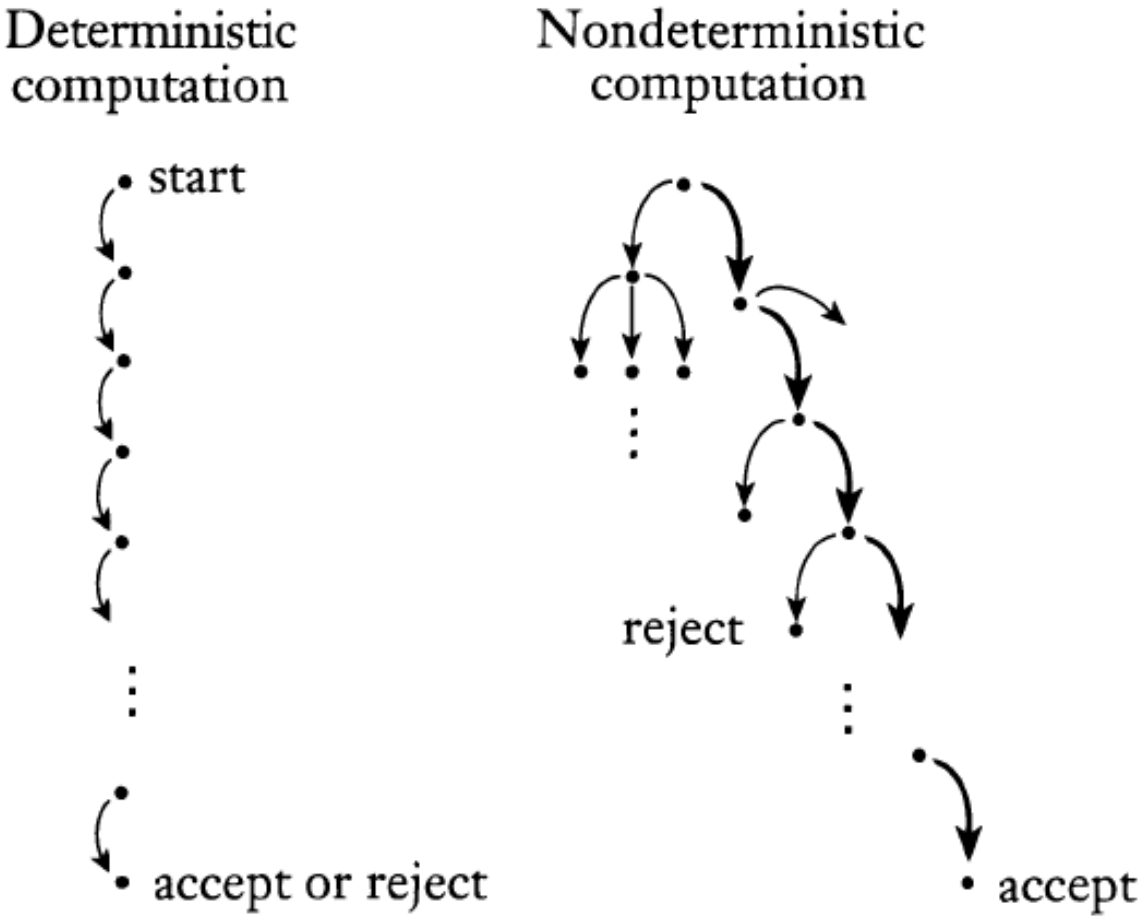
Accepts strings containing 001



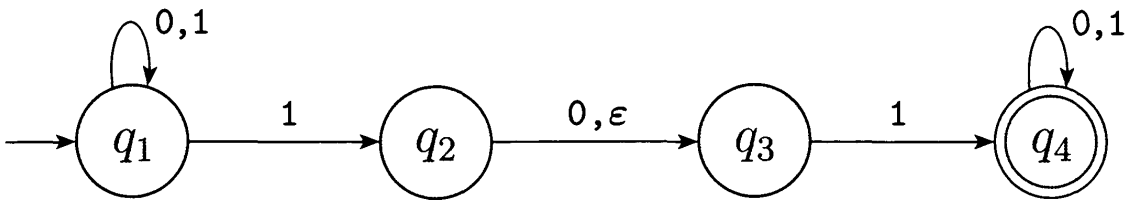
Nondeterministic Finite Automaton N_1

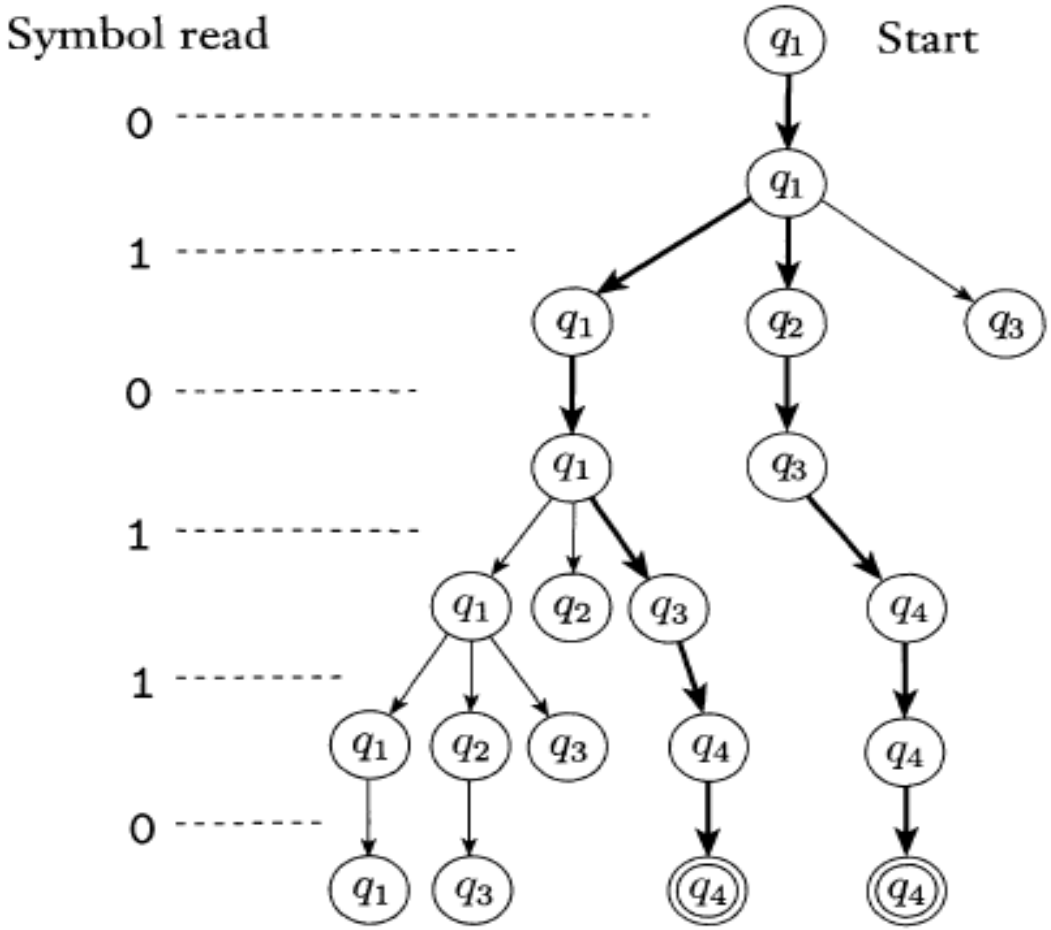


Deterministic and Non-Deterministic Computations with an accepting branch

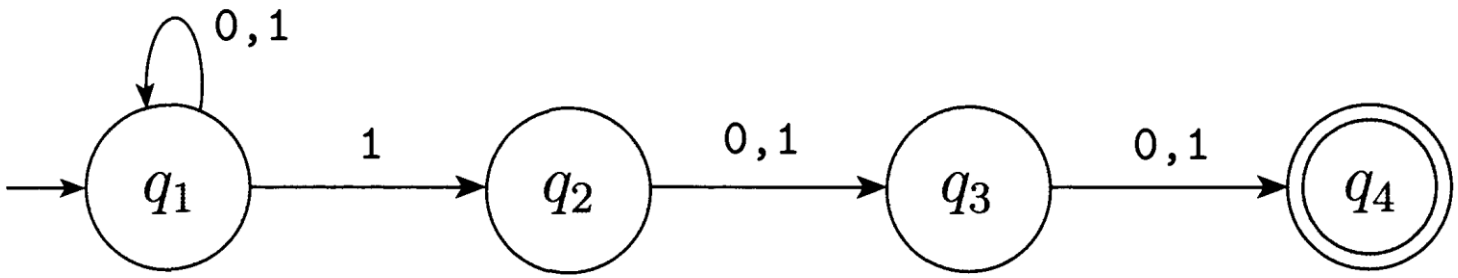


The computation of N_1 on input **010110**

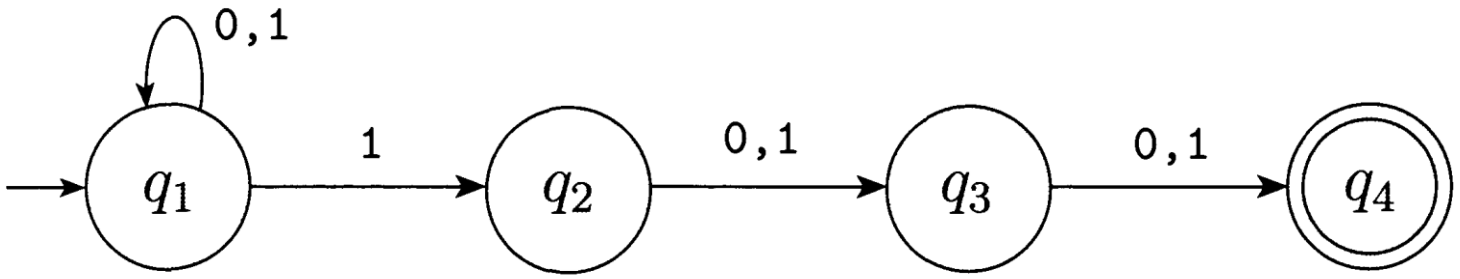




Nondeterministic Finite Automaton N_2



Nondeterministic Finite Automaton N_2

