Foundations of Computing

Lecture 4

Arkady Yerukhimovich

January 23, 2025

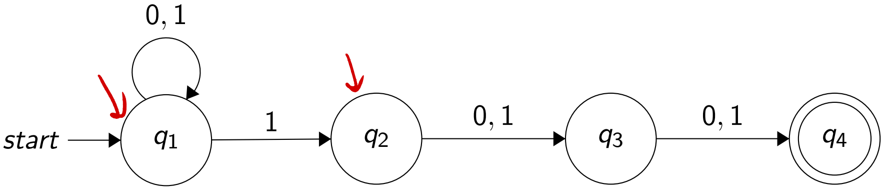
Outline

Lecture 3 Review

- Regular Languages
- Nondeterministic Finite Automata
- Understanding Nondeterminism

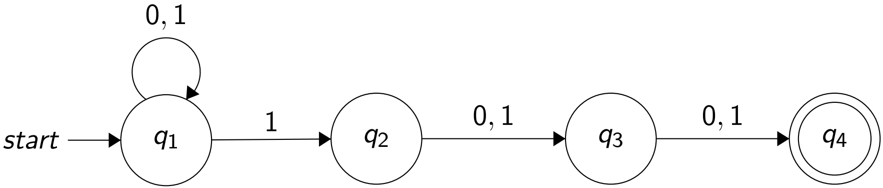
Outline

NFA Example 1



Question: Does M accept 01101? 01001?

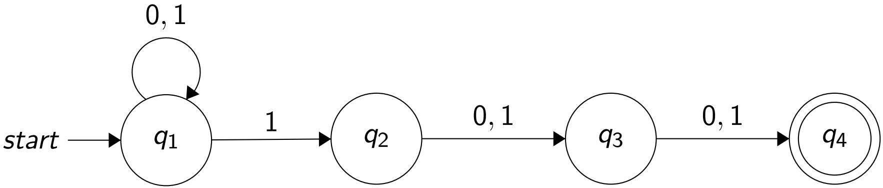
NFA Example 1



Question: Does M accept 01101? 01001?

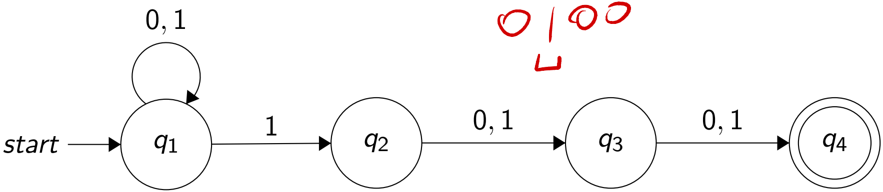
Question: What is $L(M)$?

NFA Example 1



Question: Does M accept 01101? 01001?
Question: What is $L(M)$?
Answer: Strings in $\{0, 1\}^*$ with a 1 as third from the end

NFA Example 1

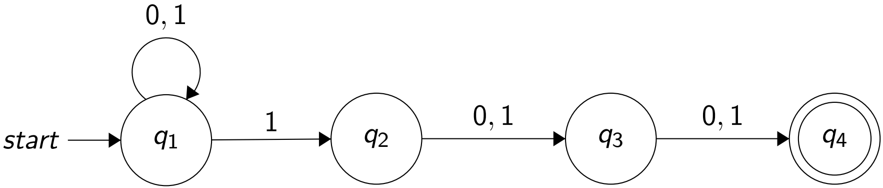


Question: Does M accept 01101? 01001?
Question: What is $L(M)$?
Answer: Strings in $\{0, 1\}^*$ with a 1 as third from the end

How does it work?

- M waits in q_1 until it "guesses" that it is 3 symbols from the end

NFA Example 1

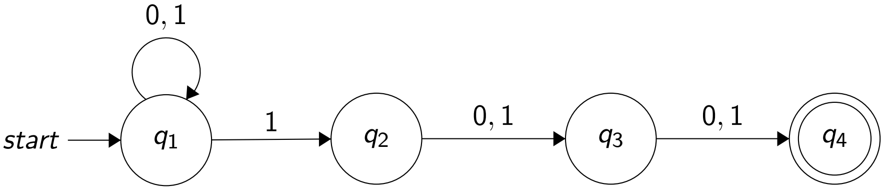


Question: Does M accept 01101? 01001?
Question: What is $L(M)$?
Answer: Strings in $\{0, 1\}^*$ with a 1 as third from the end

How does it work?

