ITP20002 Discrete Mathematics

A Brief Introduction to Halting Problem

Shin Hong

Halting Problem: Overview

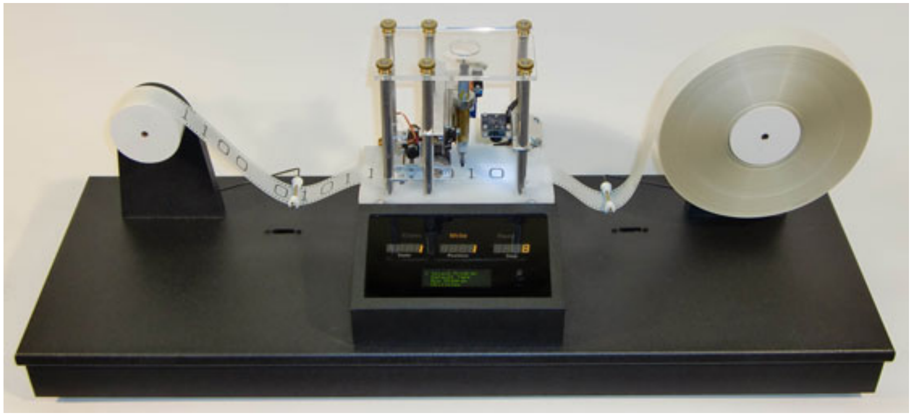
- Halting problem is a famous problem which is proven to have no **algorithm** as its solution (i.e., a *undecidable* problem)

- **Halting problem**: for a given arbitrary **program** and an arbitrary input, determine whether or not the program terminates within finite steps (i.e., halts) when it runs with the input.

- **Theorem**. There is no algorithm that solves the Halting problem.
  - There is no program that always returns the correct determination of whether a given arbitrary program terminates with a given arbitrary input, or not within a finite time.

- **Proof**. takes the proof-by-contradiction strategy.
  1. assume that there is an algorithm for the Halting problem.
  2. shows that the assumption follows algorithms are uncountable.
  3. conclude that the assumption is wrong by the contradiction, thus there is no algorithm that solves the Halting problem.

Historical Aspects


A Turning Machine In The Classic Style http://www.aturingmachine.com/

- In 1936, Alan Turing presented the proof that there is no program that solves the halting problem.

- A mathematical model of a general program called *Turing machine* was first presented in the proof, and thereafter Tuning machine has been widely used as a foundational model to analyze the complexity of computer algorithm in the complexity theories.

Setting: Model Algorithms as C programs

- Suppose that every algorithm can be represented as a C program which receives a finite sequence of characters from STDIN as input, and generates a finite sequence of characters to STDOUT as output
  - In the original proof, an algorithm is modeled as a Tuning machine.

- A C program can be represented as a finite sequence of characters (i.e., a text file).

- It is possible to list valid C programs in a lexicographical order
  - A valid C program is a text file from which a C compiler successfully generates an executable file (i.e., satisfies all C syntaxes; executable).
  - The set of all text files is countable, thus, the set of valid C programs (i.e., a subset) is also countable.

- Let P_i denote the i-th valid C program in the lexicographical order.

Setting: Input

- The set of all inputs of algorithm is countable.
  - An algorithm may have multiple inputs.
  - An input to an algorithm can be represented as a finite sequence of characters.
  - Thus, multiple inputs of an algorithm can be represented as a finite sequence.
  - Thus, the set of all inputs is countable because all finite sequences of a countable set is also countable.

- An input of a C program can be represented as a text file given to STDIN.

- It is possible to list all inputs in a lexicographical order.

- Let I_j denotes the j-th input in the lexicographical order.

Setting: Program Execution

- The execution of a program with an input may or may not terminate in a finite number of steps
  - A program never terminates when it runs with an input if the program is stuck in an infinite loop/recursion.

- Let there be a predicate T for a pair of program P_i and input I_j that returns *true* for iff P_i terminates within a finite number of steps when it runs with I_j

- We can think of a function m which merges a given program and an input as an input (i.e., a sequence of characters)
  - m^{-1} is the inverse function of m, which receives a sequence of characters and returns the corresponding pair of a program and an input

Proof (1/3)

- Theorem. No algorithm solves the Halting problem.

- Lemma. No algorithm solves the Halting problem if there is no program P_H that determines the termination of an arbitrary program with an arbitrary input in a finite number of steps
  - The behavior of P_H for an arbitrary input I_k :
    1. $(P_i, I_j) := m^{-1}(I_k)$
    2. if P_i is determined to be terminated in finite steps for I_j, return "1"
    3. Otherwise, return "0"

- Proof: Prove the lemma by proof-by-contradiction.

