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 requires no input), does P halt?

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

Proof by contraction:

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.

QED

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. The algorithm A has the following steps.

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:

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. The algorithm A has the following steps.

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:

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.

QED