CS 361: Theory of Computation

Lecture 1: Introduction & Logistics

Shikha Singh

Announcements & Logistics

- **Welcome to the first class!**
- Things to grab: **Exercise 1** (due next lecture) and **Lecture 1 Handout**
- Make sure to sign the attendance sheet that is circulating
- Course website (https://williams-cs.github.io/cs361-f25/)

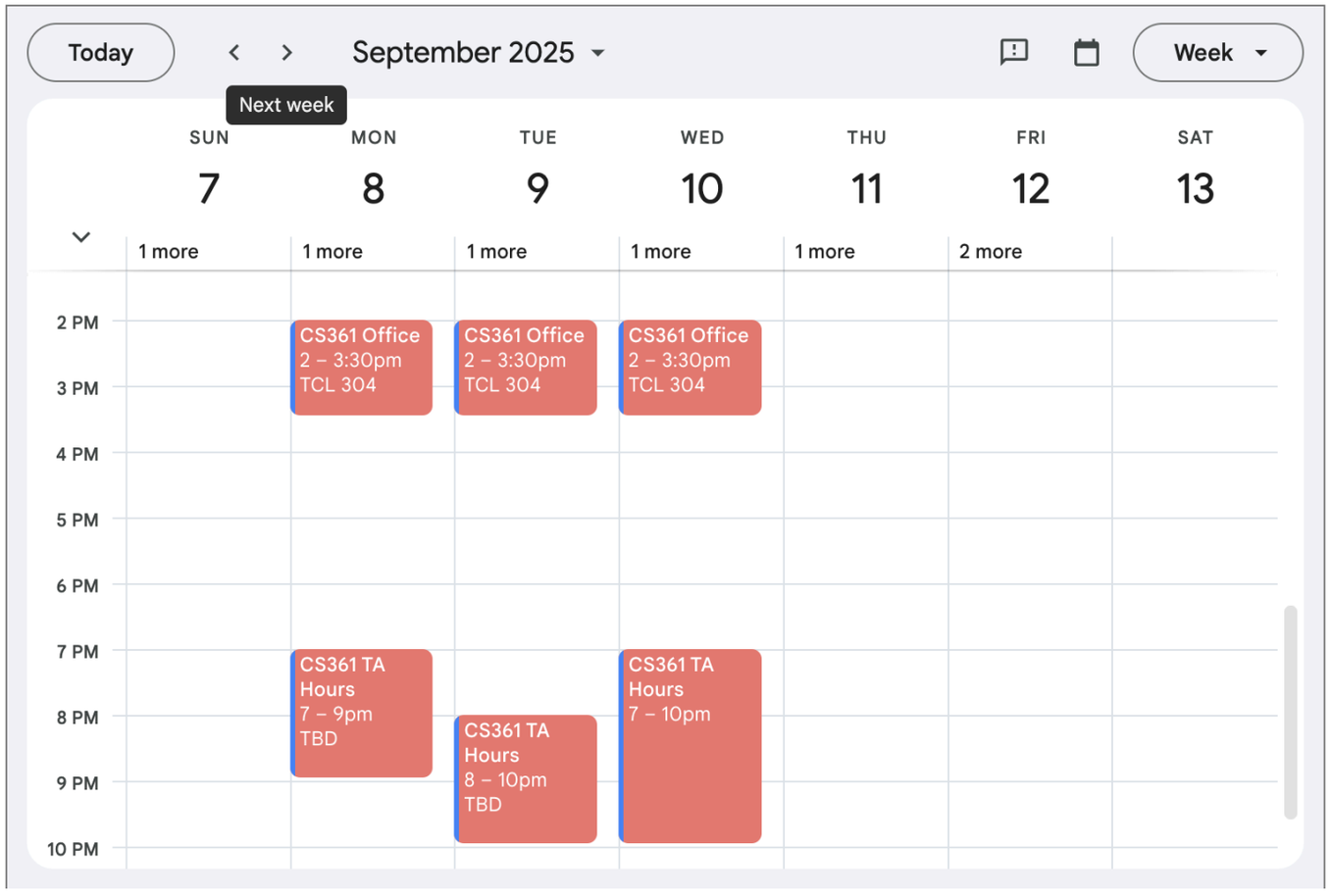
CSCI 361 - Fall 2025

Theory of Computation

Home | Lectures | Assignments | Resources | CS@Williams

Home

Instructor:	Shikha Singh
Email:	ss32@williams.edu
GLOW page:	CSCI 361 GLOW
Office Hours:	Check the calendar below.
Lectures:	TR 9:55 am - 11:10 am
Classroom	Schow 30A
Textbook	Introduction to the Theory of Computation by Michael Sipser (3rd ed)
	Problem sets will typically be due Wed @ 10 pm



About Me

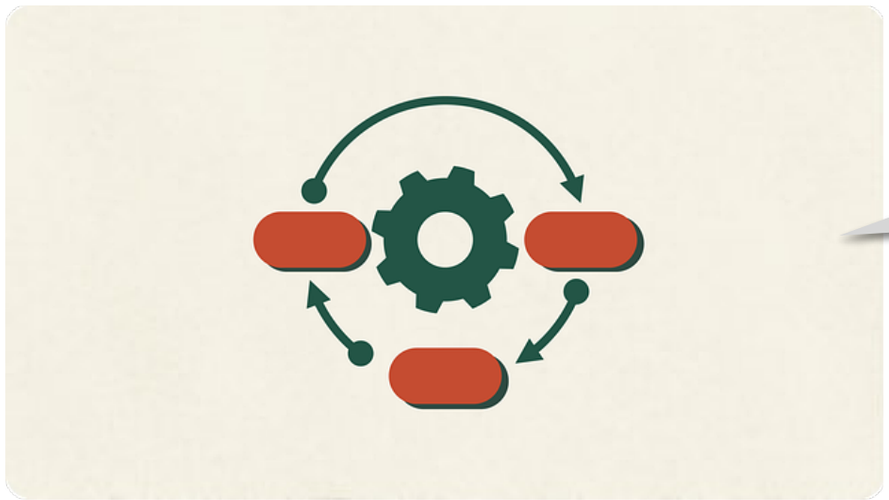
- Go by Shikha or Prof Singh (she/her)
- 7th year at Williams
- Typically teach intro and theory courses
    - CS 134, CS 256, CS 357, CS 361
- Research area
    - Data structures
    - Algorithmic Game Theory
    - Complexity Theory
    - New direction: Algorithms with learned predictions

What is Theory of Computation?

What is Computation?

What is Computation

Input

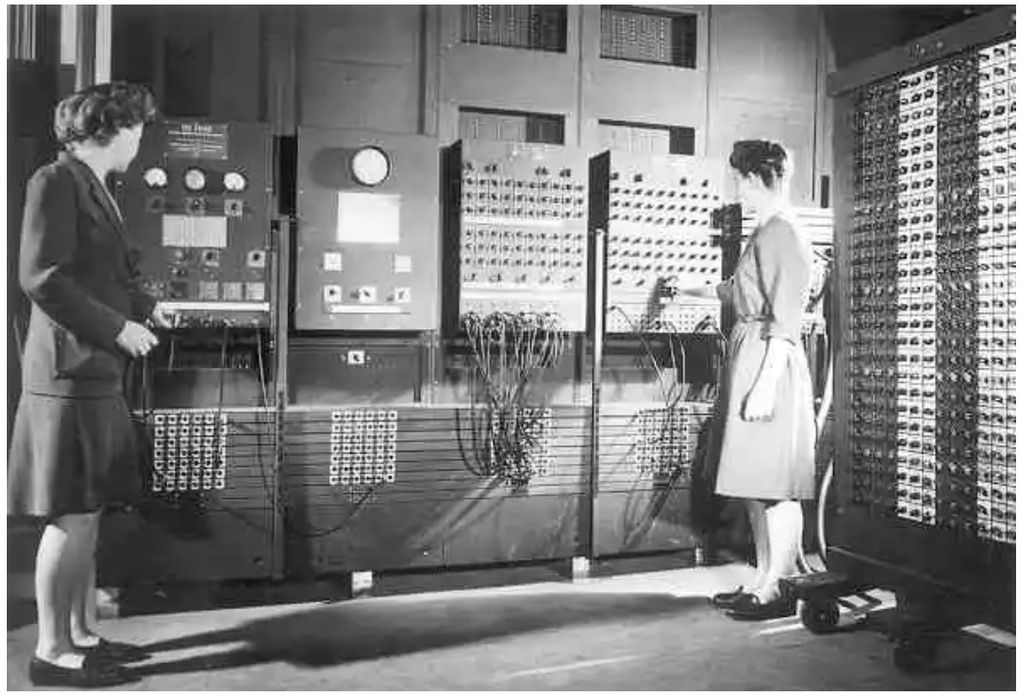


"Computer"

Output

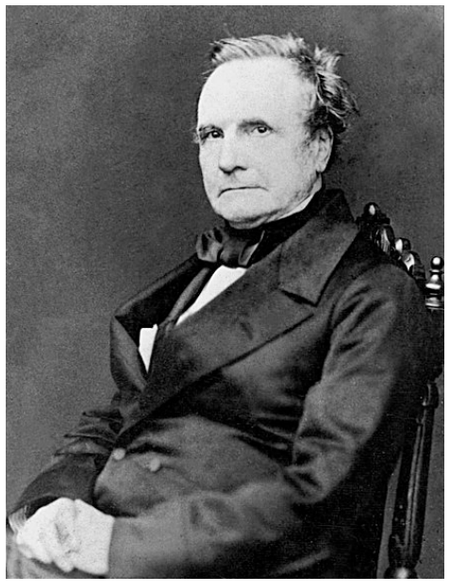
Computers Now

Computers: Early 20th Century



Earliest: Analytic Engine and Note G

Charles Babbage's Analytic Engine was the first universal computing device ever designed. It followed his Difference Engine for evaluating polynomial functions in the 1830s. Both machines were mechanical and were never fully built at the time. However, sophisticated algorithms were written for it by Babbage and Ada Lovelace. In particular, Lovelace's Note G is the first ever published algorithm.



