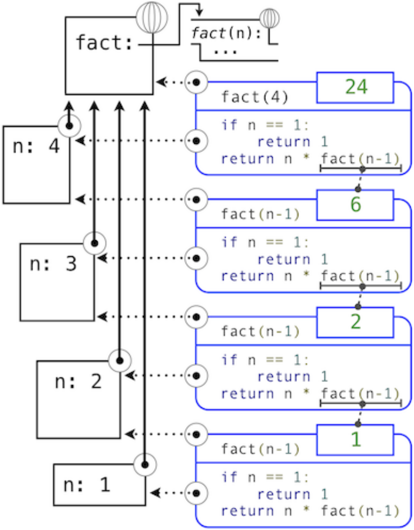
The Environment Model of Evaluation

Jim Royer

CIS 352

March 22, 2019



References

- *Structure and Interpretation of Computer Programs, 2/e, §3.2: The Environment Model of Evaluation,* by Harold Abelson and Gerald Sussman, MIT Press, 1996. https://mitpress.mit.edu/sicp/full-text/book/book.html
- William Cook, *Anatomy of Programming Languages*, Chapters 3 and 4, http://www.cs.utexas.edu/~wcook/anatomy/anatomy.htm

LFP = LC + λ + function application + variables

LFP Expressions

$$
\begin{aligned}
E ::= {} & n \mid b \mid \ell \mid E \; iop \; E \mid E \; cop \; E \mid \textbf{if } E \textbf{ then } E \textbf{ else } E \\
& \mid \, !E \mid E := E \mid \textbf{skip} \mid E;\, E \mid \textbf{while } E \textbf{ do } E \\
& \mid \underbrace{x \mid \lambda x.E \mid E\; E}_{\text{the } \lambda\text{-calculus}} \mid \textbf{let } x = E \textbf{ in } E
\end{aligned}
$$

where

- $x \in \mathbb{V}$, an infinite set of variables
- $n \in \mathbb{Z}$ (integers), $b \in \mathbb{B}$ (booleans), $\ell \in \mathbb{L}$ (locations)
- $iop \in$ (integer-valued binary operations)
- $cop \in$ (boolean-valued binary comparisons)

We focus on the (λ-calculus + **let**) part of LFP.

Application via substitution and its problems

Call by name

$$\Downarrow\text{-}cbn: \quad \frac{\langle E_1, s \rangle \Downarrow \langle \lambda x.E_1', s' \rangle \quad \langle E_1'[E_2/x], s' \rangle \Downarrow \langle V, s'' \rangle}{\langle (E_1\ E_2), s \rangle \Downarrow \langle V, s'' \rangle}$$

Call by value

$$\Downarrow\text{-}cbv: \quad \frac{\langle E_1, s \rangle \Downarrow \langle \lambda x.E_1', s' \rangle \quad \langle E_2, s' \rangle \Downarrow \langle V_2, s'' \rangle \quad \langle E_1'[V_2/x], s'' \rangle \Downarrow \langle V, s''' \rangle}{\langle (E_1\ E_2), s \rangle \Downarrow \langle V, s''' \rangle}$$

- Call-by-name and call-by-value are defined above via *substitution*.
- Substitution is:

  dandy for nailing down sensible meanings of application.

  stinko for everyday implementations.

  E.g., An implementation via substitution constantly needs to modify a program's source code.

Idea: In place of substituting a value v for a variable x:

- Keep a dictionary of variables *&* their values.
- When you need the value of x, look it up.

Environments (*Warning:* Scary Greek letters)

Definition

An environment is just a table of *variables* and associated *values*.

Consider an expression $e = \textbf{if } z \textbf{ then } x \textbf{ else } y + 2$.

- With environment $\{ x \mapsto 3, y \mapsto 4, z \mapsto \textbf{tt} \}$, e evaluates to 3.
- With environment $\{ x \mapsto 8, y \mapsto 5, z \mapsto \textbf{ff} \}$, e evaluates to 7.
- Etc.

$\text{lookup}(\rho, x)$

returns the value (if any) of x in environment ρ.

$\text{update}(\rho, x, v)$

returns a new environment $\rho[x \mapsto v]$

($\rho[x \mapsto v]$ is just like ρ except x has value v.)

