Solution notes

Introduction to Functional Programming 2005 – Paper 12 Question 10 (RGR)

(*a*) Definitions. The constructor names `Nil` and `Cons` can be varied as long as they are used consistently, and a name in place of the wildcard `_` is acceptable. Trailing semicolons are fine throughout.

(*i*) `datatype 'a seq = Nil | Cons of 'a * (unit -> 'a seq)` [1 mark]

(*ii*) `fun head (Cons (x, _)) = x` [1 mark]

(*iii*) `fun tail (Cons (_, xs)) = xs ()` [1 mark]

(*b*) `pick`. Two equivalent variations are given here, the second using curried functions in place of an explicit lambda function. 2 marks for a working sequence, and 2 marks for correctly building the list of all remaining elements at each step. −1 for a reversed list. [4 marks]

```
fun pick lst =
  let fun f _ [] = Nil
        | f prev (x::xs) =
            Cons ((x, prev @ xs),
                  fn () => f (prev @ [x]) xs)
  in
    f [] lst
  end

fun pick lst =
  let fun f _ [] () = Nil
        | f prev (x::xs) () =
            Cons ((x, prev @ xs),
                  f (prev @ [x]) xs)
  in
    f [] lst ()
  end
```

(*c*) `explodeseq`. Two equivalent variations are given here, the second using curried functions in place of an explicit lambda function. 2 marks for correctly decoding the input sequence, 2 for correctly generating an output sequence (even if the elements of the sequence are not correct), and 2 for a correct inner loop that explodes each list. [6 marks]

```
fun explodeseq Nil = Nil
  | explodeseq (Cons (lst, rest)) =
    let fun f [] = explodeseq (rest ())
          | f (x::xs) = Cons (x, fn () => f xs)
    in
      f lst
    end

fun explodeseq Nil = Nil
  | explodeseq (Cons (lst, rest)) =
    let fun f [] () = explodeseq (rest ())
          | f (x::xs) () = Cons (x, f xs)
    in
      f lst ()
    end
```

(*d*) `implodeseq`. 2 marks for correctly decoding the input sequence, 2 for correctly generating an output sequence (even if the elements of the sequence are not correct), 2 for correctly gathering lists of length `len` (−1 if the lists are in reverse order), and 1 for correctly handling the last (possibly short) list. [7 marks]

```
fun implodeseq len inseq =
  let fun f res _ Nil = Cons (res, fn () => Nil)
        | f res n (Cons (x, rest)) =
            if n = len
              then Cons (res, fn () => f [x] 1 (rest ()))
              else f (res @ [x]) (n + 1) (rest ())
  in
    f [] 0 inseq
  end
```