The SLOTH Programming Language

Mike Davis

April 4, 2012

Introduction

This document describes the SLOTH programming language. The key features of SLOTH are lazy evaluation of sequences and stack based programming.

A Short Tour of SLOTH

This section gives a short tour of SLOTH.

Literals

This section describes the type literals which the interpreter accepts.

BOOLEAN (→ boolval)
: The literals '@t' and '@f' produce the boolean values true and false, respectively. The capitalized versions produce the same effect.

INTEGER (→ intval)
: Any sequence that starts with a decimal digit and contains only decimal digits will cause the decimal value of the input to be pushed to the top of the stack.

SYMBOL (→ symbol)
: Any sequence beginning with a colon, e.g. ':symbol' causes a symbol reference to be pushed onto the stack. All symbols are global and there is only one symbol with a given name in the system at a time.

STRING (→ string)
: Any sequence beginning and ending with a double quotation represents a string. There is not character quoting and the string must end before the next newline.

[optional-forms . . .] ($\rightarrow$ listsequence)

Any sequence surrounded by square brackets generates a list on the stack. The forms can be literals, generators, or other forms.

Generators

This section describes generators, i.e., input that takes nothing from the stack and leaves one or more objects on the stack.

naturals ($\rightarrow$ sequence)

The literal 'naturals' causes a seqeunce object to be pushed to the top of the stack. This sequence object generates the values from 1 to $2^{31} - 1$.

fibonacci ($\rightarrow$ sequence)

The literal 'fibonacci' causes a sequence object to be pushed to the top of the stack. This sequence object generates the Fibonacci sequence 1, 1, . . ., $2^{31} - 1$.

Stack Manipulators

This section describes commands that take input from the stack, possibly writing output back to the stack.

dup (form $\rightarrow$ form form)

The command 'dup' makes a copy of the top element of the stack onto the stack.

eval (rawform $\rightarrow$ resultform)

The command 'eval' removes the top form from the stack and evaluates it. The result of the evaluation is pushed back on the stack. The result may be a message if evaluation resulted in an error. If the evaluation results in an infinite sequence, the command never returns. Literals evaluate to themselves. Sequences evalute to a non-lazy evaluations of their contents, in order. Therefore, if an infinite sequence, e.g. 'naturals', is evaluated, evaluation will never terminate.

pop (form $\rightarrow$)

The command 'pop' removes the top element from the stack.

print (form $\rightarrow$)

The command 'print' removes the top element from the stack and displays a printable representation to the standard output.

printall (forms ... →)

The command 'printall' removes all elements from the stack and displays a printable representation of each to the standard output. The top of stack is printed first.

swap (form1 form2 → form2 form1)

The command 'swap removes two elements from the stack and and puts them back onto the stack in the opposite order.

Operations

This section describes operations that take 1 or more arguments from the stack and produce a new lazy form that is pushed on the stack. In any case, if the described inputs are incorrect, then a message form is left on the stack. (The interpreter will print out these messages.)

append (sequence1 sequence2 → sequence)

The 'append' command takes two sequences from the stack and pushes onto the stack a sequence that is all of the items from sequence1 followed by all the items from sequence2.

count (sequence → intval)

The 'count' command takes a sequence off the stack and pushes onto the stack a lazy integer value representing the number of top level forms in the sequence.

drop (sequence intval → sequence)

The 'drop' command takes a sequence and an integer from the stack and pushes onto the stack a sequence that is the input sequence with the first intval items removed. If the intval is negative, zero items are removed.

flatten (sequence → sequence)

The 'flatten' command takes a sequence from the stack and pushes a sequence that contains all the leaf items from the original sequence as a single sequence. For example, if the input is [[1 2 3] [4 5 6]], then the output is [1 2 3 4 5 6].

gfibonacci (intval intval → sequence)

The 'gfibonacci' command takes two integer values off the stack and pushes a lazy sequence that is the "generalized" Fibonacci sequence seeded by those two values. The first two values are the inputs, and each value after that is the sum of its two prececessors. For example, if the seeds are a and b, the values are $a, b, a + b, a + 2b, 2a + 3b, 3a + 5b, \ldots$. The normal Fibonacci sequence is seeded by $1, 1$.

minmax (sequence → sequence)

The 'minmax' command takes a sequence off the stack and pushes a lazy sequence onto the stack. If the input sequence is empty, the output is empty. Otherwise, the output is a sequence of length two containing the minimum value in the input sequence and the maximum value in the input sequence.

repeat (form → sequence)

The 'repeat' command takes a form off the stack and returns a lazy sequence that returns form forever.

set (form symbol → form)

The 'set' command takes a form and a symbol off the stack, sets the value of the symbol to form, and pushes the form back onto the stack.

take (sequence intval → sequence)

The 'take' command takes a sequence and an integer value off the stack and pushes onto the stack a lazy sequence consisting of the first intval values from the sequence.

takeif (boolean-sequence sequence → sequence)

The 'takeif command takes two sequences off the stack and pushes onto the stack a lazy sequence representing consisting of items from the second sequence where the corresponding position in the first sequence is @t.

Operators

This section describes mathematical and logical operators. Unary operators take a single form from the stack, while binary operators can take two.

For a binary operator, if the inputs are a pair of sequences, the output is a lazy sequence where $Out_i = In_{1,i} \odot In_{2,i}$, where $\odot$ is the documented operation, Out is the output sequence and In_1 and In_2 are the first and second output sequences. i ranges up to the length of the shorter input sequence. If the inputs are both non-sequences, the output is a non-sequence also. If one input is a non-sequence and the other a sequence, evaluation occurs as if the non-sequence were a lazy sequence containing an infinite number of copies of the input.

For a unary argument, an input that is a sequence produces an output that is also a sequence. An input that is not a sequence produces an output that is not a sequence.

$+$ (form1 form2 → form)

The '$+$' operator takes two inputs and returns their sum. It only works on integers and sequences of integers.

− (form1 form2 → form)
The '−' operator takes two inputs and returns their difference, form1 − form2. It only works on integers and sequences of integers.

∗ (form1 form2 → form)
The '∗' operator takes two inputs and returns their product, form1 ∗ form2. It only works on integers and sequences of integers.

/ (form1 form2 → form)
The '/' operator takes two inputs and returns their product, form1 / form2. It only works on integers and sequences of integers. An error form is returned if form2 is zero.

% (form1 form2 → form)
The '%' operator takes two inputs and returns their product, form1 % form2. It only works on integers and sequences of integers. An error form is returned if form2 is zero.