L1 Syntax and Documentation

Arthur Giesel Vedana

November 23, 2016

Contents

Introduction

The $L1$ programming language is a functional language with eager left-to-right evaluation. It has a simple I/O system supporting only direct string operations. It is a trait based strongly and statically typed language supporting both explicit and implicit typing.

This document both specifies the $L1$ language and shows its implementation in F#. It is divided into 4 categories:

1. Abstract Syntax and Semantics

   This defines the abstract syntax and semantics for the functional language. It only contains the bare minimum for the language to function, without any syntactic sugar.

2. Concrete Syntax

   This is the actual syntax when programming for $L1$. This defines all operators, syntactic sugar and other aspects of the language.

3. Implementation

   Technical aspects on how $L1$ is implemented in F#, showing the interpreter, evaluator and type inference.

4. Change log

   A chronological list of changes made both to the language definition and its implementation.

1 Abstract Syntax and Semantics

1.1 Abstract Syntax

Programs in $L1$ are terms e belonging to the following abstract syntax tree:

$$\begin{array}{lcl}
e & ::= & n \\
 & | & b \\
 & | & c \\
 & | & x \\
 & | & e_1\ op\ e_2 \\
 & | & if\ e_1\ then\ e_2\ else\ e_3 \\
 & | & e_1\ e_2 \\
 & | & fn\ x : T \Rightarrow e \\
 & | & fn\ x \Rightarrow e \\
 & | & rec\ x_1 : T_1 \rightarrow T_2\ \ x_2 : T_1 \Rightarrow e \\
 & | & rec\ x_1\ x_2 \Rightarrow e \\
 & | & let\ x : T = e_1\ in\ e_2 \\
 & | & let\ x = e_1\ in\ e_2 \\
 & | & nil \\
 & | & e_1 :: e_2 \\
 & | & isempty\ e \\
 & | & hd\ e \\
 & | & tl\ e \\
 & | & raise \\
 & | & try\ e_1\ with\ e_2 \\
 & | & skip \\
 & | & e_1\ ;\ e_2 \\
 & | & input \\
 & | & output\ e \\
 & & \\
T & ::= & Int \mid Bool \mid Char \mid Unit \\
 & | & T_1 \rightarrow T_2 \mid T\ list \\
 & & \\
x & ::= & \{x_0, x_1, \ldots\} \\
 & & \\
b & ::= & True \mid False \\
n & ::= & \mathbb{N} \\
c & ::= & `char' \\
char & ::= & ASCII\ characters \\
 & & \\
op & ::= & opNum \mid opEq \mid opIneq \mid opBool \\
opNum & ::= & + \mid - \mid * \mid \div \\
opEq & ::= & = \mid \neq \\
opIneq & ::= & < \mid \leq \mid > \mid \geq \\
boolOp & ::= & \wedge \mid \vee
\end{array}$$

1.2 Operational Semantics

The $L1$ language is evaluated using a big-step evaluation with environments. This evaluation reduces a term into a value directly, not necessarily having a rule of evaluation for every possible term. To stop programmers from creating programs that cannot be evaluated, a type inference system will be specified later.

Value A value is the result of the evaluation of a term in big-step. This set of values that is different from the set of terms of $L1$, even though they share many similarities.

Environment An environment is a mapping of identifiers to values that is extended each time a *let* declaration in encountered. Because the environment stores only values, this means that $L1$ has eager evaluation.

Below are the definitions of both values and environments:

$$
\begin{array}{lcl}
env & ::= & \{\} \mid \{x \rightarrow v\} \cup env \\
\\
v & ::= & n \\
 & \mid & b \\
 & \mid & c \\
 & \mid & nil \\
 & \mid & v_1 :: v_2 \\
 & \mid & raise \\
 & \mid & skip \\
 & \mid & \langle x, e, env \rangle \\
 & \mid & \langle x_1, x_2, e, env \rangle
\end{array}
$$

The values $\langle x, e, env\rangle$ and $\langle x_1, x_2, e, env\rangle$ are closures and recursive closures, respectively. They represent the result of evaluating functions and recursive functions, and store the environment at the moment of evaluation. This means that $L1$ has static scope, since closures capture the environment at the moment of evaluation and $L1$ has eager evaluation.

