22/10/2023 @ 2:00 PM

CSSD 609: THEORY OF COMPUTATION, 2023/2024

M.Sc./M.PHIL. in COMPUTER SCIENCE, GHANA COMMUNICATION TECHNOLOGY UNIVERSITY (GCTU), TESANO-ACCRA.

LECTURE 2 – REGULAR LANGUAGES AND FINITE AUTOMATA.

FACILITATED BY: PROF. NANA YAW ASABERE AND MR. MARCELLINUS KUUBOORE.

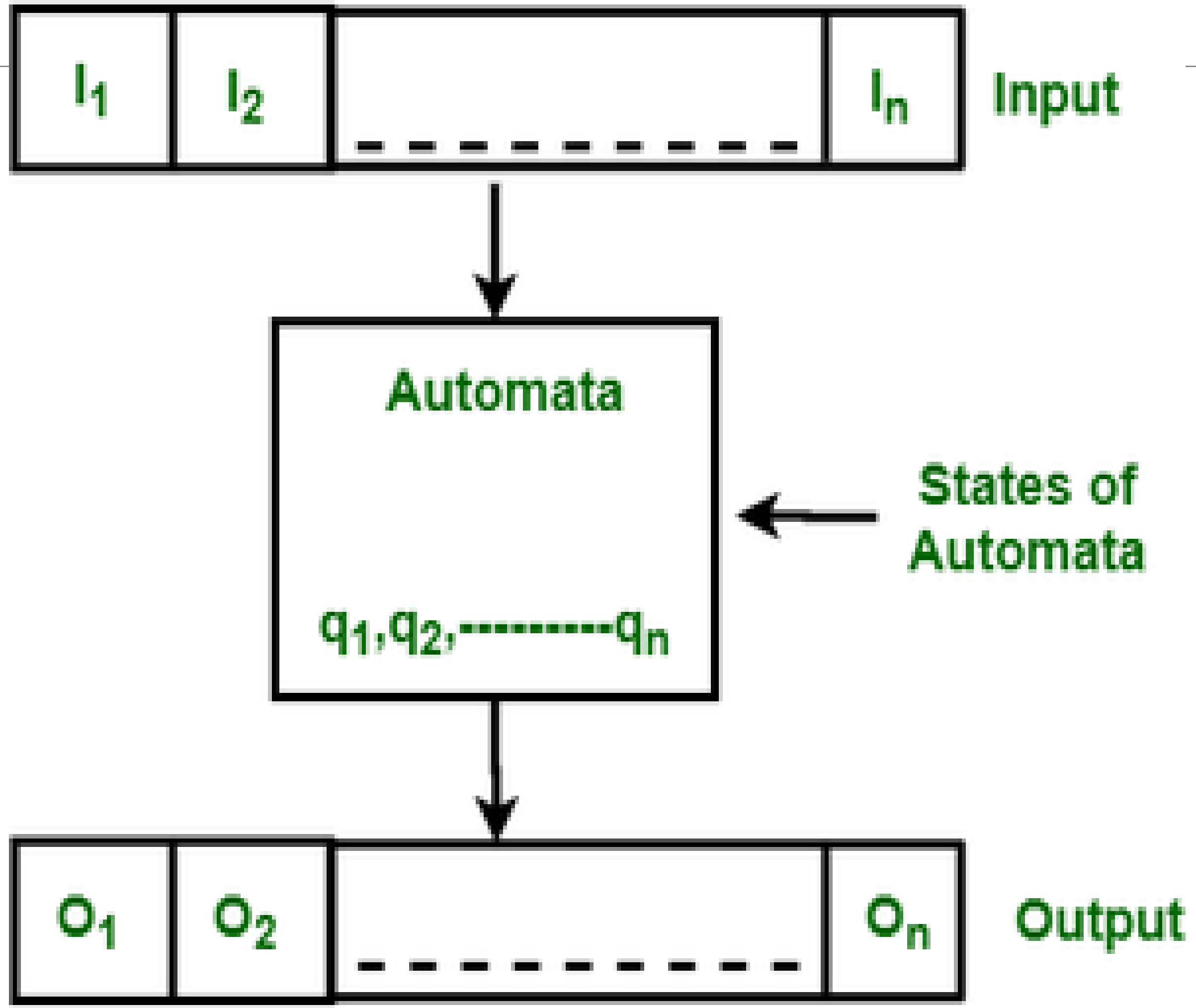
LECTURE OUTLINE

- Introduction to Regular Languages and Finite Automata
- Importance of Regular Languages and Finite Automata in Computer Science
- Deterministic Finite Automata (DFA)
- Non-deterministic Finite Automata (NFA)
- Equivalence Between Regular Expressions and NFAs/DFAs
- Closure Properties of Regular Languages

WHAT IS "FINITE AUTOMATA"

- **Finite Automata(FA)** is the simplest machine to recognize patterns. It is used to characterize a Regular Language.
- Also it is used to analyze and recognize Natural language Expressions.
- The finite automata or finite state machine is an abstract machine that has five elements or tuples.
- It has a set of states and rules for moving from one state to another but it depends upon the applied input symbol.
- Based on the states and the set of rules the input string can be either accepted or rejected.

