Theory of Computation

Lecture 1
Theory of Computation
• Course information
• Overview of the area
• Finite Automata

Sofya Raskhodnikova

Course Information

1. Instructor
2. Course website
3. Prerequisites
4. Lectures
5. Textbook
6. Syllabus
7. Homework
8. Grading policy
9. Collaboration policy
10. Exams and grading

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)

This course
• Theory basics
  – Models for machines
    – Models for the problems machine 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

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.
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

- \( L_1 = \{\text{binary representations of natural numbers divisible by 2}\} \)
- \( L_2 = \{\text{binary representations of primes}\} \)
- \( L_3 = \{\text{sequences of decimal numbers, separated by commas, that can be divided into 2 groups with the same sum}\} \)
  - \((5,3,1,3) \in L_3\)
  - alphabet = \{0,1,9,comma\}
- \( L_4 = \{\text{C programs that loop forever when run}\} \)
- \( L_5 = \{\text{representations of graphs containing a Hamiltonian cycle}\} \)
  - \{(1,2,3,4,5); (1,2),(1,3),(2,3),…\}

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!!!**