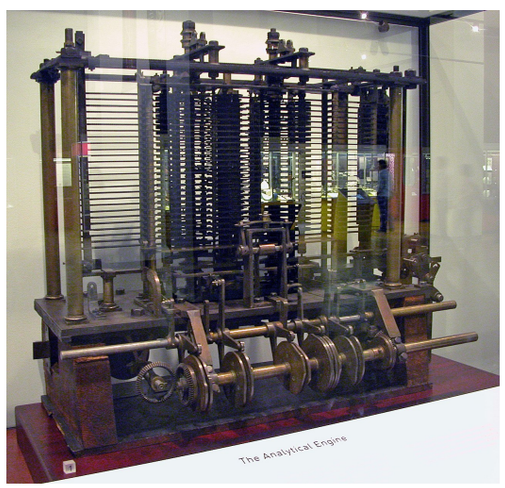


Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

Number of Operation.	Nature of Operation.	Variables acted upon.	Variables receiving results.	Indication of change in the value on any Variable.	Statement of Results.	Data.			Working Variables.									Result Variables.			
						$^1V_1$ 0 0 0 1 [1]	$^1V_2$ 0 0 0 2 [2]	$^1V_3$ 0 0 0 4 [n]	$^0V_4$ 0 0 0 0 [ ]	$^0V_5$ 0 0 0 0 [ ]	$^0V_6$ 0 0 0 0 [ ]	$^0V_7$ 0 0 0 0 [ ]	$^0V_8$ 0 0 0 0 [ ]	$^0V_9$ 0 0 0 0 [ ]	$^0V_{10}$ 0 0 0 0 [ ]	$^0V_{11}$ 0 0 0 0 [ ]	$^0V_{12}$ 0 0 0 0 [ ]	$^0V_{13}$ ..... 0 0 0 0 [ ]	$^1V_{21}$ B_1, in a decimal fraction. [B_1]	$^1V_{22}$ B_2, in a decimal fraction. [B_2]	$^1V_{23}$ B_3, in a decimal fraction. [B_3]	$^0V_{24}$... 0 0 0 0 [B_7]
1	×	$^1V_2 \times ^1V_3$	$^1V_4, ^1V_5, ^1V_6$	$\left\{\begin{matrix}^1V_2 = {}^1V_2 \\ ^1V_3 = {}^1V_3\end{matrix}\right\}$	$= 2n$	...	2	n	$2n$	$2n$	$2n$											
2	−	$^1V_4 - ^1V_1$	$^2V_4$	$\left\{\begin{matrix}^1V_4 = {}^2V_4 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n-1$	1	...	...	$2n-1$													
3	+	$^1V_5 + ^1V_1$	$^2V_5$	$\left\{\begin{matrix}^1V_5 = {}^2V_5 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n+1$	1	...	...	...	$2n+1$												
4	÷	$^2V_5 \div ^2V_4$	$^1V_{11}$	$\left\{\begin{matrix}^2V_5 = {}^1V_5 \\ ^2V_4 = {}^0V_4\end{matrix}\right\}$	$= \frac{2n-1}{2n+1}$	...	...	...	0	0	...	...	...	...	...	$\frac{2n-1}{2n+1}$						
5	÷	$^1V_{11} \div ^1V_2$	$^2V_{11}$	$\left\{\begin{matrix}^1V_{11} = {}^2V_{11} \\ ^1V_2 = {}^1V_2\end{matrix}\right\}$	$= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$	...	2	...	...	...	...	...	...	...	...	$\frac{1}{2} \cdot \frac{2n-1}{2n+1}$						
6	−	$^0V_{13} - ^2V_{11}$	$^1V_{13}$	$\left\{\begin{matrix}^2V_{11} = {}^0V_{11} \\ ^0V_{13} = {}^1V_{13}\end{matrix}\right\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$	...	...	...	...	...	...	...	...	...	...	0	.....	$-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$				
7	−	$^1V_3 - ^1V_1$	$^1V_{10}$	$\left\{\begin{matrix}^1V_3 = {}^1V_3 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= n-1\ (=3)$	1	...	n	...	...	...	...	...	...	$n-1$							
8	+	$^1V_2 + ^0V_7$	$^1V_7$	$\left\{\begin{matrix}^1V_2 = {}^1V_2 \\ ^0V_7 = {}^1V_7\end{matrix}\right\}$	$= 2+0 = 2$	...	2	...	...	...	...	2										
9	÷	$^1V_6 \div ^1V_7$	$^2V_{11}$	$\left\{\begin{matrix}^1V_6 = {}^1V_6 \\ ^0V_{11} = {}^2V_{11}\end{matrix}\right\}$	$= \frac{2n}{2} = A_1$	...	...	...	...	...	$2n$	2	...	...	...	$\frac{2n}{2} = A_1$						
10	×	$^1V_{21} \times ^2V_{11}$	$^1V_{12}$	$\left\{\begin{matrix}^1V_{21} = {}^1V_{21} \\ ^2V_{11} = {}^2V_{11}\end{matrix}\right\}$	$= B_1 \cdot \frac{2n}{2} = B_1 A_1$	...	...	...	...	...	...	...	...	...	...	$\frac{2n}{2} = A_1$	$B_1 \cdot \frac{2n}{2} = B_1 A_1$	.....	B_1			
11	+	$^1V_{12} + ^1V_{13}$	$^2V_{13}$	$\left\{\begin{matrix}^1V_{12} = {}^0V_{12} \\ ^1V_{13} = {}^2V_{13}\end{matrix}\right\}$	$= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$ ...	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$				
12	−	$^1V_{10} - ^1V_1$	$^2V_{10}$	$\left\{\begin{matrix}^1V_{10} = {}^2V_{10} \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= n-2\ (=2)$	1	...	...	...	...	...	...	...	...	$n-2$							
13	−	$^1V_6 - ^1V_1$	$^2V_6$	$\left\{\begin{matrix}^1V_6 = {}^2V_6 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n-1$	1	...	...	...	...	$2n-1$											
14	+	$^1V_1 + ^1V_7$	$^2V_7$	$\left\{\begin{matrix}^1V_1 = {}^1V_1 \\ ^1V_7 = {}^2V_7\end{matrix}\right\}$	$= 2+1 = 3$	1	...	...	...	...	...	3										
15	÷	$^2V_6 \div ^2V_7$	$^1V_8$	$\left\{\begin{matrix}^2V_6 = {}^2V_6 \\ ^2V_7 = {}^2V_7\end{matrix}\right\}$	$= \frac{2n-1}{3}$	...	...	...	...	...	$2n-1$	3	$\frac{2n-1}{3}$									
16	×	$^1V_8 \times ^3V_{11}$	$^4V_{11}$	$\left\{\begin{matrix}^1V_8 = {}^0V_8 \\ ^3V_{11} = {}^4V_{11}\end{matrix}\right\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3}$	...	...	...	...	...	...	...	0	...	...	$\frac{2n}{2} \cdot \frac{2n-1}{3}$						
17	−	$^2V_6 - ^1V_1$	$^3V_6$	$\left\{\begin{matrix}^2V_6 = {}^3V_6 \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= 2n-2$	1	...	...	...	...	$2n-2$											
18	+	$^1V_1 + ^2V_7$	$^3V_7$	$\left\{\begin{matrix}^1V_1 = {}^1V_1 \\ ^2V_7 = {}^3V_7\end{matrix}\right\}$	$= 3+1 = 4$	1	...	...	...	...	...	4										
19	÷	$^3V_6 \div ^3V_7$	$^1V_9$	$\left\{\begin{matrix}^3V_6 = {}^3V_6 \\ ^3V_7 = {}^3V_7\end{matrix}\right\}$	$= \frac{2n-2}{4}$	...	...	...	...	...	$2n-2$	4	...	$\frac{2n-2}{4}$	...	$\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = A_3\right\}$						
20	×	$^1V_9 \times ^4V_{11}$	$^5V_{11}$	$\left\{\begin{matrix}^1V_9 = {}^0V_9 \\ ^4V_{11} = {}^5V_{11}\end{matrix}\right\}$	$= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$	...	...	...	...	...	...	...	...	0								
21	×	$^1V_{22} \times ^5V_{11}$	$^0V_{12}$	$\left\{\begin{matrix}^1V_{22} = {}^1V_{22} \\ ^0V_{12} = {}^2V_{12}\end{matrix}\right\}$	$= B_3 \cdot \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3} = B_3 A_3$	...	...	...	...	...	...	...	...	0			$B_1 A_3$	.....	.....	B_3		
22	+	$^2V_{12} + ^2V_{13}$	$^3V_{13}$	$\left\{\begin{matrix}^2V_{12} = {}^0V_{12} \\ ^2V_{13} = {}^3V_{13}\end{matrix}\right\}$	$= A_0 + B_1 A_1 + B_3 A_3$	...	...	...	...	...	...	...	...	...	...	.....	0	$\left\{A_3 + B_1 A_1 + B_2 A_3^{\cdot}\right\}$				
23	−	$^2V_{10} - ^1V_1$	$^3V_{10}$	$\left\{\begin{matrix}^2V_{10} = {}^3V_{10} \\ ^1V_1 = {}^1V_1\end{matrix}\right\}$	$= n-3\ (=1)$	1	...	...	...	...	...	...	...	...	$n-3$							

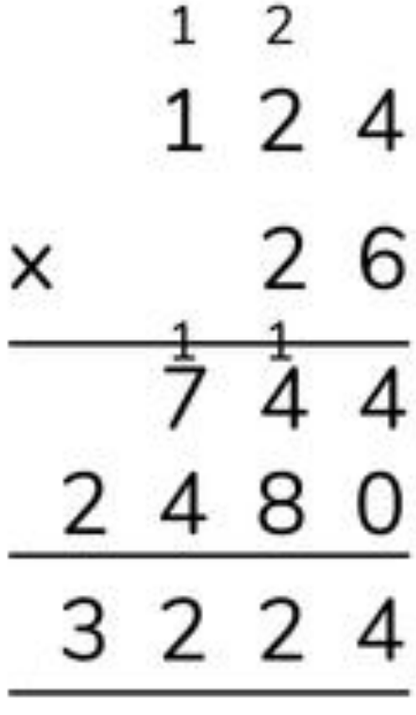
Here follows a repetition of Operations thirteen to twenty-three.

24	+	$^4V_{12} + ^0V_{24}$	$^1V_{24}$	$\left\{\begin{matrix}^4V_{12} = {}^0V_{12} \\ ^0V_{24} = {}^1V_{24}\end{matrix}\right\}$	$= B_7$	...	...	...	...	...	...	...	...	...	...	.....	.....	.....	.....	.....	.....	B_7
25	+	$^1V_1 + ^1V_3$	$^1V_3$	$\left\{\begin{matrix}^1V_1 = {}^1V_1 \\ ^1V_3 = {}^1V_3 \\ ^5V_6 = {}^0V_6 \\ ^5V_7 = {}^0V_7\end{matrix}\right\}$ by a Variable-card. by a Variable card.	$= n+1 = 4+1 = 5$	1	...	$n+1$	...	...	0	0	...	...	...								

[1843] Lovelace's Note G computes Bernoulli numbers

Algorithms are Ancient

- Algorithms have been around for thousands of years
  - Grade-school multiplication algorithm was invented by Babylonians
  - Euclid's GCD algorithm (~300 BC)




```
1: procedure GCD(a, b)
2:     while a ≠ b do
3:         if a > b then
4:             a ← a − b
5:         else
6:             b ← b − a
7:         end if
8:     end while
9:     return a
10: end procedure
```

How Do We Define an Algorithm?

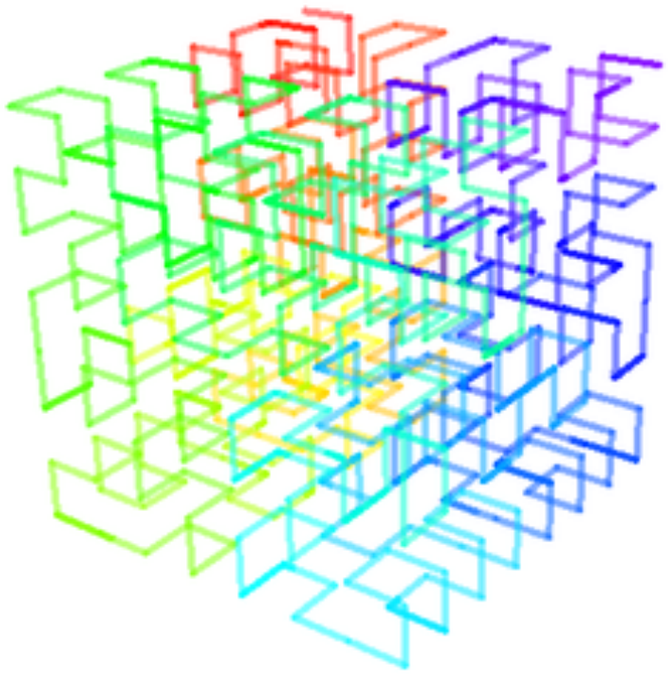
- What constitutes an algorithm? How do we define it?
    - Intuitively, step by step process
    - That eventually terminates and produces the desired output
- To design algorithms, this intuitive understanding was sufficient
- Components of an algorithm:
    - **Specification**: What is the task the algorithm performs
    - **Implementation:** How is the task accomplished
    - **Analysis:** Is it correct? Is it efficient?
- **Question.** Can all problems be solved by some algorithm?

