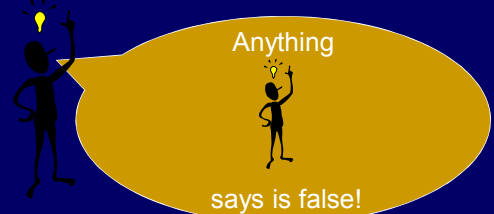


## Wonderful and Crazy Ideas in Theoretical Computer Science and Math

Ryan Williams  
and Virginia Vassilevska  
(Institute for Advanced Study)

[ryanw@ias.edu](mailto:ryanw@ias.edu), [virgi@ias.edu](mailto:virgi@ias.edu)

## Turing's Legacy: The Limits Of Computation.



## The HELLO WORLD assignment

Suppose your teacher tells you:  
Write a JAVA program to output the word  
"HELLO WORLD" on the screen and halt.

Space and time are not an issue.  
The program is for an ideal computer.

PASS for any working HELLO program,  
no partial credit.

## Teacher's Grading Program

The grading program  $G$  must be able to  
take any Java program  $P$  and grade it.

$$G(P) = \begin{cases} \text{Pass, if } P \text{ prints "HELLO WORLD"} \\ \text{Fail, otherwise.} \end{cases}$$

How exactly might such a script work?


What kind of program  
could a student who *hated*  
his/her teacher hand in?




## Nasty Program

```
n:=2;  
While (the number 2n can be written as the sum  
of two primes)  
  n++;  
Print "HELLO WORLD";
```

The nasty program is a PASS if and only if the  
Goldbach conjecture is false.



Despite the simplicity of the HELLO WORLD assignment, there is no program to correctly grade it! This can be proved.



The theory of what can and can't be computed by an ideal computer is called Computability Theory or Recursion Theory.

### The Ideal Memory Model

$\Sigma$  = finite alphabet of symbols  
 Each memory location holds one element of  $\Sigma$

"Abstract" Version: One memory location for each natural number  $0, 1, 2, \dots$

"Practical" Version: Any time you start to run out of memory, the computer contacts the factory. A maintenance person is flown by helicopter and attaches 100 Terabytes of RAM to the computer.

### Computable Functions

Fix any precise programming language, i.e., Java. A program is any finite string of symbols from  $\Sigma$  that a Java interpreter will run (won't give a syntax error)

Recall  $\Sigma^*$  is the set of all strings of symbols.

A function  $f : \Sigma^* \rightarrow \Sigma^*$  is computable if there is a program  $P$  that computes  $f$ , when  $P$  is executed on a computer with ideal memory.

That is, for all strings  $x$  in  $\Sigma^*$ ,  $P(x) = f(x)$ .

### The set of all programs is a countable set!

Fix any precise programming language, i.e., Java. A program is any finite string of symbols from  $\Sigma$  that a Java interpreter will run (won't give a syntax error)

Recall  $\Sigma^*$  is the set of all strings of symbols.

A function  $f : \Sigma^* \rightarrow \Sigma^*$  is computable if there is a program  $P$  that computes  $f$ , when  $P$  is executed on a computer with ideal memory.

That is, for all strings  $x$  in  $\Sigma^*$ ,  $P(x) = f(x)$ .


### The set of all computable functions is also a countable set!

Fix any precise programming language, i.e., Java. A program is any finite string of symbols from  $\Sigma$  that a Java interpreter will run (won't give a syntax error)


Recall  $\Sigma^*$  is the set of all strings of symbols.

A function  $f : \Sigma^* \rightarrow \Sigma^*$  is computable if there is a program  $P$  that computes  $f$ , when  $P$  is executed on a computer with ideal memory.

That is, for all strings  $x$  in  $\Sigma^*$ ,  $P(x) = f(x)$ .



There are "countably many" Java programs. Hence, there are only "countably many" computable functions.



Are there countably many functions from  $\Sigma^*$  to  $\Sigma^*$ ?

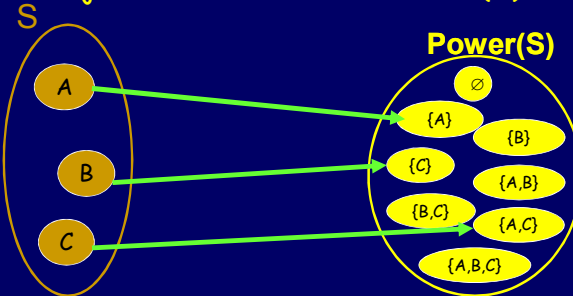
### Power Sets

Let  $S$  be a set.  
 The **power set** of  $S$  is the set of all subsets of  $S$ .

