

Assignment 2

Due by 2:30 p.m. on Thursday, February 25

The context for this assignment is Chapter 2 of *DAWOC* and the core and standard OCaml libraries.

Minimal Edits

Consider the program `edits.ml`:

```
(* edits.ml *)

type 'a edit = Transpose of int
             | Insert   of int * 'a
             | Append   of 'a
             | Delete   of int

exception Error

let transpose n xs =
  let rec trans(n, ys, xs) =
    match xs with
    | []                -> raise Error
    | [_]              -> raise Error
    | x1 :: (x2 :: xs as x2_xs) ->
      if n = 0
      then List.rev_append ys (x2 :: x1 :: xs)
      else trans(n - 1, x1 :: ys, x2_xs)
  in if n < 0 then raise Error else trans(n, [], xs)

let insert (n, z) xs =
  let rec ins(n, ys, xs) =
    match xs with
    | []                -> raise Error
    | x :: xs as x_xs ->
      if n = 0
      then List.rev_append ys (z :: x_xs)
      else ins(n - 1, x :: ys, xs)
  in if n < 0 then raise Error else ins(n, [], xs)

let append y xs = xs @ [y]
```

```

let delete n xs =
  let rec del(n, ys, xs) =
    match xs with
    | []      -> raise Error
    | x :: xs ->
      if n = 0
      then List.rev_append ys xs
      else del(n - 1, x :: ys, xs)
  in if n < 0 then raise Error else del(n, [], xs)

let doEdit(edit, xs) =
  match edit with
  | Transpose n -> transpose n xs
  | Insert(n, z) -> insert (n, z) xs
  | Append n     -> append n xs
  | Delete n     -> delete n xs

let rec doEdits(edits, xs) =
  match edits with
  | []      -> xs
  | edit :: edits -> doEdits(edits, doEdit(edit, xs))

```

This program defines a datatype `'a edit` of list editing operations, an exception `Error`, functions

```

val doEdit  : 'a edit * 'a list -> 'a list
val doEdits : 'a edit list * 'a list -> 'a list

```

for carrying out a single edit or a series of edits on a list, and several auxiliary functions. The exception `Error` is raised when an edit isn't applicable to a list.

Given lists xs and ys of type `'a list`, a *minimal edit-list taking xs to ys* is a value $edits$ of type `'a edit list` such that:

- $\text{doEdits}(edits, xs) = ys$, and
- for all values $edits'$ of type `'a edit list` such that $\text{doEdits}(edits', xs) = ys$, the length of $edits'$ is no less than that of $edits$.

Write a program `minimal-edits.ml` that defines a function

```

val minimalEdits : 'a list * 'a list -> 'a edit list list

```

such that, for all lists xs and ys of the same type, $\text{minimalEdits}(xs, ys)$ returns the list of all minimal edit-lists taking xs to ys , listed in strictly ascending order according to `compare` (so that this list doesn't contain the same edit-list more than once).

Your program should assume that `edits.ml` has already been loaded into OCaml, and it must *not* redefine the datatype, exception or functions of `edits.ml`. Your program may make use of anything defined by `edits.ml`.

Assessment

Your program will be assessed according to three criteria:

- **Correctness.** Test your program carefully, but also try to prove to yourself that it is correct. (Don't submit evidence of testing, or a correctness proof.)
- **Style.** Program in a strictly functional style, without using the imperative features of Chapter 3 of *DAWOC*. Format your program in a way that makes its structure clear. Choose meaningful names for functions, and choose other identifiers with care. Document your functions by abstractly explaining their input/output behavior. (For a given function, say that if we know that its input has some property, that its output will have some other property).
- **Efficiency.** Try to make your program efficient, but don't sacrifice correctness or style in doing so.

Submission

Begin your program with a comment containing your name. Submit your program by emailing it to me. I will acknowledge receiving it. Make sure that you retain an electronic copy of your program.