Theory of Computation

- Need a formal model of what it means to solve a problem
- Theory of Computation:
  - Building a mathematical model for computation
  - Using the model to understand the power and limits of computation
  - Gain insights that inform applications

Where it Started: Hilbert's Challenges

- **[1900-1930]** David Hilbert identifies several mathematical problems as the challenges for the coming century
- Two of these problems concerned Computer Science



1862 - 1943

Hilbert's 10th Problem

- Hilbert's 10th problem [1900]:
  - Given a multivariate polynomial with integer coefficients, is there a **process that determines in a finite number of operations** whether the equation has an integral solution

$$3x^2 - 2xy - y^2z - 7 = 0$$

$$x = 1, y = 2, z = -2$$

Integer solution

$$x^2 + y^2 + 1 = 0$$

No solution

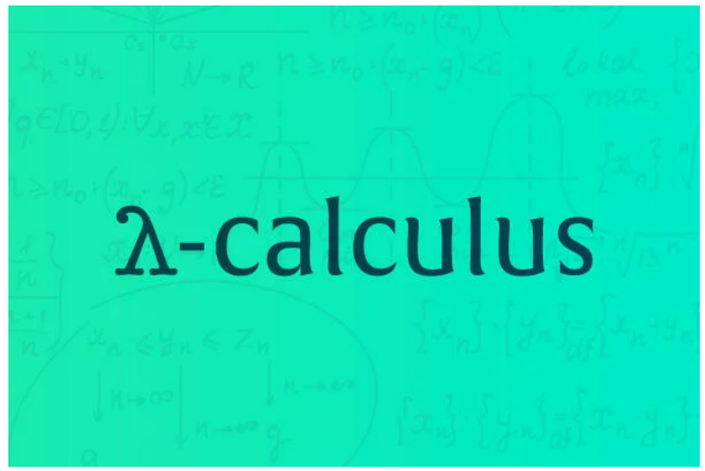
Hilbert's Entscheidungsproblem

- Hilbert believed the answer was yes but many attempts failed
- To show it was not solvable, more formalization was needed of what it means to use a "finite procedure" to solve a problem
- **Hilbert's Entscheidungsproblem** (Decision problem) [1928]:
    - **Is there a finite procedure** that determines whether a given mathematical statement is true or false?
- Hilbert again believed the answer was yes:
    - There was no such thing as an unsolvable problem

Hilbert's Tombstone: "Wir müssen wissen. Wir werden wissen = We must know. We shale know"

Attempts to Define Computation

- **[1930s: Post, Gödel, Church]** attempt to solve Hilbert's Entscheidungsproblem
  - Post machine, lambda calculus, Gödel machine



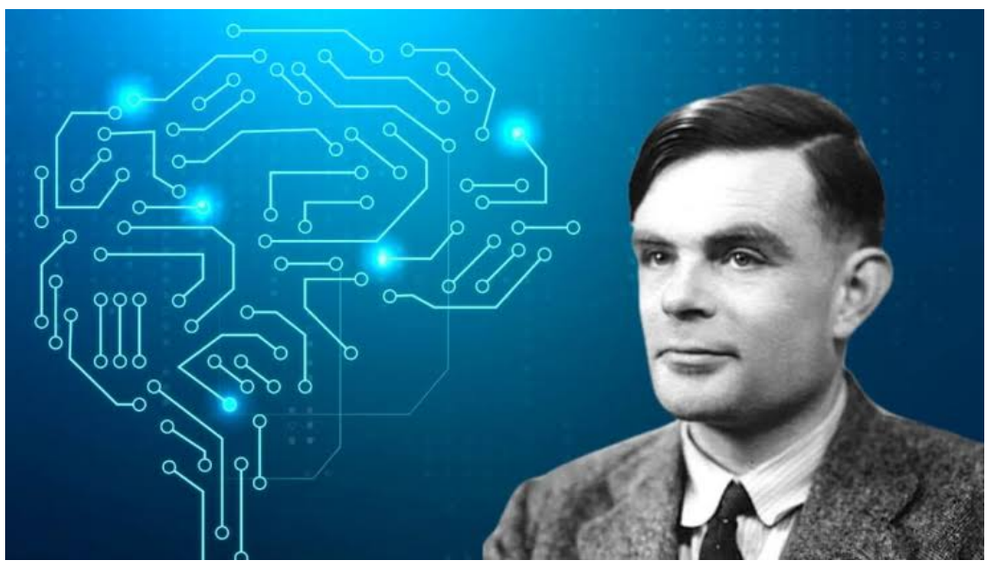
Emil Post

Kurt Gödel

Alonzo Church

Birth of Computer Science

- **[1936]** As a graduate student Alan Turing (at age 24) devised what is now called the Turing machine and used it to devise an uncomputable problem
- Church and Turing:
  - λ-computable $\equiv$ Turing computable



ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO THE ENTSCHEIDUNGSPROBLEM

By A. M. TURING.

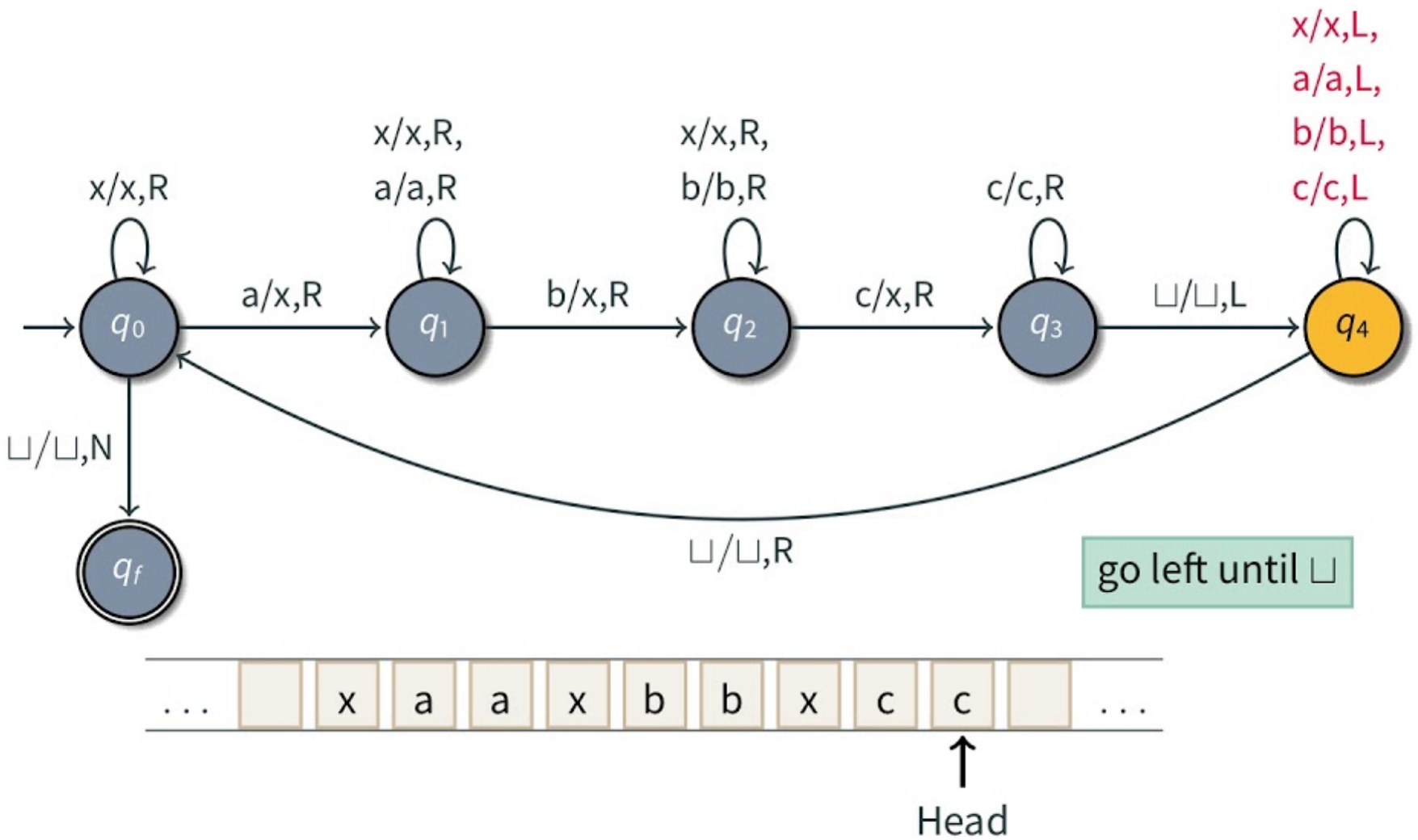
[Received 28 May, 1936.—Read 12 November, 1936.]

The "computable" numbers may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means. Although the subject of this paper is ostensibly the computable *numbers*, it is almost equally easy to define and investigate computable functions of an integral variable or a real or computable variable, computable predicates, and so forth. The fundamental problems involved are, however, the same in each case, and I have chosen the computable numbers for explicit treatment as involving the least cumbrous technique. I hope shortly to give an account of the relations of the computable numbers, functions, and so forth to one another. This will include a development of the theory of functions of a real variable expressed in terms of computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers π, e, etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In § 8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel†. These results

† Gödel, "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I", *Monatshefte Math. Phys.*, 38 (1931), 173–198.

