Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Algebraic structures; Axiomatic semantics
Section and Practice Problems

Apr 3–6, 2018

---

1 Haskell

(a) Install the Haskell Platform, via `https://www.haskell.org/platform/`.

(b) Get familiar with Haskell. Take a look at `http://www.seas.harvard.edu/courses/cs152/2018sp/resources.html` for some links to tutorials.

In particular, get comfortable doing functional programming in Haskell. Write the factorial function. Write the append function for lists.

(c) Get comfortable using monads, and the bind syntax. Try doing the exercises at `https://wiki.haskell.org/All_About_Monads#Exercises` (which will require you to read the previous sections to understand `do` notation, and their previous examples).

(d) Also, look at the file `http://www.seas.harvard.edu/courses/cs152/2018sp/sections/haskell-examples.hs`, which includes some example Haskell code (that will likely be covered in Section).

2 Algebraic structures

(a) Show that the option type, with *map* defined as in the lecture notes (Lecture 18, Section 2.2) satisfy the functor laws.

Answer: *The functor laws are:*

$$\forall f \in A \to B, g \in B \to C.\ (map\ f) \fatsemi (map\ g) = map\ (f \fatsemi g) \qquad \textit{Distributivity}$$

$$map\ (\lambda a\!:\!A.\ a) = (\lambda a\!:\!T_A.\ a) \qquad \textit{Identity}$$

The definition of map for the option type is

$$map \equiv \lambda f\!:\!\tau_1 \to \tau_2.\ \lambda a\!:\!\tau_1\ \textbf{\textit{option}}.\ \textit{case}\ a\ \textit{of}\ \lambda x\!:\!\textbf{\textit{unit}}.\ \textit{none} | \lambda v\!:\!\tau_1.\ \textit{some}\ \ (f\ v)$$

*To show distributivity, we need to show that for all functions $f : \tau_1 \to \tau_2$ and $g : \tau_2 \to \tau_3$ we have that $\lambda x\!:\!\tau_1$ **option**. $(map\ g)\ ((map\ f)\ x)$ is equivalent to map $(\lambda x\!:\!\tau_1.\ g\ (f\ x))$*

*Recall that $\fatsemi$ indicates function composition. So the function $f \fatsemi g$ can can be expressed as $\lambda x\!:\!\tau_1.\ g\ (f\ x)$, and the function $(map\ f) \fatsemi (map\ g)$ can be expressed as $\lambda x\!:\!\tau_1$ **option**. $(map\ g)\ ((map\ f)\ x)$.*

$\lambda x{:}\tau_1$ ***option***. $(map\ g)\ ((map\ f)\ x)$
$= \lambda x{:}\tau_1$ ***option***. $(map\ g)\ ((\lambda a{:}\tau_1$ ***option***. *case* a *of* $\lambda y{:}$ ***unit***. *none*$|\lambda v{:}\tau_1$. *some* $(f\ v))\ x)$ *expand map* f
$= \lambda x{:}\tau_1$ ***option***. $(map\ g)$ (*case* x *of* $\lambda y{:}$ ***unit***. *none*$|\lambda v{:}\tau_1$. *some* $(f\ v))$ *by* β*-equivalence*
$= \lambda x{:}\tau_1$ ***option***. $(\lambda b{:}\tau_2$ ***option***. *case* b *of* $\lambda z{:}$ ***unit***. *none*$|\lambda w{:}\tau_2$. *some* $(g\ w))$
(*case* x *of* $\lambda y{:}$ ***unit***. *none*$|\lambda v{:}\tau_1$. *some* $(f\ v))$ *expand map* g
$= \lambda x{:}\tau_1$ ***option***. *let* $b =$ *case* x *of* $\lambda y{:}$ ***unit***. *none*$|\lambda v{:}\tau_1$. *some* $(f\ v)$ *in*
case b *of* $\lambda z{:}$ ***unit***. *none*$|\lambda w{:}\tau_2$. *some* $(g\ w)$ *rewrite as let expression*
$= \lambda x{:}\tau_1$ ***option***. *case* x *of* $\lambda y{:}$ ***unit***. *none*$|\lambda v{:}\tau_1$. *some* $(g\ (f\ v))$ *simplifying nested cases*
$= \lambda x{:}\tau_1$ ***option***. *case* x *of* $\lambda t{:}$ ***unit***. *none*$|\lambda v{:}\tau_1$. *some* $((\lambda y{:}\tau_1.\ g\ (f\ y))\ v)$
$= map\ (\lambda y{:}\tau_1.\ g\ (f\ y))$ *un-expanding map* $(\lambda y{:}\tau_1.\ g\ (f\ y))$

