# *The Jinja Virtual Machine (JVM)*

# *Overview*

1. Architecture, instructions, execution
2. Type system
3. Bytecode verifier: a dataflow analyzer
4. Compiler

# *Architecture*

# *A picture*

# A stack machine

$$
\begin{aligned}
\textit{jvm-state} &= \textit{addr option} \times \textit{heap} \times \textit{frame list} \\
\textit{frame} &= \textit{opstack} \times \textit{registers} \times \textit{cname} \times \textit{mname} \times \textit{pc} \\
\textit{opstack} &= \textit{val list} \\
\textit{registers} &= \textit{val list} \\
\textit{pc} &= \textit{nat}
\end{aligned}
$$

Terminology: “register” = “local variable”

# *Instructions (1)*

**datatype** *instr* $=$

| | |
|---|---|
| `Load` *nat* | load from register |
| \| `Store` *nat* | store into register |
| \| `Push` *val* | push a constant |
| \| `Pop` | remove top element |
| \| `New` *cname* | create object on heap |
| \| `Getfield` *vname cname* | fetch field from object |
| \| `Putfield` *vname cname* | set field in object |
| \| `Checkcast` *cname* | check if object is of class *C* |
| \| `Invoke` *mname nat* | call method with *n* parameters |
| \| `Return` | return from method |

# *Instructions (2)*

| | |
|---|---|
| `IAdd` | integer addition |
| `CmpEq` | equality comparison |
| `IfFalse` *int* | branch if top of stack false |
| `Goto` *int* | goto relative address |
| `Throw` | throw exception |

# Generic programs

JVM programs are just like Jinja programs, but with different method bodies.
Use *parameterized* type of programs do define both:

$$\begin{aligned}
\alpha\ \mathit{prog} &= \alpha\ \mathit{cdecl\ list} \\
\alpha\ \mathit{cdecl} &= \mathit{cname} \times \alpha\ \mathit{class} \\
\alpha\ \mathit{class} &= \mathit{cname} \times \mathit{fdecl\ list} \times \alpha\ \mathit{mdecl\ list} \\
\alpha\ \mathit{mdecl} &= \mathit{mname} \times \mathit{ty\ list} \times \mathit{ty} \times \alpha \\
\\
\mathit{J\text{-}prog} &= (\mathit{vname\ list} \times \mathit{expr})\ \mathit{prog}
\end{aligned}$$

Important:

$P \vdash C$ *sees* $F{:}T$ *in* $D$ etc are defined on $\alpha$ *prog*, not just on *J-prog*.

# JVM programs

$$\mathit{jvm\text{-}prog} = (\mathit{nat} \times \mathit{nat} \times \mathit{instr\ list} \times \mathit{ex\text{-}table})\ \mathit{prog}$$

Interpretation of method body $(\mathit{mxs}, \mathit{mxl}_0, \mathit{is}, \mathit{xt})$:

- *mxs*: maximal operand stack size
- $\mathit{mxl}_0$: number of local variables, excluding *this* and parameters.
- *is*: “bytecode”
- *xt*: *exception table* (later)

# *Execution*

# *An example*

# Functional single-step execution (1)

$$exec :: jvm\text{-}prog \Rightarrow jvm\text{-}state \Rightarrow jvm\text{-}state\ option$$

If the stack is empty, stop:

$$exec\ P\ (xp, h, []) = None$$

If there is an unhandled exception, stop:

$$exec\ P\ (\lfloor a \rfloor, h, frs) = None$$

# Functional single-step execution (2)

Otherwise, execute an instruction:

$$\begin{array}{l}
\mathit{exec}\ P\ (\mathit{None}, h, (\mathit{stk}, \mathit{loc}, C, M, \mathit{pc}) \cdot \mathit{frs}) = \\
\lfloor \mathit{let}\ i = (\mathit{instrs\text{-}of}\ P\ C\ M)_{[pc]}; \\
\quad (\mathit{xp}', h', \mathit{frs}') = \mathit{exec\text{-}instr}\ i\ P\ h\ \mathit{stk}\ \mathit{loc}\ C\ M\ \mathit{pc}\ \mathit{frs} \\
\ \mathit{in\ case}\ \mathit{xp}'\ \mathit{of}\ \mathit{None} \Rightarrow (\mathit{None}, h', \mathit{frs}') \\
\quad \mid \lfloor a \rfloor \Rightarrow \mathit{find\text{-}handler}\ P\ a\ h\ ((\mathit{stk}, \mathit{loc}, C, M, \mathit{pc}) \cdot \mathit{frs}) \rfloor
\end{array}$$

Note: if an exception is raised, $h'$ and $\mathit{stk}'$ are discarded.

