Today we will learn...

- Converting REGEX to NFA
- Converting NFA to REGEX
Soundess

All Regexes have an equivalent NFA

REGEX → NFA
All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\text{NFA}(R)) = L_1$.

Given an alphabet $\Sigma$

- $\text{NFA}(a) =$
All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If \( L(R) = L_1 \), then \( L(\text{NFA}(R)) = L_1 \).

Given an alphabet \( \Sigma \)

- \( \text{NFA}(a) = \text{char}(a) \)
- \( \text{NFA}(\epsilon) = \)
All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If \( L(R) = L_1 \), then \( L(\text{NFA}(R)) = L_1 \).

Given an alphabet \( \Sigma \)

- \( \text{NFA}(a) = \text{char}(a) \)
- \( \text{NFA}(\epsilon) = \text{nil} \)
- \( \text{NFA}(\emptyset) = \)
All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(NFA(R)) = L_1$.

Given an alphabet $\Sigma$

- $NFA(a) = \text{char}(a)$
- $NFA(\epsilon) = \text{nil}$
- $NFA(\emptyset) = \text{void}$
- $NFA(R_1 \cup R_2) =$
All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\text{NFA}(R)) = L_1$.

Given an alphabet $\Sigma$

- $\text{NFA}(a) = \text{char}(a)$
- $\text{NFA}(\epsilon) = \text{nil}$
- $\text{NFA}(\emptyset) = \text{void}$
- $\text{NFA}(R_1 \cup R_2) = \text{union(\text{NFA}(R_1), \text{NFA}(R_2))}$
- $\text{NFA}(R_1 \cdot R_2) =$
All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If \( L(R) = L_1 \), then \( L(\text{NFA}(R)) = L_1 \).

Given an alphabet \( \Sigma \)

- \( \text{NFA}(a) = \text{char}(a) \)
- \( \text{NFA}(\epsilon) = \text{nil} \)
- \( \text{NFA}(\emptyset) = \text{void} \)
- \( \text{NFA}(R_1 \cup R_2) = \text{union}(\text{NFA}(R_1), \text{NFA}(R_2)) \)
- \( \text{NFA}(R_1 \cdot R_2) = \text{append}(\text{NFA}(R_1), \text{NFA}(R_2)) \)
- \( \text{NFA}(R^*) = \)
All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If $L(R) = L_1$, then $L(\text{NFA}(R)) = L_1$.

Given an alphabet $\Sigma$

- $\text{NFA}(a) = \text{char}(a)$
- $\text{NFA}(\varepsilon) = \text{nil}$
- $\text{NFA}(\emptyset) = \text{void}$
- $\text{NFA}(R_1 \cup R_2) = \text{union}(\text{NFA}(R_1), \text{NFA}(R_2))$
- $\text{NFA}(R_1 \cdot R_2) = \text{append}(\text{NFA}(R_1), \text{NFA}(R_2))$
- $\text{NFA}(R^*) = \text{star}(\text{NFA}(R))$
All Regexes have an equivalent NFA

Lemma 1.55 (ITC)

If \( L(R) = L_1 \), then \( L(\text{NFA}(R)) = L_1 \).

Given an alphabet \( \Sigma \)

- \( \text{NFA}(a) = \text{char}(a) \)
- \( \text{NFA}(\epsilon) = \text{nil} \)
- \( \text{NFA}(\emptyset) = \text{void} \)
- \( \text{NFA}(R_1 \cup R_2) = \text{union}(\text{NFA}(R_1), \text{NFA}(R_2)) \)
- \( \text{NFA}(R_1 \cdot R_2) = \text{append}(\text{NFA}(R_1), \text{NFA}(R_2)) \)
- \( \text{NFA}(R^*) = \text{star}(\text{NFA}(R)) \)

