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

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