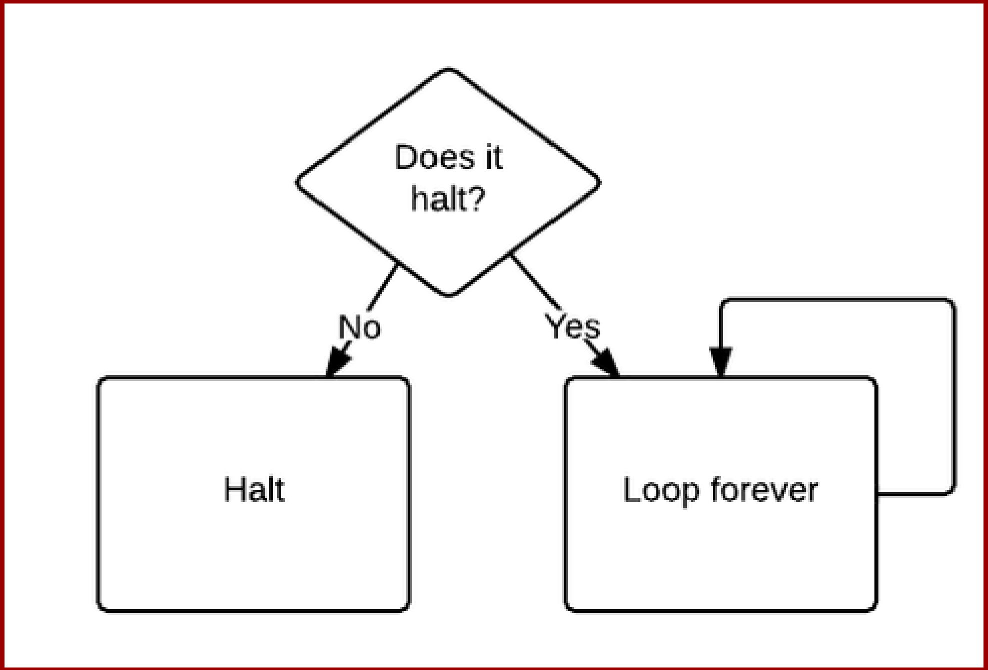
15-251: Great Theoretical Ideas in Computer Science

Lecture 8

Undecidability



Almost all Languages are undecidable

Set of all languages: $|\{S | S \subseteq \{0,1\}^*\}| = |\mathcal{P}(\{0,1\}^*)| = |\mathbb{R}|$

Set of all dec. lang.: $\leq |\{\langle M \rangle \in \{0,1\}^* | M \text{ is a decider TM}\}|$

$= |\{0,1\}^*| = |\mathbb{N}|$

$\Longrightarrow$ Most languages do not have a TM deciding them

Question:

Is it just weird languages that no one would care about which are *undecidable*?

Answer (due to Turing, 1936):

Sadly, no.

There are many natural languages one would like to compute but which are undecidable.

Many interesting Languages are undecidable

In particular, any problems related to non-wimpy / Turing equivalent computation are undecidable.

Decidable Problems

$\text{ACCEPT}_{\text{DFA}}$ = { $\langle D, x \rangle$ | D is a DFA that accepts x}

$\text{SELF-ACCEPT}_{\text{DFA}}$ = {$\langle D \rangle$ | D is a DFA that accepts $\langle D \rangle$}

$\text{EMPTY}_{\text{DFA}}$ = { $\langle D \rangle$ | D is a DFA that accepts no x}

$\text{EQUIV}_{\text{DFA}}$ =
= { $\langle D, D' \rangle$ | D and D' are DFA and L(D) = L(D')}

Theorem:

$\text{ACCEPT}_{\text{DFA}}$, $\text{SELF-ACCEPT}_{\text{DFA}}$, $\text{EMPTY}_{\text{DFA}}$ and **$\text{EQUIV}_{\text{DFA}}$** are decideable.

