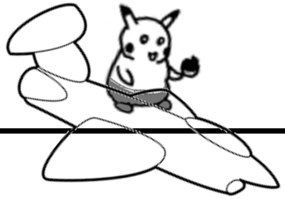


Fall 2015

Lecture 22: Finite state machines

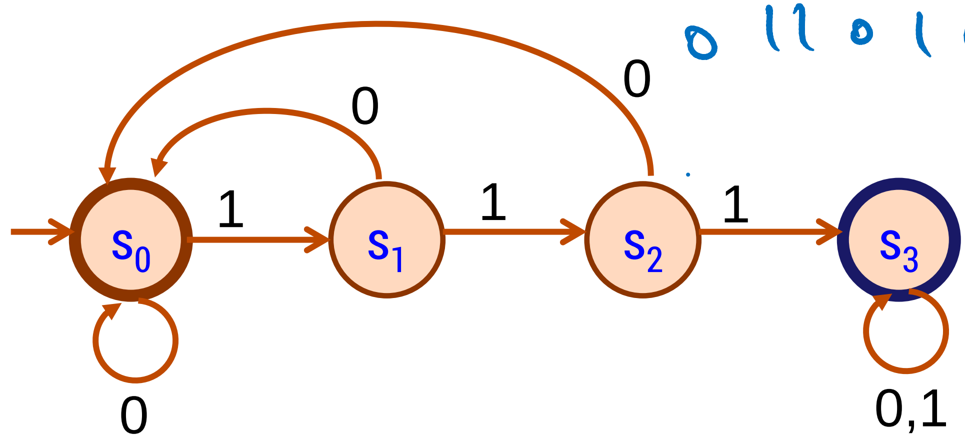
review: finite state machines

- States
- Transitions on inputs
- Start state and final states
- The language recognized by a machine is the set of strings that reach a final state

0 1110 ∈ L

0 1101 ∉ L

State	0	1
s_0	s_0	s_1
s_1	s_0	s_2
s_2	s_0	s_3
s_3	s_3	s_3



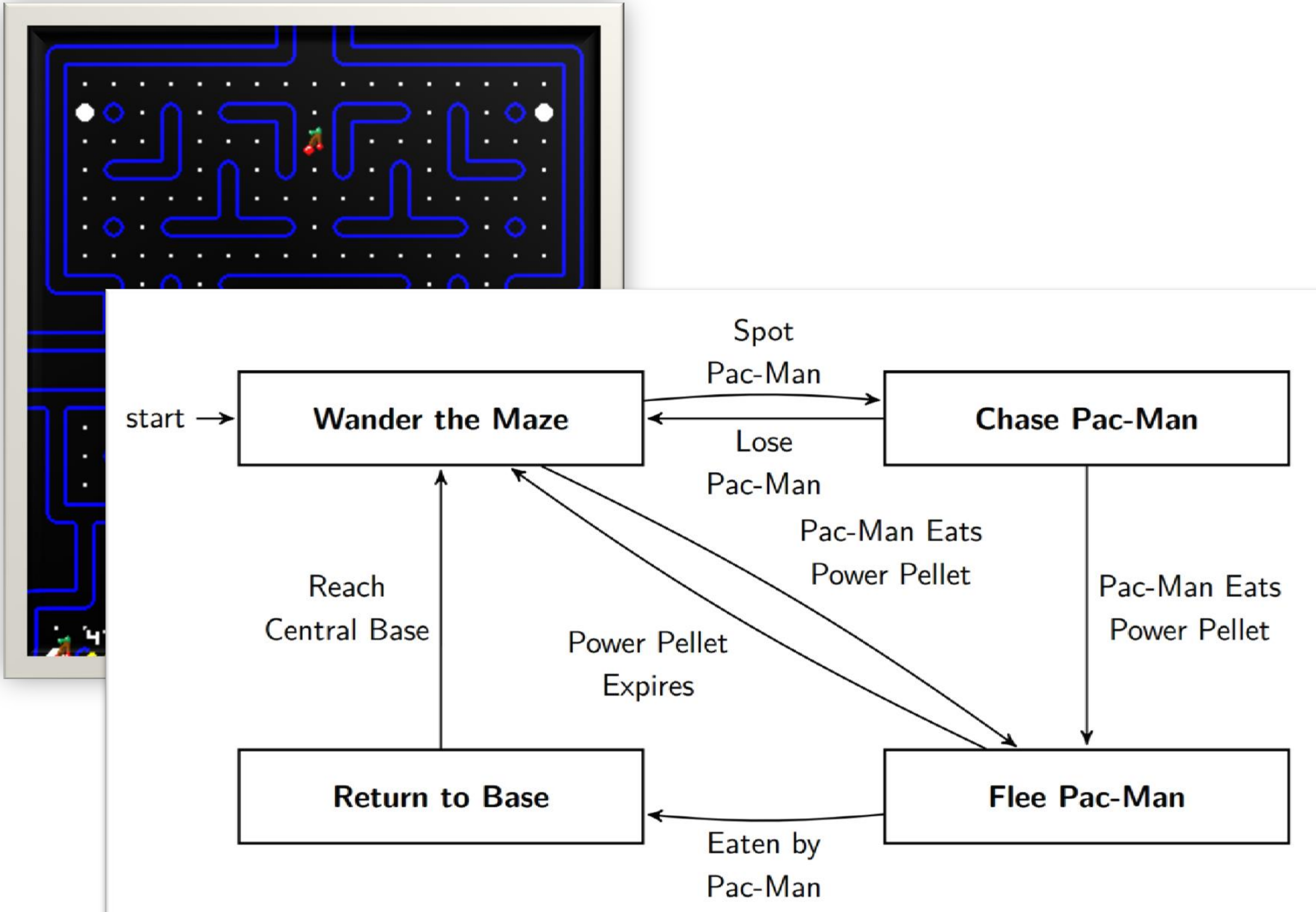
applications of FSMs (aka finite automata)