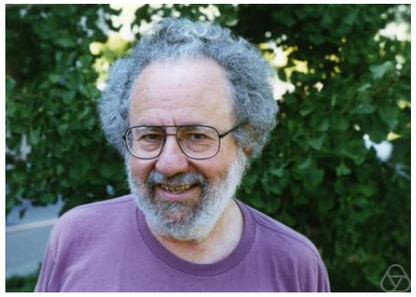

x/x,R
a/a,R
b/b,R
c/c,R
x/x,L
a/a,L
b/b,L
c/c,L
a/x,R
b/x,R
c/x,R
␣/␣,L
␣/␣,N
␣/␣,R
q_0
q_1
q_2
q_3
q_4
q_f
go left until ␣
. . .
x a a x b b x c c
. . .
Head

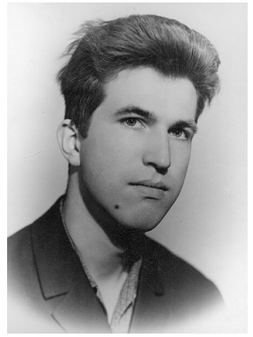
Church and Turing Thesis

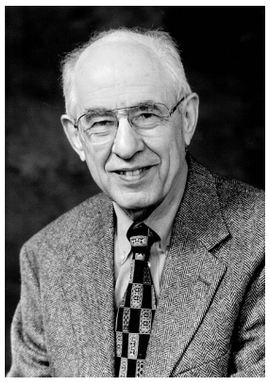
- Turing machines predates modern computer as we know it, yet there is no existing physical model that cannot be modeled by it
- **Church-Turing Thesis**:
    - Intuitive notion of "computable" is captured by functions computer by a Turing machine
    - **[Physical CT Thesis]** Any computational problem that can be solved by any physical process, can be solved by a Turing machine

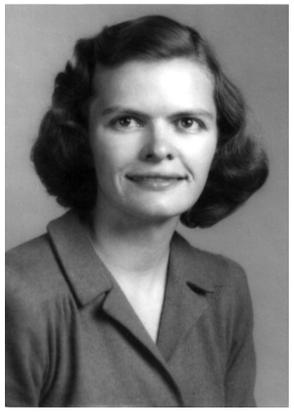
Hilbert's Challenges: Conclusion

- Hilbert's 10th problem [1900]:
  - Given a multivariate polynomial with integer coefficients, is there a **process that determines in a finite number of operations** whether the equation is solvable
  - **No:** Martin Davis, Yuri Matiyasevich, Hilary Putnam and Julia Robinson [1970]

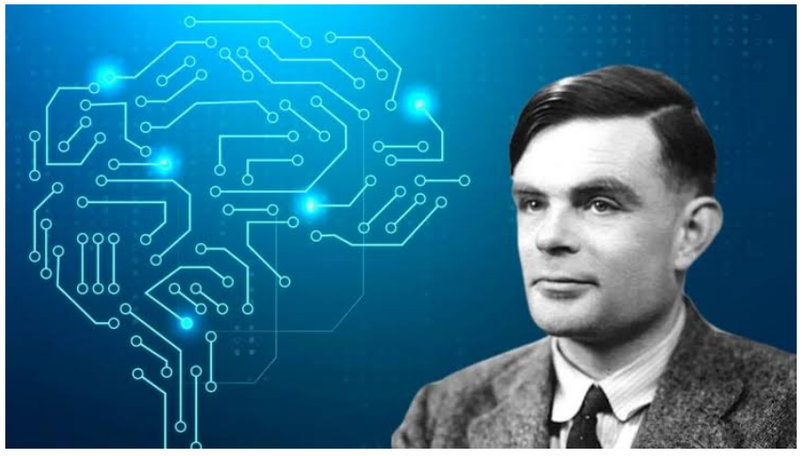








- Hilbert's Entscheidungsproblem (Decision problem) [1928]:
  - **Is there a finite procedure** that determines whether a given mathematical statement is true or false?
  - **No:** Alan Turing [1936]



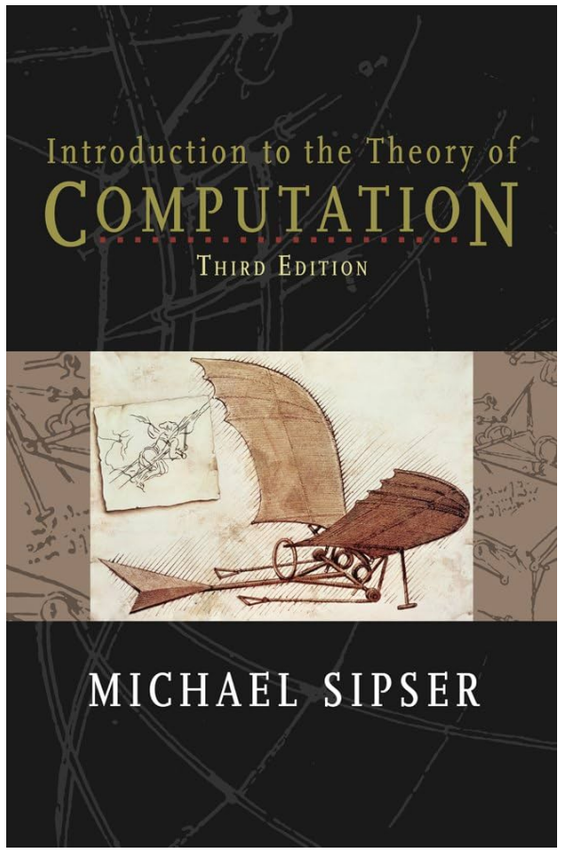
Computability and Complexity

- **Computability** of a problem:
  - Can a problem be solved by a given computational model
  - Start with restricted models: automaton
  - Build up to Turing machines (modeling modern computers)
- **Complexity** of a problem:
  - Is there an efficient algorithm to solve it?
  - Efficiency: time and space complexity
  - Practice with reductions
  - Study the hierarchy of these classes

Course Logistics

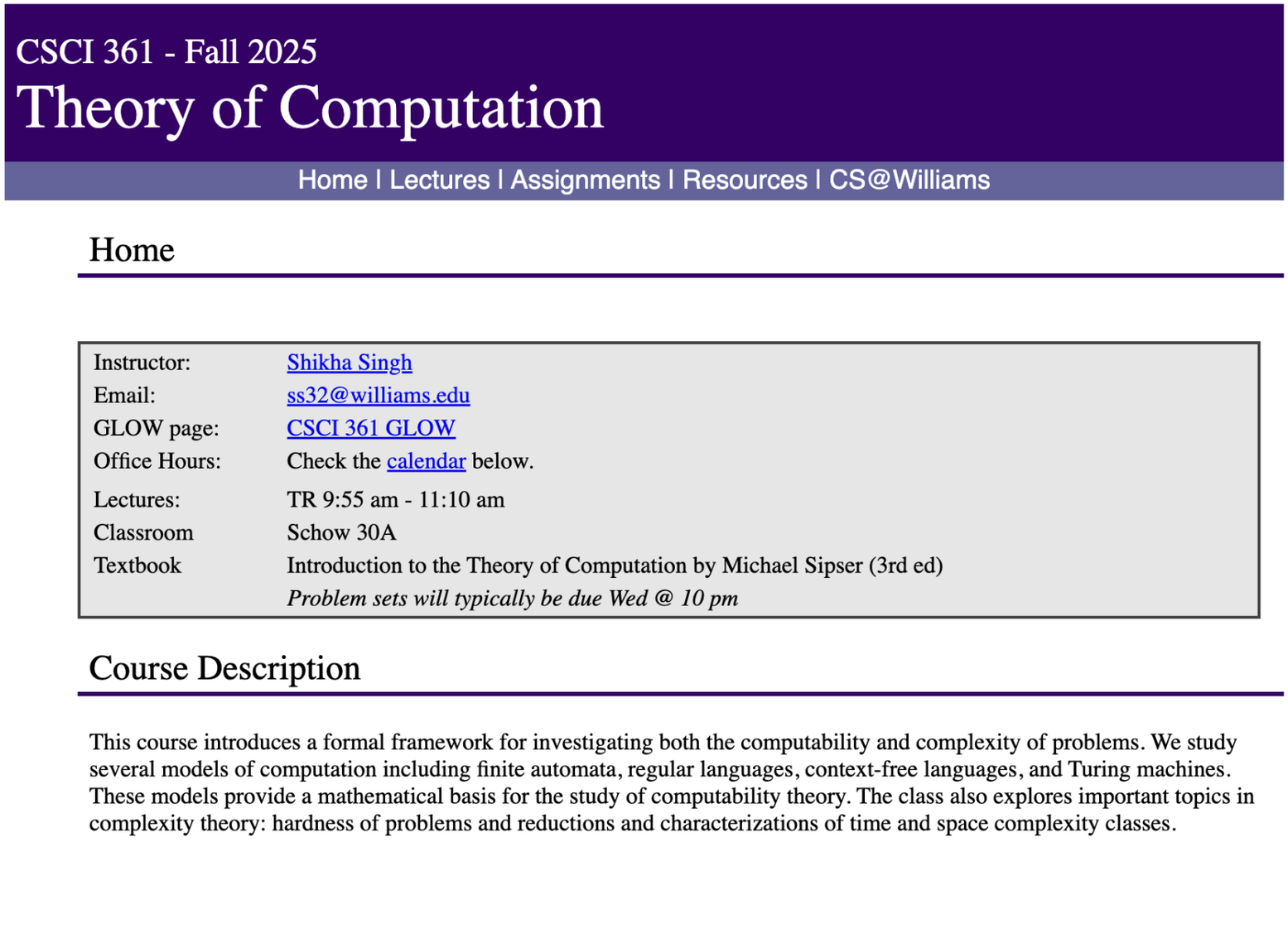
Textbooks

- Primary: Introduction to Theory of Computation (3rd ed) by Sipser
  - Will follow it pretty closely
  - Reserved at Schow if you need it
  - Unofficial PDFs floating around...
- Supplemental readings:
  - Introduction to TCS by Boaz Barak
  - Online: https://introtcs.org/public/index.html



Course Webpage

- Link: https://williams-cs.github.io/cs361-f25/
- Lecture materials, readings and assignments will be posted here
- Will occasionally use GLOW for internal documents



CSCI 361 - Fall 2025

Theory of Computation

Home | Lectures | Assignments | Resources | CS@Williams

Home

Instructor:	Shikha Singh
Email:	ss32@williams.edu
GLOW page:	CSCI 361 GLOW
Office Hours:	Check the calendar below.
Lectures:	TR 9:55 am - 11:10 am
Classroom	Schow 30A
Textbook	Introduction to the Theory of Computation by Michael Sipser (3rd ed)
	Problem sets will typically be due Wed @ 10 pm

Course Description

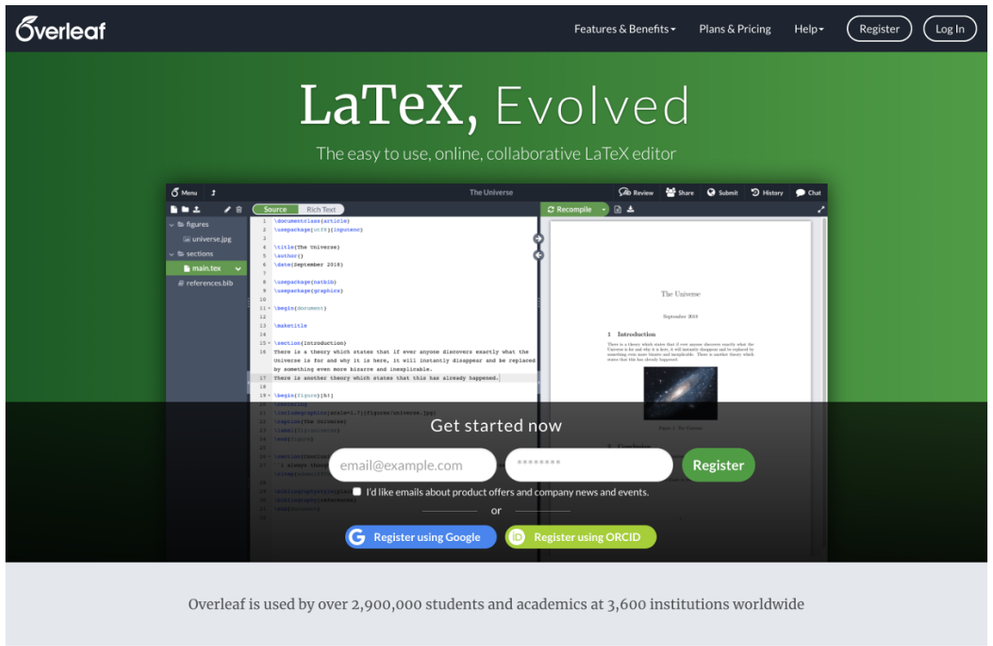
This course introduces a formal framework for investigating both the computability and complexity of problems. We study several models of computation including finite automata, regular languages, context-free languages, and Turing machines. These models provide a mathematical basis for the study of computability theory. The class also explores important topics in complexity theory: hardness of problems and reductions and characterizations of time and space complexity classes.

