UMB CS622

Turing Machines and Recursion

Monday, November 1, 2021
Announcements

• Hw6 extended deadline:
  • due Wed 11/3 11:59pm
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:

\[ B = \text{"On input } w:\text{ 1. Obtain, via the recursion theorem, own description } \langle B \rangle.\text{"} \]
The Recursion Theorem

Prove $A_{TM}$ is undecidable, by contradiction:

assume that Turing machine $H$ decides $A_{TM}$

$B = "On input $w$:"

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

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

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

Our Task:
• Create a TM that, without using recursion, prints itself.
  • How does this TM get 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:”

• This TM knows about “itself”,
  • but it does not explicitly use recursion!

Idea:
TMs can receive TMs as input; just assume input will be yourself!

“TM input”

“TM”

“argument”
  (the TM gets itself from its input!)
Self-Printing Turing Machine

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

$Q = \text{"On input string } w:\text{"}$

1. Construct the following Turing machine $P_w$. $P_w = \text{"On any input:"
   
   1. Erase input.\text{[1]}   
   2. Write } w \text{ on the tape.}
   3. Halt."

2. Output $\langle P_w \rangle$.\text{[2]}

$B = \text{"On input } \langle M \rangle, \text{ where } M \text{ is a portion of a TM:"

Compute $q(\langle M \rangle)$.}

Combine the result with $\langle M \rangle$ to make a complete TM.

3. Print the description of this TM and halt.”

$q$ creates a TM (that prints a string)\text{[1]}, and outputs it as a string (i.e., it’s “quoted”)\text{[2]}

So $q(\langle M \rangle)$ prints a “quoted” $M$

Print out two copies of the following, the second on in quotes:
SELF, Defined With The Recursion Theorem

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

- So a TM doesn’t need explicit recursion to call itself!

- What about TMs that do more than “print itself”?
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 a TM $R$ that can "obtain own description" ...

- ... instead create a TM $T$ with an extra "itself" argument ...

- ... then construct $R$ from $T$ ???
The Recursion Theorem, Pictorially

• To convert a “T” to “R”:

\[ A \rightarrow B \rightarrow T \]

(\(=P_{<BT>}\))

control for \(R\)

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

- If you want:
  - Recursive fn

- Instead create:
  - Non-recursive fn

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

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

Recursion Theorem says you can convert

It’s not clear how the recursion theorem applies to real programs?
TM and Recursive Programs

• So a TM doesn’t need explicit recursion to call itself!

• What about programs? (TM = Programs)

• Can we write recursive programs without using explicit recursion?
Interlude: Lambda

• \( \lambda = \) anonymous function, e.g. \((\lambda \ x \ x)\)
  - C++: 
    ```cpp
    [](int x){ return x; }
    ```
  - Java: 
    ```java
    (x) -> { return x; }
    ```
  - Python: 
    ```python
    lambda x : x
    ```
  - JS: 
    ```javascript
    (x) => { return x; }
    ```
A Self-Printing 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:”

```
((\ (SELF) (print2x SELF))
 "((\ (SELF) (print2x SELF))")
```

Could we write a program that does something other than print “itself”?

```
(define (print2x str)
  (printf "\n\n" str str))
```

First copy
Second copy (quoted)
Non-Printing Uses of \textit{SELF}

- Program that prints “itself”:
  \[
  \begin{aligned}
  &(((\lambda \ (\text{SELF}) \ (\text{print2x SELF})) \ \\
  &"(\lambda \ (\text{SELF}) \ (\text{print2x SELF}))")
  \end{aligned}
  \]

- Program that runs “itself” repeatedly (i.e., it infinite loops):
  \[
  (((\lambda \ (\text{SELF}) \ (\text{SELF SELF})) \ \\
  ((\lambda \ (\text{SELF}) \ (\text{SELF SELF}))))
  \]

- Loop, but do something useful each time?
  \[
  (((\lambda \ (\text{SELF}) \ (f \ (\text{SELF SELF})))) \ \\
  ((\lambda \ (\text{SELF}) \ (f \ (\lambda \ (v) \ ((\text{SELF SELF} \ v)))))))
  \]

- None of these programs use explicit recursion!

\textit{Y combinator}
Recursion Theorem Proof: Coding Demo

• Program that passes “itself” to another function:

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

Y combinator

• Function that needs “itself”

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

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

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 implies:
  - “every TM that computes on TMs has a fixed point”
  - **As code:** “every function on functions has a fixed point”

Fixed point is a TM that is unchanged by the function
Y Combinator

• `mk-recursivelfn` = a “fixed point finder”

```
(define mk-recursive-fn
  (λ (f)
    ((λ (x) (f (λ (v) ((x x) v))))
     (λ (x) (f (λ (v) ((x x) v)))))))
```

• `factorial` is the fixed point of `mk-factorial`
Summary: Where “Recursion” Comes From

• TMs are powerful enough to:
  1. Receive other TMs as input
  2. Construct other TMs
  3. Simulate other TMs

• That’s enough to achieve recursion!
Check-in Quiz 11/1

On gradescope