

Exercise 7

Bytecode Verification

self-study exercise sheet

NOTE: There will not be a regular exercise session on 10th of November, because you will take the midterm exam. Therefore this exercise sheet will NOT be discussed in an exercise session. We publish it now together with the solution to allow you to better prepare for the midterm. If you have any questions regarding this exercise sheet, please consult your assistant.

Task 1

The method `f` of class `E` has the following signature:

```
void f();
```

and one local variable `v`. The maximal stack size is equal to 1.

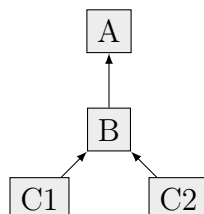
The method `f` has the following body:

```
0: iconst 5
1: istore 1
2: aload 0
3: astore 1
4: iload 1
5: iconst 1
6: iadd
7: istore 1
8: return
```

Can the provided bytecode be verified? If so then verify it, otherwise explain which line of the code causes the problem and why.

Task 2

Consider the following type hierarchy:



Suppose that the method `f` of class `E` has the following signature:

```
A f(boolean b1, boolean b2);
```

and there are three local variables `x`, `y`, `z`. It is known that the initial state is:

```
([], [E, boolean, boolean, C1, C2, A])
```

The maximal stack size is equal to 1.

The method `f` has the following body:

```

0: iload 1
1: ifeq 22
4: iload 2
5: ifeq 12
8: aload 3
9: goto 14
12: aload 4
14: astore 3
15: aload 5
17: astore 4
19: goto 0
22: aload 3
23: areturn

```

Note: In this example, `ifeq x` pops an integer from the stack and jumps to line `x` if the integer is equal to zero.

A) Verify that the program is type safe.

B) Provide the minimal type information that enables the verification of the bytecode, without a fixpoint computation.

Task 3

Consider the following Java code:

```

interface IFace {
    void m();
}
class C11 implements IFace {
    public void m() { System.out.println("C11.m"); }
}
class C12 implements IFace {
    public void m() { System.out.println("C12.m"); }
}
public class Test1 {
    public static void main( String[] args ) {
        foo(true);
        foo(false);
    }
    public static void foo( boolean param ) {
        IFace iface = null;
        if( param ) { iface = new C11(); }
        else { iface = new C12(); }
        iface.m();
    }
}

```

A) What type will be calculated for the variable `iface` of the method `foo` during bytecode verification?

B) When can we decide that `iface.m()` is safe to call, during bytecode verification, or during execution?

C) Would your answer from **B** be the same if `IFace` were a class instead of an interface? What if `IFace` were an abstract class?

Task 4

The bytecode type inference algorithm rejects a verified program if there are different stack sizes for input values of a join point.

A) Provide a bytecode program that is rejected because of this limitation but that does not cause runtime errors.

B) Is it possible to construct a bytecode verification algorithm that avoids this limitation? If yes, then provide an updated algorithm. If no, then show that it can't be done.

C) How serious is this restriction from a pragmatic perspective?

Task 5

The Java bytecode verifier is more permissive than the Java type system. Provide a program that demonstrates this.