(Proof follows by induction on the structure of \( R \).)
The **void** NFA

\[ L(\text{void}) = \emptyset \]
The **void** NFA

$L(\text{void}) = \emptyset$
The **void** NFA

$L(\text{void}) = \emptyset$

start $\rightarrow q_1$
The **nil** operator

\[ L(\text{nil}) = \{ \epsilon \} \]
The \textbf{nil} operator

$L(\text{nil}) = \{\epsilon\}$
The **nil** operator

\[ L(\text{nil}) = \{ \epsilon \} \]

\[\text{start} \rightarrow q_1\]
The `char(c)` operator

\[ L(\text{char}(c)) = \{[c]\} \]
The \texttt{char} (a) operator

$L(\texttt{char}(a)) = \{ [a] \}$
The \texttt{char(a)} operator

\[ L(\text{char}(a)) = \{ [a] \} \]
The \textbf{union}(M, N) automaton

\[ L(\text{union}(M, N)) = L(M) \cup L(N) \]
The \textbf{union}($M, N$) automaton

\[ L(\text{union}(M, N)) = L(M) \cup L(N) \]

\[ N_1 \]

\[ N_2 \]

\[ \text{union}(N_1, N_2) = ? \]
The union \((M, N)\) operator

\[ L(\text{union}(M, N)) = L(M) \cup L(N) \]

Example \(\text{union}(N_1, N_2)\)

- Add a new initial state
- Connect new initial state to the initial states of \(N_1\) and \(N_2\) via \(\epsilon\)-transitions.
The \texttt{append}(M, N) operator

\[ L(\text{append}(M, N)) = L(M) \cdot L(N) \]
The **append**\((M, N)\) operator

\[ L(\text{append}(M, N)) = L(M) \cdot L(N) \]

**Example 1:** \( L(\text{concat}(\text{char}(a), \text{char}(b))) = \{ab\} \)
The **append** \((M, N)\) operator

\[ L(\text{append}(M, N)) = L(M) \cdot L(N) \]

**Example 1:** \[ L(\text{concat(char(a), char(b)))} = \{ab\} \]

**Solution**

\[ \text{start} \rightarrow q_1 \xrightarrow{a} q_2 \]

**What did we do?** Connect the accepted states of \(N_1\) to the initial state of \(N_2\) via \(\epsilon\)-transitions.

Why not connect directly from \(q_{1,1}\) into \(q_{1,2}\)? See next slide.
Concatenation example 2

Solution
Concatenation example 2

Solution
The $\text{star}(N)$ operator

$L(\text{star}(N)) = L(N)^*$
The \textit{star}(N) operator

$L(\text{star}(N)) = L(N)^*$

Example: $L(\text{star}(\text{concat}(\text{char}(a), \text{char}(b)))) = \{ w \mid w \text{ is a sequence of } ab \text{ or empty} \}$

Solution
The \textbf{star}(N) operator

\[ L(\text{star}(N)) = L(N)^* \]

Example: \( L(\text{star}(\text{concat}(\text{char(a)}, \text{char(b)}))) = \{ w \mid w \text{ is a sequence of } ab \text{ or empty} \} \)

Solution

- create a new state \( q_{1,1} \)
- \( \epsilon \)-transitions from \( q_{1,1} \) to initial state
- \( \epsilon \)-transitions from accepted states to \( q_{1,1} \)
- \( q_{1,1} \) is the only accepted state
The $\text{star}(N)$ operator

$L(\text{star}(N)) = L(N)^*$
The **star**\(^{(N)}\) operator

\[ L(\text{star}(N)) = L(N)^* \]
Completeness

All NFAs have an equivalent Regex

$\text{NFA} \rightarrow \text{REGEX}$
Completeness

All NFAs have an equivalent Regex

Why is this result important?
Completeness

All NFAs have an equivalent Regex

Why is this result important?

If we can derive an equivalent regular expression from any NFA, then our regular expression are enough to describe whatever can be described using finite automatons.
Overview:

Converting an NFA into a regular expression

There are many algorithms of converting an NFA into a Regex. Here is the algorithm we find in the book.

1. Wrap the NFA
2. Convert the NFA into a GNFA
3. Reduce the GNFA
4. Extract the Regex
Step 1: wrap the NFA

Given an NFA $N$, add two new states $q_{\text{start}}$ and $q_{\text{end}}$ such that $q_{\text{start}}$ transitions via $\epsilon$ to the initial state of $N$, and every accepted state of $N$ transitions to $q_{\text{end}}$ via $\epsilon$. State $q_{\text{end}}$ becomes the new accepted state.

