I. Problem and Effective Solution

Yuxi Fu

BASICS, Shanghai Jiao Tong University

A Number of Scenarios

You are busy with job interviews. You have been interviewed by the heads of the personnel departments of the following companies:

- A software company developing a compiler capable of checking if a program contains a loop.
- A hardware company designing a computer that can solve some problems no existing computers can solve.
- A service provider working on a theorem prover that is supposed to answer every question about numbers.

What do you think?

The First Question

Computer Science is a science of problem solving.

The first question Computer Science must address is this:

What problems can be solved by computers?

The First Question

Computer Science is a science of problem solving.

The first question Computer Science must address is this:

What problems can be solved by computers?

Computability Theory tells us how to answer the question.

Significance of the Question

It is not only an important question in Computer Science.

It is also an important question in Logic and Philosophy.

Effective Procedure

An effective procedure consists of a finite set of instructions which, given an input from some set of possible inputs, enables us to obtain an output through a systematic execution of the instructions that terminates in a finite number of steps.

Proof vs. Verification

Unbounded search is in general not effective.

Bounded search is effective.

Proof vs. Verification

Unbounded search is in general not effective.

Bounded search is effective.

Theorem proving is in general not effective.

Proof verification is effective.

Problem and Problem Instance

How does a computer solve the Hamiltonian Cycle problem?

How is a problem instance represented in a computer?

How is the answer to a problem instance represented?

How is an effective procedure formalized?

Formalization of Problem

In a formal theory of computability, every problem instance is represented by a number and every number represents a problem instance. Every answer is also represented by a number.

A problem is a function $f : \omega \to \omega$ from numbers to numbers.

A problem is computable if it can be calculated by a program.

Problem Variety

decision problem (e.g. HCP), optimisation problem (e.g. TSP), evaluation problem (e.g. $\log(x)$), counting problem (e.g. $\sharp$PATH), ...

Decision Problem

A problem $f : \omega \to \omega$ is a decision problem if the range $rng(f)$ of f is $\{0, 1\}$, where 1 denotes a 'yes' answer and 0 a 'no' answer.

A decision problem g can be identified with the set $\{n \mid g(n) = 1\}$.

Conversely a subset A of ω can be seen as a decision problem via the characteristic function of A:

$$c_A(n) = \begin{cases} 1, & \text{if } x \in A, \\ 0, & \text{otherwise.} \end{cases}$$

Decision Problem as Predicate

A decision problem can be stated as a predicate $P(x)$ on number.

It relates to the problem-as-function viewpoint by the following characteristic function of $P(x)$:

$$c_P(n) = \begin{cases} 1, & \text{if } P(n) \text{ is valid,} \\ 0, & \text{otherwise.} \end{cases}$$

$$\begin{aligned} \text{Decision Problem} &\Leftrightarrow \text{Subset of } \omega \\ &\Leftrightarrow \text{Predicate on } \omega \end{aligned}$$

Problem I

Is the function $tower(x)$ defined below computable?

$$tower(x) = \underbrace{2^{2^{\cdots^2}}}_{x}.$$

Problem I

Is the function $tower(x)$ defined below computable?

$$tower(x) = \underbrace{2^{2^{\cdots^2}}}_{x}.$$

Theoretically it is computable.

Problem II

Consider the function f defined as follows:

$$f(n) = \begin{cases} 1, & \text{if } n > 1 \text{ and } 2n \text{ is the sum of 2 primes,} \\ 0, & \text{otherwise.} \end{cases}$$

We remark that the Goldbach Conjecture remains unsolved.

Is f computable?

Problem II

Consider the function f defined as follows:

$$f(n) = \begin{cases} 1, & \text{if } n > 1 \text{ and } 2n \text{ is the sum of 2 primes,} \\ 0, & \text{otherwise.} \end{cases}$$

We remark that the Goldbach Conjecture remains unsolved.

Is f computable?

It is a computable problem that we do not know.

Problem III

Consider the function g defined as follows:

$$g(n) = \begin{cases} 1, & \text{if there is a run of exactly } n \text{ consecutive } 7'\text{s} \\ & \text{in the decimal expansion of } \pi, \\ 0, & \text{otherwise.} \end{cases}$$

It is known that π can be calculated by $4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots\right)$.

