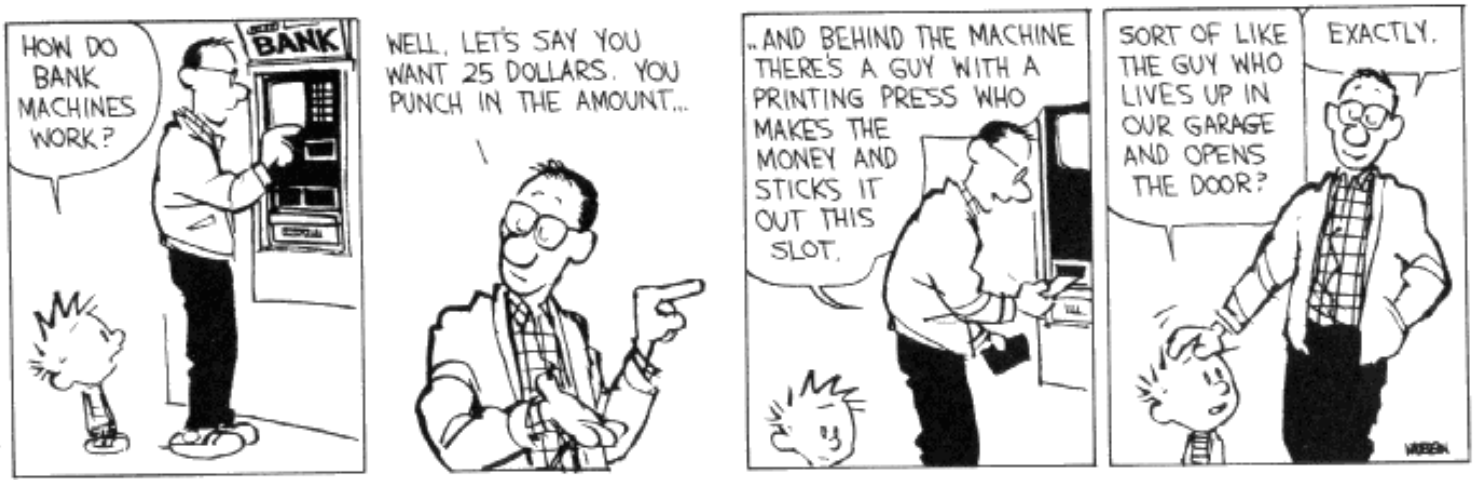
CSE 311: Foundations of Computing

Lecture 22: DFAs and Finite State Machines with Output



Reminders

- HW7 due next Wednesday
  - slightly longer than HW6, so 110 points
  - includes a more interesting structural induction

    defines 1 data type, 3 functions, and then proves some things about them
  - start early!

- If you want to test out your grammars, you can give this a try:

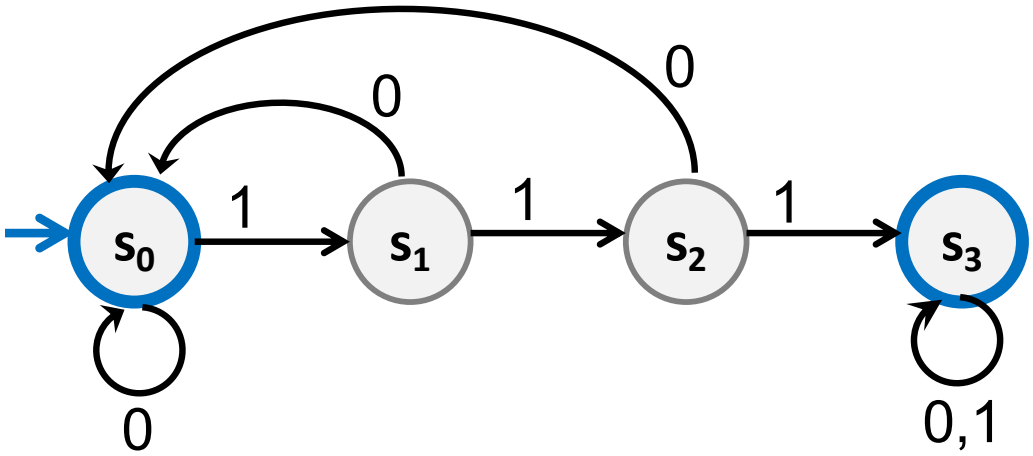
  https://homes.cs.washington.edu/~kevinz/grammar-test/

  Instructions on the page

Last class: Finite State Machines

- States
- Transitions on input symbols
- Start state and final states
- The "language recognized" by the machine is the set of strings that reach a final state from the start

Old 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

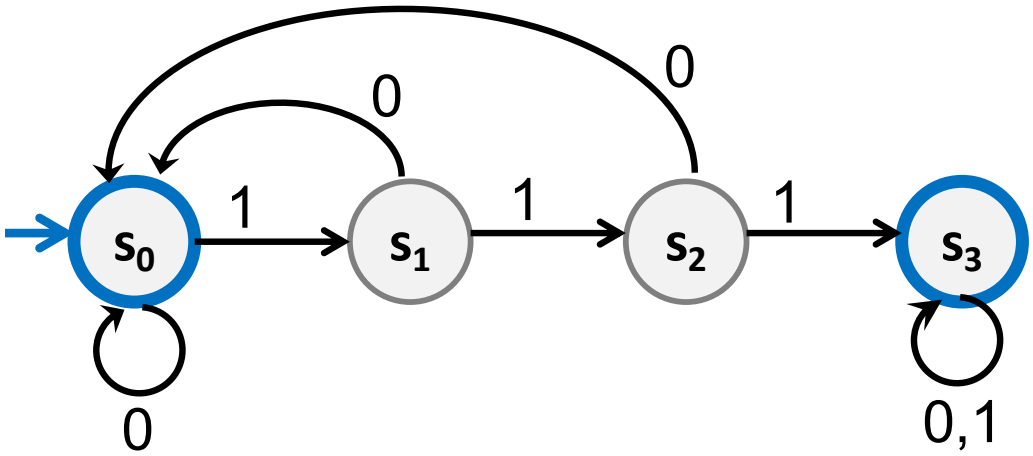


Last class: Finite State Machines

- Each machine designed for strings over some fixed alphabet Σ.

- Must have a transition defined from each state for ***every*** symbol in Σ.

