Lecture 3 Overview

- Things to remember
- Nondeterministic Finite Automaton (NFA): definition, examples
- Equivalence of DFAs and NFAs
- Introduction to regular operations and closure properties of regular languages

Things to remember

- $\emptyset$ is the empty set, i.e., the set that contains 0 elements: $\emptyset = \{\}$
- $\varepsilon$ (“epsilon”) is the empty string, i.e., the string that has 0 symbols (and hence length 0).
- We will not use $\emptyset$ or $\varepsilon$ as alphabet symbols. These are distinguished symbols that have the meanings indicated above.
- $\{\varepsilon\}$ is not an alphabet! It is a language that consists of only the empty string.
- $\emptyset$ is a language too. The language that contains no strings at all.
Nondeterminism

Nondeterminism allows several possible next states at every step.

Determinism: 

\[ \text{one next state per } s \]

Nondeterminism: 

\[ \text{set of next states per } s \]

State diagrams – NFA versus DFA

<table>
<thead>
<tr>
<th>NFA</th>
<th>DFA</th>
</tr>
</thead>
<tbody>
<tr>
<td>• a state may have 0, 1, or many exiting transitions for each symbol in the alphabet</td>
<td>• every state has exactly one exiting transition for each symbol in the alphabet</td>
</tr>
<tr>
<td>• labels on the transition arrows are symbols from the alphabet or the distinguished symbol ( \varepsilon )</td>
<td>• labels on the transition arrows are symbols from the alphabet</td>
</tr>
</tbody>
</table>
Formal Definition of NFA

**Def:** A nondeterministic finite automaton (or NFA) is a 5-tuple \( M = (Q, \Sigma, \delta, q_0, F) \), where

1. \( Q \) is a finite set of states
2. \( \Sigma \) is an alphabet
3. \( \delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow \mathcal{P}(Q) \) is the transition function
4. \( q_0 \in Q \) is the start state
5. \( F \subseteq Q \) is the set of accept (final) states.

\( \mathcal{P}(Q) \) is the set of all subsets of \( Q \).

Computation for NFAs

An NFA \( M \) accepts a string \( w \in \Sigma^* \) if there is some path to follow on \( w \) that ends in a final (i.e., accept) state.

\( M \) rejects \( w \) if all paths on \( w \) end in non-accepting states.

Formally, ...
Formal Definition of Computation for NFAs

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA and $w$ a string over $\Sigma$. $M$ accepts $w$ if there is a sequence of symbols $w_1, w_2, \ldots, w_n \in \Sigma \cup \{\varepsilon\}$, where $w = w_1w_2 \cdots w_n$, and a sequence of states $r_0, r_1, \ldots, r_n \in Q$ such that

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

Formal Definition of Computation for NFAs

Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA. $M$ recognizes a language $L$ if $L = \{w \in \Sigma^* \mid M \text{ accepts } w\}$, i.e.,

For all $w \in \Sigma^*$:

- $w \in L \Rightarrow M \text{ accepts } w$
- $w \notin L \Rightarrow M \text{ rejects } w$

Notation: $L(M)$ denotes the language recognized by $M$. 
NFA example (without $\varepsilon$-transitions)

\[ q_1 \xrightarrow{\varepsilon} q_2 \]

Without reading the next input symbol, the machine splits into two copies of itself.
One of them follows the $\varepsilon$-transition.
The other stays at $q_1$.
In parallel, each copy continues its nondeterministic computation.
NFA example (with $\varepsilon$-transitions)

\[\begin{array}{c}
\circlearrowleft 0, 1 \\
\rightarrow 1 \\
\rightarrow 0, 1 \\
\rightarrow 0, 1, \varepsilon \\
\end{array}\]
NFA example – formal description

\[ M: \]

- \[ 0 \to 1 \to 0 \]
- \[ q_0, q_1, q_2, q_3 \]
- \[ \{0, 1\} \]
- \[ \delta(q_0, \epsilon) = \emptyset \]

\[ M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \delta, q_0, \{q_3\}), \text{ where} \]

\[ \delta(q_0, 0) = \{q_0, q_1\} \]
\[ \delta(q_0, 1) = \{q_0\} \]
\[ \delta(q_0, \epsilon) = \emptyset \]

---

Why NFAs?

- NFAs seem more powerful than DFAs. We will show that they are not.
- Building NFAs is often easier than building DFAs
- NFAs may be more compact than DFAs that recognize the same language.

Is every DFA an NFA?
Equivalence of NFAs and DFAs

Two machines \( M_1, M_2 \) are equivalent if \( L(M_1) = L(M_2) \).

DFAs and NFAs recognize the same class of languages: regular languages.
To show this, we will prove the following.

Equivalence of NFAs and DFAs

**Theorem:** A language \( L \) is regular if and only if there exists an NFA that recognizes \( L \).

**Proof:** We must prove two statements:
1) If \( L \) is a regular language then there exists an NFA that recognizes it.
2) If there exists an NFA that recognizes a language \( L \), then \( L \) is regular.
Equivalence of NFAs and DFAs

1) If $L$ is a regular language then there exists an NFA that recognizes it.

Proof of 1):
Given: $L$ is regular, i.e., there exists a DFA $M$ that recognizes $L$.
Want: An NFA $N$ that recognizes $L$.
Construction: Let $N = M$. Since any DFA is an NFA, $N$ is an NFA that recognizes $L$.

Equivalence of NFAs and DFAs

2) If there exists an NFA that recognizes a language $L$, then $L$ is regular.

Proof of 2):
Given: There exists an NFA $N$ that recognizes $L$.
Want: $L$ is regular, i.e., there exists a DFA $M$ that recognizes $L$. We will build $M$.
Construction: