Great Theoretical Ideas In Computer Science

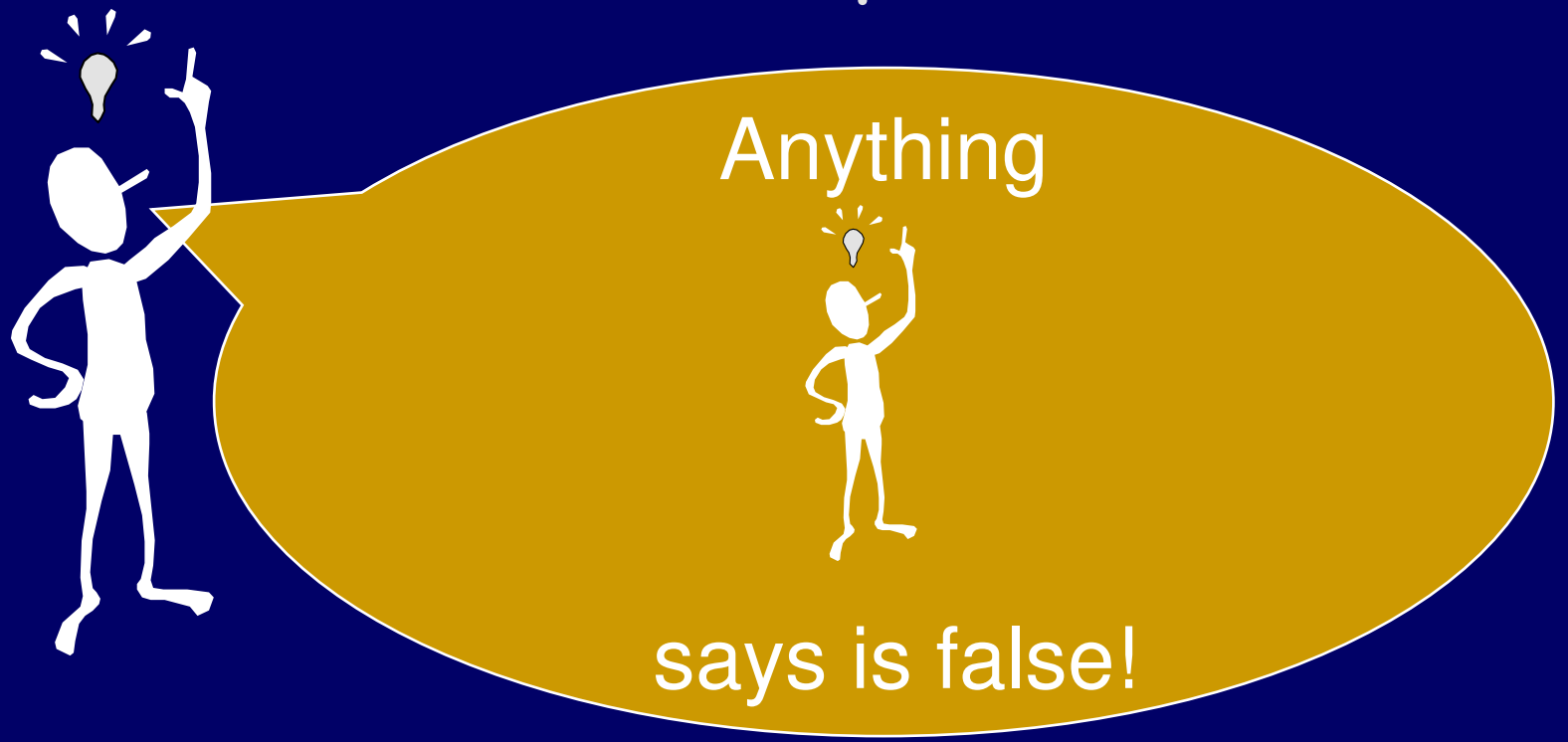
Steven Rudich

CS 15-251 Spring 2004

Lecture 26 April 20, 2003

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.

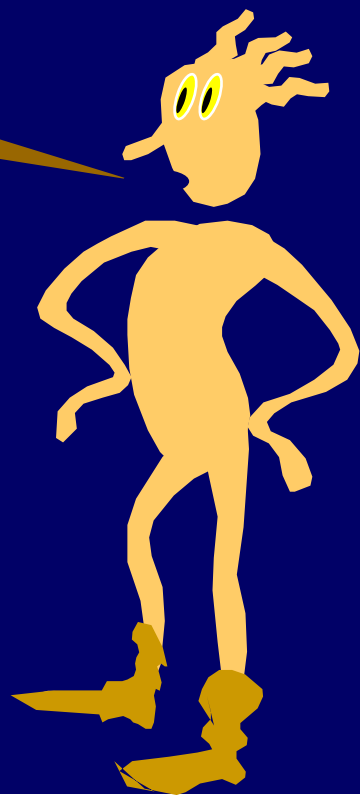
Grading Script

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



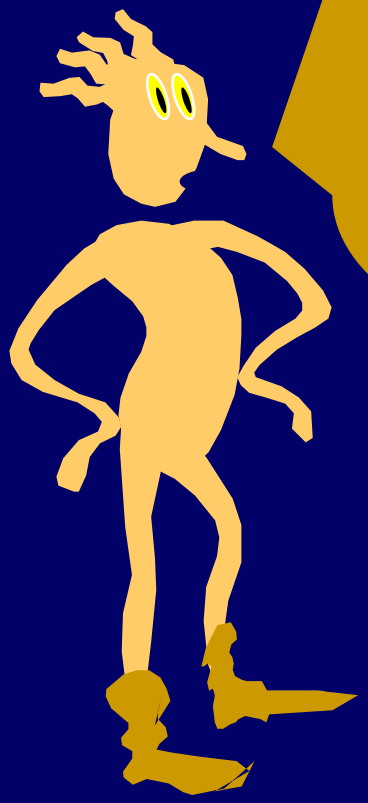
Nasty Program

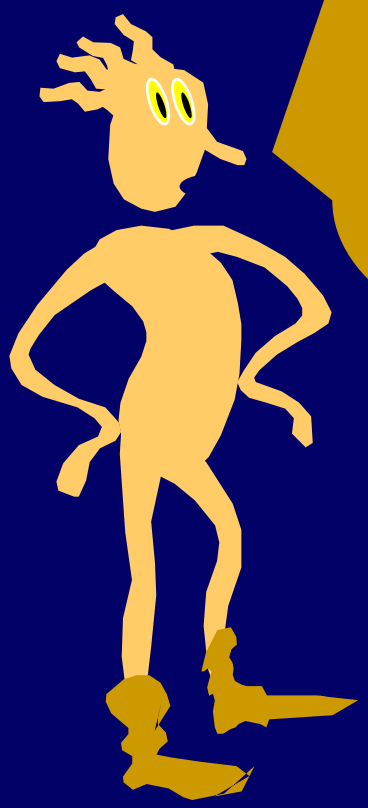
```
n:=0;
While
( n is not a counter-example
to the Riemann Hypothesis)
        n++

PRINT "HELLO"
```

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

Despite the simplicity of the HELLO 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.

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, i.e., 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 $x \in \Sigma^*$ $P(x) = f(x)$.

Countably many computable functions.

Fix any finite set of symbols, Σ. Fix any precise programming language, i.e., 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 $x \in \Sigma^*$ $P(x) = f(x)$.

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

Uncountably many functions.

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

For any subset S of Σ^* we map to the function f where:

f(x) = 1 x in S

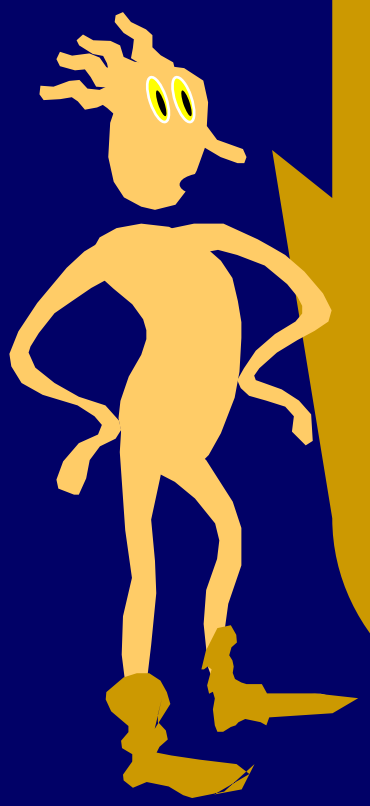
f(x) = 0 x not in S

Uncountably many functions.

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

Then the set of all f: Σ^* - > {0,1} has the same size as the powerset of Σ^*. Since Σ^* is countable its powerset is uncountably big.

Thus, most functions from Σ^* to {0,1} are not computable. Can we describe an incomputable one? Can we describe an interesting, incomputable function?



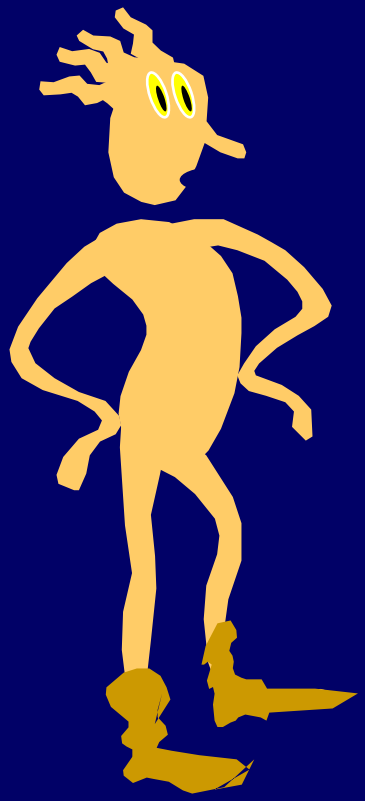
Notation And Conventions

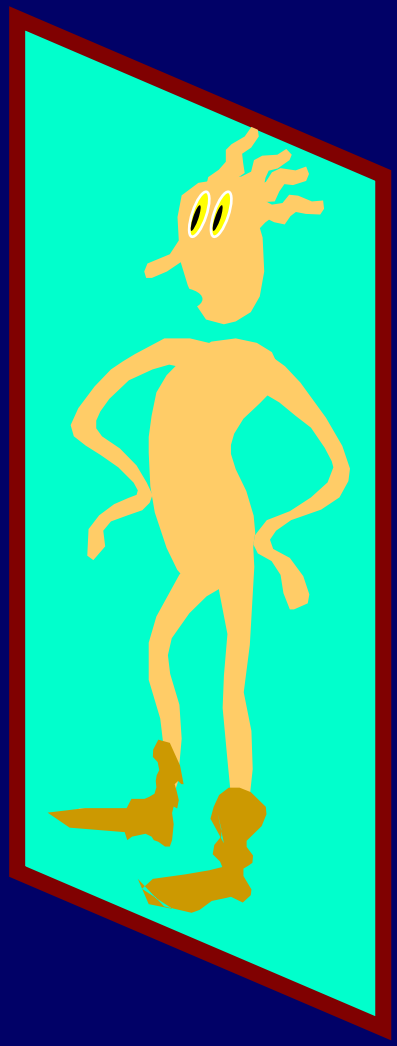
- Fix a single programming language
- When we write program P we are talking about the text of the source code for P
- P(x) means the output that arises from running program P on input x, assuming that P eventually halts
- P(x) = $\perp$ means P did not halt on x

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 Famous Halting Set: K

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

K ={ Java P | P(P) 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 to 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 CONFUSE.

CONSUSE(P):
If HALT(P) then loop_for_ever
Else return (i.e., halt)
<text of subroutine HALT goes here>

Does CONFUSE(CONFUSE) halt?

YES implies HALT(CONFUSE) = yes
thus, CONFUSE(CONSFUSE) will not halt

NO implies HALT(CONFUSE) = no
thus, CONFUSE(CONFUSE) halts

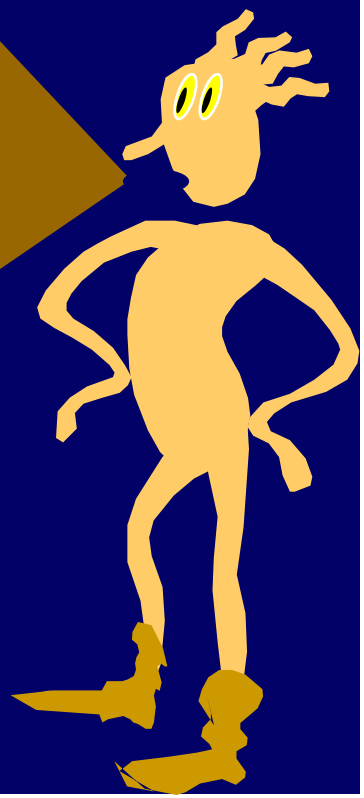
CONSUSE(P):
If HALT(P) then loop_for_ever
Else return (i.e., halt)
<text of subroutine HALT goes here>

