Chapter 9

Lecture 17

Mayers

Finite Automata

A Finite Automaton

A DFA accepting all strings ending in 111

X, S, δ, S₀, A

Finite Automata
\[
\begin{align*}
\{ y \in \Sigma^* & \mid y \in \gamma \} = \gamma^* \cup \gamma^1 \cup \gamma \cap \gamma^\epsilon \\
\text{The union of two languages } \gamma \text{ and } \gamma^1 & : \\
\{ x \in \Sigma^* & \mid x \in \gamma \} \times \{ x \in \Sigma^* & \mid x \in \gamma^\epsilon \} = \gamma \times \gamma^\epsilon \\
\text{The product of two languages } \gamma \text{ and } \gamma^\epsilon & : \\
\text{The set of strings accepted by a } FA \text{ is called the language } (\gamma) &.
\end{align*}
\]

Example of an NFA (Non-deterministic Finite Automaton)

Procedure: \( DECIDEFA(\mathcal{S}, \mathcal{A}, \mathcal{F}) \)
Theorem 9.1.1: For every regular language, there is a DFA.

• Definition 9.1.4: A set of (or language).

The Star Operation

\[
\text{If we delete the empty string, we get } \epsilon^*.
\]

Every member of \( \mathcal{L} \) is finite, but \( \mathcal{L}^* \) is infinite.

- \( \mathcal{L} \cap \emptyset = \emptyset \)
- \( \mathcal{L} \cap \mathcal{L} = \mathcal{L} \)
- \( \mathcal{L} \cap \{ \epsilon \} = \{ \epsilon \} \)

Definition 9.1.8: For \( \mathcal{L} \subseteq X^* \), the result \( \mathcal{L}^* \) of the set.

Regular Expressions
A DFA for a Module 3 Counter and its Complement

A DFA accepting $(0 + 1)^* 1 (0 + 1)^* 0$

Binary Parse Tree for $(q, a)^*$

DFA Synthesis

1. Construct a binary parse tree $T(R)$ for the given regular expression $R$.
2. Construct from $T(R)$ an NFA $N(R)$ which accepts the language.
3. Construct from $N(R)$ a DFA $D(R)$ which accepts the same language.
Rule for Product of Two Regular Expressions

Rule for Union of Two Regular Expressions
Let $\mathbb{L}$ be a language.

- For each $q \in Q$, define $t(q) = (q, 0, 0, 1, 0, 0, 0, 0)$.
- Define $t(q_0) = (q_0, 0, 0, 0, 0, 0, 0, 0)$.
- Define $t(q_f) = (q_f, 0, 0, 0, 0, 0, 0, 0)$.

Then, for any $s \in \mathbb{L}$, we have $s = q_0 \cdot t(q_1) \cdot t(q_2) \cdot \ldots \cdot t(q_n) 
\cdot t(q_{n+1}) \cdot \ldots \cdot t(q_f)$.

**Procedure**: To determine whether $s \in \mathbb{L}$, we proceed as follows:

1. **Start**: Set $q = q_0$.
2. **Loop**: For each symbol $a$ in $s$,
   - If $t(q)(a) = (q', 0, 0, 0, 0, 0, 0, 0)$, then set $q = q'$.
   - Otherwise, return False.
3. **End**: If $q = q_f$, return True; otherwise, return False.

**Example**: For the DFA in the diagram, the language is determined by the path from $q_0$ to $q_f$.

**Language**: The language $\mathbb{L}$ is the set of all strings that start with a match and end with a mismatch.
An L-Automaton Representing a Type Containing an
Infinite Number of Each Substring

\[ C_f \text{ is a set of } f \text{-cycle states } (C_1, \ldots, C_t) \]
\[ \mathcal{A}_f \text{ is the set of receive edges} \]
\[ S_0 \subseteq S \text{ is the set of initial (reset) states} \]
\[ \delta : S \times X_0 \rightarrow 2^S \text{ is the next-state function} \]
\[ S \text{ is the (finite non-empty) set of states} \]
\[ X_0 \text{ is the input alphabet (a finite non-empty set of input values)} \]
\[ \langle X, S, \delta, \mathcal{A}_f, C_f \rangle \]

An L-Automaton (L-Automata)
1. For each cycle set $C$, compute the set of edges of $\mathcal{L}$ not contained in $\mathcal{R}$. 

2. Compute the transitive closure $\mathcal{L}'$ of $\mathcal{L}$. 

3. Remove the transitive edges and test if $\mathcal{R}$. 

Procedure: Transitive-containment($\mathcal{R}$, $\mathcal{L}'$). 

An Example of a Product Automaton

Illustration of Non-containment in Cycle Set