Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 13

Self-Study Exercise Sheet

NOTE: 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 final exam. If you have any questions regarding this sheet, please consult your assistant.

Subtyping and Behavioral Subtyping

Task 1

Consider the class `X` and its only method `foo` where `ZZZ` is placeholder for a class name:

```
class X {
   /// requires x>0 ∧ (¬∃ i,j: int | 2 ≤ i,j ≤ x ∧ i*j=x)
   /// ensures result>0 ∧ result % 2 = 0
   int foo(final int x){ return (new ZZZ()).foo(x); }
}
```

Which of the four classes below could be substituted for `ZZZ` such that no contracts will be violated?

(a)
```
class A {
      /// requires x≥0
      /// ensures result = x+1
      int foo(final int x) {...}   }
```

(b)
```
class B {
      /// requires true
      /// ensures result%2 = 0
      int foo(final int x) {...}   }
```

(c)
```
class C {
      /// requires x%2 = 1
      /// ensures result = x+1
      int foo(final int x) {...}   }
```

(d)
```
class D {
      /// requires true
      /// ensures result = x*(x+1)
      int foo(final int x) {...}   }
```

Inheritance, Dynamic Method Binding, Multiple Inheritance, and Linearization

Task 2

Consider the following Java classes and interfaces:

```
public interface IA { IA g(IA x); }

public interface IB extends IA { IB g(IA x); IA g(IB x); }

public interface IC extends IA { IC g(IB x); }

class B implements IB
{
   public IB g(IA x){System.out.print("B1");return null;}
   public IC g(IB x){System.out.print("B2");return null;}
}

class C implements IC
{
   public IC g(IA x){System.out.print("C1");return null;}
   public  C g(IB x){System.out.print("C2");return null;}
}


class Main{
   public static void main(String[] args) {
      B  b  = new B();
      C  c  = new C();
      IA a1 = b;
      IA a2 = c;

      IA r1 = a1.g(a2);
      IA r2 = a2.g(b);
      IC r3 = b.g(b);
      IA r4 = c.g(a2);
      C  r5 = c.g(b);
   }
}
```

What is the output of the execution of the `Main.main` method? Explain your answer.

Task 3

Assume two Java classes `A` and `B` and assume that `B` is a subclass of `A`. Consider the following byte code:

```
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 contents of the stack and the second list indicates the contents 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 4

Consider the following Java code:

```
class Box<T extends Number> {
   private T t;

   public void set(T t) { this.t = t; }
   public T get() { return t; }
}

class Main {
   public static void main(String[] args) {
      Box<Number> b = new Box<________>();
      b.set(new ________);
      ________ c = b.get();
      System.out.println( c );
   }
}
```

and recall that `Integer <: Number <: Object`. How can you fill in the blanks in the `Main.main` method so that the code compiles and executes successfully?

(a) `Integer`, `Integer(9)`, `Integer`

(b) `Integer`, `Integer(9)`, `Object`

(c) `Number`, `Integer(9)`, `Integer`

(d) `Number`, `Integer(9)`, `Object`

(e) None of the above

Task 5

This is an extended version of a previous exam question.

Consider the following Java code:

```
interface Food {}
interface Grass extends Food {}
interface Meat extends Food{}

abstract class Animal<F extends Food> implements Meat{
   abstract void eat(F food);
   F getLunchBag(){ return lunchBag; };
   F lunchBag;
}

final class Sheep extends Animal<Grass>{ void eat(Grass f){} }
final class Wolf extends Animal<Meat>  { void eat(Meat  f){} }

class Cage { //You are allowed to modify this class
   Cage(Animal<?> animal){ this.animal = animal; }
   Animal<?> getAnimal() { return animal; }
   Animal<?> animal;
}
```

