Undecidability

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

Proved by Alan Turing in 1936

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 ______ input.
We focus on decision problems.

A decision problem ________________________________

A decision problem is
- Decidable iff ________________________________.
- Undecidable iff ________________________________

Examples of decision problems:

1. Given a propositional formula, is it satisfiable?

2. Given a predicate formula, is it valid?

3. Given a positive integer, is it prime?

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

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

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

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

(2)

(3)

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

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

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

Proof by contradiction:

Suppose that we have an algorithm $A$ to solve the Halting-no-input Problem. We can use it to solve the Halting problem.

A proof by picture: QED
Total Correctness Problem: 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:

Suppose that we have an algorithm A to solve the Total Correctness Problem. We can use it to solve the Halting-no-input Problem.

A proof by picture:

QED
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:

Suppose that we have an algorithm $A$ to solve the Partial Correctness Problem. We can use it to solve the Halting-no-input Problem.

A proof by picture: QED