Theory of Computation

CS 332

Lecture 1

Theory of Computation

- Course information
- Overview of the area
- Finite Automata

Sofya Raskhodnikova

Course information

1. Course staff
2. Course website(s)
3. Piazza bonus
4. Prerequisites
5. Textbook(s)
6. Syllabus
7. Homework logistics
8. Collaboration policy
9. Exams and grading

Tips for the course

- Concepts in this course take some time to sink in: be careful not to fall behind.
- Do the assigned reading on each topic before the corresponding lecture.
- Take advantage of office hours.
- Be active in lectures/recitations and on piazza.
- Allocate lots of time for the course: comparable to a project course, but spread more evenly.

CS 332

Tips for the course: HW

- Start working on HW early.
- Spread your HW time over multiple days.
- You can work in groups (up to 4 people), but spend 1-2 hours thinking about it on your own before your group meeting.

CS 332

Tips: learning problem solving

To learn problem solving, you have to do it:

- Try to think how you would solve any presented problem before you read/hear the answer.
- Do exercises in addition to HW.

Tips: how to read a math text

- Not like reading a mystery novel.
- The goal is not to get the answers, but to learn the techniques.
- Always try to foresee what is coming next.
- Always think how you would approach a problem before reading the solution.
- This applies to things that are not explicitly labeled as problems.

Skills we will work on

- Mathematical reasoning
- Expressing your ideas
  - abstractly (suppress inessential details)
  - precisely (rigorously)
- Mathematical modeling
- Algorithmic thinking
- Problem solving
- Having **FUN** with all of the above!!!

Could they ask me questions

about CS 332 material on job interviews?

- You bet.

What is Theory of Computation?

- You’ve learned about computers and programming
- Much of this knowledge is specific to particular computing environment

What is Theory of Computation?

- Theory
  - General ideas that apply to many systems
  - Expressed simply, abstractly, precisely
- ***Abstraction*** suppresses inessential details
- ***Precision*** enables rigorous analysis
  - Correctness proofs for algorithms and system designs
  - Formal analysis of complexity
    - Proof that there is no algorithm to solve some problem in some setting (with certain cost)

CS 332

This course

- Theory basics
  - Models for ***machines***
  - Models for the ***problems*** machines can be used to solve
  - ***Theorems*** about what kinds of machines can solve what kinds of problems, and at what cost
  - Theory needed for sequential single-processor computing
- Not covered:
  - Parallel machines
  - Distributed systems
  - Quantum computation
  - Sublinear computation
  - Real-time systems
  - Mobile computing
  - Embedded systems
  - …

CS 332

Machine models

- **Finite Automata (FAs):** machines with fixed amount of unstructured memory
  - useful for modeling chips, communication protocols, adventure games, some control systems, …
- **Pushdown Automata (PDAs):** FAs with unbounded structured memory in the form of a pushdown stack
  - useful for modeling parsing, compilers, some calculations
- **Turing Machines (TMs)**: FAs with unbounded tape
  - Model for general sequential computation (real computer).
  - ***Equivalent*** to RAMs, various programming languages models
  - Suggest general notion of ***computability***

Machine models

- **Resource-bounded TMs** (time and space bounded):
  - "not that different" on different models: "within a polynomial factor"
- **Probabilistic TMs**: extension of TMs that allows random choices

Most of these models have ***nondeterministic*** variants:

can make nondeterministic "guesses"

CS 332

Problems solved by machines

1. What is a problem?

In this course, problem is a language.

A ***language*** is a set of strings over some “alphabet”

2. What does it mean for a machine to “solve” a problem?

Examples of languages

- $\mathbf{L_1}$= {binary representations of natural numbers divisible by 2}
- $\mathbf{L_2}$= {binary representations of primes}

alphabet = {0,1}

- $\mathbf{L_3}$= {sequences of decimal numbers, separated by commas, that can be divided into 2 groups with the same sum}
  – (5,3,1,3) $\in \mathbf{L_3}$, (15,7,5,9,1) $\notin \mathbf{L_3}$.

