LambdaScript Syntax and Semantics

Alex Kozik

July 15, 2023

Contents

1 Syntax

1.1 Metavariables

Below is a list of meta-variables for different fundamental langauge constructs

$x \in Var$	Variable indentifier
$b \in \{true, false\}$	Boolean
$n \in \mathbb{N}$	Natural number
$s \in \Sigma^*$	String
$\oplus \quad \in \quad \{+, -, *, /, \%, <, >, <=$	Binary operator
$, >=, ==, ! =\}$	Unary operator

1.2 Expressions

$\langle e \rangle$ ::= n	Integer
| b	Boolean
| s	String
| ()	Nothing
| x	Identifier
| $e_1 \oplus e_2$	Binary Operation
| (e_1, e_2, ..., e_n)	Vector
| [ ]	Nil (empty list)
| e_1 :: e_2	Cons (nonempty list)
| **fn** $p \rightarrow e$	Function
| **bind** p $\leftarrow e_1$ **in** e_2	Bind expression
| **bind** p p_1 ... $p_n \leftarrow e_1$ **in** e_2	Bind expression
| **bind rec** $f \leftarrow$ **fn** p $\rightarrow e_1$ **in** e_2	Recursive function bind
| **bind rec** f p_1 ... $p_n \leftarrow e_1$ in e_2	Recursive function bind
| e_1 e_2	Function application
| **if** e_1 **then** e_2 **else** e_3	Ternary expressions
| **switch** e_0 => | $p_1 \rightarrow e_1$ ... | $p_n \rightarrow e_n$ **end**	Switch expression

1.3 Patterns

$\langle p \rangle$ ::= _	Wildcard pattern*
| x	Identifier pattern**
| ()	Nothing pattern
| b	Boolean pattern
| n	Integer pattern
| s	String pattern
| (p_1, p_2, ..., p_n)	Vector pattern
| [ ]	Nil pattern
| p_1 :: p_2	Cons pattern***

* The wildcard pattern matches any value

** The identifier pattern matches any value and produces a binding to it

*** The cons pattern matches a non empty list, but only p_1 matches the head of the list and p_2 matches the remainder of the list

1.4 Values

$\langle v \rangle$ ::= n	Integer value
| s	String value
| b	Boolean value
| ()	Nothing value
| [ ]	Nil value
| v_1 :: v_2	Cons value
| (Δ, p, e)	Function Closure

1.5 Types

$\langle t \rangle$ ::= int	Integer type
| bool	Boolean type
| str	String type
| ng	Nothing type
| t_i	Type variable
| $t_1 \rightarrow t_2$	Function type*
| [t]	List type
| (t_1, t_2, ..., t_n)	Vector type
| (t)	Parenthesized type*

* The function type operator $\rightarrow$ associates to the right

For example, the type $t_1 \rightarrow t_2 \rightarrow t_3$ is parsed as $t_1 \rightarrow (t_2 \rightarrow t_3)$

Parentheses are the highest precedence operator in the type grammar, and they can be used to counter act this.

For example

$$\text{fn f} \rightarrow \text{fn x} \rightarrow \text{f x} : (t_1 \rightarrow t_2) \rightarrow t_1 \rightarrow t_2$$

2 Dynamic Semantics

In order to discuss the dynamic semantics of the programming language, we first need to define a few things.

2.1 Dynamic Environment

LambdaScript uses an environment model to make substitutions in function bodies. The environment is an object defined as follows

$$\Delta \in Var \to Value$$

It is essentally a function from a set of variable identifiers to a set of values. Note that it is a partial function because its domain will be a subset of Var

- $\Delta(x)$ represents the value x maps to in environment Δ
- $\{\}$ is the empty environment
- $\Delta[x \to v]$ represents the environment where $\Delta(y) = v$ if $y = x$, and $\Delta(y)$ otherwise
- $D(\Delta)$ is the domain of Δ

2.2 Evaluation Relation

The evaluation relation is what describes how an expression is evaluated to a value under a certain environment

Define it as follows

$$(\Delta, e) \Rightarrow v$$

It means the following: Under environment Δ, expression e evaluates to value v

2.3 The Dynamic Semantics

2.3.1 Value

$$(\Delta, v) \Rightarrow v$$

A value always evaluates to itself

2.3.2 Variable Identifiers

$$(\Delta, x) \Rightarrow \Delta(x)$$

To evaluate an identifier x, it is simply looked up in the environment Δ

2.3.3 Vector

$$(\Delta, (e_1, e_2, ..., e_n)) \Rightarrow (v_1, v_2, ..., v_n)$$

$$(\Delta, e_1) \Rightarrow v_1$$

$$(\Delta, e_2) \Rightarrow v_2$$

$$\dots$$

$$(\Delta, e_n) \Rightarrow v_n$$

To evaluate a vector, evaluate each sub expression, then construct a new vector with the values

2.3.4 Cons

$$(\Delta, e_1 :: e_2) \Rightarrow v_1 :: v_2$$

$$(\Delta, e_1) \Rightarrow v_1$$

$$(\Delta, e_2) \Rightarrow v_2$$

To evaluate a cons expression, evaluate the two operands, then return the first argument prepended to the second

3 Static Semantics