CS652 Smalltalk VM Operational Semantics

Terence Parr

April 6, 2015

$T \bowtie x$	Resolve x in scope T
$o \in \texttt{X}$	o is instance of X
$\mathbf{v} \in \texttt{STObject}$	a single object
$\boldsymbol{l}_i \in \texttt{STObject}$	the i^{th}argument or local variable object
$o_{class} \in \texttt{STMetaClassObject}$	Metaclass (type) of object o
$o_{class_{class}} = o_{class}$	A metaclass object is its own type
$o_{superclass} \in \texttt{STMetaClassObject}$	Superclass (type) of object o
o_{field_i}	The i^{th}field of object o
$f_{literal_i}$	The i^{th}literal of method f
$f_s^{block_i} \in \texttt{BlockDescriptor}$	The i^{th}block of method f associated with instance self=s
$f_s^{block_i}[\_, \_, \_] \in \texttt{BlockContext}$	The i^{th}block of method f invoked with self=s
$f_s^{block_i}[\_, \_, \_]^d \in \texttt{BlockContext}$	The i^{th}block of method f invoked with self=s and having depth d counting from zero at the method block; e.g., `f [|x| [|y|]]` has a method block at depth 0 with `x` and a nested block at depth 1 with `y`
$\gamma \in \texttt{MethodContext}^*$	Stack of method invocations growing to the right
$\delta \in \texttt{STObject}^*$	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); $f \in$ `MethodContext`; l_i is local variable or argument, indexed from 0 and arguments first; δ is the operand stack; f *can also represent a nested code block not just a method*
$f[ip, l_0, ..l_{n-1}, \delta]$	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**
initial state	$state_0 = (\mathbb{S}[\texttt{nil}, \texttt{true}, \texttt{false}, \texttt{Transcript}], \texttt{main}_m[0, \epsilon, \epsilon])$ for $m \in$ `MainClass`; program terminates if $\exists\ state_0 \Rightarrow^* (\mathbb{S}', \epsilon)$
`nil`	$(\mathbb{S}, \gamma f[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+1, \_, \delta\, \texttt{nil}])$
`self`	$(\mathbb{S}, \gamma f_s[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f_s[ip+1, \_, \delta\, s])$
`true`	$(\mathbb{S}, \gamma f[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+1, \_, \delta\, \texttt{true}])$
`false`	$(\mathbb{S}, \gamma f[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+1, \_, \delta\, \texttt{false}])$
`push_char` c	$(\mathbb{S}, \gamma f[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+3, \_, \delta\, c]))]$
`push_int` i	$(\mathbb{S}, \gamma f[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+5, \_, \delta\, i])$
`push_float` i	$(\mathbb{S}, \gamma f[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+5, \_, \delta\, \mathit{intBitsToFloat}(i)])$
`push_field` i	$(\mathbb{S}, \gamma f_s[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f_s[ip+3, \_, \delta\, s_{field_i}])$
`push_local` $0, i$	$(\mathbb{S}, \gamma f[ip, \cdots l_i \cdots, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+5, \cdots l_i \cdots, \delta\, l_i])$
`push_local` $n > 0, i$	$(\mathbb{S}, \gamma g^{block}[\_, \cdots \boldsymbol{l}_i \cdots, \_]^{d-n} \cdots g^{block'}[ip, \_, \_]^{d-1} \cdots g^{block''}[ip, \_, \delta]^d) \ \Rightarrow$ $(\mathbb{S}, \gamma \cdots g^{block''}[ip+5, \_, \delta \boldsymbol{l}_i]^d)$
`push_literal` i	$(\mathbb{S}, \gamma f[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+3, \_, \delta\, f_{literal_i}])$
`push_global` i	$(\mathbb{S}, \gamma f[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+3, \_, \delta\, \mathbb{S}[f_{literal_i}]])$
`push_array` n	$(\mathbb{S}, \gamma f[ip, \_, \delta\, a_1..a_n]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+3, \_, \delta A])$ where $A = \mathit{Array}(a_1..a_n)$
`store_field` i	$(\mathbb{S}, \gamma f_s[ip, \_, \delta\, \mathbf{v}]) \ \Rightarrow\ (\mathbb{S}[s_{field_i} = \mathbf{v}], \gamma f_s[ip+3, \_, \delta\, \mathbf{v}])$
`store_local` n, i	$(\mathbb{S}, \gamma f[ip, \cdots l_i \cdots, \delta\, \mathbf{v}]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+5, \cdots l_{i-1} \mathbf{v}\, l_{i+1} \cdots, \delta\, \mathbf{v}])$
`pop`	$(\mathbb{S}, \gamma f[ip, \_, \delta\, \mathbf{v}]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+1, \_, \delta])$
`send` n, i	$(\mathbb{S}, \gamma f[ip, \_, \delta\, r\, p_1..p_n]) \Rightarrow (\mathbb{S}, \gamma f[ip+5, \_, \delta]\ (r_{class} \bowtie f_{literal_i})_r[0, p_1..p_n, \epsilon])$
`send_super` n, i	$(\mathbb{S}, \gamma f[ip, \_, \delta\, r\, p_1..p_n]) \Rightarrow (\mathbb{S}, \gamma f[ip+5, \_, \delta]\ (r_{superclass} \bowtie f_{literal_i})_r[0, p_1..p_n, \epsilon])$
`block` i	$(\mathbb{S}, \gamma f[ip, \_, \delta]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip+3, \_, \delta\, f_s^{block_i}])$
`block_return`	$(\mathbb{S}, \gamma f[ip, \_, \delta]\ g^{block}[\_, \_, \delta'\, \mathbf{v}]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip, \_, \delta\, \mathbf{v}])$
(*method local*) `return`	$(\mathbb{S}, \gamma f[ip, \_, \delta]\ g[\_, \_, \delta'\, \mathbf{v}]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip, \_, \delta\, \mathbf{v}])$
(*method nonlocal*) `return`	$(\mathbb{S}, \gamma f[ip, \_, \delta]\ g_s[\_, \_, \_]\ \cdots\ h[\_, \_, \_]\ g_s^{block}[\_, \_, \delta'\, \mathbf{v}]) \ \Rightarrow\ (\mathbb{S}, \gamma f[ip, \_, \delta\, \mathbf{v}])$
`dbg` i, loc	$(\mathbb{S}, \gamma f[ip, \_, \_]) \Rightarrow (\mathbb{S}[\mathit{file}{=}f_{literal_i}, \mathit{line}{=}\mathit{loc}[31{:}8], \mathit{col}{=}\mathit{loc}[7{:}0]], \gamma f[ip+7, \_, \_])$ 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**
ϵ	ϵ (object `Code.None`)	
`class T : S [ ]`	ϵ	

