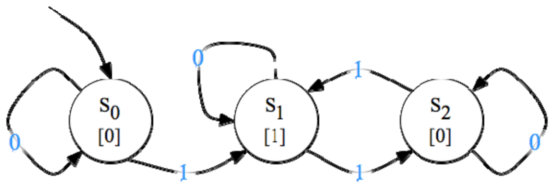
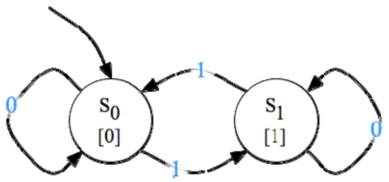
Lecture 21

- Logistics
  - HW8 due on Friday, HW9 due a week from today (last one)
  - Lab --- make sure you are done before the end of next week.
  - Midterm 2: mean 74, median 75, std 15.
- Last lecture
  - Robot ant in maze
  - Started on FSM simplification a little bit
- Today
  - More on FSM simplification

FSM Minimization

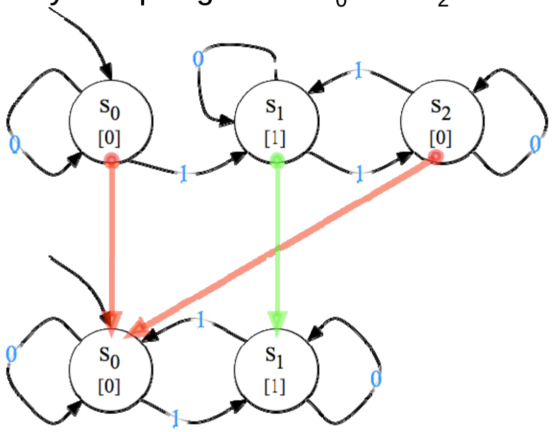
- Two simple FSMs for odd parity checking





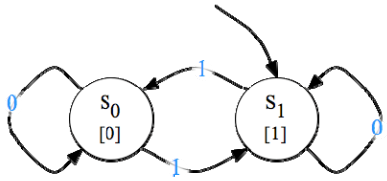
Collapsing States

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



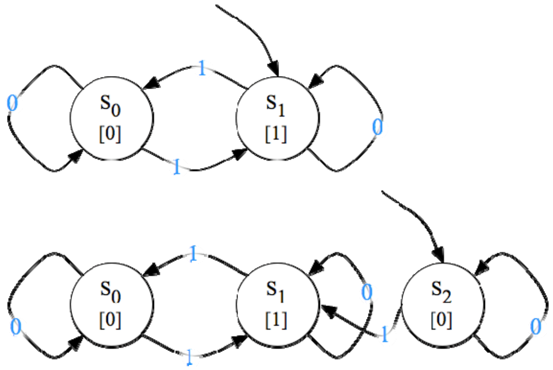
FSM Design on the Cheap

- 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.”



