Lecture 5

Last time:

- If $L$ is automata then there is an NFA $N$ so that $L = L(N)$
  
  Pf. $N$ can be interpreted as $N$

- If $L$ is regular then there is an NFA $N$ so that $L = L(N)$

  Pf. Apply Thompson’s alg

$$\text{DFA } \rightarrow \text{ NFA } \leftarrow \text{ regex}$$

today

In conclusion: auto $\leftrightarrow$ reg

$$\text{NFA } \rightarrow \text{ DFA: Subset construction (power set)}$$

Idea: in prod construction: simulated being in two machines at once.

$\rightarrow$ here: build a DFA that simulates nondeterminism in the NFA by keeping track of all possible states we can be in

Every subset of $\Sigma$ represents a possible set of states we can be in simultaneously.

$L = \text{nfa } N = (Q, \Sigma, \delta, q_0, F)$
Given NFA $N = (Q, \Sigma, S, s, A)$
build DFA $M = (Q', \Sigma, S', s', A')$

$Q' = \mathcal{P}(Q)$ ← power set.

$S' = \{ x \in S \mid x \in \epsilon\text{-reach}(S) \}$

$A' = \{ T \subseteq Q \mid T \cap A \neq \emptyset \}$

$\delta'(T, a) = \bigcup_{q \in T} \delta^*(q, a)$

NFA acceptance:
$\delta^*(s, w) \cap A \neq \emptyset$

NFA states:

In NFA:
$\delta^*(q, w) = \begin{cases} \epsilon\text{-reach}(q) & \text{if } w = \epsilon \\ \epsilon\text{-reach}(q) \cup \delta'(T', a) & \text{if } w = ax \end{cases}$

where $T' = \bigcup_{t \in T} \{ t \}$

$L = \{ w \mid \text{second to last symbol in } w \text{ is } 0 \}$

DFA by hand:

\[ \text{Graphical representation of a DFA} \]
NFA:

```
S ↦ o → S ↦ o1 → b
```

NFA → DFA

```
φ ↦ o1 → b
φ ↦ o1 → a
φ ↦ o01 → b
φ ↦ o01 → a
φ ↦ 01 → b
φ ↦ 01 → a
φ ↦ o1 → b
φ ↦ o1 → a
φ ↦ 01 → b
φ ↦ 01 → a
φ ↦ o01 → b
φ ↦ o01 → a
φ ↦ 001 → b
φ ↦ 001 → a
```

⚠️ \( P(\omega) = 2^{\omega} \) if \( \omega \) is large?

\((0+1)^* \rightarrow \text{Thompson}\)
Incremental Subset Construction

- Build the part of the DFA I need.

<table>
<thead>
<tr>
<th>q</th>
<th>ε-read(q)</th>
<th>S'(q,0)</th>
<th>S'(q,1)</th>
<th>Λ'</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>sabcdfg</td>
<td>sabcdfg</td>
<td>sabcdfg</td>
<td>✓</td>
</tr>
<tr>
<td>S'</td>
<td>sabcdfg</td>
<td>sabcdfg</td>
<td>sabcdfg</td>
<td>✓</td>
</tr>
<tr>
<td>S'</td>
<td>sabcdfg</td>
<td>sabcdfg</td>
<td>sabcdfg</td>
<td>✓</td>
</tr>
</tbody>
</table>

Simpler DFA:

NFA → regex (state elimination)

(informally) GNFAs/ expression NFAs/expression automata

like NFA's but transitions can be arbitrary regexes,
extactly one transition between each pair of states
(missing transitions are φ)

require: start state have no meaning transitions?
require: Start state have no meaning transitions
only one accept state w/ no outgoing transitions

 enforce by adding new start/accept state if necessary.

if two states, then these are start/accept

else, pick qrep other than start/accept
& rip it out

if we do this repeatedly, end up w/ just start/accept done.

ex.
2. nip out \( c \).

Before:
\[
\begin{align*}
& a \\ & \xrightarrow{a} c \\ & \xrightarrow{a} b
\end{align*}
\]

After:
\[
\begin{align*}
& a \\ & \xrightarrow{00^*1} a \\
& \xrightarrow{a} 00^*1 + 0 \\ & \xrightarrow{b} b
\end{align*}
\]

3. nip out \( a \)

\[
S \xrightarrow{\epsilon} a \xrightarrow{b} S \xrightarrow{\epsilon} b
\]

4. nip out \( b \)

\[
S \xrightarrow{b} \xrightarrow{\epsilon} f \xrightarrow{(00^*1)^* (00^*1 + 0)} (0\epsilon1)^*
\]

\[\text{Ripping out states in different orders may give different but equivalent regexes.}\]

1. normalize.

\[
\begin{align*}
& S \\
& \xrightarrow{\epsilon} a \\
& \xrightarrow{b} \xrightarrow{\epsilon} \xrightarrow{0\epsilon1} \xrightarrow{0\epsilon1}
\end{align*}
\]
2. rip out `a`

```
before
S → a → c
S → a → b
S → a → c
```

```
after
S → c
S → b
S → c
```

3. rip out `c`

```
before
S → c → b
S → c → b
S → c → c
```

```
after
S → c → b
S → c → b
S → c → b
```

4. rip out `b`

```
S → o → c
```

```
S → o → (o+1)*
S → o → (o+1)*
S → o → (o+1)*
```

\[(0*1)^* (0*1 + 0)(0+1)^* \equiv (0 + o(0+1)^*(0+1))(0+1)^* \equiv 0(0+1)^*\]

In general, state elimination gives larger regexes than optimal.
Useful (?) rule

\[ A + B B^* A = B^* A \]

\( \xi_1, \xi_2, \xi_3, \xi_4 \)