The Halting Problem

Lecture 14

Binghui Peng

Announcement

- Homework 3 come out today, due April 12th, 11:59.
- Midterm grade would come out on Thursday

Today's lecture: Is every language decidable?

Prerequisite 1: String encoding of TM

A Turing machine itself can be written down as a finite string.
Therefore a machine can receive as input:

1. an ordinary string w,
2. the description M of another machine,
3. a pair of description and ordinary string $\langle M, w \rangle$.

Prerequisite 2: Universal Computation Theorem

We can imagine a **universal Turing machine**. The universal Turing machine receives input $\langle M, w \rangle$, where M is a machine and w is a string. Then it

1. decodes the pair encoded by $\langle M, w \rangle$,
2. simulates machine M on input w,
3. accepts or rejects exactly when M does.

The proof is not hard, but we omit the proof in the lecture

The Halting Problem

Definition. The decision problem of HALT

$$\mathrm{HALT} = \{\langle M, w\rangle : M \text{ halts on input } w\}.$$

That is, given a program and an input, determine whether the program eventually stops.

Main Theorem

Theorem: HALT is undecidable.

Main Theorem

Theorem: HALT is undecidable.

In another word, there is no single Turing machine (or program or algorithm) that correctly decides, for every pair (M, w), whether M halts on input w.

Main Theorem

Theorem: HALT is undecidable.

In another word, there is no single Turing machine (or program or algorithm) that correctly decides, for every pair (M, w), whether M halts on input w.

Proof strategy: **Diagonalization**

Proof by contradiction: Suppose a TM for HALT Exists

Assume for contradiction that there is a decider H for HALT. So on every encoded input $\langle M, w \rangle$, machine H halts and answers:

1. yes if M halts on w,
2. no if M does not halt on w.

We will use H to build a machine that behaves paradoxically on itself.

The Diagonal Machine D

Construct a new machine D that works as follows on input M:

1. Run H on $\langle M, M \rangle$.
2. If H says that M halts on input M, then loop forever.
3. If H says that M does not halt on input M, then halt immediately.

Run D on Its Own Description

$D(M)$ does the opposite of $H(\langle M, M \rangle)$.

Run D on Its Own Description

$D(M)$ does the opposite of $H(\langle M, M \rangle)$.

Now run D on itself.

Run D on Its Own Description

$D(M)$ does the opposite of $H(\langle M, M \rangle)$.

Now run D on itself.

There are only two possibilities.

Run D on Its Own Description

$D(M)$ does the opposite of $H(\langle M, M \rangle)$.

Now run D on itself.

There are only two possibilities.

- **Case 1:** $H(\langle D, D \rangle)$ says "halts." Then D loops forever on input D.

Run D on Its Own Description

$D(M)$ does the opposite of $H(\langle M, M \rangle)$.

Now run D on itself.

There are only two possibilities.

- **Case 1:** $H(\langle D, D \rangle)$ says "halts." Then D loops forever on input D.
- **Case 2:** $H(\langle D, D \rangle)$ says "does not halt." Then D halts on input D.

In both cases we get a contradiction. Therefore our assumption was false.

What We Have Shown

Interpretation of non-decidablity

- There is **no single algorithm/program** always correctly decides halting (for every problem and every input)
- It does not mean that, given a specific program and input, we can not determine whether it halts or not

The halting problem is not an isolated curiosity. Once we know one natural problem is undecidable, many others follow.