NC STATE UNIVERSITY

Finite State Machines

Reading: Finite-State Machines Packet

NC STATE UNIVERSITY

Caffeine Break

- When purchasing soda from a vending machine we insert nickels (n), dimes (d), and quarters (q) (assume no paper money)
- We select a soda (s) when we have enough money
- We can model a vending machine's behavior using a finite state machine (FSM)
  - Abstract model of a system (physical, biological, mechanical, electronic, or software)
  - Consists of **inputs**, **states**, and **transitions**
  - States can be initial or final
  - Inputs cause transitions depending on the current state
- **For a vending machine, what are the possible inputs, states, and transitions?**

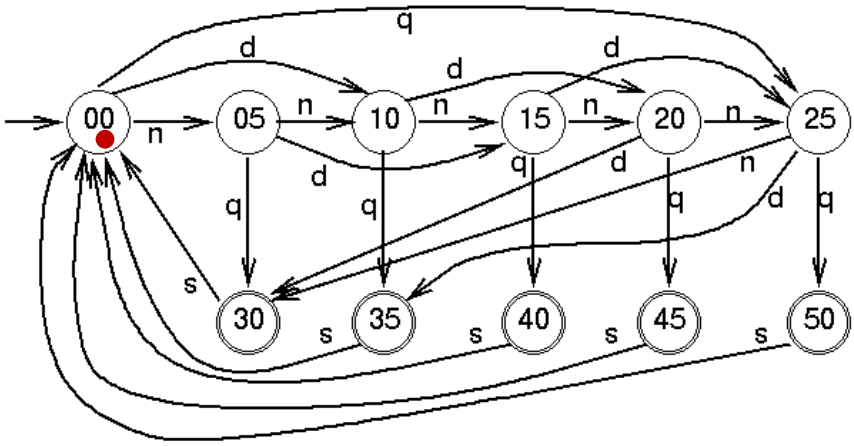


NC STATE UNIVERSITY

Vending Machine FSM Break Down

- **Inputs**: n, d, q, s
- **State**: circles/bubbles labeled with the amount of money in the machine for a single purchase
  - Initial State: 00 (i.e. no money is in the machine)
  - Final States: states with a total input greater than or equal to 30 cents – represented by double circles
- **Transitions**: arrows labeled with the inputs that caused the change of state

NC STATE UNIVERSITY



Assume a soda costs 30¢
Potential Valid Input: n, d, q, s
Potential Invalid Input: n, s

NC STATE UNIVERSITY

Finite State Machines

- Key components
  - Finite (countable) number of states
    - Represent internal "memory"
    - Store information about what has happened before
  - Transitions
    - Response of the system to its environment (input)
    - Depends on current state and current input
    - Often, but not always, results in a change of state

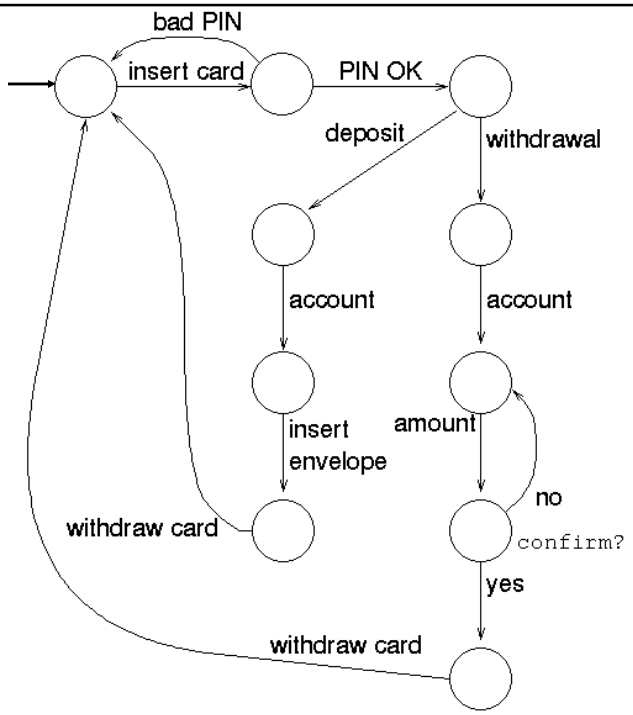
6

NC STATE UNIVERSITY

ATM FSM

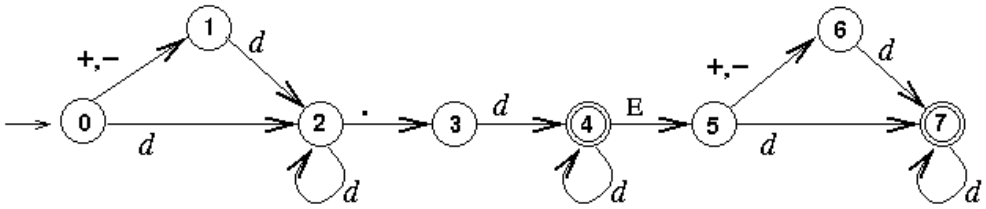
- Let's create an FSM for an ATM machine
- What are user inputs?
- What states can the machine be in?
  - States do not always have to have a label, which means a higher level of abstraction
    - a state could represent a UI screen prompting the user for the amount of money to withdrawal
- What are the transitions generated by inputs?
  - Transitions can also represent high level abstractions of user inputs

