Verifying Preconditions

Summary: In the laboratory, you will consider mechanisms for verifying the preconditions of procedures. You will also consider some issues in the documentation of such procedures.

Contents:

- Preparation
- Exercises
  - Exercise 1: Are They All Spots?
  - Exercise 2: Differentiating Between Errors
  - Exercise 3: Finding Values
  - Exercise 4: Changing Colors
  - Exercise 5: Weighted Color Averages
- For Those With Extra Time
  - Extra 1: Substitution in Lists
  - Extra 2: Substituting Colors, Revisited
  - Extra 3: Substituting Colors, Revisited
- Notes on the Problems
  - Notes on Problem 1: Are They All Spots?
  - Notes on Exercise 3: Finding Values

Preparation

Add the following definitions to your definitions window. The first, four, spot.new, spot.col, spot.row, and spot.color, you should have defined when we were first building the spot type. The next, spot?, is a procedure you may have written on your own. Finally, spot-list.leftmost, comes from the reading.

```
(define spot.new
  (lambda (col row color)
    (list col row color)))
(define spot.col car)
(define spot.row cadr)
(define spot.color caddr)

(define spot?
  (lambda (value)
    (and (list? value)
         (= (length value) 3)
         (integer? (spot.col value))
         (integer? (spot.row value))
         (rgb? (spot.color value))))

(define spot-list.leftmost
  (lambda (spots)
```


(if (or (not (list? spots))
  (null? spots)
  (not (all-spots? spots)))
  (throw "spot-list.leftmost: requires a non-empty list of spots")
  (if (null? (cdr spots))
      (car spots)
      (spot.leftmost (car spots) (spot-list.leftmost (cdr spots))))))

Exercises

Exercise 1: Are They All Spots?

You may note that spot-list.leftmost requires an all-spots? procedure. You should have
written that procedure for a recent homework assignment. If you don’t have it at hand, here’s a definition,

```
(define all-spots?
  (lambda (lst)
    (or (null? lst)
        (and (spot? (car lst))
             (all-spots? (cdr lst))))))
```

a. What preconditions should all-spots? have?

b. Is it necessary to test those preconditions? Why or why not?


Exercise 2: Differentiating Between Errors

Revise the definition of spot-list.leftmost so that it prints a different (and appropriate) error
message for each error condition.

I’d recommend that you use cond rather than if in writing this revised version.

Exercise 3: Finding Values

a. Document (using the six-P style), define, and test a procedure named index-of that takes a value,
val, and a list, vals as its arguments and returns the index of val in vals. You should use 0-based
indices, so that the initial value in a list is at index 0.

```
> (index-of color.red (list color.red color.green color.blue color.yellow))
0
> (index-of color.blue (list color.red color.green color.blue color.yellow))
2
```

b. Arrange for index-of to explicitly signal an error (by invoking the throw procedure) if val does
not occur at all as an element of vals.
c. Some programmers return special values to signal an error to the caller, rather than throw an error. If val does not occur as an element of vals, why might it be better to have index-of to return a special value (such as -1 or #f) rather than throwing an error? Explain your answer.

d. If val does not occur as an element of vals, why might be better to have index-of throw an error?

When you’re done thinking about these questions, add index-of to your library. This is a very useful procedure.

**Exercise 4: Changing Colors**

Consider the following procedure, that increments the red component of color by 64.

```
;;; Procedure:
;;;   rgb.much-redder
;;; Parameters:
;;;   color, an RGB color
;;; Purpose:
;;;   To produce a color that is much redder than color.
;;; Produces:
;;;   newcolor, a color
;;; Preconditions:
;;;   FORTHCOMING
;;; Postconditions:
;;;   (rgb.red new-color) = (+ 64 (rgb.red color))
(define rgb.much-redder
  (lambda (color)
    (rgb.new (+ 64 (rgb.red color)) (rgb.green color) (rgb.blue color))))
```

a. What preconditions must be met in order for rgb-much-redder to meet its postconditions?

b. Should we test those preconditions? Why or why not?