1.2.1 Big-Step Rules

$$\text{env} \vdash n \Downarrow n \qquad \text{(BS-NUM)}$$

$$\text{env} \vdash b \Downarrow b \qquad \text{(BS-BOOL)}$$

$$\text{env} \vdash c \Downarrow c \qquad \text{(BS-CHAR)}$$

$$\frac{\text{env}(x) = v}{\text{env} \vdash x \Downarrow v} \qquad \text{(BS-IDENT)}$$

Numerical Operations The $L1$ language only supports integers, so all operations are done on integer numbers. This means that the division always results in a whole number, truncated towards zero.

$$\frac{\text{env} \vdash e_1 \Downarrow n_1 \qquad \text{env} \vdash e_2 \Downarrow n_2 \qquad ||n|| = ||n_1|| + ||n_2||}{\text{env} \vdash e_1 + e_2 \Downarrow n} \quad \text{(BS-+)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow n_1 \qquad \text{env} \vdash e_2 \Downarrow n_2 \qquad ||n|| = ||n_1|| - ||n_2||}{\text{env} \vdash e_1 - e_2 \Downarrow n} \quad \text{(BS–)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow n_1 \qquad \text{env} \vdash e_2 \Downarrow n_2 \qquad ||n|| = ||n_1|| * ||n_2||}{\text{env} \vdash e_1 - e_2 \Downarrow n} \quad \text{(BS-}*\text{)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow n_1 \qquad \text{env} \vdash e_2 \Downarrow 0}{\text{env} \vdash e_1 \div e_2 \Downarrow raise} \quad \text{(BS-}\div\text{Zero)}$$

$$\frac{\begin{array}{cc}\text{env} \vdash e_1 \Downarrow n_1 & \text{env} \vdash e_2 \Downarrow n_2 \\ ||n_2|| \neq 0 & ||n|| = ||n_1|| \div ||n_2||\end{array}}{\text{env} \vdash e_1 \div e_2 \Downarrow n} \quad \text{(BS-}\div\text{)}$$

Equality Operations The equality operators (= and ≠) allow comparison of certain terms with other terms of the same kind. In this way, it is a polymorphic operator, being usable in different contexts. Even so, it is important to realize that it only compares values of the same kind (numbers with numbers, characters with characters, etc).

$$\frac{\text{env} \vdash e_1 \Downarrow n_1 \qquad \text{env} \vdash e_2 \Downarrow n_2 \qquad ||n_1|| = ||n_2||}{\text{env} \vdash e_1 = e_2 \Downarrow True} \quad \text{(BS-=NumTrue)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow n_1 \qquad \text{env} \vdash e_2 \Downarrow n_2 \qquad ||n_1|| \neq ||n_2||}{\text{env} \vdash e_1 = e_2 \Downarrow False} \quad \text{(BS-=NumFalse)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow c_1 \qquad \text{env} \vdash e_2 \Downarrow c_2 \qquad ||c_1|| = ||c_2||}{\text{env} \vdash e_1 ineqOPe_2 \Downarrow True} \quad \text{(BS-=CharTrue)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow c_1 \qquad \text{env} \vdash e_2 \Downarrow c_2 \qquad ||c_1|| \neq ||c_2||}{\text{env} \vdash e_1 ineqOPe_2 \Downarrow True} \quad \text{(BS-=CharFalse)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow b_1 \qquad \text{env} \vdash e_2 \Downarrow b_2 \qquad ||b_1|| = ||b_2||}{\text{env} \vdash e_1 = e_2 \Downarrow True} \quad \text{(BS-=BoolTrue)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow b_1 \qquad \text{env} \vdash e_2 \Downarrow b_2 \qquad \|b_1\| \neq \|b_2\|}{\text{env} \vdash e_1 = e_2 \Downarrow \mathit{False}} \quad (\text{BS-=}\textsc{BoolFalse})$$

$$\frac{\text{env} \vdash e_1 \Downarrow \mathit{nil} \qquad \text{env} \vdash e_2 \Downarrow \mathit{nil}}{\text{env} \vdash e_1 = e_2 \Downarrow \mathit{True}} \quad (\text{BS-=}\textsc{NilTrue})$$