Input
Step 1: wrap the NFA

Given an NFA $N$, add two new states $q_{start}$ and $q_{end}$ such that $q_{start}$ transitions via $\epsilon$ to the initial state of $N$, and every accepted state of $N$ transitions to $q_{end}$ via $\epsilon$. State $q_{end}$ becomes the new accepted state.

Input

Output
Step 2: Convert an NFA into a GNFA

A GNFA has regular expressions in the transitions, rather than the inputs.

- For every edge with $a_1, \ldots, a_n$ convert into $a_1 + \cdots + a_n$

Input

```plaintext
\[
\begin{align*}
\text{start} & \rightarrow q_{1,1} & q_{1,1} & \rightarrow q_{1,2} & q_{1,2} & \rightarrow q_{2,2} & q_{2,2} & \rightarrow q_{2,1} \\
& \quad | \quad \epsilon \quad | \quad 1 \quad | \quad 0, \epsilon \quad | \quad 0 \quad | \quad 0 \quad | \quad \epsilon \\
q_{3,2} & \rightarrow & q_{3,2} & \rightarrow & q_{4,2} & \rightarrow & q_{2,2} & \rightarrow \\
& \quad | \quad 1 \quad | \quad & \quad | \quad 1 \quad | \quad & \quad | \quad \epsilon \\
q_{2,1} & \rightarrow & & & & & & \\
\end{align*}
\]
```
Step 2: Convert an NFA into a GNFA

A GNFA has regular expressions in the transitions, rather than the inputs.

- For every edge with $a_1, \ldots, a_n$ convert into $a_1 + \cdots + a_n$
Step 3: Reduce the GNFA

While there are more than 2 states:
- pick a state and its incoming/outgoing edges, and convert it to transitions
Step 3.1: compress state $q_{1,2}$

\[
\text{compress}(q_{1,1} \xrightarrow{\epsilon} q_{1,2} \xrightarrow{0+\epsilon} q_{2,2}) = q_{1,1} \xrightarrow{\epsilon (0+\epsilon)} q_{2,2}
\]

\[
\text{compress}(q_{1,1} \xrightarrow{\epsilon} q_{1,2} \xrightarrow{1} q_{3,2}) = q_{1,1} \xrightarrow{\epsilon \cdot 1} q_{3,2}
\]

**Input**

![Diagram of the compressed states]
Step 3.1: compress state $q_{1,2}$

$$\text{compress}(q_{1,1} \to q_{1,2} \to q_{2,2}) = q_{1,1} \to q_{2,2}$$
$$\text{compress}(q_{1,1} \to q_{1,2} \to q_{3,2}) = q_{1,1} \to q_{3,2}$$

Each state that connects to $q_{1,2}$ must connect to every state that $q_{1,2}$ connects to. Some $q_{1,1}$ must connect with $q_{2,2}$ and $q_{1,1}$ must connect with $q_{3,2}$. 
Step 3.2: compress state $q_{2,2}$
Step 3.2: compress state $q_{2,2}$

Each state that connects to $q_{2,2}$ must connect to every state that $q_{2,2}$ connects to. Som $q_{1,1}$ must connect with $q_{2,1}$ and $q_{3,2}$ must connect with $q_{2,1}$.

Input

Output

compress($q_{1,1} \xrightarrow{0+\epsilon} q_{2,2} \xrightarrow{\epsilon} q_{2,1}$) = $q_{1,1} \xrightarrow{(0+\epsilon)\cdot\epsilon} q_{2,2}$

compress($q_{3,2} \xrightarrow{0} q_{2,2} \xrightarrow{\epsilon} q_{2,1}$) = $q_{3,2} \xrightarrow{0\cdot\epsilon} q_{2,1}$
Step 3.3: compress state $q_{3,2}$

After compressing a state, we must merge the new node with any old node (in red).