Undecidable Problems

ACCEPT$_{TM}$ = { $\langle M, x\rangle$ | M is a TM that accepts x}

SELF-ACCEPT$_{TM}$ = {$\langle M\rangle$| M is a TM that accepts $\langle M\rangle$ }

EMPTY$_{TM}$ = { $\langle M\rangle$ | M is a TM that accepts no x}

EQUIV$_{TM}$ =
= {$\langle M, M'\rangle$| M and M' are TMs and L(M) = L(M')}

Theorem:

ACCEPT$_{TM}$, SELF-ACCEPT$_{TM}$, EMPTY$_{TM}$ and **EQUIV$_{TM}$** are undecideable.

Example: Program Equivalence

Given a program P and a program P' we would like to automatically decide whether both do the same thing.

Formally:

EQUIV_{TM} =

= { $\langle P, P' \rangle$ | P and P' are Python programs and L(D) = L(D')}

Useful for:

- Compiler Optimization
- Matching programs to their specification
- Autograder for 112 or 251 ☺

Example: 112 Autograder

First 112 assignment: Write a "Hello World" program.

Given a program P submitted by a student we want to automatically decide whether P does the right thing.

We want an algorithm A such that:

$$A(\langle P \rangle) = \begin{cases} \text{pass} & \text{iff P outputs "Hello World" and} \\ \text{fail} & \text{otherwise} \end{cases}$$

More formally we want to decide the language

HELLO = {$\langle M \rangle$| M is a TM that outputs "Hello World" when run on the empty intput}

Simple enough, no?

Not at CMU!

Hello World HW Submission #1

```
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);}
```

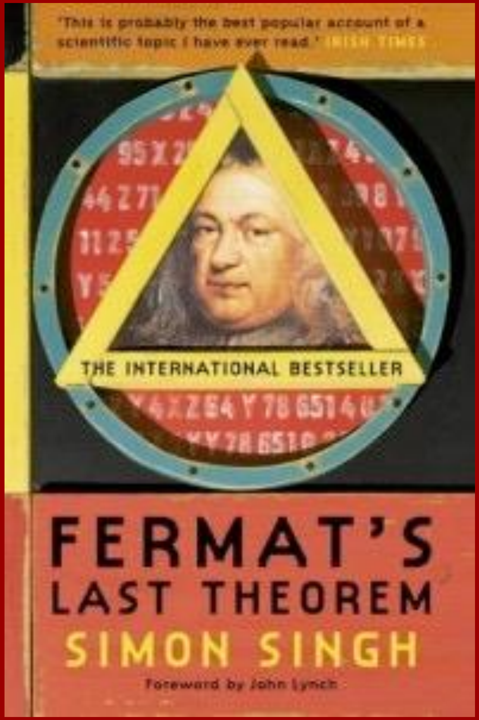
This C program prints out all the lyrics of

The Twelve Days Of Christmas.

Ok, so let just run the program P and check the output.

Hello World HW Submission #2

```
def HelloWorld():
    t = 3
    while (True):
        for n in xrange(3, t+1):
            for x in xrange(1, t+1):
                for y in xrange(1, t+1):
                    for z in xrange(1, t+1):
                        if (x**n + y**n == z**n):
                            return "Hello World"
        t += 1
```



Terminates and outputs "Hello World" if and only if Fermat's Last Theorem is false.

Hello World HW Submission #3

```
numberToTest := 2;
flag := 1;
while flag = 1 do
    flag := 0;
    numberToTest := numberToTest + 2;
    for p from 2 to numberToTest do
        if IsPrime(p) and IsPrime(numberToTest−p) then
            flag := 1;
            break;
        end if
    end for
end do
print(“HELLO  WORLD”)
```

Terminates and outputs “Hello World”

if and only if Goldbach’s Conjecture is false.

A simple undecidable language

Autograder / Hello World problem:

Given a program P, is it terminating and outputting "Hello World"?

HELLO = {$\langle M \rangle$| M is a TM that outputs "Hello World" on the empty intput ε}

Halting problem:

Given a program P, is it terminating?

