

CS 61A Week 6 Lab

Monday afternoon, Tuesday, or Wednesday morning

1. Load the **Scheme-1** interpreter from the file

```
~cs61a/lib/scheme1.scm
```

To start the interpreter, type **(scheme-1)**. Familiarize yourself with it by evaluating some expressions. Remember: you have all the Scheme primitives for arithmetic and list manipulation; you have **lambda** but not higher-order functions; you don't have **define**. To stop the **scheme-1** interpreter and return to **STk**, just evaluate an illegal expression, such as **()**.

1a. Trace in detail how a simple procedure call such as

```
((lambda (x) (+ x 3)) 5)
```

is handled in **scheme-1**.

1b. Try inventing higher-order procedures; since you don't have **define** you'll have to use the Y-combinator trick, like this:

```
Scheme-1: ((lambda (f n)           ; this lambda is defining MAP
             ((lambda (map) (map map f n))
              (lambda (map f n)
                (if (null? n)
                    '()
                    (cons (f (car n)) (map map f (cdr n)))))) ))
first           ; here are the arguments to MAP
'(the rain in spain))
(t r i s)
```

1c. Since all the Scheme primitives are automatically available in **scheme-1**, you might think you could use **STk**'s primitive **map** function. Try these examples:

```
Scheme-1: (map first '(the rain in spain))
```

```
Scheme-1: (map (lambda (x) (first x)) '(the rain in spain))
```

Explain the results.

1d. Modify the interpreter to add the **and** special form. Test your work. Be sure that as soon as a false value is computed, your **and** returns **#f** without evaluating any further arguments.

For the rest of the lab, start by reading *SICP* section 2.3.3 (pages 151–161).

2. *SICP* ex. 2.62.

3. The file `~cs61a/lib/bst.scm` contains the binary search tree procedures from pages 156–157 of *SICP*. Using **adjoin-set**, construct the trees shown on page 156.