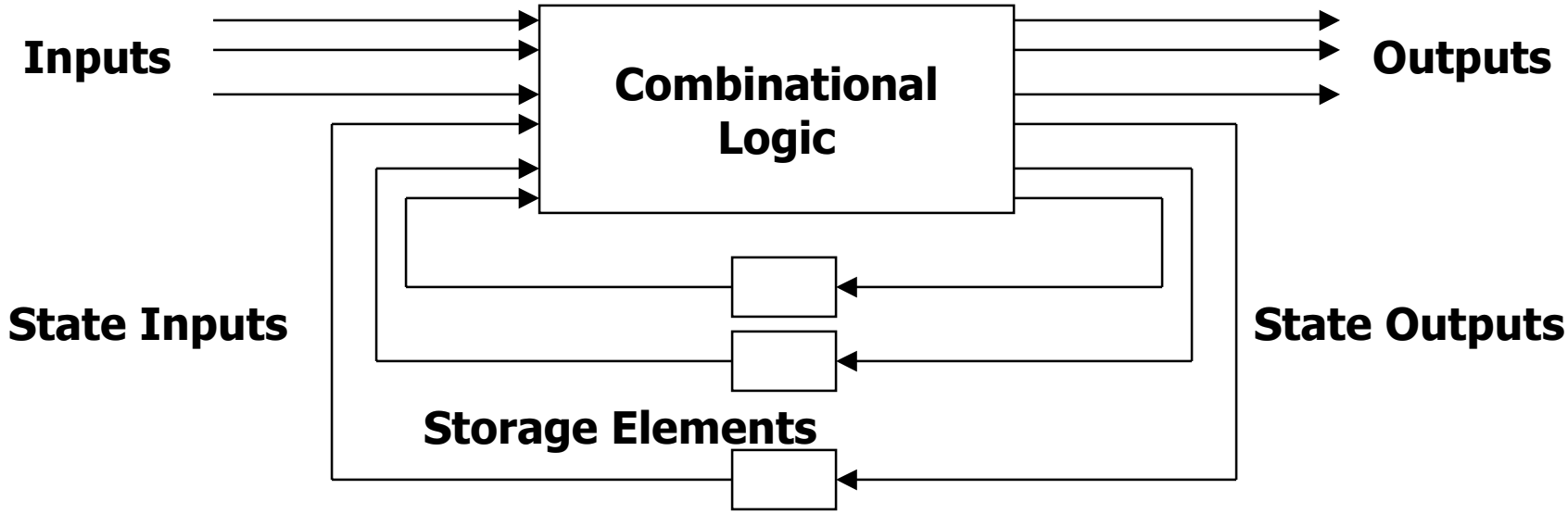
Sequential logic implementation

- Sequential systems
  - primitive sequential elements
  - combinational logic
- Models for representing sequential circuits
  - finite-state machines (Moore and Mealy)
  - representation of memory (states)
  - changes in state (transitions)
- Basic sequential circuits
  - shift registers
  - counters
- Design procedure
  - state diagrams
  - state transition table
  - next state functions

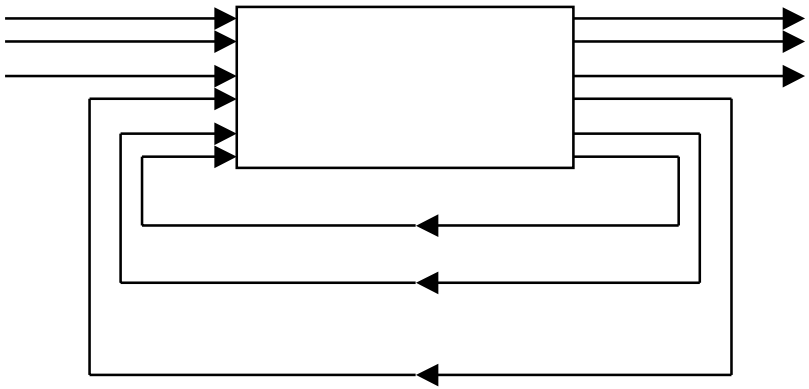
Abstraction of state elements

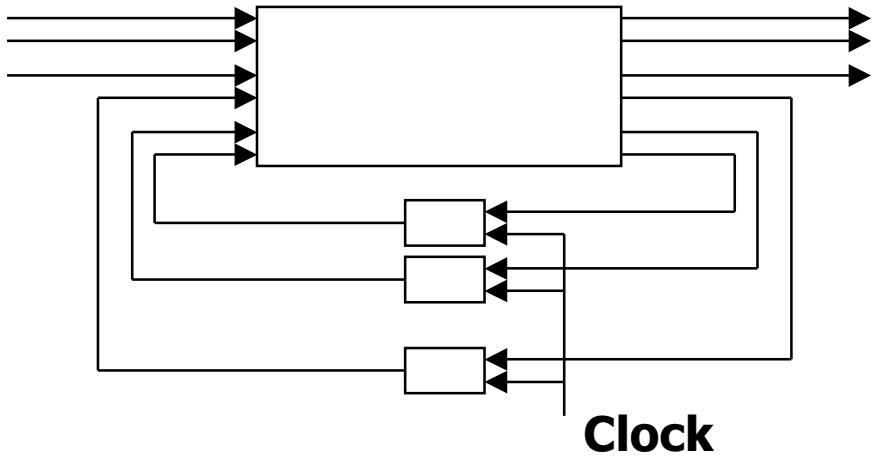
- Divide circuit into combinational logic and state
- Localize the feedback loops and make it easy to break cycles
- Implementation of storage elements leads to various forms of sequential logic



Forms of sequential logic

- Asynchronous sequential logic – state changes occur whenever state inputs change (elements may be simple wires or delay elements)
- Synchronous sequential logic – state changes occur in lock step across all storage elements (using a periodic waveform - the clock)

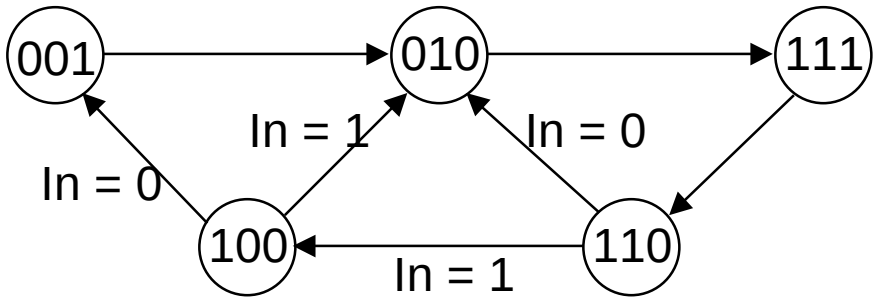




Finite state machine representations

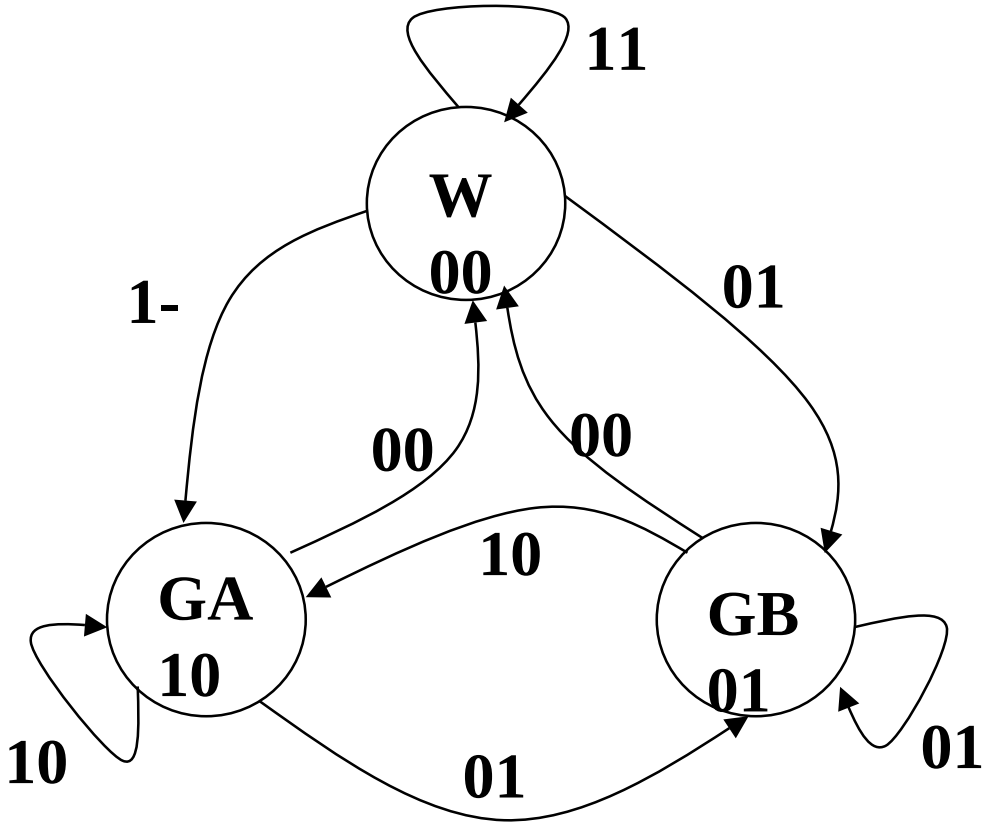
- States: determined by possible values in sequential storage elements
- Transitions: change of state
- Clock: controls when state can change by controlling storage elements