alphabet = {0,1,…,9,comma}

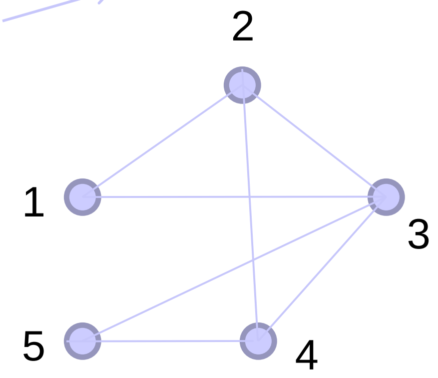
- $\mathbf{L_4}$= {C programs that loop forever on some input}
- $\mathbf{L_5}$= {representations of graphs containing a ***Hamiltonian cycle***}

visits each node exactly once

  – {(1,2,3,4,5); (1,2),(1,3),(2,3),…}

vertices **edges**

alphabet = all symbols: digits, commas, parens



CS 332

Theorems about classes of languages

We will define classes of languages and prove theorems about them:

- **inclusion**: Every language recognizable (i.e., solvable) by a FA is also recognizable by a TM.
- **non-inclusion**: Not every language recognizable by a TM is also recognizable by a FA.
- **completeness**: "Hardest" language in a class
- **robustness**: alternative characterizations of classes
  - e.g., FA-recognizable languages by regular expressions (UNIX)

Why study theory of computation?

- a ***language*** for talking about program behavior
- feasibility (what can and cannot be done)
  - halting problem, NP-completeness
- analyzing correctness and resource usage
- computationally hard problems are essential for cryptography
- computation is fundamental to understanding the world
  - cells, brains, social networks, physical systems all can be viewed as computational devices
- IT IS **FUN**!!!

CS 332

Is it useful for developers?

Boss, I can’t find an efficient algorithm. I guess I ‘m just too dumb.

Boss, I can’t find an efficient algorithm, because no such algorithm is possible.