To show identity, we need to show that map $\lambda a{:}\tau.\ a$ *is equivalent to* $\lambda a{:}\tau$ ***option***. a.

map $\lambda a{:}\tau.\ a$
$= \lambda b{:}\tau$ ***option***. *case* b *of* $\lambda x{:}$ ***unit***. *none*$|\lambda v{:}\tau$. *some* $((\lambda a{:}\tau.\ a)\ v)$ *expand map* $(\lambda a{:}\tau.\ a)$
$= \lambda b{:}\tau$ ***option***. *case* b *of* $\lambda x{:}$ ***unit***. *none*$|\lambda v{:}\tau$. *some* v *by* β*-equivalence*
$= \lambda b{:}\tau$ ***option***. b

(b) Consider the list type, τ **list**. Define functions *return* and *bind* for the list monad that satisfy the monad laws. Check that they satisfy the monad laws.

Answer: *We define return and bind so that they represent a set of possible values that could be produced by a computation. That is, we use lists to represent the possible values of a nondeterministic computation. There are other ways to define return and bind on lists, for example, as a stream of results produced by a computation.*

Here return will take a value of type τ *and return a list that contains the value as its only element. bind will take a list of* τ*, take a function* f *from* τ *to lists of* τ'*, and apply* f *to every element of the list (using the function map, defined in class), to get a list of lists of* τ'*. We then use a utility function flatten to flatten the list of lists of* τ' *to a list of* τ'*. (In the definition of flatten,* a *acts as an accumulator.)*

return $\triangleq \lambda x{:}\tau.\ x :: [\,]$
bind $\triangleq \lambda xs{:}\tau$ ***list***. $\lambda f{:}\tau \to \tau'$ ***list***. *flatten* $(map\ f\ xs)$
flatten $\triangleq$ *let* $fl = \mu f{:}\tau'$ ***list*** $\to (\tau'$ ***list***$)$ ***list*** $\to \tau'$ ***list***.
$\lambda a{:}\tau'$ ***list***. $\lambda x{:}(\tau'$ ***list***$)$ ***list***. *if isempty?* x *then* a *else* f $(append\ a\ (head\ x))$ $(tail\ x)$ *in*
$fl\ [\,]$

3 Axiomatic semantics

(a) Consider the program

$$c \equiv \mathsf{bar} := \mathsf{foo}; \textbf{while}\ \mathsf{foo} > 0\ \textbf{do}\ (\mathsf{bar} := \mathsf{bar} + 1; \mathsf{foo} := \mathsf{foo} - 1).$$

Write a Hoare triple $\{P\}\ c\ \{Q\}$ that expresses that the final value of bar is two times the initial value of foo.

Answer:

$$\{v = \mathsf{foo}\}\ \mathsf{bar} := \mathsf{foo}; \textbf{while}\ \mathsf{foo} > 0\ \textbf{do}\ (\mathsf{bar} := \mathsf{bar} + 1; \mathsf{foo} := \mathsf{foo} - 1)\ \{\mathsf{bar} = 2 \times v\}$$

Note that v is a logical variable, and we are using it to provide a name for the initial value of foo*. Note also that the Hoare triple could have said more things about the program. For example, the post condition could have included that* foo *is equal to zero.*

(b) Prove the following Hoare triples. That is, using the inference rules from Section 1.3 of Lecture 19, find proof tree with the appropriate conclusions.