WHAT IS “COMPUTATION” – IN RELATION TO AUTOMATA?

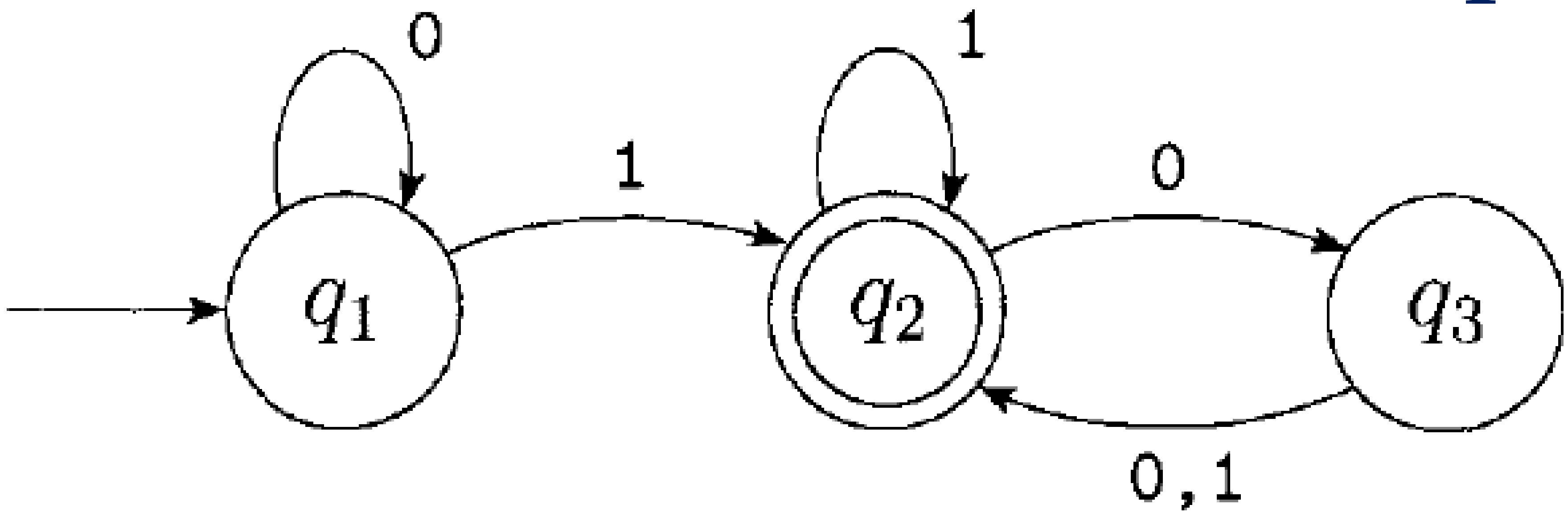


FORMAL DEFINITION OF A FINITE AUTOMATON

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

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

THE FINITE AUTOMATON M_1



- This figure depicts a finite automaton called M_1. It has three states namely $q_1, q_2, and\ q_3$.
- The ***Start state*** is $\boldsymbol{q_1}$**,**→○. Its indicated by the arrow pointing at it from nowhere.
- The ***Accept State*** q_2, is the one with double circle. The arrows going from one state to another are called ***Transition.***

