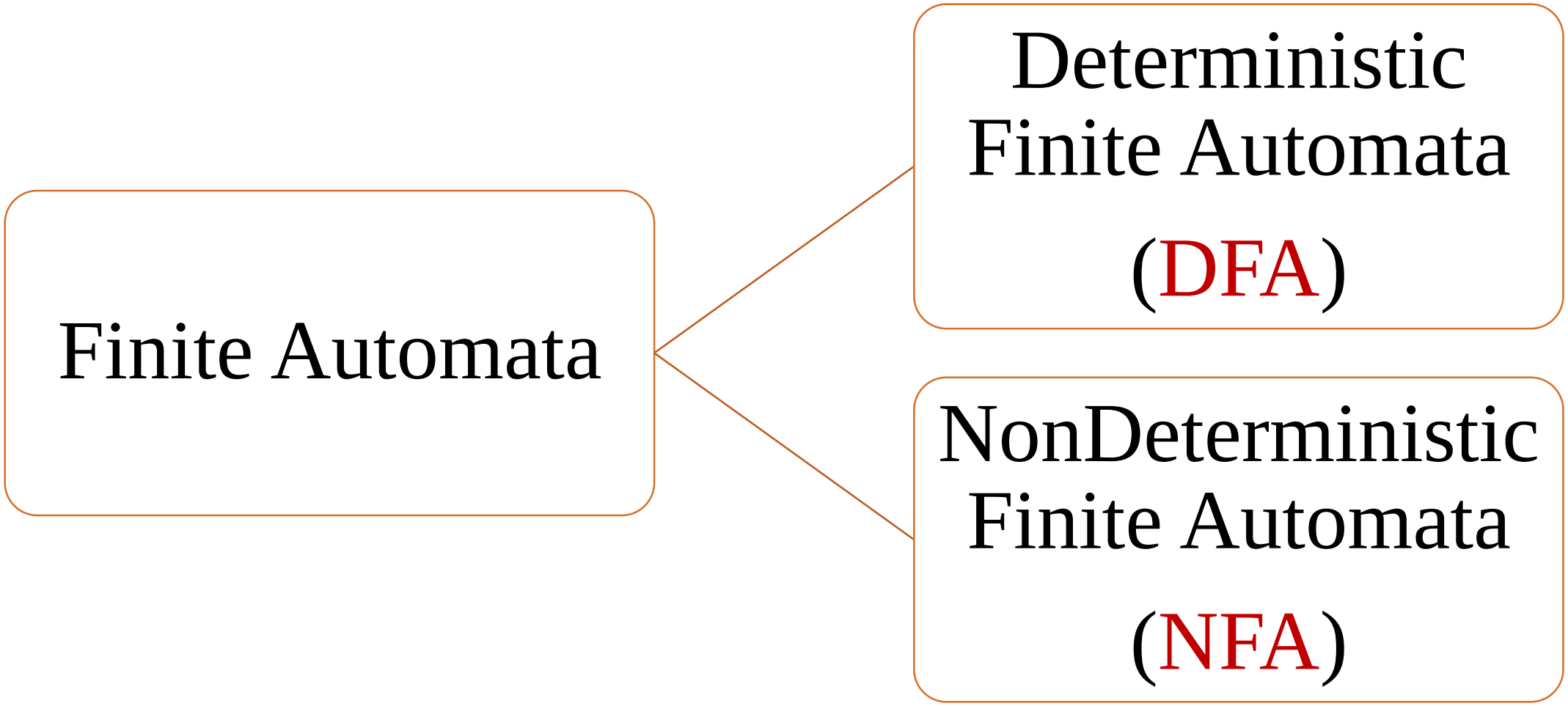
ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

Finite Automata

What is a Finite Automaton?

- Remembers only a finite amount of information.
- Information represented by its *state*.
- State changes in response to *inputs*.
- Rules that tell how the state changes in response to inputs are called *transitions*.

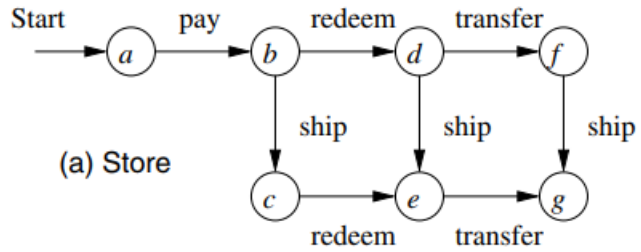
Finite Automata



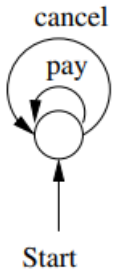
An Informal Picture of Finite Automata

- We investigate protocols that support “electronic money”
- The Ground Rules
  - There are three participants: the customer, the store and the bank
  - For simplicity, we assume that there is only one "money" file in existence.
  - Interaction among the three participants is thus limited to five events:
    1. The customer may decide to pay. That is, the customer sends the money to the store.
    2. The customer may decide to cancel. The money is sent to the bank with a message that the value of the money is to be added to the customer's bank account
    3. The store may ship goods to the customer
    4. The store may redeem the money. That is the money is sent to the bank with a request that its value be given to the store.
    5. The bank may transfer the money by creating a new, suitably encrypted money file and sending it to the store

