Programming Languages
CSCI-GA.2110.001 Fall 2020

Homework 2
Due Sunday, December 13 at 11:55pm

1. (a) Sometimes students in my class confuse substitution and β-reduction in the lambda calculus. In particular, they have a hard time distinguishing
   (1) $(\lambda x.E)[M/x]$ (a substitution)
   from β-reduction applied to the following expression:
   (2) $(\lambda x.E)M$
   Provide the result of the substitution shown in (1) above and the result of performing β-reduction on the expression shown in (2) above. Explain why the results are different.
   (b) Give the actual expression, i.e. $(\lambda ...)$, representing the Y combinator in the lambda calculus.
   (c) Show, as I did in class, that if Y is the expression you gave above, it has the property $Yf \Leftrightarrow f(Yf)$.
   (d) Write the definition of a recursive function (other than factorial) using the Y combinator. Show a series of reductions of an expression involving that function which illustrates how it is, in fact, recursive (as I did in class for factorial).
   (e) Summarize, in your own words, what the two Church-Rosser theorems state.

2. (a) Write a function in ML whose type is:
   `('a -> 'b) -> ('b list -> 'c list) -> ('c -> 'd) -> 'a -> 'd`
   (b) What is the type of the following function (try to answer without running the ML system)?

```
fun foo f (op >) x (y,z) =
   let fun bar w = if x > z then y else w
   in  f (bar [1,2,3])
   end
```

   (c) Provide an intuitive explanation of how the ML type inferencer would infer the type that you gave as the answer to the previous question.

3. (a) As discussed in class, what are the three features that a language must have in order to be considered object oriented?
   (b) What is the "subset interpretation of suptyping"?
   (c) Provide an intuitive answer, and give an example, showing why class derivation in Java satisfies the subset interpretation of subtyping.
   (d) Provide examples in Scala of (1) why function subtyping cannot be covariant in the input type, and (2) why function subtyping cannot be contravariant in the output type. Briefly explain each example.
   (e) Provide an intuitive answer of why subtyping of functions in Scala satisfies the subset interpretation of subtyping.

4. In Java generics, subtyping on instances of generic classes is invariant. That is, two different instances `C<A>` and `C<B>` of a generic class `C` have no subtyping relationship, regardless of a subtyping relationship between `A` and `B` (unless, of course, A and B are the same class).

   (a) Write a function (method) in Java that illustrates why, even if `B` is a subtype of `A`, `C<B>` should not be a subtype of `C<A>`. That is, write some Java code that, if the compiler allowed such covariant subtyping among instances of a generic class, would result in a run-time type error.

   (b) Modify the code you wrote for the above question that illustrates how Java allows a form of polymorphism among instances of generic classes, without allowing subtyping. That is, make the function you wrote above be able to be called with many different instances of a generic class.

5. (a) Consider the following Scala definition of a tree type, where each node contains a value.

```
abstract class Tree[T <: Ordered[T]]
case class Node[T <: Ordered[T]](v:T, l:Tree, r:Tree) extends Tree[T]
case class Leaf[T <: Ordered[T]](v:T) extends Tree[T]
```

   Ordered is a built-in trait in Scala (see `http://www.scala-lang.org/api/2.12.0/scala/math/Ordered.html`). Write a Scala function that takes a Tree[T], for any ordered T, and returns the minimum (smallest) value in the tree. Be sure to use good Scala programming style.

   (b) In Scala, write a generic class definition that supports covariant subtyping among instances of the class. For example, define a generic class C[E] such that if class B is a subtype of class A, then C[B] is a subtype of C[A].

   (c) Give an example of the use of your generic class from part (b).

   (d) In Scala, write a generic class definition that supports contravariant subtyping among instances of the class. For example, define a generic class C[E] such that if class B is a subtype of class A, then C[A] is a subtype of C[B].

   (e) Give an example of the use of your generic class from part (d).

6. (a) What is the advantage of a mark-and-sweep garbage collector over a reference counting collector?

   (b) What is the advantage of a copying garbage collector over a mark and sweep garbage collector?

   (c) Write a brief description of generational copying garbage collection.

   (d) Write, in the language of your choice, the procedure `delete(x)` in a reference counting GC system, where `x` is a pointer to a structure (e.g. object, struct, etc.) and `delete(x)` reclaims the structure that `x` points to. Assume that there is a free list of available blocks and `addToFreeList(x)` puts the structure that `x` points to onto the free list.