Concepts of Object-Oriented Programming
AS 2021

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 8

Bytecode Verification and Parametric Polymorphism

November 19, 2021

Task 1

(from a previous exam)

Assume two Java classes `A` and `B`, where `B` is a subclass of `A`. Consider the following bytecode:

```
0: aload 1
1: astore 2
2: goto 0
```

and assume that the input to the initial node of this code is `([],[A,A,B])`, where the first list indicates the content of the stack and the second list indicates the content of the registers.

After running the bytecode type inference algorithm, what is the inferred input to the initial node?

(a) `([],[A,A,A])`

(b) `([],[A,A,B])`

(c) `([],[A,B,B])`

(d) Nothing is inferred – the type inference does not terminate

(e) Nothing is inferred – the type inference rejects the program

Task 2

Consider the following Java code:

```
interface IFace { void m(); }

class Cl1 implements IFace {
   public void m() { System.out.println("Cl1.m"); }
}
class Cl2 implements IFace {
   public void m() { System.out.println("Cl2.m"); }
}

public class Test {
   public static void main(String[] args) {
      foo(true);
      foo(false);
   }
   public static void foo(boolean param) {
      IFace iface = null;
      if (param) { iface = new Cl1(); }
      else { iface = new Cl2(); }
```

```
        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 3

(from a previous exam)

Consider an *incorrect* bytecode verifier called *BuggyVerifier*, in which due to a bug the `aload` rule assumes that the loaded element is stored at the bottom of the stack instead of at the top (see the formal description below), while all the other rules are implemented correctly.

$$\begin{aligned}&\text{aload } n:\\ &\quad (S, R) \rightarrow (\underline{S.R(n)}, R),\\ &\quad \text{if } 0 \leq n < \mathit{MR} \ \wedge \ R(n) <: \mathit{Object} \ \wedge \ |S| < \mathit{MS}\end{aligned}$$

Assume that the initial state (stack and registers) is $([\,], [A, B])$, with the maximum number of registers $\mathit{MR} = 2$ and the maximum stack size $\mathit{MS} = 2$. A and B are classes such that $B <: A$.

A) Write a short bytecode program that is accepted by *BuggyVerifier*, but is **not** accepted by a correct bytecode verifier. Clearly mark the line at which the correct verifier detects an error, and briefly describe the error.

You can use in your solution all the bytecode operations seen during the lectures. As a reminder, here are some of them:

- `iconst` n: create on the stack a value n of type `int`.
- `iload` n: load on top of the stack an element of type `int` from the n-th register.
- `astore` n: remove an object from the top of the stack and store it in the n-th register.
- `goto` n: continue the execution from the operation at label n.

B) Is it possible that *BuggyVerifier* incorrectly accepts a program that overflows the stack, by pushing more than *MS* elements? Write yes or no, then motivate your answer.

Task 4

A) Compare dynamic type checking with the `dynamic` keyword to static type inference with `var` in C#:

- Give a correct program which can be realized with `dynamic` but not with `var`.
- Give an incorrect program which will be accepted by the compiler with `dynamic` but not with `var`.

B) C#'s most general type is `object`. Similar to `var` and `dynamic`, you can write `object x = ...` with an expression of any type on the right-hand side.

- Given a compiling program using `var`. Can we replace all `var` keywords by `object` and add explicit casts in the right places so that the program compiles and runs as before?
- Given a compiling program using `dynamic`. Can we replace all `dynamic` keywords by `object` and add explicit casts in the right places so that the program compiles and runs as before?

For both questions, either informally describe how to do the replacement, or give a counter-example where the transformation will always produce a program that does not compile or behaves differently. Note that explicit casts to `dynamic` are not allowed in the transformation.

C) Assume now a language like C#, but with covariant return types and contravariant parameter types. Given four classes `A`, `B`, `C` and `D`:

```
class A { int m (int x); }
class B { void m (dynamic x); }
class C { dynamic m (int x); }
class D { dynamic m (dynamic x); }
```

Develop a subtyping rule for the `dynamic` type annotation and informally explain the reasoning behind it. What are the potential subtypes among the four classes above?

Task 5

In this task, you have to implement (using three different approaches) a list in Java that supports the following two methods (where `i` represents an index):

```
public void add(int i, Object el)
public Object get(int i)
```

Discuss the advantages and the limitations of the three different approaches below.

A) Implement the list using only one class without generics.

B) Implement the list using one abstract class/interface and then (some) subclasses that implement it for different types.

C) Implement the list using generic types.

Task 6

(from a previous midterm)

Consider the following Java program, which is rejected by the Java compiler:

```
class Logger<T> {
    public void log(T t) {
        System.out.println(t.loggerString());
    }
}
```

A) If the code above were allowed to compile without errors, what could go wrong? To answer, write a brief code sample that uses `Logger` in a way which causes a failure at runtime.

B) How can we fix the class `Logger` so that it compiles, while preserving its functionality? You should not modify the method `log`, but otherwise can change or add any code. Your solution should include all details required to check that `Logger` is a valid Java class.

C) Assume that class `Logger` has been fixed to resolve the problem from point **A**. Let `A` and `B` be two classes such that `A` is a supertype of `B` and `Logger<A>` and `Logger<B>` are valid instantiations. Consider the following method:

```java
void foo(Logger<A> logA) {
  Logger<B> logB = logA;
  logB.log(new B());
}
```

The Java compiler rejects this code. Is the code safe? That is, if it were allowed to compile, would it run without failure?

D) Suppose we relax the Java type system rules to allow contravariant generics.

- Will the method `foo` compile now?
- What are two situations that will require dynamic checks in order to enable contravariant generics in a language, without limiting what a developer can write in a generic class?

Task 7

(from a previous exam)

A) Recall the Java interface `Comparable<T>` that was shown in the lecture:

```java
public interface Comparable<T> {
    public int compareTo(T other);
}
```

$$\text{The method } \texttt{compareTo} \text{ returns } \begin{cases} 1 & \text{if } \texttt{this} \text{ is greater than } \texttt{other} \\ 0 & \text{if } \texttt{this} \text{ is equal to } \texttt{other} \\ -1 & \text{if } \texttt{this} \text{ is less than } \texttt{other} \end{cases}$$

Suppose we want to turn `Comparable` into an abstract class with an additional helper method `greaterThan`, that returns true if and only if `this` is greater than `other`.

Assume the following implementation:

```java
public abstract class Comparable<T> {
    public abstract int compareTo(T other);

    public boolean greaterThan(T other) {
        return other.compareTo(this) < 0;
    }
}
```

A.1) Why does this implementation not type check?

A.2) Fix the type error by changing only the body of `greaterThan`, while preserving the intended semantics of the method.

A.3) Fix the type error by changing only the class signature and the signature of the method `compareTo`.

B) Suppose we have the following class:

```
class A<X                              ,Y                              > {
    X a;
    Y b;
}
```

Consider a variable `v` whose type is `A<S,T>` where `S` and `T` satisfy the type bounds that you have to insert above. Your type bounds have to guarantee that for all sequences of `a` and `b` accesses on `v` (e.g., `v.b.a.b.a.a.b.b`) the following two properties hold:

- The static type of a sequence ending in `a` is `S`.
- The static type of a sequence ending in `b` is `T`.