### The Environment Model of Evaluation

Jim Royer  
CIS 352  
March 31, 2015

---

#### LFP = LC + \( \lambda \) + function application + variables

**LFP Expressions**

\[
E ::= n \mid b \mid \ell \mid E \text{ iop } E \mid E \text{ cop } E \mid \text{ if } E \text{ then } E \text{ else } E \\
| \text{ !E} \mid E := E \mid \text{ skip} \mid E; E \mid \text{ while } E \text{ do } E \\
| x \mid \lambda x. E \mid E E \mid \text{ let } x = E \text{ in } E
\]

where

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

We focus on the \((\lambda\text{-calculus} + \text{let})\) part of LFP.

---

#### Application via substitution and its problems

**Call by name**

\[
\llbracket-\text{cbn} : \langle E_1, s \rangle \llbracket \to \langle \lambda x. E'_1, s' \rangle \to \langle E'_1[E_2/x], s' \rangle \to \langle V, s'' \rangle \to \langle (E_1 E_2), s' \rangle \llbracket \to \langle V, s'' \rangle
\]

**Call by value**

\[
\llbracket-\text{cbv} : \langle E_1, s \rangle \llbracket \to \langle \lambda x. E'_1, s' \rangle \to \langle E_2, s' \rangle \to \langle V_2, s'' \rangle \to \langle E_1[V_2/x], s'' \rangle \to \langle V, s''' \rangle \to \langle (E_1 E_2), s' \rangle \llbracket \to \langle V, s''' \rangle
\]

- 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 and their values and when you need the value of \( x \), look it up.

---

### References

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

Consider an expression \( e = \text{if } z \text{ then } x \text{ else } y + 2 \).
- With environment \( \{ x \mapsto 3, y \mapsto 4, z \mapsto \text{tt} \} \), \( e \) evaluates to 3.
- With environment \( \{ x \mapsto 8, y \mapsto 5, z \mapsto \text{ff} \} \), \( e \) evaluates to 7.
- Etc.

\( \text{lookup}(\rho, x) \) returns the value (if any) of \( x \) in environment \( \rho \).

\( \text{update}(\rho, x, v) \) returns a new environment \( \rho[x \mapsto v] \) (\( \rho[x \mapsto v] \) is just like \( \rho \) except \( x \) has value \( v \)).

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

---

### 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 \( \rho \) and state \( s \) evaluates to value \( v \) and state \( s' \).

\[
\begin{align*}
\text{Var:} & \quad \rho \vdash \langle x, s \rangle \Downarrow_V \langle v, s \rangle \\
& \qquad (v = \text{lookup}(\rho, x)) \\
\text{Let:} & \quad \rho \vdash \langle e_1, s \rangle \Downarrow_V \langle v_1, s' \rangle \\
& \qquad \rho[x \mapsto v_1] \vdash \langle e_2, s' \rangle \Downarrow_V \langle v_2, s'' \rangle \\
& \quad \rho \vdash \langle \text{let } x = e_1 \text{ in } e_2, s \rangle \Downarrow_V \langle v_2, s'' \rangle
\end{align*}
\]

**Examples/Exercises:** Let \( \rho = \{ x \mapsto 7, y \mapsto 3 \} \).
- \( \rho \vdash \langle x + y, s \rangle \Downarrow_V ?? \)
- \( \rho \vdash \langle \text{let } x = 1 \text{ in } x + y, s \rangle \Downarrow_V ?? \)
- \( \rho \vdash \langle \text{let } x = 1 \text{ in } (\text{let } z = 11 \text{ in } x + y + z), s \rangle \Downarrow_V ?? \)

---

The Environment Model of Evaluation 7 / 30
Dynamic Scoping, 1

Re: \(\lambda\)-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.

\[
\begin{align*}
\rho \vdash (e_1, s) & \Downarrow \langle \lambda x. e'_1, s' \rangle \\
\rho \vdash (e_2, s') & \Downarrow \langle v_2, s'' \rangle \\
\end{align*}
\]

**Dynamic-App:** \[
\begin{array}{c}
\rho \vdash (e_1, e_2, s) \Downarrow \langle (\lambda x. e'_1) e_2, s'' \rangle \\
\end{array}
\]

**Example:** Let \(\rho = \{ x \mapsto 7, y \mapsto 3 \}\) and consider
\[
\begin{align*}
\rho \vdash \langle \text{let } f = \lambda x. x + y \\
\text{in let } g = \lambda y.f(y + 100) \\
\text{in } ((f 10) + (g 0)), s \Downarrow ?? \\
\end{align*}
\]

