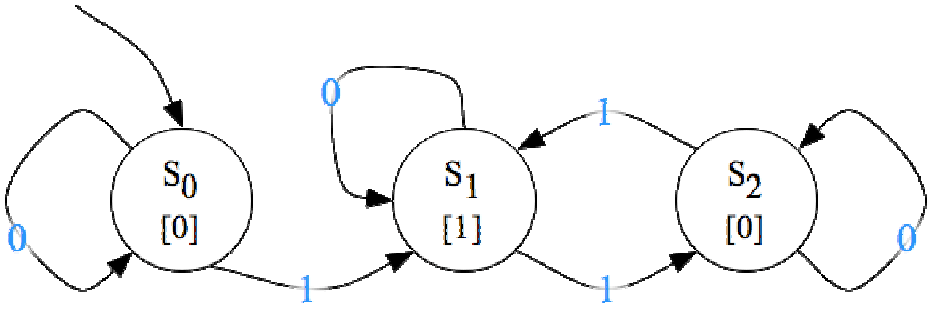
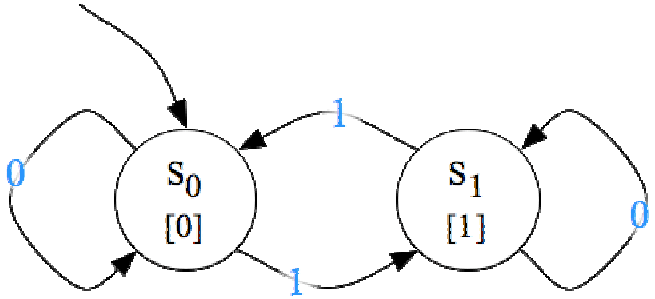
Overview

- Last lecture
  - Ant Brain?
- Today (guest lecture by Benjamin Ylvisaker)
  - State Minimization
    - Row matching method
    - Implication chart method
    - Observations about minimization and efficiency

FSM Minimization

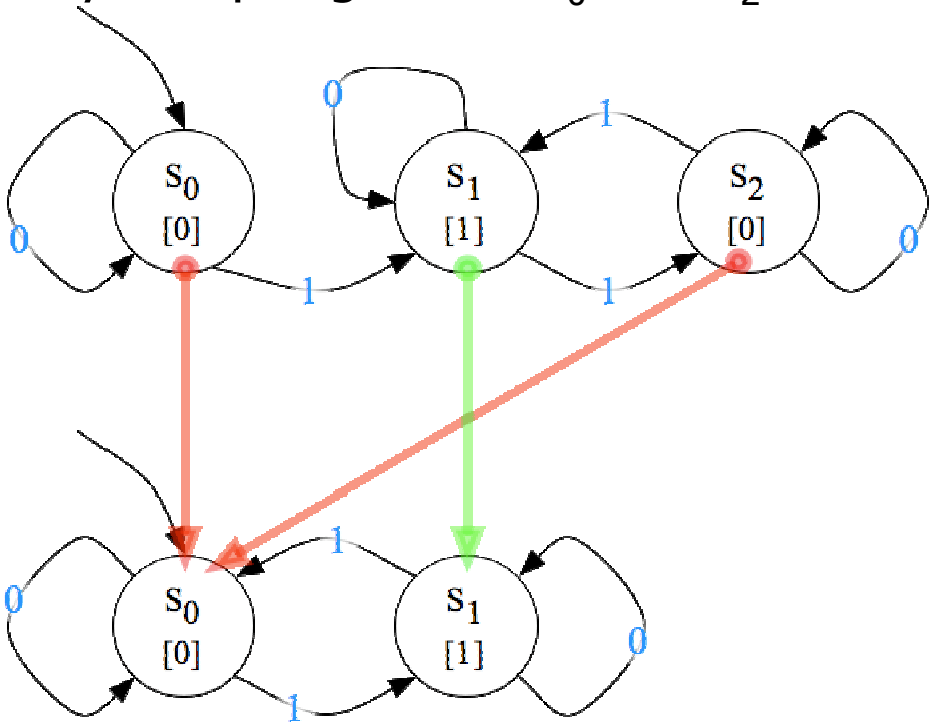
- Two simple FSMs for odd parity checking





