Correctness of the compiler

ASM for Java		ASM for the JVM
collection of packages π	$\overset{compiler}{\longmapsto}$	$cenv$ class environment
$\mathfrak{A}_0$		$\mathfrak{B}_0$
$\vdots$		$\vdots$
$\mathfrak{A}_m$		$\mathfrak{B}_{\sigma(m)}$
$\vdots$		$\vdots$
states (Java)		states (JVM)

Compiler correctness proof:

Construction of mapping $\sigma: \mathbb{N} \to \mathbb{N}$ such that

- $m \leq n \implies \sigma(m) \leq \sigma(n)$
- state $\mathfrak{A}_m$ of Java is equivalent to state $\mathfrak{B}_{\sigma(m)}$ of the JVM

Dynamic states of Java and the JVM

Java	JVM
pos	pc
$restbody$	opd
$locals$	reg
$meth$	$meth$
$frames$	$stack$
$classState$	$classState$
$globals$	$globals$
$heap$	$heap$
	$switch$

Equivalence?

Equivalence of states

Equivalence of *pos* and *pc*:

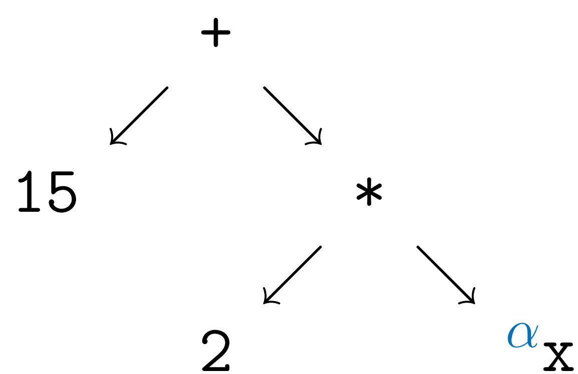
Associate to each position in method body an interval in code array.

$$\alpha \longmapsto [code(i) \mid \mathrm{beg}_\alpha \leq i < \mathrm{end}_\alpha]$$

- $restbody_n/\alpha$ not evaluated $\implies pc_{\sigma(n)} = \mathrm{beg}_\alpha$
- $restbody_n/\alpha$ evaluated $\implies pc_{\sigma(n)} = \mathrm{end}_\alpha$

Equivalence of *restbody* and *opd*:

The operand stack of the JVM can be extracted from *restbody*.



$$javaOpd(restbody, \alpha) = [15, 2]$$

Equivalence of states (continued)

Equivalence of locals **and** reg**:**

Define $\mathit{locals} \approx \mathit{reg}$ iff for each $x \in \mathrm{dom}(\mathit{locals})$:

- If $\mathit{size}(\mathcal{T}(x)) = 1$, then $\mathit{jvmVal}(\mathit{locals}(x)) = [\mathit{reg}(\overline{x})]$.
- If $\mathit{size}(\mathcal{T}(x)) = 2$, then $\mathit{jvmVal}(\mathit{locals}(x)) = [\mathit{reg}(\overline{x}), \mathit{reg}(\overline{x}+1)]$.

Equivalence of frames **and** stack**:**

$[\,] \approx [\,]$.

Assume that

1. $\mathit{frames} \approx \mathit{stack}$,
2. $\mathit{locals} \approx \mathit{reg}$,
3. reg contains correct return addresses for pos in $\mathit{restbody}$,
4. $\mathit{pc} = \mathrm{beg}_{\mathit{pos}}$ or $\mathit{pc} = \mathrm{end}_{\mathit{pos}}$ depending on $\mathit{restbody}/\mathit{pos}$. Then

$$\mathit{frames} \cdot (\mathit{meth}, \mathit{restbody}, \mathit{pos}, \mathit{locals}) \approx \mathit{stack} \cdot (\mathit{pc}, \mathit{reg}, \mathit{opd}, \mathit{meth}).$$

Problem: Correctness of subroutine return addresses.

General case: *restbody*

```
c/m(...) {
    ⋮
    abr finally {
        ⋮
        Norm finally {
            ⋮
            Exc(r) finally {
                ⋮
                pos
                ⋮
            }
        }
    }
}
```

Theorem: Correctness of the compiler

Theorem. The following invariants are true for $\alpha = pos_n$:

(reg) $locals_n \approx reg_{\sigma(n)}$

(stack) $frames_n \approx stack_{\sigma(n)}$

(beg) If $restbody_n/\alpha$ is not evaluated, then

1. $pc_{\sigma(n)} = \mathrm{beg}_\alpha$, or $\mathrm{beg}_\alpha < \mathrm{end}_\alpha$ and $code(\mathrm{beg}_\alpha) = Goto(pc_{\sigma(n)})$,
2. $opd_{\sigma(n)} = javaOpd(restbody_n, \alpha)$.

