UCLA Computer Science 131 (spring 2019) midterm
100 minutes total, open book, open notes,
No computer or any other automatic device. Write answers on test.
Please be brief; excessively long answers will be penalized.

Name: Student ID:

1	2	3	4	5	6	7	total

1a (12 minutes). For each of the following OCaml function definitions, give the type of the function and explain in words what the function does, from the caller's point of view. Assume the usual environment where '*.' means 'float' multiplication as in (3.0 *. 4.0), and where 'sin' means the 'float' trigonometric function as in (sin 1.5).

```
let q f x = f x x
```

let q a b c = c a b a' -> b -> (a' -> b -> c') -> 'c

a b a b b

'a -> 'a -> ('b -> 'a) -> 'b

```
let s = q ( *. )
```

From callers perspective, defines as

float -> float

```
let p a b x = a (b x)
```

'a -> ('a -> ('b -> 'c))

```
let c = p s sin
```

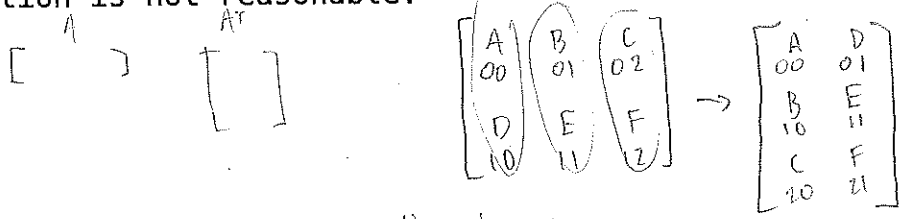
int -> calculates s

1b (3 minutes). In 1a, why does s's definition use '(*.)' and not '(fun x y -> x *. y)' or '(*.)' or simply '*.'?

It's not a function definition that takes arguments x, y. Instead it is

2. Recall that the transpose of an M×N matrix A is an N×M matrix B such that A[i][j] = B[j][i] for 0≤i≤M, 0≤j≤N.

2a (12 minutes). Suppose we represent a matrix of items as a list of list of items. Write a function loltp that does list-of-list transposition, that is, it takes a list of list of values that represents a matrix A, and returns a list of list of values that represents the transpose of A. For example, (loltp [["a";"b";"c"];["d";"e";"f"]]) returns [["a";"d"];["b";"e"];["c";"f"]]. If you cannot reasonably solve the problem in general, make sure that it at least succeeds on a 2×3 test case such as in the example given, and explain why a more-general solution is not reasonable.



Take in a list of lists, go through each one and append head to new list. Create a list of lists

```
let rec loltp lists = match lists with
  | [] -> []
  | h::t -> ;
```

take head here—and append to list created by recursive call on rest of lists.

2b (2 minutes). What is the type of your loltp function?

'a list -> 'a list

2c (4 minutes). Give an example value that you can pass to loltp that OCaml's type checking will accept but will cause a runtime error; or explain why no such value is possible.

[["a";"b";"c"];["d";"e";"f";"g"]]

3a (12 minutes). Suppose we instead represent a matrix of items in OCaml as a tuple of tuple of items. Write a function tottp that does tuple-of-tuple transposition; that is, it acts like loltp except it operates on the tuple-of-tuple representation. For example, (tottp (("a","b","c"),("d","e","f")) returns (("a","d"),("b","e"),("c","f")). Again, if you cannot reasonably solve the problem in general, make sure that it at least succeeds on a 2×3 test case such as in the example given, and explain why a more-general solution is not reasonable.

Convert to lists

3b (2 minutes). What is the type of your tottp function?

(string * string * string) * (string * string * string) → (string * string) * (string * string) * (string * str

3c (4 minutes). Give an example value that you can pass to your tottp function that OCaml's type checking will accept but will cause a runtime error; or explain why no such value is possible.

Given that the above definition is called, there is no value that will given an error. If it took more complex types a runtime error could be caused by a tuple which is a doubly nested tuple.

4. Given a grammar, a nonterminal symbol is called "nullable" if a valid parse tree rooted at the symbol contains no terminal symbols. For example, consider the following Homework 1 style grammar:

```
let nullg =
  ["Expr", [T"("; N"Expr"; T")"];
   "Expr", [N"Expr"; N"Ops"; N"Expr"];
   "Expr", [T"ID"];
   "Ops", [N"Op"; N"Op"];
   "Ops", [N"Op"; N"Op"; N"Ops"];
   "Op", [T"+"];
   "Op", [];
   "Op", [T"*"]]
```

In this grammar, the nonterminal "Op" is nullable because it can produce the empty list immediately in the second-to-last rule, and the nonterminal "Ops" is nullable because it can produce two "Op" nonterminals, each of which can produce the empty list. However, "Expr" is not nullable.

