The University of British Columbia
Department of Computer Science
Final Examination — Fall 2016
Computer Science 312
Functional and Logic Programming

Total marks: 120
Time: 150 minutes

Question 1 [20 marks]

(a) [3 marks] Consider the `foo` function defined in Haskell as follows:

```
foo [] y = y
foo (a:b) c = a: foo c b
```

Give the result of `foo [1,3] [9,7,4]`

(b) [4 marks] Write a predicate `foo(Xs, Ys, Zs)` in Prolog, such that *Zs* is the result of function `foo Xs Ys`

(c) [4 marks] What are all of the answers to the Prolog query `foo(Y, [2,7,4], Z)`? (Assume that ; is pressed after each answer).

(d) [4 marks] In Haskell, implement a function that returns the power set of a set. A power set of a set is the set of all its subsets. Sets are represented as lists without duplicates; the order of the elements does not matter. You may use recursion, list comprehensions, and the builtin functions ++, and : but no other built-in functions. It should have the following behaviour:

```
*Main> powerset [1,2,3]
[[1,2,3],[1,2],[1,3],[1],[2,3],[2],[3],[]]
*Main> powerset [2,3]
[[2,3],[2],[3],[]]
*Main> powerset [3]
[[3],[]]
*Main> powerset []
[[]]
```

(e) [5 marks] Implement the powerset function in Prolog. You can use whatever helper functions you define, but no built-in predicates. The order of the elements in any of the lists do not matter. (You can get full marks for this question if you do not use append. If you use append, whether you write it yourself or not, the maximum you can get is 3 marks.) It should have the following behaviour, perhaps with elements in a different order:

```
?- powerset([1,2,3],R).
R = [[1, 2, 3], [2, 3], [1, 3], [3], [1, 2], [2], [1], []] ;
false.
```

Question 2 [20 marks]

The idea of a merge sort is to take a list, split it into two lists of approximately equal sizes, recursively sort each of these lists, and then merge the two sorted lists into one.

David got very mixed up writing a merge sort, and wrote part in Prolog and part in Haskell. You are to write a version in the other language, keeping the structure and the naming of variables the same as much as possible. Your Prolog program must not give multiple answers.

(a) [5 marks] David's Prolog version of *split*, to split a list into two lists of approximately equal size, is:

```
split([],[],[]).
split([X],[X],[]).
split([X,Y|R],[X|R1],[Y|R2])
     :- split(R,R1,R2).
```

Give a corresponding recursive Haskell function called `split`, that takes a list and returns a pair of lists. You may use whatever Haskell language constructs you like, but do not use any functions other than those required to construct lists and tuples. `split` is the only function you should define.

(b) [5 marks] David's Haskell version of *merge*, which takes two sorted lists and returns a sorted list containing the elements of both lists.

```
merge [] l = l
merge l [] = l
merge (a:l1) (b:l2)
    | a <= b = a: merge l1 (b:l2)
    | otherwise = b: merge (a:l1) l2
```

Write a corresponding Prolog predicate called `merge` that takes in two lists and returns a list. You may use `dif` (not equals), and the comparisons =< (less than or equal to) and > (greater than), but no other predicates. `merge` is the only other function you should define.

(c) [5 marks] Give a Haskell version of merge sort that follows the description at the start of the question. It can use any of the Haskell language constructs (except for list comprehensions), but no other functions except split and merge (defined on previous page). It must have type:

```
msort :: Ord t => [t] -> [t]
```

(d) [5 marks] Give a Prolog version of merge sort. Call your predicate `msort`. It must only produce one answer (a retry should just return). You may only use the predicates of the previous page (split and merge).

Question 3 [20 marks]

This function is supposed to take a list and check whether or not it is a sorted list:

```
sorted :: [a] -> Bool
sorted [] = True
sorted (x:y:xs) = x < y && (sorted y:xs)
```

This was put into the file `sorted.hs`. As it is, the function has two compile time and one run time error. The line number and the first line of the error message are given below. When answering each subquestion, assume the bugs in previous subquestions have been fixed. For each case, briefly explain why the occurs and propose a fix.

