Practice Midterm Exam 1

CS 3520/5520, Fall 2018

September 25

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, and **it must include tests**.

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. Put each call to `interp` and in a rectangle, and show recursive calls within the same rectangle by using nested rectangles; show the `interp` result at the end of each rectangle. Number the calls to `interp` to reflect the actual order of the calls when running the interpreter.

You can omit the rectangle around the first call and everything else, leaving it implicit. If you can't get all of a box's content on one page, you can write a placeholder for a nested box and write the actual box separately (possibly on a separate page). It's ok to leave out calls to `parse` and just quote concrete syntax. Use the following abbreviations to save time and space:

X_0 = the whole expression, quoted
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]

```
(define (weed-forks [w : Weed]) : Number
  (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]

1 (interp X_0
mt-env)

2 (interp X_1
mt-env)
result is C_1 = (closV 'x `{lambda {y} {+ y x}} mt-env)

3 (interp X_2
E_1 = (extend-env (bind 'g C_1) mt-env))

4 (interp `13
E_1)
result is (numV 13)

5 (interp X_3
E_2 = (extend-env (bind 'x (numV 13)) E_1))

6 (interp `{g 6}
E_2)

7 (interp `g
E_2)
result is C_1

8 (interp `6
E_2)
result is (numV 6)

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

result is C_2

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

11 (interp `f
E_4)
result is C_2

12 (interp `x
E_4)
result is (numV 13)

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

14 (interp `y
E_5)
result is (numV 13)

15 (interp `x
E_5)
result is (numV 6)

result is (numV 19)

result is (numV 19)

result is (numV 19)

result is (numV 19)

result is (numV 19)

An alternate form of the same solution that fits more easily on multiple pages:

1 `(interp` X_0
`mt-env)`

2 `(interp` X_1
`mt-env)`
result is C_1 = `(closV 'x` `` `{lambda {y} {+ y x}} `` `mt-env)`

3 `(interp` X_2
E_1 = `(extend-env (bind 'g` C_1`) mt-env))`

4 `(interp` `` `13 ``
E_1`)`
result is `(numV 13)`

5 `(interp` X_3
E_2 = `(extend-env (bind 'x (numV 13))` E_1`))`
BOX A
BOX B
result is `(numV 19)`

result is `(numV 19)`

result is `(numV 19)`

BOX A =

6 `(interp` `` `{g 6} ``
E_2`)`

7 `(interp` `` `g ``
E_2`)`
result is C_1

8 `(interp` `` `6 ``
E_2`)`
result is `(numV 6)`

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

result is C_2

BOX B =

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

11 `(interp` `` `f ``
E_4`)`
result is C_2

12 `(interp` `` `x ``
E_4`)`
result is `(numV 13)`

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

14 `(interp` `` `y ``
E_5`)`
result is `(numV 13)`

15 `(interp` `` `x ``
E_5`)`
result is `(numV 6)`

result is `(numV 19)`

result is `(numV 19)`