- Implementation of regular expression matching in programs like `grep`
- Control structures for sequential logic in digital circuits
- Algorithms for communication and cache-coherence protocols
  - Each agent runs its own FSM
- Design specifications for reactive systems
  - Components are communicating FSMs

applications of FSMs (aka finite automata)

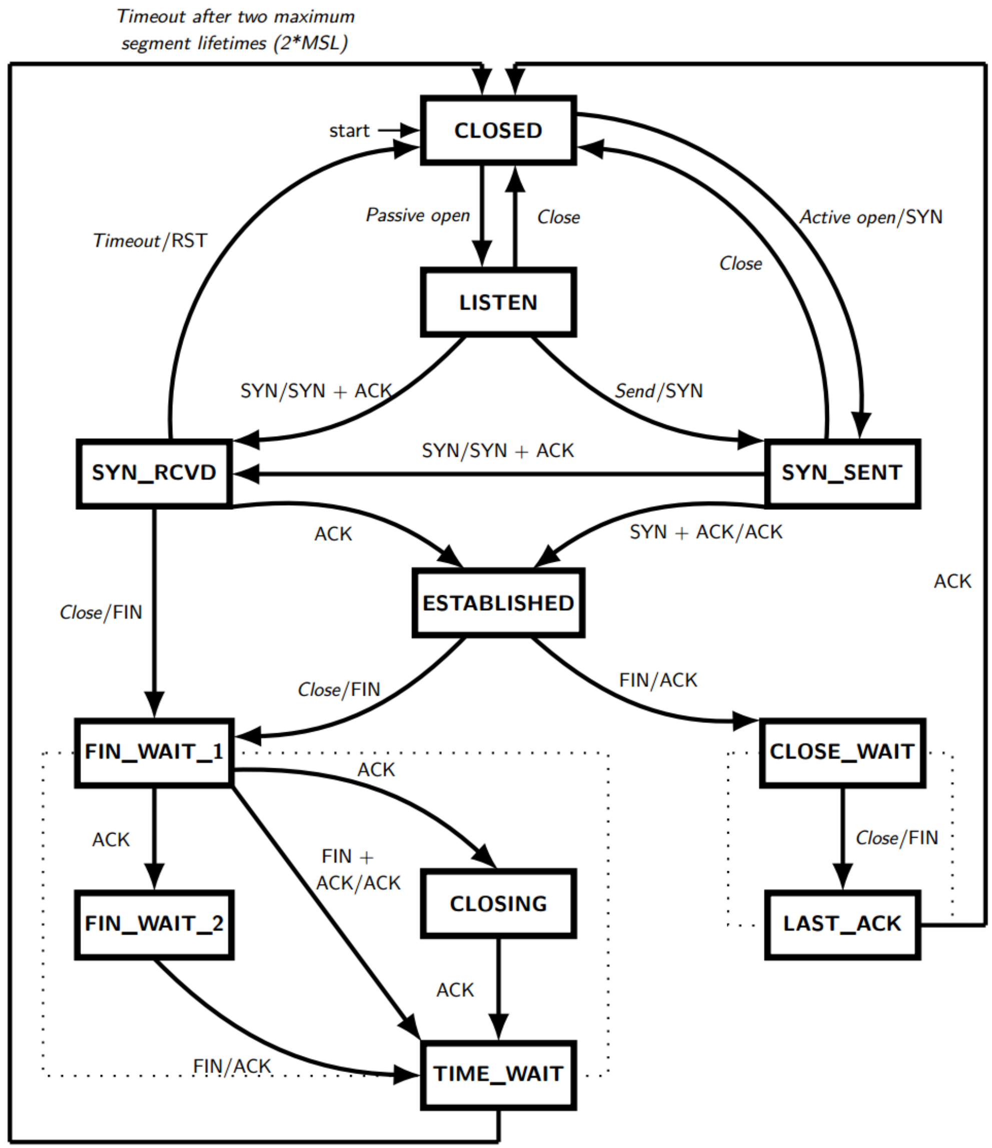
- Formal verification of systems
  - Is an unsafe state reachable?
- Computer games
  - FSMs provide worlds to explore
  - Character AI
- Minimization algorithms for FSMs can be extended to more general models used in
  - Text prediction
  - Speech recognition

waka waka



start
Wander the Maze
Spot
Pac-Man
Chase Pac-Man
Lose
Pac-Man
Pac-Man Eats
Power Pellet
Pac-Man Eats
Power Pellet
Reach
Central Base
Power Pellet
Expires
Return to Base
Flee Pac-Man
Eaten by
Pac-Man

