IN22080

2. Finite Automata

Einar Broch Johnsen

University of Oslo

19th January 2022



Department of Informatics



University of Oslo

The Basics

- **Set**: an unordered collection of distinct objects called *elements*
  - $\{a, b\} = \{a, a, b\} = \{b, a\}$
- **Set union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$
- **Set intersection**: $A \cap B = \{x \mid x \in A \text{ and } x \in B\}$
- **Set complement**: $\bar{A} = \{x \mid x \notin A\}$
- **de Morgan's laws**:
  - $\overline{A \cup B} = \bar{A} \cap \bar{B}$ and
  - $\overline{A \cap B} = \bar{A} \cup \bar{B}$
- **Power Set**: $\mathcal{P}(A) = \{S \mid S \subseteq A\}$
  - Example: $\mathcal{P}(\{0, 1\}) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$.

The Basics

- **Tuple**: ordered collection of objects
  - $(a, a, b) \neq (a, b)$
- **Cartesian product**: $A \times B = \{(a, b) \mid a \in A, b \in B\}$
- **Function**: $f : A \to B$.
  - Assigns to each element $a \in A$ a unique element $f(a) \in B$.

Strings

- Let Σ be an alphabet (non-empty finitie set of symbols)
- A **string** is a finite sequence of symbols from Σ
- Empty string ε

- **Example:** *abracadabra* is a string over the alphabet $\Sigma = \{a, b, c, \ldots, z\}$

- A **language** = set of strings over an alphabet

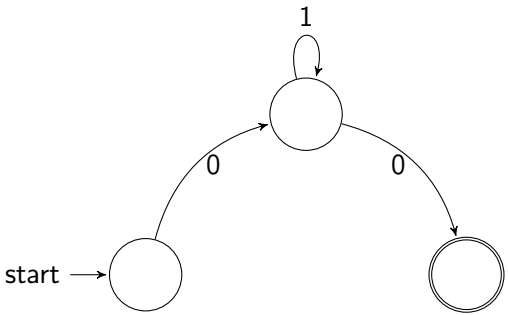
Finite Automata

- Computational model of a computer with finite memory
- Takes an input string w and decides whether to *accept* or *reject*
- Can be used to answer such questions as
  - "Is w a palindrome?"
  - "Is w a valid program in a given programming language?"
- Usually depicted as a graph for ease of reading:
  - Nodes represent *states* in which the automaton can be
  - Edges between nodes represent the transition between states given a parsed input
  - Always exactly one start node: 

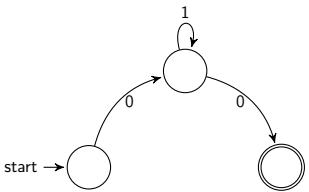
  - . . . as well as some accept states: 

Finite Automata

Example:



What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;
- the final state must be an accepting state.
- the example automaton accepts all inputs, *words*, that start and end with 0, with only 1's in between.

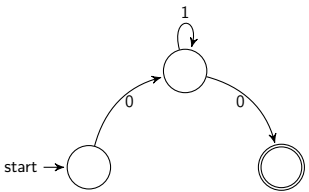
Deterministic Finite Automata

Definition

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

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

What does it mean for a finite automaton to "accept" an input w?



- starting at the start state, for each symbol in the input, follow a corresponding transition edge to the next state;
- the entire input must be parsed;
- the final state must be an accepting state.
- the example automaton accepts all inputs, *words*, that start and end with 0, with only 1's in between.

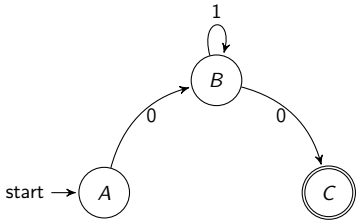
Acceptance for Deterministic Finite Automata

Definition

A DFA $(Q, \Sigma, \delta, q_0, F)$ *accepts* an input $w = w_1 w_2 \cdots w_n$ if there exists a sequence of states $s_0 \cdots s_n$ such that

1. s_0 is the start state q_0
2. $\delta(s_i, w_{i+1}) = s_{i+1}$ (a valid transition is chosen for the currently parsed input symbol)
3. $s_n \in F$, i.e., is an accept state.

Example:



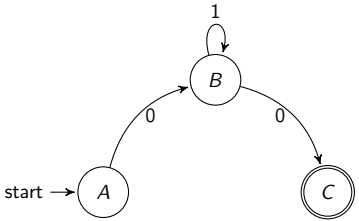
For $w = w_1 w_2 w_3 w_4 w_5 = 01110$,
$s = s_0 s_1 s_2 s_3 s_4 s_5 = ABBBBC$

Regular Languages

Given an alphabet Σ a language L is a set of words $w = w_1 \cdots w_n$ such that each $w_i \in \Sigma$.