Evaluating variable x in environment $\rho \quad \equiv \quad \text{lookup}(\rho, x)$.

Revising call-by-value big-step semantics, 1

Definition

$\rho \vdash \langle e, s \rangle \Downarrow_V \langle v, s' \rangle$ means that expression e with environment ρ and state s evaluates to value v and state s'.

$$\textit{Var:}\ \frac{}{\rho \vdash \langle x, s \rangle \Downarrow_V \langle v, s \rangle}\ (v = \mathrm{lookup}(\rho, x))$$

$$\textit{Let:}\ \frac{\rho \vdash \langle e_1, s \rangle \Downarrow_V \langle v_1, s' \rangle \qquad \rho[x \mapsto v_1] \vdash \langle e_2, s' \rangle \Downarrow_V \langle v_2, s'' \rangle}{\rho \vdash \langle \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2, s \rangle \Downarrow_V \langle v_2, s'' \rangle}$$

Examples/Exercises: Let $\rho = \{ x \mapsto 7, y \mapsto 3 \}$.

- $\rho \vdash \langle x + y, s \rangle \Downarrow_V ??$
- $\rho \vdash \langle \mathbf{let}\ x = 1\ \mathbf{in}\ x + y, s \rangle \Downarrow_V ??$
- $\rho \vdash \langle \mathbf{let}\ x = 1\ \mathbf{in}\ (\mathbf{let}\ z = 11\ \mathbf{in}\ x + y + z), s \rangle \Downarrow_V ??$

Revising call-by-value big-step semantics, 2

Preliminary versions of these rules:

$$\textit{App:}\ \frac{\begin{array}{rcl} \rho \vdash \langle e_1, s \rangle & \Downarrow_V & \langle \lambda x.e_1', s' \rangle \\ \rho \vdash \langle e_2, s' \rangle & \Downarrow_V & \langle v_2, s'' \rangle \\ \rho[x \mapsto v_2] \vdash \langle e_1', s'' \rangle & \Downarrow_V & \langle v, s''' \rangle \end{array}}{\rho \vdash \langle (e_1\ e_2), s \rangle \ \Downarrow_V \ \langle v, s''' \rangle} \qquad \textit{Fun:}\ \frac{}{\rho \vdash \langle \lambda x.e, s \rangle \Downarrow_V \langle \lambda x.e, s \rangle}$$

Examples/Exercises: Let $\rho = \{ x \mapsto 7, y \mapsto 3 \}$.

- $\rho \vdash \langle \textbf{let}\ f = \lambda x.(x+y)\ \textbf{in}\ (f\ 10), s \rangle \Downarrow_V ??$

!!! $\rho \vdash \langle \textbf{let}\ f = \lambda x.(x+y)\ \textbf{in}\ (\textbf{let}\ y = 100\ \textbf{in}\ (f\ 10)), s \rangle \Downarrow_V ??$

Scoping

Definition (Variable Scope)

The scope of a variable binding/declaration is the region of a program where the binding is valid, i.e., when you use the variable, it uses that declaration for the binding (meaning) of the name.

A Java example (static/lexical scoping)

```
{
    int i = 23;
    for (int i = 1; i<11; i++) { ...}
    System.out.println(i);
    ...
}
```

- the outer i's scope
- the inner i's scope

Dynamic Scoping, 1

Re: λ-expressions, functions, procedures, etc.,
there are two sorts of environments you have to worry about:

1. The environment in force when the function was *created*.
2. The environment in force when the function is *applied*.

$$\textit{Dynamic-App:}\ \frac{\rho \vdash \langle e_1, s \rangle \Downarrow_V \langle \lambda x.e_1', s' \rangle \qquad \rho \vdash \langle e_2, s' \rangle \Downarrow_V \langle v_2, s'' \rangle \qquad ☞\ \rho[x \mapsto v_2] \vdash \langle e_1', s'' \rangle \Downarrow_V \langle v, s''' \rangle}{\rho \vdash \langle (e_1\ e_2), s \rangle \Downarrow_V \langle v, s''' \rangle}$$