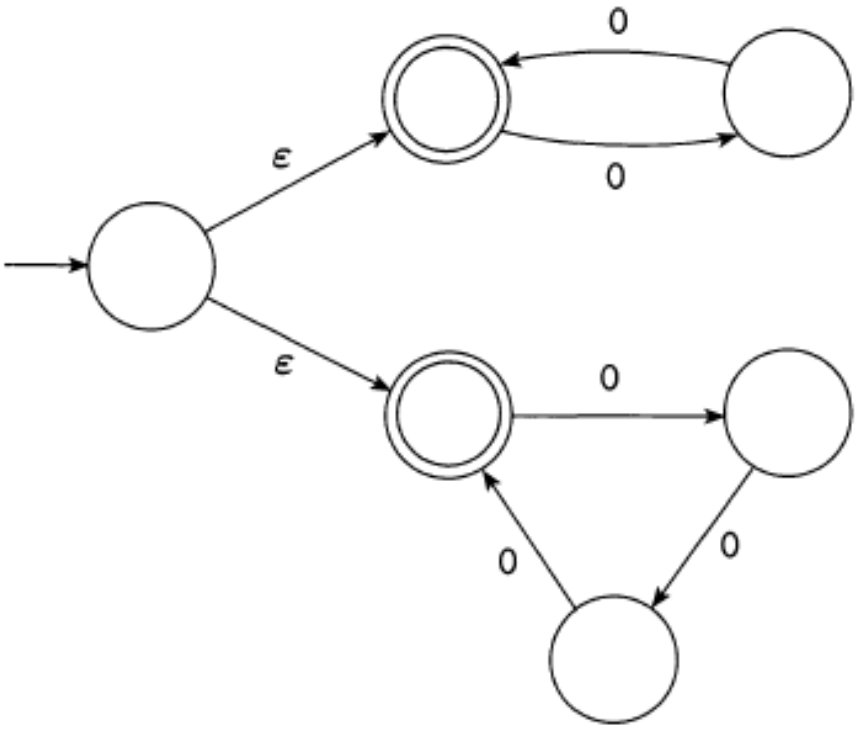


Let A be the language consisting of all strings over {0, 1} containing a 1 in the third position from the end

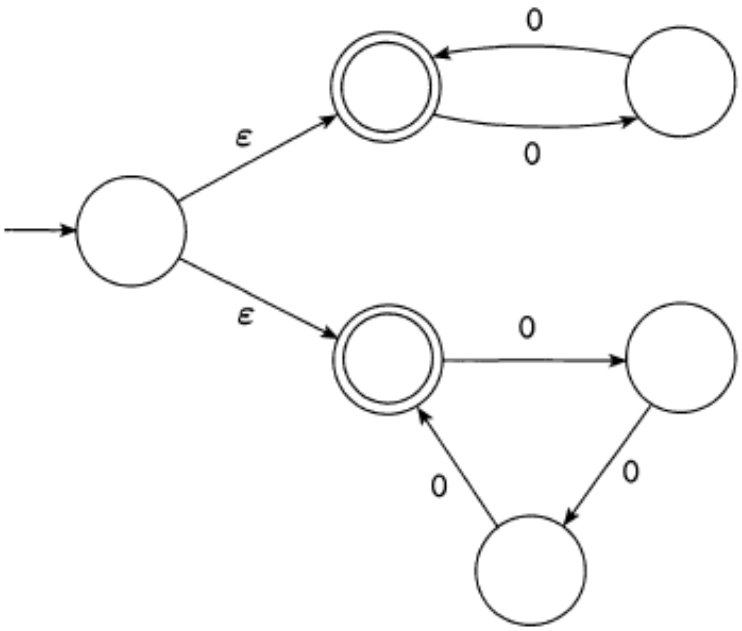
e.g. 000100, 00111, 00101

NFA N_3

NFA N_3 that has an input alphabet {0} consisting of a single symbol. An alphabet containing only one symbol is called a ***unary alphabet.***



NFA N_3

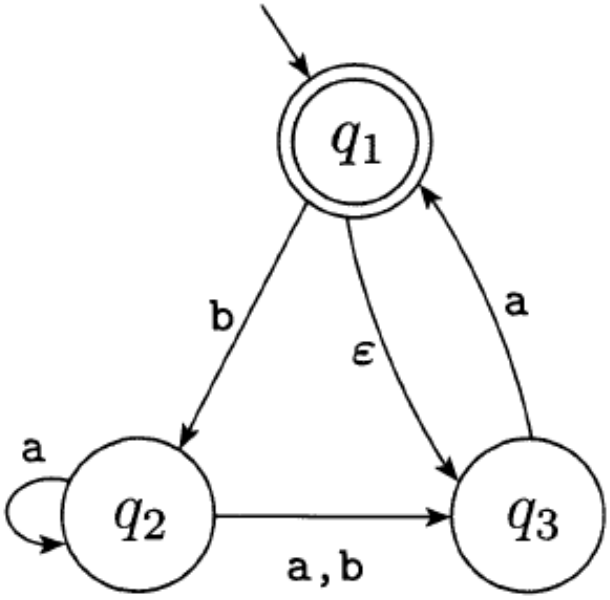


This machine N_3 accepts all strings of the form 0^k where k is a multiple of 2 or 3.