Old 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

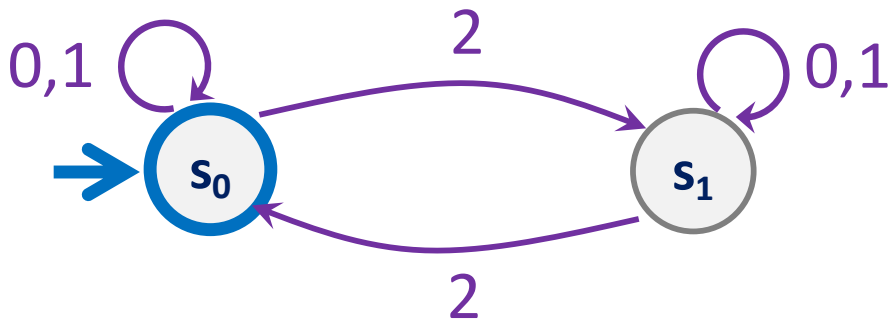


Strings over {0, 1, 2}

M_1: Strings with an even number of 2's

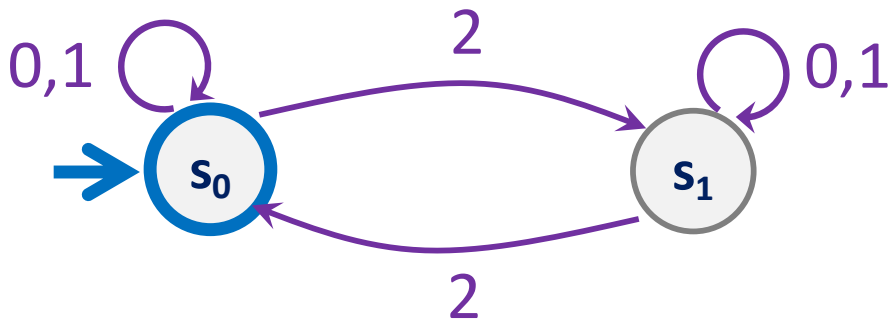
Strings over {0, 1, 2}

M_1: Strings with an even number of 2's

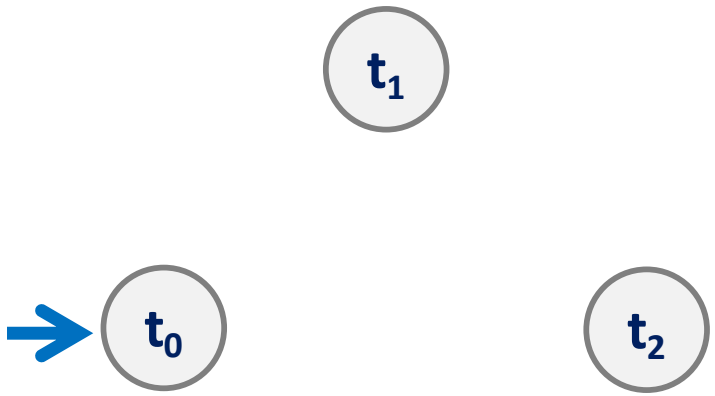


Strings over {0, 1, 2}

M_1: Strings with an even number of 2's

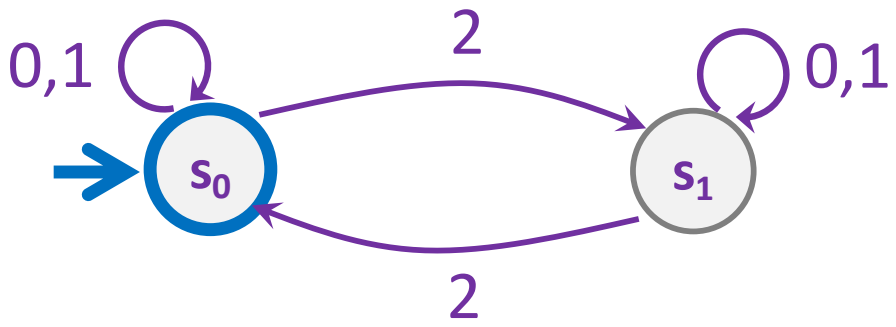


M_2: Strings where the sum of digits mod 3 is 0

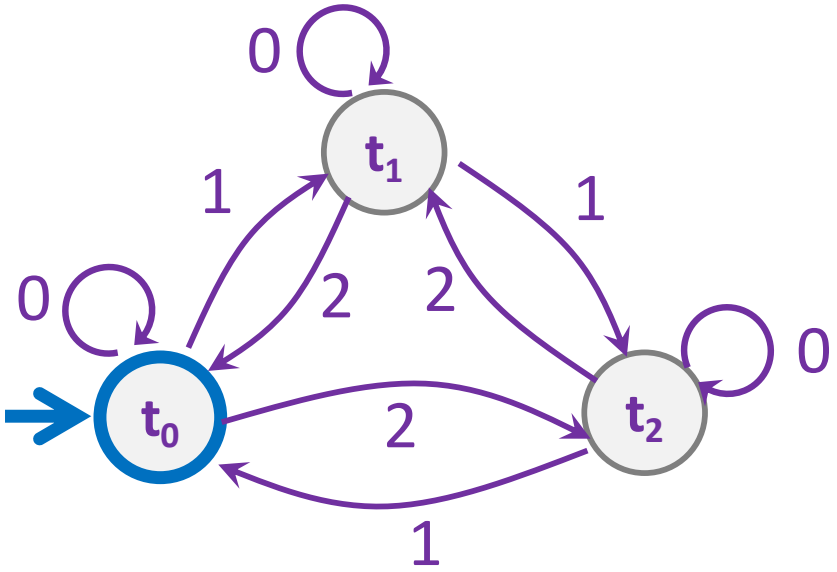


Strings over {0, 1, 2}

M_1: Strings with an even number of 2's



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



FSM as abstraction of Java code

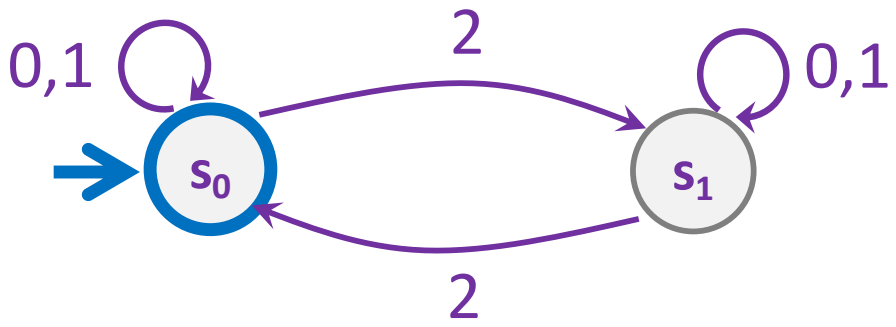
```
boolean sumCongruentToZero(String str) {
    int sum = 0;  // state
    for (int i = 0; i < str.length(); i++) {
        if (str.charAt(i) == '2')
            sum = (sum + 2) % 3;
        if (str.charAt(i) == '1')
            sum = (sum + 1) % 3;
    }
    return sum == 0;
}
```

FSM to Java code

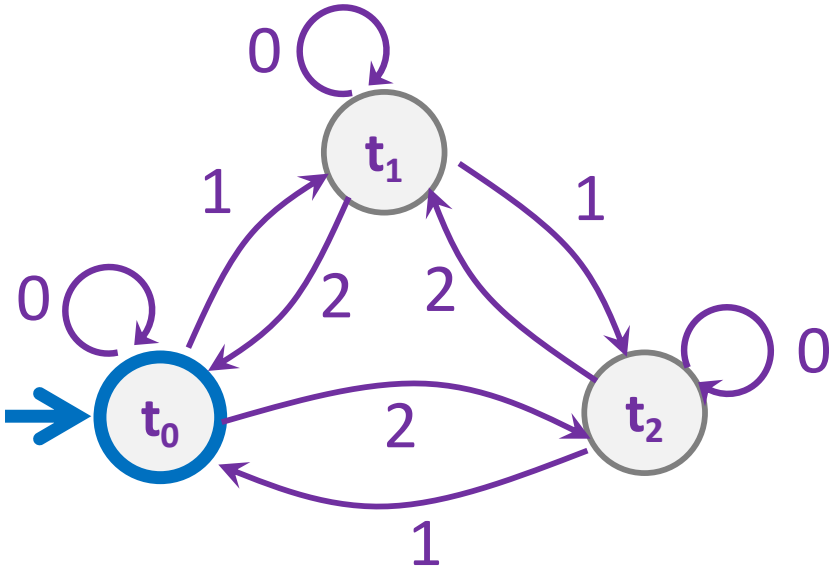
```
int[][] TRANSITION = {...};

boolean sumCongruentToZero(String str) {
   int state = 0;
   for (int i = 0; i < str.length(); i++) {
      int d = str.charAt(i) - '0';
      state = TRANSITION[state][d];
   }
   return state == 0;
}
```

Strings over {0, 1, 2}

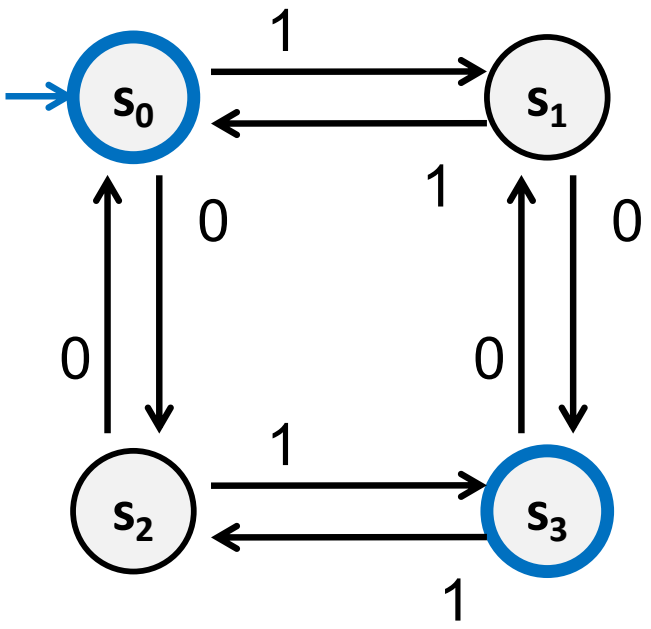
M_1: Strings with an even number of 2's



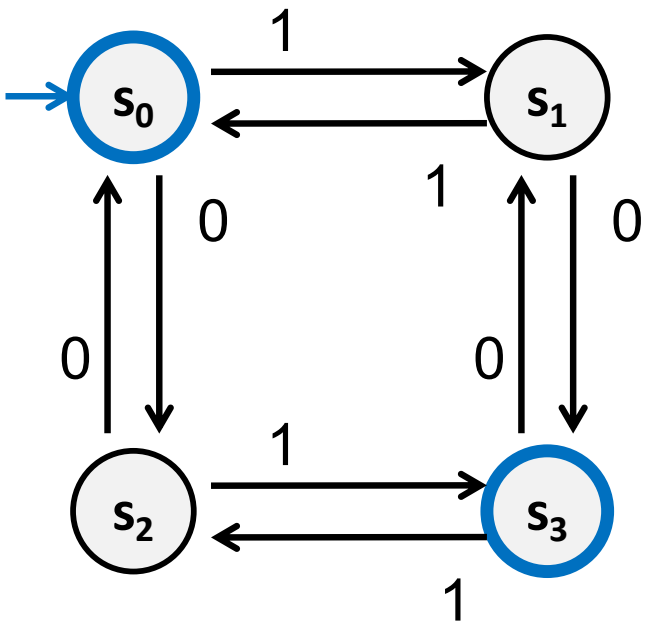
M_2: Strings where the sum of digits mod 3 is 0



What language does this machine recognize?



What language does this machine recognize?

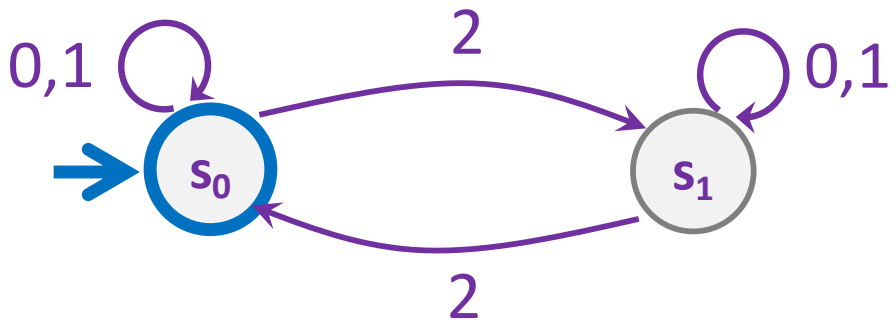


The set of all binary strings with # of 1's ≡ # of 0's (mod 2) (both are even or both are odd).

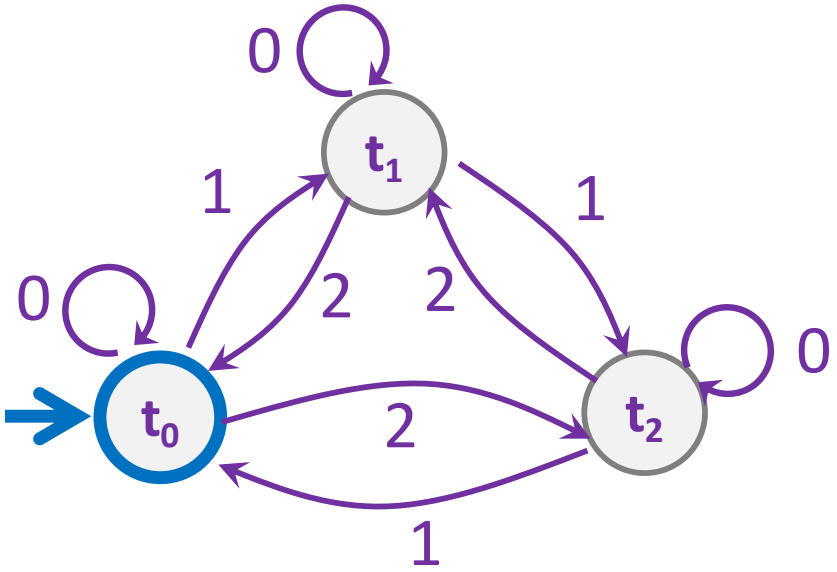
Can you think of a simpler description?