Example: Let $\rho = \{ x \mapsto 7, y \mapsto 3 \}$ and consider

$$\begin{aligned} \rho \vdash \langle\ & \textbf{let}\ f = \lambda x.x + y \\ & \quad \textbf{in let}\ g = \lambda y.f(y + 100) \\ & \qquad \textbf{in}\ ((f\ 10) + (g\ 0)), s \rangle \Downarrow_V ?? \end{aligned}$$

Dynamic Scoping, 2

$$\textit{Dynamic-App:}\quad \frac{\begin{array}{rcl} \rho \vdash \langle\, e_1, s \,\rangle & \Downarrow_V & \langle\, \lambda x.e_1', s' \,\rangle \\ \rho \vdash \langle\, e_2, s' \,\rangle & \Downarrow_V & \langle\, v_2, s'' \,\rangle \\ ☞\ \rho[x \mapsto v_2] \vdash \langle\, e_1', s'' \,\rangle & \Downarrow_V & \langle\, v,, s''' \,\rangle \end{array}}{\rho \vdash \langle\, (e_1\ e_2), s \,\rangle \quad \Downarrow_V \quad \langle\, v, s''' \,\rangle}$$

Under dynamic scoping, when you apply a function in environment

$$((\lambda x.e_1')\ e_2) \quad \text{in environment } \rho$$

you evaluate e_1' in environment $\rho[x \mapsto v_2]$.

Question:

Is this a bug or a feature?

Dynamic Scoping, 3

$$\textit{Dynamic-App:}\ \frac{\begin{array}{rcl} \rho \vdash \langle\, e_1, s \,\rangle & \Downarrow_V & \langle\, \lambda x.e_1', s' \,\rangle \\ \rho \vdash \langle\, e_2, s' \,\rangle & \Downarrow_V & \langle\, v_2, s'' \,\rangle \\ \rho[x \mapsto v_2] \vdash \langle\, e_1', s'' \,\rangle & \Downarrow_V & \langle\, v, s''' \,\rangle \end{array}}{\begin{array}{rcl} \rho \vdash \langle\, (e_1\ e_2), s \,\rangle & \Downarrow_V & \langle\, (v, s''' \,\rangle \end{array}}$$

What goes *right* under dynamic scoping?

$$\begin{array}{l} \textbf{let}\ f = \lambda n.\ \textbf{if}\ n \leq 0\ \textbf{then}\ 1\ \textbf{else}\ n * (f\ (n-1)) \\ \quad \textbf{in}\ (f\ 3) \end{array}$$

History

Discovered and formalized in early (≈1960s) Lisp implementations.

Lexical Scoping, 1

Re: λ-expressions, functions, procedures, etc.,
there are two sorts of environments you have to worry about:

1. The environment in force when the function is *created*.
2. The environment in force when the function is *applied*.

- In human language, statements need to be understood in context:

  Such a fact is probable, but undoubtedly false.

  —Edward Gibbon in "Decline and Fall of the Roman Empire"
- When Gibbon was writing "probable" meant "well-recommended".
- So in reading Gibbon we have to use a 1700's English dictionary.
- We pull a similar trick for functions.

Lexical Scoping, 2

Definition

A closure, $e\rho$, is an expression e with an environment ρ such that $fv(e) \subseteq \text{domain}(\rho)$, i.e., all of e's free variables are in ρ's dictionary.

Ideas:

- A λ-expression evaluates to a closure.
- When we create a λ-expression, we "close" it with its definition-time environment.

$$\textit{Lexical-Fun:}\quad \frac{}{\rho \vdash \langle\, \lambda x.e, s \,\rangle \Downarrow_{\mathsf{V}} \langle\, (\lambda x.e)\rho, s \,\rangle}$$

- When we apply a function (i.e., closure $(\lambda x.e')\rho'$), we evaluate e' in $\rho'[x \mapsto v]$, where v is the value of the argument.

