Automata Theory and Formal Grammars: Lecture 3

Regular Expressions and Languages
Regular Expressions and Languages

Last Time

- Deterministic Finite Automata (DFAs) and their Languages
- Closure Properties of DFA Languages (the product construction)
- Nondeterministic Finite Automata (NFAs) and their Languages
- Relating DFAs and NFAs (the subset construction)

Today

- Regular Expressions and Regular Languages
- Properties of Regular Languages
- Relating NFAs and regular expressions: Kleene’s Theorem
NFAs: Finishing Up
Sipser uses a more general definition than I gave last week:

**Definition** A nondeterministic finite automaton with empty transitions (NFA$\varepsilon$) is a quintuple $\langle Q, \Sigma, q_0, \delta, A \rangle$ where:

- $Q$ is a finite set of states;
- $\Sigma$ is the input alphabet;
- $q_0 \in Q$ is the start state;
- $A \subseteq Q$ is the set of accepting states; and
- $\delta : Q \times \Sigma \cup \{\varepsilon\} \to 2^Q$ is the transition function.
Theorem: The set of NFA languages is identical to the set of NFA\(\varepsilon\) languages.

Proof?

One direction is trivial: An NFA (without empty transitions) is an NFA\(\varepsilon\) where for all \(q\):

\[\delta(q, \varepsilon) = \emptyset\]
The Subset Construction for NFA$\varepsilon$

Let $N = \langle Q, \Sigma, q_0, \delta, A \rangle$ be a NFA$\varepsilon$.
We want to construct a DFA $D(N)$ accepting the same language.
States in $D(N)$ will be sets of states from $N$.
Let $P$ range over states of $D(N)$.
$P \in 2^Q$, that is, $P \subseteq Q$.

$$D(N) = \langle 2^Q, \Sigma, \delta(q_0, \varepsilon), \delta_D N, A_D N \rangle$$

where

$$\delta_D N (P, a) = \bigcup_{q \in P} \delta^*(q, a)$$

$$A_D N = \{ P \mid P \in 2^Q \text{ and } P \cap A \neq \emptyset \}$$
Consider the NFA $M$ given by $K = \{q_0, q_1, q_2\}$, $\Sigma = \{0, 1, 2\}$, $s = q_0$, $F = \{q_2\}$ with transition relation $\Delta$ given below:

<table>
<thead>
<tr>
<th>q</th>
<th>$\sigma$</th>
<th>$\Delta(q, \sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>0</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_0$</td>
<td>$\varepsilon$</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>1</td>
<td>$q_1$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$\varepsilon$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>2</td>
<td>$q_2$</td>
</tr>
</tbody>
</table>

$L(M) = \{0\}^* \{1\}^* \{2\}^*$. 
The resulting DFA $M'$ has $K' = \{\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}, \emptyset\}$, $s' = \{q_0, q_1, q_2\}$, $F = \{\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}\}$ and $\delta'$:

<table>
<thead>
<tr>
<th>q</th>
<th>$\sigma$</th>
<th>$\delta'(q,\sigma)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0, q_1, q_2}$</td>
<td>0</td>
<td>${q_0, q_1, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_1, q_2}$</td>
<td>1</td>
<td>${q_1, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_1, q_2}$</td>
<td>2</td>
<td>${q_2}$</td>
</tr>
<tr>
<td>${q_1, q_2}$</td>
<td>0,1</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>${q_1, q_2}$</td>
<td>1</td>
<td>${q_1, q_2}$</td>
</tr>
<tr>
<td>${q_1, q_2}$</td>
<td>2</td>
<td>${q_2}$</td>
</tr>
<tr>
<td>${q_2}$</td>
<td>0,1,2</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\emptyset$</td>
<td>0,1,2</td>
<td>$\emptyset$</td>
</tr>
</tbody>
</table>
Another example

Let $\Sigma = \{a_1, ..., a_n\}$ where $n \geq 2$.

Let $L = \{w \mid \exists i. a_i \text{ does not appear in } w\}$.

For example, if $\Sigma = \{a_1, a_2, a_3\}$ then $a_1a_1a_2 \in \Sigma$ but $a_1a_2a_3 \notin \Sigma$.

