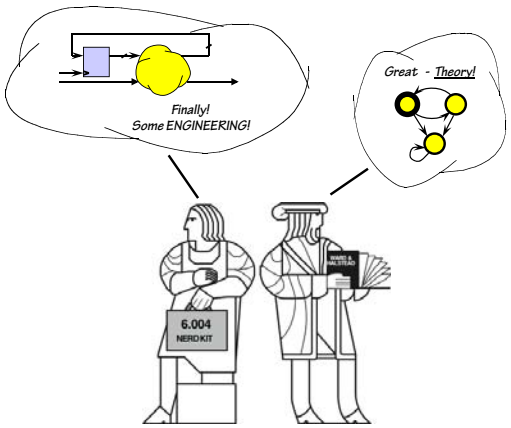
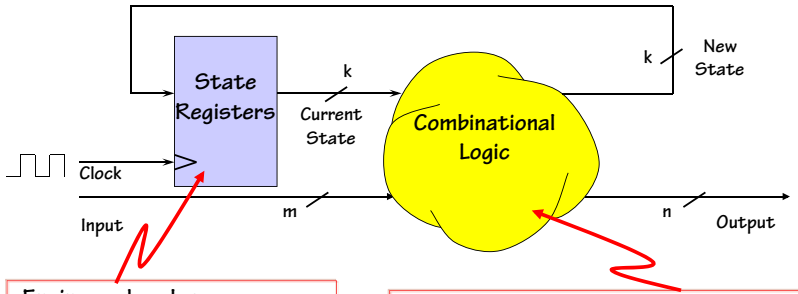
6.004 Computation Structures
Spring 2009

(Synchronous) Finite State Machines



Lab 2 is due Thursday

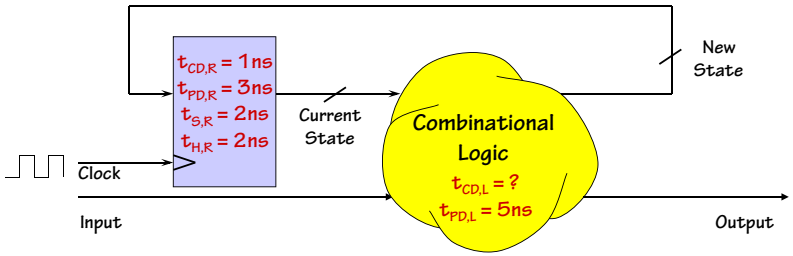
Our New Machine



- Engineered cycles
- Works only if dynamic discipline obeyed
- Remembers k bits for a total of 2^k unique combinations

- Acyclic graph
- Obeys static discipline
- Can be exhaustively enumerated by a truth table of 2^{k+m} rows and k+n output columns

Must Respect Timing Assumptions!



Questions:

- Constraints on T_{CD} for the logic?
- Minimum clock period?
- Setup, Hold times for Inputs?

$t_{CD,R}$ (1 ns) + $t_{CD,L}$(?) > $t_{H,R}$(2 ns)
$t_{CD,L}$ > 1 ns

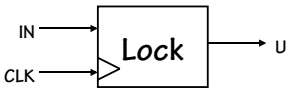
$t_{CLK} > t_{PD,R} + t_{PD,L} + t_{S,R}$ > 10nS

$t_S = t_{PD,L} + t_{S,R}$ = 7 nS
$t_H = t_{H,R} - t_{CD,L}$ = 1 nS

We know how fast it goes… But what can it do?

A simple sequential circuit…

Lets make a digital binary *Combination Lock:*



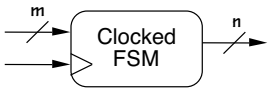
How many registers do I need?

Specification:

- One input ("0" or "1")
- One output ("Unlock" signal)
- UNLOCK is 1 if and only if:

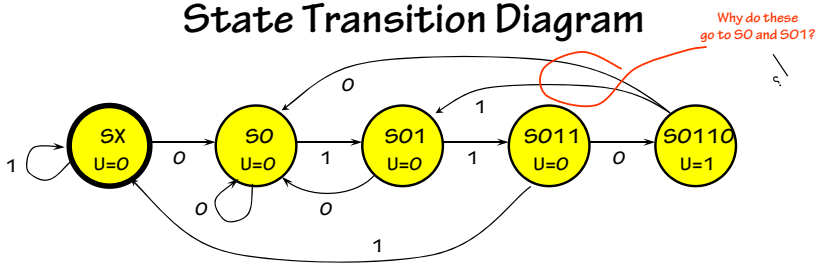
  Last 4 inputs were the "combination": 0110

