Java Bytecode Speci cation and Veri cation

Liia B rdy
NR A Sophia A tipolis
Liia B rdy sophia i ria fr

Mariela Pa lo a
NR A Sophia A tipolis
Mariela Pa lo a sophia i ria fr

Septem er

Abstract

We propose a framework for establishing the correctness of untrusted Java bytecode components w.r.t. to complex functional and/or security policies. To this end, we de ne a bytecode speci cation language (BCSL) and a weakest precondition calculus for sequential Java bytecode. BCSL and the calculus are expressive enough for verifying non-trivial properties of programs, and cover most of sequential Java bytecode, including exceptions, subroutines, references, object creation and method calls.

Our approach does not require that bytecode components are provided with their

code is accompanied by a proof for its safety w.r.t. to some safety property and the code receiver has just to generate the verification conditions and type check the proof against them. The proof is generated automatically by the certifying compiler for properties like well typedness or safe memory access. As the certifying compiler is designed to be completely automatic, it will not be able to deal with rich functional or security properties.

We propose a bytecode verification framework with the following features:

- a bytecode specification language and a compiler from source program annotations into bytecode annotations. Thus, bytecode can benefit from the source specification and does not need to be accompanied by its source code.

- verification condition generator over Javabyteco

Java
Weakest Precondition
Calculus

Java
Proof obligations

C *C*

3 Related Work

We now review research works which treat similar problematic.

JVer [8] is a tool for verifying that downloaded Java bytecode programs do not abuse client computational resources. The bytecode programs are annotated with pre and postconditions written in a subset of JML speci cation language. The tool, however, doesnot support a compiler from high level speci cation annotations into bytecode annotations.

```
public class ListArray {
  Object[] list;
  //@requires list != null;
  //@ensures \result == (\exists int i;0 <= i &&
              i < list.length && list[i] == o ) ;
  public boolean isElem(Object obj)
  {
    int i = 0;
    //@loop_modifies i;
    //@loop_invariant i <= list.length && i >=0
    //@  && (\forall int k;0 <= k && k < i ==>
    //@  list[k] != obj);
    for (i = 0; i <
```

loop frame condition, which declares the locations that can be modi ed during a loop iteration. We were inspired for this by the JML

the loops in a method are compiled to a unique method attribute: whose syntax is given in Fig. 4. This attribute is an array of data structures eac program functional properties.

The proposed weakest precondition wp supports all Java bytecode sequential instructions except for oating point arithmetic instructions and 64 bit data (long and double types), including exceptions, object creation, references and subroutines. The calculus is de ned over the method control ow graph and supports BCSL annotation, i.e. bytecode method's speci cation like preconditions, normal and exceptional postconditions, class invariants, assertions at particular program point among which loop invariants.

In Fig. 5, we show the wp rules for some bytecode instructions. As the examples show the wp function takes three arguments: the instruction for which we calculate the precondition, the instruction's postcondition and the exceptional postcondition function ^{exc} which for any exception Exc returns the corresponding exceptional postcondition ^{exc}(Exc). The function wp must satisfy the following property: if the instruction ins starts execution in a state where the predicate wp(ins; ; ^{exc}) holds then if it terminates normally then the poststatemust satisfy the predicate and if terminates on exception Exc then the poststate must satisfy ^{exc}(Exc). In the draft paper [16], we show that the wp function has this property (i.e. the calculus is correct). The proof is done by de ning.719d (do.96264 Tf6o776 09.96264 Tf 5era11.88 Td (cTd 1onal)Tj 5sem024 0 T ( )i.e.)Tj 21ics358 0 T

5.1

wp(invoke m

[14] G. Necula. Compiling With Proofs. PhD thesis, Carnegie Mellon University, 1998.

[15] G. C. Necula and P. Lee. The