Programming Languages
CSCI-GA.2110.001 Fall 2014

Final Exam
Answers

1. (a) Define the following terms: imperative language, functional language, object oriented language.

   An imperative language is a programming language with the following features: (1) variables represent modifiable locations in memory, (2) statements ("imperatives") modify the contents of those locations in memory, and (3) control flow operations serve to direct execution over the statements.

   A functional language is a programming language with the following features: (1) variables represent values and are not modifiable, (2) functions are "first class" in that they are treated as values, so can be passed as parameters, returned as values, etc.

   An object oriented language is a programming language with the following features: (1) data and the procedures that operate on that data are encapsulated together within a type ("class"), (2) inheritance is used to define a new (child) class based on an existing (parent) class, and (3) subtyping with dynamic dispatch allows an object of a child class to be used anywhere an object of the parent class is expected, with the choice of which methods to call (the parent's or the child's) determined at run time.

   (b) Give an example of a term in the lambda calculus that will be reduced to a normal form in fewer steps (applications of β-reduction) using applicative order evaluation than using normal order evaluation. Show the reduction sequences using applicative order evaluation and using normal order evaluation (should be short!).

   (λ x. + x x) (+ 3 4) would reduce to normal form using applicative order evaluation as follows: (λ x. + x x) (+ 3 4) $\Rightarrow$ (λ x. + x x) 7 $\Rightarrow$ + 7 7 $\Rightarrow$ 14.

   **It would reduce to normal form using normal order evaluation, requiring an additional step, as follows: (λ x. + x x) (+ 3 4) $\Rightarrow$ + (+ 3 4) (+ 3 4)
   $\Rightarrow$ + 7 (+ 3 4) $\Rightarrow$ + 7 7 $\Rightarrow$ 14.**

   (c) What would the following program print if it were written in a language that used pass-by-name parameter passing?

```
program exam;
  i: integer := 0;
  function f(x: integer):Integer
     sum: integer := 0;
  begin
     while (i < 10) do
       sum := sum + x;
       i : = i + 1;
     end while;
     return sum;
  end f;
begin (* program *)
  print(f(i+3));
end exam;
```

   75 (which is (0+3)+(1+3)+(2+3)+...+ (9+3) = 3+4+5+...+12)

2. (a) Write a Scheme function `maxlist` that finds the largest number stored anywhere in a list. Assume the list can contain sublists, but the only atomic elements will be numbers. For example, `(maxlist '(1 (9 2 (10 5) 6) 8))` should return 10.

```
(define (maxlist L)
    (cond ((null? L) (display "Error: Max doesn't exist") (newline))
          ((and (null? (cdr L)) (pair? (car L))) (maxlist (car L)))
          ((null? (cdr L)) (car L))
          ((pair? (car L)) (max (maxlist (car L)) (maxlist (cdr L))))
          (else (max (car L) (maxlist (cdr L))))))
```

   (b) In your Scheme interpreter, why wasn't `apply` defined in your library file, but instead was inserted into `*global-env*`?

   This question is not relevant to our class.

   (c) Write a Scheme function `f` that, given a list `L`, returns a function that, given a list `L2`, returns `L` if `(maxlist L)` is greater than `(maxlist L2)`, and returns `L2` otherwise. For example,

```
> (define g (f '(1 2 (3 4))))
> (g '(1 3 5))
(1 3 5)
> (g '(1 (2 3)))
(1 2 (3 4))
```

   Answer:

```
(define (f L) (lambda (L2) (if (> (maxlist L) (maxlist L2)) L L2)))
```

3. (a) Define a function in ML whose type is `('a -> 'b) -> ('a -> 'b)`.

```
fun f g = fn x => g x
```

   or, more simply,

```
fun f g x = g x
```

   (b) What is the type of the following function?

```
fun foo f g x y = let fun bar z = f(g(x,y), z)
                  in bar 3
                  end
```

```
('a * int -> 'b) -> ('c * 'd -> 'a) -> 'c -> 'd -> 'b
```

   The type variable names you choose may, of course, be different.