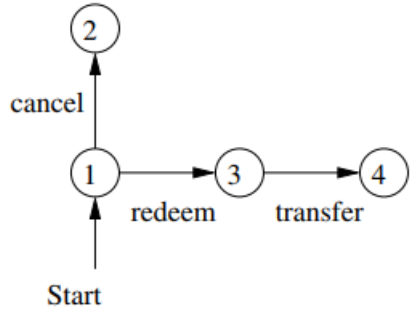
An Informal Picture of Finite Automata



(a) Store



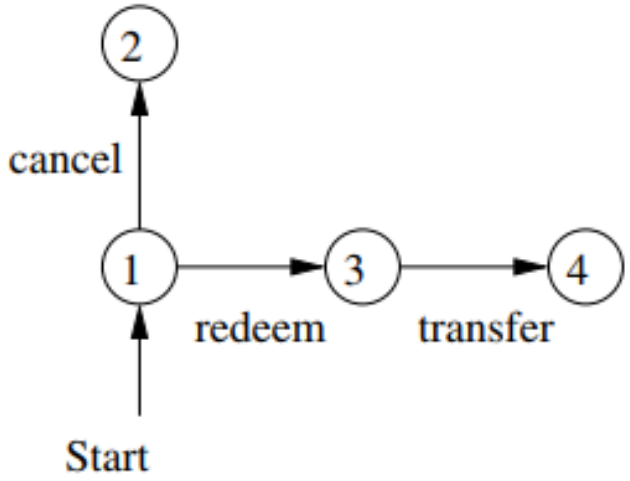
(b) Customer



(c) Bank

An Informal Picture of Finite Automata

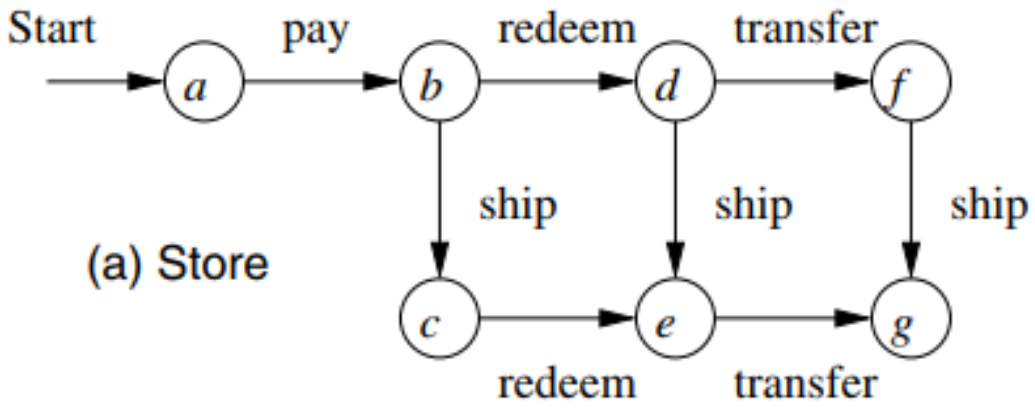
- The start state is state 1; it represents the situation where the bank has issued the money file in question but has not been requested either to redeem it or to cancel it.
- If a cancel request is sent to the bank by the customer, then the bank restores the money to the customer's account and enters state 2.
- When in state 1, the bank may receive a redeem request from the store -> it goes to state 3 and sends the store a transfer message, with a new money file that now belongs to the store. After sending the transfer message, the bank goes to state 4.
  - In that state, it will neither accept cancel or redeem requests nor will it perform any other actions regarding this particular money file



(c) Bank

An Informal Picture of Finite Automata

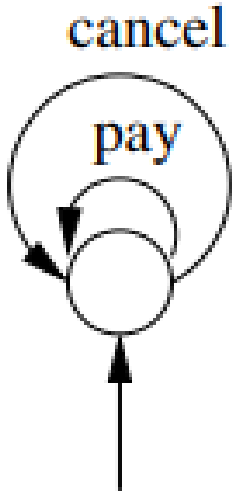
- The store starts out in *state a*.
- When the customer orders the goods by performing the *pay action*, the store enters *state b*.
  - In this state, the store begins both the shipping and redemption processes.
  - If the goods are shipped first, then the store enters *state c*, where it must still redeem the money from the bank and receive the transfer of an equivalent money file from the bank.
  - Alternatively, the store may send the redeem message first, entering *state d*.
    - From *state d*, the store might next ship, entering *state e*, or it might next receive the transfer of money from the bank, entering *state f*.
- From *state f* we expect that the store will eventually ship, putting the store in *state g*, where the transaction is complete and nothing more will happen.
- In *state e*, the store is waiting for the transfer from the bank. Unfortunately, the goods have already been shipped, and if the transfer never occurs the store is out of luck.



An Informal Picture of Finite Automata

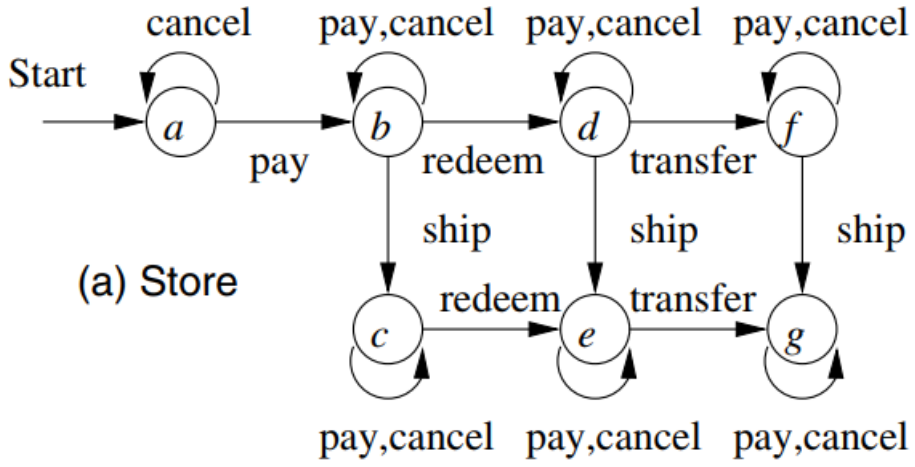
- This automaton has only one state, reffecting the fact that the customer “can do anything”.

- The customer can perform the *pay* and *cancel* actions any number of times, in any order, and stays in the lone state after each action.

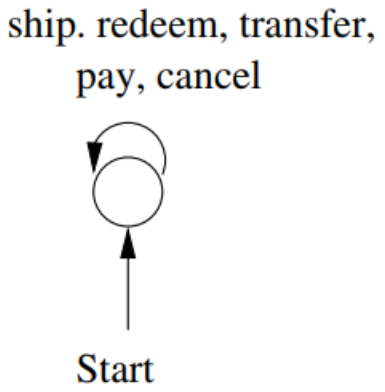


(b) Customer

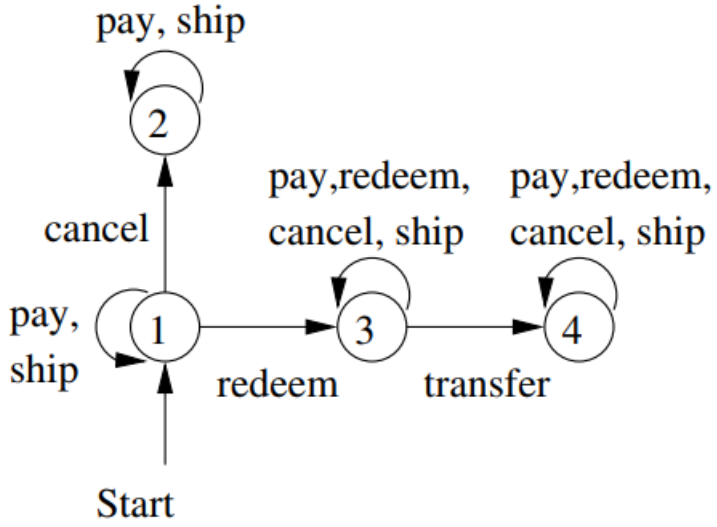
Enabling the Automata to Ignore Actions



(a) Store



(b) Customer



(c) Bank

ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

Deterministic Finite Automata

Deterministic Finite Automata

- A formalism for defining languages, consisting of:
  1. A finite set of *states* (Q, typically).
  2. An *input alphabet* (Σ, typically).
  3. A *transition function* (δ, typically).
  4. A *start state* (q_0, in Q, typically).
  5. A set of *final states* ($F \subseteq Q$, typically).
     - "Final" and "accepting" are synonyms.

