Assignment 2 – PostFix SOS

Elizabeth Brooks, CSCI 512 27 February 2017

1 Operational Semantics

The execution of a PostFix program can be described as a step-by-step process with operational semantics. Program execution proceeds by the application of a fixed set of rules on the configurations of an abstract machine. Axioms, rewrite rules, and progress rules determine the transitions between configurations, or states of the machine.

2 S-Expresssion Grammar

Symbolic-expressions (s-expressions) may be used to represent abstract trees by parenthesized linear text strings. The following domains and rules define the syntax for the PostFix language.

Syntactic Domains and Production Rules

P ∈ Prog ::= (postfix $N_{numargs}$ Q_{body})	[Program]
Q ∈ CommandSeq ::= C^*	[CommandSequence]
C ∈ Command ::= N_{num}	[IntVal]
| pop	[Pop]
| swap	[Swap]
| Aop	[ArithOp]
| Rop	[RelOp]
| nget	[NumGet]
| sel	[Select]
| exec	[Execute]
| (Q_{coms})	[ExecutableSequence]

A ∈ ArithmeticOperator = {add, sub, mul, div, rem}

R ∈ RelationalOperator = {lt, eq, gt}

N ∈ IntLit = {..., -2, -1, 0, 1, 2, ...}

3 Abstract Machine

An abstract machine provides a mathematical model for program execution. Since we are interested in executing PostFix programs without the use of a separate stack for the input integer values, the abstract machine is simply the given command sequence pre-pended with the n-tuple of numeric inputs. For example:

(postfix 1 4 add 5 mul 6 sub 7 div) $\xrightarrow{[3]}$ [3, 4, add, 5, mul, 6, sub, 7, div]
[3, 4, add, 5, mul, 6, sub, 7, div] [IntVal]
[7, 5, mul, 6, sub, 7, div] [ArithOp]
[7, 5, mul, 6, sub, 7, div] [IntVal]
[35, 6, sub, 7, div] [ArithOp]
[35, 6, sub, 7, div] [IntVal]
[29, 7, div] [ArithOp]
[29, 7, div] [IntVal]
[4] [ArithOp]
[4] $\xrightarrow{[OF]}$ 4

4 Small-Step Operational Semantics

Small-step operational semantics (SOS) formally describes program execution in terms of rewrite rules, which specify the steps necessary to progress and transform configurations of the abstract machine.

The following PostFix language L is deterministic and strongly normalizing. Furthermore, elements A ∈ AnsExp are considered to be valid answers to Legal program P ∈ Prog, given inputs I ∈ Inputs.

Domains

V ∈ Value = IntLit + CommandSeq
S ∈ SeqInputs = Value*
FinalInputs = {S | (length S) ≥ 1 and (nth 1 S) = (IntLit↣Value N) for some N ∈ IntLit}
Inputs = IntLit*
AnsExp = IntLit

SOS

Suppose the PostFix SOS is a five-tuple and has the form PFSOS = ⟨CF, ⇒, FC, IF, OF⟩, where:

CF = CommandSeq
FC = FinalInputs
IF : (Prog × Inputs) → CF
= λ ⟨(postfix N Q), [N_1, ..., N_n]⟩.
    if N = $\bar{n}$
    then ⟨[(IntLit↣Value N_n), ..., (IntLit↣Value N_1)]@Q⟩
    else ⟨[]Command⟩
    end
OF : FC → AnsExp
    = λ ⟨S'@(IntLit↣Value N).[]Command⟩.(IntLit↣AnsExp N)

In the above PostFix SOS, for n ∈ Int, $\bar{n}$ stands for the IntLit N that denotes n. CF is the domain of configurations for the abstract machine and FC is the set of final configurations, which is a subset of IrreducibleS containing all configurations that are considered to be final states in the execution of the PostFix program. The input function, IF, maps a PostFix program and its numeric inputs to an initial configuration consisting of a body command sequence joined with an n-tuple of numeric inputs pre-pended in reverse order. A stuck configuration is returned to indicate an error if an invalid command sequence or list of numeric inputs are encountered. The output function, OF, maps a final configuration to an the answer domain and returns the integer literal at the end of the list of numeric inputs. There should not be any commands remaining in the command sequence at the end of program execution.

4.1 Rewrite Rules

The commands of PostFix programs are interpreted in a linear order, therefore, the contents of an executable sequence can be used only when they are pre-pended to the single stream of commands that is being executed by the PostFix abstract machine. Thus, the next command to be executed is

at the front of the command stream, which leads to a simple pattern for the rewrite rules below.

Transitions, which appear only in rule consequents, are all of the form $\langle$S@C_{first}.Q$\rangle \Rightarrow \langle$S'@Q'$\rangle$ where Q' is either the same as Q or is the result of pre-pending some commands onto the front of Q. The command at the head of the current command sequence, C_{firs}, is consumed by the application of the given rule.

