Chapter 1 of [Sipser 1996], 3rd edition. Section 1.3.
Finite automata and regular expressions are equivalent

Theorem

A language is regular if and only if some regular expression describes it.

Proof ideas

1. If a language $A$ is described by a regular expression $R$ then $A$ is recognized by an NFA, therefore $A$ is regular

   There is an NFA $N$ such that $N$ recognizes $L(R)$

2. If a language $A$ is regular, it means that it is recognized by a DFA. Then we can always deduce a regular expression from it.

   Turn DFA into equivalent regular expression
By induction on the length of $R$:

- Base cases ($R$ has length 1):
  - $R = \{a\}$
  - $R = \epsilon$
  - $R = \emptyset$
Part 1: From regular expressions to NFAs

By induction on the length of $R$:

▷ Base cases ($R$ has length 1):
  ▶ $R = \{a\}$
  ▶ $R = \epsilon$
  ▶ $R = \emptyset$

▷ Inductive case: let $R$ have length $k > 1$. Assume that for any smaller regular expression, there is an NFA.

$R$ may be one of the following cases:
  ▶ $R = R_1 \cup R_2$
  ▶ $R = R_1 R_2$
  ▶ $R = (R_1)^*$
Part 2: From DFAs to regular expressions

1. Define Generalized Nondeterministic Finite Automaton (GNFA in short). Instead of \( \delta : Q \times \Sigma \rightarrow Q \), we use \( \delta : Q \times RE \rightarrow Q \)
   - Arrows labelled with regular expressions
   - Blocks of symbols instead of one symbol at a time
   - One start and one accept state

2. How to convert any DFA to an equivalent GNFA

3. Algorithm to convert any GNFA to an equivalent GNFA with 2 states

4. Convert a 2-state GNFA to an equivalent RE.
Step 1: DFA to GNFA

▷ Add unique and distinct start and accept states

▷ Edges with multiple labels become regexp labels

▷ If internal states \((q_1, q_2)\) don’t have an edge between them, add one labeled with \(\emptyset\)
  ▶ This should be such that \(q_{\text{start}}\) has no incoming edges and \(q_{\text{accept}}\) has no outgoing edges.
Step 2: Eliminate states from GNFA

While machine has more than 2 states:

- Pick and internal state, rip it out
- Re-label the arrows with regular expressions to account for the missing state
Step 2: Eliminate states from GNFA

While machine has more than 2 states:

▷ Pick and internal state, rip it out
▷ Re-label the arrows with regular expressions to account for the missing state
Step 2: Eliminate states from GNFA

While machine has more than 2 states:

▷ Pick and internal state, rip it out
▷ Re-label the arrows with regular expressions to account for the missing state

```
1
\[a\]  \[\text{start} \rightarrow \]
\[b\]  \[\rightarrow \]
\[2\]  \[a, b\]

s
\[\emptyset\]  \[\epsilon\]  \[\emptyset\]  \[\rightarrow \]
\[\emptyset\]  \[\epsilon\]  \[\rightarrow \]
\[a \cup b\]  \[\rightarrow \]

f
\[\rightarrow \]
\[a \cup b\]  \[\rightarrow \]
\[1\]  \[a\]  \[\rightarrow \]
\[\emptyset\]  \[\epsilon\]  \[\rightarrow \]
\[b\]  \[\rightarrow \]
```

```
s
\[\emptyset\]  \[\epsilon\]  \[\rightarrow \]
\[a \cup b\]  \[\rightarrow \]
\[2\]  \[a \cup b\]  \[\rightarrow \]
```

5 / 10
Step 2: Eliminate states from GNFA

While machine has more than 2 states:

- Pick and internal state, rip it out
- Re-label the arrows with regular expressions to account for the missing state
Step 2: Eliminate states from GNFA

While machine has more than 2 states:

- Pick and internal state, rip it out
- Re-label the arrows with regular expressions to account for the missing state
A GNFA is a tuple \((Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})\)

- \(Q\) is the set of states, \(\Sigma\) is the finite alphabet (not regexps)
- \(q_{\text{start}}\): initial state (unique, no incoming edges)
- \(q_{\text{accept}}\): accepting state (unique, no outgoing edges)
- \(\delta: (Q \setminus \{q_{\text{accept}}\}) \times (Q \setminus \{q_{\text{start}}\}) \to R\)
  - \(R\) is the set of all regexps over \(\Sigma\)

A GNFA accepts a string \(w \in \Sigma^*\) if \(w = w_1, \ldots, w_k\), with each \(w_i \in \Sigma^*\) and a sequence of states \(q_0, \ldots, q_k\) exists such that:

- \(q_0 = q_{\text{start}}\) is the start state
- \(q_k = q_{\text{accept}}\) is the accept state
- for each \(i\), we have \(w_i \in \mathcal{L}(R_i)\), where \(R_i = \delta(q_{i-1}, q_i)\), i.e. \(R_i\) is the expression on the arrow from \(q_{i-1}\) to \(q_i\)
CONVERT

Given a DFA $M$, let $G$ be its GNFA. CONVERT($G$) yields the equivalent regexp.

1. Let $k$ be the number of states of $G$

2. If $k = 2$, then $G$, return the regexp labeling its single transition

3. Select any state $q_{\text{rip}} \in Q \setminus \{q_{\text{start}}, q_{\text{accept}}\}$ and let $G'$ be the GNFA $(Q', \Sigma, \delta', q_{\text{start}}, q_{\text{accept}})$ such that

$$Q' = Q \setminus \{q_{\text{rip}}\}$$

and for any $q_i \in Q' \setminus \{q_{\text{accept}}\}$ and any $q_j \in Q' \setminus \{q_{\text{start}}\}$, let

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4)$$

for $R_1 = \delta(q_i, q_{\text{rip}})$, $R_2 = \delta(q_{\text{rip}}, q_{\text{rip}})$, $R_3 = \delta(q_{\text{rip}}, q_j)$ and $R_4 = \delta(q_i, q_j)$

4. Return CONVERT($G'$)
Theorem

Given any GNFA $G$, $\text{CONVERT}(G)$ is equivalent to $G$.

Proof idea

By induction on $k$, the number of states of $G$.

▷ Base step: $k = 2$

Show that the regexp labeling its single arrow describe all accepting strings of $G$

▷ Inductive step: assume it holds for $k - 1$. Show that $G$ and $G'$ are equivalent (i.e. accept the same words), then by the induction hypothesis so it will be for $\text{CONVERT}(G')$. 
Limits of finite automata

Are the following languages regular?

\[ L_1 = \{ w \mid w \text{ has equal number of 1s and 0s} \} \]

\[ L_2 = \{ w \mid w \text{ has equal number of occurrences of 01 and 10} \} \]