CS164 Programming Languages and Compilers **Spring 2024**

Written Assignment 3 Solutions

Assigned: February 22 **Due:** March 1 at 3:00pm

1. Examine the following ChocoPy program and list all erroneous lines. For each erroneous line, briefly explain the error. (Note: For this question, only include compile-time errors caught during semantic analysis.):

```
1.  x:int = 1
2.  y:int = 2
3.  z:Dog = None
4.  class Dog(object):
5.      y:str = "y"
6.  def f(x:int)->int:
7.      y:int = 3
8.      nonlocal x
9.      x = 0
10.     return x
11. def g(y:int):
12.     global y
13.     y = 3
14. g(y)
15. f(len(z.y))
```

Lines with errors:

- Line 8: `x` is defined locally and globally, but there is no `x` that is nonlocal.
- Line 12: `y` is already defined locally.

2. The Java language has arrays and allows inheritance. In Java you are allowed to assign a value of type `T[]` to a variable of type `U[]` so long as the type `T` is a subtype of `U`. This however, turns out to be unsound despite being able to pass Java's type checking rules. Give an example of Java code that would pass Java's static type checking but would cause an error in execution. Also describe why this error occurs in your example.

   The following is one out of many valid answers:

```
Integer[] A;
Object[] B;
A = new Integer[10];
B = A;
B[0] = new String("ERROR");
```

   This code will successfully compile, since for each assignment the static type of the assigned value is a subtype of the destination variable's type. However, we cannot place a `String` value into an array `A` which was declared to only hold `Integers`; the last line causes a `java.lang.ArrayStoreException`.

3. Write Java code that would print "dynamic" if executed with dynamically scoped environments, but "static" if executed with statically scoped environments.

```
public class Solution {
    const String word = "static";
    public String get_word() {
        return word;
    }
    public String has_local_word() {
        String word = "dynamic";
        return get_word();
    }
    public static void main(String[] args){
        System.out.println(has_local_word());
    }
}
```

4. Imagine that we have a series of user defined types in ChocoPy as follows: `Animal` is a class, and `Dog`, `Cat`, and `Bear` all derive from the `Animal` class. There are two classes `Poodle` and `Pug` which derive from the `Dog` class. There are two classes `Siamese` and `Persian` which derive from the `Cat` class. There is one class `Oski` which derives from the `Bear` class.

   (a) Let `po` be of static type `Poodle` and let `pu` be of static type `Pug`. What is the type of `[po, pu]` (a list containing `po` and `pu`)?

   `[Dog]`, since `Dog` is the least upper bound of the element types `Poodle` and `Pug`.

   (b) Let `o` be of static type `Oski` and let `b` be of static type `Bear`. What is the type of `[o, b]`?

   `[Bear]`, since `Bear` is the least upper bound of the element types `Oski` and `Bear`.

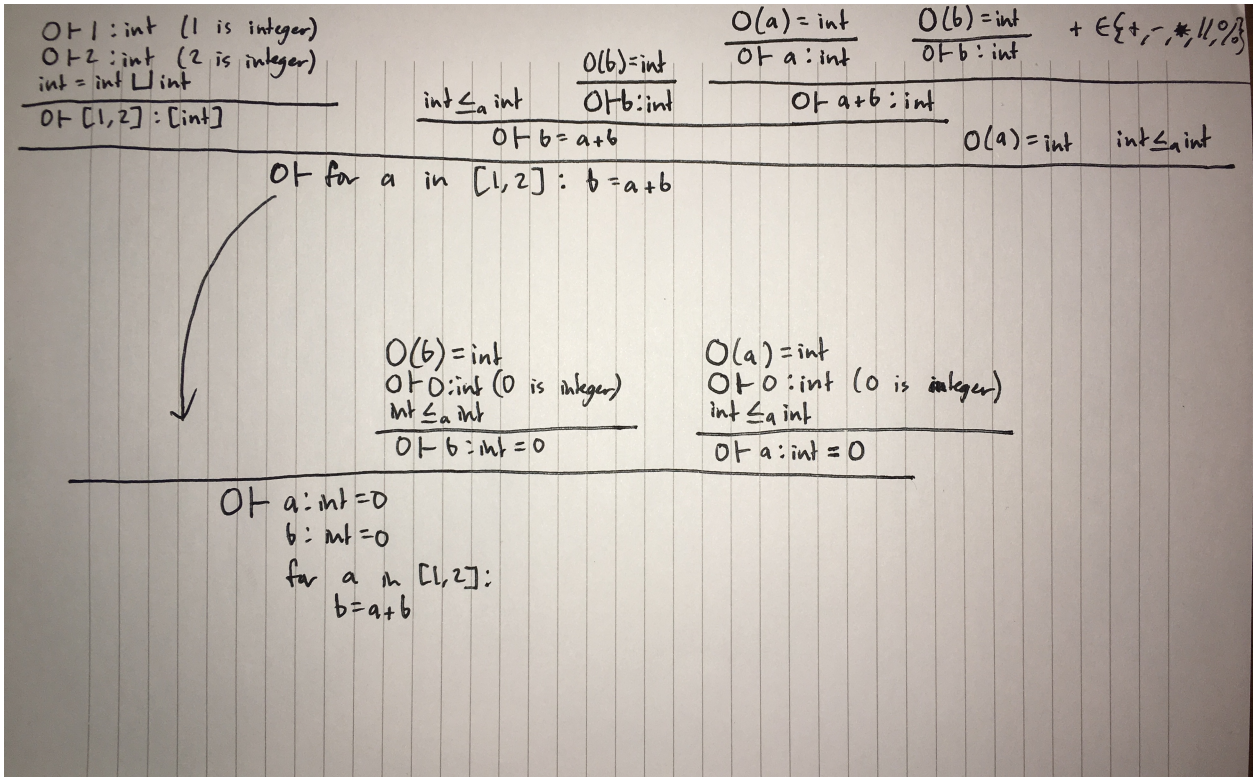
   (c) Let `p` be of static type `Pug` and let `s` be of static type `Siamese`. Also let `o` be of static type `Oski`. What is the type of `[p, s, o]`?

   `[Animal]`, since `Animal` is the least upper bound of the element types `Pug`, `Siamese`, and `Oski`.

5. The following ChocoPy program satisfies all type-checking rules. Express the typing derivation of this program in an inverted-tree representation:

```
a : int = 0
b : int = 0
for a in [1, 2]:
    b = a + b
```

That is, the the judgment that the program is well-typed should be at the bottom of the tree, and any hypotheses (which are themselves typing judgments) used for this judgment should be just above, and so on. You can omit M, C, and R in the typing judgments. Assume the initial environment is $O = \{int/a, int/b\}$.



6. In Question 1, you were asked to find the erroneous lines in the program, which largely dealt with issues of scope. Now we will focus on typing rules. For the following ChocoPy program, please list the lines (if any) that do not type check, and provide brief justification.

```
1. x : int = 3
2. y : str = "a"
3. def f() -> object:
4.     return 0
5. y = y + x
6. x = None
```

(a) Line 5: This line does not type check because we cannot apply the addition rule for either int's or str's since each rule requires both expressions to have same type (both int's or both str's, respectively).

(b) Line 6: `<None>` is not assignment compatible to type `int`.

Note: Line 4 type checks because `int` derives from `object`.