Definition: Languages generated by regular expressions are called regular languages.

Main Theorem: Regular languages are exactly the languages decided by DFAs (or NFAs, equivalently).

\[
\begin{array}{cc}
? & \rightarrow \text{E-NFA} \\
& \downarrow \\
\text{R.E.} & \rightarrow \text{NFA} \\
& \downarrow \\
? & \rightarrow \text{DFA} \\
\end{array}
\]

Theorem (R.E. \(\rightarrow\) E-NFA):

For any regular language \(L\), there exists an E-NFA \(N\) such that \(L = L(N)\).

Proof

Assume: \(L = L(r)\) for a regular expression \(r\)

Goal: Construct E-NFA \(N_r\) with \(L(N_r) = L(r)\).

Proof by structural induction, i.e., induction on \# of operators in \(r\).

Examples:
- \(r = (a+b)^*c\) \hspace{1cm} \#r = 3 = \#s + \#t + 1
- \(s = (a+b)^*\) \hspace{1cm} \#s = 2
- \(t = c\) \hspace{1cm} \#t = 0

Base case: \#r = 0 \(\Rightarrow r = \varepsilon, \emptyset\) or \(r = a\) for an \(a \in \Sigma\)

- \(N_\emptyset \rightarrow \circ\)
- \(N_\varepsilon \rightarrow \circ \xrightarrow{\varepsilon} \circ\)
- \(N_a \rightarrow \circ \xrightarrow{a} \circ\)

Induction step:

Assume that for all \(r\) with \#r \(\leq n\), there is an E-NFA \(N_r\) with \(L(N_r) = L(r)\).

Consider an \(r\) with \#r = n+1.

\[3 \text{ cases } \begin{cases} r = s + t \\ r = s \cdot t \\ r = s^* \end{cases}\]

in each case \(\#s, \#t \leq n\)
By the induction hypothesis, there exist E-NFAs $N_s, N_t$ with $L(N_s) = L(s), L(N_t) = L(t)$.

Furthermore, we may assume $N_s$ and $N_t$ each have exactly one accepting state! [Convenient for our pictures below]

Why? Otherwise, add a new accepting state, add E-transitions from all the other accepting states to it, and then make them non-accepting.

E.g. $\xrightarrow{\epsilon}$ $\circ \quad \Rightarrow \quad \circ \quad \circ$

$\xrightarrow{\epsilon} \quad \circ \quad \Rightarrow \quad \circ \quad \circ$

Case I: $r = s + t$

$\xrightarrow{\epsilon} \quad \circ \quad \Rightarrow \quad \circ \quad \circ$

$\xrightarrow{\epsilon} \quad \circ \quad \Rightarrow \quad \circ \quad \circ$

$L(N_r) = L(N_s) \cup L(N_t)$

$= L(s) \cup L(t)$

$= L(s + t) = L(r)$ ✓

Case II: $r = s \cdot t$

$\xrightarrow{\epsilon} \quad \circ \quad \Rightarrow \quad \circ \quad \circ$

$\xrightarrow{\epsilon} \quad \circ \quad \Rightarrow \quad \circ \quad \circ$

Case III: $r = s^*$

$\xrightarrow{\epsilon} \quad \circ \quad \Rightarrow \quad \circ \quad \circ$

$\xrightarrow{\epsilon} \quad \circ \quad \Rightarrow \quad \circ \quad \circ$

✓
Example  \( r = a^* + b, c \)  \((r = \varepsilon)\)

which simplifies to

\[
\quad \xrightarrow{\varepsilon, a, b, c}
\]

(\text{but not further!})


---

**Converting DFAs to Regular Expressions**  \(\text{ (see H\text{-}M\text{-}U 3.2.1-3.2.2)}\)

**Theorem**  
Let \( M \) be a DFA. 
Then \( L(M) \) is regular.

**Proof idea**  
Given \( M = (Q, \Sigma, \delta, q_0, F) \) 
Construct regexp \( r \) such that \( L(r) = L(M) \).

Two proofs in the text: (and they work directly on NFAs and \( \varepsilon \)-NFAs)

