# BU CS 332 – Theory of Computation

**Please point a browser to**
**https://forms.gle/u6qQagXBYZn4DYKy6**
**for in-class polls**

- Lecture 1:
  - Course information
  - Overview

Reading:

Sipser Ch 0

Mark Bun

January 22, 2024

# Course Information

# Course Staff



- <u>Me:</u> **Mark** Bun  (he/him/his)
  - Office hours: Tue 5-6:15, Thu 2-3:30
  - Research interests: Theory of computation (!)
    More specifically: Computational complexity, data privacy, cryptography, foundations of machine learning



- <u>Teaching Fellow:</u> Diptaksho Palit
  - Office hours: Wed 4-5:30, Fri 9-10
  - Research interests: Sublinear algorithms, property testing, meta-complexity



- <u>Teaching Assistant:</u> Anming Gu
  - Office hours: Wed 9:30-10:30, Thu 3:30-5

# Course Webpage

https://cs-people.bu.edu/mbun/courses/332_S24/

Serves as the syllabus and schedule

Check back frequently for updates!

## CS 332: Elements of the Theory of Computation, Spring 2024

### Course Overview

This course is an introduction to the theory of computation. This is the branch of computer science that aims to understand which problems can be solved using computational devices and how efficiently those problems can be solved. To be able to make precise statements and rigorous arguments, computational devices are modeled using abstract mathematical "models of computation." The learning objectives of the course are to:

- Foremost, understand how to rigorously reason about computation through the use of abstract, formal models.
- Learn the definitions of several specific models of computation including finite automata, context-free grammars, and Turing machines, learn tools for analyzing their power and limitations, and understand how they are used in other areas of computer science.
- Learn how fundamental philosophical questions about the nature of computation (Are there problems which cannot be solved by computers? Can every problem for which we can quickly verify a solution also be solved efficiently?) can be formalized as precise mathematical problems.
- Gain experience with creative mathematical problem solving and develop the ability to write correct, clear, and concise mathematical proofs.
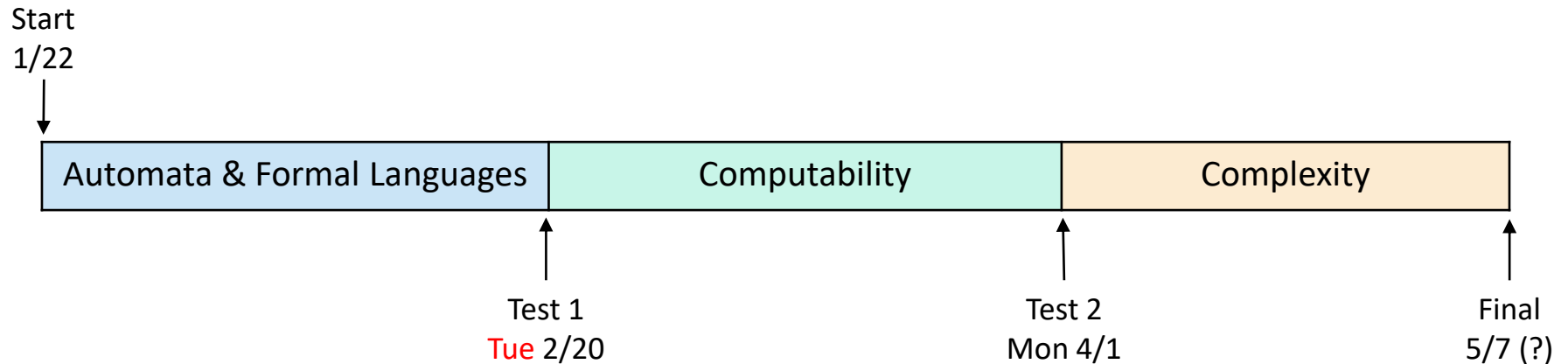
| | |
|---|---|
| **Instructor:** | Mark Bun, mbun [at] bu [dot] edu |
| **Instr. Office Hours:** | Tue 5-6:15 PM |
| | Thu 2-3:30 PM |
| **Teaching Fellow:** | Diptaksho Palit, dpalit [at] bu [dot] edu |
| **TF Office Hours:** | Wed 4-5:30 PM |
| | Fri 9-10 AM |
| **Teaching Assistant:** | Anming Gu, agu2002 [at] bu [dot] edu |
| **TA Office Hours:** | Wed 9:30-10:30 AM |
| | Thu 3:30-5 PM |
| **Class Times:** | Mon, Wed 2:30-3:45 PM (PHO 211) |
| **Discussion Sections:** | Tue 9:30-10:20 AM (FLR 123) |
| | Tue 11:15 AM-12:05 PM (FLR 123) |
| | Tue 12:30-1:20 PM (FLR 123) |
| **Solution Review Session:** | Fri 3:30-4:30 PM |

