CSC 320 - Lecture 04

#languages #regular #closure #DFA #NFA

Regular Languages

Closure Properties (2)

Theorem. If L_1 and L_2 are regular languages over alphabet Σ then $L_1 \cap L_2$ is a regular language.

Proof. Since L_1 and L_2 are regular languages there exists finite automata M_1 and M_1 with $L_1 = L(M_1) \text{ and } L_2 = L(M_2)$.

Idea. Construct a DFA M that accepts exactly the strings accepted by M_1 and M_2; similar to previous proof but need to pay attention what strings must be accepted by M.

Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$. We construct $M = (Q, \Sigma, \delta, q_0, F)$ as follows...

- $Q = \{(r_1, r_2) | r_1 \in Q_1, r_2 \in Q_2\}$.
- For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$: $\delta((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
- $q_0 = (q_1, q_2)$
- $F = \{(r_1, r_2) | r_1 \in F_1 \text{ AND } r_2 \in F_2\}$.

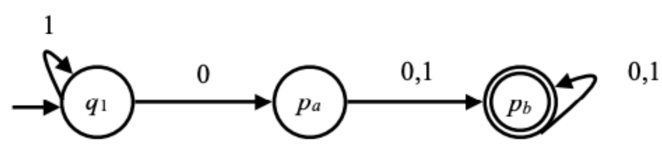
M recognizes $L_1 \cap L_2$

Example

- Let $\Sigma = \{0, 1\}$
- $L_1 =$ is the set of all strings that, after a possible prefix of 1s, consist of at least one 0 followed by at least one symbol.
- $L_2 =$ is the set of all strings of length at exactly 1.

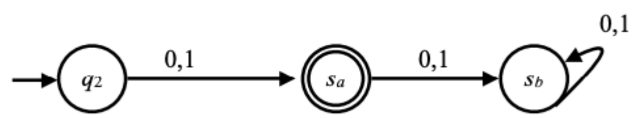
M_1

M_1



M_2

M_2

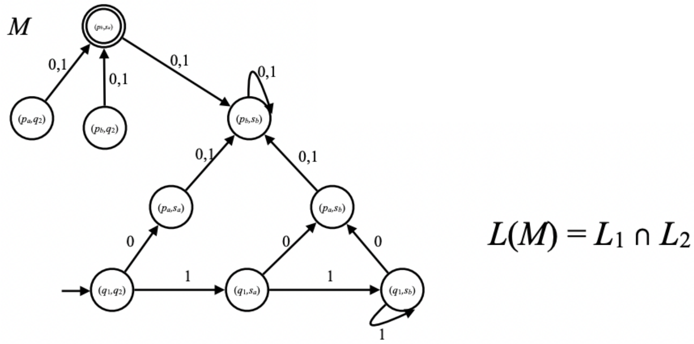


Q	**0**	**1**
(q_1, q_2)	(p_a, s_a)	(q_1, s_a)
(q_1, s_a)	(p_a, s_b)	(q_1, s_b)
(q_1, s_b)	(p_a, s_b)	(q_1, s_b)
(p_a, q_2)	(p_b, s_a)	(p_b, s_a)
(p_a, s_a)	(p_b, s_b)	(p_b, s_b)
(p_a, s_b)	(p_b, s_b)	(p_b, s_b)
(p_b, q_2)	(p_b, s_a)	(p_b, s_a)
(p_b, s_a)	(p_b, s_b)	(p_b, s_b)
(p_b, s_b)	(p_b, s_b)	(p_b, s_b)

Start: (q_1, q_2)

Q	**0**	**1**
(q_1, q_2)		
	(p_b, s_a)	(p_b, s_a)
	(p_b, s_a)	(p_b, s_a)
(p_b, s_a)		

M



Closure Properties (3)

Theorem. If L_1 and L_2 are regular languages over alphabet Σ then $L_1 L_2$ is a regular language.