(i) $\vdash \{\mathsf{baz} = 25\}\ \mathsf{baz} := \mathsf{baz} + 17\ \{\mathsf{baz} = 42\}$

Answer:

$$\textsc{Cons}\ \frac{\vDash \mathsf{baz} = 25 \Rightarrow \mathsf{baz} + 17 = 42 \qquad \textsc{Asg.}\ \dfrac{}{\vdash \{\mathsf{baz} + 17 = 42\}\ \mathsf{baz} := \mathsf{baz} + 17\ \{\mathsf{baz} = 42\}}}{\vdash \{\mathsf{baz} = 25\}\ \mathsf{baz} := \mathsf{baz} + 17\ \{\mathsf{baz} = 42\}}$$

(ii) $\vdash \{\textbf{true}\}\ \mathsf{baz} := 22; \mathsf{quux} := 20\ \{\mathsf{baz} + \mathsf{quux} = 42\}$

Answer:

$$\textsc{Consq.}\ \frac{\vDash \textbf{true} \Rightarrow 22 + 20 = 42 \qquad \textsc{Seq.}\ \dfrac{\textsc{Asg.}\ \dfrac{}{\{22 + 20 = 42\}\ \mathsf{baz} := 22\ \{\mathsf{baz} + 20 = 42\}} \qquad \textsc{Asg.}\ \dfrac{}{\{\mathsf{baz} + 20 = 42\}\ \mathsf{quux} := 20\ \{\mathsf{baz} + \mathsf{quux} = 42\}}}{\{22 + 17 = 42\}\ \mathsf{baz} := 22; \mathsf{quux} := 20\ \{\mathsf{baz} + \mathsf{quux} = 42\}}}{\{\textbf{true}\}\ \mathsf{baz} := 22; \mathsf{quux} := 20\ \{\mathsf{baz} + \mathsf{quux} = 42\}}$$

(iii) $\vdash \{\mathsf{baz} + \mathsf{quux} = 42\}\ \mathsf{baz} := \mathsf{baz} - 5; \mathsf{quux} := \mathsf{quux} + 5\ \{\mathsf{baz} + \mathsf{quux} = 42\}$

Answer: *Let* $c \equiv \mathsf{baz} := \mathsf{baz} - 5; \mathsf{quux} := \mathsf{quux} + 5$.

$$\textsc{Consq.}\ \frac{\vDash \mathsf{baz} + \mathsf{quux} = 42 \Rightarrow \mathsf{baz} - 5 + \mathsf{quux} + 5 = 42 \qquad \dfrac{\vdots}{\vdash \{\mathsf{baz} - 5 + \mathsf{quux} + 5 = 42\}\ c\ \{\mathsf{baz} + \mathsf{quux} = 42\}}}{\vdash \{\mathsf{baz} + \mathsf{quux} = 42\}\ c\ \{\mathsf{baz} + \mathsf{quux} = 42\}}$$

where the elided tree is

$$\textsc{Seq}\ \frac{\textsc{Asg}\ \dfrac{}{\{\mathsf{baz} - 5 + \mathsf{quux} + 5 = 42\}\ \mathsf{baz} := \mathsf{baz} - 5\ \{\mathsf{baz} + \mathsf{quux} - 5 = 42\}} \qquad \textsc{Asg}\ \dfrac{}{\{\mathsf{baz} + \mathsf{quux} - 5 = 42\}\ \mathsf{quux} := \mathsf{quux} + 5\ \{\mathsf{baz} + \mathsf{quux} = 42\}}}{\vdash \{\mathsf{baz} - 5 + \mathsf{quux} + 5 = 42\}\ c\ \{\mathsf{baz} + \mathsf{quux} = 42\}}$$

(iv) $\vdash \{\textbf{true}\}\ \textbf{if}\ \mathsf{y} = 0\ \textbf{then}\ \mathsf{z} := 2\ \textbf{else}\ \mathsf{z} := \mathsf{y} \times \mathsf{y}\ \{\mathsf{z} > 0\}$

