Vocabulary of the trustful JVM$_{\mathcal{E}}$

Instructions:

$$\begin{aligned} \mathit{Instr} = \ & \ldots \\ & \mid \mathit{Athrow} \\ & \mid \mathit{Jsr}(\mathit{Offset}) \\ & \mid \mathit{Ret}(\mathit{RegNo}) \end{aligned}$$

Universes:

$$\begin{aligned} \mathit{Exc} = \mathit{Exc}(\mathit{from} \ & : \mathit{Pc}, \\ \mathit{upto} \ & : \mathit{Pc}, \\ \mathit{handle} \ & : \mathit{Pc}, \\ \mathit{type} \ & : \mathit{Class}) \end{aligned}$$

$$\mathit{Switch} = \ldots \mid \mathit{Throw}(\mathit{Ref}) \mid \mathit{ThrowInit}(\mathit{Ref})$$

Trustful execution of JVM$_{\mathcal{E}}$ instructions

$$
\begin{array}{l}
execVM_E(instr) = \\
\quad execVM_O(instr) \\
\quad \textbf{case } instr \textbf{ of} \\
\qquad Athrow \rightarrow \textbf{let } [r] = take(opd, 1) \\
\qquad\qquad\qquad \textbf{if } r \neq null \textbf{ then } switch := Throw(r) \\
\qquad\qquad\qquad\quad \textbf{else } raise(\texttt{"NullPointerException"}) \\
\qquad Jsr(s) \rightarrow opd := opd \cdot [pc + 1] \\
\qquad\qquad\quad\; pc \;\; := s \\
\qquad Ret(x) \rightarrow pc := reg(x) \\
\qquad Prim(p) \rightarrow \textbf{let } ws = take(opd, argSize(p)) \\
\qquad\qquad\qquad \textbf{if } p \in divMod \wedge sndArgIsZero(ws) \textbf{ then} \\
\qquad\qquad\qquad\quad raise(\texttt{"ArithmeticException"})
\end{array}
$$

Trustful execution of $\mathrm{JVM}_{\mathcal{E}}$ instructions (continued)

$$
\begin{array}{l}
\mathit{GetField}(\_, c/f) \rightarrow \textbf{let } [r] = \mathit{take}(\mathit{opd}, 1) \\
\qquad\qquad\qquad\qquad \textbf{if } r = \mathit{null} \textbf{ then} \\
\qquad\qquad\qquad\qquad\quad \mathit{raise}(\texttt{"NullPointerException"}) \\
\mathit{PutField}(\_, c/f) \rightarrow \textbf{let } [r] \cdot \mathit{ws} = \mathit{take}(\mathit{opd}, 1 + \mathit{size}(c/f)) \\
\qquad\qquad\qquad\qquad \textbf{if } r = \mathit{null} \textbf{ then} \\
\qquad\qquad\qquad\qquad\quad \mathit{raise}(\texttt{"NullPointerException"}) \\
\mathit{InvokeSpecial}(\_, c/m) \rightarrow \\
\quad \textbf{let } [r] \cdot \mathit{ws} = \mathit{take}(\mathit{opd}, 1 + \mathit{argSize}(c/m)) \\
\quad \textbf{if } r = \mathit{null} \textbf{ then } \mathit{raise}(\texttt{"NullPointerException"}) \\
\mathit{InvokeVirtual}(\_, c/m) \rightarrow \\
\quad \textbf{let } [r] \cdot \mathit{ws} = \mathit{take}(\mathit{opd}, 1 + \mathit{argSize}(c/m)) \\
\quad \textbf{if } r = \mathit{null} \textbf{ then } \mathit{raise}(\texttt{"NullPointerException"}) \\
\mathit{Checkcast}(c) \rightarrow \textbf{let } r = \mathit{top}(\mathit{opd}) \\
\qquad\qquad\qquad\qquad \textbf{if } r \neq 0 \wedge \neg(\mathit{classOf}(r) \sqsubseteq c) \textbf{ then} \\
\qquad\qquad\qquad\qquad\quad \mathit{raise}(\texttt{"ClassCastException"})
\end{array}
$$

Trustful execution of $\mathrm{JVM}_{\mathcal{E}}$ instructions (continued)

Raising exceptions:

$raise(c) = (switch := Call(\texttt{Object/"<fail"} \cdot c \cdot \texttt{">"}, [\,]))$

Body of $\texttt{Object/"<fail"} \cdot c \cdot \texttt{">"}$:

$New(c)$
$Dupx(0, 1)$
$InvokeSpecial(\texttt{void}, c/\texttt{<init>}())$
$Athrow$

Catching exceptions:

