1. Functions and Environments

To develop our approximation of local variables, we needed to move from a substitution-based view of evaluation to an environment-based view. We’ll have to do something similar for functions. So, let’s get started!

- Omitted rules for numeric constants, because they don’t behave any different than they did in the last version
- Again, reusing syntax for 2-place and 3-place evaluation relations

We should confirm that it works. Let’s try some simple reductions:

```
\emptyset \vdash \lambda a. \lambda b. b \Downarrow \lambda a. \lambda b. b
\emptyset \vdash 3 \Downarrow 3
\{ a \mapsto 3 \} \vdash \lambda b. b \Downarrow \lambda b. b
\emptyset \vdash (\lambda a. \lambda b. b) 3 \Downarrow \lambda b. b
\emptyset \vdash 2 \Downarrow 2
\{ b \mapsto 2 \} \vdash b \Downarrow 2
\emptyset \vdash (\lambda a. \lambda b. b) 3 2 \Downarrow 2
```

Looks good so far!

```
\emptyset \vdash \lambda a. \lambda b. a \Downarrow \lambda a. \lambda b. a
\emptyset \vdash 3 \Downarrow 3
\{ a \mapsto 3 \} \vdash \lambda b. a \Downarrow \lambda b. a
\emptyset \vdash (\lambda a. \lambda b. a) 3 \Downarrow \lambda b. a
\emptyset \vdash 2 \Downarrow 2
\{ b \mapsto 2 \} \vdash a \Downarrow 3
\emptyset \vdash (\lambda a. \lambda b. a) 3 2 \Downarrow 3
```

What’s gone wrong?

- We’re trying to use variable $a$ when it’s not apparently in scope. Fair enough—this shouldn’t be derivable.
- Variable $a$ should have gotten its meaning in reducing the left-hand argument, but it didn’t. This is the real problem.
- Missing one aspect of substitution—although evaluation doesn’t touch $\lambda$s, substitution does!

Solution: $\lambda$ terms need to carry their defining environments with them!

- Means we don’t have to reintroduce substitution
- Combination of a function and its environment called a closure.

2. Closures

Let’s recap our language:
3. Typing Functions

\[ X \ni x \]
\[ \mathcal{V} \ni v ::= z \mid \lambda H \cdot t \]
\[ \mathcal{E} \ni t ::= z \mid t_1 \odot t_2 \mid x \mid \lambda x.t \mid t_1 t_2 \]

- New value form: closures. Package environment with function
- Values no longer subset of terms... but can think of \( \lambda H \cdot t \) as being syntax for \( (\lambda x.t)[v_i/y_i] \) where \( H = \{ y_i \mapsto v_i \} \).

Now we can adjust evaluation rules to construct and use closures.

\[
\begin{array}{llll}
H \vdash \lambda x.t \downarrow \lambda H \cdot t & H \vdash t_1 \downarrow \lambda H' \cdot t & H \vdash t_2 \downarrow w & H'[x \mapsto w] \vdash t \downarrow v \\
\end{array}
\]

Does this work?

\[
\begin{array}{llll}
\emptyset \vdash \lambda a.\lambda b. b \downarrow \lambda a.\lambda b. b & \emptyset \vdash 3 \downarrow 3 & \{ a \mapsto 3 \} \vdash \lambda b.b \downarrow \lambda (a \mapsto 3) b.b & \{ a \mapsto 3 \} \vdash \lambda b.b \downarrow \lambda (a \mapsto 3) b.b \\
\emptyset \vdash (\lambda a.\lambda b.b) 3 \downarrow \lambda (a \mapsto 3) b.b & \\ 
\emptyset \vdash \lambda a.\lambda b.b \downarrow \lambda (a \mapsto 3) b.b & \emptyset \vdash 2 \downarrow 2 & \{ a \mapsto 3, b \mapsto 2 \} \vdash b \downarrow 2 \\
\emptyset \vdash \lambda (a.\lambda b.b) 3 2 \downarrow 2 & \\
\emptyset \vdash \lambda (a.\lambda b.b) 3 2 \downarrow 2 & \\
\emptyset \vdash \lambda (a.\lambda b.b) 3 2 \downarrow 3 & \emptyset \vdash \lambda a.\lambda b.b \downarrow \lambda (a \mapsto 3) b.b & \emptyset \vdash 2 \downarrow 2 & \{ a \mapsto 3, b \mapsto 2 \} \vdash a \downarrow 3 \\
\emptyset \vdash \lambda (a.\lambda b.b) 3 2 \downarrow 3 & \\
\emptyset \vdash \lambda (a.\lambda b.b) 3 2 \downarrow 3 & \\
\emptyset \vdash \lambda a.\lambda b.b \downarrow \lambda (a \mapsto 3) b.b & \emptyset \vdash 2 \downarrow 2 & \{ a \mapsto 3, b \mapsto 2 \} \vdash a \downarrow 3 \\
\emptyset \vdash \lambda (a.\lambda b.b) 3 2 \downarrow 3 & \\
\end{array}
\]