EXAMPLE

Input: finite string
Output: Accept or Reject

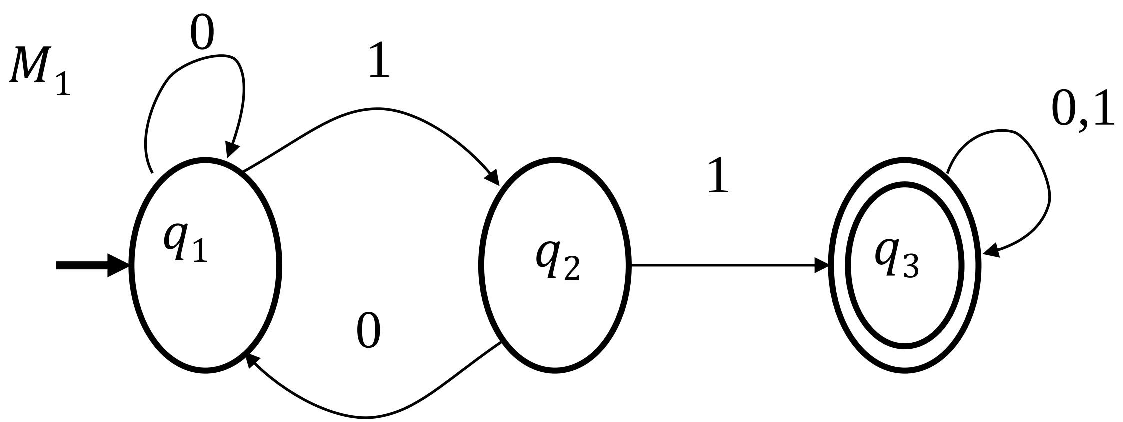
Computation process: Begin at start state,
read input symbols, follow corresponding transitions,
Accept if end with accept state, Reject if not.

Examples: 01101 → Accept
00101 → Reject

M_1 accepts exactly those strings in A where
$A = \{w \mid w \text{ contains substring } 11\}$.

Say that A is the language of $M1$ and that $M1$ recognizes A and that $A = L(M1)$.

FINITE AUTOMATA – FORMAL DEFINITION EXAMPLE



$M_1 = (Q, \Sigma, \delta, q_1, F)$

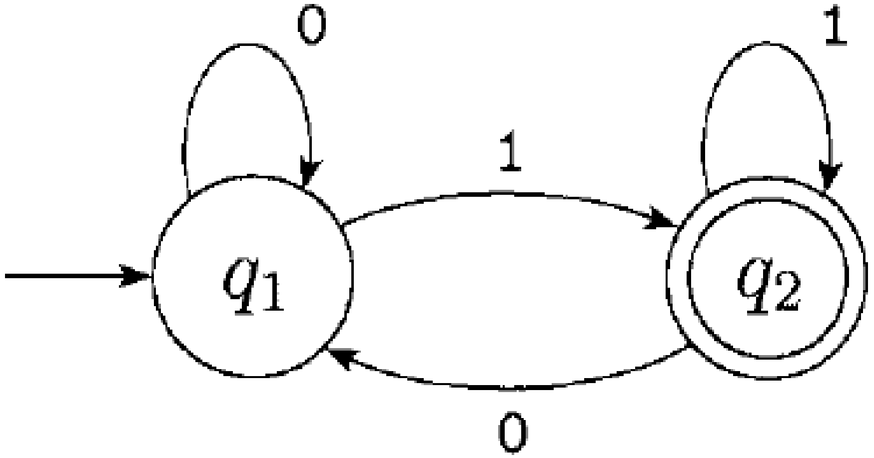
$Q = \{q_1, q_2, q_3\}$

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

$F = \{q_3\}$

$\delta =$	0	1
q_1	q_1	q_2
q_2		
q_3		

EXAMPLE OF FINITE AUTOMATA



- Let's consider the state diagram of the finite automaton M_2.
- State diagram of the two-state finite automaton M_2.
- In the formal description $M_2 = (\{q_1, q_2\}, \{0, 1\}, \delta, q_1, \{q_2\})$. The transition function δ *is*

	0	1
q_1	q_1	q_2
q_2	q_1	q_2

- The M_2 accepts all string that end in a 1. thus $L(M_2) = \{w|w$ ends in a 1$\}$.

FINITE AUTOMATA – COMPUTATION

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each w_i is a member of the alphabet Σ. Then M ***accepts*** w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with three conditions:

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

Condition 1 says that the machine starts in the start state. Condition 2 says that the machine goes from state to state according to the transition function. Condition 3 says that the machine accepts its input if it ends up in an accept state. We say that M ***recognizes language*** A if $A = \{w | M \text{ accepts } w\}$.

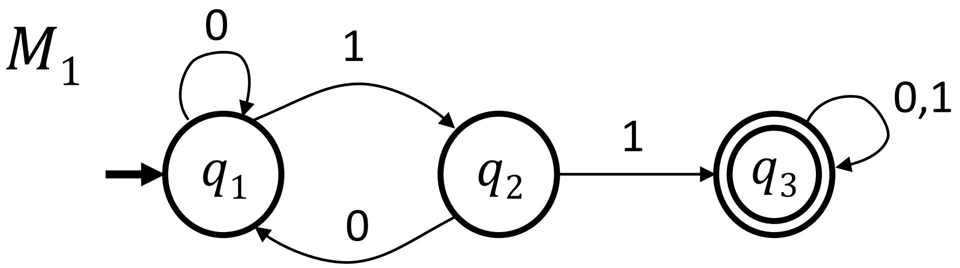
REGULAR LANGUAGE

- **When is a language regular?:** If we are able to construct one of the following: DFA or NFA or $\varepsilon - NFA\ or\ regular\ expression$.

