Harvard CS 121 and CSCI E-121

Lecture 1: Introduction and Overview

Harry Lewis

September 3, 2013

Introduction to the Theory of Computation

Computer Science 121 and CSCI E-121

Objective:

Make a *theory* out of the idea of *computation*.

What is “computation”?

- Paper + Pencil Arithmetic

$$\begin{array}{rr} & 121 \\ + & 99 \\ \hline & 220 \end{array}$$

- Abacus
- Calculator w/moving parts (Babbage wheels, Mark I)
- Ruler & compass geometry constructions
- Digital Computers

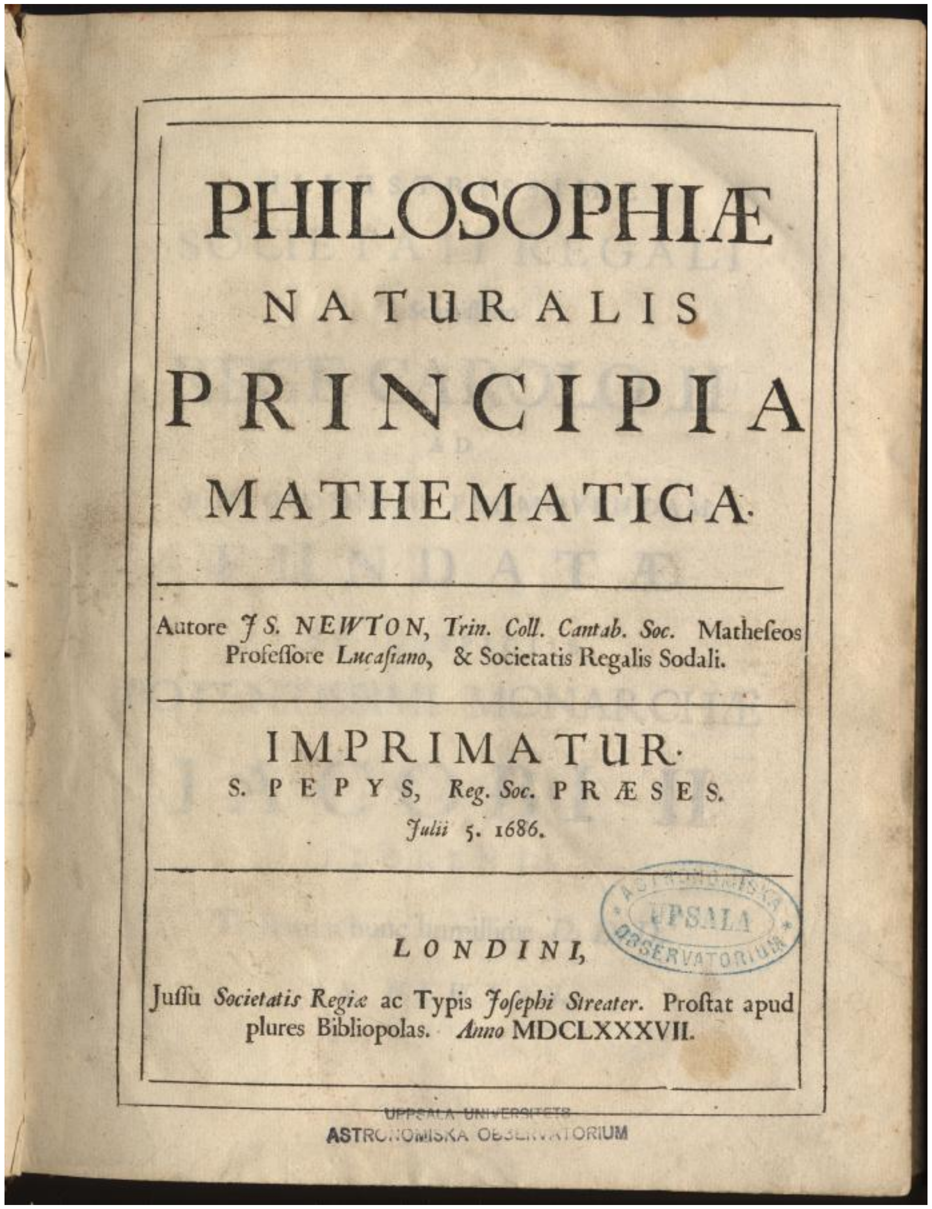
Further computing devices

- Programs in C, Java.
- The Internet and other distributed systems.
- Cells/DNA?
- The human brain?
- Quantum computers?

