NFA/DFA, Relation to Regular Languages

Lecture 6
NFA recap

• Last lecture, we saw these objects called NFAs…

• Like DFA, but with a weird transition function: choices!

• DFA is a special case of NFA (how?)
NFA recap

- Last lecture, we saw these objects called NFAs...

3 models for (Regular) Languages:

Regular Expression

DFA

NFA
NFA recap

Kleene’s Theorem

Regular Expression = DFA = NFA
NFA+$\varepsilon$: Formally

- I want to be able to change my state without consuming input
NFA+$\varepsilon$: Formally

- I want to be able to change my state without consuming input

- On input 10001?
NFA+ε: Formally

\[ N = (\Sigma, Q, \delta, s, A) \]

\( \Sigma \): alphabet  \( Q \): state space  \( s \): start state  \( A \): set of accepting states

\[
\delta : Q \times \{\Sigma \cup \epsilon\} \rightarrow \mathcal{P}(Q)
\]

We say  \( q \xrightarrow{w} \text{N} p \) if \( \exists \ a_1, \ldots, a_t \in \Sigma \cup \{\epsilon\} \) and \( q_1, \ldots, q_{t+1} \in Q \), such that
\( w = a_1 \ldots a_t, \ q_1 = q, \ q_{t+1} = p \), and \( \forall \ i \in [1, t], \ q_{i+1} \in \delta(q_i, a_i) \)

\[
L(N) = \{ \ w | s \xrightarrow{w} \text{N} p \ \text{for some} \ \ p \in A \ \} \]

e.g., \( \delta(1, o) = \{2\} \), \( \delta(1, x) = \emptyset \), \( \delta(1, \epsilon) = \{2\} \).
NFA+$\varepsilon$: Formally

We define the $\varepsilon$-reach of a state $p$:

- $p$ itself
- any state $q$ such that $r \xrightarrow{\varepsilon}_N q$ for some $r$ in the $\varepsilon$-reach of $p$

Means that there is a sequence of $\varepsilon$-transitions from $p$ to $q$

e.g., $\delta(1,o) = \{2\}$, $\delta(1,x) = \emptyset$, $\delta(1,\varepsilon) = \{2\}$.  

$\varepsilon$-reach($\{1\}$) = $\{1, 2, 3, 0\}$
Get rid of nothing

Can modify any NFA $N$, to get an NFA $N_{\text{new}}$ without $\varepsilon$-moves

$$N_{\text{new}} = (\Sigma, Q_{\text{new}}, \delta_{\text{new}}, s_{\text{new}}, A_{\text{new}})$$

$$Q_{\text{new}} = Q$$

$$s_{\text{new}} = s$$

$$A_{\text{new}} = \{ q \mid \varepsilon\text{-reach}(q) \text{ includes a state in } A \}$$

$$\delta_{\text{new}}(q, a) = \bigcup_{p \in \varepsilon\text{-reach}(q)} \delta(p, a)$$

e.g.: $\delta_{\text{new}}(1, o) = \{0, 2, 3, 4, 5\}$
Get rid of nothing

Can modify any NFA \( N \), to get an NFA \( N_{\text{new}} \) without \( \varepsilon \)-moves

\[
N_{\text{new}} = (\Sigma, Q_{\text{new}}, \delta_{\text{new}}, s_{\text{new}}, A_{\text{new}})
\]

\[Q_{\text{new}} = Q\]

\[s_{\text{new}} = s\]

\[A_{\text{new}} = \{ q \mid \varepsilon-\text{reach}(q) \text{ includes a state in } A \}\]

\[
\delta_{\text{new}}(q, a) = \bigcup_{p \in \varepsilon-\text{reach}(q)} \delta(p, a)
\]

**Theorem:** \( L(N) = L(N_{\text{new}}) \)
NFA+$\varepsilon$: Formally
NFA-\(\varepsilon\)

\[
\delta_{\text{new}}(q, a) = \bigcup_{p \in \varepsilon-\text{reach}(q)} \delta(p, a)
\]
\[
\delta_{\text{new}}(q, a) = \bigcup_{p \in \varepsilon - \text{reach}(q)} \delta(p, a)
\]
\[ \delta_{\text{new}}(q, a) = \bigcup_{p \in \varepsilon - \text{reach}(q)} \delta(p, a) \]
NFA-∈

• Same NFA!
Theorem: A language $L$ can be described by a regular expression if and only if $L$ is the language accepted by a DFA.
Kleene’s theorem

DFA

NFA+ε

Regular Expressions
Kleene’s theorem

DFA -> NFA+\(\varepsilon\) -> Regular Expressions

Do Nothing

1 -> 2 -> 3
Kleene’s theorem

DFA $\rightarrow$ Do Nothing $\rightarrow$ NFA+$\varepsilon$ $\rightarrow$ Regular Expressions

1 2 3
DFA from NFA (aka the subset construction)

NFA: $N = (\Sigma, Q, \delta, s, A)$

$\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$

assume no $\varepsilon$-moves
NFA
NFA to DFA

NFA: \( N = (\Sigma, Q, \delta, s, A) \)
\[ \delta : Q \times \Sigma \rightarrow \mathcal{P}(Q) \]

DFA: \( M_N = (\Sigma, Q', \delta', s', A') \)
\[ \delta' : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q) \]
\[ Q' = 2^Q = \mathcal{P}(Q) \]
\[ s' = \{s\} \]

Deterministic state is now a set of (non-deterministic) states

\[ A' = \{\text{all subsets } P \text{ of } Q \text{ s.t. } P \cap A \neq \emptyset\} \]

Theorem: \( L(N) = L(M_N) \)
\[ \delta' : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q) \]
\[ \delta'(P, a) = \bigcup_{q \in P} \delta(q, a) \]
NFA to DFA

• There are too many states in this DFA, more than necessary.