- We often talk about a DFA in "five-tuple" notation: $A = (Q, \Sigma, \delta, q_0, F)$
  - A: name of the DFA

The Transition Function

- Takes two arguments: a state and an input symbol.
- δ(q, a) = the state that the DFA goes to when it is in state *q* and input *a* is received.
- Note: always a next state – add a dead state if no transition.

How a DFA Processes Strings

- The "language" of the DFA: the set of all strings that the DFA accepts.
- How the DFA decides whether or not to "accept" a sequence of input symbols?
  - Suppose $a_1a_2\ldots a_n$ is a sequence of input symbols.
  - We start out with the DFA in its start state, q_0
  - We consult the transition function δ, say $\delta(q_0, a_1) = q_1$ to find the state that the DFA A enters after processing the first input symbol a_1
  - We process the next input symbol a by evaluating $\delta(q_1, a_2)$; let us suppose this state is q_2
  - We continue in this manner, finding states $q_3, q_4, \ldots, q_n$ such that $\delta(q_{i-1}, a_i) = q_i$ for each i
  - If q_n is a member of F, then the input $a_1a_2\ldots a_n$ is accepted, and if not then it is "rejected".

Example 1

- Specify a DFA that accepts all and only the strings of 0's and 1's that have the sequence *01* somewhere in the string.
  - We can write this language L as:

    {w | w is of the form x01y for some strings x and y consisting of 0's and 1's only}

  - Another equivalent description using parameters x and y to the left of the vertical bar is:

    {x01y | x and y are any strings of 0's and 1's}

  - Strings in the language include 01, 11010, 100011, ….
  - Strings NOT in the language: ω, 0, 111000, ….

=> What do we know about an automaton that can accept this language L?

Example 1

- Specify a DFA that accepts all and only the strings of 0's and 1's that have the sequence *01* somewhere in the string.
  - First its input alphabet is $\sum = \{0, 1\}$.
  - It has some set of states, Q, of which one, say q_0 is the start state.
  - To decide whether *01* is a substring of the input, A needs to remember:
    1. Has it already seen *01*? If so, then it accepts every sequence of further inputs; i.e., it will only be in accepting states from now on.
    2. Has it never seen *01*, but its most recent input was 0, so if it now sees a 1, it will have seen *01* and can accept everything it sees from here on?
    3. Has it never seen *01*, but its last input was either nonexistent (it just started) or it last saw a *1*? In this case, A cannot accept until it first sees a *0* and then sees a *1* immediately after.

Example 1

- Specify a DFA that accepts all and only the strings of 0's and 1's that have the sequence *01* somewhere in the string.
  - Condition (3) represented by the start state, q_0.
  - When just starting, we need to see a *0* and then a *1*.

  - If we are in state q_0
    - If we next see a *1*, then we are no closer to seeing *01*, and so we must stay in state q_0. That is $\delta(q_0, 1) = q_0$.
    - If we next see a 0, we are in condition (2). That is, we have never seen *01*, but we have our 0. Thus, let us use q_2 to represent condition (2). Our transition from q_0 on input *0* is $\delta(q_0, 0) = q_2$.

  - Now let us consider the transitions from state q_2
    - If we see a *0*, we are no better of than we were, but no worse either. We have not seen *01*, but *0*was the last symbol, so we are still waiting for a *1*. State q_2 describes this situation perfectly, so we want $\delta(q_2, 0) = q_2$.
    - If we see a *1*, we now know there is a *0* followed by a *1*. We can go to an accepting state, which we shall call q_1, and which corresponds to condition (1) above. That is, $\delta(q_2, 1) = q_1$

  - Finally, we must design the transitions for state q_1. In this state, we have already seen a *01* sequence, so regardless of what happens, we shall still be in a situation where we've seen 01. That is, $\delta(q_1, 0) = \delta(q_1, 1) = q_1$.

Example 1

- Specify a DFA that accepts all and only the strings of 0's and 1's that have the sequence *01* somewhere in the string.
  - Thus $Q = \{q_0, q_1, q_2\}$.
  - Start state: q_0
  - Accepting state: q_1; that is, $F = \{q_1\}$.

  $\Rightarrow$The complete specification of the automaton A that accepts the language L of strings that have a *01* substring, is

$$A = (Q, A, \delta, q0, F),$$
$$Q = \{q_0, q_1, q_2\}$$
$$A = \{0, 1\}$$
$$\delta: \delta(q_0, 1) = q_0, \delta(q_0, 0) = q_2, \delta(q_2, 0) = q_2, \delta(q_2, 1) = q_1, \delta(q_1, 0) = q_1, \delta(q_1, 1) = q_1$$
$$q_0: q_0$$
$$F = \{q_1\}$$

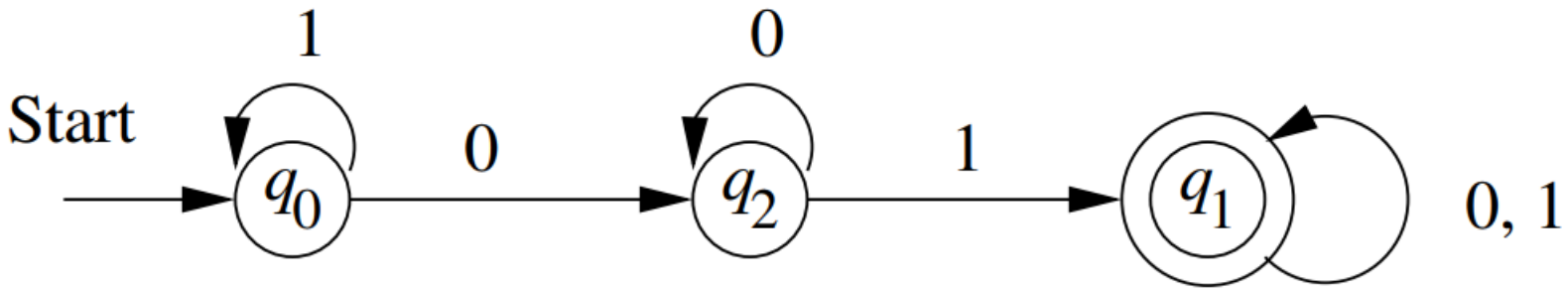
Simpler Notations for DFA's

- Specifying a DFA as a *five-tuple* with a detailed description of the δ transition function is both tedious and hard to read.