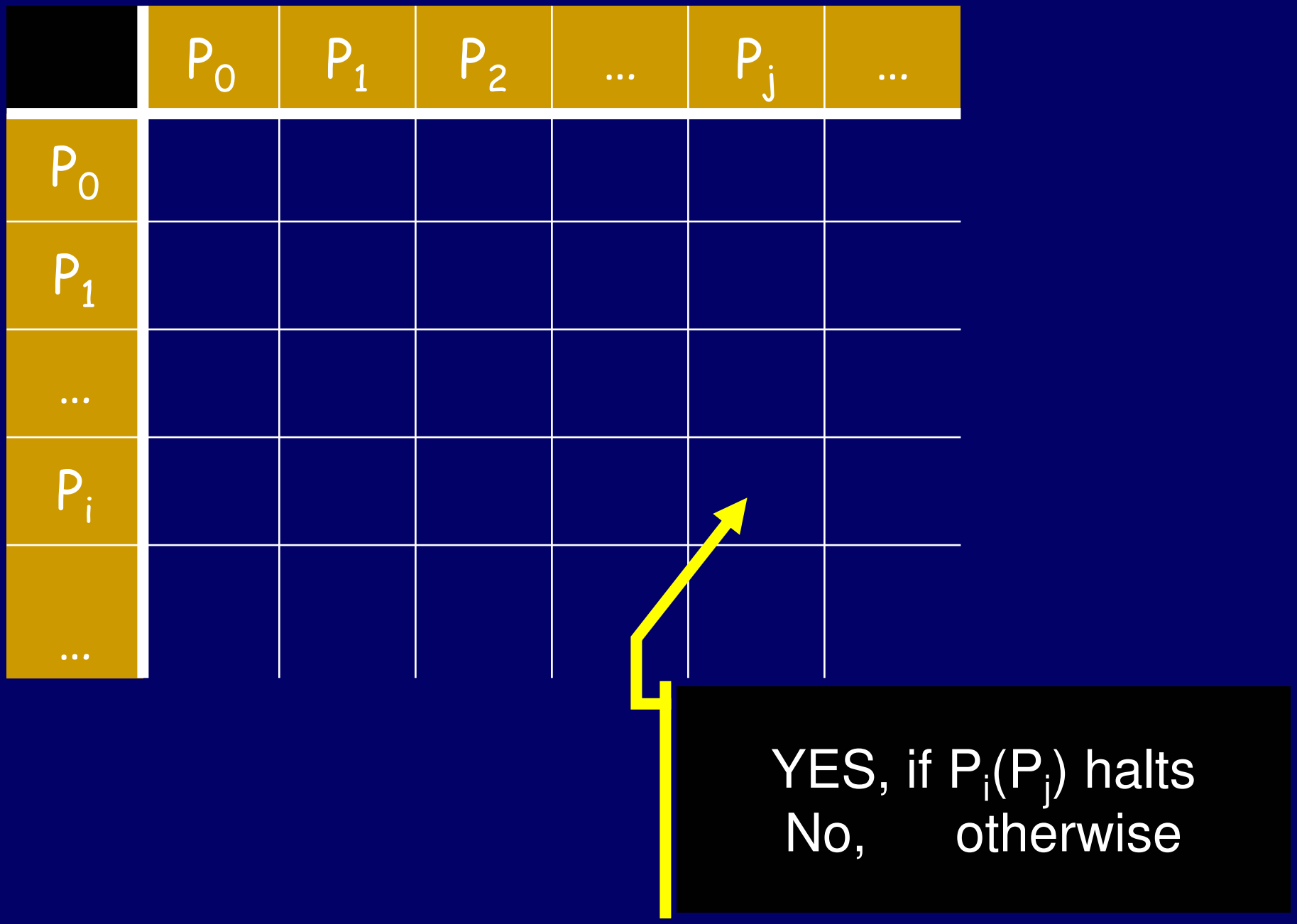
Does CONFUSE(CONFUSE) halt?

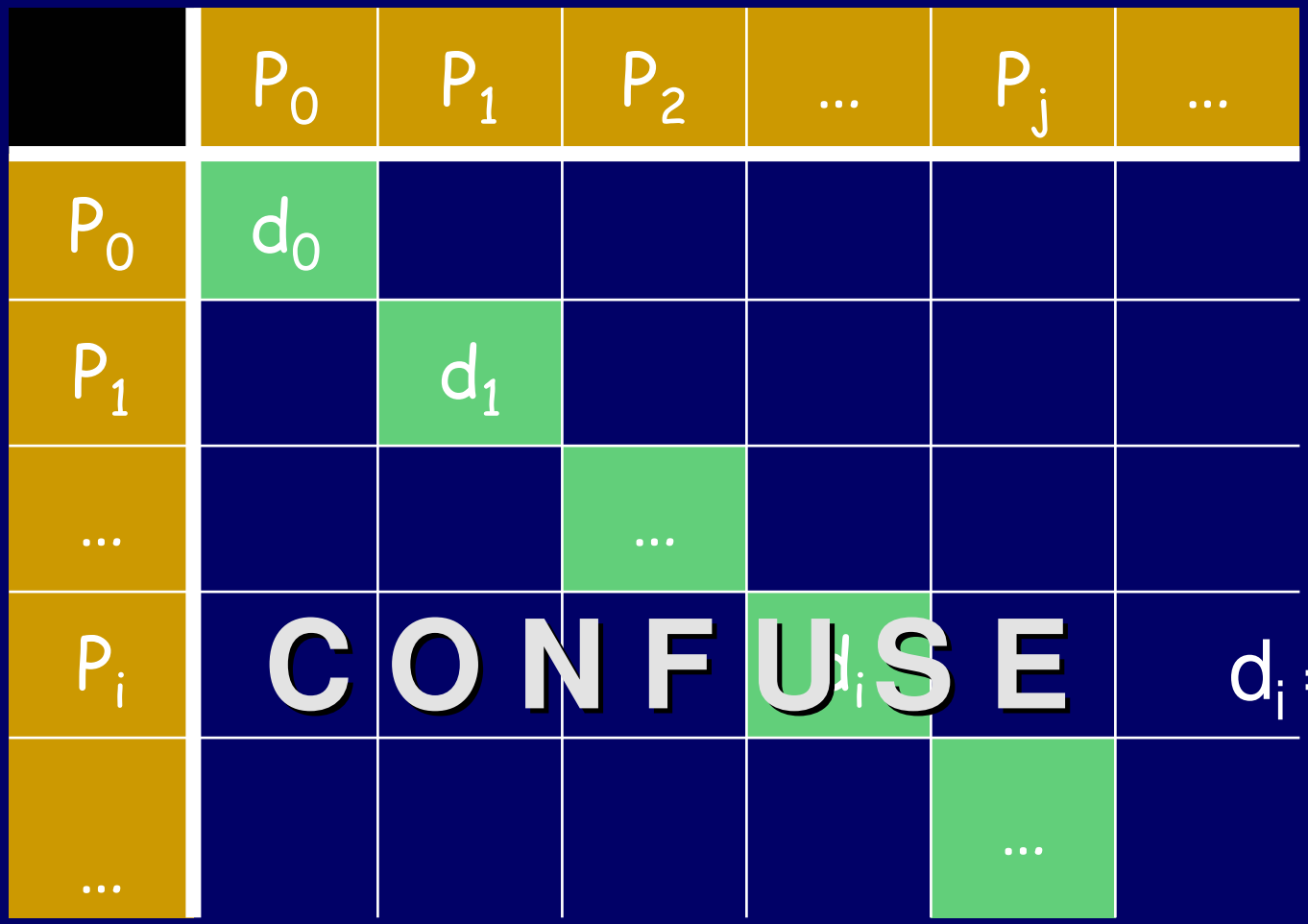
YES implies HALT(CONFUSE) = yes
thus, CONFUSE(CONSFUSE) will not halt