We write the power set as  $\text{Power}(S)$ .

**Proposition:** If  $S$  is finite, then  $\text{Power}(S)$  has cardinality  $2^{|S|}$

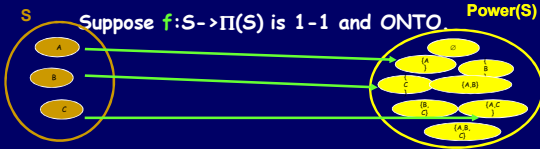
**Theorem:** For every  $S$ , there is no bijection between  $S$  and  $\text{Power}(S)$



Suppose  $f : S \rightarrow \Pi(S)$  is a bijection.

**Theorem:** For every  $S$ , there is no bijection between  $S$  and  $\text{Power}(S)$

Suppose  $f : S \rightarrow \Pi(S)$  is 1-1 and ONTO.



Let  $\text{WEIRD} = \{ x \mid x \in S, x \notin f(x) \}$

There's some  $y$  in  $S$  such that  $f(y) = \text{WEIRD}$

Is  $y$  in  $\text{WEIRD}$ ? YES or NO?

if  $y$  in  $\text{WEIRD}$ , then  $y \in S$  and  $y \notin f(y) \rightarrow \text{WEIRD}$

So  $y$  is not in  $\text{WEIRD}$ .

**Contradiction**

$y \in S$  and  $y \notin \text{WEIRD} = f(y) \dots$  So  $y$  is in  $\text{WEIRD} \dots$

**Theorem:** There are uncountably many functions!

There is a bijection between

- The set of all subsets of  $\Sigma^*$  (the powerset of  $\Sigma^*$ )
- The set of all functions  $f : \Sigma^* \rightarrow \{0,1\}$

Take a subset  $S$  of  $\Sigma^*$ , we map it to the function  $f$  where:

$f(x) = 1 \quad x \text{ in } S$   
 $f(x) = 0 \quad x \text{ not in } S$

## Uncountably many functions.

There is a bijection between

- The set of all subsets of  $\Sigma^*$  (the powerset of  $\Sigma^*$ )
- The set of all functions  $f: \Sigma^* \rightarrow \{0,1\}$

So the set of all  $f: \Sigma^* \rightarrow \{0,1\}$  has the same size as the powerset of  $\Sigma^*$

But  $\Sigma^*$  is countable, so the powerset of  $\Sigma^*$  is uncountable!

(No bijection between  $S$  and  $\text{Power}(S)$ !)

So there are functions from  $\Sigma^*$  to  $\{0,1\}$  that are not computable.

Can we describe an incomputable one?  
Can we describe an interesting, incomputable function?



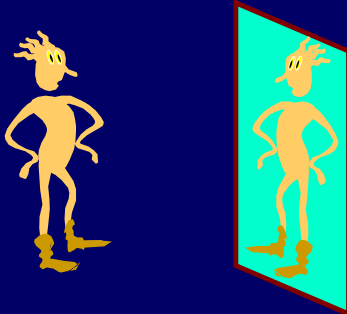
## Notation And Conventions

- Fix any programming language
- When we refer to "program  $P$ " we mean the text of the source code for  $P$
- $P(x)$  is the final output of program  $P$  on input  $x$ , assuming that  $P$  eventually halts

## $P(P)$

It follows from our conventions that  $P(P)$  is the output obtained when we run  $P$  on the text of its own source code.

$P(P)$  ... So that's what I look like



## The Famous Halting Set: $K$

$K$  is the set of all programs  $P$  such that  $P(P)$  halts.

$$K = \{ \text{Program } P \mid P(P) \text{ halts} \}$$

## The Halting Problem

Is there a program HALT such that:

HALT(P) = yes, if P(P) halts  
HALT(P) = no, if P(P) does not halt

## The Halting Problem $K = \{P \mid P(P) \text{ halts} \}$

Is there a program HALT such that:

HALT(P) = yes, if  $P \in K$   
HALT(P) = no, if  $P \notin K$

HALTS decides whether or not any given program is in K.

