Principal bytecode type assignments

More specific: A type assignment $(\mathit{regV}_i, \mathit{opdV}_i)_{i \in \mathcal{V}}$ is more specific than $(\mathit{regT}_i, \mathit{opdT}_i)_{i \in \mathcal{D}}$, if the following three conditions are satisfied:

- $\mathcal{V} \subseteq \mathcal{D}$
- $\mathit{regV}_i \sqsubseteq_{\mathrm{reg}} \mathit{regT}_i$ for each $i \in \mathcal{V}$
- $\mathit{opdV}_i \sqsubseteq_{\mathrm{seq}} \mathit{opdT}_i$ for each $i \in \mathcal{V}$

Verification of method μ:

Attempt to construct a bytecode type assignment $(\mathit{regV}_i, \mathit{opdV}_i)_{i \in \mathcal{V}}$

$\mathcal{V} = \mathit{dom}(\mathit{visited})$, set of visited code indices

$\mathcal{C} = \mathit{dom}(\mathit{changed})$, set of changed code indices

Completeness of the verifier

Theorem. If the method μ has a bytecode type assignment $(regT_i, opdT_i)_{i \in \mathcal{D}}$, then during the verification process $(regV_i, opdV_i)_{i \in \mathcal{V}}$ is always more specific than $(regT_i, opdT_i)_{i \in \mathcal{D}}$ and no `VerifyError` occurs.

Conclusion: If bytecode can be typed, then the verifier computes a principal ($=$ most specific) bytecode type assignment.

Soundness of the verifier

Theorem. During the verification process the following invariants are satisfied where $\mathcal{C} = dom(changed)$ and $\mathcal{V} = dom(visited)$:

I1. $\mathcal{C} \subseteq \mathcal{V}$ and $\mathcal{V}$ is a set of valid code indices of the method μ.

I2. Code index 0 belongs to $\mathcal{V}$.

I3. Let $[\tau_1, \ldots, \tau_n] = argTypes(\mu)$ and $c = classNm(\mu)$. If μ is a

(a) class initialization method: $regV_0 = \emptyset$.

(b) class method: $\{0 \mapsto \tau_1, \ldots, n-1 \mapsto \tau_n\} \sqsubseteq_{\mathrm{reg}} regV_0$.

(c) instance method: $\{0 \mapsto c, 1 \mapsto \tau_1, \ldots, n \mapsto \tau_n\} \sqsubseteq_{\mathrm{reg}} regV_0$.

(d) constructor: $\{0 \mapsto InInit, 1 \mapsto \tau_1, \ldots, n \mapsto \tau_n\} \sqsubseteq_{\mathrm{reg}} regV_0$.

(The constructor of class `Object` is treated as an instance method.)

I4. The list $opdV_0$ is empty.

I5. If $i \in \mathcal{V} \setminus \mathcal{C}$, then $check(\mu, i, regV_i, opdV_i)$ is true.

I6. If $i \in \mathcal{V} \setminus \mathcal{C}$ and $(j, regS, opdS) \in succ(\mu, i, regV_i, opdV_i)$, then $j \in \mathcal{V}$, $regS \sqsubseteq_{\mathrm{reg}} regV_j$ and $opdS \sqsubseteq_{\mathrm{seq}} opdV_j$.

Soundness of the verifier (continued)

17. If $i \in \mathcal{V} \setminus \mathcal{C}$, $code(i) = Ret(x)$ and $regV_i(x) = \texttt{retAddr}(s)$, then for all reachable $j \in \mathcal{V} \setminus \mathcal{C}$ with $code(j) = Jsr(s)$:

(a) $j + 1 \in \mathcal{V}$,

(b) $regV_i \sqsubseteq_{\text{reg}} mod(s) \lhd regV_{j+1}$,

(c) $opdV_i \sqsubseteq_{\text{seq}} opdV_{j+1}$,

(d) $regV_j \sqsubseteq_{\text{reg}} mod(s) \ntriangleleft regV_{j+1}$,

(e) if $\texttt{retAddr}(\ell)$ occurs in $mod(s) \ntriangleleft regV_{j+1}$, then each code index which belongs to s belongs to l,

(f) neither $(c, k)_{new}$ nor $InInit$ occur in $mod(s) \ntriangleleft regV_{j+1}$.