- Sequential logic
  - sequences through a series of states
  - based on sequence of values on input signals
  - clock defines time divisions between states and input values



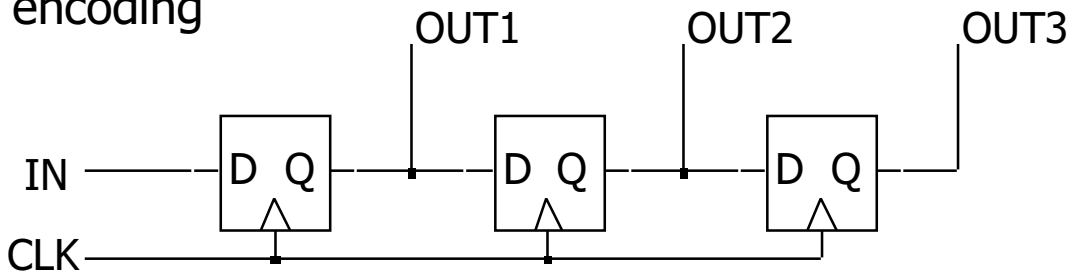
Example finite state machine diagram

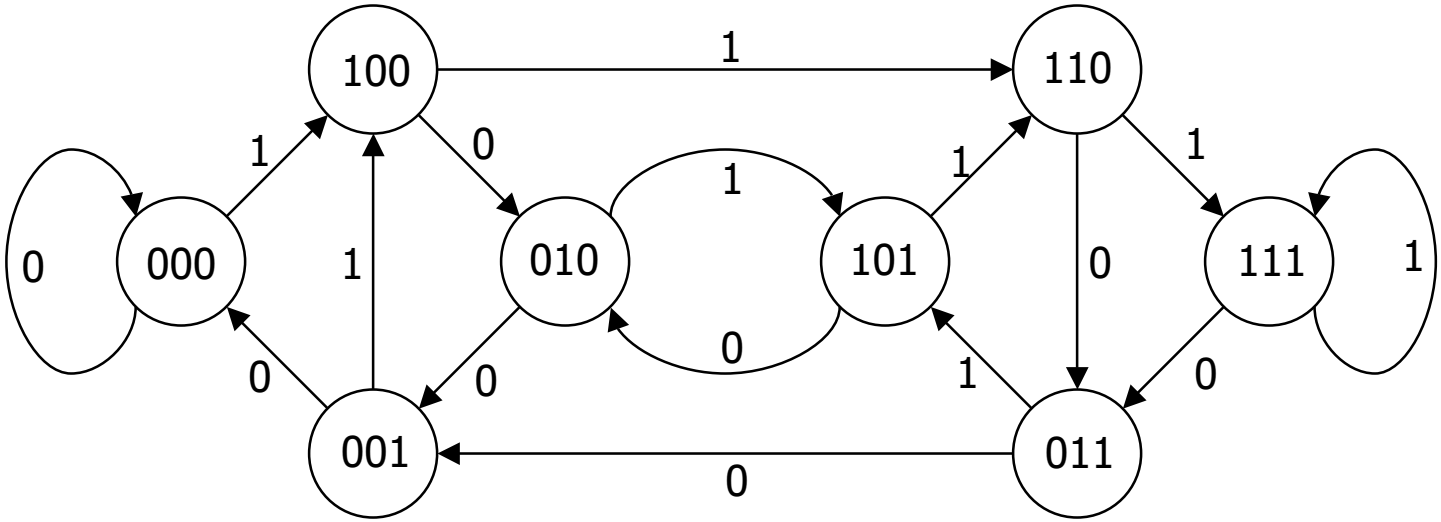
- Mutex from Introduction
  - Two inputs
  - Three states, one for each possible output configuration
  - Outputs a function of state only (Moore machine)



Shift Register Example

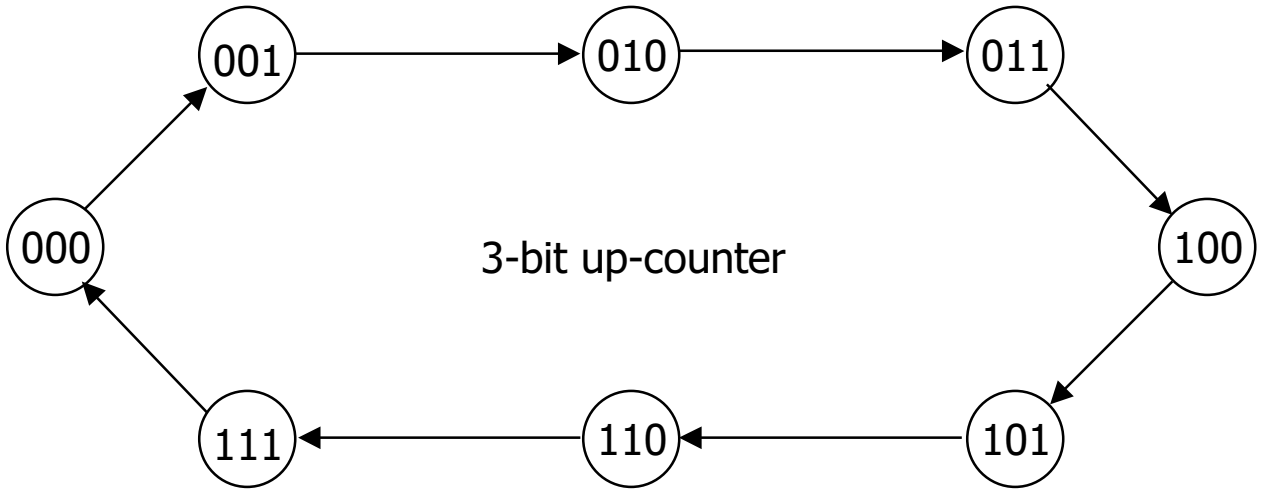
- Complex State Diagram, Simple Logic
  - input value shown on transition arcs
  - State encoding = output encoding





