State Machines

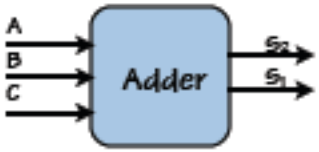
The first problem set is now online!
(Due on Jan. 31st, 2011)

Review

Much of what we call "computation" involves looking answers up in a table

- Reminiscent of those Flash Cards from grade school
- KEY IDEA: Discretize the space of allowable inputs and outputs
- DIGITAL: The ultimate discretization-- binary, only two choices, 0 and 1
- Enumerate all possible input combinations, and "memo"ize the answers (outputs)
- Leads to COMPOSABLE system blocks (i.e. gates, adders, multipliers, etc.)

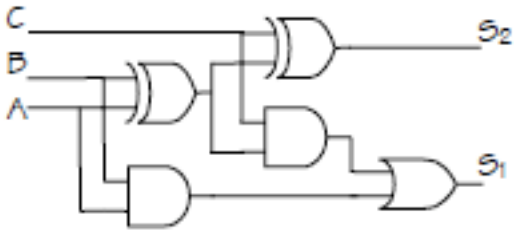
Inputs			Outputs	
A	B	C	S_2	S_1
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

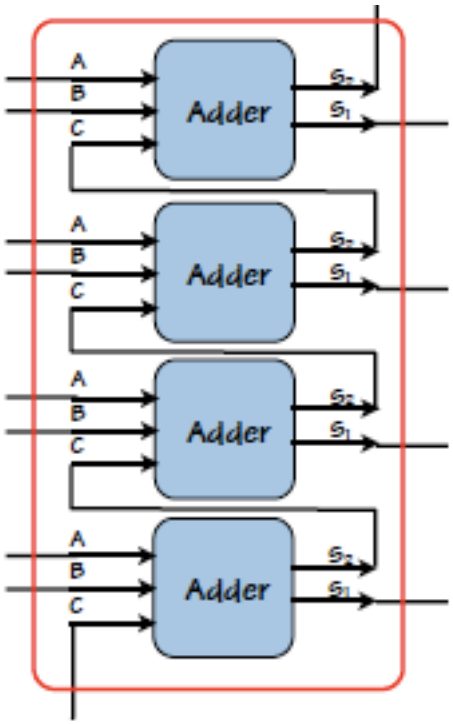


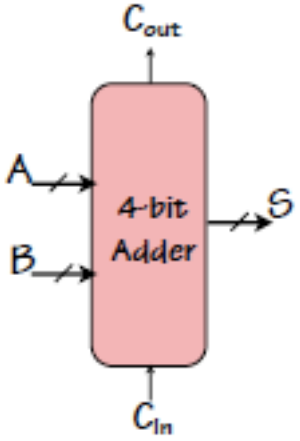
Composition

The key to building large systems...

Boxes around boxes....



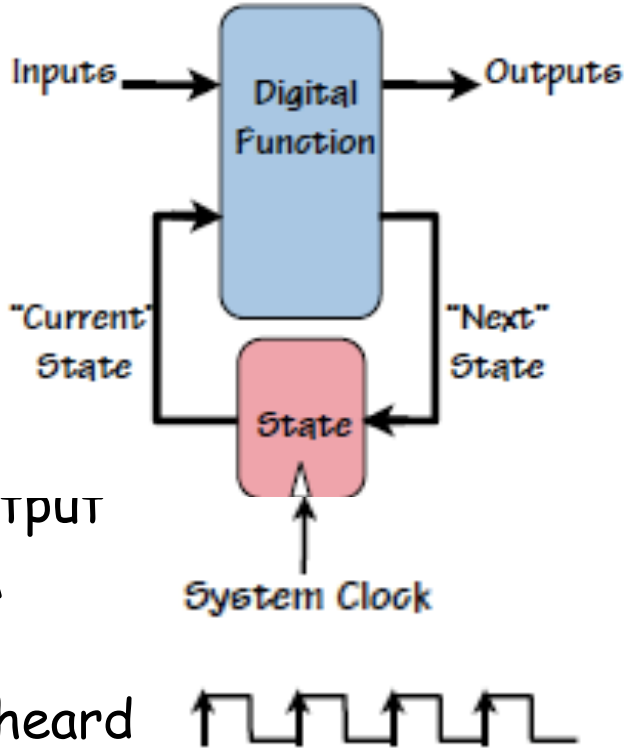




One More Trick With Tables

- Feedback - Feed some of our system's outputs back as inputs
- How do we make sure outputs are settled and ready to go?
- We GATE them with a special STATE block..
  - On the transition of a control signal, it simultaneously copies its inputs to it output
  - This control signal is called the SYSTEM CLOCK...