# Relational notation

$$P \vdash \sigma \xrightarrow{\text{jvm}}_1 \sigma' = (\mathit{exec}\ P\ \sigma = \lfloor \sigma' \rfloor)$$

$\xrightarrow{\text{jvm}}$ is the reflexive transitive closure of $\xrightarrow{\text{jvm}}_1$

# Functional method lookup

$$method :: \alpha\ prog \Rightarrow cname \Rightarrow mname \Rightarrow cname \times ty\ list \times ty \times \alpha$$

$$P \vdash C\ sees\ M: Ts \rightarrow T = m\ in\ D \Longrightarrow$$

$$method\ P\ C\ M = (D, Ts, T, m)$$

Fetching instructions:

$$instrs\text{-}of\ P\ C\ M \equiv$$

$$(let\ (D, Ts, T, (mxs, mxl_0, is, xt)) = method\ P\ C\ M\ in\ is)$$

# *Functional fields lookup*

$$\mathit{fields} :: \alpha\ \mathit{prog} \Rightarrow \mathit{cname} \Rightarrow ((\mathit{vname} \times \mathit{cname}) \times \mathit{ty})\ \mathit{list}$$

$$P \vdash C\ \mathit{has\text{-}fields}\ \mathit{FDTs} \Longrightarrow \mathit{fields}\ P\ C = \mathit{FDTs}$$

# Instruction execution

$$\mathit{exec\text{-}instr} :: \mathit{instr} \Rightarrow \mathit{jvm\text{-}prog} \Rightarrow \mathit{heap} \Rightarrow \mathit{opstack} \Rightarrow \mathit{registers} \Rightarrow \mathit{cname} \Rightarrow \mathit{mname} \Rightarrow \mathit{pc} \Rightarrow \mathit{frame\ list} \Rightarrow \mathit{jvm\text{-}state}$$

$$\mathit{exec\text{-}instr}\ (\texttt{Load}\ n)\ P\ h\ stk\ loc\ C\ M\ pc\ frs = (\mathit{None}, h, (loc_{[n]} \cdot stk, loc, C, M, pc + 1) \cdot frs)$$

$$\mathit{exec\text{-}instr}\ (\texttt{Store}\ n)\ P\ h\ (v \cdot stk)\ loc\ C\ M\ pc\ frs = (\mathit{None}, h, (stk, loc[n := v], C, M, pc + 1) \cdot frs)$$

# Push, Pop

$$\mathit{exec\text{-}instr}\ (\mathsf{Push}\ v)\ P\ h\ stk\ loc\ C\ M\ pc\ frs =$$
$$(\mathit{None}, h, (v \cdot stk, loc, C, M, pc + 1) \cdot frs)$$

$$\mathit{exec\text{-}instr}\ \mathsf{Pop}\ P\ h\ (v \cdot stk)\ loc\ C\ M\ pc\ frs =$$
$$(\mathit{None}, h, (stk, loc, C, M, pc + 1) \cdot frs)$$

# New

$$
\begin{aligned}
&\textit{exec-instr}\ (\mathsf{New}\ C)\ P\ h\ stk\ loc\ C_0\ M\ pc\ frs = \\
&(\textit{case new-Addr h of} \\
&\ \textit{None} \Rightarrow \\
&\quad (\lfloor \textit{addr-of-sys-xcpt OutOfMemory} \rfloor, h, \\
&\quad (stk, loc, C_0, M, pc) \cdot frs) \\
&\ \mid \lfloor a \rfloor \Rightarrow \\
&\quad (\textit{None}, h(a \mapsto (C, \textit{init-fields}\ (\textit{fields}\ P\ C))), \\
&\quad (\textit{Addr}\ a \cdot stk, loc, C_0, M, pc + 1) \cdot frs))
\end{aligned}
$$

