PPL182 – Assignment 2, Due To 01.05.18

Roy Ash, Ron Korine

Question 1 - General Terminology

1.1 What is a “special form”? – a parenthesized expression that opens with a syntactic keyword (such as: define, lambda etc.). The most important thing to specify about it is that unlike non-special form, a special form follows a special evaluation rule for it.

1.2 What is an “atomic expression”? – an atomic expression in scheme is one of the following types of expressions:
-Literal Numbers: written as numbers, for instance: ‘1’ or ‘2’
-Literal Booleans: can be one of the following: #t/#f for true/false accordingly
-Primitives Procedures: procedures such as arithmetic primitive operators like (+, -, *, /) and comparison operators (<, >, =, <=, >=).

1.3 What is a “compound expression”? – a parenthesized expression with a specific structure in which the leftmost sub-expression of the form is an operator and the rest of the sub-expressions are operands.

1.4 What is a “primitive expression”? – expressions which their evaluation is built in the interpreter and which are not explained by the semantics of the language. Primitive expressions include: primitive operations (such as + or -) and primitive literal values (such as numbers and boolean values).

1.5
1. ‘+’: primitive, atomic
2. ‘5’: primitive, atomic
3. ‘x’: atomic
4. ((lambda (x) x) 5): compound

1.6 Side effect

1.7 equivalent

1.8 ((lambda (x y z) (* (x z) y)) (lambda (x) (+ x 1)) ((lambda (y) (- y 22)) 23) 6)

1.9
```
(define check1
  (lambda ()
    (display "x")
    #t))

(define check2
  (lambda ()
    (display "y")
    #f))
```

PPL182 – Assignment 2, Due To 01.05.18

Roy Ash, Ron Korine

```
(define check3
  (lambda ()
    (display "z")
    #t))

(define check
  (lambda (x y z)
    (and (x) (y) (z))))

(check check1 check2 check3) → xy#f
```

כפי שניתן לראות הפונקציה and מפעילה את הפונקציות check1, check2, check3 אחת אחרי השנייה ועוצרת לאחר הפעלת check2 אשר מחזירה false ולא מפעילה את check3. ולכן and פועלת על סמך עקרון Shortcut semantics, שבמסגרתו פונקציות מחזירות ערך false מיד ברגע בו הן מודעות לכך שאלמנט לא עומד בתנאים לקבל ערך true.

1.10. 1. הפונקציות foo ו-goo שקולות פונקציונאלית מכיוון שלכל קלט x הפונקציות הנ״ל יתנו את הערך x+1, וכן הן זורקות אקספשן\נכנסות ללופ אין סופי\עוצרות ומחזירות ערך זהה על אותם הקלטים. אף על פי כן, יש side effect עבור הפונקציה goo (שהינו display) שהופך את foo ו-goo ללא שקולות מבחינת תכנותית, קרי הן לא מתנהגות אותו הדבר כשמביאים בחשבון את ה-side-effects.

2. 1. Evaluate ((define x 12)) [compound special form]
   Evaluate (12) [Atomic]
   Return value: 12
   Add binding: <<x>,12> to the Global Environment
   Return value: void
   Evaluate ((lambda (x))(+ x(+(/ x 2)x))) [compound non-special form]
   (value 0, value1) = (Evaluate(lambda(x)(+ x(+(/ x 2)x)), Evaluate(x))
   Evaluate(lambda(x)) (+ x(+(/ x 2)x))) [compound special form lambda]
   Replace the var x with the 12 (+ x(+(/ x 2)x)))
   Evaluate((+ x(+(/ x 2)x)))) [compound, but non-special form]
   Evaluate(+) [Atomic]
   Return Value: #<Procedure: +>
   Evaluate(x) [Atomic]
   Return Value: 12 (From Global Environment)
   Evaluate ((+(/ x 2)x)) [compound, non-special form]
   (value 0, value 1, value2) = (Evaluate(+), Evaluate((/ x 2)), Evaluate(x))
   Evaluate(+) [Atomic]
   Return Value: #<Procedure: +>
   Evaluate((/ x 2)) [compound non-special form]
   (value 0, value 1, value 2) = (Evaluate(/), Evaluate(x), Evaluate(2))
   Evaluate(/) [Atomic]

Return Value: #<Procedure: />
Evaluate(x) [Atomic]
Return Value: 12 (From Global Environment)
Evaluate(2) [Atomic]
Return Value: 2
Apply-Procedure + on vals: 12, 2
Return Value: 6
Apply-Procedure + on vals: 6, 12
Return Value: 18
Apply-Procedure + on vals: 12, 18
Return Value: 30
And that is the general and ending returning value (Return Value: 30)

2.2. Evaluate((define last (lambda (l) (if (empty? (cdr l)) (car l) (last (cdr l))))) [compound special form]
evaluate((lambda (l) (if (empty? (cdr l)) (car l) (last (cdr l)))) [compound special form]
return value: (closure (l) (if (empty? (cdr l)) (car l) (last (cdr l)))
add the binding ≪last>, (closure (l) (if (empty? (cdr l)) (car l) (last (cdr l)))> to the GE
return value: void
2.3. do 2.2
evaluate((last '(1 2))) [compound non-special form]
(val0, val1)=(evaluate (last), evaluate('(1 2)))
evaluate(last) [atomic]
return value: (closure (l) (if (empty? (cdr l)) (car l) (last (cdr l)))
evaluate('(1 2)) [compound literal expression]
return value: '(1 2) [compound value]
replace l by '(1 2) in (if (empty? (cdr l)) (car l) (last (cdr l))
evaluate(if (empty? (cdr '(1 2))) (car '(1 2)) (last (cdr '(1 2)))) [compound special form]
return value: #f
evaluate((last (cdr '(1 2)))) [compound non-special form]
(val0, val1)=(evaluate(last), evaluate((cdr '(1 2))))
evaluate(last) [atomic]
rerurn value: return value: #<procedure: last>
evaluate((cdr '(1 2))) [compound non-special form]
(val0, val1)=(evaluate(cdr), evaluate('(1 2)))
evaluate(cdr) [atomic]
return value: #<procedure: cdr>
evaluate('(1 2)) [compound value]
return value: '(1 2) [compound value]
apply procedure cdr on '(1 2)
return value: '(2)
apply last on '(2)
return 2
evaluate((empty? (cdr '(1 2)))) [compound non-special form]
(val0, val1)=(evaluate(empty?), evaluate((cdr '(1 2))))

```
    evaluate(empty?) [atomic]
    return value: #<procedure: empty?>
    evaluate((cdr '(1 2))) [compound special form]
      (val0, val1)=(evaluate(cdr), evaluate('(1 2)))
      evaluate(cdr) [atomic]
      return value: #<procedure: cdr>
      evaluate('(1 2)) [compound value]
      return value: '(1 2) [compound value]
      apply procedure cdr on '(1 2)
    return value: '(2)
  apply empty? on '(2)
  return #t
return value: 2
return value 2
return value 2
```

3.

Binding Instance	Appears first at line	Scope	Line #s of bound occurrences
fib?	1	Universal Scope	4, 6
n	1	Lambda body (1)	2-4
y	5	Universal Scope	6
triple	1	Universal Scope	4
x	1	Lambda body (1)	3
y	2	Lambda body (2)	3
z	3	Lambda body (3)	3

4. ממומש בסקים