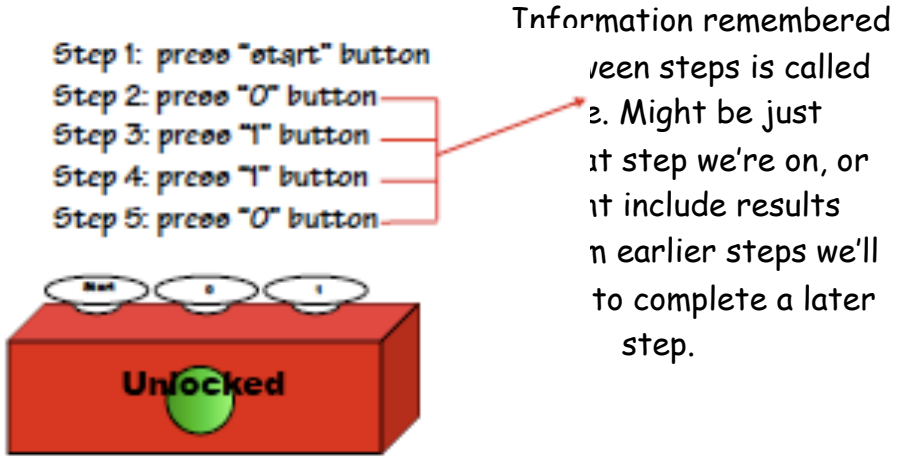
    a.k.a. the "*Hertz" specification you've heard about



State = Sequential Logic

This is one of the BIG ideas behind computers. State allows us to control sequences of events since the outputs aren't merely functions of the inputs, but also of the current state.

Example: Digital combination lock w/ 3 buttons ("start", "0" and "1"):



Information remembered
ween steps is called
e. Might be just
it step we're on, or
it include results
n earlier steps we'll
to complete a later
step.

Implementing a State Machine

We can easily extend our Lookup table approach to include state machines by

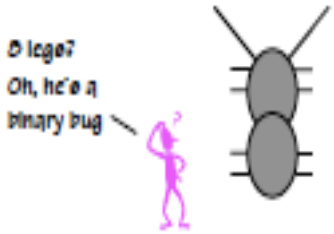
1) Adding a new input called the "CURRENT" State

2) Adding a new output called the "NEXT" state

Current state	"start"	"1"	"0"	Next state	unlock
---	1	---	---	start 000	0
start 000	0	0	1	digit1 001	0
start 000	0	1	0	error 101	0
start 000	0	0	0	start 000	0
digit1 001	0	1	0	digit2 010	0
digit1 001	0	0	1	error 101	0
digit1 001	0	0	0	digit1 001	0
digit2 010	0	1	0	digit3 011	0
digit3 011	0	0	1	unlock 100	0
unlock 100	0	1	0	error 101	1
unlock 100	0	0	1	error 101	1
unlock 100	0	0	0	unlock 100	1
error 101	0	---	---	error 101	0

6 different states → encode using 3 bits

Example State Machine: Roboant

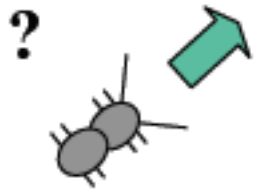


SENSORS: antennae L and R, each 1 if in contact with something.
ACTUATORS: Forward Step F, ten-degree turns TL and TR (left, right)

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

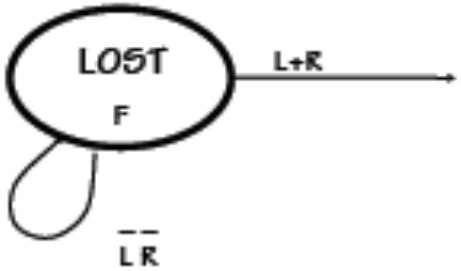
STRATEGY: "Right antenna to the wall"