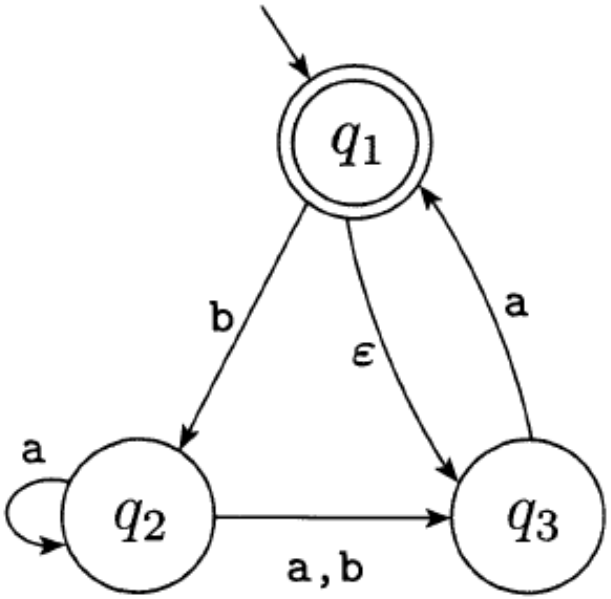
(**Note**: Superscript denotes repetition, not numerical exponentiation)

N_3 accepts the strings , 00, 000, 0000, and 000000, but not 0 or 00000.

NFA N_4



NFA N_4



This machine N_4 accepts the strings *ε, a, baba,* and *baa,*

It does not accept the strings b, bb, and babba.

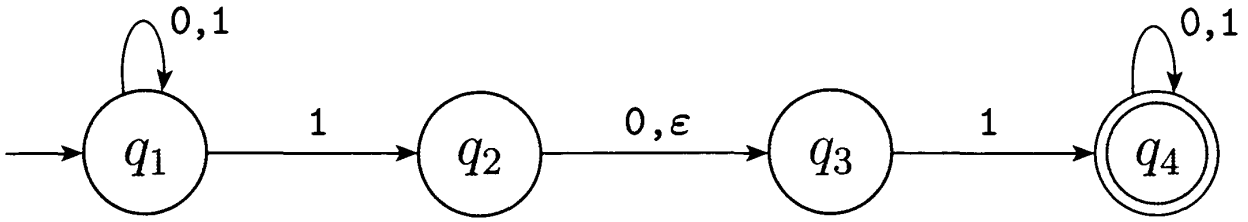
Formal definition of a Nondeterministic Finite Automaton (NFA)

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

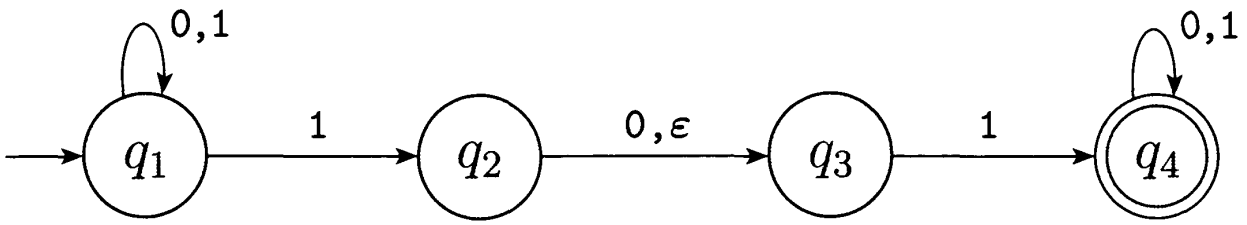
1. **Q** is a finite set called the ***states,***
2. **Σ** is a finite set called the ***alphabet,***
3. $\boldsymbol{\delta: Q \times \Sigma_\varepsilon \rightarrow P(Q)}$ is the ***transition function,***
4. $\boldsymbol{q_o \in Q}$ is the ***start state***, and
5. $\boldsymbol{F \subseteq Q}$ is the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

Note: $\boldsymbol{\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}}$

NFA N_1



NFA N_1



The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, where

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

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

,

4. q_1 is the start state, and
5. $F = \{q_4\}$.

Formal definition of computation for an NFA

The formal definition of computation for an NFA is similar to that for a DFA. Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA and w a string over the alphabet Σ. Then we say that N ***accepts*** w if we can write w as $w = y_1 y_2 \cdots y_m$, where each y_i is a member of Σ_ε and a sequence of states $r_0, r_1, \ldots, r_m$ exists in Q with three conditions:

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

Condition 1 says that the machine starts out in the start state. Condition 2 says that state r_{i+1} is one of the allowable next states when N is in state r_i and reading y_{i+1}. Observe that $\delta(r_i, y_{i+1})$ is the *set* of allowable next states and so we say that r_{i+1} is a member of that set. Finally, condition 3 says that the machine accepts its input if the last state is an accept state.

Equivalence of NFAS and DFAS

Deterministic and nondeterministic finite automata recognize the same class of languages. (Regular languages)

- Equivalence of NFAs and DFAs is useful because describing an NFA for a given language sometimes is much easier than describing a DFA for that language.

Two machines are equivalent if they recognize the same language.

PROOF (1)

Theorem-1

Every nondeterministic finite automaton has an equivalent deterministic finite automaton.

PROOF IDEA:

If a language is recognized by an NFA, then we must show the existence of a DFA that also recognizes it. The idea is to convert the NFA into an equivalent DFA that simulates the NFA.

PROOF (2)

If k is the number of states of the NFA, it has $\mathbf{2^k}$ subsets of states. Each subset corresponds to one of the possibilities that the DFA must remember, so the DFA simulating the NFA will have $\mathbf{2^k}$ states.

Now we need to figure out which will be the start state and accept states of the DFA, and what will be its transition function.

PROOF (3)

PROOF Let $N = (Q, \Sigma, \delta, q_0, F)$ be the NFA recognizing some language A. We construct a DFA $M = (Q', \Sigma, \delta', q_0', F')$ recognizing A. Before doing the full construction, let's first consider the easier case wherein N has no ε arrows. Later we take the ε arrows into account.

1. $Q' = \mathcal{P}(Q)$.
   Every state of M is a set of states of N. Recall that $\mathcal{P}(Q)$ is the set of subsets of Q.
2. For $R \in Q'$ and $a \in \Sigma$ let $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$.
   If R is a state of M, it is also a set of states of N. When M reads a symbol a in state R, it shows where a takes each state in R. Because each state may go to a set of states, we take the union of all these sets. Another way to write this expression is

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).$$

PROOF (4)

3. $q_0' = \{q_0\}$.
   M starts in the state corresponding to the collection containing just the start state of N.
4. $F' = \{R \in Q' \mid R \text{ contains an accept state of } N\}$.
   The machine M accepts if one of the possible states that N could be in at this point is an accept state.

The notation $\bigcup_{r \in R} \delta(r, a)$ means: the union of the sets $\delta(r, a)$ for each possible r in R.

