What computers just *cannot* do.

COS 116, Spring 2010

Adam Finkelstein

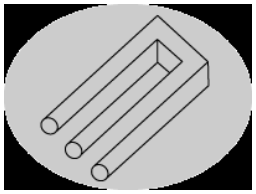
“What computers can’t do.”

“Prof, what’s with all the negative thinking?!?”

- An obvious motivation:
  Understand the limits of technology

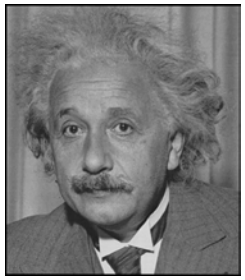
Power of negative thinking…

Often, impossibility result → deep insight



Examples

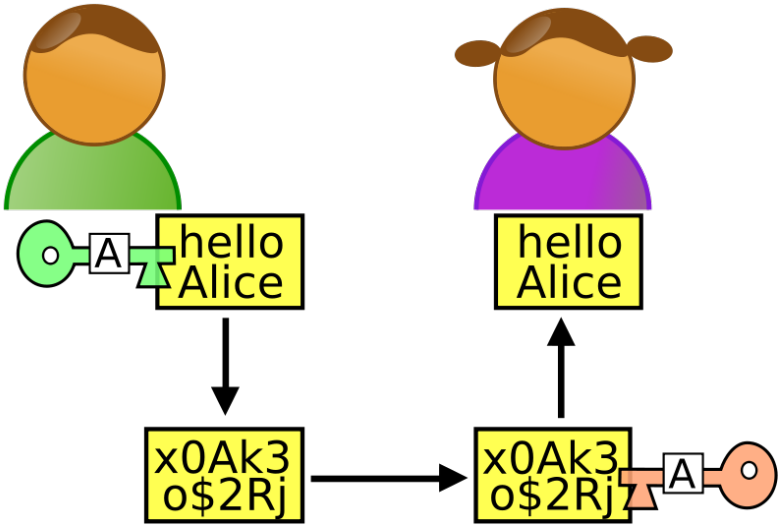
- Impossibility of trisecting angle with ruler and compass (Galois) → Group Theory and much of modern math



- Nothing travels faster than light → Relativity and modern physics

Uses of negative thinking in computer science……

CAPTCHA (CMU Group)
Computer generated test that Computers (at least with current algorithmic knowledge) seem unable to solve pass.



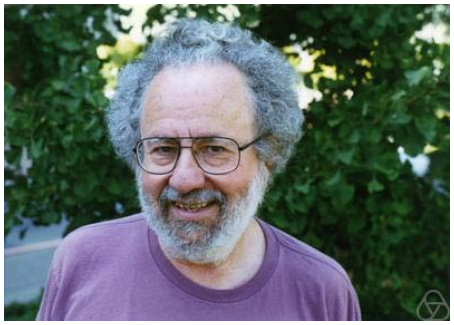
Cryptography

"Tasks that computers cannot do fast enough"; topic of future lecture

Today: Tasks that are going to be unsolvable by a computer (no matter how long it runs)

- … the story has many sidestories, characters, and thoughtprovoking consequences

Reading (first 10 pages by Thurs):
What is computation? By Martin Davis



In Mathematics…..

"Can mathematicians be replaced by machines?"

[Hilbert, 1900]

Math is axiomatic

Axioms – Set of statements

Derivation rules – finite set of rules for deriving new statements from axioms

Theorems – Statements that *can* be derived from axioms in a finite number of steps

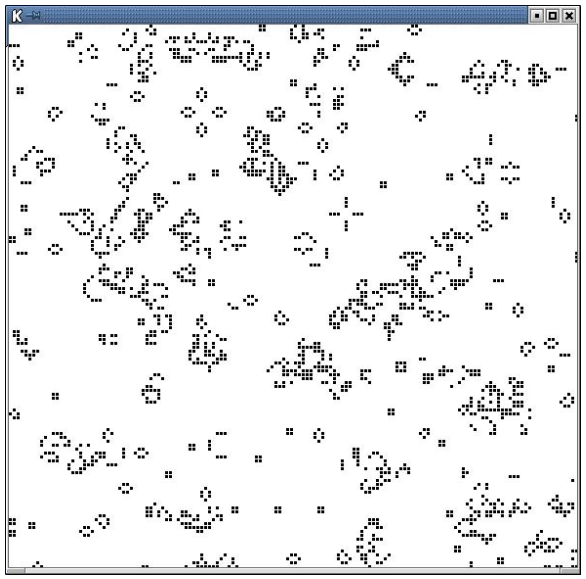
Mathematician – Person who tries to determine whether or not a statement is a theorem.

