CSC207 Week 10

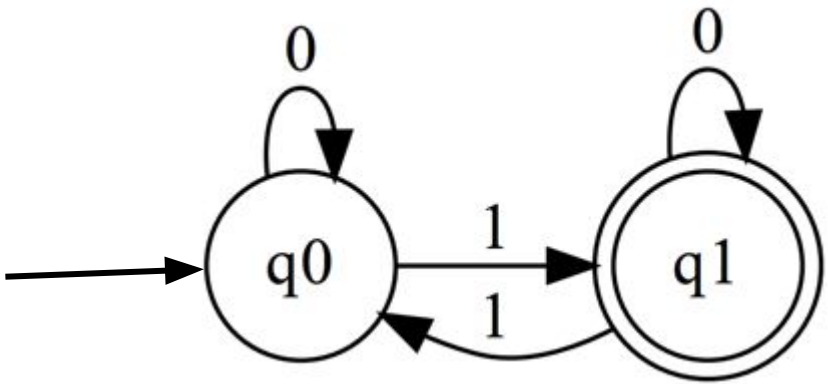
Larry Zhang

Finite State Machine for String Matching

What is an Finite State Machine (FSM)?

An FSM, a.k.a. deterministic finite automaton or DFA, is a **computational model** which can be used to solve the string-matching problems.

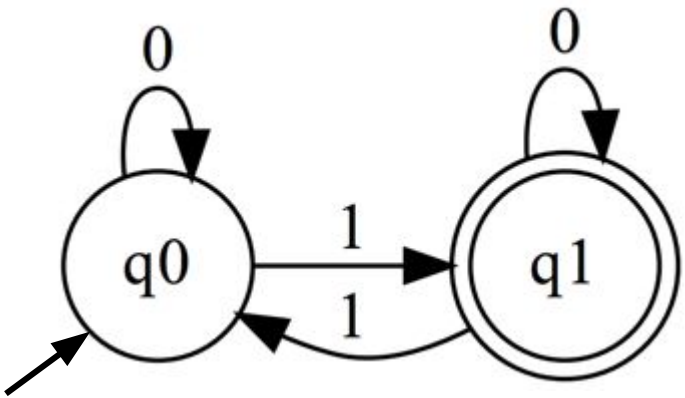
- It can also be used for other things like digital circuit design (will learn in CSC258)



String Matching Problem

- Describe a **set of strings** (a.k.a, a “language” (learn in CSC236)), namely ***L***
  - *e.g., all binary strings with an odd number 0’s*
  - *e.g., all strings that do not contain “CSC” as a substring*
  - *e.g., all strings that match “*`^a(bc)+.*\d{3}e$`*”*
- Given any string **s**, how to efficiently determine whether **s** belongs to **L**?
- If the the language is “regular” (learn in CSC236), then it is possible efficiently solve this problem using a computer with a finite amount of memory; otherwise, a computer cannot solve it.

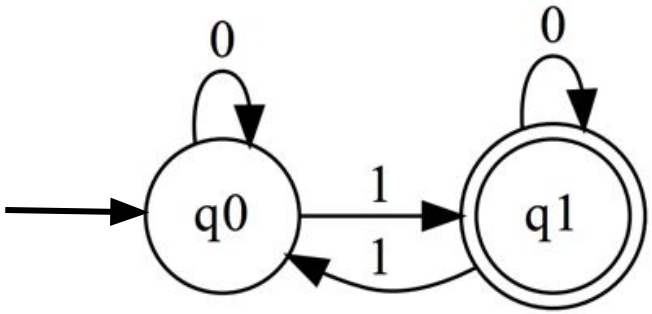
FSM: Example



Say we want to test string w = 0110

- **q0** is the **initial state**
- **q1** is the **accepting (final) state**
- read each character of w from left to right, each character causes a **state transition** in the flowchart, following one of the arrows.
- After scanning the whole string, if the current state is at the **accepting state (q1)** then the string is “**accepted**” by the FSM.
- Otherwise, the string is “**rejected**”.

