Specification and Rules for LANG

March 26, 2023

Syntax

$v, x \in \textsc{Name} \qquad n \in \mathbb{Z} \qquad b \in \{\mathsf{True}, \mathsf{False}\} \qquad c \in \{\mathsf{a}, \mathsf{b}, \ldots\} \qquad \bullet \in \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{\&\&}, \texttt{||}, \texttt{++}, \texttt{==}, \texttt{>}, \texttt{<}, \texttt{>=}, \texttt{<=}\}$

$$str \in \textsc{String} ::= \texttt{"}\ c^*\ \texttt{"}$$

$$list(t) \in \textsc{List}(t) ::= [\ t(,\ t)^*\ ] \mid [\,]$$

$$f \in \textsc{BuiltIn} ::= \mathsf{head} \mid \mathsf{tail}$$

$$pat \in \textsc{Pattern} ::= b \qquad :($$

$$
\begin{aligned}
e \in \textsc{Expr} ::=\ & v \mid n \mid b \\
\mid\ & \texttt{'}\ c\ \texttt{'} \\
\mid\ & str \\
\mid\ & list(e) \\
\mid\ & e \bullet e \\
\mid\ & \mathsf{fun}\ v\ \texttt{->}\ e \\
\mid\ & e\ e \\
\mid\ & \mathsf{let}\ v = e\ \mathsf{in}\ e \\
\mid\ & \mathsf{if}\ e\ \mathsf{then}\ e\ \mathsf{else}\ e \\
\mid\ & \mathsf{case}\ e\ \mathsf{of}\ \texttt{\{}\ \ (\ pat\ \texttt{->}\ e\ \texttt{;}\ )^*\ \texttt{\}} \\
\mid\ & f\ e_1\ \ldots\ e_n \qquad \text{where } \mathsf{arity}(f) = n
\end{aligned}
$$

$$val \in \textsc{Value} ::= n \mid b \mid c \mid str \mid list(val) \mid (\textbf{closure}\ x \to e, \rho)$$

$$\rho, \sigma \in \textsc{LocalEnv} \subseteq (\textsc{Name} \times \textsc{Value})^*$$

$$D \in \textsc{GlobalEnv} \subseteq (\textsc{Name} \times \textsc{Expr})^*$$

$$stm \in \textsc{Assign} ::= v = e$$

$$prog \in \textsc{Program} ::= (stm\ \texttt{\textbackslash n})^*$$

Denotational Semantics

$$
\begin{array}{lll}
\llbracket \cdot \rrbracket_\rho : \textsc{LocalEnv} \times \textsc{Expr} \to \textsc{Value} & & \\
\llbracket v \rrbracket_\rho & = v & v \in \textsc{Value} \\
\llbracket x \rrbracket_\rho & = \begin{cases} \mathsf{lookup}(x, \rho) & \text{if } x \in \mathsf{dom}(\rho) \\ \llbracket e_x \rrbracket_\varnothing & \text{if } x \in \mathsf{dom}(D) \\ \bot & \text{otherwise} \end{cases} & \text{where } e_x = \mathsf{lookup}(x, D) \\
\llbracket\ [e_1, \ldots, e_n]\ \rrbracket & = [\llbracket e_1 \rrbracket_\rho, \ldots, \llbracket e_n \rrbracket_\rho] & \\
\llbracket e_1 \bullet e_2 \rrbracket_\rho & = \mathsf{runBinOp}(\bullet, v_1, v_2) & \text{where } v_1 = \llbracket e_1 \rrbracket_\rho \text{ and } v_2 = \llbracket e_2 \rrbracket_\rho \\
\llbracket \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \rrbracket_\rho & = \begin{cases} \llbracket e_2 \rrbracket_\rho & \text{if } \llbracket e_1 \rrbracket_\rho = \mathsf{True} \\ \llbracket e_3 \rrbracket_\rho & \text{if } \llbracket e_1 \rrbracket_\rho = \mathsf{False} \\ \bot & \text{otherwise} \end{cases} & \\
\llbracket \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \rrbracket_\rho & = \llbracket e_2 \rrbracket_{\rho[x \mapsto v]} & \text{where } v = \llbracket e_1 \rrbracket_\rho \\
\llbracket \mathsf{fun}\ x\ \texttt{->}\ e \rrbracket_\rho & = (\textbf{closure}\ x \to e, \rho) & \\
\llbracket e_1\ e_2 \rrbracket_\rho & = \begin{cases} \llbracket e \rrbracket_{\sigma[x \mapsto v]} & \text{if } v_1 = (\textbf{closure}\ x \to e, \sigma) \\ \bot & \text{otherwise} \end{cases} & \text{where } v_1 = \llbracket e_1 \rrbracket_\rho \text{ and } v_2 = \llbracket e_2 \rrbracket_\rho \\
\llbracket f\ e_1\ \ldots\ e_n \rrbracket_\rho & = \begin{cases} \mathsf{runInternal}(f, v_1, \ldots, v_n) & \text{if } \mathsf{arity}(f) = n \\ \bot & \text{otherwise} \end{cases} & \text{where } v_i = \llbracket e_i \rrbracket_\rho \text{ and } f \in \textsc{BuiltIn}
\end{array}
$$