\[
\text{compress}(q_{1,1} \xrightarrow{1} q_{3,2} \xrightarrow{0} q_{3,2} \xrightarrow{0} q_{2,1}) + q_{1,1} \xrightarrow{0+\epsilon} q_{2,1} = q_{1,1} \xrightarrow{(10^*0) + (0+\epsilon)} q_{2,2}
\]
\[
\text{compress}(q_{1,1} \xrightarrow{1} q_{3,2} \xrightarrow{0} q_{3,2} \xrightarrow{1} q_{4,2}) = q_{3,2} \xrightarrow{10^*1} q_{2,1}
\]

Input

```
start → q_{1,1} → q_{3,2} → q_{2,1}
```

Output

```
start → q_{1,1} → q_{4,2} → q_{2,1}
```

10^*0 + \epsilon + 0
Step 3.3: compress state $q_{4,2}$

After compressing a state, we must merge the new node with any old node (in red).

\[
\text{compress}(q_{1,1} \xrightarrow{10^*1} q_{4,2} \xrightarrow{\varepsilon} q_{2,1}) + q_{1,1} \xrightarrow{10^*1+0+\varepsilon} q_{2,1} = q_{1,1} \xrightarrow{(10^*1\varepsilon)+(10^*0+0+\varepsilon)} q_{2,2}
\]

Input

\[
\begin{align*}
\text{start} & \rightarrow q_{1,1} \\
q_{1,1} & \xrightarrow{10^*1} q_{4,2} \\
q_{4,2} & \xrightarrow{\varepsilon} q_{2,1} \\
q_{2,1} & \xrightarrow{10^*0+\varepsilon+0} \text{start}
\end{align*}
\]

Output

\[
\begin{align*}
\text{start} & \rightarrow q_{1,1} \\
q_{1,1} & \xrightarrow{10^*0+\varepsilon+10^*1+0} q_{2,1}
\end{align*}
\]

Result: $10^*1 + 10^*0 + 0 + \varepsilon$
Exercise 1.66

Convert an NFA into a Regex

1. Convert the NFA into an NFA (same)

![Diagram of an NFA]

2. Wrap the NFA
Exercise 1.66

Convert an NFA into a Regex

1. Convert the NFA into an NFA (same)

2. Wrap the NFA
Exercise 1.66

Convert an NFA into a Regex

3. Convert NFA into GNFA

Before
Exercise 1.66

Convert an NFA into a Regex

3. Convert NFA into GNFA

Before

After
Exercise 1.66

Convert an NFA into a Regex

4. Compress state.

Before
Exercise 1.66

Convert an NFA into a Regex

4. Compress state.

Before

After
Exercise 1.66

Convert an NFA into a Regex

5. Compress state.

Before

![Diagram of an NFA with states and transitions]

- Start state: $q_{1,1}$
- Transition: $a^*b$ from $q_{1,1}$ to $q_{2,2}$
- Transition: $b + a$ from $q_{2,2}$ to $q_{2,1}$
- Transition: $\epsilon$ from $q_{2,2}$ to $q_{2,1}$
Exercise 1.66

Convert an NFA into a Regex

5. Compress state.

Before

After
Exercise 8

Convert an NFA into a Regex

Before

![NFA Diagram]

- Start state: $q_1$
- Transitions:
  - $q_1 \xrightarrow{\epsilon} q_4$
  - $q_1 \xrightarrow{a} q_2$
  - $q_2 \xrightarrow{a} q_2$
  - $q_2 \xrightarrow{b} q_3$
  - $q_3 \xrightarrow{b} q_3$
Exercise 8

Convert an NFA into a Regex

Before

After
Exercise 8

Convert an NFA into a Regex

Before
Exercise 8

Convert an NFA into a Regex

Before

After
Exercise 8
Convert an NFA into a Regex

Before
Exercise 8

Convert an NFA into a Regex

Before

After
Exercise 8

Convert an NFA into a Regex

Before

```
start → q₁,₁
        |     | ε     | ab⁺b
q₁,₁   → q₄,2 → q₂,₁
        |     |       |
ε + ab⁺b
```
Exercise 8

Convert an NFA into a Regex

Before

After