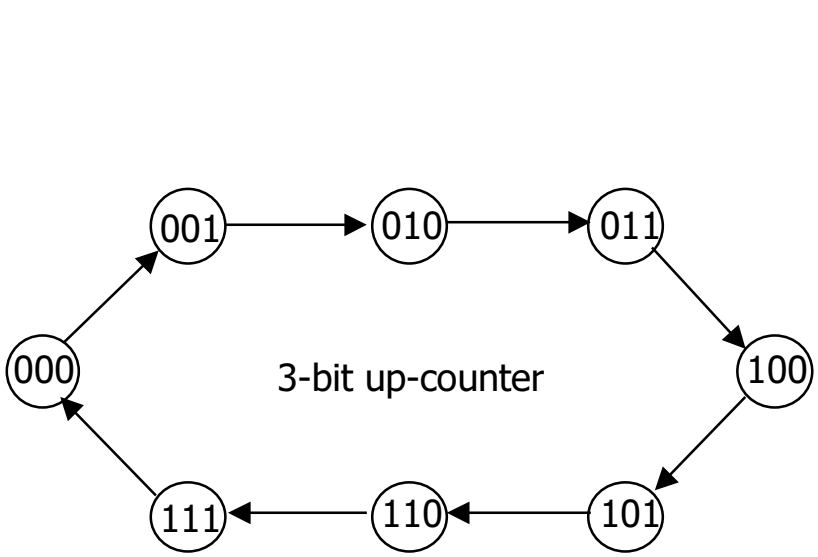
Counters Example

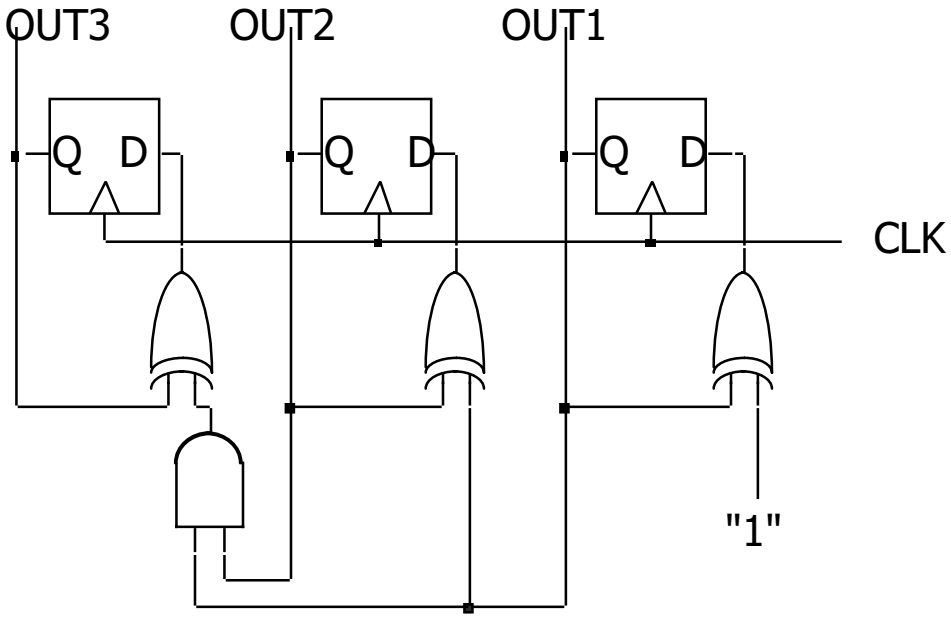
- Counters
  - proceed through well-defined sequence of states in response to enable
- Many types of counters: binary, BCD, Gray-code
  - 3-bit up-counter: 000, 001, 010, 011, 100, 101, 110, 111, 000, ...
  - 3-bit down-counter: 111, 110, 101, 100, 011, 010, 001, 000, 111, ...



How do we turn a state diagram into logic?

- Counter
  - 3 flip-flops to hold state
  - logic to compute next state
  - clock signal controls when flip-flop memory can change
    - wait long enough for combinational logic to compute new value
    - don't wait too long as that is low performance



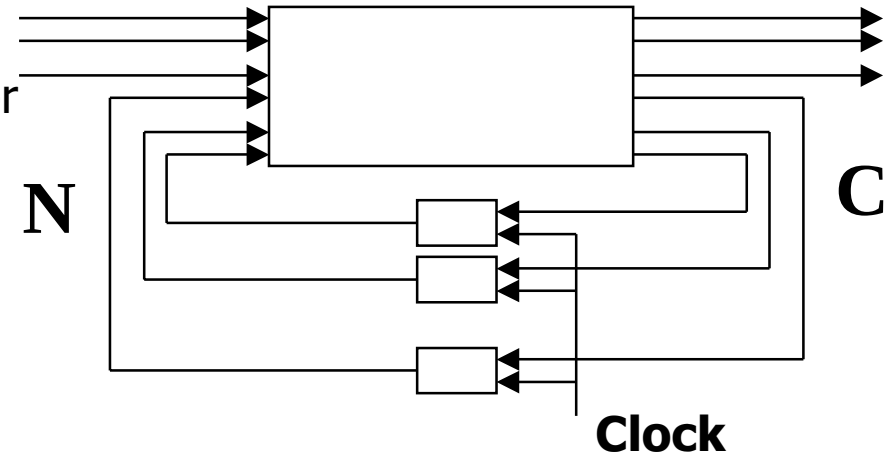


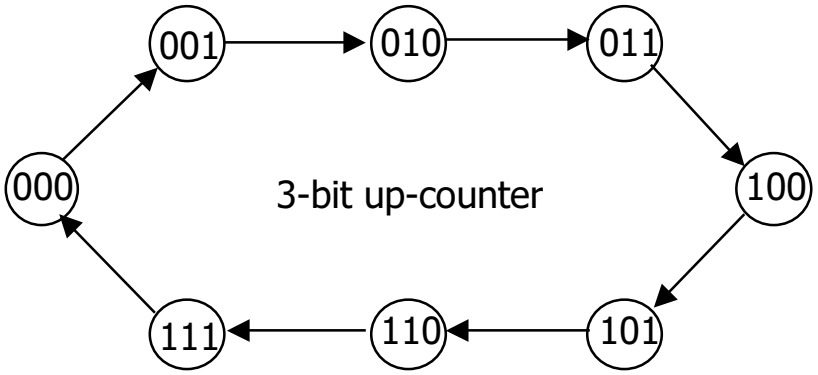
FSM design procedure

- Start with counters
  - simple because output is just state
  - simple because no choice of next state based on input
- State diagram to state transition table
  - tabular form of state diagram
  - like a truth-table
- State encoding
  - decide on representation of states
  - for counters it is simple: just its value
- Implementation
  - flip-flop for each state bit
  - combinational logic based on encoding

FSM design procedure: state diagram to encoded state transition table

- Tabular form of state diagram
- Like a truth-table (specify output for all input combinations)
- Encoding of states: easy for counters – just use value





current state		next state	
0	000	001	1
1	001	010	2
2	010	011	3
3	011	100	4
4	100	101	5
5	101	110	6
6	110	111	7
7	111	000	0

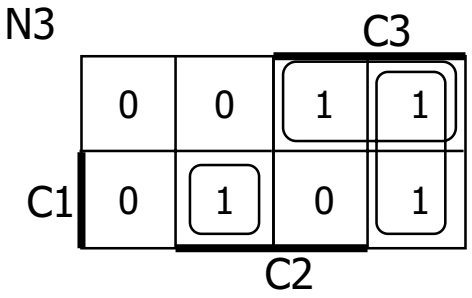
Implementation

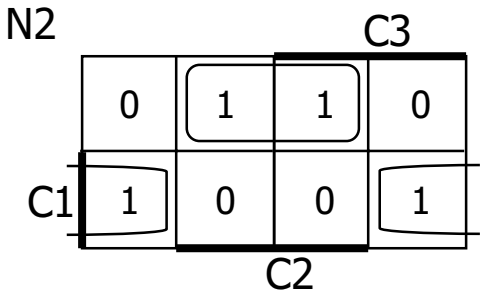
- D flip-flop for each state bit
- Combinational logic based on encoding

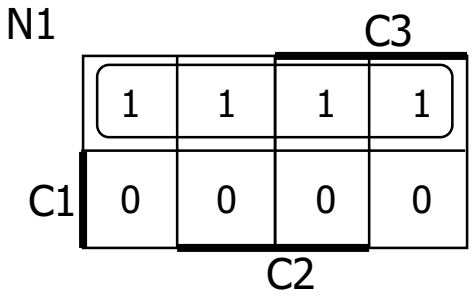
C3	C2	C1	N3	N2	N1
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

notation to show function represent input to D-FF

N1 := C1'
N2 := C1C2' + C1'C2
    := C1 xor C2
N3 := C1C2C3' + C1'C3 + C2'C3
    := C1C2C3' + (C1' + C2')C3
    := (C1C2) xor C3



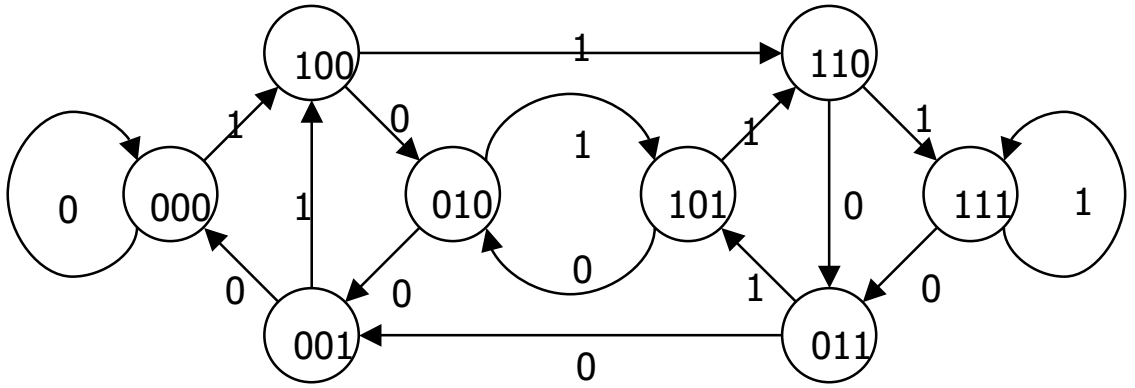




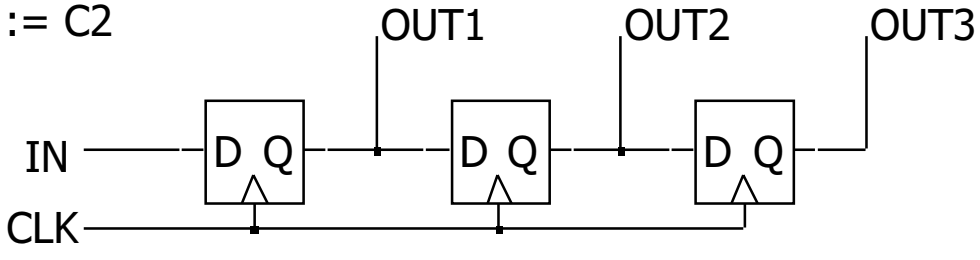
Another example

- Shift register
  - input determines next state

In	C1	C2	C3	N1	N2	N3
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	1	0	1
1	0	1	1	1	0	1
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	1	1	1