Strings over {0, 1, 2}

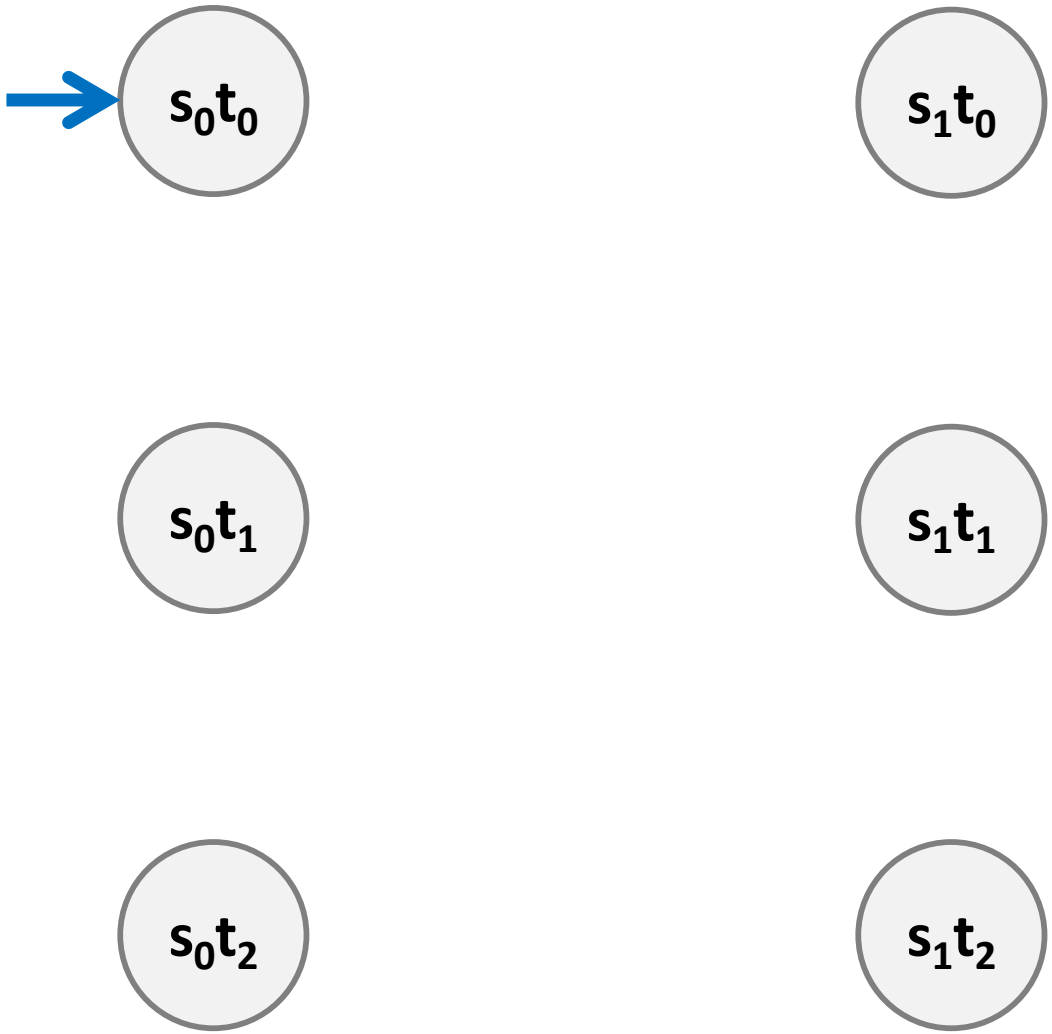
M_1: Strings with an even number of 2's



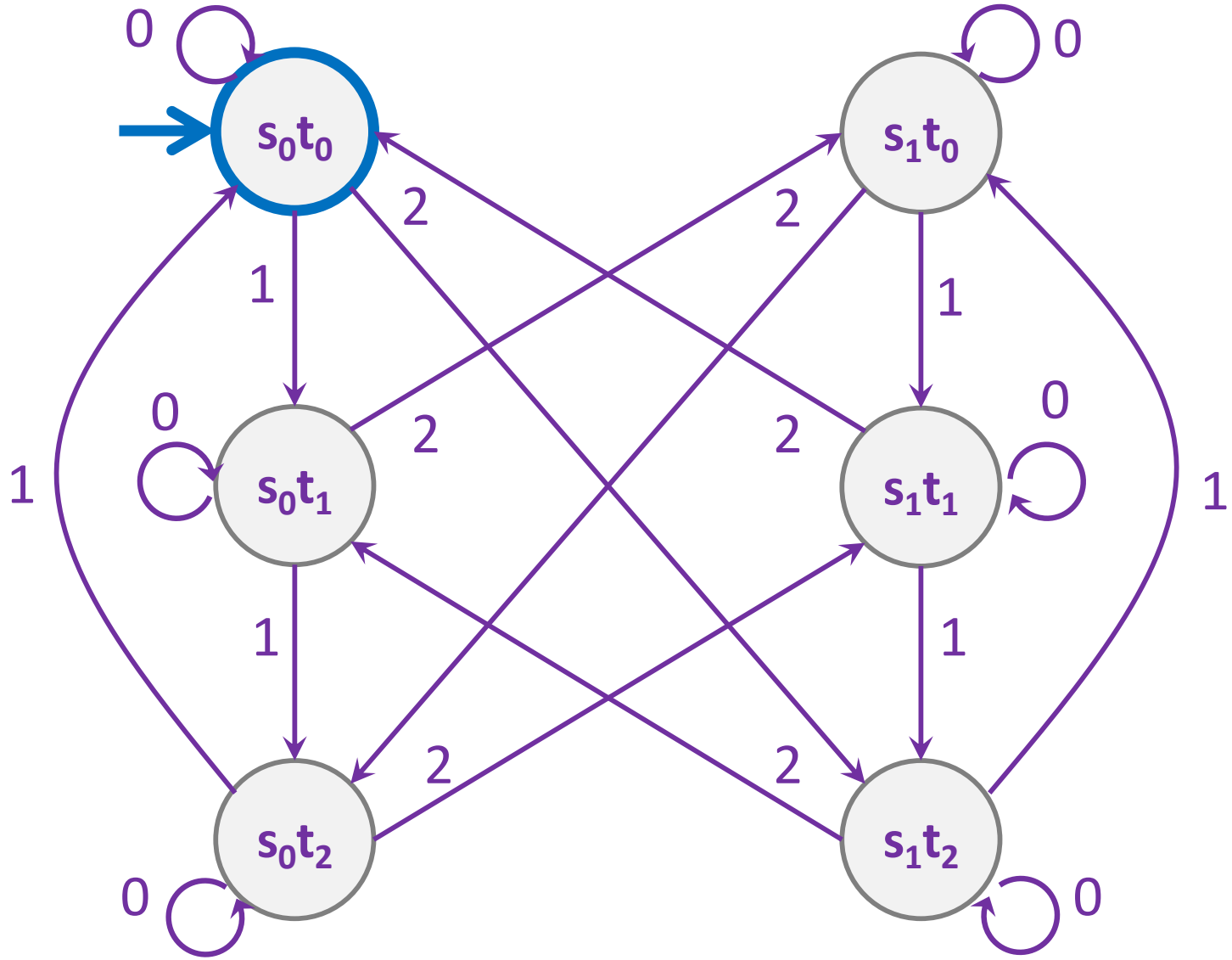
M_2: Strings where the sum of digits mod 3 is 0



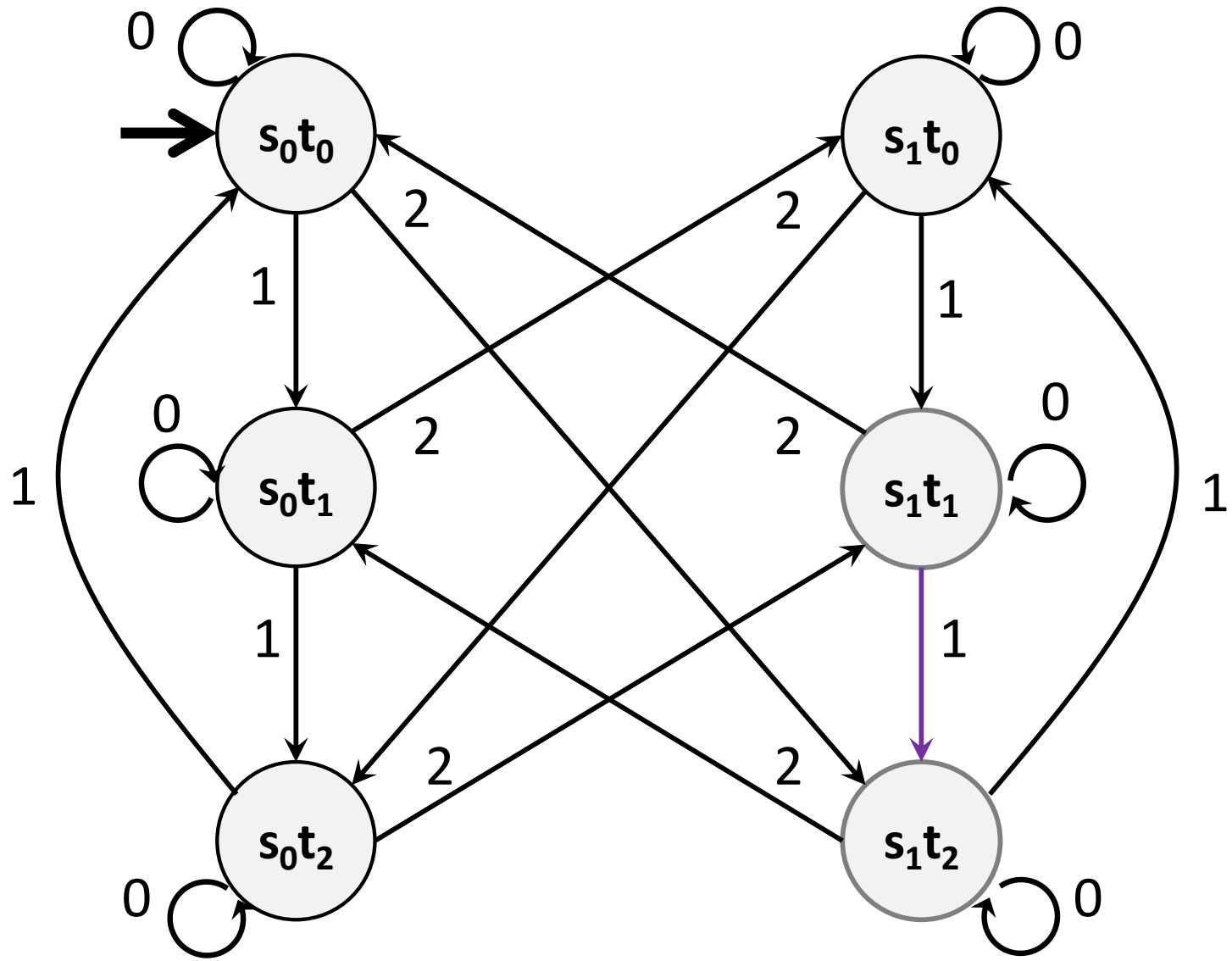
Strings over {0,1,2} w/ even number of 2's and mod 3 sum 0