- M waits in q_1 until it "guesses" that it is 3 symbols from the end
- Uses the rest of the states to verify that 1 is third from the end

NFA Example 1



Question: Does M accept 01101? 01001?
Question: What is $L(M)$?
Answer: Strings in $\{0, 1\}^*$ with a 1 as third from the end

How does it work?

- M waits in q_1 until it "guesses" that it is 3 symbols from the end
- Uses the rest of the states to verify that 1 is third from the end
- DFA doing the same thing would have to track the last three bits seen – requires 8 states

Example 2 – OR statement

$L = \{x | x \in \{0, 1\}^*$ and x contains

1. the substring 101, or
2. the substring 010$\}$

Example 2 – OR statement

$L = \{x | x \in \{0, 1\}^*$ and x contains

1. the substring 101, or
2. the substring 010$\}$

Algorithm:

1. Guess which of 101 or 010 occur in the string
2. Verify (using DFA) that this is indeed the case

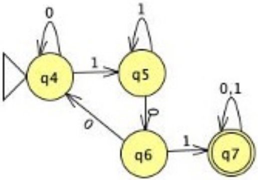
Example 2 – OR statement

$L = \{x | x \in \{0,1\}^*$ and x contains

1. the substring 101, or
2. the substring 010$\}$

Algorithm:

1. Guess which of 101 or 010 occur in the string
2. Verify (using DFA) that this is indeed the case



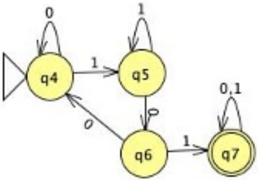
Example 2 – OR statement

$L = \{x | x \in \{0,1\}^* \text{ and } x \text{ contains}$

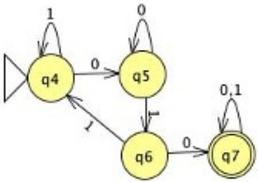
1. the substring 101, or
2. the substring 010$\}$

Algorithm:

1. Guess which of 101 or 010 occur in the string
2. Verify (using DFA) that this is indeed the case



DFA for prop. (1)

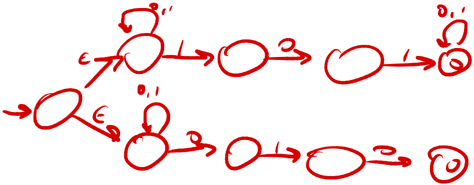


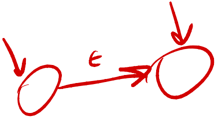
DFA for prop. (2)

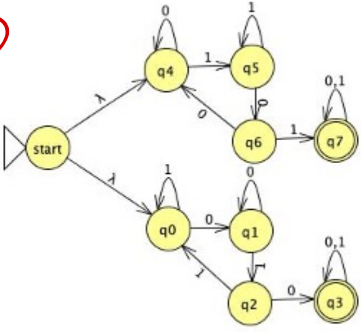
Example 2 – OR statement

$L = \{x | x \in \{0,1\}^* \text{ and } x \text{ contains}$

1. the substring 101, or
2. the substring 010$\}$



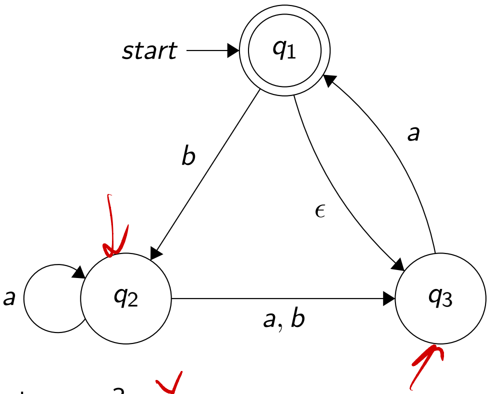




NFA for L

Quiz

Quiz



1. Does N accept $w = \epsilon$? Y
2. Does N accept $w = aaa$? Y
3. Does N accept $w = babba$? N
4. Does N accept $w = abaaba$? Y

NFA Summary

- NFAs are much simpler to design
- Only need to verify that inputs have correct form
- Ability to "guess" when some checkable property occurs is very useful

NFA Summary

- NFAs are much simpler to design
- Only need to verify that inputs have correct form
- Ability to “guess” when some checkable property occurs is very useful

Question

Are NFAs more powerful than DFAs?

