SELF-KNOWLEDGE AND A PROGRAM THAT PRINTS ITSELF

CAN WE EVER REALLY KNOW OURSELVES?

- YOUR BRAIN HAS 10 BILLION NEURONS.
- IS IT POSSIBLE TO

  KNOW / REMEMBER / REPRESENT

  ALL THOSE NEURAL CONNECTIONS IN THAT SAME BRAIN?
- ON AVERAGE, YOU HAVE LESS THAN 1 NEURON TO USE TO STORE THE INFO ABOUT EACH NEURON.

CAN A PROGRAM EVER KNOW / REPRESENT / PROCESS ON ITSELF?

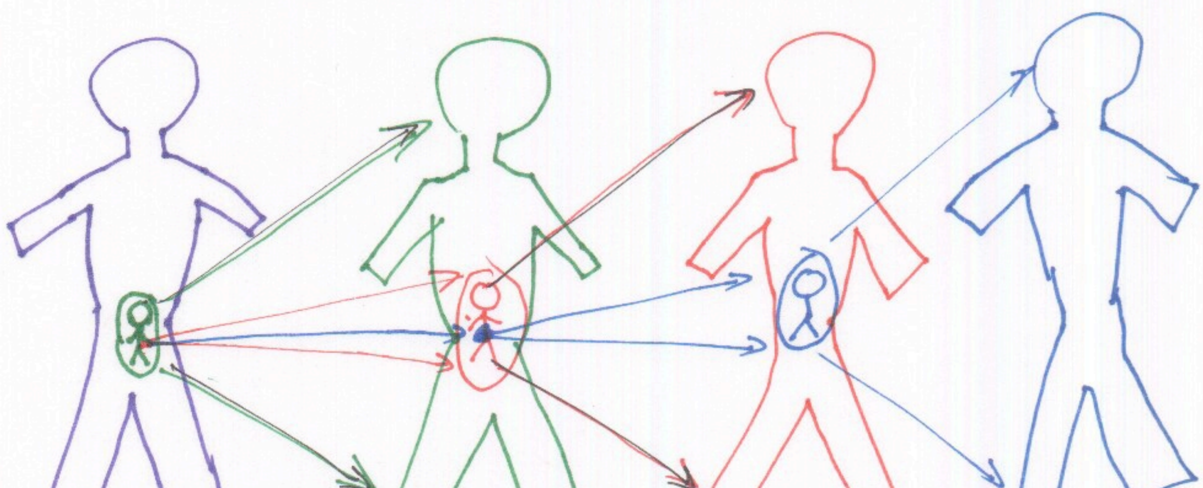
YES!

A PARADOX IN THE EARLY DAYS OF BIOLOGY:

FROM ONE ANIMAL, IS BORN ANOTHER "COPY."

EACH PARENT CONTAINS ~~A~~ SOME TINY, TINY LITTLE PEOPLE, WHO MERELY GROW INTO FULL INDIVIDUALS.

BUT WHAT ABOUT THEIR CHILDREN?



.... NAH!

COMPUTER VIRUSES.

MUST MAKE A COPY OF THEMSELVES.

FOR SIMPLICITY, JUST IMAGINE
PRINTING A COPY OF YOURSELF.

APPROACH #1

```
USE A POINTER, P.
MAKE P POINT TO FIRST
    BYTE OF THE CODE
FOR 157 TIMES...
    PRINT *P
    P ← P+1
END
```

SOLUTION USED BY BIOLOGICAL LIFE:

DNA: "CODE" WHICH IS "EXECUTED" TO PRODUCE PROTEINS

DNA: USED AS "DATA" WHEN DNA IS COPIED.

COUNTER MEASURE:

THE O.S. FLAGS MEMORY AS:
* EXECUTABLE
* READ/WRITE.

THE MACHINE PREVENTS ALL PROGRAMS FROM "READING" CODE BYTES.

APPROACH #2

RECURSION THEOREM
COMPUTER VIRUSES.

GOAL: WRITE A PROGRAM THAT PRINTS ITSELF.