tcp



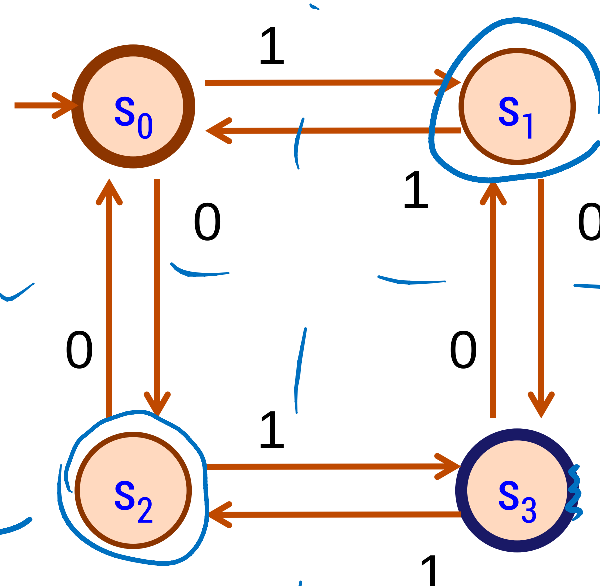
Timeout after two maximum
segment lifetimes (2*MSL)
start
CLOSED
Passive open
Close
Timeout/RST
Active open/SYN
Close
LISTEN
SYN/SYN + ACK
Send/SYN
SYN_RCVD
SYN/SYN + ACK
SYN_SENT
ACK
SYN + ACK/ACK
ESTABLISHED
ACK
Close/FIN
Close/FIN
FIN/ACK
FIN_WAIT_1
CLOSE_WAIT
ACK
ACK
FIN +
ACK/ACK
Close/FIN
FIN_WAIT_2
CLOSING
LAST_ACK
ACK
FIN/ACK
TIME_WAIT

what language does this machine recognize?

1 0 1 0 1 0

1 1 0 1

L = { strings with odd number of 0s and odd # of 1s }



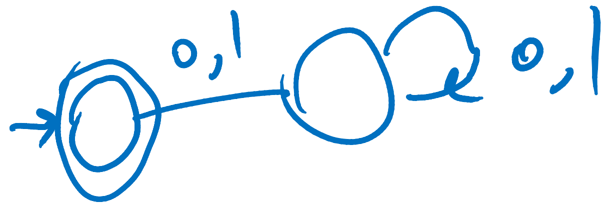
→ L = { odd # of 1s, even # of 0s }

L = { odd # of 0s, even # of 1s }

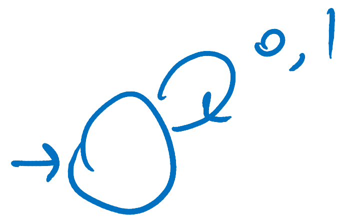
P(n) = { ∀ s: len(s) = n, if $\#_0(s) = odd$, $\#_1(s) = odd$ → s_3 }

can we recognize these languages with DFAs?