Definition

A language L is a *regular language* if there exists a DFA M that accepts each word in L, i.e., $L = \{w \mid M \text{ accepts } w\}$.



describes

$$L_1 = \{w \mid w \text{ starts and ends with 0, with only 1's in between}\}.$$

$\rightarrow L_1$ is regular

Regular Languages

Since languages are sets, we can apply various operations on them:

- Union: the union of two languages L_1 and L_2 is $L_1 \cup L_2 = \{w \mid w \in L_1 \text{ or } w \in L_2\}$
- Intersection: similarly, $L_1 \cap L_2 = \{w \mid w \in L_1 \text{ and } w \in L_2\}$.
- Concatenation: $L_1 L_2 = \{w \mid w = w_1 w_2, w_1 \in L_1, w_2 \in L_2\}$
- Kleene star: $L_1^* = \{x_1 x_2 \cdots x_k \mid k \geq 0, \text{ each } x_i \in L_1\}$

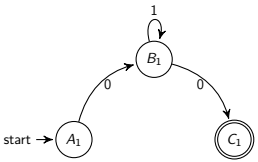
Theorem

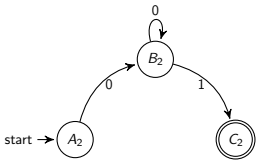
The class of regular languages is closed under union [intersection], i.e., the union [intersection] of two regular languages is regular.

Proof idea: We multitask! Construct "product" automaton that runs both DFA's in parallel: $(Q_1 \times Q_2, \Sigma, \delta, F)$ where

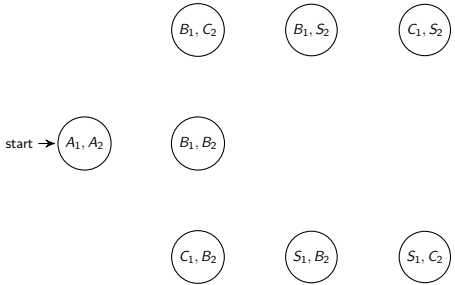
- $\delta((s_1, s_2), w_i) := (\delta_1(s_1, w_i), \delta_2(s_2, w_i))$
- $F = \{(s_1, s_2) \mid s_1 \text{ or } s_2 \text{ is an accepting state}\}$ for union,
- $F = \{(s_1, s_2) \mid s_1 \text{ and } s_2 \text{ is an accepting state}\}$ for intersection

Union of Regular Languages: Example

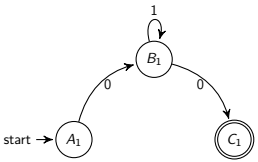


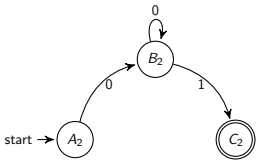


with implicit sink states S_1, S_2.

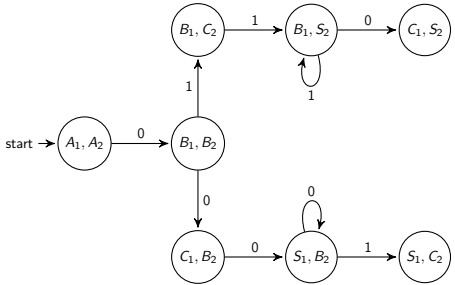


Union of Regular Languages: Example

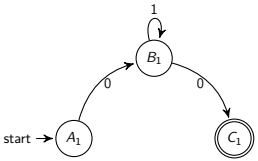


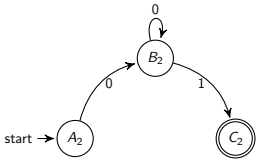


with implicit sink states S_1, S_2.

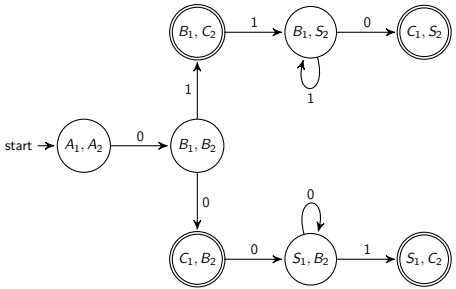


Union of Regular Languages: Example





with implicit sink states S_1, S_2.



Nondeterministic Finite Automata

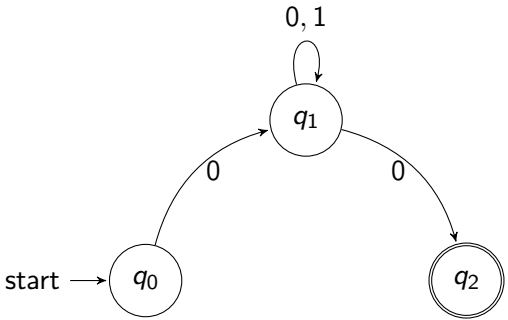
For DFA, the transition function δ gave,
for a given state and input symbol,
precisely one following state.

$\rightarrow$ **determinism**

Now we allow for multiple possible "next" states.

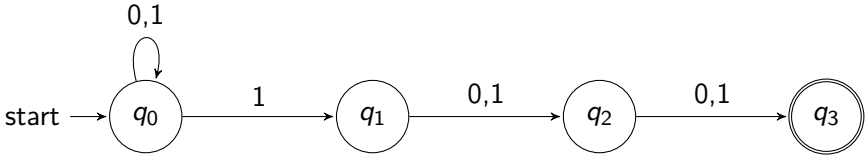
$\rightarrow$ **nondeterminism**

NFA - An Example



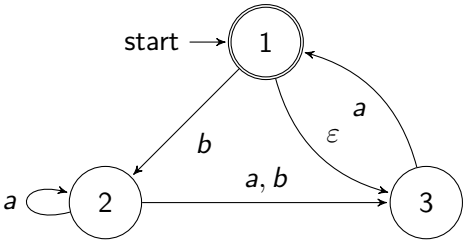
This language consists of all 0,1 sequences starting and ending with 0.

NFA - Another Example



This language consists of all 0,1 sequences with a 1 in the third position from the end

NFA - an Example with Empty Transitions



Nondeterministic Finite Automata

Definition

A nondeterministic finite automaton (NFA) 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. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function, and
4. $F \subseteq Q$ is the set of accepting states.

Acceptance for Nondeterministic Finite Automata

- What is the acceptance condition for NFA?
- Remember acceptance for DFA: a sequence of states such that the transition function gives us the next state in the sequence
- For NFA, *one* of the next states in each transition must be the next state in the sequence

Definition (NFA)

An NFA $(Q, \Sigma, \delta, q_0, F)$ *accepts* an input $w = w_1 w_2 \cdots w_n$ if there exists a sequence of states $s_0 \cdots s_n$ such that

1. s_0 is the start state q_0
2. $s_{i+1} \in \delta(s_i, w_{i+1})$ (a valid next state is chosen for the currently parsed input symbol)
3. $s_n \in F$, i.e., is an accept state.

NFA

- Notice that DFAs are special cases of NFAs.
- DFAs accept regular languages, but what languages do NFAs accept?
- As it turns out: regular languages!
- In other words, in a sense, DFA=NFA.

DFA=NFA

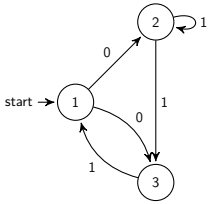
Theorem

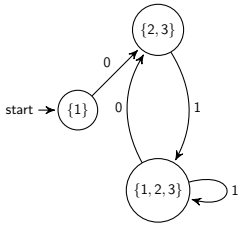
Every NFA $N = (Q, \Sigma, \delta, q_0, F)$ has an equivalent DFA $M = (Q', \Sigma, \delta', q_0', F')$.

Proof idea:

- Each state in the NFA can have multiple possible following states.
- We need to simultaneously keep track of all these possible following states in *one* state in the DFA.
- Since the "set of possible following states" in the NFA could be any subset of the state set Q, the DFA's state set Q' must be $\mathcal{P}(Q)$.
- Let us first assume there are no ε transitions.

DFA=NFA: Example





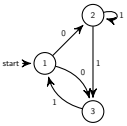
- Let us construct the DFA.
- Clearly $q_0' = \{q_0\}$.
- Skal det være noen transisjoner fra tilstanden $\{1\}$?
- Skal det være noen transisjoner fra tilstanden $\{1, 2\}$?
- Skal det være noen transisjoner fra tilstanden $\{1, 2, 3\}$?

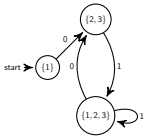
NFA=DFA

What about the transition function δ'?

- A state R in the DFA M corresponds to a set of states in the NFA N.
- For an input w_i at state R, we need to consider *all* possible following states from the set of possible states R. Or, more formally,

$$\begin{aligned}\delta'(R, w_i) &= \bigcup_{r \in R} \delta(r, w_i) \\ &= \{q \in Q \mid q \in \delta(r, w_i) \text{ for some } r \in R\}\end{aligned}$$





NFA=DFA

- Since an NFA accepts an input if *any* of the possible computations ends in an accept state, $F' = \{R \subseteq Q \mid R \text{ contains a state } r \in F\}$.

