

Computation Histories and PCP

(not Probabilistically Checkable Proofs)
(also not Angel Dust)

CS 81: Computability and Logic

November 28, 2011

UNDECIDABILITY, SO FAR

1. Acceptance for TMs is semidecidable but not decidable.

$$A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ a TM, } M \text{ accepts } w \}$$

2. *Non-Acceptance* for TMs is not semidecidable (hence not decidable).
3. TM Halting is semidecidable but not decidable.

$$H = \{ \langle M, w \rangle \mid M \text{ a TM, } M \text{ halts on } w \}$$

REDUCTIONS

Given problems P and Q , we say that

$$P \leq Q$$

if a solution to Q would let us solve P as well.

- ✓ I.e., (P is “not fundamentally more difficult” Q .)
- ✓ We say “ P **reduces to** Q .”

In Math, we often show we can solve a problem X by taking advantage of previously-solved problem Q (i.e., prove $X \leq Q$)

In Theocomp, we typically prove a problem X *hard* by showing a solution to X would also solve the hard problem P (i.e., prove $P \leq X$).

REDUCTIONS

To prove that P reduces to Q ($P \leq Q$), it suffices to prove:

- ✓ If we have a solver for Q , then we can use it to solve any instance of P .
- ✓ I.e., show you could construct a P -solver if you could make calls to a Q -solving subroutine.

Commonly, we instead prove a “mapping reduction”:

- ✓ For every instance of P , we can construct an instance of Q with the same yes/no answer.

Why is this enough?

Computation Histories

- ✓ Step-by-step recordings of a TM computation.
- ✓ In practice, a “trick” for showing more problems not decidable.

States as Strings

- ✓ We can describe the configuration of any TM using a string $C = xqy \in \Gamma^*$.
 - ✓ $x \in \Gamma^*$ = symbols to the left of head
 - ✓ $q \in Q$ = current control state
 - ✓ $y \in \Gamma^*$ = symbols under and to the right of head
- ✓ Over the course of a computation, we have
 - ✓ $\dots \Rightarrow x_1q_1y_1 \Rightarrow x_2q_2y_2 \Rightarrow x_3q_3y_3 \Rightarrow \dots$
 - ✓ If the TM halts, we can represent the whole history of the computation as a single (finite) string!
 - ✓ Traditionally written $\#C_1\#C_2\#C_3\#\dots\#C_n\#$

Checking a History

- ✓ Checker: a Turing Machine C that, given $\langle M, h \rangle$, checks whether h is a history of TM M .
 - ✓ Consecutive states should be equal, except around the head (where the change corresponds to the transition table of M).
- ✓ Can check whether h is a halting (or an accepting) history by looking at the last control state.

Digression: LBAs

- ✓ In fact, the CH can be checked for validity by a less-than-general TM called an LBA.
 - ✓ LBA = “Linear Bounded Automaton,” a TM that can only use the part of the tape containing input.
 - ✓ An LBA can have a large tape alphabet and can “mark” tape cells. It just can’t grow its tape.
- ✓ More powerful than a DFA
 - ✓ Number of potential states grows with input size
 - ✓ DFA wouldn’t be able to check a computation history.

Accepting for LBAs

✓ $A_{\text{LBA}} = \{ \langle M, w \rangle \mid M \text{ a LBA accepting } w \}$ is decidable.

✓ Proof: When running M on w , there are at most

$$n := |K| \times |\Gamma|^{|w|} \times |w|$$

distinct “states” during the computation.

✓ So,

✓ Run M on w .

✓ If computation takes longer than n steps, it's in an infinite loop; M doesn't accept w .

Emptiness for LBAs

✓ $E_{LBA} = \{ \langle M \rangle \mid M \text{ a LBA, } L(M) = \emptyset \}$ isn't decidable.

✓ Proof: $A_{TM} \leq E_{LBA}$.

All_{CFG}

All_{CFG} = { $\langle G \rangle$ | G a CFG, $L(G) = \Sigma^*$ } is undecidable.

Proof: $A_{TM} \leq All_{CFG}$.

- ✓ Key: given $\langle M, w \rangle$ create a PDA/CFG for strings that **aren't** accepting computation histories!
 - ✓ PDA accepts strings that
 - ✓ Don't start with q_0w
 - ✓ Or, don't end with $xq_{accept}y$
 - ✓ Or, two successive configurations don't match properly
 - ✓ Hack: need to reverse every other configuration.
- ✓ The grammar for this PDA is Σ^* iff M, w has no finite, accepting history.