18. If $i \in \mathcal{V}$ and $\texttt{retAddr}(s)$ occurs in $regV_i$, then i belongs to s.
If $i \in \mathcal{V}$ and $\texttt{retAddr}(s)$ occurs in $opdV_i$, then $i = s$.

Verifying $\mathsf{JVM}_{\mathcal{I}}$ — the diligent $\mathsf{JVM}_{\mathcal{I}}$

Dynamic functions:

$$
\begin{aligned}
&regV &&: \mathit{Nat} \to Map(\mathit{RegNo}, \mathit{VerifyType})\\
&opdV &&: \mathit{Nat} \to \mathit{VerifyType}^*\\
&visited &&: \mathit{Nat} \to \mathit{Bool}\\
&changed &&: \mathit{Nat} \to \mathit{Bool}
\end{aligned}
$$

Transition rules:

$$
\begin{aligned}
&diligentVM_I =\\
&\quad \textbf{if } dom(changed) \neq \emptyset \textbf{ then}\\
&\qquad verifyScheme_I(code, maxOpd, propagateVM_I, succ_I, check_I)\\
&\quad \textbf{else}\\
&\qquad trustfulVM_I
\end{aligned}
$$

Verifying JVM$_{\mathcal{I}}$ (continued)

$verifyScheme_I(code, maxOpd, propagateVM, succ, check) =$
  choose $pc \in dom(changed)$
    if $check(code(pc), maxOpd, pc, regV_{pc}, opdV_{pc})$ **then**
      $changed(pc) := undef$
      $propagateVM(code, succ, pc)$
    else
      $halt :=$ `"Verification failed"`

$propagateVM_I(code, succ, pc) =$
  forall $(s, regS, opdS) \in succ(code(pc), pc, regV_{pc}, opdV_{pc})$
    $propagateSucc(code, s, regS, opdS)$

Verifying JVM$_{\mathcal{I}}$ (continued)

$$
\begin{array}{l}
propagateSucc(code, s, regS, opdS) = \\
\quad \textbf{if } s \notin dom(visited) \textbf{ then} \\
\qquad \textbf{if } validCodeIndex(code, s) \textbf{ then} \\
\qquad\quad regV_s := \{(x,t) \mid (x,t) \in regS, validReg(t,s)\} \\
\qquad\quad opdV_s := [\textbf{if } validOpd(t,s) \textbf{ then } t \textbf{ else unusable} \mid t \in opdS] \\
\qquad\quad visited(s) \;\; := True \\
\qquad\quad changed(s) := True \\
\qquad \textbf{else} \\
\qquad\quad halt := \texttt{"Verification failed (invalid code index)"} \\
\quad \textbf{elseif } regS \sqsubseteq_{\mathrm{reg}} regV_s \wedge opdS \sqsubseteq_{\mathrm{seq}} opdV_s \textbf{ then } skip \\
\quad \textbf{elseif } length(opdS) = length(opdV_s) \textbf{ then} \\
\qquad regV_s \qquad := regV_s \sqcup_{\mathrm{reg}} regS \\
\qquad opdV_s \qquad := opdV_s \sqcup_{\mathrm{opd}} opdS \\
\qquad changed(s) := True \\
\quad \textbf{else } halt := \texttt{"Propagate failed"}
\end{array}
$$

Merging of verify types

Valid return addresses:

$$\begin{aligned} validReg(\texttt{retAddr}(l), pc) &= pc \in belongsTo(l) \\ validReg(t, pc) &= True \end{aligned}$$

$$\begin{aligned} validOpd(\texttt{retAddr}(l), pc) &= (l = pc) \\ validOpd(t, pc) &= True \end{aligned}$$

Merging of verify types:

$$t_1 \sqcup t_2 = \textbf{if } t_1 = t_2 \textbf{ then } t_1 \textbf{ else } \texttt{unusable}$$
$$rs_1 \sqcup rs_2 = rs_1 \cup rs_2$$

$$opdS \sqcup_{\text{opd}} opdV = [s \sqcup v \mid (s, v) \in zip(opdS, opdV)]$$

$$regS \sqcup_{\text{reg}} regT = \{(x, t) \mid (x, t) \in R, t \neq \texttt{unusable}\}$$
$$\textbf{where } R = \{(x, regS(x) \sqcup regT(x)) \mid x \in dom(regS) \cap dom(regT)\}$$

