GEMS OF TCS

UNDECIDABILITY

Sasha Golovnev
March 9, 2021
Everything is a Bit String

• Input to an algorithm is a string
EVERYTHING IS A BIT STRING

• Input to an algorithm is a string
• Algorithm itself is a string
EVERYTHING IS A BIT STRING

• Input to an algorithm is a string

• Algorithm itself is a string

• Every string is an algorithm
Everything is a Bit String

- Input to an algorithm is a string
- Algorithm itself is a string
- Every string is an algorithm
- Given input, algorithm
Everything is a Bit String

- Input to an algorithm is a string
- Algorithm itself is a string
- Every string is an algorithm
- Given input, algorithm
  - either eventually outputs some value
Everything is a Bit String

- Input to an algorithm is a string
- Algorithm itself is a string
- Every string is an algorithm
- Given input, algorithm
  - either eventually outputs some value
  - or never halts
Halting Problem
INFINITE LOOPS

```python
i = 0
while i <= 5:
    print('Infinite loop')
    i += 1
```
INFINITE LOOPS

```python
i = 0
while i <= 5:
    print('Infinite loop')
```

```python
x = True
while x:
    print('Infinite loop')
```
Halting Problem

- Function HALT is defined as follows.
Halting Problem

- Function HALT is defined as follows.
  - The first input is algorithm A
HALTING PROBLEM

- Function HALT is defined as follows.
  - The first input is algorithm A
  - The second input is string x
Halting Problem

- Function HALT is defined as follows.
  - The first input is algorithm $A$
  - The second input is string $x$
  - $\text{HALT}(A, x) = 1$ if $A$ halts on input $x$
Halting Problem

- Function HALT is defined as follows.
  - The first input is algorithm A
  - The second input is string x
  - HALT(A, x) = 1 if A halts on input x
  - HALT(A, x) = 0 if A enters infinite loop on input x
APPLICATIONS OF HALTING PROBLEM

• Algorithm for HALT will help to design bug-free soft (and hardware)
APPLICATIONS OF HALTING PROBLEM

- Algorithm for HALT will help to design bug-free soft (and hardware)
- Algorithm for HALT will (eventually) solve many mathematical problems
APPLICATIONS OF HALTING PROBLEM

- Algorithm for **HALT** will help to design bug-free soft (and hardware)
- Algorithm for HALT will (eventually) solve many mathematical problems
  - Goldbach’s conjecture

\[
\text{Every even number (>2) } n \text{ can be written as a sum of two primes: } n = p_1 + p_2
\]

This is true for every \( n \leq 10^{18} \)

**Algorithm A:**

\[
\text{for } n = 4 \text{ to } \infty, \text{ if } n \text{ is even, } n \not\equiv p_1 + p_2 \text{ for any } p_1, p_2 < n, \text{ then HALT}
\]

\[
\text{HALT(A)} \text{ if it tells me } \text{ then A halts, conj is false, otherwise conj is true}
\]
APPLICATIONS OF HALTING PROBLEM

- Algorithm for HALT will help to design bug-free soft (and hardware)
- Algorithm for HALT will (eventually) solve many mathematical problems
  - Goldbach’s conjecture
  - Collatz conjecture

$f(n) = \begin{cases} \frac{n}{2}, & \text{if } n \text{ is even} \\ 3n+1, & \text{if } n \text{ is odd} \end{cases}$

- Whatever $n$ you start with, you get to 1
  - $\forall n \leq 10^{20}$

12 $\xrightarrow{f(n)}$ 6 $\xrightarrow{f(c)}$ 3 $\xrightarrow{}$ 10 $\xrightarrow{}$ 5 $\xrightarrow{}$ 16 $\xrightarrow{}$ 8 $\xrightarrow{}$ 4 $\xrightarrow{}$ 2 $\xrightarrow{}$ 1
Applications of Halting Problem

- Algorithm for HALT will help to design bug-free soft (and hardware)
- Algorithm for HALT will (eventually) solve many mathematical problems
  - Goldbach’s conjecture
  - Collatz conjecture
  - Twin (cousin/sexy) prime conjecture
Applications of Halting Problem

- Algorithm for HALT will help to design bug-free soft (and hardware)
- Algorithm for HALT will (eventually) solve many mathematical problems
  - Goldbach’s conjecture
  - Collatz conjecture
  - Twin (cousin/sexy) prime conjecture
  - Odd perfect number
APPLICATIONS OF HALTING PROBLEM

• Algorithm for HALT will help to design bug-free soft (and hardware)

• Algorithm for HALT will (eventually) solve many mathematical problems
  • Goldbach’s conjecture
  • Collatz conjecture
  • Twin (cousin/sexy) prime conjecture
  • Odd perfect number
  • ...
Clearly, every function can be computed given sufficient time
Except this is **not** true
HALTING IS UNDECIDABLE

Assume there is alg $H$ that solves Halting Problem

What does $H'(H')$ do?