PROOF (5)

Now we need to consider the ε arrows. To do so we set up an extra bit of notation. For any state R of M we define $E(R)$ to be the collection of states that can be reached from R by going only along ε arrows, including the members of R themselves. Formally, for $R \subseteq Q$ let

$$E(R) = \{q |\ q \text{ can be reached from } R \text{ by traveling along 0 or more } \varepsilon \text{ arrows}\}.$$

Then we modify the transition function of M to place additional fingers on all states that can be reached by going along ε arrows after every step. Replacing $\delta(r, a)$ by $E(\delta(r, a))$ achieves this effect. Thus

$$\delta'(R, a) = \{q \in Q |\ q \in E(\delta(r, a)) \text{ for some } r \in R\}.$$

PROOF (6)

Additionally we need to modify the start state of M to move the fingers initially to all possible states that can be reached from the start state of N along the ε arrows. Changing q_0' to be $E(\{q_0\})$ achieves this effect. We have now completed the construction of the DFA M that simulates the NFA N.

The construction of M obviously works correctly. At every step in the computation of M on an input, it clearly enters a state that corresponds to the subset of states that N could be in at that point. Thus our proof is complete.

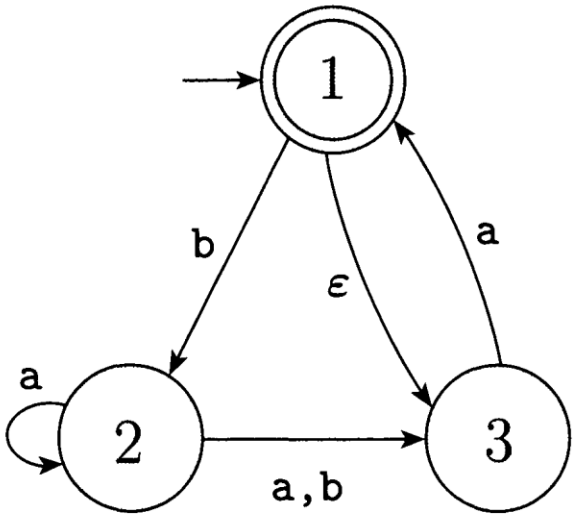
A language is regular if and only if some nondeterministic finite automaton recognizes it.

One direction of the "if and only if" condition states that a language is regular if some NFA recognizes it. Theorem 1 shows that any NFA can be converted into an equivalent DFA. Consequently, if an NFA recognizes some language, so does some DFA, and hence the language is regular. The other direction of the "if and only if" condition states that a language is regular only if some NFA recognizes it. That is, if a language is regular, some NFA must be recognizing it. Obviously, this condition is true because a regular language has a DFA recognizing it and any DFA is also an NFA.

PROOF – EXAMPLE - (1)

NFA N_4

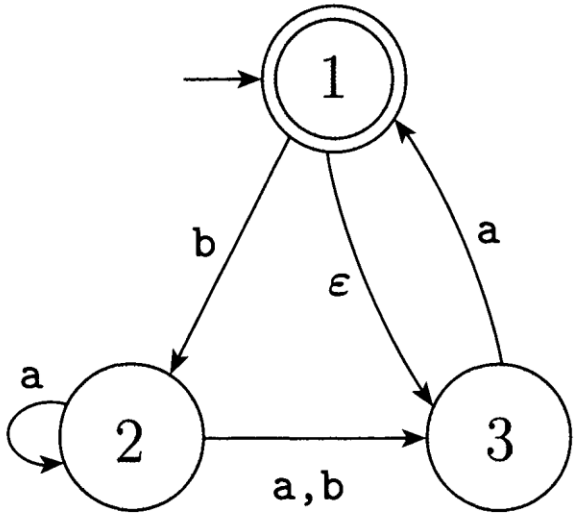
Thus in the formal description of $N_4 = (Q, \{a,b\}, \delta, 1, \{1\})$, the set of states Q is {I, 2, 3} as shown in the following figure.



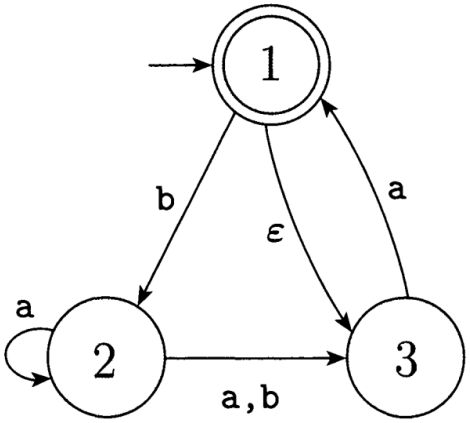
PROOF – EXAMPLE - (2)

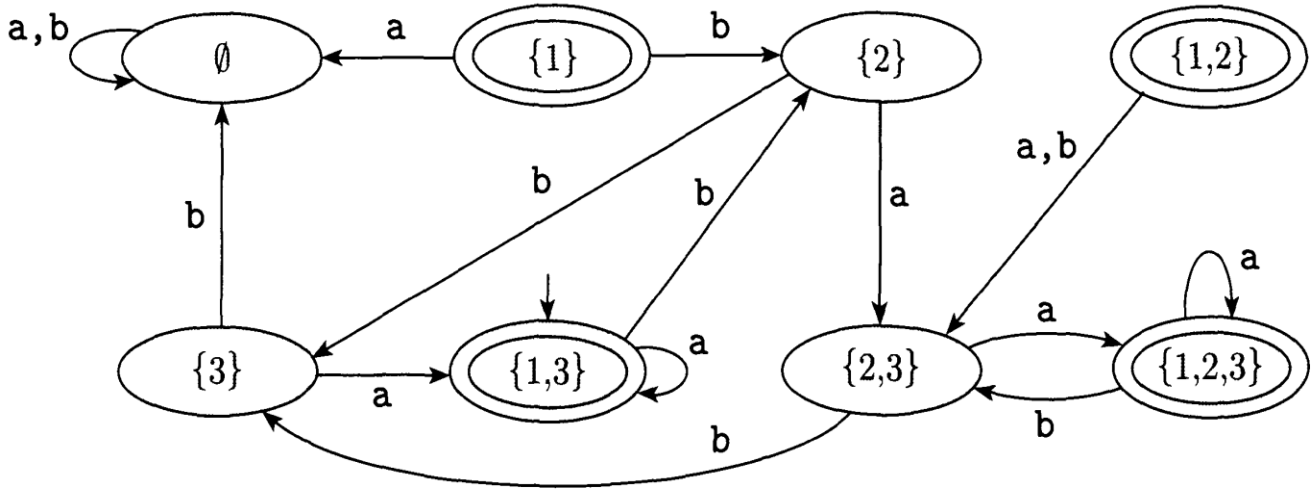
To construct a DFA D that is equivalent to N_4, we first determine D's states. N_4 has three states, $\{1, 2, 3\}$, so we construct D with eight states, one for each subset of N_4's states. We label each of D's states with the corresponding subset. Thus D's state set is

$$\{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$



PROOF – EXAMPLE - (3)





PROOF – EXAMPLE - (4)

Next, we determine the start and accept states of D. The start state is $E(\{1\})$, the set of states that are reachable from 1 by traveling along ε arrows, plus 1 itself. An ε arrow goes from 1 to 3, so $E(\{1\}) = \{1, 3\}$. The new accept states are those containing N_4's accept state; thus $\{\{1\}, \{1,2\}, \{1,3\}, \{1,2,3\}\}$.

Finally, we determine D's transition function. Each of D's states goes to one place on input `a` and one place on input `b`. We illustrate the process of determining the placement of D's transition arrows with a few examples.