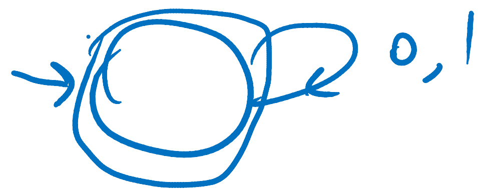
- ε



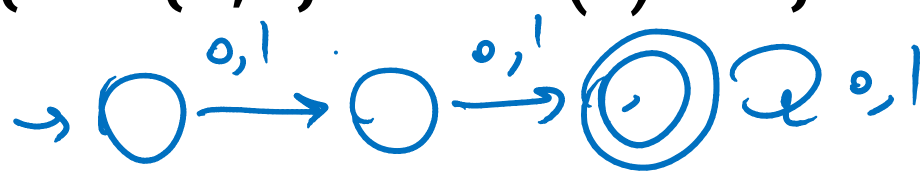
- $\emptyset$



- Σ^*

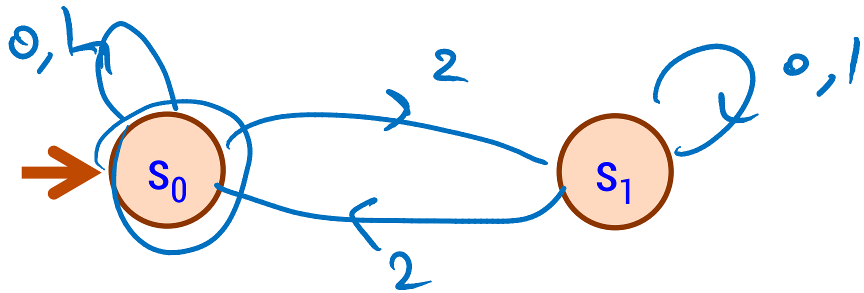


- $\{ x \in \{0,1\}^* : \text{len}(x) > 1\}$

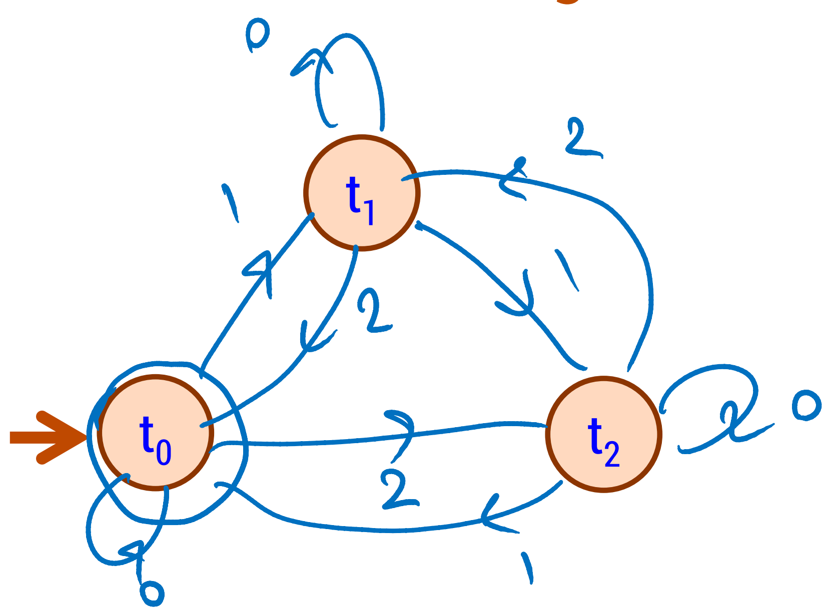


strings over {0, 1, 2}*

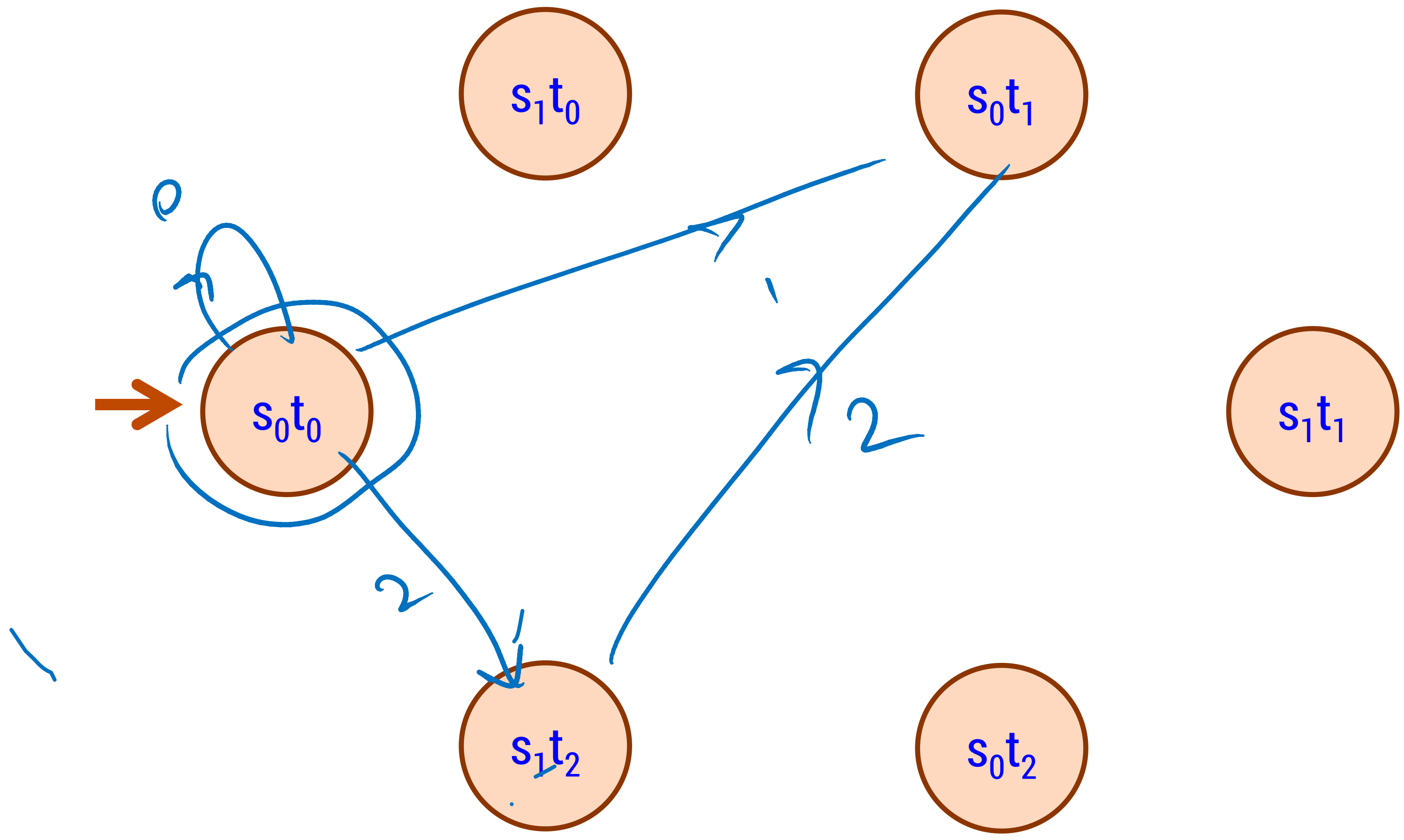
M_1: Strings with an even number of 2's



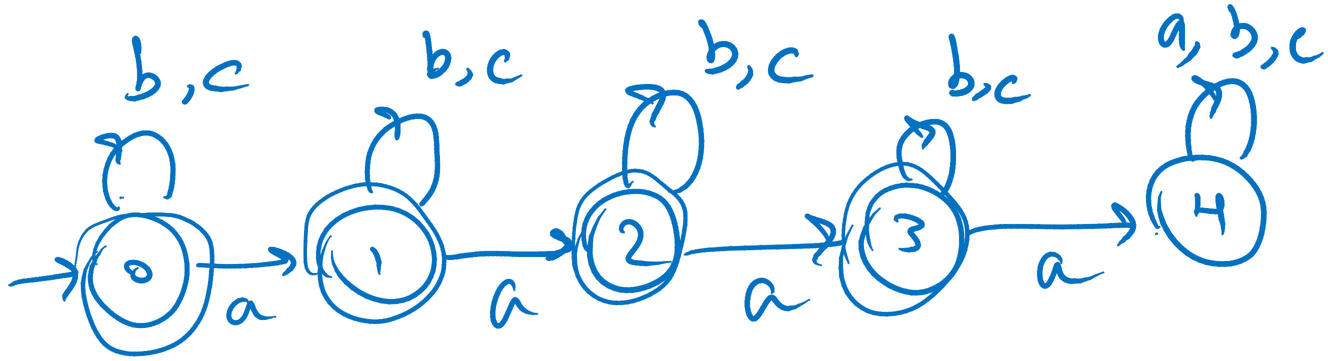
M_2: Strings where the sum of digits mod 3 is 0



