Theoretical Computation

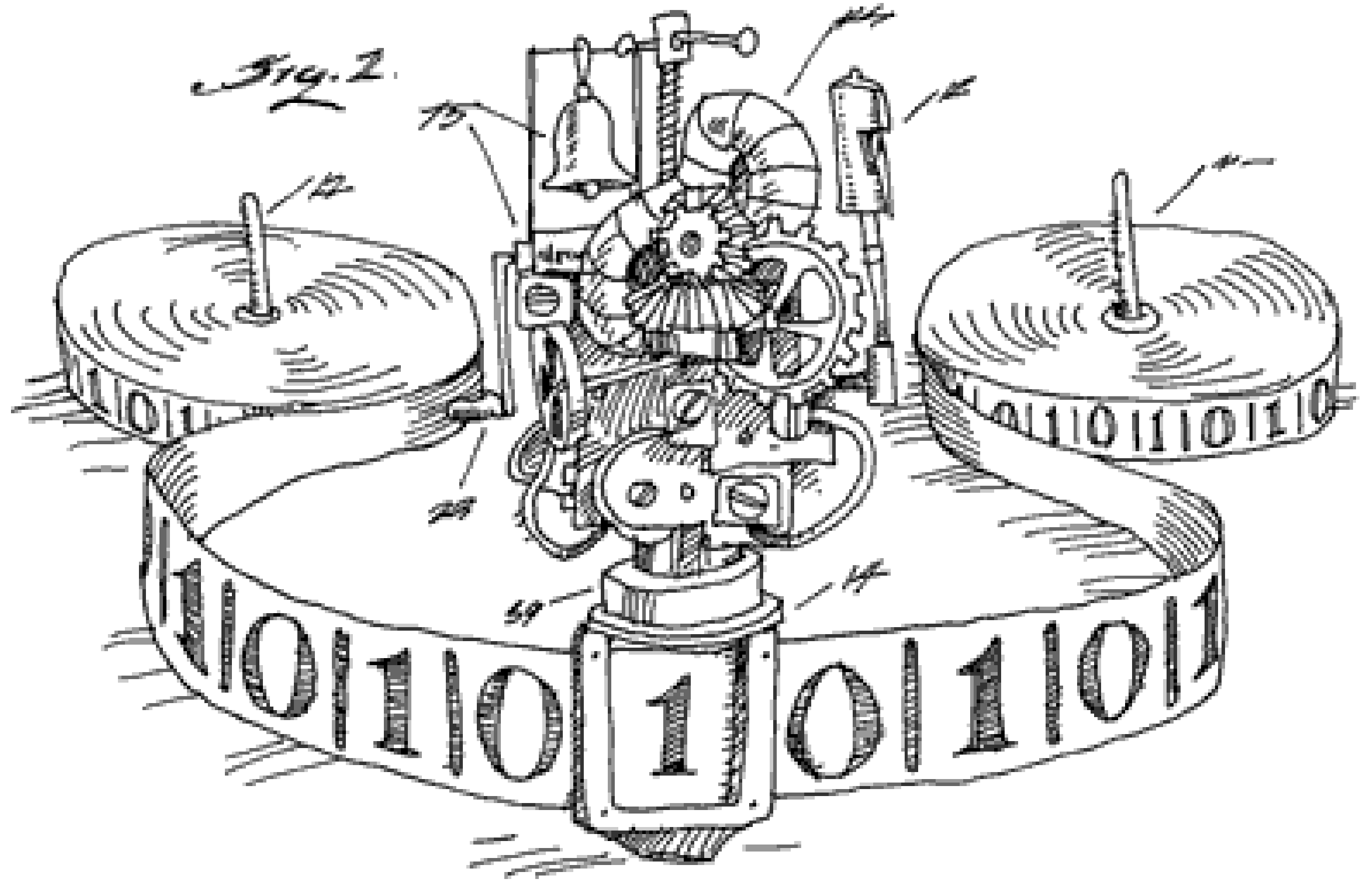

Image: http://jonfwilkins.blogspot.com.au

Motivation: Why do we study theoretical computation?

- Compilers
- Language Theory
- Hilbert Problems
  - Problem 10: Find an algorithm to determine if a polynomial with integer coefficients has an integer solution

Turing Machine

- Set of States
- Input Alphabet
- Output Alphabet
- Transition Function
- Infinite Tape with Read/Write Head

Turing Machine Example

- Symmetrical string of 1's and 0's
- See whiteboard

Church Turing Thesis

- Any algorithm can be represented as a turing machine!

Terminology

- Decision problem = Language
- Accept / Reject
- Halt / Loop
- Decide
- Recognize
- Corecognize

Problem Syntax

- PROBLEM-NAME = {input | accept condition}
- SYM =

  {str | str is a symmetrical string of 1s and 0s}
- The problem (or language) “SYM” takes strings as input and accepts exactly when the input string is a symmetrical string of 0s and 1s

Self Referentiality need not be feared

Image: xkcd.com

Self Referentiality in real life

- count-lines
- Compilers

Proof by contradiction

- To prove some statement
- Assume it's false
- Show that this results in a contradiction
- The initial assumption must be wrong
- Therefore the statement is true

The Halting Problem

- HALT = {<P>, x | P(x) halts}
- Given some program description and some input for that program, does the program ever terminate on the input?
- How can you write a program to decide HALT?

You can't!

- Proof: see whiteboard

More HALT

- It is recognizable
- It is NOT co-recognizable
- To make it easy to prove this, establish that a problem is decidable if and only if it is both recognizable and co-recognizable
- Proof on whiteboard

Two important results

- HALT is not decidable
- HALT is not co-recognizable
- That is, (NOT-HALT) is not recognizable.
- NOT-HALT = {<P>, x | P(x) loops forever}

Reductions

- Proof technique for proving non-decidability or non-recognizability
- Form of proof by contradiction
- Assume the problem is decidable (recognizable), then construct a turing machine that decides (recognizes) HALT (NOT-HALT), creating a contradiction.

Reduction Example

- A = {<P>, <Q>, x | P(x) runs for at least as many steps as Q(x)}
- If a program doesn't halt, it is said to run for infininty steps, so if both P(x) and Q(x) loop forever, (<P>, <Q>, x) is accepted.
- Is this decidable/recognizable?

Not Recognizable

- Proof: see whiteboard

Dove-Tailing

- Technique for traversing multiple (potentially) infinite lists
- We can't just traverse one after the other
- Simple example: write a program that will eventually print out any given fraction with integer numerator and denominator

Printing Fractions

	1	2	3	4
1	1/1	1/2	1/3	1/4
2	2/1	2/2	2/3	2/4
3	3/1	3/2	3/3	3/4
4	4/1	4/2	4/3	4/4

- “Print one row, then the next row, etc” isn't going to work
- Better approach:
  1/1, 2/1, 1/2, 3/1, 2/2, 1/3, 4/1, ...

Why is this relevant?

- We can execute multiple programs inside other programs without fear that one of them doesn't halt.
- For example:
  A = {<P> | P(i) halts for at least one 0 <= i <= 10}
- Is this problem recognizable/decidable?

Recognizable

- Recognizer uses dovetailing: see whiteboard
- Not decidable – if it was decidable we could use it to decide HALT

Turing Machine Excercises

- Even binary nuber
- Odd number of 0s before the first 1
- Equal number of 0s and 1s
- The same string repeated twice
- Alternating 0s and 1s
- All 0s, then all 1s

More Turing Machine Exercises

- Two binary numbers seperated by a “#”, where the second number is 1 greater than the first