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 halts
  - either eventually outputs some value
  - or never halts or inf loop

Halting Problem

Infinite Loops

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

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

Every even number (>2) n can be written as a sum of two primes: $n = p_1 + p_2$

This is true for every $n \le 10^{18}$

Alg A: for $n = 4$ to ∞, n is even
If $n \neq p_1 + p_2$ for any $p_1, p_2 < n$,
Then HALT

If Conj is true Then A runs forever
If Conj is false Then A halts

HALT(A) if it tells me than 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

whatever n you start with, you get to 1

True $\forall n \leq 10^{20}$

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

$$12 \xrightarrow{f(n)} 6 \xrightarrow{f(\cdot)} 3 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow$$
$$\rightarrow 4 \rightarrow 2 \rightarrow 1$$

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

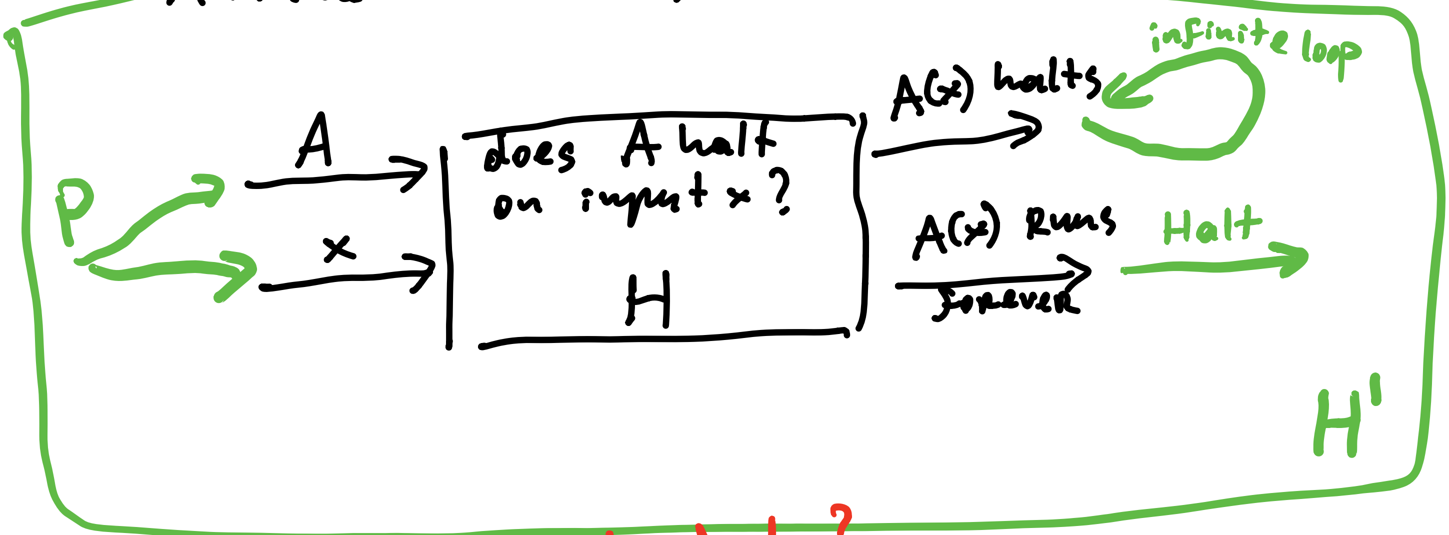
- Algorithm for HALT will help to design bug-free soft (and hardware)
- Algorithm for HALT will (eventually) solve many mathematical problems
    - Goldbach's conjecture
    - Collatz conjecture
    - Twin (cousin/sexy) prime conjecture
    - Odd perfect number
    - ...

Clearly, every function can be computed given sufficient time

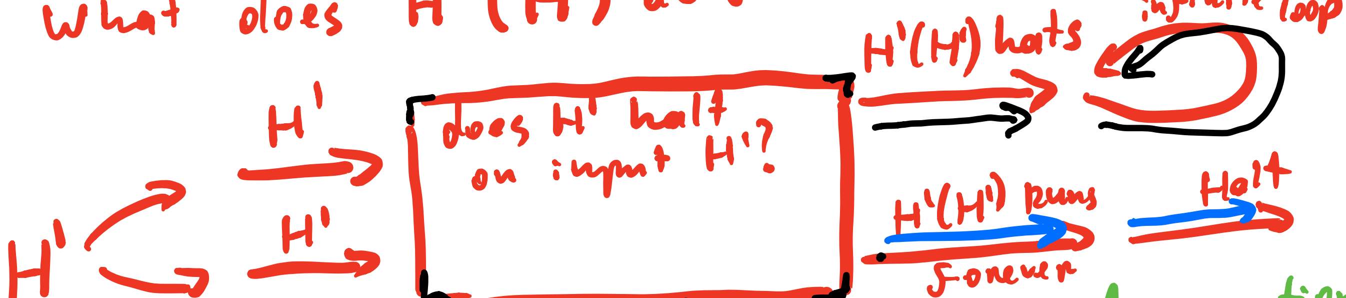
Except this is **not** true

Halting is Undecidable

Assume there is alg H that solves Halting Problem



What does H'(H') do?