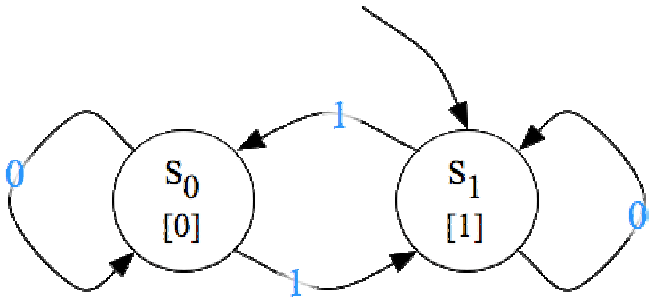
Collapsing States

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



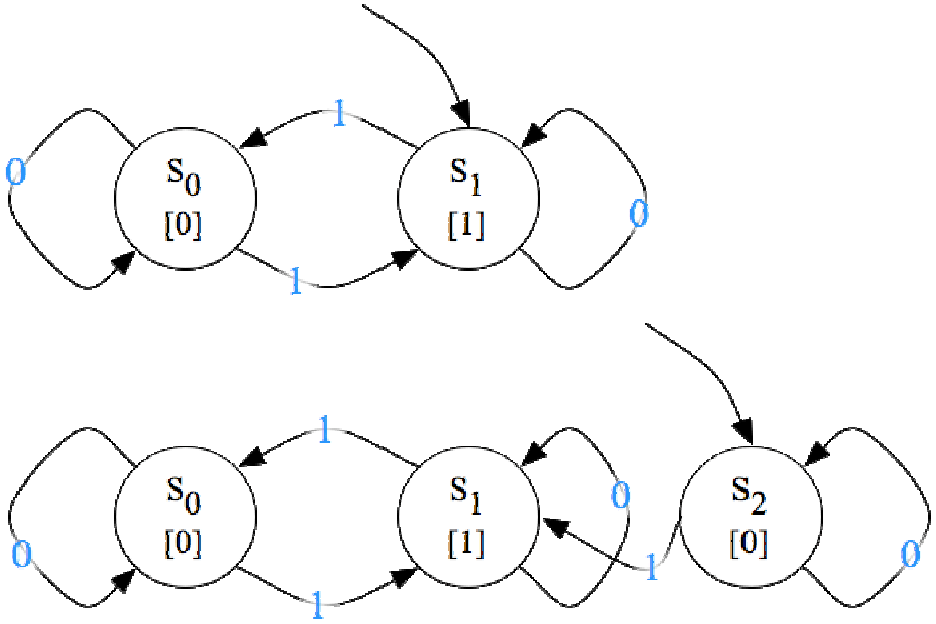
FSM Design on the Cheap

- 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."



