Practice Mid-Term Exam 1

CS 5510, Fall 2015

September 24

Instructions: You have eighty minutes to complete this open-book, open-note, closed-interpreter exam. Please write all answers in the provided space, plus the back of the exam if necessary.
Note on actual exam: The exam may refer to the `env.rkt`, `lambda.rkt`, and `store-with.rkt` interpreters. If you need the interpreters for reference to answer the questions, please bring a copy (paper or electronic) with you.

1) [15 pts] Given the following grammar:

⟨weed⟩ = `leaf`
| `(branch` ⟨weed⟩ ⟨weed⟩`)`
| `(stem` ⟨weed⟩`)`

Provide a `define-type` declaration for `Weed` that is a suitable representation for ⟨weed⟩s.

2) [25 pts] Implement the function `weed-forks`, takes a `Weed` and returns the number of `branch`es that it contains. Your implementation must follow the shape of the data definition.

3) [20 pts] For each of the following expressions, a show the store that would be returned with the program's value when using the `store-with.rkt` interpreter. Instead of nested `override-store`s, you can show the store as a `list` of `cell`s. Recall that locations are allocated starting at `1`.

a)
```
{box {+ 1 2}}
```

b)
```
{let {[b {box {+ 1 2}}]}
  {begin
    {set-box! b 4}
    {box 5}}}
```

c)
```
{let {[f {lambda {x}
           {box x}}]}
  {set-box! {f 0} {f 1}}}
```

d)
```
{let {[f {lambda {x}
           {box x}}]}
  {let {[b {f 0}]}
    {set-box! b b}}}
```

4) [40 pts] The following expression is evaluated using the `lambda.rkt` interpreter:

```
{let {[g {lambda {x} {lambda {y} {+ y x}}}]}
  {let {[x 13]}
    {let {[f {g 6}]}
      {f x}}}}
```

(**Note:** the actual exam will also use `lambda.rkt`.) Describe a trace of the evalaution in terms of arguments to an `interp` function for every call. (There will be 15 calls.) The `interp` function takes two arguments — an expression and an environment — so show both for each call. For each number, variable, and `lambda` expression, show the result value, which is immediate. (To simplify your job, you do not need to show results for other expressions, but you can show results if you prefer.) Use the back of the exam for additional space, and use the following abbreviations to save time:

X_0 = the whole expression
X_1 = `{lambda {x} {lambda {y} {+ y x}}}`
X_2 = `{let {[x 13]} {let {[f {g 6}]} {f x}}}`
X_3 = `{let {[f {g 6}]} {f x}}`

Answers

1) [15 pts]

```
(define-type Weed
  [leaf]
  [stem (rest : Weed)]
  [branch (left : Weed)
          (right : Weed)])
```

2) [25 pts]

```
; weed-forks : Weed -> num
(define (weed-forks w)
  (type-case Weed w
   [leaf () 0]
   [stem (rest) (weed-forks rest)]
   [branch (l r) (+ 1
                    (weed-forks l)
                    (weed-forks r))]))
(test (weed-forks (leaf)) 0)
(test (weed-forks (stem (leaf))) 0)
(test (weed-forks (stem (branch (leaf) (leaf)))) 1)
(test (weed-forks (branch (branch (leaf) (leaf)) (leaf))) 2)
```

3) [20 pts]

a) `(list (cell 1 (numV 3)))`

b) `(list (cell 2 (numV 5)) (cell 1 (numV 4)) (cell 1 (numV 3)))`

c) `(list (cell 1 (boxV 2)) (cell 2 (numV 1)) (cell 1 (numV 0)))`

c) `(list (cell 1 (boxV 1)) (cell 1 (numV 0)))`

4) [40 pts]

expr = $\boxed{X_0}$
env = `mt-env`

expr = $\boxed{X_1}$
env = `mt-env`
result = `(closV 'x` $\boxed{\texttt{\{lambda \{y\} \{+ y x\}\}}}$ `mt-env)` = C_1

expr = $\boxed{X_2}$
env = `(extend-env (bind 'g` C_1`) mt-env)` = E_1

expr = $\boxed{\texttt{13}}$
env = E_1
result = `(numV 13)`

expr = $\boxed{X_3}$

env = `(extend-env (bind 'x (numV 13)) ` E_1 `) = ` E_2

expr = `{g 6}`
env = E_2

expr = `g`
env = E_2
result = C_1

expr = `6`
env = E_2
result = `(numV 6)`

expr = `{lambda {y} {+ y x}}`
env = `(extend-env (bind 'x (numV 6)) mt-env) = ` E_3
result = `(closV 'y` `{+ y x}` E_3`) = ` C_2

expr = `{f x}`
env = `(extend-env (bind 'f` C_2`)` E_2`) = ` E_4

expr = `f`
env = E_4
result = C_2

expr = `x`
env = E_4
result = `(numV 13)`

expr = `{+ y x}`
env = `(extend-env (bind 'y (numV 13)) ` E_3`) = ` E_5

expr = `y`
env = E_5
result = `(numV 13)`

expr = `x`
env = E_5
result = `(numV 6)`