$match(pc, c, Exc(f, u, h, t)) = f \leq pc \wedge pc < u \wedge c \preceq_{\mathrm{h}} t$
$handler(m, pc, c) = e$
where $[e] \cdot \_ = [e \mid e \in excs(m), match(pc, c, e)]$
$escapes(m, pc, c) = \nexists e \in excs(m) : match(pc, c, e)$

Trustful execution of $\mathrm{JVM}_{\mathcal{E}}$ instructions (continued)

$$
\begin{array}{l}
switchVM_E = \\
\quad switchVM_C \\
\quad \textbf{case } switch \textbf{ of} \\
\qquad Call(meth, args) \rightarrow \textbf{if } isAbstract(meth) \textbf{ then} \\
\qquad\qquad\qquad\qquad raise(\texttt{"AbstractMethodError"}) \\
\qquad InitClass(c) \rightarrow \textbf{if } unusable(c) \textbf{ then} \\
\qquad\qquad\qquad\qquad raise(\texttt{"NoClassDefFoundError"})
\end{array}
$$

Trustful execution of $\mathrm{JVM}_{\mathcal{E}}$ instructions (continued)

$$
\begin{array}{l}
switchVM_E = \\
\quad \textbf{case } switch \textbf{ of} \\
\quad\quad Throw(r) \rightarrow \textbf{if } \neg escapes(meth, pc, classOf(r)) \textbf{ then} \\
\quad\quad\quad\quad \textbf{let } exc = handler(meth, pc, classOf(r)) \\
\quad\quad\quad\quad\quad pc \;\; := handle(exc) \\
\quad\quad\quad\quad\quad opd \;\; := [r] \\
\quad\quad\quad\quad\quad switch := Noswitch \\
\quad\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad\quad \textbf{if } methNm(meth) = \texttt{"<clinit>"} \textbf{ then} \\
\quad\quad\quad\quad\quad\quad \textbf{if } \neg(classOf(r) \preceq_{\mathrm{h}} \texttt{Error}) \textbf{ then} \\
\quad\quad\quad\quad\quad\quad\quad raise(\texttt{"ExceptionInInitializerError"}) \\
\quad\quad\quad\quad\quad\quad\quad pc := undef \\
\quad\quad\quad\quad\quad\quad \textbf{else } switch := ThrowInit(r) \\
\quad\quad\quad\quad\quad \textbf{else } popFrame(0, [\,])
\end{array}
$$

Trustful execution of $\mathrm{JVM}_{\mathcal{E}}$ instructions (continued)

$$
\begin{array}{l}
switchVM_E = \\
\quad \textbf{case } switch \textbf{ of} \\
\qquad ThrowInit(r) \rightarrow \textbf{let } c = classNm(meth) \\
\qquad\qquad\qquad\qquad classState(c) := Unusable \\
\qquad\qquad\qquad\qquad popFrame(0, [\,]) \\
\qquad\qquad\qquad\qquad \textbf{if } \neg superInit(top(stack), c) \textbf{ then} \\
\qquad\qquad\qquad\qquad\quad switch := Throw(r)
\end{array}
$$

$$
\begin{array}{l}
superInit((\_, \_, \_, m), c) = \\
\quad methNm(m) = \texttt{"<clinit>"} \wedge super(classNm(m)) = c
\end{array}
$$

$$
trustfulVM_E = trustfulScheme_C(execVM_E, switchVM_E)
$$

Compilation of Java$_{\mathcal{E}}$ statements

$$\mathcal{S}(\texttt{throw}\ exp;) = \mathcal{E}(exp) \cdot Athrow$$

$$\mathcal{S}(\texttt{try}\ stm\ \texttt{catch}\ (c_1\, x_1)\, stm_1 \ldots \texttt{catch}\ (c_n\, x_n)\, stm_n\ ) =$$
$$\quad \text{try} \cdot \mathcal{S}(stm) \cdot \text{tryEnd} \cdot Goto(\text{end}) \cdot$$
$$\quad \text{handle}_1 \cdot Store(\texttt{addr}, \overline{x_1}) \cdot \mathcal{S}(stm_1) \cdot Goto(\text{end}) \cdot$$
$$\quad \vdots$$
$$\quad \text{handle}_n \cdot Store(\texttt{addr}, \overline{x_n}) \cdot \mathcal{S}(stm_n) \cdot Goto(\text{end}) \cdot$$
$$\quad \text{end}$$

$$\mathcal{S}(stm_1\ \texttt{finally}\ stm_2) =$$
$$\quad \text{try}_\text{f} \cdot \mathcal{S}(stm_1) \cdot Jsr(\text{fin}) \cdot Goto(\text{end}) \cdot$$
$$\quad \text{default} \cdot Store(\texttt{addr}, \overline{exc}) \cdot Jsr(\text{fin}) \cdot Load(\texttt{addr}, \overline{exc}) \cdot Athrow \cdot$$
$$\quad \text{fin} \cdot Store(\texttt{addr}, \overline{ret}) \cdot \mathcal{S}(stm_2) \cdot Ret(\overline{ret}) \cdot$$
$$\quad \text{end}$$