Syllabus and Grade Breakdown

- Posted on course webpage
  - https://williams-cs.github.io/cs361-f25/handouts/syllabus.pdf
- Grading breakdown:
  - Attendance & Class Participation (5 %)
  - Assignments (20 %)
  - Daily Exercises (10%)
  - Midterms (20 + 20 %)
  - Final Exam (25 %)

Problem Sets / Assignments

- Problem sets for practicing concepts from class
    - Open ended, frequently proofs
- Must be typeset in LaTeX (using provided template in Overleaf)
- Anonymized grading: no name/ID on HWs
- Submit via Gradescope (Course ID: **7XZ2KZ**)
- Assignments will be released on Thursdays and due on Wed at 10 pm



Assignment 1

- First assignment has been released and due Sept 10 Wed at 10 pm
- Based on today's lecture and reading for Tuesday's lecture

CSCI 361 - Fall 2025

Theory of Computation

Home | Lectures | Assignments | Resources | CS@Williams

Assignments

Assignments will be posted here with due dates.

Due Date	Problem Set
Sep 10	Assignment 1 (DFAs and Regular Expressions)

Daily Exercises

- Due each day at the beginning of class:
  - Please grab the next lecture's exercise sheet
- Very short pencil & paper question based on the reading
- Graded mostly on completion
  - Low stakes questions to familiarize you with upcoming definitions
- No late submission accepted:
  - Lowest two will be dropped

Attendance & Class Participation

- Attendance is required in this class
- I like interaction in my classes!
- Most lectures will include in-class problem solving
- Incentive to attend and engage in class: 5% of final grade
- Everyone can miss two-classes at no penalty
    - Otherwise, if you need to miss, reach out ahead of time
- Help build a good community in class
    - Come prepared and help each other!

Bottom line. *Help create a vibrant, positive, and inclusive classroom environment!*

Honor Code

- No collaboration or help on daily exercises - do individually
- Collaboration on problem sets is allowed: can discuss high-level ideas, clarifications, examples to understand the question, etc
    - Should not discuss low-level details, do joint writing
- HW problems are low-stakes practice for exams: should not search the internet/ChatGPT with question specific prompts
    - Shortcuts take away valuable learning opportunities
    - Best way to **train** for the exam is to ***sit with and work through*** the HW problems
- You must arrive at on your own and understand the work you submit

Bottom line. *Any work that is not your own is a violation of the Honor Code.*

How to Succeed in This Class

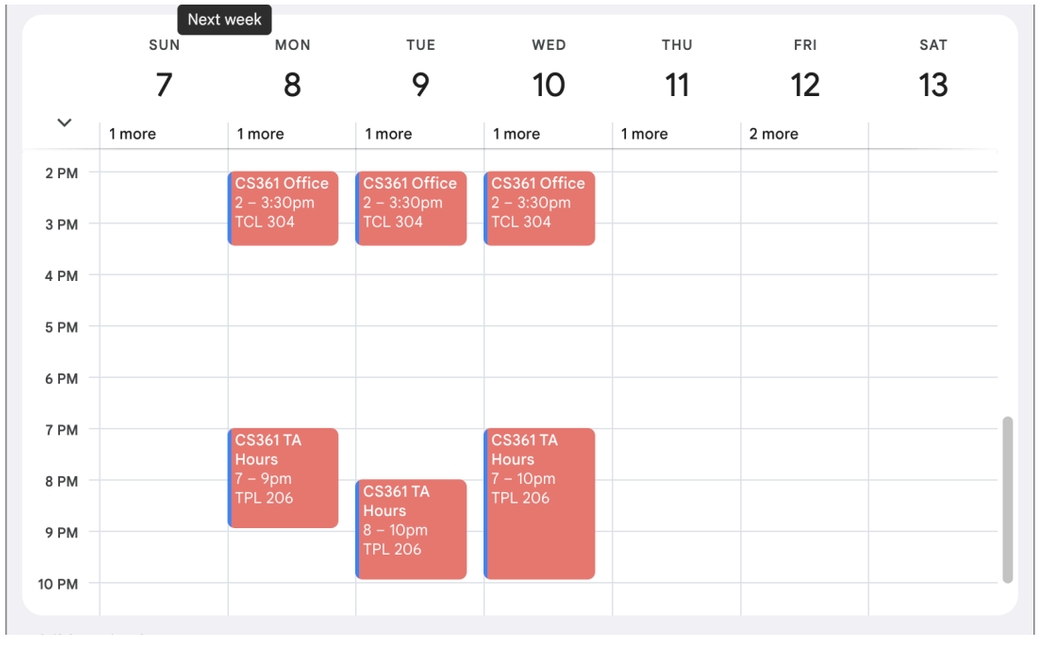
- Do the readings and bring questions to lecture
    - The textbook is very accessible
- Struggle through the problem sets
    - No substitute for trying out different ideas and hitting dead ends
    - This is a workout for your thinking muscles
    - But when stuck.... seek help!
- Proofs are not very different from code
    - Need to edit, polish, debug and craft a "good" proof
    - Read the feedback on problem sets and look at sample solutions to get a sense of the expectations on formalism

Course Support

- Instructor Office Hours:
  - Mon, Tues & Wed: **2 - 3.25 pm** in TCL 304
- Two TAs: Juan Mendez and Luke Zanuck
- TA help hours (starting Monday):
  - Mon 7- 9 pm, Tues 8-10 pm, Wed 7 - 10 pm in TPL 206

Juan

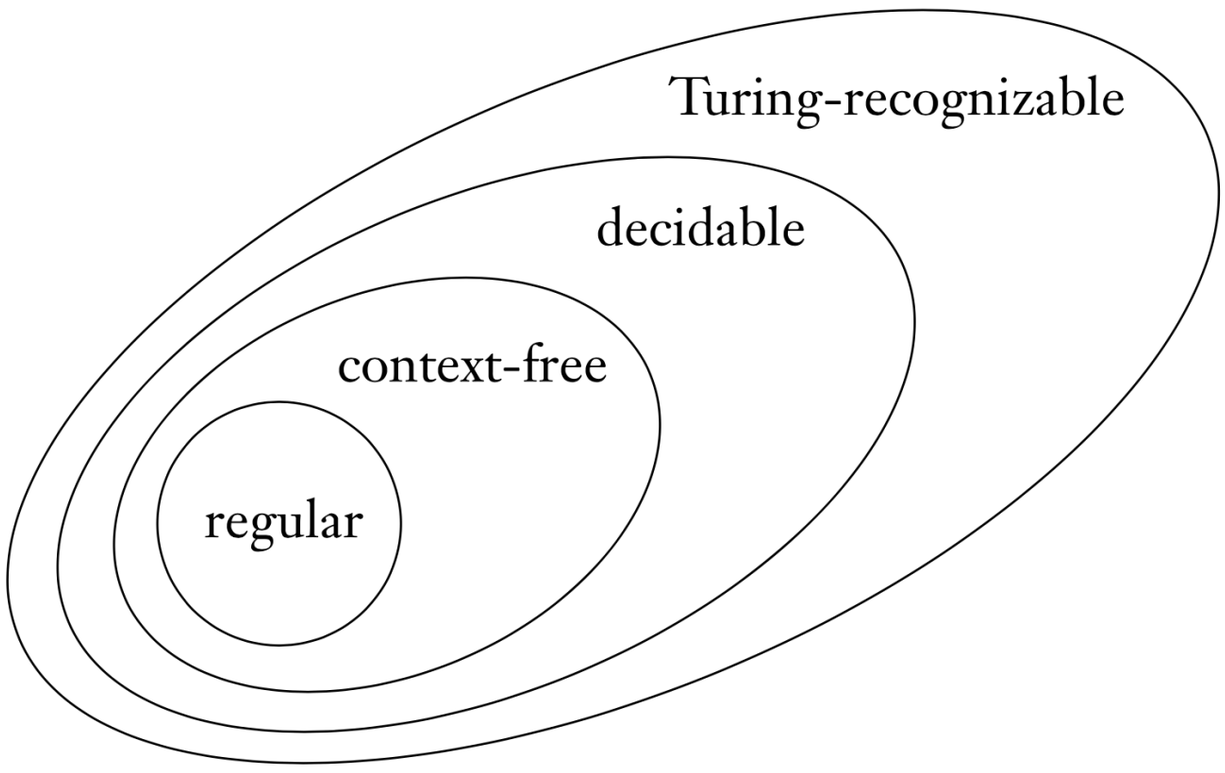
Luke



Course Progression

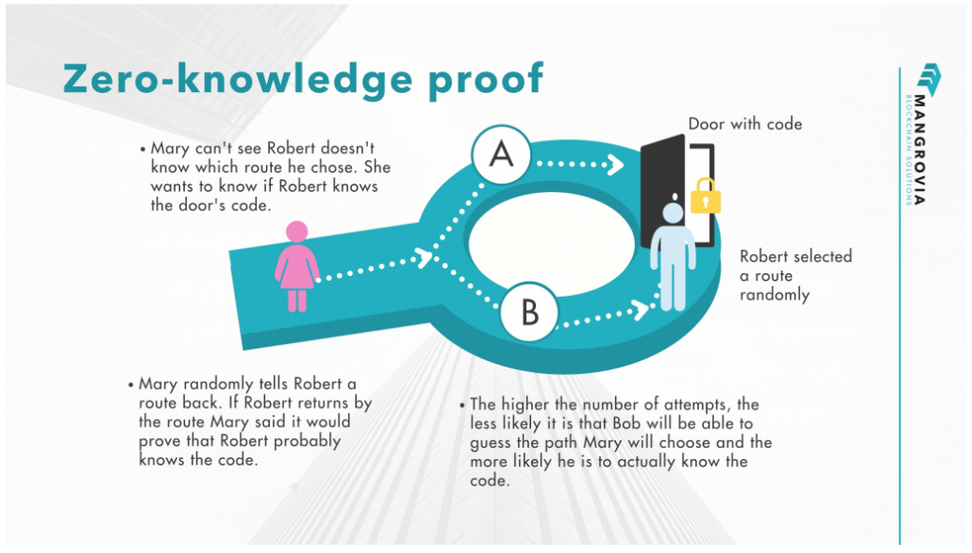
- Finite automaton
- Push-down automaton
- Turing machine