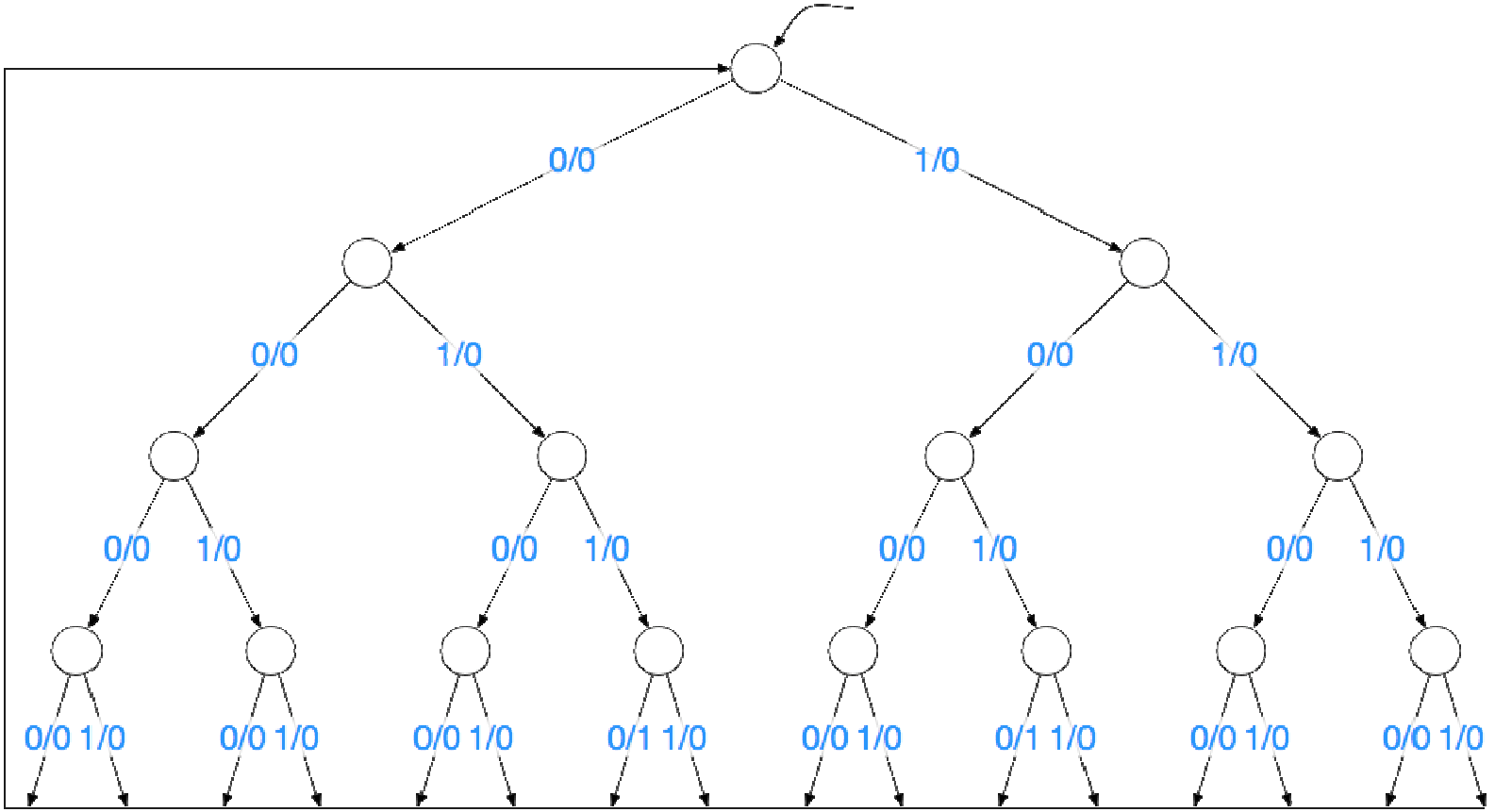
Two Methods for FSM Minimization

- Row matching
  - Easier to do by hand
  - Misses minimization opportunities
  - More pessimistic
- Implication table
  - Guaranteed to find the most reduced FSM
  - More complicated algorithm (but still relatively easy to write a program to do it)
  - More optimistic