# Course Structure

Start
1/22

| Automata & Formal Languages | Computability | Complexity |
|:---:|:---:|:---:|

Test 1
Tue 2/20

Test 2
Mon 4/1

Final
5/7 (?)

## Grading

- Homework (45%): Roughly 11 of these
- In-class tests (42%):
  - Test 1 (12%)
  - Test 2 (12%)
  - Final (18%)
- Participation (13%): In-class polls, discussions, HW self-assessments, HW0, etc.

# Homework Policies

- Weekly assignments due Thursday @ 11:59PM
- No late days
- Lowest homework score will be dropped

- Homework to be submitted via Gradescope
  - Entry code: 7DNP62

- You are encouraged to typeset your solutions in LaTeX (resources available on course webpage)

- HW0 out, due Th 1/25 (some housekeeping, brief review)
- HW1 to be released on Th 1/25, due Th 2/1

# Homework Policies: Collaboration

- You are encouraged to work with your classmates to discuss most of the homework problems

- HOWEVER:
    - You may collaborate with at most 3 other students
    - You must acknowledge your collaborators and write "Collaborators: none" if you worked alone
    - You must write your solutions by yourself
    - You may not share written solutions
    - You may not search for solutions on the web or ask Chat-GPT to produce solutions for you
    - You may not receive help from anyone outside the course (including students from previous years)

- Some problems will be marked "INDIVIDUAL". **No collaboration** is allowed on these problems.

# Collaboration Dilemma

If you worked alone on a homework assignment...

(a) You don't need to include a collaboration statement

(b) You should invent a fake collaborator and acknowledge them appropriately

(c) You should include the collaboration statement "Collaborators: none"

(d) You should hurriedly call up three of your friends in the class at 11:55PM, briefly discuss the problems, and acknowledge them
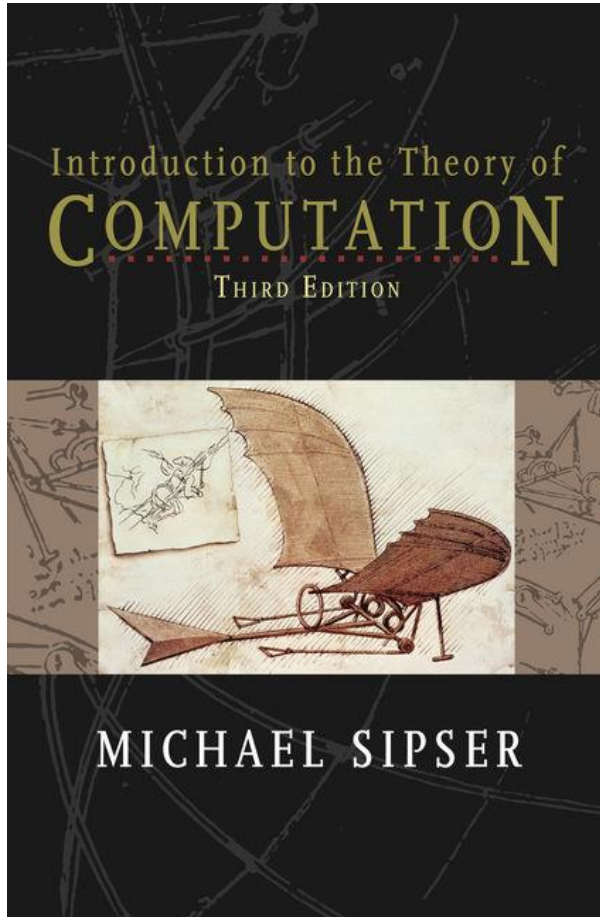
# Homework Policies: Collaboration

Details of the collaboration policy may be found here:
https://cs-people.bu.edu/mbun/courses/332_F24/handouts/collaboration-policy.pdf

