Lecture 25

◆ Logistics
  - HW8 posted today, due 12/5
  - Lab9 this week
  - No Class for the rest of the week!

◆ Last lecture
  - Robot ant in maze
  - Started on FSM simplification a little bit

◆ Today
  - More on FSM simplification

The “WHY” slide

◆ FSM minimization
  - It is best to minimize FSM before expressing it as a logic circuit. As you saw in the ant robot example, minimization step is about looking for some patterns and merging states. There are systematic ways to do this (rather than the way we’d done it for the ant example) and we will learn them here.
Two Methods for FSM Minimization

- **Row matching**
  - Easier to do by hand
  - Misses minimization opportunities

- **Implication table**
  - Guaranteed to find the most reduced FSM
  - More complicated algorithm (but still relatively easy to write a program to do it)

Simple row matching does not guarantee most reduced state machine

<table>
<thead>
<tr>
<th>Present State</th>
<th>X=0</th>
<th>X=1</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>S₀</td>
<td>S₁</td>
<td>0</td>
</tr>
<tr>
<td>S₁</td>
<td>S₁</td>
<td>S₂</td>
<td>1</td>
</tr>
<tr>
<td>S₂</td>
<td>S₂</td>
<td>S₁</td>
<td>0</td>
</tr>
</tbody>
</table>

CSE370, Lecture 25
The Implication chart method

- Here’s a slightly funkier FSM as an example

Step 1: Draw the table
Step 2: Consider the outputs

Step 3: Add transition pairs
Step 3: Add transition pairs

Step 4 (repeated): Consider transitions
Final reduced FSM

Odd parity checker revisited

<table>
<thead>
<tr>
<th>Present State</th>
<th>X=0</th>
<th>X=1</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S₀</td>
<td>S₂</td>
<td>S₁</td>
<td>0</td>
</tr>
<tr>
<td>S₁</td>
<td>S₁</td>
<td>S₂</td>
<td>1</td>
</tr>
<tr>
<td>S₂</td>
<td>S₂</td>
<td>S₁</td>
<td>0</td>
</tr>
</tbody>
</table>
More complex state minimization

- Multiple input example

Minimized FSM

- Implication chart method
  - cross out incompatible states based on outputs
  - then cross out more cells if indexed chart entries are already crossed out
Minimized FSM

<table>
<thead>
<tr>
<th>state</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</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>S4</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>0</td>
</tr>
<tr>
<td>S4</td>
<td>S0</td>
<td>S1</td>
<td>S2</td>
<td>S5</td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>S1</td>
<td>S4</td>
<td>S0</td>
<td>S5</td>
<td>0</td>
</tr>
</tbody>
</table>

Minimizing incompletely specified FSMs

- Equivalence of states is transitive when machine is fully specified.
- But it's not transitive when don't cares are present.

E.g., state output:

- S0 X 0: S1 is compatible with both S0 and S2.
- S1 1 X: but S0 and S2 are incompatible.
- S2 X 1:

- Hard to determining best grouping of states to yield the smallest number of final states.
Minimizing FSMs isn’t always good

- Two FSMs for 0->1 edge detection

Minimal state diagram -> not necessarily best circuit
A little perspective

- These kinds of optimizations are what CAD (Computer Aided Design) / EDA (Electronic Design Automation) is all about.
- The interesting problems are almost always computationally intractable to solve optimally.
- People really care about the automation of the design of billion-transistor chips.