Eq_{CFG}

$Eq_{CFG} = \{ \langle G_1, G_2 \rangle \mid L(G_1) = L(G_2) \}$ is undecidable.

Proof: $All_{CFG} \leq Eq_{CFG}$.

Post Correspondence Problem

Emil Post



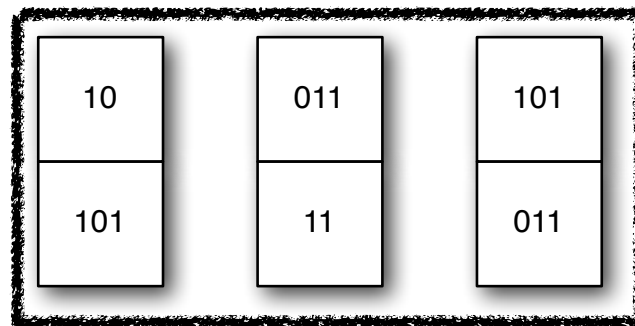
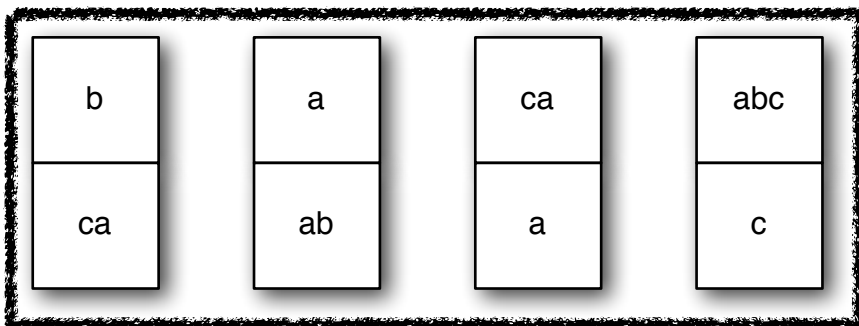
- ✓ Named after logician Emil Post (1897-1954)
- ✓ studied fundamental models of computation
- ✓ “scooped” by Gödel, Turing, and Church

Why PCP?

- ✓ Trivial problem to state
 - ✓ Looks nothing like Turing Machines
 - ✓ A child can understand it
 - ✓ Superficially, doesn't look that hard
- ✓ Can reduce PCP to other problems, showing them undecidable

PCP

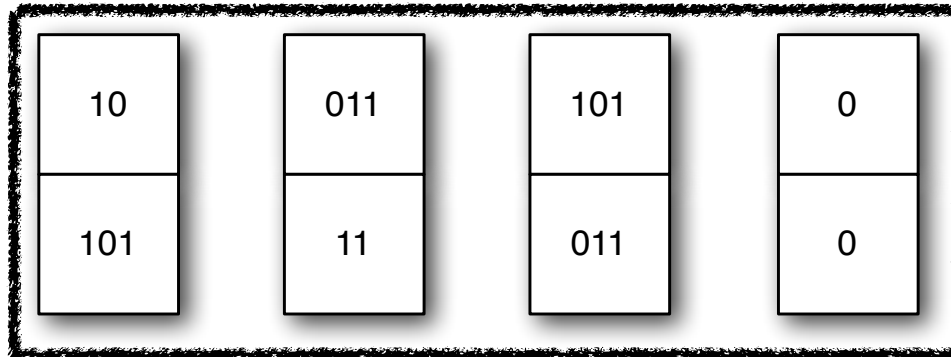
- ✓ Given a set of “dominos” (pairs of strings), find a finite sequence of dominos, repeats allowed, where the top line and bottom line spell out the same string.



