Gems of TCS

Undecidability

Sasha Golovnev

October 5, 2022

Alan Turing

1912–1954

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

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

Infinite Loops

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

```
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
    - $\text{HALT}(A, x) = 1$ if A halts on input x
    - $\text{HALT}(A, x) = 0$ 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

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

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

REMARKS

- Easy to solve for one input and one algorithm

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

Compiler

- Takes

Compiler

- Takes
  - String A describing algorithm
  - String x describing algorithm's input

Compiler

- Takes
  - String A describing algorithm
  - String x describing algorithm's input
- Outputs $A(x)$

Compiler

- Takes
  - String A describing algorithm
  - String x describing algorithm's input
- Outputs $A(x)$

- Compiler itself is an algorithm, too!

Undecidable Problem

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

Diagonalization

Reduction from Diag to HALT

- 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

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

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

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