Q1	Q2	Q3	Q4	Q5	TOTAL
55	50	50	25	–	200

bonus

Definition of Programming Languages

CPSC 311 2015W1

University of British Columbia

Practice Final Examination—Episode One, **plus Q3**

Joshua Dunfield

Student Name:

Signature:

Student Number:

INSTRUCTIONS

- This is a CLOSED BOOK / CLOSED NOTES examination.
- Write all answers ON THE EXAMINATION PAPER.

  Try to write your answers in the blanks or boxes given. If you can't, try to write them elsewhere on the same page, or on one of the worksheet pages, and **LABEL THE ANSWER**. We can't give credit for answers we can't find.
- Blanks are suggestions. You may not need to fill in every blank.

Question 1 [55 points]: "If it's 'dynamic', it *must* be better."

The following rules define an environment-based semantics for lexically-scoped functions, lam, and dynamically-scoped functions, ds-lam.

$\boxed{env \vdash e \Downarrow v}$ Under environment env, expression e evaluates to value v

$$\frac{}{env \vdash (\mathsf{num}\ n) \Downarrow (\mathsf{num}\ n)}\ \text{Env-num} \qquad \frac{env \vdash e1 \Downarrow (\mathsf{num}\ n1) \qquad env \vdash e2 \Downarrow (\mathsf{num}\ n2)}{env \vdash (\mathsf{add}\ e1\ e2) \Downarrow (\mathsf{num}\ n1{+}n2)}\ \text{Env-add}$$

$$\frac{env \vdash e1 \Downarrow v1 \qquad x{=}v1, env \vdash e2 \Downarrow v2}{env \vdash (\mathsf{with}\ x\ e1\ e2) \Downarrow v2}\ \text{Env-with}$$

$$\frac{lookup(env, x) = e}{env \vdash (\mathsf{id}\ x) \Downarrow e}\ \text{Env-id} \qquad \frac{lookup(env, x)\ \text{undefined}}{env \vdash (\mathsf{id}\ x)\ \mathsf{unknown\text{-}id\text{-}error}}\ \text{Env-unknown-id}$$

$$\frac{}{env \vdash (\mathsf{lam}\ x\ e1) \Downarrow \big(\mathsf{clo}\ env\ (\mathsf{lam}\ x\ e1)\big)}\ \text{Env-lam} \qquad \frac{}{env \vdash (\mathsf{clo}\ env_{old}\ e) \Downarrow (\mathsf{clo}\ env_{old}\ e)}\ \text{Env-clo}$$

$$\frac{env \vdash e1 \Downarrow (\mathsf{clo}\ env_{old}\ (\mathsf{lam}\ x\ eB)) \qquad env \vdash e2 \Downarrow v2 \qquad x{=}v2, env_{old} \vdash eB \Downarrow v}{env \vdash (\mathsf{app}\ e1\ e2) \Downarrow v}\ \text{Env-app}$$

$$\frac{}{env \vdash (\mathsf{ds\text{-}lam}\ x\ e1) \Downarrow (\mathsf{ds\text{-}lam}\ x\ e1)}\ \text{Env-ds-lam} \qquad \frac{env \vdash e1 \Downarrow (\mathsf{ds\text{-}lam}\ x\ eB) \qquad env \vdash e2 \Downarrow v2 \qquad x{=}v2, env \vdash eB \Downarrow v}{env \vdash (\mathsf{app}\ e1\ e2) \Downarrow v}\ \text{Env-ds-app}$$

Assume that $lookup(env, x)$ returns the **leftmost** binding of x. For example:

$$lookup\Big(\big(x{=}(\mathsf{num}\ 2), x{=}(\mathsf{num}\ 1), \emptyset\big),\ x\Big) \ = \ (\mathsf{num}\ 2)$$

