The University of Melbourne
School of Computing and Information Systems

Semester 2, 2019 Sample Assessment

COMP30020
Declarative Programming

[redacted] Sample Answers Included [redacted]

Reading Time: 15 minutes

Writing Time: 2 hours

This paper has 9 pages.

Total marks for this paper: 100

Authorised Materials:

Writing instruments (e.g., pens, pencils, erasers, rulers).
No other materials and no electronic devices are permitted.

Instructions to Invigilators:

The exam paper must remain in the exam room and be returned to the subject coordinator.

Instructions to Students:

The marks for each question are listed at the beginning of the question. You should attempt all questions. Use the number of marks allocated to a question as a rough indication of the time to spend on it. We have tried to provide ample space for your answers; do not take the amount of space provided for an answer as an indication of how much you need to write.

This paper must *not* be lodged with the university library.

This page intentionally left blank

Question 1 [12 marks]

For each of the following Haskell expressions, give its **type** (which may be a function type, may include type variables, and may include type class constraints) or indicate that it represents a type error. You need not write anything other than the type, or that it is an error.

(a) `(<)`

`(<) :: Ord a => a -> a -> Bool` ⇐

(b) `map (+3)`

`Num a => [a] -> [a]` ⇐

(c) `foldr`

`(a -> b -> b) -> b -> [a] -> b` ---or--- `Foldable t => (a -> b -> b) -> b -> t a -> b` ⇐

(d) `Nothing`

`Maybe a` ⇐

(e) `zip [True,True,False]`

`[a] -> [(Bool, a)]` ⇐

(f) `flip filter "hello"`

`(Char -> Bool) -> [Char]` —**or**— `(Char -> Bool) -> [Char]` ⇐

Question 2 [12 marks]

For each of the following Haskell expressions, give its **value**, or explain why it will produce an error or fail to terminate. Assume the `Data.List` library is loaded, which defines the `sort` function.

(a) `map (length < 3) [[1],[1,2,3]]`

Type error ⇐

(b) `filter (not.(==3)) [1,2,3]`

`[1,2]` ⇐

(c) `let e = head [] in 3`

`3` ⇐

(d) `map fst $ filter snd $ zip "abcde" [True,True,False,True]`

`"abd"` ⇐

(e) `head $ sort $ zip [3,0,0,2,0] $ reverse [9,0,0,4,8]`

`(0,0)`

⇛

(f) `map snd $ sort $ zip "decl" [1..]`

`[3,1,2,4]`

⇛

Question 3 [30 marks]

Consider the following Haskell type for ternary trees:

```
data Ttree t = Nil | Node3 t (Ttree t) (Ttree t) (Ttree t)
```

Suppose we have a `Ttree` of Doubles and we want a function to find the average of the numbers in the tree. Write a Haskell function which performs this task. If the `Ttree` is empty, your function should return 0.0. Include type declarations for all your functions. To obtain maximum marks, your code should use a single traversal over the tree and have $O(N)$ worst case time complexity.

Sample Answer to Question 3 ⇐

Solution using custom fold function for Ttrees

```
ttreeAverage1 :: Ttree Double -> Double
ttreeAverage1 Nil = 0.0
ttreeAverage1 tt =
  let (sum,count) = foldTtree plusCount (0.0,0.0) tt
  in  sum / count


plusCount :: Double -> (Double, Double) -> (Double, Double)
plusCount n (sum,count) = (sum+n, count+1)


foldTtree :: (a -> b -> b) -> b -> Ttree a -> b
foldTtree _ acc Nil = acc
foldTtree f acc (Node3 n left mid right)
  = let acc1 = foldTtree f acc  right
        acc2 = foldTtree f acc1 mid
        acc3 = foldTtree f acc2 left
    in  f n acc3
```

Alternative full-credit solution

```
ttreeAverage2 :: Ttree Double -> Double
ttreeAverage2 Nil = 0.0
ttreeAverage2 tt =
  let (sum,count) = ttAverage' tt
  in  sum / count

ttAverage' :: Ttree Double -> (Double,Double)
ttAverage' Nil = (0.0,0.0)
ttAverage' (Node3 n left mid right) =
  let (suml,countl) = ttAverage' left
      (summ,countm) = ttAverage' mid
```

```
      (sumr,countr) = ttAverage' right
  in  (n+suml+summ+sumr, 1+countl+countm+countr)
```