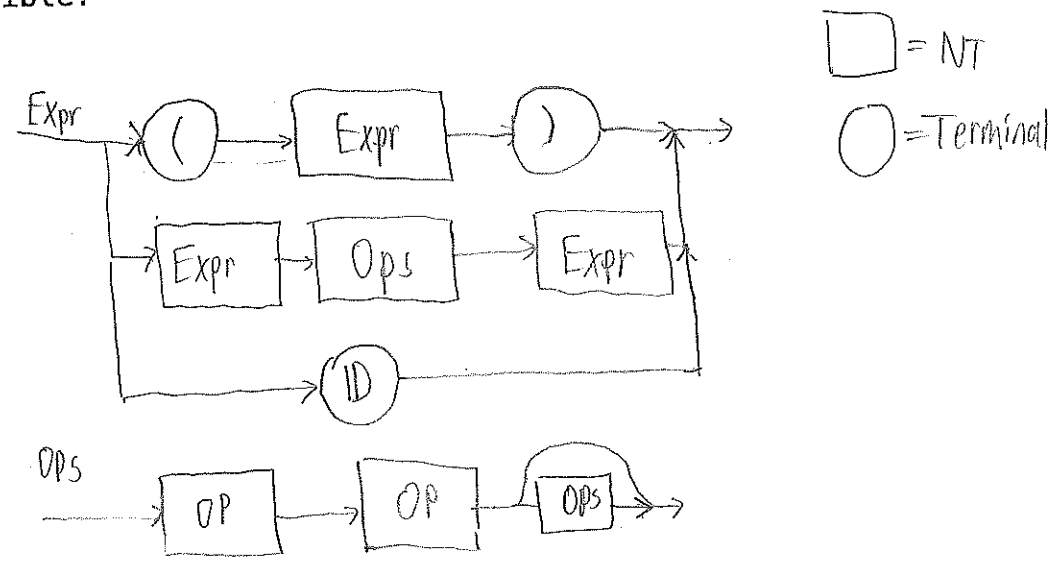
4a (12 minutes). Write an OCaml function (nullables G) that returns the set of nullable nonterminals in the grammar G, representing the set as a list. The members of the returned list can be in any order and the list can contain duplicates. For examples, (nullables nullg) might return ["Ops";"Op"]. Your function can assume the functions (subset A B), (equal_sets A B), (set_union A B), (set_intersection A B), (set_diff A B), and (computed_fixed_point EQ F X) that were assigned in Homework 1. However, your function should not use any functions in the OCaml standard library. You can write auxiliary functions to help implement your function.

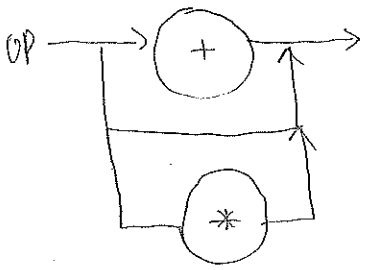
For left hand side return

let rec nullables grammar =

4b (5 minutes). If you translate 'nullg' to Homework 2 style, will it cause the corresponding matcher to loop in some cases? If so, give an example of how the matcher would loop; if not, explain why not.

4c (10 minutes). Convert 'nullg' to a syntax diagram that is as simple as possible.

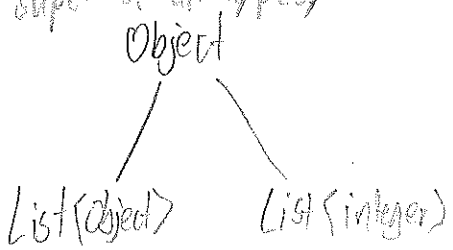




5 (4 minutes). In OCaml, 'int list' is a subtype of ''a list'. However, in Java, 'List<Integer>' is not a subtype of 'List<Object>'. Explain the seeming discrepancy, and give some other List type that 'List<Integer>' *is* a subtype of in Java.

Java is strongly typed
List(Integer) is not a subtype of List(object)
bc List(object) is actually a subtype of object along with list(integer), but it is NOT a supertype of Object (which is a super of all types)

List(integer) is a subtype of object



6 (6 minutes). In C, the _Noreturn keyword marks functions that do not return; for example, the 'exit' function is declared this way:

```
_Noreturn void exit(int);
```

because it accepts an integer argument and never returns. Compilers can optimize calls to _Noreturn functions, e.g., by generating code that does not bother to save the call's return address (because the return address is never used).

Is a function type containing the _Noreturn keyword a subtype of the same function type without _Noreturn? Or vice versa? Or neither? Briefly explain.

C operates on structural equivalence meaning that types are the same if they have the same behavior
Considering that behavior with _Noreturn and without it yields different results, I can assert that these aren't the same. I think _Noreturn is a subtype because it is possible to define a function without noReturn that has identical behavior.

7 (12 minutes). The Java designers were willing to give up some performance in exchange for reproducible results. For example, although C allows a compiler to evaluate a call's arguments in any order, Java requires the compiler to evaluate them left-to-right. Java's rule prevents some optimizations but means that code is more likely to yield the same results on different platforms. A Java compiler is still allowed to execute the arguments out of order for speed, so long as the user can't tell the difference.

A significant reliability problem in Java comes from race conditions in multithreaded programs. Couldn't the Java designers have traded performance in exchange for avoiding race conditions? That is, couldn't the Java designers have said that a Java compiler must evaluate multithreaded code as if the first runnable thread is the only running thread? (By "first" I mean the earliest-created thread.) As before, the Java compiler would still be allowed to execute code in parallel for speed, so long as the user can't tell the difference other than in performance.

If this idea is impossible, explain why that is so. Or if it is possible but impractical, explain why. Or if it is a reasonably practical suggestion, give a good reason why the Java designers did not take the suggestion.

To the best of my understanding, I thought the designers of Java tried something similar to this. They attempted to implement a locking mechanism on methods with a complementary unlocking method. These spin locks have one problem though — they have an extremely detrimental effect on efficiency because essentially only one lock may be held at a time. This is probably why the Java designers decided not to do this.