Lost in Space



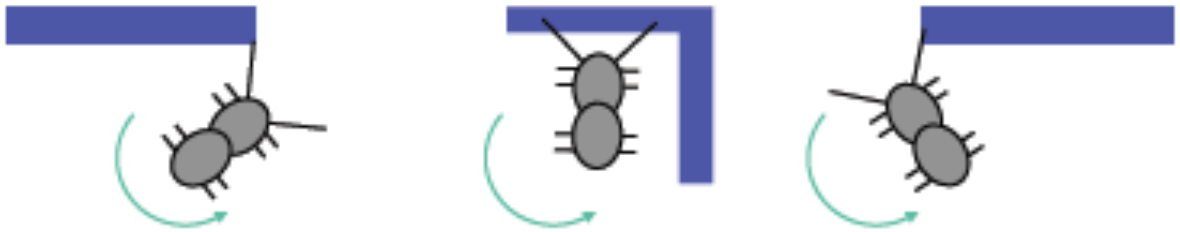
Current	L	R	TL	TR	F	Next
LOST	0	0	0	0	1	LOST
LOST	-	1	0	0	1	???
LOST	1	-	0	0	1	???

Action: Go forward until we hit something.



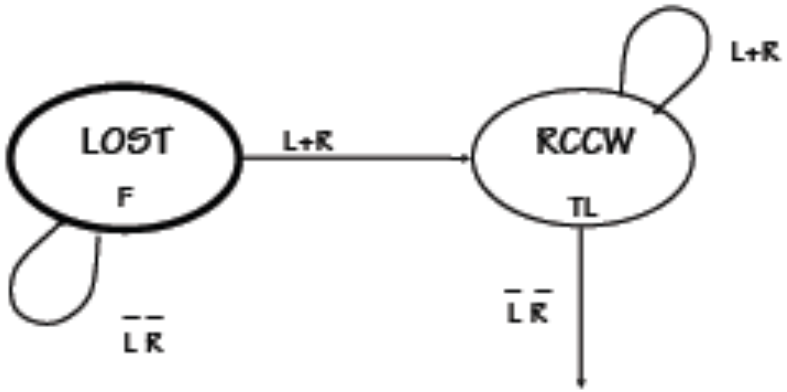
"lost" is the initial state

Bonk!

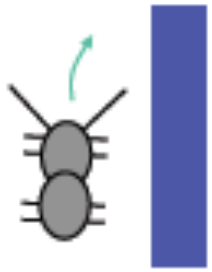


Current	L	R	TL	TR	F	Next
LOST	0	0	0	0	1	LOST
LOST	1	-	0	0	1	RCCW
LOST	-	1	0	0	1	RCCW
RCCW	1	-	1	0	0	RCCW
RCCW	-	1	1	0	0	RCCW
RCCW	0	0	1	0	0	???