**Important:** Sign this document to affirm you understand it, and turn it in via Gradescope by 11:59PM, Thu 1/25

# Textbook

Introduction to the Theory of Computation (Third Edition)
 by Michael Sipser

- It's fine if you want to use an older edition, but section numbers may not be the same

- Other resources available on course webpage

# Participation

- Your class participation score (13% of the course grade) will be determined by

    - Answering in-class polls on Google Forms

    - Handing in completed worksheets at the end of each discussion section

    - Performing brief self-assessments after reviewing posted homework solutions

- You can also increase your participation score by participating thoughtfully in lecture, discussion, office hours, and on Piazza

# Piazza

- We will use Piazza for announcements and discussions
  - Ask questions here and help your classmates
  - Please use private messages / email sparingly

  https://piazza.com/bu/spring2024/cs332

# Expectations and Advice for Succeeding in CS 332

# Learning Objectives

- <u>Understand how to rigorously reason about computation using mathematical models</u>

- Learn about several specific models of computation, how to analyze them, and how they are used throughout computer science

- Learn how to pose deep philosophical questions about the nature of computation as precise mathematical problems

- Gain experience with problem solving and develop proof-writing skills

# Our (the Course Staff's) Responsibilities

- Guide you through difficult parts of the material in lecture

- Encourage active participation in lectures / section

- Assign practice problems and homework that will give you a deep understanding of the material

- Give detailed (formative) feedback on assignments

- Be available outside of class (office hours, Piazza)

- Regularly solicit feedback to improve the course

# Your Responsibilities

- Concepts in this course take <u>time</u> to sink in. Keep at it, and be careful not to fall behind.

- Do the assigned reading on each topic before the corresponding lecture

- Take advantage of office hours

- Participate actively in lectures/sections and on Piazza

- Allocate lots of time for the course: comparable to a project-based course, but spread more evenly

# Prerequisites

*This class is fast-paced and assumes experience with mathematical reasoning and algorithmic thinking*

**You must have passed CS 330 – Intro to Algorithms**

This means you should be comfortable with:

- Set theory
- Functions and relations
- Graphs
- Pigeonhole principle
- Propositional logic

- Asymptotic notation
- Graph algorithms (BFS, DFS)
- Dynamic programming
- NP-completeness

Come talk to me if you have concerns about your preparation for the course

# Advice on Homework

- Start working on homework early! You can get started as soon as it's assigned.

- Spread your homework time over multiple days.

- You may work in groups (of up to 4 people), but <u>think about each problem for at least 30 minutes before your group meeting</u>.

- To learn problem solving, you have to do it:
  - Try to think about how you would solve <span style="color:red">any</span> presented problem before you read/hear the answer
  - Do exercises in the textbook in addition to assigned homework problems

# Advice on Reading

- Not like reading a novel
- The goal is not just to find the answers, but to learn and understand the techniques
- Always try to predict what's coming next
- Always think about how you would approach a problem before reading the solution – maybe you'll discover a different way of solving it!
- This applies to things that are not explicitly labeled as exercises or problems

# Academic Integrity

**Extremely important:** Read and understand the Collaboration and Honesty policy before you sign it

*Violations of the collaboration policy…will result in an automatic failing grade and will be reported to the Academic Conduct Committee (ACC). The ACC often suspends or expels students deemed guilty of plagiarism or other forms of cheating.*

If you find yourself in a desperate situation:

- Hand in as much of the assignment as you're able to complete

- Remember the lowest HW grade is dropped

- Talk to us! We want to help

…cheating is seriously not worth it

# Course Overview

# Objective

Build a *theory* out of the idea of *computation*

# What is "computation"

- Examples:
  - Paper + pencil arithmetic
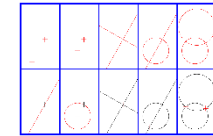  - Abacus
  - Mechanical calculator
  - Ruler and compass geometry constructions
  - Java/C programs on a digital computer

- For us: Computation is the processing of information by the unlimited application of a finite set of operations or rules

# Other examples of computation?

https://www.youtube.com/watch?v=OltvGZUvpvw

# What do we want in a "theory"?

- General ideas that apply to many different systems
- Expressed simply, abstractly, and precisely

