2 Foundations of Computer Science (LCP)

This question has been translated from Standard ML to OCaml

The function `perms` returns all $n!$ permutations of a given n-element list.

```
let rec perms = function
| [] -> [[]]
| xs ->
    let rec perms1 xs ys =
      match xs with
      | [] -> []
      | x::xs ->
          List.map (List.cons x) (perms (List.rev ys @ xs)) @
                   perms1 xs (x::ys)
    in
      perms1 xs []
```

(*a*) Explain the ideas behind this code, including the function `perms1` and the expression `List.map (List.cons x)`. What value is returned by `perms [1; 2; 3]`? [7 marks]

(*b*) A student modifies `perms` to use an OCaml type of lazy lists, where `appendq` and `mapq` are lazy list analogues of `@` and `List.map`.

```
let rec lperms = function
| [] -> Cons ([], fun () -> Nil)
| xs ->
    let rec fun perms1 xs ys = function
    | [] -> Nil
    | x::xs ->
        appendq (mapq (List.cons x) (lperms (List.rev ys @ xs)))
                (perms1 xs (x::ys))
    in
      perms1 xs []
```

Unfortunately, `lperms` computes all $n!$ permutations as soon as it is called. Describe how lazy lists are implemented in OCaml and explain why laziness is not achieved here. [5 marks]

(*c*) Modify the function `lperms`, without changing its type, so that it computes permutations upon demand rather than all at once. [8 marks]

All OCaml code must be explained clearly and should be free of needless complexity.