$\mathcal{JVML}_{00}$

$\mathcal{JVML}_{00}$ formalizes a subset of the possible instructions of a method in a class A, with argument types t_1, ... t_l, result type t, and

$$\mathsf{Stack} = \mathsf{l}_1, \mathsf{Locals} = \mathsf{l}_2, \mathsf{ArgsSize} = \mathsf{l}_3.$$

Syntax

$$\textit{instruction} \quad ::= \quad \mathsf{inc} \mid \mathsf{pop} \mid \mathsf{store}\ x \mid \mathsf{load}\ x \mid \mathsf{if}\ L \mid \mathsf{halt}$$

where x is the number of a local variable,
L is the label of some instruction.

Operational Semantics

$\mathsf{P} : 0..\mathrm{k} \longrightarrow$ *instruction* stands for the program.
The value of k is   .

pc is the program counter

$f : 0..\mathrm{j} \longrightarrow$ *integer* describes the contents of local variables, and *integer* represents both integers and addresses.
The value of j is   .

s is a list of values, and stands for the operand stack. The symbol $\cdot$ represents concatenation.
The maximal length of the list is   .

Execution has the format $\mathsf{P} \vdash pc, f, s \rightsquigarrow pc', f', s'$

$$\frac{\mathsf{P}[pc] = \mathsf{inc}}{\mathsf{P} \vdash pc, f, n{\cdot}s \rightsquigarrow pc{+}1, f, (n{+}1){\cdot}s}
\qquad
\frac{\mathsf{P}[pc] = \mathsf{pop}}{\mathsf{P} \vdash pc, f, v{\cdot}s \rightsquigarrow pc{+}1, f, s}$$

$$\frac{\mathsf{P}[pc] = \mathsf{load}\ x}{\mathsf{P} \vdash pc, f, s \rightsquigarrow pc{+}1, f, f(x){\cdot}s}
\qquad
\frac{\mathsf{P}[pc] = \mathsf{store}\ x}{\mathsf{P} \vdash pc, f, v{\cdot}s \rightsquigarrow pc{+}1, f[x \mapsto v], s}$$

$$\frac{\mathsf{P}[pc] = \mathsf{if}\ L}{\mathsf{P} \vdash pc, f, O{\cdot}s \rightsquigarrow pc{+}1, f, s}
\qquad
\frac{\mathsf{P}[pc] = \mathsf{if}\ L, \quad n \neq 0}{\mathsf{P} \vdash pc, f, n{\cdot}s \rightsquigarrow L, f, s}$$

Question: Why is there no operational semantics rule for halt?

The Type System

t $\in$*Type* = { int } $\cup$ { A $\mid$ A the name of a class }

We introduce type vectors $(\mathit{Type})^*$, where the symbol $\cdot$ indicates concatenation, *e.g.* int$\cdot$A$\cdot$int is such a type vector. We have F_i, S_i:

F_i : (*Type*)^n stands for types of local variables as at instruction i.
For example, $F_3[2]$ is the type of the 2nd local variable at instruction 3.
The value of n is .

S_i : (*Type*)^m stands for types of stack operands as at instruction i.
For example, $S_3[2]$ is the type of the 2nd operand at instruction 3.
The value of m is .

The range of i is .

The judgment

$$F, S, \mathsf{i} \vdash \mathsf{P}$$

asserts that the *i*-th instruction of P is well-formed with respect to *F* and *S*. The judgment is defined on next slide.

The judgment

$$F, S \vdash \mathsf{P}$$

guarantees that program P is well-formed with respect to *F* and *S*, *i.e.* that all its instructions are well-formed with respect to *F* and *S*, *i.e.* :

$$\frac{\forall i \in Dom(\mathsf{P}) : \quad F, S, \mathsf{i} \vdash \mathsf{P}}{F, S \vdash \mathsf{P}}$$

$$\frac{\mathsf{P}[i] = \mathsf{inc} \quad F_{i+1} = F_i \quad S_{i+1} = S_i = \mathsf{int}\cdot\alpha \quad i+1 \in Dom(\mathsf{P})}{F, S, \mathsf{i} \vdash \mathsf{P}}$$

$$\frac{\mathsf{P}[i] = \mathsf{if}\ L \quad F_{i+1} = F_L = F_i \quad \mathsf{t}\cdot S_{i+1} = \mathsf{t}\cdot S_L = S_i \quad i+1 \in Dom(\mathsf{P}),\ L \in Dom(\mathsf{P})}{F, S, \mathsf{i} \vdash \mathsf{P}}$$

$$\frac{\mathsf{P}[i] = \mathsf{pop} \quad F_{i+1} = F_i \quad \mathsf{t}\cdot S_{i+1} = S_i \quad i+1 \in Dom(\mathsf{P})}{F, S, \mathsf{i} \vdash \mathsf{P}}$$

$$\frac{\mathsf{P}[i] = \mathsf{load}\ x \quad x \in Dom(F_i) \quad F_{i+1} = F_i \quad S_{i+1} = F_i[x]\cdot S_i \quad i+1 \in Dom(\mathsf{P})}{F, S, \mathsf{i} \vdash \mathsf{P}}$$

$$\frac{\mathsf{P}[i] = \mathsf{store}\ x \quad x \in Dom(F_i) \quad F_{i+1} = F_i[x \mapsto \mathsf{t}] \quad \mathsf{t}\cdot S_{i+1} = S_i \quad i+1 \in Dom(\mathsf{P})}{F, S, \mathsf{i} \vdash \mathsf{P}}$$

$$\frac{\mathsf{P}[i] = \mathsf{halt}}{F, S, \mathsf{i} \vdash \mathsf{P}}$$

Note that

- F_{i+1} in general depends on F_i.