Abstraction *du jour:* Finite State Machines



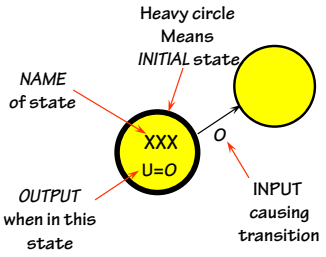
- A FINITE STATE MACHINE has
  - k STATES: $S_1 \ldots S_k$ (one is "initial" state)
  - m INPUTS: $I_1 \ldots I_m$
  - n OUTPUTS: $O_1 \ldots O_n$
  - Transition Rules s'(s, I) for each state s and input I
  - Output Rules Out(s) for each state s

State Transition Diagram

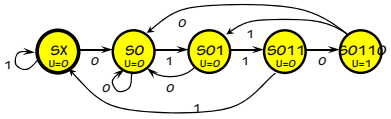


Designing our lock …

- Need an initial state; call it SX.
- Must have a separate state for each step of the proper entry sequence
- Must handle other (erroneous) entries



Yet Another Specification



IN	Current State		Next State	Unlock
0	SX	000	S0	0010
1	SX	000	SX	0000
0	S0	001	S0	0010
1	S0	001	S01	0110
0	S01	011	S0	0010
1	S01	011	S011	0100
0	S011	010	S0110	1000
1	S011	010	SX	0000
0	S0110	100	S0	0011
1	S0110	100	S01	0111

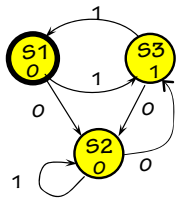
The assignment of codes to states can be arbitrary, however, if you choose them carefully you can greatly reduce your logic requirements.

All state transition diagrams can be described by truth tables…

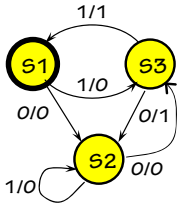
Binary encodings are assigned to each state (a bit of an art)

The truth table can then be simplified using the reduction techniques we learned for combinational logic

