Exercise 60 (Type Inference)

Give a derivation tree for the following statement, and so determine the type \( \tau \):

\[
[] \vdash \lambda xyz. x y (y z) : \tau
\]

Exercise 61 (Recursive let)

Recursive let expressions are one way (besides \( Y \)-combinators) to add recursion to \( \lambda \rightarrow \).

\[
t ::= x \mid (t_1 t_2) \mid (\lambda x. t) \mid \text{letrec } x = t_1 \text{ in } t_2
\]

a) Modify the standard typing rule for let to create a suitable rule for letrec.

b) Give a derivation tree for the following statement, and so determine the type \( \tau \):

\[
[] \vdash \text{letrec } x = \lambda y. x (x y) \text{ in } x \in x : \tau
\]

Homework 62 (let-Polymorphism)

Give a derivation tree for the following statement, and so determine the type \( \tau \):

\[
[z : \tau_0] \vdash \text{let } x = \lambda yz. z y y \text{ in } x \in (x z) : \tau
\]

Homework 63 (Constructive logic)

a) Prove the following statement using the calculus for intuitionistic propositional logic:

\[
((c \rightarrow b) \rightarrow b) \rightarrow (c \rightarrow a) \rightarrow ((a \rightarrow b) \rightarrow b)
\]

Hint: To make your proof tree more compact, you may remove unneeded assumptions to the left of the \( \vdash \) during the proof as you see fit. For example, the following step is valid:

\[
\frac{p \vdash p}{p, q \vdash p}
\]

b) Give a well-typed expression in \( \lambda \rightarrow \) with the type

\[
((\gamma \rightarrow \beta) \rightarrow \beta) \rightarrow (\gamma \rightarrow \alpha) \rightarrow ((\alpha \rightarrow \beta) \rightarrow \beta)
\]

(You don’t need to give the derivation tree.)