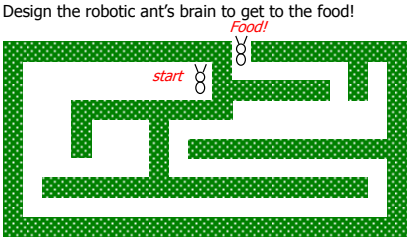
Lecture 21

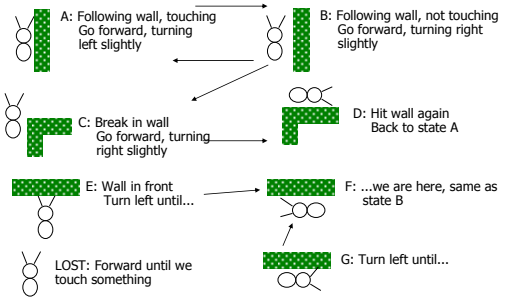
- Logistics
  - HW7 due Wednesday
  - Lab 9 this week and next week
- Last lecture
  - A bigger FSM example: Hungry Robot Ant in Maze --- Started
- Today
  - Continue on the same example
  - FSM simplification

Robotic ant in a maze

- Robot ant, physical maze
  - Maze has no islands
  - Corridors are wider than ant
  - Design the robotic ant's brain to get to the food!



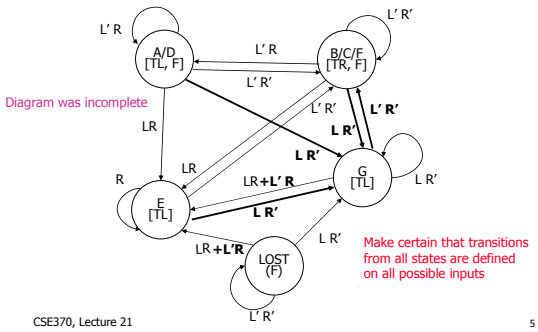
Robot Ant behavior



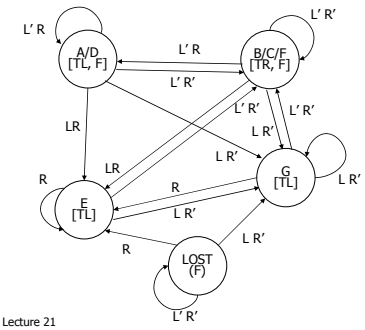
Notations

- Sensors on L and R antennae
  - Sensor = "1" if touching wall; "0" if not touching wall
    - L'R' ≡ no wall
    - L'R ≡ wall on right
    - LR' ≡ wall on left
    - LR ≡ wall in front
- Movement
  - F ≡ forward one step
  - TL ≡ turn left slightly
  - TR ≡ turn right slightly

1. State Diagram from last class



1. State Diagram before Minimization



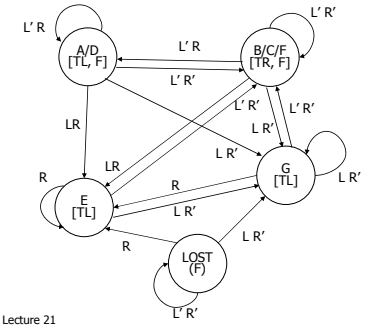
2. State Transition Table

- Using symbolic states and outputs

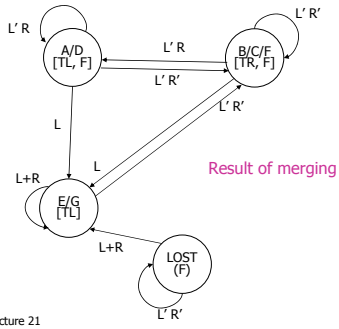
state	L	R	next state	outputs
LOST	0	0	LOST	F
LOST	X	1	E	F
LOST	1	0	G	F
E	0	0	B	TL
E	X	1	E	TL
E	1	0	G	TL
G	0	0	B	TL
G	X	1	E	TL
G	1	0	G	TL
A	0	0	B	TL, F
A	0	1	A	TL, F
A	1	0	G	TL, F
A	1	1	E	TL, F
B	0	0	B	TR, F
B	0	1	A	TR, F
B	1	0	G	TR, F
B	1	1	E	TR, F