Answer:

$$\textsc{Cons}\ \frac{\vDash \textbf{true} \wedge \mathsf{y} = 0 \Rightarrow 2 > 0 \qquad \textsc{Asg}\ \dfrac{}{\{2 > 0\}\ \mathsf{z} := 2\ \{\mathsf{z} > 0\}} \qquad \vDash \mathsf{z} > 0 \Rightarrow \mathsf{z} > 0}{\vdash \{\textbf{true} \wedge \mathsf{y} = 0\}\ \mathsf{z} := \mathsf{y} \times \mathsf{y}\ \{\mathsf{z} > 0\}}$$

Where here the assertion $\textbf{true} \wedge \mathsf{y} = 0 \Rightarrow 2 > 0$ *is always valid because* $\vDash 2 > 0$.

$$\textsc{Cons}\,\frac{\vDash \textbf{true} \wedge \neg(\mathsf{y}=0) \Rightarrow \mathsf{y}\times\mathsf{y} > 0 \qquad \textsc{Asg}\,\dfrac{}{\{\mathsf{y}\times\mathsf{y} > 0\}\ \mathsf{z} := \mathsf{y}\times\mathsf{y}\ \{\mathsf{z} > 0\}} \qquad \vDash \mathsf{z} > 0 \Rightarrow \mathsf{z} > 0}{\vdash \{\textbf{true} \wedge \neg(\mathsf{y}=0)\}\ \mathsf{z} := \mathsf{y}\times\mathsf{y}\ \{\mathsf{z} > 0\}}$$

The assertion $\textbf{true} \wedge \neg(\mathsf{y} = 0) \Rightarrow \mathsf{y}\times\mathsf{y} > 0$ *is valid because either* $\vDash \mathsf{y}\times\mathsf{y} > 0$ *or* $\nvDash \textbf{true} \wedge \neg(\mathsf{y} = 0)$. *To see this we can simplify* $\nvDash \textbf{true} \wedge \neg(\mathsf{y} = 0)$ *to* $\nvDash \neg(\mathsf{y} = 0)$, *and then to* $\vDash \mathsf{y} = 0$. *And it is always the case that either* $\vDash \mathsf{y} = 0$ *or* $\vDash \mathsf{y}\times\mathsf{y} > 0$.
Combining the two above trees, we can get

$$\textsc{If}\,\frac{\textsc{Cons}\,\dfrac{\vdots}{\vdash \{\textbf{true} \wedge \mathsf{y} = 0\}\ \mathsf{z} := 2\ \{\mathsf{z} > 0\}} \qquad \textsc{Cons}\,\dfrac{\vdots}{\vdash \{\textbf{true} \wedge \neg(\mathsf{y}=0)\}\ \mathsf{z} := \mathsf{y}\times\mathsf{y}\ \{\mathsf{z} > 0\}}}{\vdash \{\textbf{true}\}\ \textbf{if}\ \mathsf{y} = 0\ \textbf{then}\ \mathsf{z} := 2\ \textbf{else}\ \mathsf{z} := \mathsf{y}\times\mathsf{y}\ \{\mathsf{z} > 0\}}$$

(v) $\vdash \{\textbf{true}\}\ \mathsf{y} := 10; \mathsf{z} := 0; \textbf{while}\ \mathsf{y} > 0\ \textbf{do}\ \mathsf{z} := \mathsf{z} + \mathsf{y}\ \{\mathsf{z} = 55\}$

Answer: *This is a "trick" question in that the loop never terminates. (This wasn't intentional; Prof Chong made a mistake when writing the question. But luckily the Hoare triple is still valid!)*
Let's consider the while loop. So the loop invariant we will use is $\mathsf{y} > 0$.

$$\textsc{While}\,\frac{\dfrac{\vdots}{\vdash \{\mathsf{y} > 0 \wedge \mathsf{y} > 0\}\ \mathsf{z} := \mathsf{z} + \mathsf{y}\ \{\mathsf{y} > 0\}}}{\vdash \{\mathsf{y} > 0\}\ \textbf{while}\ \mathsf{y} > 0\ \textbf{do}\ \mathsf{z} := \mathsf{z} + \mathsf{y}\ \{\mathsf{y} > 0 \wedge \mathsf{y} \leq 0\}}$$

Note that the post condition is $\mathsf{y} > 0 \wedge \mathsf{y} \leq 0$. *This is equivalent to* **false**! *And* **false** *implies anything. In particular, we have that* $\vDash \mathsf{y} > 0 \wedge \mathsf{y} \leq 0 \Longrightarrow \mathsf{z} = 55$.