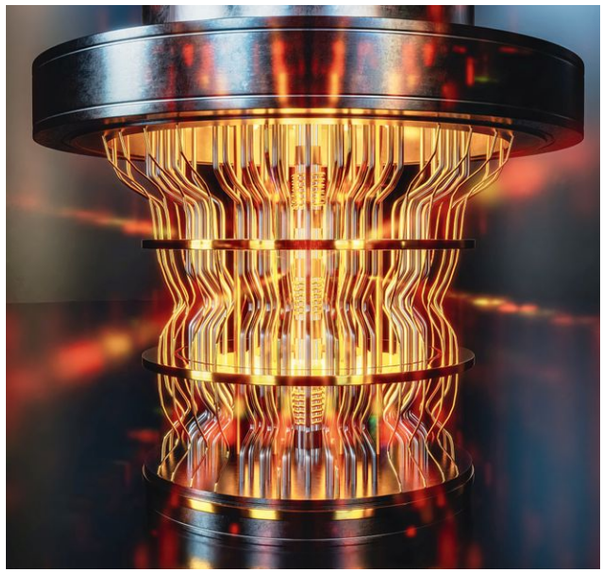
- Sequencing, repetition
- Function calls/ simple recursion
- Simple loops as well as multiple recursive calls

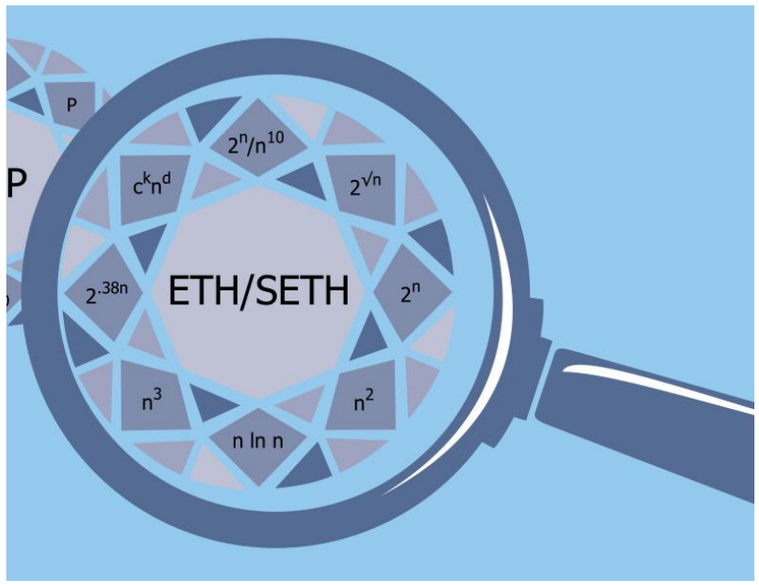


Topic Outline

- Week 1: Sets, Languages & Finite Automaton
- Week 2-3: Regular Languages
- Week 4: Context-free grammars
- Week 5-7: Turing machines and computability
- Week 8-10: Time and Space Complexity, Reductions
- Week 11-12: Advanced topics, student presentations







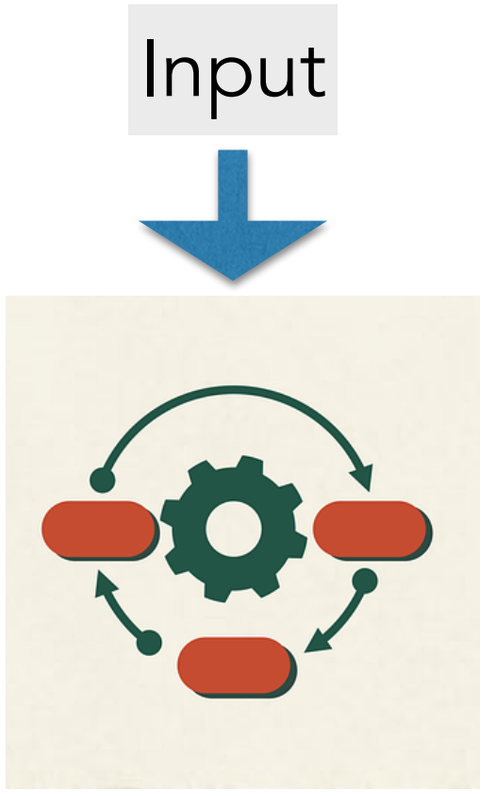
Is this Stuff Useful?

- Typical computer scientist attitude: urgency to "build"
- CSCI 361 is not about building or coding, but....
    - Concepts have stood the test of time and led to many insights
    - Old paradigm: Good theory informs good practice
- Also, concepts in this course have proved particularly applicable
    - **Automaton & RegEx** → Scanners, circuit design, cellular automaton
    - **Context-free grammars** → Building parsers for compilers
    - **Computability and halting program** → Program verification
    - **Formal systems and logic** → Foundation of AI and Databases
    - **Complexity Theory** → Cryptography and Security

Data Representation

Representing Data

- Mathematically modeling input/output?
- Input/output can be any object:
    - Images, text, electrical signals, social network, etc
- What is the typical approach in CS?
    - "Encode" this data into text/strings
    - Specifically binary strings



Encodings

- Not specific to CS languages:
  - Spoken languages encode spoken sounds through letters and words
- **Alphabet** Σ = a non-empty and finite set made up of symbols
- **String** s: a finite (possibly empty) sequence of symbols from Σ
  - $s = a_1 a_2 \cdots a_n$ where each $a_i \in \Sigma$
- Binary strings with $\Sigma = \{0,1\}$
  - ϵ (empty string)
  - $01,\ 000,\ 1110000 \cdots 111$
- Does the choice of alphabet matter?

A B C D E
F G H I K
L M N O
P Q R S T
V X Y Z

अ आ इ ई उ ऊ
ऋ ॠ ऌ ॡ
ए ऐ ओ औ
क ख ग घ ङ
च छ ज झ ञ
ट ठ ड ढ ण
त थ द ध न
प फ ब भ म
य र ल व
श ष स ह

Encoding Input Data as String

- **Length** of a string s denoted $|s| =$ the number of symbols in s
- $\Sigma^* =$ set of all finite-length strings over Σ
- Example:
  - $\{0,1\}^* = \{\epsilon,0,1,00,01,10,11,000,\ldots\}$
  - $\{a\}^* = \{\epsilon, a, aa, aaa, \ldots\}$
- **Note**: Σ^* is an infinite set, but each string in it has a finite length
- Given a set A of objects, an **encoding** is an injective (one-to-one) function that maps A to Σ^*
  - No two objects have the same encoding

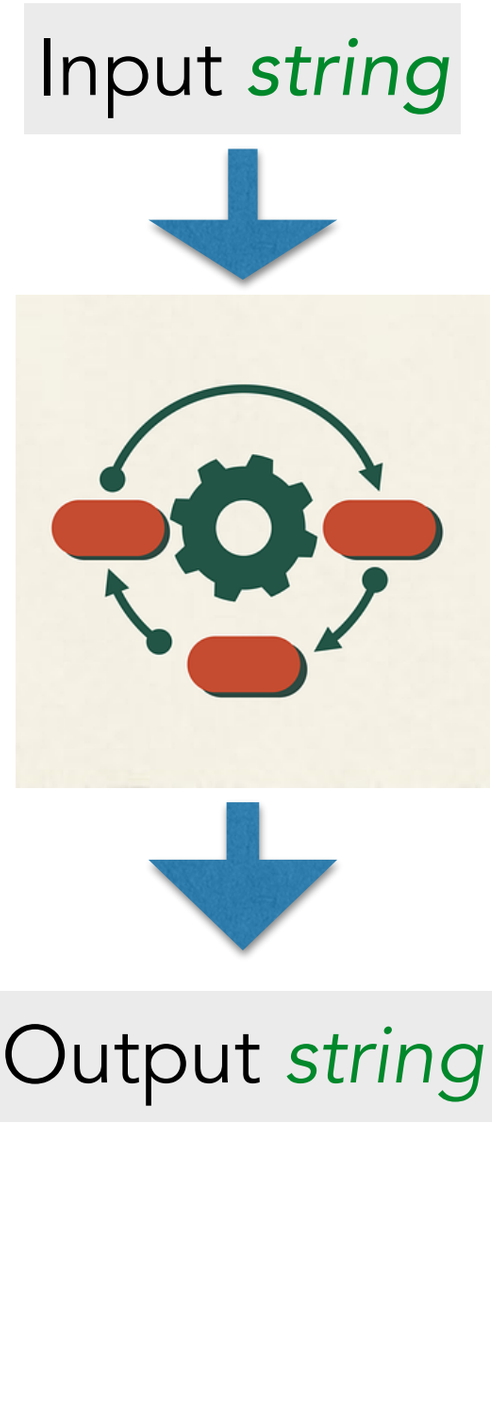
What Can We Encode?

- **Aside.** Can we encode everything?
  - Encodability = Countability (will come back to this)
    - How do we encode uncountable sets (e.g. $\mathbb{R}$)?
    - Approximation (with some precision)
- Often restricting to binary alphabet:
  - Can encode an alphabet with $|\Sigma| = k$ in binary using $\lceil \log_2 k \rceil$ bits
  - This extra factor is constant wrt size of input

Takeaway: In theory of computation, all input/output data is a string

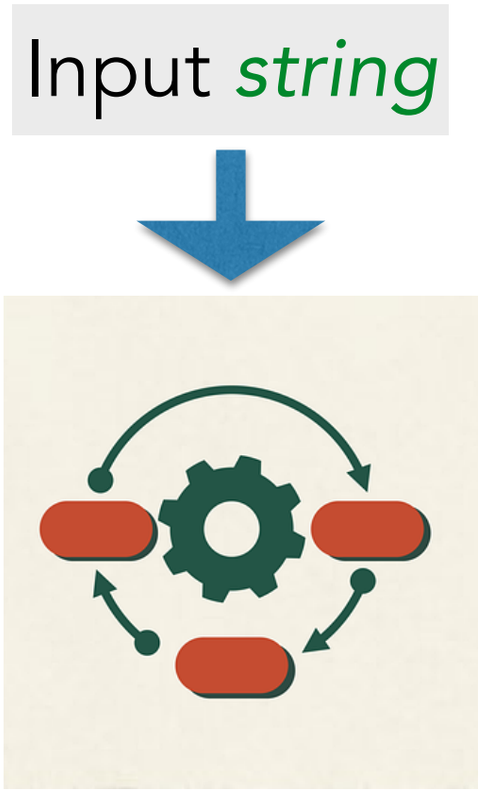
Back to Computation

- Input and output are strings over an alphabet
  - Input can be any finite length string
  - Output is a finite length string
- Now we need to define computation
  - For every input, there is an output
- What mathematical object captures this?
  - A function f from $\Sigma^* \to \Sigma^*$
- **Question.** Is computation just a function?
  - We need to know **how** to transform the input to the output



