Programming Languages
CSCI-GA.2110.001 Fall 2021

Final Exam

Please either put your answers directly on this sheet or put them in a plain text, Word, or PDF document. Then, upload the document to the Assignments→Final Exam tab on the course website.

1. **True/False (20 points, 2 points each)**. Circle or indicate the correct response.

   (a) T F Generic case classes in Scala can be declared as covariantly or contravariantly subtyped.

   (b) T F In ML, a function that takes a single parameter can be polymorphic so that its parameter is either a list of integers or a list of reals, but not a list of strings.

   (c) T F In the λ-calculus, 3 can be converted to (λx. x) 3 using β-conversion.

   (d) T F A grammar is a finite way of specifying a potentially infinite set of strings.

   (e) T F Suppose, in ML, the type of function `foo` is `'a list -> 'b list -> int`. In a call `foo L1 L2`, the type of the elements of the list `L1` must be different from the type of the elements of the list `L2`.

   (f) T F In the subset interpretation of subtyping, if class B is a subtype of class A then since a B object may contain more fields than an A object, the set of all A objects is a subset of the set of all B objects.

   (g) T F The regular expression `((a | b)* c)*` denotes the set containing the empty string and all strings consisting of at least one `c` and zero or more `a`'s and `b`'s.

   (h) T F On the very first garbage collection for a running program, a copying GC would copy the same number of live objects as a generational copying GC would (assuming the From heap in the copying GC is the same size as the youngest heap in the generational GC).

   (i) T F In Scheme, the `let*` construct is equivalent to nested `let` constructs.

   (j) T F The Church Rosser Theorem II implies that applicative order reduction, if it terminates, takes fewer steps than normal order reduction.

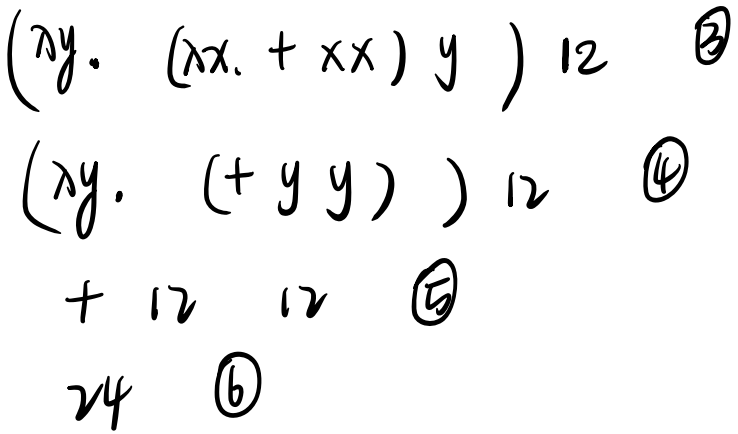
2. **Multiple Choice (28 points, 4 points each)**. Circle or indicate all answers that are correct, if any, for each question (there may be more than one correct answer or none at all).

   (a) Using applicative order reduction in the λ-calculus, the expression

   $$((\lambda x.\ (\lambda y.\ (x\ y))\ (\lambda x.\ +\ x\ x))\ ((\lambda y.\ *\ y\ 2)\ 6))$$

   reduces to normal form in how many reduction steps?

   i. None, there is no normal form.
   ii. 5
   iii. 6
   iv. 7



(b) In the λ-calculus, how would the result of the substitution $(\lambda x.\ E)[M/x]$ be different from the result of β-reduction applied to $(\lambda x.\ E)M$?

i. The result of the substitution would be $(\lambda x.\ E)$ and the result of the β-reduction would not be $(\lambda x.\ E)$.

ii. There would be no difference in the results, since β-reduction is defined as a substitution.

iii. The result of the substitution would be $(\lambda x.\ E)$ and the result of the β-reduction would be $(\lambda x.\ E[M/x])$.

iv. The result of the substitution would be $(\lambda x.\ E)$ and the result of the β-reduction would be $E[M/x]$.

(c) In Scala, if a generic class `C` is declared as "`class C[+T] {...}` " (where "`...`" is additional code), then:

T and super T

i. None of `C`'s methods can take a `T` object as a parameter.

ii. None of `C`'s methods can return a `T` object.

iii. Any generic class `D[]` that extends `C[]` must also be declared using `+` (e.g. "`D[+T]`").

iv. If B is a subtype of A, then `C[A]` is a subtype of `C[B]`.

