Type Checking Exercises

CS 4610 — Spring 2017

This Review Set asks you to prepare written answers to questions on type checking. Each of the questions has a short answer. You may discuss this Review Set with other students and work on the problems together.

1 Definitions and Background

1. Define the following terms and give examples where appropriate.
   (a) Semantic Analysis:

   (b) Variable Declaration:

   (c) Variable Definition:

   (d) Scoping Rules:

   (e) Symbol Table:

   (f) Type:

   (g) Type System:

   (h) Type Checking:

   (i) Type Inference:

   (j) Soundness:

(k) Type Environment:

(l) Free Variable:

(m) Dynamic Type:

(n) Static Type:

2. Describe some key differences between statically- and dynamically-typed languages. What are some advantages of each approach?

2 Type Checking

1. C++, unlike COOL, supports multiple inheritance. For example, the following class hierarchy is legal in C++:

```
  A
 / \
B   C
 \ /
  D
```

   Allowing multiple inheritance changes the way we define and use the least upper bound (*lub*) function on types.

   (a) Explain, using at least one example, why it is necessary to change *lub*.

   (b) Describe how you might implement *lub* for multiple inheritance. Be brief.

2. The Java programming language includes arrays. The Java language specification states that if s is an array of elements of class S, and t is an array of elements of class T, then the assignment $s = t$ is allowed as long as T is a subclass of S.

   This typing rule for array assignments turns out to be unsound. Java works around this by inserting runtime checks to throw an exception if arrays are used unsafely.

   Consider the following Java program, which type checks according to the preceeding rule:

```java
class Mammal { String name; }

class Dog extends Mammal {
  void beginBarking() { ... }
}

class Main {
  public static void main(String argv[]) {
    Dog x[] = new Dog[5];
    Mammal y[] = x;

    // Insert your code here






























  }
}
```

   Add code to the `main` method so that the resulting program is a valid Java program (i.e., it compiles), but running that program triggers one of the aforementioned runtime checks. Include a brief explanation of how your program exhibits the problem.

3. The following typing judgments have one or more flaws. For each judgment, list the flaws and explain how they affect the judgment.

   (a)

$$\frac{\begin{array}{c} O \vdash e_0 : T \\ O \vdash T \leq T_0 \\ O \vdash e_1 : T_1 \end{array}}{O[x/T_0] \vdash \textsf{let } x : T_0 \leftarrow e_0 \textsf{ in } e_1 : T_1} \ (let-init)$$

   (b)

$$\frac{\begin{array}{c} O(\mathrm{id}) = T_0 \\ O \vdash e_1 : T_1 \\ T_0 \leq T_1 \end{array}}{O \vdash \mathrm{id} \leftarrow e_1 : T_1} \ (assign)$$

   (c)

$$\frac{T \leq C}{\vdash \mathrm{SELF\_TYPE}_C \leq T} \ (self-type)$$

(d)

$$\frac{\begin{array}{c} O, M, C \vdash e_0 : T_0 \\ \dots \\ O, M, C \vdash e_n : T_n \\ T_0 \leq T \\ M(T_0, f) = (T'_1, \dots, T'_n, T'_{n+1}) \\ T'_{n+1} \neq \text{SELF\_TYPE} \\ \forall 1 \leq i \leq n \ . \ T_i \leq T'_i \end{array}}{O, M, C \vdash e_0 @ T.f(e_1, \dots, e_n) : T'_{n+1}} \ (static - dispatch - self)$$

4. Draw the AST for the following Cool snippet and annotate each node with its associated type (following the Cool typing rules). You may assume that class E has been defined an that it has a function `set_var`, which has a single formal parameter of type `Int`.

```
let num : Int <- in_int() in
let x : Int <- 1 in
  {
     (let y : Int <- 1 in
        while y <= num loop
           {
              x <- x * y;
              y <- y + 1;
           }
        pool
     );
     (new E).set_var(x);
  }
```