**THEOREM:** There is no program that can solve the halting problem!  
(Alan Turing 1937)

Suppose a program HALT, solving the halting problem, existed:

HALT(P) = yes, if P(P) halts  
HALT(P) = no, if P(P) does not halt

We will call HALT as a subroutine in a new program called WEIRD.

The Program WEIRD(P):

If HALT(P) then go into an infinite loop.  
Else stop.

<Put text of subroutine HALT here>

Does WEIRD(WEIRD) halt or not?

YES implies HALT(WEIRD) = yes  
but then, WEIRD(WEIRD) will infinite loop

NO implies HALT(WEIRD) = no  
but then, WEIRD(WEIRD) halts

The Program WEIRD(P):

If HALT(P) then go into an infinite loop.  
Else stop.

<Put text of subroutine HALT here>

Does WEIRD(WEIRD) halt or not?

YES implies HALT(WEIRD) = yes  
but then, WEIRD(WEIRD) will infinite loop

**CONTRADICTION**

NO implies HALT(WEIRD) = no  
but then, WEIRD(WEIRD) halts

Turing's argument is just like the **DIAGONALIZATION** argument from the theory of infinities.



	$P_0$	$P_1$	$P_2$	...	$P_j$	...
$P_0$						
$P_1$						
...						
$P_i$						
...						

YES, if  $P_i(P_j)$  halts  
NO, otherwise

	$P_0$	$P_1$	$P_2$	...	$P_j$	...
$P_0$	$d_0$					
$P_1$		$d_1$				
...						
$P_i$					$d_i$	
...						

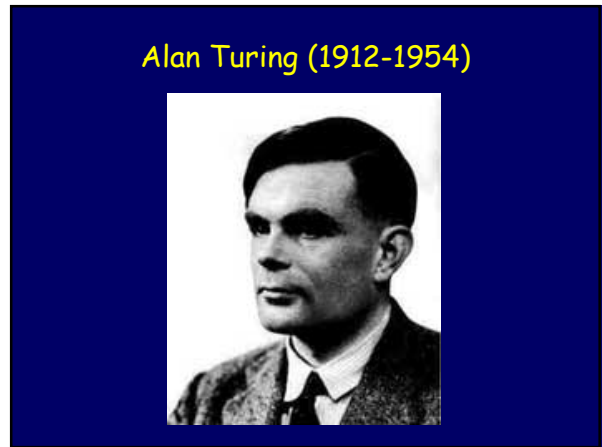
$d_i = \text{HALT}(P_i)$

YES, if  $P_i(P_j)$  halts  
NO, otherwise

	$P_0$	$P_1$	$P_2$	...	$P_j$	...
$P_0$	$d_0$					
$P_1$		$d_1$				
...						
$P_i$	<b>WEIRD</b>				$d_i$	
...						

$d_i = \text{HALT}(P_i)$

WEIRD( $P_i$ ) halts iff  $d_i = \text{NO}$   
The WEIRD row contains the opposite of the diagonal...



Is there a real number that can be described, but not computed?

Consider the real number between 0 and 1, which has a 1 in the  $i^{\text{th}}$  decimal place if  $P_i$  is in  $K$ , and 0 otherwise

## Computability Theory: Vocabulary Lesson

We call a set  $S \subseteq \Sigma^*$  decidable if there is a program  $P$  such that:

$P(x) = \text{yes}$ , if  $x \in S$

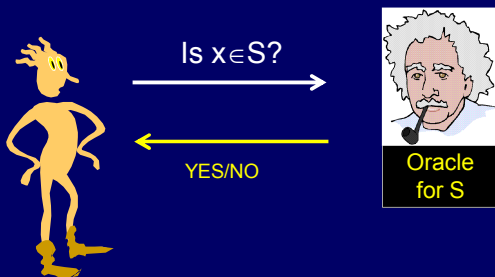
$P(x) = \text{no}$ , if  $x \notin S$

We already know:  $K$  is **undecidable**

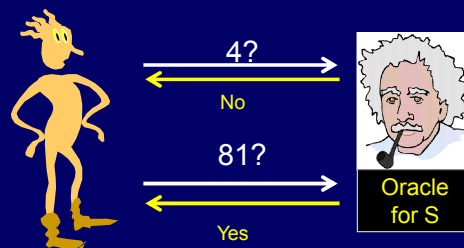
Now that we have established that the Halting Set  $K$  is undecidable, we can use it as a starting point for more “natural” undecidability results.



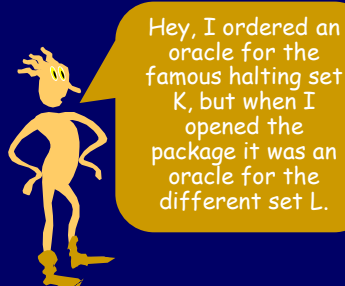
## Oracle For Set $S$



## Example Oracle $S = \text{Odd Naturals}$

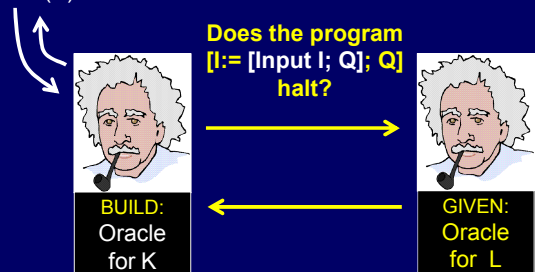


$L = \text{the set of programs that take no input and halt}$



$L = \text{the set of programs that take no input and halt}$

$P = [\text{Input } I; Q]$   
Does  $P(P)$  halt?



Thus, if L were decidable then K would be as well.  
 (If there were a program for L, there'd be one for K, too!)  
 We already know K is not decidable. Therefore L is also not decidable!



HELLO = the set of programs that print HELLO and halt

Does P halt?

Let P' be P with all print statements removed.



**BUILD:**  
Oracle  
for L

Does [P'; Print HELLO] ever print HELLO?



**GIVEN:**  
HELLO  
Oracle

If there were a program for HELLO, then there'd be a program for L.  
 But L is not decidable.  
 So HELLO is not decidable.



**EQUAL = All <P,Q> such that P and Q have identical outputs on all inputs**

Does P equal HELLO?

Let H = [Print HELLO]



**BUILD:**  
HELLO  
Oracle

Are P and H equal?

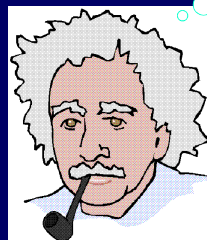


**GIVEN:**  
EQUAL  
Oracle

Halting with input,  
 Halting without input,  
 The "Hello World" assignment, and  
 EQUAL are not decidable.



PHILOSOPHICAL  
INTERLUDE



## CHURCH-TURING THESIS

Any well-defined procedure that can be grasped and performed by the human mind and pencil/paper, can be computed on a conventional digital computer with no bound on its memory.

The Church-Turing Thesis is NOT a theorem. It is a statement of belief about the universe we live in.

Your opinion will be influenced by your religious, scientific, and philosophical beliefs.

## Empirical Intuition

No one has ever given a counter-example to the Church-Turing thesis. That is, no one has given a concrete example of something that humans can compute in a consistent and well defined way, that also can't be programmed on a computer. The thesis is true.

## Mechanical Intuition

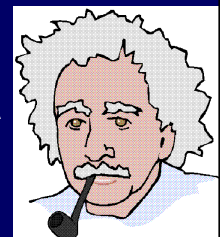
The brain is a machine. The components of the machine obey physical laws. In principle, an entire brain can be simulated step by step on a digital computer. Thus, any thoughts of such a brain can be computed by a simulating computer. The thesis is true.


## Spiritual Intuition

The mind consists of part matter and also part soul. Soul, by its very nature, cannot be reduced to physical laws. Thus, the action and thoughts of the brain cannot be simulated or reduced to simple components and rules. The thesis is false.


There are many other viewpoints you might have concerning the Church-Turing Thesis.

But this ain't philosophy class!





Problems that have no obvious relation to halting, or even to computation can encode the Halting Problem in non-obvious ways. These problems are all undecidable.



Do these theorems about the limits of computation tell us something about the limitations of human thought?

### Self-Reference Puzzle

Write a program that prints its own code out as output.

No calls to the operating system, or to memory external to the program.

(You don't need to use a specific programming language, just your own "English pseudocode" will do.)