Is g computable?

Problem III

Consider the function g defined as follows:

$$g(n) = \begin{cases} 1, & \text{if there is a run of exactly } n \text{ consecutive } 7'\text{s} \\ & \text{in the decimal expansion of } \pi, \\ 0, & \text{otherwise.} \end{cases}$$

It is known that π can be calculated by $4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots\right)$.

Is g computable?

We do not know if this problem is computable.

Problem IV

Consider the function i defined as follows:

$$i(x, n, t) = \begin{cases} 1, & \text{if } n \text{ is the machine code of a } C \text{ program } P \\ & \text{and } P(x) \text{ terminates in } t \text{ steps,} \\ 0, & \text{otherwise.} \end{cases}$$

Is i computable?

Problem IV

Consider the function i defined as follows:

$$i(x, n, t) = \begin{cases} 1, & \text{if } n \text{ is the machine code of a } C \text{ program } P \\ & \text{and } P(x) \text{ terminates in } t \text{ steps,} \\ 0, & \text{otherwise.} \end{cases}$$

Is i computable?

The function i is intuitively computable.

Halting Problem (Turing, 1936)

Consider the function h defined as follows:

$$h(x,n) = \begin{cases} 1, & \text{if } n \text{ is the machine code of a } C \text{ program } P \\ & \text{and } P(x) \text{ terminates,} \\ 0, & \text{otherwise.} \end{cases}$$

This is the Halting Problem, a well known undecidable problem.
In other words there does not exist any C program calculating h.

Post Correspondence Problem (Post, 1946)

Suppose $\alpha_1, \ldots, \alpha_N$ and $\beta_1, \ldots, \beta_N$ are finite words over $\{0, 1\}$.

PCP is the problem of deciding if there is some $K > 0$ and some numbers $i_1, \ldots, i_K \in \{1, \ldots, N\}$ such that

$$\alpha_{i_1} \ldots \alpha_{i_K} = \beta_{i_1} \ldots \beta_{i_K}.$$

It is a well-known undecidable problem.

Busy Beaver Problem (Redó, 1962)

The set of n-state TM's with the alphabet $\{\square, 1\}$ is finite.

The n-state busy beaver function BB is defined as follows:

$$BB(n) = \begin{cases} \text{starting from the blank tape containing } \square \text{ solely,} \\ \text{the largest number of 1s on the tape of an} \\ n \text{ state TM after the machine halts.} \end{cases}$$

The function BB is well defined.
It grows faster than any computable function.

Busy Beaver Problem (Redó, 1962)

A related function T counts the time to calculate BB:

$$T(n) = \begin{cases} \text{starting from the blank tape containing } \square \text{ solely,} \\ \text{the largest number of shifts an } n \text{ state TM} \\ \text{has made before the machine halts.} \end{cases}$$

Clearly $T(n) > BB(n)$.

Let P be a program, written in your favourite programming language, that executes without any input.

Let $ol(P)$ be the length of the output of P, if it exists.

The following problem is not computable:

$$\ell(n) \quad = \quad \max\{ol(P) \mid \text{the length of } P \text{ is } n\}\,.$$

In theory we can solve Goldbach Conjecture using following strategy:

- Design an algorithm that tries to refute the conjecture.
- Suppose the algorithm is implemented by an n-state TM. Run the machine until it has made $T(n)$ shifts.

If the machine does not stop after $T(n)$ shifts, we conclude that GC is true. Otherwise GC must be false since a counter example has been found.

In theory we can solve Goldbach Conjecture using following strategy:

- Design an algorithm that tries to refute the conjecture.
- Suppose the algorithm is implemented by an n-state TM. Run the machine until it has made $T(n)$ shifts.

If the machine does not stop after $T(n)$ shifts, we conclude that GC is true. Otherwise GC must be false since a counter example has been found.

Why not try to solve GC using this strategy?

What we have done so far is to convince you that to study computability one might as well look for a theory of

computable functions.

Onwards to the theory of recursive functions!

Reference Book

Nigel Cutland.

An Introduction to Recursive Function Theory. CUP, 1980.

Robert Soare.

Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets. Springer-Verlag, 1987.

Hartley Rogers.

Theory of Recursive Functions and Effective Computability. McGraw-Hill, 1967.