- Generality
  - Independence from Technology: Applies to the future as well as the present
  - Abstraction: Suppresses inessential details

- Precision: Can prove formal mathematical theorems
  - Positive results (what *can* be computed): correctness of algorithms and system designs
  - Negative results (what *cannot* be computed): proof that there is no algorithm to solve some problem in some setting (with certain cost)

# Parts of a Theory of Computation

- Models for machines (computational devices)
- 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

This course: Sequential, single-processor computing

Not covered:

- Parallel machines
- Real-time systems
- Distributed systems
- Mobile computing
- Quantum computation
- Embedded systems

# What is a (Computational) Problem?

A single question with infinitely many instances

Examples:

Parity: Given a string consisting of $a$'s and $b$'s, does it contain an even number of $a$'s?

Primality: Given a natural number $x$ (represented in binary), is $x$ prime?

Halting Problem: Given a C program, can it ever get stuck in an infinite loop?

For us: Focus on *decision* problems (yes/no answers) on *discrete* inputs

# What is a (Computational) Problem?

For us: A problem will be the task of determining whether a *string* is in a *language*

# What is a (Computational) Problem?

For us: A problem will be the task of determining whether a *string* is in a *language*

- **Alphabet**: A finite set $\Sigma$

  Ex. $\Sigma = \{a, b, \dots, z\}$

- **String:** A finite concatenation of alphabet symbols

  Ex. $bqr, ababb$

  $\varepsilon$ denotes empty string, length 0

  $\Sigma^*$ = set of all strings using symbols from $\Sigma$

- **Language:** A set $L$ of strings

# Examples of Languages

Parity: Given a string consisting of $a$'s and $b$'s, does it contain an even number of $a$'s?

$\Sigma =$                   $L =$

Primality: Given a natural number $x$ (represented in binary), is $x$ prime?

$\Sigma =$                   $L =$

Halting Problem: Given a C program, can it ever get stuck in an infinite loop?

$\Sigma =$                   $L =$

# Primality language

Which best represents the language corresponding to the Primality problem? (I.e., strings over the alphabet $\{0, 1\}$ that are binary representations of prime numbers.)

Let's say the most significant bit is on the left, so "100" is the binary representation of 4.

(a) $\{2, 3, 5, 7, \dots\}$

(b) $\{10, 11, 101, 111, \dots\}$

(c) $\{11, 111, 11111, 1111111, \dots\}$

(d) $\{11, 011, 101, 110, 111, 0111, \dots\}$

# Machine Models

Computation is the processing of information by the **unlimited application** of a **finite set** of operations or rules

| Input | $a$ | $b$ | $a$ | $a$ | | | |
|-------|-----|-----|-----|-----|--|--|--|

...
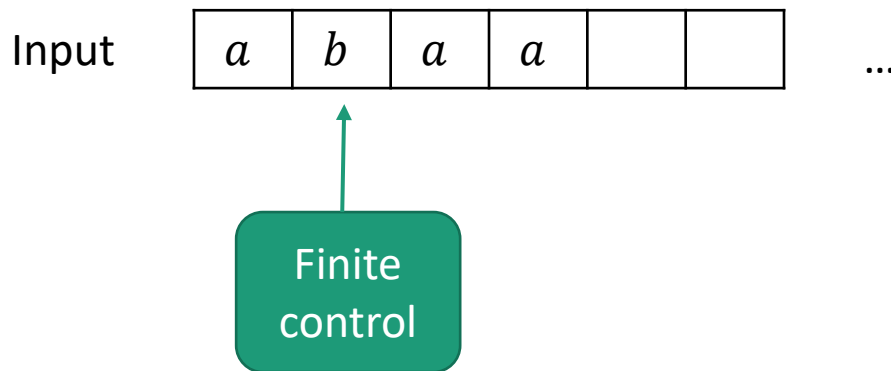
**Finite control**

Abstraction: We don't care how the control is implemented. We just require it to have a finite number of states, and to transition between states using fixed rules.

