Introduction to the Theory of Computation

Lecture 12: Regular expressions & NFAs

Tiago Cogumbreiro
Today we will learn...

- NFA reduction graphs
- Converting REGEX to NFA
- Converting NFA to REGEX
Exercise

Strings that interleave one "a" with one "b"
Examples: "a", "b", "ab", "ba", "aba", "bab", "abab", "baba"
Exercise

Strings that interleave one "a" with one "b"
Examples: "a", "b", "ab", "ba", "aba", "bab", "abab", "baba"

- We start in an accepting state
- Reading an a moves us to $q_2$ which expects a b
- Reading a b moves us to $q_3$ which expects an a
- All states are accepting. **However, not all strings are accepted.**
Acceptance in an NFA

Acceptance is path finding

The given string must be a path from the starting node into the accepting node.

- NFAs can have **multiple** possible paths because of nondeterminism, contrary to DFAs!
Acceptance in an NFA

Acceptance of $\text{abbaba}$
Acceptance in an NFA

Acceptance of $ab\texttt{baba}$
Acceptance in an NFA

Acceptance of $ababa$

Diagram:

- Start state: $q_1$
- Transitions:
  - $a, b$ from $q_1$ to $q_1$
  - $a$ from $q_1$ to $q_2$
  - $b$ from $q_2$ to $q_3$
  - $a$ from $q_3$ to $q_4$
  - Loop $a, b$ at $q_4$

- Additional diagrams:
  - Transition from $q_1$ to $q_1$ on $a$
  - Transition from $q_2$ to $q_3$ on $b$
  - Transition from $q_3$ to $q_3$ on $b$
Acceptance in an NFA

Acceptance of abbaba
Acceptance in an NFA

Acceptance of \textit{ababa}
Acceptance in an NFA

Acceptance of \texttt{abbab}
Acceptance in an NFA

Acceptance of $\text{abbaba}$
Acceptance in an NFA

- There are multiple concurrent possible paths and a current state
- Given a current state, if there are no transitions for a given input, the path ends
- Once we reach the final path, we check if there are accepting states
Epsilon transitions
Epsilon transitions

Exercise 2

Let $\Sigma = \{a, b\}$. Give an NFA with four states that recognizes the following language

$$\{w \mid w \text{ contains the strings } aba \text{ or } aa\}$$

Note

- NFAs can also include $\epsilon$ transitions, which may be taken without consuming an input.
Exercise 2: acceptance of $aaba$

Interleave input with $\epsilon$.

Read $a$
Exercise 2: acceptance of $a\epsilon aba$

Interleave input with $\epsilon$.

Read $\epsilon$
Exercise 2: acceptance of $aaba$

Interleave input with $\epsilon$.

Read $a$
Exercise 2: acceptance of aabεa

Interleave input with ε.

Read ε
Exercise 2: acceptance of $aa\overline{b}a$

Interleave input with $\epsilon$.

Read $b$
Exercise 2: acceptance of aaba

Interleave input with $\epsilon$.

Read a
Exercise 2: acceptance of $aaba\epsilon$

Interleave input with $\epsilon$.

Read $\epsilon$
Exercise 2: acceptance of aaba

Interleave input with $\epsilon$.

Read $\epsilon$
Note $\epsilon$ transitions in the initial state

We looked at $\epsilon$ in the middle of the state diagram. Let us observe their effect in the initial state.
Exercise 2: acceptance of bd

Read $\epsilon$
Exercise 2: acceptance of \( bd \)

Read b
Exercise 2: acceptance of bd

Read $\epsilon$ and then read $d$
Exercise 2: acceptance of bd
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 $L(NFA(R)) = L_1$.

Given an alphabet $\Sigma$

- $NFA(a) = \text{char}(a)$
- $NFA(\epsilon) = \text{nil}$
- $NFA(\emptyset) =$
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 $\Sigma$

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

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

$L(\text{nil}) = \{ \epsilon \}$

![Diagram](image.png)
The \texttt{char(c)} operator

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

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

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

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

\[
L(\text{union}(M, N)) = L(M) \cup L(N)
\]
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 \textbf{append}(M, N) \textit{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**

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 \textbf{star}(N) operator

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

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

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

Solution
The **star**\((N)\) operator

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

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

Solution

- create a new state \(q_{1,1}\)
- \(\varepsilon\)-transitions from \(q_{1,1}\) to initial state
- \(\varepsilon\)-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 \textbf{star}(N) operator

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

All NFAs have an equivalent Regex

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

Output
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 $q_{1,2}$

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

\[
\text{compress}(q_{1,1} \xrightarrow{\epsilon} q_{1,2} \xrightarrow{1} q_{3,2}) = q_{1,1} \xrightarrow{\epsilon \cdot 1} q_{3,2}
\]
Step 3.1: compress state $q_{1,2}$

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

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

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 $q_{2,2}$

Input

Diagram:

- Start state $q_{1,1}$
- State $q_{3,2}$
- State $q_{2,2}$
- State $q_{4,2}$
- State $q_{2,1}$

Transitions:
- $1$ from $q_{1,1}$ to $q_{3,2}$
- $0 + \epsilon$ from $q_{2,2}$ to $q_{2,1}$
- $\epsilon$ from $q_{2,2}$ to $q_{2,1}$
- $\epsilon$ from $q_{3,2}$ to $q_{4,2}$
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. So $q_{1,1}$ must connect with $q_{2,1}$ and $q_{3,2}$ must connect with $q_{2,1}$.

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

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

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

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

\[
\text{compress}(q_{1,1} \xrightarrow{1} q_{3,2} \xrightarrow{0} q_{3,2} \xrightarrow{1} q_{4,2}) = q_{3,2} \xrightarrow{10^*1} q_{2,1}
\]
Step 3.3: compress state $q_{4,2}$

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

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

**Input**

```
start \rightarrow q_{1,1} \xrightarrow{10^*1} q_{4,2} \xrightarrow{\epsilon} q_{2,1} \xrightarrow{10^*0+\epsilon+0} q_{2,1}
```

**Output**

```
start \rightarrow q_{1,1} \xrightarrow{10^*0+\epsilon+10^*1+0} q_{2,1}
```

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

Convert a DFA into a Regex

1. Convert the DFA into an NFA (same)

2. Wrap the NFA
Exercise 1.66

Convert a DFA into a Regex

1. Convert the DFA into an NFA (same)

2. Wrap the NFA
Exercise 1.66

Convert a DFA into a Regex

3. Convert NFA into GNFA

Before

![Diagram of NFA]

\( q_{1,1} \) — EPSILON — \( q_{1,2} \) — 'a' — \( q_{2,2} \) — 'a,b' — \( q_{2,1} \)
Exercise 1.66

Convert a DFA into a Regex

3. Convert NFA into GNFA

Before

After
Exercise 1.66

Convert a DFA into a Regex

4. Compress state.

Before
Exercise 1.66

Convert a DFA into a Regex

4. Compress state.

Before

After
Exercise 1.66

Convert an DFA into a Regex

5. Compress state.

Before
Exercise 1.66

Convert an DFA into a Regex

5. Compress state.

Before

After

\[ a^*b((b+a)^*) \]
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
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