- ✓ Theorem: PCP is undecidable.
- ✓ Proof: Halting \leq MPCP \leq PCP

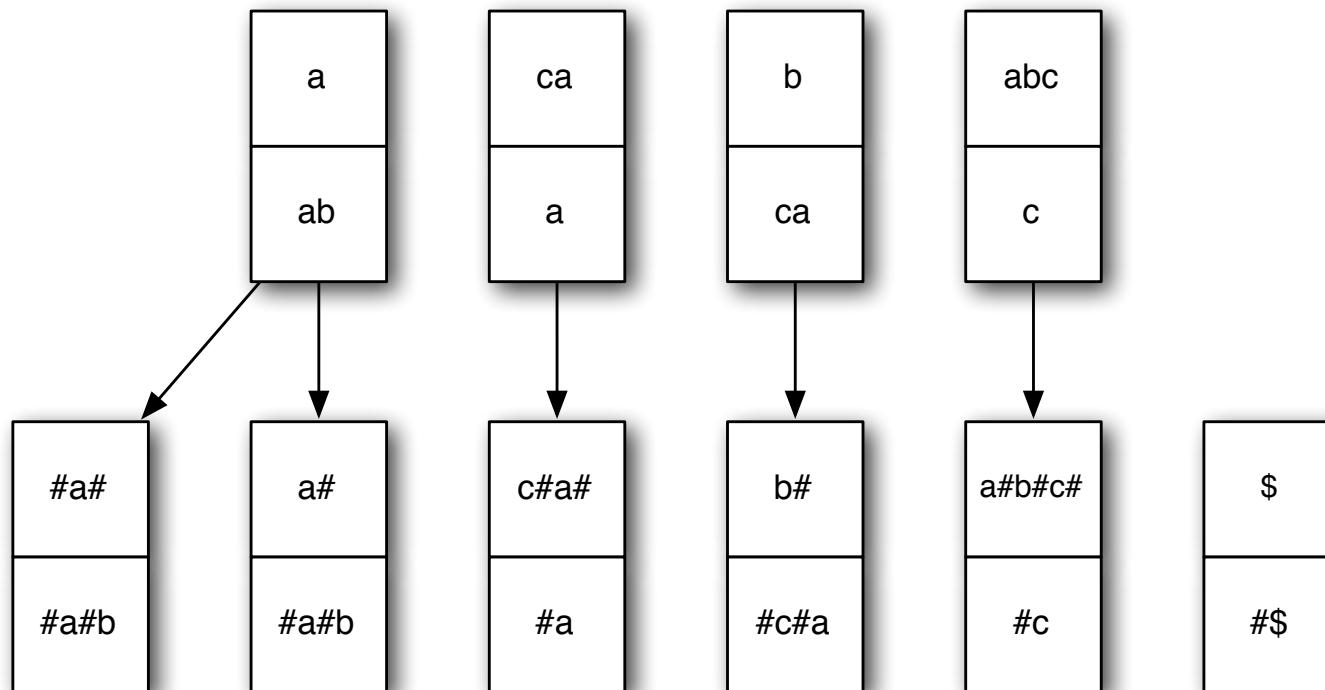
Modified PCP (MPCP)

- ✓ Like PCP, but solution starts with first domino.
- ✓ The following **MPCP** instance has no solution.



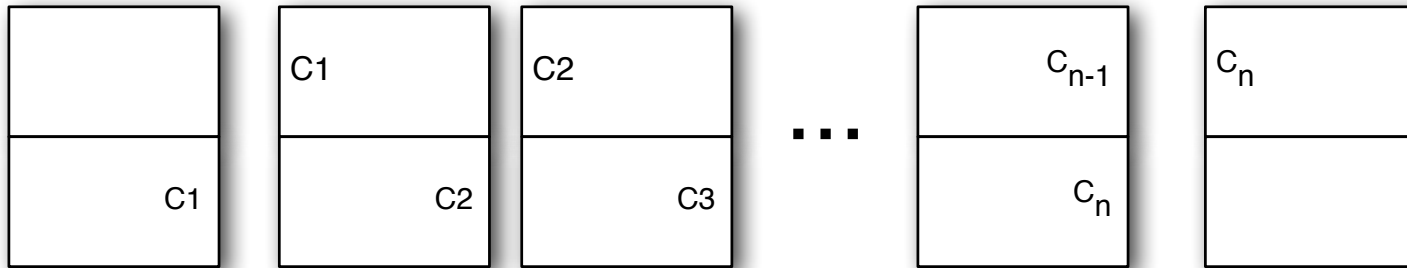
MPCP \leq PCP

✓ Given an instance of MPCP, solve by translating dominos and doing PCP.



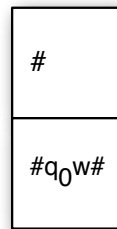
Halts \leq MPCP

- ✓ Idea: if a TM halts, it has a computation history
- ✓ Given a $\langle M, w \rangle$, construct dominos whose solution would yield such a history (on top and bottom); use MPCP-solver to check for a solution.