Case 1. $H'(H')$ halts $\Rightarrow$ $H'(H')$ infinite loop
Case 2. $H'(H')$ infinite loop $\Rightarrow$ $H'$ halts

$H'$ halts $\Rightarrow$ $H'$ infinite loop
$H'$ forever $\Rightarrow$ $H'$ halt

Assumption was wrong $\Rightarrow$ No alg $H$ for HALT
\[ H'(P) : \]
\[ b = H(C, P) \]
\[ \text{if } b = 0 : \]
\[ \text{while } \text{True} : \]
\[ \text{else} \]
\[ \text{Return} \]

\[ H' \text{ doesn’t exist because} \]
\[ H'(H') \text{ cannot halt or run forever} \]

\[ \therefore H \text{ cannot exist} \]
• Easy to solve for one input and one algorithm

For one fixed \( A \), one fixed \( x \)

It's easy to decide if \( A(x) \) halts or not

Alg One: always outputs "halts"
Alg Two: always outputs "inF loop"
REMARKS

- Easy to solve for one input and one algorithm
- But impossible to solve for all inputs and algorithms
Remarks

- Easy to solve for one input and one algorithm
- But impossible to solve for all inputs and algorithms
- Result holds for all computational models
REMARKS

- Easy to solve for one input and one algorithm
- But impossible to solve for all inputs and algorithms
- Result holds for all computational models
- All non-trivial properties of algorithms are undecidable
Compiler
• Takes
COMPILER

• Takes
  • String A describing algorithm
  • String x describing algorithm’s input
• Takes
  • String A describing algorithm
  • String x describing algorithm’s input
• Outputs A(x)
• Takes
  • String A describing algorithm
  • String x describing algorithm’s input
• Outputs A(x)

• Compiler itself is an algorithm, too!
UNDECIDABLE PROBLEM

Un computable

- Function $A_{\text{diag}}(x)$ is defined as follows
**Undecidable Problem**

- Function $A_{\text{diag}}(x)$ is defined as follows

- If the algorithm $x$ on input $x$ outputs 1, then $A_{\text{diag}}(x) = 0$
UNDECIDABLE PROBLEM

- Function $A_{\text{diag}}(x)$ is defined as follows

- If the algorithm $x$ on input $x$ outputs 1, then $A_{\text{diag}}(x) = 0$

- If the algorithm $x$ on input $x$ outputs other value or never halts, then $A_{\text{diag}}(x) = 1$

\[\begin{align*}
x(x) & \quad \text{runs forever} \\
x(x) & \quad \text{outputs 0} \\
x(x) & \quad \text{outputs "cat"} \\
x(x) & \quad \text{doesn't compile}
\end{align*}\]
There is no alg for $A_{	ext{diag}}$. 

Assume alg $A$ solves our problem: 

$A(A) \neq A_{\text{diag}}(A) \Rightarrow$ contradiction. 

There is no alg for $A_{\text{diag}}$. 

$A_{\text{diag}}(0) = 0$

$A_{\text{diag}}(1) = 1$

$A_{\text{diag}}(00) = 1$

$A_{\text{diag}}(01) = 0$
Reduction from Diag to HALT

We already know that Diag is undecidable, we’ll use this to prove that HALT is undecidable

• Assume there exists an algorithm for HALT
Reduction from Diag to HALT

• Assume there exists an algorithm for HALT

• Given input \( x \), we check if the algorithm \( x \) halts on \( x \)

\[
\text{HALT}(x, x)
\]
REDUCTION FROM DIAG TO HALT

• Assume there exists an algorithm for HALT

• Given input $x$, we check if the algorithm $x$ halts on $x$

• If it doesn’t halt, output 1

$$A_{\text{diag}}(x) = 1$$
Reduction from Diag to HALT

• Assume there exists an algorithm for HALT

• Given input x, we check if the algorithm x halts on x

• If it doesn’t halt, output 1

• If it halts and outputs 1, output 0

\[ A_{\text{diag}}(x) = 0 \]
Reduction from Diag to HALT

• Assume there exists an algorithm for HALT

• Given input x, we check if the algorithm x halts on x

• If it doesn’t halt, output 1

• If it halts and outputs 1, output 0

• If it halts and outputs something else, output 1
Summary

First proof: Assuming HALT can be solved $\Rightarrow$ design $H'$ such $H'(H')$ cannot halt or run forever.

Second proof: Diagonalization defines problem s.t. it differs from every alg $A$ on at least one input (for example, input $A$) from this problem cannot be solved by any algorithm.

Third proof: Assuming HALT can be solved, we solved Diog-contradiction, HALT is undecidable.