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

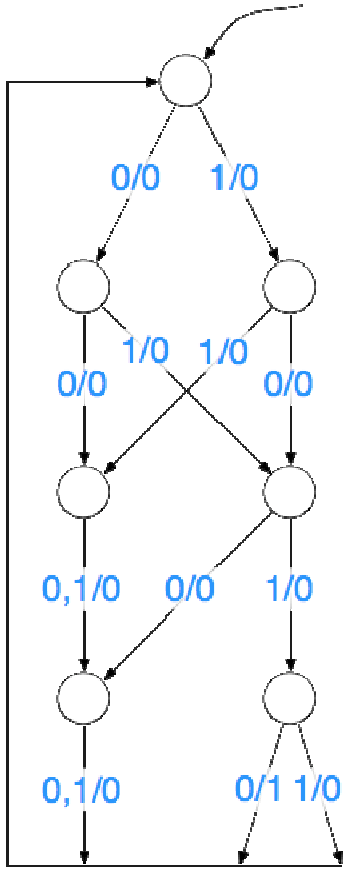
Merge until no more rows match

		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

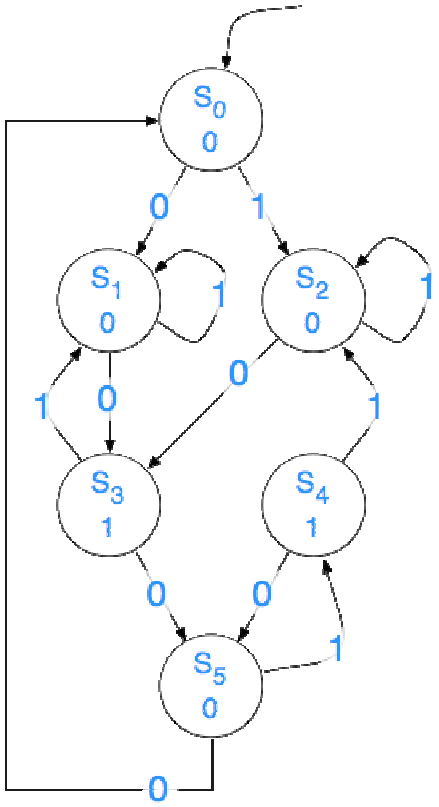
		Next State		Output	
Input Sequence	Present State	X=0	X=1	X=0	X=1
Reset	S_0	S_1	S_2	0	0
0	S_1	$S_{3'}$	$S_{4'}$	0	0
1	S_2	$S_{4'}$	$S_{3'}$	0	0
00 or 11	$S_{3'}$	$S_{7'}$	$S_{7'}$	0	0
01 or 10	$S_{4'}$	$S_{7'}$	$S_{10'}$	0	0
Not (011 or 101)	$S_{7'}$	S_0	S_0	0	0
011 or 101	$S_{10'}$	S_0	S_0	1	0