   (c) Give an intuitive description of how the type you gave in your previous answer would be inferred by the compiler.

   The variables `x` and `y` don't have any constraints, so their types are `'c` and `'d`, respectively. Since `g` takes `(x,y)` as a parameter, g's input type is `'c * 'd`. The result of `g(x,y)` is unconstrained (call it `'a`), so g's type is `'c * 'd -> 'a`. The parameter `z` to `bar` is an `int`, since `bar` is applied to 3. Since the input to `f` is a tuple containing the result of `g(x,y)` and `z`, the input type of f is `'a * int` and the result type is unconstrained (call it `'b`). The result type of `foo` is the same as the result type of `bar`, which is the same as the result type of `f`, which is `'b`. Thus, the type of foo is the type given above.

(d) Define in ML a polymorphic datatype, `'a mylist`, whose values are either an empty list, written `nil`, or a non-empty list, created by writing an expression of the form `cons(x,xs)`, where x is of type `'a` and `xs` is of type `'a mylist`. For example, once you have defined the `mylist` datatype, you could write,

```
val L = cons(3, cons(4, nil))
```

Answer:

```
datatype 'a mylist = nil | cons of 'a * 'a mylist
```

(e) Write a polymorphic function `max` whose type is `('a * 'a -> bool) -> 'a mylist -> 'a` that computes the maximum element of an `'a mylist`, where the first parameter is a "greater than" operator (and should be used as `>` within `max`). Assume that `max` will not be called on the empty list (so you don't have to handle that case).

Answer:

```
fun max (op >) nil = raise error
 |  max (op >) (cons(x,nil)) = x
 |  max (op >) (cons(x,xs)) = let val y = max (op >) xs
                               in if x > y then x else y
                               end
```

4. (a) Give a simple example in Java of the overriding of a method (in a child class), as well as the dynamic dispatching of that method.

Answer:

```
class A {
    String foo() { return "I'm an A";}
}

class B extends A {
    String foo() { return "I'm a B"; }
}

public class C {
    static void bar(A x) {
        System.out.println(x.foo());
    }

    public static void main(String args[]) {
        A a = new A();
        B b = new B();
        bar(a);
        bar(b);
    }

}
```

(b) Why doesn't Java allow subtyping between instances of a generic class? Give an example in Java where such subtyping, if it were allowed, would cause a problem. Briefly explain your example.

Because allowing subtyping among instances of a generic class could cause type errors. For example, given

```
class C<T> extends ArrayList<T> {}

class A {}
class B extends A { public int x; }

class D {

    static void f(C<A> c) {
        A a = new A();
        c.add(a);
    }

    public static void main(String args[]) {
        C<B> c = new C<B>();
        f(c);
        B b = c.get(0);
        b.x++;
    }
}
```

if, in `main()`, the object `c` of type `C<B>` (which extends `ArrayList<B>`) were allowed to be passed to `f()`, then an `A` object would be inserted into `c`, whose elements are `B`'s. This would be a type error, since, as shown in `main()`, the program could access an element of `c` and treat it as a `B`, even though it may possibly be an `A`.

(c) Suppose there is a Java generic class, `C<T>`, that implements the `Collection<T>` interface, so that the method `boolean add(T x)` adds a `T` object to the collection. Write a static method `insert()` that is as polymorphic as possible and takes two parameters:

- a `Car` object (where `Car` is derived from `Vehicle`), and
- any `C<>` object into which a `Car` object can be inserted.

Your `insert()` method should add the `Car` object to the `C<>` object using the `add()` method.

Answer:

```
void insert(C<? super Car> c, Car x) {
    c.add(x);
}
```

(d) Suppose class `Porsche` is derived from `Car`. Given your definition of `insert()`, and given a variable `c` of type `C<Car>` and a variable `p` of type `Porshe`, would it be possible to call `insert(c,p)`? Explain why or why not.