Consider the following expression, shown in concrete syntax (left) and in abstract syntax (right).

```
{with {y 100}
 {with {f {with {y 10}
               {lam x {+ x y}}}}
       {with {y 2}
             {app f y}}}}
```

$$\Big(\mathsf{with}\ y\ (\mathsf{num}\ 100)$$
$$\quad \Big(\mathsf{with}\ f\ \big(\mathsf{with}\ y\ (\mathsf{num}\ 10)$$
$$\qquad\qquad (\mathsf{lam}\ x\ (\mathsf{add}\ (\mathsf{id}\ x)\ (\mathsf{id}\ y)))\big)$$
$$\qquad \big(\mathsf{with}\ y\ (\mathsf{num}\ 2)$$
$$\qquad\qquad (\mathsf{app}\ (\mathsf{id}\ f)\ (\mathsf{id}\ y))\big)\Big)\Big)$$

Q1a [10 points] Complete the **abstract syntax tree** for the above expression. $\Longrightarrow$

```
      with
     /  |  \
    y  num
       100
```

Question 1 [55 points]: "If it's 'dynamic', it *must* be better." (cont.)

Q1b [10 points] If we evaluate the above expression in the empty environment, what value do we get?

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Q1c [10 points] While evaluating the above expression, we will evaluate the body of the lam. When we evaluate that expression, (add (id x) (id y)), what is the complete environment?

_ _

Q1d [15 points] **Warning:** Dynamic scope ahead!

If we evaluate the expression that is **almost** the same, but has ds-lam in place of lam, as shown **below**, what value do we get? _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

```
{with {y 100}
 {with {f {with {y 10}
              {ds-lam x {+ x y}}}}
      {with {y 2}
            {app f y}}}}
```

```
(with y (num 100)
  (with f (with y (num 10)
                (ds-lam x (add (id x) (id y))))
     (with y (num 2)
           (app (id f) (id y)))))
```

Q1e [10 points] (No more dynamic scope. Yay!)

This will be a question about substitution.
It should be roughly similar to Q1 on the midterm (and the practice midterm).

Worksheet (i)

Worksheet (ii)

Question 2 [50 points]: Little Perennials II

The *expression strategy*, *value strategy*, and *lazy evaluation* are different ways of evaluating a function application (app $e1$ $e2$). All strategies evaluate $e1$ to (lam x eB), but they differ in how they handle $e2$:

- The expression strategy evaluates eB with x bound to a *thunk* containing $e2$. (The thunk (thk env $e2$) saves the current environment env, to make sure we *don't* use dynamic scoping.)
- The value strategy evaluates $e2$ to a value $v2$, then evaluates eB with x bound to that value.
- Lazy evaluation creates a *lazy thunk*, $\ell \triangleright$(lazy-thk env $e2$), in the store, and evaluates eB with x bound to (lazy-ptr ℓ). If (id x) is evaluated, we evaluate $e2$ to a value $v2$, and replace $\ell \triangleright$(lazy-thk env $e2$) with $\ell \triangleright v2$ (rule SEnv-lazy-ptr). If (id x) is evaluated again, rule SEnv-lazy-ptr-done looks up the value $v2$, without evaluating $e2$ again.

If you need to, you can refer to the following evaluation rules:

$\boxed{env; S \vdash e \Downarrow v; S'}$ Under environment env and store S, expression e evaluates to v with updated store S'

$$\frac{}{env; S \vdash (\mathsf{lam}\ x\ e1) \Downarrow \big(\mathsf{clo}\ env\ (\mathsf{lam}\ x\ e1)\big); S}\ \text{SEnv-lam} \qquad \frac{}{env; S \vdash (\mathsf{clo}\ env_{\mathrm{old}}\ e) \Downarrow (\mathsf{clo}\ env_{\mathrm{old}}\ e); S}\ \text{SEnv-clo}$$

$$\frac{env; S \vdash e1 \Downarrow (\mathsf{clo}\ env_{\mathrm{old}}\ (\mathsf{lam}\ x\ eB)); S1 \qquad env; S1 \vdash e2 \Downarrow v2; S2 \qquad x{=}v2, env_{\mathrm{old}}; S2 \vdash eB \Downarrow v; S'}{env; S \vdash (\mathsf{app}\ e1\ e2) \Downarrow v; S'}\ \text{SEnv-app-}\textbf{value}$$