**Question:** Is this a bug or a feature?

---

Dynamic Scoping, 2

\[
\begin{align*}
\rho \vdash (e_1, s) & \Downarrow \langle \lambda x. e'_1, s' \rangle \\
\rho \vdash (e_2, s') & \Downarrow \langle v_2, s'' \rangle \\
\end{align*}
\]

**Dynamic-App:** \[
\begin{array}{c}
\rho[x \mapsto v_2] \vdash \langle e'_1, s'' \rangle \Downarrow \langle v_2, s''' \rangle \\
\end{array}
\]

Under dynamic scoping, when you apply a function in environment

\((\lambda x. e'_1) e_2\) in environment \(\rho[x \mapsto v_2]\)

you evaluate \(e'_1\) in environment \(\rho[x \mapsto v_2]\).

---

Lexical Scoping, 1

Re: \(\lambda\)-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

**History**

Discovered and formalized in early (\(\approx 1960\)) Lisp implementations.
Lexical Scoping, 2

Definition

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

Ideas:

- A $\lambda$-expression evaluates to a closure.
- When we create a $\lambda$-expression, we “close” it with its definition-time environment.

$$\text{Lexical-Fun: } \rho \vdash (\lambda x.s, s) \Downarrow \langle (\lambda x.e')\rho', s \rangle$$

- 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.

$$\rho \vdash (e_1, s) \Downarrow \langle (\lambda x.e'_1')\rho_1', s' \rangle$$

$$\rho \vdash (e_2, s') \Downarrow \langle v_2, s'' \rangle$$

$$\text{Lexical-App: } \rho'[x \mapsto v_2] \vdash (e_1', s''') \Downarrow \langle v, s''' \rangle$$

$$\rho \vdash ((e_1, e_2), s) \Downarrow \langle (v, s'''') \rangle$$

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

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

Lexical Scoping, 3

$$\text{Lexical-Fun: } \rho \vdash (\lambda x.s, s) \Downarrow \langle (\lambda x.e)\rho, s \rangle$$

a closure

$$\rho \vdash (e_1, s) \Downarrow \langle (\lambda x.e'_1)\rho_1, s' \rangle$$

$$\rho \vdash (e_2, s') \Downarrow \langle v_2, s'' \rangle$$

$$\text{Lexical-App: } \rho'[x \mapsto v_2] \vdash (e_1', s''') \Downarrow \langle v, s''' \rangle$$

$$\rho \vdash ((e_1, e_2), s) \Downarrow \langle (v, s'''') \rangle$$

Puzzle 1

- The $e\rho$ notation is an effort to keep things from getting even-more heavy-handed.
- An alternative would be something like: $\text{close}(e, \rho)$.

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

$$e_1 = \text{let } q = \lambda a.(a + b) \text{ in } \text{let } a = 5 * b \text{ in } (q 100)$$

What is the value of $e_1$ in environment $\rho_1$ under call-by-value with (a) lexical scoping? (b) dynamic scoping?
### Puzzle 1(a): Call-by-value, lexical scoping

<table>
<thead>
<tr>
<th>Tag</th>
<th>Environment</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_1 )</td>
<td>[ a \mapsto 1, \ b \mapsto 2 ]</td>
<td>( \text{let } q = \ldots )</td>
</tr>
<tr>
<td>( e_1 )</td>
<td>( \text{let } a = 5 \times b ) in ( \text{let } b = a \times b ) in ( (q \ 100) )</td>
<td>( q \mapsto (\lambda a. (a + b)) ) ( \rho_1 )</td>
</tr>
</tbody>
</table>

Value of \( e_1 \rho_1 \): 102

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

<table>
<thead>
<tr>
<th>Tag</th>
<th>Environment</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_1 )</td>
<td>[ a \mapsto 1, \ b \mapsto 2 ]</td>
<td>( \text{let } q = \ldots )</td>
</tr>
<tr>
<td>( e_1 )</td>
<td>( \text{let } a = 5 \times b ) in ( \text{let } b = a \times b ) in ( (q \ 100) )</td>
<td>( q \mapsto (\lambda a. (a + b)) ) ( \rho_1 )</td>
</tr>
</tbody>
</table>

Value of \( e_1 \rho_1 \): 120

### Puzzle 2

\( \rho_1 = [a \mapsto 1, \ b \mapsto 2] \)

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

What is the value of \( e_2 \) in environment \( \rho_1 \) under call-by-value with
(a) lexical scoping?
(b) dynamic scoping?

