15-251

Great Theoretical Ideas in Computer Science

What does this do?

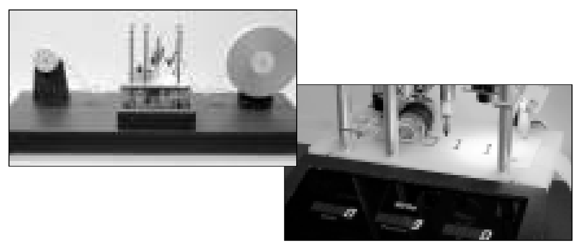
```
_(__,___,____){___/__<=1?_(__,___+1,___
_):!(___%__)?_(__,___+1,0):___%__==___
/
__&&!____?(printf("%d\t",___/__),_(__,_
__+1,0)):___%__>1&&___%__<___/__?_(
__,1+
___,____+!(___/__%(___%__))):___<__*__
?_(__,___+1,____):0;}main(){_(100,0,0);}
```

What does this do?

```
#include <stdio.h>
main(t,_,a)char *a;{return!0<t?t<3?main(-79,-13,a+main(-87,1-_,
main(-86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,main(-94,-
27+t,a)&&t==2?_<13? main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-
72?main(_,t,"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+,/w{%+,/w#q#n+,/#
{l,+,/n{n+,/+#n+,/#\ ;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;#q#n'){)#}w'){){nl]'/+#n';d}rw'
i;# \ ){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K {rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i; :{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \ ;;{nl'-
{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'rdq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/") :t<-50?_==*a?putchar(31[a]):main(-
65,_,a+1):main((*a=='/')+t,_,a+1)
:0<t?main(2,2,"%s"):*a=='/'||main(0,main(-61,*a, "!ek;dc i@bK'(q)-
[w]*%n+r3#l,{}:\nuwloca-O;m .vpbks,fxntdCeghiry"),a+1);}
```

Turing's Legacy: The Limits Of Computation

Lecture 24 (November 11, 2010)



From the last lecture:

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

We saw that

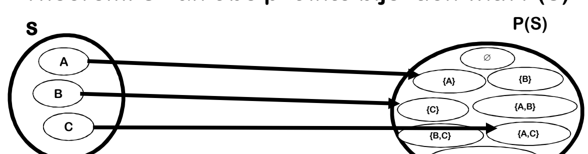
computable $\Rightarrow$ describable

but do we also have

describable $\Rightarrow$ computable?

We'll answer this question today…

Theorem: S can't be put into bijection with P(S)



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

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

Since f is onto, exists $y \in S$ such that $f(y) = \text{CONFUSE}_f$.
Is y in CONFUSE_f?

Let $CONFUSE_f$ = { x | x $\in$ S, x $\notin$ f(x) }

Let y $\in$ S such that f(y) = $CONFUSE_f$.

Is y in $CONFUSE_f$?

Suppose y in CONFUSE

Then y $\notin$ f(y) — by def of CONFUSE

But f(y) = CONFUSE — By choice of y

$\Rightarrow$ y $\notin$ CONFUSE

Contradiction!

Suppose y not in CONFUSE

So y $\notin$ f(y) — hey, f(y) = CONFUSE

$\Rightarrow$ y in CONFUSE — By defn of CONFUSE

Contradiction!

Theorem: a set S can't be put into bijection with its power set P(S)

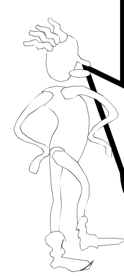
Computable Function

Fix a finite set of symbols, Σ
Fix a precise programming language, e.g., Java

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

A function f : $\Sigma^* \to \Sigma^*$ is computable if there is a program P that when executed on an ideal computer, computes f.
That is, for all strings x in Σ^*, f(x) = P(x).

Hence: countably many computable functions!



There are only countably many Java programs.

Hence, there are only countably many computable functions.