```
class Zoo{
  void feedAnimal(Cage cage){ /*code given in each section*/ }
  <F extends Food>void feed(F food, Animal<F> animal){animal.eat(food);}

   void manage(){ /*your code here*/ }
}
```

Clearly a `Wolf` can eat a `Sheep` but not the other way around. In the following subtasks we explore if relaxing some of the Java type rules can lead to a situation where a `Sheep` can eat a `Wolf` - that is, the method `eat` is called on an object of the dynamic type `Sheep` with an argument object of the dynamic type `Wolf`. All the code you give in the answers to the following sections is in the same package as the code above, and must type check in standard Java. The top method that is called in all sections is `Zoo.manage`. You can assume that method call arguments are evaluated left to right, and that the program is sequential. You are not allowed to use reflection, raw types, or type-casts.

A) Assume the following body of `Zoo.feedAnimal(Cage cage)`, which is rejected by the Java type checker:

```
{ feed(cage.getAnimal().lunchBag,cage.getAnimal()); }
```

Make a `Sheep` eat a `Wolf` assuming the body of `feedAnimal` is exempted from the type checker. Show all necessary code. You are only allowed to change the `Cage` class and provide the body of the `Zoo.manage` method.

B) Assume the following body of `Zoo.feedAnimal(Cage cage)`, which is rejected by the Java type checker:

```
{feed(cage.animal.getLunchBag(),cage.animal);}
```

Can you make a `Sheep` eat a `Wolf` if the body of `feedAnimal` is exempted from the type checker? If so, show all necessary code, otherwise explain why not. You are only allowed to change the `Cage` class, provide the body of the `Zoo.manage` method, and add new classes.

C) Answer the question in the previous section, assuming the field `Cage.animal` is `final`. Explain your answer. Reminder: Java's `final` fields can be assigned to only in the constructor of the class that declares them.

D) Assume the following body of `Zoo.feedAnimal(Cage cage)`, which is rejected by the Java type-checker:

```
{feed(cage.animal.lunchBag,cage.animal);}
```

Can you make a `Sheep` eat a `Wolf` if the body of `feedAnimal` is exempted from the type-checker? If so, show all necessary code, otherwise explain why not.

E) Which of the above cases, that is safe in the sequential case, is unsafe in a multithreaded program? For each such case, explain what can happen in the multithreaded case that cannot in the sequential case.

F) The current Java rule for evaluating an expression (including a method call) with wildcard typed arguments is to capture each wildcard in the arguments separately. Propose a more lenient wildcard capture rule than current Java, that is typesafe and accept all the above cases that you deem safe.
Hint: define "stable" paths that cannot be modified by calls.

Task 6

Suppose that we have a language with the information hiding rules of Java, but with structural subtyping. What should be the subtyping relations between the following three classes?

```
class A {int foo();}
class B {protected int foo();}
class C {public int foo();}
```

Task 7

Consider the class `Hour`, defined as follows:

```
public class Hour {
   protected int h=0;
   /// invariant h>=0 && h<24

   public void set(int h) {
      if(h>=0 && h<24) this.h=h;
   }
}
```

What is the external interface of `Hour`?

Can we extend the code, without changing the class, so that the invariant is broken? If yes, provide an example, and propose how to fix the class.

Task 8 Information Hiding

Consider the following Java program consisting of two packages `BTC` and `B2X`:

```
1  package BTC;
2
3  public class Chain {
4
5      /// ensures result <= 2
6      _______ int max_size() {
7          return 2;
8      }
9  }
10
11 package B2X;
12 import BTC.*;
13
14 public class Chain2x extends Chain {
15
16     /// ensures result <= 4
17     protected int max_size() {
18         return 4;
19     }
20 }
```

A) What is the *most permissive* access modifier for the method `max_size()` in class `Chain` such that class `Chain2x` is a *behavioral subtype* of `Chain`? Assume that we *do not use* specification inheritance. **Fill the blank above with your answer**. Explicitly write `default` for a default access modifier. Write `none` if no choice of access modifier allows `Chain2x` to be a behavioral subtype of `Chain`.

B) We now add a class `Block` and a subclass `Block2x` to package `BTC`:

