

# Lecture T3: Computability



Hilbert

“ Every mathematical problem can be solved. We are convinced of that. After all, one of the things that attracts us most when we apply ourselves to a mathematical problem is precisely that within us we always hear the call: here is the problem, search for the solution, you can find it by pure thought, for in mathematics there is no *ignorabimus*. ”

## A Puzzle ("Post's Correspondence Problem")

Given a set of cards:

- N card types (can use as many of each type as possible).
- Each card has a top string and bottom string.

Example 1:

BAB	A	AB	BA	N = 4
A	ABA	B	B	
0	1	2	3	

Puzzle:

- Is it possible to arrange cards so that top and bottom strings are the same?

## A Puzzle ("Post's Correspondence Problem")

Given a set of cards:

- N card types (can use as many of each type as possible).
- Each card has a top string and bottom string.

Example 1:

BAB	A	AB	BA	N = 4
A	ABA	B	B	
0	1	2	3	

Puzzle:

- Is it possible to arrange cards so that top and bottom strings are the same?

Solution 1.



A	BA	BAB	AB	A
ABA	B	A	B	ABA
1	3	0	2	1

## A Puzzle ("Post's Correspondence Problem")

Given a set of cards:

- N card types (can use as many of each type as needed).
- Each card has a top string and bottom string.

Example 2:

A	ABA	B	A	N = 4
BAB	B	A	B	
0	1	2	3	

Puzzle:

- Is it possible to arrange cards so that top and bottom strings are the same?

## A Puzzle ("Post's Correspondence Problem")

Given a set of cards:

- N card types (can use as many of each type as needed).
- Each card has a top string and bottom string.

Example 2:

A	ABA	B	A	N = 4
BAB	B	A	B	
0	1	2	3	

Puzzle:

- Is it possible to arrange cards so that top and bottom strings are the same?

Solution 2.



5

## PCP Puzzle Contest

S[	X	X	11A	1	[A	]	[	B1	B]	[1A]E
S[11111X][	1X	A	A1	1	[B	]	[	1B	A]	E
0	1	2	3	4	5	6	7	8	9	10

Contest:

- Additional restriction: string must start with 'S'.
- Be the first to solve this puzzle!
  - extra credit for first correct solution
- Check solution by putting STRING ONLY (blanks and line breaks OK) in a file `solution.txt`, then type
 

```
pcp126 < solution.txt
```

Hopeless challenge for the bored:

- Write a program that reads a set of Post cards, and determines whether or not there is a solution.

6

## Overview

Formal language.

- Rigorously express computational problems.
- Ex:  $L = \{ 2, 3, 5, 7, 11, 13, 17, \dots \}$

Abstract machines recognize languages.

- Ex. Is 977 prime? Is 977 in L?
- Essence of computers.

This lecture:

- What is an "algorithm"?
- Is it possible, in principle, to write a program to solve any problem (recognize any language)?

7

## Background

Abstract models of computation help us learn:

- Nature of machines needed to solve problems.
- Relationship between problems and machines.
- Intrinsic difficulty of problems.

As we make machines more powerful, we can recognize more languages.

- Are there languages that no machine can recognize?



- Are there limits on the power of machines that we can imagine?

Pioneering work in the 1930's. (Princeton == center of universe)

- Turing, Church, von Neumann, Gödel. (inspiration from Hilbert)
- Automata, languages, computability, complexity, logic, rigorous definition of "algorithm."

8

# Undecidable Problems

## Hilbert's 10th Problem.

- “Devise a process according to which it can be determined by a finite number of operations whether a given multivariate polynomial has an integral root.”

- Example 1:  $f(x, y, z) = 6x^3yz^2 + 3xy^2 - x^3 - 10$



- Example 2:  $f(x, y) = x^2 + y^2 - 3$



- Example 3:  $f(x, y, z) = x^n + y^n - z^n$



Andrew Wiles, 1995

11

# Undecidable Problems

## Hilbert's 10th Problem.

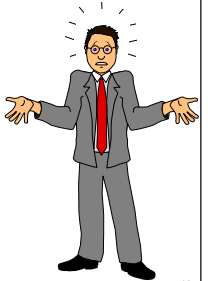
- “Devise a process according to which it can be determined by a finite number of operations whether a given multivariate polynomial has an integral root.”