For us computation will be

Processing information by unlimited application
of a finite set of operations or rules

What do we want in a “theory”?



PHILOSOPHIÆ

NATURALIS

PRINCIPIA

MATHEMATICA.

Autore *J S.* NEWTON, *Trin. Coll. Cantab. Soc.* Matheſeos Profeſſore *Lucaſiano*, & Societatis Regalis Sodali.

IMPRIMATUR.

S. PEPYS, *Reg. Soc.* PRÆSES.

Julii 5. 1686.

LONDINI,

Juſſu *Societatis Regiæ* ac Typis *Joſephi Streater*. Proſtat apud plures Bibliopolas. *Anno* MDCLXXXVII.

What we would like to get past

What we would like to get past

- “This must be hard because I can’t figure out how do it”

What we would like to get past

- “This must be hard because I can’t figure out how do it”
- “This must be hard because I can’t figure out how do it and neither can anybody else, including a lot of really smart people”

What we would like to get past

- “This must be hard because I can’t figure out how do it”
- “This must be hard because I can’t figure out how do it and neither can anybody else, including a lot of really smart people”
- “This method seems to get the right answer on every case I’ve tried”

What we would like to get past

- “This must be hard because I can’t figure out how do it”
- “This must be hard because I can’t figure out how do it and neither can anybody else, including a lot of really smart people”
- “This method seems to get the right answer on every case I’ve tried”
- “It’s never crashed while I was testing it so let’s ship it”

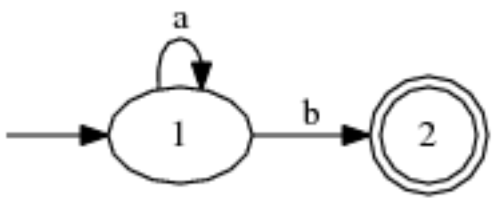
What do we want in a "theory"?

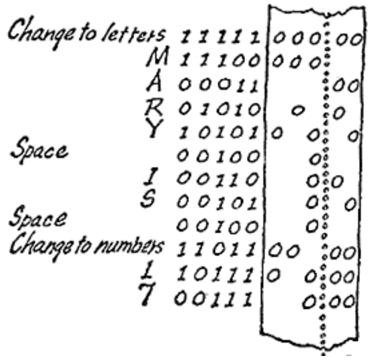
- **Precision**
  - Mathematical, formal.
  - Can *prove* theorems about computation, both positive (what can be computed) and negative (what cannot be computed).
- **Generality**
  - Technology-independent, applies to the future as well as the present
  - Abstraction: ignores inessential details (though it may pay to restore them later)

“Inessential” details

- Simple models are easier to reason about
- Once you have some model, it may pay to restore detail previously ignored
- Example: The UPS delivery truck
- Example: Boarding an airplane

An automaton is an abstraction





A machine reading symbols from a tape

A system receiving discrete impulses over time

A chip

Representing “Information”

- Alphabet

  Ex: $a, b, c, \ldots, z$.

- Strings: finite concatenation of alphabet symbols, order matters

  Ex: $qaz, abbab$

  $\varepsilon =$ empty string (length 0; sometimes e)

- Inputs (& outputs) of computations are strings.

  $\Rightarrow$ we focus on *discrete* computations

Computational Problems (i.e. Tasks)

A single question that has infinitely many different instances

- *PARITY*: given a string x, does it have an even number of a's?
- *MAJORITY*: given a string x, does it have more a's than b's?

Problems are defined extensionally: a problem is

- the set of all instances of the question to which the answer is positive
- the set of all $\langle$question, answer$\rangle$ pairs

Examples of computational problems on numbers

- *PRIMALITY*: given a number x, is x prime?
- *ADDITION*: given two numbers x, y, compute $x + y$.
- A numerical ”problem” is a set of binary or decimal numerals

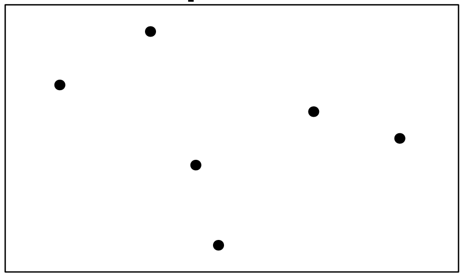
Examples of computational problems about computer programs

- C *SYNTAX:* given a string of ASCII symbols, does it follow the syntax rules for the C programming language?
- *HALTING PROBLEM*: given a computer program (say in C), can it ever get stuck in an infinite loop?

