SICP Exercise Solutions for Section 2.1

Paul L. Snyder

March 25, 2014

Contents

1 2.1.1 Example: Arithmetic Operations for Rational Numbers

1.1 Exercise 2.1:

1.1.1 Problem

Define a better version of `make-rat` that handles both positive and negative arguments. `make-rat` should normalize the sign so that if the rational number is positive, both the numerator and denominator are positive, and if the rational number is negative, only the numerator is negative.

1.1.2 Solution

2 2.1.2 Abstraction Barriers

2.1 Exercise 2.2:

2.1.1 Problem

Consider the problem of representing line segments in a plane. Each segment is represented as a pair of points: a starting point and an ending point. Define a constructor `make-segment` and selectors `start-segment` and `end-segment` that define the representation of segments in terms of points. Furthermore, a point can be represented as a pair of numbers: the x coordinate and the y coordinate. Accordingly, specify a constructor `make-point` and selectors `x-point` and `y-point` that define this representation. Finally, using your selectors and constructors, define a procedure `midpoint-segment` that takes a line segment as argument and returns its midpoint (the point whose coordinates are the average of the coordinates of the endpoints). To try your procedures, you'll need a way to print points:

```
(define (print-point p)
  (newline)
  (display "(")
  (display (x-point p))
  (display ",")
  (display (y-point p))
  (display ")"))
```

2.1.2 Solution

2.2 Exercise 2.3:

2.2.1 Problem

Implement a representation for rectangles in a plane. (Hint: You may want to make use of Exercise 2-2.) In terms of your constructors and selectors, create procedures that compute the perimeter and the area of a given rectangle. Now implement a different representation for rectangles. Can you design your system with suitable abstraction barriers, so that the same perimeter and area procedures will work using either representation?

2.2.2 Solution

3 2.1.3 What Is Meant by Data?

3.1 Exercise 2.4:

3.1.1 Problem

Here is an alternative procedural representation of pairs. For this representation, verify that `(car (cons x y))` yields `x` for any objects `x` and `y`.

```
(define (cons x y)
  (lambda (m) (m x y)))

(define (car z)
  (z (lambda (p q) p)))
```

What is the corresponding definition of `cdr`? (Hint: To verify that this works, make use of the substitution model of section 1.1.5.)

3.1.2 Solution

3.2 Exercise 2.5:

3.2.1 Problem

Show that we can represent pairs of non-negative integers using only numbers and arithmetic operations if we represent the pair a and b as the integer that is the product $2^a 3^b$. Give the corresponding definitions of the procedures `cons`, `car`, and `cdr`.

3.2.2 Solution

3.3 Exercise 2.6:

3.3.1 Problem

In case representing pairs as procedures wasn't mind-boggling enough, consider that, in a language that can manipulate procedures, we can get by without numbers (at least insofar as non-negative integers are concerned) by implementing 0 and the operation of adding 1 as

```
(define zero (lambda (f) (lambda (x) x)))

(define (add-1 n)
  (lambda (f) (lambda (x) (f ((n f) x)))))
```

This representation is known as "Church numerals", after its inventor, Alonzo Church, the logician who invented the λ calculus.

Define `one` and `two` directly (not in terms of `zero` and `add-1`). (Hint: Use substitution to evaluate `(add-1 zero)`). Give a direct definition of the addition procedure '+' (not in terms of repeated application of 'add-1').

3.3.2 Solution

4 2.1.4 Extended Exercise: Interval Arithmetic

4.1 Exercise 2.7:

4.1.1 Problem

Alyssa's program is incomplete because she has not specified the implementation of the interval abstraction. Here is a definition of the interval constructor:

```
(define (make-interval a b) (cons a b))
```

Define selectors `upper-bound` and `lower-bound` to complete the implementation.

4.1.2 Solution

4.2 Exercise 2.8:

4.2.1 Problem

Using reasoning analogous to Alyssa's, describe how the difference of two intervals may be computed. Define a corresponding subtraction procedure, called `sub-interval`.

4.2.2 Solution

4.3 Exercise 2.9:

4.3.1 Problem

The "width" of an interval is half of the difference between its upper and lower bounds. The width is a measure of the uncertainty of the number specified by the interval. For some arithmetic operations the width of the result of combining two intervals is a function only of the widths of the argument intervals, whereas for others the width of the combination is not a function of the widths of the argument intervals. Show that the width of the sum (or difference) of two intervals is a function only of the widths of the intervals being added (or subtracted). Give examples to show that this is not true for multiplication or division.