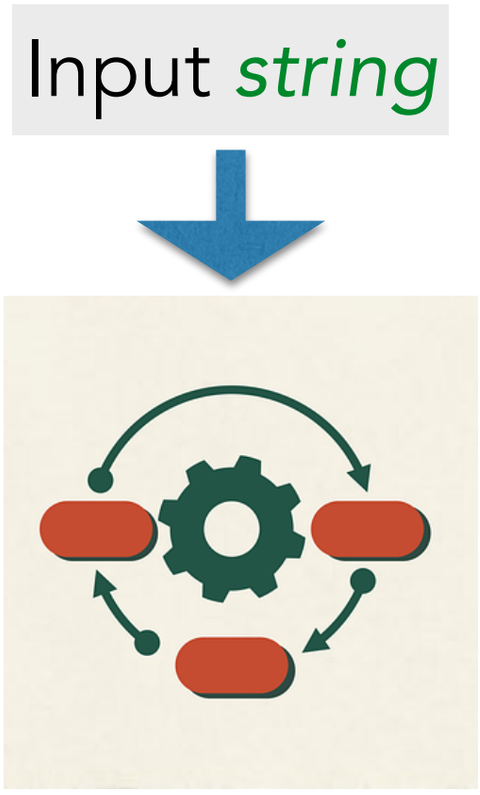
Function Problem

- Specification of a computational problem
- Function problem:
  - A function of the form $f : \Sigma^* \rightarrow \Sigma^*$
  - Specifies input, output pairs
- A computer/algorithm **solves** function problem f if its input/output behavior matches f
- Examples:
  - Reverse function
  - Sort function
  - isPrime



Decision Problem

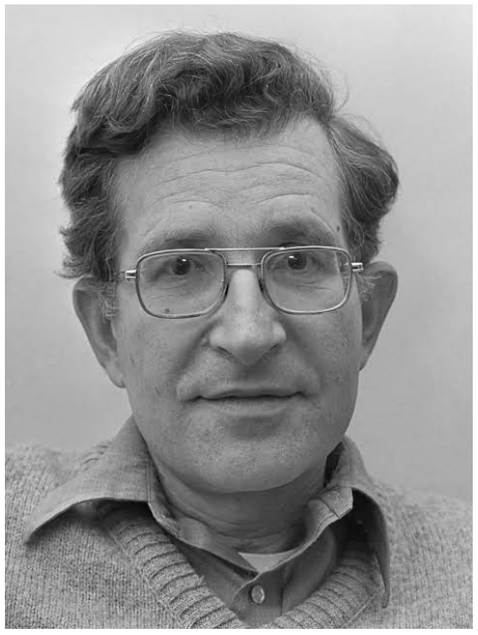
- A further convenient restriction on output:
  - Only consider decision problems
- A decision problem is a function $f : \Sigma^* \to \{0, 1\}$
- Examples:
  - Given a graph, is there a clique of size k?
  - Given a number, is it prime?
- This restriction simplifies the study of computation, without losing much



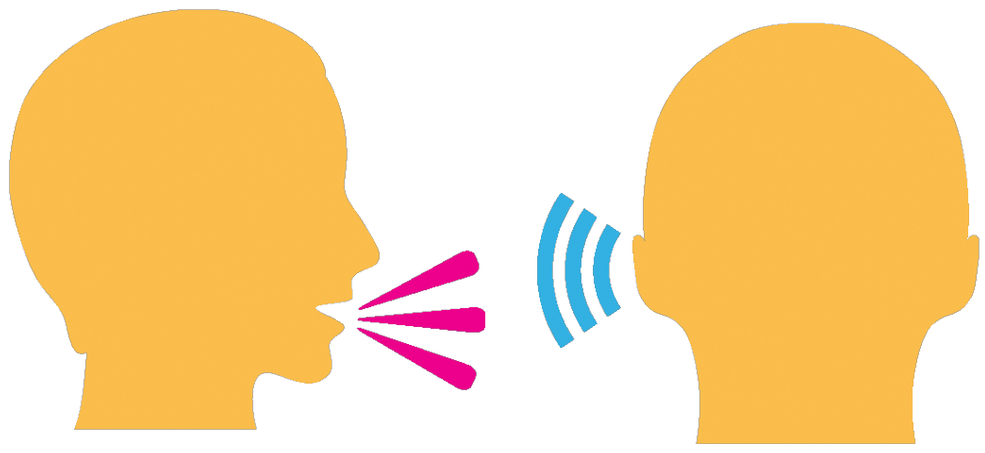
Languages

- Language: any **set** L of finite-length strings over an alphabet Σ
  - That is, any set $L \subseteq \Sigma^*$
  - Intuitively, a language is set of words over an alphabet
- Examples for $\Sigma = \{0, 1\}$
  - $L = \emptyset$
  - $L = \Sigma^*$
  - $L = \{1,01,001,0001,\ldots,\}$
- One-to-one mapping between decision problem f and language L:
  - $f(s) = 1$ if and only if $s \in L$

Influence of Chomsky



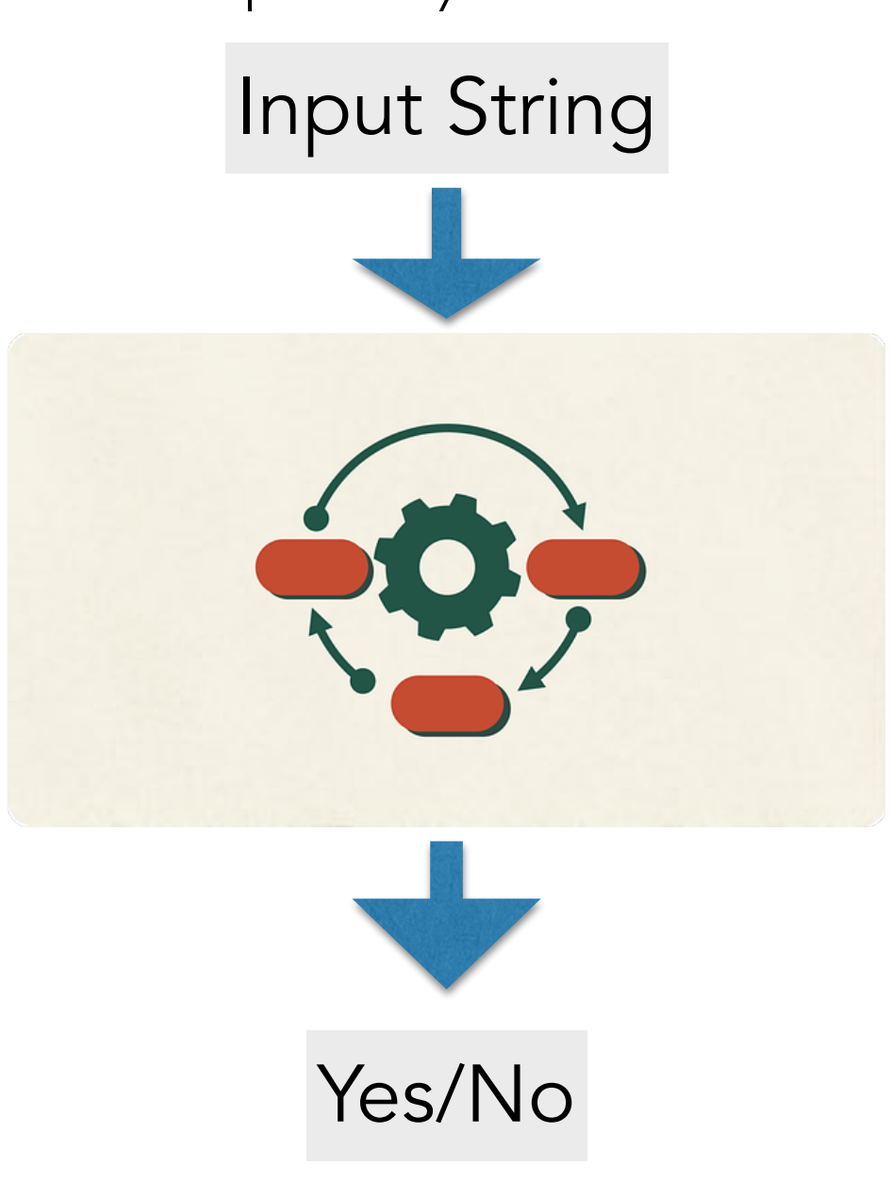
- We will study computation through the lens of languages
- Influence of linguist Chomsky on computation
- A **grammars** ***generates*** a language (akin to speaking)
  - Any string in the language can be generated using the rules of the grammar
- A **machine** ***recognizes*** a language (akin to listening)
  - If a given input string is in a language, the machine will "accept" (output true), otherwise "reject" (output false)



Takeways: Computation & Languages

- **Computation**: manipulation of input to reach desired output (yes/no)
- **Computational problem**: specification of the input/output pairs
- **Algorithm**: step by step approach to find out if output is yes or no

- **Language** is just a set of strings that represent the "yes" instances of a decision problem
    - e.g. all even length binary strings (is this binary string of even length?)

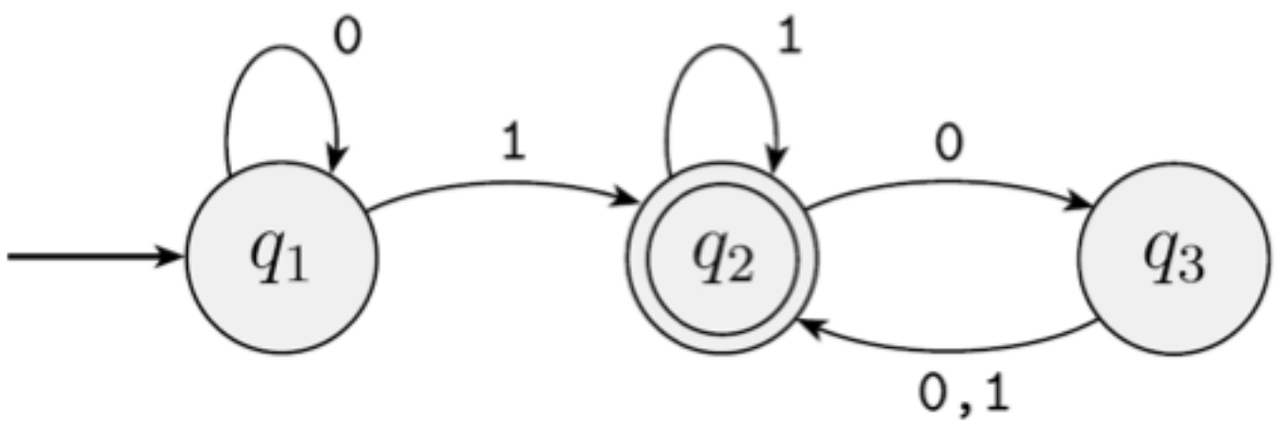


Finite State Automata

Simplest Form of Computation

Deterministic Finite Automata

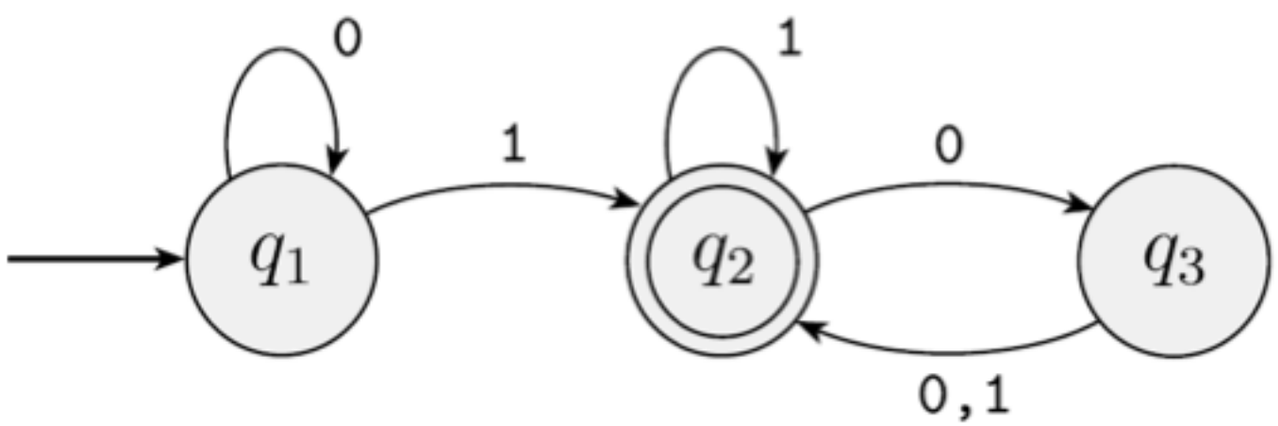
- A **machine** ***recognizes*** a language (akin to listening)
  - If a given input string is in a language, the machine will "accept" (output true), otherwise "reject" (output false)
- **Question.** What language is recognized by this machine?
  - Try some example strings