- Problem resolved in very surprising way. (Matijasevič, 1970)



- How can we assert such a mind-boggling statement?



12

# Undecidable Problems

## Hilbert's 10th Problem.

## Post's Correspondence Problem.

## Halting Problem.

- Write a C program that reads in another program and its inputs, and decides whether or not it goes into an infinite loop.
  - infinite loop often signifies a bug

- Program 1.

```
- 8 6 4 2 4 2 4 2 4 2 4 2 4
- 9 7 5 3 1
```



```
odd.c
...
while (x > 1) {
    if (x > 2)
        x = x - 2;
    else
        x = x + 2;
}
```

14

# Undecidable Problems

## Hilbert's 10th Problem.

## Post's Correspondence Problem.

## Halting Problem.

- Write a C program that reads in another program and its inputs, and decides whether or not it goes into an infinite loop.
  - infinite loop often signifies a bug

- Program 2.

```
- 8 4 2 1
- 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```



```
hailstone.c
...
while (x > 1) {
    if (x % 2 == 0)
        x = x / 2;
    else
        x = 3*x + 1;
}
```

15

## Undecidable Problems

Hilbert's 10th Problem.  
Post's Correspondence Problem.  
Halting Problem.  
Program Equivalence.  
Optimal Data Compression.  
Virus Identification.

Impossible to write C program to solve any of these problem!

19

## TM : As Powerful As TOY Machine

Turing machines are strictly more powerful than FSA, PDA, LBA because of infinite tape memory.

- Power = ability to recognize languages.

Turing machines are at least as powerful as a TOY machine:

- Encode state of memory, PC, etc. onto Turing tape.
- Develop TM states for each instruction.
- Can do because all instructions:
  - examine current state
  - make well-define changes depending on current state

Works for all real machines.

- Can simulate at machine level, gate level, . . . .

20

## TM : Equal Power as TOY and C

Turing machines are equivalent in power to C programs.

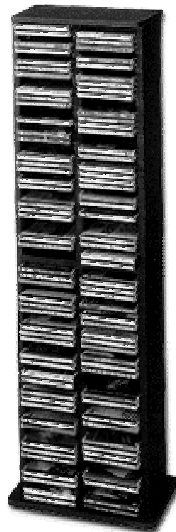
- C program  $\Rightarrow$  TOY program (Lecture A2)
- TOY program  $\Rightarrow$  TM (previous slide)
- TM  $\Rightarrow$  C program (TM simulator, Lecture T2)

Works for all real programming languages.



Is this assumption reasonable?

Assumption: TOY machine and C program have unbounded amount of memory. Otherwise TM is strictly more powerful.



21

## Church-Turing Thesis

Church-Turing thesis (1936):

- Q. Which problems can a Turing machine solve?  
A. Any problem that any real computer can solve.

"Thesis" and not a mathematical theorem.



Implications:

- Provides rigorous definition for **algorithm**.
- Universality among computational models.
  - if a problem can be solved by TM, then it can be solved on EVERY general-purpose computer.
  - if a problem can't be solved by TM, then it can't be solve on ANY physical computer

22

## Evidence Supporting Church-Turing Thesis

### Imagine TM with more power.

- Composition of TM's, multiple heads, more tapes, 2D tapes.
- Nondeterminism.

### Different ways to define "computable."

- TM, RAM machine, circuits, grammar,  $\lambda$ -calculus,  $\mu$ -recursive functions, cellular automata, [Conway's game of life](#).

### Conventional computers.

- ENIAC, TOY, Pentium III, . . .

### New speculative models of computation.

- DNA computers, quantum computers, soliton computers.

23

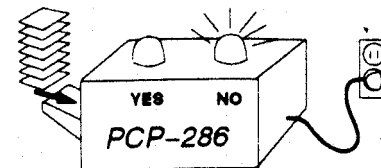
## A More Powerful Computer

### Post machine (PCP-286).

- Input: set of Post cards.
- Output.
  - YES light if PCP is solvable for these cards
  - NO light if PCP has no solution

### PCP is strictly more powerful than:

- Turing machine.
- TOY machine.
- C programming language.
- iMac.
- Any conceivable super-computer.



### Why doesn't it violate Church-Turing thesis?

24

## TM: A General Purpose Machine

### Each TM solves one particular problem.

- Ex: is the integer  $x$  prime?
- Analogy: computer algorithm.
- Similar to ancient special-purpose computers (Analytic Engine) prior to von Neumann stored-program computers.