- Its called a regular language if some finite automation recognizes it.

- $L(M) = \{w |\, M \text{ accepts } w\}$ $L(M)$ is the language of $M - M$ recognizes $L(M)$

REGULAR LANGUAGES – EXAMPLES



$L(M_1) =$
$\{w|\ w \text{ contains substring } 11\} = A$

<u>Therefore A is regular</u>

More examples:

- Let $B = \{w|\ w$ has an even number of 1s$\}$
- B is regular (make automaton for practice).
- Let $C = \{w|\ w$ has equal numbers of 0s and 1s$\}$
- C is <u>not</u> regular (we will prove).

REGULAR OPERATIONS

Regular operations. Let A, B be languages:

- **Union:** $A \cup B = \{w \mid w \in A \text{ or } w \in B\}$
- **Concatenation:** $A \circ B = \{xy \mid x \in A \text{ and } y \in B\} = AB$
- **Star:** $A^* = \{x_1 \dots x_k \mid \text{each } x_i \in A \text{ for } k \geq 0\}$

  Note: $\varepsilon \in A^*$ always

Example. Let $A = \{\text{good, bad}\}$ and $B = \{\text{boy, girl}\}$.

- $A \cup B = \{\text{good, bad, boy, girl}\}$
- $A \circ B = AB = \{\text{goodboy, goodgirl, badboy, badgirl}\}$
- $A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad}, \dots\}$

Regular expressions

- Built from Σ, members $\Sigma, \emptyset, \varepsilon$ [Atomic]
- By using $\cup, \circ, *$ [Composite]

Examples:

- $(0 \cup 1)^* = \Sigma^*$ gives all strings over Σ
- $\Sigma^* 1$ gives all strings that end with 1
- $\Sigma^* 11 \Sigma^* =$ all strings that contain $11 = L(M_1)$

DETERMINISTIC FINITE AUTOMATA (DFA)

- DFA is a fundamental concept in automata theory and formal language theory, used to recognize and accept strings in regular languages.

- A DFA is a mathematical model and an abstract machine that operates in a deterministic manner, meaning that for each state and input symbol, there is exactly one transition.

- DFAs are used for tasks such as pattern matching, lexical analysis in compilers, and various string processing tasks. They provide a clear and efficient method for recognizing regular languages.

DETERMINISTIC FINITE AUTOMATA (DFA)

- DFAs are vital concepts in formal language theory, and they play a central role in understanding the hierarchy of formal languages, with regular languages being the simplest in this hierarchy.

COMPONENTS AND CHARACTERISTICS OF A DFA

- **States (Q):** A DFA consists of a finite set of states. Each state represents a distinct configuration of the machine. States are typically represented by symbols such as q0, q1, q2, and so on.
- **Alphabet (Σ):** The alphabet is a finite set of input symbols. These symbols are the characters that can be read by the DFA. Common examples include binary (0, 1) or alphanumeric characters.
- **Transition Function (δ):** The transition function δ defines how the DFA moves from one state to another based on the input symbol. It is a function that takes a current state and an input symbol and returns the next state. Formally, $\delta: Q \times \Sigma \rightarrow Q$.

COMPONENTS AND CHARACTERISTICS OF A DFA (Cont'd)

- **Start State (q0):** The start state is the initial state of the DFA, where the machine begins processing input.
- **Accepting States (F):** The accepting states, also known as final states, are a subset of the set of states. When the DFA reaches an accepting state after processing an entire input string, it accepts that string. This indicates that the input string belongs to the language recognized by the DFA.
- **Non-accepting States:** All other states are non-accepting states. If the DFA reaches a non-accepting state after processing an input string, it rejects that string.

COMPONENTS AND CHARACTERISTICS OF A DFA (Cont'd)

- **Acceptance of Strings:** A DFA accepts a string if, after processing all the input symbols, it ends up in an accepting state. If it ends up in a non-accepting state, it rejects the string.

- **Deterministic Behavior:** The most crucial feature of a DFA is its deterministic behavior. For a given state and input symbol, there is only one possible next state. This deterministic property simplifies the design, analysis, and implementation of DFAs.

NON-DETERMINISM

- Non-determinism is a concept in automata theory and theoretical computer science that represents a different mode of operation in finite automata, particularly in Nondeterministic Finite Automata (NFAs).