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

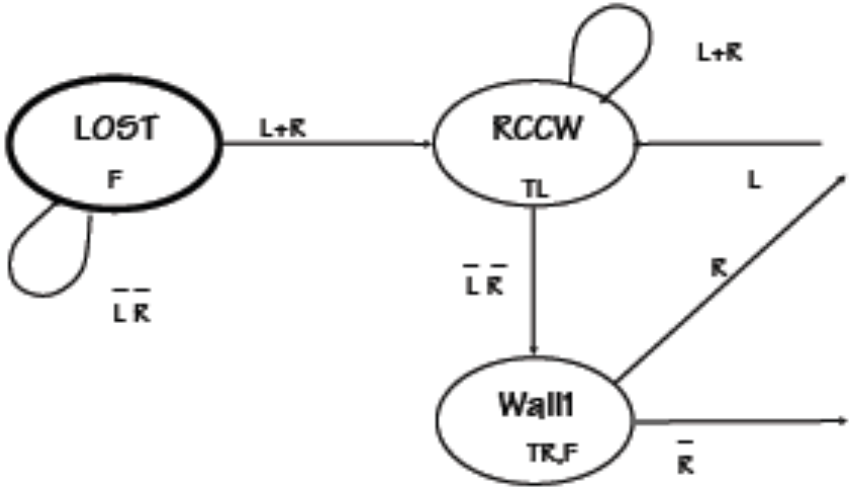


A Little to the Right…



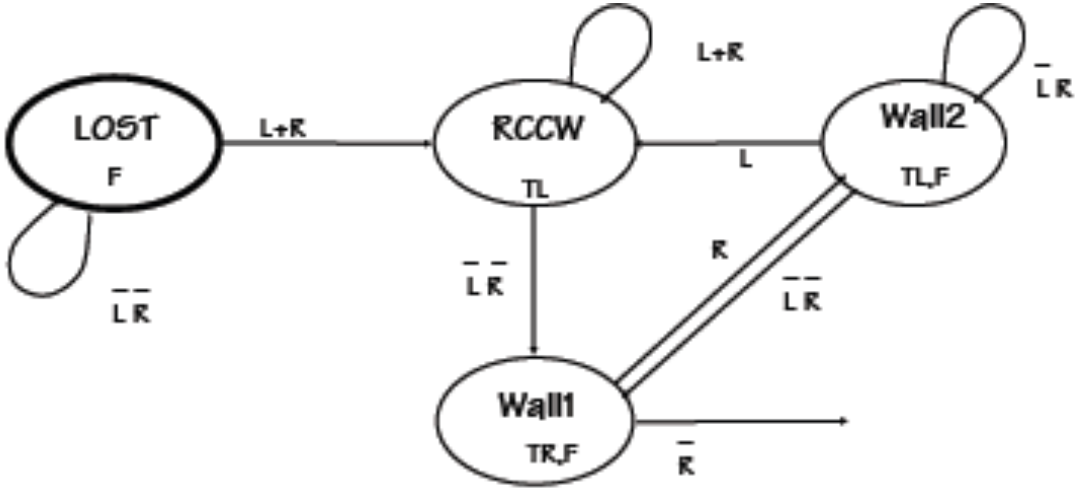
Action: Step and turn right a little, look for wall

Current	L	R	TL	TR	F	Next
LOST	0	0	0	0	1	LOST
LOST	1	-	0	0	1	RCCW
LOST	-	1	0	0	1	RCCW
RCCW	1	-	1	0	0	RCCW
RCCW	-	1	1	0	0	RCCW
RCCW	0	0	1	0	0	Wall1
Wall1	-	1	0	1	1	???
Wall1	-	0	0	1	1	???



Then a Little to the Left

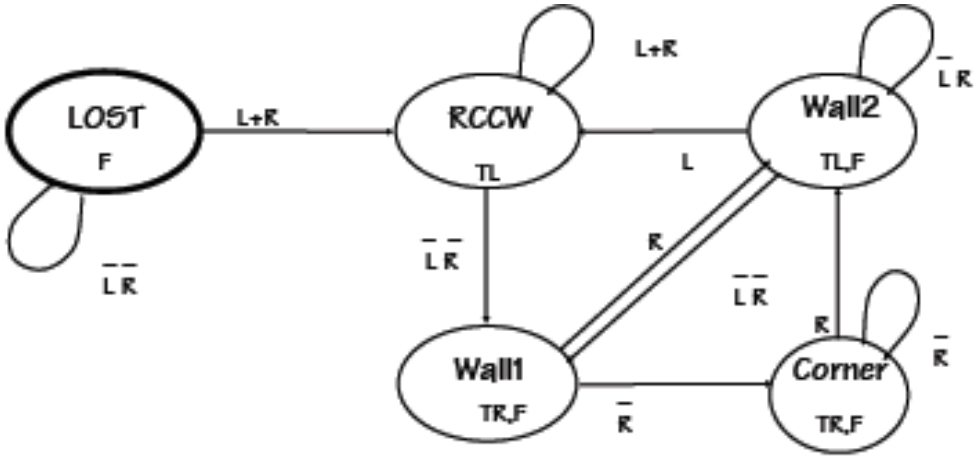
Action: Step and turn left a little, till not touching (again)