HALT_ε = {$\langle M \rangle$ | M is a TM terminating on ε}

HALT = {$\langle M, x \rangle$ | M is a TM terminating on x}

The Halting Problem is Undecidable (1936)

The Halting Problem is Undecidable

Theorem:

The language

HALT = {$\langle M, x \rangle$ **| M is a TM terminating on x}**

is undecidable.

Proof:

Assume for the sake of contradiction that M_{HALT} is a decider TM which decides HALT.

The Halting Problem is Undecidable

Here is the description of another TM called D, which uses M_{HALT} as a subroutine:

D:

> Given as input ⟨M⟩, the encoding of a TM M:
>
> D executes M_{HALT}(⟨M, ⟨M⟩⟩).
>
> If this call accepts, D enters an infinite loop.
>
> If this call rejects, D halts (say, it accepts).

In other words… D(⟨M⟩) loops if M(⟨M⟩) halts,
halts if M(⟨M⟩) loops.

The Halting Problem is Undecidable

Assume M_{HALT} is a decider TM which decides HALT.

We can use it to construct a machine D such that

$$D(\langle M \rangle) \quad \text{loops} \quad \text{if } M(\langle M \rangle) \text{ halts,}$$
$$\qquad\qquad \text{halts} \quad \text{if } M(\langle M \rangle) \text{ loops.}$$

Time for the contradiction:

Does D(⟨D⟩) loop or halt?

By definition, if it loops it halts and if it halts it loops.

Contradiction. $\blacksquare$

BTW: This is essentially just

Cantor's Diagonal Argument.

D(⟨M⟩) loops if M(⟨M⟩) halts, halts if M(⟨M⟩) loops

The set of all TM's is countable, so list it:

	$\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	$\langle M_5\rangle$	$\cdots$
M_1	halts	halts	loops	halts	loops	
M_2	loop	loops	loops	loops	loops	
M_3	halts	loops	halts	halts	halts	
M_4	halts	halts	halts	halts	loops	
M_5	halts	loops	loops	halts	loops	
$\vdots$						

How could D be on this list?

What would the diagonal entry be??

D(⟨M⟩) loops if M(⟨M⟩) halts, halts if M(⟨M⟩) loops

The set of all TM's is countable, so list it:

	⟨M_1⟩	⟨M_2⟩	⟨M_3⟩	⟨M_4⟩	⟨M_5⟩	…
M_1	halts	halts	loops	halts	loops	
M_2	loop	loops	loops	loops	loops	
M_3	halts	loops	halts	halts	halts	
M_4	halts	halts	halts	halts	loops	
M_5	halts	loops	loops	halts	loops	
⋮						

The Halting Problem is Undecidable

Theorem:

There exists no program P_{HALT}.py which, given as input a program file P.py and an input file input.txt, always correctly outputs whether program P.py terminates on this input.

Proof:

Assume for the sake of contradiction that

P_{HALT}.py exists.

The Halting Problem is Undecidable

Here is the pseudo-code of another program Impossible.py, which uses P_{HALT}.py as a subroutine:

Impossible.py:

Input: Program file P.py

Copy P.py to input.txt

run P_{HALT}.py, as a subroutine, with P.py and input.txt, i.e.,

result = P_{HALT}.py P.py input.txt

If result == halts then loop forever

else terminate

The Halting Problem is Undecidable

Imp.py:

Input: Program file P.py

Copy P.py to input.txt

run P_{HALT}, as a subroutine, with P.py and input.txt, i.e.,

result = P_{HALT}.py P.py   input.txt

If   result == halts   then loop forever

else   terminate

Now running Imp.py with Imp.py as input leads to the execution of P_{HALT}.py Imp.py Imp.py followed by doing the opposite of what P_{HALT} Imp.py Imp.py predicted would happen.

Given some code,
determine if it terminates.

It's not: "we don't know how to solve it efficiently".

It's not: "we don't know if it's a solvable problem".

We know that it is unsolvable by any algorithm.

We know that it is unsolvable by any algorithm, any mechanism, any human being, anything in this world and any (physical) world we can imagine.