Outline

Nondeterministic Finite Automaton – Formal Definition

Nondeterministic Finite Automaton (NFA)

An NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

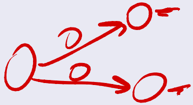
- Q is a finite set of states
- Σ is a finite input alphabet
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to P(Q)$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

Recall:
$P(Q)$ is the power set of Q, i.e., the set of all subsets of Q

Nondeterministic Finite Automaton – Formal Definition

Nondeterministic Finite Automaton (NFA)

An NFA is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:



- Q is a finite set of states
- Σ is a finite input alphabet
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to P(Q)$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

Recall:
$P(Q)$ is the power set of Q, i.e., the set of all subsets of Q

Changes:

1. Transition function allows empty symbol (ϵ)
2. Output of transition function is a set of states $\in P(Q)$, not a single state in Q

DFAs == NFAs

Theorem

For every NFA N there exists an equivalent DFA M

DFAs == NFAs

Theorem

For every NFA N there exists an equivalent DFA M

Intuition:

- Recall how we simulated NFA N by highlighting a set of nodes

DFAs == NFAs

Theorem

For every NFA N there exists an equivalent DFA M

Intuition:

- Recall how we simulated NFA N by highlighting a set of nodes
- Each transition moved us to a new set of nodes

DFAs == NFAs

Theorem

For every NFA N there exists an equivalent DFA M

Intuition:

- Recall how we simulated NFA N by highlighting a set of nodes
- Each transition moved us to a new set of nodes
- Accept if any of the highlighted nodes end in accept state

DFAs == NFAs

Theorem

For every NFA N there exists an equivalent DFA M

Intuition:

- Recall how we simulated NFA N by highlighting a set of nodes
- Each transition moved us to a new set of nodes
- Accept if any of the highlighted nodes end in accept state

A little more detail:

- Let node of DFA M represent set of “highlighted” nodes

DFAs == NFAs

Theorem

For every NFA N there exists an equivalent DFA M

Intuition:

- Recall how we simulated NFA N by highlighting a set of nodes
- Each transition moved us to a new set of nodes
- Accept if any of the highlighted nodes end in accept state

A little more detail:

- Let node of DFA M represent set of “highlighted” nodes
- Define δ to move to new set of highlighted nodes

DFAs == NFAs

Theorem

For every NFA N there exists an equivalent DFA M

Intuition:

- Recall how we simulated NFA N by highlighting a set of nodes
- Each transition moved us to a new set of nodes
- Accept if any of the highlighted nodes end in accept state

A little more detail:

- Let node of DFA M represent set of "highlighted" nodes
- Define δ to move to new set of highlighted nodes
- Accept states are ones in which at least one node is an accept node

DFAs == NFAs

Theorem

For every NFA N there exists an equivalent DFA M

Intuition:

- Recall how we simulated NFA N by highlighting a set of nodes
- Each transition moved us to a new set of nodes
- Accept if any of the highlighted nodes end in accept state

A little more detail:

- Let node of DFA M represent set of "highlighted" nodes
- Define δ to move to new set of highlighted nodes
- Accept states are ones in which at least one node is an accept node
- Can deal with ϵ edges by "placing more fingers" on resulting nodes

Making it Formal

Let N be an NFA recognizing L. Contruct DFA M recognizing L

Making it Formal

Let N be an NFA recognizing L. Contruct DFA M recognizing L

1. $Q' = P(Q)$ – power set of Q

Making it Formal

Let N be an NFA recognizing L. Contruct DFA M recognizing L

1. $Q' = P(Q)$ – power set of Q
2. For $R \in Q'$ and $a \in \Sigma$, let

$R = \{r_1, r_2\}$

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

Look at transitions from all states in set R and map to set that gives results of all these transitions