$$\textit{Lexical-App:}\quad \frac{\begin{array}{rcl} \rho \vdash \langle\, e_1, s \,\rangle & \Downarrow_{\mathsf{V}} & \langle\, (\lambda x.e_1')\rho_1', s' \,\rangle \\ \rho \vdash \langle\, e_2, s' \,\rangle & \Downarrow_{\mathsf{V}} & \langle\, v_2, s'' \,\rangle \\ \rho_1'[x \mapsto v_2] \vdash \langle\, e_1', s'' \,\rangle & \Downarrow_{\mathsf{V}} & \langle\, v, s''' \,\rangle \end{array}}{\rho \vdash \langle\, (e_1\ e_2), s \,\rangle \Downarrow_{\mathsf{V}} \langle\, (v, s''' \,\rangle}$$

Lexical Scoping, 3

$$\textit{Lexical-Fun:}\ \frac{}{\rho \vdash \langle\, \lambda x.e, s \,\rangle \Downarrow_{\mathsf{V}} \langle\, \underbrace{(\lambda x.e)\rho}_{\text{a closure}}, s \,\rangle}$$

$$\textit{Lexical-App:}\ \frac{\begin{array}{rcl} \rho \vdash \langle\, e_1, s \,\rangle & \Downarrow_{\mathsf{V}} & \langle\, \overbrace{(\lambda x.e_1')\rho_1'}^{\text{a closure}}, s' \,\rangle \\ \rho \vdash \langle\, e_2, s' \,\rangle & \Downarrow_{\mathsf{V}} & \langle\, v_2, s'' \,\rangle \\ \rho_1'[x \mapsto v_2] \vdash \langle\, e_1', s'' \,\rangle & \Downarrow_{\mathsf{V}} & \langle\, v, s''' \,\rangle \end{array}}{\rho \vdash \langle\, (e_1\ e_2), s \,\rangle \Downarrow_{\mathsf{V}} \langle\, (v, s''' \,\rangle}$$

Examples/Exercises: Let $\rho = \{\, x \mapsto 7, y \mapsto 3 \,\}$.

- $\rho \vdash \langle\, \textbf{let}\ f = \lambda x.(x+y)\ \textbf{in}\ (f\ 10),\ s \,\rangle \Downarrow_{\mathsf{V}}$??
- $\rho \vdash \langle\, \textbf{let}\ f = \lambda x.(x+y)\ \textbf{in}\ \ (\textbf{let}\ y = 100\ \textbf{in}\ (f\ 10)),\ s \,\rangle \Downarrow_{\mathsf{V}}$??
- $\rho \vdash \langle\, \textbf{let}\ f = \lambda n.\ \textbf{if}\ n \le 0\ \textbf{then}\ 1\ \textbf{else}\ n * (f\ (n-1))\ \textbf{in}\ (f\ 3),\ s \,\rangle \Downarrow_{\mathsf{V}}$??

Puzzle 1

$$
\begin{aligned}
\rho_1 =& [a \mapsto 1,\ b \mapsto 2] \\
e_1 =& \textbf{let } q = \lambda a.(a+b) \textbf{ in} \\
& \quad \textbf{let } a = 5 * b \textbf{ in} \\
& \quad\quad \textbf{let } b = a * b \textbf{ in} \\
& \quad\quad\quad (q\ 100)
\end{aligned}
$$

What the value of e_1 in environment ρ_1 under call-by-value with

(a) lexical scoping?

(b) dynamic scoping?

Puzzle 1(a): Call-by-value, lexical scoping

$$\rho_1 = [a \mapsto 1,\ b \mapsto 2]$$

$$\begin{aligned} e_1 = \textbf{let } & q = \lambda a.(a+b) \textbf{ in} \\ & \textbf{let } a = 5 * b \textbf{ in} \\ & \quad \textbf{let } b = a * b \textbf{ in} \\ & \qquad (q\ 100) \end{aligned}$$

tag	Environment	Expression
ρ_1:	$\boxed{\begin{array}{c} a \mapsto 1 \\ b \mapsto 2 \end{array}}$	**let** $q = \dots$
ρ_2:	$\boxed{q \mapsto (\lambda a.(a+b))\rho_1} \to \rho_1$	**let** $a = \dots$
ρ_3:	$\boxed{a \mapsto 10} \to \rho_2$	**let** $b = \dots$
ρ_4:	$\boxed{b \mapsto 20} \to \rho_3$	$(q\ 100)$
ρ_5:	$\boxed{a \mapsto 100} \to \rho_1$	$(a+b)$