Partial credit solution using two passes

```
-- Partial credit, two-pass solution.  This answer would still get
-- most of the marks.
ttreeAverage3 :: Ttree Double -> Double
ttreeAverage3 Nil = 0.0
ttreeAverage3 tt = ttSum tt / ttCount tt

ttSum :: Ttree Double -> Double
ttSum Nil = 0.0
ttSum (Node3 n left mid right) =
  n + ttSum left + ttSum mid + ttSum right

ttCount :: Ttree Double -> Double
ttCount Nil = 0.0
ttCount (Node3 n left mid right) =
  1 + ttCount left + ttCount mid + ttCount right
```

Question 4 [10 marks]

Give the formal semantics (meaning) of the following Prolog program. Recall that the formal semantics of a logic program is the set of *ground unit clauses* that would give the same answers to all queries as the program itself. English descriptions of the meanings of the programs will receive no credit.

```
p(a).    p(b).

q(X,Y) :- p(X), p(Y).

r(a,c).    r(d,b).

s(X,Y) :- q(X,Y), r(X,_), r(_,Y).
```

Sample Answer to Question 4

⇛

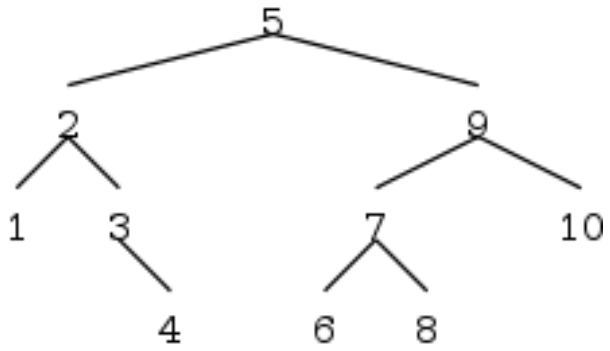
```
p(a).   p(b).
q(a,a).   q(a,b).   q(b,a).   q(b,b).
r(a,c).   r(d,b).
s(a,b).
```

Question 5 [20 marks]

For this question, we will represent a set of integers as a binary tree in Prolog, using the atom `empty` to represent an empty tree or node, and `tree(L,N,R)` to represent a node with label `N` (an integer), and left and right subtrees `L` and `R`. Naturally, we also insist that `N` be strictly larger than any label in `L` and strictly smaller than any in `R`. We do not require that the tree be balanced. For example,

```
tree(tree(tree(empty, 1, empty),
          2,
          tree(empty, 3, tree(empty, 4, empty))),
     5,
     tree(tree(tree(empty,6,empty),
               7,
               tree(empty,8,empty)),
          9,
          tree(empty, 10, empty)))
```

is one possible representation of the set of numbers from 1 to 10. It might be visualized as



Write a predicate `intset_insert(N, Set0, Set)` such that `Set` is the same as `Set0`, except that `N` is a member of `Set`, but may or may not be a member of `Set0`. That is, either `N` is a member of `Set0` and `Set = Set0`, or `N` is not a member of `Set0` and is a member of `Set`, and other than that, `Set` is the same as `Set0`. This predicate must work as long as `N` is bound to an integer and `Set0` is ground.

Hint: Prolog's arithmetic comparison operators are `<`, `>`, `=<` (not `<=`), and `>=`. You can also use `=` and `\=` for equality and disequality.

Sample Answer to Question 5 ⇐

```
intset_insert(N, empty, tree(empty,N,empty)).
intset_insert(N, tree(Left,Val,Right), Result) :-
```

```
        (   N = Val
        ->  Result = tree(Left,Val,Right)
        ;   N < Val
        ->  Result = tree(Left1,Val,Right),
            intset_insert(N, Left, Left1)
        ;   Result = tree(Left,Val,Right1),
            intset_insert(N, Right, Right1)
        ).
```

Question 6 **[16 marks]**

Following is a definition of a Prolog predicate to compute the sum of a list of numbers.

```
sumlist([], 0).
sumlist([N|Ns], Sum) :-
        sumlist(Ns, Sum0),
        Sum is N + Sum0.
```

Fill in the blanks in the following transformation of this code to be tail recursive.

```
sumlist(List, Sum) :- [sumlist(List, 0, Sum).]
⇛
sumlist( [[], Sum, Sum] ).
⇛sumlist([N|Ns], Sum0, Sum) :-
        [Sum1 is Sum0 + N] ,
⇛       sumlist( [Ns, Sum1, Sum] ).
⇛
```

— End of Exam —