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

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

x = True
while x:
    print('Infinite loop')
• Function HALT is defined as follows.
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 $x$
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 \( x \)
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)
  - 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

...
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

- But impossible to solve for all inputs and algorithms

- Result holds for all computational models

- All non-trivial properties of algorithms are undecidable
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

• Takes

• Outputs

• Compiler itself is an algorithm, too!
• 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 $A_{\text{diag}}(x) = 0$

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