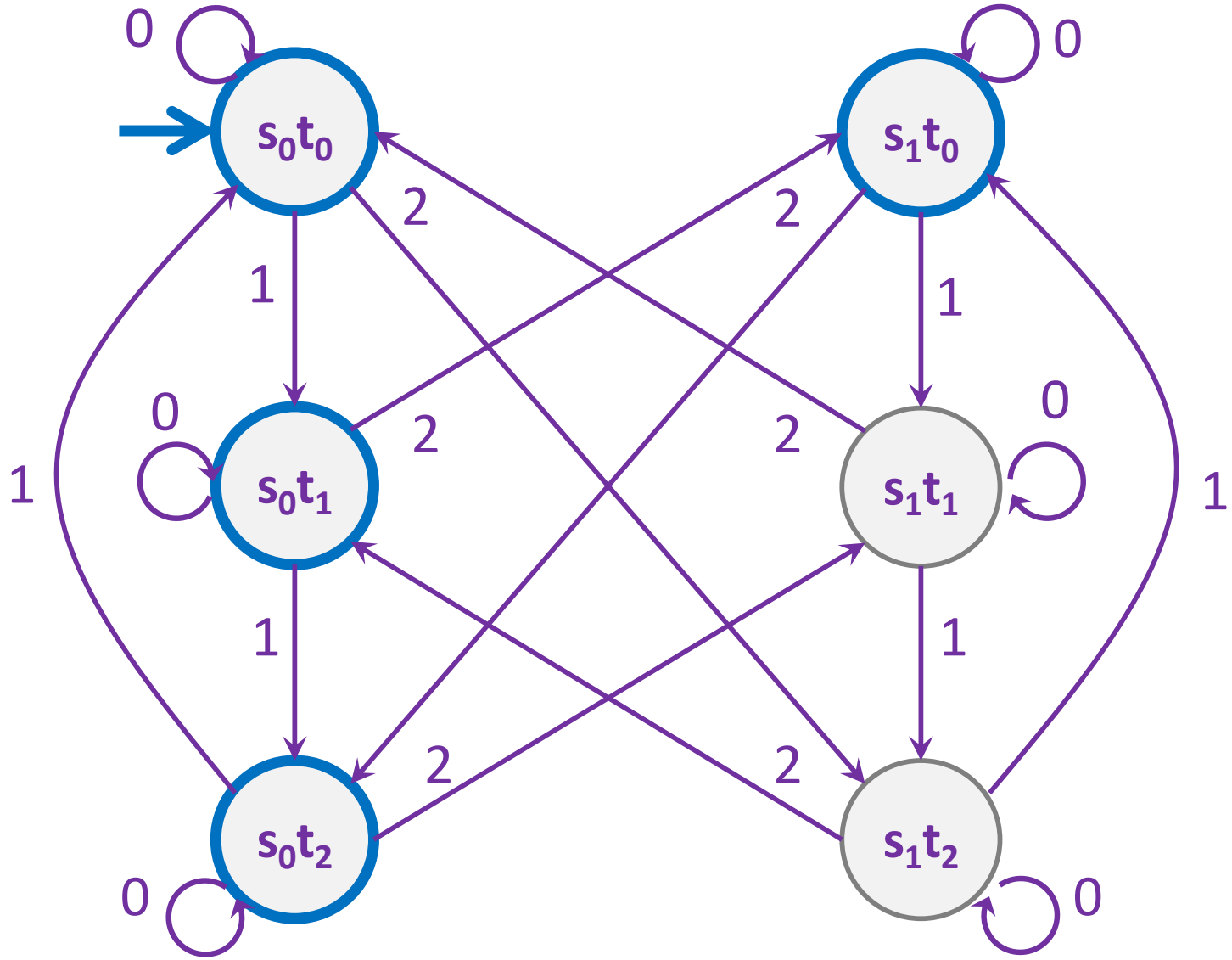
Strings over {0,1,2} w/ even number of 2's and mod 3 sum 0



Strings over {0,1,2} w/ even number of 2's OR mod 3 sum 0?

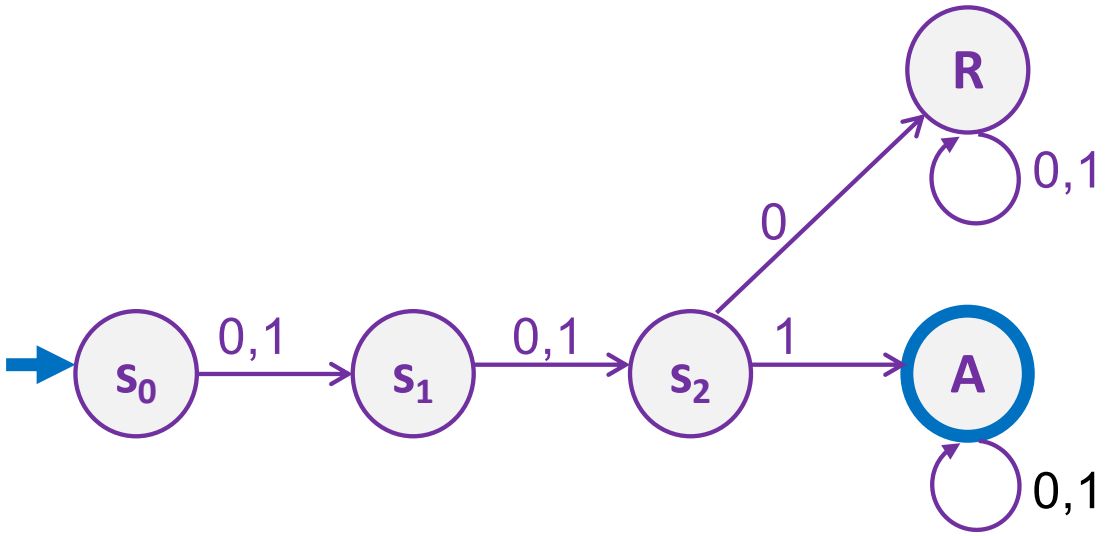


Strings over {0,1,2} w/ even number of 2's OR mod 3 sum 0



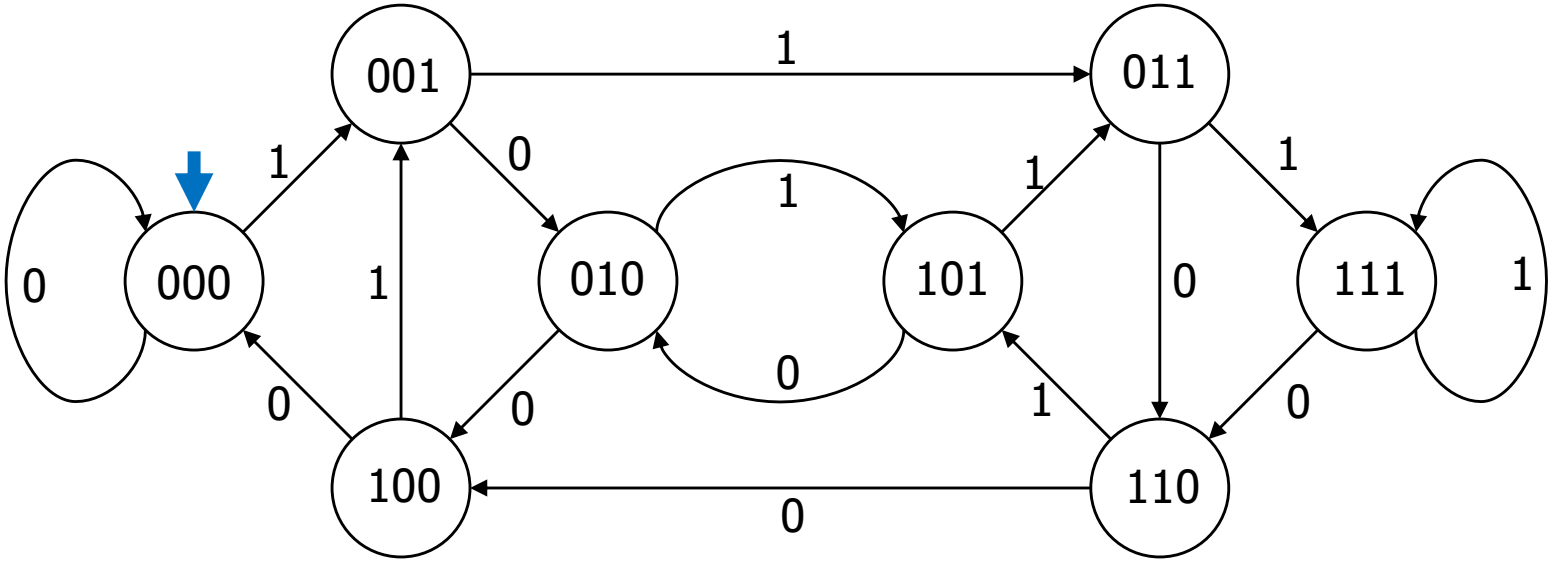
The set of binary strings with a 1 in the 3rd position from the start