$$\frac{\text{env} \vdash e_1 \Downarrow v_1 :: v_2 \qquad \text{env} \vdash e_2 \Downarrow \mathit{nil}}{\text{env} \vdash e_1 = e_2 \Downarrow \mathit{False}} \quad (\text{BS-=}\textsc{NilFalse1})$$

$$\frac{\text{env} \vdash e_1 \Downarrow \mathit{nil} \qquad \text{env} \vdash e_2 \Downarrow v_1 :: v_2}{\text{env} \vdash e_1 = e_2 \Downarrow \mathit{False}} \quad (\text{BS-=}\textsc{NilFalse2})$$

$$\frac{\begin{array}{c}\text{env} \vdash e_1 \Downarrow h_1 :: tl_1 \qquad \text{env} \vdash e_2 \Downarrow h_2 :: tl_2 \\ \text{env} \vdash h_1 = h_2 \Downarrow \mathit{False}\end{array}}{\text{env} \vdash e_1 = e_2 \Downarrow \mathit{False}} \quad (\text{BS-=}\textsc{ListFalse})$$

$$\frac{\begin{array}{c}\text{env} \vdash e_1 \Downarrow h_1 :: tl_1 \qquad \text{env} \vdash e_2 \Downarrow h_2 :: tl_2 \\ \text{env} \vdash h_1 = h_2 \Downarrow \mathit{True} \qquad \text{env} \vdash tl_1 = tl_2 \Downarrow b\end{array}}{\text{env} \vdash e_1 = e_2 \Downarrow b} \quad (\text{BS-=}\textsc{ListTrue})$$

$$\frac{\text{env} \vdash e_1 = e_2 \Downarrow \mathit{False}}{\text{env} \vdash e_1 \neq e_2 \Downarrow \mathit{True}} \quad (\text{BS-}{\neq}\textsc{True})$$

$$\frac{\text{env} \vdash e_1 = e_2 \Downarrow \mathit{True}}{\text{env} \vdash e_1 \neq e_2 \Downarrow \mathit{False}} \quad (\text{BS-}{\neq}\textsc{False})$$

Inequality Operations The inequality operators function much in the same way as the equality operators. The only difference is that they do not allow comparison of certain kinds of terms (such as booleans) when such terms do not have a clear ordering to them.

To reduce the number of rules, some rules are condensed for all inequality operators ($<, \leq, >, \geq$). The comparison done on numbers is the ordinary numerical comparison. For characters, the ASCII values are compared numerically.

$$\frac{\text{env} \vdash e_1 \Downarrow n_1 \qquad \text{env} \vdash e_2 \Downarrow n_2 \qquad \|n_1\|\ \mathit{opIneq}\ \|n_2\|}{\text{env} \vdash e_1\ \mathit{opIneq}\ e_2 \Downarrow \mathit{True}} \quad (\text{BS-}\textsc{IneqNumTrue})$$

$$\frac{\text{env} \vdash e_1 \Downarrow n_1 \qquad \text{env} \vdash e_2 \Downarrow n_2 \qquad \neg\, ||n_1||\, opIneq\, ||n_2||}{\text{env} \vdash e_1\ opIneq\ e_2 \Downarrow True}\ (\text{BS-IneqNumFalse})$$

$$\frac{\text{env} \vdash e_1 \Downarrow c_1 \qquad \text{env} \vdash e_2 \Downarrow c_2 \qquad ||c_1||\, opIneq\, ||c_2||}{\text{env} \vdash e_1\ opIneq\ e_2 \Downarrow True}\ (\text{BS-IneqCharTrue})$$

$$\frac{\text{env} \vdash e_1 \Downarrow c_1 \qquad \text{env} \vdash e_2 \Downarrow c_2 \qquad \neg\, ||c_1||\, opIneq\, ||c_2||}{\text{env} \vdash e_1\ opIneq\ e_2 \Downarrow True}\ (\text{BS-IneqCharFalse})$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil \qquad \text{env} \vdash e_2 \Downarrow nil}{\text{env} \vdash e_1 < e_2 \Downarrow False}\ (\text{BS-<Nil})$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil \qquad \text{env} \vdash e_2 \Downarrow nil}{\text{env} \vdash e_1 \leq e_2 \Downarrow True}\ (\text{BS-}\leq\text{Nil})$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil \qquad \text{env} \vdash e_2 \Downarrow nil}{\text{env} \vdash e_1 > e_2 \Downarrow False}\ (\text{BS->Nil})$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil \qquad \text{env} \vdash e_2 \Downarrow nil}{\text{env} \vdash e_1 \geq e_2 \Downarrow True}\ (\text{BS-}\geq\text{Nil})$$