ACCEPT is undecidable

Theorem:

ACCEPT = $\{\langle M, x\rangle \mid M$ is a TM which accepts $x\}$

is undecidable.

We could use the same diagonalization proof for ACCEPT.

But maybe there is an easier way …

Particularly, ACCEPT seems clearly harder than HALT.

After all, how can I decide if a program accepts if I don’t even know if it halts.

ACCEPT is undecidable

Theorem:

ACCEPT = {⟨M, x⟩ | M is a TM which accepts x}

is undecidable.

New Proof Strategy:

Try to show that:

ACCEPT is at least as hard as HALT

$\Longleftrightarrow$

HALT is at most as hard as ACCEPT

$\Longleftrightarrow$

HALT would be easy if ACCEPT were easy

ACCEPT is undecidable

Theorem:

ACCEPT = {⟨M, x⟩ | M is a TM which accepts x}

is undecidable.

Proof (by contradiction):

Assume ACCEPT is decidable then show that HALT would be also decidable:

Suppose M_{ACCEPT} is a TM deciding ACCEPT.

Here is a description of a TM deciding HALT:

"Given ⟨M, x⟩, run $M_{ACCEPTS}$(⟨M, x⟩). If it accepts, then accept.

Reverse the accept & reject states in ⟨M⟩, forming ⟨M'⟩.

Run $M_{ACCEPTS}$(⟨M', x⟩). If it accepts (i.e., M rejects x), then accept.

Else reject."

New Proof Strategy summarized:

Want to show:

Problem L is undecidable

New Proof Strategy:

Deciding L is at least as hard as deciding HALT

$\Longleftrightarrow$

HALT would be easy if L were easy

$\Longleftrightarrow$

HALT reduces to L

$\Longleftrightarrow$

HALT $\leq_T$ L

Reductions

Definition:

Language A *reduces to* language B means:

"It is possible to decide A using an algorithm for deciding B as a subroutine."

Notation: $A \leq_T B$ (T stands for Turing).

Think, "A is no harder than B".

Reductions

Fact:

Suppose $A \leq_T B$; i.e., A reduces to B.

If B is decidable, then so is A.

We actually used the contrapositive:

Fact:

Suppose $A \leq_T B$; i.e., A reduces to B.

If A is undecidable, then so is B.

Note that “$A \leq_T B$” is a stronger statement than proving that A is decidable under the assumption that B is decidable.

Reductions

Reductions are the main technique
for showing undecidability.

Interesting:
We use a positive statement, i.e., the existence of a reduction algorithm, in order to prove a negative (impossibility) result.

Reductions (HALT $\leq_T$ ACCEPT)

Theorem:

For ACCEPT = $\{\langle M, x\rangle \mid M$ is a TM which accepts $x\}$
we have HALT $\leq_T$ ACCEPT.

Proof:

Suppose M_{ACCEPT} is a subroutine deciding ACCEPT.

Here is a description of a TM deciding HALT:

"Given $\langle M, x\rangle$, run $M_{ACCEPTS}(\langle M, x\rangle)$. If it accepts, then accept.

Reverse the accept & reject states in $\langle M\rangle$, forming $\langle M'\rangle$.

Run $M_{ACCEPT}(\langle M', x\rangle)$. If it accepts (i.e., M rejects x), then accept.

Else reject."

More Reductions (ACCEPT $\leq_T$ ALL)

Theorem:

ALL = $\{\langle M \rangle \mid M$ accepts all strings$\}$ is undecidable.

(and therefore its compliment EMPTY is as well)

Proof: (ACCEPT $\leq_T$ ALL)

Suppose M_{ALL} is a subroutine deciding ALL.

Here is a description of a TM deciding ACCEPT:

"Given $\langle M, x \rangle$, write down the description $\langle M_x \rangle$ of a TM M_x which does this:

"Overwrite the input with x and then run M."

Call subroutine M_{ALL} on input $\langle M_x \rangle$. Accept if it accepts, reject otherwise"

(Note that M_x behaves the same on all inputs and in particular we have that M_x accepts all strings if and only if M accepts x.)