(exp) If α is an $\mathcal{E}$-position, $restbody_n/\alpha = v$ and v is a value or a finite sequence of values, then

1. $pc_{\sigma(n)} = \mathrm{end}_\alpha$,
2. $opd_{\sigma(n)} = javaOpd(restbody_n, \alpha) \cdot jvmVal(v)$.

Theorem: Correctness of the compiler (continued)

(bool1) If α is a $\mathcal{B}_1(lab)$-position and $restbody_n/\alpha = True$, or if α is a $\mathcal{B}_0(lab)$-position and $restbody_n/\alpha = False$, then

1. $pc_{\sigma(n)} = lab$,
2. $opd_{\sigma(n)} = javaOpd(restbody_n, \alpha)$.

(bool2) If α is a $\mathcal{B}_1(lab)$-position and $restbody_n/\alpha = False$, or if α is a $\mathcal{B}_0(lab)$-position and $restbody_n/\alpha = True$, then

1. $pc_{\sigma(n)} = \mathrm{end}_\alpha$,
2. $opd_{\sigma(n)} = javaOpd(restbody_n, \alpha)$.

(new) If $body(meth_n)/\alpha = \texttt{new}\ c$ and $restbody_n/\alpha = ref$, then

1. $pc_{\sigma(n)} = \mathrm{end}_\alpha$,
2. $opd_{\sigma(n)} = javaOpd(restbody_n, \alpha) \cdot [ref, ref]$.

Theorem: Correctness of the compiler (continued)

(stm) If α is an $\mathcal{S}$-position and $restbody_n/\alpha = Norm$, then

1. $pc_{\sigma(n)} = \mathrm{end}_\alpha$,
2. $opd_{\sigma(n)} = []$.

(abr) If $restbody_n/\alpha = abr$ and abr is not an exception, then

1. $opd_{\sigma(n)} = []$,
2. $pc_{\sigma(n)}$ is a continuation for abr at position α wrt. $reg_{\sigma(n)}$.

(exc) If $restbody_n/\alpha = Exc(r)$ and $body(meth_n)/\alpha \neq$ `static` _, then

1. $switch_{\sigma(n)} = Throw(r)$,
2. $\mathrm{beg}_\alpha \leq pc_{\sigma(n)}$,
3. $pc_{\sigma(n)} < \mathrm{end}_\alpha$, or α is an $\mathcal{E}$-position and $pc_{\sigma(n)} < \mathrm{end}_{up(\alpha)}$,
4. there is no $(f, u, \_, c) \in \mathcal{X}(\alpha)$ such that $f \leq pc_{\sigma(n)} < u$ and $classOf(r) \preceq_{\mathrm{h}} c$.

Theorem: Correctness of the compiler (continued)

(exc-clinit) If $restbody_n/\alpha = Exc(r)$ and $body(meth_n)/\alpha = \texttt{static}\,\_$, then $switch_{\sigma(n)} = ThrowInit(r)$.

(clinit) Assume that $restbody_n/\alpha = \texttt{static}\,\_$ and $c = classNm(meth_n)$.
If $c \neq \texttt{Object}$ and not $initialized(super(c))$, then $switch_{\sigma(n)} = InitClass(super(c))$, otherwise $switch_{\sigma(n)} = Noswitch$.

(fin) $reg_{\sigma(n)}$ contains correct return addresses for α in $restbody_n$.

If nothing is said about $switch$, then $switch_{\sigma(n)} = Noswitch$.

Proof. By induction on n. 83 cases on 22 pages. $\square$

Properties of the exception table $\mathcal{X}({}^{\alpha}stm)$

Lemma. The exception table has the following properties:

1. If $(f, u, \_, \_) \in \mathcal{X}(\alpha)$, then $\mathrm{beg}_{\alpha} \leq f$ and $u \leq \mathrm{end}_{\alpha}$.
2. If β is a position inside ${}^{\alpha}stm$ and h is a handler which occurs in the table $\mathcal{X}(\alpha)$ before the subtable $\mathcal{X}(\beta)$, then the interval protected by h is disjoint to the interval $\{i \mid \mathrm{beg}_{\beta} \leq i < \mathrm{end}_{\beta}\}$.
3. If β is a direct subposition of ${}^{\alpha}stm$ and β is not the position of a `try` Block, or a `try-catch` statement, then the intervals of handlers in $\mathcal{X}(\alpha)$ which do not belong to $\mathcal{X}(\beta)$ are disjoint to $\{i \mid \mathrm{beg}_{\beta} \leq i < \mathrm{end}_{\beta}\}$.

