COMPSCI 501: Formal Language Theory

Lecture 19: The Recursion Theorem

Marius Minea
marius@cs.umass.edu

University of Massachusetts Amherst

6 March 2019

Recursion and Self-reference

- Can a Machine Self-Reproduce?
- How can we express recursion ?
  let f(...) = ... f(...) ...
  but Turing machines don't have *names* for self-reference
- Can a machine *access* its own description?
  and can it *use* that description?

A Self-Reproducing Program

Program contains:

- a tabular description of (part of) itself
- code to print that description (and the remainder)

```
char s[ ] = {
        '\t',
        '0',
        '\n',
        '}',
        ';',
        '\n',
        '\n',
        '/',
        '*',
        '\n',
        (213 lines deleted)
        0
};

/*
 * The string s is a
 * representation of the body
 * of this program from '0'
 * to the end.
 */

main( )
{
        int i;

        printf("char\ts[ ] = {\n");
        for(i=0; s[i]; i++)
                printf("\t%d, \n", s[i]);
        printf("%s", s);
}
```

(output of this program will be self-reproducing)

Ken Thompson, "Reflections on Trusting Trust"

Self-Reference

Lemma: For any string w, we can design a Turing machine P_w that prints w and then halts.

Reworded: There is a computable function from strings to strings, $q : \Sigma^* \to \Sigma^*$, so that $q(w)$ is the description of a Turing machine P_w that prints w and then halts.

TM Q computes $q(w)$:
On input string w:

1. Construct the following TM P_w:
   "On any input
   1. Erase input
   2. Write w on tape
   3. Halt."
2. Output $\langle P_w \rangle$

A Self-Printing Turing Machine: Intuition

Print two copies of the following, the second one in quotes:
"Print two copies of the following, the second one in quotes:"

The second line is a representation of the first.
It provides to the first part (the "program") a representation of itself, so the program can use and process it.

Constructing a Self-Printing Turing Machine

- We compose $SELF$ of two parts, A and B, so $SELF$ prints out $\langle SELF \rangle = \langle AB \rangle$
- A should print B: just take $A = P_{\langle B \rangle}$.
  so need description of B.
- Can't define B based on A in the same way (circular)
- But can have B compute A from the output of A!
- $\langle B \rangle$ was left on tape when A finished
- so B can obtain its own description $\langle B \rangle$
- B can compute $q(\langle B \rangle) = A$
- B combines A and B and writes $\langle AB \rangle = \langle SELF \rangle$ on tape

Self-Printing Turing Machine: Definition

$B =$ Given input $\langle M \rangle$ describing TM (fragment) M:

1. Compute $q(\langle M \rangle)$
2. Combine result with $\langle M \rangle$ into a TM
3. Print description of this TM and halt

$A = P_{\langle B \rangle}$

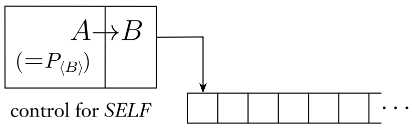


FIGURE 6.2
Schematic of *SELF*, a TM that prints its own description

Control starts at A, who produces B and passes control.

Recursion Theorem

SELF prints its own description.
But can a TM obtain its own description *and* use it to compute?

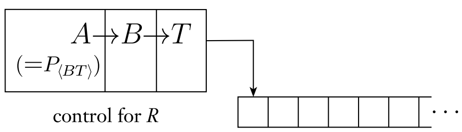
Let T be a TM that computes a function $t : \Sigma^* \times \Sigma^* \to \Sigma^*$. There is a Turing machine R that computes a function $r : \Sigma^* \to \Sigma^*$:

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

T is arbitrary TM that takes TM description and input
R operates like T with description of R filled in.

We can now say "obtain own description and use it" when needed.

Recursion Theorem



Construction in three parts:

- T is the given TM.
- A is $q(\langle BT \rangle)$
  - we want to preserve the input w
  - change q to *append* TM description to tape content
  - after running A, we have $w\langle BT \rangle$ on tape
- B: read tape, apply q, get $\langle A \rangle$
  - combine A, B, T into single machine
  - encode this with w on tape: $\langle R, w \rangle$, pass control to T

Revisiting: A_{TM} is undecidable

Assume decider $H(\langle M, w \rangle)$.

Previous proof: Construct $Test(\langle M \rangle) = \neg H(\langle M, \langle M \rangle \rangle)$

Run $Test(\langle Test \rangle) = \neg H(\langle Test, \langle Test \rangle \rangle) = \neg Test(\langle Test \rangle)$, contradiction

New proof: Construct the following TM B:
On input w:

1. Obtain own description $\langle B \rangle$ via recursion theorem
2. Run H on input $\langle B, w \rangle$
3. Do the opposite of H (accept/reject)

B does the opposite of what H says it does: contradiction

Minimal Turing Machines

Def. Say M is minimal if there is no equivalent TM with a shorter description than $\langle M \rangle$ (number of symbols).

Let $MIN_{\mathsf{TM}} = \{\langle M \rangle | M \text{ is a minimal } TM\}$.

Theorem: MIN_{TM} is not Turing-recognizable.

Proof: by contradiction. Assuming an enumerator E, construct C:

C = "On input w:

1. Obtain own description $\langle C \rangle$ via recursion theorem
2. Run enumerator E until getting a machine D longer than C
3. Simulate D on w."

D exists, since MIN_{TM} is infinite.

Since C simulates D, they are equivalent.

But C is shorter than D, so D should not be in E's list.

Fixed Point Version of Recursion Theorem

Def. A fixed point (fixpoint) of a function f is a value x that is unchanged by applying the function: $f(x) = x$

For any transformation on Turing machines, there is a TM unchanged by that transformation. Formally:

Theorem: Let $t : \Sigma^* \to \Sigma*$ be a computable function.
Then there is a TM F for which $t(\langle F \rangle)$ is equivalent to F.

F = "On input w:

1. Obtain own description $\langle F \rangle$ via recursion theorem
2. Compute $t(\langle F \rangle)$ to obtain a TM G
3. Simulate G on w."

Since F simulates G, they are equivalent!