Administrative Notes

No class on Thursday.

We will end lecture at 3pm today, rather than 3:30pm.

Makeup lectures:

- 11am-12:30pm on Monday, February 26 and
- 11am-12:30pm on Monday, March 4

If you haven't picked up your textbook/notes, see Bob Untiedt in 776 Soda.

FL: A Functional Language

$$
\begin{array}{rcll}
E & \in & \text{Exp} & \\
L & \in & \text{Lit} & \\
I & \in & \text{Identifier} & \\
B & \in & \text{Boollit} = \{\texttt{\#t}, \texttt{\#f}\} & \\
N & \in & \text{Intlist} = \{..., -2, -1, 0, 1, 2, ...\} & \\
O & \in & \text{Primop} = \{\texttt{not}?, +, -, +, /, ...\} & \\
E & ::= & L & [Literal] \\
 & | & I & [VariableReference] \\
 & | & (\texttt{primop} O_{name}\ E^*_{arg}) & [PrimitiveAppl] \\
 & | & (\texttt{proc}\ I_{formal}\ E_{body}) & [Abstraction] \\
 & | & (\texttt{call} E_{operator} E_{operand}) & [Application] \\
 & | & (\texttt{if} E_{test} E_{consequent} E_{alternate}) & [Branch] \\
 & | & (\texttt{pair} E_{left} E_{right}) & [Pairing] \\
 & | & (\texttt{rec} I_{var} E_{body}) & [Recursion] \\
L & ::= & \texttt{\#u} & [UnitLiteral] \\
 & | & B & [BooleanLiteral] \\
 & | & N & [IntegerLiteral] \\
 & | & (\texttt{symbol} I) & [SymbolicLitera]
\end{array}
$$

Denotational Semantics of FLK

$$
\begin{array}{rclcl}
x & \in & Expressible & = & (Value + Error)_\perp \\
v & \in & Value & = & Unit + Bool + Int + Sym + Pair + Procedure \\
 & & Unit & = & \{unit\} \\
i & \in & Int & = & \{..., -2, -1, 0, 1, 2, ...\} \\
b & \in & Bool & = & \{true, false\} \\
y & \in & Sym & = & \text{Identifier} \\
a & \in & Pair & = & Expressible \times Expressible \\
p & \in & Procedure & = & Expressible \to Expressible \\
 & & Error & = & Unit \\
e & \in & Environment & = & Identifier \to Expressible
\end{array}
$$

What can we tell from these domain equations (using the domain names to give some intuition about how these domains will be used)?

- Procedures are expressible values. They can be passed as argument to other Procedures (since they take Expressible values), stored in pairs, etc.
- Procedures are, at least potentially, lazy (non-strict), since they are represented a functions that may take in a bottom ($\perp$) value.

Helping Functions

We've seen these before in Postfix. The denotation semantics are bit less cumbersome if we use helping function on the domains.

$$error : Expressible = (Error \rightarrow Expressible\ \ unit)$$

$$with\text{-}value : Expressible \rightarrow (Value \rightarrow Expressible) \rightarrow\ Expressible$$
$$= \lambda xf.\ \textbf{matching}\ x$$
$$(Value \rightarrow Expressible\ v)\ ||\ (fv)$$

$$with\text{-}boolean : Expressible \rightarrow (Bool \rightarrow Expressible) \rightarrow\ Expressible$$

and others for *with-integer*, *with-procedure*, *with-pair*, *with-symbol*.

Helping Function on Environments

An environment is a mapping from identifiers to values. It is used to keep track of scopes. Note that it is not the same as the mapping from variables (locations) to values, which we will in an imperative language.

$$e \in Environment = Identifier \to Expressible$$

In FLK as defined, all values that can be expressed (including functions, bottom, etc.) can be bound to values. However, in general, we call the values that can be bound *denotable* values.

- *Denotable values:* values that can be named by identifiers in the language.
- *Expressible values:* values that can be the result of an expression evaluation in the language. The TGR book uses the word *Computable* to refer to these.

More "Abstract" Semantic Domains

In order to point out differences and similarities between different languages, we use some standard domains, as well as some that are language-specific.

$$
\begin{array}{rcll}
c & \in & Computation & = \text{to be filled in} \\
d & \in & Denotable & = \text{to be filled in} \\
v & \in & Value & = \text{to be filled in} \\
p & \in & Procedure & = Denotable \to Computation \\
\beta & \in & Binding & = (Denotable + Unbound)_\perp \\
e & \in & Environment & = Identifier \to Binding \\
 & & Unbound & = \{unbound\}
\end{array}
$$

The evaluation function is:

$$\mathcal{E} : \text{Exp} \to Environment \to Computation$$

Environment Operations

empty-env, bind, and extend-env

Strict vs. Lazy Evaluation

Lazy evaluation (call by name in TGR):

$d \in Denotable = Computation$

$$\mathcal{E}[\![I]\!] = \lambda e\ .\ (with\text{-}denotable(lookup e I)(\lambda d\ .\ d))$$

$$\begin{aligned}\mathcal{E}[\![(\texttt{call} E_1 E_2) = \lambda\ e.(\ & with\text{-}procedure(\mathcal{E}[\![E_1]\!]e)\\ & (\lambda p.(p(\mathcal{E}[\![E_2]\!]e))))\end{aligned}$$

Eager evaluation (call by value in TGR):

$d \in Denotable = Value$

$$\mathcal{E}[\![I]\!] = \lambda e\ .\ (with\text{-}denotable(lookup e I)(\lambda d\ .\ (val\text{-}to\text{-}comp d)))$$

$$\begin{aligned}\mathcal{E}[\![(\texttt{call} E_1 E_2) = \lambda\ e.(\ & with\text{-}procedure(\mathcal{E}[\![E_1]\!]e)\\ & (\lambda p.(with\text{-}value(\mathcal{E}[\![E_2]\!]e)p)))\end{aligned}$$

Pairs

Non-Strict Pairs

$a \in Pair = Value \times Value$

$\mathcal{E}[\![(\texttt{pair} E_1 E_2)]\!] =$

$\quad \lambda e\ .\ (val\text{-}to\text{-}comp(Pair \rightarrow Value\langle(\mathcal{E}[\![E_1]\!]e)(\mathcal{E}[\![E_2]\!]e)\rangle\ )))$