(vi) $\vdash \{\textbf{true}\}\ \mathsf{y} := 10; \mathsf{z} := 0; \textbf{while}\ \mathsf{y} > 0\ \textbf{do}\ (\mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1)\ \{\mathsf{z} = 55\}$

Answer: *This is what the previous question was actually meant to be.... The loop invariant we will use is that* $\mathsf{y} \geq 0 \wedge \mathsf{z} = 10 + 9 + \cdots + (\mathsf{y} + 1)$ *which we can write as* $\mathsf{y} \geq 0 \wedge \mathsf{z} = \sum_{i=\mathsf{y}+1}^{10} i$.
Let's first of all prove that the loop invariant is established when the program enters the loop (we leave part of the proof tree elided, as an exercise for the reader):

$$\textsc{Cons.}\,\frac{\vDash \textbf{true} \Longrightarrow 10 = 10 \wedge 0 = 0 \qquad \dfrac{\vdots}{\vdash \{10 = 10 \wedge 0 = 0\}\ \mathsf{y} := 10; \mathsf{z} := 0;\ \{\mathsf{y} = 10 \wedge \mathsf{z} = 0\}} \qquad \vDash (\mathsf{y} = 10 \wedge \mathsf{z} = 0) \Longrightarrow \mathsf{y} \geq 0 \wedge \mathsf{z} = \textstyle\sum_{i=\mathsf{y}+1}^{10}$}{\vdash \{\textbf{true}\}\ \mathsf{y} := 10; \mathsf{z} := 0;\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = \textstyle\sum_{i=\mathsf{y}+1}^{10} i\}}$$

Now let's show that it is in fact a loop invariant. For brevity let $S \equiv \sum_{i=\mathsf{y}+1}^{10} i$ *and* $S' \equiv \sum_{i=\mathsf{y}-1+1}^{10} i$.

$$\textsc{While}\,\frac{\vDash \mathsf{y} \geq 0 \wedge \mathsf{z} = S \wedge \mathsf{y} > 0 \Longrightarrow \mathsf{y} - 1 \geq 0 \wedge \mathsf{z} + \mathsf{y} = S' \qquad \dfrac{\overset{1}{\vdots}}{\vdash \{\mathsf{y} - 1 \geq 0 \wedge \mathsf{z} + \mathsf{y} = S'\}\ \mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S\}} \qquad \vDash \mathsf{y} \geq 0 \wedge \mathsf{z} = S}{\vdash \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S \wedge \mathsf{y} > 0\}\ \mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S\}}}{\vdash \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S\}\ \textbf{while}\ \mathsf{y} > 0\ \textbf{do}\ (\mathsf{z} := \mathsf{z} + \mathsf{y}; \mathsf{y} := \mathsf{y} - 1)\ \{\mathsf{y} \geq 0 \wedge \mathsf{z} = S \wedge \mathsf{y} \leq 0\}}$$

where $\overset{1}{\vdots}$ *is the following derivation (where* $S \equiv \sum_{i=\mathsf{y}+1}^{10} i$ *and* $S' \equiv \sum_{i=\mathsf{y}-1+1}^{10} i$*):*

$$\textsc{Seq}\ \frac{\textsc{Asg}\ \dfrac{}{\vdash \{\mathsf{y}-1\geq 0\wedge \mathsf{z}+\mathsf{y}=S'\}\ \mathsf{z}:=\mathsf{z}+\mathsf{y}\ \{\mathsf{y}-1\geq 0\wedge \mathsf{z}=S'\}} \qquad \textsc{Asg.}\ \dfrac{}{\vdash \{\mathsf{y}-1\geq 0\wedge \mathsf{z}=S'\}\ \mathsf{y}:=\mathsf{y}-1\ \{\mathsf{y}\geq 0\wedge \mathsf{z}=S\}}}{\vdash \{\mathsf{y}-1\geq 0\wedge \mathsf{z}+\mathsf{y}=S'\}\ \mathsf{z}:=\mathsf{z}+\mathsf{y};\mathsf{y}:=\mathsf{y}-1\ \{\mathsf{y}\geq 0\wedge \mathsf{z}=S\}}$$