$$\frac{env; S \vdash e1 \Downarrow (\mathsf{clo}\ env_{\mathrm{old}}\ (\mathsf{lam}\ x\ eB)); S1 \qquad x{=}(\mathsf{thk}\ env\ e2), env_{\mathrm{old}}; S1 \vdash eB \Downarrow v; S2}{env; S \vdash (\mathsf{app}\ e1\ e2) \Downarrow v; S2}\ \text{SEnv-app-}\textbf{expr}$$

$$\frac{env; S \vdash e1 \Downarrow (\mathsf{clo}\ env_{\mathrm{old}}\ (\mathsf{lam}\ x\ eB)); S1 \qquad x{=}(\mathsf{lazy\text{-}ptr}\ \ell), env_{\mathrm{old}}; \ell \triangleright (\mathsf{lazy\text{-}thk}\ env\ e2),\ S1 \vdash eB \Downarrow v; S2}{env; S \vdash (\mathsf{app}\ e1\ e2) \Downarrow v; S2}\ \text{SEnv-app-}\textbf{lazy}$$

$$\frac{\textit{lookup-loc}(S, \ell) = (\mathsf{lazy\text{-}thk}\ env_{\mathrm{arg}}\ e2) \qquad env_{\mathrm{arg}}; S \vdash e2 \Downarrow v; S1 \qquad \textit{update-loc}(S1, \ell, v) = S2}{env; S \vdash (\mathsf{lazy\text{-}ptr}\ \ell) \Downarrow v; S2}\ \text{SEnv-lazy-ptr}$$

$$\frac{\textit{lookup-loc}(S, \ell) = v2 \qquad v2 \neq (\mathsf{lazy\text{-}thk}\ \cdots\ \cdots)}{env; S \vdash (\mathsf{lazy\text{-}ptr}\ \ell) \Downarrow v2; S}\ \text{SEnv-lazy-ptr-done}$$

$$\frac{env_{\mathrm{old}}; S \vdash e \Downarrow v; S'}{env; S \vdash (\mathsf{thk}\ env_{\mathrm{old}}\ e) \Downarrow v; S'}\ \text{SEnv-thk} \qquad \frac{\textit{lookup}(env, x) = e \qquad env; S \vdash e \Downarrow v; S'}{env; S \vdash (\mathsf{id}\ x) \Downarrow v; S'}\ \text{SEnv-id}$$

$$\frac{env; S \vdash e1 \Downarrow (\mathsf{num}\ n1); S1 \qquad env; S1 \vdash e2 \Downarrow (\mathsf{num}\ n2); S'}{env; S \vdash (\mathsf{add}\ e1\ e2) \Downarrow (\mathsf{num}\ n1 + n2); S'}\ \text{SEnv-add}$$

$$\frac{env; S \vdash e1 \Downarrow (\mathsf{num}\ n1); S1 \qquad env; S1 \vdash e2 \Downarrow (\mathsf{num}\ n2); S'}{env; S \vdash (\mathsf{sub}\ e1\ e2) \Downarrow (\mathsf{num}\ n1 - n2); S'}\ \text{SEnv-sub}$$

Question 2 [50 points]: Little Perennials II, continued

Consider the following expression:

```
{app {app {lam x {lam y {+ y y}}}
          {- 7 1}}
     {- 10 1}}
```

$$\Big(\mathsf{app}\ \big(\mathsf{app}\ (\mathsf{lam}\ x\ (\mathsf{lam}\ y\ (\mathsf{add}\ (\mathsf{id}\ y)\ (\mathsf{id}\ y))))\ (\mathsf{sub}\ (\mathsf{num}\ 7)\ (\mathsf{num}\ 1))\big)\ (\mathsf{sub}\ (\mathsf{num}\ 10)\ (\mathsf{num}\ 1))\Big)$$

Q2a [15 points] If we implement the **SEnv-app-value** rule (and *not* the SEnv-app-expr and SEnv-app-lazy rules) and evaluate the above expression, we will perform

____ addition(s), and ____ subtraction(s).

Q2b [10 points] Now we switch from the value strategy to the expression strategy. Complete the derivation tree for the second premise of SEnv-app-expr.