A more efficient solution

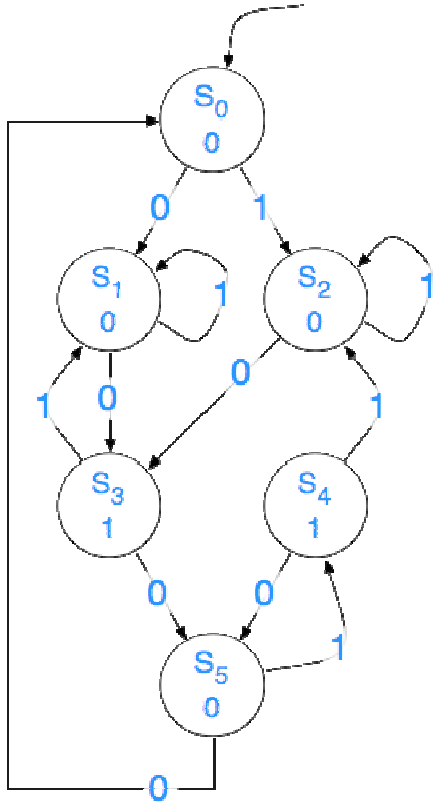


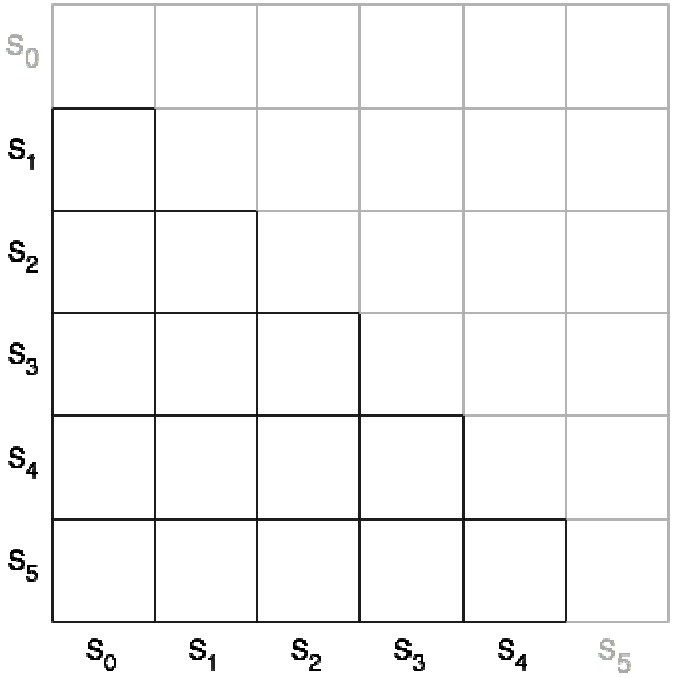
The Implication table method

- Here's a slightly funkier FSM

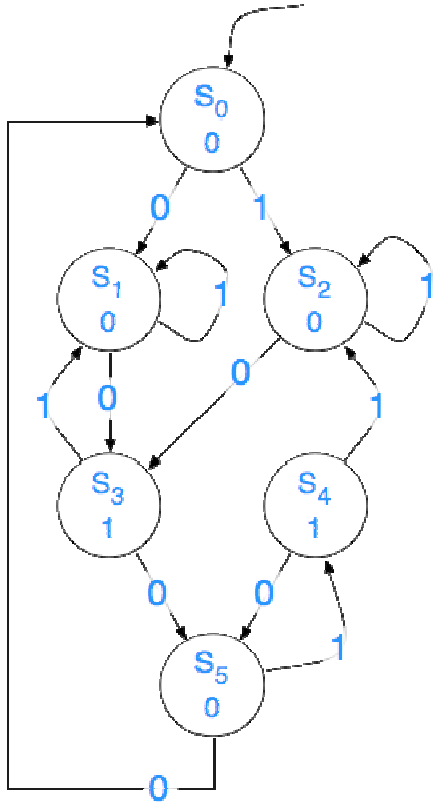


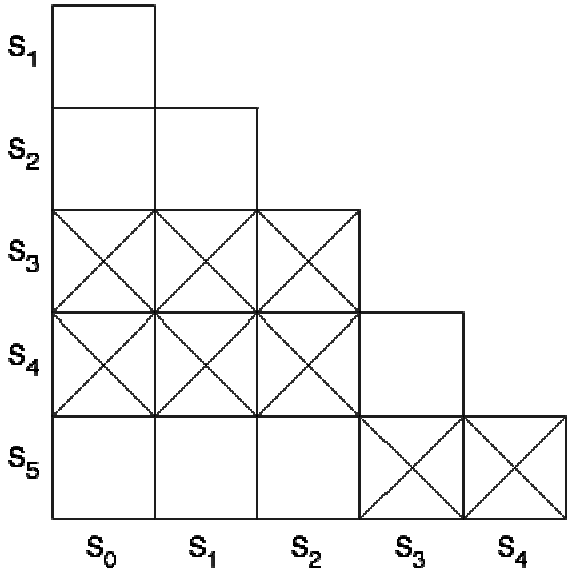
Step 1: Draw the table



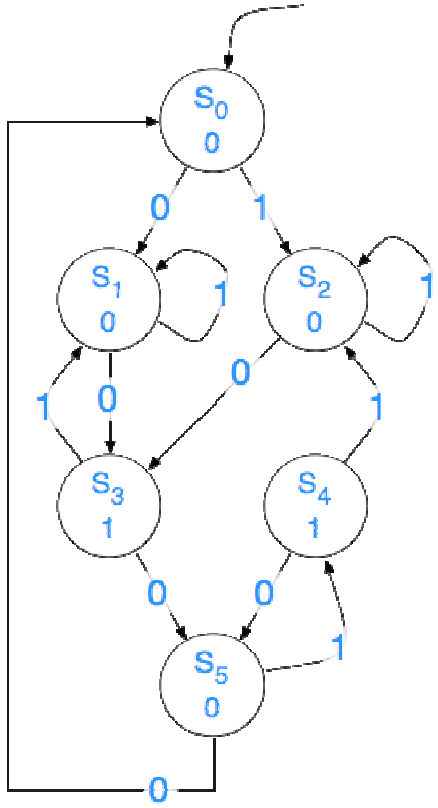