Ans (Goedel,Turing, etc.): Computers cannot discover all math truths; in fact no axiomatic system can capture all math truths

Understanding complex systems (or even simple systems)….

Can a simple set of mathematical equations “solve” problems like:

“Given starting configuration for the game of life, determine whether or not cell (100,100) is ever occupied by a critter.”



John Conway

Ans: Problem is unsolvable by computers. So no easy “theory” to explain the outcomes of game of life.

Automated software checking?

e.g. Windows Vista:
50-million line program

Can computers check whether or not it will ever crash?

Ans: No computer program can solve the task of
checking if a given piece of code will ever crash (or “hang up”)

How can one prove result of prev. slide?

A. Turing

- Fix a simple computational model, Turing-Post pseudocode
- Argue that this simple model can simulate all *realizable* computational models (anything a computer can do, a T.P. program can do too)
- Show that a T.P. program cannot solve the computational task

OK, but how do you prove that T-P pseudocode cannot solve the computational task?

Ans. Do the reading; discussion next time.

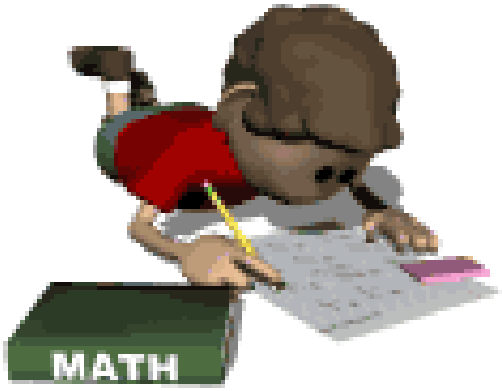
Discussion Time

(reconstructing Turing's thought process)

What is a computation?

What is a computation?

A formalization of an age-old notion



Basic Elements

- **Scratch Pad**
- **Step-by-step description of what to do (“program”); should be finite!**
- **At each step:**
  - **Can only scan a fixed number of symbols**
  - **Can only write a fixed number of symbols**

Turing's model

- 1 dimensional unlimited scratchpad ("infinite tape")
- Only symbols are 0 or 1 (tape has a finite number of 1s)
- Can only scan/write one symbol per step
- Program looks like →

1. PRINT 0
2. GO LEFT
3. GO TO STEP 1 IF 1 SCANNED
4. PRINT 1
5. GO RIGHT
6. GO TO STEP 5 IF 1 SCANNED
7. PRINT 1
8. GO RIGHT
9. GO TO STEP 1 IF 1 SCANNED
10. STOP

The Doubling Program

Example:
What does this program do?

1. PRINT 0
2. GO RIGHT
3. GO TO STEP 1 if 1 SCANNED
4. GO TO STEP 2 if 0 SCANNED

Discussion Time

Can this computational model do every computation that pseudocode can?

How do we implement arithmetic instructions, arrays, loops?

Surprising facts about this simple model

- It can do everything that pseudocode can do

Hence it can “simulate” any other physical system, and in particular simulate any other physically realizable “computer.”

[CHURCH-TURING THESIS”]

THIS MODEL CAPTURES THE NOTION OF “COMPUTATION” ----TURING

Representing programs in binary

Recall: Numbers and letters can be written in binary.

A program can also be represented by a string of bits!

“Code” for a program

= Binary Representation

Many conventions possible (e.g., ASCII)

Davis’s convention:

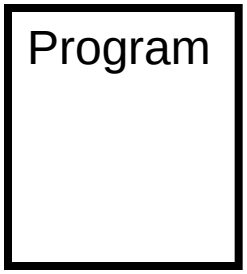
Code	Instruction
000	PRINT 0
001	PRINT 1
010	GO LEFT
011	GO RIGHT
$1010\underbrace{\ldots}_{i}01$	GO TO STEP i IF 0 IS SCANNED
$1101\underbrace{\ldots}_{i}10$	GO TO STEP i IF 1 IS SCANNED
100	STOP

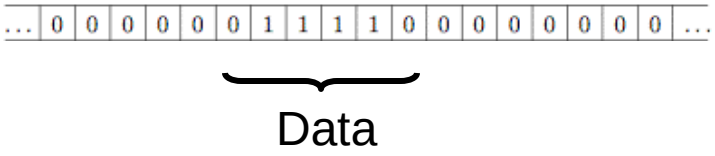
P ⟷ Code (P)

Programs and Data

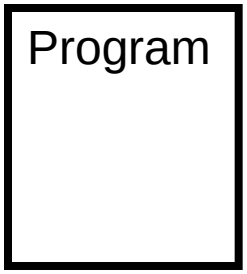
A False Dichotomy!