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

<table>
<thead>
<tr>
<th>Tag</th>
<th>Environment</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_1 )</td>
<td>[ a \mapsto 1, \ b \mapsto 2 ]</td>
<td>( \text{let } p = \ldots )</td>
</tr>
<tr>
<td>( e_2 )</td>
<td>( \text{let } p = \lambda a. (a + b) ) in ( \text{let } q = \lambda b. (a + (p \ b)) ) in ( \text{let } a = 10 ) in ( \text{let } b = 20 ) in ( (q \ 100) )</td>
<td>( p \mapsto (\lambda a. (a + b)) ) ( \rho_1 )</td>
</tr>
</tbody>
</table>

Value of \( e_2 \rho_1 \): 103

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

<table>
<thead>
<tr>
<th>Tag</th>
<th>Environment</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_1 )</td>
<td>[ a \mapsto 1, \ b \mapsto 2 ]</td>
<td>( \text{let } q = \ldots )</td>
</tr>
<tr>
<td>( e_2 )</td>
<td>( \text{let } p = \lambda a. (a + b) ) in ( \text{let } q = \lambda b. (a + (p \ b)) ) in ( \text{let } a = 10 ) in ( \text{let } b = 20 ) in ( (q \ 100) )</td>
<td>( q \mapsto (\lambda b. (a + (p \ b))) ) ( \rho_2 )</td>
</tr>
</tbody>
</table>

Value of \( e_2 \rho_1 \): 103
**Lexical Scoping, 5: What about call-by-name?**

**Call by name**

$$\text{Subst-App-cbn: } \frac{\langle E_1, s \rangle \triangleright_{N} \langle \lambda x. E'_1, s' \rangle \langle E'_1[E_2/x], s' \rangle \triangleright_{N} \langle V, s'' \rangle}{\langle (E_1 E_2), s \rangle \triangleright_{N} \langle V, s'' \rangle}$$

**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 $\lambda$-expressions.


---

**Lexical Scoping, 6**

**The Call-By-Name Version**

$$\text{Lexical-App: } \rho \vdash \langle e_1, s \rangle \triangleright_{N} \langle \lambda x. E'_1, s' \rangle \rho[x \mapsto e_2] \triangleright_{N} \langle (e'_1, s') \triangleright_{N} \langle V, s'' \rangle \triangleright_{N} \langle (V, s''') \rangle$$

**Var:**

$$\rho \vdash \langle x, s \rangle \triangleright_{N} \langle s', s'' \rangle (e'_p = \text{lookup}(\rho, x))$$

Call-by-name/dynamic-scoping makes very little sense, … but we are implementing it any way in Homework 10.
Puzzle 3

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

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

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

What are \( v_1 \) and \( s_1 \) in
\[ \rho_0 \vdash (e_0, s_0) \Downarrow_V (v_1, s_1) \]

Puzzle 3(b): Call-by-name

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

What are \( v_1 \) and \( s_1 \) in
\[ \rho_0 \vdash (e_0, s_0) \Downarrow_N (v_1, s_1) \]

Puzzle 4

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

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?
The Environment Model of Evaluation

Puzzle 4(a): Call-by-value

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

What are \( v_1 \) and \( s_1 \) in

\[ \rho_0 \vdash (e_0, s_0) \Downarrow_v (v_1, s_1)? \]

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

Recursion under lexical scoping, 1

Recall:

\[ E ::= \ldots \mid \text{rec } x.E \]

Informally: “\text{rec } x.E” reads recursively define \( x \) to be \( E \).

The big-step operational semantics is given by:

\[ \text{unfolding}_{\text{sub}}: \]
\[ \begin{align*}
\langle E[\text{rec } x.E]/x, s \rangle & \Downarrow \langle V, s' \rangle \\
\langle \text{rec } x.E, s \rangle & \Downarrow \langle V, s' \rangle
\end{align*} \]

Puzzle 4(b): Call-by-name

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

What are \( v_1 \) and \( s_1 \) in

\[ \rho_0 \vdash (e_0, s_0) \Downarrow_v (v_1, s_1)? \]

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

Recursion under lexical scoping, 2

The substitution-based version of unfold

\[ \text{unfolding}_{\text{sub}}: \]
\[ \begin{align*}
\rho[\text{rec } x.E] \vdash \langle E, s \rangle \Downarrow \langle V, s' \rangle \\
\rho \vdash \langle \text{rec } x.E, s \rangle \Downarrow \langle V, s' \rangle
\end{align*} \]

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

\[ \text{unfolding}_{\text{env}}: \]
\[ \begin{align*}
\rho \vdash \langle \text{rec } x.E, s \rangle \Downarrow \langle V, s' \rangle
\end{align*} \]

Try:

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