Assignment 4

COMP 302 Programming Languages and Paradigms
Prakash Panangaden

Solutions to the written questions

Question 3[20 points] Informally derive the type for the apply-list function defined below. This function takes a list of functions and produces a single function which is the composite of all of them. If the list is empty it returns the identity function.

The code is shown below.

```ml
let rec apply_list l =
  match l with
  | [] -> (fun x -> x)
  | f::fs -> (fun x -> apply_list(fs)(f x))
```

Solution We see right away that apply_list is some kind of function so its type is assumed to be $\alpha \rightarrow \beta$, with $l$ being of type $\alpha$. We see from the first clause of the match that $l$ is some kind of list so we decide that $\alpha = \gamma - \text{list}$. We see that a function $\text{fun } x \rightarrow x$ is returned and this must be of type $\delta \rightarrow \delta$ so we have that $\beta = \delta \rightarrow \delta$. Now from the second clause we see that there is a function being returned here, so this function also has to be of type $\delta \rightarrow \delta$. Thus the $x$ occurring here must be of type $\delta$ and hence $f$ must be of type $\delta \rightarrow \theta$. However the result ($fx$) : $\theta$ is fed as an argument to the function being returned by apply_list which is of type $\delta \rightarrow \delta$. This means that $\theta = \delta$. So $f$ is of type $\delta \rightarrow \delta$. Hence $\gamma = \delta \rightarrow \delta$. Thus we deduce finally the type for apply_list to be

$$(\delta \rightarrow \delta) - \text{list} \rightarrow (\delta \rightarrow \delta).$$

Question 4[15 points] When the following expression is evaluated; the actual result is 7, as shown in the execution script below. Draw environment diagrams showing the bindings just as

1. the let u =2 block is opened,
2. the execution of f(4) is about to begin and
3. the execution of the body of the function is about to begin.
let x = 1 in
let f =
    let x = (let u = 2 in u + x) in
    fun z -> x + z
f(4);;

val x : int = 1
val f : (int -> int)
val it : int = 7

Solution:

Just after the \( u = 2 \) block is opened we have the evaluation of the body \( u + x \) about to begin. The names \( f \) and the second \( x \) have not been bound yet because the expressions which are going to produce the values have not been bound yet.

When the execution of \( f(4) \) is about to begin the \( u \) block is closed and it has vanished. The names \( f \) and \( x \) now have bindings. The binding \( (x, 3) \) is closed but it has become trapped by the closure pointer so, unlike \( u \), it is not removed.

Please turn over.
When the execution of the body of the function is about to begin we have the situation shown below. The parameter \( z \) has been bound to 4 and this binding points to the same place as the closure pointer points to.