value of $e_1\rho_1$: 102

Puzzle 1(b): Call-by-value, dynamic scoping

$\rho_1 = [a \mapsto 1,\ b \mapsto 2]$

$$\begin{aligned} e_1 = &\ \textbf{let}\ q = \lambda a.(a+b)\ \textbf{in} \\ &\quad \textbf{let}\ a = 5 * b\ \textbf{in} \\ &\quad\quad \textbf{let}\ b = a * b\ \textbf{in} \\ &\quad\quad\quad (q\ 100) \end{aligned}$$

tag	Environment	Expression
ρ_1:	$\boxed{a \mapsto 1,\ b \mapsto 2}$	$\textbf{let}\ \ q = \ldots$
	$\uparrow$	
ρ_2:	$\boxed{q \mapsto (\lambda a.(a+b))}$	$\textbf{let}\ \ a = \ldots$
	$\uparrow$	
ρ_3:	$\boxed{a \mapsto 10}$	$\textbf{let}\ \ b = \ldots$
	$\uparrow$	
ρ_4:	$\boxed{b \mapsto 20}$	$(q\ 100)$
	$\uparrow$	
ρ_5:	$\boxed{a \mapsto 100}$	$(a+b)$

value of $e_1 \rho_1$: 120

Puzzle 2

$$
\begin{aligned}
\rho_1 &= [a \mapsto 1,\ b \mapsto 2] \\
e_2 &= \textbf{let } p = \lambda a.(a+b) \textbf{ in} \\
&\qquad\quad \textbf{let } q = \lambda b.(a+(p\ b)) \textbf{ in} \\
&\qquad\qquad \textbf{let } a = 10 \textbf{ in} \\
&\qquad\qquad\quad \textbf{let } b = 20 \\
&\qquad\qquad\qquad \textbf{in } (q\ 100)
\end{aligned}
$$

What is the value of e_2 in environment ρ_1 under call-by-value with

(a) lexical scoping?

(b) dynamic scoping?

Puzzle 2(a): Call-by-value, lexical scoping

$\rho_1 = [a \mapsto 1,\ b \mapsto 2]$

$e_2 = \textbf{let}\ p = \lambda a.(a+b)\ \textbf{in}$
$\quad \textbf{let}\ q = \lambda b.(a+(p\ b))\ \textbf{in}$
$\quad\quad \textbf{let}\ a = 10\ \textbf{in}$
$\quad\quad\quad \textbf{let}\ b = 20$
$\quad\quad\quad\quad \textbf{in}\ (q\ 100)$

tag	Environment	Expression
ρ_1:	$\boxed{\begin{array}{c} a \mapsto 1 \\ b \mapsto 2 \end{array}}$	**let** $p = \ldots$
	$\uparrow$	
ρ_2:	$\boxed{p \mapsto (\lambda a.(a+b))\rho_1}$	**let** $q = \ldots$
	$\uparrow$	
ρ_3:	$\boxed{q \mapsto (\lambda b.(a+(p\,b)))\rho_2}$	**let** $a = \ldots$
	$\uparrow$	
ρ_4:	$\boxed{a \mapsto 10}$	**let** $b = \ldots$
	$\uparrow$	
ρ_5:	$\boxed{b \mapsto 20}$	$(q\ 100)$
ρ_6:	$\boxed{b \mapsto 100} \rightarrow \rho_2$	$a + (p\ b)$
ρ_7:	$\boxed{a \mapsto 100} \rightarrow \rho_1$	$(a+b)$

value of $e_2\rho_1$: 1+(100+2) = 103

Puzzle 2(b): Call-by-value, dynamic scoping

$\rho_1 = [a \mapsto 1,\ b \mapsto 2]$

$e_2 = \textbf{let}\ p = \lambda a.(a + b)\ \textbf{in}$
$\quad \textbf{let}\ q = \lambda b.(a + (p\ b))\ \textbf{in}$
$\quad\quad \textbf{let}\ a = 10\ \textbf{in}$
$\quad\quad\quad \textbf{let}\ b = 20$
$\quad\quad\quad\quad \textbf{in}\ (q\ 100)$