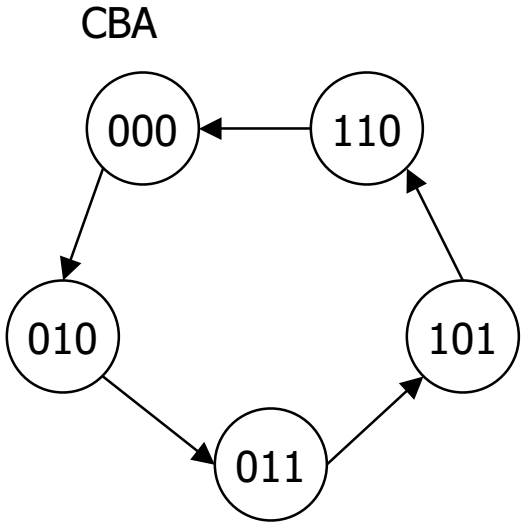


N1 := In
N2 := C1
N3 := C2



More complex counter example

- Complex counter
  - repeats 5 states in sequence
  - not a binary number representation
- Step 1: derive the state transition diagram
  - count sequence: 000, 010, 011, 101, 110
- Step 2: derive the state transition table from the state transition diagram



Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	x	x	x
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	x	x	x
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	x	x	x

note the don't care conditions that arise from the unused state codes

More complex counter example (cont'd)

- Step 3: K-maps for next state functions

C+

A \ BC	00	01	11	10
0	0	0	X	0
1	X	1	X	1

B+

A \ BC	00	01	11	10
0	1	1	0	X
1	X	0	X	1

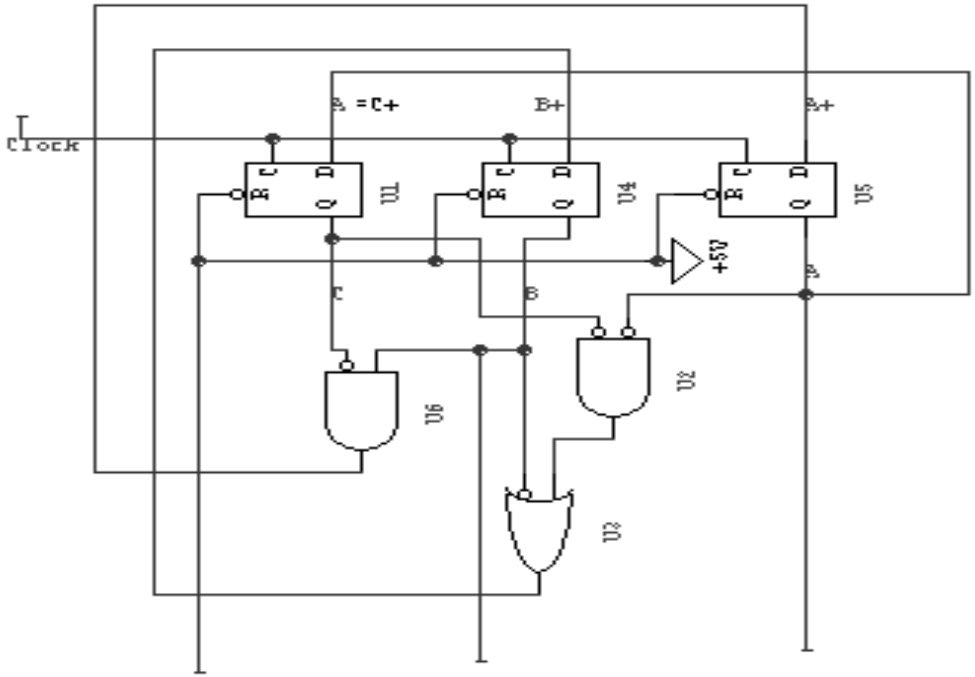
A+

A \ BC	00	01	11	10
0	0	1	0	X
1	X	1	X	0

C+ := A

B+ := B' + A'C'

A+ := BC'



Self-starting counters (cont'd)

- Re-deriving state transition table from don't care assignment

C+

A \ BC	00	01	11	10
0	0	0	0	0
1	1	1	1	1

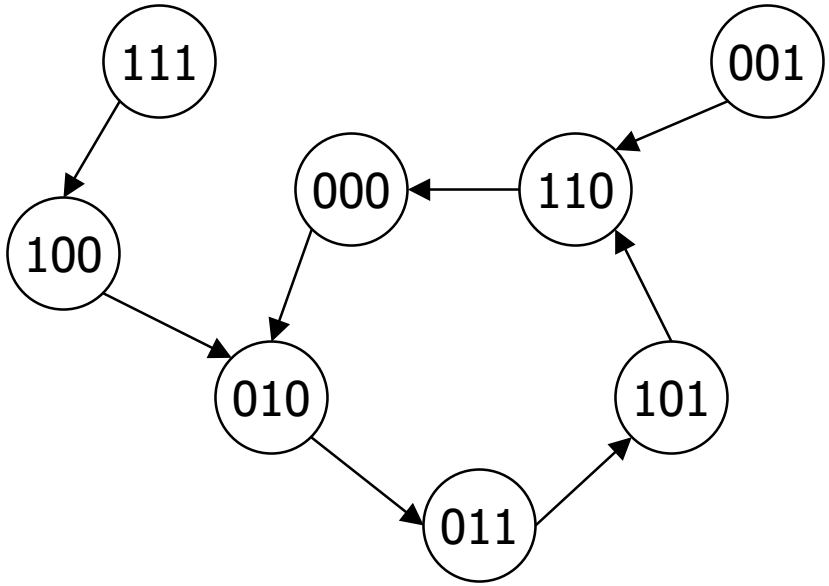
B+

A \ BC	00	01	11	10
0	1	1	0	1
1	1	0	0	1

A+

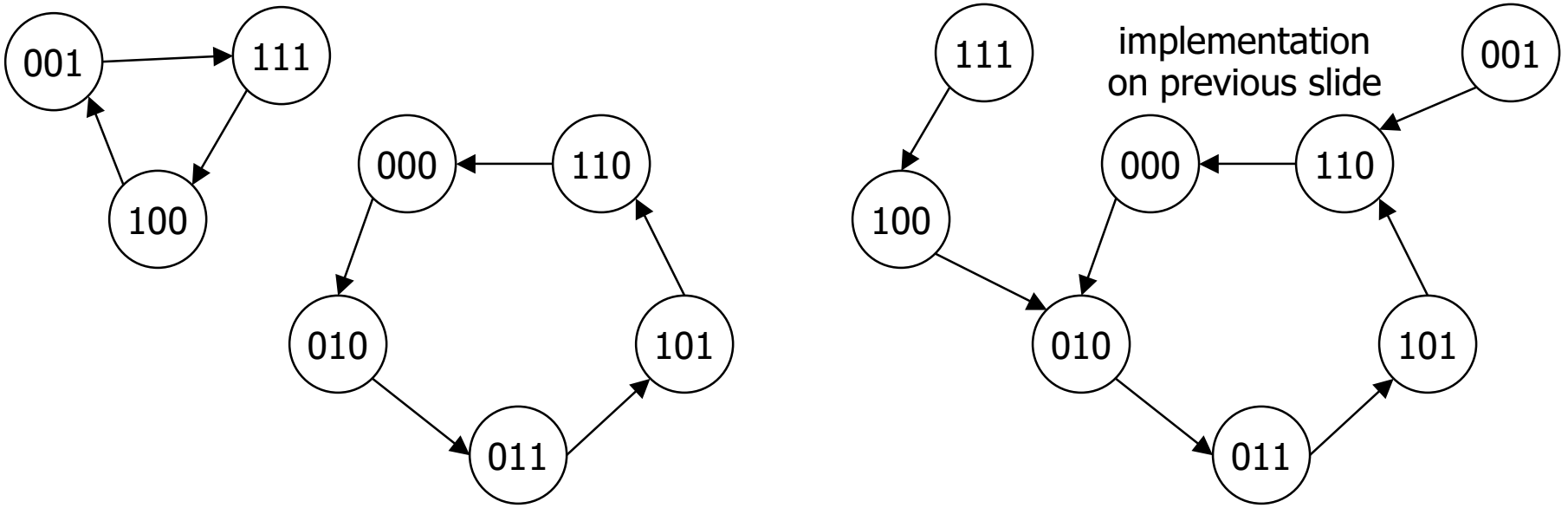
A \ BC	00	01	11	10
0	0	1	0	0
1	0	1	0	0

Present State			Next State		
C	B	A	C+	B+	A+
0	0	0	0	1	0
0	0	1	1	1	0
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	1	1	0	0



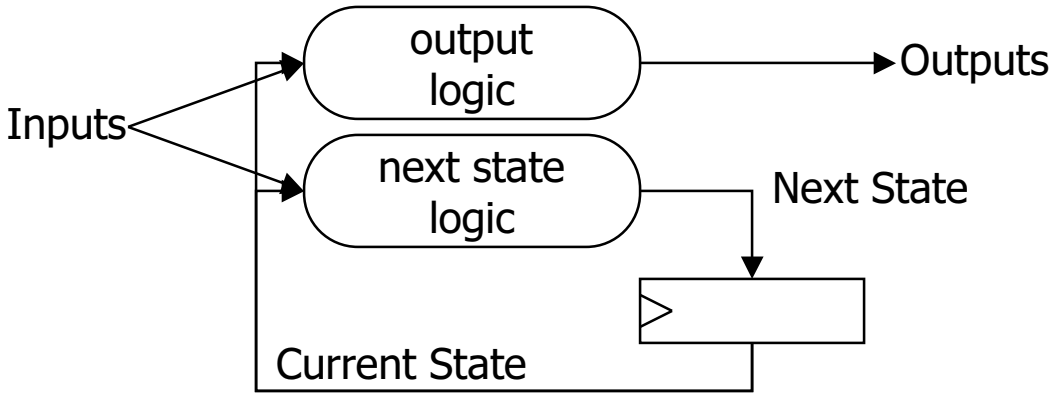
Self-starting counters

- Start-up states
  - at power-up, counter may be in an unused or invalid state
  - designer must guarantee that it (eventually) enters a valid state
- Self-starting solution
  - design counter so that invalid states eventually transition to a valid state
  - may limit exploitation of don't cares



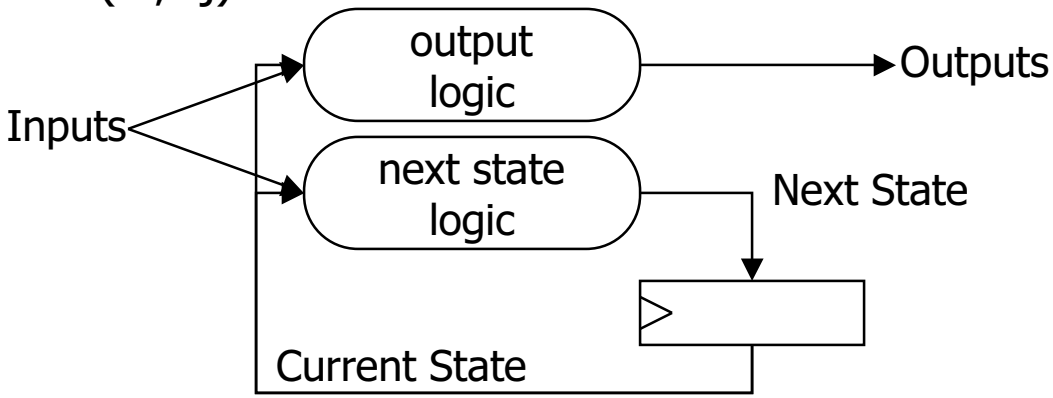
State machine model

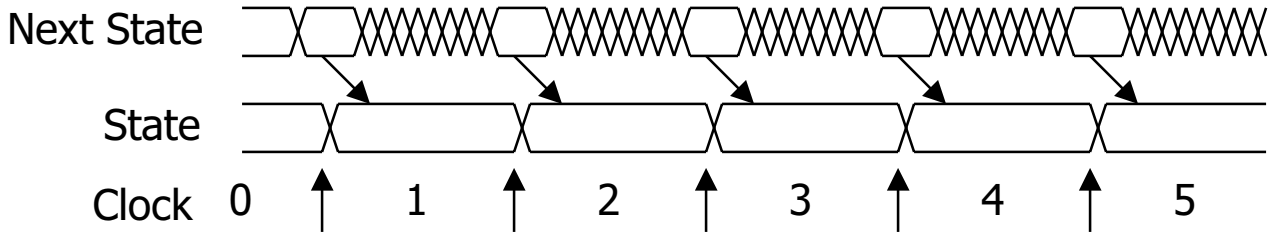
- Values stored in registers represent the state of the circuit
- Combinational logic computes:
  - next state
    - function of current state and inputs
  - outputs
    - function of current state and inputs (Mealy machine)
    - function of current state only (Moore machine)



State machine model (cont'd)

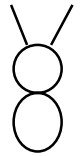
- States: S1, S2, ..., Sk
- Inputs: I1, I2, ..., Im
- Outputs: O1, O2, ..., On
- Transition function: Fs(Si, Ij)
- Output function: Fo(Si) or Fo(Si, Ij)

