# Lecture T4: Computability

## Overview

**Formal language.**
- Rigorously express computational problems.
- Ex: L = { 2, 3, 5, 7, 11, 13, 17, . . . }

**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)?



## Background

**Abstract models of computation help us learn:**
- Nature of machines needed to solve problems.
- Relationship between problems and machines.
- Intrinsic difficulty of problems.

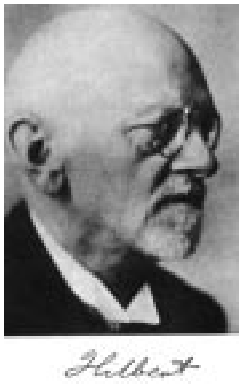
**As we make machines more powerful, we can recognize more languages.**
- Are there languages that no machine can recognize?
- Are there limits on the power of machines that we can imagine?

**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."



## Undecidable Problems

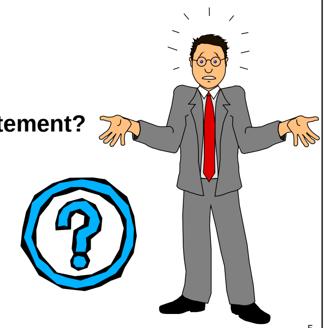
**Hilbert's 10th Problem**
- "*Devise a process according to which it can be determined by a finite number of operations whether a given multivariate polynomial has an integral root.*"
- Example 1: `f(x,y,z)` = $6x^3yz^2 + 3xy^2 - x^3 - 10$
- Example 2: `f(x,y)` = $x^2 + y^2 - 3$



# 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?





# Undecidable Problems

**Hilbert's 10th Problem**

**Post's Correspondence Problem**

- N card types (can use as many of each type as possible).
- Each card has a top string and bottom string.
- Can you arrange cards so that top and bottom strings are the same?
- Example 1:

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

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



# Undecidable Problems

**Hilbert's 10th Problem**

**Post's Correspondence Problem**

- N card types (can use as many of each type as possible).
- Each card has a top string and bottom string.
- Can you arrange cards so that top and bottom strings are the same?
- Example 2:

