Recap: regular languages
- DFAs
  - \( L \) is regular if \( L = L(M) \), \( M \) a DFA
- NFAs
  - Equivalence Theorem: \( \forall \text{NFAs} \ N, \exists \text{a DFA} \ M \ \text{s.t.} \ L(N) = L(M) \)
  - Closure Theorem

Today: regular expressions

Proof idea for equivalence theorem:

Given an NFA \( N = (Q_N, \Sigma, S_N, q_0, F_N) \), construct a DFA

\[ M = (Q_M, \Delta, \Sigma, S_m, q_m, F_m) \]

where

\[ Q_M = \{ S | S \subseteq Q_N \}, \]
\[ \delta: \text{for every } S \subseteq Q_M, a \in \Sigma, \]
\[ S(S, a) = T, T \subseteq Q_N \]

\( T \) is the set of states where \( N \) can take you starting at \( q \) and reading symbol \( a \).

(Actually, \( T \) is the set of states you can get to from \( S \) after empty transitions, then \( a \), then more empty transitions)

It can be helpful to eliminate empty transitions first.

Note: a NFA converted to a DFA has an exponential number of states.

Closure theorem: If \( A, B \) are regular languages, so are \( A \cup B, A \circ B, \) and \( A^* \).
Idea: If a string \( s \) is in \( A^* \), it accepts parts of the string, which are in \( A \), then empty transitions back to the start. If it accepts the string, the string can be broken into parts which are in \( A \).

(For homework: explain correctness, but don't go overboard.)

Regular expressions: Consider \( L = \{ w \mid w \text{ begins and ends with } 0 \} \).

A regular expression for \( L \) is \((0(01)*0) \cup 0\).

\( L = \{ w \mid w \text{ has an even number of } 0 \text{'s} \} \).

Regular expression: \((1^*01^*01^*)^* \cup 1^*\)

Def: \( R \) is a regular expression over \( \Sigma \) if it is one of:

1. \( R = a \) for \( a \in \Sigma \). \( L(R) = \{ a \} \).
2. \( R = \epsilon \) (empty string). \( L(R) = \{ \epsilon \} \).
3. \( R = \emptyset \) (empty set). \( L(R) = \emptyset \).
4. \( R = (R_1 \cup R_2) \) for shorter regexes \( R_1, R_2 \). \( L(R) = L(R_1) \cup L(R_2) \).
5. \( R = (R_1 \cdot R_2) \). \( L(R) = L(R_1) \cdot L(R_2) \).
6. \( R = (R_1)^* \). \( L(R) = L(R_1)^* \).

Back to examples: \((0(01)*0) \cup 0\)

Implied concatenation.

More examples:

1. \( \Sigma = \{a, n\} \). \( R = \text{ann} \). \( L(R) = \{ \text{ann} \} \).
2. \( R = (\text{ann})^* \). \( L(R) = \{ \epsilon, \text{ann}, \text{annann}, \ldots \} \).
3. \( R = \text{ann}^\ast = \text{ann}(a^\ast) \). \( L(R) = \{ \text{ann}, \text{annann}, \text{annannann}, \ldots \} \).

(Star operates first.)
4. \( R = \text{anna} \cup \text{may} = (\text{anna}) \cup (\text{may}) \)
   (Concatenation operator before union)

5. \((\text{anna} \cup \text{may})^* = (\text{anna} \cup \text{may})(\text{anna} \cup \text{may})^*\)
   (This is just a shorthand.)

6. \( R = \text{anna} \circ \emptyset = \emptyset \subseteq L(R) : L(\text{anna}) \circ L(\emptyset) \)

7. \( R = \text{anna} \circ \emptyset = \text{anna} \)
   but no such \( w_2 \) exists.

8. \( R \circ \emptyset^* = \emptyset \)
   \( L(\emptyset^*) = \{ w \mid w \text{ is a concatenation of } 0 \text{ or more strings in } \emptyset \} \)
   and \( \emptyset \) is the concatenation of 0 strings

* \( VR, RV, R \circ V, R, R \circ R = R \)
* \( R \circ \emptyset = R, R \cup \emptyset = \text{anna} \cup \text{may} \) or may not be \( R \).

Theorem (Equivalence for regexes): \( L \) is regular iff \( L = L(R) \)
for some regex \( R \).

Proof: \( \frac{}{1. \text{ If } L = L(R), \text{ then } L \text{ is regular.}} \)
\( \frac{}{2. \text{ If } L = L(R), \text{ then } L \text{ is regular.}} \)

Proof by induction on the length of the regex.

Base case: \( R \) of type 0, 1, and 3.

Type 0:
\[ \text{Type 0:} \]

Type 1:
\[ \text{Type 1:} \]

Type 3:
\[ \text{Type 3:} \]
Inductive step:
Type C1: \( R = R_1 \cup R_2 \). By the inductive hypothesis, \( L(C_1) \) and \( L(C_2) \) are regular, so \( L(C_1 \cup C_2) \) is regular by the closure theorem.

Type C2: Same, but with concatenation:
\( R = R_1 \cdot R_2 \). By ind. hypothesis, \( L(C_1) \) and \( L(C_2) \) are regular, so \( L(C_1 \cdot C_2) \) is regular.

Type C3: Same, but with star.

Q Generalize from an example: DFA recognizing strings in \( \{0, 1, 2\} \).

\[ M = \begin{array}{c}
q_0 \to 1 \quad 2 \\
1 \to 1 \\
0 \to 2
\end{array} \]

\[ L(M) = \{ w \in \{0, 1, 2\}^* \mid \sum w_i \equiv 0 \pmod{3} \} \]

Roadmap: Convert \( M \) into a GNFA (generalized NFA), where a GNFA is an automaton in which transitions are labelled with regexes, there is a unique \( q_{\text{start}} \), there is a unique \( q_{\text{accept}} \), and for any \( (q, r) \) where \( q \neq q_{\text{start}} \) and \( q \neq q_{\text{accept}} \), there is a transition from \( q \) to \( r \).
GNFA accepts if there is a computation history that reaches \( q_{\text{accept}} \) by consuming elements of regular expressions on transitions.

Transition function:

\[
\begin{array}{ccc}
\text{From} & \text{To} & \text{Symbol} \\
q_{\text{start}} & q_0 & \emptyset \\
q_{\text{start}} & q_1 & \emptyset \\
q_0 & q_0 & \emptyset \\
q_1 & q_1 & 1 \\
q_1 & q_2 & 2 \\
q_1 & q_{\text{accept}} & \emptyset \\
\end{array}
\]

Removing state \( q_{\text{trip}} \):

These GNFA would accept the same strings.

We need to update the transitions between every state after removing \( q_{\text{trip}} \).