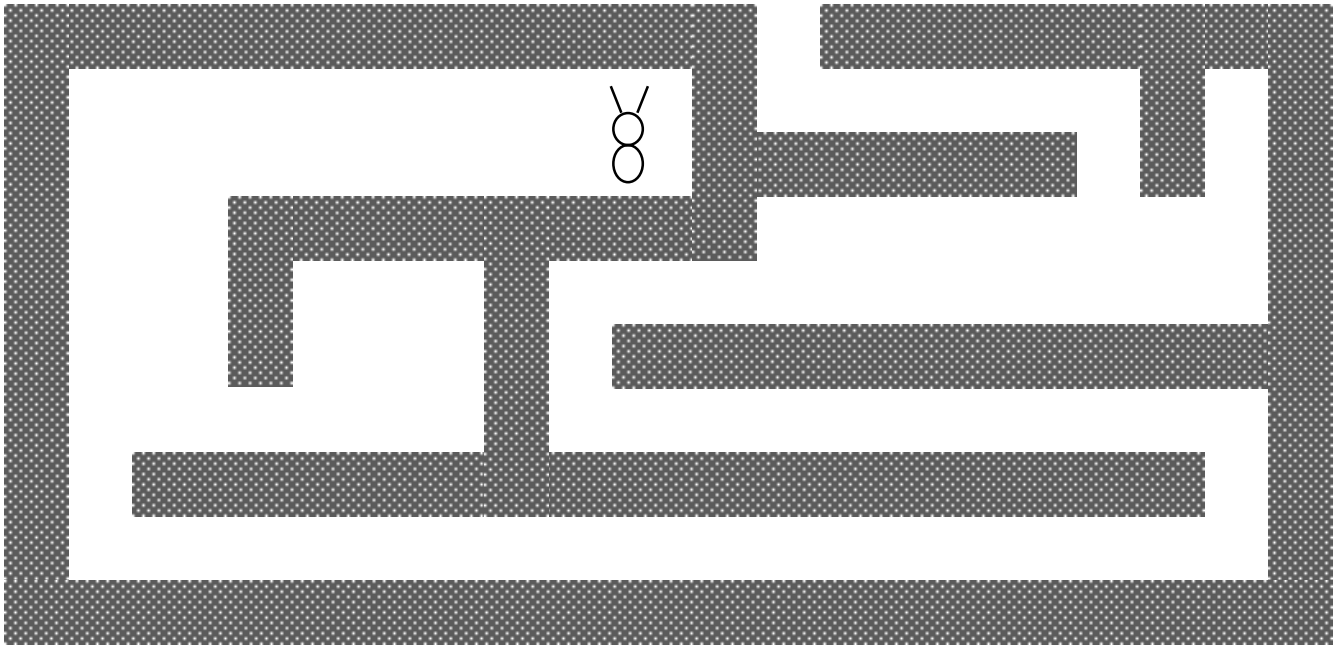




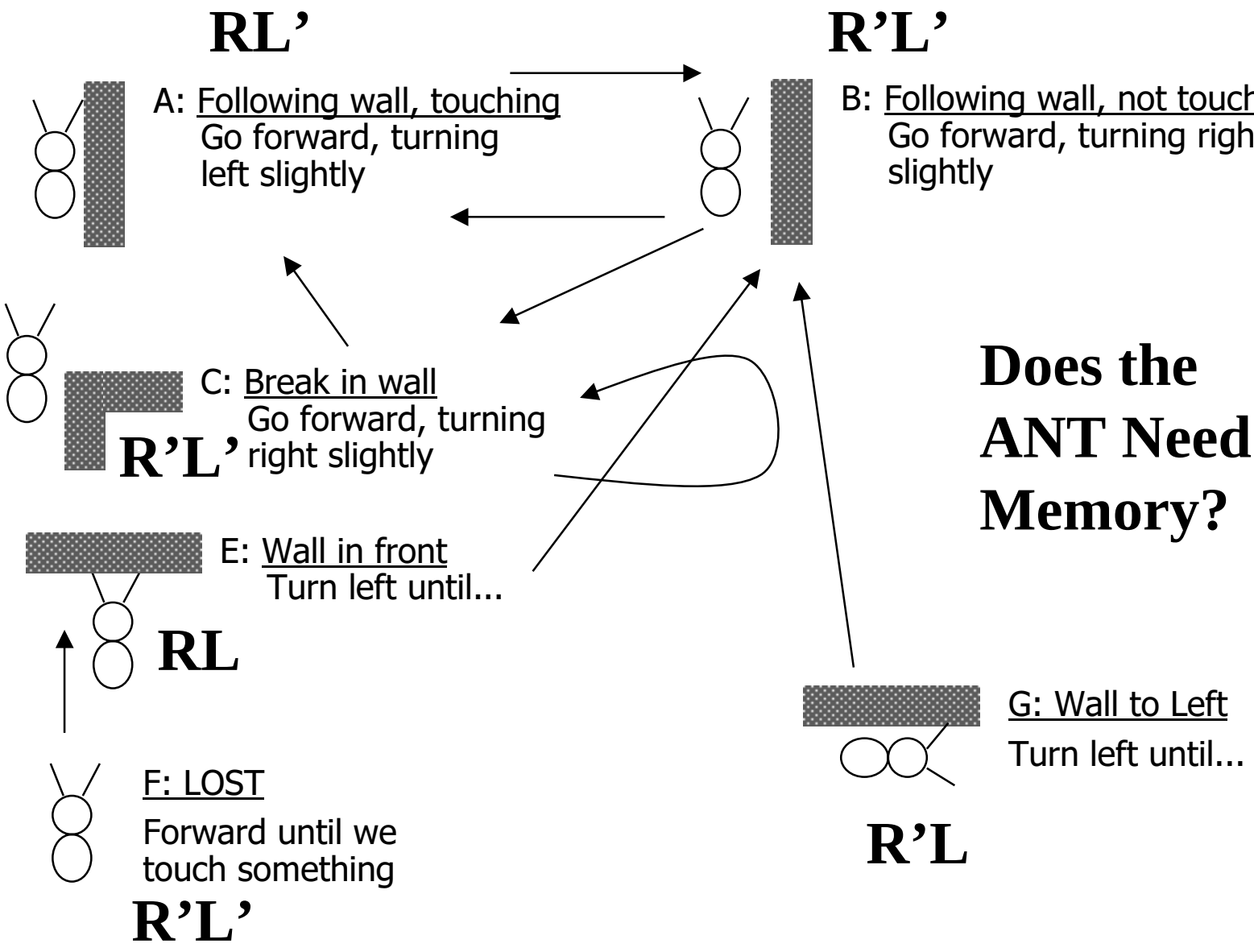
Example: ant brain (Ward, MIT)

- Sensors: L and R antennae, 1 if in touching wall
- Actuators: F - forward step, TL/TR - turn left/right slightly
- Goal: find way out of maze
- Strategy: keep the wall on the right





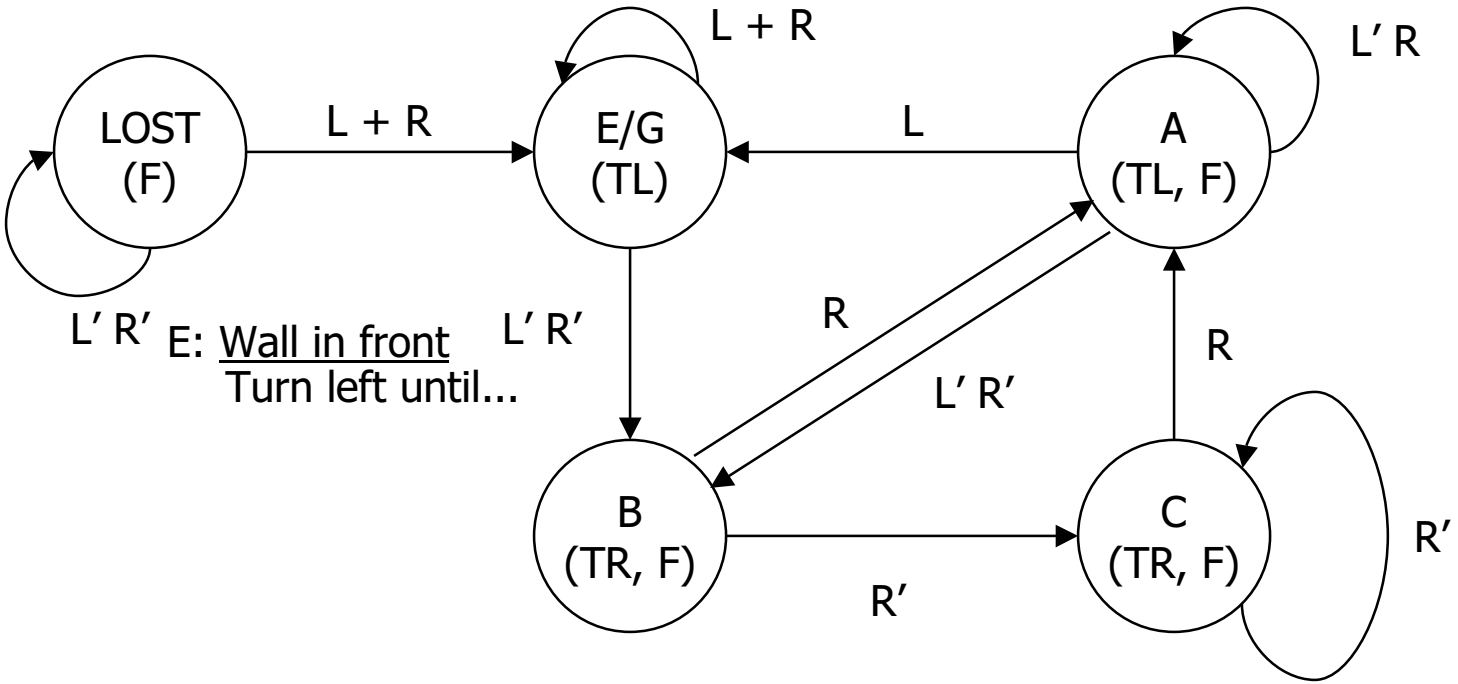
Ant behavior: And States of Mind



Designing an ant brain

A: Following wall, touching
B: Following wall, not touching
C: Break in wall, not touching
E: Wall in Front
G: Wall to Left
F: Lost

- State diagram

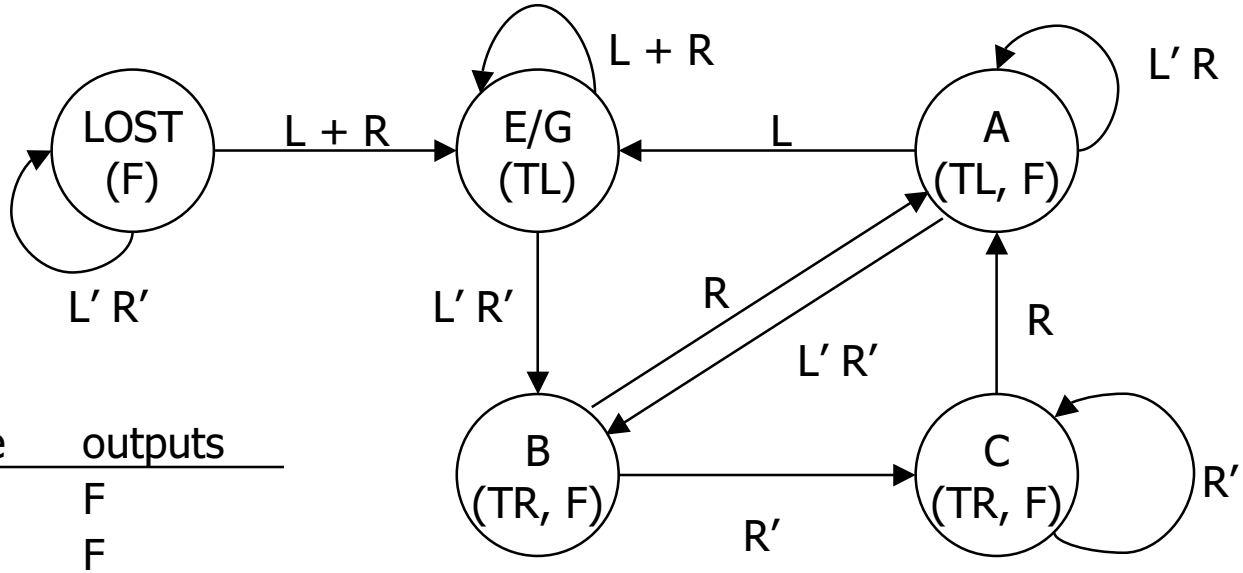


Synthesizing the ant brain circuit

- Encode states using a set of state variables
  - arbitrary choice - may affect cost, speed
- Use transition truth table
  - define next state function for each state variable
  - define output function for each output
- Implement next state and output functions using combinational logic
  - 2-level logic (ROM/PLA/PAL)
  - multi-level logic
  - next state and output functions can be optimized together

Transition truth table

- Using symbolic states and outputs



state	L	R	next state	outputs
LOST	0	0	LOST	F
LOST	–	1	E/G	F
LOST	1	–	E/G	F
A	0	0	B	TL, F
A	0	1	A	TL, F
A	1	–	E/G	TL, F
B	–	0	C	TR, F
B	–	1	A	TR, F
...	...	...	...	...