4.3.2 Solution

4.4 Exercise 2.10:

4.4.1 Problem

Ben Bitdiddle, an expert systems programmer, looks over Alyssa's shoulder and comments that it is not clear what it means to divide by an interval that spans zero. Modify Alyssa's code to check for this condition and to signal an error if it occurs.

4.4.2 Solution

4.5 Exercise 2.11:

4.5.1 Problem

In passing, Ben also cryptically comments: "By testing the signs of the endpoints of the intervals, it is possible to break `mul-interval` into nine

cases, only one of which requires more than two multiplications." Rewrite this procedure using Ben's suggestion.

After debugging her program, Alyssa shows it to a potential user, who complains that her program solves the wrong problem. He wants a program that can deal with numbers represented as a center value and an additive tolerance; for example, he wants to work with intervals such as 3.5 ± 0.15 rather than $[3.35, 3.65]$. Alyssa returns to her desk and fixes this problem by supplying an alternate constructor and alternate selectors:

```
(define (make-center-width c w)
  (make-interval (- c w) (+ c w)))

(define (center i)
  (/ (+ (lower-bound i) (upper-bound i)) 2))

(define (width i)
  (/ (- (upper-bound i) (lower-bound i)) 2))
```

Unfortunately, most of Alyssa's users are engineers. Real engineering situations usually involve measurements with only a small uncertainty, measured as the ratio of the width of the interval to the midpoint of the interval. Engineers usually specify percentage tolerances on the parameters of devices, as in the resistor specifications given earlier.

4.5.2 Solution

4.6 Exercise 2.12:

4.6.1 Problem

Define a constructor `make-center-percent` that takes a center and a percentage tolerance and produces the desired interval. You must also define a selector `percent` that produces the percentage tolerance for a given interval. The `center` selector is the same as the one shown above.

4.6.2 Solution

4.7 Exercise 2.13:

4.7.1 Problem

Show that under the assumption of small percentage tolerances there is a simple formula for the approximate percentage tolerance of the product of

two intervals in terms of the tolerances of the factors. You may simplify the problem by assuming that all numbers are positive.

4.7.2 Solution

4.8 Exercise 2.14:

4.8.1 Problem

After considerable work, Alyssa P. Hacker delivers her finished system. Several years later, after she has forgotten all about it, she gets a frenzied call from an irate user, Lem E. Tweakit. It seems that Lem has noticed that the formula for parallel resistors can be written in two algebraically equivalent ways:

$$\frac{r_1 r_2}{r_1 + r_2}$$

and

$$\frac{1}{1/r_1 + 1/r_2}$$

He has written the following two programs, each of which computes the parallel-resistors formula differently:

```
(define (par1 r1 r2)
  (div-interval (mul-interval r1 r2)
                (add-interval r1 r2)))

(define (par2 r1 r2)
  (let ((one (make-interval 1 1)))
    (div-interval one
                  (add-interval (div-interval one r1)
                                (div-interval one r2)))))
```

Lem complains that Alyssa's program gives different answers for the two ways of computing. This is a serious complaint.

Demonstrate that Lem is right. Investigate the behavior of the system on a variety of arithmetic expressions. Make some intervals A and B, and use them in computing the expressions A/A and A/B. You will get the most insight by using intervals whose width is a small percentage of the center value. Examine the results of the computation in center-percent form (see Exercise 2.12).

4.8.2 Solution

4.9 Exercise 2.15:

4.9.1 Problem

Eva Lu Ator, another user, has also noticed the different intervals computed by different but algebraically equivalent expressions. She says that a formula to compute with intervals using Alyssa's system will produce tighter error bounds if it can be written in such a form that no variable that represents an uncertain number is repeated. Thus, she says, `par2` is a "better" program for parallel resistances than `par1`. Is she right? Why?

4.9.2 Solution

4.10 Exercise 2.16:

4.10.1 Problem

Explain, in general, why equivalent algebraic expressions may lead to different answers. Can you devise an interval-arithmetic package that does not have this shortcoming, or is this task impossible? (Warning: This problem is very difficult.)

4.10.2 Solution