both: even number of 2's and sum mod 3 = 0

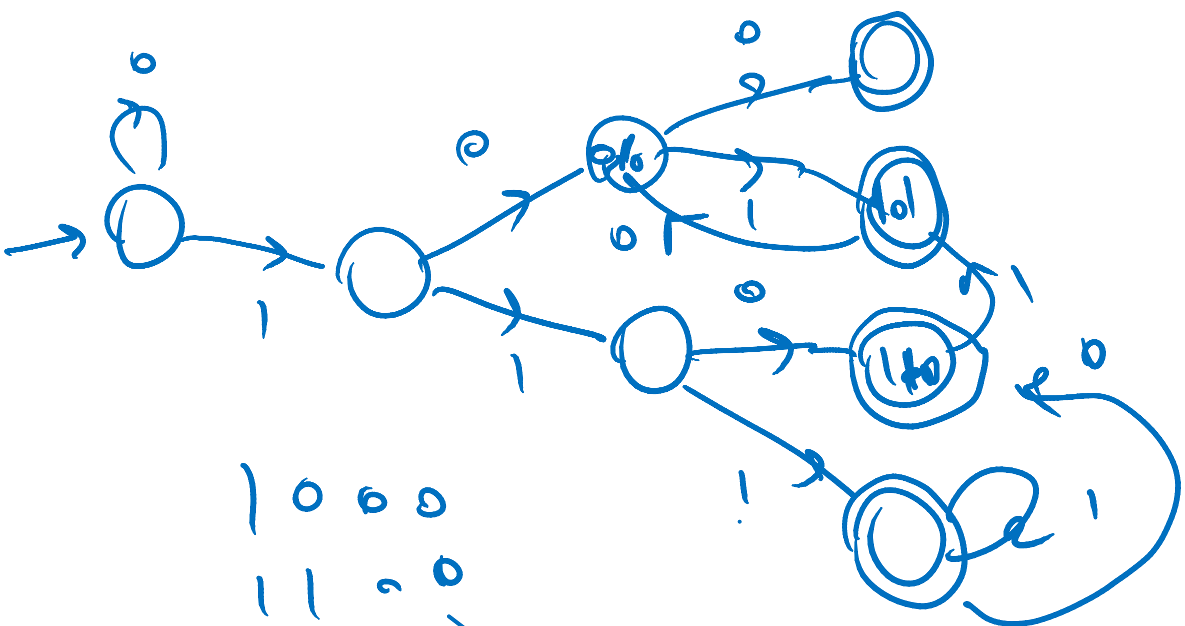


DFA that accepts strings of a's, b's, c's with no more than 3 a's



FSM that accepts binary strings with a 1 three positions from the end

00100 ∈ L
00010 ∉ L
00110 ∈ L

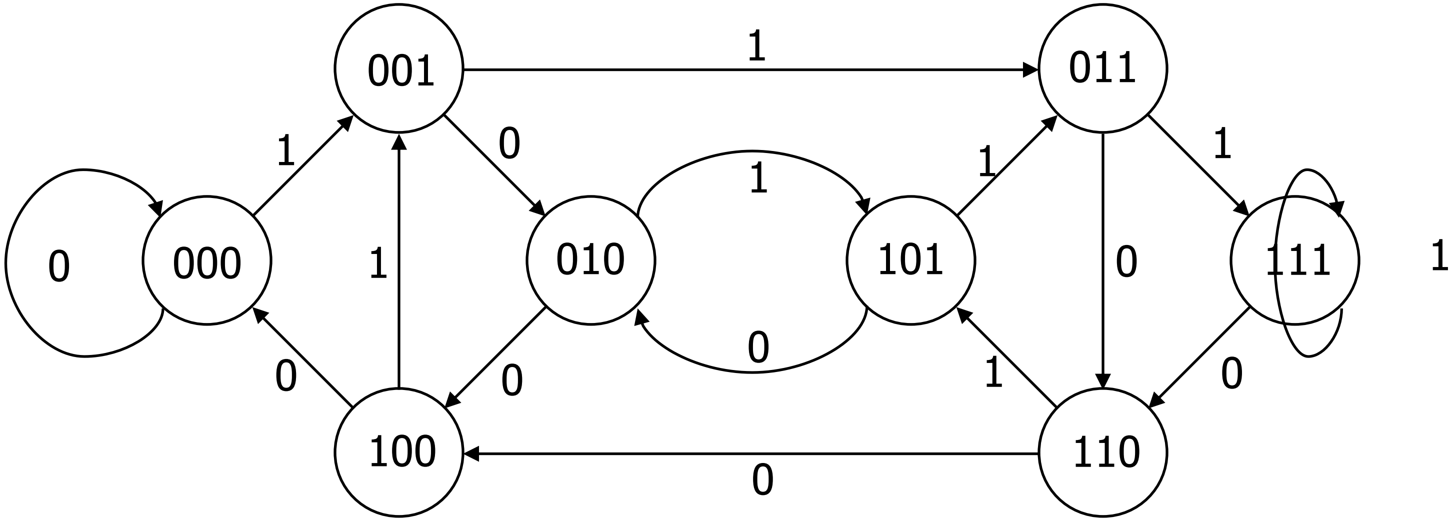


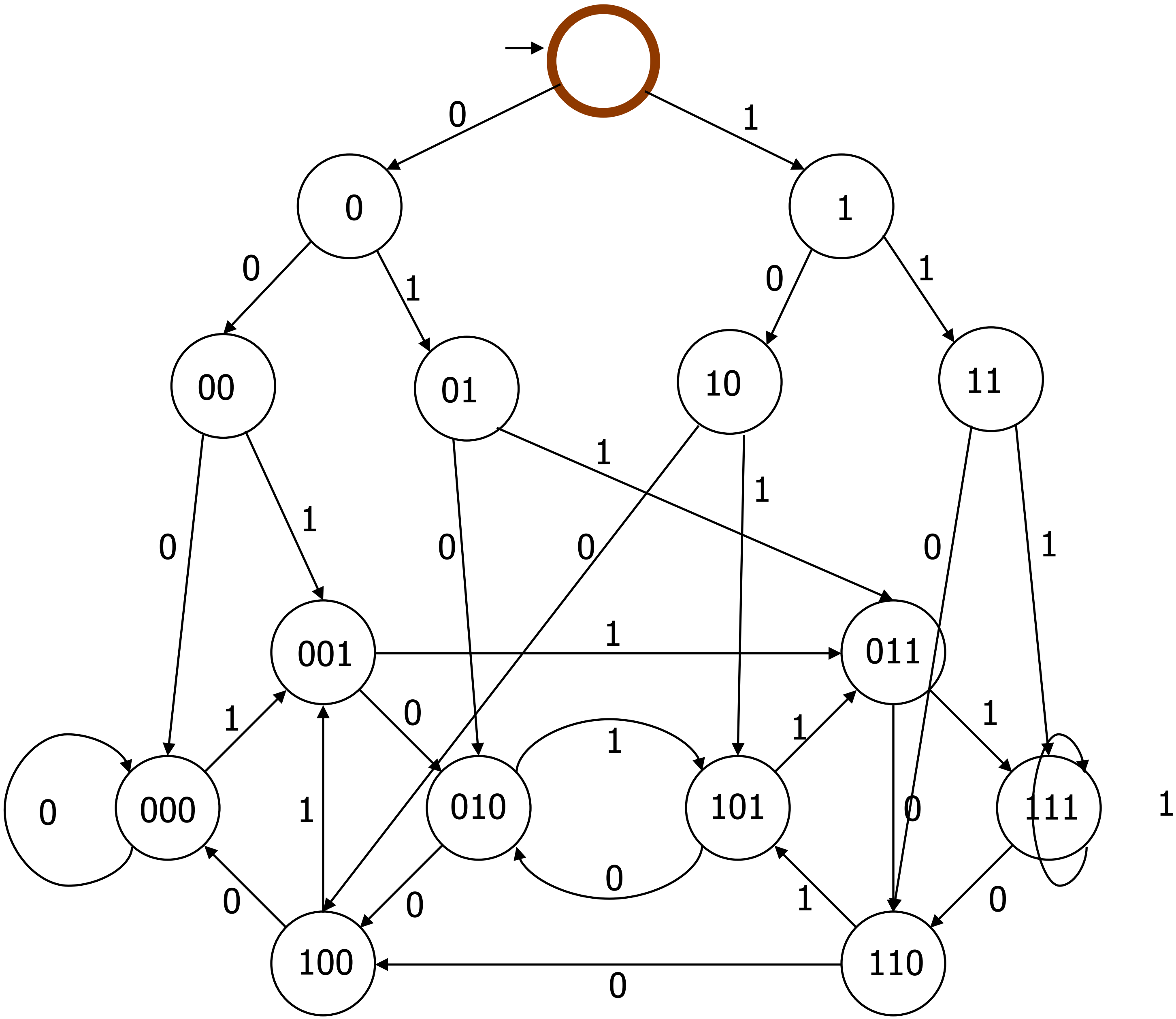
1000
1100
11100

