CS420, Ch 6.1: Turing Machines and Recursion

Mon, April 12, 2021
Announcements

• HW8 past due

• HW9 due Sun 4/18 11:59pm EST

• **Last Unit:** Time Complexity
  • P, NP, NP-Completeness ...
  • Starting Wed 4/14

• **Reminder:** no class next Monday 4/19
  • Patriot’s Day
Recursion in Programming

```
(define (factorial n)
  (if (zero? n)
      1
      (* n (factorial (sub1 n)))))
```

In most programming languages, you can call a function recursively, even before it’s completely defined!
Turing Machines and Recursion

• We’ve been saying: “A Turing machine models programs.”

• **Q:** Is a recursive program modeled by a Turing machine?

• **A:** Yes!
  • But it’s not explicit.
  • In fact, it’s a little complicated.
  • Need to prove it ...

• **Today:** The Recursion Theorem

Where’s the recursion in this definition???
The Recursion Theorem

You can write a TM description like this: Prove $A_{TM}$ is undecidable, by contradiction:
assume that Turing machine $H$ decides $A_{TM}$

$B = \text{"On input w:
  1. Obtain, via the recursion theorem, own description } \langle B \rangle.\
  2. Run } H \text{ on input } \langle B, w \rangle.\
  3. Do the opposite of what } H \text{ says. That is, accept if } H \text{ rejects and reject if } H \text{ accepts."
}$

This is the non-existent “$D$” machine, the TM that does the opposite of itself, defined using recursion!
How can a TM “obtain it’s own description?”

How does a TM even know about “itself” before it’s completely defined?
A (Simpler) Coding Exercise

Your task:
• Write a program that, without using recursion, prints itself.
  • Such a program obviously must have knowledge about “itself”

• An example, in English:

Print out two copies of the following, the second one in quotes:
“Print out two copies of the following, the second one in quotes:”

• A program that does this knows about “itself”,
  • but it does not explicitly use recursion!
Interlude: Lambda

• $\lambda =$ anonymous function value, e.g. ($\lambda (x) \; x$)
  
  • **C++**: [](int \ x){ return \ x; }
  
  • **Java**: $(x) \to \{} \; \text{return} \; x; \}$
  
  • **Python**: `lambda x : x`
  
  • **JS**: $(x) \to \{} \; \text{return} \; x; \}$
My Self-Reproducing Program

Print out two copies of the following, the second one in quotes:
“Print out two copies of the following, the second one in quotes:”

```
((\ (the-following) (print2x-2ndquoted the-following))
 "(\ (the-following) (print2x-2ndquoted the-following)))"
```

```
(define (print2x-2ndquoted str)
 (printf "(~a\n ~v)\n" str str))
```

First copy  Second copy (quoted)

(could have inlined this)
Self-Reproducing Turing Machine

The following TM $Q$ computes $q(w)$.

$Q = \text{“On input string } w$:}
\begin{align*}
1. &\text{ Construct the following Turing machine } P_w. \\
&\;P_w = \text{“On any input:} \\
&\;\quad 1. \text{ Erase input.} \\
&\;\quad 2. \text{ Write } w \text{ on the tape.} \\
&\;\quad 3. \text{ Halt.”} \\
2. &\text{ Output } \langle P_w \rangle.\]

$P_{<B>}$ is TM that writes $<B>$ on tape (TM's pass args by writing it on tape)

$q$ creates $P_{<B>}$

Print out two copies of the following, the second on in quotes:

“Print out two copies of the following, the second on in quotes:”
Recursive Program that Prints Itself

\[ SELF = \text{"On any input: \ 1. Obtain, via the recursion theorem, own description } \langle SELF \rangle. \ 2. \text{ Print } \langle SELF \rangle. \"} \]

• Our program contains “itself” even though it has no recursion!

• We don’t need explicit recursion to write recursive programs!