  Assume that there exists P_H that always returns the equivalent result with T in a finite time.

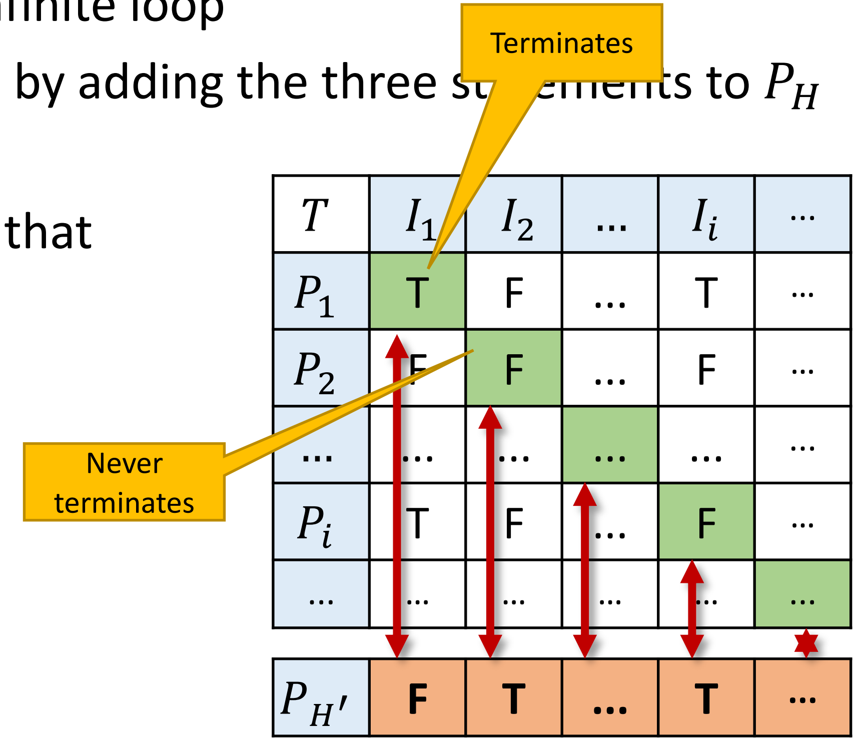
Proof (2/3)

- Define another program $P_{H'}$ using P_H
  - $P_{H'}$ extends P_H (which is a C program) as follows:
    - the behavior of $P_{H'}$ for a given input I_k

```
1.    $tmp$ := $P_H(m(P_k,\ I_k))$
2.    if $tmp$ is "0", return "1"
3.    if $tmp$ is "1", go into an infinite loop
```

  - Note that it is trivial to construct $P_{H'}$ by adding the three statements to P_H

- The assumption and the definition imply that $P_{H'}$ is different from all P_i for $i \in \mathbb{N}$, since $P_i(I_i)$ terminates in a finite time iff $P_H(I_i)$ never terminates in a finite time.
  - As long as P_H behaves as assumed.



T	I_1	I_2	…	I_i	…
P_1	T	F	…	T	…
P_2	F	F	…	F	…
…	…	…	…	…	…
P_i	T	F	…	F	…
…	…	…	…	…	…
$P_{H'}$	**F**	**T**	**…**	**T**	**…**

Proof (3/3)

- The conclusion that there is no $i \in \mathbb{N}$ such that $P_{H'} \equiv P_i$ implies that the set of programs is not countable
  (i.e., no one-to-one correspondence with Natural number).

- The contradiction between the definition of programs (i.e., the set of all programs is countable) and the existence of $P_{H'}$ (i.e., the set of all programs is uncountable) implies that the assumption is not true.

- Thus, there is no program that solves the Halting problem.

- Since there is no program for the Halting problem, we can conclude that there is no algorithm that answers to the Halting problem in all cases.

Implication

- A problem can be proven to be undecidable if there is a finite sequence of steps to transform the problem into the Halting problem.

- We know that it is impossible to write a program that reads the source code of a program and then perfectly determines whether or not the program satisfies a certain property (e.g., free from null-pointer deferecence).
    - If there is a bug finder that aims to say a program has a bug or no bug, we can always come up with a counter-example program for which the bug finder returns a wrong answer
    - Proving the correctness of a given program (verification) may require a unique and create strategy specialized for the given program