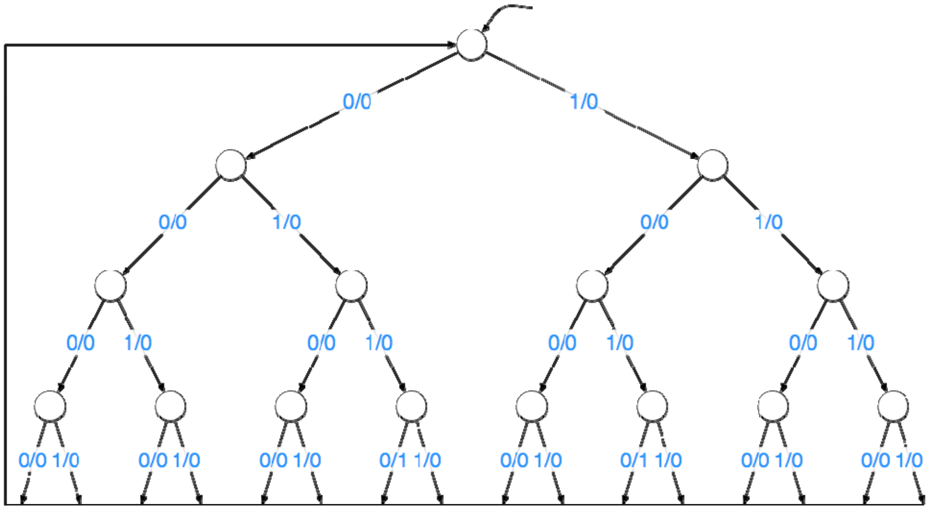
Two Methods for FSM Minimization

- 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

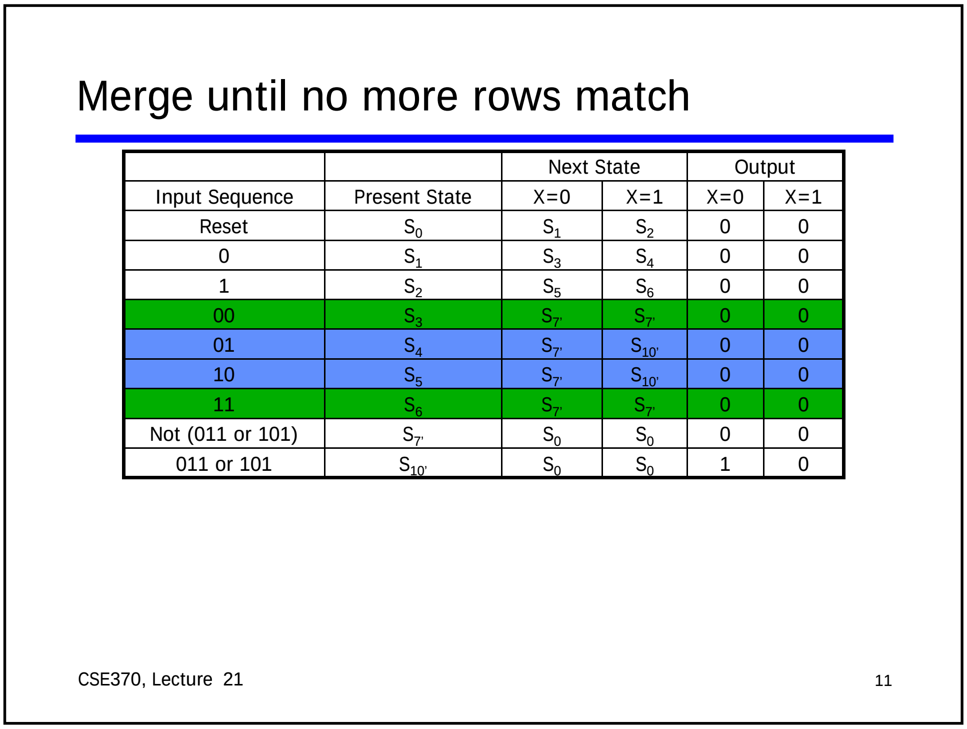


Find matching rows

		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

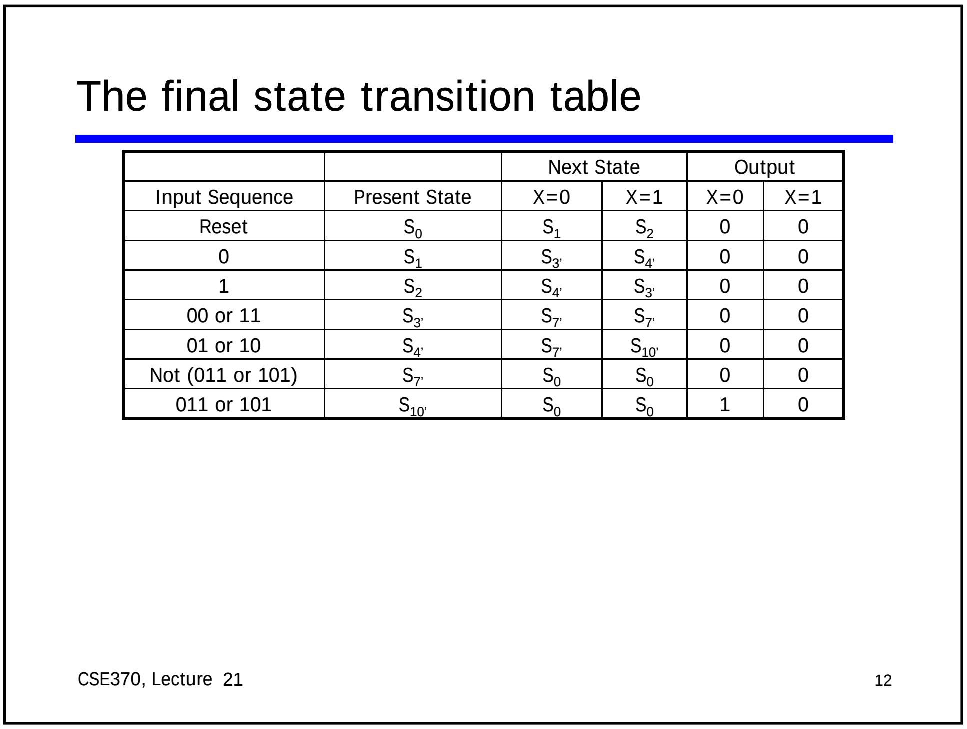
Merge the matching rows

		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

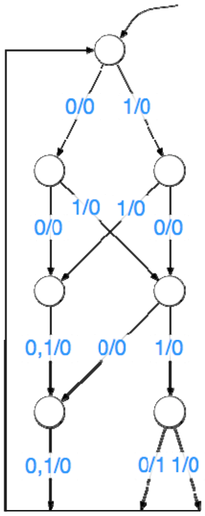
		Next State		Output	
Input Sequence	Present State	X=0	X=1	X=0	X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	S_3	S_4	0	0
1	S_2	S_5	S_6	0	0
00	S_3	$S_{7'}$	$S_{7'}$	0	0
01	S_4	$S_{7'}$	$S_{10'}$	0	0
10	S_5	$S_{7'}$	$S_{10'}$	0	0
11	S_6	$S_{7'}$	$S_{7'}$	0	0
Not (011 or 101)	$S_{7'}$	S_0	S_0	0	0