tag	Environment	Expression
ρ_1:	$a \mapsto 1$, $b \mapsto 2$	$\textbf{let}\ p = \ldots$
ρ_2:	$p \mapsto (\lambda a.(a+b))$	$\textbf{let}\ q = \ldots$
ρ_3:	$q \mapsto (\lambda b.(a + (p\, b)))$	$\textbf{let}\ a = \ldots$
ρ_4:	$a \mapsto 10$	$\textbf{let}\ b = \ldots$
ρ_5:	$b \mapsto 20$	$(q\ 100)$
ρ_6:	$b \mapsto 100$	$a + (p\ b)$
ρ_7:	$a \mapsto 100$	$(a + b)$

value of $e_2 \rho_1$: 10+(100+100) = 210

Lexical Scoping, 4: Closures + States = Objects

Suppose $(new\ v)$ returns a fresh location initialize to v.

Warning: The following is tormented LFP; `return` is as in HW10.

$$\begin{array}{l}
\textbf{let}\ mkbox = \lambda x.(\textbf{let}\ bx = (new\ x)\ \textbf{in}\ (\lambda y.\{\ bx := !bx + y;\ \textbf{return}\ !bx\ \}));\\
\quad \textbf{in let}\ \ u = (mxbox\ 10);\\
\quad\quad \textbf{in let}\ \ v = (mxbox\ (100 + (u\ 5)))\\
\quad\quad\quad \textbf{in}\ \ ((u\ 0) + (v\ 0))
\end{array}$$

[Trace this thing]

In more familiar notation, *mkbox* is roughly:

$$\begin{array}{l}
\textbf{function}\ mxbox(x) = \{\ \ \textbf{var}\ bx = (new\ x);\\
\qquad\qquad\qquad\qquad\quad \textbf{return}\ (\textbf{function}\ foo(v)\\
\qquad\qquad\qquad\qquad\qquad\qquad \{\ bx := !bx + v;\ \textbf{return}\ !bx\ \});\ \}
\end{array}$$

In Java terms:
- box is a class
- *mkbox* is a box-constructor
- u and v are instance methods
- bx is an instance variable.

Lexical Scoping, 5: What about call-by-name?

Call by name

$$\textit{Subst-App-cbn:}\quad \frac{\langle\, E_1, s \,\rangle \Downarrow_{\mathsf{N}} \langle\, \lambda x.E_1', s' \,\rangle \qquad \langle\, E_1'[E_2/x], s' \,\rangle \Downarrow_{\mathsf{N}} \langle\, V, s'' \,\rangle}{\langle\, (E_1\ E_2), s \,\rangle \Downarrow_{\mathsf{N}} \langle\, V, s'' \,\rangle}$$

Question:

With environments, how do we simulate substituting the unevaluated E_2 for x in E_1' that call-by-name requires?

Answer:

Thunks $\equiv$ closures of arbitrary expressions, not just λ-expressions.

History of the term: `http://www.retrologic.com/jargon/T/thunk.html`

The Call-By-Name Version

$$\textit{Lexical-App:}\ \frac{\rho \vdash \langle\, e_1, s \,\rangle \Downarrow_{\mathsf{N}} \langle\, \overbrace{(\lambda x.e_1')\rho_1'}^{\text{a closure}}, s' \,\rangle \qquad \rho[x \mapsto \overbrace{e_2\rho}^{\text{thunk}}] \vdash \langle\, e_1', s' \,\rangle \Downarrow_{\mathsf{N}} \langle\, v, s'' \,\rangle}{\rho \vdash \langle\, (e_1\ e_2), s \,\rangle \quad \Downarrow_{\mathsf{N}} \quad \langle\, (v, s'' \,\rangle}$$

$$\textit{Var:}\ \frac{\rho' \vdash \langle\, e', s \,\rangle \Downarrow_{\mathsf{N}} \langle\, v', s' \,\rangle}{\rho \vdash \langle\, x, s \,\rangle \Downarrow_{\mathsf{N}} \langle\, v', s' \,\rangle}\ (e'\rho' = \text{lookup}(\rho, x))$$

Call-by-name/dynamic-scoping makes very little sense,
…but we are implementing it any way in Homework 10.