The set of binary strings with a 1 in the 3rd position from the start

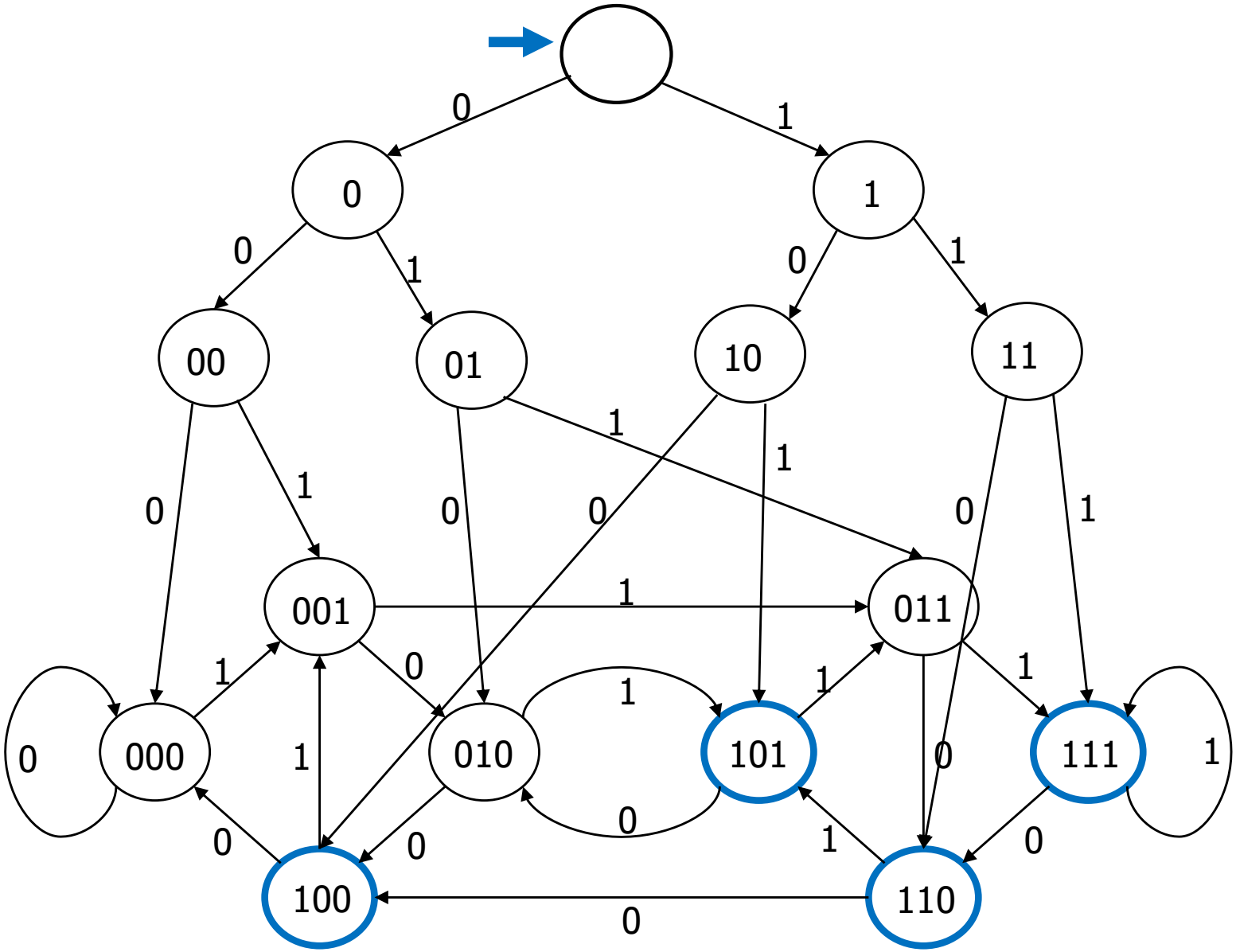


The set of binary strings with a 1 in the 3rd position from the end

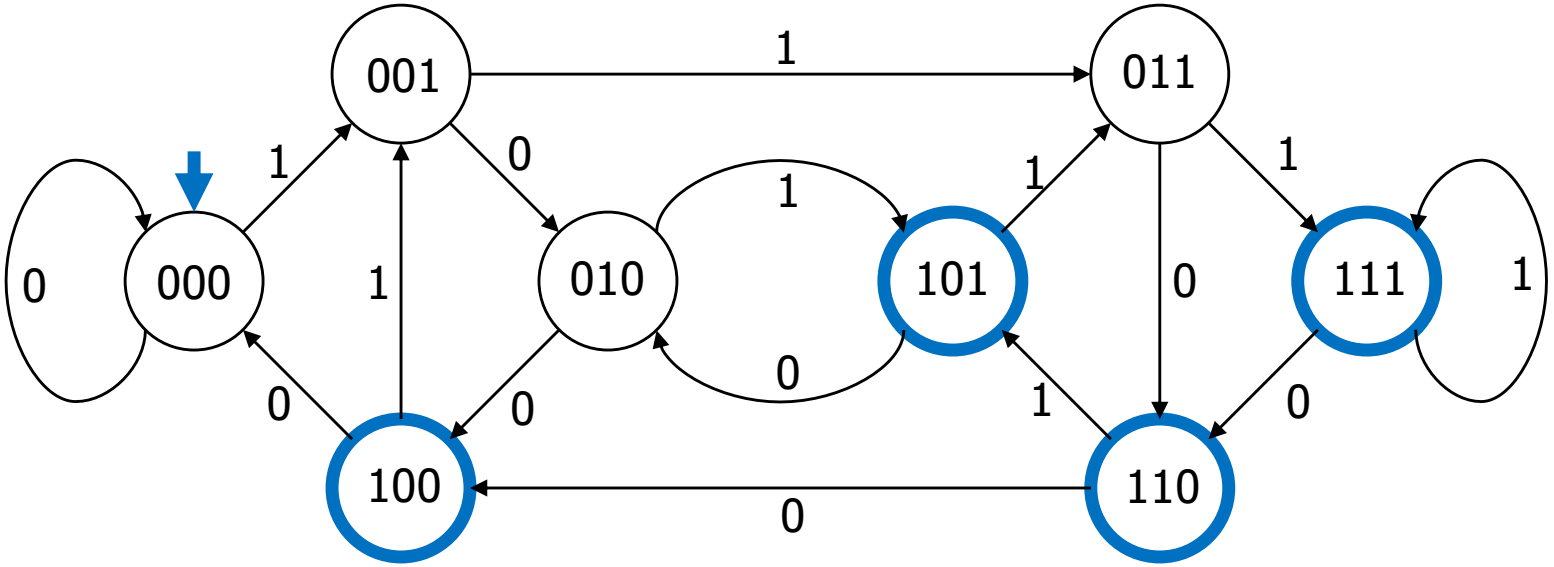
3 bit shift register "Remember the last three bits"



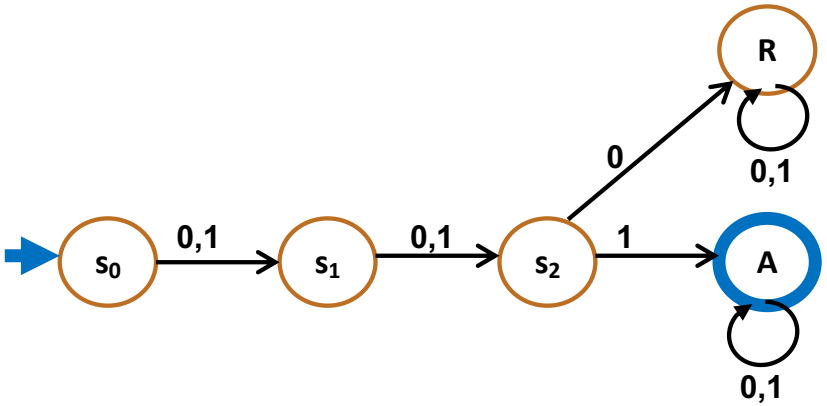
The set of binary strings with a 1 in the 3rd position from the end