3 bit shift register

"Remember the last three bits"

000 →(1) 001 → 010 → 101 - - -

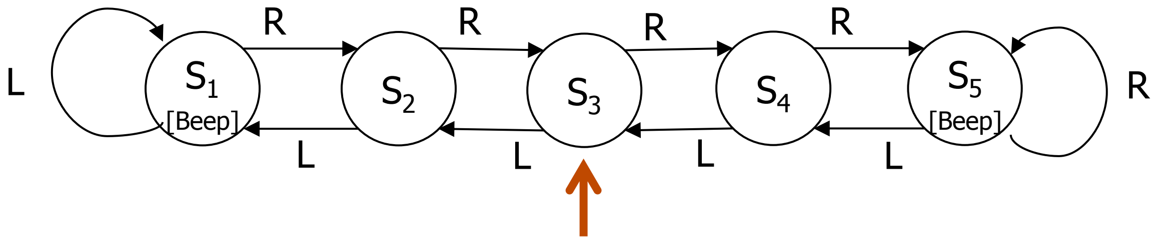



0
1
0
1
0
1
0
1
00
01
10
11
0
1
0
1
0
0
1
0
1
001
011
1
1
0
1
1
0
000
1
010
1
101
0
111
1
0
0
1
0
0
100
110
0

FSMs with output

"Tug-of-war"

	Input		Output
State	L	R	
s_1	s_1	s_2	Beep
s_2	s_1	s_3	
s_3	s_2	s_4	
s_4	s_3	s_5	
s_5	s_4	s_5	Beep