# Machine Models

- <u>Finite Automata (FAs):</u> Machine with a finite amount of unstructured memory

Input

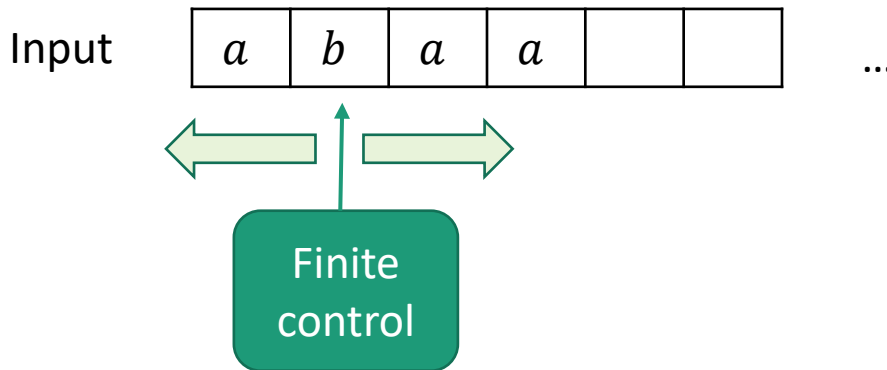| $a$ | $b$ | $a$ | $a$ | | | ...

Finite control
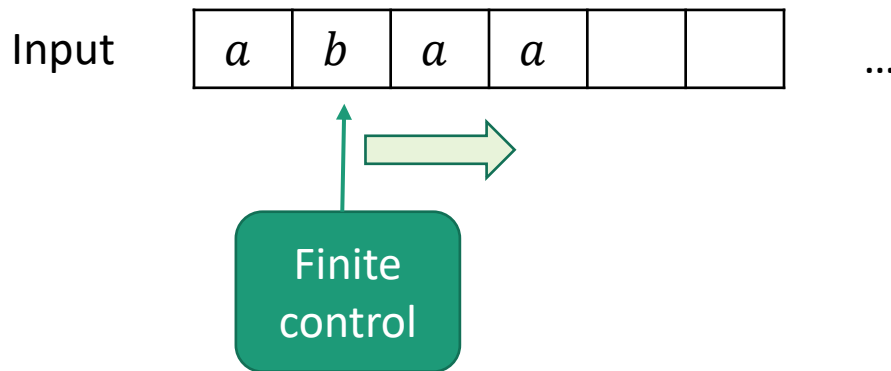
Control scans left-to-right
Can check simple patterns
Can't perform unlimited counting

Useful for modeling chips, simple control systems, choose-your-own adventure games…

# Machine Models

- <u>Turing Machines (TMs):</u> Machine with unbounded, unstructured memory

Input

| $a$ | $b$ | $a$ | $a$ |  |  |

...

Control can scan in both directions
Control can both read and <u>write</u>

Finite control

Model for general sequential computation
Church-Turing Thesis: Everything we intuitively think of as "computable" is computable by a Turing Machine

# What theorems would we like to prove?

We will define classes of languages based on which machines can recognize them

**Inclusion:** Every language recognizable by a FA is also recognizable by a TM

**Non-inclusion:** There exist languages recognizable by TMs which are not recognizable by FAs

**Completeness:** Identify a "hardest" language in a class

**Robustness:** Alternative definitions of the same class

   <u>Ex.</u> Languages recognizable by FAs = regular expressions

# Why study theory of computation?

- You'll learn how to formally reason about computation
- You'll learn the technology-independent foundations of CS

Philosophically interesting questions:

- Are there well-defined problems which cannot be solved by computers?
- Can we always find the solution to a puzzle faster than trying all possibilities?
- Can we say what it means for one problem to be "harder" than another?

# Why study theory of computation?

- You'll learn how to formally reason about computation
- You'll learn the technology-independent foundations of CS

## Connections to other parts of science:

- Finite automata arise in compilers, AI, coding, chemistry
https://cstheory.stackexchange.com/a/14818
- Hard problems are essential to cryptography
- Computation occurs in cells/DNA, the brain, economic systems, physical systems, social networks, etc.

# Why do you want to study the theory of computation?

(a) I want to learn new ways of thinking about computation

(b) I like math and want to see how it's used in computer science

(c) I'm excited about the philosophical questions about the nature of computation

(d) I want to practice problem solving and algorithmic thinking

(e) I want to develop a "computational perspective" on other areas of math/science

(f) I actually wanted to take CS 320 or CS 350 but they were full

# Why study theory of computation?

## Practical knowledge 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 exists."

## Will you be asked about this material on job interviews?
No promises, but a true story…