Foundations of Computer Science
Lecture 27

Unsolvable Problems

No Automatic Program Verifier for Hello-World
No Ultimate Debugger or Algorithm for PCP
The Complexity Zoo
Last Time: Turing Machines

Intuitive notion of algorithm \( \equiv \) Turing Machine
Solvable problem \( \equiv \) Turing-decidable

\[ \mathcal{L} = \{ \langle G \rangle \mid G \text{ is connected} \} \]

\[ \langle G \rangle = 2; 1; 3; 4 \# 1,2; 2,3; 1,3; 3,4 \]

(\( \langle G \rangle \) is the encoding of graph \( G \) as a string.)

\[ M = \text{Turing Machine that solves graph connectivity} \]
\[ \text{input: } \langle G \rangle, \text{ the encoding of a graph } G. \]
   1: Check that \( \langle G \rangle \) is a valid encoding of a graph and mark the first vertex in \( G \).
   2: REPEAT: Find an edge in \( G \) between a marked and an unmarked vertex.
      Mark the unmarked node or GOTO step 3 if there is no such edge.
   3: REJECT if there is an unmarked vertex remaining in \( G \); otherwise ACCEPT.

To tell your friend on the other coast about this fancy Turing Machine \( M \),
encode its description into the bit-string \( \langle M \rangle \) and send over the telegraph.

You want to solve a different problem? Build another Turing Machine!
Today: Unsolvable Problems

1. Programmable Turing Machines.

2. Examples of unsolvable problems.
   - Post’s Correspondence Problem (PCP)?
   - HalfSum?
   - Auto-Grade?
   - Ultimate-Debugger?

3. $\mathcal{L}_{TM}$: The language recognized by a Universal Turing Machine.
   - $\mathcal{L}_{TM}$ is undecidable – cannot be solved!

4. Auto-Grade and Ultimate-Debugger do not exist.

5. What about HalfSum?
Programmable Turing Machine: Universal Turing Machine

A Turing Machine $M$ has a binary encoding $\langle M \rangle$. Its input $w$ is a binary string.

$\langle M \rangle \# w$ can be the input to another Turing Machine $U_{TM}$.

$$U_{TM}(\langle M \rangle \# w) = \begin{cases} 
\text{halt with ACCEPT} & \text{if } M(w) = \text{halt with ACCEPT}; \\
\text{halt with REJECT} & \text{if } M(w) = \text{halt with REJECT}; \\
\text{loop forever} & \text{if } M(w) = \text{loop forever};
\end{cases}$$

$U_{TM}$ outputs on $\langle M \rangle \# w$ whatever $M$ outputs on $w$. $U_{TM}$ simulates $M$.

**Challenge:** $U_{TM}$ is fixed but can simulate any $M$, even one with a million states.

Entire simulation is done on the tape.
Post’s Correspondence Problem (PCP) and **HALFsum**

**PCP:** Consider 3 dominos:

\[
\begin{array}{c}
\text{d}_1 \\
0 \\
100
\end{array}
\begin{array}{c}
\text{d}_2 \\
01 \\
00
\end{array}
\begin{array}{c}
\text{d}_3 \\
110 \\
11
\end{array}
\]

\[
d_3d_2d_3d_1 = 110011100110011100 = 11001101 = \text{Top and bottom strings match.}
\]

That’s the goal.

**INPUT:** Dominos \(\{d_1, d_2, \ldots, d_n\}\). For example \(\left\{\begin{array}{c}
\text{10} \\
101
\end{array}, \begin{array}{c}
\text{011} \\
11
\end{array}, \begin{array}{c}
\text{101} \\
011
\end{array}\right\}\).

**TASK:** Can one line up finitely many dominos so that the top and bottom strings match?

**HalfSum:** Consider the multiset \(S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\}\), and subset \(A = \{1, 3, 4, 9\}\).

\[
\text{sum}(A) = 17 = \frac{1}{2} \times \text{sum}(S).
\]

**INPUT:** Multiset \(S = \{x_1, x_2, \ldots, x_n\}\). For example, \(S = \{1, 1, 1, 3, 4, 4, 5, 6, 9\}\).

**TASK:** Is there a subset whose sum is \(\frac{1}{2} \times \text{sum}(S) = \frac{1}{2} \times (x_1 + x_2 + \cdots + x_n)\)?
Your first CS assignment: Write a program to print “Hello World!” and halt.

