Exercise 1 (Recursive let)

Recursive let expressions are one way (besides Y-combinators) to add recursion to λ→.

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

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

b) Considering type inference, what is the problematic property of this rule compared to the rule for let?

Exercise 2 (Type Inference in Haskell (2))

Extend the implementation of the type inference algorithm from the last exercise with let and letrec constructs.

Exercise 3 (Peirce’s Law in Intuitionistic Logic)

Prove the following variant of Peirce’s Law in intuitionistic logic:

\[ (((((P \rightarrow Q) \rightarrow P) \rightarrow P) \rightarrow Q) \rightarrow Q \]
**Homework 4 (Fixed-point combinator)**

Let

\[ S = \lambda abcdefghijklmnopqrstuvwxyz. \; r(thisisafixedpointcombinator) \]

and

\[ E = \ldots \]

Show that \( E \) is a fixed-point combinator.

**Homework 5 (let-Polymorphism)**

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

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

**Homework 6 (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:

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

b) Give a well-typed expression in \( \lambda^\tau \) 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.)