OUR PROGRAMMING LANGUAGE?

VARIABLES, ASSIGNMENT

STRINGS

PRINT STATEMENTS.

```
x ← 'world'
PRINT 'Hello'
PRINT x
```

TURING MACHINES: The tape is used as MEMORY.

MODERN COMPUTERS: Variables are used for MEMORY.

WE'LL IGNORE

* PRINTING NEWLINES
* ESCAPING QUOTES.

```
'4 o'clock'
'4 o\'clock'
9H4 O'CLOCK
```

$\underbrace{\text{4 O'CLOCK}}_{\text{9 chars}}$

PROGRAMS THAT PRINT THEMSELVES ARE CALLED "QUINES".

PRINT 'Hello'

PRINT 'PRINT 'Hello''

PRINT 'PRINT 'PRINT 'Hello'''

PRINT 'PRINT 'PRINT 'PRINT 'Hello''''

⋮

STEP 1:

```
X ← '    ?    '

PRINT X
```

STEP 2:

```
X ← '          '
PRINT 'X ← ''
PRINT X
PRINT '''
```

STEP 3:

```
X ← '          '
PRINT 'X ← ''
PRINT X
PRINT '''
PRINT X
```

~~STEP 2 4~~

STEP 4:

X ← 'PRINT 'X←'' PRINT X PRINT ''' PRINT X'

```
PRINT 'X←''
PRINT X
PRINT '''
PRINT X
```

EXECUTING THIS:

X ← 'PRINT 'X←'' PRINT X PRINT ''' PRINT X'

```
PRINT 'X←''
PRINT X
PRINT '''
PRINT X
```

IN THE "C" LANGUAGE

Note: ASCII 34 = double quote char (")

```
printf("Hello %c %s %c", 34, "world", 34);
```

Hello "world"

```
main () {
    char *x = "                ";
    printf(x, 34, x, 34);
}
```

```
main () {
    char *x = %c %s %c;
    printf(x, 34, x, 34);
}
```

QUINE:

A T.M. THAT PRINTS ITS OWN DESCRIPTION

APPROACH TO IMPLEMENTING "QUINE" ON A T.M.

BREAK THE TASK INTO 2 STEPS.

A really long string of 0's and 1's

STEP A:

X ← '<step B>'

STEP B:

PRINT OUT <STEP A>
(We get to use X!)

PRINT OUT <STEP B>
(We get to use X!)

GOALS:
EACH STEP IS A T.M./SUBROUTINE.
EXECUTE STEP A, THEN STEP B.
THIS SHOULD WRITE OUT

$$\underbrace{\langle \text{STEP A} \rangle \mid \langle \text{STEP B} \rangle}_{\text{OUR "QUINE" TM.}} \mid \cdots$$

Let P_w be a Turing Maching that prints out w.

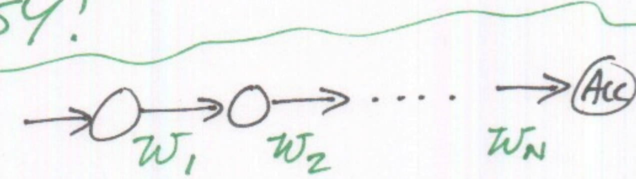
P_{10110} writes 10110 on the tape.

Let $\langle P_w \rangle$ be the representation of Turing Machine P_w

---

Given a string w, could you build a T.M. to write out w?

SURE! EASY!

P_w looks like 

This task is clearly computable. A computable function g does it

$$g: \Sigma^* \to \Sigma^*$$

$$g(w) \to \langle P_w \rangle$$

Given a string

Representation of a simple TM to write w on the tape.

STEP A:

Write a long string on the tape.
Call this string X.

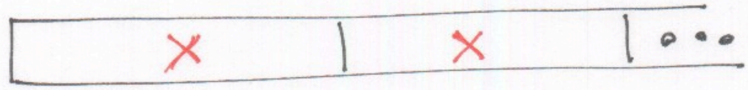
The string will turn out to be <STEP B>
We dont know the string yet.
We can't finish coding STEP A yet.
Once we know X, we can easily finish coding step A.