$$\dfrac{\dfrac{}{\emptyset;\emptyset \vdash (\mathsf{lam}\ x\ eBody) \Downarrow (\mathsf{clo}\ \emptyset\ (\mathsf{lam}\ x\ eBody));\emptyset}\,\text{SEnv-lam} \qquad \boxed{\quad\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\quad}}{\emptyset;\emptyset \vdash \Big(\mathsf{app}\ \big(\mathsf{lam}\ x\ \underbrace{(\mathsf{lam}\ y\ (\mathsf{add}\ (\mathsf{id}\ y)\ (\mathsf{id}\ y)))}_{eBody}\big)\ \underbrace{(\mathsf{sub}\ (\mathsf{num}\ 7)\ (\mathsf{num}\ 1))}_{eSub}\Big) \Downarrow \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_}\,\text{SEnv-app-}\mathbf{expr}$$

Q2c [10 points] If we implement the **SEnv-app-expr** rule **instead** of SEnv-app-value, and evaluate the above expression, we will perform

____ addition(s), and ____ subtraction(s).

Q2d [15 points] If we implement the **SEnv-app-lazy** rule **instead** of SEnv-app-value, and evaluate the above expression, we will perform

____ addition(s), and ____ subtraction(s).

Question 3 [50 points]: Big Log

The small-step semantic interpreters we have seen so far do not include side effects. Interesting side effects include state and input/output.

An interpreter can be extended with input/output by introducing a `print` construct, and we can model the effect of printing by appending to an output buffer B:

$\boxed{\mathrm{B}; e \longrightarrow \mathrm{B}'; e'}$ With starting buffer B, expression e steps to expression e' and an updated buffer B'

Reduction rules:

$$\frac{}{\mathrm{B}; (\mathsf{add}\ (\mathsf{num}\ n1)\ (\mathsf{num}\ n2)) \longrightarrow \mathrm{B}; (\mathsf{num}\ n1{+}n2)}\ \text{Step-add}$$

$$\frac{}{\mathrm{B}; (\mathsf{with}\ x\ v1\ e2) \longrightarrow \mathrm{B}; \mathit{subst}(e2, x, v1)}\ \text{Step-with}$$

$$\frac{\mathrm{B2} = \mathit{append}(\mathrm{B1}, v)}{\mathrm{B1}; (\mathsf{print}\ v) \longrightarrow \mathrm{B2}; v}\ \text{Step-print}$$

Context rule:

$$\frac{\mathrm{B}; e \longrightarrow \mathrm{B}'; e'}{\mathrm{B}; \mathcal{C}[e] \longrightarrow \mathrm{B}'; \mathcal{C}[e']}\ \text{Step-context}$$

Rule Step-context uses the following evaluation contexts:

$$\begin{aligned} \mathcal{C} ::= &\ [\,] \\ &\mid (\mathsf{add}\ \mathcal{C}\ e) \\ &\mid (\mathsf{add}\ v\ \mathcal{C}) \\ &\mid (\mathsf{with}\ x\ \mathcal{C}\ e) \\ &\mid (\mathsf{print}\ \mathcal{C}) \end{aligned}$$

This question uses the following **define-type** definitions, and functions with the following signatures:

```
(define-type Res    ; Type of result of 'reduce' and 'step':
  [RES (B Buffer?) (e E?)])   ; a buffer and an expression

(define-type Buffer
  [buffer/empty]
  [buffer/append (head Buffer?) (tail E?)])

; append-buffer : Buffer E → Buffer
;
; (append-buffer B1 v) = append(B1, v)

; value? : E → boolean
;
; Returns #true iff e is a num.
```

Go to the next page.