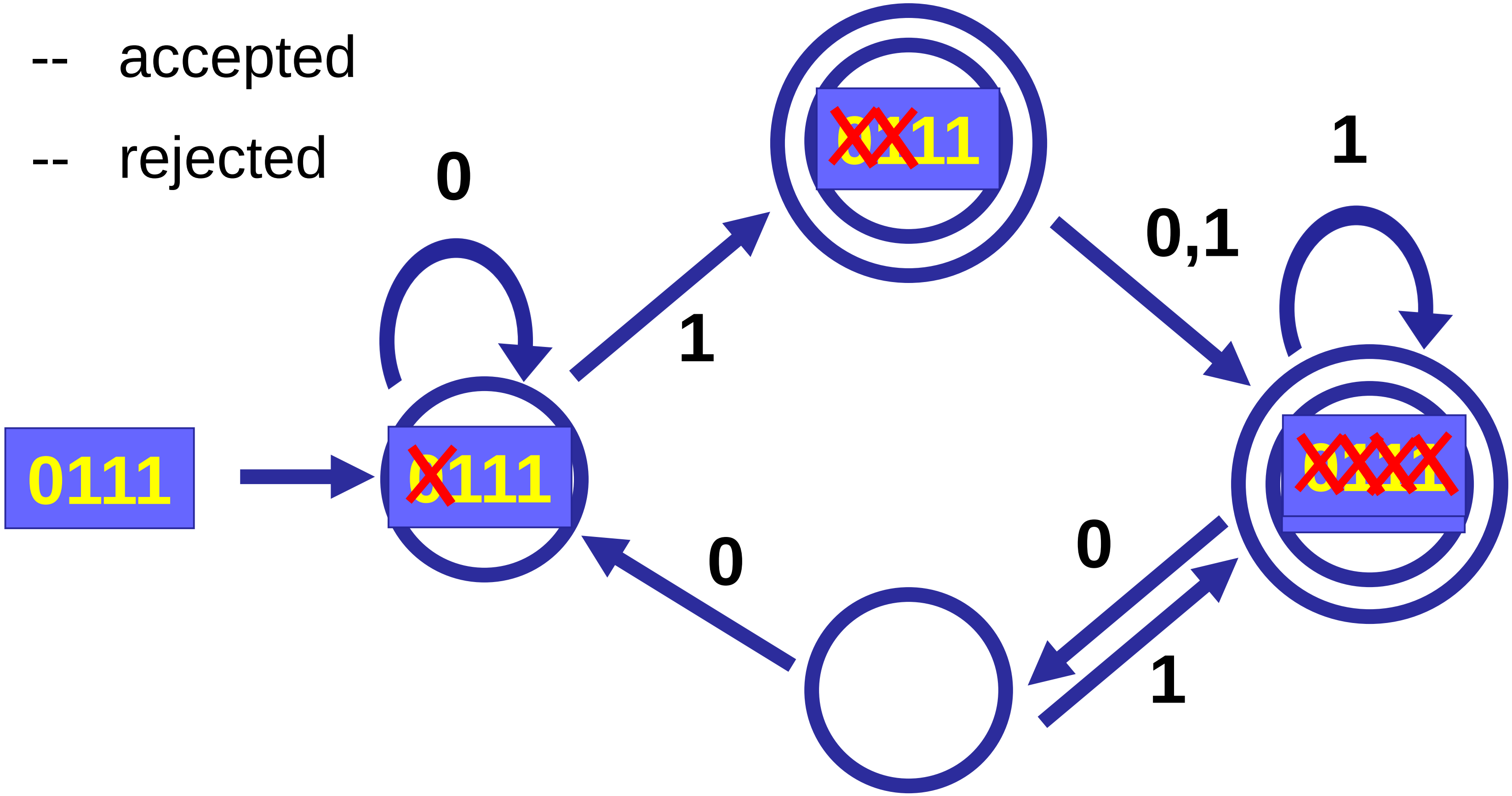
Parts of the course

I. Automata Theory
II. Computability Theory
III. Complexity Theory

Finite automata (FA)