# Getfield

$$
\begin{aligned}
&\mathit{exec\text{-}instr}\ (\texttt{Getfield}\ F\ C)\ P\ h\ (v \cdot \mathit{stk})\ \mathit{loc}\ C_0\ M\ \mathit{pc}\ \mathit{frs} = \\
&(\mathit{let}\ \mathit{xp}' = \\
&\qquad \mathit{if}\ v = \mathit{Null}\ \mathit{then}\ \lfloor \mathit{addr\text{-}of\text{-}sys\text{-}xcpt}\ \mathit{NullPointer} \rfloor\ \mathit{else}\ \mathit{None}; \\
&\quad \mathit{Addr}\ a = v;\ \lfloor (D, \mathit{fs}) \rfloor = h\ a \\
&\ \mathit{in}\ (\mathit{xp}', h, (\mathit{the}\ (\mathit{fs}\ (F, C)) \cdot \mathit{stk}, \mathit{loc}, C_0, M, \mathit{pc} + 1) \cdot \mathit{frs}))
\end{aligned}
$$

# Putfield

$$
\begin{aligned}
&\textit{exec-instr}\ (\texttt{Putfield}\ F\ C)\ P\ h\ (v \cdot r \cdot stk)\ loc\ C_0\ M\ pc\ frs = \\
&(\textit{let}\ xp' = \\
&\quad \textit{if}\ r = \textit{Null}\ \textit{then}\ \lfloor \textit{addr-of-sys-xcpt NullPointer} \rfloor \\
&\quad \textit{else}\ \textit{None}; \\
&\quad \textit{Addr}\ a = r;\ \lfloor (D, fs) \rfloor = h\ a; \\
&\quad h' = h(a \mapsto (D, fs((F, C) \mapsto v))) \\
&\textit{in}\ (xp', h', (stk, loc, C_0, M, pc + 1) \cdot frs))
\end{aligned}
$$

# Checkcast

$$
\begin{aligned}
&\mathit{exec\text{-}instr}\ (\texttt{Checkcast}\ C)\ P\ h\ (v \cdot \mathit{stk})\ \mathit{loc}\ C_0\ M\ \mathit{pc}\ \mathit{frs} = \\
&(\mathit{let}\ \mathit{xp}' = \\
&\qquad \mathit{if}\ \neg\ \mathit{cast\text{-}ok}\ P\ C\ h\ v \\
&\qquad \mathit{then}\ \lfloor \mathit{addr\text{-}of\text{-}sys\text{-}xcpt}\ \mathit{ClassCast} \rfloor\ \mathit{else}\ \mathit{None} \\
&\ \mathit{in}\ (\mathit{xp}', h, (v \cdot \mathit{stk}, \mathit{loc}, C_0, M, \mathit{pc} + 1) \cdot \mathit{frs}))
\end{aligned}
$$

# Invoke

$$
\begin{aligned}
&\mathit{exec\text{-}instr}\ (\texttt{Invoke}\ M\ n)\ P\ h\ stk\ loc\ C_0\ M_0\ pc\ frs = \\
&(\mathit{let}\ ps = \mathit{take}\ n\ stk;\ r = stk_{[n]};\\
&\quad xp' = \\
&\quad\ \mathit{if}\ r = \mathit{Null}\ \mathit{then}\ \lfloor \mathit{addr\text{-}of\text{-}sys\text{-}xcpt}\ \mathit{NullPointer} \rfloor\ \mathit{else}\ \mathit{None};\\
&\quad \mathit{Addr}\ a = r;\ \lfloor (C, \_) \rfloor = h\ a;\\
&\quad (D, M', Ts, mxs, mxl_0, ins, xt) = \mathit{method}\ P\ C\ M;\\
&\quad f' = ([], [r] @ \mathit{rev}\ ps @ \mathit{replicate}\ mxl_0\ \mathit{arbitrary}, D, M, 0)\\
&\ \mathit{in}\ (xp', h, f' \cdot (stk, loc, C_0, M_0, pc) \cdot frs))
\end{aligned}
$$

# Return

$$
\begin{aligned}
&\mathit{exec\text{-}instr}\ \texttt{Return}\ P\ h\ (v \cdot \mathit{stk}_0)\ \mathit{loc}_0\ C_0\ M_0\ \mathit{pc}\ \mathit{frs} = \\
&(\mathit{if}\ \mathit{frs} = []\ \mathit{then}\ (\mathit{None}, h, [])\\
&\ \mathit{else}\ \mathit{let}\ (\mathit{stk}, \mathit{loc}, C, M, \mathit{pc}) \cdot \_ = \mathit{frs};\\
&\qquad (D, \mathit{Ts}, T, m) = \mathit{method}\ P\ C_0\ M_0;\ n = |\mathit{Ts}|\\
&\quad \mathit{in}\ (\mathit{None}, h,\\
&\qquad (v \cdot \mathit{drop}\ (n+1)\ \mathit{stk}, \mathit{loc}, C, M, \mathit{pc}+1) \cdot \mathit{tl}\ \mathit{frs}))
\end{aligned}
$$

