CSE 311: Foundations of Computing

Lecture 24: FSM Minimization & NFAs
Last class: Finite State Machines

- States
- Transitions on input symbols
- Start state and final states
- The “language recognized” by the machine is the set of strings that reach a final state from the start

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
Last class: Finite State Machines

• Each machine designed for strings over some fixed alphabet $\Sigma$.

• Must have a transition defined from each state for every symbol in $\Sigma$.

<table>
<thead>
<tr>
<th>Old State</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_0$</td>
<td>$s_0$</td>
<td>$s_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$s_0$</td>
<td>$s_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$s_0$</td>
<td>$s_3$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>$s_3$</td>
<td>$s_3$</td>
</tr>
</tbody>
</table>
State Machine Design Recipe

Given a language, how do you design a state machine for it?

Need enough states to:

• Decide whether to accept or reject at the end
• Update the state on each new character
State Machine Design Recipe

$M_2$: Strings where the sum of digits mod 3 is 0
State Machine Design Recipe

\( M_2: \) Strings where the sum of digits mod 3 is 0

Can we get away with two states?

- One for 0 mod 3 and one for everything else
State Machine Design Recipe

$M_2$: Strings where the sum of digits mod 3 is 0

Can we get away with two states?
• One for 0 mod 3 and one for everything else

This would be enough to decide at the end!

But can’t update the state on each new character
State Machine Design Recipe

M₂: Strings where the sum of digits mod 3 is 0

Can we get away with two states?
• One for 0 mod 3 and one for everything else

This would be enough to decide at the end!

But can’t update the state on each new character:
• If you’re in the “not 0 mod 3” state, and the next character is 1, which state should you go to?
State Machine Design Recipe

M₂: Strings where the sum of digits mod 3 is 0

So, we need three states.
What information should we track?
Strings over \{0, 1, 2\}

\(M_2: \) Strings where the sum of digits mod 3 is 0
State Machine Design Recipe

Given a language, how do you design a state machine for it?

Need enough states to:

• Decide whether to accept or reject at the end
• Update the state on each new character
Strings over \{0,1,2\} w/ even number of 2’s and mod 3 sum 0
Strings over \{0,1,2\} w/ even number of 2’s and mod 3 sum 0
Strings over \{0,1,2\} w/ even number of 2's OR mod 3 sum 0
The set of binary strings with a 1 in the 3rd position from the start
The set of binary strings with a 1 in the 3\textsuperscript{rd} position from the start
The set of binary strings with a 1 in the 3\textordmasco rth position from the end
3 bit shift register  
“Remember the last three bits”
The set of binary strings with a 1 in the 3\textsuperscript{rd} position from the end.
The set of binary strings with a 1 in the 3\textsuperscript{rd} position from the end
The beginning versus the end
State Minimization

• Many different FSMs (DFAs) for the same problem

• Take a given FSM and try to reduce its state set by combining states
  – Algorithm will always produce the unique minimal equivalent machine (up to renaming of states) but we won’t prove this
State Minimization Algorithm

• Put states into groups

• Try to find groups that can be collapsed into one state
  – states can keep track of information that isn’t necessary to determine whether to accept or reject

• Group states together until we can prove that collapsing them can change the accept/reject result
  – find a specific string \( x \) such that:
    - starting from state A, following edges according to \( x \) ends in accept
    - starting from state B, following edges according to \( x \) ends in reject
  – (algorithm below could be modified to show these strings)
State Minimization Algorithm

1. Put states into groups based on their outputs (whether they accept or reject)
State Minimization Algorithm

1. Put states into groups based on their outputs (whether they accept or reject)

2. Repeat the following until no change happens
   a. If there is a symbol $s$ so that not all states in a group $G$ agree on which group $s$ leads to, split $G$ into smaller groups based on which group the states go to on $s$

3. Finally, convert groups to states
State Minimization Example

Put states into groups based on their outputs (or whether they accept or reject)
State Minimization Example

Put states into groups based on their outputs (or whether they accept or reject)
### State Minimization Example