IN FACT, WE CAN JUST USE $q(x)$ TO DO THE ENTIRE CODING OF STEP A, ONCE WE KNOW X !!

STEP B:

WHEN IT STARTS, THE TAPE CONTAINS A LONG STRING X.

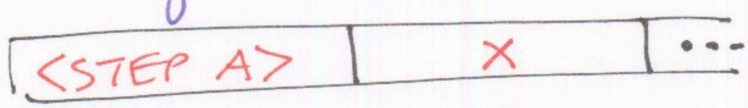
MAKE A COPY OF X:



USE g AS A SUBROUTINE.

CALL IT ON X TO COMPUTE $g(x)$

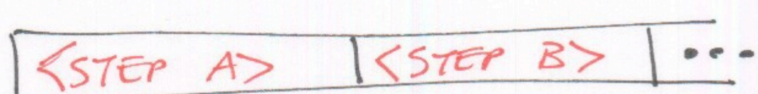
Note: $g(X) =$ <STEP A>



AND NOW THE MAGIC!

LET X BE THE DESCRIPTION OF STEP B. X = <STEP B>

OUR TAPE CONTAINS:



WE ARE DONE CODING STEP B.
WE NOW KNOW <STEP B>.
GO BACK AND FINISH UP CODING STEP A.

THE RECURSION THEOREM

OPERATIONS YOU MIGHT DO ON A T.M.

* COUNT THE NUMBER OF STATES.
* CHECK TO SEE IF ACCEPT IS EVEN REACHABLE FROM INITIAL STATE.
* CHECK TO SEE IF THE T.M. ACCEPTS w.
* etc.

APPROACH:

Build a TM to do this

$$t(\langle M \rangle, w)$$

Perhaps you'd like to run this TM on itself:

$$t(\langle t \rangle, w)$$

⇒ You have to pass t its own description.

Is there any other way?

YES

You can build a T.M. r that does exactly what t would do if passed a description of itself.

$$r(w) = t(\langle r \rangle, w)$$

RECURSION THEOREM

If you can build ~~a~~ t, then there exists another TM that does the same thing but COMPUTES ITS OWN DESCRIPTION instead of having to take it as an input.

Recursion Theorem

Let T be some Turing machine that computes some function t.

$$t : \Sigma^* \times \Sigma^* \to \Sigma^* \quad / \quad t(\langle M \rangle, w)$$

Then there will always exist another Turing Machine R that does the same thing as t when t is applied to a description of itself.

That is R computes the function $r : \Sigma^* \to \Sigma^*$

and for every w ...

$$r(w) = t(\langle R \rangle, w)$$

THE RECURSION THEOREM: SOME RESULTS

Bottom Line

Whenever we are specifying a Turing Machine algorithm, We can say

Obtain, via the recursion theorem, a description of self, <SELF>.

Or, more concisely,

$X \leftarrow \langle SELF \rangle$

The Quine Program (short version):

```
X ← <SELF>
PRINT X
```

The recursion theorem says: this is an entirely legal T.M. program.

THEOREM (PREVIOUSLY PROVEN)

$$A_{TM} = \{\langle M, w\rangle \mid M \text{ is a T.M that ACCEPTS } w\}$$

IS UNDECIDABLE

NEW PROOF

Assume H decides A_{TM}.

CONSTRUCT MACHINE B:

INPUT: w

$X \leftarrow \langle B\rangle$ GET <SELF> via Recursion THEOREM

Run H on $\langle B\rangle, w$

Do the opposite:

if H ACCEPTS THEN REJECT

if H REJECTS THEN ACCEPT

Running B on input w does the opposite of what H says ~~#~~ B does. Therefore H is wrong. H can't be deciding A_{TM}.

THE "SIZE" OF A TURING MACHINE:

$|\langle M \rangle|$ = Number of symbols in the description of M.

DEFINITION

A Turing Machine M is "MINIMAL" if there is no Turing machine equivalent to M with a shorter description.

