A Program Logic for Bytecode[1]

Fabian Bannwart Peter Müller

Presented by Moritz Hoffmann

April 15, 2013

Objective

A sound and complete Hoare-style logic to apply Proof-Carrying Code on bytecode.

Motivation

- Intermediate languages are part of standardized execution environments, i.e. JVM and .NET.
- Formal reasoning on source level
- Improve and speed up JIT

Proof-carrying code

- Translate verified source code to verified bytecode
- Annotate intermediate language with proofs
- Efficient run-time verification of proof carrying code

Problem

- Code is compiled to intermediate language
- Source proof must also be transformed

Goal

Develop proof-transforming compiler

Bytecode Language

Bytecode language $\mathrm{VM_K}$

- Used to model classes, methods and instructions
- No exception handling
- Programs are well typed
- Object Store models heap
- Stack
- Similar to JVM and CLI bytecode instructions

Hoare-style rules for every included instruction

Program Logic

[The program logic] allows to formally verify that implementations satisfy interface specifications given as pre- and postconditions.

Method specification $\{P\}$ comp $\{Q\}$

Instruction specification $\mathcal{A} \vdash \{E_i\}\ i : I_i$

Method Sequence of instruction specifications
$\forall i \in \{0, \ldots, |body(T@m)| - 1\} : (\mathcal{A} \vdash \{E_i\}\ i : I_i)$

$$\frac{\forall i \in \{0, \ldots, |body(T@m)| - 1\} : (\mathcal{A} \vdash \{E_i\}\ i : I_i)}{\mathcal{A} \vdash \{E_o\}\ body(T@m)\ \{E_{|body(T@m)|-1}\}}$$

Method Specification

- Method specification: $\{P\}$ comp $\{Q\}$
- `comp` is a method implementation $T@m$ of a virtual interface $T : m$
- Support for virtual methods
- Contains language independent rules to connect method specifications to programming logic

Instruction Specification

- Instruction specification: $\{E_i\}\ i : I_i$
- $\{E_i\}$ is the local weakest precondition
- *shift* and *unshift* model stack operations.

I_i	$\mathtt{wp}^1_{\mathtt{p}}(I_i)$
`pushc` v	$unshift(E_{i+1}[v/s(0)])$
`pop` x	$(shift(E_{i+1}))[s(0)/x]$
$\mathtt{binop}_{\mathtt{op}}$	$(shift(E_{i+1}))[s(1)\ \mathtt{op}\ s(0))/s(1)]$

- Other operations: `pushv`, `goto`, `brtrue`, `checkcast`, `newobj`, `getfield`, `putfield`, `return`

Application

$$\{p = P\}\mathsf{Math}:\mathsf{abs}$$
$$\{(P \geq 0 \Rightarrow \texttt{result} = P) \wedge (P < 0 \Rightarrow \texttt{result} = -P)\}$$

Since $\mathsf{Math} : \mathsf{abs}$ is defined in a class without subclasses, we can apply the following rule:

$$\frac{\{p = P \wedge \tau(\texttt{this}) = \mathsf{Math} \wedge \texttt{this} \neq \texttt{null}\}\ body(Math@abs)\ \{Q\}}{\{p = P \wedge \tau(\texttt{this}) = \mathsf{Math}\}\ Math@abs\ \{Q\}}$$

Impact

- Development of proof-transforming compiler producing proof-carrying code
- Foundation to understanding complication of `break` and try/catch/finally clauses

Remaining issue

- Only one method parameter p is covered by the logic.
- Logic does not handle type checking

Proof Math : abs

$$\{p = P \wedge \tau(\mathtt{this}) = \mathtt{Main} \wedge \mathtt{this} \neq \mathtt{null}\} 0 : \mathtt{pushv}\ p$$
$$\{(s(0) < 0 \Rightarrow P < 0) \wedge (s(0) \geq 0 \Rightarrow P \geq 0) \wedge p = P\} 1 : \mathtt{pushc}\ 0$$
$$\left\{\begin{array}{r}(s(1) < s(0) \Rightarrow P < 0) \\ \wedge (s(1) \geq s(0) \Rightarrow P \geq 0) \wedge p = P\end{array}\right\} 2 : \mathtt{binop}_{\geq}$$
$$\{(s(0) < 0 \Rightarrow P < 0) \wedge (s(0) \geq 0 \Rightarrow P \geq 0) \wedge p = P\} 3 : \mathtt{brtrue}\ 8$$
$$\{P < 0 \wedge p = P\} 4 : \mathtt{pushc}\ 0$$
$$\{P < 0 \wedge s(0) - p = -P\} 5 : \mathtt{pushv}\ p$$
$$\{P < 0 \wedge s(1) - s(0) = -P\} 6 : \mathtt{binop}_{-}$$
$$\{P < 0 \wedge s(0) = -P\} 7 : \mathtt{goto}\ 9$$
$$\{P \geq 0 \wedge p = P\} 8 : \mathtt{pushv}\ p$$
$$\{(P \geq 0 \Rightarrow s(0) = P) \wedge (P < 0 \Rightarrow s(0) = -P)\} 9 : \mathtt{pop\ result}$$
$$\{(P \geq 0 \Rightarrow \mathtt{result} = P) \wedge (P < 0 \Rightarrow \mathtt{result} = -P\} 10 : \mathtt{return}$$

Transformation of Source Proofs

$$\mathcal{S}\left(\frac{\frac{T}{\{e \wedge P\}\ S\ \{P\}}}{\{P\}\ \texttt{while}(e)S\ \{\neg e \wedge P\}}\right) =$$

$$\begin{array}{rc}
\{P\}l_1 : & \texttt{goto}\ l_3 \\
\{e \wedge P\}l_2 : & \mathcal{S}\left(\frac{T}{\{e \wedge P\}\ S\ \{P\}}\right) \\
\{P\}l_3 : & \mathcal{S}_E(P, e) \\
\{shift(P) \wedge s(0) = e\}l_4 : & \texttt{brtrue}\ l_2 \\
\{P \wedge \neg e\} &
\end{array}$$

For Further Reading I

Fabian Bannwart and Peter Müller.
A program logic for bytecode.
Electronic Notes in Theoretical Computer Science, 141(1):255–273, 2005.