More Reductions (ACCEPT $\leq_T$ EMPTY)

Theorem:

We also have ACCEPT $\leq_T$ EMPTY.

Proof: (ACCEPT $\leq_T$ EMPTY)

Suppose M_{EMPTY} is a subroutine deciding EMPTY.

Here is a description of a TM deciding ACCEPT:

"Given $\langle M, x\rangle$, write down the description $\langle M_x\rangle$ of a TM M_x which does this:

"Overwrite the input with x and then run M."

Call subroutine M_{EMPTY} on input $\langle M_x\rangle$. Reject if it accepts else reject."

More Reductions (ALL,EMPTY $\leq_T$ EQUIV)

Theorem:

EQUIV = $\{\langle M,M' \rangle \mid L(M) = L(M')\}$ is undecidable.

Proof: (ALL $\leq_T$ EQUIV and EMPTY $\leq_T$ EQUIV)

Suppose M_{EQUIV} is a subroutine deciding EQUIV.

Here is a description of a TM deciding ALL:

"Given $\langle M \rangle$ write down the description $\langle M' \rangle$ of a TM M' which always accepts / rejects.

Then call subroutine M_{EQUIV} on input $\langle M,M' \rangle$."

Poll – Test your Intuition

We just showed:

HALT $\leq_T$ ACCEPT $\leq_T$ EMPTY $\leq_T$ EQUIV

and

ACCEPT $\leq_T$ ALL $\leq_T$ EQUIV

Which of the following, do you believe also hold?

HALT $\leq$ EMPTY

HALT $\leq$ EQUIV

EMPTY $\leq$ ACCEPT

EQUIV $\leq$ EMPTY

EQUIV $\leq$ HALT

Poll – Test your Intuition

We just showed:

HALT $\leq_T$ ACCEPT $\leq_T$ EMPTY $\leq_T$ EQUIV

and

ACCEPT $\leq_T$ ALL $\leq_T$ EQUIV

Which of the following, do you believe also hold?

HALT $\leq$ EMPTY

HALT $\leq$ EQUIV

EMPTY $\leq$ ACCEPT

EQUIV $\leq$ EMPTY

EQUIV $\leq$ HALT

More Reductions (EMPTY $\leq_T$ HALT)

Theorem:

HALT, ACCEPT, EMPTY are all equally hard.

Proof: (EMPTY $\leq_T$ HALT)

Suppose M_{HALT} is a subroutine deciding HALT.

Here is a description of a TM deciding EMPTY:

"Given ⟨M⟩, write down the description ⟨M'⟩ of a TM M' which does this:

"For t=1 to ∞

run M on each string of length at most t for t steps

If any execution terminates and accepts then terminate (+ accept)"

Then call subroutine M_{HALT} on input ⟨M', ε⟩ but reverse the accept/reject."

More Undecidability

Theorem:

HALT, ACCEPT, EMPTY are all equally hard.

What about EQUIV and ALL?

Fun Fact #1:

EQUIV and ALL are harder than HALT and so are

TOTAL = {$\langle M \rangle$ | M halts on all inputs x}

FINITE = {$\langle M \rangle$ | L(M) is finite}

and in fact all these problems are equally hard.

Fun Fact #2:

There is an infinite hierarchy of harder and harder undecidable languages.

More Undecidability

Fun Fact:

There is an infinite hierarchy of harder and harder undecidable languages. (which however still only covers countably many languages)

How does one define / construct this hierarchy?

Look at TMs which have a subroutine/oracle that solves HALT. These oracle TMs can solve ACCEPT and other equivalent problems easily BUT they cannot decide if an oracle TM given to them halts. This makes the HALTing problem for oracle TMs even harder. …

Question:

Do all undecidable problems involve TM's?