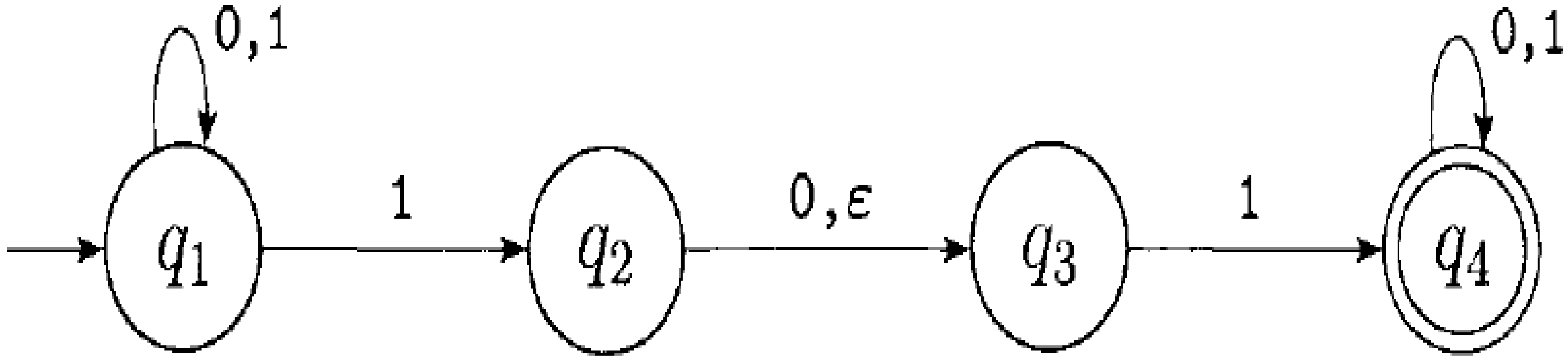
- NFAs contrast with Deterministic Finite Automata (DFAs), are primarily in their acceptance criteria and the transitions between states.

NON-DETERMINISM (Cont'd)

- Non-determinism is a crucial concept in automata theory because it allows for more expressive and concise representations of certain language classes. It also adds a layer of complexity in analysis and implementation.

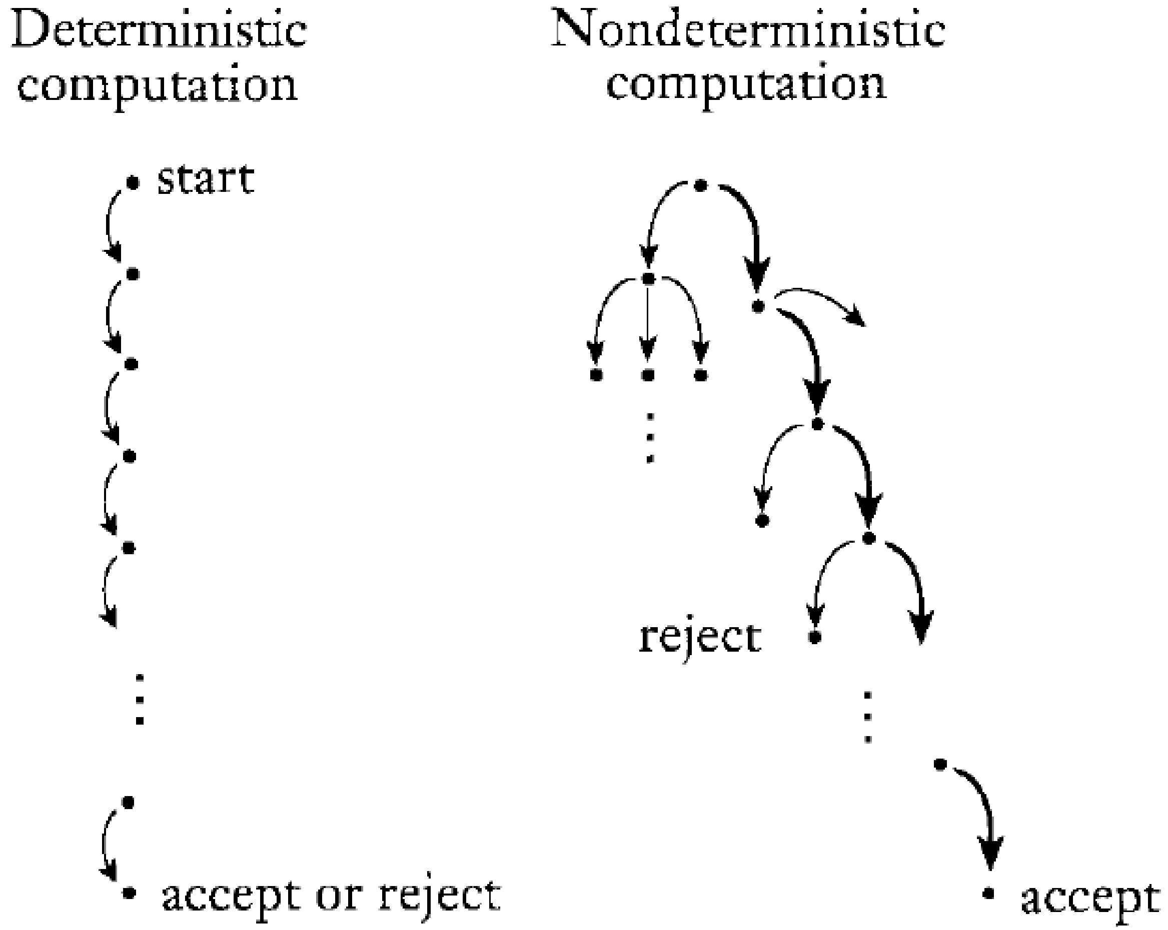
- While DFAs are typically easier to construct and analyze, NFAs can be more intuitive for describing patterns and languages with multiple possible paths.