PROOF – EXAMPLE - (5)

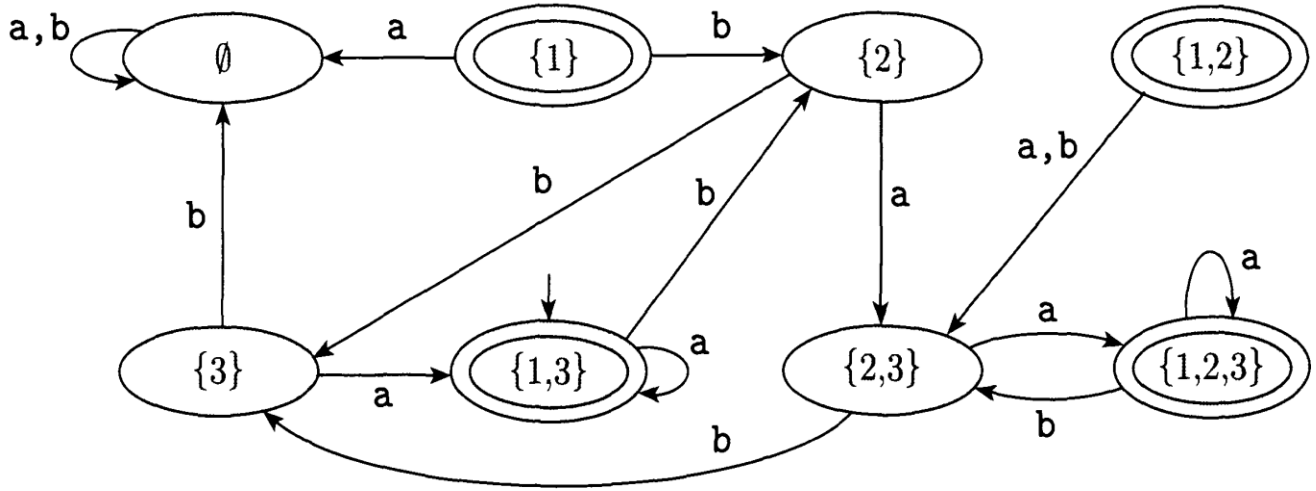
In D, state $\{2\}$ goes to $\{2,3\}$ on input a, because in N_4, state 2 goes to both 2 and 3 on input a and we can't go farther from 2 or 3 along ϵ arrows. State $\{2\}$ goes to state $\{3\}$ on input b, because in N_4, state 2 goes only to state 3 on input b and we can't go farther from 3 along ϵ arrows.

State $\{1\}$ goes to $\emptyset$ on a, because no a arrows exit it. It goes to $\{2\}$ on b. Note that the procedure in Theorem 1 specifies that we follow the ϵ arrows *after* each input symbol is read. An alternative procedure based on following the ϵ arrows before reading each input symbol works equally well, but that method is not illustrated in this example.

PROOF – EXAMPLE - (6)

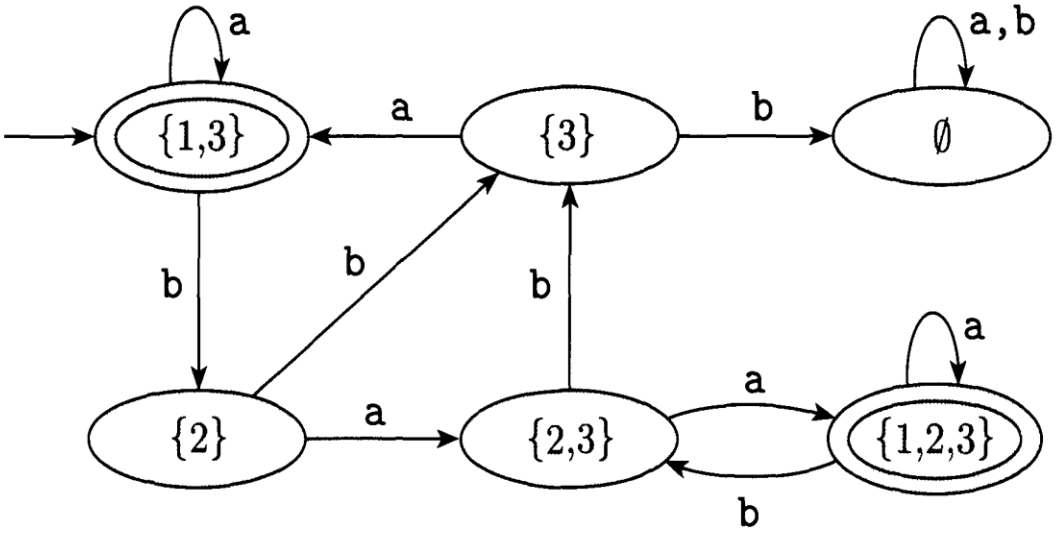
State $\{3\}$ goes to $\{1,3\}$ on a, because in N_4, state 3 goes to 1 on a and 1 in turn goes to 3 with an ε arrow. State $\{3\}$ on b goes to $\emptyset$.

State $\{1,2\}$ on a goes to $\{2,3\}$ because 1 points at no states with a arrows and 2 points at both 2 and 3 with a arrows and neither points anywhere with ε arrows. State $\{1,2\}$ on b goes to $\{2,3\}$. Continuing in this way we obtain the following diagram for D.



PROOF – EXAMPLE - (7)

We may simplify this machine by observing that no arrows point at states $\{1\}$ and $\{1, 2\}$, so they may be removed without affecting the performance of the machine. Doing so yields the following figure.



REGULAR LANGUAGE

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

REGULAR LANGUAGE – FORMAL DEFINITION

The collection of regular languages over an alphabet Σ is defined recursively as follows:

- The empty language Ø, and the empty string language {ε} are ***regular languages.***
- For each $a \in \Sigma$ (a belongs to Σ), the **singleton** language {a} is a ***regular language.***
- If A and B are regular languages, then $A \cup B$ (union), $A \bullet B$ (concatenation), and A^* (**Kleene star**) are regular languages.
- No other languages over Σ are regular.

REGULAR OPERATIONS (1)

In arithmetic, the basic objects are numbers and the tools are operations for manipulating them, such as + and x.

In the theory of computation the objects are languages and the tools include operations specifically designed for manipulating them.

REGULAR OPERATIONS (2)

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

- **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**: $A^* = \{x_1 x_2 \ldots x_k | \, k \geq 0 \text{ and each } x_i \in A\}$.

REGULAR OPERATIONS (3)

The star operation applies to a single language rather than to two different languages. That is, the star operation is a ***unary operation*** instead of a ***binary operation***.

It works by attaching any number of strings in A together to get a string in the new language. Because "any number" includes 0 as a possibility, the empty string ε is always a member of A*, no matter what A is.

REGULAR OPERATIONS (4)

Example:

Let the alphabet Σ be the standard 26 letters $\{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$. If $A = \{\mathtt{good}, \mathtt{bad}\}$ and $B = \{\mathtt{boy}, \mathtt{girl}\}$, then

$$A \cup B = \{\mathtt{good}, \mathtt{bad}, \mathtt{boy}, \mathtt{girl}\},$$

$$A \circ B = \{\mathtt{goodboy}, \mathtt{goodgirl}, \mathtt{badboy}, \mathtt{badgirl}\}, \text{ and}$$

$$A^* = \{\varepsilon, \mathtt{good}, \mathtt{bad}, \mathtt{goodgood}, \mathtt{goodbad}, \mathtt{badgood}, \mathtt{badbad}, \mathtt{goodgoodgood}, \mathtt{goodgoodbad}, \mathtt{goodbadgood}, \mathtt{goodbadbad}, \ldots\}.$$

REGULAR OPERATIONS (5)

Let **N** = {I, 2, 3, . .. } be the set of natural numbers.

When we say that ***N*** is closed under multiplication we mean that, for any x and y in N, the product *x* x *y* also is in ***N***.

In contrast N is not closed under division, as 1 and 2 are in **N** but 1/2 is not.

Generally speaking, a collection of objects is closed under some operation if applying that operation to members of the collection returns an object still in the collection.