- There are two preferred notations for describing automata:
  - A *transition diagram*
  - A *transition table*

Transition Diagrams

- A transition diagram for a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is a graph defined as follows:
  - For each state in Q there is a node
  - For each state q in Q and each input symbol a in Σ, let $\delta(q, a) = p$.
    Then the transition diagram has an arc from node q to node p, labeled a.
    - If there are several input symbols that cause transitions from q to p, then the transition diagram can have one arc labeled by the list of these symbols.
  - There is an arrow into the start state q_0, labeled *Start*. This arrow does not originate at any node.
  - Nodes corresponding to accepting states (those in F) are marked by a *double circle*. States not in F have a *single circle.*



The transition diagram for the DFA accepting all strings with a substring 01

Transition Tables

- A transition table is a conventional tabular representation of a function like δ that takes two arguments and returns a value.
  - The rows of the table correspond to the states
  - The columns correspond to the inputs
  - The entry for the row corresponding to state *q* and the column corresponding to input *a* is the state *δ(q, a)*.

- *The start state is marked with an arrow*
- *The accepting states are marked with a star*

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Transition table for the DFA of Example 1

Extending the Transition Function to Strings

- The DFA defines a language (informally): the set of all strings that result in a sequence of state transitions from the start state to an accepting state.
- In terms of the transition diagram: the language of a DFA is the set of labels along all the paths that lead from the start state to any accepting state.
- Now we need to make the notion of the language of a DFA precise

  => we define an *extended transition function* ($\hat{\delta}$) that describes what happens when we start in any state and follow any sequence of inputs.

- The *extended transition function* $\hat{\delta}$ is a function that takes a state q and a string w and returns a state p - the state that the automaton reaches when starting in state q and processing the sequence of inputs w.
  - $\hat{\delta}(q, \varepsilon) = q$
    - That is, if we are in state q and read no inputs then we are still in state q
  - Suppose w is a string of the form xa; that is, a is the last symbol of w, and x is the string consisting of all but the last symbol.
    - For example: $w = 1101$ is broken into $x = 110$ and a = 1.
    - Then $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$

Example 2

Design a DFA to accept the language L = {*w*|*w* has both an even number of *0's* and an even number of *1's*}

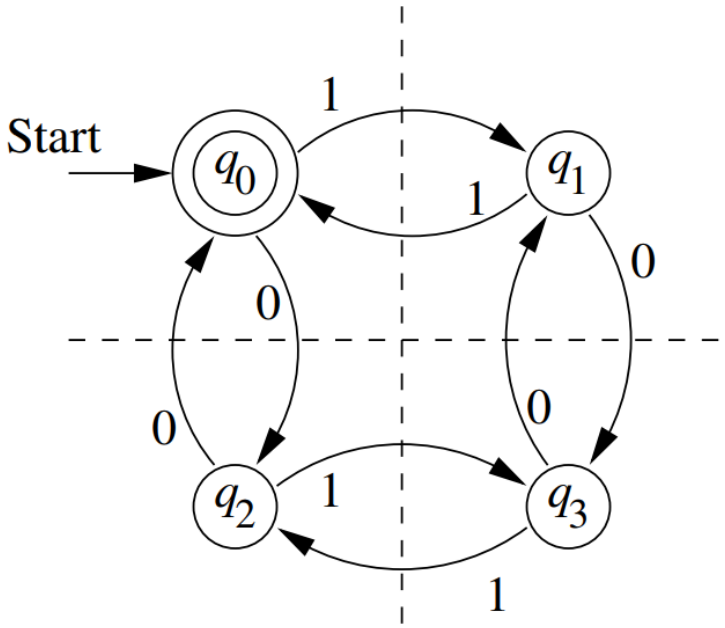
⇒The job of the states of this DFA is to count both the number of 0's and the number of 1's but count them modulo 2.

⇒That is, the state is used to remember whether the number of 0's seen so far is even or odd

⇒and also to remember whether the number of 1's seen so far is even or odd.

⇒There are thus four states, which can be given the following interpretations:

- q_0: Both the number of 0's seen so far and the number of 1's seen so far are even
- q_1: The number of 0's seen so far is even but the number of 1'sseen so far is odd
- q_2: The number of 1's seen so far is even but the number of 0's seen so far is odd
- q_3: Both the number of 0's seen so far and the number of 1's seen so far are odd
- State $q_0$is both the start state and the lone accepting state.
  - It is the start state because before reading any inputs the numbers of 0's and 1's seen so far are both zero and zero is even.



Example 2

- Represent the DFA of Example 2 by
  - a five-tuple
  - a transition table

M = (Q, Σ, δ, q0, F),

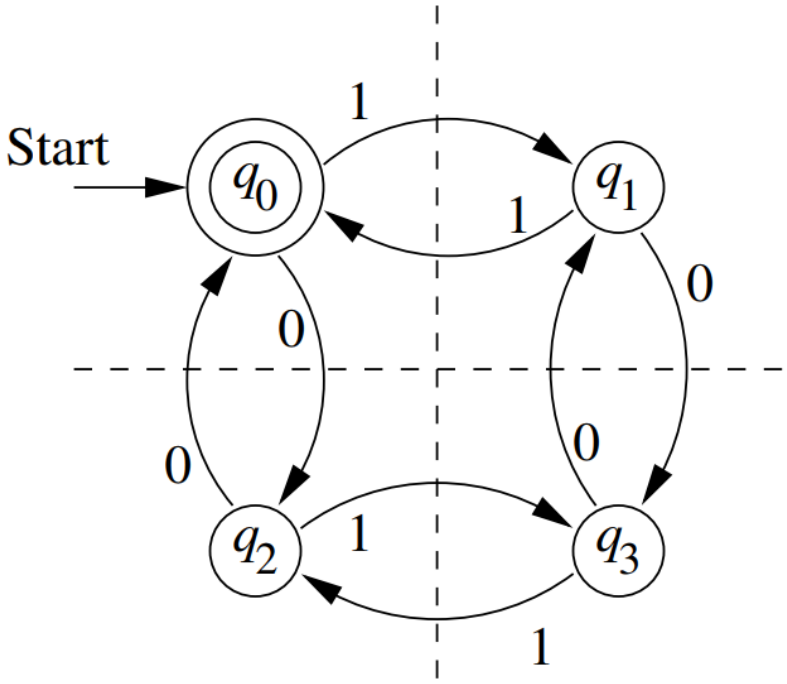
Q = {q_0, q_1, q_2, q_3}

Σ = {0, 1}

δ: $\delta(q_0, 1) = q_1$, $\delta(q_0, 0) = q_2$,
$\delta(q_1, 0) = q_3$, $\delta(q_1, 1) = q_0$,
$\delta(q_2, 0) = q_0$, $\delta(q_2, 1) = q_3$,
$\delta(q_3, 0) = q_1$, $\delta(q_3, 1) = q_2$

q_0: q_0

F = {q_0}



	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Extended Transition Function

- The check involves computing $\hat{\delta}(q, w)$ for each prefix w of *110101* starting at ε and going in increasing size.
- The summary of this calculation is:
  - $\hat{\delta}(q_0, \varepsilon) = q_0$
  - $\hat{\delta}(q_0, 1) = \delta(\hat{\delta}(q_0, \varepsilon), 1) = \delta(q_0, 1) = q_1$
  - $\hat{\delta}(q_0, 11) = \delta(\hat{\delta}(q_0, 1), 1) = \delta(q_1, 1) = q_0$
  - $\hat{\delta}(q_0, 110) = \delta(\hat{\delta}(q_0, 11), 0) = \delta(q_0, 0) = q_2$
  - $\hat{\delta}(q_0, 1101) = \delta(\hat{\delta}(q_0, 110), 1) = \delta(q_2, 1) = q_3$
  - $\hat{\delta}(q_0, 11010) = \delta(\hat{\delta}(q_0, 1101), 0) = \delta(q_3, 0) = q_1$
  - $\hat{\delta}(q_0, 110101) = \delta(\hat{\delta}(q_0, 11010), 1) = \delta(q_1, 1) = q_0$

	0	1
$* \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

q_0 is a member of F, then *110101* is accepted.