CS1: 700+ submissions!

Naturally, we do not grade these by hand.

**Auto-Grade**: runs each submission and determines if its correct.

What does **Auto-Grade** say for this program:

```plaintext
n = 4;
while(n > 0){
    if(n is not a sum of two primes){
        print("Hello World!") and exit;
    }
    n ← n + 2;
}
```
Wouldn’t it be nice to have the **Ultimate-Debugger**.

\[
\text{Halts}(n = 4; \text{while}(n > 0)\{ \text{if}(n \text{ is not a sum of two primes})\{ \text{print("Hello World!") and exit;} \}\text{n} \leftarrow n + 2;\}) = \begin{cases} \text{YES} & \text{if program halts} \\ \text{NO} & \text{if program infinitely loops} \end{cases}
\]

- We can grade the students program correctly.
- We can solve Goldbach’s conjecture.
- Just think what you could do with **Ultimate-Debugger**.
  - No more infinite looping programs.
Verification: Does A Program Successfully Terminate?

\[ \mathcal{L}_{\text{TM}} = \{ \langle M \rangle \# w \mid M \text{ is a Turing Machine and } M \text{ accepts } w \} \].

\( U_{\text{TM}} \) is a recognizer for \( \mathcal{L}_{\text{TM}} \).

Is there a Turing Machine \( A_{\text{TM}} \) which decides \( \mathcal{L}_{\text{TM}} \)?

- A decider must always halt with an answer.
- \( U_{\text{TM}} \) may loop forever if \( M \) loops forever on \( w \).
- Question: What do these mean: \( M(\langle M \rangle) \) and \( A_{\text{TM}}(\langle M \rangle \# \langle M \rangle) \)?

A diabolical Turing Machine \( D \) built from \( A_{\text{TM}} \):

1. Run \( A_{\text{TM}} \) with input \( \langle M \rangle \# \langle M \rangle \).
2. If \( A_{\text{TM}} \) accepts then REJECT; otherwise (\( A_{\text{TM}} \) rejects) ACCEPT

\( D \) does the opposite of \( A_{\text{TM}} \). Is \( D \) a decider?
Theorem. $A_{\text{TM}}$ does not exist ($\mathcal{L}_{\text{TM}}$ Cannot be Solved)

$A_{\text{TM}}$ exists $\rightarrow$ $D$ exists.

$D$ exists means it will appear on the list of all Turing Machines,

\[
\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \langle M_4 \rangle, \langle D \rangle, \ldots
\]

Consider what happens when $M_i$ runs on $\langle M_j \rangle$, that is $A_{\text{TM}}(\langle M_i \rangle \# \langle M_j \rangle)$.

<table>
<thead>
<tr>
<th>$A_{\text{TM}}(\langle M_i \rangle # \langle M_j \rangle)$</th>
<th>$\langle M_1 \rangle$</th>
<th>$\langle M_2 \rangle$</th>
<th>$\langle M_3 \rangle$</th>
<th>$\langle M_4 \rangle$</th>
<th>$\langle D \rangle$</th>
<th>$\ldots$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle M_1 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_2 \rangle$</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_3 \rangle$</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle M_4 \rangle$</td>
<td>ACCEPT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\langle D \rangle$</td>
<td>REJECT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>ACCEPT</td>
<td>ACCEPT Reject</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ldots$</td>
</tr>
</tbody>
</table>

$D(\langle M_i \rangle)$ does the opposite of $A_{\text{TM}}(\langle M_i \rangle \# \langle M_i \rangle)$.
ULTIMATE-DEBUGGER and AUTO-GRADE Don’t Exist

No general program/algorithm to analyze any other program $M$ and tell if $M$ will accept or not a particular input.

No ULTIMATE-DEBUGGER to analyze other programs and tell if they halt.

No AUTO-GRADE for CS-1 programs.

No solver for PCP.

Suppose ULTIMATE-DEBUGGER $H_{TM}$ exists and decides if any other program halts.

We can use $H_{TM}$ to construct a solver $A_{TM}$ for $L_{TM}$.

\[
A_{TM} = \text{Turing Machine derived from } H_{TM} \text{ (the decider for } L_{HALT})
\]

input: $\langle M \rangle \#w$ where $M$ is a Turing Machine and $w$ an input to $M$.

1: Run $H_{TM}$ on input $\langle M \rangle \#w$. If $H_{TM}$ rejects, then REJECT.