$$\frac{\text{env} \vdash e_1 \Downarrow v_1 :: v_2 \qquad \text{env} \vdash e_2 \Downarrow nil}{\text{env} \vdash e_1 < e_2 \Downarrow False}\ (\text{BS-<ListNil})$$

$$\frac{\text{env} \vdash e_1 \Downarrow v_1 :: v_2 \qquad \text{env} \vdash e_2 \Downarrow nil}{\text{env} \vdash e_1 \leq e_2 \Downarrow False}\ (\text{BS-}\leq\text{ListNil})$$

$$\frac{\text{env} \vdash e_1 \Downarrow v_1 :: v_2 \qquad \text{env} \vdash e_2 \Downarrow nil}{\text{env} \vdash e_1 > e_2 \Downarrow True}\ (\text{BS->ListNil})$$

$$\frac{\text{env} \vdash e_1 \Downarrow v_1 :: v_2 \qquad \text{env} \vdash e_2 \Downarrow nil}{\text{env} \vdash e_1 \geq e_2 \Downarrow True}\ (\text{BS-}\geq\text{ListNil})$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil \qquad \text{env} \vdash e_2 \Downarrow v_1 :: v_2}{\text{env} \vdash e_1 < e_2 \Downarrow True}\ (\text{BS-<NilList})$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil \qquad \text{env} \vdash e_2 \Downarrow v_1 :: v_2}{\text{env} \vdash e_1 \leq e_2 \Downarrow True} \quad (\text{BS-}{\leq}\text{NilList})$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil \qquad \text{env} \vdash e_2 \Downarrow v_1 :: v_2}{\text{env} \vdash e_1 > e_2 \Downarrow False} \quad (\text{BS-}{>}\text{NilList})$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil \qquad \text{env} \vdash e_2 \Downarrow v_1 :: v_2}{\text{env} \vdash e_1 \geq e_2 \Downarrow False} \quad (\text{BS-}{\geq}\text{NilList})$$

$$\frac{\begin{array}{cc}\text{env} \vdash e_1 \Downarrow h_1 :: tl_1 & \text{env} \vdash e_2 \Downarrow h_2 :: tl_2 \\ \text{env} \vdash h_1 = h_2 \Downarrow False & \text{env} \vdash h_1 \ opIneq\ h_2 \Downarrow b\end{array}}{\text{env} \vdash e_1 \ opIneq\ e_2 \Downarrow b} \quad (\text{BS-IneqListHead})$$

$$\frac{\begin{array}{cc}\text{env} \vdash e_1 \Downarrow h_1 :: tl_1 & \text{env} \vdash e_2 \Downarrow h_2 :: tl_2 \\ \text{env} \vdash h_1 = h_2 \Downarrow True & \text{env} \vdash tl_1 \ opIneq\ tl_2 \Downarrow b\end{array}}{\text{env} \vdash e_1 \ opIneq\ e_2 \Downarrow b} \quad (\text{BS-IneqListTail})$$

Logical Operations The logical operators $\wedge$ (AND) and $\vee$ (OR) both have a short-circuit evaluation. This means that, if the result of the operation can be determined from the first operand, the second one is not evaluated.

$$\frac{\text{env} \vdash e_1 \Downarrow True}{\text{env} \vdash e_1 \vee e_2 \Downarrow True} \quad (\text{BS-}{\vee}\text{Short})$$

$$\frac{\text{env} \vdash e_1 \Downarrow False \qquad \text{env} \vdash e_2 \Downarrow b}{\text{env} \vdash e_1 \vee e_2 \Downarrow b} \quad (\text{BS-}{\vee})$$

$$\frac{\text{env} \vdash e_1 \Downarrow False}{\text{env} \vdash e_1 \wedge e_2 \Downarrow False} \quad (\text{BS-}{\wedge}\text{Short})$$