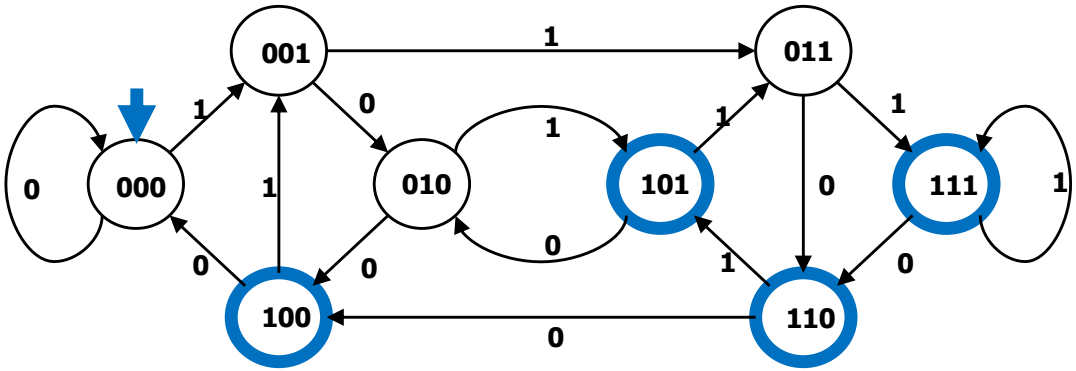


The set of binary strings with a 1 in the 3rd position from the end



The beginning versus the end





Adding Output to Finite State Machines

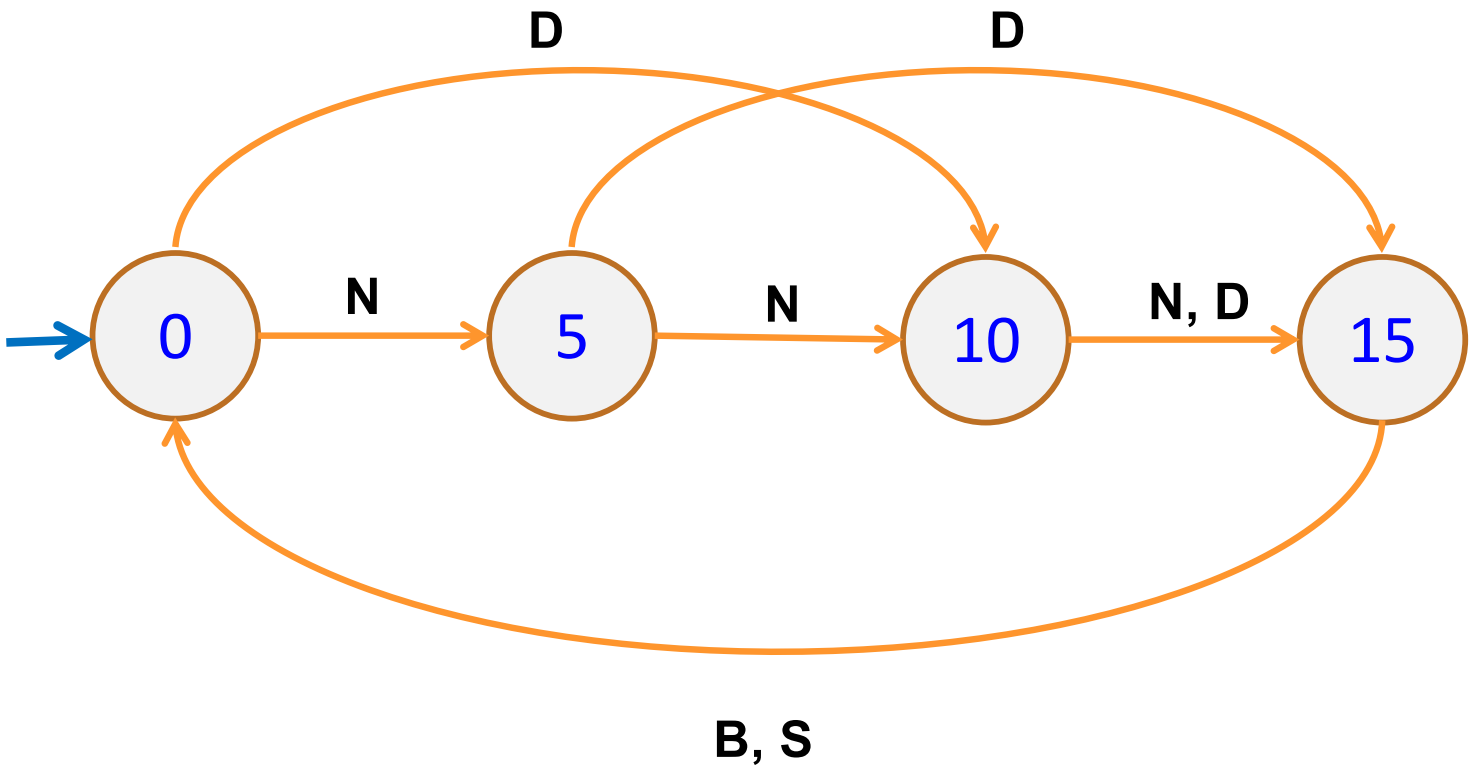
- So far we have considered finite state machines that just accept/reject strings
  - called “Deterministic Finite Automata” or DFAs
- Now we consider finite state machines *with output*
  - These are the kinds used as controllers

