

Java Bytecode Verification Framework for Complex Policies

Mariela Pavlova

INRIA, Sophia-Antipolis, France

Outline

- 1 Context**
- 2 New approach for bytecode verification
- 3 Bytecode verification scheme for complex policies
 - Bytecode Modeling Language
 - Compiler from JML to BML
 - Verification condition generator
 - Relation for proof obligation on source and bytecode
- 4 Conclusion. Towards a PCC for complex policies

Context

- Security becomes part of the overall software quality especially today when software applications enter into our daily life
- Source code validation - a necessary stage in the software development

Java source code verification

- Java Modeling Language
 - method pre and postconditions
 - loop invariants
 - class invariants, history constraints
 - assertions at particular program point
 - special keywords for denoting the method result, the value of a variable in the method prestate
- Verification condition generator.
 - the Loop tool
 - esc/java
 - Jack
 - Key
 - Krakatoa
- decision procedure.
 - Interactive procedures - Coq, PVS
 - automated procedures - Simplify

Example

Example

sum of the natural numbers smaller than k

```
// @requires k >= 0 ;
// @ensures result == k*(k+1)/2;
public void m(int k){
    int sum = 0;
    // @loop_modifies sum, i;
    // @loop_invariant i >= 0
    //                && i <=k && sum == i*(i-1)/2;
    for (int i = 0; i < k; i++){
        sum = sum + i;
    }
}
```

What is source validation good for?

- Source validation useful for the software development process
- the aforementioned scheme can deal with complex functional requirements
- However, for scenarios like mobile code, source validation is not appropriate

Mobile code scenarios

Mobile code scenarios

Two sides

- code producer - develops an application
- code client - receives the executable code but he potentially may not trust the code. How will the client assure that the code does not his internal invariants?
- source verification not appropriate

Existing solutions

Traditional PCC

- the code producer -
 - develops an application and builds an evidence for its correctness
 - ships the applications and the evidence to the client
- the code client -
 - generates verification conditions over the application
 - checks that the evidence is a certificate for the generated verification conditions

Proof Carrying Code (PCC)

traditional PCC

Traditional approach

- annotations inferred automatically
- proof inferred automatically

traditional PCC is good for

may check for properties like well - typedness, safe read -write memory access

but

cannot deal with complex functional properties

Outline

- 1 Context
- 2 New approach for bytecode verification**
- 3 Bytecode verification scheme for complex policies
 - Bytecode Modeling Language
 - Compiler from JML to BML
 - Verification condition generator
 - Relation for proof obligation on source and bytecode
- 4 Conclusion. Towards a PCC for complex policies

Motivations

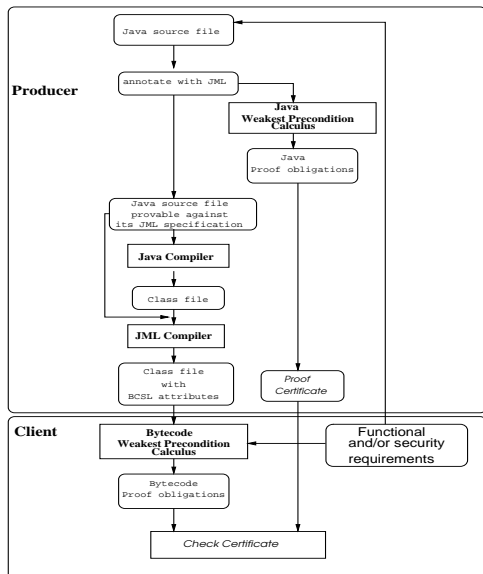
a bytecode verification framework which

- benefits from source verification techniques
- thus, suitable for complex security and functional properties

Components of a bytecode verification framework

- Bytecode Modeling Language (BML)
- Compiler from JML to BML
- verification condition generator
- Relation with source verification

Architecture



Outline

- 1 Context
- 2 New approach for bytecode verification
- 3 Bytecode verification scheme for complex policies**
 - Bytecode Modeling Language
 - Compiler from JML to BML
 - Verification condition generator
 - Relation for proof obligation on source and bytecode
- 4 Conclusion. Towards a PCC for complex policies

Syntax and Semantics of BML

- follows closely the semantics of JML
- stack expressions - necessary for specifying properties in intermediate program points of methods

Encoding of BML

Features

- Java compiler independence
- does not violate the JVM performance
- correspond to a desugared version of JML

Why a compiler from JML to BML?

other approaches for writing bytecode specification

- automatic inference of annotation not decidable especially for complex policies.
- writing manually specifications for bytecode not easy

our approach

make the Java bytecode benefit from the source specification

Compiler

Takes as input

- a source file specified with JML
- the corresponding class file supplied for every method with **Local_Variable_Table** and **Line_Number_Table**

Stages

- desugaring of JML
- linking
- find the places to which an inter method specification must hold
- compiling specification into userdefined attributes in the class file

Produces

• a class file containing attributes with JML specifications

Example. Annotated bytecode of the method sum

Example

```
// @requires reg(1)>=0
0 const 0
1 store 2
2 const 0
3 store 3
4 goto 10
5 load 2
6 load 3
7 add
8 store 2
9 iinc 3 //LOOP END
// @loop_modifies reg(2), reg(3)
// @loop_invariant reg(3)>=0 && reg(3)<=reg(1)&&
    reg(2)==reg(3)*(reg(3)-1)/2
10 load 3 //LOOP ENTRY
11 load 1
12 if_icmplt 5
13 return
// @ensures \ result==reg(1)*(reg(1)+1)/2
```

Verification condition generator. Design

- covers the most important feature of Java - object manipulation and creation, exceptions, method invocations, arithmetic, stack manipulation etc.
- contract based approach
- uses frame conditions for initialising properly non - modified locations in loops
- works on reducible control flow graphs
- proven sound under the hypothesis that the control flow graph is reducible

Relation for proof obligation on source and bytecode

holds if

- the compiler is non optimizing
- the compiler has certain properties like : reducible graphs, loop and exception handler preservation, statements

Equivalence modulo

- names - Java names are compiled into indexes of the constant pool or elements in the method local variable table
- types - Java basic types integer, short, byte and boolean are compiled in the bytecode into integers

How can it be used?

Context

- complex security or functional requirements
- verification on source code easier and allows correcting bugs or errors, correcting or completing annotation
- the bytecode certificate then can be constructed from the source certificate

Outline

- 1 Context
- 2 New approach for bytecode verification
- 3 Bytecode verification scheme for complex policies
 - Bytecode Modeling Language
 - Compiler from JML to BML
 - Verification condition generator
 - Relation for proof obligation on source and bytecode
- 4 Conclusion. Towards a PCC for complex policies**

What has been done upto now?

- the BML language
- the equivalence between source and bytecode verification conditions
- implementation in Jack the bytecode verification condition generator and the JML2BML compiler

To do

- encoding of the proof certificate
- a smaller verification condition generator, useful for on-device checking
- relation between certificate over source and bytecode programs produced with an optimizing compiler