CSCI-GA.2110.001 Programming Languages	December 7, 2015

Solutions to Problem 1 of Homework 2

Name: GOWTHAM GOLI (N17656180) *Due: Monday, December 14*

Provide regular expressions for defining the syntax of the following.

(a) As we discussed in class, the expression $(\lambda x.(xx))(\lambda x.(xx))$ has no normal form. Write another expression that has no normal form. Make sure that your expression is distinct from $(\lambda x.(xx))(\lambda x.(xx))$, i.e. that it wouldnt be convertible to $(\lambda x.(xx))(\lambda x.(xx))$. Hint: Think about how youd write a non-terminating expression in a functional language

Solution:

□

(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)..

Solution:

Let FIB = $Y(\lambda f.\lambda n$ if (= n 0) 1 else if (= n 1) 1 (+ (f (- n 1)) (f (-n 2))))

and $p = \lambda f.\lambda n$ if (= n 0) 1 else if (= n 1) 1 (+ (f (- n 1)) (f (-n 2)))

$\implies$ FIB $= Y(p) = p(Y(p))$

Consider the evaluation of fibonnaci of 4

$$\text{FIB } 4 = \underbrace{(\lambda f.\lambda n\ldots)}_{\text{p}}\underbrace{(Y(\underbrace{\lambda f.\lambda n\ldots}_{\text{p}}))}_{\text{FIB}}\ 4$$

$$\underset{\beta}{\Longrightarrow} (\lambda n \text{ if } (= n\ 0)\ 1 \text{ else if } (= n1)\ 1\ (+\ (\text{FIB } (\text{- } n\ 1))\ (FIB\ (\text{- n } 2))))\ 4$$

$$\underset{\beta}{\Longrightarrow} \lambda n \text{ if } (=4\ 0)\ 1 \text{ else if } (=\ 4\ 1)\ 1\ (+\ (\text{FIB } (\text{- } 4\ 1))\ (\text{FIB } (\text{-}4\ 2))$$

$$\underset{\delta}{\Longrightarrow} (+\ (\text{FIB } (\text{- } 4\ 1))\ (\text{FIB } (\text{-}4\ 2)))$$

$$\underset{\delta}{\Longrightarrow} (+\ (\text{FIB } 3)\ (\text{FIB } 2))$$

On further beta and delta reductions, FIB 2 reduces to (+ 1 1) = 2 and FIB 3 reduces to (+ (FIB 2) 1) = 3. Thus FIB 4 evaluates to 5 □

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

Solution:

$$
\begin{aligned}
Y &= \lambda f.(\lambda x.(\text{f (x x)}))(\lambda x.(\text{f (x x)})) \\
Y\ f &= \lambda f.(\lambda x.(\text{f (x x)}))(\lambda x.(\text{f (x x)}))\ f \\
&\underset{\beta}{\Longrightarrow} (\lambda x.(\text{f (x x)}))(\lambda x.(\text{f (x x)})) \\
&\underset{\beta}{\Longrightarrow} f((\lambda x.(\text{f (x x)}))(\lambda x.(\text{f (x x)}))) \\
&= f(Y\ f)
\end{aligned}
$$

□

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

Solution:

For a given lambda expressions, there could be multiple ways of reducing it to a normal form. However some particular order of reductions might not always terminate to a normal form.

Church-Rosser theorem 1 states that if any two different order of reductions of a given lambda expression terminate then they will result in the same normal form

Church-Rosser throrem 2 states that if there is some order of reductions of a given lambda expression that terminates to a normal form then Normal Order reduction will definitely terminate. (By theorem 1 it terminates to the same normal form) □

CSCI-GA.2110.001 Programming Languages	December 7, 2015
Solutions to Problem 2 of Homework 2	
Name: GOWTHAM GOLI (N17656180)	*Due: Monday, December 14*

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

Solution:

It makes the static checking of the types possible and the type checker sound and complete

□

(b) Write a function in ML whose type is $('a \rightarrow' b) \rightarrow ('b \text{ list} \rightarrow' c \text{ list}) \rightarrow' a \rightarrow' c$.

Solution:

□

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

Solution:

□

(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.

Solution:

□

CSCI-GA.2110.001 Programming Languages December 7, 2015

Solutions to Problem 3 of Homework 2

Name: GOWTHAM GOLI (N17656180) *Due: Monday, December 14*

Consider the following package specification for an Ada package that implements a queue of integers.

```
package queue is
  function extract return integer;
  function insert(x: integer);
end queue;
```

(a) Why would this package not be said to implement an abstract data type (ADT) for a queue?

Solution:

□

(b) Modify the above package specification, and implement a simple package body (that performs no error checking), so that a queue is an ADT.

Solution:

□

CSCI-GA.2110.001 Programming Languages December 7, 2015

Solutions to Problem 4 of Homework 2

Name: GOWTHAM GOLI (N17656180) *Due: Monday, December 14*

(a) As discussed in class, what are the three features that a language must have in order to considered object oriented?

Solution:

$\square$

(b) i. What is the subset interpretation of suptyping?

Solution:

$\square$

ii. Provide an intuitive answer, and give an example, showing why class derivation in Java satisfies the subset interpretation of subtyping.

Solution:

$\square$

iii. Provide an intuitive answer, and give an example, showing why subtyping of functions in Scala satisfies the subset interpretation of subtyping.

Solution:

$\square$

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

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

Solution:

$\square$

(d) 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).

i. 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

Solution:

□

ii. 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.

Solution:

□

(e) i. 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].

Solution:

□

ii. Give an example of the use of your generic class.

Solution:

□

(f) i. 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]

Solution:

□

ii. Give an example of the use of your generic class

Solution:

□

CSCI-GA.2110.001 Programming Languages	December 7, 2015

Solutions to Problem 5 of Homework 2

Name: GOWTHAM GOLI (N17656180) *Due: Monday, December 14*

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

Solution:

□

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

Solution:

□

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

Solution:

□

(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

Solution:

□