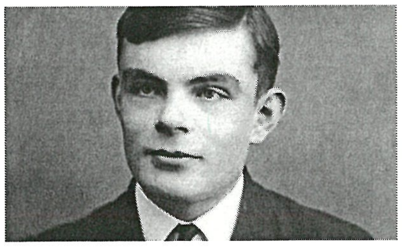
Undecidability

There are problems that cannot be solved by computer programs (i.e. algorithms) even assuming unlimited time and space.



Proved by Alan Turing (and Alonzo Church) in 1936

(Watch The Imitation Game!)

(Watch Hidden Figures!)

What is a computer program/algorithm?

- At the time, there were no electronic computers. A computer referred to a person who computes.
- Turing's idea of a "computer program" was a list of instructions that a person could follow.
- For us, an algorithm could refer to any of the following:
    - Racket, C, and C++ programs
    - Turing machines
    - High-level pseudo-code

What does it mean for an algorithm to solve a problem?

- The algorithm must produce the correct output for every input.

We focus on decision problems.
A decision problem has YES/NO answers.

A decision problem is

- Decidable iff there exists an algorithm to solve it.
- Undecidable iff there does not exist an algorithm to solve it.

Examples of decision problems:

1. Given a propositional formula, is it satisfiable?
This problem is decidable. Here is an algorithm: Construct a truth table for the propositional formula. The formula has to have a finite number of variables, so the truth table will be finite. Check whether there is a T in the final column of the truth table. If there is a T in the column, then the formula is satisfiable. Otherwise, the formula is not satisfiable.

2. Given a predicate formula, is it valid?
Undecidable

3. Given a positive integer, is it prime?
This problem is decidable. Here is an algorithm: Let the given positive integer be n. For every integer k from 2 up to n-1, try dividing n by k. If k divides n, then n is not prime. If k does not divide for any k, then n is prime.

4. Given a program and a Hoare triple, does the program satisfy the Hoare triple under partial correctness?

<u>Undecidable</u>

5. Given a program and a Hoare triple, does the program satisfy the Hoare triple under total correctness?

<u>Undecidable</u>

6. Given two programs, do the two programs produce the same output for every input?

<u>Undecidable</u>

7. Given a program and an input, does the program terminate on the input?

<u>Undecidable</u>

This is also called the halting problem. This is one of the first known problems to be proven to be undecidable.

The Halting Problem:
Given a program P and an input I, will P halt on I?

- "Halts" means "terminates" or "does not get stuck".
- One of the first known undecidable problems

The Halting Theorem: The halting problem is undecidable. That is, there does not exist an algorithm H which solves the halting problem for every program P and input I.

Proof by contradiction:

Assume that there exists an algorithm H(P,I), which solves the halting problem for every program P and input I.

We need to derive a contradiction, which shows that H does not exist.

Our approach:
We will construct an algorithm X(P), which takes a program P as input. We will show that H always gives the wrong answer when predicting whether the program X halts on the input X. That is,
- If H(X,X) returns yes, then X does not halt on X.
- If H(X,X) returns no, then X halts on X.

The algorithm X(P) does the following three things:

(1) Makes two copies of P.
(2) Runs H(P,P).
(3) X behaves differently depending on the result of H(P,P).
    (1) If H(P,P) outputs yes, X goes into an infinite loop and runs forever.
    (2) If H(P,P) outputs no, X terminates.

Let's compare the result of X(X) and the output of H(X,X).

If H(X,X) returns yes, then X goes into an infinite loop and does not halt on X. However, H predicted that X halts on X. H's prediction was wrong. This contradicts our assumption that H solves the halting problem for any program and input.

If H(X,X) returns no, then X terminates and halts on X. However, H predicted that X does not halt on X. H's prediction was wrong. This again contradicts our assumption that H solves the halting problem for any program and input.

Therefore, our assumption must be wrong, and H does not exist.