The transition relation, $\Rightarrow$, is a binary relation on configurations that defines the allowable transitions between configurations. A transition relation is deterministic if for every cf $\in$ ReducibleS there is exactly one cf' such that cf $\Rightarrow$ cf'.

4.1.1 Axioms

An axiom represents the collection of all configuration pairs that respectively match the left-hand-side (LHS) and right-hand-side (RHS) of the transition pattern. Since an axiom has no antecedents, it is determined solely by its consequent, which is a transition pattern. A transition pattern is similar to a transition except that the LHS and RHS may contain domain variables. Furthermore, a transition pattern may be considered a schema representing all the transitions that match the pattern.

Note that a simpler notation is used to express the following PostFix rewrite rules, rather than the strict metalanguage notation described in the Design Concepts in Programming Languages text appendix. For example,

$\langle$S@(IntLit$\rightarrowtail$Command N).Q$\rangle \Rightarrow \langle$S@(IntLit$\rightarrowtail$Value N).Q$\rangle$ [num]

becomes

$\langle$S@N.Q$\rangle \Rightarrow \langle$S@N.Q$\rangle$ [num]

for improved readability.

SOS Rules

$\langle$S@N.Q$\rangle \Rightarrow \langle$S@N.Q$\rangle$ [num]

$\langle$S(Q_{exec}).$Q_{rest}\rangle \Rightarrow \langle$S@($Q_{exec}$).$Q_{rest}\rangle$ [seq]

$\langle$S@V.pop.Q$\rangle \Rightarrow \langle$S@Q$\rangle$ [pop]

$\langle[V_{N_{size}}, ..., V_1]@N_{index}$.nget.Q$\rangle \Rightarrow \langle[V_{N_{size}}, ..., V_1]@V_{N_{index}}$.Q$\rangle$ [nget]

where (compare gt N_{index} 0) $\wedge\neg$(compare gt N_{index} N_{size})

$\wedge$ ($V_{N_{index}} \in$ IntLit)

$\langle$S@V_2.V_1.swap.Q$\rangle \Rightarrow \langle$S@$V_1$.$V_2$.Q$\rangle$ [swap]

$\langle$S@0.V_{true}.V_{false}.sel.$Q_{rest}\rangle \Rightarrow \langle$S@$V_{false}$.$Q_{rest}\rangle$ [sel-false]

$\langle$S@N_{test}.V_{true}.V_{false}.sel.Q_{rest} $\rangle \Rightarrow \langle$S@$V_{true}$.$Q_{rest}\rangle$ [sel-true]

where $N_{test} \neq 0$

$\langle$S@(Q_{exec}).exec.$Q_{rest}\rangle \Rightarrow \langle$S@$Q_{exec}$@$Q_{rest}\rangle$ [execute]

$\langle$S@N_2.N_1.A.Q$\rangle \Rightarrow \langle$S@$N_{ans}$.Q$\rangle$ [arithop]

where N_{ans} = (calculate A N_2 N_1)

$\langle$S@N_2.N_1.R.Q$\rangle \Rightarrow \langle$S@1.Q$\rangle$ [relop-true]

where (compare R N_2 N_1)

$\langle$S@N_2.N_1.R.Q$\rangle \Rightarrow \langle$S@0.Q$\rangle$ [relop-false]

where $\neg$(compare R N_2 N_1)

4.2 Operational Execution

By applying the developed operational semantics to the earlier PostFix example, the integer representation of the answer is returned by the output function, OF. The transition path taken from the initial to the final configuration is shown below.

(IF $\langle$(postfix 1 4 add 5 mul 6 sub 7 div), [3]$\rangle$)

= $\langle$3, 4, add, 5, mul, 6, sub, 7, div$\rangle$

$\Rightarrow \langle$3, 4, add, 5, mul, 6, sub, 7, div$\rangle$ [num]

$\Rightarrow \langle$7, 5, mul, 6, sub, 7, div$\rangle$ [arithop]

$\Rightarrow \langle$7, 5, mul, 6, sub, 7, div$\rangle$ [num]

$\Rightarrow \langle$35, 6, sub, 7, div$\rangle$ [arithop]

$\Rightarrow \langle$35, 6, sub, 7, div$\rangle$ [num]

$\Rightarrow \langle$29, 7, div$\rangle$ [arithop]

$\Rightarrow \langle$29, 7, div$\rangle$ [num]

$\Rightarrow \langle$4$\rangle \in$ FC [arithop]

$\Rightarrow$ (OF $\langle$4$\rangle$) = 4

Thus, eight transitions were made using the developed PostFix SOS to reach the final configuration and return the integer value at the end of the input sequence.