Puzzle 3

$$\begin{aligned} \rho_0 &= \emptyset \\ s_0 &= [\,\ell \mapsto 0] \\ e_0 &= \textbf{let}\; g = \lambda x.\{\, \ell := !\ell + 1;\; \textbf{return}\;\; x \,\}; \\ &\qquad\quad \textbf{in let}\; z = (g\,100) \\ &\qquad\qquad \textbf{in}\; (z + z + z) \end{aligned}$$

Consider $\rho_0 \vdash (e_0, s_0) \Downarrow_{?} (v_1, s_1)$.

What are v_1 and s_1 we use lexical scoping and

(a) call-by-value evaluation?

(b) call-by-name evaluation?

Puzzle 3(a): Call-by-value

$$\rho_0 = \emptyset$$
$$s_0 = [\,\ell \mapsto 0]$$
$$e_0 = \textbf{let}\ g = \lambda x.\{\,\ell :=!\ell + 1;\ \textbf{return}\ \ x\,\};\ \textbf{in let}\ z = (g\,100)\ \textbf{in}\ (z + z + z)$$

What are v_1 and s_1 in

$$\rho_0 \vdash (e_0, s_0) \Downarrow_V (v_1, s_1)?$$

tag	Environment	Exp./State
ρ_0:	$\boxed{}$	$\textbf{let}\ \ g = \ldots$ $[\ell \mapsto 0]$
	$\uparrow$	
ρ_1 :	$\boxed{g \mapsto (\lambda x.\{\ldots\})\rho_0}$	$\textbf{let}\ \ z = (g\,100)$ $[\ell \mapsto 1]$
	$\uparrow$	
ρ_2 :	$\boxed{z \mapsto 100}$	$z + z + z$ $[\ell \mapsto 1]$

$$v_1 = 300$$
$$s_1 = [\ell \mapsto 1]$$

Puzzle 3(b): Call-by-name

$$\rho_0 = \emptyset$$
$$s_0 = [\,\ell \mapsto 0]$$
$$e_0 = \textbf{let}\ g = \lambda x.\{\,\ell := !\ell + 1;\ \textbf{return}\ \ x\,\};\ \textbf{in let}\ z = (g\,100)\ \textbf{in}\ (z + z + z)$$

What are v_1 and s_1 in

$$\rho_0 \vdash (e_0, s_0) \Downarrow_{\mathsf{N}} (v_1, s_1)?$$

tag	Environment	Exp./State
ρ_0:	$\boxed{}$	$\textbf{let}\ \ g = \ldots$ $[\ell \mapsto 0]$
	$\uparrow$	
ρ_1 :	$\boxed{g \mapsto (\lambda x.\{\ldots\})\rho_0}$	$\textbf{let}\ \ z = (g\,100)$ $[\ell \mapsto 0]$
	$\uparrow$	
ρ_2 :	$\boxed{z \mapsto (g\,100)\rho_1}$	$z + z + z$ $[\ell \mapsto 3]$

$$v_1 = 300$$
$$s_1 = [\ell \mapsto 3]$$

Puzzle 4

$$
\begin{aligned}
\rho_0 &= \emptyset \\
s_0 &= [\,\ell \mapsto 0] \\
e_0 &= \textbf{let } g = \lambda x.\{\,\ell := !\ell + 1;\ \textbf{return}\ \ x\,\}; \\
&\qquad\quad \textbf{in let } h = \lambda y.2; \\
&\qquad\qquad \textbf{in } (h\,(g\,89))
\end{aligned}
$$

Consider $\rho_0 \vdash (e_0, s_0) \Downarrow_{?} (v_1, s_1)$.

What are v_1 and s_1 we use lexical scoping and

(a) call-by-value evaluation?

(b) call-by-name evaluation?