- S_{i+1} in general depends on S_i.

- All instructions, except for halt, expect the next instruction to be defined in P.

- All instructions, except for store, leave F_i unmodified.

Also, note that type rules define both *type checking* and *type inference* (or combinations):

A *type checker* is given a program with type declarations, and checks whether it is type correct. E.g., Java compilers so type checking.
In this case, a type checker would be given F, S and P, and would check whether $F, S \vdash \mathsf{P}$ holds.

A *type inference* system is given a program without type declarations, and tries to find type declarations that would make the program type correct. E.g., Haskell compilers do type inference.
In this case, a type inference system would be given P, and would try to find F and S such that $F, S \vdash \mathsf{P}$ holds.

The Java verifier is something in between, because it is given
and tries to construct so that $F, S \vdash \mathsf{P}$ holds.

Guarantees given by the type system

Theorem If $F, S \vdash$P and $i \in Dom($P$)$, and f , and s , then:

- **Type preservation** If P $\vdash i, f, s \;\leadsto\; i', f', s'$ then , and , and .

- **Progress** If there do not exist i', f', s', such that P $\vdash i, f, s \;\leadsto\; i', f', s'$, then .

Type rules with subtypes

We extend types to contain a type $\top$, which denotes *any* type:

$$\mathsf{t} \in \mathit{Type} = \{ \ \mathsf{int} \ \} \cup \{ \ \top \ \} \cup \{ \mathsf{A} \mid \mathsf{A} \text{ the name of a class } \}$$

We extend the subtype relationship for classes

$$\frac{\mathsf{t} \text{ a subclass of } \mathsf{t}' \text{ in } \mathsf{P}}{\mathsf{P} \vdash \mathsf{t} \leq \mathsf{t}'} \qquad \frac{\mathsf{t} \text{ is a type in } \mathsf{P}}{\mathsf{P} \vdash \mathsf{t} \leq \top}$$

We extend the subtype relationship to type vectors

$$\frac{\mathsf{P} \vdash \mathsf{t}_j \leq \mathsf{t}'_j \qquad \forall \mathsf{j} \in 1, ..., \mathsf{m}}{\mathsf{P} \vdash \mathsf{t}_1 \cdot ... \cdot \mathsf{t}_m \leq \mathsf{t}'_1 \cdot ... \cdot \mathsf{t}'_m}$$

$$\frac{\begin{array}{c}\mathsf{P}[i]{=}\mathsf{inc}\\ \mathsf{P}\vdash F_i \leq F_{i+1}\\ \mathsf{P}\vdash S_i \leq S_{i+1}, \quad \exists S : with S_i = \mathsf{int}\cdot S\\ i+1 \in Dom(\mathsf{P})\end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}}$$

$$\frac{\begin{array}{c}\mathsf{P}[i]{=}\mathsf{if}\ L\\ \mathsf{P}\vdash F_i \leq F_{i+1},\ \mathsf{P}\vdash F_i \leq F_L\\ \mathsf{P}\vdash S_i \leq \mathsf{t}\cdot S_{i+1},\ \mathsf{P}\vdash S_i \leq \mathsf{t}\cdot S_L\\ i+1 \in Dom(\mathsf{P}),\ L \in Dom(\mathsf{P})\end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}}$$

$$\frac{\begin{array}{c}\mathsf{P}[i]{=}\mathsf{pop}\\ \mathsf{P}\vdash F_i \leq F_{i+1}\\ \mathsf{P}\vdash S_i \leq \mathsf{t}\cdot S_{i+1}\\ i+1 \in Dom(\mathsf{P})\end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}}$$

$$\frac{\begin{array}{c}\mathsf{P}[i]{=}\mathsf{load}\ x\\ x \in Dom(F_i)\\ \mathsf{P}\vdash F_i \leq F_{i+1}\\ \mathsf{P}\vdash F_i[x]\cdot S_i \leq S_{i+1}\\ i+1 \in Dom(\mathsf{P})\end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}}$$

$$\frac{\begin{array}{c}\mathsf{P}[i]{=}\mathsf{store}\ x\\ x \in Dom(F_i)\\ \mathsf{P}\vdash F_i[x \mapsto \mathsf{t}] \leq F_{i+1}\\ \mathsf{P}\vdash S_i \leq \mathsf{t}\cdot S_{i+1}\\ i+1 \in Dom(\mathsf{P})\end{array}}{F, S, \mathsf{i} \vdash_s \mathsf{P}}$$

$$\frac{\mathsf{P}[i]{=}\mathsf{halt}}{F, S, \mathsf{i} \vdash_s \mathsf{P}}$$

As before, the judgement $F, S \vdash_s$ P guarantees that program P is well-formed with respect to ***F*** and **S**, and is defined as follows:

$$\frac{\forall i \in Dom(\mathsf{P}) : \quad F, S, \mathsf{i} \vdash_s \mathsf{P}}{F, S \vdash_s \mathsf{P}}$$

- In general, $\mathsf{P} \vdash F_i \leq F_{i+1}$, *i.e.* the types of local variables may be less precise at next instruction.

- In general, $\mathsf{P} \vdash S_i \leq S_{i+1}$, *i.e.* the types of stack may be less precise at next instruction.

- The task of the verifier can be understood as the search for appropriate F and S, for given P and F_1, S_1, so that $F, S \vdash_s$ P.