Definition of a Finite Automaton

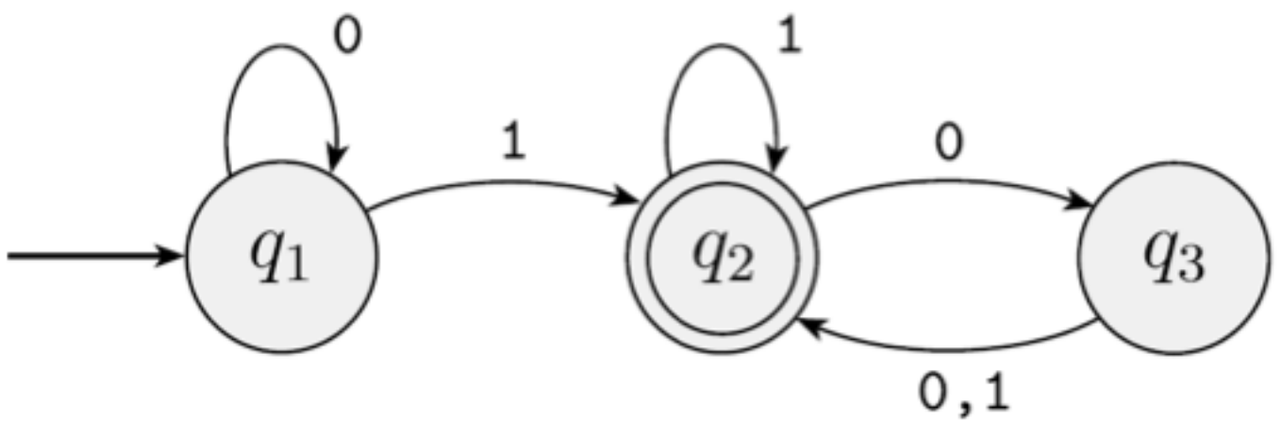
A finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

- Q is a finite set called the states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma \rightarrow Q$ is the transition function,
- $q_o \in Q$ is the start state and $F \subseteq Q$ is the set of accept states.

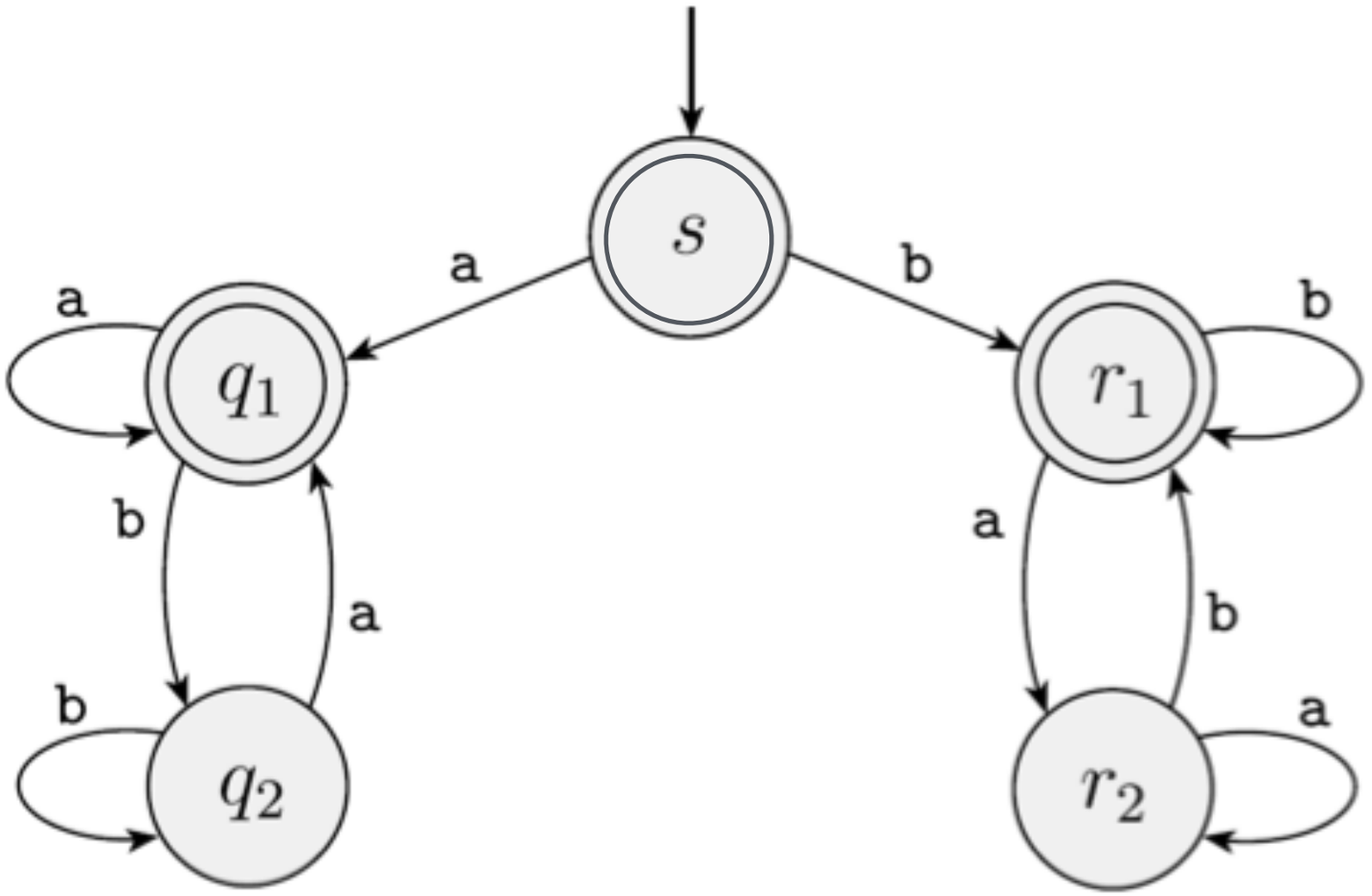


Language of a Machine

- The set of all strings accepted by a finite automaton M is called the language of machine M, and is written $L(M)$.
    - Say M **recognizes** language $L(M)$
- We will define M accepts w more formally
- Intuitive it is the strings on which it reaches an accepting state



What Language?



Automaton Computation

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and let $w = w_1 w_2 \cdots w_n$ be a string where each $w_i \in \Sigma$. Then M ***accepts*** w if there is a sequence of $r_0, r_1, \ldots, r_n$ in Q such that
  - $r_0 = q_0$
  - $\delta(r_i, w_{i+1}) = r_{i+1}$ for $i = 0,1,\ldots,n-1$ and
  - $r_n \in F$

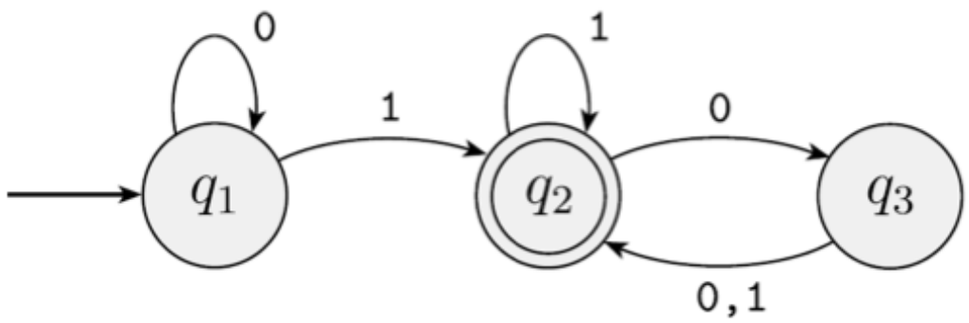
Extended Transition Function

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA
- Transition function $\delta : Q \times \Sigma \to Q$ is often extended to $\delta^* : Q \times \Sigma^* \to Q$ where $\delta^*(q, w)$ is defined as the state the DFA ends up in if it starts at q and reads the string w
- Alternate definition of M accepts $w \iff \delta^*(q_0, w) \in F$

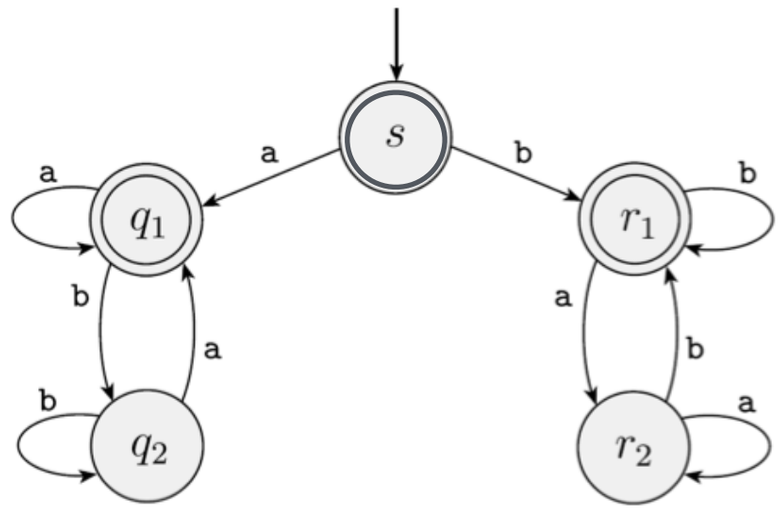
Language of a Machine

- The set of all strings accepted by a finite automaton M is called the language of machine M, and is written $L(M)$.
    - Say M **recognizes** language $L(M)$

M_1



M_2



$L(M_1) = \{w \mid w$ contains at least one 1 and an even number of zeroes follow the last $1\}$

$L(M_2) = \{w \mid w \in \{a, b\}^*$ that starts and ends with the same symbol$\}$

Regular Languages

- **Definition**. A language is called a **regular** language if some deterministic finite automaton recognizes it.

- Thus, to show a language L is regular, we must design a DFA M that recognizes it, that is, $L(M) = L$

    - M accepts $w \iff w \in L$

Practice with DFAs

- Show that the following languages are regular by drawing the state diagram of a DFA that recognizes it:
- $\{w \in \{0,1\}^* \mid w$ contains an even number of 1s $\}$
- $\{w \in \{a,b\}^* \mid w$ contains the substring $aba\ \}$

Class Exercises

- Show that the following are regular:
- $L_1 = \{w \mid w$ is a string of 0s and 1s containing an even number of 1s $\}$
- $L_2 = \{w \mid w$ is a string of as and bs containing the substring aba $\}$