```
 1 package BTC;
 2
 3 public class Block {
 4
 5     protected int num;
 6     /// invariant: 1 <= num
 7
 8     public Block(int n) {
 9         num = (n < 1 ? 1 : n);
10     }
11
12 }
13
14 public class Block2x extends Block {
15
16     /// invariant: 2 <= num
17     protected Block pred;
18     /// invariant: pred != null ==> pred.num < num
19
20     public Block2x(int n, Block b) {
21         super(n < 1 ? 2 : 2*n);
22         pred = (b != null && 2 <= b.num && b.num < num ? b : null);
23     }
24
25 }
```

B.1 Do the invariants in `Block` and `Block2x` satisfy the requirements of *behavioral subtyping*? Assume that we *do not use* specification inheritance. **Briefly explain your answer**.

B.2 A class `C` is *correct* with respect to its invariants if all constructors of `C` establish the invariants *of the new object* and all exported methods `m` of `C` preserve the invariants *of the receiver object*, that is, the invariant holds in the poststate of `m` provided that it held in the prestate of `m`. Are classes `Block` and `Block2x` correct with respect to their invariants? **Briefly explain your answer**.

C) We now want to extend the code in part **B** with *methods that preserve the invariants of the class in which they are declared* but that make it possible to violate the invariants of `Block2x` from client code in another package.

C.1 Extend the code in part **B** with a method that preserves the invariants of the class in which it is declared but makes it possible to violate the *first invariant* of class `Block2x` (i.e., `2 <= num`) from client code in package `B2X`. **Specify in which class you want to declare the method, write the method, and write the client code that violates the invariant.**

C.2 How can you prevent the code that you wrote in part **C.1** from violating the invariant by further extending the code in part **B**? *You are not allowed to modify existing code.* **Write the code that fixes the specific problem you exploited in part C.1.**

C.3 Extend the code in part **B** with a method that preserves the invariants of the class in which it is declared but makes it possible to violate the *second invariant* of class `Block2x` (i.e., `pred != null ==> pred.num < num`) from client code in package `B2X` *in a way that cannot be prevented by further extending the code in part B*. **Specify in which class you want to declare the method, write the method, and write the client code that violates the invariant.**

Task 9

Consider the following class definitions in the context of the read-only type system taught in the course:

```
class C {
   public D f;
   void foo(readonly C other) {...}
}

class D { E g; }

class E {}
```

Let `a` and `b` be non-null references of type `C`. Which of the following statements is true:

(a) The call `a.foo(b)` is guaranteed not to change the value of `b.f`, but may change the value of `b.f.g`

(b) The call `a.foo(b)` is guaranteed not to change the value of `b.f` and neither the value of `b.f.g`

(c) The assignment `other.f.g = new E();` may appear in the code of `foo`

(d) None of the above is correct

Task 10

In the following question we do not consider the owners-as-modifiers discipline. We are only concerned with the topology of the ownership type system.

Consider the assignment:

`o.f = p.g;`

and assume that `o.f` and `p.g` have the same static type.

A) The assignment is forbidden if `o.f` has ownership modifier `lost`. Show an example to demonstrate why we need this rule to preserve the topological invariant.

B) If the ownership modifier of `o.f` is `any`, then what are the requirements for the assignment to be legal?

C) If `o.f` has ownership modifier `lost` can we upcast `o.f` to an `any` reference and make the assignment legal? Why (not)?

Non-null Types and Initialization

Task 11

Consider the following Java code:

```
public class A {
    public static int a1;
    public static int a2;

    static {
        a1 = D.d * 3;
        new B<String>();
        new B<Integer>();
        a2 = 17 + D.d + a1;
    }
}

public class B<T> {
    public static int b;
    static {
        b = A.a2;
        A.a1 += 10;
    }
}

public class C {
    static { A.a2 += 100; }
}

public class D extends C { public static int d = A.a2; }

public class Main {
    public static void main(String[] args){
        if (1 == 2){
            System.out.println(D.d);
        }else{
            System.out.println(A.a2);
        }
    }
}
```

What is the result of compiling the code and running the `Main.main` method?

(a) The code does not compile

(b) The program does not terminate or aborts with a stack overflow

(c) The program terminates normally and prints `127`

(d) The program terminates normally and prints `417`

(e) The program terminates normally and prints `437`

(f) None of the above

Task 12

Consider a Java class `Vector`, representing a 2 dimensional vector:

```
public class Vector {
```

```
    public Number x; // Remark: Number is a super-interface for
    public Number y; //   Integer, Double, etc.