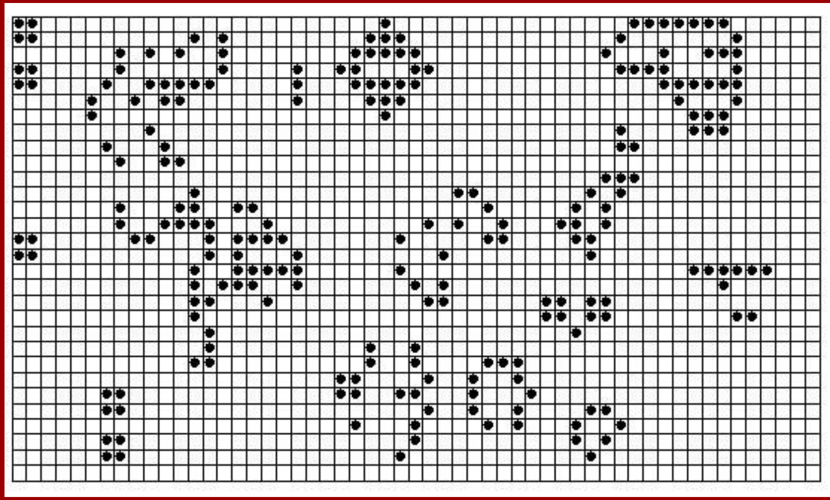
Answer:

No! Some very different problems are undecidable!

Cellular Automata

Input: A CA with its initial configuration.

E.g. a game of life pattern



Theorem: Deciding whether the input CA loops is an undecidable problem.

Post's Correspondence Problem

Input: A finite collection of "dominoes", having strings written on each half.

E.g.:

a	a	bcc
ab	cabc	c

Definition: A match is a sequence of dominoes, repetitions allowed, such that

top string = bottom string.

Post’s Correspondence Problem

Input: A finite collection of “dominoes”, having strings written on each half.

E.g.:

a	a	bcc
ab	cabc	c

Match:

a	bcc	a	bcc	= abccabcc
ab	c	cabc	c	= abccabcc

Post’s Correspondence Problem

Input: A finite collection of “dominoes”,
having strings written on each half.

Task: Output YES if and only if there is a match.

Theorem (Post, 1946): **Undecidable.**
There is no algorithm solving this problem.

(More formally, PCP = {⟨Domino Set⟩ : there’s a match}
is an undecidable language.)

Post’s Correspondence Problem

Input: A finite collection of “dominoes”,
having strings written on each half.

Task: Output YES if and only if there is a match.

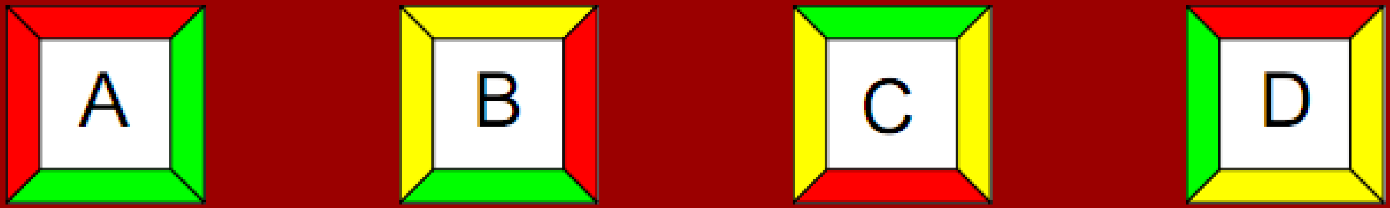
Theorem (Post, 1946): **Undecidable**.

Two-second proof sketch:

Given a TM M, you can make a domino set such that the only matches are execution traces of M which end in the accepting state. Hence ACCEPTS $\leq_T$ PCP.

Wang Tiles

Input: Finite collection of “Wang Tiles” (squares) with colors on the edges. E.g.,



Task: Output YES if and only if it’s possible to make an infinite grid from copies of them, where touching sides must color-match.

Theorem (Berger, 1966): **Undecidable.**

Richardson's Problem

Input: A set S of rational numbers.

What you can do: Make an expression E using the numbers in S, the numbers π and $\ln(2)$, the variable x, and operations $+$, $-$, $\cdot$, sin, exp, abs.

Question: Can you make an E such that $E \equiv 0$?