Finally, we can use the fact that $\vDash \mathsf{y}\geq 0\wedge \mathsf{z}=S\wedge \mathsf{y}\leq 0 \implies \mathsf{z}=55$ *to construct a proof of the desired triple (where* $c\equiv \mathsf{y}:=10;\mathsf{z}:=0;$ **while** $\mathsf{y}>0$ **do** $(\mathsf{z}:=\mathsf{z}+\mathsf{y};\mathsf{y}:=\mathsf{y}-1)$*):*

$$\textsc{Cons}\ \frac{\vDash \textbf{true}\implies \textbf{true} \qquad \textsc{Seq}\ \dfrac{\vdots}{\vdash \{\textbf{true}\}\ c\ \{\mathsf{y}\geq 0\wedge \mathsf{z}=S\wedge \mathsf{y}\leq 0\}} \qquad \vDash \mathsf{y}\geq 0\wedge \mathsf{z}=S\wedge \mathsf{y}\leq 0\implies \mathsf{z}=55}{\vdash \{\textbf{true}\}\ c\ \{\mathsf{z}=55\}}$$

4 Environment Semantics

For Homework 5, the monadic interpreter you will be using uses environment semantics, that is, the operational semantics of the language uses a map from variables to values instead of performing substitution. This is a quick primer on environment semantics.

An environment ρ maps variables to values. We define a large-step operational semantics for the lambda calculus using an environment semantics. A configuration is a pair $\langle e, \rho\rangle$ where expression e is the expression to compute and ρ is an environment. Intuitively, we will always ensure that any free variables in e are mapped to values by environment ρ.

The evaluation of functions deserves special mention. Configuration $\langle \lambda x.\, e, \rho\rangle$ is a function $\lambda x.\, e$, defined in environment ρ, and evaluates to the *closure* $(\lambda x.\, e, \rho)$. A closure consists of code along with values for all free variables that appear in the code.

The syntax for the language is given below. Note that closures are included as possible values and expressions, and that a function $\lambda x.\, e$ is *not* a value (since we use closures to represent the result of evaluating a function definition).

$$\begin{aligned} e &::= x \mid n \mid e_1 + e_2 \mid \lambda x.\, e \mid e_1\ e_2 \mid (\lambda x.\, e, \rho) \\ v &::= n \mid (\lambda x.\, e, \rho) \end{aligned}$$

Note than when we apply a function, we evaluate the function body using the environment from the closure (i.e., the lexical environment, ρ_{lex}), as opposed to the environment in use at the function application (the dynamic environment).

$$\frac{}{\langle x, \rho\rangle \Downarrow \rho(x)} \qquad \frac{}{\langle n, \rho\rangle \Downarrow n} \qquad \frac{\langle e_1, \rho\rangle \Downarrow n_1 \quad \langle e_2, \rho\rangle \Downarrow n_2}{\langle e_1 + e_2, \rho\rangle \Downarrow n}\, n = n_1 + n_2$$

$$\frac{}{\langle \lambda x.\, e, \rho\rangle \Downarrow (\lambda x.\, e, \rho)} \qquad \frac{\langle e_1, \rho\rangle \Downarrow (\lambda x.\, e, \rho_{lex}) \quad \langle e_2, \rho\rangle \Downarrow v_2 \quad \langle e, \rho_{lex}[x \mapsto v_2]\rangle \Downarrow v}{\langle e_1\ e_2, \rho\rangle \Downarrow v}$$

For convenience, we define a rule for let expressions.

$$\frac{\langle e_1, \rho\rangle \Downarrow v_1 \quad \langle e_2, \rho[x \mapsto v_1]\rangle \Downarrow v_2}{\langle \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2, \rho\rangle \Downarrow v_2}$$

(a) Evaluate the program $\mathsf{let}\ f = (\mathsf{let}\ a = 5\ \mathsf{in}\ \lambda x.\, a + x)\ \mathsf{in}\ f\ 6$. Note the closure that f is bound to.