Question 3 [50 points]: Big Log, continued

Q3a [15 points] In the function `reduce` (below), implement the rule Step-print.

```
; reduce : Buffer E → (or Res false)
; Given a buffer B and expression e, return (RES B2 e2) where B; e ⟶ B2; e2
; using a reduction rule, or #false if no reduction rule can be applied.
(define (reduce B e)
  (type-case E e
    [add (e1 e2)
         (if (and (value? e1) (num? e1)
                  (value? e2) (num? e2))
             (RES B
                  (num (+ (num-n e1) (num-n e2))))
             #false)]
    ; ...
    [print (e1)

       ------------------------------------------------

       ------------------------------------------------

       ------------------------------------------------

       ------------------------------------------------
    ]))
```

Q3b [10 points] In the function `step` (below), implement the evaluation context (print $\mathcal{C}$).

```
; step : Buffer E → (or Res false)
; Given a buffer B and expression e, return (RES B2 e2) where B; e ⟶ B2; e2
; using rule Step-context, or #false if no derivation of B; e ⟶ B2; e2 exists.
(define (step B e)
  (or (reduce B e)
      (type-case E e
        [add (e1 e2)
           (if (step B e1)
               (type-case Res (step B e1)   ; C ::= (add C e2)
                 [RES (B2 s1)
                      (RES B2 (add s1 e2))])
               (if (and (value? e1) (step B e2))
                   (type-case Res (step B e2)   ; C ::= (add v C)
                     [RES (B2 s2)
                        (if s2
                            (RES B2 (add e1 s2))
                            #false)])
                   #false)
               )])]
        ; ...
        [print (e1)

           --------------------------------------------------

           --------------------------------------------------

           --------------------------------------------------

           --------------------------------------------------
```

```
-------------------------------------------------------------------- ])))
```

Question 3 [50 points]: Big Log, continued

Q3c [25 points] We can introduce concurrency into our language by adding angelic nondeterminism:

$$\frac{}{\mathsf{B}; (\mathsf{par}\ v1\ e2) \longrightarrow \mathsf{B}; v1}\ \text{Step-par-left}$$

$$\frac{}{\mathsf{B}; (\mathsf{par}\ e1\ v2) \longrightarrow \mathsf{B}; v2}\ \text{Step-par-right}$$

$$\begin{aligned} \mathcal{C} &::= \cdots \\ &\mid (\mathsf{par}\ \mathcal{C}\ e) \\ &\mid (\mathsf{par}\ e\ \mathcal{C}) \end{aligned}$$

Let $\longrightarrow^*$ be the transitive/reflexive closure of $\longrightarrow$.

That is, $e \longrightarrow^* e'$ if either $e' = e$, or $e \longrightarrow e2$ and $e2 \longrightarrow^* e'$.

Suppose our program is this expression e:

$$e \;=\; \Big(\mathsf{print}\ \big(\mathsf{par}\ (\mathsf{with}\ r2\ (\mathsf{print}\ (\mathsf{num}\ 2))\ (\mathsf{num}\ 22))\ (\mathsf{with}\ r3\ (\mathsf{print}\ (\mathsf{num}\ 3))\ (\mathsf{num}\ 33))\big)\Big)$$

The final result of e is not always the same; both the buffer and the resulting value can vary.

For example:

$$\langle\,\rangle; e \longrightarrow^* \langle(\mathsf{num}\ 2), (\mathsf{num}\ 22)\rangle; (\mathsf{num}\ 22)$$

by the intermediate steps

$$\begin{aligned}
\langle\,\rangle; e &\longrightarrow \langle(\mathsf{num}\ 2)\rangle; \Big(\mathsf{print}\ \big(\mathsf{par}\ (\mathsf{with}\ r2\ (\mathsf{num}\ 2)\ (\mathsf{num}\ 22))\ (\mathsf{with}\ r3\ (\mathsf{print}\ (\mathsf{num}\ 3))\ (\mathsf{num}\ 33))\big)\Big) \\
&\longrightarrow \langle(\mathsf{num}\ 2)\rangle; \Big(\mathsf{print}\ \big(\mathsf{par}\ (\mathsf{num}\ 22)\ (\mathsf{with}\ r3\ (\mathsf{print}\ (\mathsf{num}\ 3))\ (\mathsf{num}\ 33))\big)\Big) \\
&\longrightarrow \langle(\mathsf{num}\ 2)\rangle; \big(\mathsf{print}\ (\mathsf{num}\ 22)\big) \\
&\longrightarrow \langle(\mathsf{num}\ 2), (\mathsf{num}\ 22)\rangle; (\mathsf{num}\ 22)
\end{aligned}$$