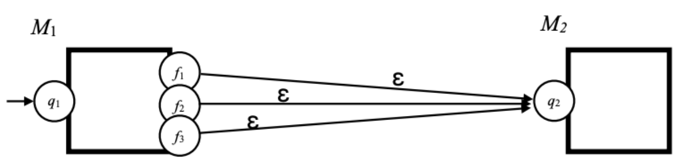
Proof. Since L_1 and L_2 are regular languages there exist DFA M_1 and M_2 with $L_1 = L(M_1)$ and $L_2 = L(M_2)$.

Idea. Construct a finite automaton M that accepts exactly all the strings of which the first part is accepted by M_1 and the second part by M_2.

New Idea. Construct a nondeterministic finite automaton M that accepts exactly the strings where the first part is accepted by first M_1 and the second one by M_2. (We want to use NFAs)!

Regular Languages Are Closed Under Concatenation

M



M inherits all states from DFAs M_1 and M_1 with...

- M's **start state** is M_1's start state.
- M's **final states** are M_2's final states.
- M's **transitions** consists of:
    - All transitions of M_1's and all transitions of M_2's.
    - ε-transitions between each state that corresponds to a final state in M_1 and the state the corresponds to M_2's start state.

Nondeterminism and Nondeterministic Finite Automata

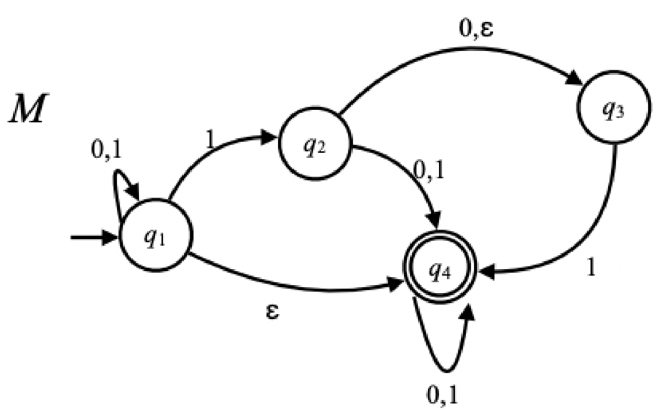
Abstraction that allows us to consider extension of ordinary computation. Simultaneous execution paths are permitted. Strings are accepted if any execution path is an accepting one.

Proving that regular languages are closed under concatenation: we show this for nondeterministic fine automata and their corresponding language.

Then we show that the nondeterministic finite automata (NFA) recognize the same class of languages as deterministic ones: regular languages.

NFA Example

$M$$



Note. You do not need to have a transition for each element in the alphabet. Likewise, you can have multiple transitions for each element in the alphabet.

Note. You can also include a transition to the empty set in your state diagram, but it is not needed (it is implied).

$M = (\{q_1, q_2, q_3, q_4\}, \{0, 1\}, \delta, q_1, \{q_4\})$ with $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ defined by...

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

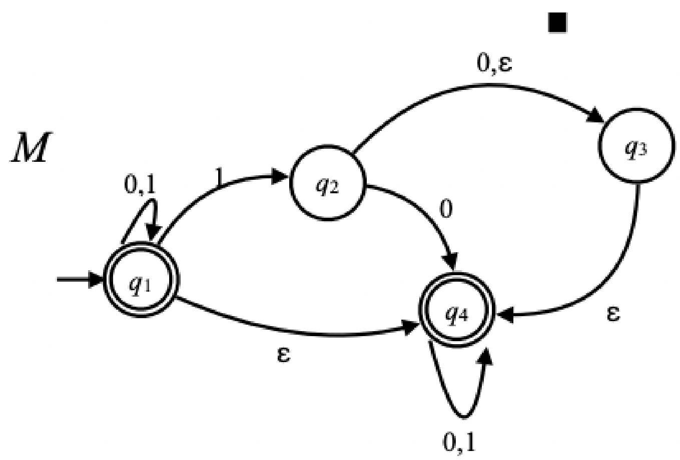
- $w_1 = 01100$
- $w_2 = 0$
- $w_3 = \varepsilon$

Nondeterministic Finite Automata (NFA)

- Transitions go from states to **sets of states**.
    - Starting from a state q and reading a symbol a can have transitions into more than one state.
        - $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$, where $\mathcal{P}(Q)$ is the powerset of Q.
- We allow **empty transitions** (ε-transitions).
    - Do not read any symbol from the input.