7



FSM and Programming Languages

- FSMs may be used as a precise description of programming language syntax
  - Used in compilers
- Example: Real Constants in Pascal
  - A constant floating-point number in Pascal is written as a string of digits that contain a decimal point and optionally a sign
  - There must be at least one digit before and after the decimal point
  - Floating-point data may also be represented using Pascal scientific notation
    - Sign, following by a floating-point number, followed by E, another sign, and an integer (+ signs may be omitted)

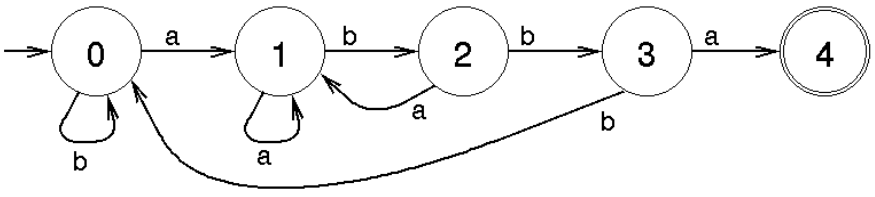


d is shorthand for digit, or 0–9

- What states does the FSM move through with the following inputs? Are these valid inputs (i.e. do they lead to a final state)?
  - 0.9
  - 9.
  - .9
  - 9.0
  - -9.3E15
  - 1.0.0

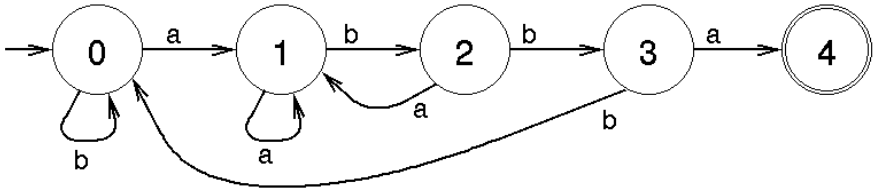
Text Processing FSMs

- Let's start with a substring example
- Suppose our alphabet consists only of 'a' and 'b'
- We can write an FSM that will search for the string "abba"
- An input string like "aababbb**abba**…" contains "abba" if we reach state 4
- Everything after finding the substring doesn't matter



Text Processing FSMs (2)

Input		a	a	b	a	b	b	b	a	b	b	a
State	0	1	1	2	1	2	3	0	1	2	3	4



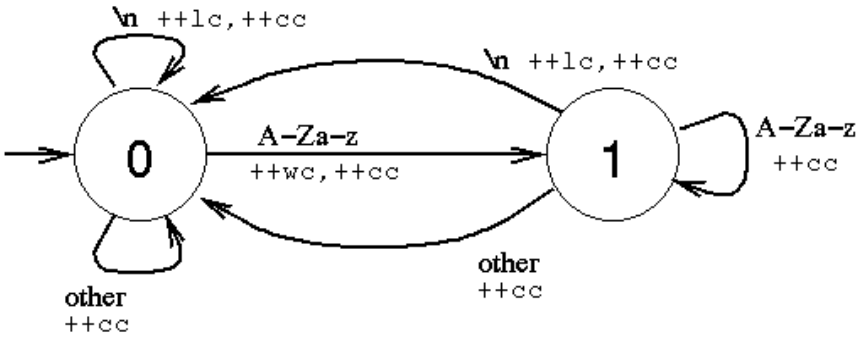
Does our FSM recognize the string “abba” in “aababbbabba”?

Text Processing and Code!

- Consider the following
  - Word: maximal sequence of zero or more upper- and zero or more lower-case **letters in a given string of text**
  - For the string `"Go State!"`
    - Is `"State!"` a word?
    - Is `"tat"` a word?
    - How about `"irregardless"`?
  - `wc` is the word count, initially 0
  - `lc` is the line count, initially 0
  - `cc` is the character count, initially 0
- Create an FSM that recognizes “words” and counts the number of words, lines, and characters (this is an implementation of the `wc` command on Linux)
  - Inputs, States, Transitions
  - Other data we need to track?