NON-DETERMINISM (Cont'd)

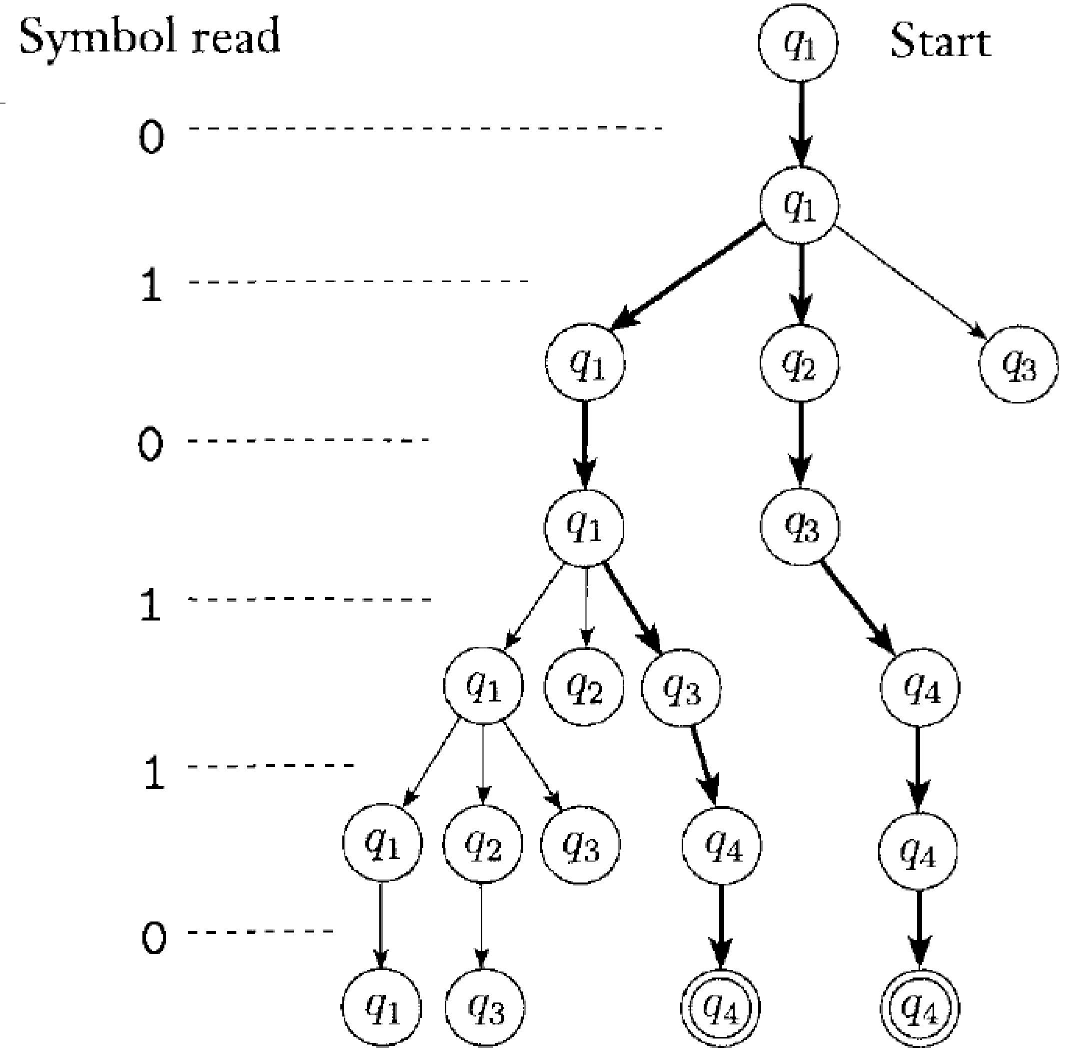


The non-deterministic finite automaton N_1

DETERMINISTIC & NON-DETERMINISTIC COMPUTATION WITH AN ACCEPTING BRANCH



THE COMPUTATION OF N_1 ON INPUT 010110



FORMAL DEFINITION OF NFA

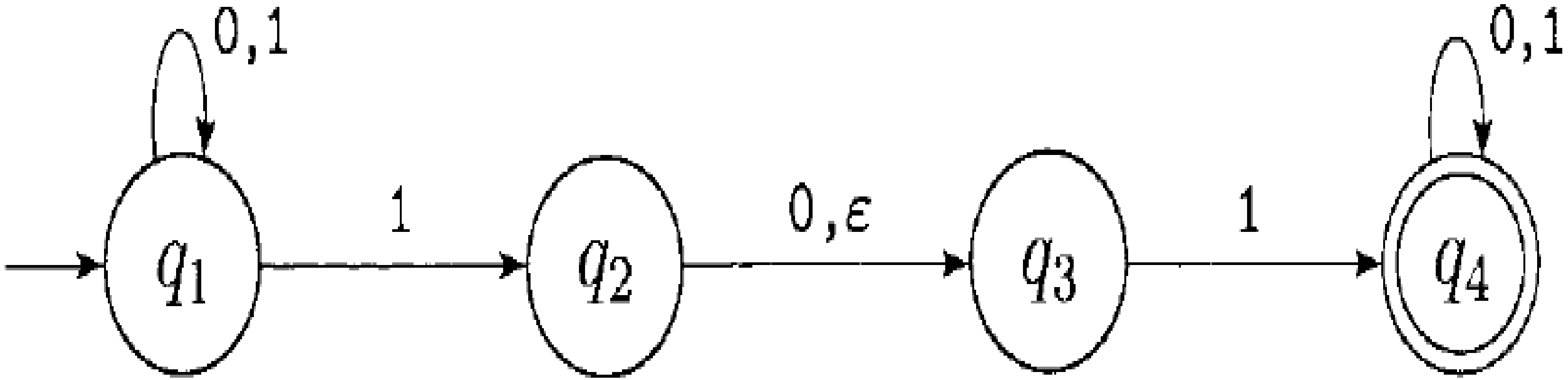
- The formal definition of NFA is similar to that of a DFA.
- They both have states, an input alphabet, a transition function, a start state, and a collection of accept states.

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. $\delta \colon Q \times \Sigma_\varepsilon \longrightarrow \mathcal{P}(Q)$ 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 OF NFA

- Recall the NFA N_1:
- The formal description of N_1 is $(Q, \Sigma, \delta, q_1, F)$, Where



EXAMPLE OF NFA SOLUTION

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. $F = \{q_4\}$.

EQUIVALENCE OF NFAs & DFAs

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

$$\delta'(R, a) = \bigcup_{r \in R} \delta(r, a).$$ [4]

EQUIVALENCE OF NFAs & DFAs (Cont'd)

3. $q_0{}' = \{q_0\}$.
   M starts in the state corresponding to the collection containing just the start state of N.

4. $F' = \{R \in Q' |\ R \text{ contains an accept state of } N\}$.
   The machine M accepts if one of the possible states that N could be in at this point is an accept state.

CLOSURE UNDER THE REGULAR OPERATIONS

- Amin is to prove that the union, concatenation, star of regular languages are till regular.

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.

CLOSURE UNDER THE REGULAR OPERATIONS (Cont'd)

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.
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 $a \in \Sigma_\varepsilon$,

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

CLASS OF REGULAR LANGUAGE UNDER THE CONCATENATION OPERATION

- If we have regular language A_1 & A_2 and want to show that $A_1 \, o \, A_2$ is regular.
- The idea is to take to NFAs N_1 & N_2 for A_1 & A_2, and combine them into the new NFA N as did for the case of union, but it's different way now.

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.