**Exercise 5: Weighted Color Averages**

In a number of exercises, we were required to blend two colors. For example, we blended colors in a variety of ways to make interesting images, and we made a color more grey by averaging it with grey. In blending two colors, we are, in essence, creating an average of the two colors, but an average in which each color contributes a different fraction.

For this problem, we might write a procedure, (rgb.weighted-average fraction color1 color2) that makes a new color, each of whose components is computed by multiplying the corresponding component of color1 by fraction and adding that to the result of multiplying the corresponding component of color2 by (1-fraction). For example, we might compute the red component with
a. What preconditions should rgb.weighted-average have? (Think about restrictions on percent, color1, and color2.)

b. How might you formally specify the postconditions for rgb.weighted-average?


d. Write the code for rgb.weighted-average, making sure to test for each precondition.

For Those With Extra Time

Extra 1: Substitution in Lists

Consider a procedure, (list.substitute lst old new), that builds a new list by substituting new for old whenever old appears in lst.

> (list.substitute (list "black" "red" "green" "blue" "black") "black" "white")
(list "white" "red" "green" "blue" "white")
> (list.substitute (list "black" "red" "green" "blue" "black") "yellow" "white")
(list "black" "red" "green" "blue" "black")
> (list.substitute null "yellow" "white")
()

a. Document this procedure, making sure to carefully consider the preconditions.

b. Implement this procedure, making sure to check the preconditions.

Extra 2: Substituting Colors, Revisited

Consider a procedure, (spots.substitute spots old new), that, given a list of spots and two colors, makes a new copy of spots by using the color new whenever old appeared in the original spots.

a. What preconditions does this procedure have?

b. Implementing this procedure, using the husk-and-kernel structure to ensure that old and new are rgb colors and that spots is a list of colors, before starting the recursion.

Extra 3: Substituting Colors, Revisited

Write a procedure, (image.substitute image old new), that, given an image, makes a new copy of image by using the color new whenever old appeared in the original image. Ensure that this procedure verifies its preconditions.

> (image.substitute "~/home/rebelsky/glimmer/samples/rebelsky-stalkernet.jpg" rgb.red rgb.blue)
image.substitute: Expected an image as the first parameter
> (image.substitute (image.load "~/home/rebelsky/glimmer/samples/rebelsky-stalkernet.jpg") "puce" "red")
image.substitute: Expected an rgb color as the second parameter
Notes on the Problems

Notes on Problem 1: Are They All Spots?

Here are some possible solutions.

Using that strategy, I just fill in spot? or spots? for ____?.

a. The all-spots? procedure needs a list as a parameter.

b. It depends on how we will use all-spots?. If we are sure that it will only be called correctly (e.g., after we’ve already tested that the parameter is a list or in a context in which we can prove that the parameter is list), then it need not check its preconditions. Otherwise, it should check its preconditions.

Notes on Exercise 3: Finding Values

a. Here’s an initial version of index-of

```scheme
(define index-of
  (lambda (val vals)
    ;; If the value appears first in the list
    (if (equal? (car val) vals)
      0
      ;; Otherwise, we find the index in the cdr. Since we’ve
      ;; thrown away the car in finding that index, we need to add 1.
      (+ 1 (index-of val (cdr vals))))))
```

We might also write this procedure tail-recursively.

```scheme
(define index-of
  (lambda (val vals)
    (index-of-helper val 0 vals)))

(define index-of-helper
  (lambda (val skipped remaining)
    (if (equal? val (car remaining))
      skipped
      (index-of-helper val (+ skipped 1) (cdr remaining))))))
```

You should be able to figure out what preconditions to test and how to test them.

c. If index-of explicitly checks its precondition using member?, we end up duplicating work. That is, we scan the list once to see if the value is there, and once to see its index. Even if index-of does not explicitly check its precondition, the caller may be called upon to do so, which still duplicates the work. By having index-of return a special value, we permit the client to have index-of do both.

d. In some cases, programs should stop when there is no index for a specified value. For example, a program that tries to look up a grade for a student should not continue if the student does not appear in the list. There are also some instances in which careless programmers do not check the return value, which can lead to unpredictable behavior.