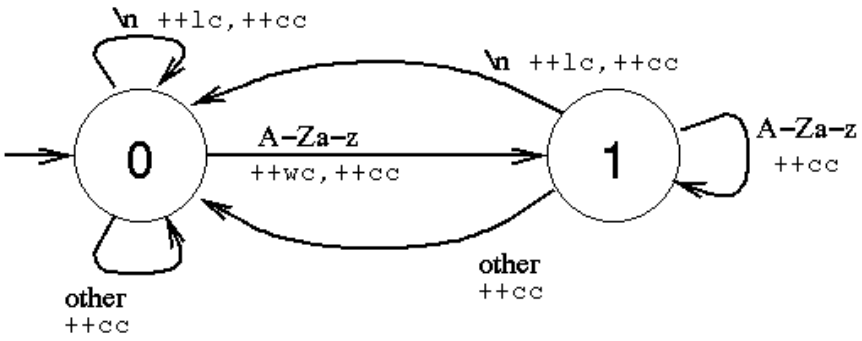
Word FSM

- Contains both inputs (`\n, A-Za-z`, and other) and actions (`++lc, ++cc, ++wc`)
- What do states 0 and 1 represent?
- Why no final state?



Transition Table for Word FSM

State	A-Za-z	\n	Other
0	1: ++wc, ++cc	0: ++lc, ++cc	0: ++cc
1	1: ++cc	0:++lc, ++cc	0: ++cc



NC STATE UNIVERSITY

Translating an FSM into Code

- Standard idiom: while-switch idiom for text processing FSMs
  - While loop gets each character in the text
  - Switch on the current state
  - Then handle the input based on the specific state
  - Don't forget to increment loop variable!

```
int currentState = 0; // start state
char c = '';
int i = 0;
while (i < string.length()) {
    c = string.charAt(i);
    switch (currentState) {
        case 0:
            if (c is a given input) { /* do task */ }
            else if (c is another input) { /* do task */}
            break;
        case 1:
            if (c is a given input) { /* do task */ }
            else if (c is another input) { /* do task */}
            break;
    }
    i++;
}
```


NC STATE UNIVERSITY

Translating an FSM into Code

- Could also be implemented using a for loop and if statements
  - For loop gets each character in the text
  - First layer of conditionals related to current state
  - Then handle the input based on the specific state
  - Loop variable increment handled in the loop declaration

```
int currentState = 0; // start state
char c = '';
for (int i = 0; i < string.length(); i++) {
    c = string.charAt(i);
    if (currentState == 0) {
        if (c is a given input) { /* do task */ }
        else if (c is another input) { /* do task */}
    } else if (currentState == 1) {
        if (c is a given input) { /* do task */ }
        else if (c is another input) { /* do task */}
    }
}
```

NC STATE UNIVERSITY

Switch Statements

- Switch statement is a control structure similar to if/else
- Cases are executed based on an integer value
  - Later versions of Java allow for switching on strings
  - But for text processing FSMs, we're working on a character-level so int works fine.
- If the calculated value does not match any listed values then no case code is executed, unless…
- The optional default case is implemented
- Remember to include break after the statements for each case!!! (unless you explicitly intend to fall through)

```
switch (<integer expression>) {
   case <value>:
      <statements>;
      break;
   case <value>:
      <statements>;
      break;
   …
   default:   //optional
      <statements>;
      <break>;
}
```


NC STATE UNIVERSITY

Switch Examples

```
Scanner in = new Scanner(System.in);
System.out.println("Place?");
int place = in.nextInt();

switch (place) {
  case 1:
    System.out.println("1st!");
    break;
  case 2:
    System.out.println("2nd!");
    break;
  case 3:
    System.out.println("3rd");
    break;
}
```

```
Scanner in = new Scanner(System.in);
System.out.println("Place?");
int place = in.nextInt();

switch (place) {
  case 1:
  case 2:
  case 3:
    System.out.println("Medal!");
    break;
 default:
  System.out.println("No medal");
  break;
}
```

WordCounter

- Take a look at the code implementation in the FSM reading (Figure 7 on Page 9)

Practice FSM

- Create an FSM that would recognize the acronym “lol” (case insensitive) in a String of input (like a text message)
  - By word, I mean that “lol” would be surrounded by whitespace, end with punctuation (non-letter and non-number), or be the only text in a String
    - Lol - YES
    - Lollipop - NO
    - LOL. - YES
    - Alol - NO
- Design your FSM: http://madebyevan.com/fsm/
- Implement your FSM in a method containsLol() that accepts a String of text and returns true if the String contains “lol”

References

- “Finite-State Machines,” by Suzanne Balik and Matthias Stallmann