Let's test a few strings



- 0
  - q0 -> q0, **rejected**
- 1
  - q0 -> q1, **accepted**
- 100
  - q0 -> q1 -> q1 -> q1, **accepted**
- 110
  - q0 -> q1 -> q0 -> q0, **rejected**
- 10101
  - q0 -> q1 -> q1 -> q0 -> q0 -> q1 -> q1, **accepted**
- 10101001
  - q0 -> q1 -> q1 -> q0 -> q0 -> q1 -> q1 -> q1 -> q0, **rejected**

Guess what kind of strings are accepted by this FSM?
ANS: Strings with an odd number of 1's!

This matching algorithm is **efficient**:
Going through the tested string just once

This FSM determines if a string is a member of the following regular language: {w ∈ {0, 1}* | w consists of an odd number of 1's}

When you use Regex in Java

This methods compiles the regex into the FSM

```
Pattern p = Pattern.compile("CSC[0-9][0-9][0-9]H5(F|S)");
Matcher m = p.matcher("CSC207H5F");
System.out.println(m.matches());
```

This method tests the string using the FSM.

Exercise: Given a Regex, Compile an FSM

Regex: `^a.*b$`

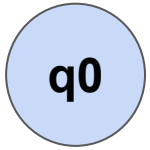
All strings that start with a and end with b

(assume the character can only be “a” or “b”)

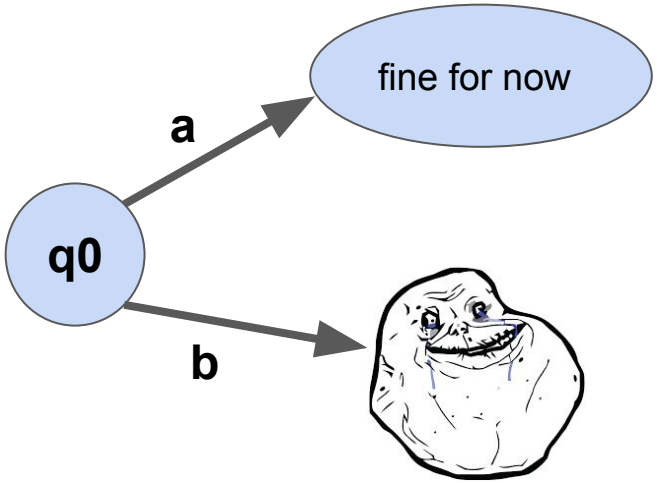
L: All strings that start with a and end with b

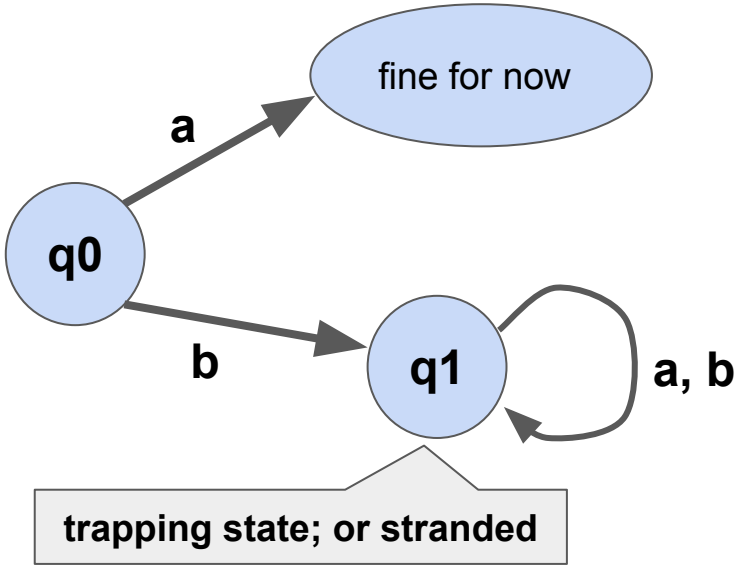
Thinking process ....

Start from an initial state q0 ...