Same outputs and transitions ⇒ can merge states

1. Before Minimization



3. State Minimization



Result of merging states E and G

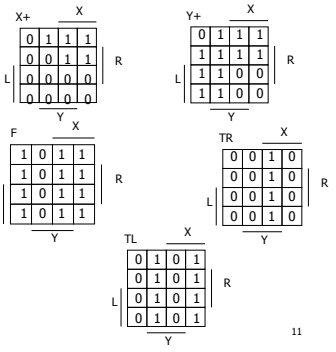
4. State encoding

state	L	R	next state	outputs
LOST	0	0	LOST	F
LOST	X	1	E	F
LOST	1	0	E	F
E	0	0	B	TL
E	X	1	E	TL
E	1	0	E	TL
A	0	0	B	TL, F
A	0	1	A	TL, F
A	1	X	E	TL, F
B	0	0	B	TR, F
B	0	1	A	TR, F
B	1	X	E	TR, F

state X	state Y	L	R	next state X+	next state Y+	outputs F	outputs TR	outputs TL
0	0	0	0	0	0	1	0	0
0	0	X	1	0	1	1	0	0
0	0	1	0	0	1	1	0	0
0	1	0	0	1	1	0	0	1
0	1	X	1	0	1	0	0	1
0	1	1	0	0	1	0	0	1
1	0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	0	1
1	0	1	X	0	1	1	0	1
1	1	0	0	1	1	1	1	0
1	1	0	1	1	0	1	1	0
1	1	1	X	0	1	1	1	0

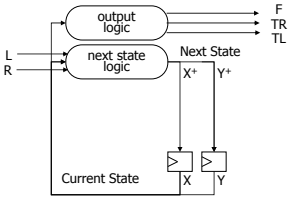
5. Next state logic minimization

state X	state Y	L	R	next state X+	next state Y+	outputs F	outputs TR	outputs TL
0	0	0	0	0	0	1	0	0
0	0	X	1	0	1	1	0	0
0	0	1	0	0	1	1	0	0
0	1	0	0	1	1	0	0	1
0	1	X	1	0	1	0	0	1
0	1	1	0	0	1	0	0	1
1	0	0	0	1	1	1	0	1
1	0	0	1	1	0	1	0	1
1	0	1	X	0	1	1	0	1
1	1	0	0	1	1	1	1	0
1	1	0	1	1	0	1	1	0
1	1	1	X	0	1	1	1	0



6. Circuit Implementation

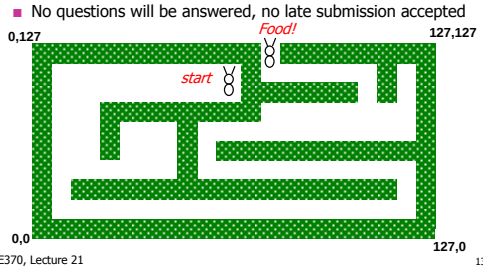
- Outputs are a function of the current state only - Moore machine



Extra credit (worth 10pts equivalent in a midterm)

Design the robotic ant's brain with virtual maze representation

- Due last day in class, Friday, March 13; printouts only
- Graded on clarity and completeness of explanation
- No questions will be answered, no late submission accepted



The maze

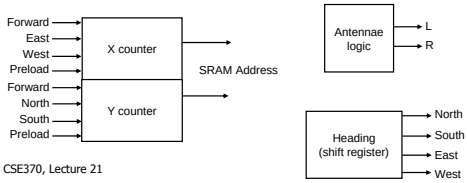
- Virtual maze
  - 128 × 128 grid
    - Stored in memory
    - 16384 8-bit words
  - *YX* is maze addresses
    - *X* is the ant's horizontal position (7 bits)
    - *Y* is the ant's vertical position (7 bits)
  - Each memory location says
    - 00000001 ≡ No wall
    - 00000010 ≡ North wall
    - 00000100 ≡ West wall
    - 00001000 ≡ South wall
    - 00010000 ≡ East wall
    - 00100000 ≡ Exit

Can have multiple walls
Example: 00001100
⇒ Walls on South and East

Design of different components

Predesigned:

Submit the designs for:



Recommendations

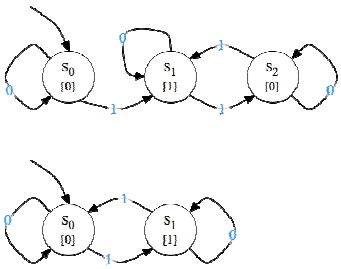
- Memory controller
  - Move horizontally: Increment or decrement *X*
  - Move vertically: Increment or decrement *Y*
- Shift register for heading
  - N: 0001
  - W: 0010
  - S: 0100
  - E: 1000
  - Rotate right when ant turns right
  - Rotate left when ant turns left
- Combinational logic for antennae logic

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.

