The University of Melbourne
Department of Computing and Information Systems

Semester 2, 2015 Sample Assessment

COMP30020/COMP90048 Declarative Programming

Reading Time: 15 minutes Total marks for this paper: 100

Writing Time: 2 hours

This paper has 5 pages.

Authorised Materials:

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

Instructions to Invigilators:

Each student should receive a script book.

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

Instructions to Students:

Answer each question in the script book provided. **Begin each question on a fresh page, and write the question number in the upper right corner of the page.** Any unreadable answers will be considered wrong.

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.

This paper should *not* be lodged with Baillieu Library.

This page intentionally left blank

Question 1 [12 marks]

What would be printed if the following expressions are typed into `ghci`? Recall `:t` tells `ghci` to just print the type of the expression. If any expressions would result in errors, just give the general nature of the error, for example "type error", rather than detailed error messages.

(a) `:t (<)` Ord t => t -> t -> Bool

(b) `:t map (+3)` Foldable t, Num a => t a -> t a

(c) `:t zip [True,True,False]` [b] -> [(Bool, b)]

(d) `map (length < 3) [[1],[1,2,3]]` type error map (\x -> length x < 3) [[1], [1,2,3]]

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

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

Question 2 [18 marks]

In Haskell, the less than operator (`<`) can be applied to many different data types, including new types created by the programmer. Briefly (in two or three paragraphs) explain the feature of the Haskell type system which allows this. We would normally expect `<` to obey certain laws such as transitivity (`x<y` and `y<z` implies `x<z`). Does the Haskell implementation enforce such laws? If so, explain how. If not, explain whether it is important for the programmer to make sure they are obeyed.

Haskell has a strong type class system which allows this. For example, the less than operator can be applied to any type of data which belongs to the Ord class. So as long as a new type of data created by the programmer belongs to Ord class, the less than operator can be applied to the data. The Haskell doesn't enforce the transitivity law, but the it is important that the programmer makes sure they are obeyed since otherwise it will be unreliable when multiple data is compared.

Question 3 [30 marks]

Consider the following Haskell type for ternary trees:

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

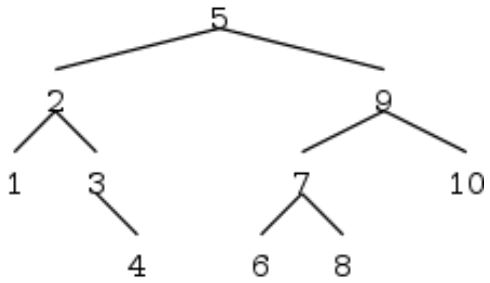
Suppose we have a `Ttree` of Doubles and we want a function to find the average of the numbers in the tree. Write a Haskell function which performs this task. If the `Ttree` is empty, your function should return 0.0. Include type declarations for all your functions. To obtain maximum marks, your code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

Question 4 [30 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.

Question 5 **[10 marks]**

Following is a definition of a Prolog predicate to compute the sum of a list of numbers. Transform this definition to be tail recursive. You need not show the steps of your transformation.

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

— End of Paper —