The Language of a DFA

- The language of a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is denoted *L(A)* and is defined by

  $L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ is in } F\}$

- That is, the language of *A* is the set of strings *w* that take the start state *q* to one of the accepting states.

- If *L* is *L(A)* for some DFA *A*, then we say *L* is a *regular language*.
  - If *A* is the DFA of *Example 1* then *L(A)* is the set of all strings of *0's* and *1's* that contain a substring *01*.
  - If *A* is the DFA of *Example 2* then *L(A)* is the set of all strings of *0's* and *1's* whose numbers of *0's* and *1's* are both even.

Exercises

Exercise 1:

Give DFA's accepting the following languages over the alphabet {0, 1}:

1. The set of all strings ending in 00
2. The set of all strings with three consecutive 0's (not necessarily at the end)
3. The set of strings with 001 as a substring.

Exercises

Exercise 2:

Given the alphabet {0,1}, construct a DFA which recognizes those elements of the universal language with an odd number of zeros.

Exercises

Exercise 3:

Given the alphabet {a,b}, construct a DFA which recognizes string with length “3” of the universal language.

Exercises

Exercise 4:

We want to design a device that, given a string which consists of binary numbers, will be able to find if the keyword “1011” is included in the input string and it also would be used as a basis to count the number of times this keyword is included.

For instance, for the input string 0101011011011, the device would detect two occurrences of the keyword (the “1” in the seventh position is not considered as the beginning of a new apparition).

It is required to design the corresponding DFA.

Exercises

Exercise 5:

In several programming languages, comments are included between the marks “/*”and “*/”.

Let L be the language of every string of comments limited by these marks.

Then, every element in L begins /* and ends with */, but it does not include any intermediate */.

To simplify the problem, consider that the input alphabet is {a, b, /,*}.

Indicate the DFA which recognizes L.

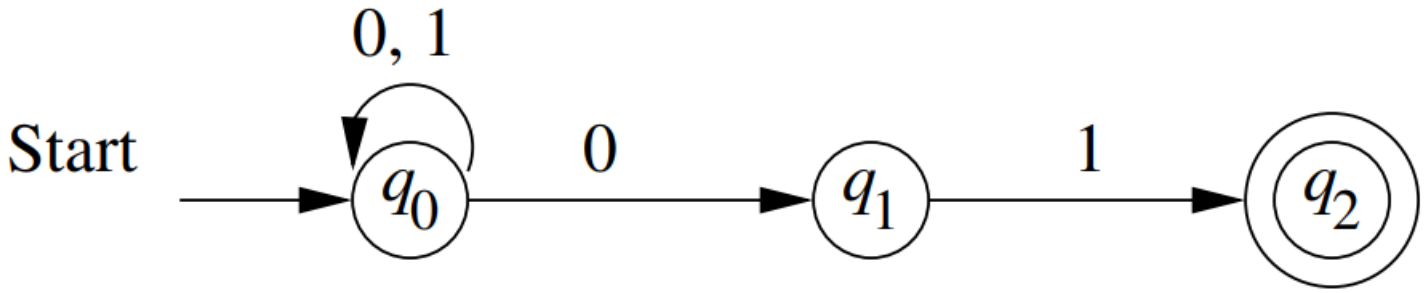
ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG VIỆT - HÀN
Vietnam - Korea University of Information and Communication Technology

Nondeterministic Finite Automata (NFA)

An Informal View of NFA

- Like the DFA, an NFA has:
  - a finite set of states (Q, typically).
  - a finite set of input symbols (Σ, typically),
  - one start state (q_0, in Q, typically),
  - a set of accepting states ($F \subseteq Q$, typically),
  - a transition function (δ, typically).
- The difference between the DFA and the NFA is in the type of transition function
  - For the NFA, δ is a function that takes *a state* and *input symbol* as arguments (like the DFA's transition function), but returns a set of zero, one or more states (rather than returning exactly one state, as the DFA must)

Example 3



An NFA accepting all strings that end in 01

Definition of Nondeterministic Finite Automata

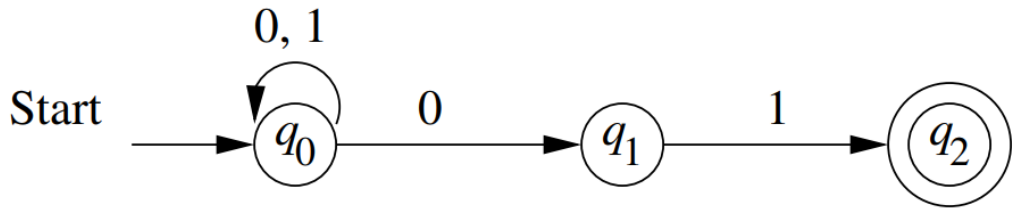
- An NFA is represented essentially like a DFA

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

where:

- Q is a finite *set of states*
- Σ is a finite *set of input symbols*
- q_0, a member of Q, is the *start state*
- F, a subset of Q, is the set of *final* (or *accepting*) *states*
- δ, the *transition function*
  - is a function that takes a state in Q and an input symbol in Σ as arguments and returns a subset of Q
  - Notice that the only *difference between an NFA and a DFA* is in the type of value that δ returns:
    - a set of states in the case of an NFA
    - a single state in the case of a DFA

Example 3:
NFA accepting all strings that end in 01



- By five-tuple: $A = (Q, \Sigma, \delta, q_0, F)$

  $Q = \{q_0, q_1, q_2\}$
  $\Sigma = \{0, 1\}$
  $q_0: q_0$
  $F = \{q_2\}$
  δ: $\delta(q_0, 0) = \{q_0, q_1\}$
  $\delta(q_0, 1) = \{q_0\}$
  $\delta(q_1, 1) = \{q_2\}$
  $\delta(q_1, 0) = \phi$
  $\delta(q_2, 0) = \phi$