# IAdd, CmpEq

$$\textit{exec-instr}\ \texttt{IAdd}\ P\ h\ (\textit{Intg}\ i_2 \cdot \textit{Intg}\ i_1 \cdot \textit{stk})\ \textit{loc}\ C\ M\ \textit{pc}\ \textit{frs} =$$
$$(\textit{None}, h, (\textit{Intg}\ (i_1 + i_2) \cdot \textit{stk}, \textit{loc}, C, M, \textit{pc} + 1) \cdot \textit{frs})$$

$$\textit{exec-instr}\ \texttt{CmpEq}\ P\ h\ (v_2 \cdot v_1 \cdot \textit{stk})\ \textit{loc}\ C\ M\ \textit{pc}\ \textit{frs} =$$
$$(\textit{None}, h, (\textit{Bool}\ (v_1 = v_2) \cdot \textit{stk}, \textit{loc}, C, M, \textit{pc} + 1) \cdot \textit{frs})$$

# Goto, IfFalse

$$\mathit{exec\text{-}instr}\ (\texttt{Goto}\ i)\ P\ h\ stk\ loc\ C\ M\ pc\ frs =$$
$$(\mathit{None}, h, (stk, loc, C, M, \mathit{nat}\ (\mathit{int}\ pc + i)) \cdot frs)$$

$$\mathit{exec\text{-}instr}\ (\texttt{IfFalse}\ i)\ P\ h\ (v \cdot stk)\ loc\ C\ M\ pc\ frs =$$
$$(\mathit{let}\ pc' = \mathit{if}\ v = \mathit{Bool\ False}\ \mathit{then}\ \mathit{nat}\ (\mathit{int}\ pc + i)\ \mathit{else}\ pc + 1$$
$$\mathit{in}\ (\mathit{None}, h, (stk, loc, C, M, pc') \cdot frs))$$

# Throw

$$
\begin{aligned}
&\mathit{exec\text{-}instr}\ \texttt{Throw}\ P\ h\ (v \cdot stk)\ loc\ C\ M\ pc\ frs = \\
&(\mathit{let}\ xp' = \\
&\qquad \mathit{if}\ v = \mathit{Null}\ \mathit{then}\ \lfloor \mathit{addr\text{-}of\text{-}sys\text{-}xcpt}\ \mathit{NullPointer} \rfloor \\
&\qquad \mathit{else}\ \lfloor \mathit{the\text{-}Addr}\ v \rfloor \\
&\ \mathit{in}\ (xp', h, (v \cdot stk, loc, C, M, pc) \cdot frs))
\end{aligned}
$$

# *Exceptions*

# Exception table

$$\textit{ex-table} = \textit{ex-entry list}$$
$$\textit{ex-entry} = \textit{pc} \times \textit{pc} \times \textit{cname} \times \textit{pc} \times \textit{nat}$$

Interpretation of entry $(f, t, C, h, d)$:

- Entry catches exceptions of class $\preceq^* C$ throw in code range $f$ to $t$.
- Handler code starts at instruction $h$ and needs operand stack of height $d+1$.

Expected invariant: operand stack must have height $\geq d+1$ when the exception is caught.

# Exception handling

$$
\begin{aligned}
&\mathit{exec}\ P\ (\mathit{None}, h, (\mathit{stk}, \mathit{loc}, C, M, \mathit{pc}) \cdot \mathit{frs}) = \\
&\lfloor \mathit{let}\ i = (\mathit{instrs\text{-}of}\ P\ C\ M)_{[pc]};\\
&\quad (\mathit{xp}', h', \mathit{frs}') = \mathit{exec\text{-}instr}\ i\ P\ h\ \mathit{stk}\ \mathit{loc}\ C\ M\ \mathit{pc}\ \mathit{frs}\\
&\ \mathit{in\ case}\ \mathit{xp}'\ \mathit{of}\ \mathit{None} \Rightarrow (\mathit{None}, h', \mathit{frs}')\\
&\quad \mid \lfloor a \rfloor \Rightarrow \mathit{find\text{-}handler}\ P\ a\ h\ ((\mathit{stk}, \mathit{loc}, C, M, \mathit{pc}) \cdot \mathit{frs}) \rfloor
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{find\text{-}handler} :: \\
&\quad \mathit{jvm\text{-}prog} \Rightarrow \mathit{addr} \Rightarrow \mathit{heap} \Rightarrow \mathit{frame\ list} \Rightarrow \mathit{jvm\text{-}state}
\end{aligned}
$$

# Find handler

Exception: $h\ a = \lfloor (C, \_) \rfloor$

- Search frames from left to right, popping them.
- In each frame $(stk, loc, C_0, M, pc)$ search exception table of $M$ from left to right.
- An entry $(f, t, D, h, d)$ catches the exception iff $f \leq pc \wedge pc < t \wedge P \vdash C \preceq^* D$.