NO implies HALT(CONFUSE) = no
thus, CONFUSE(CONFUSE) halts

Turing's argument is essentially the reincarnation of 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



$d_i = HALT(P_i)$

CONFUSE(P_i) halts iff d_i = no

The CONFUSE row contains the negation of the diagonal.

Alan Turing (1912-1954)

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

Consider the real number whose binary expansion has a 1 in the i^{th} position iff $P_i \in K$.

Computability Theory: Vocabulary Lesson

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

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

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

We already know: K is undecidable

Computability Theory: Vocabulary Lesson

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

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

Is K Enumerable?

Enumerating K

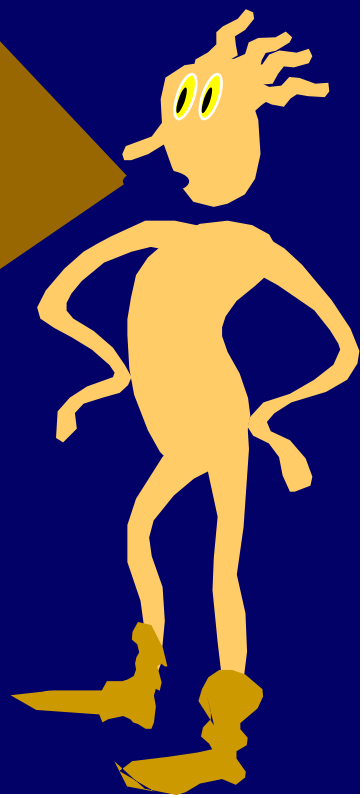
For n = 0 to forever do

{Loop through w = all strings of length < n do:
    {If w(w) halts in n steps then Output w}
}

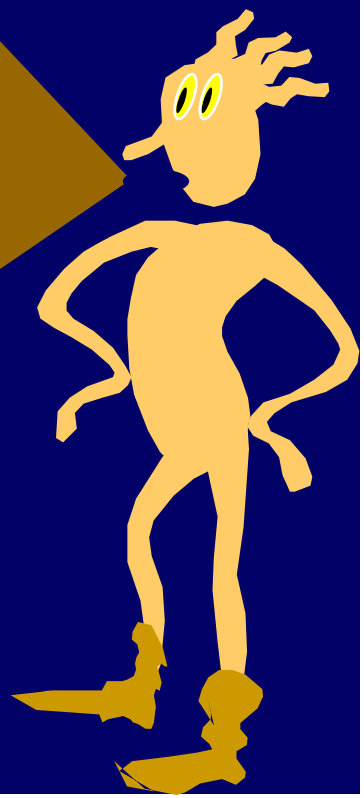
K is NOT decidable, but it is enumerable!

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

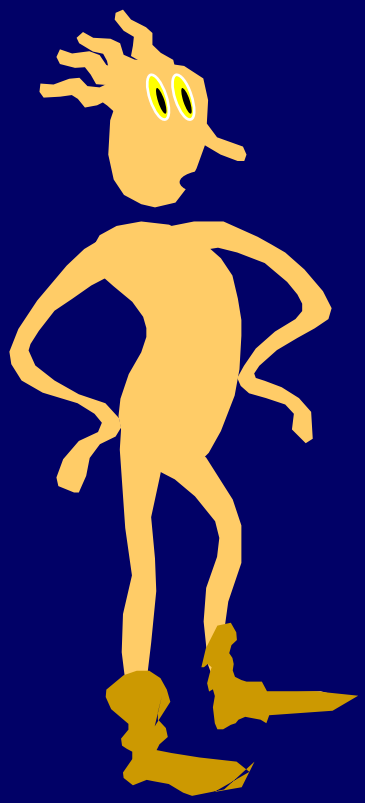
Is K' enumerable?