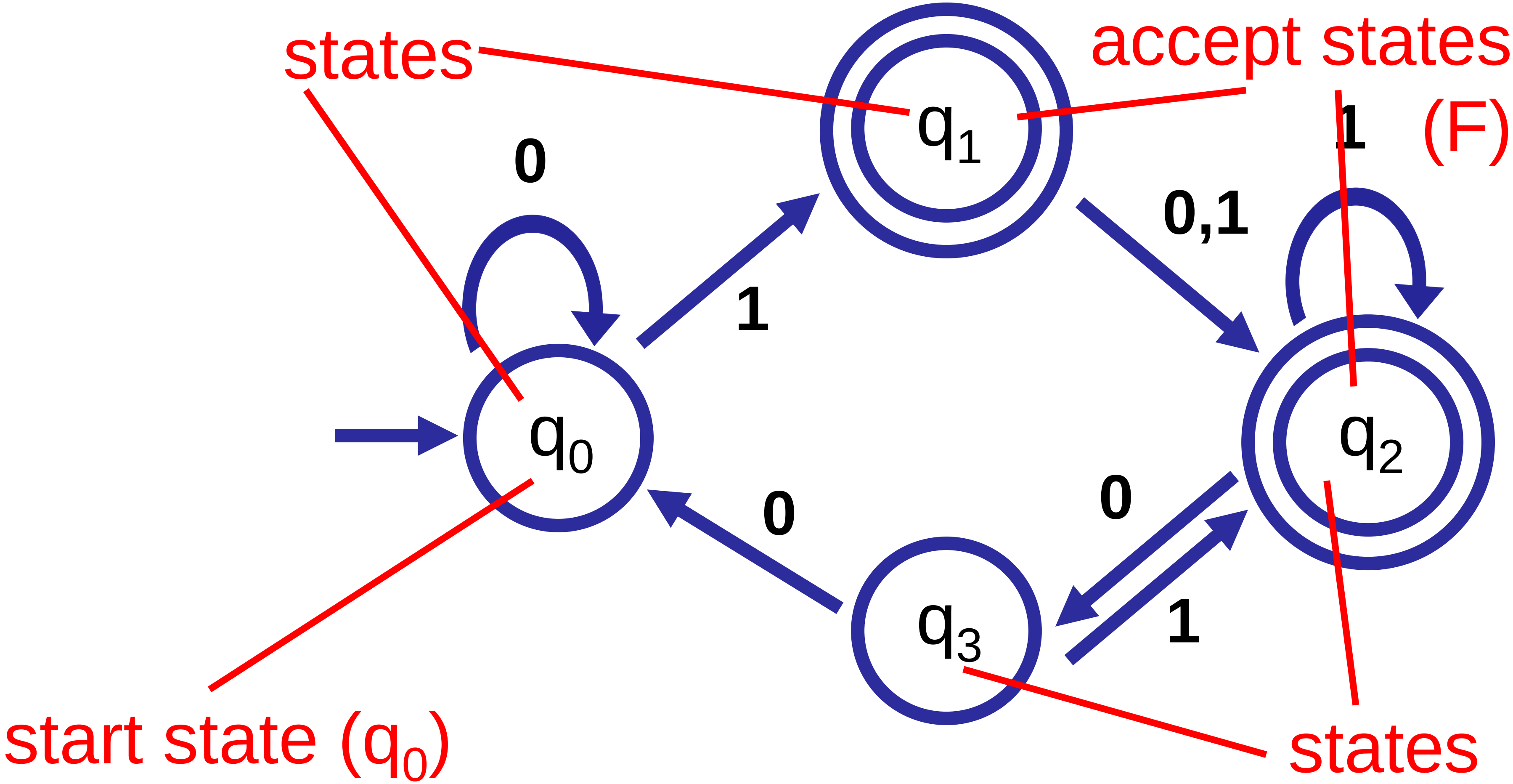
0111 -- accepted

01110 -- rejected



Each string is either accepted or rejected by the automaton depending on whether it is in an accept state at the end.

Anatomy of finite automaton



CS 332 Formal Definition

A ***finite automaton*** is a 5-tuple $\mathbf{M = (Q, \Sigma, \delta, q_0, F)}$

$\mathbf{Q}$ is the set of states

$\mathbf{\Sigma}$ is the alphabet

$\mathbf{\delta : Q \times \Sigma \to Q}$ is the transition function

$\mathbf{q_0 \in Q}$ is the start state

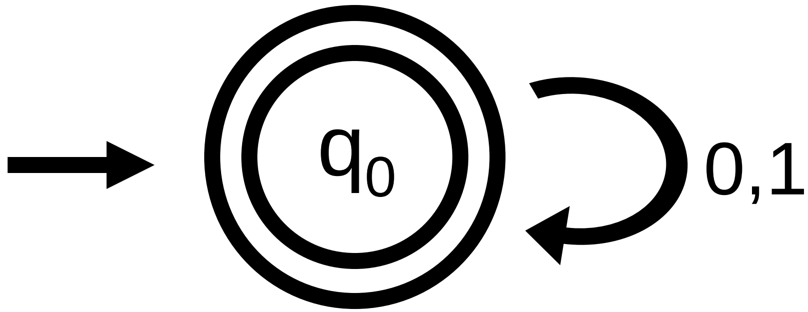
$\mathbf{F \subseteq Q}$ is the set of accept states

$\mathbf{L(M)}$ = the ***language*** of machine M
= set of all (finite) strings machine M accepts

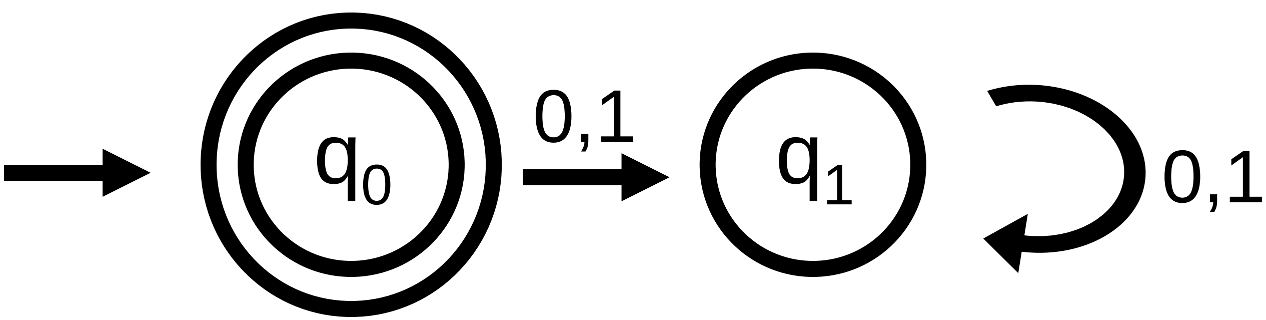
M ***recognizes*** the language L(M)

