“Every mathematical problem can be solved. We are convinced of that. After all, one of the things that attracts us most when we apply ourselves to a mathematical problem is precisely that within us we always hear the call: here is the problem, search for the solution, you can find it by pure thought, for in mathematics there is no ignorabimus.”

A Puzzle: Post’s Correspondence Problem

Given a set of cards:
- N card types (can use as many of each type as needed).
- Each card has a top string and bottom string.

Example 1:

<table>
<thead>
<tr>
<th>BAB</th>
<th>A</th>
<th>AB</th>
<th>BA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ABA</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Puzzle:
- Is it possible to arrange cards so that top and bottom strings are the same?

A Puzzle: Post’s Correspondence Problem

Example 2:

<table>
<thead>
<tr>
<th>A</th>
<th>ABA</th>
<th>B</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAB</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Puzzle:
- Is it possible to arrange cards so that top and bottom strings are the same?

PCP Puzzle Contest

Contest:
- Additional restriction: string must start with ‘S’.
- Be the first to solve this puzzle!
  - extra credit for first correct solution
- Check solution from Lectures web page.

Hopeless challenge for the bored:
- Write a program that reads a set of Post cards, and determines whether or not there is a solution.
Background

Abstract models of computation help us learn:
- Nature of machines needed to solve problems.
- Relationship between problems and machines.
- Intrinsic difficulty of problems.

Deep questions.
- Are there problems that no machine can solve?
- Are there limits on the power of machines that we can imagine?

Pioneering work in the 1930’s.  (Princeton == center of universe)
- Gödel, Turing, Church, von Neumann.  (inspiration from Hilbert)
- Automata, languages, computability, complexity, logic, rigorous definition of “algorithm.”

A Notational Simplification

Decision problems.
- Rigorously express computational problems as yes/no queries.
- Captures essence of computation.
- Cleaner to understand and study.

Example.  Is 977 a prime number?

This lecture:
- What is an ”algorithm”?
- Is it possible, in principle, to write a program to solve any problem?

Church-Turing Thesis

Church-Turing Thesis (1936).
Q. Which decision problems can a Turing machine solve?
A. Any decision problem that any real computer can solve.

"Thesis" and not a mathematical theorem.
- Can’t be proved because we can’t precisely define solving a problem (computability).

Implications:
- Provides rigorous definition for ALGORITHM.
  - describing an algorithm = building a TM
- Universality among computational models.
  - if a problem can be solved by some TM, then it can be solved on EVERY general-purpose computer
  - if a problem can’t be solved by any TM, then it can’t be solved on ANY physical computer

Evidence Supporting Church-Turing Thesis

Imagine TM with more power.
- Composition of TM’s, multiple heads, more tapes, 2D tapes.
- Nondeterminism.

Different ways to define ”computable."
- TM, RAM machine, circuits, grammar, \(\lambda\)-calculus, \(\mu\)-recursive functions, cellular automata, Conway’s game of life.

Conventional computers.
- ENIAC, TOY, Pentium 4, . . .

New speculative models of computation.
- DNA computers, quantum computers, soliton computers.
TM : As Powerful As TOY Machine

Power = ability to solve more problems.

Turing machines are at least as powerful as a TOY machine:
- Encode state of memory, registers, PC, onto Turing tape.
- Design TM states for each instruction.
- Can do because all instructions:
  - examine current state
  - make well-defined changes depending on current state

Works for all real machines.
- Can simulate at machine level, gate level, . . .

TM : Power Equal to TOY and C

Turing machines are equivalent in power to C programs.
- C program \(\Rightarrow\) TOY program (Lecture A2)
- TOY program \(\Rightarrow\) TM (previous slide)
- TM \(\Rightarrow\) C program (TM simulator, Lecture T1)

Works for all real programming languages.

Assumption: TOY machine and C program have unbounded amount of memory. Otherwise TM is strictly more powerful.

Undecidable Problems

Hilbert’s 10th Problem.
- “Devise a process according to which it can be determined by a finite number of operations whether a given multivariate polynomial has an integral root.”

- Example 1: \(f(x, y, z) = 6x^3yz^2 + 3xy^2 - x^3 - 10\)

- Example 2: \(f(x, y) = x^2 + y^2 - 3\)

Hilbert’s 10th Problem.
- “Devise a process according to which it can be determined by a finite number of operations whether a given multivariate polynomial has an integral root.”

- Problem resolved in very surprising way. (Matijasevic, 1970)

- How can we assert such a mind-boggling statement?
Undecidable Problems

Hilbert’s 10th Problem.
Post’s Correspondence Problem.
Program Equivalence.
Optimal Data Compression.
Virus Identification.