Step 2: Consider the outputs



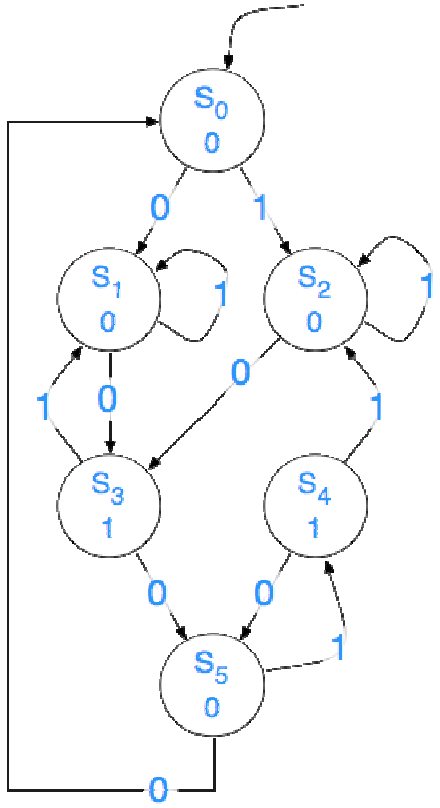


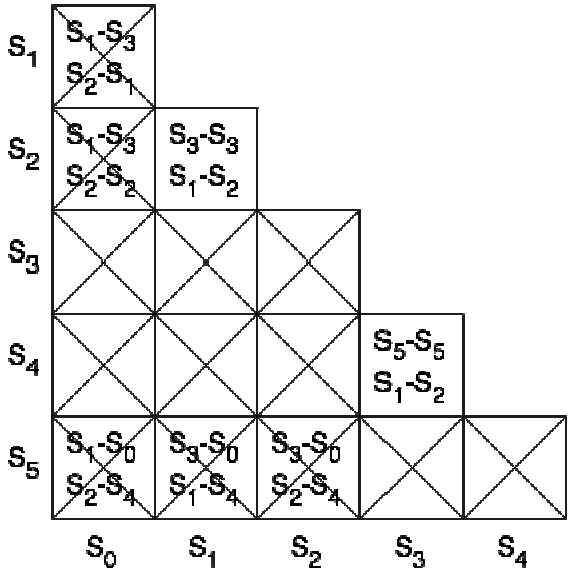
Step 3: Add transition pairs



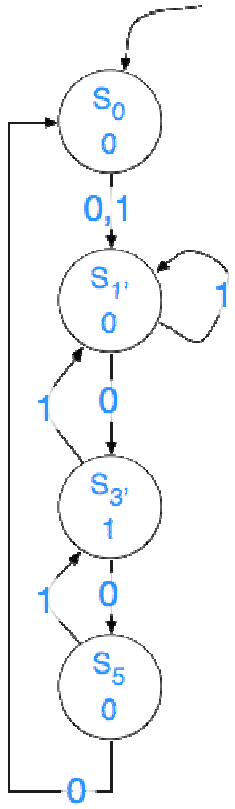
S_1	S_1-S_3 S_2-S_1			
S_2	S_1-S_3 S_2-S_2	S_3-S_3 S_1-S_2		
S_3				
S_4				S_5-S_5 S_1-S_2
S_5	S_1-S_0 S_2-S_4	S_3-S_0 S_1-S_4	S_3-S_0 S_2-S_4		
	S_0	S_1	S_2	S_3	S_4