CS 332

Examples of FAs

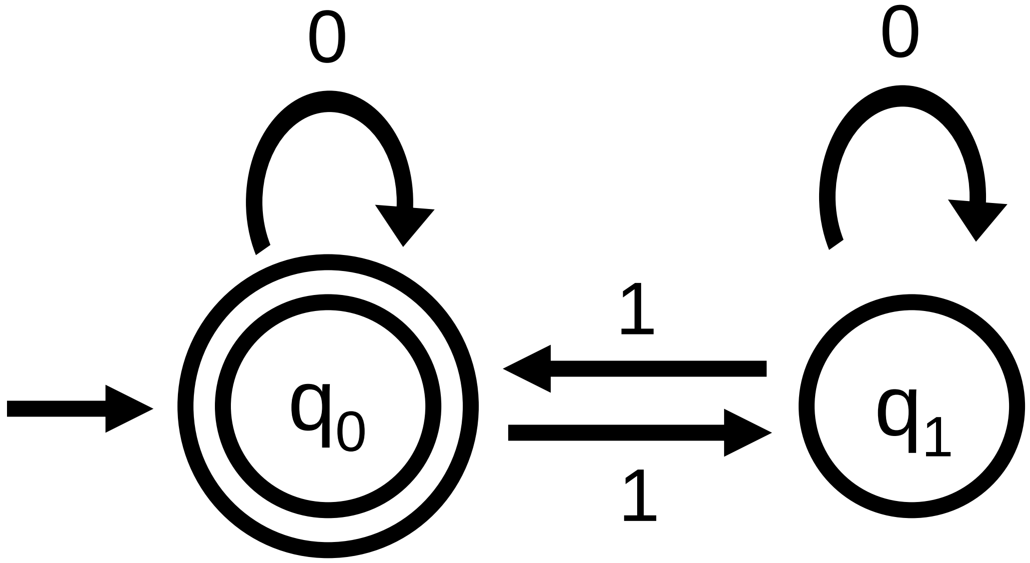


L(M) = {~~w~~ | w is a string of 0s and 1s}



L(M) = {ε} where ε denotes the empty string

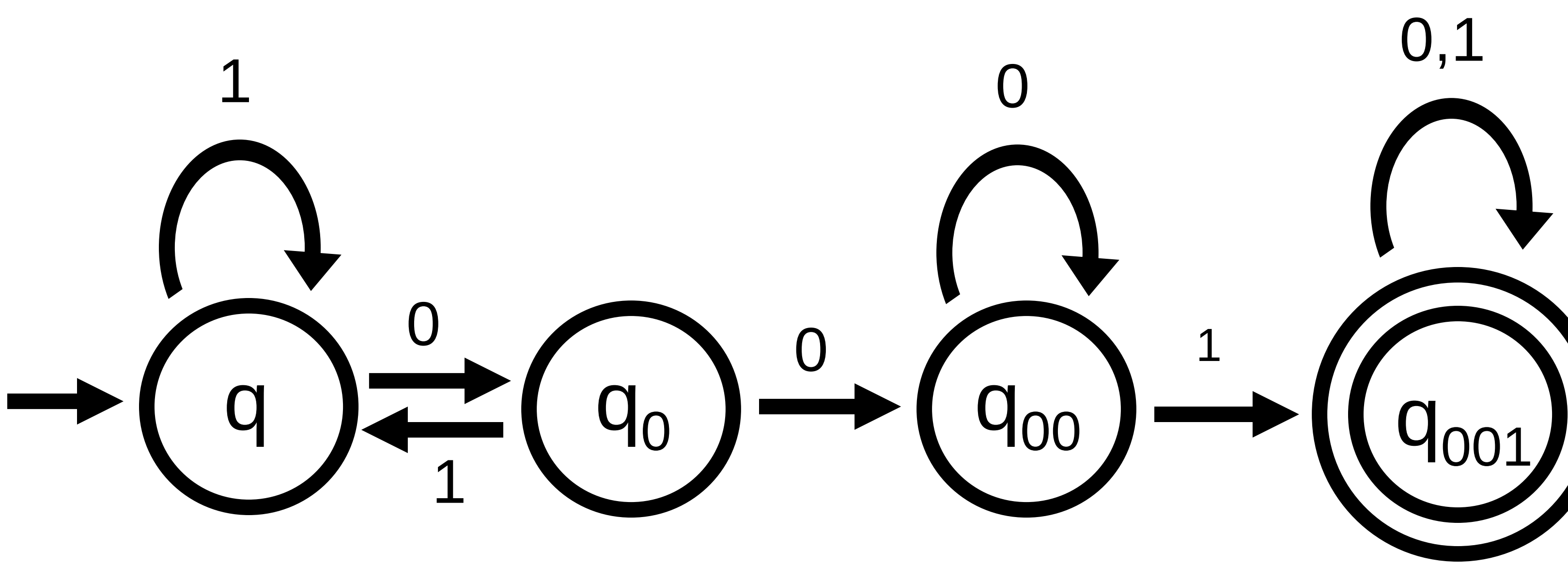
Examples of FAs



L(M) = {w | w has an even number of 1s}

Examples of FAs

Build an automaton that accepts all (and only those) strings that contain 001



CS 332

Formal definition of FA

$M = (Q, \Sigma, \delta, q_0, F)$ **where**

