Structure and Interpretation of Computer Programs

Exercise solutions

Konstantinos Chousos

June 7, 2023

Introduction

This document contains my solutions to the exercises of the *Abelson, Sussman & Sussman (2002) Structure and Interpretation of Computer Programs, MIT Press* book. It will be updated to reflect my current progress.

Instead of reading from the PDF, I study SICP through its TeXInfo format[1]. This way, I read the material, interact with the REPL[2] and take notes all from my editor of choice, Emacs.

This document is written in Org-Mode. The code blocks are evaluated using Org-Babel, which is also used to *tangle* the resulting sicp.rkt file and *weave* this very PDF, as in literate programming parlance.

Racket is used for implementing the exercises, using the `sicp` package[3]. Other possible options are guile and MIT-scheme.

[1] https://www.neilvandyke.org/sicp-texi/
[2] see "Read-Eval-Print-Loop"
[3] https://docs.racket-lang.org/sicp-manual/index.html

Chapter 1

1.1

Below is a sequence of expressions. What is the result printed by the interpreter in response to each expression? Assume that the sequence is to be evaluated in the order in which it is presented.

The interpreter's expected results are in the comments under each expression.

```
10
;; 10
(+ 5 3 4)
;; 12
(- 9 1)
;; 8
(/ 6 2)
;; 3
(+ (* 2 4) (- 4 6))
;; 6
(define a 3)
;;
(define b (+ a 1))
;;
(+ a b (* a b))
;; 19
(= a b)
;; #f
(if (and (> b a) (< b (* a b)))
    b
    a)
;; 4
(cond ((= a 4) 6)
      ((= b 4) (+ 6 7 a))
      (else 25))
;; 16
(+ 2 (if (> b a) b a))
;; 6
(* (cond ((> a b) a)
         ((< a b) b)
         (else -1))
   (+ a 1))
;; 16
```

1.2

Translate the following expression into prefix form.

$$\frac{5+4+(2-(3-(6+4/5)))}{3(6-2)(2-7)} \tag{1}$$

```
(/ (+ 5 4 (- 2 (- 3 (+ 6 (/ 4 5)))))
   (* 3 (- 6 2) (- 2 7)))
```

Results

```
-37/150
```

1.3

Define a procedure that takes three numbers as arguments and returns the sum of the squares of the two larger numbers.

Instead of searching for the two larger numbers, we just find the smallest. This way, we don't need to think about duplicate cases etc.

```
(define (proc a b c)
  (cond ((and (> a c) (> b c))
         (+ (* a a) (* b b)))
        ((and (> a b) (> c b))
         (+ (* a a) (* c c)))
        ((and (> b a) (> c a))
         (+ (* b b) (* c c)))))
```

Listing 1: Sum of the two-out-of-three larger numbers

```
<<larger-squares>>
(proc 4 2 3)
```

Listing 2: Testing the previous procedure

Results

```
25
```

1.4

Observe that our model of evaluation allows for combinations whose operators are compound expressions. Use this observation to describe the behavior of the following procedure:

```
(define (a-plus-abs-b a b)
  ((if (> b 0) + -) a b))
```

The operator to be used depends on the evaluation of the `if` statement.