main	*main*
	`self`
	`return`	

`f <primitive:#`*primitive-name*`>`	ϵ	

`f [` *body* `]`	ϵ	$\texttt{f}_{code} =$
		body
		`pop`
		`self`
		`return`

operator `[` *body* `]`	ϵ	$operator_{code} =$
		body
		`pop`
		`self`
		`return`

`a:x b:y` $\cdots$ `c:z [` *body* `]`	ϵ	$\texttt{a:b:c:}_{code} =$
		body
		`pop`
		`self`
		`return`

$\underbrace{\texttt{[}args\vert\ \ \vert locals\vert\ \texttt{]}}_{\texttt{f}^{block_i}}$	`block` i	$\texttt{f}_{block_i} =$
		`nil`
		`block_return`

$\underbrace{\texttt{[}\ body\ \texttt{]}}_{\texttt{f}^{block_i}}$	`block` i	$\texttt{f}_{block_i} =$ *body* `block_return`
$instr_1.\, instr_2.\, \cdots\, instr_n$	$instr_1$ `pop` $instr_2$ `pop` $\cdots$ $instr_n$	

Figure 3: Smalltalk Class/Method/Block Compilation Rules

Smalltalk fragment	**Visitor method result**	**Side-effects**
`class T` $[\|\mathrm{x}\|\cdots[\cdots\ x{:=}\,expr$	$expr$ `store_field` i	
`f:x` $[\cdots\ x{:=}\,expr$	$expr$ `store_local` $0, i$	
`f` $[\|\mathrm{x}\|\cdots\ x{:=}\,expr$	$expr$ `store_local` $0, i$	
$\underbrace{\texttt{f:x}\ [\cdots[}_{\Delta=\#scopes}\ \cdots\ x{:=}\,expr$	$expr$ `store_local` Δ, i	
$\texttt{f}\ \ \texttt{[}\cdots\underbrace{[\|\mathrm{x}\|\cdots[}_{\Delta}\cdots\ x{:=}\,expr$	$expr$ `store_local` Δ, i	
$\hat{}\,expr$	$expr$ `return`	
`f [`$\cdots\ expr\, w$	$expr$ `send` $0, i$	$\mathtt{f}^{block_j}_{literal_i}$ = "w"
`f [`$\cdots$ `super` w	$expr$ `send_super` $0, i$	$\mathtt{f}^{block_j}_{literal_i}$ = "w"
`f [`$\cdots\ expr_1\ op\ expr_2$	$expr_1$ $expr_2$ `send` $1, i$	$\mathtt{f}^{block_j}_{literal_i}$ = "op"
`f [`$\cdots\ expr\ w_1{:}x_1\ w_2{:}x_2\ \cdots\ w_n{:}x_n$	$expr$ `send` n, i	$\mathtt{f}^{block_j}_{literal_i}$ = "$w_1{:}w_2{:}\cdots w_n{:}$"
`f [`$\cdots$ `super` $w_1{:}x_1\ w_2{:}x_2\ \cdots\ w_n{:}x_n$	$expr$ `send_super` n, i	$\mathtt{f}^{block_j}_{literal_i}$ = "$w_1{:}w_2{:}\cdots w_n{:}$"
99	`push_int 99`	
`1.2`	`push_float floatToIntBits(1.2)`	
`'a string'`	`push_literal` i	$\mathtt{f}^{block_j}_{literal_i}$ = "$astring$"
`nil`	`nil`	
`self`	`self`	
`true`	`true`	
`false`	`false`	

Figure 4: Smalltalk Expression Compilation Rules