Verifying JVM$_{\mathcal{C}}$ — the diligent JVM$_{\mathcal{C}}$

$$
\begin{aligned}
&\mathit{initVerify}(\mathit{meth}) = \\
&\quad \mathit{visited}(0) \quad := \mathit{True} \\
&\quad \mathit{changed}(0) := \mathit{True} \\
&\quad \mathit{regV}_0 \qquad := \mathit{formals}(\mathit{meth}) \\
&\quad \mathit{opdV}_0 \qquad := [\,] \\
&\quad \textbf{forall } i \in \mathit{dom}(\mathit{visited}), i \neq 0 \\
&\qquad \mathit{visited}(i) \quad := \mathit{undef} \\
&\qquad \mathit{changed}(i) := \mathit{undef} \\
&\qquad \mathit{regV}_i \qquad := \mathit{undef} \\
&\qquad \mathit{opdV}_i \qquad := \mathit{undef}
\end{aligned}
$$

Verifying JVM$_\mathcal{E}$ — the diligent JVM$_\mathcal{E}$

$$
\begin{array}{l}
\mathit{propagateVM}_E(\mathit{code}, \mathit{succ}, \mathit{pc}) = \\
\quad \mathit{propagateVM}_I(\mathit{code}, \mathit{succ}, \mathit{pc}) \\
\quad \textbf{case } \mathit{code}(\mathit{pc}) \textbf{ of} \\
\qquad \mathit{Jsr}(s) \rightarrow \mathit{enterJsr}(s) := \{\mathit{pc}\} \cup \mathit{enterJsr}(s) \\
\qquad\qquad\qquad\quad \textbf{forall } (i, x) \in \mathit{leaveJsr}(s), i \notin \mathit{dom}(\mathit{changed}) \\
\qquad\qquad\qquad\qquad \textbf{if } \mathit{regV}_i(x) = \mathtt{retAddr}(s) \textbf{ then} \\
\qquad\qquad\qquad\qquad\quad \mathit{propagateJsr}(\mathit{code}, \mathit{pc}, s, i) \\
\qquad \mathit{Ret}(x) \rightarrow \textbf{let } \mathtt{retAddr}(s) = \mathit{regV}_{\mathit{pc}}(x) \\
\qquad\qquad\qquad\quad \mathit{leaveJsr}(s) := \{(\mathit{pc}, x)\} \cup \mathit{leaveJsr}(s) \\
\qquad\qquad\qquad\quad \textbf{forall } j \in \mathit{enterJsr}(s), j \notin \mathit{dom}(\mathit{changed}) \\
\qquad\qquad\qquad\qquad \mathit{propagateJsr}(\mathit{code}, j, s, \mathit{pc})
\end{array}
$$

Remark:

$$\mathit{enterJsr}(s) = \{j \in \mathcal{V} \mid \mathit{code}(j) = \mathit{Jsr}(s)\}$$

$$\mathit{leaveJsr}(s) \supseteq \{(i, x) \mid i \in \mathcal{V},\ \mathit{code}(i) = \mathit{Ret}(x),\ \mathit{regV}_i(x) = \mathtt{rA}(s)\}$$

Verifying JVM$_\mathcal{E}$ (continued)

$$
\begin{aligned}
&propagateJsr(code, j, s, i) = \\
&\quad propagateSucc(code, j+1, regJ \oplus mod(s) \lhd regV_i, opdV_i) \\
&\quad \textbf{where} \\
&\qquad regJ = \{(x,t) \mid (x,t) \in mod(s) \ntriangleleft regV_j, validJump(t,s) \land \\
&\qquad\qquad\qquad t \neq (\_,\_)_{new} \land t \neq InInit\}
\end{aligned}
$$

$$
\begin{aligned}
validJump(\texttt{retAddr}(l), s) &= belongsTo(s) \subseteq belongsTo(l) \\
validJump(t, s) &= True
\end{aligned}
$$

Remark: Invariant **I7** remains true.