

# CS652 Smalltalk VM Operational Semantics

Terence Parr

April 3, 2015

$T \bowtie x$	Resolve $x$ in scope $T$
$o \in X$	$o$ is instance of $X$
$v \in \text{STObject}$	a single object
$l_i \in \text{STObject}$	the $i^{\text{th}}$ argument or local variable object
$o_{\text{class}} \in \text{STMetaClassObject}$	Metaclass (type) of object $o$
$o_{\text{class}_{\text{class}}} = o_{\text{class}}$	A metaclass object is its own type
$o_{\text{superclass}} \in \text{STMetaClassObject}$	Superclass (type) of object $o$
$o_{\text{field}_i}$	The $i^{\text{th}}$ field of object $o$
$f_{\text{literal}_i}$	The $i^{\text{th}}$ literal of method $f$
$f_s^{\text{block}_i} \in \text{BlockDescriptor}$	The $i^{\text{th}}$ block of method $f$ associated with instance $\text{self}=s$
$f_s^{\text{block}_i}[-, -, -] \in \text{BlockContext}$	The $i^{\text{th}}$ block of method $f$ invoked with $\text{self}=s$
$f_s^{\text{block}_i}[-, -, -]^d \in \text{BlockContext}$	The $i^{\text{th}}$ block of method $f$ invoked with $\text{self}=s$ and having depth $d$ counting from zero at the method block; e.g., <code>f [ x  [ y ]]</code> has a method block at depth 0 with <code>x</code> and a nested block at depth 1 with <code>y</code>
$\gamma \in \text{MethodContext}^*$	Stack of method invocations growing to the right
$\delta \in \text{STObject}^*$	Operand stack of objects growing to the right
$\mathbb{S}$	The state of the VM system dictionary
$(\mathbb{S}, \gamma)$	VM state is the system dictionary and a method invocation stack with zero or more elements
$(\mathbb{S}, \gamma) \Rightarrow (\mathbb{S}', \gamma')$	VM state transition
$(\mathbb{S}, \gamma) \Rightarrow^* (\mathbb{S}', \gamma')$	Zero-or-more state transitions
$f_s[ip, l_0, ..l_{n-1}, \delta]$	Method invocation context that derived from sending message $f$ to receiver $s$ (self); $f \in \text{MethodContext}$ ; $l_i$ is local variable or argument, indexed from 0 and arguments first; $\delta$ is the operand stack; <i><math>f</math> can also represent a nested code block not just a method</i>
$f[ip, l_0, ..l_{n-1}, \delta]$	Same as previous but the receiver is unknown or irrelevant
$f[ip, -, -]$	A method invitation context with “don’t care” for locals and operand stack

Figure 1: Smalltalk VM Bytecode Specification Notation

Bytecode Instruction	Transition
<i>initial state</i>	$state_0 = (\mathbb{S}[\text{nil}, \text{true}, \text{false}, \text{Transcript}], \text{main}_m[0, \epsilon, \epsilon])$ for $m \in \text{MainClass}$ ; program terminates if $\exists state_0 \Rightarrow^* (\mathbb{S}', \epsilon)$
<b>nil</b>	$(\mathbb{S}, \gamma f[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 1, -, \delta \text{nil}])$
<b>self</b>	$(\mathbb{S}, \gamma f_s[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f_s[ip + 1, -, \delta s])$
<b>true</b>	$(\mathbb{S}, \gamma f[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 1, -, \delta \text{true}])$
<b>false</b>	$(\mathbb{S}, \gamma f[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 1, -, \delta \text{false}])$
<b>push_char</b> $c$	$(\mathbb{S}, \gamma f[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 3, -, \delta c])$
<b>push_int</b> $i$	$(\mathbb{S}, \gamma f[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, -, \delta i])$
<b>push_float</b> $i$	$(\mathbb{S}, \gamma f[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, -, \delta \text{intBitsToFloat}(i)])$
<b>push_field</b> $i$	$(\mathbb{S}, \gamma f_s[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f_s[ip + 3, -, \delta s_{field_i}])$
<b>push_local</b> $0, i$	$(\mathbb{S}, \gamma f[ip, \dots l_i \dots, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, \dots l_i \dots, \delta l_i])$
<b>push_local</b> $n > 0, i$	$(\mathbb{S}, \gamma g^{block}[-, \dots l_i \dots, -]^{d-n} \dots g^{block'}[ip, -, -]^{d-1} \dots g^{block''}[ip, -, \delta]^d) \Rightarrow$ $(\mathbb{S}, \gamma \dots g^{block''}[ip + 5, -, \delta l_i]^d)$
<b>push_literal</b> $i$	$(\mathbb{S}, \gamma f[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 3, -, \delta f_{literal_i}])$
<b>push_global</b> $i$	$(\mathbb{S}, \gamma f[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 3, -, \delta \mathbb{S}[f_{literal_i}]])$
<b>push_array</b> $n$	$(\mathbb{S}, \gamma f[ip, -, \delta a_1..a_n]) \Rightarrow (\mathbb{S}, \gamma f[ip + 3, -, \delta A])$ where $A = \text{Array}(a_1..a_n)$
<b>store_field</b> $i$	$(\mathbb{S}, \gamma f_s[ip, -, \delta \mathbf{v}]) \Rightarrow (\mathbb{S}[s_{field_i} = \mathbf{v}], \gamma f_s[ip + 3, -, \delta \mathbf{v}])$
<b>store_local</b> $n, i$	$(\mathbb{S}, \gamma f[ip, \dots l_i \dots, \delta \mathbf{v}]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, \dots l_{i-1} \mathbf{v} l_{i+1} \dots, \delta \mathbf{v}])$
<b>pop</b>	$(\mathbb{S}, \gamma f[ip, -, \delta \mathbf{v}]) \Rightarrow (\mathbb{S}, \gamma f[ip + 1, -, \delta])$
<b>send</b> $n, i$	$(\mathbb{S}, \gamma f[ip, -, \delta r_{p_1..p_n}]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, -, \delta] (r_{class} \bowtie f_{literal_i})_r[0, p_1..p_n, \epsilon])$
<b>send_super</b> $n, i$	$(\mathbb{S}, \gamma f[ip, -, \delta r_{p_1..p_n}]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, -, \delta] (r_{superclass} \bowtie f_{literal_i})_r[0, p_1..p_n, \epsilon])$
<b>block</b> $i$	$(\mathbb{S}, \gamma f[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 3, -, \delta f_s^{block_i}])$
<b>block_return</b>	$(\mathbb{S}, \gamma f[ip, -, \delta] g^{block}[-, -, \delta' \mathbf{v}]) \Rightarrow (\mathbb{S}, \gamma f[ip, -, \delta \mathbf{v}])$
(method local) <b>return</b>	$(\mathbb{S}, \gamma f[ip, -, \delta] g[-, -, \delta' \mathbf{v}]) \Rightarrow (\mathbb{S}, \gamma f[ip, -, \delta \mathbf{v}])$
(method nonlocal) <b>return</b>	$(\mathbb{S}, \gamma f[ip, -, \delta] g_s[-, -, -] \dots h[-, -, -] g_s^{block}[-, -, \delta' \mathbf{v}]) \Rightarrow (\mathbb{S}, \gamma f[ip, -, \delta \mathbf{v}])$
<b>dbg</b> $i, loc$	$(\mathbb{S}, \gamma f[ip, -, -]) \Rightarrow (\mathbb{S}[file=f_{literal_i}, line=loc[31:8], col=loc[7:0]], \gamma f[ip + 7, -, -])$ Set VM current filename to $f_{literal_i}$ and split $loc$ into char position (indexed from 0) from lower 8 bits and line number from the upper 24 bits.