Current	L	R	T L	T R	F	Next
LOST	0	0	0	0	1	LOST
LOST	1	-	0	0	1	RCCW
LOST	-	1	0	0	1	RCCW
RCCW	1	-	1	0	0	RCCW
RCCW	-	1	1	0	0	RCCW
RCCW	0	0	1	0	0	Wall1
Wall1	-	1	0	1	1	Wall2
Wall1	-	0	0	1	1	???
Wall2	0	0	1	0	1	Wall1
Wall2	0	1	1	0	1	Wall2
Wall2	1	-	1	0	1	RCCW

Dealing with Corners

Action: Step and turn right until we hit perpendicular wall



Current	L	R	T L	T R	F	Next
LOST	0	0	0	0	1	LOST
LOST	1	-	0	0	1	RCCW
LOST	-	1	0	0	1	RCCW
RCCW	1	-	1	0	0	RCCW
RCCW	-	1	1	0	0	RCCW
RCCW	0	0	1	0	0	Wall1
Wall1	-	1	0	1	1	Wall2
Wall1	-	0	0	1	1	Corner
Wall2	0	0	1	0	1	Wall1
Wall2	0	1	1	0	1	Wall2
Wall2	1	-	1	0	1	RCCW
Corner	-	1	0	1	1	Wall2
Corner	-	0	0	1	1	Corner

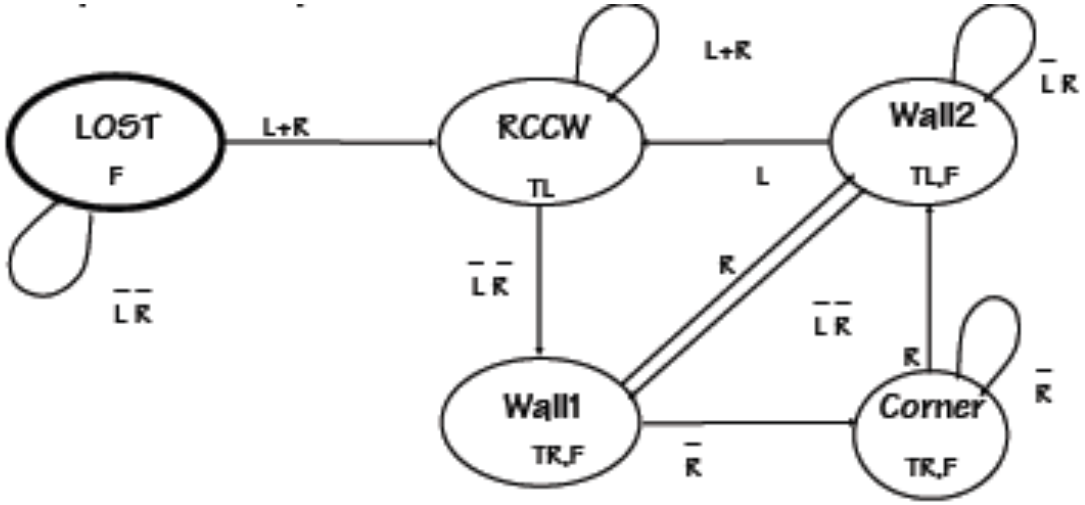
Equivalent State Reduction

Observation: Si ≡ Sj if

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

Reduction Strategy:

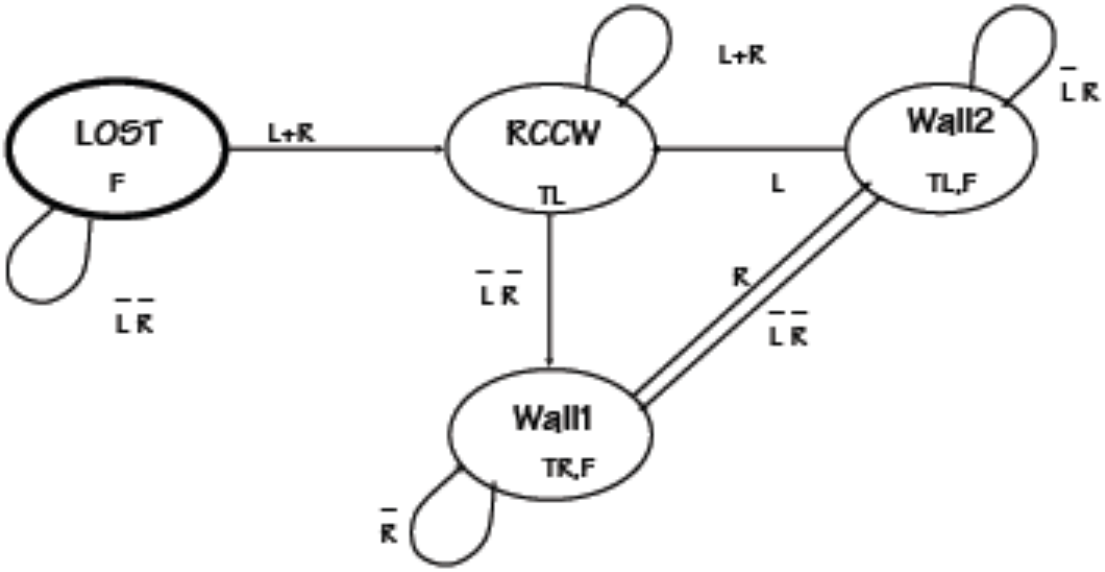
Find pairs of equivalent states, MERGE them.



Current	L	R	T L	T R	F	Next
LOST	0	0	0	0	1	LOST
LOST	1	-	0	0	1	RCCW
LOST	-	1	0	0	1	RCCW
RCCW	1	-	1	0	0	RCCW
RCCW	-	1	1	0	0	RCCW
RCCW	0	0	1	0	0	Wall1
Wall1	-	1	0	1	1	Wall2
Wall1	-	0	0	1	1	Corner
Wall2	0	0	1	0	1	Wall1
Wall2	0	1	1	0	1	Wall2
Wall2	1	-	1	0	1	RCCW
Corner	-	1	0	1	1	Wall2
Corner	-	0	0	1	1	Corner

An Evolutionary Step

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

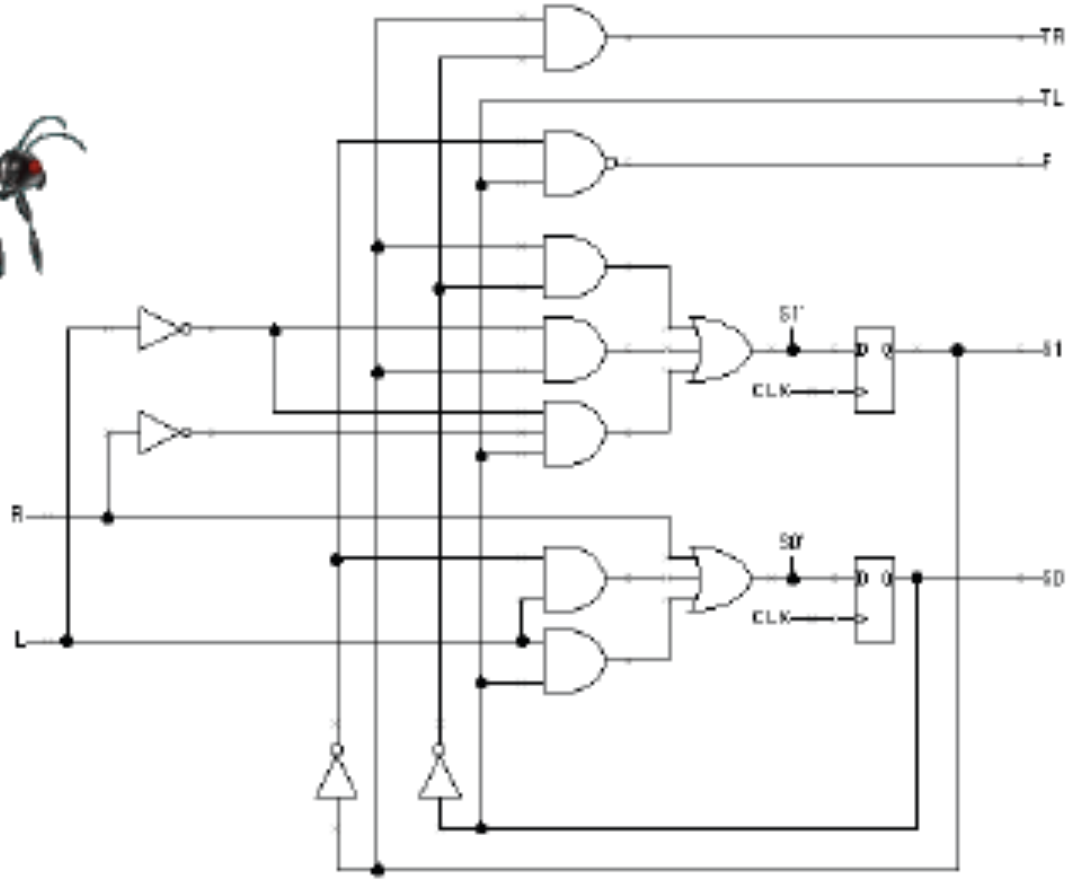


Behaves exactly as previous (5-state) state machine!

