6.1 The Recursion Theorem

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?

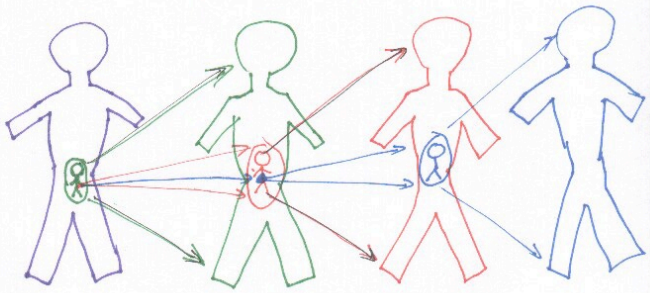
Yes!

A PARADOX IN THE EARLY DAYS OF BIOLOGY:

FROM ONE ANIMAL, IS BORN ANOTHER "COPY."

EACH PARENT CONTAINS ~~&~~ 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
```

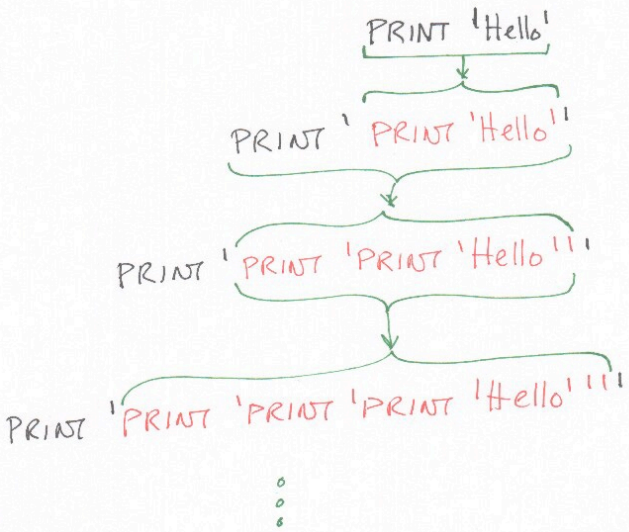
TURING MACHINES: The tape is used as MEMORY.
MODERN COMPUTERS: Variables are used for MEMORY.

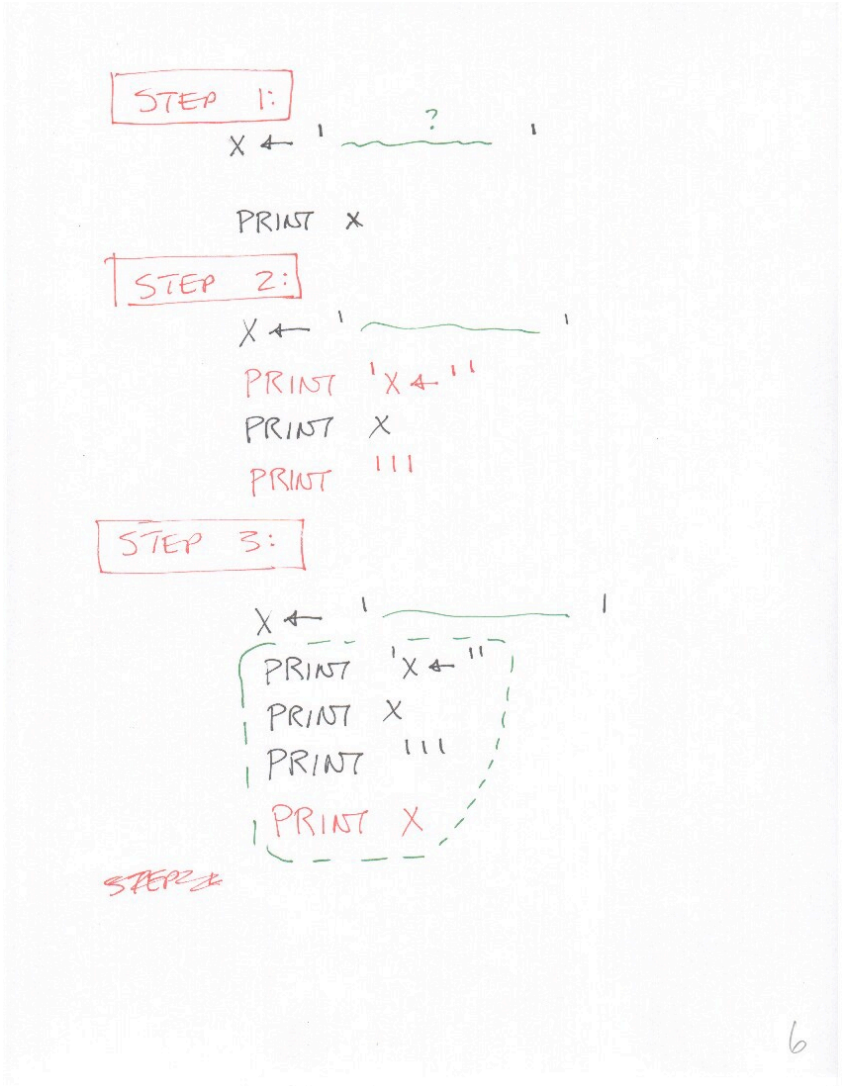
WE'LL IGNORE
* PRINTING NEWLINES
* ESCAPING QUOTES.

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

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

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
```

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);
}
```

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:

$X \leftarrow$ '<step B>'

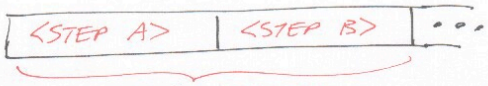
Step B:

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



our "Quine" TM.

Let P_w be a Turing Maching that prints out w.

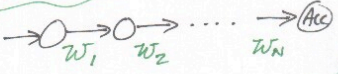
P_{10110} Writes 10110 on the tape.

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.

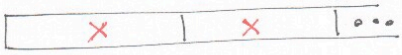
The string will turn out to be ⟨STEP B⟩
We don't know the string yet.
We can't finish coding STEP A yet.
Once we know X, we can easily finish coding step A.

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

STEP B:

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

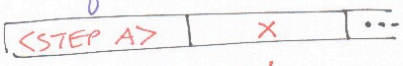
MAKE A COPY OF X:



USE g AS A SUBROUTINE.

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

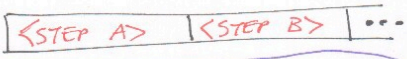
Note: $g(X) = \langle \text{STEP A} \rangle$



AND NOW THE MAGIC!

LET X BE THE DESCRIPTION OF STEP B. $X = \langle \text{STEP B} \rangle$

OUR TAPE CONTAINS:



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

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 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...

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

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 ← <SELF>

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, $\langle C \rangle$.
Run E until it prints
out a machine D with
a longer description than C.
Simulate D on w.

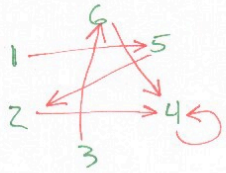
MIN_{TM} is infinite, so we'll eventually
find a machine D which is longer.
C simulates D; therefore they are equivalent.
D cannot be minimal
CONTRADICTION!

DEFINITION

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

Example

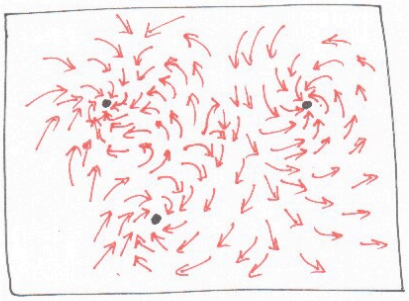
x	f(x)
1	5
2	4
3	6
4	4
5	2
6	4



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 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.