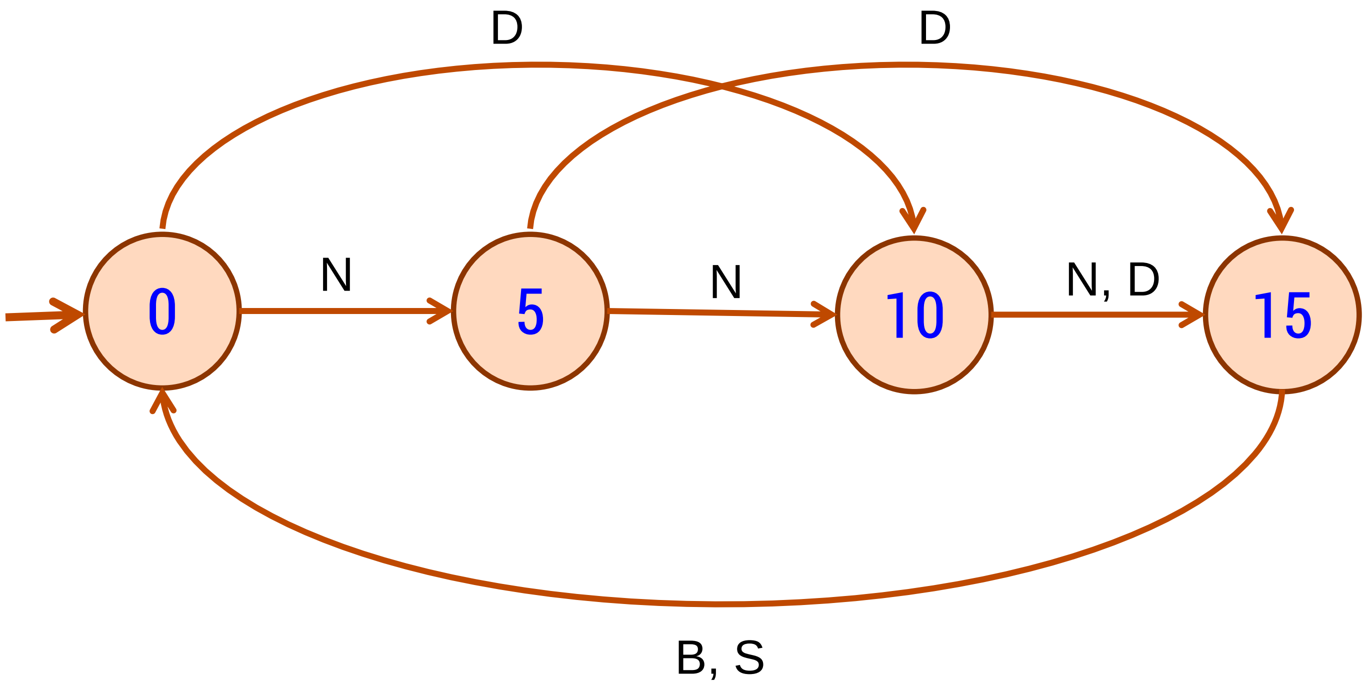
vending machine

We're only making $5.50/hour writing regular expressions.

Let's design a vending machine.

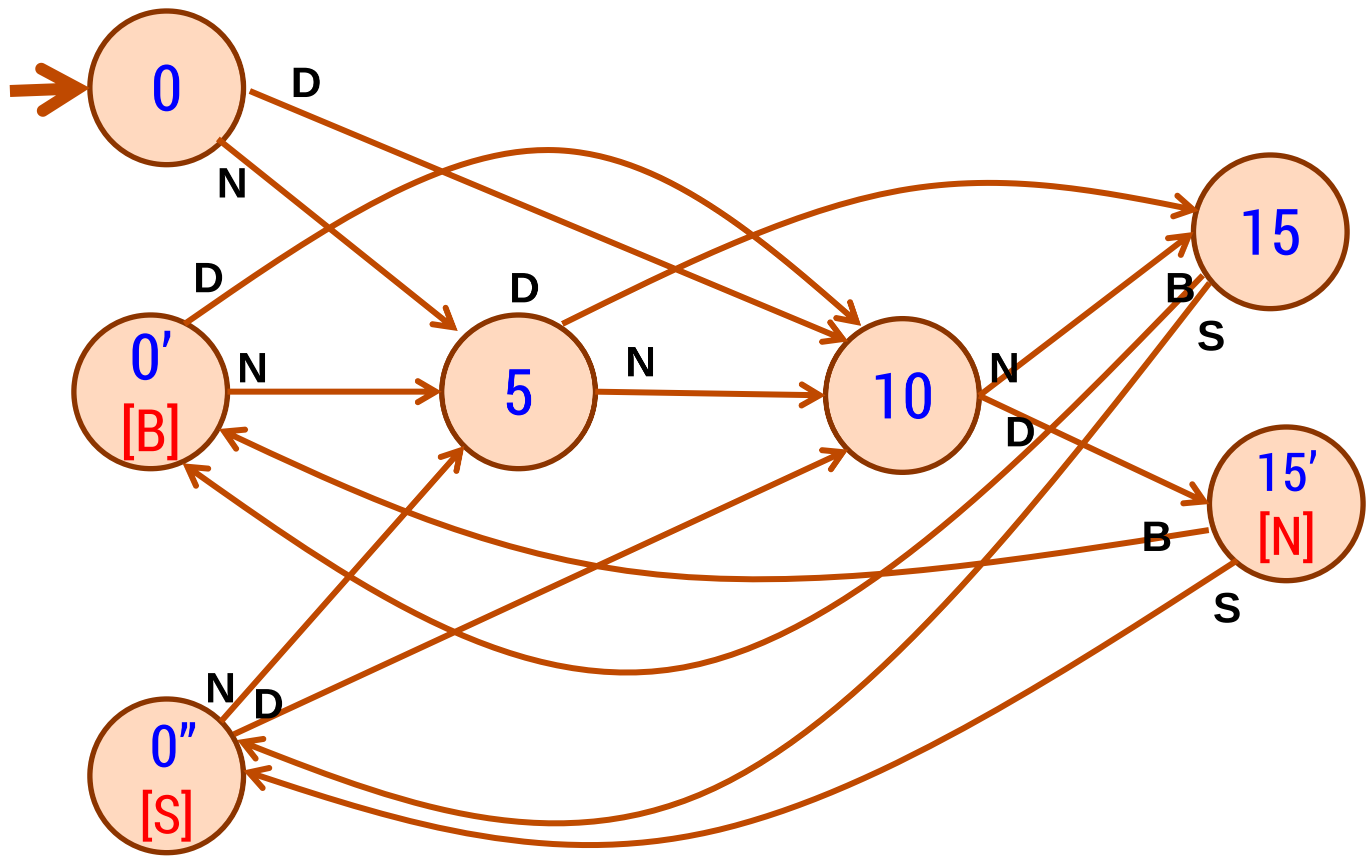
Vending spec:
Enter 15 cents in dimes or nickels
Press **S** or **B** for a candy bar