  In that case replace the frame by $(Addr\ a \cdot drop\ (|stk| - d)\ stk, loc, C, M, h)$.
- If no catching entry is found in any frame: stop in state $(\lfloor a \rfloor, h, [])$.

# *A defensive JVM*

# *Aggressive versus defensive*

- *exec-instr* models *aggressive* execution:

  Computation cannot get stuck
  because all functions are total.

  But new state sometimes underdefined:
  what does Pop do if $stk = []$?

- *check-instr* $::$ *instr* $\Rightarrow \ldots \Rightarrow$ *bool* (to come)
  models *defensive* execution:

  checks if instruction is “safe” to execute in current state.

# *Why defensive execution*

- Agressive execution models the real implementation: no runtime checks, eg if *stk* is [].
- Defensive execution models potential problems, eg buffer underflow!

We shall show:
For well-typed bytecode, *check-instr* is always true when *exec-instr* is called.

The *raison d'être* for the defensive model:
To define precisely which conditions we do *not* need to check at runtime.

# check-instr

*check-instr* ::
*instr* $\Rightarrow$ *jvm-prog* $\Rightarrow$ *heap* $\Rightarrow$ *opstack* $\Rightarrow$ *registers*
$\Rightarrow$ *cname* $\Rightarrow$ *mname* $\Rightarrow$ *pc* $\Rightarrow$ *frame list* $\Rightarrow$ *bool*

Same arguments as *exec-instr*

# Some easy checks (1)

$$\mathit{check\text{-}instr}\ (\texttt{Load}\ n)\ P\ hp\ stk\ loc\ C\ M_0\ pc\ frs = (n < |loc|)$$

$$\mathit{check\text{-}instr}\ (\texttt{Store}\ n)\ P\ hp\ stk\ loc\ C_0\ M_0\ pc\ frs =$$

$$(0 < |stk| \wedge n < |loc|)$$

$$\mathit{check\text{-}instr}\ (\texttt{Push}\ v)\ P\ hp\ stk\ loc\ C_0\ M_0\ pc\ frs = (\neg\ \mathit{is\text{-}Addr}\ v)$$

$$\mathit{check\text{-}instr}\ \texttt{Pop}\ P\ hp\ stk\ loc\ C_0\ M_0\ pc\ frs = (0 < |stk|)$$

$$\mathit{check\text{-}instr}\ (\texttt{New}\ C)\ P\ hp\ stk\ loc\ C_0\ M_0\ pc\ frs = \mathit{is\text{-}class}\ P\ C$$

# Some easy checks (2)

$$\mathit{check\text{-}instr}\ \texttt{IAdd}\ P\ hp\ stk\ loc\ C_0\ M_0\ pc\ frs =$$
$$(1 < |stk| \wedge \mathit{is\text{-}Intg}\ (hd\ stk) \wedge \mathit{is\text{-}Intg}\ (hd\ (tl\ stk)))$$

$$\mathit{check\text{-}instr}\ \texttt{CmpEq}\ P\ hp\ stk\ loc\ C_0\ M_0\ pc\ frs = (1 < |stk|)$$

$$\mathit{check\text{-}instr}\ (\texttt{Goto}\ b)\ P\ hp\ stk\ loc\ C_0\ M_0\ pc\ frs =$$
$$(0 \leq \mathit{int}\ pc + b)$$

$$\mathit{check\text{-}instr}\ (\texttt{IfFalse}\ b)\ P\ hp\ stk\ loc\ C_0\ M_0\ pc\ frs =$$
$$(0 < |stk| \wedge \mathit{is\text{-}Bool}\ (hd\ stk) \wedge 0 \leq \mathit{int}\ pc + b)$$

$$\mathit{check\text{-}instr}\ \texttt{Throw}\ P\ hp\ stk\ loc\ C_0\ M_0\ pc\ frs =$$
$$(0 < |stk| \wedge \mathit{is\text{-}Ref}\ (hd\ stk))$$