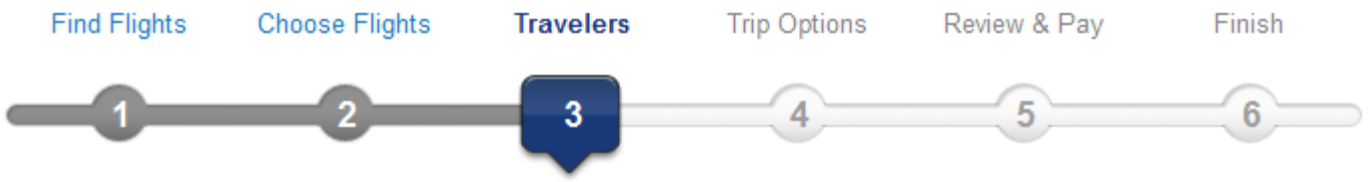
Finite State Machines

Object states, state transition diagrams, FSMs diagrams as algorithms

Applying states to objects

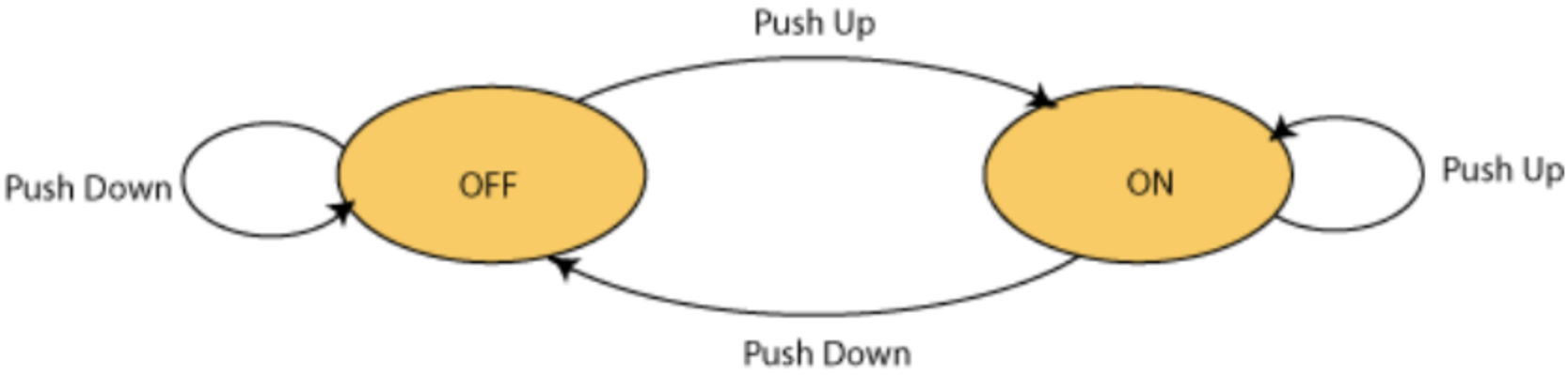
For some objects, it makes sense to consider the state of the object. Examples:

- Ordinary light switch: on, off
- Traffic light: red, yellow, green
- Metrocard purchase transaction: type selection (refill, new), payment method choice, payment, card fill, card return, change return
- Airplane ticketing: choose flight, specify passengers, select payment, on-hold, ticketed, cancelled



Simple example: state diagram

Here is a diagram for an ordinary electric light with a single switch.



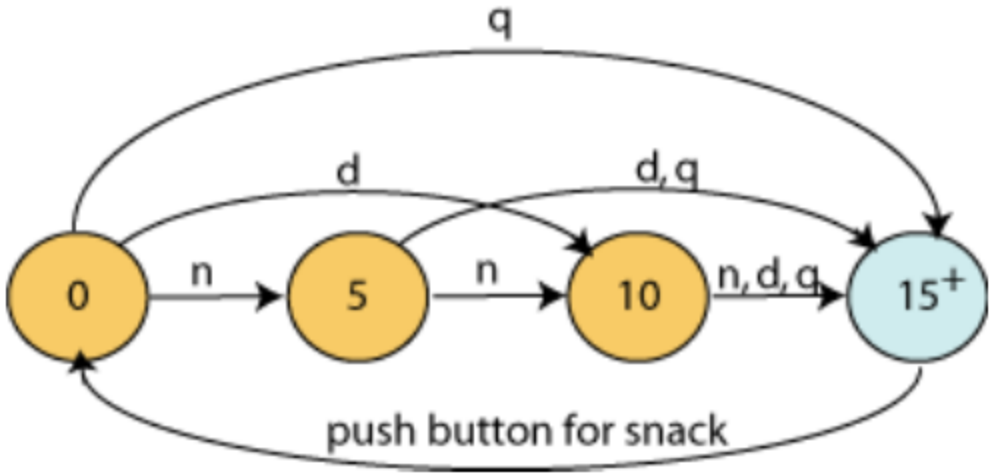
- Light **states** (OFF, ON) are ovals.
- **Transitions** between states are labeled, directed arcs. Each arc in this diagram corresponds to a possible user action.

