

# 15-251

## Great Theoretical Ideas in Computer Science

What does this do?

```
(_,_,_){_/_<=1?(_,_+1,_
_):!(_%_)?(_,_+1,0):_%==_
/_
_&&!_?(printf("%d\t",_/_),_(_,_
_+1,0)):_%>1&&_%<_/_?(_
_,1+
_,_+!(_/_%(_%_))):_<_*_
?(_,_+1,_):0;}main(){_(100,0,0);}
```

## Turing's Legacy: The Limits Of Computation

Lecture 25 (November 18, 2007)



This lecture will change the way you  
think about computer programs...

Many questions which appear easy at first  
glance are impossible to solve in general

## The HELLO assignment

Write a JAVA program to output the word  
"HELLO" 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.

## Grading Script

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

$$G(P) = \begin{cases} \text{Pass, if P prints only the word} \\ \text{"HELLO" and halts.} \\ \text{Fail, otherwise.} \end{cases}$$

How exactly might such a script work?

What does this do?

```
(_,_,_){_/_<=1?(_,+1,_):!(_%_)?(_,+1,0):_%==_/_&&!_?(printf("%d\t",_/_),_(+1,0)):_%>1&&_%</_?(_,+1,_)!(_/_%(_%_)):_<_*?(_,+1,):0;}main(){_(100,0,0);}
```

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



### Nasty Program

```
n:=0;
while (n is not a counter-example
to the Riemann Hypothesis) {
  n++;
}
print "Hello";
```

The nasty program is a PASS if and only if the Riemann Hypothesis is false.

A TA nightmare: Despite the simplicity of the HELLO assignment, there is no program to correctly grade it!

And we will prove this.



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



From the last lecture:

Are all reals describable? NO  
Are all reals computable? NO

We saw that  
computable  $\Rightarrow$  describable  
but do we also have  
describable  $\Rightarrow$  computable?

The "grading function" we just described is not computable! (We'll see a proof soon.)

## Computable Function

Fix a finite set of symbols,  $\Sigma$   
Fix a precise programming language, e.g., Java

A program is any finite string of characters that is syntactically valid.

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

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

Hence: countably many computable functions!

There are only countably many Java programs.

Hence, there are only countably many computable functions.



## Uncountably Many Functions

The functions  $f: \Sigma^* \rightarrow \{0,1\}$  are in 1-1 onto correspondence with the subsets of  $\Sigma^*$  (the powerset of  $\Sigma^*$ ).

Subset  $S$  of  $\Sigma^* \Leftrightarrow$  Function  $f_S$

$x$ in $S$	$\Leftrightarrow$	$f_S(x) = 1$
$x$ not in $S$	$\Leftrightarrow$	$f_S(x) = 0$

Hence, the set of all  $f: \Sigma^* \rightarrow \{0,1\}$  has the same size as the power set of  $\Sigma^*$ , which is uncountable.

Countably many computable functions.

Uncountably many functions from  $\Sigma^*$  to  $\{0,1\}$ .

Thus, most functions from  $\Sigma^*$  to  $\{0,1\}$  are not computable.



Can we explicitly describe an uncomputable function?

Can we describe an interesting uncomputable function?



## Notation And Conventions

Fix a single programming language (Java)

When we write program  $P$  we are talking about the text of the source code for  $P$

↓  
 $P(x)$  means the output that arises from running program  $P$  on input  $x$ , assuming that  $P$  eventually halts.

$P(x) = \perp$  means  $P$  did not halt on  $x$

## The meaning of $P(P)$

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

## The Halting Set $K$

Definition:

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

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

## The Halting Problem

Is there a program  $\text{HALT}$  such that:

$\text{HALT}(P) =$  yes, if  $P(P)$  halts  $\Rightarrow P \in K$   
 $\text{HALT}(P) =$  no, if  $P(P)$  does not halt  $\equiv P \notin K$

*deciding membership in  $K$*

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

Suppose a program  $\text{HALT}$  existed that solved the halting problem.