Intuitively, the NFA would work in the following manner:

- Guess the symbol $a_i$ that is missing from the input.
- If no symbol is missing, move to a dead state.
- If a symbol $a_i$ is missing, go to state $q_i$.
- If in state $q_i$ you ever encounter $a_i$, move to a dead state.
- Otherwise eat the remaining symbols and accept.
Another Example (continued)

For the construction of the NFA we need one starting state $q_0$ and one state for each symbol in the alphabet, $q_1, \ldots, q_n$.

There are $\varepsilon$-transitions from $q_0$ into each of $q_1, \ldots, q_n$, and self-loops on each of $q_1, \ldots, q_n$ labeled with the states that are legal.

What happens when we use the construction to produce a DFA accepting this language?

The equivalent DFA $M'$ has initial state $s' = \{q_0, q_1, q_2, q_3, \ldots, q_n\}$. 
Regular Languages

This course: a study of the computing power needed to “process” different kinds of languages.

The first class of languages we will study: regular languages.

Regular languages are defined using regular expressions.
Regular Expressions

... a notation for defining languages.

**Definition**  Let \( \Sigma \) be an alphabet. Then the set \( \mathcal{R}(\Sigma) \) of *regular expressions* over \( \Sigma \) is defined recursively as follows.

\[
\begin{align*}
\emptyset & \in \mathcal{R}(\Sigma) \\
\varepsilon & \in \mathcal{R}(\Sigma) \\
a & \in \mathcal{R}(\Sigma) \quad \text{if} \ a \in \Sigma \\
\mathcal{R} + \mathcal{S} & \in \mathcal{R}(\Sigma) \quad \text{if} \ \mathcal{R} \in \mathcal{R}(\Sigma) \ \text{and} \ \mathcal{S} \in \mathcal{R}(\Sigma) \\
\mathcal{R} \circ \mathcal{S} & \in \mathcal{R}(\Sigma) \quad \text{if} \ \mathcal{R} \in \mathcal{R}(\Sigma) \ \text{and} \ \mathcal{S} \in \mathcal{R}(\Sigma) \\
\mathcal{R}^* & \in \mathcal{R}(\Sigma) \quad \text{if} \ \mathcal{R} \in \mathcal{R}(\Sigma)
\end{align*}
\]
Comments about Regular Expressions

The previous definition just gives the syntax of regular expressions: \(\circ, \cup, \ast\) are symbols that we will shortly give an interpretation to.

**Examples**  Let \(\Sigma = \{a, b\}\). The following are regular expressions in \(\mathcal{R}(\Sigma)\).

- \(a\)
- \((a + (b \circ b))\ast\)
- \(((b\ast) \circ ((a \circ a) + b)) \circ \emptyset\)

**Notation**

Usually, \(\circ\) will be omitted.

Also, to reduce parentheses, we will adopt the following precedence:

\(\ast > \circ > \cup\).

So \(((b\ast) \circ ((a \circ a) + b)) \circ \emptyset\) can be written as \(b\ast(aa + b)\emptyset\).
Derived Operations

We will sometimes use the following derived operations on regular expressions.

\[ r^+ = r \circ (r^*) \]

\[ r^i = \begin{cases} \varepsilon & \text{if } i = 0 \\ r \circ (r^{i-1}) & \text{otherwise} \end{cases} \]

E.g. \((b + a)^2 = (b + a) \circ (b + a) \circ \varepsilon\)
To make connection with languages precise, we need to define a **semantics** for regular expressions saying what they “mean”.

- Semantics will be given in form of function $\mathcal{L} : \mathcal{R}(\Sigma) \rightarrow 2^{\Sigma^*}$.

- For any regular expression $r$, $\mathcal{L}(r) \subseteq \Sigma^*$ will be the language defined by $r$. 
The Semantics of Regular Expressions

**Definition** Fix alphabet $\Sigma$. Then $L : \mathcal{R}(\Sigma) \to 2^{\Sigma^*}$ is defined as follows.

\[
L(r) = \begin{cases} 
\emptyset & \text{if } r = \emptyset \\
\{\varepsilon\} & \text{if } r = \varepsilon \\
\{a\} & \text{if } r = a \text{ and } a \in \Sigma \\
L(s_1) \cup L(s_2) & \text{if } r = s_1 + s_2 \\
L(s_1) \circ L(s_2) & \text{if } r = s_1 \circ s_2 \\
(L(s))^* & \text{if } r = s^*
\end{cases}
\]

**Definition** $L \subseteq \Sigma^*$ is a regular language if there is a regular expression $r$ such that $L = L(r)$.

(Note: This is a denotational semantics.)