Recap: Finite State Automaton

\[ A = (\Sigma, Q, q_0, \delta, F) \]

- \( \Sigma \) \hspace{0.5cm} \text{alphabet}
- \( Q \) \hspace{0.5cm} \text{states (nodes in the graph)}
- \( q_0 \in Q \) \hspace{0.5cm} \text{initial state (with } \rightarrow \text{ sign in drawing)}
- \( \delta \subseteq Q \times \Sigma \times Q \) \hspace{0.5cm} \text{transitions (labeled edges in the graph)}
- \( F \subseteq Q \) \hspace{0.5cm} \text{final states (double circles)}

\[
\delta = \{(q_0, a, q_0), (q_0, b, q_1), (q_1, a, q_1), (q_1, b, q_1)\}
\]
Question

- Design an automaton that recognizes strings over $\Sigma = \{0, 1\}$ that do not contain the substring $010$
Question

- Design an automaton that recognizes strings over $\Sigma = \{0, 1\}$ that do not contain the substring $010$

Answer

![Diagram of an automaton](image)
Question

- Design an automaton that recognizes strings over $\Sigma = \{0, 1\}$ that end in the substring 01
Question

- Design an automaton that recognizes strings over $\Sigma = \{0, 1\}$ that end in the substring 01

Answer

Can you design an automaton for this language with only 3 states?
Question

- Design an automaton that recognizes all numbers written in binary that are divisible by 2. For example, the automaton should accept the words 0, 10, 100, 110, ⋯ (leading zeros are ok)
Question

- Design an automaton that recognizes all numbers written in binary that are divisible by 2. For example, the automaton should accept the words 0, 10, 100, 110, ⋅⋅⋅ (leading zeros are ok)

Answer
Types of Finite State Automata

- **Deterministic Finite Automata (DFA)**
  - $\delta$ is a function $(Q, \Sigma) \mapsto Q$
  - One transition per input per state
  - All examples so far

- **Nondeterministic Finite Automata (NFA)**
  - $\delta$ is a function $(Q, \Sigma) \mapsto 2^Q$
  - Can have multiple transitions for one input in a given state
Computation of a DFA

- For each input string there is exactly one path in a DFA.

![Diagram of a DFA showing states 0 and 1 with transitions for 0 and 1 inputs.]

<table>
<thead>
<tr>
<th>Path</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Path</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Computations of an NFA

For an input string there are multiple possible computation paths in an NFA.

Word is accepted if there is a path in the computation tree that leads to an accepting state.
Undefined transitions go to a trap state where no input can be accepted
Epsilon transition allows an NFA to change its state spontaneously without consuming any symbol from input

Example
NFA that accepts all strings of the form $0^k$ where $k$ is a multiple of 2 or 3
DFA vs. NFA

**DFA:**

```
0 1
2
0 1
0 1
0 1
```

- NFA for a language can be smaller and easier to construct than DFA
- An implementation of an NFA normally has backtracking
- An implementation of a DFA normally requires only as many steps as the input length

**NFA:**

```
0, 1
0, 1
```

```
0, 1
1, 0, 1
2
```
Exercise

Question

- Construct an NFA that recognizes all strings over $\Sigma = \{a, b, c\}$ that do not contain all the alphabet symbols $a$, $b$ and $c$. 

Answer

Let's start with a regular expression $(a | b)^* | (b | c)^* | (a | c)^*$. 

\[ a,b \in \epsilon \]

\[ a,c \in \epsilon \]

\[ c,b \in \epsilon \]
Exercise

Question

• Construct an NFA that recognizes all strings over $\Sigma = \{a, b, c\}$ that do not contain all the alphabet symbols $a$, $b$ and $c$.

Answer

• Let’s start with a regular expression

$$(a|b)^* \mid (b|c)^* \mid (a|c)^*$$
From NFA to DFA

- For every NFA there exists an equivalent DFA that accepts the same set of strings
- NFAs could be exponentially smaller (succinct)
- Idea: keep track of a set of all possible states in which the automaton could be
- View this finite set as one state of new automaton
From NFA to DFA: Example

When processing if we see a set exactly the same as a set constructed earlier we mark it.
From NFA to DFA: Example

When processing if we see a set exactly the same as a set constructed earlier we mark it.

\[ \{q_0\} \]
From NFA to DFA: Example

When processing if we see a set exactly the same as a set constructed earlier we mark it. For example:

- \( \{ q_0 \} \) is mapped to \( \{ q_0, q_1 \} \) on input 'a'.
- \( \{ q_0, q_1 \} \) is mapped to \( \times \) on input 'b'.

This process continues until all states are accounted for.
From NFA to DFA: Example

- When processing if we see a set exactly the same as a set constructed earlier we mark it.

\[
\begin{align*}
\{q_0\} & \xrightarrow{a} \{q_0, q_1\} \\
\{q_0\} & \xrightarrow{b} \{q_1\} \\
\{q_0, q_1\} & \xrightarrow{b} \{q_1, q_4, q_2\}
\end{align*}
\]
From NFA to DFA: Example

When processing if we see a set exactly the same as a set constructed earlier we mark it.

\[ \{q_0\} \xrightarrow{a} \{q_0, q_1\} \xrightarrow{a} \{q_0, q_1, q_2\} \]

\[ \{q_0\} \downarrow b \times \{q_1, q_4, q_2\} \]

\[ \{q_0, q_1\} \downarrow b \times \{q_1, q_4, q_2\} \]
When processing if we see a set exactly the same as a set constructed earlier we mark it.
• When processing if we see a set exactly the same as a set constructed earlier we mark it
From NFA to DFA: Example

- When processing if we see a set exactly the same as a set constructed earlier we mark it.
From NFA to DFA: Example

- When processing if we see a set exactly the same as a set constructed earlier we mark it
From NFA to DFA: Example

- When processing if we see a set exactly the same as a set constructed earlier we mark it.
Exercise

Question

• Construct an NFA for the regular expression \((bb^*)|a^*\) over alphabet \(\{a, b\}\) and determinize it.