Vending Machine

Enter 15 cents in dimes or nickels

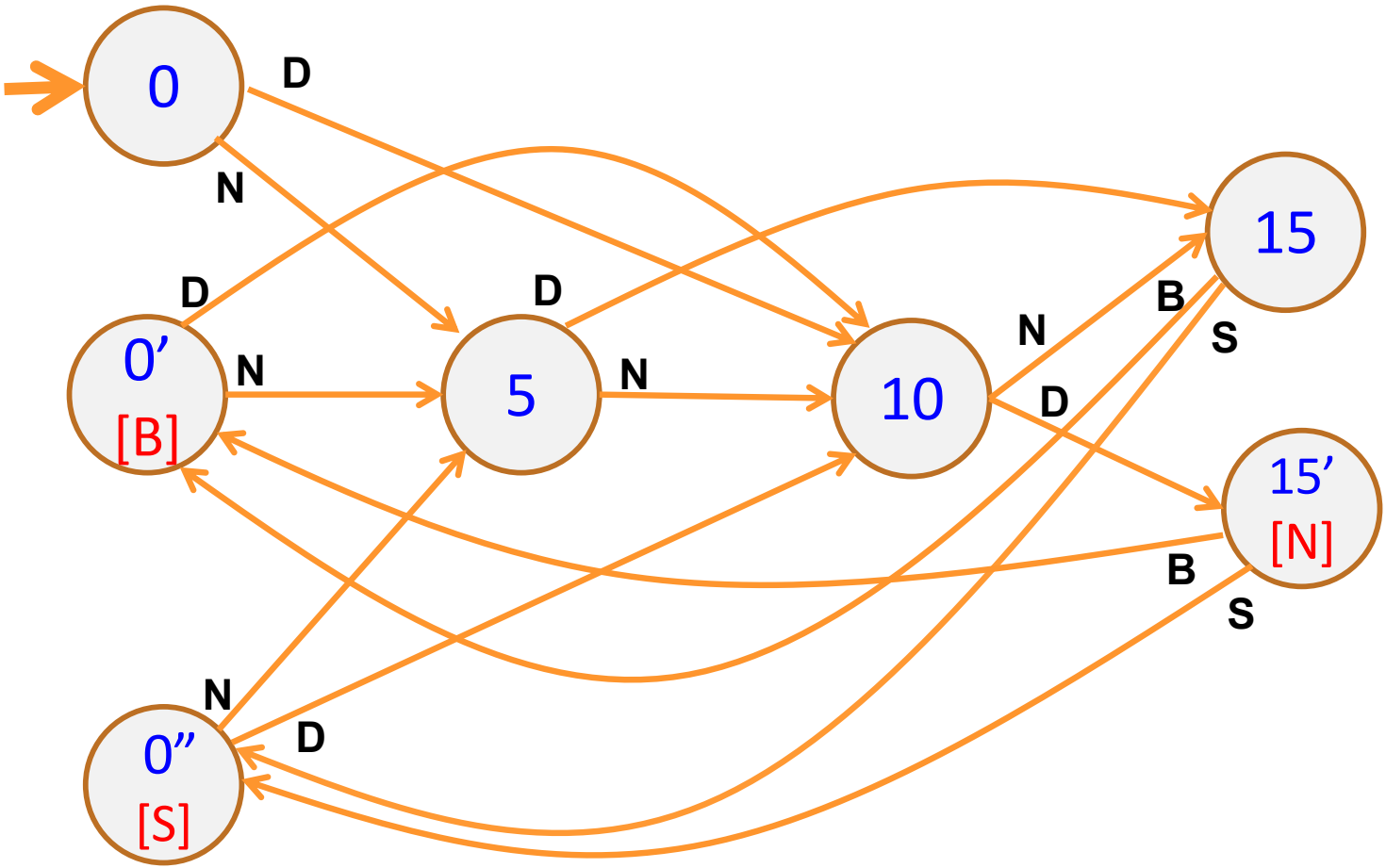
Press S or B for a candy bar

Vending Machine, v0.1



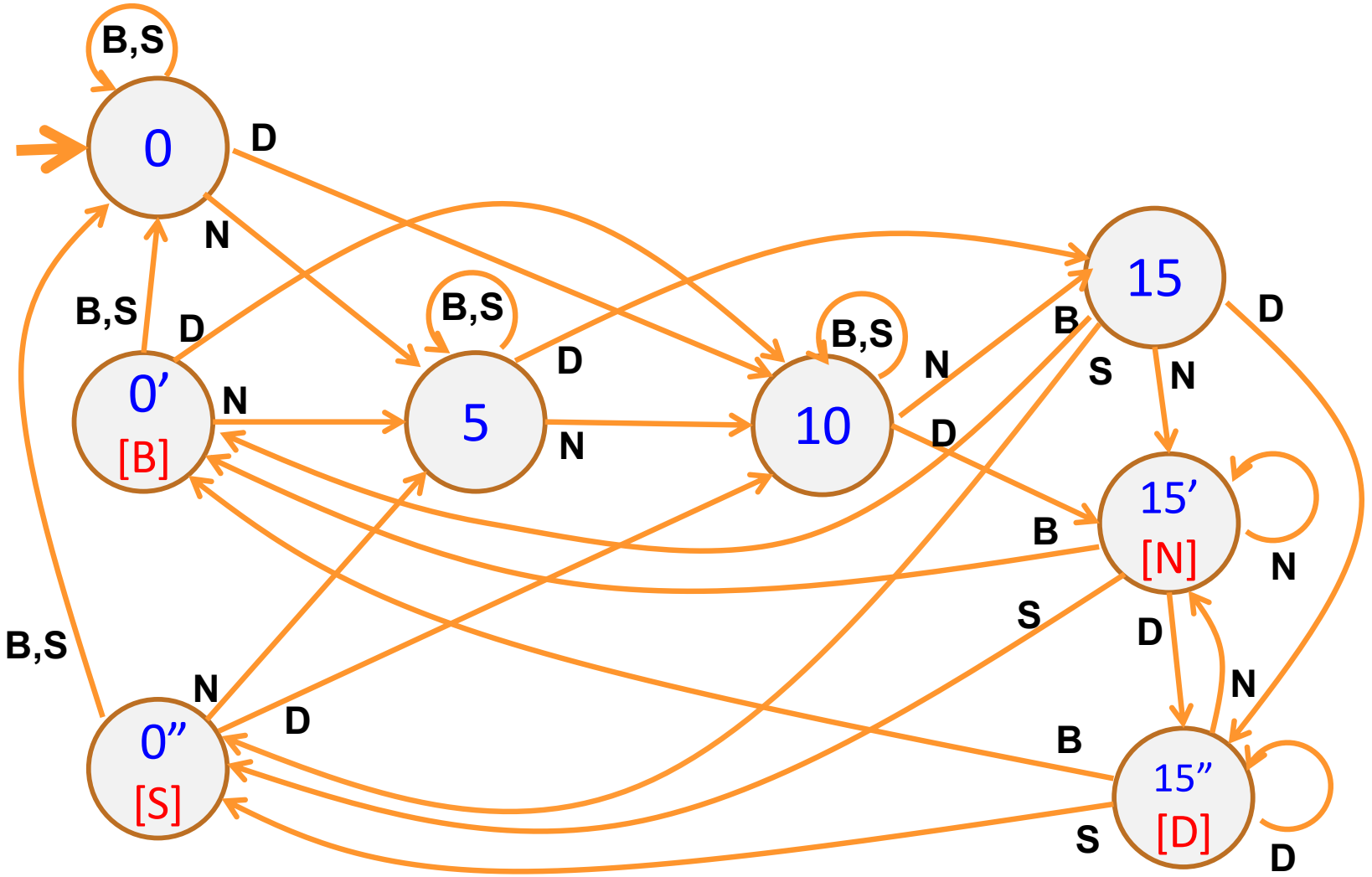
Basic transitions on **N** (nickel), **D** (dime), **B** (butterfinger), **S** (snickers)

Vending Machine, v0.2



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

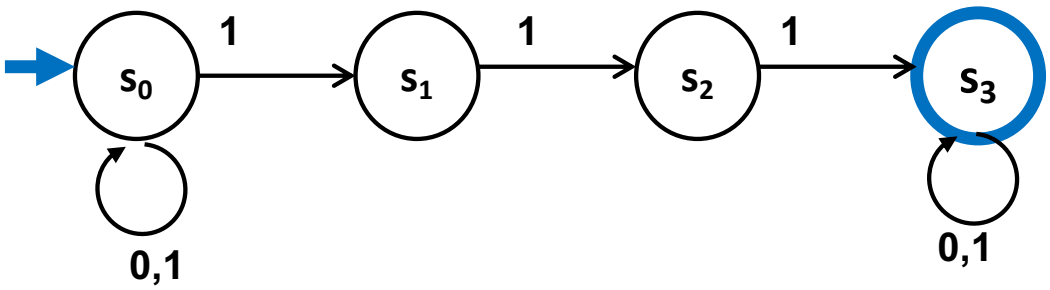
Vending Machine, v1.0



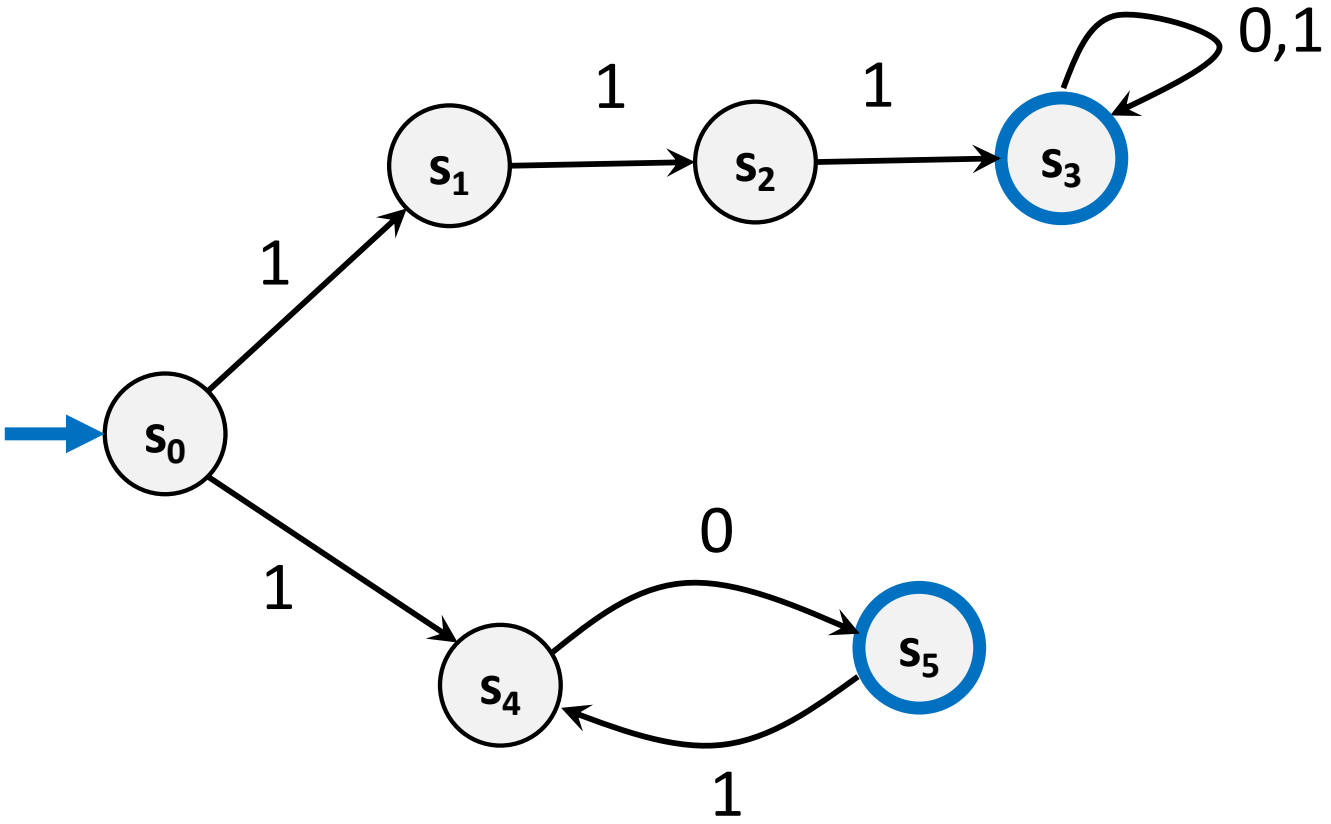
Adding additional “unexpected” transitions to cover all symbols for each state

Nondeterministic Finite Automata (NFA)

- Graph with start state, final states, edges labeled by symbols (like DFA) but
  - Not required to have exactly 1 edge out of each state labeled by each symbol— can have 0 or >1
  - Also can have edges labeled by empty string ε
- **Definition:** x is in the language recognized by an NFA if and only if <u>some</u> valid execution of the machine gets to an accept state

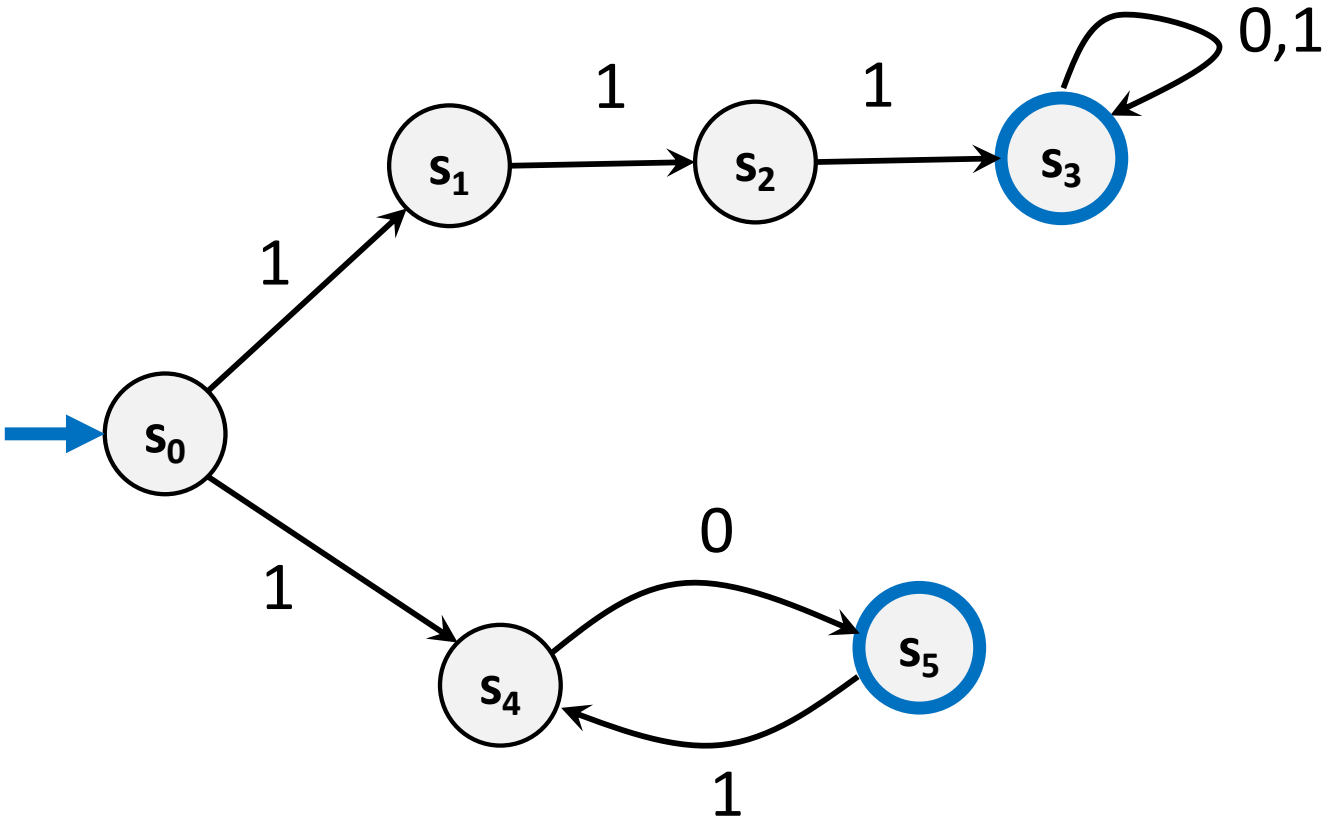


Consider This NFA



What language does this NFA accept?