$$\frac{\text{env} \vdash e_1 \Downarrow True \qquad \text{env} \vdash e_2 \Downarrow b}{\text{env} \vdash e_1 \wedge e_2 \Downarrow b} \quad (\text{BS-}{\wedge})$$

$$\frac{\text{env} \vdash e_1 \Downarrow True \qquad \text{env} \vdash e_2 \Downarrow v}{\text{env} \vdash if\ e_1\ then\ e_2\ else\ e_3 \Downarrow v} \quad (\text{BS-IfTrue})$$

$$\frac{\text{env} \vdash e_1 \Downarrow False \qquad \text{env} \vdash e_3 \Downarrow v}{\text{env} \vdash if\ e_1\ then\ e_2\ else\ e_3 \Downarrow v} \quad (\text{BS-IfFalse})$$

$$\frac{\text{env} \vdash e_1 \Downarrow \langle x, e, env \rangle \qquad \text{env} \vdash e_2 \Downarrow v_2 \qquad \{x \rightarrow v_2\} \cup \text{env} \vdash e \Downarrow v}{\text{env} \vdash e_1\ e_2 \Downarrow v} \tag{BS-AppFn}$$

$$\frac{\text{env} \vdash e_1 \Downarrow \langle x_1, x_2, e, env \rangle \qquad \text{env} \vdash e_2 \Downarrow v_2 \qquad \{x_2 \rightarrow v_2, x_1 \rightarrow \langle x_1, x_2, e, env \rangle\} \cup \text{env} \vdash e \Downarrow v}{\text{env} \vdash e_1\ e_2 \Downarrow v} \tag{BS-AppRec}$$

$$\text{env} \vdash fn\ x : T \Rightarrow e \Downarrow \langle x, e, env \rangle \tag{BS-Fn}$$

$$\text{env} \vdash fn\ x \Rightarrow e \Downarrow \langle x, e, env \rangle \tag{BS-Fn2}$$

$$\text{env} \vdash rec\ x_1 : T_1 \rightarrow T_2\ \ x_2 : T_1 \Rightarrow e \Downarrow \langle x_1, x_2, e, env \rangle \tag{BS-Rec}$$

$$\text{env} \vdash rec\ x_1\ x_2 \Rightarrow e \Downarrow \langle x_1, x_2, e, env \rangle \tag{BS-Rec2}$$

$$\frac{\text{env} \vdash e_1 \Downarrow v \qquad \{x \rightarrow v\} \cup \text{env} \vdash e_2 \Downarrow v_2}{\text{env} \vdash let\ x : T = e_1\ in\ e_2 \Downarrow v_2} \tag{BS-Let}$$

$$\frac{\text{env} \vdash e_1 \Downarrow v \qquad \{x \rightarrow v\} \cup \text{env} \vdash e_2 \Downarrow v_2}{\text{env} \vdash let\ x = e_1\ in\ e_2 \Downarrow v_2} \tag{BS-Let2}$$

$$\text{env} \vdash nil \Downarrow nil \tag{BS-Nil}$$

$$\frac{\text{env} \vdash e_1 \Downarrow v \qquad \text{env} \vdash e_2 \Downarrow nil}{\text{env} \vdash e_1 :: e_2 \Downarrow v :: nil} \tag{BS-List}$$

$$\frac{\text{env} \vdash e_1 \Downarrow v_1 \qquad \text{env} \vdash e_2 \Downarrow h_2 :: tl_2}{\text{env} \vdash e_1 :: e_2 \Downarrow v_1 :: (h_2 :: tl_2)} \tag{BS-List2}$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil}{\text{env} \vdash isempty\ e_1 \Downarrow True} \tag{BS-EmptyTrue}$$

$$\frac{\text{env} \vdash e_1 \Downarrow h_1 :: tl_1}{\text{env} \vdash isempty\ e_1 \Downarrow False} \tag{BS-EmptyFalse}$$

$$\frac{\text{env} \vdash e_1 \Downarrow h_1 :: tl_1}{\text{env} \vdash hd\ e_1 \Downarrow h_1} \qquad \text{(BS-H}{\small\text{EAD}}\text{)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil}{\text{env} \vdash hd\ e_1 \Downarrow raise} \qquad \text{(BS-H}{\small\text{EAD}}\text{E}{\small\text{MPTY}}\text{)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow h_1 :: tl_1}{\text{env} \vdash tl\ e_1 \Downarrow tl_1} \qquad \text{(BS-T}{\small\text{AIL}}\text{)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow nil}{\text{env} \vdash tl\ e_1 \Downarrow raise} \qquad \text{(BS-T}{\small\text{AIL}}\text{E}{\small\text{MPTY}}\text{)}$$

Exceptions Some programs can be syntactically correct but still violate the semantics of the $L1$ language, such as a dividing by zero or trying to access the head of an empty list. In these scenarios, the term is evaluated as the *raise* value.