    public Vector (Number x, Number y) {
        this.x = x;
        this.y = y;
    }
}
```

Suppose that in some other class we write the following method to calculate the length of the vector represented by a `Vector` object:

```
public double vectorLength(Vector c) {
    double x = c.x.doubleValue();
    double y = c.y.doubleValue();
    return Math.sqrt(x * x + y * y);
}
```

A) This implementation is unsafe - when executed it may throw exceptions. Why? Is this a reasonable behavior?

B) Add a pre-condition for the method, specifying what is required to be safe.

C) Suppose you are allowed to modify the signature of the method to include non-null type annotations. To what extent can you weaken the necessary pre-condition?

D) Suppose that you are also allowed to upgrade the class `Vector` to include reasonable non-null type annotations. How does this affect your previous answer? Do these changes to the class seem reasonable?

Task 13

Consider the following three classes (declared in the same package):

```
public class Person {
    Dog? dog; // people might have a dog

    public Person() { }
}

public class Dog {
    Person! owner; // Dogs must have an owner
    Bone! bone;    // Dogs must have a bone
    String! breed; // Dogs must have a breed

    public Dog(Person owner, String breed) {
        this.owner = owner;
        this.bone = new Bone(this);
        this.breed = breed;
    }
}

public class Bone {
    Dog! dog;         // Bones must belong to a dog..

    public Bone(Dog toOwn) {
        this.dog = toOwn;
    }
}
```

A) Annotate the code with non-null and construction type annotations where they are necessary. Explain why the code now type-checks according to construction types.

B) Could we provide constructors for classes `Dog` and `Bone` with no parameters?

C) Now, suppose a (possibly mad) scientist wants to extend the implementations of these classes with some genetic engineering. Firstly, we want to be able to "clone" a bone. We can add the following method to class `Bone` to make a copy of an existing bone, and assign it to another `Dog`:

```
public Bone clone(Dog toOwn) {
      return new Bone(toOwn);
   }
```

However, our scientist would like to go further, and be able to clone dogs. A cloned `Dog` should also have its bone cloned along with it, but may be assigned to a new owner: we add the following extra constructor and method to class `Dog`:

```
Dog(Dog toClone, Person newOwner) {
   this.owner = newOwner;
   this.breed = toClone.breed;
   this.bone = new Bone(this);
}

public Dog clone(Person toOwn) {
   return new Dog(this, toOwn);
}
```

However, our scientist would like to go still further, and be able to clone people. A cloned `Person` should also have its dog (if any) cloned along with it: we add the following extra constructor and method to class `Person`:

```
Person(Person toClone) {
   Dog? d = toClone.dog;
   if(d!=null) {
      this.dog = new Dog(d, this);
   }
}

public Person clone() {
   return new Person(this);
}
```

Annotate this extra code with appropriate non-null and construction types annotations. You should guarantee that each of the `clone` methods (which belong to the public interface) return a committed reference. You should ensure that your answers guarantee that all of the code type-checks - explain your choices.

Hint: think carefully about how constructor calls are typed, and what happens if the constructors are called in more than one situation.

Reflection

Task 14

Which of the following is the *defining* characteristic of reflection?

(a) It allows for much simpler code

(b) It enables more flexibility

(c) It allows a program to observe and modify its own structure and behavior

(d) It is not statically safe

(e) It may hurt performance

(f) None of the above

Task 15

Consider the following Java code:

```
void foo() throws java.lang.Exception {
  LinkedList<String> xs = new LinkedList<String>();
  xs.add("A"); xs.add("B"); xs.add("C");

  Class<?> c = xs.getClass();
  Method remove = c.getMethod("remove");
  xs.add(remove.invoke(xs));
}
```

which uses the following methods of class `LinkedList<E>`

```
  public E remove()
  public boolean add(E e)
```

Which of the following statements is true? The invocation of ...

(a) `c.getMethod("remove")` is rejected by the compiler

(b) `c.getMethod("remove")` raises an exception (at run time)

(c) `remove.invoke(xs)` is rejected by the compiler

(d) `remove.invoke(xs)` raises an exception (at run time)

(e) `xs.add(...)` is rejected by the compiler

(f) `xs.add(...)` raises an exception (at run time)