1. By adding states in one at a time [3.2.1]
2. By eliminating states from \( M \) one at a time \([3.2.2]\)

**State Elimination proof**

Idea: Start with an NFA \( N \). 
If \( N \) has \( \leq 2 \) states, it is easy!

eg. \( N = \xrightarrow{a} b \) \(\Rightarrow r_N = (c^*a d^* b)^* c^* a d^* \)

or \( r_N = (c + a d^* b)^* a d^* \)

Try to simplify to this case by eliminating states between the initial and final states.

Example \( \xrightarrow{a} b \) \(\Rightarrow \xrightarrow{a b} c \) \(\Rightarrow \xrightarrow{a b + c} \)

This creates generalized automata with edges labeled by regular expressions.
In general, it will look like...

\[ A_1 \xrightarrow{r_1} s \xrightarrow{t_1} C_1 \]
\[ A_2 \xrightarrow{r_2} B \xrightarrow{t_2} C_2 \]
\[ A_3 \xrightarrow{r_3} B \xrightarrow{t_3} \]

where \( r_1, r_2, \ldots, s, t_1, t_2, \ldots \) are all regexps.

It may be that some \( A_i \) and \( C_j \) states are the same!

Now to eliminate \( B \), add edges from each \( A_i \) to each \( C_j \) as:

\[ A_i \xrightarrow{r_i \cdot s^* \cdot t_j} C_j \]

Then simplify any possible multi-edges, e.g.,

\[ A \xrightarrow{r_1} C \xrightarrow{r_2 + r_3} C \]

Idea: At each step of simplification, the arrow from state \( q \) to state \( q' \) should be labeled by a regexp representing all possible ways of going from \( q \) to \( q' \) in the original NFA that only go through eliminated states.
Example:

- First, change it so there is only one accepting state

- Now eliminate \( q_2 \):

- Now eliminate \( q_3 \):

- With only two states remaining, we can read off the answer:

\[
r = (00 + (1+01)(01+11)^*(00+10))(o + (1+01)(01+11)^*(e+0+1))
\]

Note: The answer you get will depend on the order in which you eliminate states. In exercises, show your work!

It is a good idea to eliminate less-connected states first.

This is not yet a proof, but you should convince yourself that the idea works. A formal proof is in H-M-U 3.2.1.

\[ \Rightarrow \] DFAs and NFAs all accept, and regexps generate exactly the set of regular languages.
Examples:

- \( L = \{ \omega \in \{0,1\}^* \mid n_0(\omega) \text{ is odd} \} \)

\[
\begin{align*}
&1^* 0 1^* (0 1^* 0^*)^* 1^* & \times \\
&1^* 0 1^* (0 1^* 0^*)^* & \checkmark \\
&1^* 0 (1^* 0 1^* 0^*)^* & \times \\
&1^* 0 (1^* 0 1^* 0^*)^* & \times \\
&1^* 0 (1^* 0 1^* 0^*)^* & \checkmark \\
&1^* 0 (1^* 0 1^* 0^*)^* & \checkmark \\
&(1 + 01^* 0)^* 01^* & \checkmark
\end{align*}
\]

- \( L = \{ \omega \in \{0,1\}^* \mid \omega \text{ ends with 1 if does not contain } 0003 \}

\((1 + 01^*)(1 + 01)\)

- \( L = \{ \omega \in \{0,1\}^* \mid n_0(\omega), n_1(\omega) \text{ are both even} \}

\((00 + 11 + (01 + 10)(00 + 11)^*(01 + 10))^*\)

Look at a prefix and find the shortest one that balances the parities of \( n_0 \) and \( n_1 \).

Example: Convert to a regexp:

\[
\begin{array}{c}
\rightarrow \\
\end{array}
\]