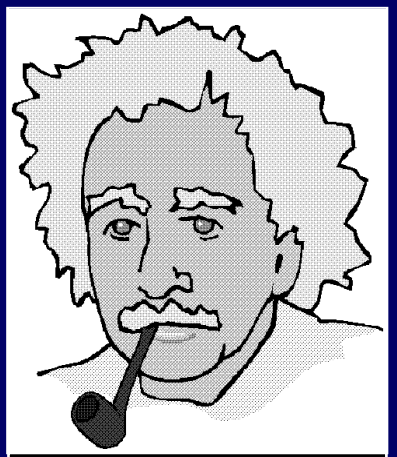
Now that we have established that the Halting Set is undecidable, we can use it for a jumping off points for more “natural” undecidability results.



Oracle For Set S



Is $x \in S$?



Oracle for S

YES/NO

Example Oracle
S = Odd Naturals

4?

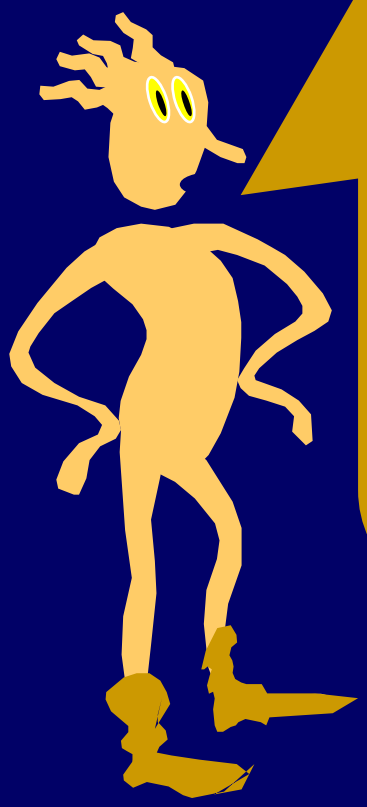
No

81?

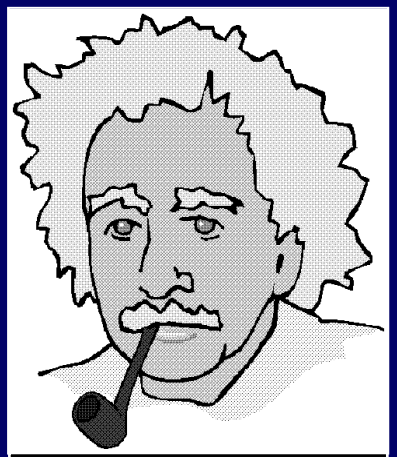
Yes

Oracle for S

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



Hey, I order an oracle for the famous halting set K, but when I opened the package it was an oracle for the different set K_0.



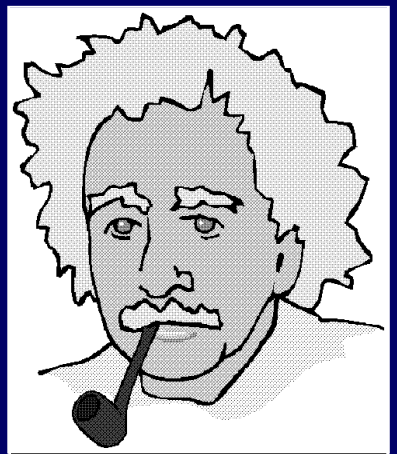
GIVEN: Oracle for K_0

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

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

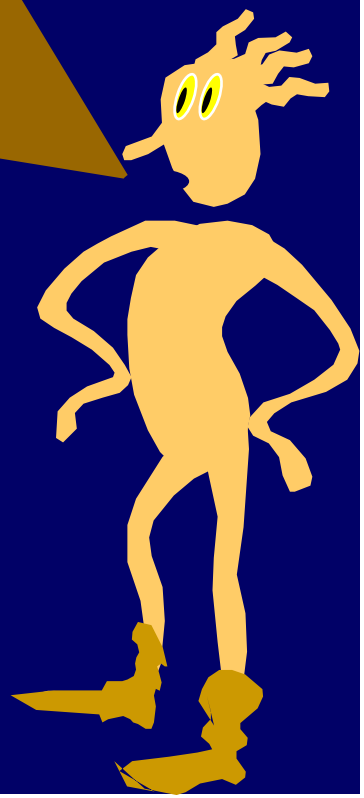
BUILD:
Oracle
for K

Does [I:=P;Q] halt?



GIVEN:
Oracle
for K_0

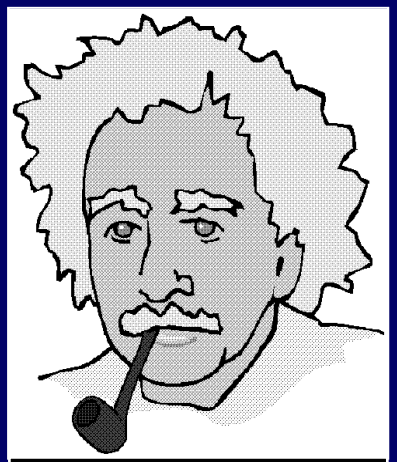
Thus, if K_0 were decidable then K would be as well. We already know K is not decidable, hence K_0 is not decidable.