Puzzle 4(a): Call-by-value

$$\rho_0 = \emptyset$$
$$s_0 = [\,\ell \mapsto 0]$$
$$e_0 = \textbf{let}\ g = \lambda x.\{\,\ell := !\ell + 1;\ \textbf{return}\ \ x\,\};\ \textbf{in let}\ h = \lambda y.2\ \textbf{in}\ (h\,(g\,89))$$

What are v_1 and s_1 in

$$\rho_0 \vdash (e_0, s_0) \Downarrow_{\mathsf{V}} (v_1, s_1)?$$

tag	Environment	Exp./State
ρ_0:	$\boxed{}$	**let** $g = \ldots$ $[\ell \mapsto 0]$
ρ_1 :	$\boxed{g \mapsto (\lambda x.\{\ldots\})\rho_0} \uparrow$	**let** $h = \ldots$ $[\ell \mapsto 0]$
ρ_2 :	$\boxed{h \mapsto (\lambda y.2)\rho_1} \uparrow$	$(h(g\,89))$ $[\ell \mapsto 0]$
ρ_3:	$\boxed{x \mapsto 89} \to \rho_0$	$\{\,\ell := !\ell + 1;$ **return** $x\}$ $[\ell \mapsto 1]$
ρ_4:	$\boxed{y \mapsto 89} \to \rho_1$	2 $[\ell \mapsto 1]$

$$v_1 = 2$$
$$s_1 = [\ell \mapsto 1]$$

Puzzle 4(b): Call-by-name

$\rho_0 = \emptyset$

$s_0 = [\,\ell \mapsto 0]$

$e_0 = \textbf{let}\ g = \lambda x.\{\,\ell := !\ell + 1;$
$\quad \textbf{return}\ \ x\,\};$
$\quad \textbf{in let}\ h = \lambda y.2$
$\quad \textbf{in}\ (h\,(g\,89))$

What are v_1 and s_1 in

$$\rho_0 \vdash (e_0, s_0) \Downarrow_V (v_1, s_1)?$$

tag	Environment	Exp./State
ρ_0:	$\boxed{}$	$\textbf{let}\ \ g = \ldots$ $[\ell \mapsto 0]$
	$\uparrow$	
ρ_1 :	$\boxed{g \mapsto (\lambda x.\{\ldots\})\rho_0}$	$\textbf{let}\ \ h = \ldots$
	$\uparrow$	$[\ell \mapsto 0]$
ρ_2 :	$\boxed{h \mapsto (\lambda y.2)\rho_1}$	$(h(g\ 89))$ $[\ell \mapsto 0]$
ρ_4:	$\boxed{y \mapsto (g\,89)\rho_2} \rightarrow \rho_1$	2 $[\ell \mapsto 0]$

$$v_1 = 2$$
$$s_1 = [\ell \mapsto 0]$$

Recursion under lexical scoping, 1

Recall:

$$E ::= \dots \mid \mathbf{rec}\ x.E$$

Informally: "$\mathbf{rec}\ x.E$" reads *recursively define x to be E.*

The big-step operational semantics is given by:

$$\textit{unfolding}_{subst}\!: \quad \frac{\langle\, E[(\mathbf{rec}\ x.E)/x], s \,\rangle \Downarrow \langle\, V, s' \,\rangle}{\langle\, \mathbf{rec}\ x.E, s \,\rangle \Downarrow \langle\, V, s' \,\rangle}$$

Recursion under lexical scoping, 2

The substitution-based version of unfold

$$\textit{unfolding}_{subst}\colon\ \frac{\langle\, E[(\mathbf{rec}\ x.E)/x], s\,\rangle \Downarrow \langle\, V, s'\,\rangle}{\langle\, \mathbf{rec}\ x.E, s\,\rangle \Downarrow \langle\, V, s'\,\rangle}$$

An environment-based version of unfold *(There are better ways!)*

$$\textit{unfolding}_{env}\colon\ \frac{\rho[x \mapsto (\mathbf{rec}\ x.E)] \vdash \langle\, E, s\,\rangle \Downarrow \langle\, V, s'\,\rangle}{\rho \vdash \langle\, \mathbf{rec}\ x.E, s\,\rangle \Downarrow \langle\, V, s'\,\rangle}$$

Try:

$$\vdash \langle\, \mathbf{rec}\ z.(\mathbf{if}\ !\ell > 0\ \mathbf{then}\ (\ell := !\ell - 1;\ z)\ \mathbf{else\ skip}), \{\, \ell \mapsto 2\,\}\,\rangle \Downarrow ??$$