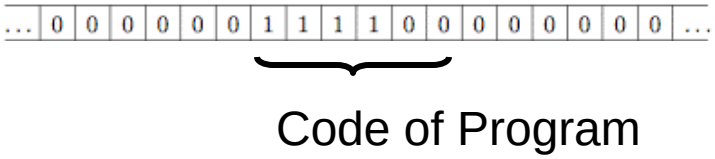
Usual viewpoint -



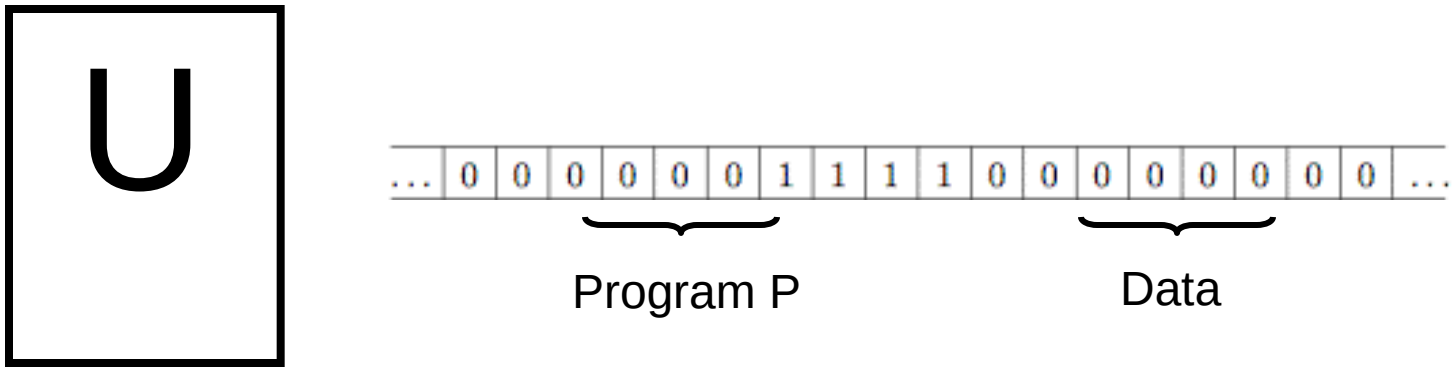


But can have -





Universal Program U

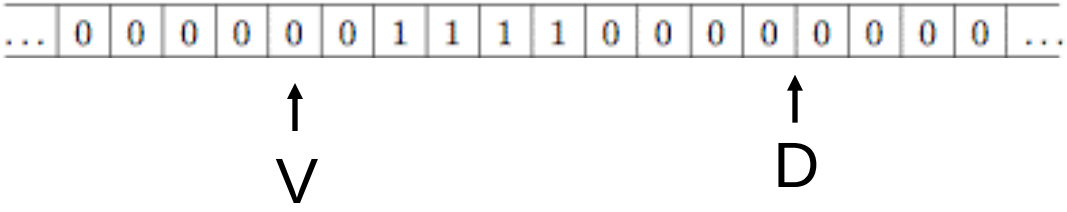


- U “simulates” what P would do on that data

(Sometimes also known as “interpreter”; basis of modern technologies)

Automated Bug Checking Revisited

Halting Problem



Let P = program such that code(P) = V.
Does P halt on data D?

IDEAS???

Trivial Idea: Simulate P using universal program U.
If P halts, will eventually detect.

Problem: But if P never halts, neither does the simulation.

Next Time: Halting Problem is unsolvable by another program

Read this proof in the Davis article, and try to understand.

Ponder the meaning of “Proof by contradiction.”
How convincing is such a proof?

“When something’s not right, it’s wrong…” -Bob Dylan

Homework for next Thurs posted this afternoon.
For discussion next time: Write Turing-Post program that prints the bit sequence 101 infinitely often.
Also write the binary code of this program.