Computational problems from pure and applied mathematics

- *DIOPHANTINE EQUATIONS*: Given a polynomial equation (e.g. $x^2 + 3xyz - 44z^3 = 0$), does it have an integer solution?

- *TRAVELLING SALESMAN PROBLEM*: Given a set of ‘cities’ in the plane, what is the fastest way to visit them all?



- *GRAPH 2-COLORING (3-COLORING)*: Given a set of people, can they be partitioned into 2 groups so that every pair of people in each group gets along? (3 groups?)

More examples of computational problems

- *REGISTER ALLOCATION*
- *MULTIPROCESSOR SCHEDULING*
- *PROTEIN FOLDING*
- *DECODING ERROR-CORRECTING CODES*
- *NEURON TRAINING*
- *AUCTION WINNER*
- *MIN-ENERGY CONFIGURATION OF A GAS*
- ...

The (Mathematical) Idea of a Language

- A **language:** any set of strings.
- “Solving a yes/no computational problem”
  $\Leftrightarrow$ “Deciding if a string is in a given language”

Examples of Languages

- All words in the *American Heritage Dictionary*

  $\{a, aah, aardvark, \ldots, zyzzva\}$

  Mathematically simple, because it's <u>finite</u>!

- All strings with an even number of a's.

  $\{\varepsilon, b, bb, aa, baa, aba, baa, \ldots\}$

  <u>Note:</u> "ε" denotes the string of length 0 – the empty string

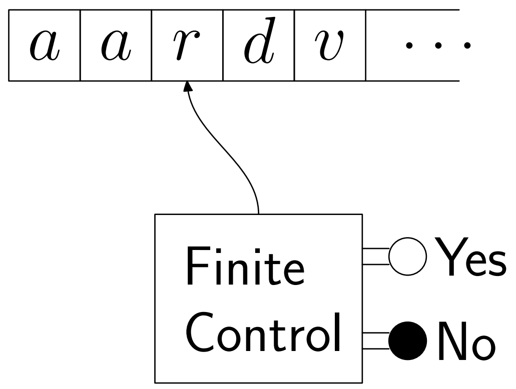
  Infinite – but simple membership <u>rule</u>

- All syntactically correct C programs

  (counting space and newline as characters)

Computational Models

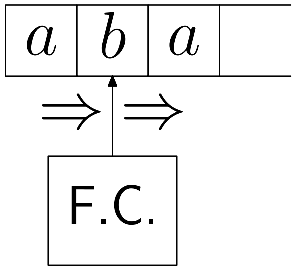
What is a computer? First try: a mathematical automaton.



We don't care how the control is implemented – only that it have a <u>finite</u> number of states and change states based on <u>fixed rules</u>

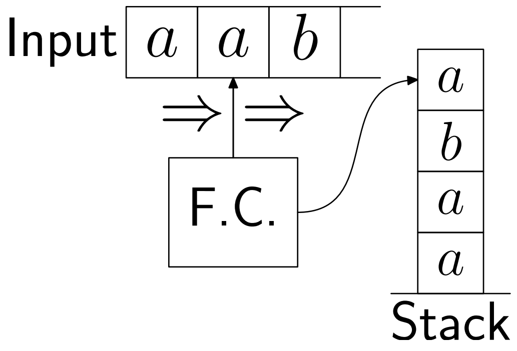
Kinds of Automata

Finite Automata



- Head scans left to right
- Check simple patterns
- Finite Table Lookup
- Can't count without limit

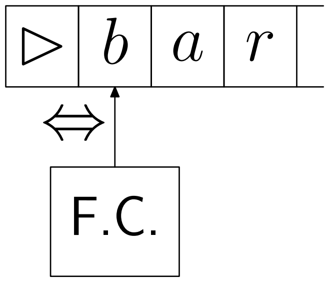
Pushdown Automata



- Use stack to count, balance parentheses
- Check many syntax rules

A model for general-purpose computers

Turing Machines



- Control is still finite
- Head moves left and right, reads, and writes

Q1: What computational problems can be solved by these automata?

- Finite Automata recognize the **regular languages**.

  A regular language is one that can be described by a *regular expression*, e.g.

a^*	generates $\{\varepsilon, a, aa, aaa, \ldots\}$
$* =$ "any number of"	
$(ab)^*$	generates $\{\varepsilon, ab, abab, ababab, \ldots\}$
$(a^*ab)^*a^*$	generates $\{???\}$
$(a \cup ab)^*$	generates $\{???\}$

Q1: What computational problems can be solved by these automata?

Pushdown Automata: the **context-free languages**. A PDA can determine whether or not strings are generated by any fixed *context-free grammar*, e.g.

$$\left\{\begin{array}{l} S \to aSb \\ S \to \varepsilon \end{array}\right\} \text{ generates } \{\varepsilon, ab, aabb, aaabbb, \ldots\}$$

Note: this is not the same as a^*b^*!

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$$

CFGs as models for natural languages

$$\left\{ \begin{array}{rcl} \langle \text{sentence} \rangle & \rightarrow & \langle \text{noun} - \text{phrase} \rangle\_\langle \text{verb} \rangle \\ \langle \text{noun} - \text{phrase} \rangle & \rightarrow & \langle \text{noun} \rangle \mid \langle \text{adjective} \rangle\_\langle \text{noun} - \text{phrase} \rangle \\ \langle \text{noun} \rangle & \rightarrow & \text{cat} \mid \text{dog} \mid \text{mouse} \\ \langle \text{adjective} \rangle & \rightarrow & \text{black} \mid \text{hungry} \\ \langle \text{verb} \rangle & \rightarrow & \text{jumps} \mid \text{barks} \end{array} \right\}$$

generates $\{\text{black\_dog\_jumps}, \text{hungry\_black\_cat\_barks}, \ldots\}$

More powerful models

Turing machines: the **computable languages**

- Captures our intuitive notion of “computable” **(Church–Turing Thesis)**.
- TMs equivalent in expressiveness to C programs, LISP programs, Pentium CPU, (hypothetical) quantum computers, ...
- Concept of computability is independent of technology!

Church’s Thesis (Church-Turing Thesis)

Intuitive notion of “computable”

$$\equiv$$

Formal notion of “computable by a Turing Machine”

Are there non-computable languages?

Yes – in fact “almost all” languages are not computable

What are some examples?

[Problems to avoid!]

Q2: Are there computational problems that <u>cannot</u> be solved by these automata?

- Yes — in fact “almost all” problems are not computable.
- But what are some examples? [Problems to avoid!]
- A non-regular problem?
- A non-context-free problem?
- Non-computable problems?

Classifying languages

	FA/ regular?	PDA/ context free?	TM/ computable?
PARITY			
MAJORITY			
PRIMALITY			
C *SYNTAX*			
HALTING			
TSP			
2-COLORING			
DIOPHANTINE EQ.			

Q3: Are there computable problems that cannot be solved efficiently?

- A problem need not be uncomputable to be practically unsolvable (It may just take too long!)

- Theory of relative difficulty of problems

$\longrightarrow$ Based on resources required:

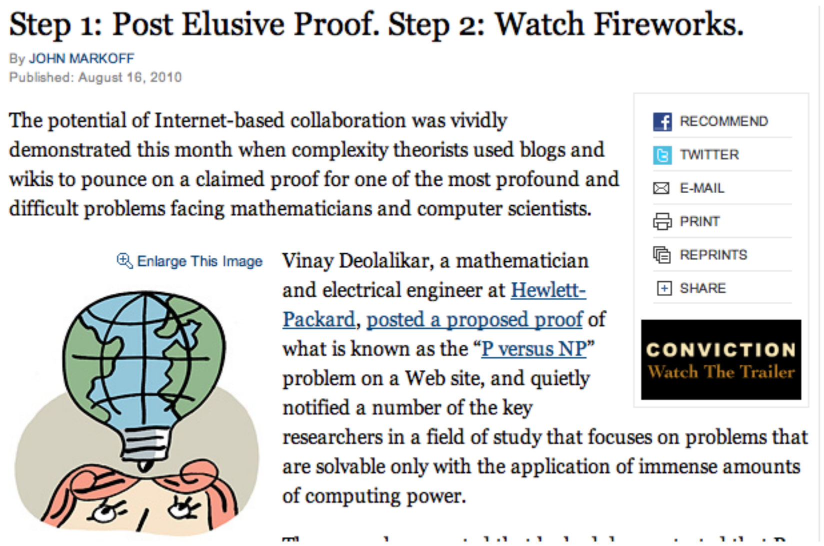
  - Time
  - Memory
  - …

The NP-Complete Problems

TRAVELLING SALESMAN PROBLEM, GRAPH 3-COLORING, MULTIPROCESSOR SCHEDULING, PROTEIN FOLDING, ...

Do they have efficient algorithms? Either all do or none do!