Example

NFA M



$M = (\{q_1, q_2, q_3, q_4\}, \{0, 1\}, \delta, q_1, \{q_4\})$ with $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ defined by...

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

Formal Definition Nondeterministic Finite Automaton

A **nondeterministic finite automaton (NFA)** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ with...

- Q is a finite set of states.
- Σ is an alphabet.
- Function $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q)$ is the transition function.
- $q_0 \in Q$ is the start state.
- $F \subseteq Q$ is the set of accept (or final) states.

NFA Computation

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a NFA and $w = w_1 w_2 ... w_n$ a string over Σ. Then M **accepts** w if we can write $w = y_1 y_2 ... y_n$ with $y_i \in \Sigma \cup \{\varepsilon\}$ and there is a sequence of states $r_0, r_2, \ldots, r_m \in Q$, such that...

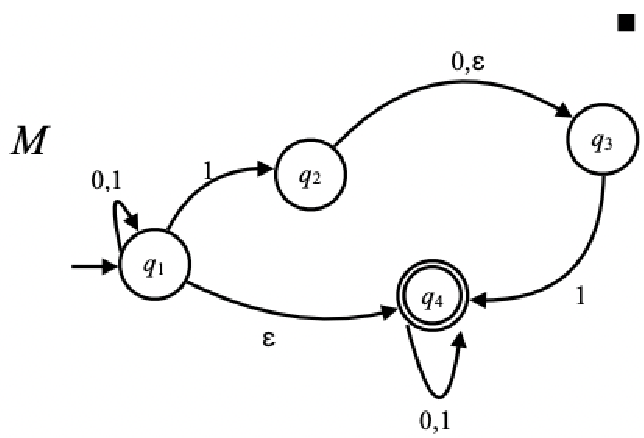
1. $r_0 = q_0$
2. $r_{i+1} \in \delta(r_i, y_{i+1})$
3. $r_m \in F$

Then M **recognizes** L if $L = L(M) = \{w \in \Sigma^* | M \text{ accepts } w\}$ (same as a DFA).

- If a machine M that does not accept any string then $L(M) = \emptyset$ (same as DFA).

NFA Example

M



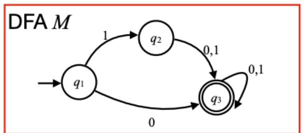
- Input $w = 1101$ is accepted by M. (q_1, q_1, q_2, q_3, q_4 is a state sequence.) (You could also have $q_1, q_1, q_1, q_1, q_1, q_4$ where we read 1101ε).
  - To show $w \in L(M)$: rewrite $w = 1\varepsilon 101$
  - State sequence for $q_1, q_2, q_3, q_4, q_4, q_4$
- $w : q_1, q_4, q_4, q_4, q_4$ is another sequence of states for w for (1101).

Note. Choosing sequence q_1, q_1, q_1, q_1 for $w = 1101$ does not yield acceptance.

NFAs and DFAs

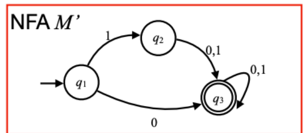
- A DFA can be considered a special case of NFA.
  - Main Difference: definition of a transition function.
  - Caution: Formal definition is different!