Finite State Machines

- A **Finite State Machine** (or FSM) is a model of computation.
- The model consists of a finite number of states and the transitions between them.
- States:
  - Represent internal information about what has already happened (memory).
  - FSM is in exactly one state at any given time.
- Transitions:
  - Response of the system to input.
  - Depend entirely on current state and current input.
  - Can (and most often do) change the FSM from one state to another.
- FSMs are actually mathematical models, and they make up part of the study of automata theory.

Example: vending machine

A vending machine for a 15-cent snack. The machine accepts quarters, dimes, and nickels only. It does not give back change.



- The starting state is 0.
- There is no end state since the machine is always ready to accept coins.

Transition table

The vending machine FSM can be described in a **transition table**.

- All columns except the first correspond to input to the machine.
- Each row in the table corresponds to a state and its transitions. The first cell on the row names the state. The remaining cells are the next state given the column input.

States	N	D	Q	Push Button
0	5	10	15+	0
5	10	15+	15+	5
10	15+	15+	15+	10
15+	15+	15+	15+	0

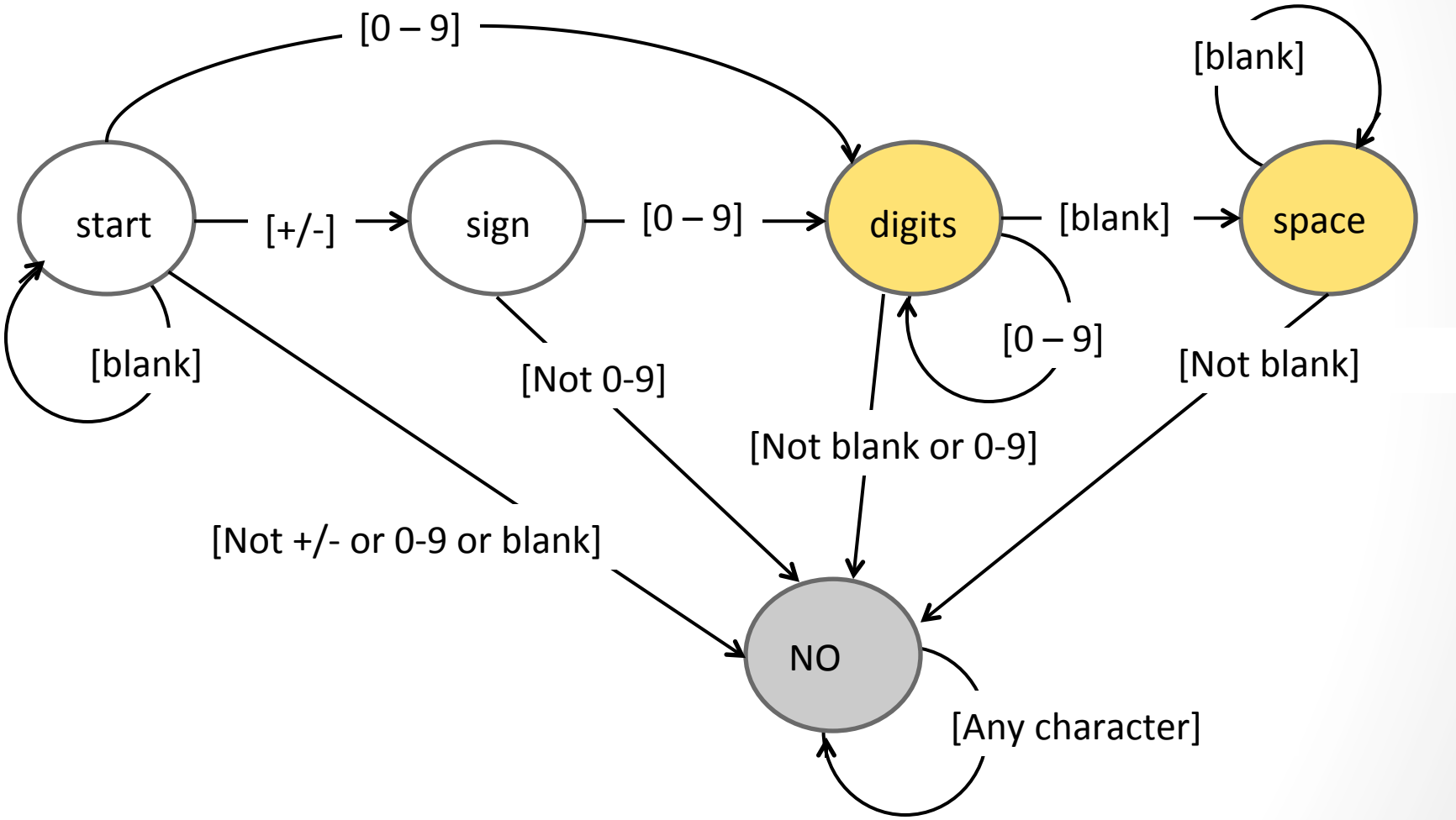
FSMs and programming language issues

FSMs can be used for describing language syntax. For example, an integer is a sequence of characters such that:

- First non-whitespace character is +, -, or a digit.
- There must be at least one digit.
- Successive characters are digits.