Synthesis

- 5 states : at least 3 state variables required (X, Y, Z)
  - state assignment (in this case, arbitrarily chosen)

LOST - 000
E/G - 001
A - 010
B - 011
C - 100

state X,Y,Z	L	R	next state X', Y', Z'	outputs F	TR	TL
0 0 0	0	0	0 0 0	1	0	0
0 0 0	0	1	0 0 1	1	0	0
...	...	...	...	...		
0 1 0	0	0	0 1 1	1	0	1
0 1 0	0	1	0 1 0	1	0	1
0 1 0	1	0	0 0 1	1	0	1
0 1 0	1	1	0 0 1	1	0	1
0 1 1	0	0	1 0 0	1	1	0
0 1 1	0	1	0 1 0	1	1	0
...	...	...	...	...		

it now remains to synthesize these 6 functions

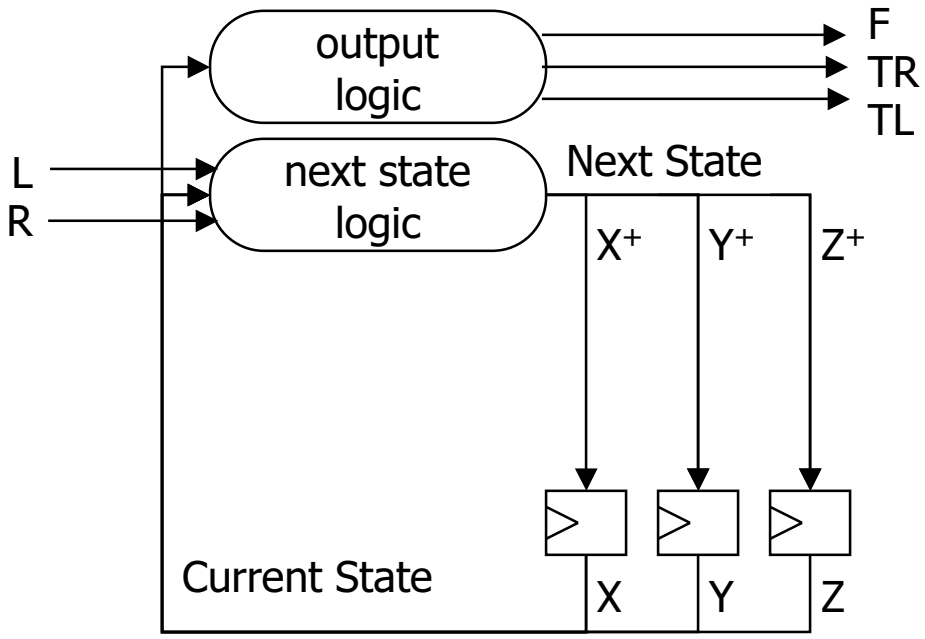
e.g.

$TR = X + Y Z$

$X^+ = X R' + Y Z R' = R' TR$

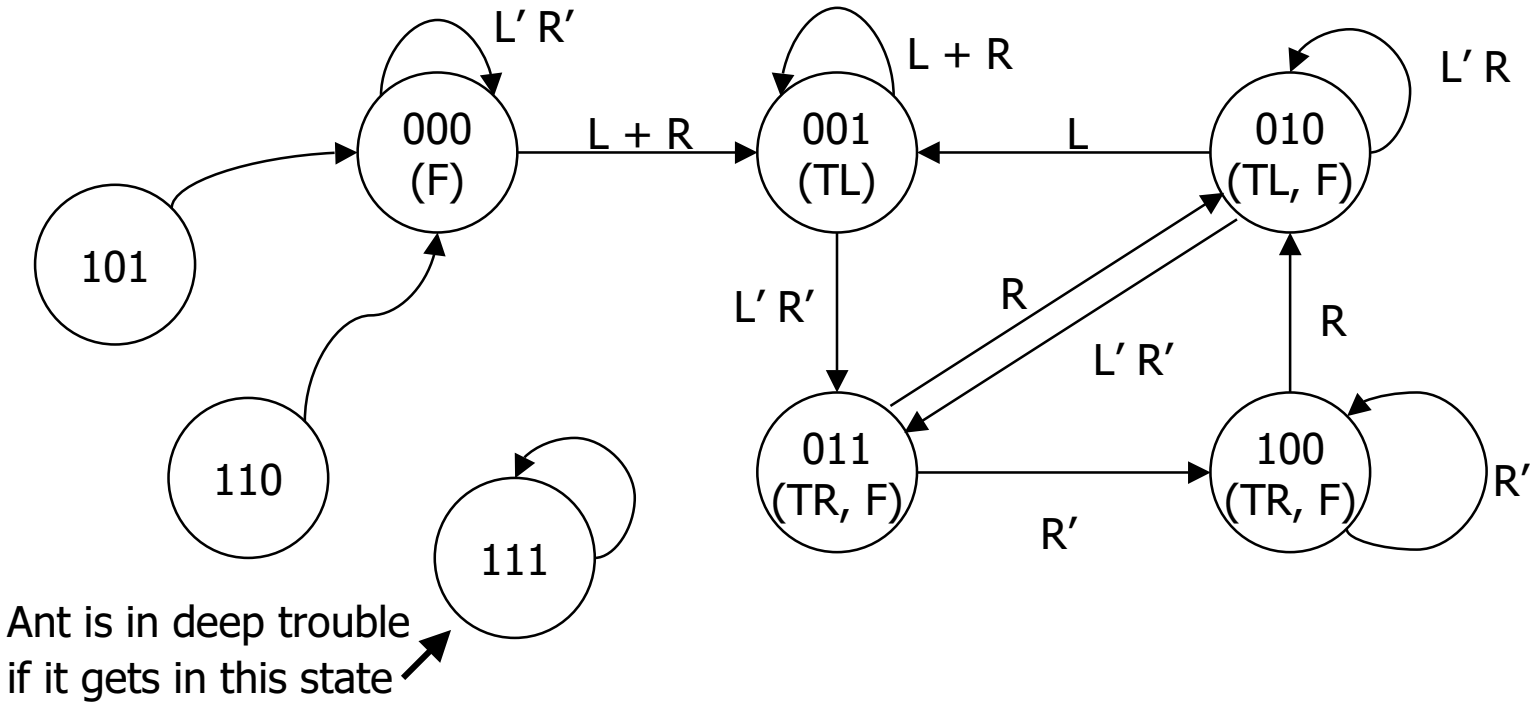
Circuit implementation

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



Don’t cares in FSM synthesis

- What happens to the "unused" states (101, 110, 111)?
- They were exploited as don't cares to minimize the logic
  - if the states can't happen, then we don't care what the functions do
  - if states do happen, we may be in trouble



