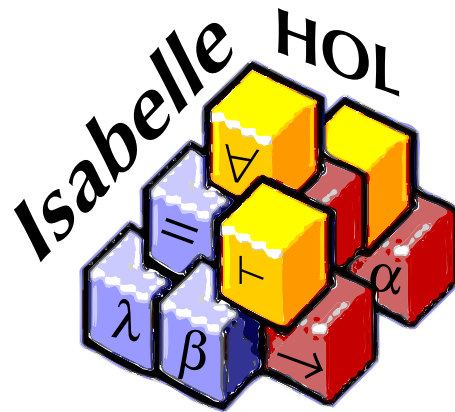

Verified Lightweight Bytecode Verification

Gerwin Klein and Tobias Nipkow
TU München



in part supported by DFG, project Bali

<http://isabelle.in.tum.de/Bali/>

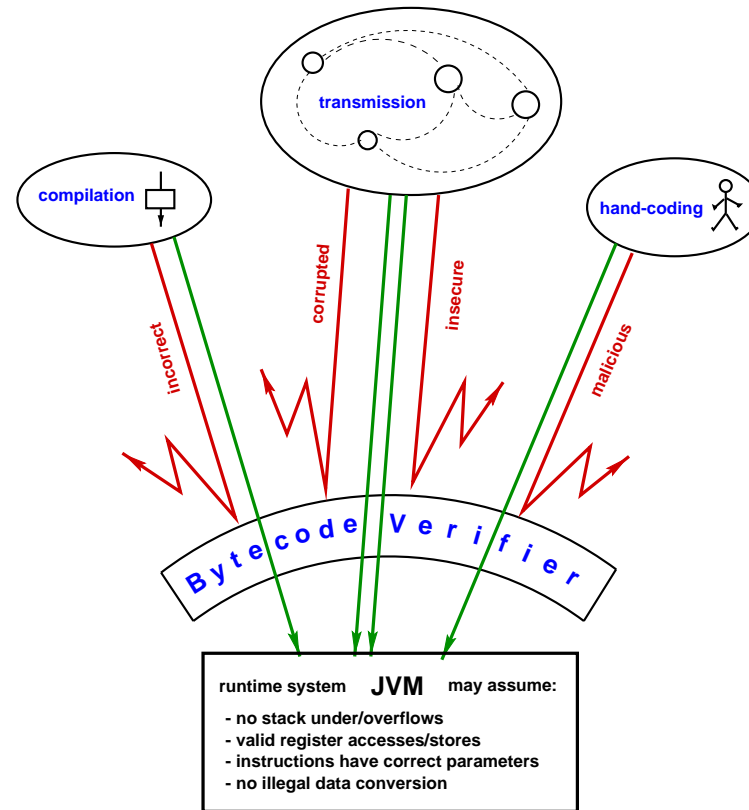
Roadmap

- μ Java
- Bytecode Verification
- Lightweight Bytecode Verification
- Verified Lightweight Bytecode Verification

μ Java

- Java embedded in Isabelle/HOL
- Everything left out but classes
- Formalizing Java, Bytecode, JVM, Bytecode Verifier, Compiler ...

Bytecode Verification



Bytecode Verification

Sun's JVM Spec

- **Iterative** process to collect type information of opstack prior to execution
- Describes **implementation** of the Bytecode Verifier

μ Java's BV Spec

- **Specification** of the Bytecode Verifier
- **checks** if a JVM program satisfies the given **static type** information
- Based on work by Zhenyu Qian

Order on Types

Supertype relation

$$G \vdash a' \quad \preceq_o \text{ Unusable} = \text{True}$$

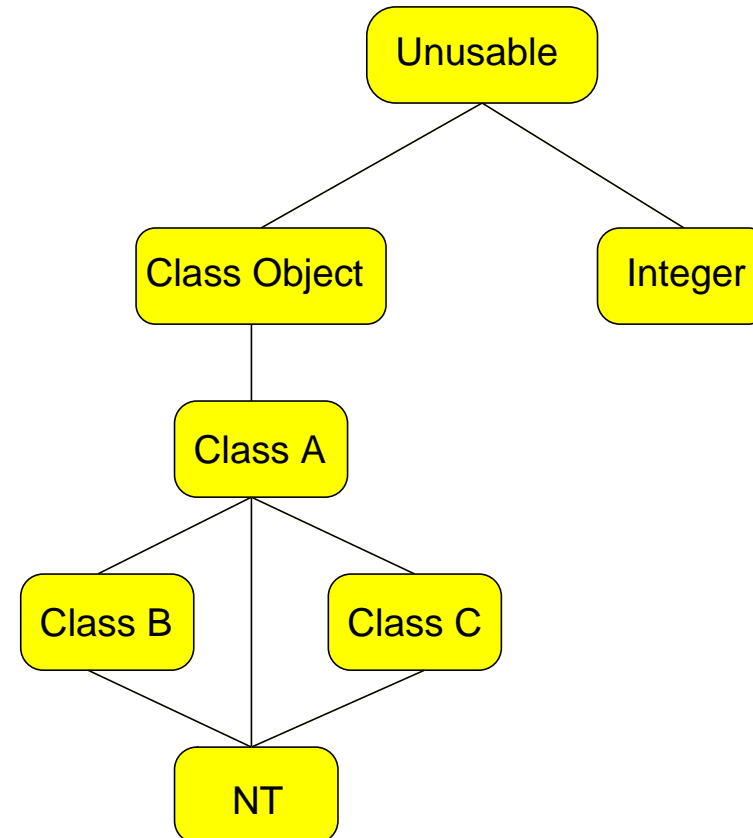
$$G \vdash \text{Unusable} \quad \preceq_o \text{ Usable } t = \text{False}$$

$$G \vdash \text{Usable } t' \quad \preceq_o \text{ Usable } t = G \vdash t' \preceq t$$