### Goal: "general purpose machine" that can solve many problems.

- Simulate the operations of any special-purpose TM.
- Analogy: computer than can execute any algorithm.
- How?



25

## Representation of a Turing Machine

### Special-purpose TM consists of 3 ingredients.

- TM program.
- Initial tape contents.
- Current TM state.

26

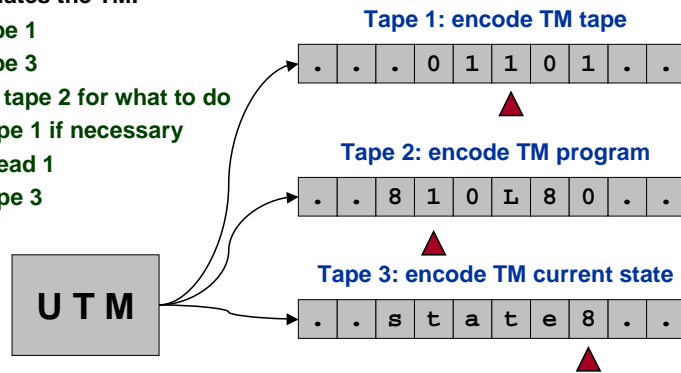
## Universal Turing Machine

### Universal Turing Machine (UTM),

- A specific TM that simulates operations of any TM.

### How to create.

- Encode 3 ingredients of TM using 3 tapes.
- UTM simulates the TM.
  - read tape 1
  - read tape 3
  - consult tape 2 for what to do
  - write tape 1 if necessary
  - move head 1
  - write tape 3







27

## Universal Turing Machine

### Universal Turing Machine (UTM),

- A specific TM that simulates operations of any TM.

### How to create.

- Encode 3 ingredients of TM using 3 tapes.
- UTM simulates the TM.
- Like the fetch-increment-execute cycle of TOY.
  - 
  - 
  - 
- Can reduce 3-tape TM to single tape one.
  - 

28

## Implications of Universal Turing Machine

### Existence of UTM has profound implications.

- "Invention" of general-purpose computer.
  - stimulated development of stored-program computers (von Neumann machines)
- "Invention" of software.
- Universal framework for studying limitations of computing devices.
- Can simulate any machine (including itself)!

29

## Halting Problem

### Halting problem.

- Devise a TM that reads in another TM (encoded in binary) and its initial tape, and determines whether or not that TM would ever reach a yes or no state.
- Write a C program that reads in another program and its inputs, and determines whether or not it goes into an infinite loop.

### Theorem (Alan Turing, 1937). Halting problem is undecidable.

- No TM can solve the halting problem.  
(not possible to write a C program either)

### Proof intuition.

- Self-reference.
- Grelling's paradox.

31

## Warmup: Grelling's Paradox

### Grelling's paradox:

- Divide all adjectives into two categories:
  - autological: self-descriptive
  - heterological: not self-descriptive

autological adjectives
pentasyllabic
awkwardnessful
recherché
...

heterological adjectives
bisyllabic
palindromic
edible
...

- How do we categorize heterological?

32

## Warmup: Grelling's Paradox

### Grelling's paradox:

- Divide all adjectives into two categories:
  - autological: self-descriptive
  - heterological: not self-descriptive

autological adjectives
pentasyllabic
awkwardnessful
recherché
heterological

heterological adjectives
bisyllabic
palindromic
edible

- How do we categorize heterological?
  - suppose it's autological



33

## Warmup: Grelling's Paradox

### Grelling's paradox:

- Divide all adjectives into two categories:
  - autological: self-descriptive
  - heterological: not self-descriptive

autological adjectives
pentasyllabic
awkwardnessful
recherché
<del>heterological</del>

heterological adjectives
bisyllabic
palindromic
edible
heterological

- How do we categorize heterological?
  - suppose it's heterological



34

## Warmup: Grelling's Paradox

### Grelling's paradox:

- Divide all adjectives into two categories:
  - autological: self-descriptive
  - heterological: not self-descriptive

autological adjectives
pentasyllabic
awkwardnessful
recherché
<del>heterological</del>

heterological adjectives
bisyllabic
palindromic
edible
<del>heterological</del>

- How do we categorize heterological?
  - not possible
  - we can't have words with these meanings!  
(or we can't partition adjectives into these two groups)

35

## Halting Problem Proof

Assume the existence of  $\text{Halt}(f,x)$  that takes as input: **any function  $f$  and its input  $x$** , and outputs **yes** if  $f(x)$  halts, and **no** otherwise.

- We prove  $\text{Halt}(f, x)$  can't exist by contradiction.
- Note:  $\text{Halt}(f, x)$  always returns **yes** or **no**. (infinite loop not possible)