Valid State Diagrams



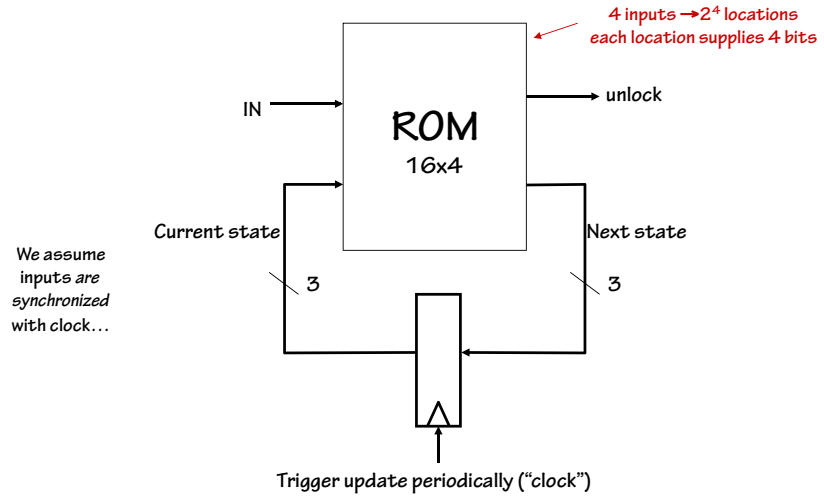
MOORE Machine: Outputs on States



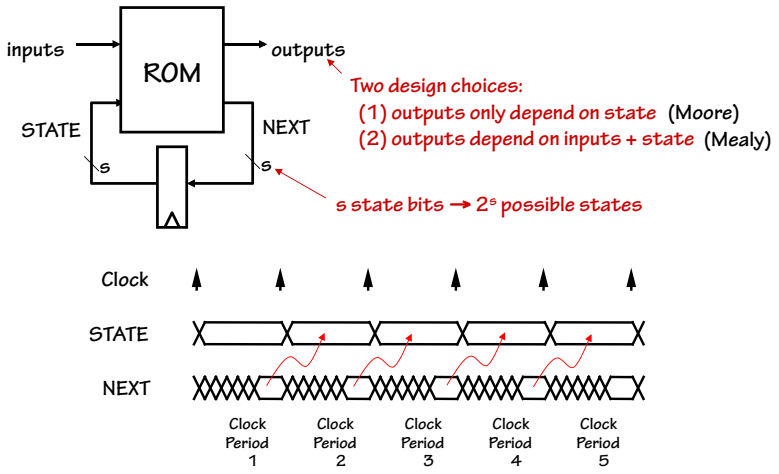
MEALY Machine: Outputs on Transitions

- Arcs leaving a state must be:
- (1) mutually exclusive
  - *can't have two choices for a given input value*
- (2) collectively exhaustive
  - *every state must specify what happens for each possible input combination. "Nothing happens" means arc back to itself.*

Now put it in Hardware!



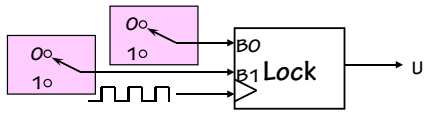
Discrete State, Time



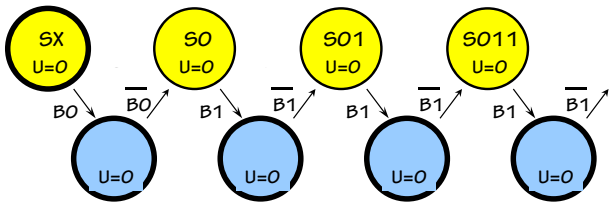
Asynchronous Inputs - I

Our example assumed a single clock transition per input. What if the "button pusher" is unaware of, or not synchronized with, the clock?

What if each button input is an asynchronous 0/1 level? How do we prevent a single button press, e.g., from making several transitions?



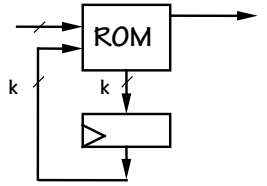
But what About the Dynamic Discipline?



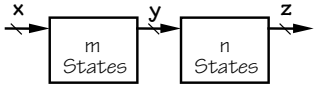
Use *intervening states* to synchronize button presses!

FSM Party Games

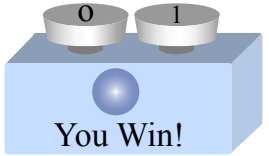
1. What can you say about the number of states?



2. Same question:



3. Here's an FSM. Can you discover its rules?



What's My Transition Diagram?

0=OFF,
1=ON?

"1111" =
Surprise!

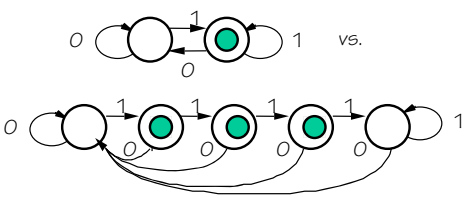


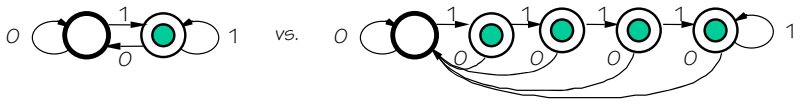
Figure by MIT OpenCourseWare.

- If you know NOTHING about the FSM, you're never sure!
- If you have a BOUND on the number of states, you can discover its behavior:

  K-state FSM: Every (reachable) state can be reached in < k steps.

BUT ... states may be equivalent!

FSM Equivalence



ARE THEY DIFFERENT?

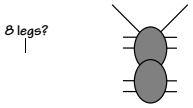
NOT in any practical sense! They are EXTERNALLY INDISTINGUISHABLE, hence interchangeable.

> FSMs *EQUIVALENT* iff every input sequence yields identical output sequences.

ENGINEERING GOAL:

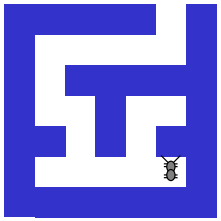
- HAVE an FSM which *works...*
- WANT <u>simplest</u> (ergo cheapest) equivalent FSM.

Lets build an *Ant*



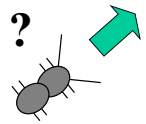
- SENSORS: antennae L and R, each 1 if in contact with something.
- ACTUATORS: Forward Step F, ten-degree turns TL and TR (left, right).