Case 1. H'(H') halts → H'(H') infinite loop

Case 2. H'(H') infinite loop → H' halts

⇒ Assumption was wrong ⇒ No alg H for HALT

```
H'(P):
    b = H(P, P)
    if b == 0:
        while True:
    else
        return
```

H' doesn't exist because H'(H') cannot halt or run forever

⇒ H cannot exist

Remarks

- 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 <u>all</u> inputs and algorithms

Remarks

- Easy to solve for one input and one algorithm
- But impossible to solve for <u>all</u> 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

it's undecidable to understand
- if Alg always outputs the same value
- if Alg ever outputs 0
- ...

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! is a string 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

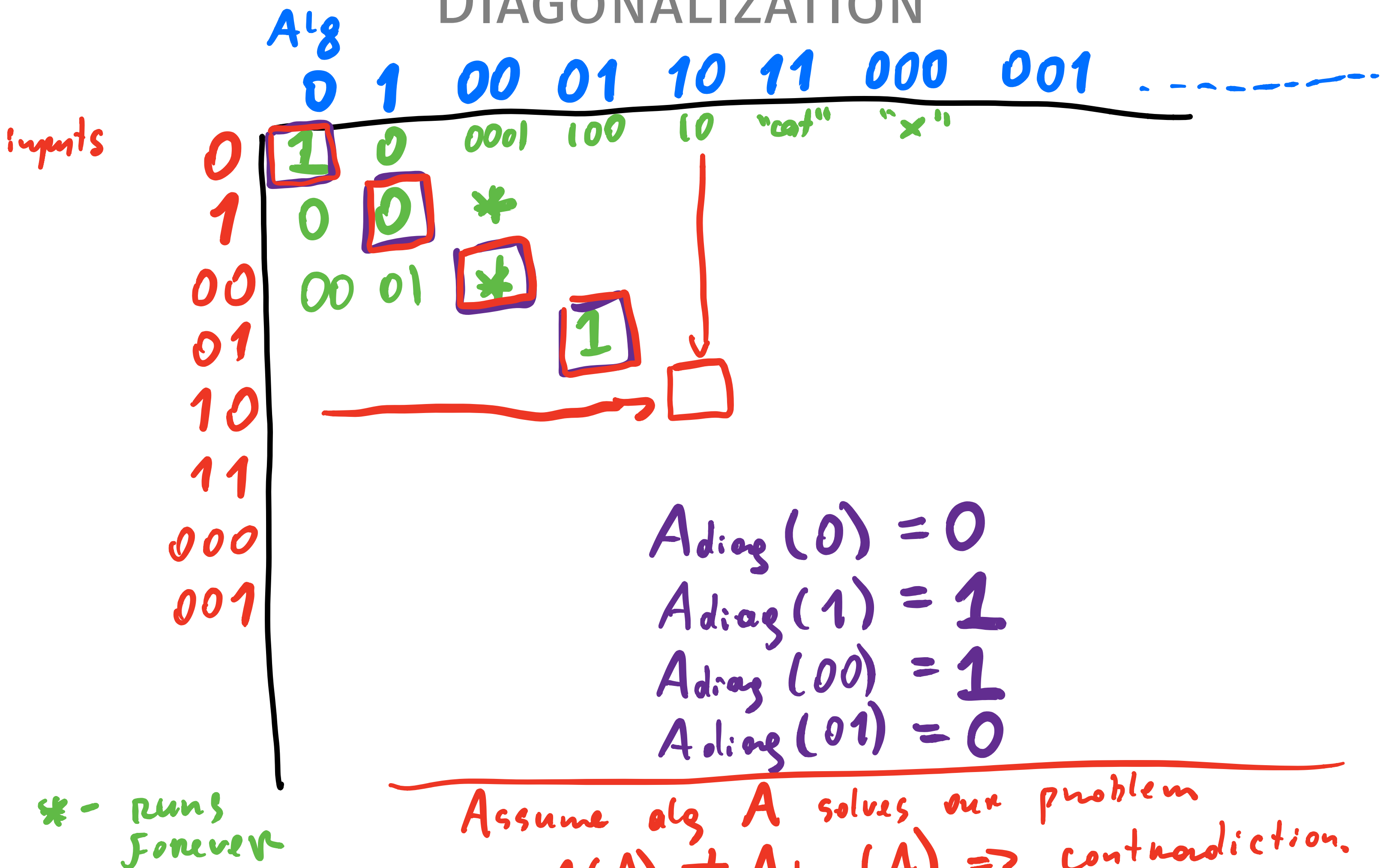
- If the algorithm x on input x outputs 1, then $A_{\text{diag}}(x) = 0$

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

$x(x)$ runs forever
$x(x)$ outputs 0
$x(x)$ outputs "cat"
$x(x)$ doesn't compile

Diagonalization



$A_{diag}(0) = 0$
$A_{diag}(1) = 1$
$A_{diag}(00) = 1$
$A_{diag}(01) = 0$

Assume alg A solves our problem

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

There is no alg for A_{diag}.

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

$HALT(x, x)$

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

$A_{diag}(x) = 1$

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

$A_{diag}(x) = 0$

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

$A_{diag}(x) = 1$

Summary

First proof: Assuming HALT can be solved ⟹ design H' such H'(H') cannot halt or run forever

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

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