

Specifying the Behavior of Expressions

Exercise 3.1

$\lfloor (\text{value-of } \langle\langle x \rangle\rangle \rho) \rfloor = 10$

$\lfloor (\text{value-of } \langle\langle 3 \rangle\rangle \rho) \rfloor = 3$

$\lfloor (\text{value-of } \langle\langle v \rangle\rangle \rho) \rfloor = 5$

$\lfloor (\text{value-of } \langle\langle i \rangle\rangle \rho) \rfloor = 1$

Exercise 3.2

A $val \in \text{ExpVal}$ must be that which is in $\text{Int} + \text{Bool}$. Then a $val \in \text{ExpVal}$ for which $\lfloor [val] \rfloor \neq val$ is where $val \in \text{Bool}$, such as $val = \text{true}$.

Exercise 3.3

We are able to describe the arithmetic operations in terms of subtraction. We cannot do so if we chose addition.

Exercise 3.4

Let $\rho = [x=[33], y=[22]]$.

$$\frac{\text{(value-of-program } \langle\langle \text{if zero? } (-(x, 11)) \text{ then } -(y, 2) \text{ else } -(y, 4) \rangle\rangle \text{)}}{\frac{\text{(value-of } \langle\langle \text{if zero? } (-(x, 11)) \text{ then } -(y, 2) \text{ else } -(y, 4) \rangle\rangle \rho \text{)}}{\frac{\text{(value-of } \langle\langle \text{zero? } (-(x, 11)) \rangle\rangle \rho \text{)} = (\text{bool-val } \#f)}{\frac{\text{(value-of } \langle\langle -(y, 4) \rangle\rangle \rho \text{)}}{[18]}}$$

Exercise 3.5

$$\frac{
 \begin{array}{l}
 (\text{value-of } \langle\langle \text{let } x = 7 \\
 \quad \text{in let } y = 2 \\
 \text{in let } y = \text{let } x = -(x, 1) \text{ in } -(x, y) \\
 \quad \text{in } -(-(x, 8), y) \rangle\rangle \rho_0)
 \end{array}
 }{
 \text{skib}
 }$$

Exercise 3.6

```

(minus-exp (exp1 expression?))

(minus-exp (exp1)
  (value-of (diff-exp (const-exp 0)
    exp1)
    env))

```

Exercise 3.7

```

(add-exp
  (exp1 expression?)
  (exp2 expression?))
(mul-exp
  (exp1 expression?)
  (exp2 expression?))
(div-exp
  (exp1 expression?)
  (exp2 expression?))

(add-exp (exp1 exp2)
  (num-val (+ (expval->num (value-of exp1 env))
    (expval->num (value-of exp2 env))))))
(mul-exp (exp1 exp2)
  (num-val (* (expval->num (value-of exp1 env))
    (expval->num (value-of exp2 env))))))

```

```

                                (expval->num (value-of exp2 env))))))
(div-exp (exp1 exp2)
  (let ((val2 (expval->num (value-of exp2 env))))
    (if (= 0 val2)
        (report-division-by-zero)
        (num-val (/ (expval->num (value-of exp1 env))
                     val2))))))

```

Exercise 3.8

```

(equal?-exp
  (exp1 expression?)
  (exp2 expression?))
(greater?-exp
  (exp1 expression?)
  (exp2 expression?))
(less?-exp
  (exp1 expression?)
  (exp2 expression?))

(equal?-exp (exp1 exp2)
  (bool-val (= (expval->num (value-of exp1 env))
               (expval->num (value-of exp2 env)))))
(greater?-exp (exp1 exp2)
  (bool-val (> (expval->num (value-of exp1 env))
               (expval->num (value-of exp2 env)))))
(less?-exp (exp1 exp2)
  (bool-val (< (expval->num (value-of exp1 env))
               (expval->num (value-of exp2 env)))))

```

Exercise 3.9

We express a `list-val` as the Scheme cons of two expressed values. To select from the cons we use the Scheme `car` and `cdr`. Alternatively, we could have represented the `list-val` case in `expval->list` in the defined

language as (list-val (first rest)), and use cases to grab either component in the body of car-exp and cdr-exp in value-of.

```
(emptylist-val)
(list-val
 (first expval?)
 (rest expval?))

(define expval->list
  (lambda (val)
    (cases expval val
      (list-val (first rest) (cons first rest))
      (else (report-expval-extractor-error 'list val))))))

(cons-exp
 (exp1 expression?)
 (exp2 expression?))
(car-exp
 (exp1 expression?))
(cdr-exp
 (exp1 expression?))
(null?-exp
 (exp1 expression?))
(emptylist-exp)

(cons-exp (exp1 exp2)
          (list-val (value-of exp1 env)
                    (value-of exp2 env)))
(car-exp (exp1)
         (car (expval->list exp1)))
(cdr-exp (exp1)
         (cdr (expval->list exp1)))
(null?-exp (exp1))
```

```
(let ((val1 (value-of exp1 env)))
  (cases expval val1
    (emptylist-val () (bool-val #t))
    (else (bool-val #f)))))
(emptylist-exp ()
  (emptylist-val))
```