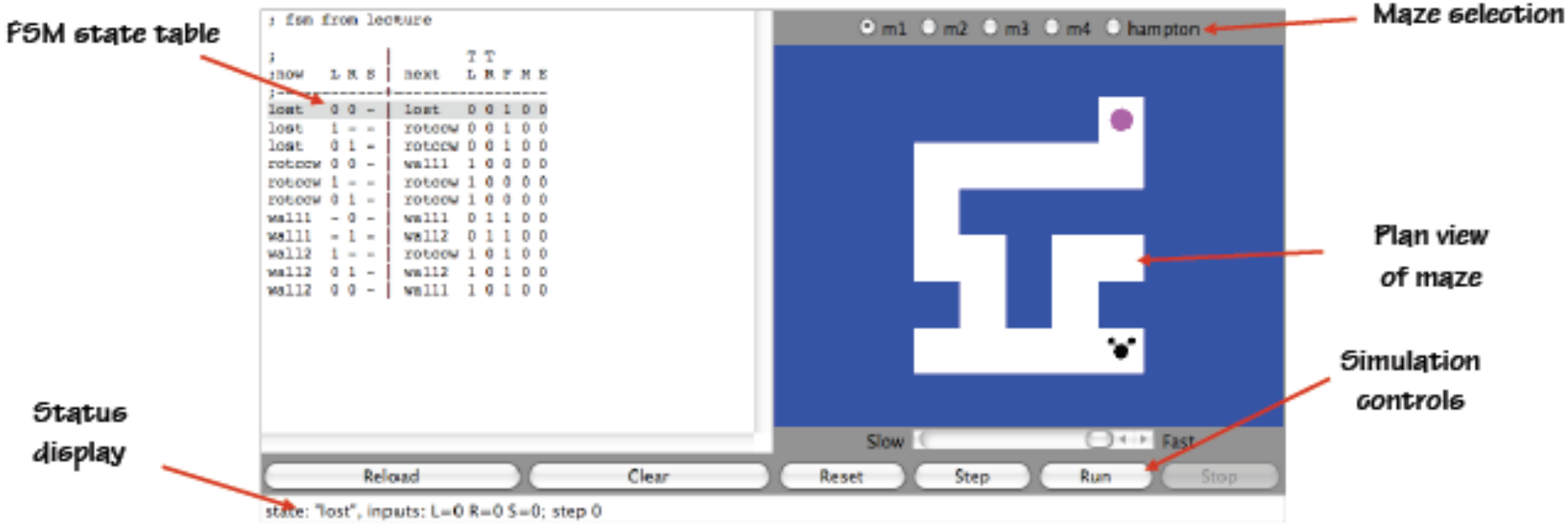
Current	L	R	TL	TR	F	Next
LOST	0	0	0	0	1	LOST
LOST	1	-	0	0	1	RCCW
LOST	0	1	0	0	1	RCCW
RCCW	1	-	1	0	0	RCCW
RCCW	0	1	1	0	0	RCCW
RCCW	0	0	1	0	0	Wall1
Wall1	-	1	0	1	1	Wall2
Wall1	-	0	0	1	1	Wall1
Wall2	0	0	1	0	1	Wall1
Wall2	0	1	1	0	1	Wall2
Wall2	1	-	1	0	1	RCCW

An Ant Schematic

Are you telling me that my essence amounts to no more than 15 gates!



Super Roboant



Featuring an improved Roboant that can add crumbs (M), erase crumbs (E), and sense (S) crumbs along its path.