• Construct the DFA incrementally instead, by performing BFS on the DFA graph.

• Prepare a table as follows
\[
\begin{array}{cccc}
\delta'(P,0) & \delta'(P,1) & q' \in A' \\
\hline
s & s & as & bs & No \\
as & as & ats & bs & No \\
bs & bs & as & bts & No \\
ats & ats & ats & bts & Yes \\
ibs & bts & ats & bts & Yes \\
\end{array}
\]
The diagram illustrates a deterministic automaton with states labeled as, bs, s, bts, ats, a, and t. The transitions are labeled with inputs 0 and 1, and the output actions are as, bs, ats, bts.

The table summarizes the transitions for inputs 0 and 1:

<table>
<thead>
<tr>
<th>P</th>
<th>ε</th>
<th>δ'(P,0)</th>
<th>δ'(P,1)</th>
<th>q' ∈ A'</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>s</td>
<td>as</td>
<td>bs</td>
<td>No</td>
</tr>
<tr>
<td>as</td>
<td>as</td>
<td>ats</td>
<td>bs</td>
<td>No</td>
</tr>
<tr>
<td>bs</td>
<td>bs</td>
<td>as</td>
<td>bts</td>
<td>No</td>
</tr>
<tr>
<td>ats</td>
<td>ats</td>
<td>ats</td>
<td>bts</td>
<td>Yes</td>
</tr>
<tr>
<td>bts</td>
<td>bts</td>
<td>ats</td>
<td>bts</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Kleene’s theorem

DFA \rightarrow \text{Do Nothing} \rightarrow \text{NFA} + \varepsilon \rightarrow \text{Regular Expressions}

\rightarrow \text{Subset Construction}
Kleene’s theorem

- DFA
- NFA+ε
- Regular Expressions

Do Nothing

Subset Construction

1 2 3
NFAs from Regular Languages

**Theorem (Thompsons Algorithm):** Every regular language is accepted by an NFA.

We will show how to get from regular expressions to NFA+$\varepsilon$, but in a **particular way. One accepting state only!**
Single Final State Form

Can compile a given NFA so that there is only one final state (and there is no transition out of that state)
**Theorem**: Every regular language is accepted by an NFA.

**Proof**: Recall definition of Regular Language.

<table>
<thead>
<tr>
<th>Atomic expressions (Base cases)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\emptyset$</td>
<td>$L(\emptyset) = \emptyset$</td>
</tr>
<tr>
<td>$w$ for $w \in \Sigma^*$</td>
<td>$L(w) = {w}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inductively defined expressions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$(r_1 + r_2)$</td>
<td>$L(r_1 + r_2) = L(r_1) \cup L(r_2)$</td>
</tr>
<tr>
<td>$(r_1r_2)$</td>
<td>$L(r_1r_2) = L(r_1)L(r_2)$</td>
</tr>
<tr>
<td>$(r^*)$</td>
<td>$L(r^<em>) = L(r)^</em>$</td>
</tr>
</tbody>
</table>
Theorem: Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Base Case 1: L = ∅
NFAs from Regular Languages

**Theorem**: Every regular language is accepted by an NFA.

**Proof**: Recall definition or Regular Language.

Base Case 1: $L = \emptyset$

What is a NFA for $L$?

![Diagram](image)
**Theorem**: Every regular language is accepted by an NFA.

**Proof**: Recall definition or Regular Language.

Base Case 2: $L = \{\varepsilon\}$

What is a NFA for $L$?
Theorem: Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Base Case 3: \( L = \{a\} \), some string in \( \Sigma^* \) (e.g. HW2)

What is a NFA for \( L \)?

![Diagram of NFA](image)
Theorem: Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 1: \( L = A \cup B \)

What is a NFA for \( L \)?
Closure Under Union
**Theorem**: Every regular language is accepted by an NFA.

**Proof**: Recall definition or Regular Language.

Inductive case 1: \( L = A \cup B \)

What is a NFA for \( L \)?
Theorem: Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 2: \( L = AB \)

What is a NFA for \( L \)?
Closure Under Concatenation

\[ \varepsilon \]
Theorem: Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 2: $L=AB$

What is a NFA for $L$?
NFAs from Regular Languages

**Theorem**: Every regular language is accepted by an NFA.

**Proof**: Recall definition or Regular Language.

Inductive case 3: \( L = A^* \)

What is a NFA for \( L \)?
Closure Under Kleene Star

\[ s \xrightarrow{\varepsilon} C \xrightarrow{\varepsilon} t \]
**Theorem**: Every regular language is accepted by an NFA.

**Proof**: Recall definition or Regular Language.

Inductive case 3: \( L = A^* \)

What is a NFA for \( L \)?
NFAs from Regular Languages

**Theorem**: Every regular language is accepted by an NFA.

**Proof**: Recall definition or Regular Language.

**Inductive case 3**: $L = A^*$

Why not?
**Theorem**: Every regular language is accepted by an NFA.

**Proof**: Recall definition or Regular Language.

Inductive case 3: $L = A^*$
**Theorem**: Every regular language is accepted by an NFA.

**Proof**: Recall definition or Regular Language.

Inductive case 3: \( L = A^* \)
Theorem: Every regular language is accepted by an NFA.

Proof: Recall definition or Regular Language.

Inductive case 3: $L = A^*$
NFAs & Regular Languages

Example: $L$ given by regular expression $(10+1)^*$