The University of Melbourne

Department of Computer Science and Software Engineering

433–326

Declarative Programming

Semester 1, 2010

Exam Duration: Two hours.

Reading Time: Fifteen minutes.

Length: This paper has 3 pages including this cover page.

Authorised Materials: None.

Instructions to Invigilators:

Instructions to Students: Answer all of the questions in the script book provided. Start your answer for each question on a separate page and write the question number at the top of the page. The number of marks for each question (totaling 60) is given; where no breakdown is given for multi-part questions, each part is worth the same number of marks. Minor errors in Haskell and Mercury syntax etc, will be tolerated but you should add comments if you feel it will aid understanding of what you have written.

Calculators: Calculators are not permitted.

Baillieu Library: Exam paper to be held by the library.

Question 1 [5 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 map`

(b) `map tail [[1,2],[]]`

(c) `filter (length.(==2)) [[1,2],[]]`

(d) `foldl (+) 4 [1,2]`

(e) `let e = 3:e in head e`

Question 2 [6 marks]

Assume the following definitions are part of a Mercury module.

```
:- pred append3(list(T), list(T), list(T), list(T)).
:- mode append3(in, in, in, out) is det.
:- mode append3(out, out, out, in) is nondet.

append3(A, B, C, ABC) :-
    append(A, B, AB), append(AB, C, ABC).
```

What would be computed by the following calls? Give bindings for all the variables in the call (assume they are all mode `out`) for each solution to the call; if there are no solutions just write "no solutions". If any calls would result in errors, just give the general nature of the error, for example "type error", rather than detailed error messages.

(a) `append3([1], [[]], [2,3], Ds)`

(b) `append3(As, Bs, Cs, [4])`

(c) `append3(As, As, Cs, [5,5])`

Question 3 [5 marks]

In Mercury it is always possible to declare procedures to be `nondet`. This makes programming simpler in that you don't have to think about determinism. However, most Mercury procedures written by experienced programmers are declared `det`. Briefly explain two important advantages of giving more restrictive determinism declarations (such as `det`) where possible.

Question 4 [4+5+5=14 marks]

The C type system supports *structs*, which contain the values of several fields, and *unions*, which also have several fields but can only hold the value of one field at any one time.

(a) How are C structs and unions related to the algebraic data types of Haskell and Mercury?

(b) Describe two important advantages of algebraic data types compared to structs and unions.

(c) The type systems of Haskell and Mercury also support *parametric polymorphism*. Explain how this is an advantage compared to C.

Question 5 [6+6+6+12=30 marks]

Consider the Haskell type definitions below. The first is for ranges of integers: `Range min max` represents the range of integers from `min` to `max-1`, inclusive. The intention is `min<max` always (there are no empty ranges). The second is for ternary (three-way) trees.

```
data Range = Range Int Int
data Ttree a = Nil | Node3 a (Ttree a) (Ttree a) (Ttree a)
```

(a) Write a Haskell function

```
foldr_ttree :: (a->b) -> b -> Ttree a -> b
```

which is analogous to *foldr* for lists.

(b) Suppose we want to take a ternary tree of ranges and return the range with maximum size, where the size of `Range min max` is `max-min`. If there are several ranges with the same maximal size, any one can be returned. Write a function `max_range` (you should choose an appropriate type signature for it) which does this using `foldr_ttree` to traverse the tree.

(c) Assume the trees of ranges are arranged like binary search trees, in that ranges with small `min` and `max` are in the left subtree, those with large `min` and `max` are in the right subtree and ranges which overlap with the range at the root are in the middle subtree. Specifically, for each valid tree of ranges `Node3 (Range rmin rmax) l m r`, all ranges `Range min max` in `l` have the property `max<=rmin`, all ranges in `r` have the property `rmax<=min` and all ranges in `m` have neither of these properties. Write a function to insert a range into a tree of ranges, maintaining these properties.

(d) Given the properties of trees of ranges in (c) above, write a more efficient version of `max_range` which can avoid searching parts of the tree. Note that the size of the range depends on the *difference* between `max` and `min`. It may be that both `min` and `max` are large but the size of the range is small, for example. Hint: when you visit a left subtree there is a constraint on the `max` value in ranges, when you visit a right subtree there is a constraint on `min` values and if you have both constraints you may be able to avoid searching.