vending machine v0.1



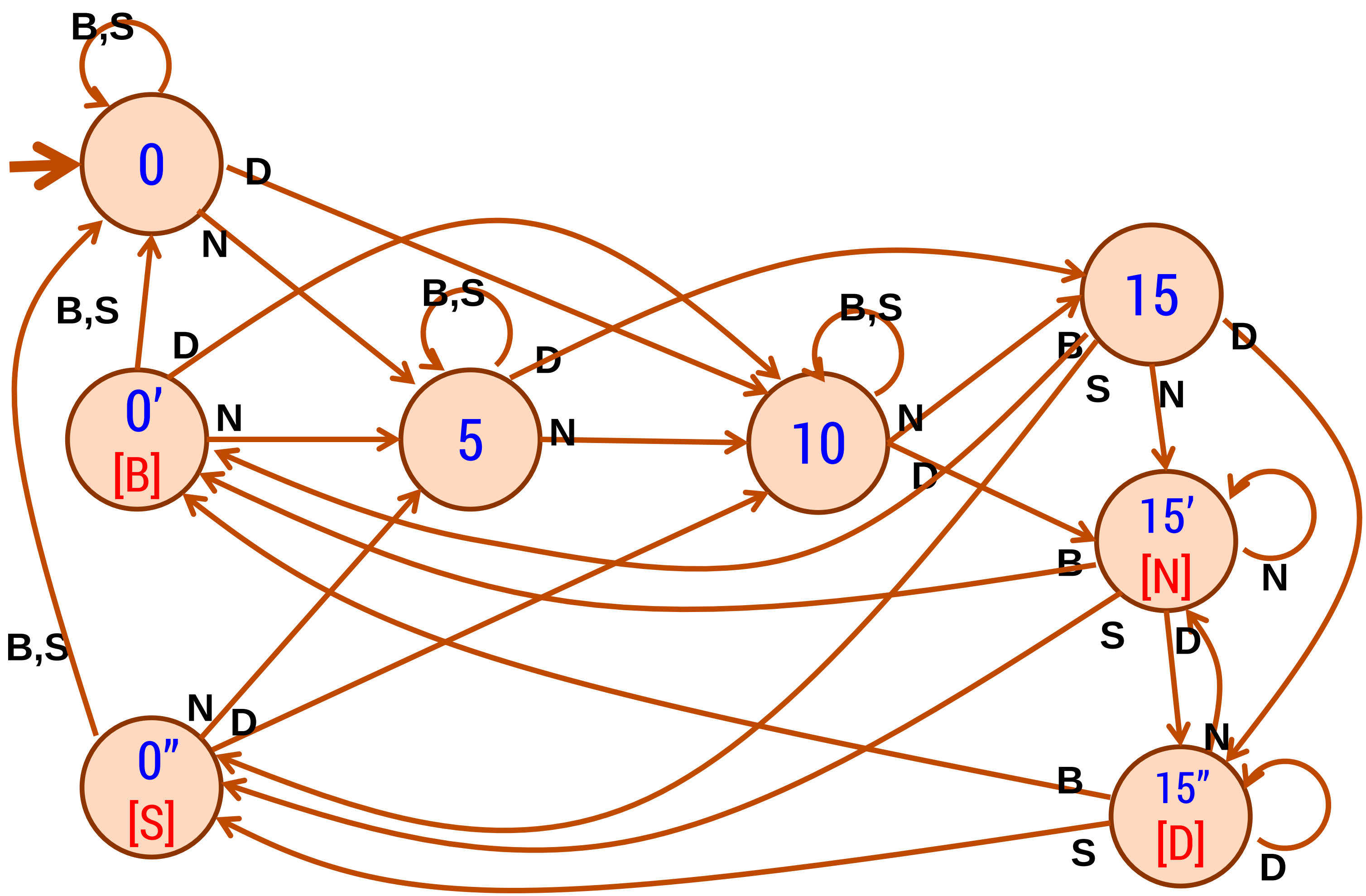
Basic transitions on N (nickel), D (dime), B (butterfinger), S (snickers)

vending machine v0.2



Adding output to states: N – Nickel, S – Snickers, B – Butterfinger

vending machine v1.0



Adding additional “unexpected” transitions