Fill in the intermediate computation steps:

$$\langle\,\rangle; e \longrightarrow \langle(\mathsf{num}\ 3)\rangle; \Big(\mathsf{print}\ \big(\mathsf{par}\ (\mathsf{with}\ r2\ (\mathsf{print}\ (\mathsf{num}\ 2))\ (\mathsf{num}\ 22))\ (\mathsf{with}\ r3\ (\mathsf{num}\ 3)\ (\mathsf{num}\ 33))\big)\Big)$$

$\longrightarrow$ ____

$\longrightarrow$ ____

$\longrightarrow$ ____

$\longrightarrow$ ____

Note: Different solutions are possible!

Question 4 [25 points]: The Criminal Cats of West 11th Avenue

The subsumption principle states that, if $A1 <: A2$ (meaning that $A1$ is a subtype of $A2$), then any value of type $A1$ can safely be used wherever a value of type $A2$ is required.

Each part of this question proposes one or more subtyping rules. In each part, determine whether the rules proposed maintain the subsumption principle or violate it. If they violate the subsumption principle, give an example of an expression of type $A1$ that cannot be safely used where an expression of type $A2$ is expected.

Assume, in all the parts, that we have the following subtyping and typing rules that allow us to distinguish positive (≥ 0) rational numbers from negative (≤ 0) rational numbers through types Pos and Neg, which are both subtypes of Rat.

Also assume there is a function `print-pos` : Pos $\rightarrow$ Pos that can only print positive rationals, and will crash if given a rational that is less than zero.

$$\frac{}{\textsf{Pos} <: \textsf{Rat}} \qquad \frac{}{\textsf{Neg} <: \textsf{Rat}} \qquad \frac{n \geq 0}{\Gamma \vdash (\textsf{num}\ n) : \textsf{Pos}} \qquad \frac{n \leq 0}{\Gamma \vdash (\textsf{num}\ n) : \textsf{Neg}} \qquad \frac{\Gamma \vdash e : A}{\Gamma \vdash (\textsf{ref}\ e) : \textsf{Ref}\ A}$$

Example Proposed rule:

$$\frac{}{\textsf{Rat} <: \textsf{Pos}}$$

Does this proposed rule: ☐ maintain the subsumption principle, or

✓ violate it (example expression: (num −3) ____________)

Q4a
[10 points] In this problem, the abstract syntax of an expression that creates a ref is $(\textsf{ref}\ e)$.

Proposed rule:

$$\frac{A <: A' \qquad A' <: A}{(\textsf{Ref}\ A) <: (\textsf{Ref}\ A')}$$

Does this proposed rule: ☐ maintain the subsumption principle, or

☐ violate it (example expression: ______________________)

where $A =$ ___________ and $A' =$ ___________

Q4b
[15 points] The feline criminals of West 11th Avenue (perhaps confused about the meaning of "Rat") have proposed that any function taking a Rat as its argument can be used as a Rat.

Recall some relevant typing rules:

$$\frac{\Gamma \vdash e1 : \textsf{Rat} \qquad \Gamma \vdash e2 : \textsf{Rat}}{\Gamma \vdash (\textsf{add}\ e1\ e2) : \textsf{Rat}} \qquad \frac{x : A, \Gamma \vdash eBody : B}{\Gamma \vdash (\textsf{lam}\ x\ eBody) : (A \rightarrow B)} \qquad \frac{\Gamma \vdash e1 : (A \rightarrow B) \qquad \Gamma \vdash e2 : B}{\Gamma \vdash (\textsf{app}\ e1\ e2) : B}$$

Proposed subtyping rule:

$$\frac{}{(\textsf{Rat} \rightarrow B) <: \textsf{Rat}}$$

Does this proposed rule: ☐ maintain the subsumption principle, or

☐ violate it (example expression: ______________________)

where $B =$ ___________

Worksheet (iii)