Step 4 (repeated): Consider transitions



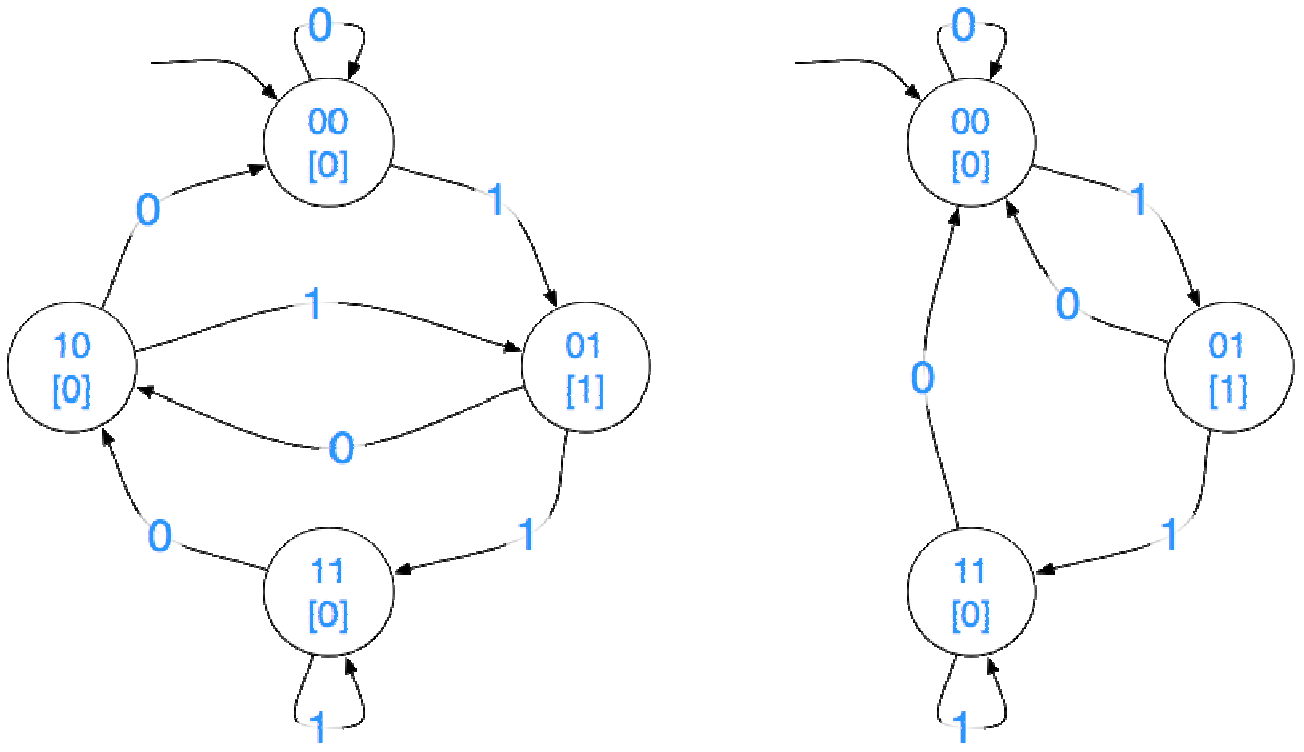


Final reduced FSM

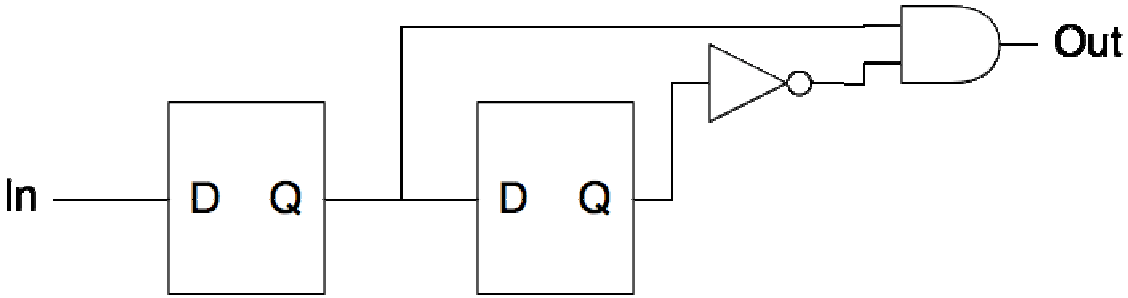


Minimizing FSMs isn't always good

- Two FSMs for 0->1 edge detection



An implementation of the first 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

In the next few lectures

- More about FSMs
- Some methods and tricks for improving them