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State 0</th>
<th>Next State 1</th>
<th>Next State 2</th>
<th>Next State 3</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>S0</td>
<td>S3</td>
<td>S1</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S1</td>
<td>S3</td>
<td>S2</td>
<td>S4</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>S1</td>
<td>S0</td>
<td>S4</td>
<td>S5</td>
<td>1</td>
</tr>
<tr>
<td>S4</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S5</td>
<td>0</td>
</tr>
<tr>
<td>S5</td>
<td>S1</td>
<td>S4</td>
<td>S0</td>
<td>S5</td>
<td>0</td>
</tr>
</tbody>
</table>

State transition table

Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol $s$ so that not all states in a group $G$ agree on which group $s$ leads to, split $G$ based on which group the states go to on $s$
Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol \( s \) so that not all states in a group \( G \) agree on which group \( s \) leads to, split \( G \) based on which group the states go to on \( s \)
Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol $s$ so that not all states in a group $G$ agree on which group $s$ leads to, split $G$ based on which group the states go to on $s$
State Minimization Example

Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol $s$ so that not all states in a group $G$ agree on which group $s$ leads to, split $G$ based on which group the states go to on $s$
State Minimization Example

Put states into groups based on their outputs (or whether they accept or reject)

If there is a symbol $s$ so that not all states in a group $G$ agree on which group $s$ leads to, split $G$ based on which group the states go to on $s$
State Minimization Example

Finally convert groups to states:

Can combine states S0-S4 and S3-S5.

In table replace all S4 with S0 and all S5 with S3
Minimized Machine

State transition table:

<table>
<thead>
<tr>
<th>present state</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S3</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>S0</td>
<td>S3</td>
<td>S1</td>
<td>S3</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S1</td>
<td>S3</td>
<td>S2</td>
<td>S0</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>S1</td>
<td>S0</td>
<td>S0</td>
<td>S3</td>
<td>0</td>
</tr>
</tbody>
</table>

State transition table:

1. S0 transitions to S1 on input 1.
2. S1 transitions to S3 on input 1.
3. S2 transitions to S1 on input 1.
4. S3 transitions to S1 on input 1.
A Simpler Minimization Example

The set of all binary strings with # of 1’s ≡ # of 0’s (mod 2).
A Simpler Minimization Example

Split states into accept/reject groups

Every symbol causes the DFA to go from one group to the other so neither group needs to be split
The set of all binary strings with \# of 1’s \equiv \# of 0’s (mod 2).

= The set of all binary strings with even length.
Nondeterministic Finite Automata (NFA)

- Graph with start state, final states, edges labeled by symbols (like DFA) but
  - Not required to have exactly 1 edge out of each state labeled by each symbol—can have 0 or >1
  - Also can have edges labeled by empty string $\varepsilon$

- **Definition:** $x$ is in the language recognized by an NFA if and only if some valid execution of the machine gets to an accept state

![Diagram of NFA](image-url)
Consider This NFA

What language does this NFA accept?
Consider This NFA

What language does this NFA accept?

\[ 10(10)^* \cup 111(0 \cup 1)^* \]
NFA $\varepsilon$-moves

- Start state: $s_0$
- Terminal states: $s_0, s_1, t_1, t_2$
- Transitions:
  - $s_0 \xrightarrow{0,1} s_1$
  - $s_0 \xrightarrow{\varepsilon} s_1$
  - $s_0 \xrightarrow{\varepsilon} t_0$
  - $s_1 \xrightarrow{2} t_1$
  - $s_1 \xrightarrow{0,1} s_1$
  - $t_0 \xrightarrow{0} t_0$
  - $t_0 \xrightarrow{1} t_1$
  - $t_0 \xrightarrow{2} t_2$
  - $t_1 \xrightarrow{1} t_2$
  - $t_1 \xrightarrow{2} t_1$
  - $t_2 \xrightarrow{0} t_2$
  - $t_2 \xrightarrow{1} t_1$

Initial state: $q$
Accepted states: $t_0, t_1, t_2$
NFA $\varepsilon$-moves

Strings over \{0,1,2\} w/\ even \ # \ of \ 2's \ OR \ sum \ to \ 0 \ mod \ 3
NFA for set of binary strings with a 1 in the 3rd position from the end
NFA for set of binary strings with a 1 in the 3rd position from the end
Compare with the smallest DFA
Summary of NFAs

• Generalization of DFAs
  – drop two restrictions of DFAs
  – every DFA is an NFA

• Seem to be more powerful
  – designing is easier than with DFAs

• Seem related to regular expressions