${\rm CS}652$ Smalltalk VM Operational Semantics

Terence Parr

April 6, 2015

| $T \bowtie x$ | Resolve x in scope T | |
|---|--|--|
| $o \in X$ | o is instance of X | |
| $\mathbf{v} \in \mathtt{STObject}$ | a single object | |
| $oldsymbol{l}_i \in 	exttt{STObject}$ | the i^{th} argument or local variable object | |
| $o_{class} \in \mathtt{STMetaClassObject}$ | 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 | |
| $f_s^{block_i} \in 	exttt{BlockDescriptor}$ | The i^{th} block of method f associated with instance self= s | |
| $f_s^{block_i}[extsf{-},	extsf{-},	extsf{-}] \in 	extsf{BlockContext}$ | The i^{th} block of method f invoked with self= s | |
| $f_s^{block_i}[_,_,_]^d \in 	exttt{BlockContext}$ | 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 | |
| $\delta \in \mathtt{STObject}^*$ | Operand stack of objects growing to the right | |
| S | The state of the VM system dictionary | |
| (\mathbb{S},γ) | 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); $f \in \texttt{MethodContext}; l_i$ 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 | $state_0 = (\mathbb{S}[\mathtt{nil}, \mathtt{true}, \mathtt{false}, \mathtt{Transcript}], \mathtt{main}_m[0, \epsilon, \epsilon])$ | |
| | for $m \in \text{MainClass}$; program terminates if $\exists state_0 \Rightarrow^* (S', \epsilon)$ | |
| nil | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta]) \ \Rightarrow \ (\mathbb{S}, \gamma f[ip+1, \underline{\ }, \delta \mathtt{nil}])$ | |
| self | $(\mathbb{S}, \gamma f_s[ip, \neg, \delta]) \Rightarrow (\mathbb{S}, \gamma f_s[ip+1, \neg, \delta s])$ | |
| true | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta]) \ \Rightarrow \ (\mathbb{S}, \gamma f[ip+1, \underline{\ }, \delta \mathtt{true}])$ | |
| false | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta]) \ \Rightarrow \ (\mathbb{S}, \gamma f[ip+1, \underline{\ }, \delta \mathtt{false}])$ | |
| ${\tt push_char}\ c$ | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta]) \ \Rightarrow \ (\mathbb{S}, \gamma f[ip + 3, \underline{\ }, \delta c])]$ | |
| $\mathtt{push_int}\ i$ | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, \underline{\ }, \delta i])$ | |
| ${\tt push_float}\ i$ | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, \underline{\ }, \delta \ intBitsToFloat(i)])$ | |
| $\mathtt{push_field}\ i$ | $(\mathbb{S}, \gamma f_s[ip, -, \delta]) \Rightarrow (\mathbb{S}, \gamma f_s[ip + 3, -, \delta s_{field_i}])$ | |
| ${\tt push_local}\ 0, i$ | $(\mathbb{S}, \gamma f[ip, \cdots l_i \cdots, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, \cdots l_i \cdots, \delta l_i])$ | |
| ${\tt push_local}\ n>0, i$ | $(\mathbb{S}, \gamma g^{block}[\underline{\ }, \cdots \underline{\ }i_i \cdots, \underline{\ }]^{d-n} \cdots g^{block'}[ip, \underline{\ }, \underline{\ }]^{d-1} \cdots g^{block''}[ip, \underline{\ }, \delta]^d) \ \Rightarrow$ | |
| | $(\mathbb{S}, \gamma \cdots g^{block''}[ip+5, _, \delta l_i]^d)$ | |
| ${\tt push_literal}\ i$ | $(\mathbb{S}, \gamma f[ip, J, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 3, J, \delta f_{literal_i}])$ | |
| ${\tt push_global}\ i$ | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 3, \underline{\ }, \delta \mathbb{S}[f_{literal_i}]])$ | |
| ${\tt push_array}\ n$ | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta a_1a_n]) \Rightarrow (\mathbb{S}, \gamma f[ip+3, \underline{\ }, \delta A]) \text{ where } A = Array(a_1a_n)$ | |
| $\mathtt{store_field}\;i$ | $(\mathbb{S}, \gamma f_s[ip, \neg, \delta \mathbf{v}]) \Rightarrow (\mathbb{S}[s_{field_i} = \mathbf{v}], \gamma f_s[ip + 3, \neg, \delta \mathbf{v}])$ | |
| $\mathtt{store_local}\ n, i$ | $(\mathbb{S}, \gamma f[ip, \cdots l_i \cdots, \delta \mathbf{v}]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, \cdots l_{i-1}\mathbf{v} l_{i+1} \cdots, \delta \mathbf{v}])$ | |
| pop | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta \mathbf{v}]) \ \Rightarrow \ (\mathbb{S}, \gamma f[ip+1, \underline{\ }, \delta])$ | |
| $\mathtt{send}\ n, i$ | $(\mathbb{S}, \gamma f[ip, \neg, \delta r p_1p_n]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, \neg, \delta] \left(r_{class} \bowtie f_{literal_i}\right)_r [0, p_1p_n, \epsilon])$ | |
| $\mathtt{send_super}\ n, i$ | $(\mathbb{S}, \gamma f[ip, \neg, \delta r p_1p_n]) \Rightarrow (\mathbb{S}, \gamma f[ip + 5, \neg, \delta] (r_{superclass} \bowtie f_{literal_i})_r[0, p_1p_n, \epsilon])$ | |
| $\mathtt{block}\; i$ | $(\mathbb{S}, \gamma f[ip, \cdot, \delta]) \Rightarrow (\mathbb{S}, \gamma f[ip + 3, \cdot, \delta f_s^{block_i}])$ | |
| block_return | $(\mathbb{S}, \gamma f[ip, \mathbf{x}, \delta] \ g^{block}[\mathbf{x}, \mathbf{y}, \delta' \mathbf{v}]) \ \Rightarrow \ (\mathbb{S}, \gamma f[ip, \mathbf{x}, \delta \mathbf{v}])$ | |
| $(method\ local)$ return | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta] \ g[\underline{\ }, \underline{\ }, \delta' \mathbf{v}]) \ \Rightarrow \ (\mathbb{S}, \gamma f[ip, \underline{\ }, \delta \mathbf{v}])$ | |
| $(method\ nonlocal)$ return | $(\mathbb{S}, \gamma f[ip, \underline{\ }, \delta] \ g_s[\underline{\ }, \underline{\ }, \underline{\ }] \ \cdots \ h[\underline{\ }, \underline{\ }, \underline{\ }] \ g_s^{block}[\underline{\ }, \underline{\ }, \delta' \mathbf{v}]) \ \Rightarrow \ (\mathbb{S}, \gamma f[ip, \underline{\ }, \delta \mathbf{v}])$ | |
| $dbg\; i, loc$ | $(\mathbb{S}, \gamma f[ip, _, _]) \Rightarrow (\mathbb{S}[file=f_{literal_i}, line=loc[31:8], col=loc[7:0]], \gamma f[ip+7, _, _])$ 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 |
|--|--|---|
| ϵ | $\epsilon \; (ext{object Code.None})$ | |
| class T : S [] | ϵ | |
| main | $\mid main \mid$ | |
| | self | |
| | return | |
| <pre>f <pre>frimitive:#primitive-name></pre></pre> | ϵ | |
| f [] | ϵ | $f_{code} =$ |
| | | self |
| | | return |
| f [body] | ϵ | $f_{code} =$ |
| | | body |
| | | pop |
| | | self |
| , , , , , | | return |
| operator [body] | ϵ | $operator_{code} =$ |
| | | body |
| | | pop |
| | | self |
| $\mathtt{a} \colon \mathtt{x} \ \mathtt{b} \colon \mathtt{y} \ \mathtt{c} \colon \mathtt{z} \ [\ \mathit{body} \]$ | | return |
| a. x b. y c. z [<i>bouy</i>] | ϵ | $a:b:c:_{code} =$ |
| | | body |
| | | pop |
| | | self |
| [args locals] | \mid block i | $egin{array}{c} \mathtt{return} \ \mathtt{f}_{block_i} = \end{array}$ |
| | DIGGR V | nil |
| \mathtt{f}^{block_i} | | block_return |
| [body] | block i | ${f f}_{block_i} =$ |
| f^{block_i} | | body |
| • | | block_return |
| $instr_1.instr_2. \cdots instr_n$ | $instr_1$ | |
| | pop | |
| | $instr_2$ | |
| | pop | |
| | | |
| | $\int instr_n$ | |