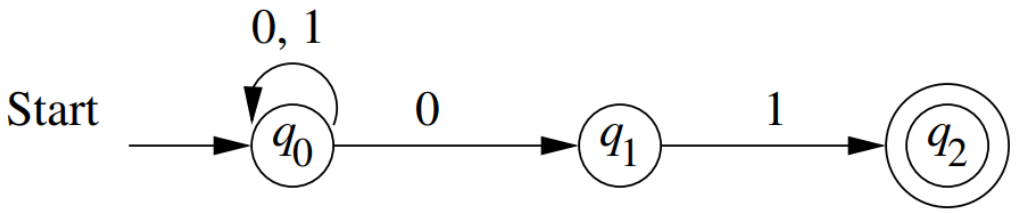
- By transition table

	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0\}$
q_1	$\emptyset$	$\{q_2\}$
$*q_2$	$\emptyset$	$\emptyset$

The Extended Transition Function

- As for DFA's, we need to extend the transition function δ of an NFA to a function $\hat{\delta}$ that
  - takes a state q and a string of input symbols *w*
  - and returns the set of states that the NFA is in
    - if it starts in state q and processes the string *w*
  - $\hat{\delta}(q, \varepsilon) = q$
    - That is, without reading any input symbols, we are only in the state we began in.
  - $\hat{\delta}(q, w) = ???$
    - Suppose *w* is of the form *w* = *xa*, where
      - *a* is the final symbol of *w* and
      - *x* is the rest of *w*
    - Also suppose that $\hat{\delta}(q, x) = \{p_1, p_2, ..., p_k\}$
    - Let $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2, \ldots, rm\}$
    - Then $\hat{\delta}(q, w) = \{r_1, r_2, \ldots, rm\}$

Example 4



- Let us use $\hat{\delta}$ to describe the processing of input 00101 by the NFA of Example 3

$\hat{\delta}(q_0, 00101)$ = ????

$\hat{\delta}(q_0, \varepsilon) = \{q_0\}$

$\hat{\delta}(q_0, 0) = \delta(\hat{\delta}(q_0, \varepsilon), 0) = \delta(q_0, 0) = \{q_0, q_1\}$

$\hat{\delta}(q_0, 00) = \delta(\hat{\delta}(q_0, 0), 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \varnothing = \{q_0, q_1\}$

$\hat{\delta}(q_0, 001) = \delta(\hat{\delta}(q_0, 00), 1) = \delta(q_0, 1)) \cup \delta(q_1, 1) = \{q_0, q_2\}$

$\hat{\delta}(q_0, 0010) = \delta(\hat{\delta}(q_0, 001), 0) = \delta(q_0, 0)) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \varnothing = \{q_0, q_1\}$

$\hat{\delta}(q_0, 00101) = \delta(\hat{\delta}(q_0, 0010), 1) = \delta(q_0, 1)) \cup \delta(q_1, 1) = \{q_0, q_2\}$

The Language of an NFA

- An NFA accepts a string w if
  - it is possible to make any sequence of choices of next state, while reading the characters of w and
  - go from the start state to any accepting state

- If $A = (Q, \Sigma, \delta, q_0, F)$ is an NFA, then $L(A) = \{w \mid \hat{\delta}(q_0, w) \cap F \neq \phi\}$
  - $L(A)$ is the set of strings w in Σ^* such that $\hat{\delta}(q_0, w)$ contains at least one accepting state.

Exercises

- Exercise 1:

  Design a NFA to recognize set of all strings over {0, 1} that end with 0

Exercises

- Exercise 2:

  Design a NFA to recognize set of all strings over {0, 1} that contain 0

Exercises

- Exercise 3:

Design a NFA to recognize set of all strings over {0, 1} that start with 000 or end with 111

Equivalence of DFA and NFA

- Every language that can be described by some NFA can also be described by some DFA.
- The DFA in practice has about as many states as the NFA, although it often has more transitions.
- In the worst case, however, the smallest DFA can have 2^n states while the smallest NFA for the same language has only n states.

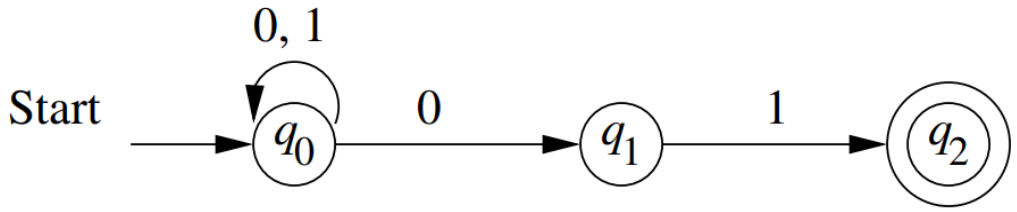
Convert a NFA to a DFA

- Convert NFA $N = (Q_N, \Sigma, \delta_N, q_0, F_N)$ to DFA $D = (Q_D, \Sigma, \delta_D, q_0, F_D)$ such that L(D) = L(N)
  - Notice that,
    - *the input alphabets* of the two automata are the same
    - *the start state of* D is the set containing only the start state of N.
    - the other components of D are constructed as follows
      - Q_D is the set of subsets of Q_N
        - if Q_N has *n* states then Q_D will have 2^n states
      - F_D is the set of subsets S of Q_N such that $S \cap F_N \neq \varnothing$
        - That is, F_D is all sets of N's states that include at least one accepting state of N
      - For each set $S \subseteq Q_N$ and for each input symbol *a* in Σ,

        $\delta_D(S, a) = \bigcup_{p \text{ in } S} \delta_N(p, a)$

        - That is, to compute $\delta_D(S, a)$ we look at all the states *p* in *S*, see what states N goes to from *p* on input *a*, and take the union of all those states

Example 5: Convert to a DFA the following NFA



- The complete subset construction

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

- Renaming the states

	0	1
A	A	A
$\rightarrow B$	E	B
C	A	D
$*D$	A	A
E	E	F
$*F$	E	B
$*G$	A	D
$*H$	E	F

- Starting in the start state B we can only reach states B, E and F.
- The other five states are inaccessible from the start state and may as well not be there.

