Midterm Exam

CSC 4330 / 6330 *Programming Language Concepts*

October 16, 2023

This midterm exam is worth 100 points (the number of points for each exercise is indicated beside the exercise). This exam is open book and open internet, you can (and should) use `GHCi` (*e.g.*, on `repl.it`), Prolog (*e.g.*, `https://swish.swi-prolog.org`), *etc.*, as long as the solution you draft is your own. As always, partial credit is available, so do what you can!

You may submit your solution in the text window on iCollege, as a `.txt` file, or in some other commonly used format, such as `.pdf`, `.docx`, *etc.* Once finished, please upload your solution to iCollege by the end of the exam period, which is at **1pm on October 18th**. The exam will go offline at 2pm, *i.e.*, submission will become *unavailable* one hour after the end of the exam period.

Exercises

1. In each of a., b. and c., the Haskell function (left) and the Prolog predicate (right) do the same thing. What do they do in each of the cases a., b. and c.?

   a. (10 points).

```
foo :: Num a => a -> Int -> a
foo _ 0 = 0
foo n m = n + foo n (m-1)
```

```
foo(_,0,0).
foo(N, M, Result) :-
   M > 0,
   M1 is M - 1,
   foo(N, M1, Result1),
   Result is N + Result1.
```

   b. (10 points).

```
bar :: Ord a => [a] -> Bool
bar xs =
   and [x <= y | (x,y) <- zip xs (tail xs)]
```

```
bar([]).
bar([_]).
bar([X,Y|Tail]) :-
   X =< Y,
   bar([Y|Tail]).
```

   c. (10 points).

```
baz :: Eq a => [a] -> Bool
baz [] = True
baz [_] = True
baz (x:xs) =
   if x == last xs
     then baz (init xs)
     else False
```

```
baz([]).
baz([_]).
baz([X|Xs]) :-
   append(Ys, [X], Xs),
   baz(Ys).
```

2. (30 points: 15 points for each of the Haskell and Prolog implementations). Write a Haskell function `evenLength :: [a] -> Bool` and the corresponding Prolog predicate `evenLength`, which returns (or resolves to) `true` when the single list argument passed to it has even length.

   Note: that these must be written *from scratch*, so no previously defined functions may be used, *e.g.*, the Prelude `length` function (or the Prolog `length` predicate) may *not* be used — your solutions will be recursive. You may, of course define auxiliary helper functions (which also must be written from scratch), *e.g.*, the appropriate `oddLength :: [a] -> Bool` might be useful in Haskell, and similarly, an `oddLength` predicate in Prolog.

   The idea is that in Haskell, *e.g.*, `evenLength [1,2,3,4]` would return `True`, and `evenLength "hey"` would return `False`, while in Prolog, *e.g.*, the query `evenLength([1,2,3,4]).` would resolve to `true`, and the query `evenLength([a,b,c]).` would resolve to `false`.

3. In mathematics, the *Cartesian product* $X \times Y$ of two sets X and Y is $\{(x, y) \mid x \in X,\ y \in Y\}$, for example, if $X = \{1, 2, 3\}$ and $Y = \{4, 5, 6\}$, then

$$X \times Y = \{(1,4),\ (1,5),\ (1,6),\ (2,4),\ (2,5),\ (2,6),\ (3,4),\ (3,5),\ (3,6)\}$$

   In Haskell, we can represent such sets X and Y using *lists*.

   a. (15 points). Write a function

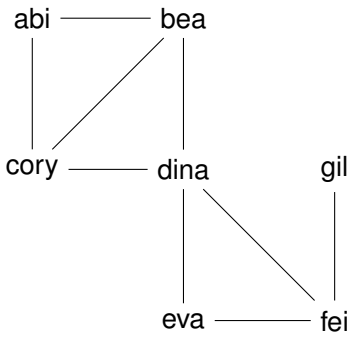
      ```
      cartesianproduct :: [a] -> [b] -> [(a,b)]
      ```

      which computes the above Cartesian product using a *list comprehension*, and the result of evaluating `cartesianproduct [1,2,3] [4,5,6]` for your definition of this function.

   b. (5 points). Why does `cartesianproduct` have the type that it does?

4. Let us go back to the network we have been studying in the previous few homework assignments. Only, this time, it is a social network involving people who are connected by a `friend` relationship, represented below: as Prolog code on the left, and a graphical representation on the right. Note that this `friend` relationship is *symmetric*, since, on the left, for each pair `X`, `Y` of people, if `friend(X,Y)`, then `friend(Y,X)`, and in the graphical representation on the right, the "edges" do not have direction.

```
friend(abi, bea).     friend(bea, abi).
friend(abi, cory).    friend(cory, abi).
friend(bea, cory).    friend(cory, bea).
friend(bea, dina).    friend(dina, bea).
friend(cory, dina).   friend(dina, cory).
friend(dina, eva).    friend(eva, dina).
friend(dina, fei).    friend(fei, dina).
friend(eva, fei).     friend(fei, eva).
friend(fei, gil).     friend(gil, fei).
```



   a. (10 points). Write the Prolog predicate `friendofafriend(X,Y)` which is `true` when `X` has a friend who is a friend of `Y`. *Note*: that this does not preclude `X` from also being a friend of `Y`, but it *does* preclude `X` from being a friend of a friend of itself, *i.e.*, `friendofafriend(X,X)` should be `false`.

b. (5 points). Use this `friendofafriend` predicate to list all of the people who are a friend of a friend of `cory`. *Note*: take care that `cory` is not a friend of a friend of `cory`, as mentioned above. Some people are listed twice, why is this?

c. (5 points). What problem do we encounter if we try to use the predicates `connection` or `path` from the previous homework assignments, and why?