$\text{HALT}(P) =$  yes, if  $P(P)$  halts  
 $\text{HALT}(P) =$  no, if  $P(P)$  does not halt

We will call  $\text{HALT}$  as a subroutine in a new program called  $\text{CONFUSE}$ .

## CONFUSE

```
CONFUSE(P)
{ if (HALT(P))
  then loop forever;    //i.e., we dont halt
  else exit;           //i.e., we halt
  // text of HALT goes here
}
```

Does  $\text{CONFUSE}(\text{CONFUSE})$  halt?

## CONFUSE

```
CONFUSE(P)
{ if (HALT(P))
  then loop forever;    //i.e., we dont halt
  else exit;           //i.e., we halt
  // text of HALT goes here }
```

Suppose  $\text{CONFUSE}(\text{CONFUSE})$  halts:  
then  $\text{HALT}(\text{CONFUSE}) = \text{TRUE}$   
 $\Rightarrow \text{CONFUSE}$  will loop forever on input  $\text{CONFUSE}$

Suppose  $\text{CONFUSE}(\text{CONFUSE})$  does not halt  
then  $\text{HALT}(\text{CONFUSE}) = \text{FALSE}$

**CONTRADICTION**

Alan Turing (1912-1954)

Theorem: [1937]

There is no program to solve the halting problem



Turing's argument is essentially the reincarnation of Cantor's Diagonalization argument that we saw in the previous lecture.



All Programs (the input)

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

Programs (computable functions) are countable, so we can put them in a (countably long) list

All Programs (the input)

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

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

All Programs (the input)

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

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

CONFUSE( $P_i$ ) halts iff  $d_i = \text{no}$   
(The CONFUSE function is the negation of the diagonal.)  
Hence CONFUSE cannot be on this list.

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



Consider the real number  $R$  whose binary expansion has a 1 in the  $j^{\text{th}}$  position iff the  $j^{\text{th}}$  program halts.



## Proof that $R$ cannot be computed

Suppose it is, and program FRED computes it. then consider the following program:

```

MYSTERY(program text P)
  for j = 0 to forever do {
    if (P == Pj)
      then use FRED to compute  $j^{\text{th}}$  bit of  $R$ 
      return YES if (bit == 1), NO if (bit == 0)
  }

```

**MYSTERY solves the halting problem!**

From the last lecture:

Are all reals describable? NO  
Are all reals computable? NO

We saw that

computable  $\Rightarrow$  describable  
but do we also have  
describable  $\Rightarrow$  computable?

## Computability

### A Vocabulary lesson

## Computability Theory: Vocabulary Lesson

We call a set  $S \subseteq \Sigma^*$  decidable or recursive 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: the halting set  $K$  is undecidable

## Decidable and Computable

Subset  $S$  of  $\Sigma^*$   $\Leftrightarrow$  Function  $f_S$

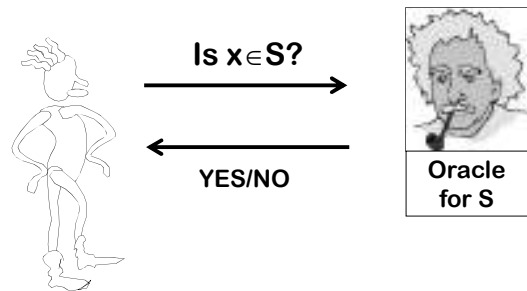
$x \in S$	$\Leftrightarrow$	$f_S(x) = 1$
$x \text{ not in } S$	$\Leftrightarrow$	$f_S(x) = 0$

Set  $S$  is decidable  $\Leftrightarrow$  function  $f_S$  is computable

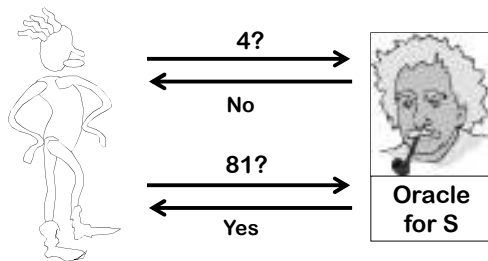
Sets are “decidable” (or undecidable), whereas functions are “computable” (or not)

## Oracles and Reductions

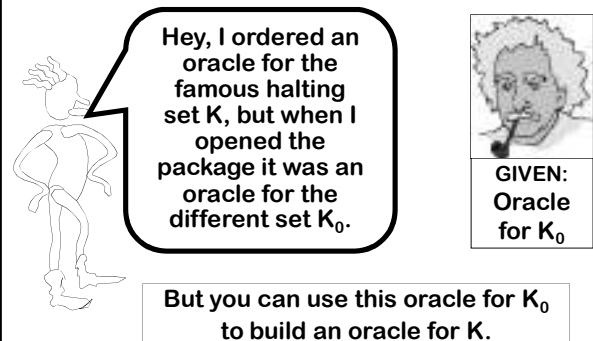
### Oracle For Set S



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

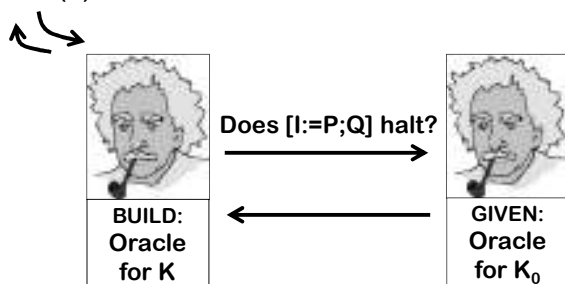


$K_0$  = the set of programs that take no input and halt



$K_0$  = the set of programs that take no input and halt

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



We've reduced the problem of deciding membership in K to the problem of deciding membership in  $K_0$ .

Hence, deciding membership for  $K_0$  must be at least as hard as deciding membership for K.

Thus if  $K_0$  were decidable then  $K$  would be as well.

We already know  $K$  is not decidable, hence  $K_0$  is not decidable.



HELLO = the set of programs that print hello and halt

Does  $P$  halt?



Let  $P'$  be  $P$  with all print statements removed. (assume there are no side effects)

Is  $[P'; \text{print HELLO}]$  a hello program?



BUILD:  
Oracle  
for  $K_0$



GIVEN:  
HELLO  
Oracle



Hence, the set HELLO is not decidable.

EQUAL = All  $\langle P, Q \rangle$  such that  $P$  and  $Q$  have identical output behavior on all inputs

Is  $P$  in set HELLO?



Let  $H_I = [\text{print HELLO}]$



BUILD:  
HELLO  
Oracle



GIVEN:  
EQUAL  
Oracle

Are  $P$  and  $H_I$  equal?



Halting with input, Halting without input, HELLO, and EQUAL are all undecidable.

## Diophantine Equations

Does polynomial  $4X^2Y + XY^2 + 1 = 0$  have an integer root? I.e., does it have a zero at a point where all variables are integers?

$D = \{\text{multivariate integer polynomials } P \mid P \text{ has a root where all variables are integers}\}$

Famous Theorem:  $D$  is undecidable!  
[This is the solution to Hilbert's 10<sup>th</sup> problem]



Hilbert



### Resolution of Hilbert's 10<sup>th</sup> Problem: Dramatis Personae



Martin Davis, Julia Robinson, Yuri Matiyasevich (1982)

### Polynomials can Encode Programs

There is a computable function  
F: Java programs that take no input  $\rightarrow$   
Polynomials over the integers

Such that  
program P halts  $\Leftrightarrow$  F(P) has an integer root

D = the set of all integer  
polynomials with integer roots

Does program P  
halt?



F(P) has  
integer root?



PHILOSOPHICAL  
INTERLUDE



### CHURCH-TURING THESIS

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



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

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

...mileage may vary

### **Empirical Intuition**

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

### **Mechanical Intuition**

The brain is a machine. The components of the machine obey fixed 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.

### **Quantum Intuition**

The brain is a machine, but not a classical one. It is inherently quantum mechanical in nature and does not reduce to simple particles in motion. Thus, there are inherent barriers to being simulated on a digital computer. The thesis is false. However, the thesis is true if we allow quantum computers.