CS415 — Written Assignment 5

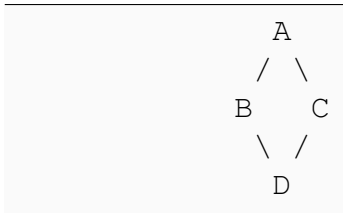
2007-03-19

This assignment asks you to prepare written answers to questions on type checking. Each of the questions has a short answer. You may discuss this assignment with other students and work on the problems together. However, your write-up should be your own individual work.

Please print your name and email address on your homework!

We need this information so that we can give you credit for the assignment and so that we can return it to you.

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



   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:

Listing 1: totally innocuous Java code

```
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)

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

(b)

$$\frac{\begin{array}{c} W \vdash P\langle T \rangle : type \\ O, M, W \vdash e_0 : P\langle T \rangle \\ O, M, W \vdash e_1 : T_1 \\ \vdots \\ O, M, W \vdash e_n : T_n \\ M(P\langle x \rangle, f) = (T'_1 \ldots T'_n, T'_{n+1}) \\ T_i \leq T'_i[x/T] \quad 1 \leq i \leq n \end{array}}{O, M, W \vdash e_0.f(e_1, \ldots e_n) : T'_{n+1}} \quad \text{[dispatch-template]}$$

Where $T_0[q/T_1]$ reads as 'replace each occurrence of the name q with type expression T_1 in type expression T_0. ' In class it was written as T_0''.

4. Recall the 'is a valid type' rules we used to type check COOL-with-templates:

$$\frac{}{W \vdash C : type} \quad \frac{}{W = P\langle t \rangle \vdash t : type} \quad \frac{W \vdash T : Type}{W \vdash P\langle T \rangle : Type}$$

Use these three rules to construct a derivation for the following COOL-with-templates code. (For an example derivation tree, see slide 44 in the 'Scoping Types' lecture notes.)

Listing 2: COOL-with-templates

```
class List<T> {
  (* ... *)
};

class Person {
  (* ... *)
};

(* Give the derivation for the type of x: *)
let x : List<List<Person> > <- ...
```