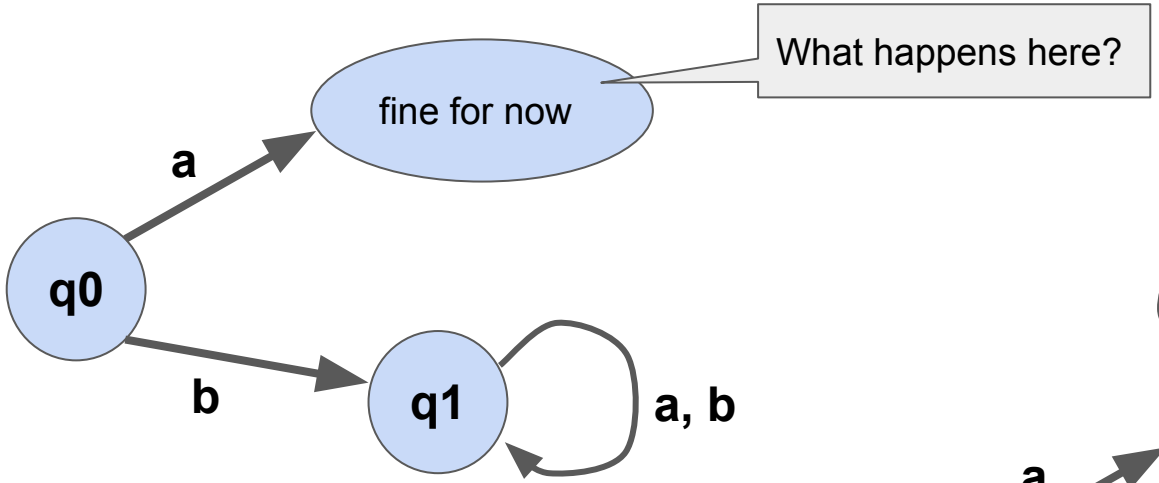
What happens if we get “**a**” or “**b**” when at q0 ?





L: All strings that start with a and end with b

Thinking process, continued

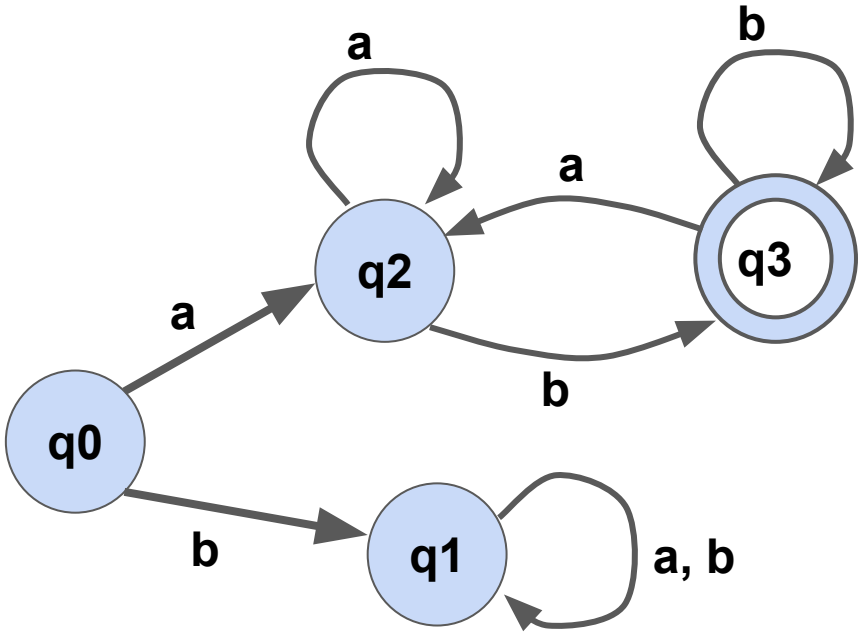


Just after getting a single “**a**”, not ready to accept.

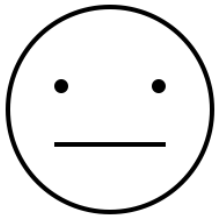
If getting a “**b**”, then ready to accept it.

If getting another “**b**”, still ready to accept.

But if getting a “**a**”, then change back to “not ready to accept” state.