State minimization

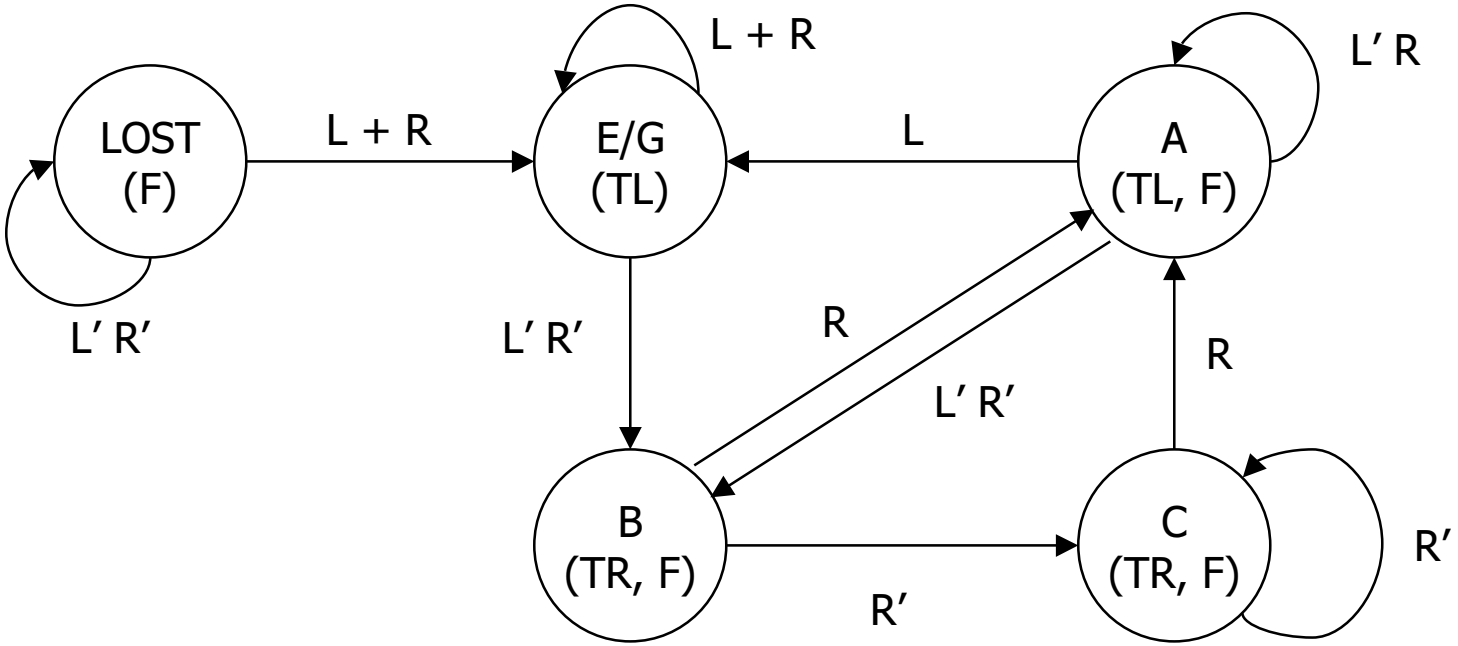
- Fewer states may mean fewer state variables
- High-level synthesis may generate many redundant states
- Two state are equivalent if they are impossible to distinguish from the outputs of the FSM, i. e., for any input sequence the outputs are the same
- Two conditions for two states to be equivalent:
  - 1) output must be the same in both states
  - 2) must transition to equivalent states for all input combinations

Ant brain revisited

- Any equivalent states?

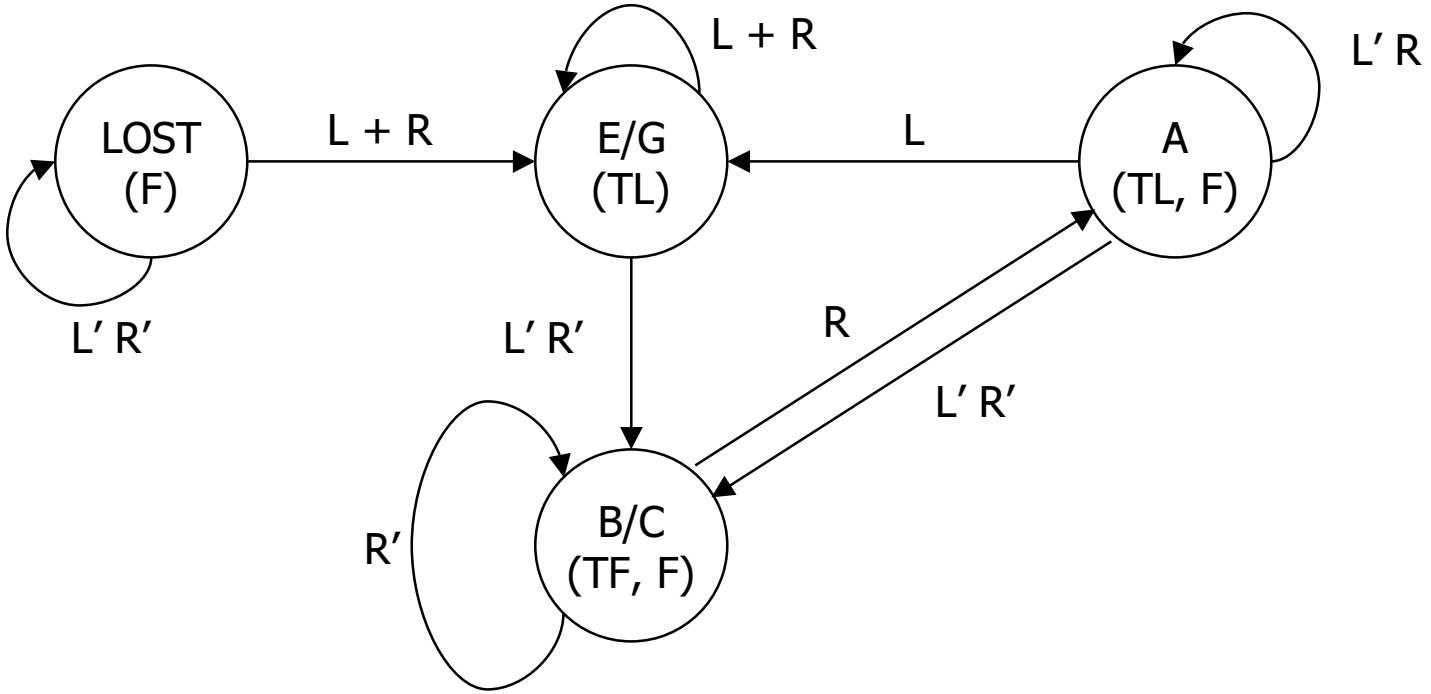
A: Following wall, touching
B: Following wall, not touching
C: Break in wall, not touching
E: Wall in Front
G: Wall to Left
F: Lost

On track
Oriented
Getting Oriented
Lost



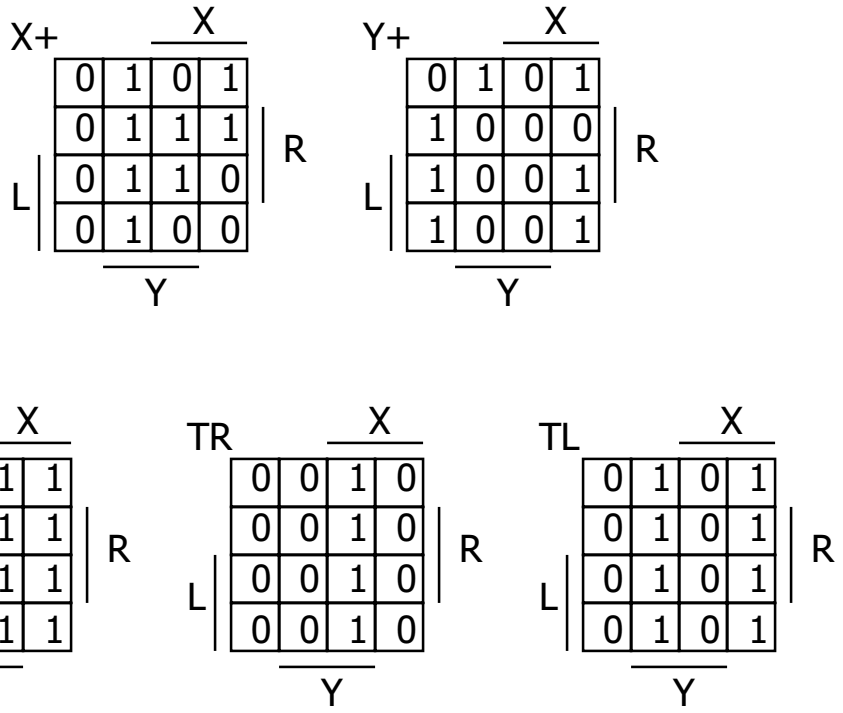
New improved brain

- Merge equivalent B and C states
- Behavior is exactly the same as the 5-state brain
- We now need only 2 state variables rather than 3



New brain implementation

state X,Y	inputs L R	next state X',Y'	outputs F TR TL
0 0	0 0	0 0	1 0 0
0 0	- 1	0 1	1 0 0
0 0	1 -	0 1	1 0 0
0 1	0 0	1 1	0 0 1
0 1	- 1	1 0	0 0 1
0 1	1 -	1 0	0 0 1
1 0	0 0	1 1	1 0 1
1 0	0 1	1 0	1 0 1
1 0	1 -	0 1	1 0 1
1 1	- 0	0 0	1 1 0
1 1	- 1	1 0	1 1 0



Do we really need 4 states?