Operational Semantics

$$\textsc{EvalJ} ::= \textsc{GlobalEnv} \times \textsc{LocalEnv} \times \textsc{Expr} \times \textsc{Value}$$

$$D, \rho \vdash e \Rightarrow v \subseteq \textsc{EvalJ}$$

$$\frac{}{D, \rho \vdash v \Rightarrow v}\ \textsc{Lit}$$

$$\frac{(x, v) \in \rho}{D, \rho \vdash x \Rightarrow v}\ \textsc{LocalVar}$$

$$\frac{(x, e) \in D \quad D, \{\} \vdash e \Rightarrow v}{D, \rho \vdash x \Rightarrow v}\ \textsc{GlobalVar}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow v_1 \quad D, \rho \vdash e_2 \Rightarrow v_2 \quad [\![\bullet]\!](v_1, v_2) = v}{D, \rho \vdash e_1 \bullet e_2 \Rightarrow v}\ \textsc{Op}$$

$$\frac{}{D, \rho \vdash \mathsf{fun}\ x\ \texttt{->}\ e \Rightarrow (\textbf{closure}\ x \rightarrow e, \rho)}\ \textsc{Abs}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow (\textbf{closure}\ x \rightarrow e, \sigma) \quad D, \rho \vdash e_2 \Rightarrow v_2 \quad D, \sigma[x \mapsto v_2] \vdash e \Rightarrow v}{D, \rho \vdash e_1\ e_2 \Rightarrow v}\ \textsc{App}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow v_1 \quad D, \rho[x \mapsto v_1] \vdash e_2 \Rightarrow v}{D, \rho \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \Rightarrow v}\ \textsc{Let}$$

$$\frac{D, \rho \vdash e_i \Rightarrow v_i \quad [\![f]\!](v_1, \ldots, v_n) = v}{D, \rho \vdash f\ e_1\ \ldots\ e_n \Rightarrow v}\ \textsc{BuiltIn}$$

$$\frac{}{D, \rho \vdash [\,] \Rightarrow [\,]}\ \textsc{ListNil} \qquad \frac{D, \rho \vdash e_i \Rightarrow v_i}{D, \rho \vdash [e_1, \ldots, e_n] \Rightarrow [v_1, \ldots, v_n]}\ \textsc{List}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow \mathsf{True} \quad D, \rho \vdash e_2 \Rightarrow v_1}{D, \rho \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \Rightarrow v_1}\ \textsc{IfTrue} \qquad \frac{D, \rho \vdash e_1 \Rightarrow \mathsf{False} \quad D, \rho \vdash e_3 \Rightarrow v_2}{D, \rho \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \Rightarrow v_2}\ \textsc{IfFalse}$$

Figure 1: Operational Semantics for LANG.

Explanation Form

$$\textsc{NoEnvEvalJ} ::= \textsc{Expr} \times \textsc{Value}$$
$$e \Rightarrow v \subseteq \textsc{NoEnvEvalJ}$$

$$\Delta \in \text{X}\textsc{EvalJ} ::= \textsc{EvalJ} \times \textsc{NoEnvEvalJ}^*$$
$$\langle D, \rho \vdash e \Rightarrow v \mid \Delta \rangle$$

$$\frac{}{\langle D, \rho \vdash v \Rightarrow v \mid [\ ]\ \rangle}\ \text{X}\textsc{Lit}$$