M and M'



$(\{q_1,q_2,q_3,q_4\},\ \{0,1\},\ \delta,\ q_1,\ \{q_3\})$ with
δ: $Q\times\Sigma\to Q$ defined by

δ	0	1
q_1	q_3	q_2
q_2	q_3	q_3
q_3	q_3	q_3



$(\{q_1,q_2,q_3,q_4\},\ \{0,1\},\ \delta,\ q_1,\ \{q_3\})$ with
δ: $Q\times(\Sigma\cup\{\varepsilon\})\to\mathcal{P}(Q)$ defined by

δ	0	1	ε
q_1	$\{q_3\}$	$\{q_2\}$	$\varnothing$
q_2	$\{q_3\}$	$\{q_3\}$	$\varnothing$
q_3	$\{q_3\}$	$\{q_3\}$	$\varnothing$

Note. Make sure you add those $\{\}$ when creating the table!

Definition: Let M_1 and M_2 each be a DFA or NFA. Then we call M_1 and M_2 **equivalent** if $L(M_1) = L(M_2)$.

- Since NFAs and DFAs produce the same set of languages we know:
  - The languages recognized by NFAs is exactly the set of regular languages.

First we show. For every DFA there exists an equivalent NFA.

For Every DFA There Exists An Equivalent NFA

Let $M = (Q, \Sigma, \delta, q_M, F)$ be a DFA. Then we can build NFA $N = (Q', \Sigma, \delta', q_N, F')$ with $L(M) = L(N)$ as follows...

- $Q' := Q$
- $q_N := q_M$
- $F' := F$
- $\delta' : Q' \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(Q')$ with $\delta'(q, a) := \{\delta(q, a)\}$ for all $a \in \Sigma$, $\delta'(q, \varepsilon) := \emptyset$.

Equivalence of NFAs and DFAs

Theorem. For every NFA there exists an equivalent DFA.

Proof.

- **Plan**. Given NFA $N = (Q, \Sigma, \delta, q_0, F)$ construct DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$ with $L(N) = L(D)$.
- **Idea**. Build D such that it simulates the computation of N.
  - To not miss any possible computation of N in simulation: when defining Q_D : create one state for **every possible subset** of Q.
  - Define δ_D for all those states in Q_D and all inputs symbols.
- **For Now**. Ignore ε-transitions in N (i.e., assume N does not have any ε-transitions; we will deal with them later).

Building D

Given NFA $N = (Q, \Sigma, \delta, q_0, F)$ (without any ε-transitions).

Build DFA $D = (Q_D, \Sigma, \delta_D, q_D, F_D)$...

- $Q_D := \mathcal{P}(Q), q_D := \{q_0\}$.
- Definition of δ_D
  - Let $S \in Q_D$ and $a \in \Sigma$; recall $S \subseteq Q$.
  - $\delta_D(S, a) := \{q \in Q | q \in \delta(s, a) \text{ for some } s \in S\}$.
- F_D: the set of all subsets of Q that contain final state of F.
  - $F_D := \{S \in Q_D | \text{ there exists a } q \in S \text{ with } q \in F\}$, that is $F_D = \{S \in Q_D | S \cap F \neq \emptyset\}$.

Note. Our construction so far only works for NFAs without ε-transitions. Thus, modify construction to also simulate NFAs with ε-transitions.

Adding ε-Transitions

Given NFA $N = (Q, \Sigma, \delta, q_0, F)$ (can have ε-transitions).

Use same construction ignoring ε-transitions. Then...

For any $S \in Q_D$ let $E(S) = \{q | q$ can be reached from some state in S by traveling 0 or more $\varepsilon\text{-transitions}\}$.