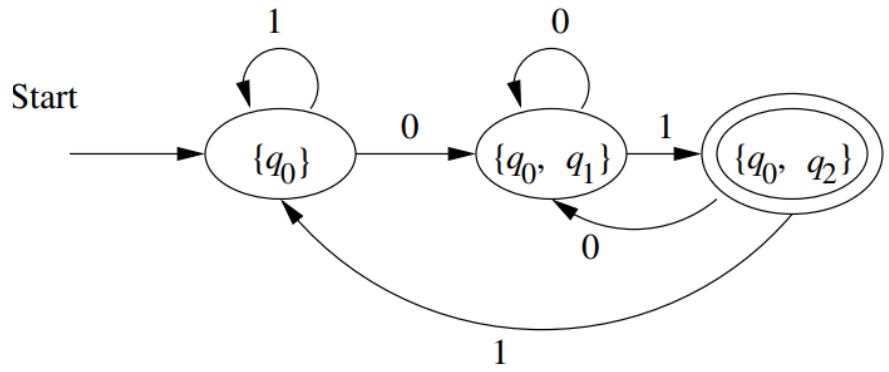
Example 5: Convert to a DFA the following NFA

- The complete subset construction

	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_1\}$	$\emptyset$	$\{q_2\}$
$*\{q_2\}$	$\emptyset$	$\emptyset$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$*\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$
$*\{q_1, q_2\}$	$\emptyset$	$\{q_2\}$
$*\{q_0, q_1, q_2\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$

- Renaming the states

	0	1
A	A	A
$\rightarrow B$	E	B
C	A	D
$*D$	A	A
E	E	F
$*F$	E	B
$*G$	A	D
$*H$	E	F



BÀI 1. Xây dựng DFA từ các NFA sau

2. NFA có bảng dịch chuyển

δ_N	a	b	c	d
→0	{1,2}	∅	{1}	{2}
1	∅	{0}	{2}	{0,1}
*2	∅	∅	∅	∅

Exercises

- Exercise 1: Convert to a DFA the following NFA

	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	$\emptyset$
$*s$	$\{s\}$	$\{s\}$

Exercises

- Exercise 2: Convert to a DFA the following NFA

	0	1
$\rightarrow p$	$\{q, s\}$	$\{q\}$
$*q$	$\{r\}$	$\{q, r\}$
r	$\{s\}$	$\{p\}$
$*s$	$\emptyset$	$\{p\}$

Exercises

- Exercise 3: Convert the following NFA to a DFA and informally describe the language it accepts.

	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r, s\}$	$\{t\}$
r	$\{p, r\}$	$\{t\}$
$*s$	$\emptyset$	$\emptyset$
$*t$	$\emptyset$	$\emptyset$

Exercises

- Exercise 4: Design NFA's to recognize the following sets of strings then convert each of your NFA's to DFA's.

a) *abc*, *abd*, and *aacd*. Assume the alphabet is {a, b, c, d}

b) *0101*, *101*, and *011*

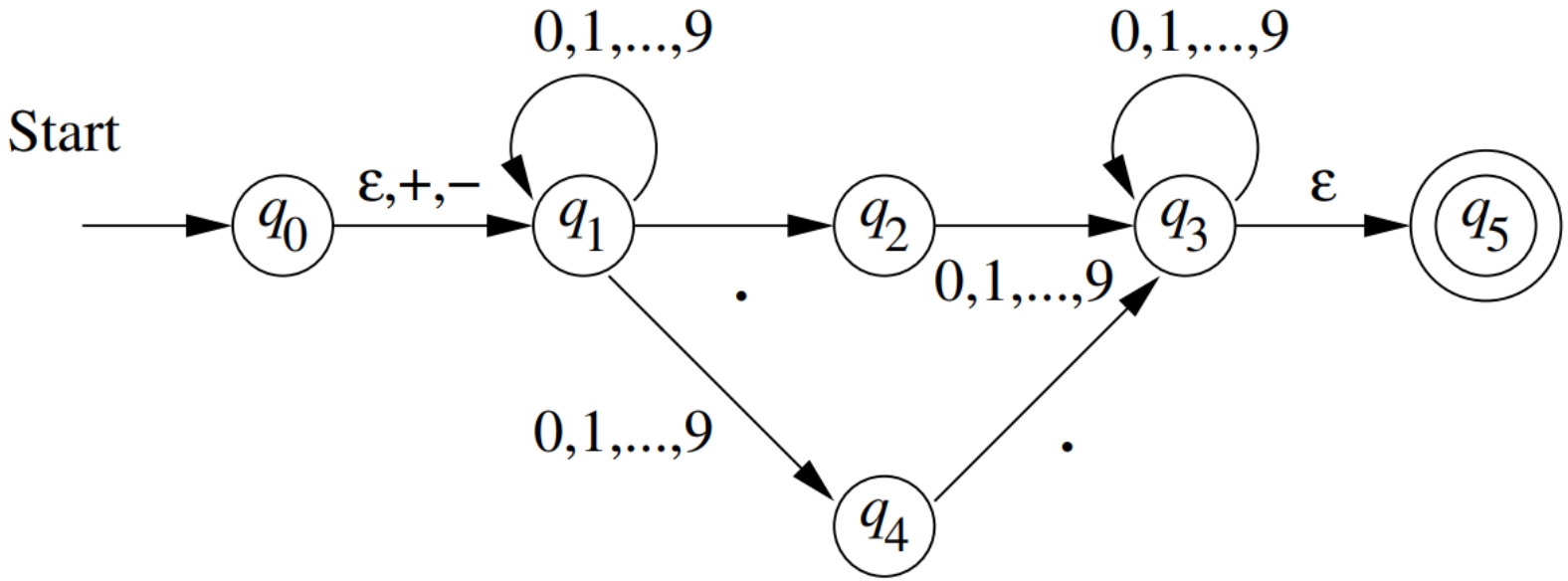
c) *ab*, *bc*, and *ca*. Assume the alphabet is {a, b, c}

Finite Automata With Epsilon Transitions

- ε–NFA: using transition diagrams with ε allowed as a label

Example 6

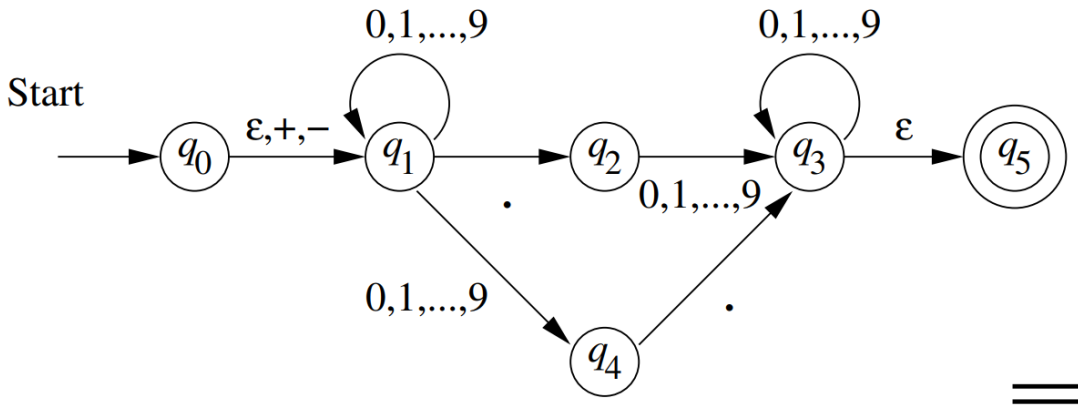
- An ε–NFA that accepts decimal numbers consisting of:
  1. An optional + or – sign,
  2. A string of digits,
  3. A decimal point, and
  4. Another string of digits. Either this string of digits, or the string (2) can be empty, but at least one of the two strings of digits must be nonempty.