You can create a FSM that takes a string and determines if it represents an integer (ignoring any leading or trailing whitespace).

FSM diagram for recognizing an integer



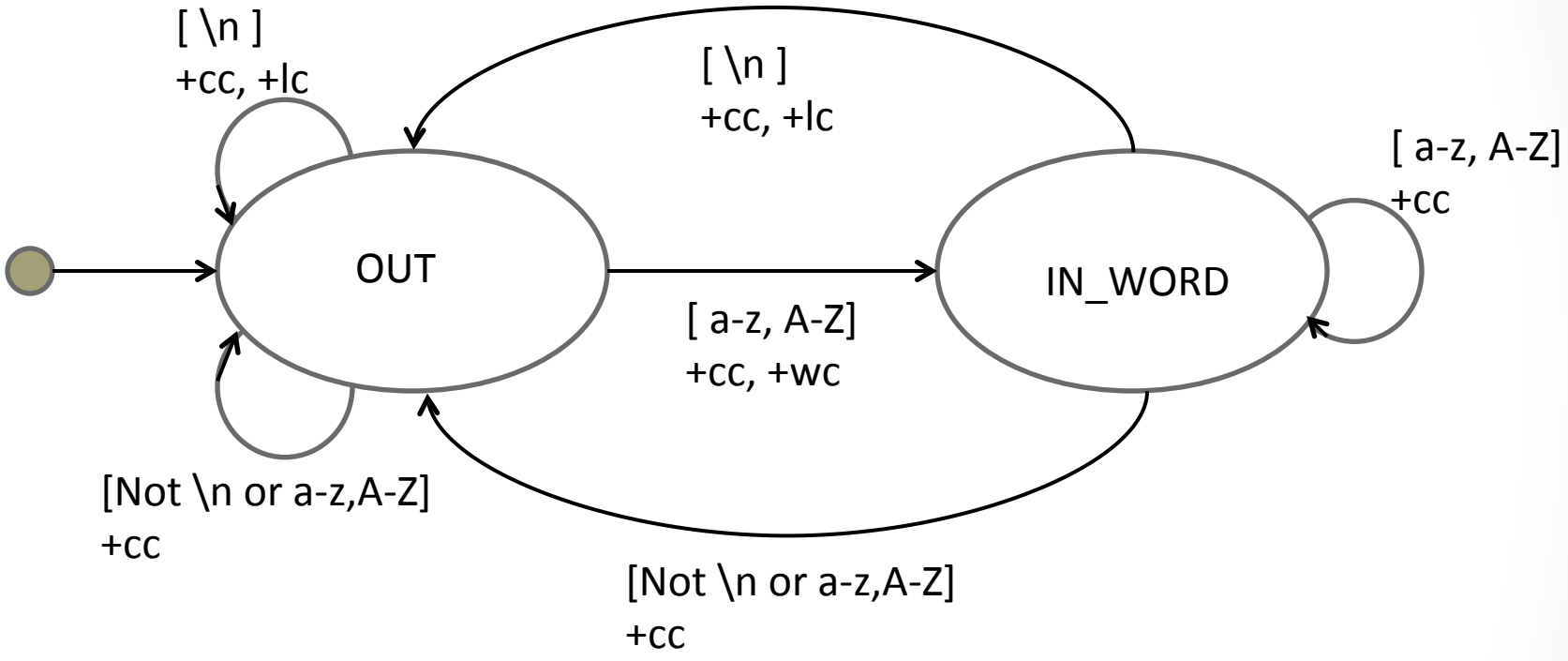
A FSM for text processing

Problem: Counting characters, words, lines.

> " A word is a maximal non-empty sequence of alphabetic characters."

- We can build a FSM as an algorithm to count
  - cc : character count
  - wc: word count
  - lc: line count
- The FSM will have two states: IN_WORD and OUT

FSM diagram, transition table



State	A-Z, a-z	\n	Not \n or A-Z,a-z
OUT	IN_WORD: +cc, +wc	OUT: +cc, +lc	OUT: +cc
IN_WORD	IN_WORD: +cc	OUT:+cc, +lc	OUT: +cc

Translating FSMs into code

Method 1: use a while loop to go through each input, with an embedded switch statement to handle the states and transitions.

```
Scanner s = new Scanner("myfile.txt");

int IN_WORD = 1;
int OUT = 0;
int state = 0;

int wc = 0, lc = 0, cc = 0;

while (sc.hasNext()) {
    state = OUT;
    String line = sc.nextLine() + "\n";
    // FSM switch code goes here
```

FSM switch code

```
// FSM switch code
for (int k  = 0; k < line.length(); k++)
   char ch = line.charAt(k);
   switch (state) {
      case OUT:
        if ( ch == '\n' ) { ++lc; ++cc; }
        else if ( Character.isLetter(ch) ) {
                                  state = IN;  ++wc; ++cc; }
        else {++cc;}
        break;
      case IN_WORD:
        if (ch == '\n' ) { state = OUT; ++lc; ++cc; }
        else if ( Character.isLetter(ch) ) { ++cc; }
        else { state = OUT; ++cc; }
        break;
      default:
        }
   }
```