| A | ABA | B | A |
|---|---|---|---|
| BAB | B | A | B |
| 0 | 1 | 2 | 3 |



# 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.
- 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;
}
```

# 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.
- Such a program would be quite useful for debugging.
  - infinite loop usually signifies a bug



# Undecidable Problems

Hilbert's 10th Problem
Post's Correspondence Problem
Halting Problem
Program Equivalence

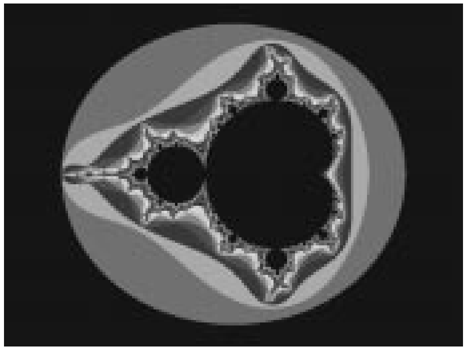
- Do two programs always produce the same output?
- Where's my bug?
- Useful for debugging.



# Undecidable Problems

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

- Find the shortest program to produce a given string or picture.





# 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.

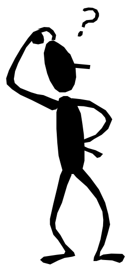
- Can simulate at machine level, gate level, . . . .

# TM : Equal Power as TOY and C

**Turing machines are equivalent in power to C programs.**

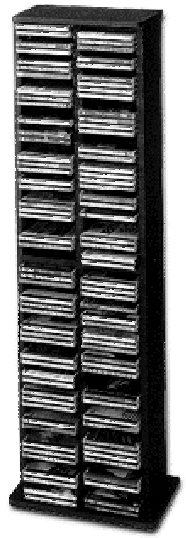
- C program ⇒ TOY program (Lecture A2)
- TOY program ⇒ TM (previous slide)
- TM ⇒ C program (TM simulator, Lecture T2)

**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.





# Church-Turing Thesis

**Church-Turing thesis (1936):**

Q. Which problems can a Turing machine solve?

A. Any problem that any 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 computer



# 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, grammar, $\lambda$-calculus, $\mu$-recursive functions.
- Conway's game of life.

**New speculative models of computation:**

- DNA computers, quantum computers, soliton computers.



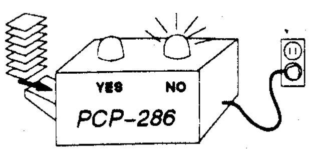
# A More Powerful Computer

**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?**

# TM: A General Purpose Machine

Each TM solves one particular problem.

- Ex: is the integer x prime?
- Analog: 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.
- Analog: computer than can execute any algorithm.
- How?



# Representation of a Turing Machine

Special-purpose TM consists of 3 ingredients.

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



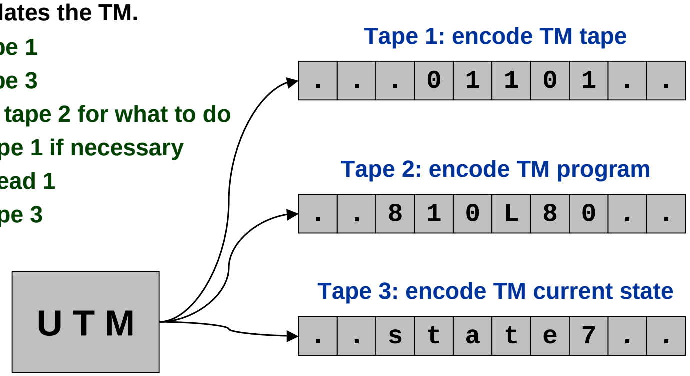
# Universal Turing Machine

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






# 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.

# Implications of Universal Turing Machine

**Existence of UTM has profound implications.**

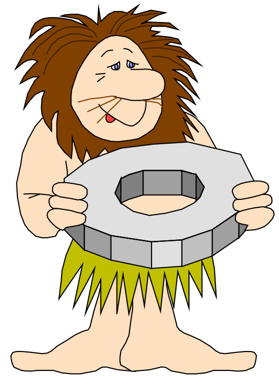
- "Invention" of general-purpose computer.
  - stimulated development of stored-program computers (von Neumann machines)
- Universal framework for studying limitations of general purpose computing devices.
- Can simulate any machine (including itself)!



# 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)
- Universal framework for studying limitations of general purpose computing devices.
- Can simulate any machine (including itself)!





# Halting Problem

**Halting problem.**

- Devise a TM that reads in another TM (encoded in binary) and its initial tape, and determines whether or not it ever reaches 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.

**Halting problem is unsolvable.**

- No TM can solve this problem.
- Not possible to write a C program either.

**We prove that the halting problem is not solvable.**

- Intuition of proof: self-reference.



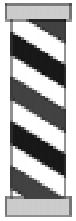
# Some Paradoxes

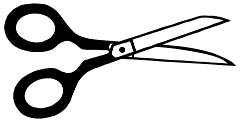
**Lying paradox:**

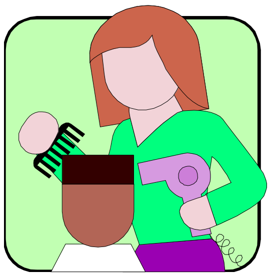
- Divide all statements into two categories: truths and lies.
- How do we classify the statement "I am lying." ?

**Barber paradox:**

- The barber that must cut hair only for all those who don’t cut their own hair.
- Should the barber cut their own hair?







# Warmup: Grelling's Paradox

**Grelling's paradox:**

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

| autological adjectives | heterological adjectives |
| --- | --- |
| pentasyllabic | bisyllabic |
| awkwardnessful | edible |
| recherché | . . . |
| heterological | |

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



# Warmup: Grelling's Paradox

**Grelling's paradox:**

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

| autological adjectives | heterological adjectives |
| --- | --- |
| pentasyllabic | bisyllabic |
| awkwardnessful | edible |
| recherché | . . . |
| ~~heterological~~ | heterological |

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



# Halting Problem Proof

**Assume the existence of Halt(P,x) that takes as input: any program P and its input x, and outputs yes if P(x) halts, and no otherwise.**

- Note: Halt(P, x) always returns yes or no (infinite loop not possible).
- Construct program Strange(P) as follows:
  - calls Halt(P, P)
  - halts if Halt(P, P) outputs no
  - goes into infinite loop if Halt(P, P) outputs yes
- In other words:
  - if P(P) does not halt then Strange(P) halts
  - if P(P) halts then Strange(P) does not halt
- Call Strange with ITSELF as input.
  - if Strange(Strange) does not halt then Strange(Strange) halts
  - if Strange(Strange) halts then Strange(Strange) does not halt
- Either way, a contradiction. Hence Halt(P,x) cannot exist.



# 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 T6.

# 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, step-by-step procedure for solving a problem.**
- **Formally, Turing machine.**

**What is a general-purpose computer?**

- **Capable of simulating any TM.**
- **UTM.**
- **iMac, Dell, Sun UltraSparc, TOY.**
  **(assuming we endow with unlimited memory)**

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

- **No.**