LambdaScript Syntax and Semantics

Alex Kozik

July 16, 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
$\oplus \in \{+, -, *, /, \%, <, >$	Binary operator
$, <=, >=, ==, != \}$	Unary operator

1.2 Expressions

$\langle e \rangle$ ::= n
| b
| s
| ()
| x
| $e_1 \oplus e_2$
| $(e_1, e_2, ..., e_n)$
| [ ]
| $e_1 :: e_2$
| **fn** $p \rightarrow e$
| **bind** p $\leftarrow e_1$ **in** e_2
| **bind** $p\ \ p_1\ ...\ p_n \leftarrow e_1$ **in** e_2
| **bind rec** $f \leftarrow$ **fn** p $\rightarrow e_1$ **in** e_2
| **bind rec** $f\ \ p_1\ ...\ p_n \leftarrow e_1$ in e_2
| $e_1\ \ e_2$
| **if** e_1 **then** e_2 **else** e_3
| **switch** e_0 => | $p_1 \rightarrow e_1$ ... | $p_n \rightarrow e_n$ **end**

Integer
Boolean
String
Nothing
Identifier
Binary Operation
Vector
Nil (empty list)
Cons (nonempty list)
Function
Bind expression
Bind expression
Recursive function bind
Recursive function bind
Function application
Ternary expressions
Switch expression

1.3 Patterns

$\langle p \rangle$ ::= _	Wildcard pattern*
$\mid$ x	Identifier pattern**
$\mid$ ()	Nothing pattern
$\mid$ b	Boolean pattern
$\mid$ n	Integer pattern
$\mid$ s	String pattern
$\mid$ $(p_1, p_2, ..., p_n)$	Vector pattern
$\mid$ [ ]	Nil pattern
$\mid$ $p_1 :: p_2$	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
$\mid$ s	String value
$\mid$ b	Boolean value
$\mid$ ()	Nothing value
$\mid$ [ ]	Nil value
$\mid$ $v_1 :: v_2$	Cons value
$\mid$ (Δ, p, e)	Function Closure

1.5 Types

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

$$\Delta \in Var \rightarrow Value$$

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
- $\Delta[x \rightarrow v]$ represents the environment where $\Delta(y) = v$ if $y = x$, and $\Delta(y)$ otherwise
- $D(\Delta)$ is the domain of Δ
- $\Delta_1 \circ \Delta_2$ represents the environment Δ where $\forall y \in D(\Delta_2)$, $\Delta(y) = \Delta_2(y)$, $\forall y \in D(\Delta_1) - D(\Delta_2)$, $\Delta(y) = \Delta_1(y)$. Otherwise, $\Delta(y)$ is not defined.

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 Basic 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

$$\frac{(\Delta, (e_1, e_2, ..., e_n)) \Rightarrow (v_1, v_2, ..., v_n)}{\begin{array}{c}(\Delta, e_1) \Rightarrow v_1 \\ (\Delta, e_2) \Rightarrow v_2 \\ \cdots \\ (\Delta, e_n) \Rightarrow v_n\end{array}}$$

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

2.3.4 Cons

$$\frac{(\Delta, e_1 :: e_2) \Rightarrow v_1 :: v_2}{\begin{array}{c}(\Delta, e_1) \Rightarrow v_1 \\ (\Delta, e_2) \Rightarrow v_2\end{array}}$$

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

2.4 Switch Statement Semantics

A switch statement uses an expression, call it e_0 and a list of branches. Each branch consists of a pattern and a body.

First, e_0 is evaluted to a value v_0 using the current environment Δ

Starting from the first branch, v_0 is compared to its pattern. If it matches, certain bindings are produced, which are used to evaluate its body. That value is then returned.

This process of comparing v_0 to the pattern of a branch continues until a match is made.

2.4.1 Pattern Matching Relation

In order to model a value matching some pattern, and producing some bindings, we will use the following relation

$$v \in p \rightarrow \Delta$$

This can be read as "value v matches pattern p and produces bindings Δ"

We will also use the following relation

$$v \notin p$$

This can be read as "value v does not patch pattern p"

Below are the semantics for each pattern

Wildcard

$$v \in \_ \rightarrow \{\}$$

Variable Identifier

$$v \in x \rightarrow \{\}[x \rightarrow v]$$

Nothing

$$() \in () \rightarrow \{\}$$

Boolean

$$b \in b \rightarrow \{\}$$

Integer

$$i \in i \rightarrow \{\}$$

String

$$s \in s \to \{\}$$

Nil / Empty List

$$[] \in [] \to \{\}$$

Vector

$$\frac{(v_1, v_2, ..., v_n) \in (p_1, p_2, ..., p_n) \to \Delta_1 \circ \Delta_2 \circ ... \circ ...\Delta_n}{\begin{array}{c} v_1 \in p_1 \to \Delta_1 \\ v_2 \in p_2 \to \Delta_2 \\ \cdots \\ v_n \in p_n \to \Delta_n \end{array}}$$

Cons

$$\frac{v_1 :: v_2 \in p_1 :: p_2 \to \Delta_1 \circ \Delta_2}{\begin{array}{c} v_1 \in p_1 \to \Delta_1 \\ v_2 \in p_2 \to \Delta_2 \end{array}}$$

2.4.2 The Semantics of Switch

$$\frac{(\Delta, \text{switch e => } |p_1 \to e_1 ... |p_n \to e_n \text{ end}) \implies v'}{\begin{array}{c} (\Delta, e) \implies v \\ v \notin p_i \text{ for } i < m \\ v \in p_m \to \Delta_m \text{ where } 1 \leq m \leq n \\ (\Delta \circ \Delta_m, e_j) \implies v' \end{array}}$$

Let's go through those statements one by one

1. $(\Delta, e) \implies v$ shows that e evalutes to v under environment Δ
2. $v \notin p_i$ for $i < m$ shows that v doesn't match the first $m-1$ patterns
3. $v \notin p_i$ for $i < m$ shows that v matches the m^{th} pattern and produces bindings Δ_m
4. $(\Delta \circ \Delta_m, e_m) \implies v'$ shows that the body of the m^{th} branch evaluates to a value v' under the external environment Δ composed with the new bindings Δ_m. v' is what the entire switch expression evaluates to.

3 Static Semantics