Prove the Correctness of FSM

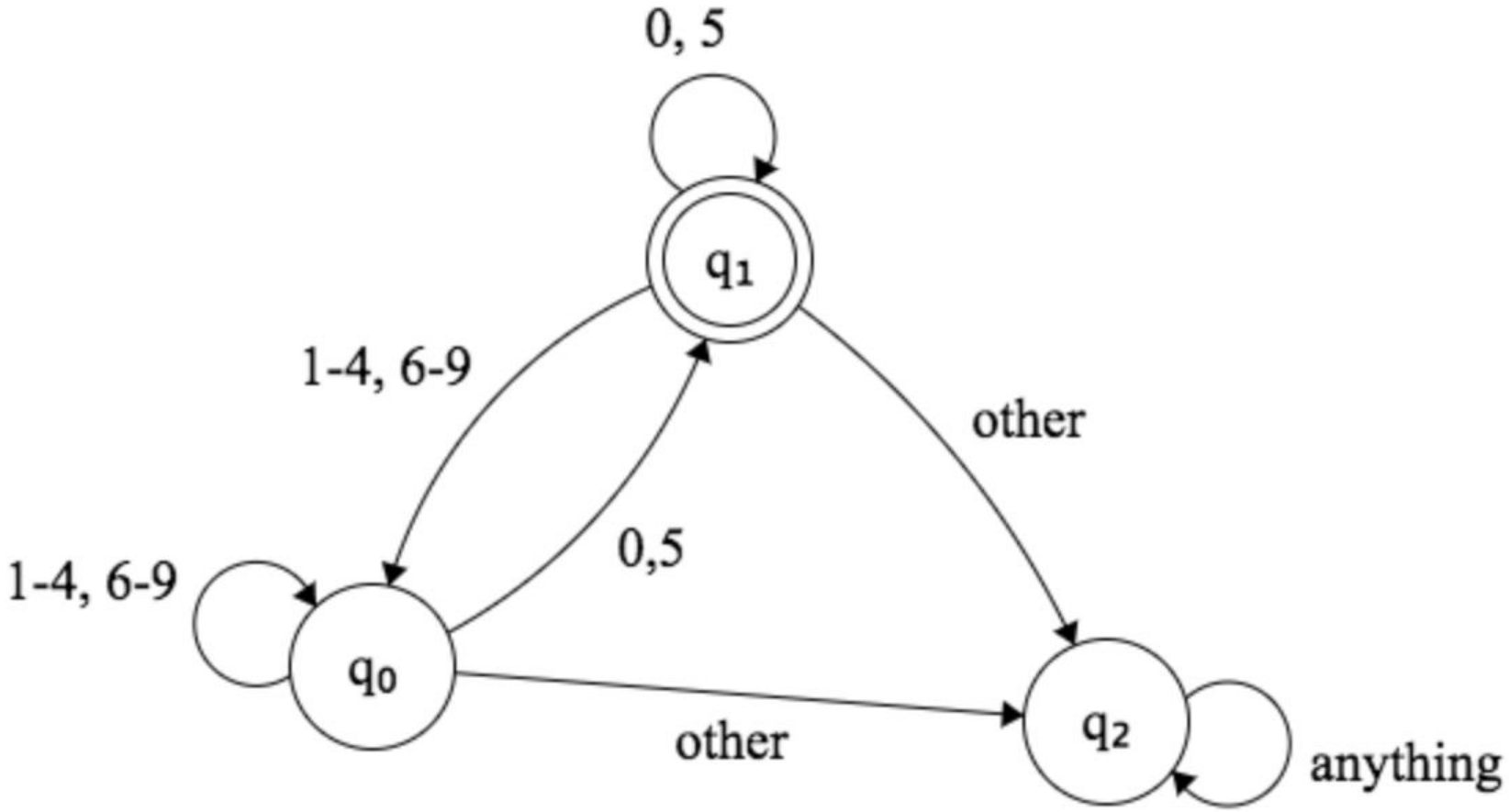


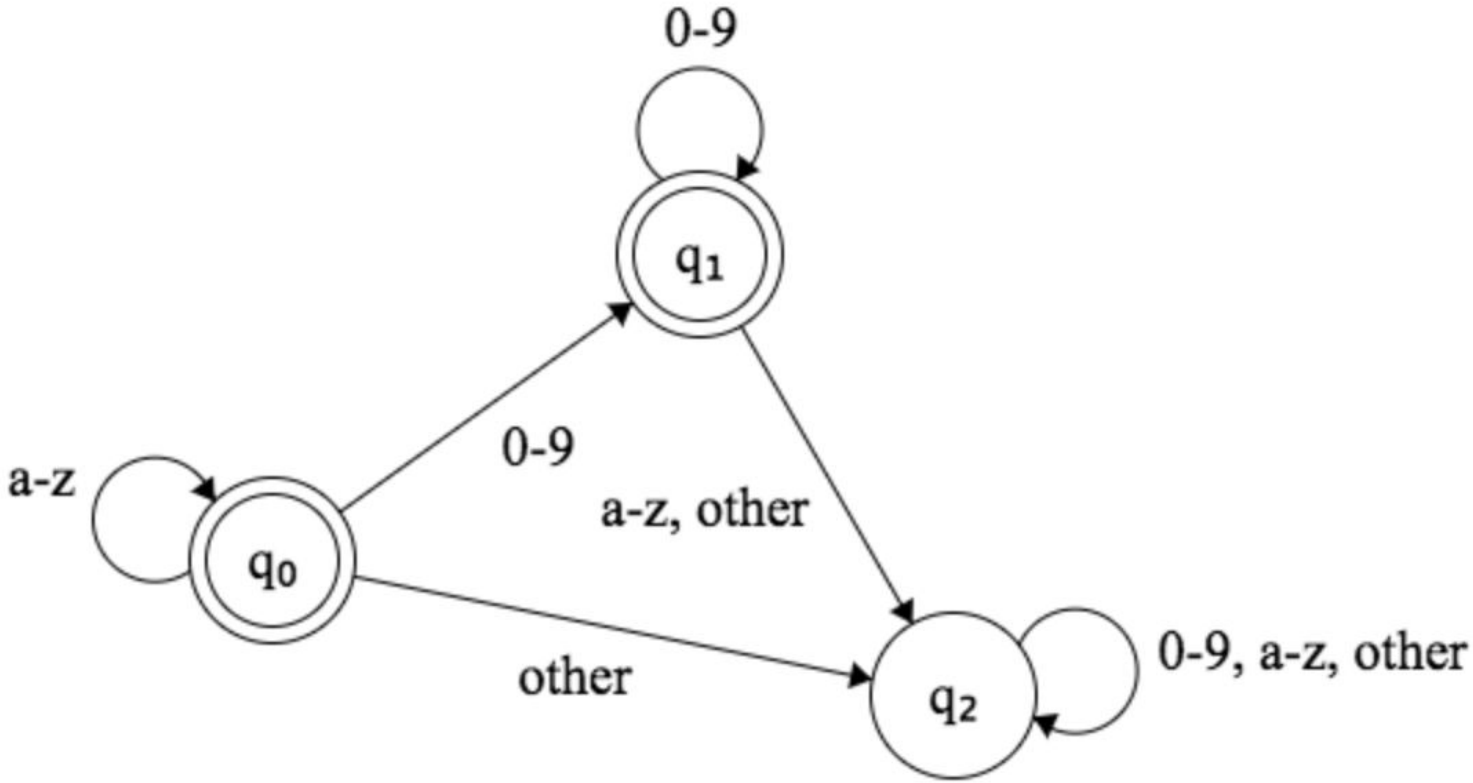
You learn it in CSC236.

DEMO #1:

`FSMExamples.java`

Exercise:
Design an FSM that accepts a string representing a number that is a multiple of 5


0, 5
q1
1-4, 6-9
other
0,5
1-4, 6-9
q0
q2
other
anything


0-9
q1
0-9
a-z, other
a-z
q0
other
q2
0-9, a-z, other

DEMO #2:

`MarksParser.java`