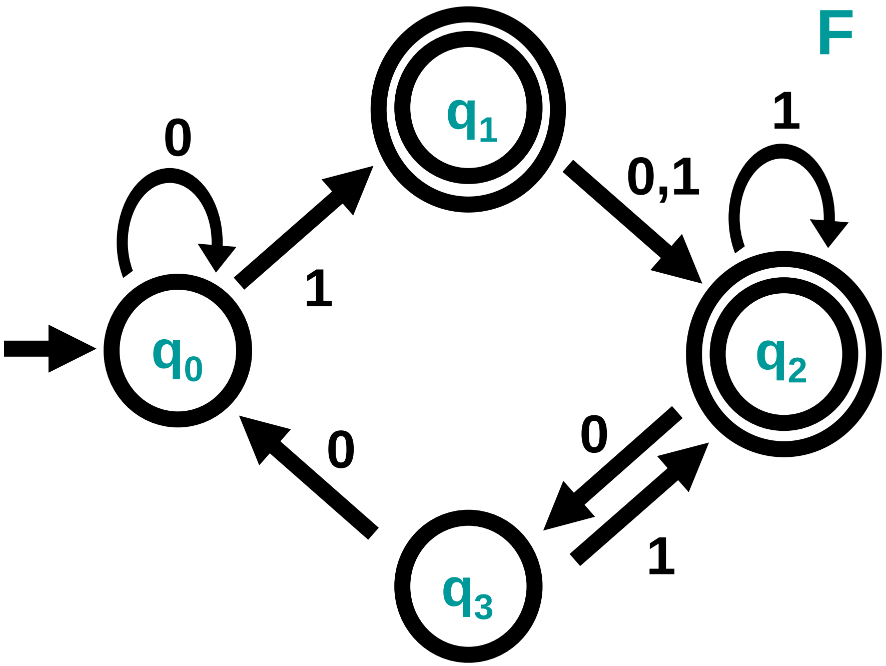
$Q = \{q_0, q_1, q_2, q_3\}$

$\Sigma = \{0,1\}$

$\delta : Q \times \Sigma \rightarrow Q$ **transition function***

$q_0 \in Q$ **is start state**

$F = \{q_1, q_2\} \subseteq Q$ **accept states**



*

δ	0	1
q_0		
q_1		
q_2		
q_3		

Language of FA

L(M) = the ***language*** of machine M

= set of all strings machine M accepts

M ***recognizes*** the language L(M)