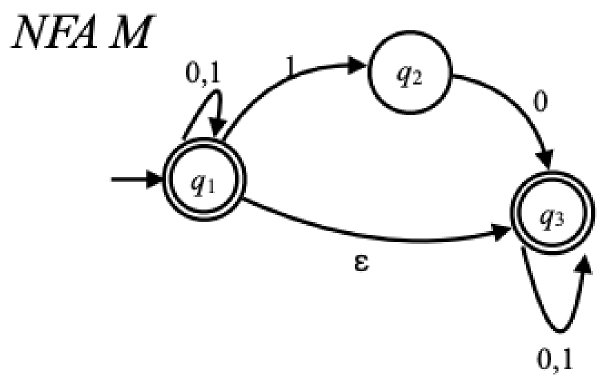
Note. This means we include states that we can get to with free hops (ε-transitions).

Modify D as follows...

- $q_D := E(\{q_0\})$
- $\delta_D(S, a) := \{q \in Q | q \in E(\delta(s, a)) \text{ for some } s \in S\}$ ■

Example

NFA M



- $Q_D := \mathcal{P}(Q)$
- $q_D := E(\{q_0\})$
- $\delta_D(S, a) := \{q \in Q | q \in E(\delta(s, a)) \text{ for some } s \in S\}$

	δ_D	0	1
	$\varnothing$	$\varnothing$	$\varnothing$
	$\{q_1\}$	**$\{q_1,q_3\}$**	**$\{q_1,q_2,q_3\}$**
	$\{q_2\}$	**$\{q_3\}$**	$\varnothing$
	$\{q_3\}$	**$\{q_3\}$**	**$\{q_3\}$**
	$\{q_1,q_2\}$	**$\{q_1,q_3\}$**	**$\{q_1,q_2,q_3\}$**
Start	**$\{q_1,q_3\}$**	**$\{q_1,q_3\}$**	**$\{q_1,q_2,q_3\}$**
	$\{q_2,q_3\}$	**$\{q_3\}$**	**$\{q_3\}$**
	$\{q_1,q_2,q_3\}$	**$\{q_1,q_3\}$**	**$\{q_1,q_2,q_3\}$**

The above is a DFA table.

Note. You should draw the failure state in a complete diagram. Bold face states are accept states.

Didn't We Say We Wanted to Study Problems and Their Solutions?

- A (**decision** or **yes/no**) **problem** is a mapping from a set of **problem instances** to Yes/No (called **yes-instances** and **no-instances**).
- Languages: abstract representation of problems.
- For a problem Π , the associated language L_Π is $L_\Pi = \{x \in \Sigma^* | x \text{ is a yes-instance of } \Pi\}$.

Yes-No-Problems and Their Languages

Examples

Sorted Sequence

- **Input**: A list of n comparable elements $e_1, e_2, \ldots, e_n$.
- **Question**: Are the elements, as given, in sorted order? That is: is it true that $e_1 \leq e_2 \leq \ldots \leq e_n$?

$L_{\text{SORTED SEQUENCE}} = \{\text{list of comparable elements } l \mid \text{the elements of } l \text{ are in sorted order}\}$.

Connected Graph

- **Input**: A simple, undirected graph $G = (V, E)$.
- **Question**: Is G connected? That is: for any pairs of vertices $x, y \in V$, does there exists a path from x to y in G?

$L_{\text{CONNECTED GRAPH}} = \{G = (V, E) | G \text{ is a simple, undirected connected graph}\}$.

Short Spanning Tree

- **Input**: A simple, undirected, edge-weighted graph $G = (V, E)$ where each edge $e \in E$ is assigned a positive integer weight $w(e)$, and integer k.
- **Question**: Does there exist a spanning tree $T = (V, E_T)$ for G where T has weight at most k? That is: T is a tree, $E_T \subseteq E$, and $\Sigma_{e \in E_T} w(e) \leq k$?

$L_{\text{SHORT SPANNING TREE}} = \{(G = (V, E), k) | k \text{ is a positive integer and } G$
$\text{ is a simple, undirected, edge-weighted graph has a spanning tree of weight at most } \ k\}$.

Previous Lecture

[Lecture03]

Next Lecture

Lecture05