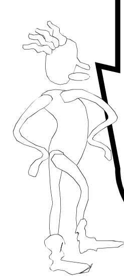
Uncountably Many Functions

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

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

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

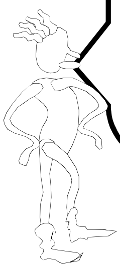
Hence, the set of all f:$\Sigma^* \to \{0,1\}$ has the same size as the power set of Σ^*, which is uncountable.



Countably many computable functions.

Uncountably many functions from Σ^* to {0,1}.

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



Can we explicitly describe an uncomputable function?

The HELLO assignment

Write a JAVA program to output the words "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.

Grading Script

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

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

How exactly might such a script work?

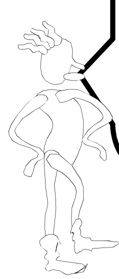
What does this do?

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

Nasty Program

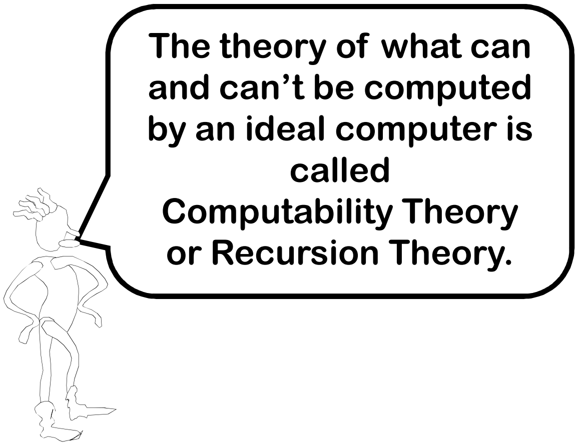
```
n:=0;
while (n is not a counter-example
     to the Riemann Hypothesis) {
     n++;
}
print "Hello";
```

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.



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 Problem

Is there a program HALT such that:

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

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

Suppose a program HALT existed that solved the halting problem.

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 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 CONFUSE(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 CONFUSE(CONFUSE) halts:
then HALT(CONFUSE) = TRUE
⇒ CONFUSE will loop forever on input CONFUSE

Suppose CONFUSE(CONFUSE) does not halt
then HALT(CONFUSE) = FALSE
⇒ CONF

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)

All Programs

	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)

All Programs

	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)

All Programs

	P_0	P_1	P_2	…	P_j	…
P_0	d_0					
P_1		d_1				
…			…			
P_i				d_i		
…					…	Let d_i = HALT(P_i)

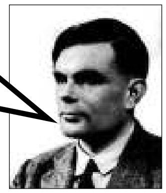
CONFUSE(P_i) halts iff d_i = no
(The CONFUSE function is the negation of the diagonal.)

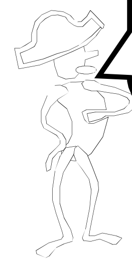
Hence CONFUSE cannot be on this list.

Alan Turing (1912-1954)

Theorem: [1937]

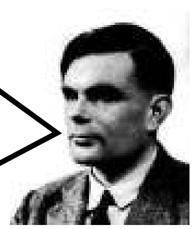
There is no program to solve the halting problem





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^{th} position iff the j^{th} program halts on input itself.



Proof that R cannot be computed

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

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

MYSTERY solves the halting problem!

The Halting Set K

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

K = { Java P | P(P) halts }

Computability Theory: Vocabulary Lesson

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

P(x) = yes, if $x \in S$
P(x) = no, if $x \notin S$

Today we saw: the halting set K is undecidable

No program can decide membership in K

Decidable and Computable

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

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

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

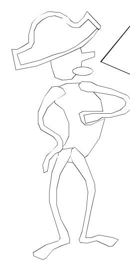
Sets are “decidable” (or “undecidable”),
functions are “computable” (or not)

Computable vs. Enumerable

Computability Theory: Some More Vocabulary

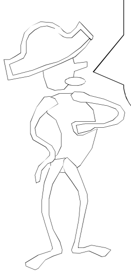
We call a set of strings $S \subseteq \Sigma^*$ enumerable
or recursively enumerable (r.e.)
if there is a program P such that:

1. P prints an (infinite) list of strings.
2. Any element on the list should be in S.
3. Each element in S appears after a finite amount of time.



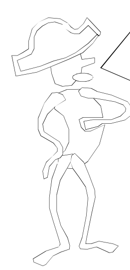
Can you enumerate all strings in Σ^* ?

```
for n = 0 to infinity do
   for all strings s of length n do
      print(s)
```



Can you enumerate all (syntactically valid) Java programs?

```
for n = 0 to infinity do
   for all strings s of length n do
      if check-syntax(s) then
         print(s)
```

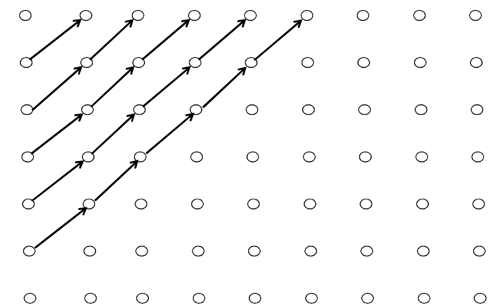


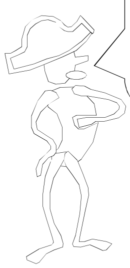
Is the halting set K enumerable?

Enumerating K

```
Enumerate-K   {
  for n = 0 to forever   {
    for W = all strings of length < n do   {
      if W(W) halts in n steps then output W;
    }
  }
}
```

(x,y) = check if program P_x halts on y^{th} input





K is not decidable but it is enumerable!

Let K’ = { Java P | P(P) does not halt}

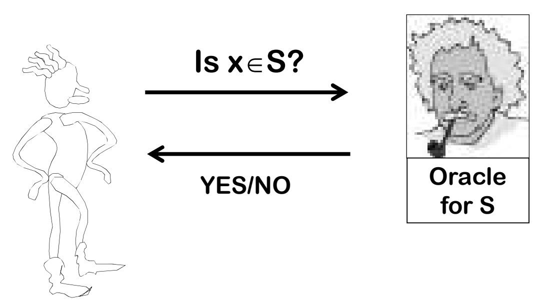
Is K’ enumerable?

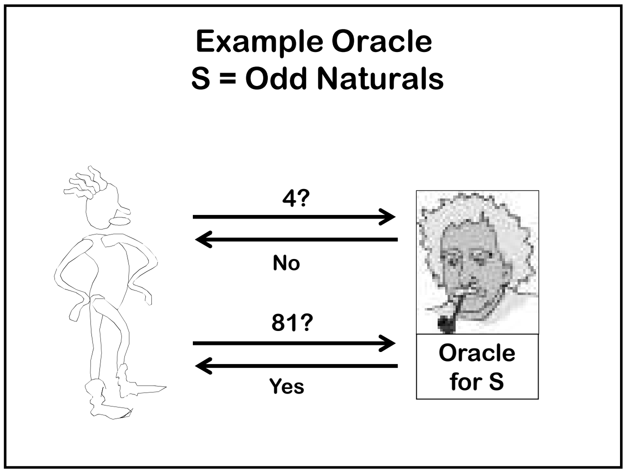
No! If both K and K’ are enumerable, then K is decidable.

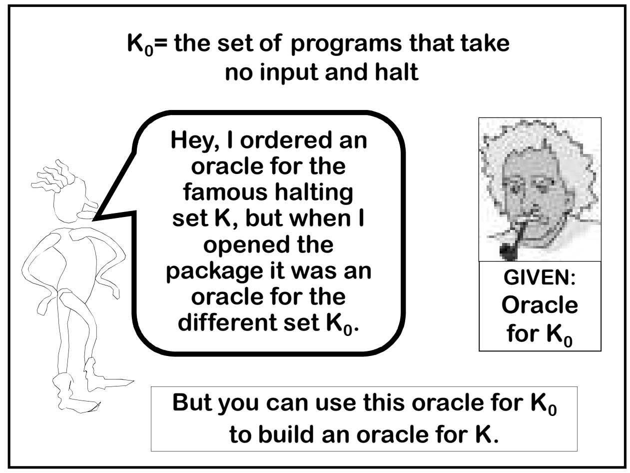
Run both enumeration programs in parallel. Every P will be eventually output in one of these, can use to decide in P in K.

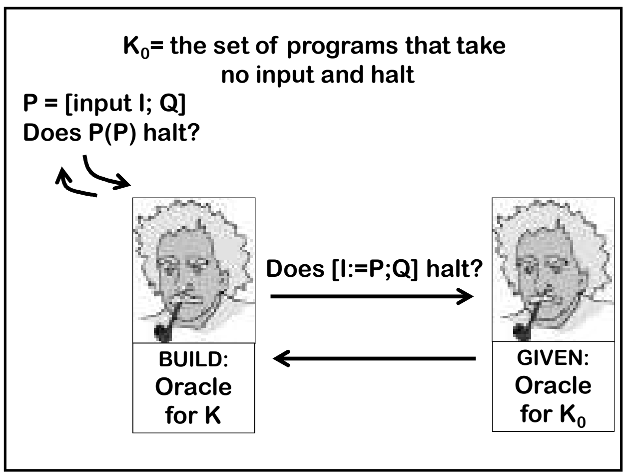
Oracles and Reductions

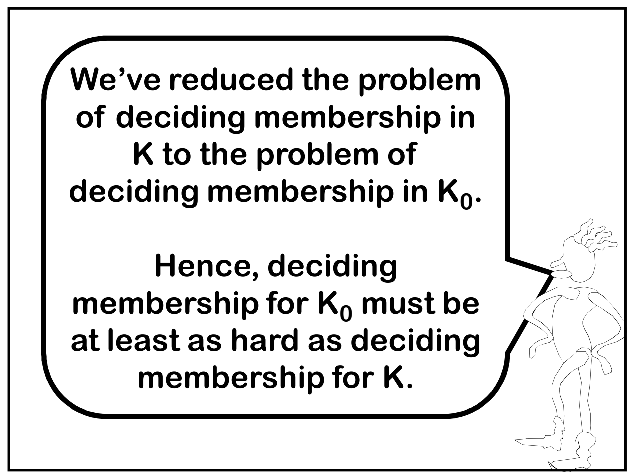
Oracle For Set S

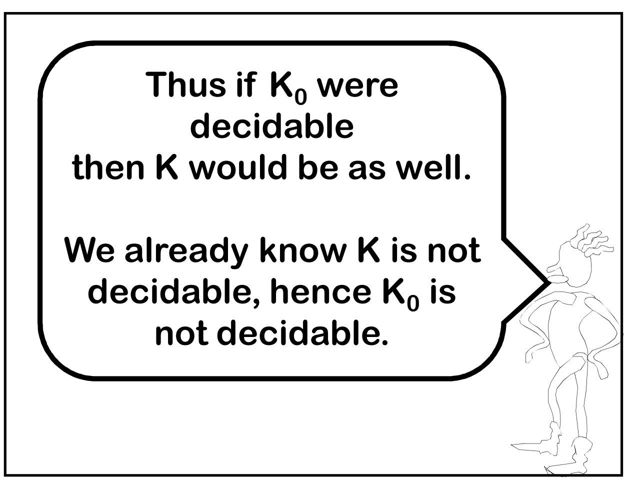


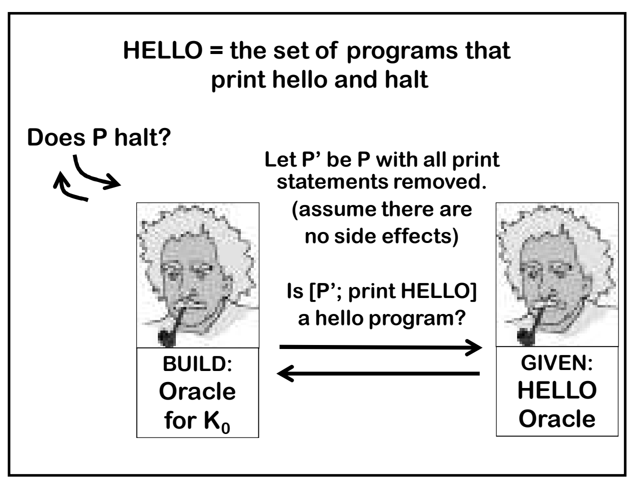

Example Oracle
S = Odd Naturals
4?
No
81?
Yes
Oracle for S


K_0= the set of programs that take no input and halt
Hey, I ordered an oracle for the famous halting set K, but when I opened the package it was an oracle for the different set K_0.
GIVEN: Oracle for K_0
But you can use this oracle for K_0 to build an oracle for K.


K_0= the set of programs that take no input and halt
P = [input I; Q]
Does P(P) halt?
Does [I:=P;Q] halt?
BUILD: Oracle for K
GIVEN: Oracle for K_0


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'; print HELLO] a hello program?
BUILD: Oracle for K_0
GIVEN: HELLO Oracle

Hence, the set HELLO is not decidable.

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

Is P in set HELLO?

Let HI = [print HELLO]

BUILD:
HELLO Oracle

Are P and HI equal?

GIVEN:
EQUAL Oracle

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 = {multivariate integer polynomials P s.t. P has root where all variables are integers}

Famous Theorem: D is undecidable!

[This is the solution to Hilbert's 10th problem]

Hilbert

Resolution of Hilbert's 10th Problem

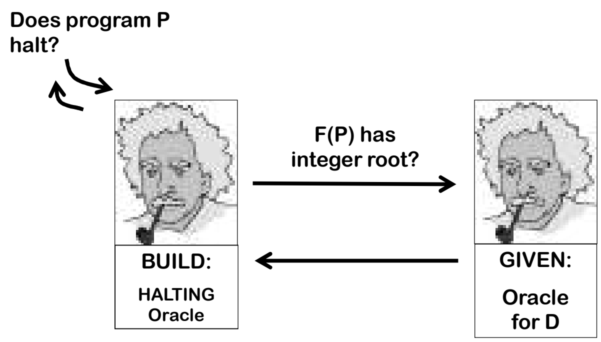
Martin Davis, Julia Robinson, Yuri Matiyasevich (in 1982)

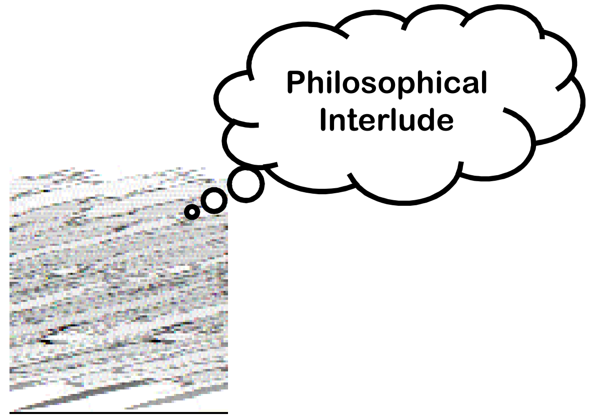
Polynomials can Encode Programs

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





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…

…your mileage may vary

Empirical Intuition

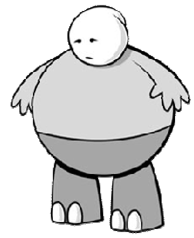
No one has ever given a counter-example to the Church-Turing thesis. I.e., no one has given a concrete example of something humans compute in a consistent and well defined way, but that can't be programmed on a computer. The thesis is true.

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.



Here's What You Need to Know…

Computable and Decidable

Halting Problem
- Definition
- Halting set K
- Proof that K is not decidable
- Diagonalization (again!)

Enumerable
- Definition
- K is enumerable

Oracles
- Reductions (super important!)