L1 Syntax and Documentation

Arthur Giesel Vedana

November 20, 2016

Contents

Introduction **ii**

1 Abstract Syntax and Semantics **1**
1.1 Abstract Syntax 1
1.2 Operational Semantics 2
1.2.1 Big-Step Rules 2

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:

$$
\begin{array}{lcl}
e & ::= & n \\
 & | & b \\
 & | & c \\
 & | & x \\
 & | & e_1 \ op \ e_2 \\
 & | & if \ e_1 \ then \ e_2 \ else \ e_3 \\
 & | & e_1 \ e_2 \\
 & | & fn \ x : T \Rightarrow e \\
 & | & fn \ x \Rightarrow e \\
 & | & rec \ x_1 : T_1 \rightarrow T_2 \ \ x_2 : T_1 \Rightarrow e \\
 & | & rec \ x_1 \ x_2 \Rightarrow e \\
 & | & let \ x : T = e_1 \ in \ e_2 \\
 & | & let \ x = e_1 \ in \ e_2 \\
 & | & nil \\
 & | & e_1 :: e_2 \\
 & | & isempty \ e \\
 & | & hd \ e \\
 & | & tl \ e \\
 & | & raise \\
 & | & try \ e_1 \ with \ e_2 \\
 & | & skip \\
 & | & e_1 \ ; \ e_2 \\
 & | & input \\
 & | & output \ e \\
 & & \\
T & ::= & Int \mid Bool \mid Char \mid Unit \\
 & | & T_1 \rightarrow T_2 \mid T \ list \\
 & & \\
x & ::= & \{x_0, x_1, \ldots\} \\
 & & \\
b & ::= & True \mid False \\
n & ::= & \mathbb{N} \\
c & ::= & `char' \\
char & ::= & \mathrm{a} \ldots \mathrm{z} \mid \mathrm{A} \ldots \mathrm{Z} \mid 0 \ldots 9 \\
 & & \\
op & ::= & opNum \mid opEq \mid opIneq \mid opBool \\
opNum & ::= & + \mid - \mid * \mid \div \\
opEq & ::= & = \mid \neq \\
opIneq & ::= & < \mid \leq \mid > \mid \geq \\
boolOp & ::= & \wedge \mid \vee
\end{array}
$$

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:

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

The values $\langle x, e, env \rangle$ and $\langle f, x, e, env \rangle$ 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

$$\text{env} \vdash n \Downarrow n \tag{BS-Num}$$

$$\text{env} \vdash b \Downarrow b \tag{BS-Bool}$$

$$\text{env} \vdash c \Downarrow c \tag{BS-Char}$$

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

Equality Operations Equality Operations

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

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

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

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

$$\frac{\text{env} \vdash e_1 \Downarrow \mathit{True} \qquad \text{env} \vdash e_2 \Downarrow \mathit{True}}{\text{env} \vdash e_1 = e_2 \Downarrow \mathit{True}} \quad (\text{BS-=}\textsc{BoolTrue}1)$$

$$\frac{\text{env} \vdash e_1 \Downarrow \mathit{False} \qquad \text{env} \vdash e_2 \Downarrow \mathit{False}}{\text{env} \vdash e_1 = e_2 \Downarrow \mathit{True}} \quad (\text{BS-=}\textsc{BoolTrue}2)$$

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

$$\frac{\text{env} \vdash e_1 \Downarrow \mathit{True} \qquad \text{env} \vdash e_2 \Downarrow \mathit{False}}{\text{env} \vdash e_1 \neq e_2 \Downarrow \mathit{True}} \quad (\text{BS-}{\neq}\textsc{BoolTrue}1)$$

$$\frac{\text{env} \vdash e_1 \Downarrow \mathit{False} \qquad \text{env} \vdash e_2 \Downarrow \mathit{True}}{\text{env} \vdash e_1 \neq e_2 \Downarrow \mathit{True}} \quad (\text{BS-}{\neq}\textsc{BoolTrue}2)$$

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

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.

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

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

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

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