$$\frac{}{\langle D, \rho \vdash x \Rightarrow v \mid [\ ]\ \rangle}\ \text{X}\textsc{Var}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow v_1 \quad D, \rho \vdash e_2 \Rightarrow v_2 \quad [\![\bullet]\!](v_1, v_2) = v \quad D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_2 \rightsquigarrow e_2'}{\langle D, \rho \vdash e_1 \bullet e_2 \Rightarrow v \mid e_1' \bullet e_2' \Rightarrow v \rangle}\ \text{X}\textsc{BinOp}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow v_1 \quad v_1 = (\textbf{closure}\ z \rightarrow e, \sigma) \quad D, \rho[x \mapsto v_1] \vdash e_2 \Rightarrow v_2 \quad D, \rho[x \mapsto v_1] : e_2 \rightsquigarrow e_2'}{\langle D, \rho \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \Rightarrow v \mid e_2' \Rightarrow v \rangle}\ \text{X}\textsc{LetFun}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow v_1 \quad D, \rho[x \mapsto v_1] \vdash e_2 \Rightarrow v \quad D, \rho[x \mapsto v_1] : e_2 \rightsquigarrow e_2' \quad D, \rho : e_1 \rightsquigarrow e_1'}{\langle D, \rho \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \Rightarrow v \mid e_1' \Rightarrow v_1, e_2' \Rightarrow v \rangle}\ \text{X}\textsc{Let}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow (\textbf{closure}\ x \rightarrow e, \sigma) \quad \langle D, \rho \vdash e_2 \Rightarrow v_2 \mid \Delta \rangle \quad D, \sigma[x \mapsto v_2] \vdash e \Rightarrow v \quad D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_2 \rightsquigarrow e_2'}{\langle D, \rho \vdash e_1\ e_2 \Rightarrow v \mid e_1'\ v_2 \Rightarrow v, e_2' \Rightarrow v_2, \Delta \rangle}\ \text{X}\textsc{App}$$

$$\frac{D, \rho \vdash e_i \Rightarrow v_i \quad D, \rho : e_i \rightsquigarrow e_i' \quad [\![f]\!](v_1, \ldots, v_n) = v}{\langle D, \rho \vdash f\ e_1\ \ldots\ e_n \Rightarrow v \mid f\ v_1\ \ldots\ v_n \Rightarrow v, e_i' \Rightarrow v_i \rangle}\ \text{X}\textsc{Builtin}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow \mathsf{True} \quad D, \rho \vdash e_2 \Rightarrow v \quad D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_2 \rightsquigarrow e_2'}{\langle D, \rho \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \Rightarrow v \mid e_1' \Rightarrow \mathsf{True}, e_2' \Rightarrow v \rangle}\ \text{X}\textsc{IfTrue}$$

$$\frac{D, \rho \vdash e_1 \Rightarrow \mathsf{False} \quad D, \rho \vdash e_3 \Rightarrow v \quad D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_3 \rightsquigarrow e_3'}{\langle D, \rho \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \Rightarrow v \mid e_1' \Rightarrow \mathsf{False}, e_3' \Rightarrow v \rangle}\ \text{X}\textsc{IfFalse}$$

Figure 2: Operational Semantics for LANG in Explanation Form

Explanatory Form for Expressions

The judgment $D, \rho : e \rightsquigarrow e'$ means that e, in the context of D and ρ, has the explanatory form e'.

$$\frac{(x, v) \in \rho \quad v \neq (\textbf{closure } y \to e, \sigma)}{D, \rho : x \rightsquigarrow v} \text{ EFVAL}$$

$$\frac{}{D, \rho : x \rightsquigarrow x} \text{ EFNAME}$$

$$\frac{D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_2 \rightsquigarrow e_2'}{D, \rho : e_1 \bullet e_2 \rightsquigarrow e_1' \bullet e_2'} \text{ EFOP}$$

$$\frac{D, \sigma : e \rightsquigarrow e' \quad \sigma = \rho \backslash x}{D, \rho : \mathsf{fun}\ x\ \texttt{->}\ e \rightsquigarrow \mathsf{fun}\ x\ \texttt{->}\ e'} \text{ EFLAM}$$

$$\frac{D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_2 \rightsquigarrow e_2'}{D, \rho : e_1\ e_2 \rightsquigarrow e_1'\ e_2'} \text{ EFAPP}$$

$$\frac{D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_2 \rightsquigarrow e_2'}{D, \rho : \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 \rightsquigarrow \mathsf{let}\ x = e_1'\ \mathsf{in}\ e_2} \text{ EFLET}$$

$$\frac{D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_2 \rightsquigarrow e_2' \quad D, \rho : e_3 \rightsquigarrow e_3'}{D, \rho : \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \rightsquigarrow \mathsf{if}\ e_1'\ \mathsf{then}\ e_2'\ \mathsf{else}\ e_3'} \text{ EFIF}$$

$$\frac{D, \rho : e_i \rightsquigarrow e_i'}{D, \rho : [e_1, \ldots, e_n] \rightsquigarrow [e_1', \ldots, e_n']} \text{ EFLIST}$$