Impossible to write C program to solve any of these problem!

Halting Problem

Halting problem.

- Devise a TM that reads in another TM (encoded in binary) and its initial tape, and determines whether or not that TM ever reaches a halt state.

Busy Beaver

Busy beaver.

- N state Turing machine over \{0, 1\} alphabet.
- Initial tape = all 0’s.
- Leaves as many 1’s on tape as possible, while still halting.

Known Turing machines.

<table>
<thead>
<tr>
<th>N</th>
<th>Ones</th>
<th>Moves</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4098</td>
<td>11,798,826</td>
</tr>
<tr>
<td>8</td>
<td>$6.7 \times 10^{47}$</td>
<td>$2.0 \times 10^{98}$</td>
</tr>
<tr>
<td>9</td>
<td>$1.2 \times 10^{99.5}$</td>
<td>$3.0 \times 10^{1730}$</td>
</tr>
</tbody>
</table>

Undecidable Problems

Halting Problem.
- Write a C program that reads in another program and its inputs, and decides whether or not it goes into an infinite loop.
  - infinite loop often signifies a bug

- Program 2.
  - 8 4 2 1
  - 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

```
while (x > 1) {
    if (x % 2 == 0)
        x = x / 2;
    else
        x = 3*x + 1;
}
```

Hailstone.c

Halting Problem

Theorem (Alan Turing, 1937). Halting problem is undecidable.
- Most famous of all undecidable problems.
- No TM can solve the halting problem.
- Not possible to write a C program either.

Proof intuition.
- Self-reference.
- Grelling’s paradox.

Warmup: Grelling’s Paradox

Grelling’s paradox:
- Divide all adjectives into two categories:
  - autological: self-descriptive
  - heterological: not self-descriptive
  - bisyllabic
  - palindromic
  - edible
  - "heterological"

- How do we categorize heterological?
- not possible
- we can’t have words with these meanings!
  (or we can’t partition adjectives into these two groups)
Assume the existence of $\text{Halt}(f, x)$ that takes as input: any function $f$ and its input $x$, and outputs yes if $f(x)$ halts, and no otherwise.

We prove $\text{Halt}(f, x)$ can’t exist by contradiction.

Note: $\text{Halt}(f, x)$ always returns yes or no.

- infinite loop not possible

```c
#define YES 1
#define NO 0

int Halt(char f[], char x[]) {
    if ( ??? )
        return YES;
    else
        return NO;
}
```

Assume the existence of $\text{Halt}(f, x)$ that takes as input: any function $f$ and its input $x$, and outputs yes if $f(x)$ halts, and no otherwise.

- Construct program $\text{Strange}(f)$ as follows:
  - calls $\text{Halt}(f, f)$
  - halts if $\text{Halt}(f, f)$ outputs no
  - goes into infinite loop if $\text{Halt}(f, f)$ outputs yes

- In other words:
  - if $f(f)$ does not halt then $\text{Strange}(f)$ halts
  - if $f(f)$ halts then $\text{Strange}(f)$ does not halt

```c
void Strange(char f[]) {
    if (Halt(f, f) == NO)
        return;
    else
        while(1)    // infinite loop
}
```

Halting problem is not "artificial."

- Undecidable problem reduced to simplest form to simplify proof.
- Closely related to practical problems.
  - Hilbert’s 10th problem, Post’s correspondence problem, program equivalence, optimal data compression

Practical implications.
- Work with limitations.
- Recognize and avoid unsolvable problems.
- Learn from structure.
  - same theory tells us about efficiency of algorithms (stay tuned)
Philosophical Implications

Caveat: ask a philosopher.

We "assume" that any step-by-step reasoning will solve any technical or scientific problem.
- "Not quite" says the halting problem.
- Anything that is like (could be) a computer has the same flaw:
  - logic
  - physical machines
  - human brain?
  - matter, universe???

A More Powerful Computer???

Post machine (PCP-286).
- Input: set of Post cards.
- Output.
  - YES light if PCP is solvable for these cards
  - NO light if PCP has no solution

PCP is strictly more powerful than:
- Turing machine.
- TOY machine.
- C programming language.
- iMac.
- Any conceivable super-computer.

Why doesn't it violate Church-Turing thesis?

Summary

What is an algorithm?
- Formally: a Turing machine.

Turing’s key ideas:
- Computing is same thing as manipulation symbols.
  - encode numbers as strings
- "Computable at all" == "Computable with a TM"
  - Church-Turing Thesis
- Existence of general-purpose computer (UTM).
  - programmable machine
- Halting Problem is undecidable.

Is it possible, in principle, to write a program to solve any problem?
- No.