Answer: *Here is a derivation of the program.*

$$\dfrac{\dfrac{\overline{\langle 5, \emptyset\rangle \Downarrow 5} \quad \overline{\langle \lambda x.\, a + x, [a \mapsto 5]\rangle \Downarrow (\lambda x.\, a + x, [a \mapsto 5])}}{\langle \textsf{let}\, a = 5\ \textsf{in}\ \lambda x.\, a + x), \emptyset\rangle \Downarrow (\lambda x.\, a + x, [a \mapsto 5])} \quad \dfrac{\vdots}{\langle f\ 6, [f \mapsto (\lambda x.\, a + x, [a \mapsto 5])]\rangle \Downarrow 11}}{\langle \textsf{let}\, f = (\textsf{let}\, a = 5\ \textsf{in}\ \lambda x.\, a + x)\ \textsf{in}\ f\ 6, \emptyset\rangle \Downarrow 11}$$

where the missing derivation is as follows (and where $\rho_0 = [f \mapsto (\lambda x.\, a + x, [a \mapsto 5])]$ *and* $\rho_1 = [a \mapsto 5, x \mapsto 6]$*)*

$$\dfrac{\dfrac{\overline{\langle f, \rho_0\rangle \Downarrow (\lambda x.\, a + x, [a \mapsto 5])} \quad \overline{\langle 6, \rho_0\rangle \Downarrow 6} \quad \dfrac{\overline{\langle a, \rho_1\rangle \Downarrow 5} \quad \overline{\langle x, \rho_1\rangle \Downarrow 6}}{\langle a + x, \rho_1\rangle \Downarrow 11}}{\langle f\ 6, \rho_0\rangle \Downarrow 11}$$

Note that f *is bound to the closure* $(\lambda x.\, a + x, [a \mapsto 5])$*. That is. the function* $\lambda x.\, a + x$ *has a lexical environment* $[a \mapsto 5]$*: when the function was defined, the variable* a *was bound to* 5*. Note that when the function is used (*$f\ 6$*), the environment does not bind* a *at all.*

(b) Suppose we replaced the rule for application with the following rule:

$$\dfrac{\langle e_1, \rho\rangle \Downarrow (\lambda x.\, e, \rho_{lex}) \quad \langle e_2, \rho\rangle \Downarrow v_2 \quad \langle e, \rho[x \mapsto v_2]\rangle \Downarrow v}{\langle e_1\ e_2, \rho\rangle \Downarrow v}$$

That is, we use the dynamic environment to evaluate the function body instead of the lexical environment.

What would happen if you evaluated the program $\textsf{let}\ f = (\textsf{let}\ a = 5\ \textsf{in}\ \lambda x.\, a + x)\ \textsf{in}\ f\ 6$ with this modified semantics?

Answer: *As noted in the answer to the previous question,* f *is bound to the closure* $(\lambda x.\, a + x, [a \mapsto 5])$*, i.e., the lexical environment for the function* $\lambda x.\, a + x$ *is* $[a \mapsto 5]$*: when the function was defined, the variable* a *was bound to* 5*. When the function is used (*$f\ 6$*), the dynamic environment does not bind* a *at all. So that means that evaluation of* $\lambda x.\, a + x$ *will get stuck. In particular, it will try to evaluate expression* $a + x$ *in environment* $[f \mapsto (\lambda x.\, a + x, [a \mapsto 5]), x \mapsto 6]$ *(that is, the dynamic environment at the call site extended with* x *mapping to* 6*), and so won't be able to evaluate the variable* a*.*