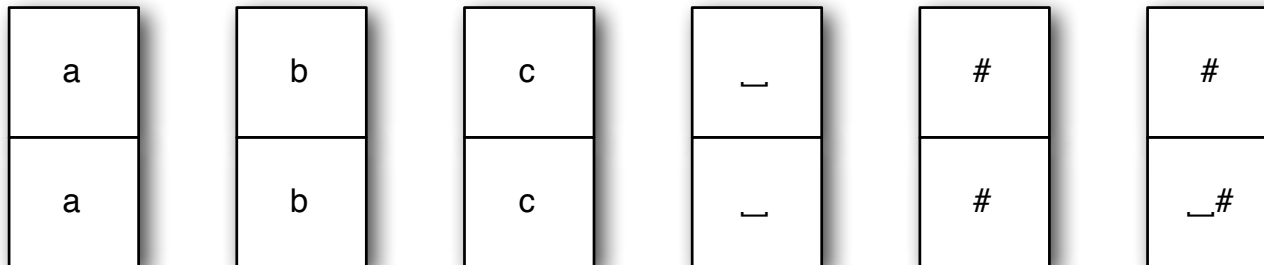


The Dominos

- ✓ First domino: set up the initial state

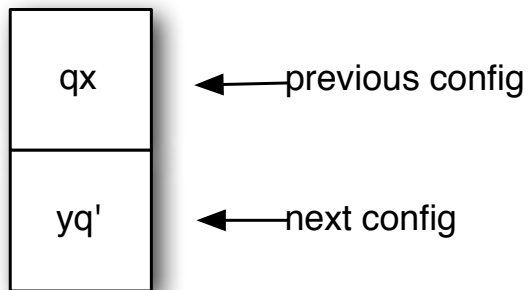


- ✓ Helper dominos: copy unchanged parts of the tape from C_i to C_{i+1} . (optional: expand the tape)

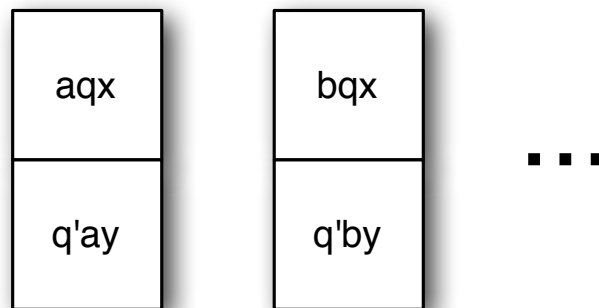


More Dominos

✓ For each transition $q, x \rightarrow q', y, R$



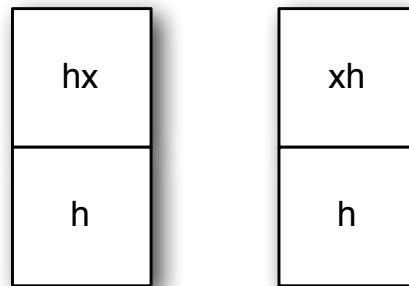
✓ For each transition $q, x \rightarrow q', y, L$



Completion Dominos

✓ Technical “Trick”

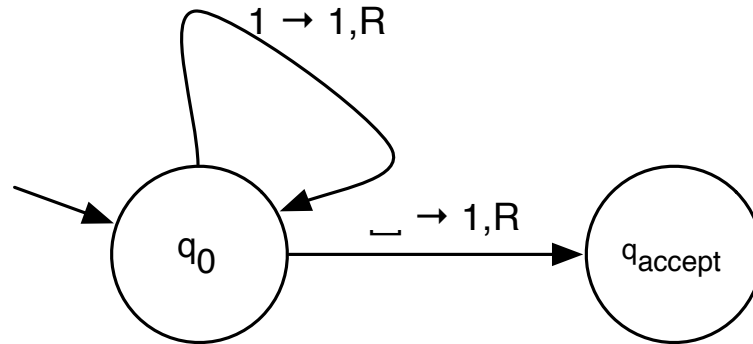
- ✓ When we reach a halting state h , we delete the configuration (one symbol at a time) until it disappears
- ✓ For each halting state h , and each symbol x , we need