The collection of regular languages is closed under all three of the regular operations.

CLOSURE UNDER THE REGULAR OPERATIONS

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

And

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

THEOREM-A (1)

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

PROOF IDEA:

We have regular languages A_1 and A_2 and want to prove that $A_1 \cup A_2$ is regular.

The idea is to take two NFAs, N_1 and N_2 for A_1 and A_2, and combine them into one new NFA, N.

THEOREM-A (2)

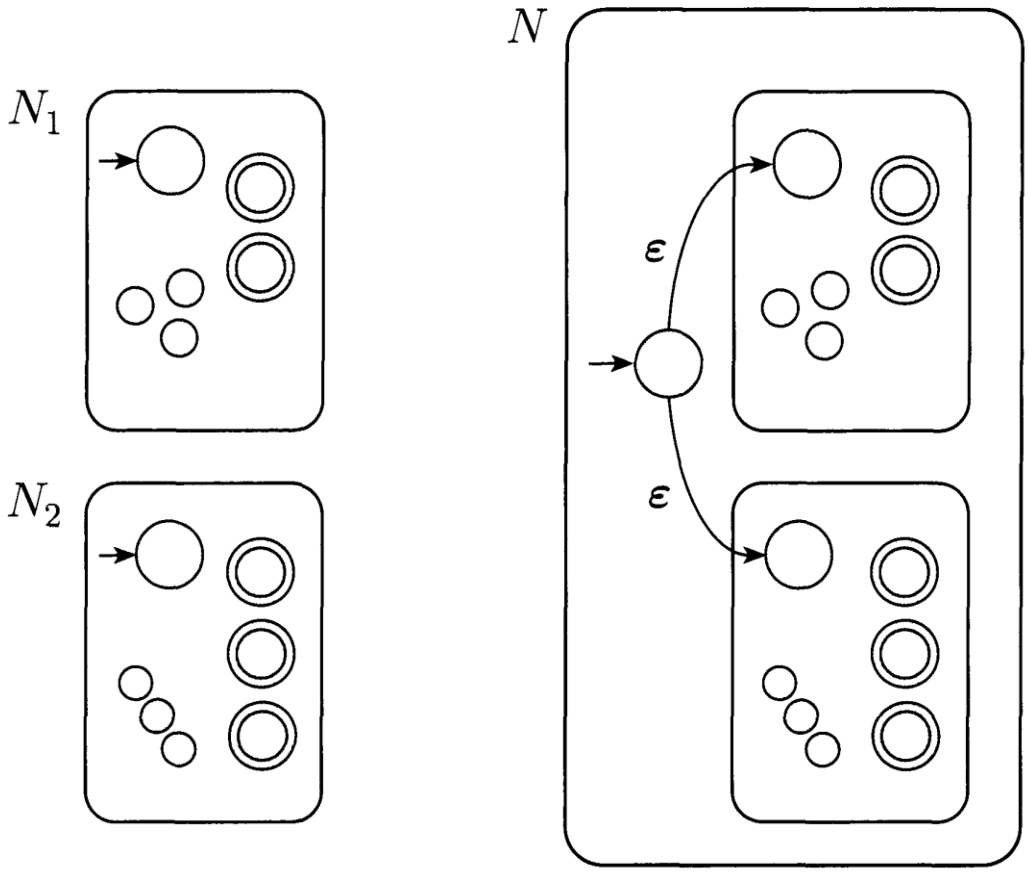
Machine N must accept its input if either N_1 and N_2 accepts this input.

The new machine has a new start state that branches to the start states of the old machines with ε arrows.

In this way the new machine non-deterministically guesses which of the two machines accepts the input. If one of them accepts the input, N will accept it, too.

THEOREM-A (3)

Construction of an NFA N to recognize A_1 U A_2



THEOREM-A (4)

PROOF

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1, and
$N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2.

Construct $N = (Q, \Sigma, \delta, q_0, F)$ to recognize $A_1 \cup A_2$.

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$.
   The states of N are all the states of N_1 and N_2, with the addition of a new start state q_0.
2. The state q_0 is the start state of N.

THEOREM-A (5)

3. The accept states $F = F_1 \cup F_2$.
The accept states of N are all the accept states of N_1 and N_2. That way N accepts if either N_1 accepts or N_2 accepts.

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-B (1)

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

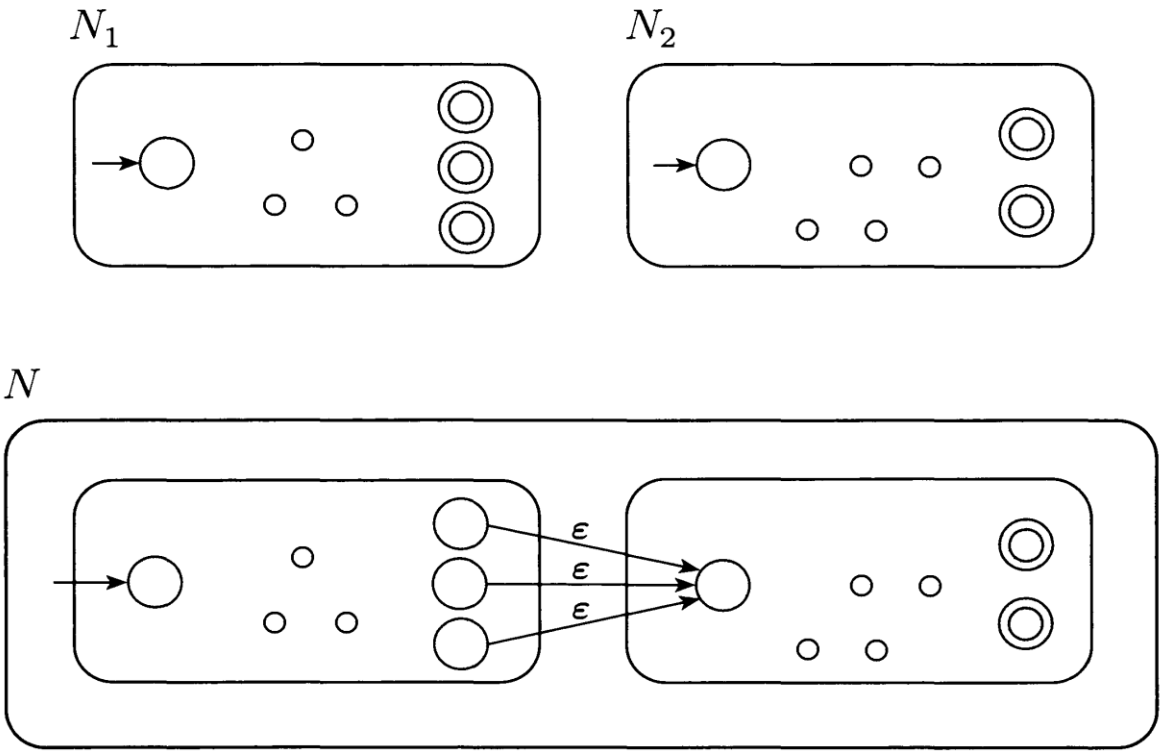
PROOF IDEA:

We have regular languages A_1 and A_2 and want to prove that $A_1 \circ A_2$ is regular.

The idea is to take two NFAs, N_1 and N_2 for A_1 and A_2, and combine them into a new NFA N as shown in Figure.

THEOREM-B (2)

Construction of *N* to recognize $A_1 \circ A_2$



THEOREM-B (3)

Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ 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. $Q = Q_1 \cup Q_2$.
   The states of N are all the states of N_1 and N_2.
2. The state q_1 is the same as the start state of N_1.