• Can we write a program that does something other than print “itself”?
Other nonrecursive programs using “itself”

- Program that prints “itself”:

  $$\left(\lambda \text{(the-following)} \left(\text{print2x-2ndquoted the-following})\right)\right)\left(\text{"(\lambda \text{(the-following)} \left(\text{print2x-2ndquoted the-following})\right)"}\right)$$

- Program that runs “itself” repeatedly (i.e., it loops):

  $$\left(\lambda \text{(x)} \left(\text{x x}\right)\right)\left(\lambda \text{(x)} \left(\text{x x}\right)\right)$$

  - Call arg fn with itself as arg
  - Don’t convert arg to string
  - Need this extra lambda bc we want to call $f$ first before looping

- Loop, but call some other function $f$ each time:

  $$\left(\lambda \text{(f)} \left(\left(\lambda \text{(x)} \left(\text{f (x x)}\right)\right)\right)\right)$$

  $$\left(\left(\lambda \text{(x)} \left(\text{f (\lambda \text{(v)} \left(\text{(x x) v})\right)\right)}\right)\right)\left(\lambda \text{(x)} \left(\text{f (\lambda \text{(v)} \left(\text{(x x) v})\right)}\right)\right)\text{ ))}

- None of these programs use explicit recursion!
The Recursion Theorem, Formally

**Recursion theorem**  Let $T$ be a Turing machine that computes a function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. There is a Turing machine $R$ that computes a function $r : \Sigma^* \rightarrow \Sigma^*$, where for every $w$,

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

**In English:**

- If you want TM $R$ that includes a step “obtain own description” ...

- ... instead create TM $T$ with an extra “itself” argument ...

- ... then construct $R$ from $T$
Recursion Theorem, A Concrete Example

• If you want:
  • Recursive fn

```
(define (factorial n) ;; R
  (if (zero? n)
      1
      (* n (factorial (sub1 n)))))
```

• Instead create:
  • Non-recursive fn

```
(define (factorial/itself ITSELF n) ;; T
  (if (zero? n)
      1
      (* n (ITSELF (sub1 n))))
```

Recursion Theorem says you can convert

But how??
Recursion Theorem, Proof

• To convert a “T” to “R”:

\[
A \rightarrow B \rightarrow T \quad (= P_{BT})
\]

control for R

\[\cdots\]

1. Construct \( A = \) program constructing \(<BT>\), and
2. Pass result to \( B \) (from before),
3. which passes “itself” to \( T \)

• Compare with \textit{SELF}:

Print out two copies of the following, the second on in quotes:
“Print out two copies of the following, the second on in quotes;”
Recursion Theorem Proof: Coding Demo

- Program that passes “itself” to another function:

\[
(\lambda f \, ((\lambda x \, (f (\lambda v \, ((x \, x) \, v)))))) \, (\lambda x \, (f (\lambda v \, ((x \, x) \, v)))))
\]

- Function that needs “itself”

\[
(define \text{factorial/itself} \, \text{ITSELF} \, n) \quad ;; \; T
(if \, (\text{zero?} \, n)
\, 1
\, (* \, n \, (\text{ITSELF} \, (\text{sub1} \, n))))
\]
Fixed Points

• A value $x$ is a fixed point of a function $f$ if $f(x) = x$
Recursion Theorem and Fixed Points

**Theorem 6.8**

Let $t : \Sigma^* \rightarrow \Sigma^*$ be a computable function. Then there is a Turing machine $F$ for which $t(\langle F \rangle)$ describes a Turing machine equivalent to $F$. Here we'll assume that if a string isn't a proper Turing machine encoding, it describes a Turing machine that always rejects immediately.

In this theorem, $t$ plays the role of the transformation, and $F$ is the fixed point.

**Proof**

Let $F$ be the following Turing machine.

$F =$ “On input $w$:
1. Obtain, via the recursion theorem, own description $\langle F \rangle$.
2. Compute $t(\langle F \rangle)$ to obtain the description of a TM $G$.
3. Simulate $G$ on $w$.”

Clearly, $\langle F \rangle$ and $t(\langle F \rangle) = \langle G \rangle$ describe equivalent Turing machines because $F$ simulates $G$.

- I.e., Recursion theorem says:
  - “every TM that computes on TMs has a fixed point”
  - **As code:** “every function on functions has a fixed point”
Y Combinator

- \texttt{mk-recursive-fn} = a “fixed point finder”

\begin{verbatim}
(define mk-recursive-fn
  (\lambda (f)
    ((\lambda (x) (f (\lambda (v) ((x x) v))))
     (\lambda (x) (f (\lambda (v) ((x x) v)))))))
\end{verbatim}
Summary: Where “Recursion” Comes From

• TMs are powerful enough to:
  1. Construct other TMs
  2. Simulate other TMs

• That’s enough to achieve recursion!
Check-in Quiz 4/12
On gradescope