Programming Languages
CSCI-GA.2110.001 Spring 2017

Homework 2
Due Monday, May 8

You should write the answers using word, latex, etc., and upload them as a PDF document. **Important: You <u>must</u> turn this in by 11:55pm on Monday, May 8. I will be posting the solutions after that.**

1. (a) In the λ-calculus, give an example of an expression which would reduce to normal form under normal-order evaluation, but not under applicative-order evaluation.

   (b) 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).

   (c) Write the actual expression in the λ-calculus representing the Y combinator, and show that it satisfies the property $\mathrm{Y(f)} = \mathrm{f(Y(f))}$.

   (d) Summarize, in your own words, what the two Church-Rosser theorems state.

2. (a) In ML, why do all lists have to be homogeneous (i.e. all elements of a list must be of the same type)?

   (b) Write a function in ML whose type is `('a -> 'b list) -> ('b -> 'c list) -> 'a -> 'c`.

   (c) What is the type of the following function (try to answer without running the ML system)?

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

   (d) 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 considered object oriented?

   (b) What is the "subset interpretation of suptyping"?

   (c) Explain why function subtyping must be contravariant in the parameter type and covariant in the result type. If necessary, provide examples to illuminate your explanation.

   (d) Provide an intuitive answer showing why function subtyping satisfies the subset interpretation of subtyping. Be sure to consider both the contravariant and covariant aspects of function subtyping.

   (e) Give an example in Scala that demonstrates subtyping of functions, utilizing both the contravariance on the parameter type and covariance on the result type.

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) 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].

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

   (c) 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].

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

   (e) 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[T], r:Tree[T]) 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/current/index.html#scala.math.Ordered`). Write a Scala function that takes a Tree[T], for any ordered T, and returns the smallest (minimum) value in the tree. Be sure to use good Scala programming style.

6. (a) What is the advantage of a reference counting collector over a mark and sweep 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)` deletes the pointer x. Assume that there is a free list of available blocks and `addToFreeList(x)` puts the structure that `x` points to onto the free list.