Notes for Friday, April 23rd

To keep in mind:

DFA:
\[ q_i \xrightarrow{a} q_j \]
maps a single state to a single state

NFA:
\[ q_i \xrightarrow{a} \{q_1, q_2\} \]
maps a single state to a set of states

NFA \rightarrow DFA:
\[ DFA = (Q, \Sigma, \delta, q_0, F) \]
\[ Q = \text{Pow}(Q') \]
Transition function seems like it’s going from a set of states, but it’s just notation. It may be useful to imagine quotes:
\[ \{q_1, q_2\} \rightarrow \{ \} \].

We have learned two ways to describe languages: DFAs and NFAs (and we have proved they are the same).

Last time, we proved languages are closed under \( \cup, \circ, * \).

Suppose we use \( \cup, \circ, * \) to describe languages. Some examples (we use \( \Sigma = \{0, 1\} = 0 \cup 1 \)):

1. \( 0 \cup 1 = \{0, 1\} \).
2. \( (0 \cup 1) \circ 0 = \{00, 10\} \)
3. \( (0 \cup 1)^* = \{0, 1\}^* \) (like \( \Sigma^* \))
4. \( (0 \cup 1)^* 0 = \{w \mid w \text{ ends in } 0\} \)
5. \( ((0 \cup 1)(0 \cup 1))^* = \{w \mid w \text{ is even}\} \)
6. \( \Sigma^* 1\Sigma = \{w \mid \text{second to last symbol of } w \text{ is } 1\} \)
7. \( \Sigma^* \emptyset = \{w \mid \exists x, y \text{ such that } w = xy \text{ and } x \in \Sigma^*, y \in \emptyset\} = \emptyset \)
8. \( A\emptyset = \emptyset \)
9. \( \emptyset^* = \{\varepsilon\} \) (because \( k \) can be 0 in the definition of * )
10. \( \varepsilon^* = \{\varepsilon\} \)

Sets of strings described by these operations are called Regular Expressions.

Definition: \( R \) is a regular expression IFF
\[ R \]
is a string over the alphabet \( \Sigma \cup \{(,), \varepsilon, \emptyset, \cup, \circ, *\} \) (we often omit \( \circ \) because we may write \( ab \) instead of \( a \circ b \))
AND
\[ R \]
is
1. \( a \in \Sigma \) OR
2. \( \varepsilon \) OR
3. \( \emptyset \) OR
4. \( R_1 \cup R_2 \), with \( R_1, R_2 \) regular expressions OR
5. \( R_1 R_2 \), with \( R_1, R_2 \) regular expressions OR
6. \( R_1^* \), with \( R_1 \) a regular expression
Parentheses are used for precedence. Without them, $\ast > \circ > \cup$.

The language of a regular expression, $L(R)$, is the set of strings defined by $R$.

Examples:

1. $L(R) = \{ w | w \text{ contains exactly 2 0's} \}$:
   
   $R = 1^*01^*01^*$

2. $L(R) = \{ w | w \text{ contains at least 2 0's} \}$:
   
   $R = \Sigma^*0\Sigma^*0\Sigma^*$

3. $L(R) = \{ w | w \text{ contains even number of 0's} \}$:
   
   $R = 1^*(1^*01^*01^*) \text{ or } 1^*(01^*01^*)$

4. $L(R) = \{ w | w \text{ does not contain 00} \}$

   Consider the “opposite”: $L(R') = \{ w | w \text{ contains 00} \}$:

   $R' = \Sigma^*00\Sigma^*$

   Ideally, we’d like $R = \Sigma^* - \Sigma^*00\Sigma^*$, but this is not allowed.

   It may help to make a DFA for $R$:

   ![DFA Diagram]

   What does not containing 00 mean?

   Answer: any 0 must be followed by 1, unless it is the final 0

   $(011^*)^* \text{ or } (011^*)^*0$

   we are still missing the $1^*$ case:

   $(1^*(011^*)^*) \cup (1^*(011^*)^*0)$

   or alternatively $1^*(011^*)^*(\varepsilon \cup 0)$

   The regular expression seems to capture the dynamics of the computation done by the DFA.

Question: are regular expressions and DFAs/NFAs equivalent?

Final example: $L(R) = \{ w | w \text{ is a valid identifier in C} \}$

$R = (A \cup B \cup \cdots \cup Z \cup a \cup b \cup \cdots \cup z \cup \_)(A \cup B \cup \cdots \cup Z \cup a \cup b \cup \cdots \cup z \cup \_ \cup 0 \cup 1 \cup \cdots \cup 9)^*$

Regular expressions are useful to describe the general rules.