What about the set of MINIMAL Turing Machines?

THEOREM

The set

$$MIN_{TM} = \{\langle M\rangle \mid M \text{ is a "minimal" Turing Machine}\}$$

is NOT TURING-RECOGNIZABLE.

PROOF

Assume MIN_{TM} is Turing Recognizable.
Then $\exists$ an enumerator E that will list them out.
Use E to construct a new machine "C", as follows:

C

INPUT: w

ALGORITHM:

Obtain, via Recursion Theorem, a description of self, <C>.

Run E until it prints out a machine D with a longer description than C.

Simulate D on w.

MIN_{TM} is infinite, so we'll eventually find a machine D which is longer.
C simulates D; therefore they are equivalent.
D cannot be minimal
CONTRADICTION!

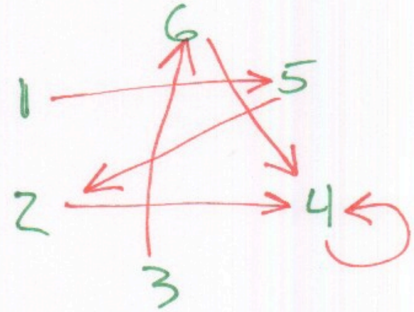
FIXED POINTS AND TURING MACHINES

DEFINITION

A "FIXED POINT" OF A FUNCTION IS A VALUE THAT IS UNCHANGED BY REPEATED APPLICATIONS OF THE FUNCTION.

Example

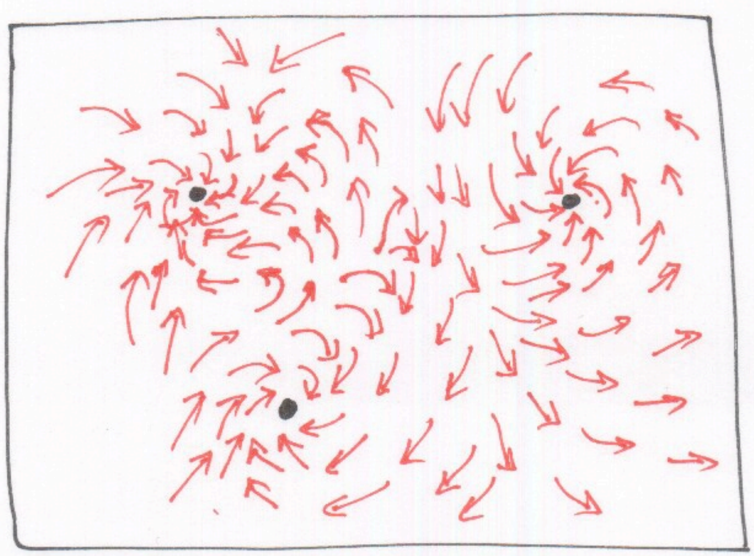
x	$f(x)$
1	5
2	4
3	6
4	4
5	2
6	4



Example

SPACE OF VALUES

FUNCTION • → •



FUNCTIONS: Computable Transformations.

VALUES: Turing Machine Descriptions.

ANOTHER VERSION OF THE RECURSION THEOREM (The "Fixed Point" Version)

For any transformation function ~~[illegible]~~ on Turing Machines,

There will always exist a Turing Machine which is unchanged by the Transformation.

THEOREM

Let t be any computable function

$$t : \Sigma^* \to \Sigma^*$$

[We can apply t to descriptions of Turing Machines: $t(\langle M \rangle)$]

Then there is a Turing Machine F such that

$t(\langle F \rangle)$ is equivalent to F.

PROOF Let F be the following TM:

INPUT: w

ALGORITHM:

OBTAIN DESCRIPTION OF SELF, $\langle F \rangle$.

COMPUTE $t(\langle F \rangle)$. TO OBTAIN A NEW Turing Machine, G

SIMULATE G on w.

G and F are equivalent.

$\langle G \rangle = t(\langle F \rangle)$

So $\langle F \rangle$ and $t(\langle F \rangle)$ are equivalent.