Widen relation

$$G \vdash t \quad \preceq t$$

$$G \vdash \text{NT} \quad \preceq \text{RefT } R$$

$$G \vdash \text{Class } B \preceq \text{Class } A, \text{ if } B \text{ subclass of } A$$


An Example

<i>pc</i>	<i>instr</i>	<i>opstack</i>	<i>locvar₀, locvar₁</i>
0	Load 0	[]	Class A, Class B
1	Aconst_null	[Class A]	Class A, Class B
2	Ifcmpeq +6	[NT, Class A]	Class A, Class B
3	Load 0	[]	Class A, Class B
4	Invoke i n	[Class A]	Class A, Class B
⋮	⋮	⋮	⋮
8	Load 1	[]	Class A, Class B
9	Goto -5	[Class B]	Class A, Class B

BV Specification

Load Instruction

$$\begin{aligned}
 & \text{wt_LS (Load } idx) \Gamma \varphi \text{ max_pc pc} = \\
 & \text{let } (ST, LT) = \varphi ! pc \text{ in} \\
 & \quad pc+1 < \text{max_pc} \wedge \\
 & \quad idx < \text{length } LT \wedge \\
 & \quad (\exists t. (LT ! idx) = \text{Usable } t \wedge \\
 & \quad \quad \Gamma \vdash (t \# ST, LT) \preceq_s \varphi ! (pc+1))
 \end{aligned}$$

idx	:	int	(index of local variable to load onto the stack)
Γ	:	jvm_prog	(class hierarchy)
φ	:	$method_type$	(static type information of the method)
max_pc	:	int	(number of instructions in the method)
pc	:	int	(current program counter)

Lightweight Bytecode Verification

Motivation

- Bytecode Verifiers are **large, complex** programs
- runtime **space** $\mathcal{O}(\max_pc \cdot (\max |opstack| + |locvar|))$
- do not fit on **Java Smartcards**

Consequently current Java Smartcards do not support dynamic reprogramming or rely on cryptographic techniques to ensure typesafe bytecode.

Lightweight Bytecode Verification

Idea

- Provide the Bytecode Verifier with **additional information** c
- Use this to achieve **linear space** $\mathcal{O}(c)$ and **linear time**

Concept

- George Necula's Proof Carrying Code
- Applied to Bytecode Verification by Eva and Christopher Rose

An Example

<i>pc</i>	<i>instr</i>	certificate	
0	Load 0		
1	Aconst_null		
2	Ifcmpeq 6		
3	Load 0		
4	Invoke i n	[Class A]	[Class A, Class B]
:	:		
8	Load 1	[]	[Class A, Class B]
9	Goto -5		

Load Instruction

$$\begin{aligned} \text{wtl_LS (Load } idx) s s' \text{ max_pc pc} = \\ \text{let } (ST, LT) = s \text{ in} \\ pc+1 < \text{max_pc} \wedge idx < \text{length } LT \wedge \\ (\exists t. (LT ! idx) = \text{Usable } t \wedge \\ (t \# ST, LT) = s') \end{aligned}$$

Certificate

$$\begin{aligned} \text{wtl_inst_option } i \Gamma rT s s' \text{ cert max_pc pc} = \\ \text{case cert ! pc of None} \quad \rightarrow \text{wtl_inst } i \Gamma rT s s' \text{ cert max_pc pc} \\ | \text{Some } s'' \rightarrow (\Gamma \vdash s \preceq_s s'') \wedge \text{wtl_inst } i \Gamma rT s'' s' \text{ cert max_pc pc} \end{aligned}$$

i : current instruction
s : current state_type
s' : next state_type
cert : certificate for the method

max_pc : number of instructions in the method
pc : current program counter
 Γ : class hierarchie
rT : return type of the method

Verified Lightweight Bytecode Verification

Correctness

Lightweight Bytecode Verification is **sound** and **tamper proof**
(*LBV*) $\text{wtl_jvm_prog } \Gamma \text{ cert} \implies \exists \Phi. \text{wt_jvm_prog } \Gamma \Phi$ (*BV*)

Proof

- construct a Φ from *cert* and show that it satisfies *wt_jvm_prog*, both
- by induction on the list of instructions
- by case distinction over the single instructions

Difficult part: Branches

Verified Lightweight Bytecode Verification

Completeness

Lightweight Bytecode Verification is **complete**

$$(BV) \quad wt_jvm_prog \Gamma \Phi \implies \exists cert. wtl_jvm_prog \Gamma cert \quad (LBV)$$

Proof

- construct *cert* by specifying a **Certifying Bytecode Verifier**
- prove that *wtl_jvm_prog* holds for this *cert*

Not yet done in Isabelle

Conclusion

**Lightweight Bytecode Verification is as good
as traditional Bytecode Verification**

It is a true alternative to trust based cryptographic methods

Statistics

- Specification: 300 lines
- Soundness proof: 450 lines
- Total efforts so far: 2.5 months

Still to do

- Formal proof of completeness
- Deriving an implementation from the specification