(a) [4 marks]

```
sorted.hs:3:29:
    Couldn't match expected type 'Bool' with actual type '[Bool]'
```

Why error occurred:
Fix:

(b) [4 marks]

```
sorted.hs:3:21:
    No instance for (Ord a) arising from a use of '<'
```

Why error occurred:
Fix:

(c) [4 marks]

```
Main> sorted [1,2,3,4]
*** Exception: finalerr.hs:(2,1)-(3,42): Non-exhaustive patterns in
function sorted
```

Why error occurred:
Fix:

(d) [6 marks] Consider the following Prolog program:

```
translate([],[],_).
translate([X|Xs],[Y|Ys],Dict) :-
    member((X,Y),Dict),
    translate(Xs,Ys,Dict).
```

and this query:

```
   ? German = [wir, betreten, feuer, trunken, himmlische, dein, heiligtum],
English = [we, enter, 'drunk with', fire, 'heavenly being', your, sanctuary],
translate(German, English, Dictionary), translate([wir, trunken, feuer], T1,
Dictionary), translate(T2, [your, heavenly, being],Dictionary).
```

What would Prolog's output be for T1 and T2 and Dictionary:

```
T1 =
T2 =
Dictionary =
```

(e) [2 marks] Given what has been learned in this course, suggest a better way to do translation.

Question 4 [20 marks]

Consider representing 9×9 matrices as functions from indices to values. That is, `m i j` is the (i,j)-th element of matrix *m*. All matrices in the question are assumed to be 9×9. All indices and values can be assumed to be of type Int.

(a) [4 marks] Write a Haskell function *transpose* that takes a matrix and returns its transpose. The transpose of martix m is a matrix m' such that $m'_{ij} = m_{ji}$.

(b) [4 marks] Write a Haskell function *replace* that takes a matrix m, indices i and j and a number v, and returns a matrix that is the same as m but with the (i,j)-th element having value v.

(c) [4 marks] Write a Haskell function *sum_all* that takes a matrix and returns the sum of all of the elements of the matrix. (You may use Haskell's +, sum, list comprehensions, foldl and/or foldr).

(d) [4 marks] Suppose someone decided to represent matrices in Prolog using a relation

$$matrix(M, I, J, V)$$

where M denotes a matrix, I and J integer indicies, and the relation is true when the (I,J)-th element of matrix M has value V.

Suppose the transpose of matrix M (defined on previous page) is represented as $transpose(M)$, where *transpose* is a Prolog function. Define the *matrix* predicate so that it also works for transposes of matrices.

(e) [4 marks] The inferred type of transpose in Haskell is not informative. What could be done in Haskell to make the type more informative? Does your solution provide any extra type-checking in Haskell? Explain why or why not.

Question 5 [20 Marks]

Consider the declaration:

```
data BSTree k v = BSEmpty
                | BSNode k v (BSTree k v) (BSTree k v)
instance (Show k, Show v) => Show (BSTree k v) where
      show t = foldr (\ e r -> (show e) ++ "\n"++ r) "" (tolist t)
tolist BSEmpty = []
tolist (BSNode key val l r) = tolist l ++ [(key,val)] ++ tolist r
```

(a) [3 marks] What do *k* and *v* represent in the first line?

(b) [3 marks] What is the inferred type of tolist?

(c) [3 marks] For the function:

```
myst t = length (tolist t)
```

Give an English description of myst that does not rely on the user knowing what length or tolist are.

(d) [3 marks] Explain in English (suitable for one of your peers who is just starting to learn Haskell) what the line that starts with `instance` and the line that follows are doing. Do not try to explain the meaning of what is on the right of =. (To get full marks you need to explain both the part on the left of => and the part on the right.)

(e) [4 marks] Define two different trees that, when shown, give the output

```
(2,"fun")
(4,"me")
```

(f) [4 marks] Define a Haskell function *value_in_tree* that takes a binary search tree defined using *BSTree* and a value *v* and returns *True* if *v* is the value for some key, and *False* otherwise.

Question 6 [20 Marks]

(a) [10 marks] Suppose you were trying to convince Haskell programmers of some of the advantages of Prolog. Give one feature of Prolog that could plausibly be included in Haskell, but is lacking in Haskell. Use full sentences in your answers.

  i) [2 marks] What is the feature?

  ii) [4 marks] Give an example in Prolog of what can be done with the feature.

  iii) [4 marks] What currently must be done in Haskell to overcome the lack of this feature?

(b) [10 marks] Suppose you were trying to convince Prolog programmers of some of the advantages of Haskell. Give one feature of Haskell that could plausibly be included in Prolog, but is lacking in Prolog. Use full sentences in your answers.

  i) [2 marks] What is the feature?

  ii) [4 marks] Give an example in Haskell of what can be done with the feature.

  iii) [4 marks] What currently must be done in Prolog to overcome the lack of this feature?