1. $\{i \mid f \leq i < u\} \subseteq \{i \mid \mathrm{beg}_{\alpha} \leq i < \mathrm{end}_{\alpha}\}$

2. $\mathcal{X}(\alpha) = [\ldots, Exc(f, u, h, t), \ldots \underbrace{\ldots\ldots}_{\mathcal{X}(\beta)} \ldots]$

3. $\mathcal{X}(\alpha) = [\ldots, Exc(f, u, h, t), \ldots \underbrace{\ldots\ldots}_{\mathcal{X}(\beta)} \ldots, Exc(f, u, h, t), \ldots]$

Continuations

Continuations for break. Code index i is a continuation for an abruption $Break(lab)$ at position α, if $finallyLabsUntil(\alpha, lab) = [\mathrm{fin}_1, \ldots, \mathrm{fin}_k]$ and

$$\begin{array}{lcl} code(i) & = & Jsr(\mathrm{fin}_1) \\ \vdots & & \vdots \\ code(i+k-1) & = & Jsr(\mathrm{fin}_k) \\ code(i+k) & = & Goto(lab_b). \end{array}$$

Continuations for continue. Code index i is a continuation for an abruption $Continue(lab)$ at position α, if $finallyLabsUntil(\alpha, lab) = [\mathrm{fin}_1, \ldots, \mathrm{fin}_k]$ and

$$\begin{array}{lcl} code(i) & = & Jsr(\mathrm{fin}_1) \\ \vdots & & \vdots \\ code(i+k-1) & = & Jsr(\mathrm{fin}_k) \\ code(i+k) & = & Goto(lab_c). \end{array}$$

Continuations (continued)

Continuations for return void. Code index i is a continuation for an abruption Return at position α, if $\mathit{finallyLabs}(\alpha) = [\mathrm{fin}_1, \ldots, \mathrm{fin}_k]$ and

$$\begin{aligned}
\mathit{code}(i) &= \mathit{Jsr}(\mathrm{fin}_1) \\
\vdots \qquad & \qquad \vdots \\
\mathit{code}(i+k-1) &= \mathit{Jsr}(\mathrm{fin}_k) \\
\mathit{code}(i+k) &= \mathit{Return}(\mathtt{void})
\end{aligned}$$

Continuations for return value. Code index i is a continuation for a $\mathit{Return}(\mathit{val})$ at position α wrt. reg, if $\mathit{finallyLabs}(\alpha) = [\mathrm{fin}_1, \ldots, \mathrm{fin}_k]$ and

$$\begin{aligned}
\mathit{code}(i) &= \mathit{Jsr}(\mathrm{fin}_1) \\
\vdots \qquad & \qquad \vdots \\
\mathit{code}(i+k-1) &= \mathit{Jsr}(\mathrm{fin}_k) \\
\mathit{code}(i+k) &= \mathit{Load}(\tau, x) \\
\mathit{code}(i+k+1) &= \mathit{Return}(\tau),
\end{aligned}$$

if $\mathit{size}(\tau) = 1$, then $\mathit{jvmVal}(\mathit{val}) = [\mathit{reg}(x)]$,
if $\mathit{size}(\tau) = 2$, then $\mathit{jvmVal}(\mathit{val}) = [\mathit{reg}(x), \mathit{reg}(x+1)]$.

Correct return addresses

Definition. We say that reg contains correct return addresses for position α in $restbody$, if the following conditions are satisfied:

(fin-norm) For each β, if $restbody/\beta = (Norm\ \texttt{finally}\ s)$ and α is in s, then $code(reg(\overline{\mathrm{ret}}_\beta)) = Goto(\mathrm{end}_\beta)$.

(fin-abr) For each β, if $restbody/\beta = (abr\ \texttt{finally}\ s)$, abr is not an exception and α is in s, then $reg(\overline{\mathrm{ret}}_\beta)$ is a continuation for abr at position β with respect to reg.

(fin-exc) For each β, if $restbody/\beta = (Exc(r)\ \texttt{finally}\ s)$ and α is in s, then $reg(\overline{\mathrm{ret}}_\beta) = \mathrm{default}_\beta + 2$ and $reg(\overline{\mathrm{exc}}_\beta) = r$.