```
Halt(f, x)
#define YES 1
#define NO 0

int Halt(char f[], char x[]) {
    if ( ??? )
        return YES;
    else
        return NO;
}
```

function  $f$  and its input  $x$  encoded as strings

36

## Halting Problem Proof

Assume the existence of  $\text{Halt}(f,x)$  that takes as input: **any function  $f$  and its input  $x$** , and outputs **yes** if  $f(x)$  halts, and **no** otherwise.

- Construct program  $\text{Strange}(f)$  as follows:
  - calls  $\text{Halt}(f, f)$
  - halts if  $\text{Halt}(f, f)$  outputs **no**
  - goes into infinite loop if  $\text{Halt}(f, f)$  outputs **yes**
- In other words:
  - if  $f(f)$  does not halt then  $\text{Strange}(f)$  halts
  - if  $f(f)$  halts then  $\text{Strange}(f)$  does not halt

```
Strange(f)
void Strange(char f[]) {
    if (Halt(f, f) == NO)
        return;
    else
        while(1)
            ; // infinite loop
}
```

37

## Halting Problem Proof

Assume the existence of  $\text{Halt}(f,x)$  that takes as input: **any function  $f$  and its input  $x$** , and outputs **yes** if  $f(x)$  halts, and **no** otherwise.

- Construct program  $\text{Strange}(f)$  as follows:
  - calls  $\text{Halt}(f, f)$
  - halts if  $\text{Halt}(f, f)$  outputs **no**
  - goes into infinite loop if  $\text{Halt}(f, f)$  outputs **yes**
- In other words:
  - if  $f(f)$  does not halt then  $\text{Strange}(f)$  halts
  - if  $f(f)$  halts then  $\text{Strange}(f)$  does not halt
- Call  $\text{Strange}$  with **ITSELF** as input.
  - if  $\text{Strange}(\text{Strange})$  does not halt then  $\text{Strange}(\text{Strange})$  halts
  - if  $\text{Strange}(\text{Strange})$  halts then  $\text{Strange}(\text{Strange})$  does not halt
- Either way, a contradiction. Hence  $\text{Halt}(f,x)$  cannot exist.



38

## Consequences

Halting problem is "not artificial."

- Undecidable problem reduced to simplest form to simplify proof.
- Closely related to practical problems.
  - Hilbert's 10th problem, Post's correspondence problem, program equivalence, optimal data compression

How to show new problem  $X$  is undecidable?

- Use fact that Halting problem is undecidable.
- Design algorithm to solve Halting problem, using (alleged) algorithm for  $X$  as a subroutine.
- See Reduction in Lecture T5.

39

## Implications

### Practical:

- Work with limitations.
- Recognize and avoid unsolvable problems.
- Learn from structure.
  - same theory tells us about efficiency of algorithms (see T5)

### Philosophical (caveat: ask a philosopher):

- We "assume" that any step-by-step reasoning will solve any technical or scientific problem.
- "Not quite" says the halting problem.
- Anything that is like (could be) a computer has the same flaw:



## Summary

### What is an algorithm?

- Informally: a step-by-step procedure for solving a problem.
- Formally: a Turing machine.

### Turing's key ideas:

- Computing is the same as manipulating symbols.
  - can encode numbers as strings
- Existence of general-purpose computer (UTM).
  - programmable machine

### What is a general-purpose computer (UTM)?

- Can be "programmed" to implement any algorithm.
- iMac, Dell, Sun UltraSparc, TOY (assuming unlimited memory).

### Is it possible, in principle, to write a program to solve any problem?

- No.