Introduction to Undecidability

Alice Gao

Lecture 22

Outline

Learning Goals

By the end of this lecture, you should be able to:

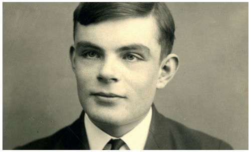
- Define decision problem.
- Define decidable and undecidable problems.
- Prove that a decision problem is decidable
  by giving an algorithm to solve it.
- Describe the halting problem.
- Prove that the halting problem is undecidable.

Exploring the limitation of computation

Most of CS focuses on what we CAN compute.

Are there problems that CANNOT be solved by a computer even with unlimited time and space?

The answer is yes. This was proved by Alan Turing in 1936.



What is a computer program/algorithm?

In the old days, there were no electronic computers. A computer refers to a person who computes. (Watch Hidden Figures.)

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

A decision problem is decidable/undecidable

We focus on decision problems. **A decision problem** is a problem which calls for an answer of yes or no, given some input.

A decision problem is **decidable** if and only if there exists an algorithm that produces the correct output for the problem for every input. Otherwise, the decision problem is **undecidable**.

CQ: Examples of decision problems

Consider a Propositional formula with n propositional variables. We can write down its truth table, which is finite (2^n rows).

CQ: Given a Propositional formula, is it satisfiable?

[✓ (A) This problem is decidable.]

(B) This problem is undecidable.

(C) I don't know.

If the formula is true in at least one row of the truth table, then it is satisfiable. Otherwise, it is not satisfiable. This algorithm will terminate because the size of the truth table is finite.

CQ: Examples of decision problems

Intuitively, there is an infinite number of valuations.
A naive algorithm can try to enumerate all the valuations and evaluate the formula under all the

CQ: Given a Predicate formula, is it valid?
(Take a guess. All answers will be marked correct.)

(A) This problem is decidable.

[✓ (B)] This problem is undecidable.

(C) I don't know.

valuations. However, this naive algorithm will not terminate.

CQ: Examples of decision problems

Given a positive integer n, determine whether d divides n for every $2 \leq d \leq n$. If there exists a $2 \leq d \leq n$ such that $d \mid n$, then n is not prime. Otherwise,

CQ: Given a positive integer, is it prime?

[✓ (A) This problem is decidable.]

(B) This problem is undecidable.

(C) I don't know.

n is prime. This algorithm will terminate because there is a finite number of values for d.

CQ: Examples of decision problems

CQ: Given a program P and an input I,
does P terminate when run with the input I?
(Take a guess. All answers will be marked correct.)

(A) This problem is decidable.

[✓ (B) This problem is undecidable.]

(C) I don't know.

This is the famous halting problem.

A few more remarks about decidable/undecidable problems

- An algorithm must terminate after finitely many steps.
- To prove that a problem is decidable, write down an algorithm to solve/decide it for every input.
- For an undecidable problem, there may exist an algorithm which gives the correct output for the problem for some particular inputs. The problem is undecidable because no algorithm can give the correct output for the problem for every input.

The Halting Problem

D = {all algorithms, all programs, all inputs}

$A(x)$: x is an algorithm $I(x)$: x is an input

$P(x)$: x is a program $S(x,y,z)$: x solves the halting problem for program y and input z.

The decision problem: Given a program P and an input I, will P halt when run with input I?

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

The Halting problem is undecidable.

There does not exist an algorithm H,
which gives the correct answer for the Halting problem
for every program P and every input I.

Exercise: Translate the above statement into a Predicate formula.

$$\neg(\exists x(A(x) \land \forall y \forall z(P(y) \land I(z) \to S(x,y,z))))$$

The Halting Problem is Undecidable - A Proof by Video

https://www.youtube.com/watch?v=92WHN-pAFCs

- Why can we feed a program as an input to itself?
  We can convert any program to a string, then we can feed the string to the program as input.
- What does the negator do?
  It negates the output of H. If H predicts that the program halts, then the negator goes into an infinite loop and does not halt. If H predicts that the program does not halt, then the negator halts. The negator is designed to make H fail at its prediction task.
- Why do we need the photocopier?
  In the video, H takes two inputs. We need to make two copies of the input. In code, we do not need the photocopier, and we can simply call H(P,P).

The Halting Problem is Undecidable

Theorem: The Halting problem is undecidable.

Proof Sketch.

We will prove this by contradiction.

Assume that there exists an algorithm H, which solves the Halting problem for every program and every input.

We will construct an algorithm X, which takes program P as input.

We will show that H gives the wrong answer when predicting whether the program X halts when run with input X. This contradicts the fact that H solves the Halting problem for every program and every input. Therefore, H does not exist.

□

Theorem: The halting problem is undecidable.
(There does not exist an algorithm H which gives the correct answer to the halting problem for every program P and every input I.)

Proof by contradiction:

Assume that there exists an algorithm $H(P, I)$, which gives the correct answer to the halting problem for every program P and every input I.

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

We construct an algorithm X(P) which takes a program P as input and does the following things:

(1) Runs $H(\underset{\uparrow \text{ program}}{P}, \underset{\nwarrow \text{ input}}{P})$ (predict whether P will halt when run w/ input P).

(2) If H(P, P) outputs $\overset{\text{P will halt on input P}}{\text{"yes"}}$, then X does not halt. (i.e. runs an infinite loop).

If H(P, P) outputs $\overset{\text{P will not halt on input P.}}{\text{"no"}}$, then X halts.

We claim that H is always wrong when predicting whether X halts when run with input X. Let's compare X(X) and H(X, X).

① If H(X, X) outputs "yes", then H predicts that X halts when run with input X, but X does not halt. H's prediction is wrong, which is a contradiction.

(2) If $H(X,X)$ outputs "no", then H predicts that X does not halt when run with input X, but X halts. H's prediction is wrong again, which is a contradiction.

Our assumption was wrong and H does not exist.

QED

Revisiting the learning goals

By the end of this lecture, you should be able to:

- Define decision problem.
- Define decidable and undecidable problems.
- Prove that a decision problem is decidable
  by giving an algorithm to solve it.
- Describe the halting problem.
- Prove that the halting problem is undecidable.