CLASS OF REGULAR LANGUAGE UNDER THE CONCATENATION OPERATION (Cont'd)

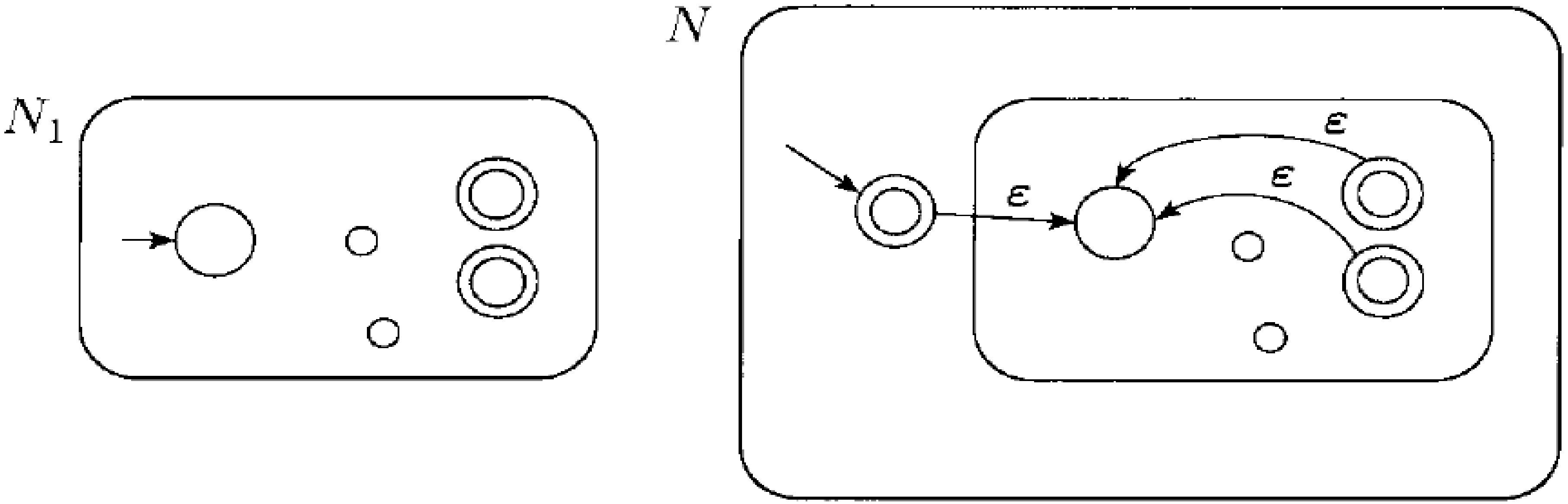
Construct $N = (Q, \Sigma, \delta, q_1, F_2)$ to recognize $A_1 \circ A_2$.

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.
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 $a \in \Sigma_\varepsilon$,

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

CLASS OF REGULAR LANGUAGE UNDER THE STAR (*) OPERATION

- If we have regular language A_1 and want to show that A_1^* also is regular.
- The idea is to take to NFAs N_1 for A_1, and modify it to recognize A_1^* as shown below.



CLASS OF REGULAR LANGUAGE UNDER THE STAR (*) OPERATION (Cont'd)

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.

CLASS OF REGULAR LANGUAGE UNDER THE STAR (*) OPERATION (Cont'd)

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

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

REGULAR EXPRESSIONS

➢ Formal Definition of a 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. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

EQUIVALENCE WITH FINITE AUTOMATA

- Both regular expression and finite automata are equivalent in their respective power.

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

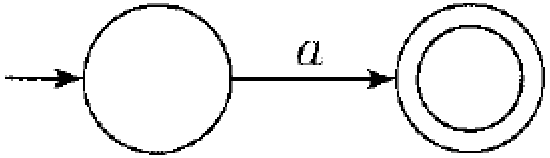
- If a language is describe by a regular expression, then it is regular.

EQUIVALENCE WITH FINITE AUTOMATA (Cont'd)

- Say that we have a regular expression R describing some language A.
- We show how to convert R into an NFA recognizing A. If an NFA recognizes A then A is regular.

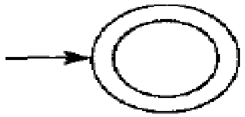
PROOF Let's convert R into an NFA N. We consider the six cases in the formal definition of regular expressions.

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



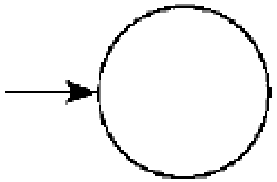
EQUIVALENCE WITH FINITE AUTOMATA (Cont'd)

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



Formally, $N = (\{q_1\}, \Sigma, \delta, q_1, \{q_1\})$, where $\delta(r, b) = \emptyset$ for any r and b.

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



Formally, $N = (\{q\}, \Sigma, \delta, q, \emptyset)$, where $\delta(r, b) = \emptyset$ for any r and b.

EQUIVALENCE WITH FINITE AUTOMATA (Cont'd)

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

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

END OF LECTURE

ANY QUESTIONS??

NEXT: NON-REGULAR LANGUAGES AND CONTEXT-FREE LANGUAGES