Successors for JVM$_{\mathcal{I}}$ instructions

$$
\begin{array}{l}
succ_I(instr, pc, regT, opdT) = \\
\quad \textbf{case } instr \textbf{ of} \\
\qquad Prim(p) \rightarrow \\
\qquad\quad \{(pc+1, regT, drop(opdT, argSize(p)) \cdot returnType(p))\} \\
\qquad Dupx(s_1, s_2) \rightarrow \\
\qquad\quad \{(pc+1, regT, drop(opdT, s_1+s_2) \cdot \\
\qquad\qquad\qquad\qquad take(opdT, s_2) \cdot take(opdT, s_1+s_2))\} \\
\qquad Pop(s) \rightarrow \{(pc+1, regT, drop(opdT, s))\} \\
\qquad Load(t, x) \rightarrow \\
\qquad\quad \textbf{if } size(t) = 1 \textbf{ then} \\
\qquad\qquad \{(pc+1, regT, opdT \cdot [regT(x)])\} \\
\qquad\quad \textbf{else} \\
\qquad\qquad \{(pc+1, regT, opdT \cdot [regT(x), regT(x+1)])\}
\end{array}
$$

Successors for JVM$_\mathcal{I}$ instructions (continued)

$$
\begin{array}{l}
succ_I(instr, pc, regT, opdT) = \\
\quad \textbf{case } instr \textbf{ of} \\
\qquad Store(t, x) \rightarrow \\
\qquad\quad \textbf{if } size(t) = 1 \textbf{ then} \\
\qquad\qquad \{(pc+1, regT \oplus \{(x, top(opdT))\}, drop(opdT, 1))\} \\
\qquad\quad \textbf{else} \\
\qquad\qquad \{(pc+1, regT \oplus \{(x, t_0), (x+1, t_1)\}, drop(opdT, 2))\} \\
\qquad\quad \textbf{where } [t_0, t_1] = take(opdT, 2) \\
\qquad Goto(o) \quad\;\; \rightarrow \{(o, regT, opdT)\} \\
\qquad Cond(p, o) \rightarrow \{(pc+1, regT, drop(opdT, argSize(p))), \\
\qquad\qquad\qquad\qquad\;\; (o, regT, drop(opdT, argSize(p)))\}
\end{array}
$$

Successors for JVM$_{\mathcal{C}}$ instructions

$$
\begin{aligned}
& succ_C(meth)(instr, pc, regT, opdT) = \\
& \quad succ_I(instr, pc, regT, opdT) \cup \\
& \quad \textbf{case } instr \textbf{ of} \\
& \qquad GetStatic(t, c/f) \rightarrow \{(pc+1, regT, opdT \cdot t)\} \\
& \qquad PutStatic(t, c/f) \rightarrow \{(pc+1, regT, drop(opdT, size(t)))\} \\
& \qquad InvokeStatic(t, c/m) \rightarrow \\
& \qquad \quad \{(pc+1, regT, drop(opdT, argSize(c/m)) \cdot t)\} \\
& \qquad Return(mt) \rightarrow \emptyset
\end{aligned}
$$

Successors for JVM$_{\mathcal{O}}$ instructions

$$
\begin{array}{l}
succ_O(meth)(instr, pc, regT, opdT) = \\
\quad succ_C(meth)(instr, pc, regT, opdT) \cup \\
\quad \textbf{case } instr \textbf{ of} \\
\qquad New(c) \rightarrow \{(pc+1, regS, opdS \cdot [(c, pc)_{new}])\} \\
\qquad\quad \textbf{where } regS = \{(x, t) \mid (x, t) \in regT, t \neq (c, pc)_{new}\} \\
\qquad\quad opdS = [\textbf{if } t = (c, pc)_{new} \textbf{ then unusable else } t \mid t \in opdT] \\
\qquad GetField(t, c/f) \rightarrow \{(pc+1, regT, drop(opdT, 1) \cdot t)\} \\
\qquad PutField(t, c/f) \rightarrow \{(pc+1, regT, drop(opdT, 1 + size(t)))\} \\
\qquad InstanceOf(c) \rightarrow \{(pc+1, regT, drop(opdT, 1) \cdot [\texttt{int}])\} \\
\qquad Checkcast(t) \rightarrow \{(pc+1, regT, drop(opdT, 1) \cdot t)\}
\end{array}
$$