Compilation of Java$_{\mathcal{E}}$ statements (continued)

$$\mathcal{S}(\texttt{continue}\ lab;) = \textbf{let}\ [\mathrm{fin}_1, \ldots, \mathrm{fin}_n] = \mathit{finallyLabsUntil}(lab)$$
$$\mathit{Jsr}(\mathrm{fin}_1) \cdot \ldots \cdot \mathit{Jsr}(\mathrm{fin}_n) \cdot \mathit{Goto}(lab_c)$$

$$\mathcal{S}(\texttt{break}\ lab;) = \textbf{let}\ [\mathrm{fin}_1, \ldots, \mathrm{fin}_n] = \mathit{finallyLabsUntil}(lab)$$
$$\mathit{Jsr}(\mathrm{fin}_1) \cdot \ldots \cdot \mathit{Jsr}(\mathrm{fin}_n) \cdot \mathit{Goto}(lab_b)$$

$$\mathcal{S}(\texttt{return};) = \textbf{let}\ [\mathrm{fin}_1, \ldots, \mathrm{fin}_n] = \mathit{finallyLabs}$$
$$\mathit{Jsr}(\mathrm{fin}_1) \cdot \ldots \cdot \mathit{Jsr}(\mathrm{fin}_n) \cdot \mathit{Return}(\texttt{void})$$

$$\mathcal{S}(\texttt{return}\ exp;) =$$
$$\textbf{if}\ \mathit{finallyCodeToExec}\ \textbf{then}$$
$$\mathcal{E}(exp) \cdot \mathit{Store}(\mathcal{T}(exp), \overline{var}) \cdot$$
$$\textbf{let}\ [\mathrm{fin}_1, \ldots, \mathrm{fin}_n] = \mathit{finallyLabs}$$
$$\mathit{Jsr}(\mathrm{fin}_1) \cdot \ldots \cdot \mathit{Jsr}(\mathrm{fin}_n) \cdot \mathit{Load}(\mathcal{T}(exp), \overline{var}) \cdot \mathit{Return}(\mathcal{T}(exp))$$
$$\textbf{else}$$
$$\mathcal{E}(exp) \cdot \mathit{Return}(\mathcal{T}(exp))$$

Definition of Java$_\mathcal{E}$ exception tables

$$
\begin{aligned}
&\mathcal{X}(\texttt{try}\ stm\ \texttt{catch}\ (c_1\, x_1)\, stm_1 \ldots \texttt{catch}\ (c_n\, x_n)\, stm_n\ ) = \\
&\quad \mathcal{X}(stm)\cdot \\
&\quad \mathcal{X}(stm_1)\cdot Exc(\mathrm{try}, \mathrm{tryEnd}, \mathrm{handle}_1, c_1)\cdot \\
&\quad \vdots \\
&\quad \mathcal{X}(stm_n)\cdot Exc(\mathrm{try}, \mathrm{tryEnd}, \mathrm{handle}_n, c_n) \\
&\mathcal{X}(stm_1\ \texttt{finally}\ stm_2) = \\
&\quad \mathcal{X}(stm_1)\cdot Exc(\mathrm{try_f}, \mathrm{default}, \mathrm{default}, \texttt{Throwable})\cdot \mathcal{X}(stm_2) \\
&\mathcal{X}(\{stm_1 \ldots stm_n\}) = \mathcal{X}(stm_1)\cdot \ldots \cdot \mathcal{X}(stm_n) \\
&\mathcal{X}(\texttt{if}\ (exp)\, stm_1\ \texttt{else}\ stm_2) = \mathcal{X}(stm_1)\cdot \mathcal{X}(stm_2) \\
&\mathcal{X}(\texttt{while}\ (exp)\, stm) = \mathcal{X}(stm) \\
&\mathcal{X}(lab:\ stm) = \mathcal{X}(stm) \\
&\mathcal{X}(\texttt{static}\ stm) = \mathcal{X}(stm) \\
&\mathcal{X}(\_) = \epsilon
\end{aligned}
$$

Why ThrowInit(r)?

```
class A {
  static { // At run-time: ArithmeticException
    int i = 0;
    System.out.println(1 / i);
  }
}
class B extends A {
  static {
    try { System.out.println("B"); }
    catch (Throwable e) {
      System.out.println(e + " caught");
    }
  }
  public static void main(String[] argv) { }
}

Exception table: Exc(0, 8, 8, Throwable)
```