THEOREM-B (4)

3. The accept states F_2 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-C (1)

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

PROOF IDEA:

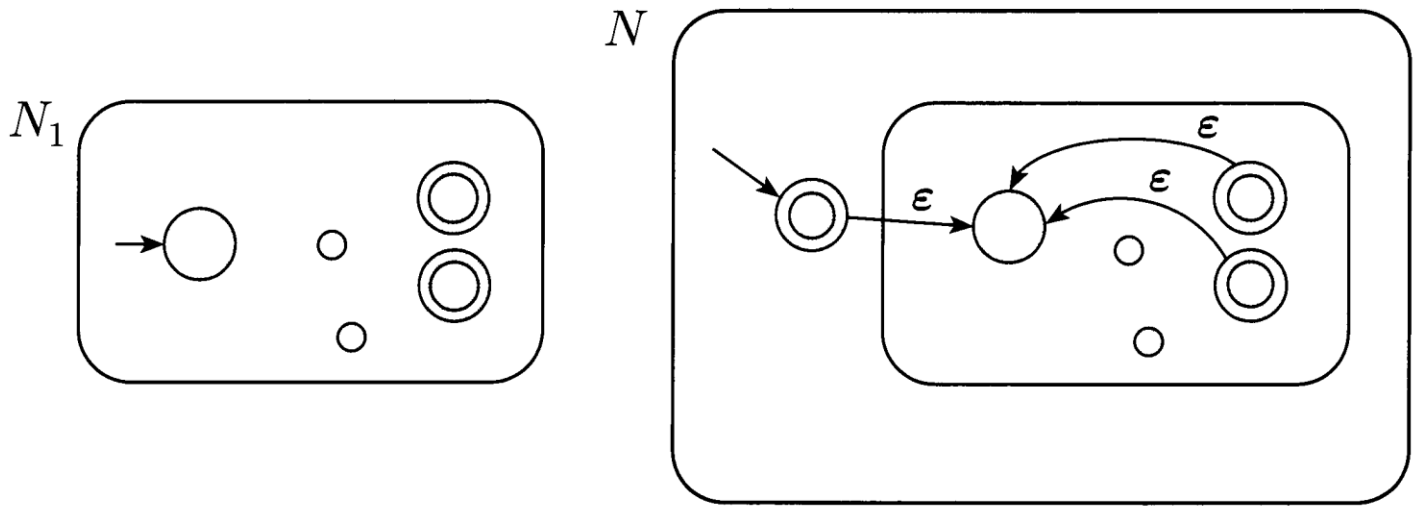
We have a regular language A_1 and want to prove that A_1^* also is regular.

We take an NFA N_1 for A_1 and modify it to recognize A_1^*, as shown in the following figure.

The resulting NFA N will accept its input whenever it can be broken into several pieces and N_1 accepts each piece.

THEOREM-C (2)

Construction of *N* to recognize A_1*



THEOREM-C (3)

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

1. $Q = \{q_0\} \cup Q_1$.
   The states of N are the states of N_1 plus a new start state.
2. The state q_0 is the new start state.
3. $F = \{q_0\} \cup F_1$.
   The accept states are the old accept states plus the new start state.

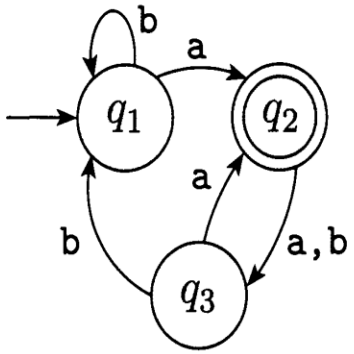
THEOREM-C (4)

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

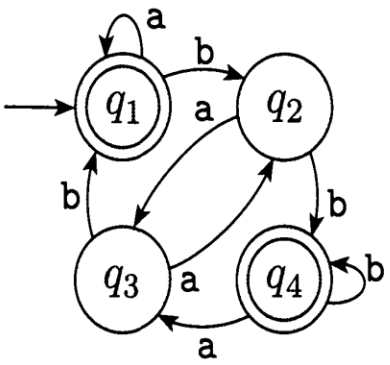
$$\delta(q,a) = \begin{cases} \delta_1(q,a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q,a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q,a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a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

EXERCISE

A**1.1** The following are the state diagrams of two DFAs, M_1 and M_2. Answer the following questions about each of these machines.

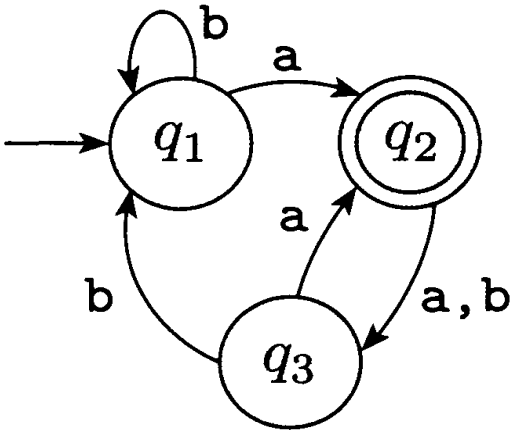


M_1



M_2

EXERCISE



M_1

a. What is the start state?

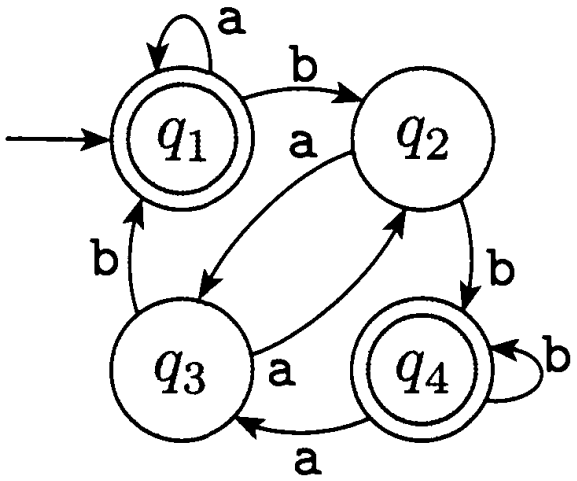
b. What is the set of accept states?

c. What sequence of states does the machine go through on input `aabb`?

d. Does the machine accept the string `aabb`?

e. Does the machine accept the string ε?

EXERCISE



M_2

a. What is the start state?

b. What is the set of accept sta...

c. What sequence of states does the machine go through on input `aabb`?

d. Does the machine accept the string `aabb`?

e. Does the machine accept the string ϵ?

EXERCISE

The formal description of a DFA M is $\big(\{q_1, q_2, q_3, q_4, q_5\}, \{\mathtt{u}, \mathtt{d}\}, \delta, q_3, \{q_3\}\big)$, where δ is given by the following table. Give the state diagram of this machine.

	u	d
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_2	q_4
q_4	q_3	q_5
q_5	q_4	q_5

EXERCISE

Q: Describe the Regular languages denoted by the following Regular expressions. Also construct the Finite Automata that recognize the regular languages generated by the following Regular Expressions.