✓ Finally,



Example: $w=11$



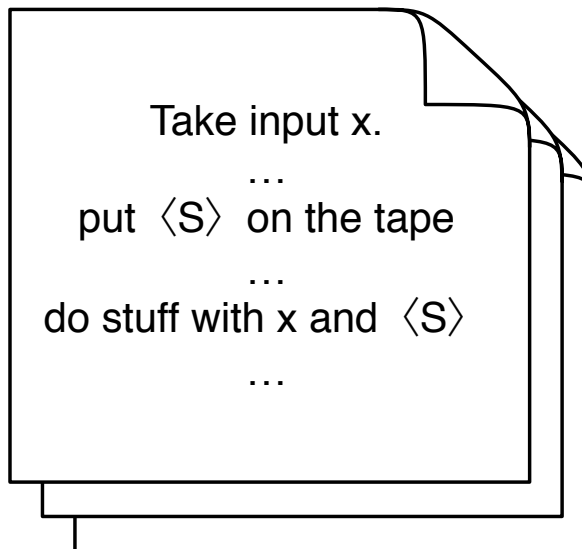
#	1	_	#	#
#q ₀ 11#	1	_	#	_#
q ₀ 1	q ₀ _	1q _a	q _a 1	q _a ##
1q ₀	1q _a	q _a	q _a	#

The Recursion Theorem

Recursion Theorem (Kleene 1938)

✓ **Informally:** A program can have access to its own description (code).

Code for TM S



Recursion Theorem (Kleene 1938)

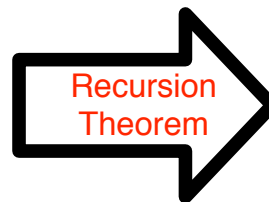
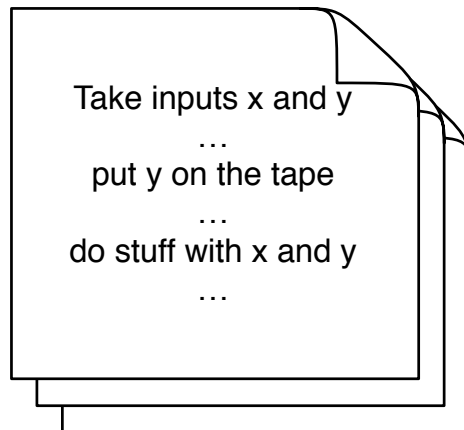
✓ **Formally:**

If R is a Turing machine computing a binary function $R(x, y)$, then there is a Turing machine S computing a unary function such that:

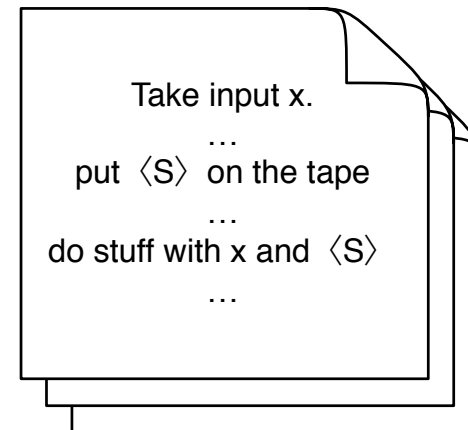
$$S(x) = R(x, \langle S \rangle)$$

where $\langle S \rangle$ is the description of S itself.

Code for TM R



Code for TM S



Application: Undecidability of A_{TM} (again)

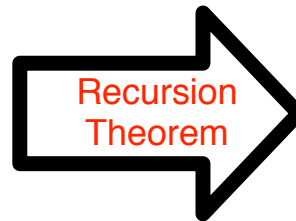
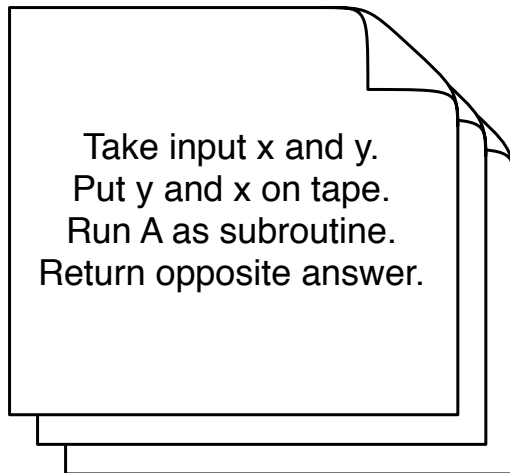
- ✓ Suppose A_{TM} were decidable using TM A .

