CSE341 - Section Problems

May 26, 2004

The following questions are meant to give you practice in understanding some of the OO-related concepts that we've been discussing in the last 2 weeks. They should also serve well as additional review for the final exam.

1. (Subtyping) For each of the following questions, determine under what conditions it is sound for the first type to be a subtype of the second:

   (a) When is $\tau_1 \rightarrow \tau_2$ a subtype of $\tau_3 \rightarrow \tau_4$?

   (b) When is $\tau_a \rightarrow \tau_b \rightarrow \tau_c$ a subtype of $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3$?

   (c) When is $(\tau_a \rightarrow \tau_b) \rightarrow (\tau_c \rightarrow \tau_d)$ a subtype of $(\tau_1 \rightarrow \tau_2) \rightarrow (\tau_3 \rightarrow \tau_4)$?

2. (Picking on Java) This program type-checks and runs:

```
class C {
  public static void f(Object x, Object arr[]) {
    arr[0] = x;
  }
  public static void main(String args[]) {
    Object o = new Object();
    C []   a = new C[10];
    f(o, a);
  }
}
```

   (a) For this program, where does the type-checker use subsumption? From what type to what type? What is Java's subtyping rule for arrays?

   (b) Does this program execute any downcasts when it runs? What happens when it runs?

   (c) Informally, what is the semantics of array-update in Java? (Start your answer with, "Array update takes an array-object a, an index i, and an object $o\ldots$". Discuss what exceptions might be thrown under what conditions and what occurs if no exceptions are thrown.)

(d) Is it possible to compile a Java program without run-time type information, even if the program has no downcasts, method overriding, or reflection? (Note that compilation must preserve the behavior you described in the previous question.)

3. (Fragile Superclasses) Assume a class-based OO language where "subclassing *is* subtyping" and there is no static overloading. Consider a class C, a class D that extends C, and a client P that uses classes C and D. Now consider each of these potential source-code changes to class C:

   (a) We add a method f to C.

   (b) We take an existing method f of C and change f from taking one argument of type T_1 to taking one argument of type T_2, where $T_1 \leq T_2$.

   (c) We take an existing method f of C and change f from taking one argument of type T_1 to taking one argument of type T_2, where $T_2 \leq T_1$.

   (d) We take an existing method f of C and make it abstract (removing its implementation, but still requiring all objects of type C to have it).

   For each of these changes:

   - Describe the conditions under which D will no longer typecheck. That is, describe all D where the definition of class D should type-check before the change of C but should not type-check after the change.
   - Describe the conditions under which P will no longer typecheck. That is, describe all P where the code for P should type-check before the change of C but should not type-check after the change.

4. (Encoding Fields) Consider an OO language with public fields. (Whether the language has classes or not is irrelevant.)

   (a) Explain how we can translate programs in this language into a similar one where all fields are private. (Hint: Add two methods per field to every object. Explain how to change method bodies to use these fields.)

   (b) Explain how we can translate programs in the language with private fields into one with *no fields at all*, but with *method update*. Method update, written `o.m := mbody`, mutates an object `o` such that its method name `m` is bound to `mbody`. Here is a silly example: This method changes `o.m` to multiply its argument by n and "optimizes" the case $n == 2$:

```
unit f(C o, int n) {
   if(n==2) then o.m := int m(int x) { x + x; }
   else o.m := int m(int x) { x * n; }
}
```