HELLO = the set of program that print hello and halt

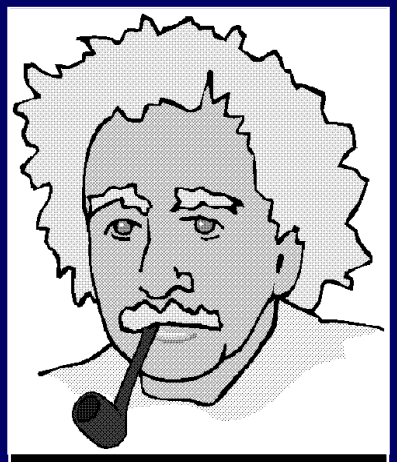
Does P halt?

Let P' be P with all print statements removed.



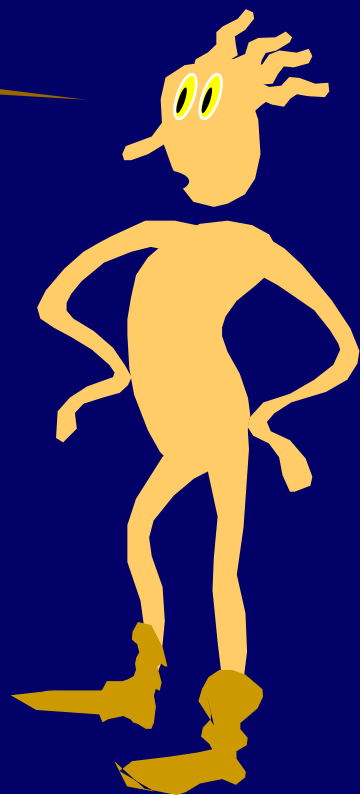
BUILD:
Oracle
for K_0

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



GIVEN:
HELLO
Oracle

HELLO is not decidable.



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

Does P equal HELLO ?

Let HI = [print HELLO]

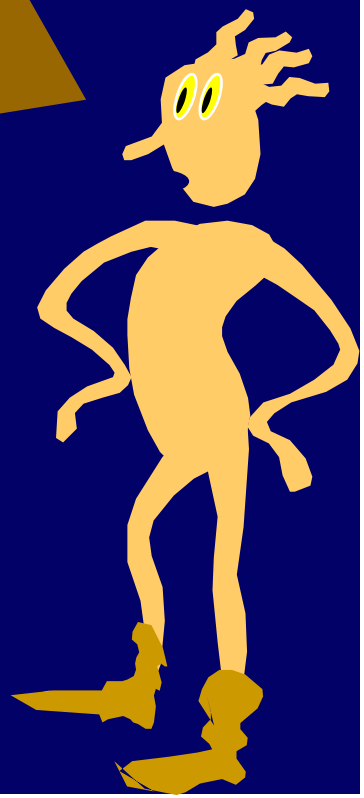
BUILD:
HELLO
Oracle

Are P and HI equal?

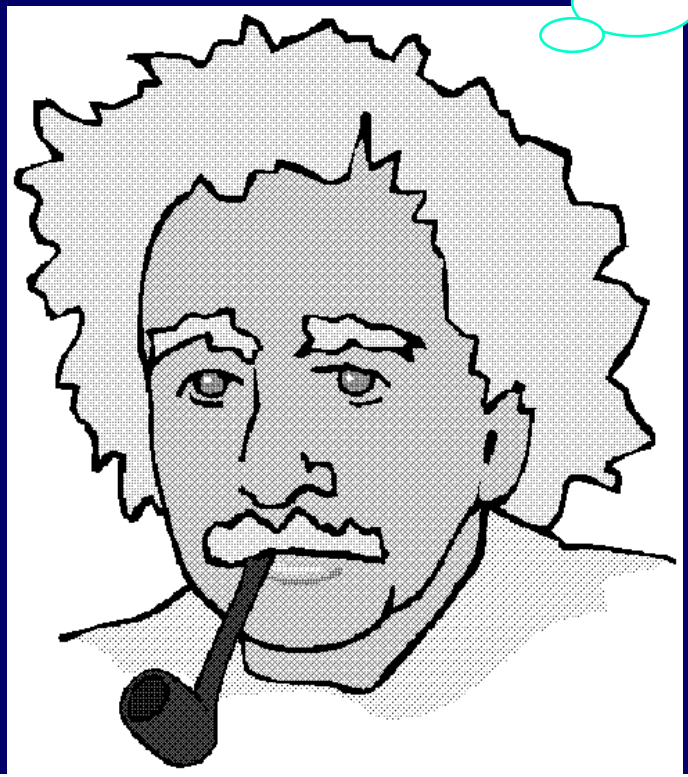
GIVEN:

EQUAL
Oracle

Halting with input, Halting without input,
Hello, and
EQUAL are 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 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.