- ✓ Define $M(x)$ as follows:
 1. Take input x ;
 2. Use A to decide whether M accepts x , i.e., whether $\langle M, x \rangle \in A_{TM}$
 3. Do the opposite.

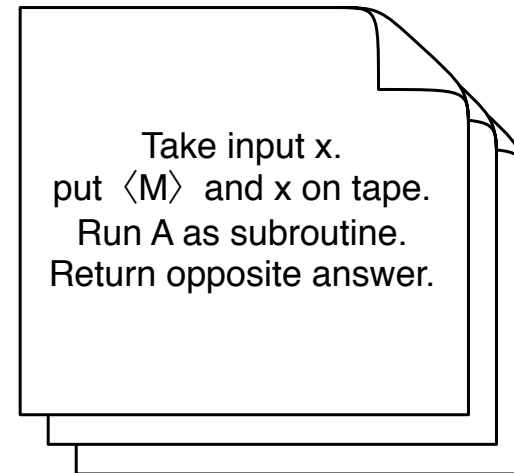
- ✓ M can't exist, so A must not exist. QED

Recursion Theorem to the rescue

Code for TM R

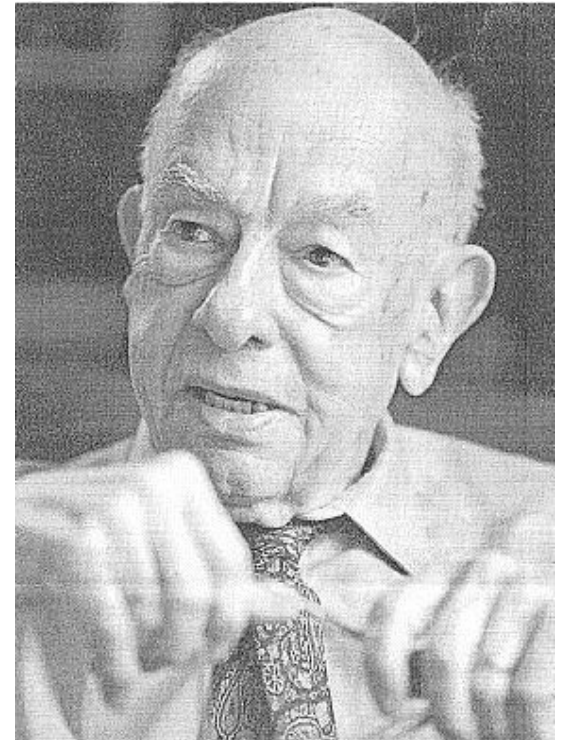


Code for TM M



Self-Printing Machines

- ✓ Even if a machine is not given a handle to its own code on its tape at the outset, there are ways for it to construct it.
- ✓ Such programs are now called “Quines”



Willard Van Orman Quine 1908-2000

A Java Quine

<http://www.knet.ro/lsantha/>

```
class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'}')};static String
s="class Q{public static void main(String[]v){char
c=34;System.out.print(s+c+s+c+';'+'}')};static String
s=";}
```

✓ javac Q.java

✓ java Q

```
class Q{public static void main(String[]v){ char
c=34;System.out.print(s+c+s+c+';'+'}')};static String
s="class Q {public static void main(String[]v){ char
c=34;System.out.print(s+c+s+c+';'+'}')};static String
s=";}
```

C and C++ Quines (authors unknown)

```
char f[] =  
"char f[] =%c%c%s%c;%cmain() {printf(f,10,34,f,34,10,10);}%c";  
main() {printf(f,10,34,f,34,10,10);}
```

```
#include <iostream>  
#define a(b) std::cout<<"#include <iostream>\n#define a(b) "<<#b<<"\nmain(){a("<<#b<<");}"  
main(){a(std::cout<<"#include <iostream>\n#define a(b) "<<#b<<"\nmain(){a("<<#b<<");}");}
```