Successors for JVM$_{\mathcal{O}}$ instructions (continued)

$$
\begin{array}{l}
succ_O(meth)(instr, pc, regT, opdT) = \\
\quad \textbf{case } instr \textbf{ of} \\
\quad\quad InvokeSpecial(t, c/m) \rightarrow \\
\quad\quad\quad \textbf{let } opdT' = drop(opdT, 1 + argSize(c/m)) \cdot t \\
\quad\quad\quad \textbf{if } methNm(m) = \texttt{"<init>"} \textbf{ then} \\
\quad\quad\quad\quad \textbf{case } top(drop(opdT, argSize(c/m))) \textbf{ of} \\
\quad\quad\quad\quad (c, o)_{new} \rightarrow \{(pc + 1, regT[c/(c, o)_{new}], opdT'[c/(c, o)_{new}])\} \\
\quad\quad\quad\quad InInit \quad \rightarrow \textbf{let } c/\_ = meth \\
\quad\quad\quad\quad\quad\quad\quad \{(pc + 1, regT[c/InInit], opdT'[c/InInit])\} \\
\quad\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad\quad \{(pc + 1, regT, opdT')\} \\
\quad\quad InvokeVirtual(t, c/m) \rightarrow \\
\quad\quad\quad \textbf{let } opdT' = drop(opdT, 1 + argSize(c/m)) \cdot t \\
\quad\quad\quad \{(pc + 1, regT, opdT')\}
\end{array}
$$

Successors for JVM$_{\mathcal{E}}$ instructions

$$
\begin{aligned}
&succ_E(meth)(instr, pc, regT, opdT) = \\
&\quad succ_O(meth)(instr, pc, regT, opdT) \cup \\
&\quad allhandlers(instr, meth, pc, regT) \cup \\
&\quad \textbf{case } instr \textbf{ of} \\
&\qquad Athrow \rightarrow \emptyset \\
&\qquad Jsr(s) \quad \rightarrow \{(s, regT, opdT \cdot [\mathtt{retAddr}(s)])\} \\
&\qquad Ret(x) \quad \rightarrow \emptyset
\end{aligned}
$$

$$
\begin{aligned}
allhandlers(Jsr(\_), m, pc, regT) &= \{\} \\
allhandlers(Goto(\_), m, pc, regT) &= \{\} \\
allhandlers(Return(\_), m, pc, regT) &= \{\} \\
allhandlers(Load(\_, \_), m, pc, regT) &= \{\} \\
allhandlers(instr, m, pc, regT) &= \\
\quad \{(h, regT, [t]) \mid (f, u, h, t) \in excs(m) \wedge f \leq pc < u\}
\end{aligned}
$$

Successors are monotonic

Lemma: Assume that

- $regV \sqsubseteq_{\text{reg}} regT$, $opdV \sqsubseteq_{\text{seq}} opdT$,
- $check(meth, pc, regT, opdT) = True$,
- $(s, regS, opdS) \in succ(meth, pc, regV, opdV)$.

Then there exists $(s, regU, opdU) \in succ(meth, pc, regT, opdT)$ such that $regS \sqsubseteq_{\text{reg}} regU$ and $opdS \sqsubseteq_{\text{seq}} opdU$.

$$
\begin{array}{ccc}
 & \begin{array}{ccc} regV & \sqsubseteq_{\text{reg}} & regT \\ opdV & \sqsubseteq_{\text{seq}} & opdT \end{array} & \\
\begin{array}{l} \text{for each successor} \\ (s, regS, opdS) \text{ of} \\ (pc, regV, opdV) \end{array} & \begin{array}{ccc} \downarrow & \quad\quad & \downarrow \end{array} & \begin{array}{l} \text{there exists a successor} \\ (s, regU, opdU) \text{ of} \\ (pc, regT, opdT) \end{array} \\
 & \begin{array}{ccc} regS & \sqsubseteq_{\text{reg}} & regU \\ opdS & \sqsubseteq_{\text{seq}} & opdU \end{array} &
\end{array}
$$

Reachability

Control transfer instructions:

$Goto(i)$, $Cond(p,i)$, $Return(t)$, $Athrow$, $Jsr(i)$, $Ret(x)$.

Successor index: A code index j is called a successor index of i (wrt. m), if one of the following conditions is true:

- $code(i)$ is not a control transfer instruction and $j = i + 1$
- $code(i) = Goto(j)$
- $code(i) = Cond(p,k)$ or $code(i) = Jsr(k)$ and $j \in \{i+1, k\}$
- There exists a handler $(f, u, j, \_) \in excs(m)$ such that $f \le i < u$ and $code(i)$ is neither Jsr, $Goto$, $Return$ nor $Load$.