(Note: The alphabet ∑ = {a, b})

1. **(b U ab*a)***
2. **(ab)* U $(aba)^+$**
3. **b ∑∑∑ a**
4. **a(b U ba)*b**
5. **ba* U ab***
6. **(ab)* U aba**

EXERCISE

7. $b\ (\Sigma\Sigma)^+\ a$
8. $(a\ U\ ba\ U\ bb)$
9. $(ab)^*\ U\ (aba)^+$
10. $(a\ U\ ba\ U\ bb)\ \Sigma^*$
11. $(a\ \Sigma^*)\ U\ (\Sigma^* b)$
12. $(a\ U\ \epsilon)(b\ U\ \epsilon)$
13. $b\ \Sigma\Sigma\Sigma\ a$
14. $(a\ \Sigma^*)\ U\ (\Sigma^* b)$
15. $aaa(\Sigma\Sigma)^*$
16. $b^*\ (ab^+)^*$
17. $b\ \Sigma\Sigma\Sigma\ a$

EXERCISE

1. **∑ = {a, b}. Construct FA for the following language**
   { *w* | *w* has even length and an odd number of a's }

2. **∑ = {0, 1}. Construct FA for the language $1^*(001^+)^*$**

3. **∑ = {a, b}. Construct FA for the following language**
   { *w* | *w* contains the substring abab, i.e., *w* = *x*abab*y* for some *x* and *y* }

4. **∑ = {0, 1}. Construct FA for the language 1^*01^+**

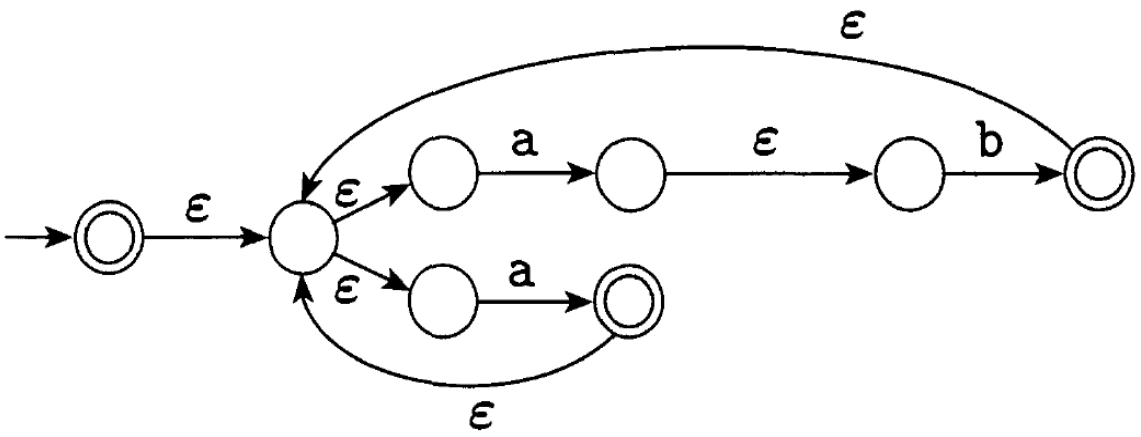
EXERCISE

1. **∑ = {a, b}. Construct FA for the following language**
   { *w* | *w* has an odd number of a's and ends with a b }

2. **∑ = {0, 1}. Construct FA for the following language**
   { *w* | *w* contains the substring 0101, i.e., *w* = *x*0101*y* for some *x* and *y* }

EXERCISE

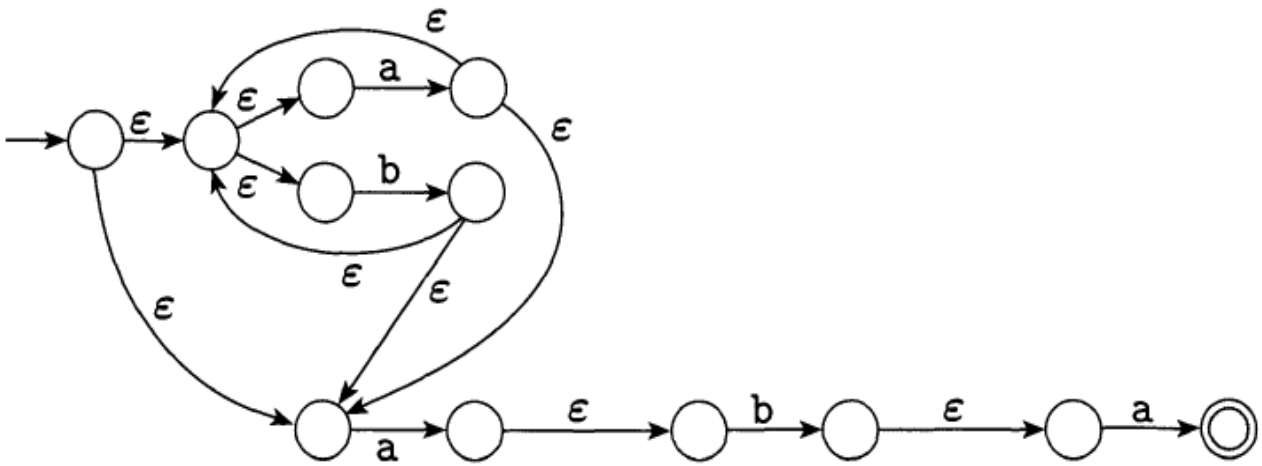
The following is the state diagram of a Finite Automaton M [where alphabet ∑ = {a, b}]



- **Give the complete formal description of machine M**
- **Does the machine M accept the string aaababa?**
- **Define the Regular language recognized by the above Finite Automaton M.**

EXERCISE

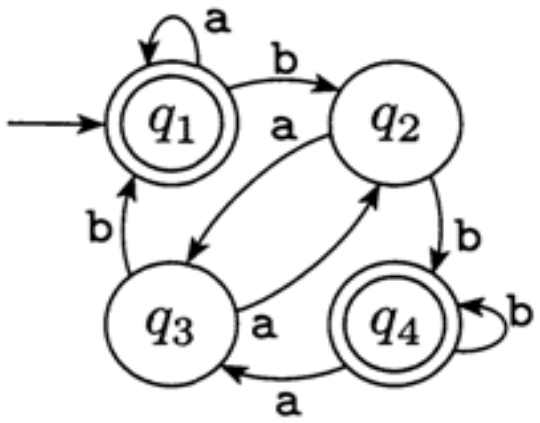
The following is the state diagram of a Finite Automaton M [where alphabet ∑ = {a, b}]



- **Give the complete formal description of machine M**
- **Does the machine M accept the string aabbbaba?**
- **Define the Regular language recognized by the above Finite Automaton M.**

EXERCISE

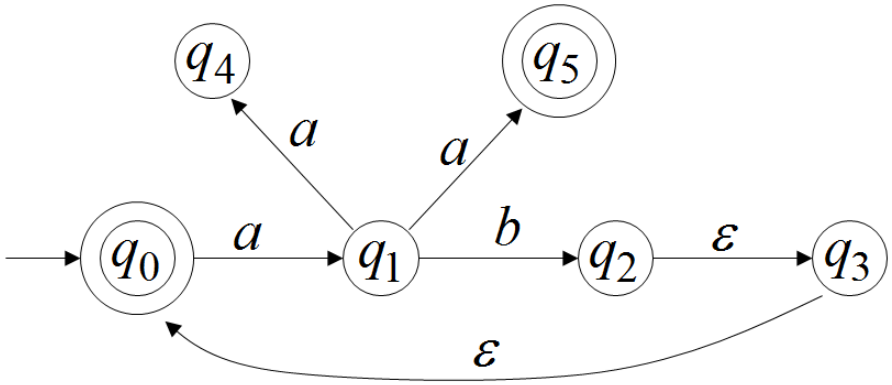
The following is the state diagram of a Finite Automaton M [where alphabet ∑ = {a, b}]



- **Give the complete formal description of machine M**
- **Does the machine M accept the string aabbaab? Write its sequence of states (i.e. trace)**

EXERCISE

The following is the state diagram of a Finite Automaton M [where alphabet ∑ = {a, b}]



- **Give the complete formal description of machine M**
- **Describe the language accepted by Finite Automaton M.**
- **What sequence of states does the machine go through on input abababaa? Does the machine accepts this input?**