Figure 3: Rules for the Explainitory Form for expressions

Proposals

Currently, we have the rule

$$\frac{D, \rho \vdash e_1 \Rightarrow (\textsf{\textbf{closure}}\ x \rightarrow e, \sigma) \quad \langle D, \rho \vdash e_2 \Rightarrow v_2 \mid \Delta \rangle \quad D, \sigma[x \mapsto v_2] \vdash e \Rightarrow v \quad D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_2 \rightsquigarrow e_2'}{\langle D, \rho \vdash e_1\ e_2 \Rightarrow v \mid e_1'\ v_2 \Rightarrow v, e_2' \Rightarrow v_2, \Delta \rangle} \text{XAPP}_1$$

which includes the explanation of the argument of an application into its own explanation. However, this rule leads to the judgment

$$\langle D, [] \vdash \textsf{length [4 + 5, 6 + 7]} \Rightarrow \textsf{2} \mid \textsf{length [9,13]} \Rightarrow \textsf{2}, \textsf{[4 + 5, 6 + 7]} \Rightarrow \textsf{[9,12]} \rangle$$

since the list expression currently yields an empty explanation. This isn't really an issue in some sense, but doesn't quite match our examples. But we also have the judgment

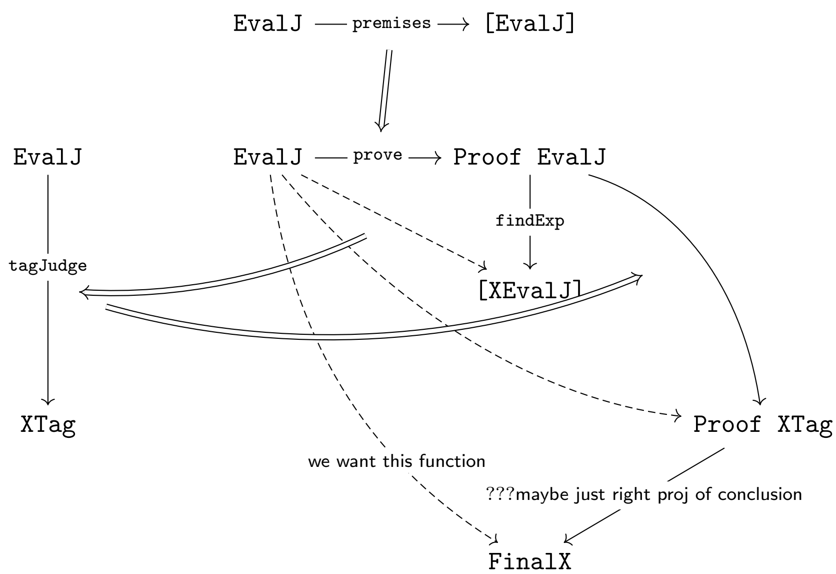
$$\langle D, [] \vdash \textsf{add (4 + 5) (6 + 7)} \Rightarrow \textsf{22} \mid \textsf{add (4 + 5) 13} \Rightarrow \textsf{22}, \textsf{6 + 7} \Rightarrow \textsf{13}, \textsf{6 + 7} \Rightarrow \textsf{13} \rangle$$

which is problmematic, as we would expect $\textsf{4 + 5} \Rightarrow \textsf{9}$ to be present if we are seeing $\textsf{6 + 7} \Rightarrow \textsf{13}$.

We should change this rule to include the explanation for the third premise,

$$\frac{D, \rho \vdash e_1 \Rightarrow (\textsf{\textbf{closure}}\ x \rightarrow e, \sigma) \quad \langle D, \rho \vdash e_2 \Rightarrow v_2 \mid \Delta \rangle \quad \langle D, \sigma[x \mapsto v_2] \vdash e \Rightarrow v \mid \Sigma \rangle \quad D, \rho : e_1 \rightsquigarrow e_1' \quad D, \rho : e_2 \rightsquigarrow e_2'}{\langle D, \rho \vdash e_1\ e_2 \Rightarrow v \mid e_1'\ v_2 \Rightarrow v, e_2' \Rightarrow v_2, \Delta, \Sigma \rangle} \text{XAPP}_2$$

which would include any explanation $D, \sigma[x \mapsto v_2] \vdash e \Rightarrow v$ might yield. Another thing to consider is to treat the explanation lists as sets, which removes the hassle of preventing duplicate judgments. It also gives us the sense that judgements should or should not be included in the final explanation and can be "thrown in".


EvalJ
premises
[EvalJ]
EvalJ
prove
Proof EvalJ
EvalJ
tagJudge
XTag
findExp
[XEvalJ]
Proof XTag
we want this function
???maybe just right proj of conclusion
FinalX