Theorem (Richardson, 1968): **Undecidable.**

Mortal Matrices

Input: Two 21×21 matrices of integers, A & B.

Question: Is it possible to multiply A and B together (multiple times in any order) to get the 0 matrix?

Theorem (Halava, Harju, Hirvensalo, 2007):
Undecidable.

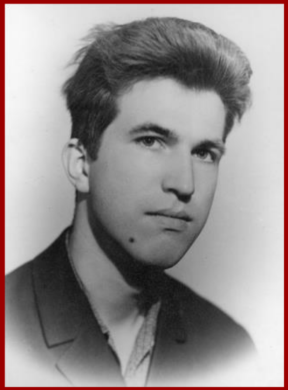
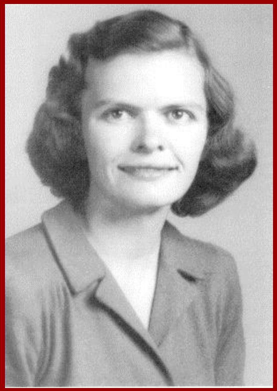
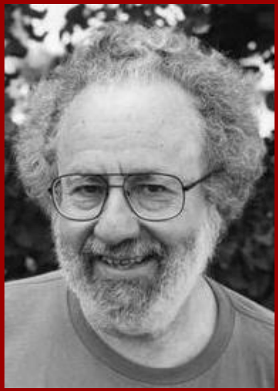
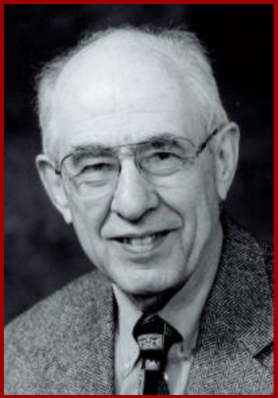
Hilbert's 10th problem

Input: Multivariate polynomial w/ integer coeffs.

Question: Does it have an integer root?

Theorem (1970): **Undecidable.**

Matiyasevich Robinson Davis Putnam

Hilbert's 10^{th} problem

Input: Multivariate polynomial w/ integer coeffs.

Question: Does it have an integer root?

Undecidable.

Question: Does it have a real root?

Decidable.

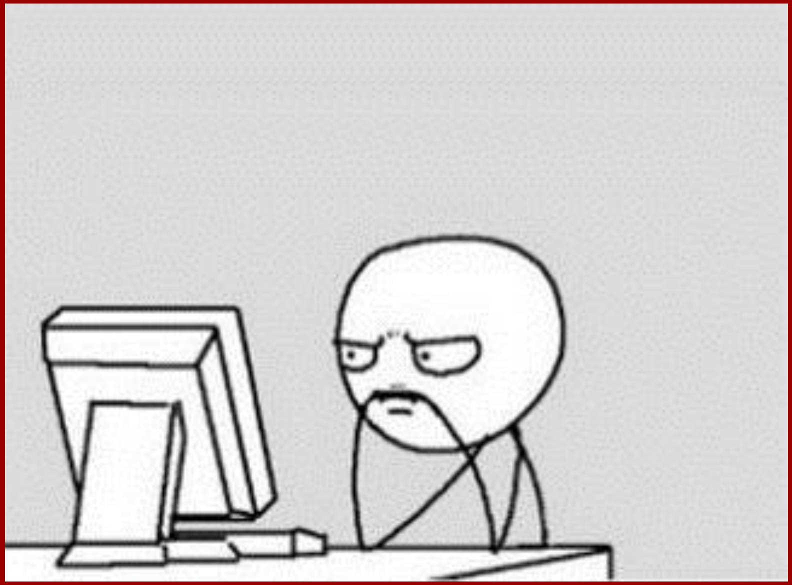
Tarski, 1951.

Question: Does it have a rational root?

Not known if it's decidable or not.

Definitions:

Halting and other Problems



Study Guide

Theorems/proofs:

Undecidability of HALT

many reduction proofs

Practice:

Diagonalization

Reductions

Programming with TM’s