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

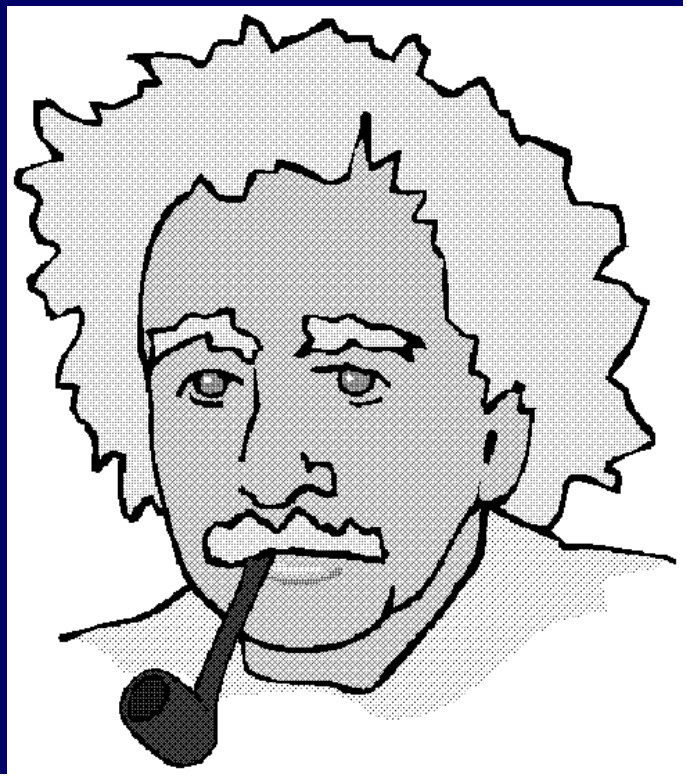
The mind consists of part matter and part soul. Soul, by its very nature, defies reduction to physical law. Thus, the action and thoughts of the brain are not simulable or reducible to simple components and rules. The thesis is false.

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!



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.

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 $EAT(SELF_{EAT})$

Auto Cannibal Maker
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.

$$4X^2Y + XY^2 = 0$$

Do this polynomial have an integer root? I.e., does it have a zero at a point where all variables are integers?

Diophantus: Given a multi-variate polynomial over the integers, does it have an integer root?

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]

Polynomials can encode programs.

There is a computable function
F: Java programs that take no input ->
Polynomials over the integers

Such that

Program P halts ←→ F(P) has an integer root

D = the set of all integers polynomials with integer roots

Does program P halt?

BUILD: HALTING Oracle

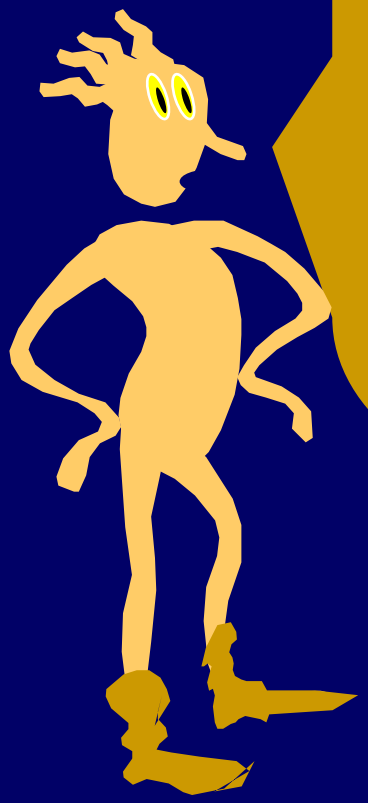
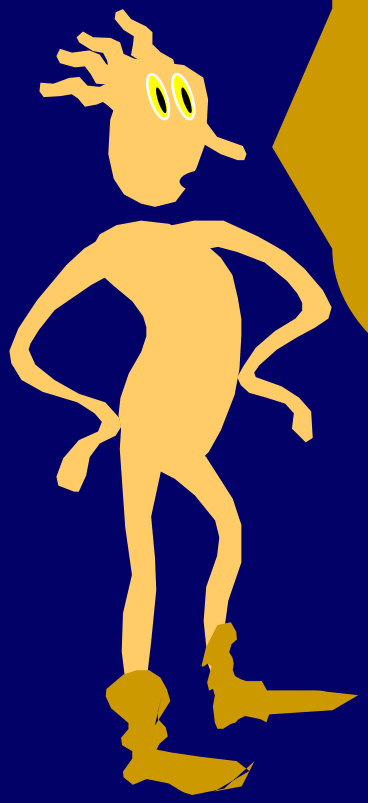
F(P)?

GIVEN: D

A Million Dollar Diophantine Problem.

Does F(Nasty Program) have a root?

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



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