

Analyzing Procedures

Summary: In the laboratory, you will explore the running time for a few algorithm variants.

Contents:

- Preparation
- Exercises
 - Exercise 1: Manual Analysis
 - Exercise 2: Automatic Analysis
 - Exercise 3: Additional Calls
 - Exercise 4: Predicting Calls
 - Exercise 5: The Brightest Color, Revisited
- For Those with Extra Time

Preparation

- a. In DrScheme, create a new file for this lab, called `analysis-examples.scm`.
- b. Add the comments and code for `reverse-1`, `reverse-2`, and `my-append` from the corresponding reading to your file.

Exercises

Exercise 1: Manual Analysis

- a. Add the following line to the beginning of `my-append` (again, immediately after the `lambda`).

```
(display (list 'my-append front back)) (newline)
```

- b. Determine how many times `my-append` is called when reversing a list of length seven using `reverse-1`.

- c. Add the following line to the kernel of `reverse-2` (immediately after the `lambda`).

```
(display (list 'reverse-2-kernel remaining reversed)) (newline)
```

- d. Determine how many times `reverse-2-kernel` is called when reversing a list of length seven using `reverse-2`.

- e. Comment out the lines that you just added by prefixing them with a semicolon.

Exercise 2: Automatic Analysis

a. Replace the `define` for `reverse-1` with `define$`, as in the following.

```
(define$ reverse-1
  (lambda (lst)
    ...))
```

b. Find out how many times `my-append` is called in reversing a list of seven elements by entering the following command in the interactions pane.

```
> (analyze (reverse-1 (list 1 2 3 4 5 6 7)) my-append)
```

c. Did you get the same answer as in the previous exercise? If not, why do you think you got a different result?

d. One potential issue is that we haven't told the analyst to include the recursive calls in `my-append`. We can do so by replacing `define` with `define$` in the definition of `my-append`.

e. Once again, find out how many times `my-append` is called in reversing a list of seven elements by entering the following command in the interactions pane.

```
> (analyze (reverse-1 (list 1 2 3 4 5 6 7)) my-append)
```

f. Did you get the same answer as in exercise 1? If not, what difference do you see?

g. Replace the `define` in `reverse-2` with `define$`.

h. Find out how many times `reverse-2-kernel` is called in reversing a list of seven elements by entering the following command in the interactions pane.

```
> (analyze (reverse-2 (list 1 2 3 4 5 6 7)) reverse-2-kernel)
```

i. Did you get the same answer as in exercise 1? If not, what difference do you see?

Exercise 3: Additional Calls

In the previous exercise, you considered only a single procedure in each case (`my-append` for `reverse-1`, `reverse-2-kernel` for `reverse-2`). Suppose we incorporate all of the other procedures. What effect does it have?

a. Find out how many total procedure calls are done in reversing a list of length seven, using `reverse-1`, with the following.

```
> (analyze (reverse-1 (list 1 2 3 4 5 6 7)))
```

b. How does that number of calls seem to relate to the number of calls to `my-append`?

c. Are there any procedures you're surprised to see?

d. Find out how many total procedure calls are done in reversing a list of length seven, using `reverse-2`, with the following.

```
> (analyze (reverse-2 (list 1 2 3 4 5 6 7)))
```

e. How does that number of calls seem to relate to the number of calls to `kernel`?

f. Are there any procedures you're surprised to see?

Exercise 4: Predicting Calls

a. Fill in the following chart to the best of your ability.

List Length	r1: Calls to <code>my-append</code>	r1: Total calls	r2: Calls to <code>kernel</code>	r2: Total calls
2				
4				
8				
16				

b. Predict what the entries will be for a list size of 32.

c. Check your results experimentally.

d. Write a formula for the columns, to the best of your ability.

Exercise 5: The Brightest Color, Revisited

Here is a third version of `rgb.brightest`.

```
(define rgb.brightest
  (lambda (colors)
    (rgb.brightest-helper (car colors) (cdr colors))))
(define rgb.brightest-helper
  (lambda (brightest-so-far remaining-colors)
    (if (null? remaining-colors)
        brightest-so-far
        (rgb.brightest-helper
         (rgb.brighter brightest-so-far (car remaining-colors))
         (cdr remaining-colors)))))
```

a. Find out how many steps this procedure takes on lists of length 2, 4, 8, and 16 in which the elements are arranged from lightest to darkest.

- b. Find out how many steps this procedure takes on lists of length 2, 4, 8, and 16 in which the elements are arranged from darkest to lightest.
- c. Find out how many steps this procedure takes on lists of length 2, 4, 8, and 16 in which the elements are in no particular order.
- d. Predict the number of steps this procedure will take on each kind of list, where the length is 32.

For Those with Extra Time

Copyright © 2007 Janet Davis, Matthew Kluber, and Samuel A. Rebelsky. (Selected materials copyright by John David Stone and Henry Walker and used by permission.) This material is based upon work partially supported by the National Science Foundation under Grant No. CCLI-0633090. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.