GOAL: Make our ant smart enough to get out of a maze like:

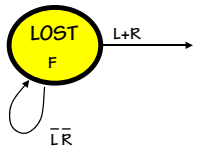


STRATEGY: "Right antenna to the wall"

Lost in space

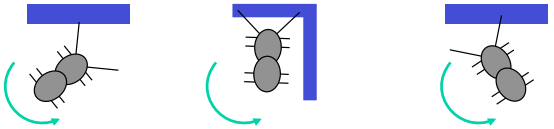


Action: Go forward until we hit something.

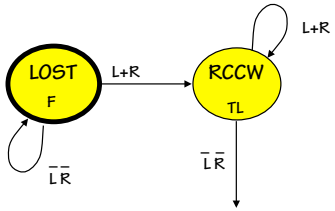


"lost" is the initial state

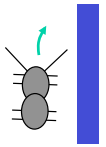
Bonk!



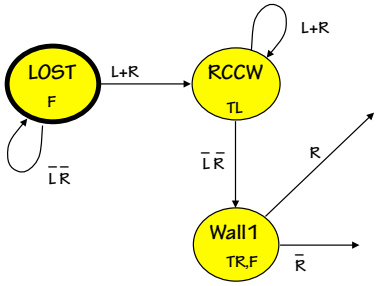
Action: Turn left (CCW) until we don't touch anymore



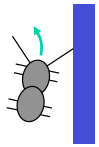
A little to the right…



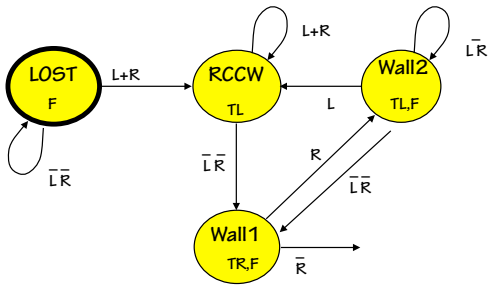
Action: Step and turn right a little, look for wall



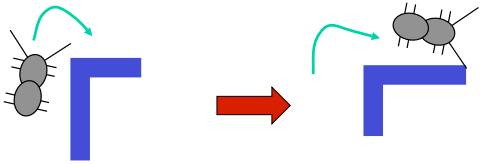
Then a little to the left



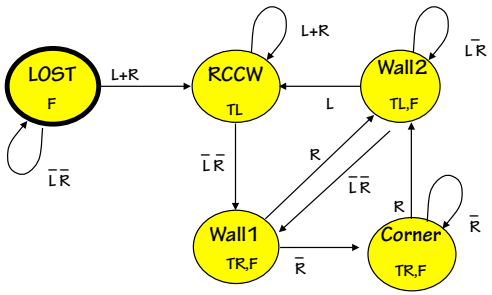
Action: Step and turn left a little, till not touching (again)



Dealing with corners



Action: Step and turn right until we hit perpendicular wall



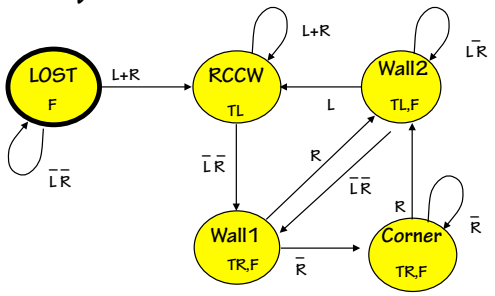
Equivalent State Reduction

Observation: $S_i \equiv S_j$ if

1. States have identical outputs; AND
2. Every input → equivalent states.

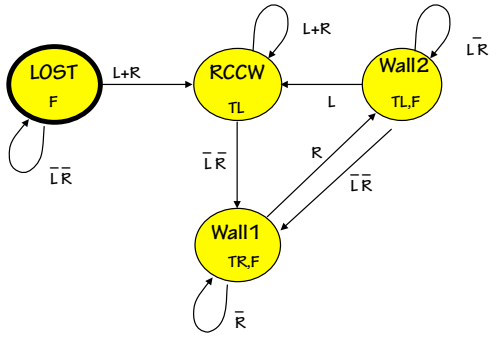
Reduction Strategy:

Find pairs of equivalent states, MERGE them.



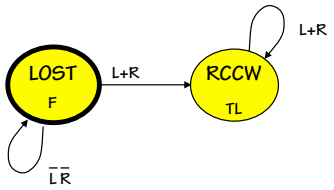
An Evolutionary Step

Merge equivalent states Wall1 and Corner into a single new, combined state.



