NFA/DFA, Relation to Regular Languages

Lecture 7
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+$\varepsilon$: Formally

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

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

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

We say $q \xrightarrow{w} N p$

$$L(N) =$$

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

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

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

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

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

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

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

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

e.g., $\delta(1,o) = \{2\}$, $\delta(1,x)=\emptyset$, $\delta(1,\varepsilon)=\{2\}$.  
$\varepsilon$-reach($\{1\}$) = \{1, 2, 3, 0\}
NFA+ε: Formally

We define the ε-reach of a state p:

- p itself
- any state q such that $\epsilon_{N,r}\rightarrow q$ for some r in the ε-reach of p

Means that there is a sequence of ε-transitions from p to q

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

$\epsilon$-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

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

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

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

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

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

\[
\delta_{\text{new}}(1, o) = \{ 0, 2, 3, 4, 5 \}
\]
Get rid of nothing

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

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

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

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

$$A_{\text{new}} = \{\}$$
Get rid of nothing

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

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

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

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

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

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

\textbf{Theorem}: $L(N) = L(N_{\text{new}})$
NFA+$\varepsilon$: Formally
\[ \delta_{\text{new}}(q, a) = \bigcup_{p \in \varepsilon - \text{reach}(q)} \delta(p, a) \]
\[ \delta_{\text{new}}(q, a) = \bigcup_{p \in \varepsilon - \text{reach}(q)} \delta(p, a) \]
\[ \delta_{\text{new}}(q, a) = \bigcup_{p \in \varepsilon - \text{reach}(q)} \delta(p, a) \]
NFA-$\varepsilon$

- Same NFA!
Kleene’s theorem

**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 \leftrightarrow NFA+\varepsilon \leftrightarrow \text{Regular Expressions}
Kleene’s theorem

- DFA
- NFA+ε
- Regular Expressions

1. Do Nothing
2. 2
3. 3

DFA

NFA+ε

Regular Expressions
Kleene’s theorem

DFA

Do Nothing

NFA+$\varepsilon$

\[ \xrightarrow{1} \]

\[ \xrightarrow{2} \]

\[ \xrightarrow{3} \]

Regular Expressions
DFA from NFA (aka the subset construction)

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

$\delta : Q \times \Sigma \rightarrow P(Q)$

assume no $\varepsilon$-moves
NFA

1001 1001 1001 1001 1001 1001
NFA to DFA

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

\[ \delta : Q \times \Sigma \to P(Q) \]

assume no \( \varepsilon \)-moves

DFA: \( M_N = (\Sigma, Q', \delta', s', A') \)

\[ Q' = 2^Q = P(Q) \]

\[ s' = \{s\} \]

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

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

Theorem: \( L(N) = L(M_N) \)

\[ \delta' : P(Q) \times \Sigma \to P(Q) \]

\[ \delta'(P, a) = \bigcup_{q \in P} \delta(q, a) \]
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
\[ P, \varepsilon, \delta'(P,0), \delta'(P,1), q' \in A' \]

<table>
<thead>
<tr>
<th></th>
<th>( P )</th>
<th>( \varepsilon )</th>
<th>( \delta'(P,0) )</th>
<th>( \delta'(P,1) )</th>
<th>( q' \in A' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>s</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>as</td>
<td>as</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bs</td>
<td>bs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ats</td>
<td>ats</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bts</td>
<td>bts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$P$</td>
<td>$\varepsilon$</td>
<td>$\delta'(P,0)$</td>
<td>$\delta'(P,1)$</td>
<td>$q' \in A'$</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>---------------</td>
<td>----------------</td>
<td>----------------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>$s$</td>
<td>$s$</td>
<td>$as$</td>
<td>$bs$</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>$as$</td>
<td>$as$</td>
<td>$ats$</td>
<td>$bs$</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>$bs$</td>
<td>$bs$</td>
<td>$as$</td>
<td>$bts$</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>$ats$</td>
<td>$ats$</td>
<td>$ats$</td>
<td>$bts$</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>$bts$</td>
<td>$bts$</td>
<td>$ats$</td>
<td>$bts$</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>
\[
P \quad \varepsilon \quad \delta'(P,0) \quad \delta'(P,1) \quad q' \in A'
\begin{array}{cccc}
\hline
P & \varepsilon & \delta'(P,0) & \delta'(P,1) & q' \in A' \\
\hline
s & s & as & bs & No \\
as & as & as & bs & No \\
bs & bs & as & bts & No \\
ats & ats & ats & bts & Yes \\
bts & bts & ats & bts & Yes \\
\hline
\end{array}
\]
Kleene’s theorem

DFA

Do Nothing

NFA+ɛ

Regular Expressions

Subset Construction
Kleene’s theorem

DFA  \(\xrightarrow{1} \)  NFA+\(\varepsilon\)  \(\xrightarrow{2} \)  Regular Expressions

Do Nothing

Subset Construction
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 or 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: $L = \emptyset$

What is a NFA for $L$?
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 \( L \)?
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 $L$?
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 \( L \)?
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
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: \( L = A^* \)

I need the new start state.
NFAs & Regular Languages

Example: \( L \) given by regular expression \((10+1)^*\)
NFAs & Regular Languages

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