2: Run $U_{TM}$ on input $\langle M \rangle \#w$ and output the decision $U_{TM}$ gives.

Exercise. Show that AUTO-GRADE does not exist.

Exercise. Show that HALF$\Sigma$UM is solvable by giving a decider.
The Landscape

DFA
(no external memory)
(regular expressions)
\{0^1, 0^{3n+1}\}

CFG
(stack)
\{0^n1^n\},
\{ww^R\}

TM-Decider
(RAM)
\{ww\}, \{0^{2n}\},
\{0^n1^n0^n\}

TM-Recognizer
\mathcal{L}_{TM}
ULTIMATE-DEBUGGER
AUTO-GRADE
PCP

Non-Recognizable
\mathcal{L}_{TM}, \mathcal{L}_{HALT}
most languages

Creator: Malik Magdon-Ismail
Unsolvable Problems: 11 / 13
The Path Forward
The Path Forward: Focus on Decidable Problems

FOCS

Theory of Computing

- Decider
  \( U_{TM} = \text{computer} \)
  \( TM = \text{Algorithm} \)

- CFG Parsing
- DFA RegExp

Discrete Math

- Proof, logic
  - Induction
- Recursion
  - Struct. Induction
- Sums, Asymptotics
- Number theory
- Graphs
- Counting
- Probability

Computability & Complexity

- Algorithms & DS
  - Approximation
  - Randomized
  - Distributed

- Cryptography

- Data
  - ML/AI/DM/NLP
  - Vision
  - Graphics
  - Comp. Finance

- Networks
  - Computers
  - Social
  - Data (e.g. www)

- Robotics
- Security
- Programming Languages
  - Compilers
  - Distributed
- Program Analysis
  - Testing
  - Verification

- DB Systems
- Parallel computing
- Operating systems
- Architecture
- Theory
  - Algorithms
  - AI
- Introduction to Algorithms
- Principles of Software
- Software Systems

Chapters 28 & 29

Unsolvable Problems: 12 / 13

Epic Disasters
...the high technology so celebrated today is essentially a mathematical technology.

“To err is human, but to really foul things up you need a computer.” – Paul Ehrlich

- **Mariner rocket explodes (1962).** Formula into code bug resulted in no smoothing of deviations.
  - Lucky Stanislav “funny feeling in my gut” Petrov thought: “surely they’d use more missiles?”
- **Therac 25 (1985).** Concurrent programming bug killed patients through massive 100× radiation overdose.
- **AT&T Lines Go Dead (1990).** 75 million calls dropped (one line of buggy code in software upgrade).
- **Pentium floating point long-division bug (1993).** Cost: $475 million – flawed division table.
- **Ariane rocket explosion (1996).** Cost: $500 million – overflow in 64-bit to 16-bit conversion.
- **Y2K (1999).** Cost: $500 billion spent because year was stored as 2 digits to save space.
- **Mars Climate Orbiter Crash (1998).** Cost: $125 million lost due to metric to imperial units bug.
- **Tesla Self-Driving Car (2016). 1 dead.** Auto-pilot didn’t “see” tractor-trailer.
- **Financial Disasters:** London Stock Exchange down due to single server bug (2009; billions of pounds of trading); Knight Capital computer glitch triggers stock sale (2012; 500 million lost and Knight’s value drops by 75%).
- **Airline Disasters:**
  - AirFrance 447 2009, **228 dead:** pitot-tube failure feeds inconsistent data to programs which then panic pilot.
  - Spanair 5022, 2008, **154 dead:** malware virus.
  - AdamAir 574, 2007, **102 dead:** navigation system errors (and pilot errors).
  - KoreanAir 801, 1997, **228 dead:** ground proximity warning system bug.
  - AeroPerú 603, 1996, **70 dead:** altimeter failures.
  - Scottish RAF Chinook, 1994, **29 dead:** faulty test program
  - AirFrance 296, 1988, **3 dead:** altimeter bug.
  - IranAir 655, 1988, **290 dead:** shot down by US Aegis combat system (misidentified as attacking military plane).
  - KoreanAir 007, 1983, **269 dead:** autopilot took plane into Soviet airspace where it got shot down.
  - Boeing 737 Max, 2018,2019, **346 dead:** attack sensor + algorithm errors.
- **Software errors cost the U.S. $60 billion annually in rework, lost productivity and actual damages.**

**Put effort to make sure your program works fully correctly all the time.**