Yes. Because `Porsche` is a subtype of `Car`, a `Porshe` object can be passed as the second parameter to `insert()`, above, which is expecting a `Car`. Since an object of type `C<Car>` satisfies the constraint `C<? super Car>`, a `C<Car>` object can be passed as the first parameter to `insert()`.

5. (a) Explain how function subtyping in Scala, where if `B <: A` then `A=>B <:B=>A`, satisfies the subset interpretation of subtyping.

Function subtyping is contravariant in the input type and covariant in the output type. This means that for any types `X` and `Y`, the type `X => Y` is the

set of all functions whose input type is a supertype of `X` (including `X` itself) and whose output type is a subtype of `Y` (including `Y` itself).

Since, under the subset interpretation of subtyping, `B <: A` means that `B` is a subset `A`, the set of functions whose input type is a supertype of `B` (including `B`) is a larger set than the set of functions whose input type is a supertype of `A`. That's because `A` is a supertype of `B`, but not vice versa. Similarly, the set of functions whose output type is a subtype of `A` (including `A` itself) is a larger set than the set of functions whose output type is a subtype of `B` (including `B` itself), since `B` is a subtype of `A`, but not vice versa. Thus, it must be the case that the set of all functions whose input type is a supertype of `B` and whose output type is a subtype of `A` is a bigger set than the set of all functions whose intput type is a supertype of `A` and whose output type is a subtype of `B`. That is `A => B` $\subseteq$ `B => A`, satisfying the subset interpretation of subtyping.

(b) Explain what the following code means:

```
trait myTrait[T <: Ordered[T]] {
  def value: T
  def foo(other: myTrait[T]):Boolean
}
```

where `Ordered[]` is a trait in Scala supporting the comparison operators.

A trait in Scala, like an interface in Java, specifies the methods that any class implementing the trait must support. In this case, `myTrait` is a generic trait parameterized by any type `T` that implements the `Ordered[T]` trait. Any class implementing `myTrait[T]` must define a `value` method that returns a `T` and a `foo` method that takes an object of a type that implements the `myTrait[T]` trait and returns a boolean.

(c) Define two unrelated Scala generic classes, `myClass[T]` and `yourClass[T]`, which both implement the `myTrait[T]` trait. The `foo()` method in both classes should use both the "this" object and the "other" object to compute its result.

```
 class myClass[T <: Ordered[T]](x:T) extends myTrait[T] {
  def value = x
  def foo(other: myTrait[T]) = value < other.value
}

class yourClass[T <: Ordered[T]](x:T) extends myTrait[T] {
  def value = x
  def foo(other: myTrait[T]) = value < other.value
}
```

(d) Can the `foo()` method of `myClass[T]` be passed an object of type `yourClass[T]` and vice versa? Explain.

Yes. `myClass[T]`'s `foo()` simply requires that its parameter implement `myTrait[T]`, which `yourClass[T]` does, and vice versa.

6. (a) What is the advantage of using a heap pointer to allocate heap structures rather than a free list?

Allocation is much faster using a heap pointer, which simply needs to be incremented. A free list must be traversed to find a free block of the right size.

(b) Why is copying GC used in conjunction with a heap pointer and mark/sweep garbage GC generally used in conjunction with a free list?

In copying GC, live objects in FROM space are copied to contiguous locations in TO space, thus the live objects are automatically compacted. Once GC is finished, the heap pointer simply needs to point to the first available location after the copied blocks. Mark/Sweep garbage collection doesn't perform compaction, the objects remain in place. Without compaction, a heap pointer can't be used – thus a free list is required.

(c) In a scenario in which the heap is large, but the number of live objects (and the amount of space occupied by those objects) is small, which of mark/sweep or copying garbage collection would you expect to consume less time? Explain?

The cost of Mark/Sweep GC is proportional to the size of the heap (due to the sweep phase), but the cost of copying GC is proportional to the number of live objects. Thus, in the given scenario, copying GC should consume less time.