Last Time

- Review of DFAs
- Regular Languages
- Regular Language Operations
- Proof of Regular Languages Closure Under Union
Nondeterminism

• Deterministic Finite Automata:
  – Given that we’re in state $r_i$ and that the next input character is $w_{i+1}$, we know what state to move to.
  – We can trace the flow of a computation across the full input $w$.

• Nondeterministic Finite Automata:
  – State $r_i$ and character $w_{i+1}$ could lead to two or more different states, or maybe to no states at all!
  – We could switch between state $r_i$ and state $r_{i+1}$ without reading anything at all!
As the Book Explains...
Why the Heck Would You Want to Do That?!?!?

• Remember our closure under union proof?
• Wouldn’t this be easier?

“I’m just gonna guess... and my guess will always be right!”
Example 1
Example 1

- states: \( Q = \{q_1, q_2, q_3, q_4\} \)
- alphabet: \( \Sigma = \{0, 1\} \)
- transition function: \( \delta: Q \times \Sigma \epsilon \rightarrow \mathcal{P}(Q) \)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>( \epsilon )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1 )</td>
<td>{q_1}</td>
<td>{q_1, q_2}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>{q_3}</td>
<td>\emptyset</td>
<td>{q_3}</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>\emptyset</td>
<td>{q_4}</td>
<td>\emptyset</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>{q_4}</td>
<td>{q_4}</td>
<td>\emptyset</td>
</tr>
</tbody>
</table>

- start state: \( q_1 \)
- accept states: \( F = \{q_4\} \)
What Strings Will $M_1$ Accept?

- Accepts 11
- Accepts 101
- Accepts 1101
- Accepts 1010101
- Accepts 100101001
- Rejects 0
- Rejects 1
- Rejects 00
- Rejects 0010
- Rejects 1001001001

Contains substring 11 or 101

Doesn’t
Example 2
Example 2

– states: \( Q = \{q_0, q_1, q_2, q_3, q_4, q_5\} \)
– alphabet: \( \Sigma = \{0\} \)
– transition function: \( \delta: Q \times \Sigma \epsilon \rightarrow \mathcal{P}(Q) \)
– start state: \( q_0 \)
– accept states: \( F = \{q_1, q_3\} \)
What Strings Will $M_2$ Accept?

Strings of length 2n or 3n

- Accepts $\epsilon$
- Accepts 00
- Accepts 000
- Accepts 0000
- Accepts 00000

Other length strings

- Rejects 0
- Rejects 00000
- Rejects 000000
- Rejects 0000000
- Rejects 00000000

Strings of length 2n or 3n

- Accepts $\epsilon$
- Accepts 00
- Accepts 000
- Accepts 0000
- Accepts 000000
Every NFA has an equivalent DFA!

• **Equivalence**: Two machines are equivalent if they recognize the same language.

• **Proof Idea**:
  – Convert the NFA into a DFA that simulates it.
  – Each DFA state represents a subset of the possible states that the NFA could be in.
  – If the NFA has \( k \) states, the DFA will have \( 2^k \) states.
In Action...

1. Determine the DFA’s states (there will be $2^3 = 8$).

2. Determine the start and accept states of the DFA.
   - Start – state which contains the NFA start state, plus any states reachable through epsilon arrows.
   - Accept – states which contain the NFA accept state.

3. Determine the DFA’s transition function.
   - Map the arrows from the NFA by thinking it through.

4. Simplify the machine by removing useless states.
In Action...

\[
\begin{align*}
\{1, 3\} & \xrightarrow{a} \{3\} & \{3\} & \xrightarrow{b} \emptyset \\
\{2\} & \xrightarrow{b} \{2, 3\} & \{2, 3\} & \xrightarrow{a} \{1, 2, 3\} \\
\{1, 3\} & \xrightarrow{b} \emptyset & \emptyset & \xrightarrow{a,b} \{1, 2, 3\}
\end{align*}
\]
So What About Those Operations on Regular Languages?
So What About Those Operations on Regular Languages?

**Proof:** Let $N_1$ recognize $A_1$, where $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and let $N_2$ recognize $A_2$, where $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Construct $N$ to recognize $A_1 \cup A_2$, where $N = (Q, \Sigma, \delta, q_0, F)$:

1. $Q = \{q_0\} \cup Q_1 \cup Q_2$
2. $\Sigma$ is the same as in $N_1$ and $N_2$
3. For any $q \in Q$ and any $a \in \Sigma_\varepsilon$, $\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$
4. $q_0$ is the new start state added.
5. $F = F_1 \cup F_2$. ■
So What About Those Operations on Regular Languages?
So What About Those Operations on Regular Languages?

Proof: Let $N_1$ recognize $A_1$, where $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and let $N_2$ recognize $A_2$, where $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Construct $N$ to recognize $A_1 \circ A_2$, where $N = (Q, \Sigma, \delta, q_0, F)$:

1. $Q = Q_1 \cup Q_2$
2. $\Sigma$ is the same as in $N_1$ and $N_2$
3. For any $q \in Q$ and any $a \in \Sigma_\varepsilon$, $\delta(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_2(q, a) \cup \{q_2\} & q \in Q_2 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_2(q, a) & q \in Q_2 \end{cases}$
4. $q_0$ is the start state of $N_1$.
5. $F = F_2$. ■
So What About Those Operations on Regular Languages?
So What About Those Operations on Regular Languages?

Proof: Let $N_1$ recognize $A_1$, where $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$.

Construct $N$ to recognize $A_1^*$, where $N = (Q, \Sigma, \delta, q_0, F)$:

1. $Q = \{q_0\} \cup Q_1$
2. $\Sigma$ is the same as in $N_1$
3. For any $q \in Q$ and any $a \in \Sigma$, $\delta(q, a) = \begin{cases} 
\delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\
\delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a \neq \epsilon \\
\{q_1\} & q = q_0 \text{ and } a = \epsilon \\
\emptyset & q = q_0 \text{ and } a \neq \epsilon
\end{cases}$
4. $q_0$ is the new start state added.
5. $F = \{q_0\} \cup F_1$. ■
Any Questions?

HOMEWORK (due 9/16)
1.7abcd, 1.8a, 1.9a, 1.10a, 1.15
Image Credits

- Slide 3: Computer thrown out window (http://www.examiner.com/images/blog/EXID20872/images/computer-thrown-out-a-window%281%29.JPG)
- Slide 4: NFA explanation images (Sipser textbook, 2nd edition)
- Slides 16, 18, 20: Operations on regular expression images (Sipser textbook, 2nd edition)