Besides violation of semantic rules, the only other term that evaluates to the *raise* value is the *raise* term, using the following rule:

$$\text{env} \vdash raise \Downarrow raise \qquad \text{(BS-R}{\small\text{AISE}}\text{)}$$

This value is propagated by (almost) all terms, climbing up the evaluation tree. This means that, for every evaluation rule, there is an alternative rule that, when a sub-term evaluates to raise, evaluates the whole term to raise. To avoid cluttering this document with the repetition of rules, these are not shown in their entirety. Below are a few examples of these propagation rules:

$$\frac{\text{env} \vdash e_1 \Downarrow raise}{\text{env} \vdash let\ x : T = e_1\ in\ e_2 \Downarrow raise} \qquad \text{(BS-L}{\small\text{ET}}\text{R}{\small\text{AISE}}\text{)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow n_1 \qquad \text{env} \vdash e_2 \Downarrow raise}{\text{env} \vdash e_1 + e_2 \Downarrow raise} \qquad \text{(BS-+R}{\small\text{AISE}}\text{)}$$

The only term that does not propagate the *raise* value is the *try* exception. Its evaluation rules are the following:

$$\frac{\text{env} \vdash e_1 \Downarrow raise \qquad \text{env} \vdash e_2 \Downarrow v}{\text{env} \vdash try\ e_1\ with\ e_2 \Downarrow v} \qquad \text{(BS-T}{\small\text{RY}}\text{R}{\small\text{AISE}}\text{)}$$

$$\frac{\text{env} \vdash e_1 \Downarrow v}{\text{env} \vdash try\ e_1\ with\ e_2 \Downarrow v} \qquad \text{(BS-T}{\small\text{RY}}\text{)}$$

Sequential Evaluation The term $e_1\ ;\ e_2$ is used when the term e_1 is only evaluated for its side effects, without any regard for its resulting value.

$$\frac{\text{env} \vdash e_1 \Downarrow \mathit{skip} \qquad \text{env} \vdash e_2 \Downarrow v}{\text{env} \vdash e_1\ ;\ e_2 \Downarrow v} \qquad \text{(BS-Sequential)}$$

The value *skip* can be obtained either by the term *skip*

$$\text{env} \vdash \mathit{skip} \Downarrow \mathit{skip} \qquad \text{(BS-Skip)}$$

or by evaluating other terms (such as *outpute*, as shown below).

Input and Output Since both input and output deal with side effects, specifying their evaluation rules is tricky.

The *output* term does not evaluate to any significant value, so we represent with the value *skip*.

$$\frac{\text{env} \vdash e \Downarrow \mathit{nil}}{\text{env} \vdash \mathit{output}\ e \Downarrow \mathit{skip}} \qquad \text{(BS-OutputEmpty)}$$

$$\frac{\text{env} \vdash e \Downarrow c :: v_1 \qquad \text{env} \vdash \mathit{output}\ v_1 \Downarrow \mathit{skip}}{\text{env} \vdash \mathit{output}\ e \Downarrow \mathit{skip}} \qquad \text{(BS-Output)}$$

This cannot be represented in the semantic rules, but the value c that results from evaluating e in rule **BS-Output** is printed on the output stream (typically the console). The rule **BS-OutputEmpty** prints a new line character on the output stream, effectively making *output* similar to "‘printLine"’ found in other languages.

The *input* term does evaluate to a significant value, but the value is nondeterministic, since it depends on the input of a user. The only guarantees that exist are that it will be either *nil* or $c :: v_2$, making this similar to "‘readLine"’ found in other languages.

$$\text{env} \vdash \mathit{input} \Downarrow v \qquad \text{(BS-Input)}$$