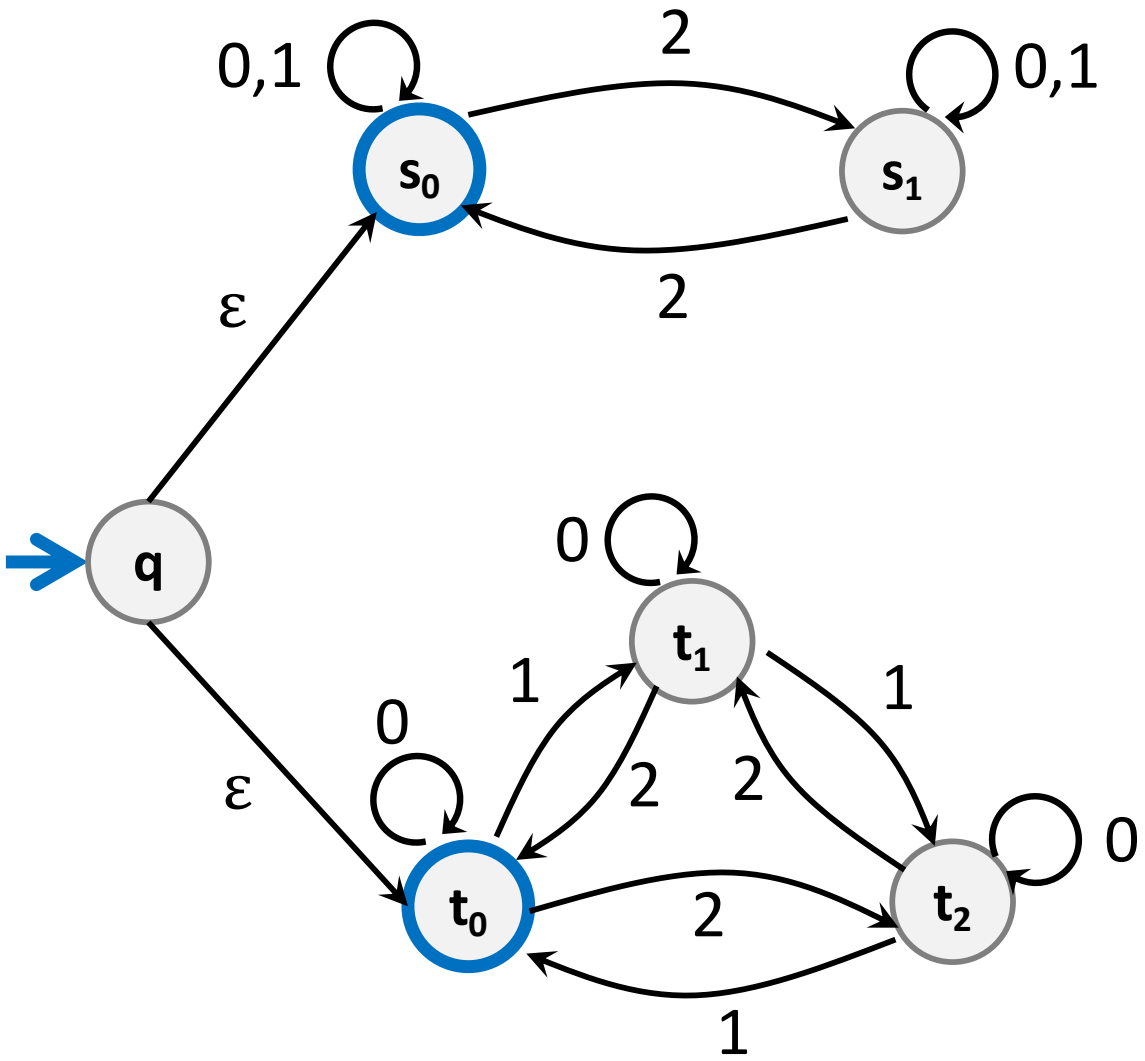
Consider This NFA



What language does this NFA accept?

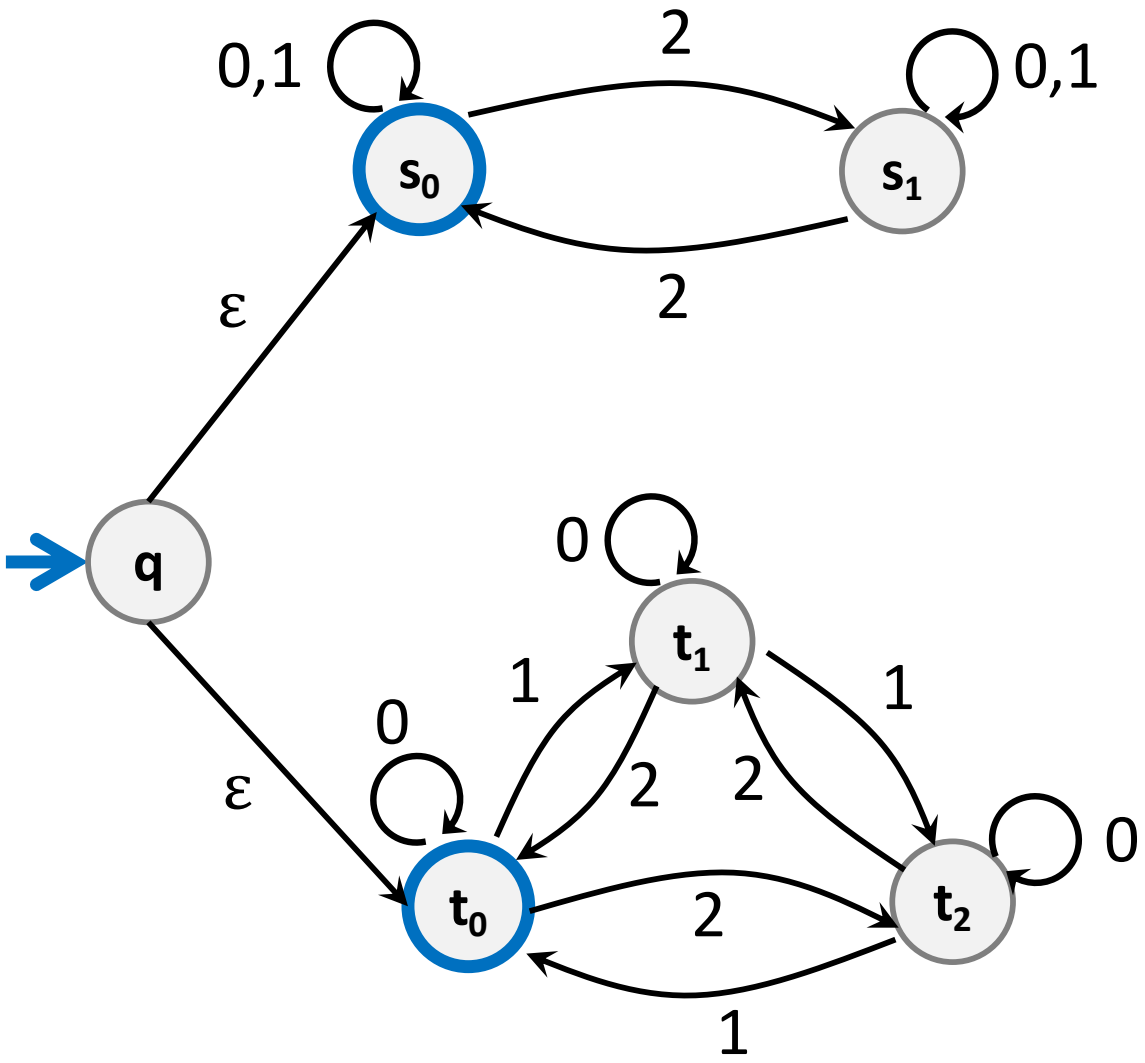
$10(10)^* \cup 111(0 \cup 1)^*$

NFA ε-moves



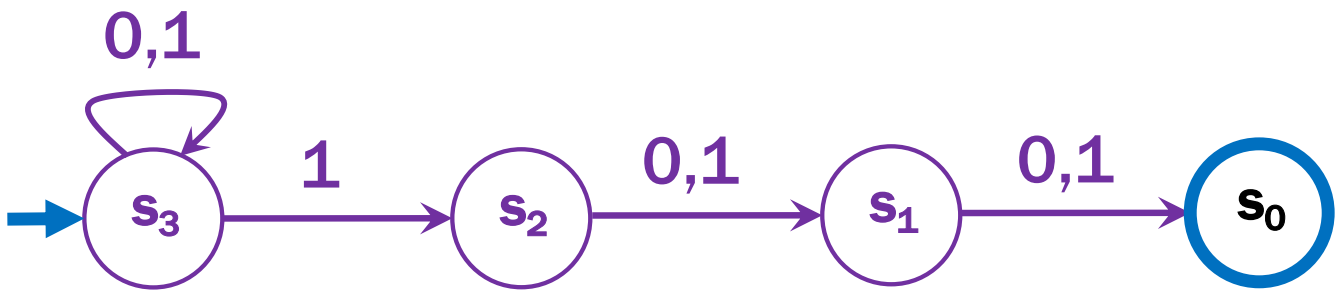
NFA ε-moves

Strings over {0,1,2} w/even # of 2's OR sum to 0 mod 3



NFA for set of binary strings with a 1 in the 3rd position from the end

NFA for set of binary strings with a 1 in the 3rd position from the end



Compare with the smallest DFA