

Intro to Theory of Computation

CS
464

LECTURE 1 Theory of Computation

- Course information
- Overview of the area
- Finite Automata

Sofya Raskhodnikova

8/25/2009

Sofya Raskhodnikova; Intro Theory of Computation

CS
464

Course information

1. Instructors
2. Course website
3. Honors credit
4. Prerequisites
5. Textbook
6. Syllabus
7. Homework logistics
8. Collaboration policy
9. Exams and grading
10. Influenza precautions

8/25/2009

Sofya Raskhodnikova; Intro Theory of Computation

L1.2

CS
464

What is Theory of Computation?

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

8/25/2009

Sofya Raskhodnikova; based on lecture notes by Nancy Lynch

L1.3

CS
464

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)

8/25/2009

Sofya Raskhodnikova; based on lecture notes by Nancy Lynch

L1.4

CS
464

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
 - Real-time systems
 - Mobile computing
 - Embedded systems
 - ...

8/25/2009

Sofya Raskhodnikova; based on lecture notes by Nancy Lynch

L1.5

CS
464

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
 - Suggests general notion of **computability**

8/25/2009

Sofya Raskhodnikova; based on lecture notes by Nancy Lynch

L1.6

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”

Problems solved by machines

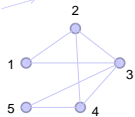
1. What is a problem?

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

- $L_1 =$ {binary representations of natural numbers divisible by 2}
 - $L_2 =$ {binary representations of primes} alphabet = {0,1}
 - $L_3 =$ {sequences of decimal numbers, separated by commas, that can be divided into 2 groups with the same sum}
 - (5,3,1,3) $\in L_3$, (15,7,5,9,1) $\notin L_3$, alphabet = {0,1,...,9,comma}
 - $L_4 =$ {C programs that loop forever when run}
 - $L_5 =$ {representations of graphs containing a *Hamiltonian cycle*}
 - {(1,2,3,4,5); (1,2),(1,3),(2,3),...}
 - visits each node exactly once
- vertices edges
- alphabet = all symbols: digits, commas, parens



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!!!

Is it useful for programmers?



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



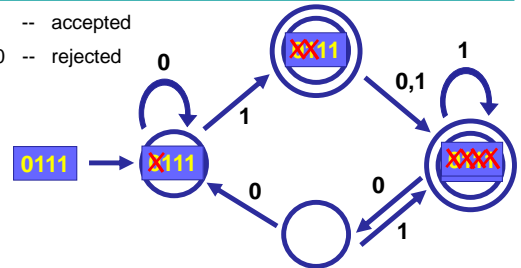
Boss, I can't find an efficient algorithm, because no such algorithm is possible.

Parts of the course

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

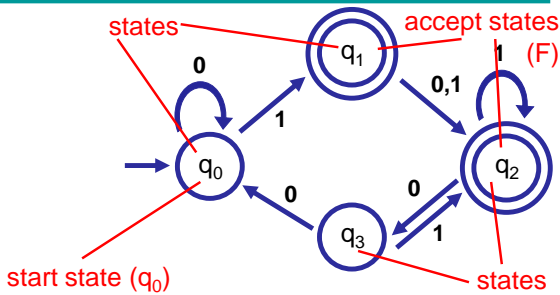
Finite automata (FA)

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



Formal Definition

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

Q is the set of states

Σ is the alphabet

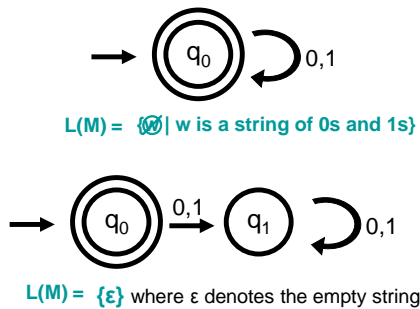
$\delta : Q \times \Sigma \rightarrow Q$ is the transition function

$q_0 \in Q$ is the start state

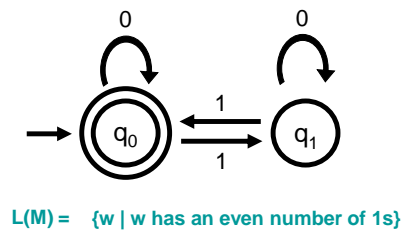
$F \subseteq Q$ is the set of accept states

$L(M)$ = the *language* of machine M
= set of all strings machine M accepts

Examples of FAs

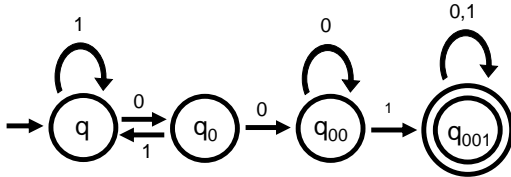


Examples of FAs



Examples of FAs

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



8/25/2009

Sofya Rashodnikova; based on slides by Nick Hopper

L1.19

Regular languages

A language is **regular if it is recognized by a finite automaton**

$L = \{ w \mid w \text{ contains } 001 \}$ is regular

$L = \{ w \mid w \text{ has an even number of } 1\text{s} \}$ is regular

Many interesting programs recognize regular languages

NETWORK PROTOCOLS

COMPILERS

GENETIC TESTING

ARITHMETIC

8/25/2009

Sofya Rashodnikova; based on slides by Nick Hopper

L1.20