rex Quine

(by a Pomona College Student)

```
a="\\";aa="a";
b="\\";bb="b";
c="=";cc="c";
d="print(

    aa,c,  b, a,a,b,  f,
    aa,aa,  c,b,aa,b,f,g,bb,c,b,
    a,b,b,f ,bb,bb,c,b,bb,b,f,g,
    cc,c,b,c ,b,f, cc,cc,c,b,cc,
    b,  f,g ,dd      ,c,b,
    d,  b,f ,      g,g,
    dd,  dd,      c,b,
    dd,  b,f      ,g,ee
    ,c,b ,e,      b,f,
    ee,  ee,      c,b,
    ee,b,f,  g,ff,c,  b,f,
    b,f,ff,ff,c,b,ff,b,f,  g,gg,
    c,b,a    ,nn,b,
    f,gg     ,gg,c,b,
    gg,b,f,  g,nn,nn,c
    ,b,nn,b,  f,g,g,d,g);";
```

continued next col.

```
dd="d";
e=")";ee="e";
f="";ff="f";
g="\n";gg="g";
nn="n";

print(

    aa,c,  b, a,a,b,  f,
    aa,aa,  c,b,aa,b,f,g,bb,c,b,
    a,b,b,f ,bb,bb,c,b,bb,b,f,g,
    cc,c,b,c ,b,f, cc,cc,c,b,cc,
    b,  f,g ,dd      ,c,b,
    d,  b,f ,      g,g,
    dd,  dd,      c,b,
    dd,  b,f      ,g,ee
    ,c,b ,e,      b,f,
    ee,  ee,      c,b,
    ee,b,f,  g,ff,c,  b,f,
    b,f,ff,ff,c,b,ff,b,f,  g,gg,
    c,b,a    ,nn,b,
    f,gg     ,gg,c,b,
    gg,b,f,  g,nn,nn,c
    ,b,nn,b,  f,g,g,d,g);
```

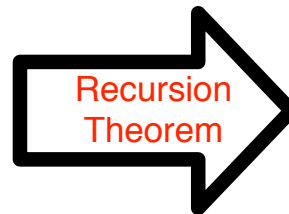
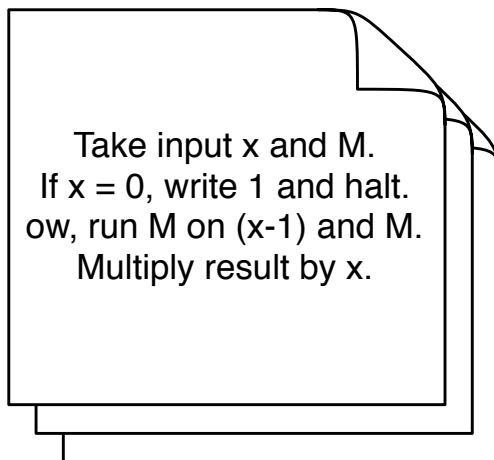
Applications of Quines

- ✓ Entertainment
- ✓ Computer viruses
- ✓ Artificial life?

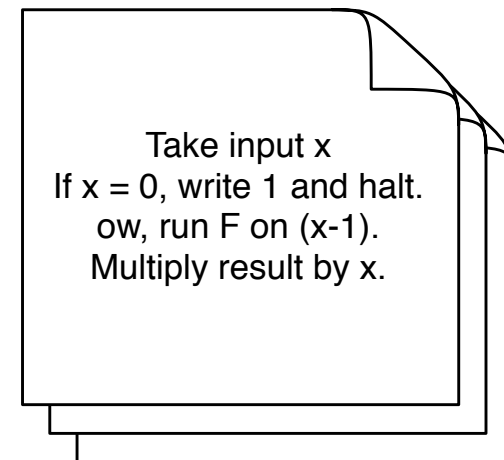
Recursion from the Recursion Theorem

- ✓ If you have $\langle M \rangle$ on the tape, you can run it.
- ✓ A machine M can put $\langle M \rangle$ on the tape.
- ✓ Therefore, a TM can compute “recursively” in the modern sense.

Code for TM R



Code for TM F



Theorem

✓ The language $\text{MIN} = \{ \langle M \rangle \mid M \text{ is minimal} \}$
is not even semidecidable.

Minimal

=

no machine with the same language has
a strictly smaller description.

Proof

- ✓ Suppose MIN **were** semidecidable, hence enumerated by some machine E.
- ✓ Consider the TM C:
 - ✓ Take input x.
 - ✓ Start running enumerator E.
 - ✓ Stop when we produce a $\langle D \rangle$ that is **strictly longer** than $\langle C \rangle$
(This must happen. Why?)
 - ✓ Simulate D running on x.
- ✓ C cannot exist (why?) , so E cannot exist.
- ✓ Thus MIN cannot be semidecidable.