QED

Proving Undecidability via Reduction

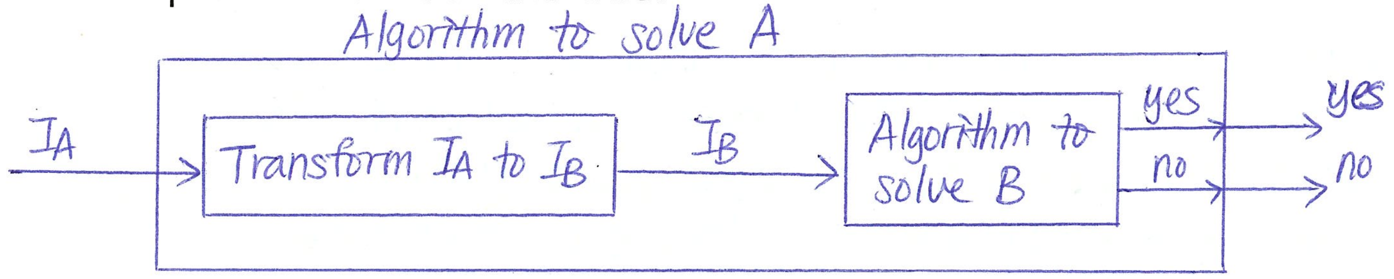
Now that we know the halting problem is undecidable. How do we prove that another problem is undecidable?

- We could prove it from scratch, or…
- We could prove that this problem is as hard as the halting problem; hence it is undecidable.

Problem A is reducible to problem B.

- An algorithm for solving B could be used as a subroutine for solving A.
- If B is decidable, then A is decidable. (If there is an algorithm to solve B, then there is an algorithm to solve A.)
- If A is undecidable, then B is undecidable.

A picture to illustrate this:



Question: Prove that problem X is undecidable.

Proof: We reduce the halting problem to problem X. (Describe the reduction). Since the halting problem is undecidable, problem X is also undecidable.

QED

Halting-no-input Problem: Given a program P (that requires no input), does P halt?

Theorem: The halting-no-input problem is undecidable.

Proof by contradiction:

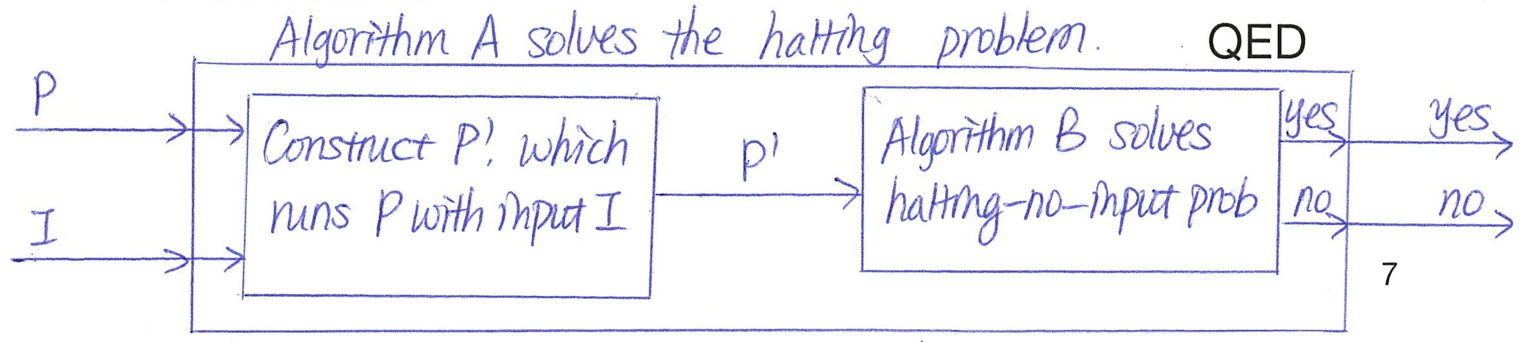
Assume that there is an algorithm B which solves the halting-no-input problem.

We will construct an algorithm A to solve the halting problem. Algorithm A works as follows:

- A takes two inputs, a program P and an input I.
- Construct a program P' as follows: P' runs P with input I and outputs the result of P(I).
- Run algorithm B with P' as the input.
- Return the result of B(P').

By our construction of algorithm A, P' halts if and only if P halts on input I. Therefore, if algorithm B solves the halting-no-input Problem for input P', then the algorithm A solves the halting problem for inputs P and I. By our assumption, algorithm B solves the halting-no-input problem. Thus, algorithm A solves the halting problem. This contradicts the fact that the halting problem is undecidable.