011 or 101	$S_{10'}$	S_0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



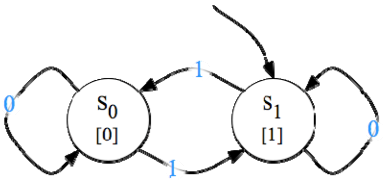
The final state transition table

		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

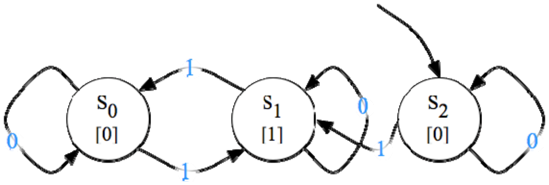
A more efficient solution



Simple row matching does not guarantee most reduced state machine

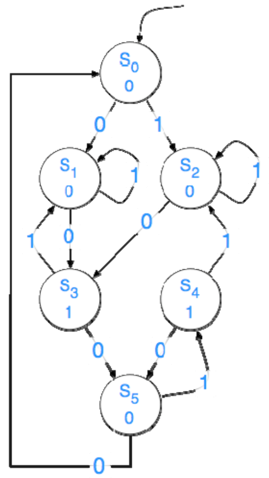


	Next State		
Present State	X=0	X=1	Output
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0

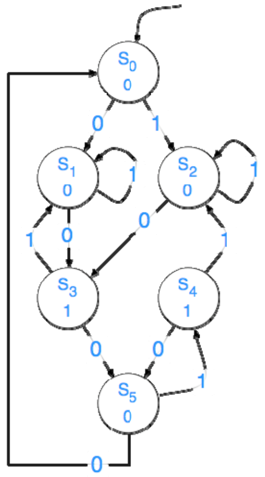


The Implication chart method

- Here's a slightly funkier FSM

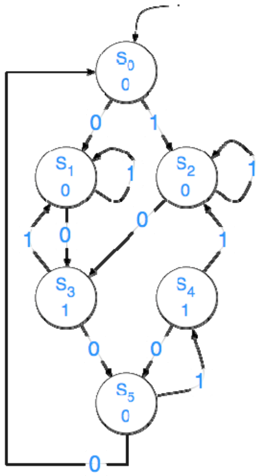


Step 1: Draw the table



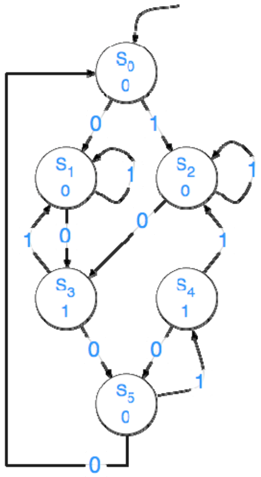
S_0						
S_1						
S_2						
S_3						
S_4						
S_5						
	S_0	S_1	S_2	S_3	S_4	S_5