Figure 2: Smalltalk VM State Transition Rules

Smalltalk fragment	Visitor method result	Side-effects
$\epsilon$	$\epsilon$ (object Code.None)	
class T : S [ ]	$\epsilon$	
main	main	
	self	
f <primitive:#primitive-name>	return	
f [ body ]	$\epsilon$	$f_{code} =$ body
		pop
		self
		return
operator [ body ]	$\epsilon$	operator <sub>code</sub> = body
		pop
		self
		return
a:x b:y ... c:z [ body ]	$\epsilon$	a:b:c <sub>code</sub> = body
		pop
		self
		return
$\underbrace{[args   locals  ]}_{f^{block_i}}$	block i	$f_{block_i} =$ nil
$\underbrace{[ body ]}_{f^{block_i}}$	block i	block_return
		$f_{block_i} =$ body
		block_return
instr <sub>1</sub> .instr <sub>2</sub> . ... instr <sub>n</sub>	instr <sub>1</sub>	
	pop	
	instr <sub>2</sub>	
	pop	
	...	
	instr <sub>n</sub>	

Figure 3: Smalltalk Class/Method/Block Compilation Rules

Smalltalk fragment	Visitor method result	Side-effects
class T [ x  ... [ ... x := expr	expr	
f : x [ ... x := expr	store_field i expr	
f [ x  ... x := expr	store_local 0, i expr	
f : x { ... [ ... x := expr	store_local 0, i expr	
Δ	store_local Δ, i	
f { [ x  ... [ ... x := expr	expr	
Δ	store_local Δ, i	
^ expr	expr	
return		
f [ ... expr w	expr send 0, i	f <sub>literal<sub>i</sub></sub> = "w"
f [ ... super w	expr send_super 0, i	f <sub>literal<sub>i</sub></sub> = "w"
f [ ... expr <sub>1</sub> op expr <sub>2</sub>	expr <sub>1</sub> expr <sub>2</sub> send 1, i	f <sub>literal<sub>i</sub></sub> = "op"
f [ ... expr w <sub>1</sub> :x <sub>1</sub> w <sub>2</sub> :x <sub>2</sub> ... w <sub>n</sub> :x <sub>n</sub>	expr send n, i	f <sub>literal<sub>i</sub></sub> = "w <sub>1</sub> :w <sub>2</sub> : ... w <sub>n</sub> :"
f [ ... super w <sub>1</sub> :x <sub>1</sub> w <sub>2</sub> :x <sub>2</sub> ... w <sub>n</sub> :x <sub>n</sub>	expr send_super n, i	f <sub>literal<sub>i</sub></sub> = "w <sub>1</sub> :w <sub>2</sub> : ... w <sub>n</sub> :"

Figure 4: Smalltalk Expression Compilation Rules