QED


Algorithm A solves the halting problem.
P
I
Construct P', which runs P with input I
P'
Algorithm B solves halting-no-input prob
yes
no
yes
no

Problem (Both-Halt): Given two programs P1 and P2, do both problems halt?

Theorem: The both-halt problem is undecidable.

Proof by contradiction:

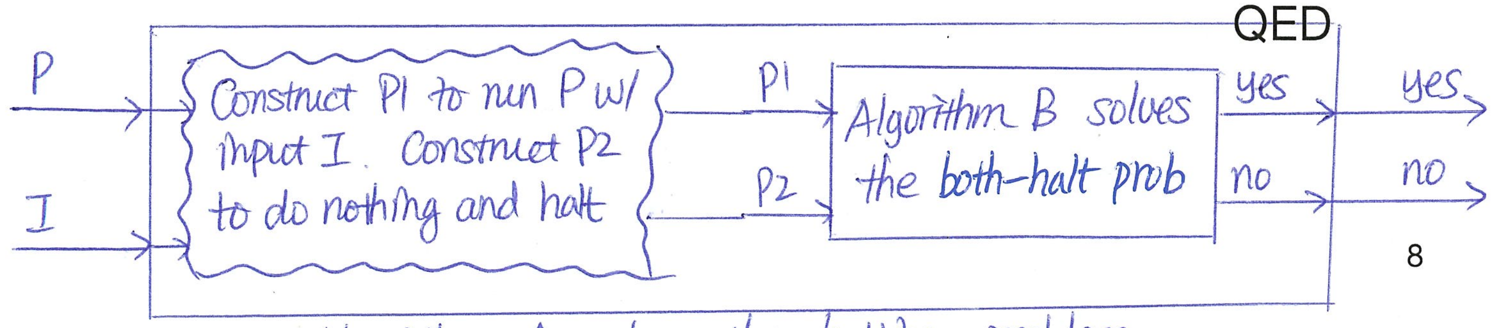
Assume that there is an algorithm B to solve the both-halt problem.

We will construct an algorithm A to solve the halting problem. Algorithm A works as follows:

- A takes two inputs: a program P and an input I.
- Construct program P1, which runs P with input I and returns the result of P(I).
- Construct program P2, which does nothing and halts.
- Run algorithm B with P1 and P2 as inputs.
- Return the result of B(P1, P2).

By the design of algorithm A, P1 and P2 both halt if and only if P halts on input I. Thus, if algorithm B solves the Both-Halt problem, then algorithm A solves the Halting problem. By our assumption, algorithm B solves the Both-Halt problem. Thus, algorithm A solves the Halting Problem. This contradicts the fact that the Halting Problem is undecidable.

QED


P
I
Construct P1 to run P w/ input I. Construct P2 to do nothing and halt
P1
P2
Algorithm B solves the both-halt prob
yes
no
yes
no

Algorithm A solves the halting problem.

Total Correctness: Given a Hoare triple (| P |) C (| Q |), does C satisfy the triple under total correctness?

Theorem: The total correctness problem is undecidable.

Proof by contradiction:

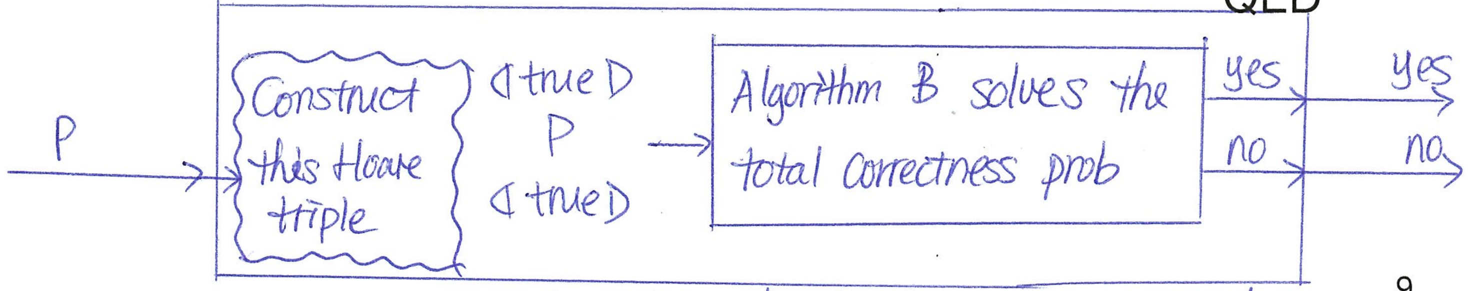
Assume that we have an algorithm B to solve the total correctness Problem.