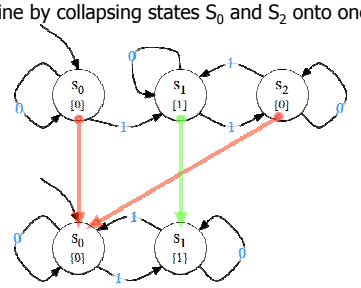
FSM Minimization

- Two simple FSMs for odd parity checking



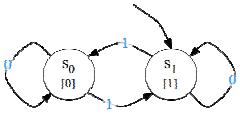
Collapsing States

- We can make the top machine match the bottom machine by collapsing states S_0 and S_2 onto one state



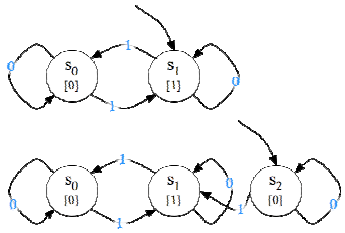
FSM Design on the Cheap

- Let's say we start with this FSM for even parity checking



FSM Design on the Cheap

- Now an enterprising engineer comes along and says, "Hey, we can turn our even parity checker into an odd parity checker by just adding one state."



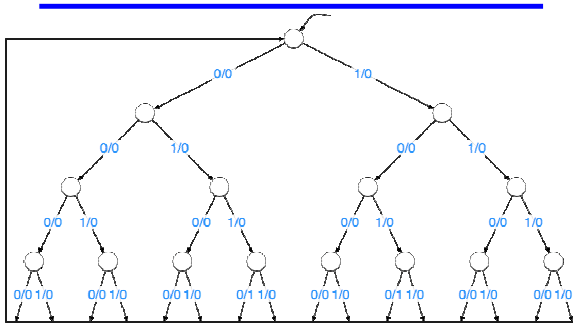
Two Methods for FSM Minimization

- 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)

A simple problem

- Design a Mealy machine with a single bit input and a single bit output. The machine should output a 0, except once every four cycles, if the previous four inputs matched one of two patterns (0110, 1010)
- Example input/output trace:
  in: 0010 0110 1100 1010 0011 …
  out: 0000 0001 0000 0001 0000 …

… and a simple solution



Find matching rows

		Next State		Output	
Input Sequence	Present State	X=0	X=1	X=0	X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	S_{10}	0	0
10	S_5	S_{11}	S_{12}	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011	S_{10}	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
101	S_{12}	S_0	S_0	1	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

Merge the matching rows

		Next State		Output	
Input Sequence	Present State	X=0	X=1	X=0	X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	S_7	S_8	0	0
01	S_4	S_9	$S_{10'}$	0	0
10	S_5	S_{11}	$S_{10'}$	0	0
11	S_6	S_{13}	S_{14}	0	0
000	S_7	S_0	S_0	0	0
001	S_8	S_0	S_0	0	0
010	S_9	S_0	S_0	0	0
011 or 101	$S_{10'}$	S_0	S_0	1	0
100	S_{11}	S_0	S_0	0	0
110	S_{13}	S_0	S_0	0	0
111	S_{14}	S_0	S_0	0	0

Merge until no more rows match

		Next State		Output	
Input Sequence	Present State	X=0	X=1	X=0	X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	$S_{7'}$	$S_{7'}$	0	0
01	S_4	$S_{7'}$	$S_{10'}$	0	0
10	S_5	$S_{7'}$	$S_{10'}$	0	0
11	S_6	$S_{7'}$	$S_{7'}$	0	0
Not (011 or 101)	$S_{7'}$	S_0	S_0	0	0
011 or 101	$S_{10'}$	S_0	S_0	1	0

The final state transition table

		Next State		Output	
Input Sequence	Present State	X=0	X=1	X=0	X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	$S_{3'}$	$S_{4'}$	0	0
1	S_2	$S_{4'}$	$S_{3'}$	0	0
00 or 11	$S_{3'}$	$S_{7'}$	$S_{7'}$	0	0
01 or 10	$S_{4'}$	$S_{7'}$	$S_{10'}$	0	0
Not (011 or 101)	$S_{7'}$	S_0	S_0	0	0
011 or 101	$S_{10'}$	S_0	S_0	1	0

A more efficient solution