Figure 3: Smalltalk Class/Method/Block Compilation Rules

| Smalltalk fragment | Visitor method result | Side-effects |
|---|---|--|
| class T $[\mathbf{x} \cdots [\cdots \ x := expr]]$ | expr | |
| $f:x [\cdots x:=expr$ | $\begin{array}{c} \mathtt{store_field} \; i \\ expr \end{array}$ | |
| $f[x \cdots x := expr]$ | $\begin{array}{c} \mathtt{store_local} \ 0, i \\ expr \end{array}$ | |
| | ${	t store_local} \ 0, i$ | |
| $\mathbf{\underline{f}:x} [\cdots] \cdots x := expr$ | expr | |
| $\Delta = \#scopes$ | $	extsf{store_local} \Delta, i$ | |
| $f [\cdots \underbrace{[\mathbf{x} \cdots]}\cdots \ x := expr$ | expr | |
| Δ | $	extsf{store_local} \Delta, i$ | |
| $^{\Delta}$ expr | expr | |
| | return | 111. |
| $	extsf{f} \left[\cdots \ expr w ight.$ | expr | $\mathbf{f}_{literal_i}^{block_j} = "w"$ |
| | $\mathtt{send}\ 0, i$ | |
| $\mathtt{f} \; [\cdots \; \mathtt{super} w$ | expr | $\mathbf{f}_{literal_i}^{block_j} = w$ |
| | ${	t send_super} \ 0, i$ | |
| $f [\cdots expr_1 op expr_2]$ | $expr_1$ | $f_{literal}^{block_j} = "op"$ |
| | $expr_2$ | titeraiz |
| | $\mathtt{send}\ 1, i$ | |
| $f \left[\cdots \ expr \ w_1:x_1 \ w_2:x_2 \cdots w_n:x_n \right]$ | expr | $\mathbf{f}_{literal_i}^{block_j} = "w_1: w_2: \cdots w_n:"$ |
| | $egin{array}{c} send \; n,i \end{array}$ | $literal_i$ 1 1 2 $literal_i$ |
| $f \left[\cdots \text{ super } w_1 : x_1 \ w_2 : x_2 \cdots w_n : x_n \right]$ | expr | $\mathbf{f}_{literal_i}^{block_j} = "w_1: w_2: \cdots w_n:"$ |
| The super w_1x_1 w_2x_2 w_nx_n | $\texttt{send_super}\ n, i$ | $ _{literal_i} - _{literal_i} - _{literal_i} - _{literal_i} - _{literal_i} - _{literal_i} - _{literal_i} - _{literal_i} - _{literal_i$ |
| 99 | push_int 99 | |
| 1.2 | <pre>push_float floatToIntBits(1.2)</pre> | |
| 'a string' | $ullet$ push_literal i | $f_{literal_i}^{block_j} = "astring"$ |
| nil | nil | inerai; |
| self | self | |
| true | true | |
| false | false | |

Figure 4: Smalltalk Expression Compilation Rules