Making it Formal

Let N be an NFA recognizing L. Contruct DFA M recognizing L

1. $Q' = P(Q)$ – power set of Q
2. For $R \in Q'$ and $a \in \Sigma$, let

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

   Look at transitions from all states in set R and map to set that gives results of all these transitions
3. $q'_0 = \{q_0\}$

Making it Formal

Let N be an NFA recognizing L. Contruct DFA M recognizing L

1. $Q' = P(Q)$ – power set of Q
2. For $R \in Q'$ and $a \in \Sigma$, let

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

   Look at transitions from all states in set R and map to set that gives results of all these transitions
3. $q_0' = \{q_0\}$
4. $F' = \{R \in Q' | R \text{ contains an accept state of } N\}$
   Accept if any state in R is an accept state

Handling ϵ transitions

Problem: We need to also get rid of any ϵ edges

Handling ϵ transitions

Problem: We need to also get rid of any ϵ edges

Intuition: For every ϵ edge, just place a new "finger" on the graph

Handling ϵ transitions

Problem: We need to also get rid of any ϵ edges

Intuition: For every ϵ edge, just place a new "finger" on the graph

Formally:

1. Let $E(R) = \{q | q \text{ can be reached from } R \text{ along } \epsilon \text{ arrows}\}$

Handling ϵ transitions

Problem: We need to also get rid of any ϵ edges

Intuition: For every ϵ edge, just place a new "finger" on the graph

Formally:

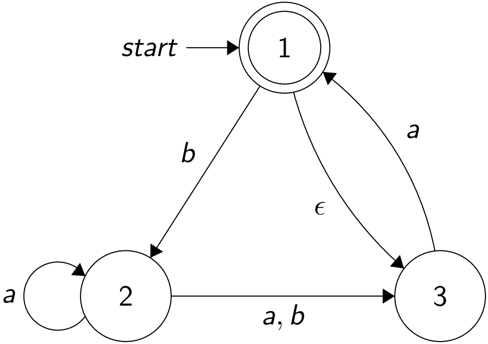
1. Let $E(R) = \{q | q \text{ can be reached from } R \text{ along } \epsilon \text{ arrows}\}$
2. Define extended transition function

$$\delta'(R, a) = \cup_{r \in R} E(\delta(r, a))$$

Map to set of states that can be reached on input a or $a\epsilon$

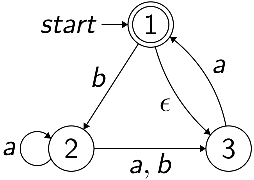
An Example: NFA $\rightarrow$ DFA

NFA $N = (Q, \Sigma, \delta, q_0, F)$



An Example: NFA $\rightarrow$ DFA

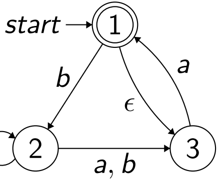
NFA $N = (Q, \Sigma, \delta, q, F)$



1. states: $Q' =$

An Example: NFA $\to$ DFA

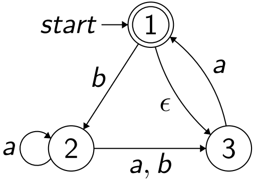
NFA $N = (Q, \Sigma, \delta, q, F)$



1. states: $Q' = P(Q) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
2. start state: $q' =$

An Example: NFA $\to$ DFA

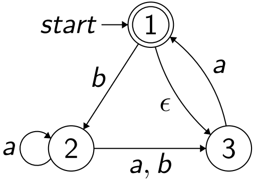
NFA $N = (Q, \Sigma, \delta, q, F)$



1. states: $Q' = P(Q) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
2. start state: $q' = E(1) = \{1, 3\}$
3. accept states: $F =$

An Example: NFA $\rightarrow$ DFA

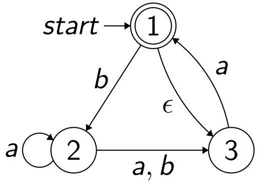
NFA $N = (Q, \Sigma, \delta, q, F)$



1. states: $Q' = P(Q) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$
2. start state: $q' = E(1) = \{1,3\}$
3. accept states: $F = \{\{1\}, \{1,2\}, \{1,3\}, \{1,2,3\}\}$