The Formal Notation for an ε–NFA

- We may represent an ε–NFA exactly as we do an NFA with one exception:
  - the transition function must include information about transitions on ε.
- Formally, we represent an ε–NFA A by $A = (Q, \Sigma, \delta, q_0, F)$, where:
  - all components have their same interpretation as for an NFA
  - except that δ is now a function that takes as arguments:
    - a state in Q, and
    - a member of $\Sigma \cup \{\varepsilon\}$
      - that is, either an input symbol, or the symbol ε
      - We require that ε, the symbol for the empty string, cannot be a member
      - of the alphabet Σ, so no confusion results

Example 6: Transition table

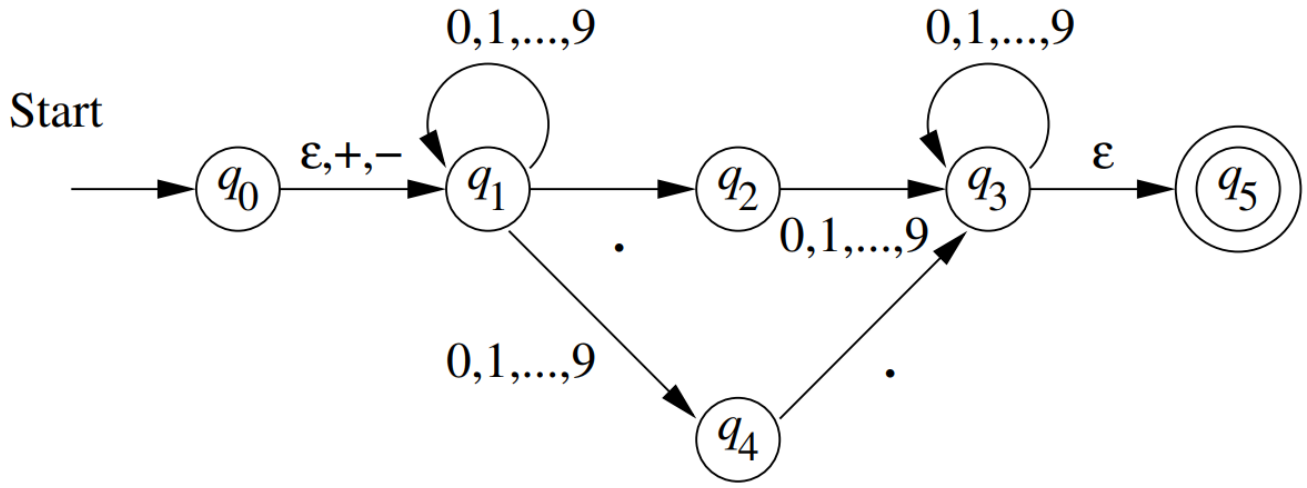


	ϵ	$+, -$	.	$0, 1, \ldots, 9$
$\rightarrow q_0$	$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\emptyset$
q_1	$\emptyset$	$\emptyset$	$\{q_2\}$	$\{q_1, q_4\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_3	$\{q_5\}$	$\emptyset$	$\emptyset$	$\{q_3\}$
q_4	$\emptyset$	$\emptyset$	$\{q_3\}$	$\emptyset$
$*q_5$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

EpsilonClosures (ε–closure)

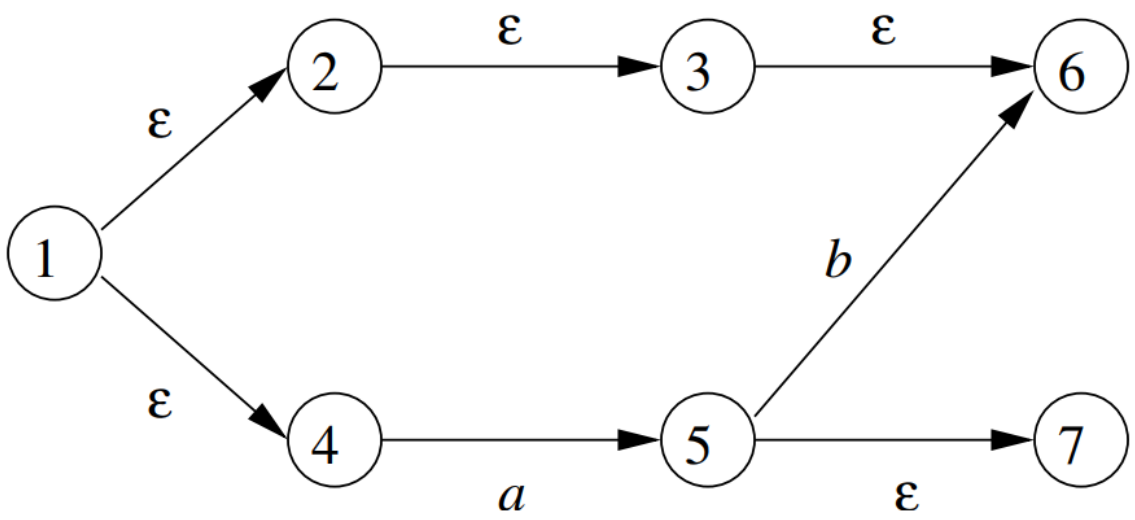
- Informally, we ε–close a state *q* by following all transitions out of *q* that are labeled ε.
- However, when we get to other states by following ε, we follow the ε-transitions out of those states, and so on,
  - eventually finding every state that can be reached from *q* along any path whose arcs are all labeled ε.
- Formally, we define the ε-closure *ECLOSE(q)* recursively, as follows:
  - State *q* is in *ECLOSE(q)*
  - If state p is in *ECLOSE(q)*, and there is a transition from state *p* to state *r* labeled ε, then *r* is in *ECLOSE(q)*.
    - More precisely, if δ is the transition function of the ε-NFA involved, and *p* is in ECLOSE(q) then ECLOSE(q) also contains all the states in δ(p, ε)

Example 7



- ECLOSE(q_0) = {q_0, q_1}
- ECLOSE(q_3) = {q_3, q_5}

Example 8



- ECLOSE(1) =
- ECLOSE(2) =
- ECLOSE(3) =
- ECLOSE(4) =
- ECLOSE(5) =
- ECLOSE(6) =
- ECLOSE(7) =

Exercises

- Exercise 1: Consider the following ε-NFA

	ϵ	a	b	c
$\rightarrow p$	$\emptyset$	$\{p\}$	$\{q\}$	$\{r\}$
q	$\{p\}$	$\{q\}$	$\{r\}$	$\emptyset$
$*r$	$\{q\}$	$\{r\}$	$\emptyset$	$\{p\}$

a) Compute the ε-closure of each state.
b) Give all the strings of length three or less accepted by the automaton.
c) Convert the automaton to a DFA.