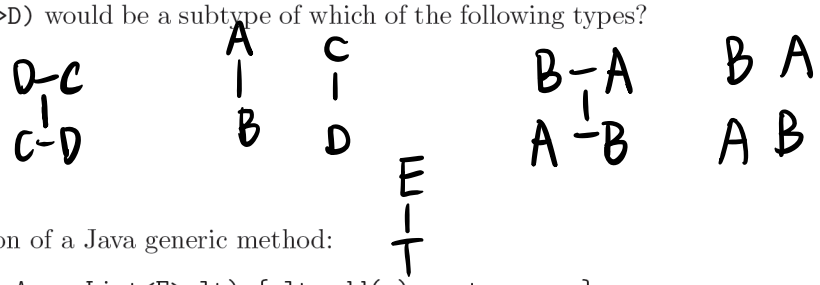
(d) In a language (like Scala) with function subtyping, if `B` is a subtype of `A` and `D` is a subtype of `C`, then `(A->B)->(C->D)` would be a subtype of which of the following types?

i. `(A->B)->(C->C)`

ii. `(A->A)->(D->C)`

iii. `(A->B)->(D->C)`

iv. `(B->B)->(C->D)`



(e) Consider the following declaration of a Java generic method:

```
<E, T extends E> T foo(T x, ArrayList<E> lt) { lt.add(x); return x; }
```

Which of the following method declarations are equivalent (in any program) to the one above? You can assume the programmer never explicitly specifies the type parameters in a call to `foo`.

i. `<T> T foo(T x, ArrayList<? super T> lt) { lt.add(x); return x; }`

ii. `<E, T extends E> T foo(T x, ArrayList<? super E> lt) { lt.add(x); return x; }`

iii. `<T> T foo(T x, ArrayList<T> lt) { lt.add(x); return x; }`

iv. `<T extends E> T foo(T x, ArrayList<E> lt) { lt.add(x); return x; }`

(f) Automatic storage reclamation (i.e. "Garbage Collection") is an improvement over programmer-specified storage reclamation for the following reasons:

i. A programmer is likely to allocate too many objects in the heap, compared to garbage collection.

ii. The free list method used by programmers is less efficient than the heap pointer method used by all garbage collectors.

iii. Programs in languages using garbage collection are less likely to have memory leaks than programs in languages with programmer-specified reclamation.

iv. Programmer-specified reclamation is more suspectible to reclaiming objects that are actually live than garbage collection is.

(g) Which of the following CFGs defines the set of all strings containing equal numbers of a's and b's (and no other letters or digits).

i. `S -> aSb | ϵ`

ii. `S -> aSb | bSa | ϵ`

iii. `S -> aS | bS | ϵ`

iv. `S -> aSa | bSb | aSb | bSa | ϵ`

3. **Fill in the blanks (15 points, 5 points each).** Your answer should be able to fit within each blank (or the equivalent on your sheet).

(a) In ML, suppose that:

- f is a function of type `('a list -> 'b list -> 'c list)`
- g is a function of type `'a -> 'b`
- h is a function of type `'b -> 'c`
- x is a variable of type `'a`

Write an expression involving every one of f, g, h, and x – and any built-in function or operator you want – such that the type of the expression is `'c`.

Answer: _____

(b) The following function, `revEq`, in Scheme takes two lists, `L1` and `L2`, and returns true if the reverse of `L2` is equal to `L1` (using `equal?` but without using `reverse`). For example, `(revEq '(1 (2 3) 4) '(4 (2 3) 1))` would return true and `(revEq '(1 (2 3) 4) '(1 (3 2) 4))` would return false.

```
(define (revEq L1 L2)
  (_____
    ((tempFn (lambda (L2 revL2)
               (cond ((null? L2) _____)
                     (else (tempFn _____))))))
     (tempFn L2 '())))
```

(c) Fill in the blanks below (only one line) so that the ML function `f` has type `('a -> 'b) -> ('a * 'b ) -> ('a list * 'b list)`

```
fun f _____ = _____
```

(Continued on next page)

4. **Short Programming Questions (10 points, 5 points each)**. Your code should be no more than 6 lines for each question.

(a) Consider the following Java generic method,

```
<T extends Vehicle> int addSpeedsUp(List<T> L) {
   int res = 0;
   for (T x: L) res += x.speed;
   return res; }
```

assuming that class `Vehicle` has already been defined and has a public `speed` field. Translate the above `addSpeedsUp` method into Scala, using Scala's built-in `List()` class and pattern matching (you can use recursion if you want). Again, you can assume class `Vehicle` has been defined in Scala.

```
def addSpeedUp[T<:Vechile ] (L:List[T]):Int = L match{
  case List( ) => 0
  case (head::res) => head.speed + addSpeedUp(res)
```

(b) Write an ML function `maxFn L x`, where `L` is a list of functions of type `'a -> int` and `x` is of type `'a`, that returns the function within `L` that returns the largest value when applied to `x` of all the functions in `L`. For example,
`maxFn [fn y => y+1, fn y => y * 2, fn y => y * -1] -4`
would return the third function of the list. You can assume that `L` is non-empty.

```
fun maxFn [e] x = e
| maxFn (l::ls) x =
    let result = maxFn ls x
    in if (l x) < (result x) then result
       else l end
```