Step 2: Consider the outputs



S_1					
S_2					
S_3	☒	☒	☒		
S_4	☒	☒	☒		
S_5				☒	☒
	S_0	S_1	S_2	S_3	S_4

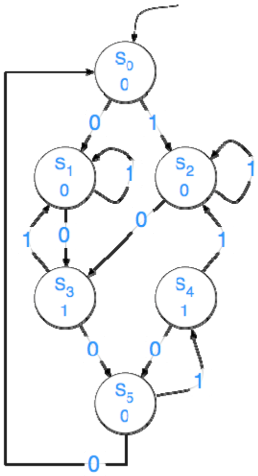
Step 3: Add transition pairs



Implied State Pairs

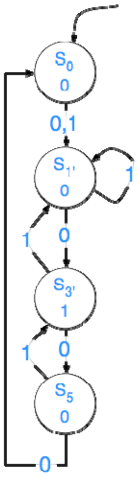
S_1	S_1-S_3 S_2-S_1				
S_2	S_1-S_3 S_2-S_2	S_3-S_3 $S.$-S_2			
S_3	☒	☒	☒		
S_4	☒	☒	☒	S_5-S_5 S_1-S_2	
S_5	S_1-S_0 S_2-S_4	S_3-S_0 $S.$-S_4	S_3-S_0 S_2-S_4	☒	☒
	S_0	S_1	S_2	S_3	S_4

Step 4 (repeated): Consider transitions

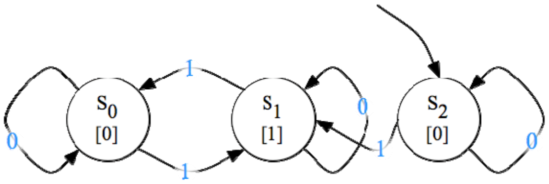


S_1	~~S_1-S_3~~ ~~S_2-S_1~~				
S_2	~~S_1-S_3~~ ~~S_2-S_2~~	S_3-S_3 $S.$-S_2			
S_3	☒	☒	☒		
S_4	☒	☒	☒	S_5-S_5 S_1-S_2	
S_5	~~S_1-S_0~~ ~~S_2-S_4~~	~~S_3-S_0~~ ~~$S.$-S_4~~	~~S_3-S_0~~ ~~S_2-S_4~~	☒	☒
	S_0	S_1	S_2	S_3	S_4

Final reduced FSM



Odd parity checker revisited

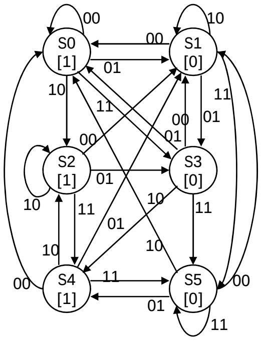


	Next State		
Present State	X=0	X=1	Output
S_0	S_0	S_1	0
S_1	S_1	S_2	1
S_2	S_2	S_1	0

S1	X	
S2	S0-S2 S1–S1	X
	S0	S1

More complex state minimization

- Multiple input example



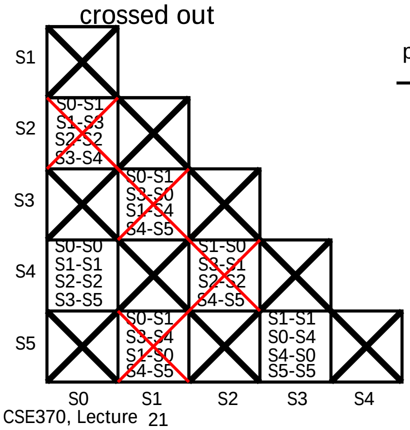
inputs here

present state	next state 00	01	10	11	output
S0	S0	S1	S2	S3	1
S1	S0	S3	S1	S4	0
S2	S1	S3	S2	S4	1
S3	S1	S0	S4	S5	0
S4	S0	S1	S2	S5	1
S5	S1	S4	S0	S5	0

symbolic state transition table

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



	S0	S1	S2	S3	S4
S1					
S2	~~S0-S1~~ ~~S1-S3~~ ~~S2-S2~~ ~~S3-S4~~				
S3		~~S0-S1~~ ~~S3-S0~~ ~~S1-S4~~ ~~S4-S5~~			
S4	S0-S0 S1-S1 S2-S2 S3-S5		~~S1-S0~~ ~~S3-S1~~ ~~S2-S2~~ ~~S4-S5~~		
S5		~~S0-S1~~ ~~S3-S4~~ ~~S1-S0~~ ~~S4-S5~~		S1-S1 S0-S4 S4-S0 S5-S5	

present state	next state 00	01	10	11	output
S0'	S0'	S1	S2	S3'	1
S1	S0'	S3'	S1	S3'	0
S2	S1	S3'	S2	S0'	1
S3'	S1	S0'	S0'	S3'	0

minimized state table
(S0==S4) (S3==S5)