- **Almost done!** But we still need to take ε transitions into account. To that end, let

$$E(R) = \{q \mid q \text{ can be reached from } R \text{ with 0 or more } \varepsilon \text{ transitions}\} \text{ for } R \subseteq Q.$$

- Then $q_0' = E(\{q_0\})$.
- Transition function δ':

$$\begin{aligned}\delta'(R, w_i) &= \bigcup_{r \in R} E(\delta(r, w_i)) \\ &= \{q \in Q \mid q \in E(\delta(r, w_i)) \text{ for some } r \in R\}\end{aligned}$$

NFA=DFA

In other words, we have just proven:

Theorem

A language is regular iff (if and only if) there exists an NFA that accepts it.

Concatenation of Regular Languages

- Remember from previous lecture we proved that RL is closed under union and intersection?
- Proof method from previous lecture:
  we could construct DFA for $L_1 \cup L_2$ and $L_1 \cap L_2$ from DFAs for L_1 and L_2
- What about concatenation of languages?

- Given two regular languages L_1, L_2, is

$$L_1 L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}$$

  a regular language?
- Can we use a similar proof method for concatenation (and Kleene star)?
- We can do this by constructing NFAs!

Concatenation of Regular Languages

- **Idea:** An automaton gets an input w and must "guess" how to split it $w = w_1 w_2$ in order to check L_1 and L_2 membership.

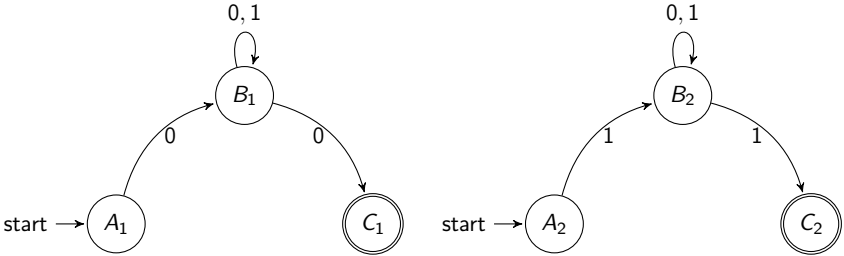
- **Example:**
  $L_1 = \{w \mid w \text{ begins and ends with } 1\}$ and $L_2 = \{w \mid w \text{ begins and ends with } 0\}$

  How to split the word $w = 10101010$? Multiple options

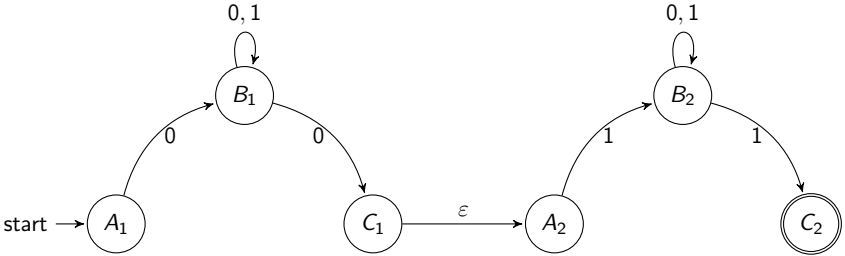
- This "guessing" is done by using nondeterminism:
  NFAs try all computational branches and accept if one leads to an accepting state.

- For concatenation: after M_1 has parsed a part of the input that led to an accept state, nondeterministically "jump" to the second automaton M_2.

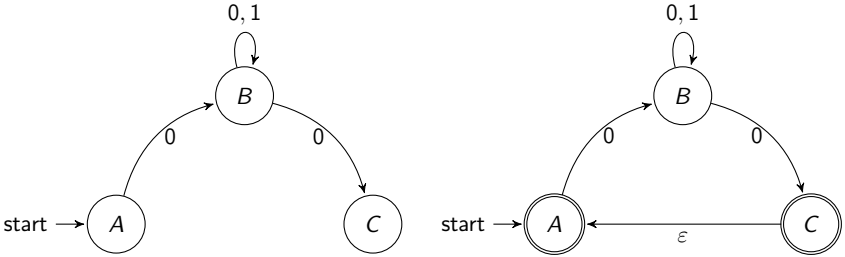
Concatenation of Regular Languages



Concatenation of Regular Languages



Kleene Star



- Let L be the language recognized by this automaton
- How can we construct an automaton that recognizes L^*?

Summary

- **Basics**: sets, de Morgan's laws, tuples, products, functions, strings
- **Deterministic finite automata (DFA)**: states, transitions, start state, accept states
- Formal definitions of DFA, acceptance, regular language (RL)
- Operations on languages: union, intersection, concatenation, Kleene star
- Theorem: RL closed under union and intersection
- **Nondeterministic finite automata (NFA)**
- NFAs recognize regular languages (just like DFAs)
- Theorem: Every NFA has an equivalent DFA
- NFAs are convenient for composing languages (concatenation, repetition)
- Theorem: RL closed under concatenation and Kleene star