Reachble: A code index j is reachable from i if there exists a finite (possibly empty) sequence of successor steps from i to j.

Subroutines

Subroutine: If i is reachable from 0 and the $code(i) = Jsr(s)$, then the code index s is called a subroutine.

Return from subroutine. A code index r is a possible return from subroutine s, if $code(s) = Store(\texttt{addr}, x)$, $code(r) = Ret(x)$ and r is reachable from $s + 1$ on a path that does not use any $Store(\_, x)$ instruction.

Belongs to a subroutine: A code index i belongs to subroutine s, if there exists a possible return r from s such that $s \leq i \leq r$.

Modified variables: Let s be a subroutine. A variable x belongs to $mod(s)$, if there exists a code index i which belongs to s such that $code(i) = Store(t, y)$ and one of the following conditions is satisfied:

- $size(t) = 1$ and $x = y$
- $size(t) = 2$ and $x = y$ or $x = y + 1$.

Bytecode type assignments

A bytecode type assignment with domain $\mathcal{D}$ for a method μ is a family $(\mathit{regT}_i, \mathit{opdT}_i)_{i \in \mathcal{D}}$ of type frames satisfying the following conditions:

T1. $\mathcal{D}$ is a set of valid code indices of the method μ.

T2. Code index 0 belongs to $\mathcal{D}$.

T3. Let $[\tau_1, \ldots, \tau_n] = \mathit{argTypes}(\mu)$ and $c = \mathit{classNm}(\mu)$. If μ is a

(a) class initialization method: $\mathit{regT}_0 = \emptyset$.

(b) class method: $\{0 \mapsto \tau_1, \ldots, n-1 \mapsto \tau_n\} \sqsubseteq_{\mathrm{reg}} \mathit{regT}_0$.

(c) instance method: $\{0 \mapsto c, 1 \mapsto \tau_1, \ldots, n \mapsto \tau_n\} \sqsubseteq_{\mathrm{reg}} \mathit{regT}_0$.

(d) constructor: $\{0 \mapsto \mathit{InInit}, 1 \mapsto \tau_1, \ldots, n \mapsto \tau_n\} \sqsubseteq_{\mathrm{reg}} \mathit{regT}_0$.

T4. The list opdT_0 is empty.

T5. If $i \in \mathcal{D}$, then $\mathit{check}(\mu, i, \mathit{regT}_i, \mathit{opdT}_i)$ is true.

T6. If $i \in \mathcal{D}$ and $(j, \mathit{regS}, \mathit{opdS}) \in \mathit{succ}(\mu, i, \mathit{regT}_i, \mathit{opdT}_i)$, then $j \in \mathcal{D}$, $\mathit{regS} \sqsubseteq_{\mathrm{reg}} \mathit{regT}_j$ and $\mathit{opdS} \sqsubseteq_{\mathrm{seq}} \mathit{opdT}_j$.

Bytecode type assignments (continued)

T7. If $i \in \mathcal{D}$, $code(i) = Ret(x)$ and $regT_i(x) = \mathtt{retAddr}(s)$, then for all reachable $j \in \mathcal{D}$ with $code(j) = Jsr(s)$:

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

(b) $regT_i \sqsubseteq_{\mathrm{reg}} mod(s) \lhd regT_{j+1}$,

(c) $opdT_i \sqsubseteq_{\mathrm{seq}} opdT_{j+1}$,

(d) $regT_j \sqsubseteq_{\mathrm{reg}} mod(s) \ntriangleleft regT_{j+1}$,

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

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

T8. If $i \in \mathcal{D}$ and $\mathtt{retAddr}(s)$ occurs in $regT_i$, then i belongs to s. If $i \in \mathcal{D}$ and $\mathtt{retAddr}(s)$ occurs in $opdT_i$, then $i = s$.

Notation.

$$X \lhd f := \{(x, y) \in f \mid x \in X\}$$

$$X \ntriangleleft f := \{(x, y) \in f \mid x \notin X\}$$

Bytecode type assignments (soundness)

Theorem. If every method in the class environment $cenv$ has a bytecode type assignment, then the defensive JVM does not halt with the message "`Runtime check failed`".

Remark. If the JVM is executing the code of a method μ which has a bytecode type assignment with domain $\mathcal{D}$, then $pc \in \mathcal{D}$.