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

NFAε

Sipser uses a more general definition than I gave last week:

$\boxed{\text{Definition}}$ A nondeterministic finite automaton with empty transitions (NFAε) is a quintuple $\langle Q, \Sigma, q_0, \delta, A \rangle$ where:

- Q is a finite set of states;
- Σ is the input alpabet;
- $q_0 \in Q$ 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.

Relating NFA and NFAε

$\boxed{\text{Theorem}}$ The set of NFA languages is identical to the set of NFAε languages.

Proof?

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

$$\delta(q, \varepsilon) = \emptyset$$

The Subset Construction for NFAε

Let $N = \langle Q, \Sigma, q_0, \delta, A \rangle$ be a NFAε.

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, $P \subseteq Q$.

$$D(N) = \langle 2^Q \, , \, \Sigma \, , \, \delta(q_0, \varepsilon) \, , \, \delta_{DN} \, , \, A_{DN} \rangle$$

where

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

$$A_{DN} = \{ \, P \mid P \in 2^Q \text{ and } P \cap A \neq \emptyset \, \}$$

Example

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 Δ given below:

q	σ	Δ(q,σ)
q_0	0	q_0
q_0	ε	q_1
q_1	1	q_1
q_1	ε	q_2
q_2	2	q_2

$\mathcal{L}(M) = \{0\}^*\{1\}^*\{2\}^*$.

Example continued

The resulting DFA M' has $K' = \{\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}, \emptyset\}$, $\underline{s}' = \{q_0, q_1, q_2\}$, $F = \{\{q_0, q_1, q_2\}, \{q_1, q_2\}, \{q_2\}\}$ and δ':

q	σ	δ' (q,σ)
$\{ q_0, q_1, q_2 \}$	0	$\{ q_0, q_1, q_2 \}$
$\{ q_0, q_1, q_2 \}$	1	$\{ q_1, q_2 \}$
$\{ q_0, q_1, q_2 \}$	2	$\{ q_2 \}$
$\{ q_1, q_2 \}$	0	$\emptyset$
$\{ q_1, q_2 \}$	1	$\{ q_1, q_2 \}$
$\{ q_1, q_2 \}$	2	$\{ q_2 \}$
$\{ q_2 \}$	0,1	$\emptyset$
$\{ q_2 \}$	2	$\{ q_2 \}$
$\emptyset$	0,1,2	$\emptyset$

Another example

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

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

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

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 ε-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 $\underline{s}' = \{q_0, q_1, q_2, q_3, ..., q_n\}$.

Regular Languages

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.

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

$$
\begin{aligned}
\underline{\emptyset} &\in \mathcal{R}(\Sigma) \\
\underline{\varepsilon} &\in \mathcal{R}(\Sigma) \\
\underline{a} &\in \mathcal{R}(\Sigma) \quad \text{if } a \in \Sigma \\
\underline{r+s} &\in \mathcal{R}(\Sigma) \quad \text{if } \underline{r} \in \mathcal{R}(\Sigma) \text{ and } \underline{s} \in \mathcal{R}(\Sigma) \\
\underline{r \circ s} &\in \mathcal{R}(\Sigma) \quad \text{if } \underline{r} \in \mathcal{R}(\Sigma) \text{ and } \underline{s} \in \mathcal{R}(\Sigma) \\
\underline{r*} &\in \mathcal{R}(\Sigma) \quad \text{if } \underline{r} \in \mathcal{R}(\Sigma)
\end{aligned}
$$

Comments about Regular Expressions

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

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

- a
- $(a + (b \circ b))*$
- $(((b*) \circ ((a \circ a) + b)) \circ \emptyset)$

$\boxed{\text{Notation}}$

Usually, $\circ$ will be omitted.

Also, to reduce parentheses, we will adopt the following precedence: $* > \circ > \cup$.

So $(((b*) \circ ((a \circ a) + b)) \circ \emptyset)$ can be written as $b*(aa + b)\emptyset$.

Derived Operations

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

$$\underline{r^+} = \underline{r \circ (r*)}$$

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

$\boxed{\text{E.g.}}$ $\underline{(b+a)^2} = \underline{(b+a) \circ (b+a) \circ \varepsilon}$

How Do Regular Expressions “Define” Languages?

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 $\mathcal{L} : \mathcal{R}(\Sigma) \to 2^{\Sigma^*}$.
- For any regular expression $\underline{r}$, $\mathcal{L}(\underline{r}) \subseteq \Sigma^*$ will be the language defined by $\underline{r}$.

The Semantics of Regular Expressions

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

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

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

(Note: This is a denotational semantics.)