The Recursion Theorem

Robert M. Keller
Harvey Mudd College
April 2010

What is this?

- A fundamental result having to do with computability and programming languages.
- Another technique that can be used to get further undecidability results.
- It was introduced as a theorem by Kleene in 1938.

Recursion Theorem: Informal Statement

- A program can have access to its own description (code).

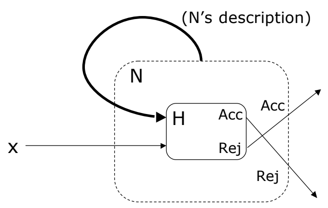
Another undecidabilty proof for A_{TM}

- This proof uses **self-reference** rather than **diagonalization**, as in our first proof.
- Suppose there is a TM that decides A_{TM}, to get contradiction.

Another undecidabilty proof for A_{TM}

- Suppose **H** is a TM that decides A_{TM}.
- Construct a machine N that behaves as follows:
  - With input x, run **H** on <**N**, x>.
    If **H** accepts, reject. If it rejects, accept.
- Note here that **N** is a description of this very machine.
- What will N do with input <N>?
  - If N accepts <N>, then H on <N, <N>> rejects, indicating N does **not** accept <N>.
  - If N does not accept <N>, then H on <N, <N>> accepts, indicating N accepts <N>.
  - Either way, we contradict the supposition of such an H.

Picture



Functional description

- H(<M, x>) = M(x) ? true : false
- N(x) = ¬H(<N, x>)
- So N(<N>)
  = ¬H(<N, <N>>) constr. of N
  = ¬(N(<N>) ? true : false) def. of H
  = N(<N>) ? false : true meaning of ¬
  = ¬N(<N>) meaning of ¬

(unless N<N> never halts,
but N must always halt if M does)

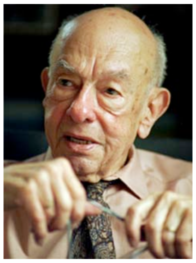
What is key in the previous proof?

- It relied on the ability of a machine to use its **own description** inside its own program.
- Is this strange?
  - An interpreter could use its own source code, for example, and interpret that code.
- Ok, but is it strange for TMs?

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”. (Would Quine like this?)

Willard Van Orman Quine (1908 - 2000)



A Java Quine (all one line. 34 is “)

```
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=";}
```

Proof

```
% 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=";}
```

source: http://www.knet.ro/lsantha/

Quines in C and C++ (authors unknown)

C Quine using numeric codes:

```
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);}
```

This C++ Quine does not use numeric codes:

```
#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<<");}");}
```

Example: A rex Quine

constructed 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 of self and others
- Computer viruses, worms, and other forms of mal-ware
  - To protect against these, it is important to know their characteristics and methods of operation.
- Artificial life
  - *cf.* von Neuman: "Theory of Self-Reproducing Automata"

Recursion Theorem Formalized

- If R is a Turing machine computing a binary function R(A, B), then there is a Turing machine S computing a unary function such that:

  S(A) = R(A, <S>)

  where <S> is the description of S itself.

From Programming Languages

- Compute a recursively-defined function **without actually using recursion**.
- This is not so hard if we allow **higher-order functions** (functions that take functions as arguments and return functions as results). These are sometimes called "functionals".

Example

- Factorial:

  fac(N) = N < 2 ? 1 : N*fac(N-1)

- How to do this *without* recursion?

Functionalize the definition

- "Factorial" functional

  f(G)(N) = N < 2 ? 1 : N*(G(G)(N-1))

- Notice the above definition is ***not recursive***.
- G could be any function argument.

Functionalize the definition

- f(G)(N) = N < 2 ? 1 : N*(G(G)(N-1))
- G could be any function argument.
- f(f) makes sense:
- f(f)(N) = N < 2 ? 1 : N*(f(f)(N-1))
- So f(f) achieves the same effect as fac.
- We might say f(f) "is" fac?
- It is more correct to say "fac is a fixed point of f" (fac satisfies the functional equation when substituted for G).
- In fact (oops), fac is the ***least fixed point*** of functional f.

f(f) makes sense

- f(f)(N) = N < 2 ? 1 : N*(f(f)(N-1))

-
  ```
      f(f)(4)
  = 4*(f(f)(3))
  = 4*3*(f(f)(2))
  = 4*3*2*(f(f(1)))
  = 4*3*2*1
  ```

Least Fixed Point?

- Least in this case means "least defined".
- That is, it is the fixed point that makes the fewest assumptions consistent with the definition of f.
- In the case of f, fac is the *only* fixed point.
- In other cases, there can be more than one, with varying degrees of defined-ness.

Example Realization (in rex)

- 1 rex > f(G)(N) = N < 2 ? 1 : N*G(G)(N-1);
- 2 rex > f(f)(10);
- 3628800

An Application of the Recursion Theorem (Sipser)

- The **length** of the description of a machine <M> is the number of symbols in <M>.
- M is called **minimal** if there is no equivalent machine having a shorter description.
- **Theorem**:

  The language { <M> | M is a minimal TM }
  is not recognizable.

Proof

- Assume that L = { <M> | M is a minimal TM } is recognizable. Then L is **enumerated** by some Turing machine **E**.
- Construct the following TM, call it **C**, which, on input w:
  - Obtain the description <**C**> of this machine.
  - Using **E,** begin enumerating L until a machine **D** appears such that <**D**> is **longer** than <**C**>. (This must happen.)
  - Behave as **D** on w.
- **C** is equivalent to **D** by construction.
- But <**D**> is longer than <**C**>, therefore **D** cannot be minimal after all. It shouldn't be in the enumeration. This contradicts the assumption that **E** enumerates only minimal machines.