Minimizing incompletely specified FSMs

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

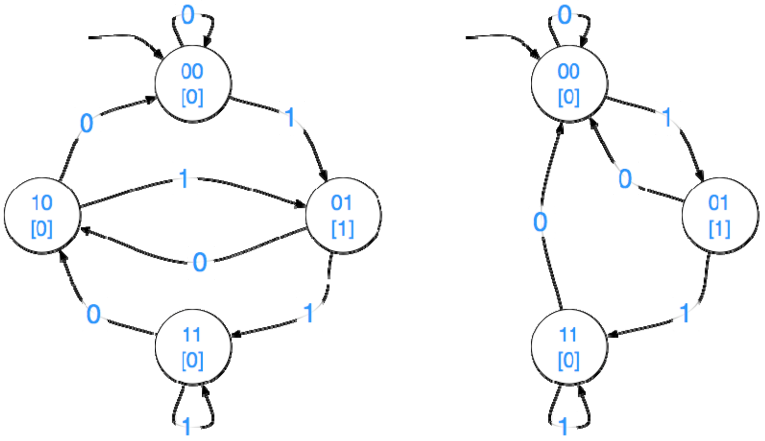
e.g., state output

S0	– 0	S1 is compatible with both S0 and S2
S1	1 –	but S0 and S2 are incompatible
S2	– 1	

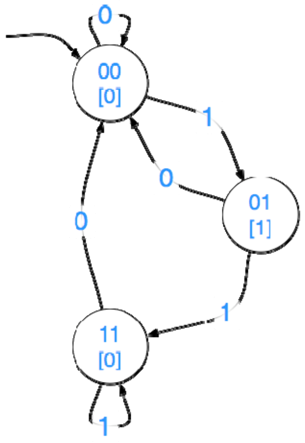
- Hard to determining best grouping of states to yield the smallest number of final states

Minimizing FSMs isn’t always good

- ◆ Two FSMs for 0->1 edge detection



Minimal state diagram -> not necessarily best circuit



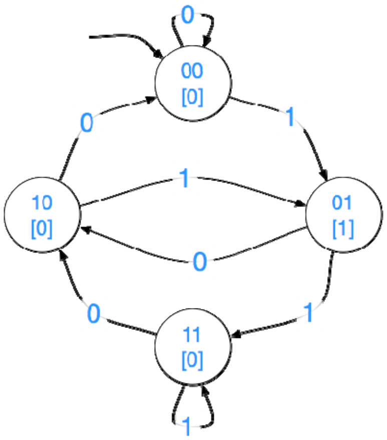
In	Q_1	Q_0	Q_1^+	Q_0^+
0	0	0	0	0
0	0	1	0	0
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	1	1	1
–	1	0	0	0

$Q_1^+ = \text{In} \ (Q_1 \text{ xor } Q_0)$

$Q_0^+ = \text{In} \ Q_1' \, Q_0'$

$\text{Out} = Q_1' \, Q_0$

Minimal state diagram -> not necessarily best circuit



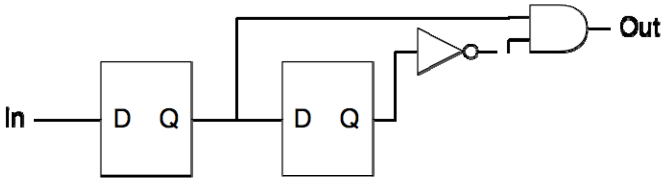
In	Q_1	Q_0	Q_1^+	Q_0^+
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	1
1	0	1	1	1
1	1	0	0	1
1	1	1	1	1

$Q_1^+ = Q_0$

$Q_0^+ = In$

$Out = Q_1' Q_0$

Circuit is simpler for non-simplified FSM



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