1999

p1q5
LCP

Foundations of Computer Science (Paper 1)

Describe ML's facilities for treating functions as data, giving examples of their use in programs. Illustrate your answer by discussing the function `foldr`:

```
fun foldr f ([], e) = e
  | foldr f (x::xs, e) = f(x, foldr f (xs,e));
```

[7 marks]

You have been asked to implement a data structure to family relationships. For each person, it should record his or her name, mother, their father, and children. As a first attempt, you have been given the following `datatype` declaration:

```
datatype person = Person of string * person * person * person list;
```

Identify two problems with this declaration that make it unusable. Modify the declaration to correct these problems [6 marks]

Consider the following, simpler data structure for associating a person with his or her children:

```
datatype famtree = B of string * famtree list;
```

Write an ML function that takes two arguments: a predicate P over family trees (a function of type `famtree->bool`) and a family tree t. The result should be the list of all subtrees of t (possibly including t itself) satisfying the predicate. For full credit, give due attention to efficiency. [7 marks]

Model Answer

'Functions as data' is extensively covered in the notes. The answer should mention passing functions as arguments to functions, which is possible in most languages, returning functions as results of other functions, and including functions as components of data structures. The function `foldr` takes a function as its argument (f) and, because it is curried, can be seen as delivering a function as its result. There are many applications of `foldr` in the notes; it can be used to define standard list functions such as append or to add up the elements of a list of integers. An answer should mention curried functions, which are an example of functions that deliver other functions as results, and that give the effect of 'partial evaluation.' Functions such as `foldr` allow programs to be expressed concisely, though sometimes the result is unreadable.

First, a conceptual problem: the data structure requires us to know a person's parents all the way back in time, which is clearly impossible. Second, a computational problem: if Elizabeth is the mother of Charles then there will be a cycle between the two nodes, but this data structure is immutable, so no cycles can be formed. A more minor corollary is that no changes can be made to the

data structures after they are built. But, as it stands now, no values of type `person` can be formed at all!

To correct the problems, we can add a `Nobody` constructor to handle unknown ancestors and also to provide a starting point for creating cycles. Also, we have to use type `ref` to allow cycles to be formed and to allow the data to be updated.

```
datatype person = Nobody
                | Person of string * person ref * person ref *
                  person list ref;
```

Here is a solution. It is not essential to use `foldr`, but it avoids the use of very expensive appending of the results for the various subtrees. Note that even if a node is accepted, its subtrees are still examined.

```
fun filtree P (t,ts) =
  let val B(_,chs) = t
      val ts2 = foldr (filtree P) (chs, ts)
  in
    if P t then t::ts2 else ts2
  end;
```