Behaves exactly as previous (5-state) FSM, but requires half the ROM in its implementation!

Building the Transition Table



S	L	R	S′	TR	TL	F
00	0	0	00	0	0	1
00	1	-	01	0	0	1
00	0	1	01	0	0	1
01	1	-	01	0	1	0
01	0	1	01	0	1	0

Implementation Details

	S	L	R	S′	TR	TL	F
LOST	00	0	0	00	0	0	1
	00	1	-	01	0	0	1
	00	0	1	01	0	0	1
RCCW	01	1	-	01	0	1	0
	01	0	1	01	0	1	0
	01	0	0	10	0	1	0
WALL1	10	-	0	10	1	0	1
	10	-	1	11	1	0	1
	11	1	-	01	0	1	1
WALL2	11	0	0	10	0	1	1
	11	0	1	11	0	1	1

Complete Transition table

S1′

LR \ S_1S_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	1
10	0	0	0	1

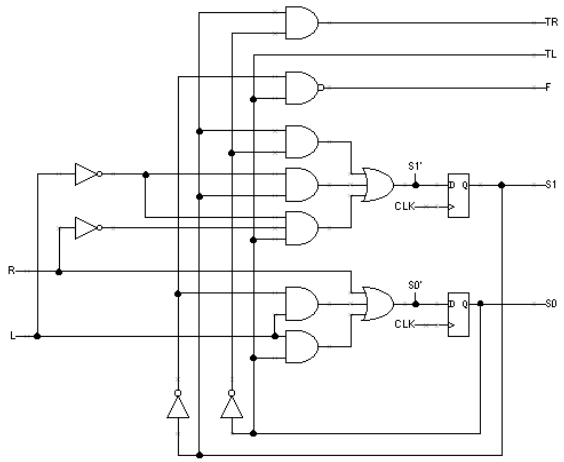
$$S_1' = S_1\overline{S_0} + \overline{L}S_1 + \overline{L}\,\overline{R}S_0$$

S0′

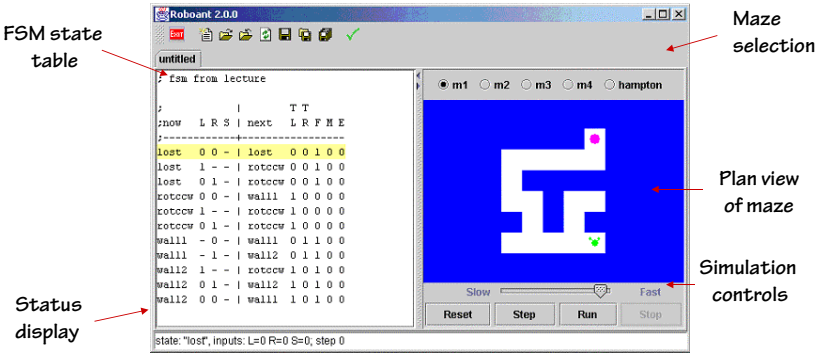
LR \ S_1S_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	0

$$S_0' = R + L\overline{S_1} + LS_0$$

Ant Schematic

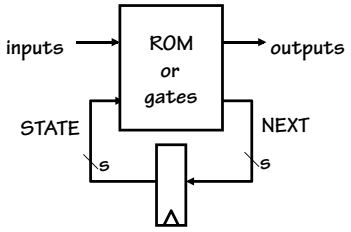


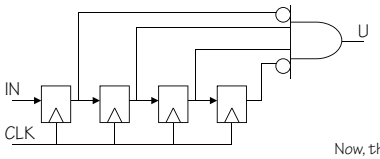
Roboant®



Featuring the new Mark-II ant: can add (M), erase (E), and sense (S) marks along its path.

Housekeeping issues…





1. Initialization? Clear the memory?
2. Unused state encodings?
   - waste ROM (use PLA or gates)
   - what does it mean?
   - can the FSM recover?
3. Choosing encoding for state?
4. Synchronizing input changes with state update?

Now, that's a funny looking state machine

Twisting you Further…

- MORE THAN ANTS:
  Swarming, flocking, and schooling can result from collections of very simple FSMs
- PERHAPS MOST PHYSICS:
  Cellular automata, arrays of simple FSMs, can more accurately model fluilds than numerical solutions to PDEs
- WHAT IF:
  We replaced the ROM with a RAM and have outputs that modify the RAM?

… You'll see FSMs for the rest of your life!