An Example: NFA $\rightarrow$ DFA

NFA $N = (Q, \Sigma, \delta, q, F)$



4. Transition function δ':

$\delta'(\emptyset, a) = \emptyset$

$\delta'(\{1\}, a) = \emptyset$

$\delta'(\{2\}, a) = \{2,3\}$

$\delta'(\{1,2\}, a) = \{2,3\}$

$\delta'(\{3\}, a) = \{1,3\}$

$\delta'(\{1,3\}, a) =$

$\delta'(\{2,3\}, a) =$

$\delta'(\{1,2,3\}, a) =$

$\delta'(\emptyset, b) =$

$\delta'(\{1\}, b) =$

$\delta'(\{2\}, b) =$

$\delta'(\{1,2\}, b) =$

$\delta'(\{3\}, b) =$

$\delta'(\{1,3\}, b) =$

$\delta'(\{2,3\}, b) =$

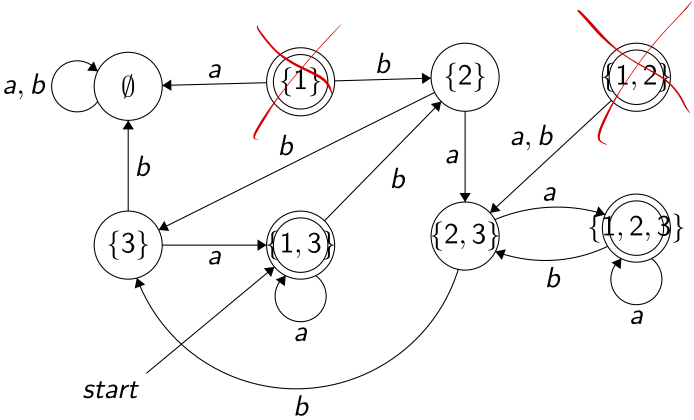
$\delta'(\{1,2,3\}, b) =$

An Example: NFA $\rightarrow$ DFA

4. Transition function δ':

$\delta'(\emptyset, a) = \emptyset$	$\delta'(\emptyset, b) = \emptyset$
$\delta'(\{1\}, a) = \emptyset$	$\delta'(\{1\}, b) = \{2\}$
$\delta'(\{2\}, a) = \{2, 3\}$	$\delta'(\{2\}, b) = \{3\}$
$\delta'(\{1, 2\}, a) = \{2, 3\}$	$\delta'(\{1, 2\}, b) = \{2, 3\}$
$\delta'(\{3\}, a) = \{1, 3\}$	$\delta'(\{3\}, b) = \emptyset$
$\delta'(\{1, 3\}, a) = \{1, 3\}$	$\delta'(\{1, 3\}, b) = \{2\}$
$\delta'(\{2, 3\}, a) = \{1, 2, 3\}$	$\delta'(\{2, 3\}, b) = \{3\}$
$\delta'(\{1, 2, 3\}, a) = \{1, 2, 3\}$	$\delta'(\{1, 2, 3\}, b) = \{2, 3\}$

An Example: NFA $\rightarrow$ DFA



Outline

A Useful Corollary

Recall that:

Definition

A language L is regular if and only if there is a DFA that recognizes it

A Useful Corollary

Recall that:

Definition

A language L is regular if and only if there is a DFA that recognizes it

Since we now know that NFAs and DFAs are equal:

Corollary

A language L is regular if and only if there is an NFA that recognizes it

A Useful Corollary

Recall that:

Definition

A language L is regular if and only if there is a DFA that recognizes it

Since we now know that NFAs and DFAs are equal:

Corollary

A language L is regular if and only if there is an NFA that recognizes it

We can now use NFAs to argue the properties of regular languages

Closure Under Union

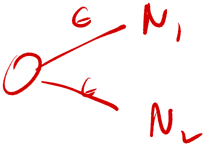
Closure Under Union

If L_1 and L_2 are both regular languages then $L_1 \cup L_2$ is also regular

$L_1 \cup L_2$ is the language consisting of all strings either in L_1 or L_2

Proof:

N_1 N_2



Closure Under Concatenation

Closure Under Concatenation

If L_1 and L_2 are both regular languages then $L_1 \circ L_2$ is also regular

$L_1 \circ L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$

Proof:

Closure Under the Star Operation

Closure Under Star Operation

If L is a regular languages then L^* is also regular

$L^* = \{0 \text{ or more strings from } L\}$

Proof: