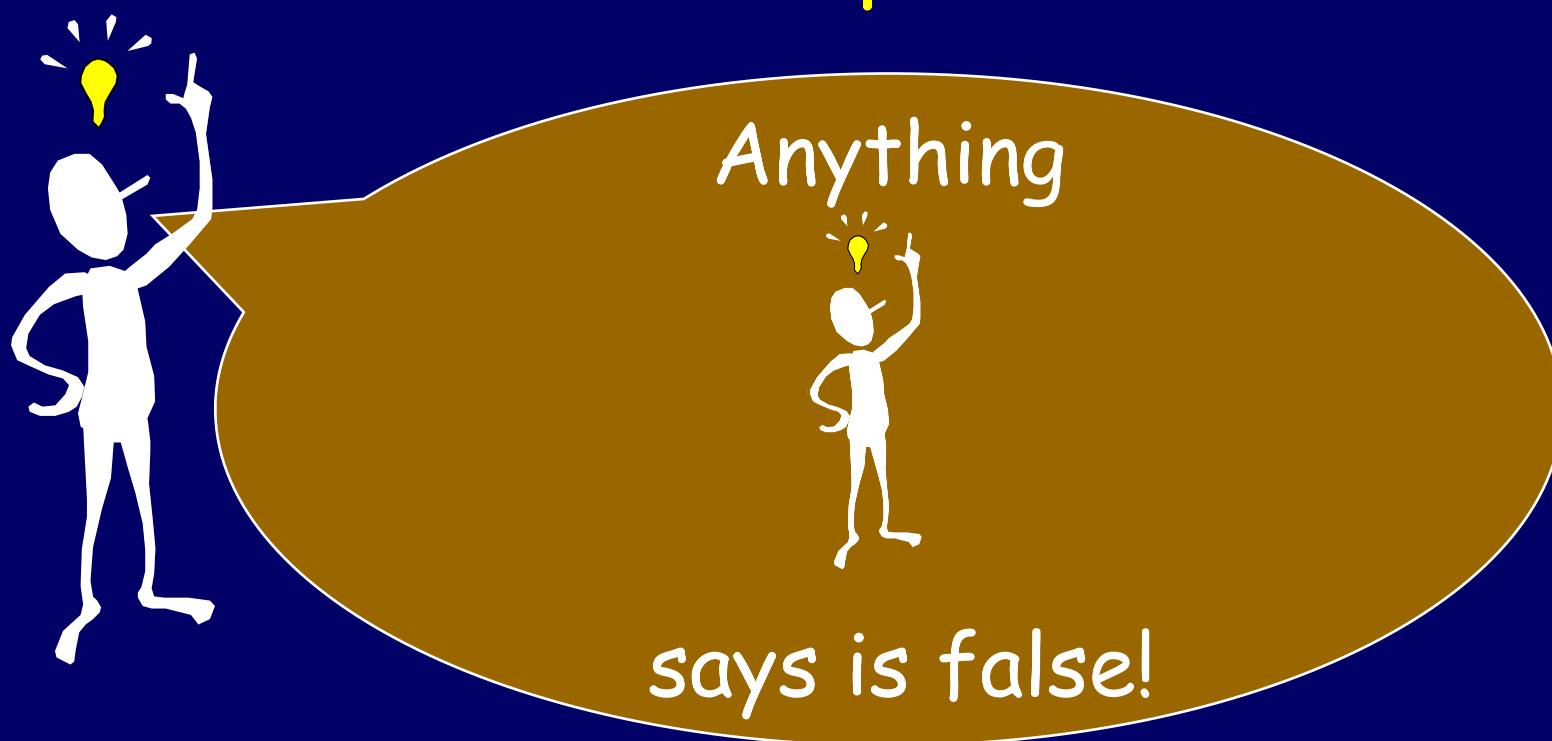
Great Theoretical Ideas In Computer Science

John Lafferty | CS 15-251 | Fall 2006
Lecture 26 | November 28, 2006 | Carnegie Mellon University

Turing's Legacy: The Limits Of Computation.



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

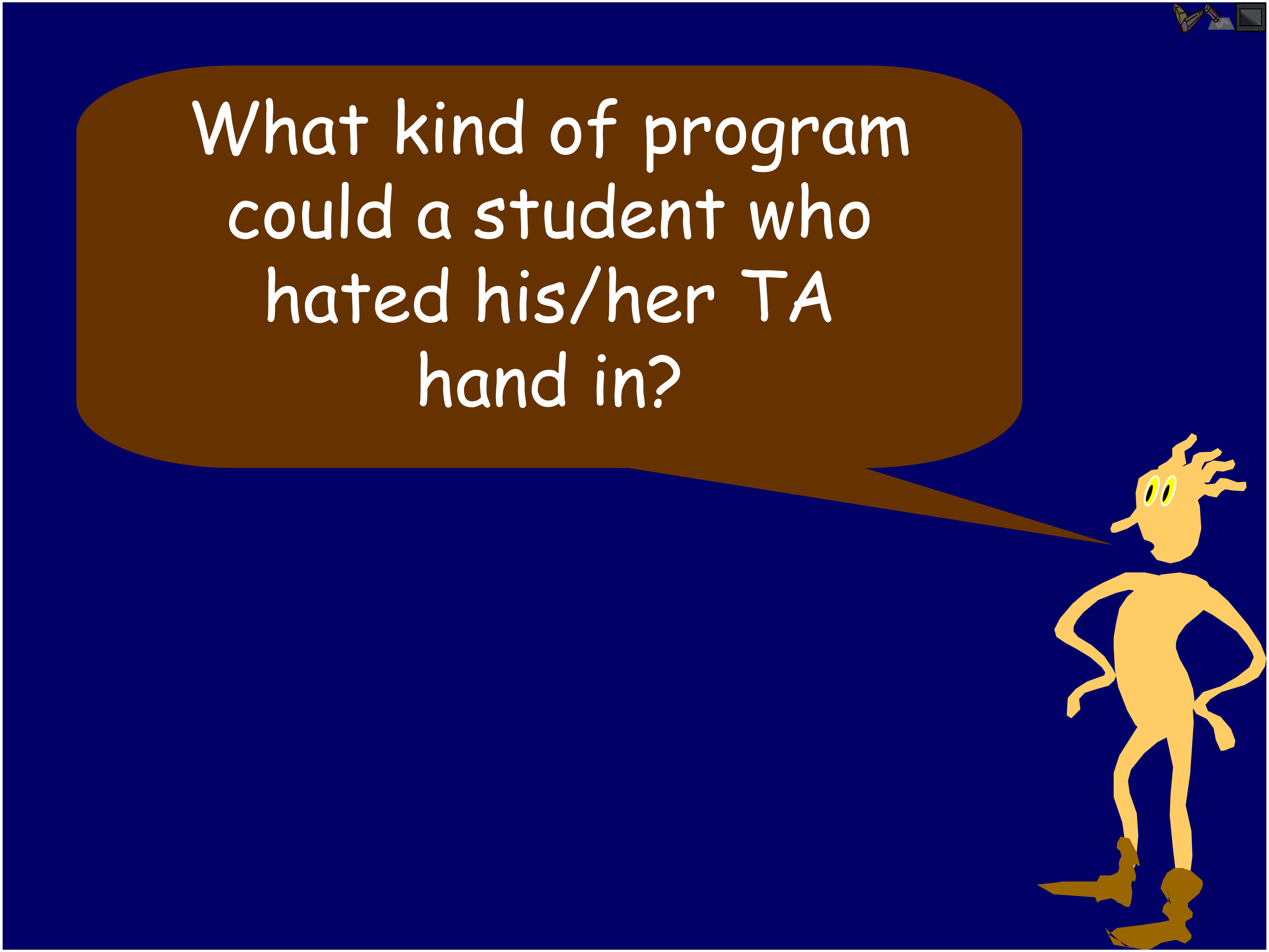
The grading script G must be able to take any Java program P and grade it.

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

How exactly might such a script work?

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);}
```


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

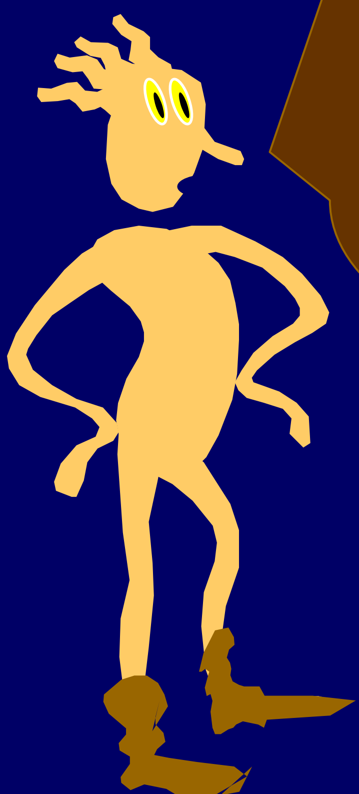
Nasty Program

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

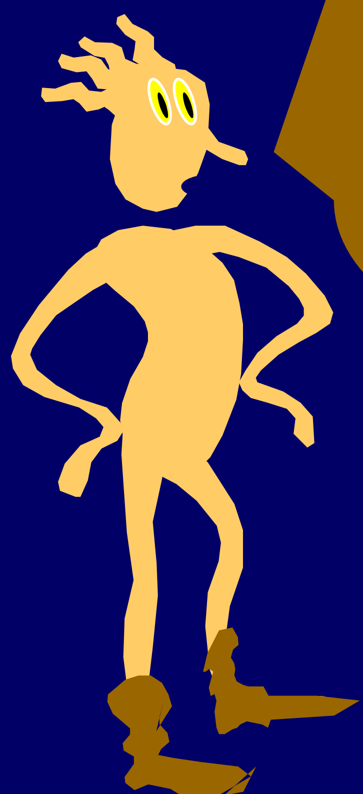
The nasty program is a PASS if and only if the Riemann Hypothesis is false.

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

And we will <u>prove</u> this.



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

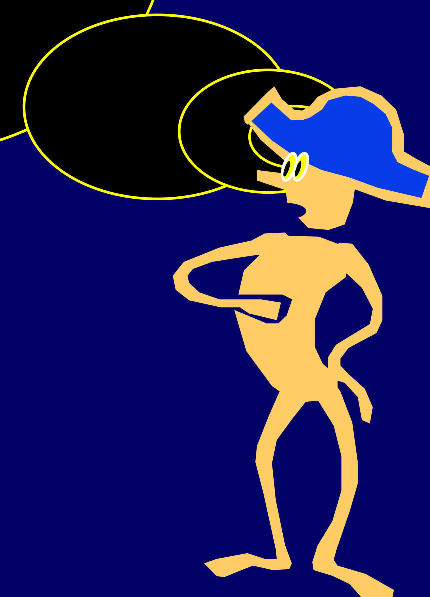


From Lecture 25:

Are all reals describable? **NO**

Are all reals computable? **NO**

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

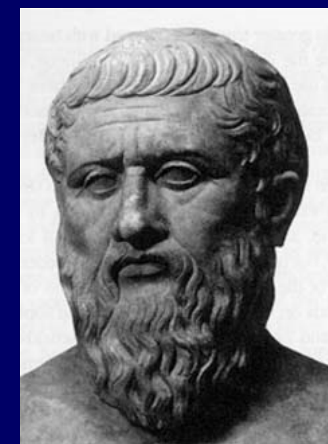


The "grading function" we just described is not computable! (We'll see a proof soon.)

Infinite RAM Model

Platonic Version:

One memory location for each natural number 0, 1, 2, …



Aristotelian Version:

Whenever you run out of memory, the computer contacts the factory. A maintenance person is flown by helicopter and attaches 100 Gig of RAM and all programs resume their computations, as if they had never been interrupted.

Computable Function

Fix any finite set of symbols, Σ.
Fix any 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 <u>computable</u> if there is a program P that when executed on an ideal computer, computes f.

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

Computable Function

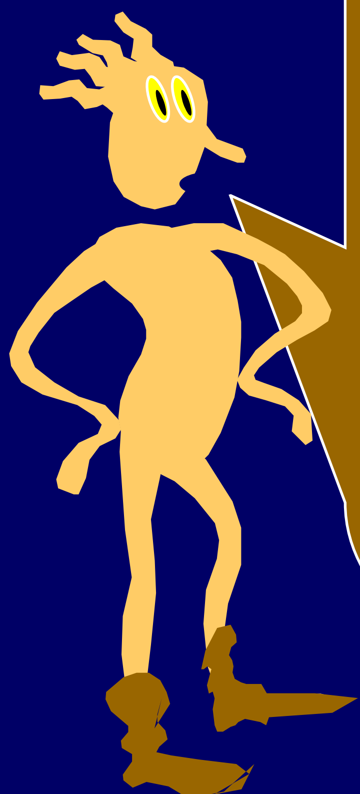
Fix any finite set of symbols, Σ.
Fix any precise programming language, e.g., Java.

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: $\Sigma^* \rightarrow \{0,1\}$ 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	$\Leftrightarrow$	$f_S(x) = 0$

Uncountably many functions

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

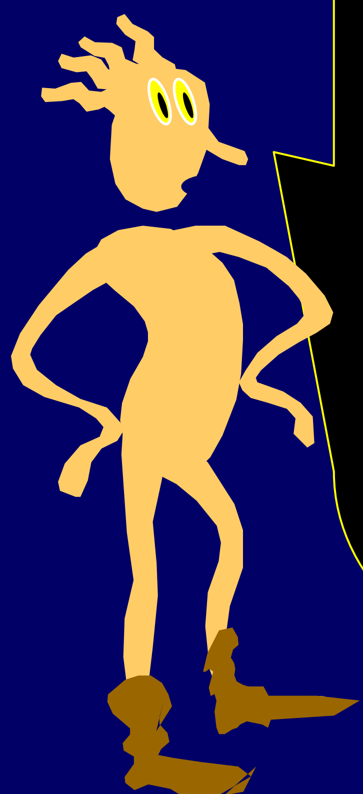
Hence, the set of all $f: \Sigma^* \rightarrow \{0,1\}$ has the same size as the power set of Σ^*.

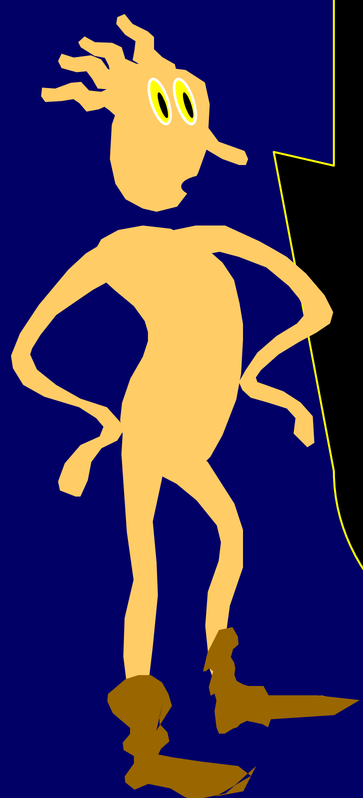
And since Σ^* is countably infinite, its power set is uncountably infinite.

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?

Can we describe an <u>interesting</u> 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

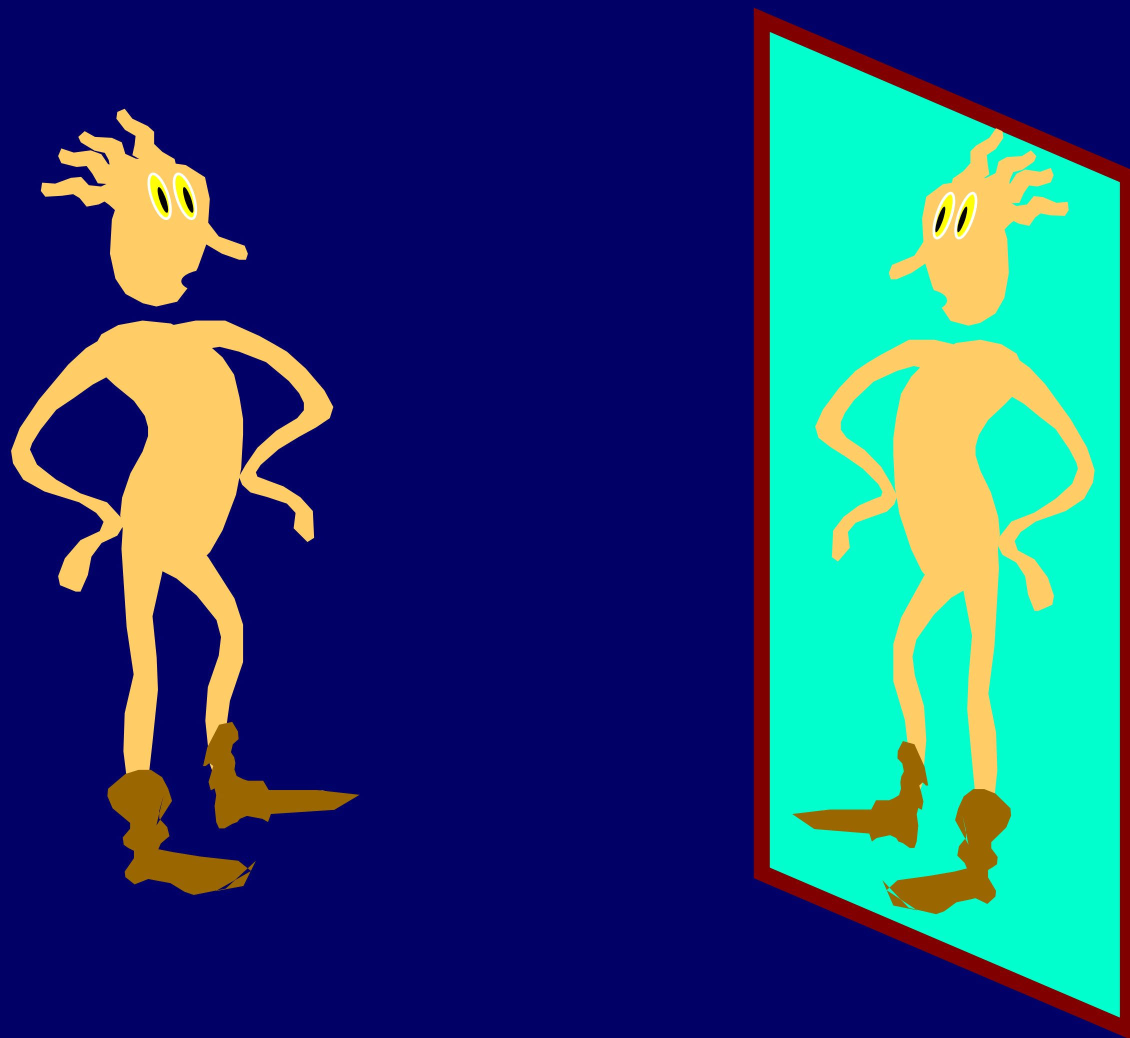
$P(x)$ means the output that arises from running program P on input x, assuming that P eventually halts.

$P(x) = \perp$ 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.

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



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 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:

$$\mathrm{HALT}(P) = \begin{cases} \text{yes, if } P \in K \\ \text{no, if } P \notin K \end{cases}$$

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

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

$\Rightarrow$ CONFUSE will loop forever on input CONFUSE

Suppose C… not halt

…

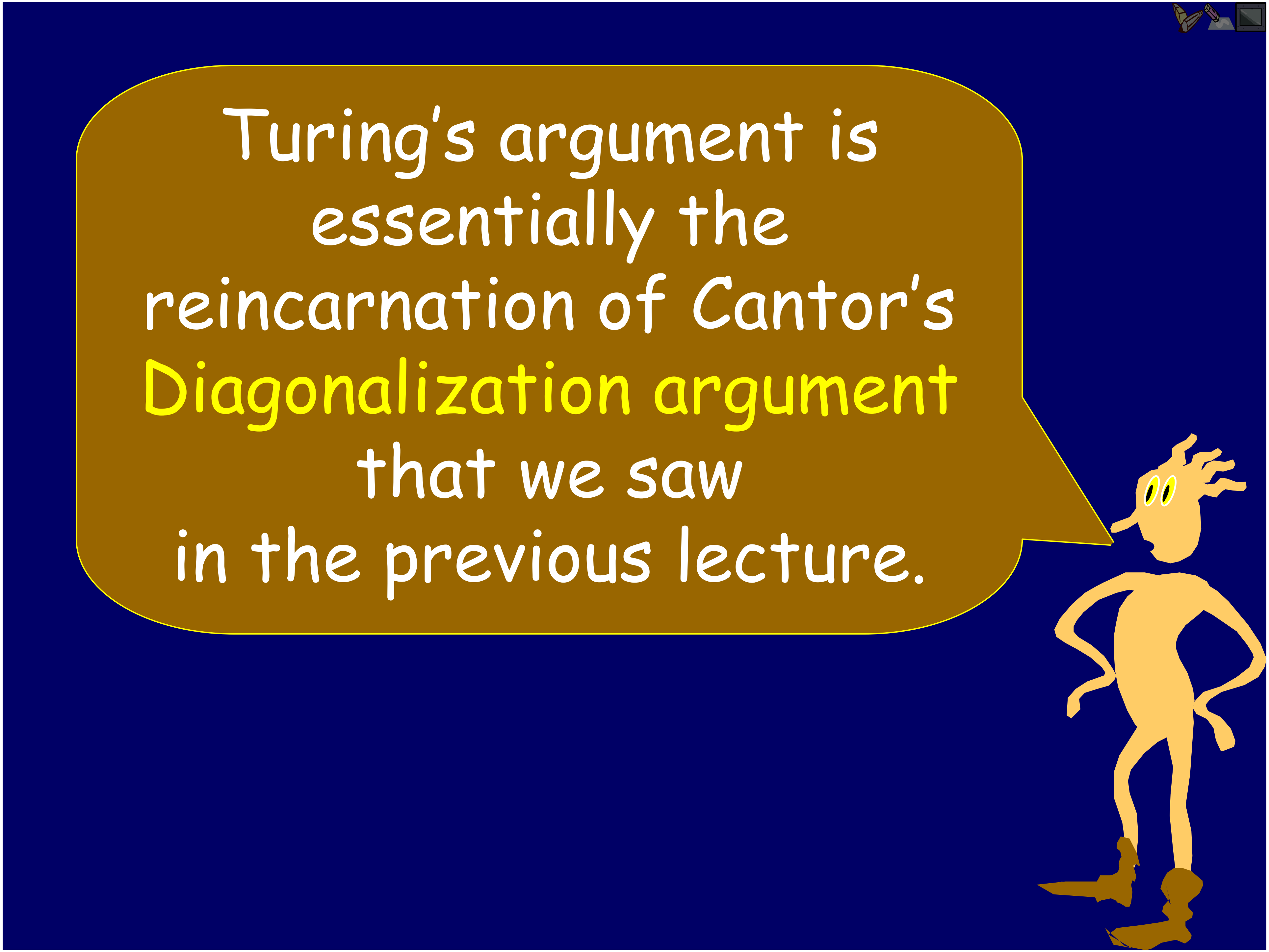
CONTRADICTION

$\Rightarrow$ CONFUSE will halt on input CONFUSE

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.)

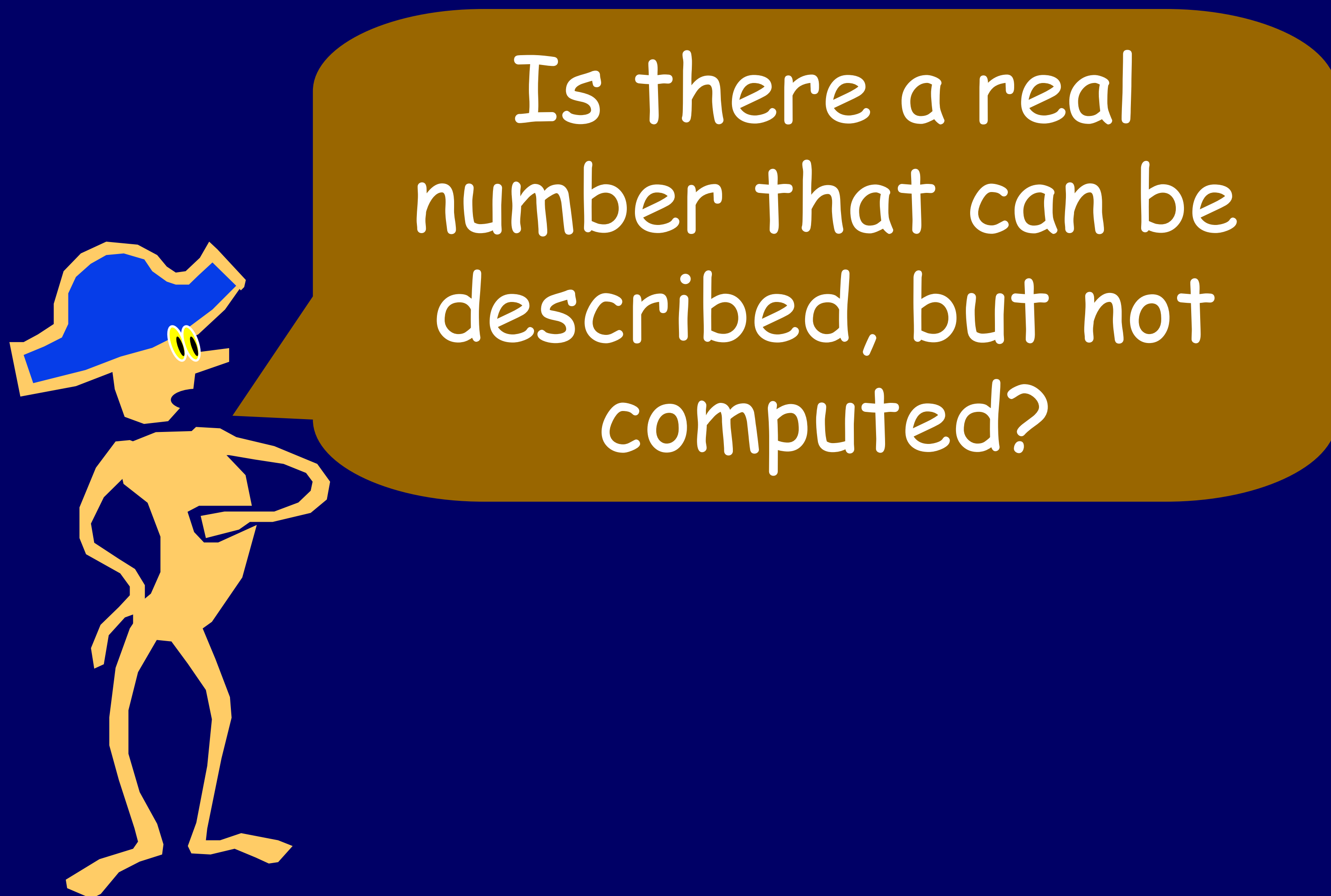
Hence CONFUSE cannot be on this list.

Alan Turing (1912-1954)

Theorem: [1937]

There is no program to solve the halting problem

From last lecture:



Consider the real number R_K whose binary expansion has a 1 in the j^{th} position iff $P_j \in K$ (i.e., if the j^{th} program halts).

Proof that R_K 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 == P_j)
       then use FRED to compute j^th bit of R_K
    return YES if (bit == 1), NO if (bit == 0)
  }
```

MYSTERY solves the halting problem!

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

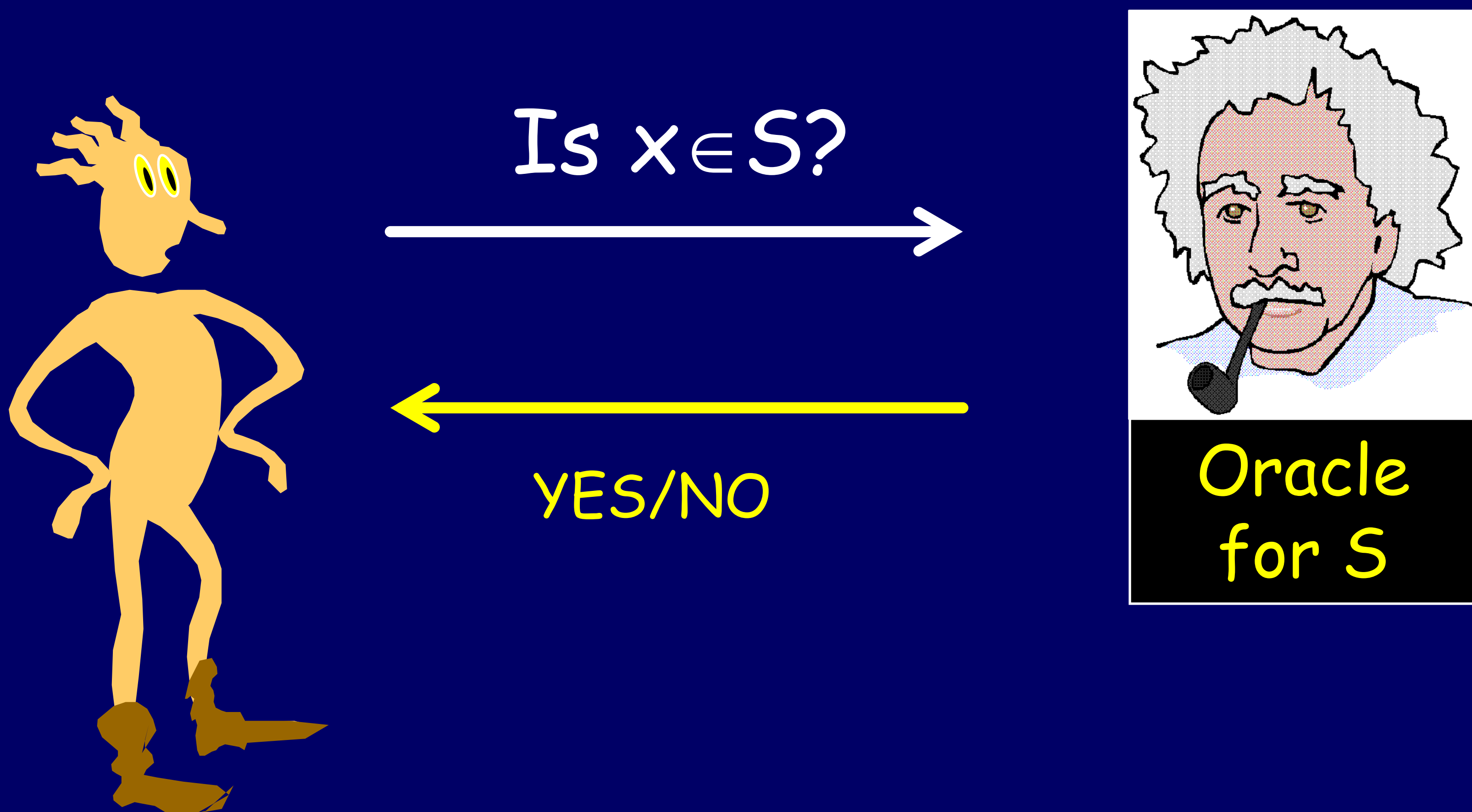
Subset S of Σ^*	$\Leftrightarrow$	Function f_S
x in S	$\Leftrightarrow$	$f_S(x) = 1$
x not in S	$\Leftrightarrow$	$f_S(x) = 0$

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

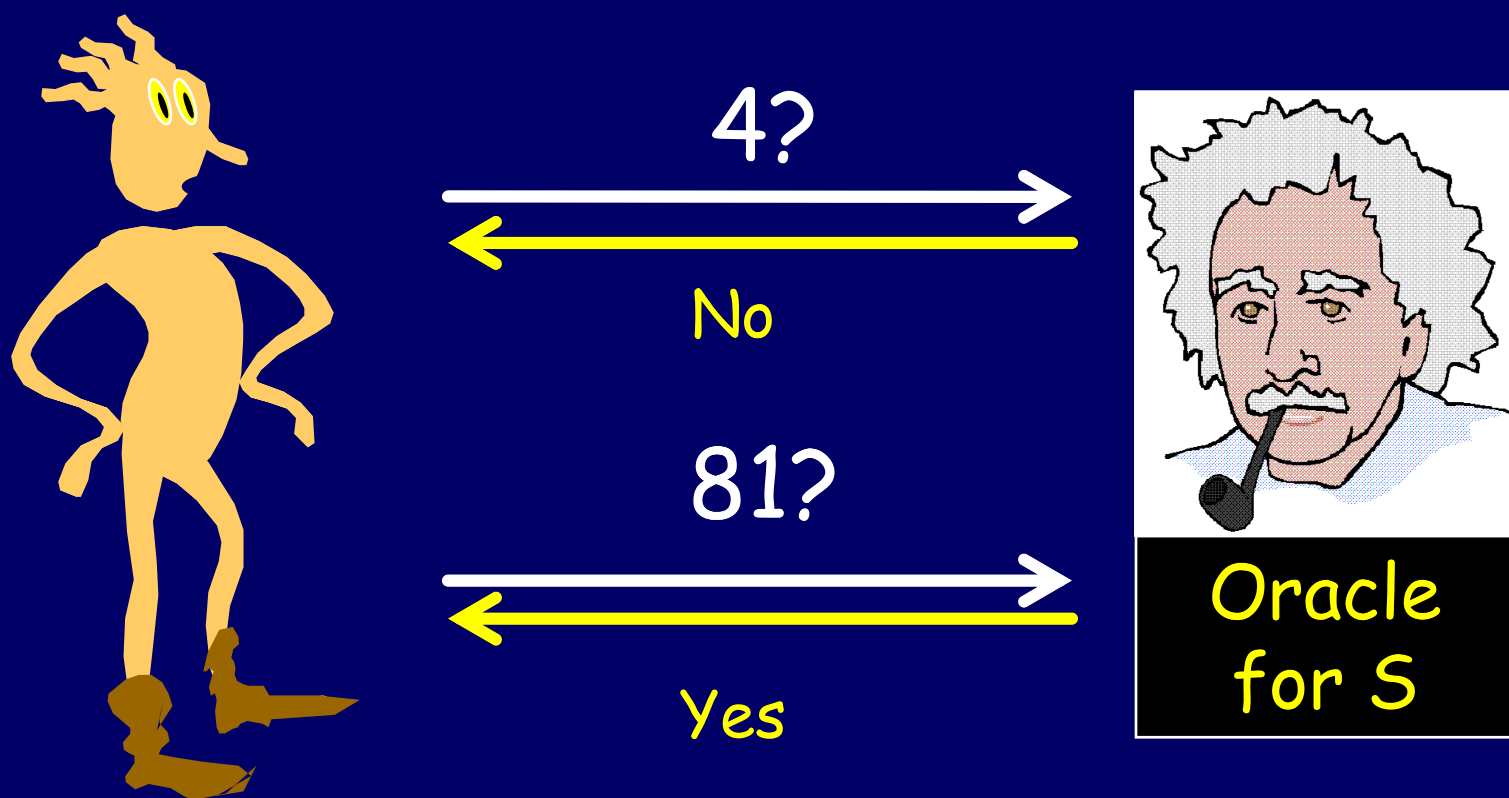
Sets are "decidable" (or undecidable), whereas functions are "computable" (or not)

Oracles and Reductions

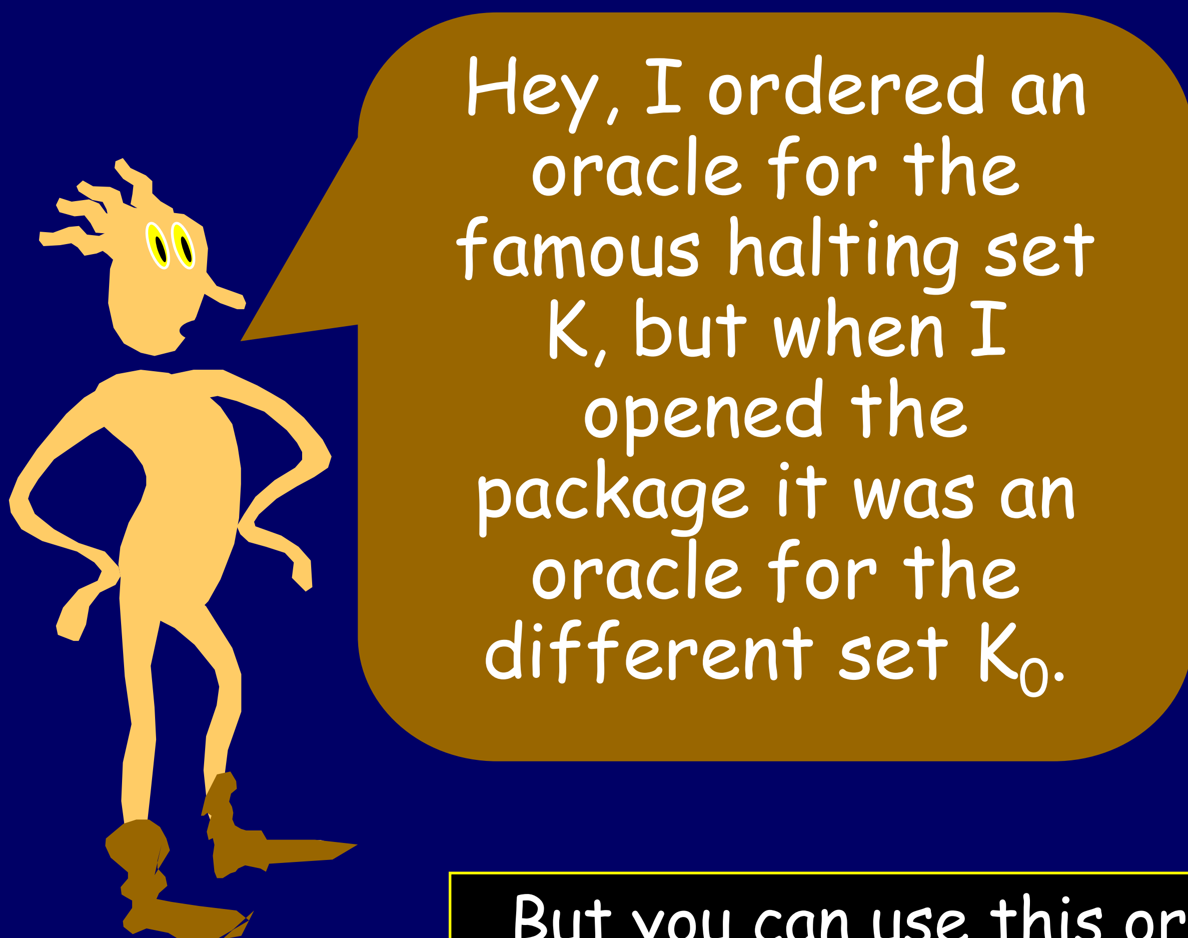
Oracle For Set S

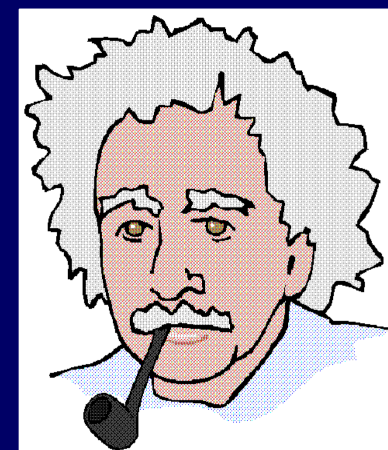


Example Oracle
S = Odd Naturals



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



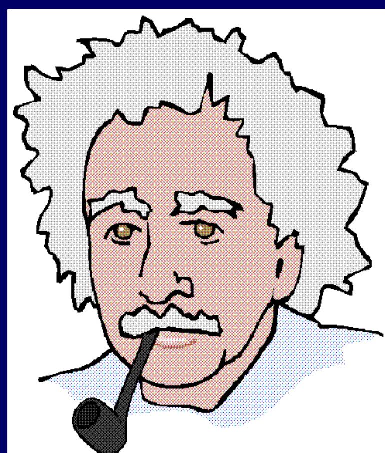


GIVEN:
Oracle for K_0

But you can use this oracle for K_0 to build an oracle for K.

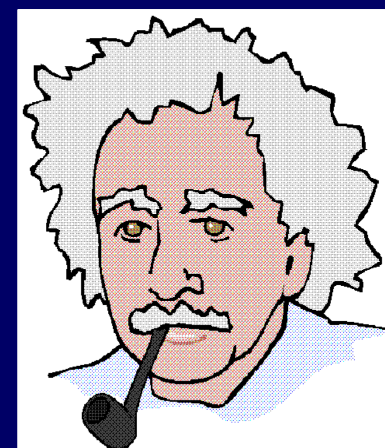
K_0= the set of programs that take no input and halt

P = [input I; Q]
Does P(P) halt?



BUILD:
Oracle for K

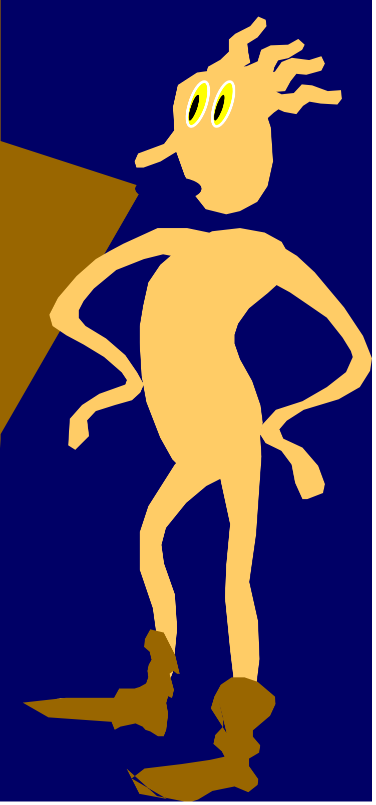
Does [I:=P;Q] halt?



GIVEN:
Oracle for K_0

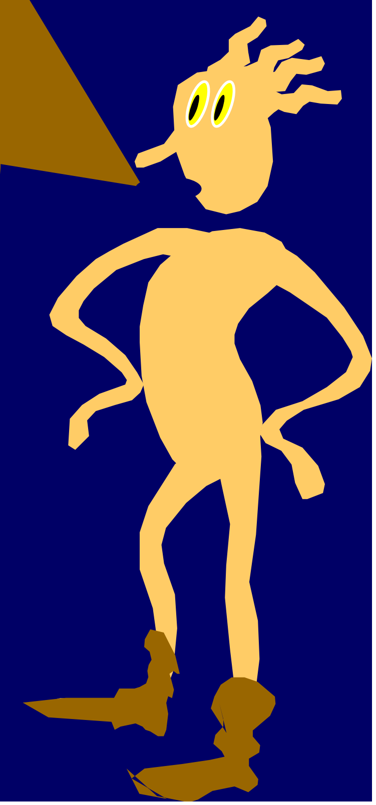
We've reduced the problem of deciding membership in K to the problem of deciding membership in K_0.

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.

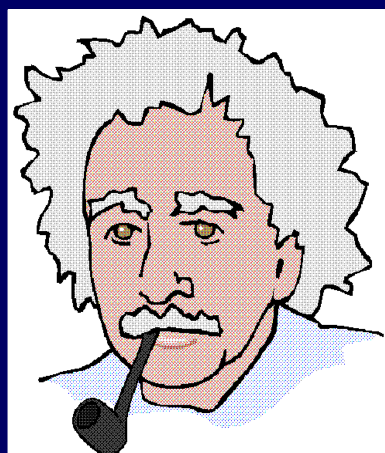
We already know K is not decidable, hence K_0 is not decidable.



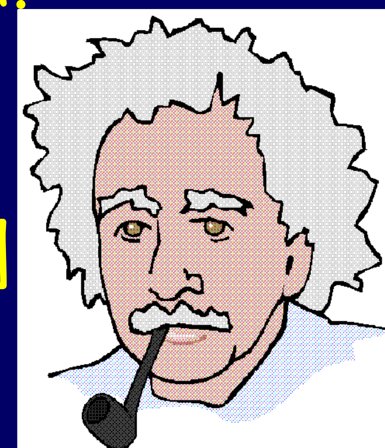
HELLO = the set of programs that print hello and halt

Does P halt?

Let P' be P with all print statements removed.

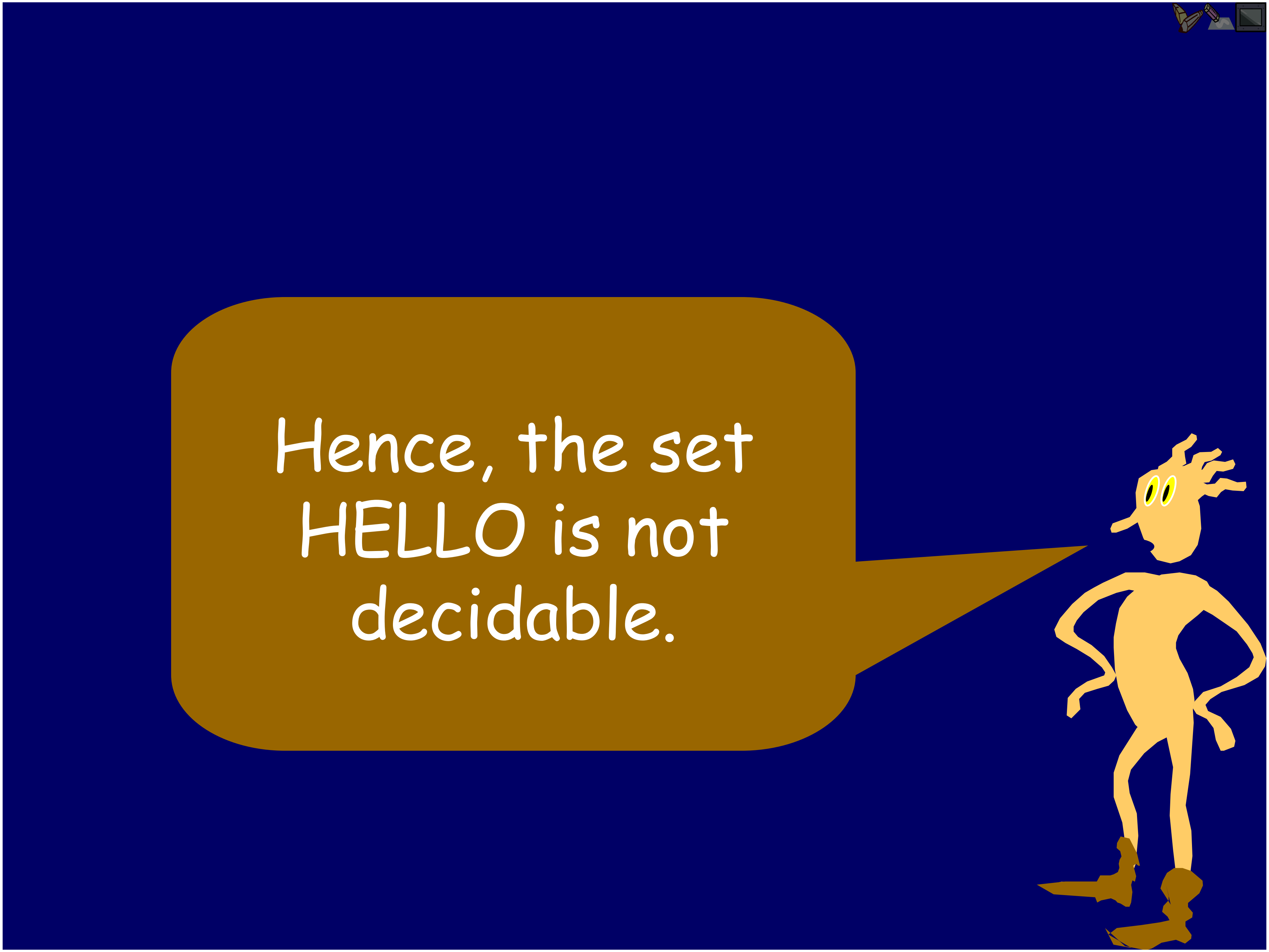


Is [P'; print HELLO] a hello program?



BUILD:
Oracle
for K_0

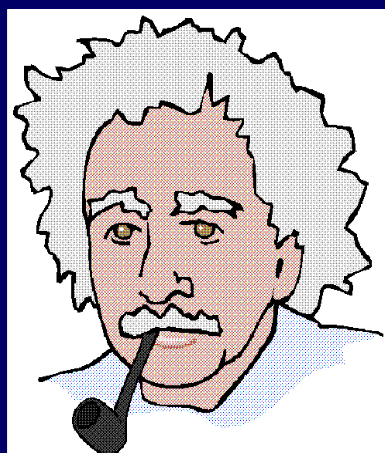
GIVEN:
HELLO
Oracle


Hence, the set HELLO is not decidable.

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

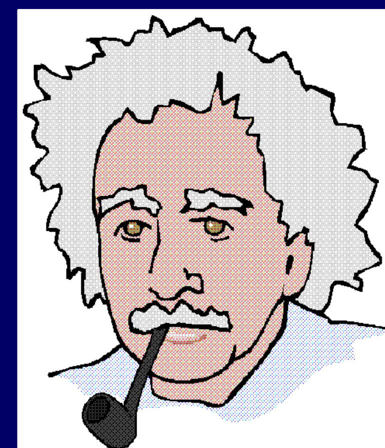
Is P in set HELLO?

Let HI = [print HELLO]



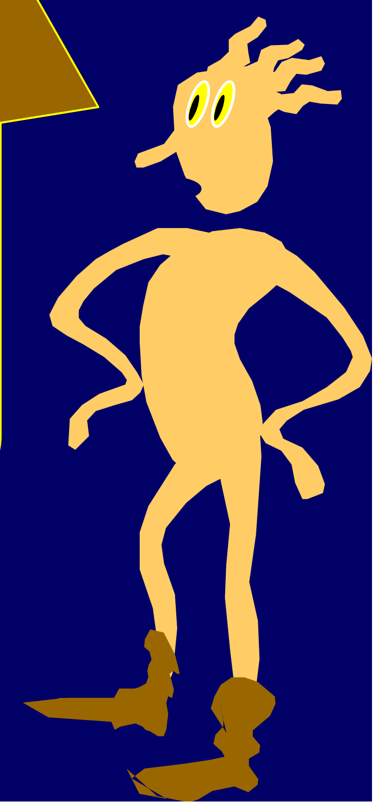
BUILD:
HELLO
Oracle

Are P and HI equal?



GIVEN:
EQUAL
Oracle

Halting with input,
Halting without input,
HELLO, and
EQUAL are all
<u>undecidable</u>.



Diophantine Equations

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

D = {multi-variant integer polynomials P | P has a 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: Dramatis Personae

http://www.goldenmuseum.com/1612Hilbert_engl.html

Martin Davis, Julia Robinson, Yuri Matiyasevich (1982)

and…

Good old Fibonaccimal…

Define "Fibonaccimal numbers" as follows:

Consider a sequence of digits $S = (a_n, a_{n-1}, \cdots, a_3, a_2, a_1)$, where (i) each $a_i \in \{0,1\}$; (ii) no consecutive 1's occurs in S, i.e., for all $1 \le i \le n-1$, $a_i + a_{i+1} \le 1$; and (iii) S has no leading zero, i.e., $a_n = 1$. We call S a "Fibonaccimal number" of length n and interpret it as the number

$$\sum_{i=1}^{n} a_i F_i$$

For example, $(1) = F_1 = 1$, $(1,0) = F_2 = 2$, $(1,0,0) = F_3 = 3$, $(1,0,1) = F_3 + F_1 = 4$, $(1,0,0,0) = F_4 = 5$, etc. Note that $(1,1,0)$ would **not** be a valid Fibonaccimal number.

- **(20 points)** Show that any positive integer N can be expressed in the "Fibonaccimal" representation.
- **(15 points)** Prove also that the representation is unique for all positive integer. That is, for each integer $N \ge 1$, if S_1 and S_2 both represent N, then the two sequences S_1 and S_2 are identical.

Zeckendorf's Theorem: Every number can be represented uniquely in the Fibonaccimal representation!

Polynomials can encode programs.

There is a computable function

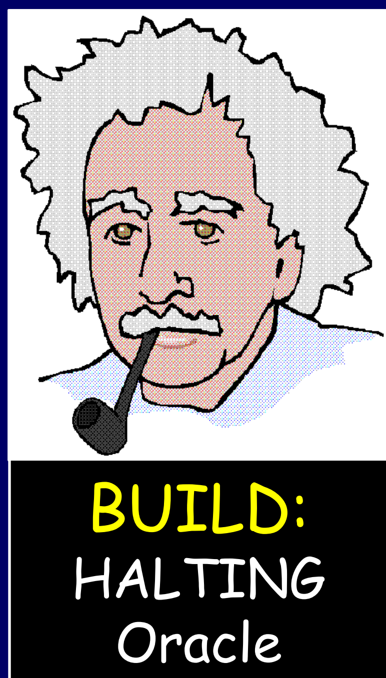
F: Java programs that take no input $\rightarrow$
Polynomials over the integers

Such that

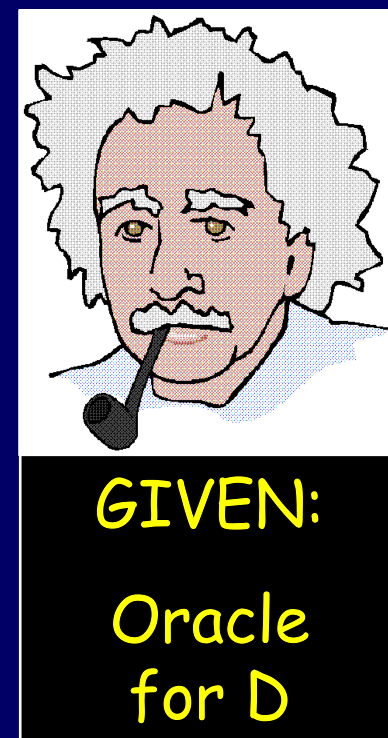
program P halts $\Leftrightarrow$ F(P) has an integer root

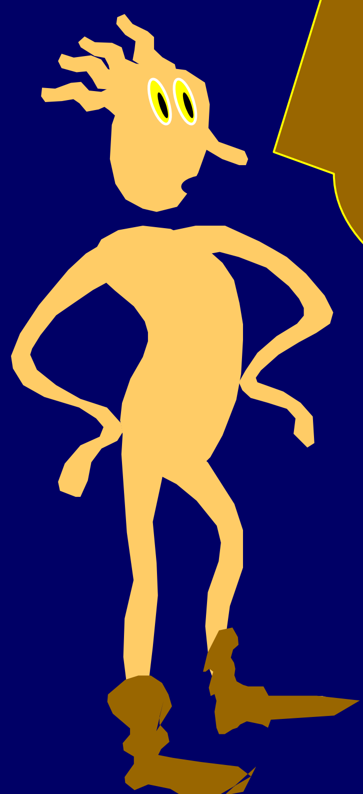
D = the set of all integer polynomials with integer roots

Does program P halt?



F(P) has integer root?




Problems that have no obvious relation to halting, or even to computation can encode the Halting Problem is non-obvious ways.

Self-Reference Puzzle

Write a program that prints itself out as output. No calls to the operating system, or to memory external to the program.

HW12: Auto Cannibal Maker

Write a program AutoCannibalMaker that takes the text of a program EAT as input and outputs a program called $SELF_{EAT}$.

When $SELF_{EAT}$ is executed, it should output $EAT(SELF_{EAT})$

Suppose HALT with no input was programmable in JAVA.

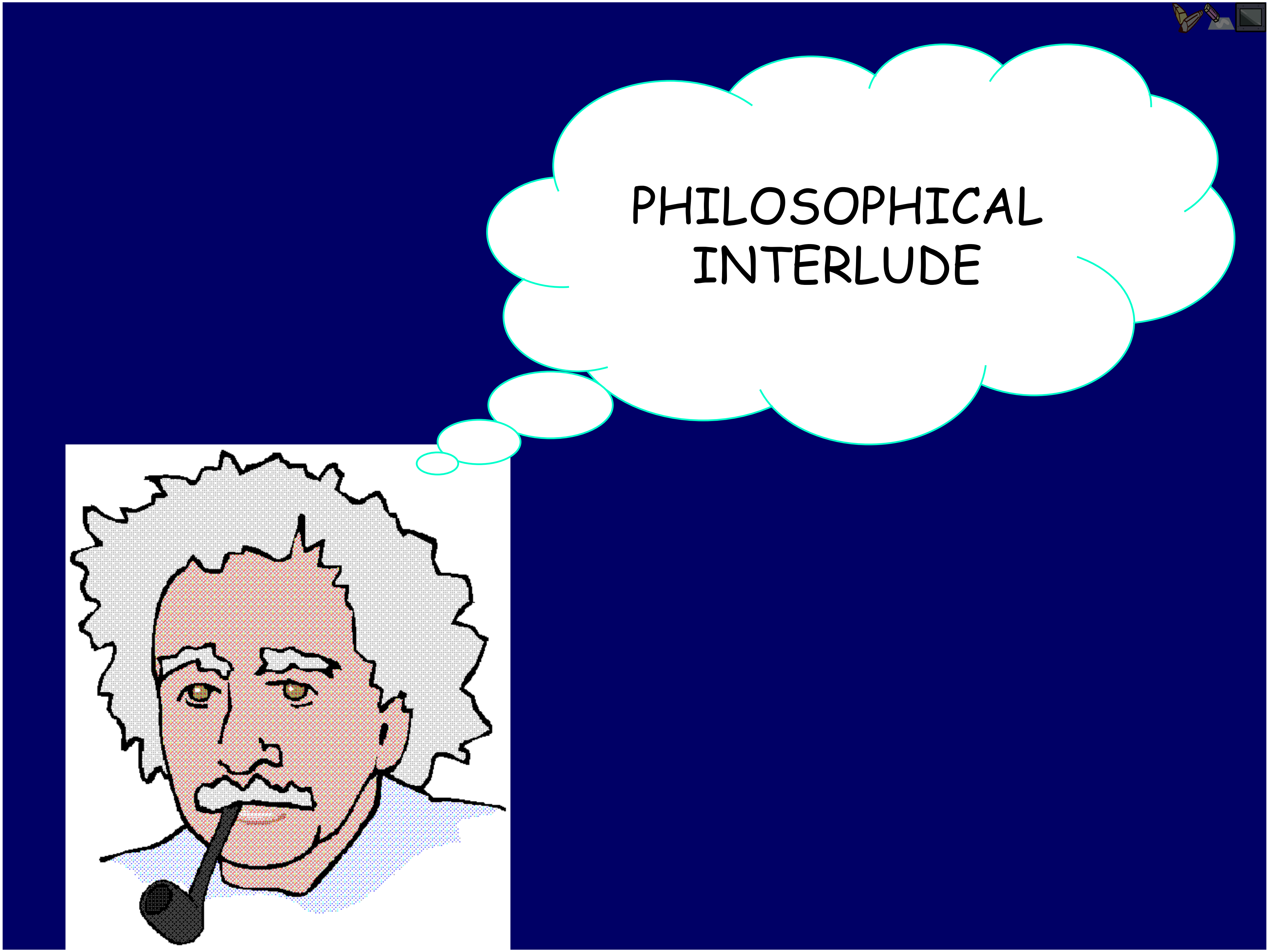
Write a program AutoCannibalMaker that takes the text of a program EAT as input and outputs a program called $SELF_{EAT}$.

When $SELF_{EAT}$ is executed it should output $EAT(SELF_{EAT})$

Let EAT(P) = halt, if P does not halt
loop forever, otherwise.

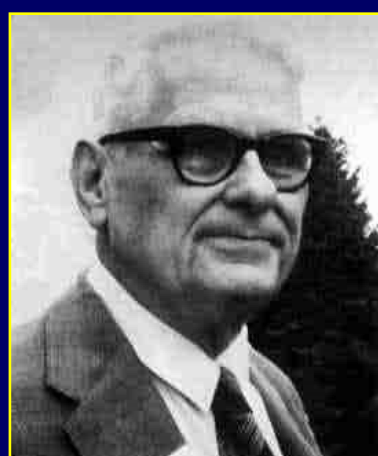
What does $SELF_{EAT}$ do?

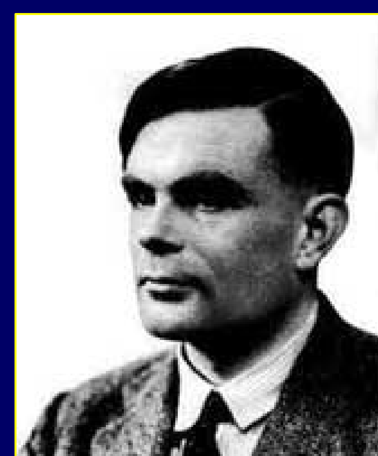
Contradiction! Hence EAT does not have a corresponding JAVA program.


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.

Spiritual Intuition

The mind consists of part matter and part soul. Soul, by its very nature, defies reduction to physical law. Thus, the action and thoughts of the brain are not simulable or reducible to simple components and rules. The thesis is false.

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.

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

But this ain't philosophy class!

Another important notion

Computability Theory: Vocabulary Lesson

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

P prints an (infinite) list of strings.

- Any element on the list should be in S.
- Each element in S appears after a finite amount of time.

Is
the halting set K
enumerable?

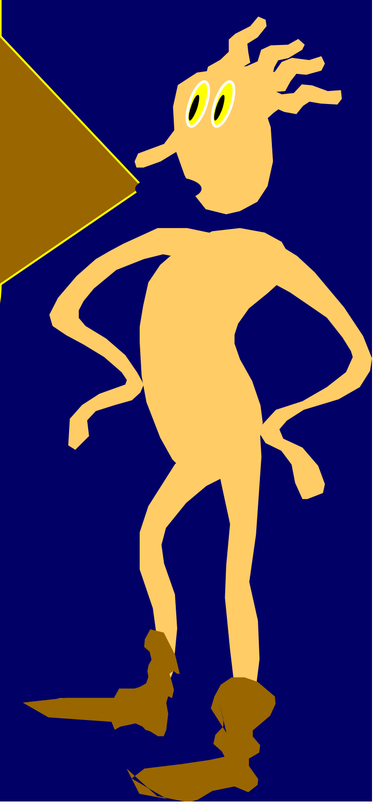
Enumerating K

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

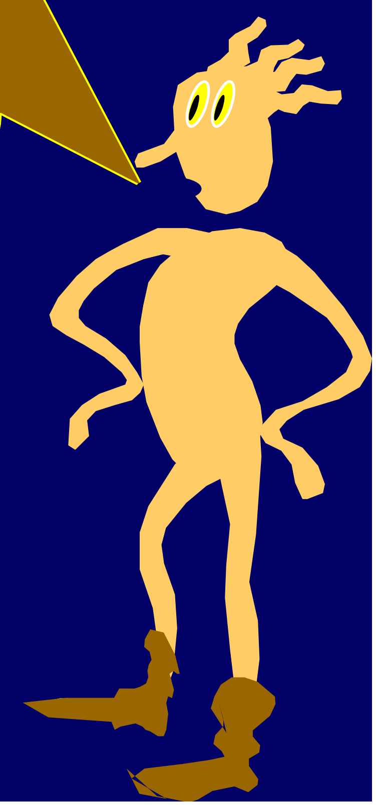
K is <u>not</u> decidable, but it is enumerable!

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

Is K' enumerable?



If both K and K' are enumerable, then K is decidable. (why?)



Now that we have established that the Halting Set is undecidable, we can use it for a jumping off points for more “natural” undecidability results.

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