Looks promising.

Seems to work!

Call by name variation: just replace \( H \in X \rightarrow \mathcal{V} \) with \( H \in X \rightarrow \mathcal{E} \) and:

\[
\begin{array}{llll}
H \vdash_{\text{cbn}} H(x) \downarrow v & H \vdash_{\text{cbn}} t \downarrow t_1 \lambda H' \cdot x.t & H'[x \mapsto t_2] \vdash_{\text{cbn}} t \downarrow v & \\
\end{array}
\]

\textit{Historical note.} Early implementations of LISP, including some still in use (ELISP), got closures wrong. Some people like to present this as a design choice; they call it “dynamic scope” or similar euphemisms. This is not a design choice, any more than \( 2 + 2 = 5 \) would be a design choice for addition. It is a system that fails to match the semantics of the \( \lambda \)-calculus.

3. Typing Functions

What can go wrong? \( 12, (\lambda c.c) + 1 \).

We need to extend our grammar of types:

\[ T \ni T ::= \text{Int} \mid T_1 \rightarrow T_2 \]

- Why don’t closures need to be reflected in the types of functions?
As before, we define a variation of the evaluation relation that characterizes the types of values:

\[ \Gamma \vdash t : T. \]

- Syntax: \( \vdash \) denotes consequence—under the assumptions in \( \Gamma \), the typing on the right holds.
- \( \Gamma : \mathcal{X} \rightarrow \mathcal{T} \) map from variables to their types.
- More about the typing relation... and the significance of our notational choices... to come.

**Typing rules:**

\[
\begin{align*}
\Gamma \vdash z : \text{Int} \quad &\quad \Gamma \vdash t_1 : \text{Int} \quad \Gamma \vdash t_2 : \text{Int} \\
\Gamma \vdash t_1 + t_2 : \text{Int} \\
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash x : \Gamma(x) \\
\Gamma[x \mapsto T_1] \vdash t : T_2 \\
\Gamma \vdash t_1 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_2 : T_1 \\
\Gamma \vdash t_1 t_2 : T_2
\end{align*}
\]

- Common notation for \( \Gamma[x \mapsto T_1] \) is \( \Gamma, x : T_1 \). May fall into this later, but not yet.
- Why don’t we have to represent the closure in the application rule?

Let’s look at some simple derivations:

\[
\begin{align*}
\{a \mapsto \text{Int}, b \mapsto \text{Int} \rightarrow \text{Int} \} \vdash a : \text{Int} \\
\{a \mapsto \text{Int} \} \vdash \lambda b.a : (\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int} \\
\emptyset \vdash (\lambda a.\lambda b.a) : \text{Int} \rightarrow (\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int} \\
\emptyset \vdash 3 : \text{Int} \\
\{c \mapsto \text{Int} \} \vdash c : \text{Int} \\
\emptyset \vdash (\lambda a.\lambda b.a) 3 : (\text{Int} \rightarrow \text{Int}) \rightarrow \text{Int} \\
\emptyset \vdash \lambda c.c : \text{Int} \rightarrow \text{Int} \\
\emptyset \vdash (\lambda a.\lambda b.a) 3 (\lambda c.c) : \text{Int}
\end{align*}
\]

- Check typing of functions at construction, not at use. So: more structure under the typing of a \( \lambda \), but less at their uses.
- Same term may have more than one typing derivation: \( \lambda a.a \) (up to \( \alpha \)-equivalence) given both \( \text{Int} \rightarrow \text{Int} \) and \( (\text{Int} \rightarrow \text{Int}) \rightarrow (\text{Int} \rightarrow \text{Int}) \).