This is the famous (and still open) **P vs. NP Question**.



Step 1: Post Elusive Proof. Step 2: Watch Fireworks.

By JOHN MARKOFF
Published: August 16, 2010

The potential of Internet-based collaboration was vividly demonstrated this month when complexity theorists used blogs and wikis to pounce on a claimed proof for one of the most profound and difficult problems facing mathematicians and computer scientists.

Enlarge This Image

Vinay Deolalikar, a mathematician and electrical engineer at Hewlett-Packard, posted a proposed proof of what is known as the "P versus NP" problem on a Web site, and quietly notified a number of the key researchers in a field of study that focuses on problems that are solvable only with the application of immense amounts of computing power.

More NP-Complete Problems

- Integer Linear Program

  Is there a solution over the positive integers to a system like this?

$$\begin{array}{rcrcrcl} x_1 & - & 4x_2 & + & x_3 & = & 0 \\ x_1 & + & x_2 & + & x_3 & \leq & 0 \\ x_1 & & & + & 7x_3 & \geq & 0 \end{array}$$

- Boolean Satisfiability

  Are there true/false values for the variables to make this formula true?

$$(x \vee y \vee z) \wedge (\neg x \vee \neg y) \wedge (\neg z \vee y)$$

[$\vee =$ “or” $\wedge =$ “and” $\neg =$ “not”]

For computer scientists

- Technology-independent foundations of CS.
- How to reason precisely about computation.
- Topics applicable to other parts of CS.

Circuit Design	Finite Automata
Distributed Computing	Finite Automata
Parsing + Compiling	Context-free Languages
Natural Language Processing	Context-free Languages
Programming Langs	Regular Expressions, Uncomputability
Artificial Intelligence	Finite Automata, Complexity Theory
Algorithm Design	Complexity Theory
Cryptography	Complexity Theory

For mathematicians

A “computational perspective” on mathematics.
Ex: which is a ‘better’ formula for the n’th Fibonacci number (1,1,2,3,5,8,13,21...)?

1. $F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$.

2. $F_n =$ the number of strings over alphabet $\{a, b\}$ of length $n-2$ with no two consecutive b’s.

Connection between computation and mathematical proofs

- Uncomputability $\leftrightarrow$ Gödel's Incompleteness Theorem.
- P vs. NP $\leftrightarrow$ "are mathematical proofs as easy to find as they are to verify?"
- Can mathematics be automatized?

Important and famous problems for Mathematics

Rich interplay between the Theory of Computation and various areas of mathematics (logic, combinatorics, algebra, number theory, probability, functional analysis, algebraic geometry, topology, ...). Many research opportunities.

For others

- How to recognize and interpret computational intractability in case it appears in your domain, e.g. *PROTEIN FOLDING*, *NEURON TRAINING*, *AUCTION WINNER-DETERMINATION*, *MIN-ENERGY CONFIGURATION OF A GAS*

- How to model computation, e.g. as it may occur in Cells/DNA, the brain, economic systems, physical systems, social networks, ...

Philosophically interesting questions

- Are there well-defined problems that cannot be solved automatically?
- Can we *always* search for a solution to a puzzle more quickly than trying all possibilities?
- Can we formalize the idea that one problem is “harder” than another?

Prerequisites

“Experience in formal mathematics at the level of CS 20.”

- Comfort reading and writing mathematical proofs.
- Sets (e.g. cardinality, powersets, cartesian products)
- Functions (e.g. one-to-one, onto, bijections)
- Relations (e.g. symmetric, transitive, reflexive)
- Graphs (e.g. directed vs. undirected)
- Proofs by induction
- Propositional logic (e.g. truth tables, De Morgan’s Laws, CNF)
- Growth rates (O notation)

You do *not* need to know/remember *all* of these concepts.

Strengthening your Mathematical Preparation

- Sipser, Chapter 0.
- Problem Set 0. Graded, but will not count. Strongly recommended!
- Open sections this week.
- Office Hours (mine & the TFs).
- Course materials for CS 20.
- Books by Solow and Rosen (see syllabus).

If you have any doubts about your preparation, come talk to me!

Other Organizational Remarks

- No handouts, everything on the Web, including these slides
- Read collaboration policy carefully and respect it
- Lecture videos available 24 hours after lecture
- Sections start meeting this week, but no sectioning until next weekend
- One section will be a ”math section”
- Use Piazza for questions, course staff monitors it
- Use of LaTeX for problem sets is mandatory