We will construct an algorithm A to solve the halting-no-input problem. Algorithm A works as follows:

- A takes one input, a program P.
- Run algorithm B with the following Hoare triple as input.
    - (| true |) P (| true |)
- Return the result of algorithm B.

Therefore, the Hoare triple is satisfied under total correctness if and only if P halts. Therefore, if algorithm B solves the total correctness problem, then algorithm A solves the halting-no-input problem. By our assumption algorithm B solves the total correctness problem. Thus, algorithm A solves the halting-no-input problem. This contradicts with the fact that the halting-no-input problem is undecidable.

QED


P
Construct this Hoare triple
(| true |)
P
(| true |)
Algorithm B solves the total correctness prob
yes
no
yes
no

Algorithm A solves the halting-no-input prob.

Partial Correctness: Given a Hoare triple (| P |) C (| Q |), Does C satisfy the triple under partial correctness?

Theorem: The partial correctness problem is undecidable.

Proof by contradiction:

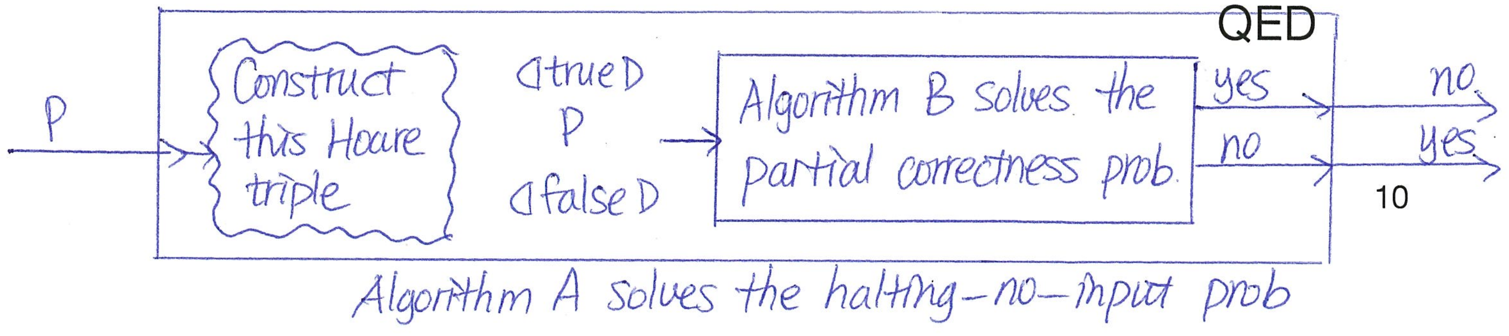
Assume that we have an algorithm B to solve the partial correctness problem.

We will construct an algorithm A to solve the halting-no-input problem. Algorithm A works as follows.

- A takes one input, a program P.
- Run algorithm B with the following Hoare triple as input.
    - (| true |) P (| false |)
- Return the result of algorithm B.

It is impossible for any program to satisfy the above Hoare triple because the post condition is false. Therefore, the Hoare triple is satisfied under partial correctness if and only if P does not halt. Therefore, if algorithm B solves the partial correctness problem, then algorithm A solves the halting-no-input problem. By our assumption algorithm B solves the partial correctness problem. Thus, algorithm A solves the halting-no-input problem. This contradicts with the fact that the halting-no-input problem is undecidable.

QED


P
Construct this Hoare triple
(| true |) P (| false |)
Algorithm B solves the partial correctness prob
yes
no
no
yes
Algorithm A solves the halting-no-input prob