Fernando Vallecillos Ruiz 1W18CG03

[1] Let curry be the transformation defined by curry(f) = $\lambda x_1, \lambda x_2.f\langle x_1, x_2\rangle$. Also, let uncurry be the transformation defined by uncurry(f) = $\lambda x.f(\text{fst } x)(\text{snd } x)$. Prove the following facts:
(a) If $\mathbf{\Gamma} \vdash f : \tau_1 \times \tau_2 \to \tau$ then $\mathbf{\Gamma} \vdash$ curry(f) : $\tau_1 \to \tau_2 \to \tau$.
(b) If $\mathbf{\Gamma} \vdash f : \tau_1 \to \tau_2 \to \tau$ then $\mathbf{\Gamma} \vdash$ uncurry(f) : $\tau_1 \times \tau_2 \to \tau$.

a)
We consider a function $f(x_1, x_2)$ taking two arguments, and having the type $(x_1, x_2) \to z$.The curried form of f is defined as:
$curry(f) = \lambda x_1.(\lambda x_2.(f(x_1, x_2)))$
Since curry takes, as input, functions with the type $(x_1 \times x_2) \to z$, one concludes that the type of curry itself is:
$curry : ((x_1 \times x_2) \to z) \to (x_1 \to (x_2 \to z))$

b)
We consider a function that takes functions with the type $(x_1 \times x_2)$, and having the type $(x_1 \times x_2) \to z$. The uncurried form of f is defined as:
$uncurry(f) = \lambda x.f(fst\ x)(snd\ x)$
Since curry takes, as input, two arguments, and having the type type $(x_1, x_2) \to z$, one concludes that the type of uncurried itself is:
$curry : (x_1 \to (x_2 \to z)) \to ((x_1 \times x_2) \to z)$

［2］ Let α list = $\mu\alpha'.unit+(\alpha\times\alpha')$. Write the recursive function map which, given a list $[v_1; v_2;...; v_n]$ and a function f as arguments, returns the list $[v'_1; v'_2; \ldots ; v'_n]$ where, for each $i \in 1, ... ; n$, v'_i is the evaluation result of applying f to v_i (i.e., the returned list is the given list but with f applied to each element). Write the function so that $\vdash$ map : τ_{map} where $\tau_{map} = \forall\alpha_1.\forall\alpha_2.\alpha_1 \text{ list} \rightarrow (\alpha_1 \rightarrow \alpha_2) \rightarrow \alpha_2 \text{ list}$. Also, show the derivation of $\vdash$ map : τ_{map}. For this question, use System F style explicit polymorphism. (Hint: map should be of the form fix $m.\Delta\alpha_1.\Delta\alpha_2.\lambda ls{:}\alpha_1 \text{ list}.\ \lambda f{:}\alpha_1\rightarrow\alpha_2....$).

fix $m.\Delta\alpha_1.\Delta\alpha_2.\lambda ls{:}\alpha_1 \text{ list}.\ \lambda f{:}\alpha_1\rightarrow\alpha_2.$if λEMPTY α_1 then () else (f α_1) (m α_2 f)

［３］ For this question, assume equi-recursive type equivalence (i.e., $\mu\alpha.\tau$ is implicitly equated with $[\mu\alpha.\tau/\alpha]$). The term $\omega = (\lambda x.x\ x)\ (\lambda x.x\ x)$ is not typable in the simple type system but is typable with recursive types. Indeed, below is the type derivation $\vdash \omega : \tau_\omega$ where $\tau_\omega = \mu\alpha.\alpha\to\alpha$:

Note that this is a correct derivation because, by equi-recursive type equality, $\tau_\omega = \mu\alpha.\alpha\to\alpha = (\mu\alpha.\alpha\to\alpha)\to(\mu\alpha.\alpha\to\alpha) = \tau_\omega \to \tau_\omega$.
Can every closed pure λ term be typed with recursive types? If yes, then write a proof of the claim. If no, then give an example of a closed pure λ term that is not typable with recursive types.

The term $\omega = \lambda x.xx$ is not typable with recursive types.