Exercise 1 (Example of Type Inference for let)

Consider the typing problem

\[ x : \alpha \vdash \text{let } y = \lambda z. z \ x \ \text{in } y \ (\lambda v. \ x) : ?\tau \]

where \( \alpha \) is a type variable.

a) Find the most general type schema \( \sigma \) with \( x : \alpha \vdash \lambda z. z \ x : \sigma \) and draw a type derivation tree.

b) Draw the type derivation tree for

\[ x : \alpha, y : \sigma \vdash y \ (\lambda v. \ x) : ?\tau \]

with the correct type for \( ?\tau \).

Exercise 2 (Recursive let)

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

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

a) Modify the standard typing rule for \( \text{let} \) to create a suitable rule for \( \text{letrec} \).

b) Considering type inference, what is the problematic property of this rule compared to the rule for \( \text{let} \)?

Exercise 3 (Type Inference in Haskell (2))

Extend the implementation of the type inference algorithm from the last exercise with \( \text{let} \) and \( \text{letrec} \) constructs.
Homework 4 (Fixed-point combinator)

Let
\[ S = \lambda abcdefghijklmnopqrstuvwxyzr. r(thisisafixedpointcombinator) \]
and
\[ \mathcal{E} = \ldots \]
Show that \( \mathcal{E} \) is a fixed-point combinator.

Homework 5 (let-Polymorphism)

Give a derivation tree for the following statement, and so determine the type \( \tau \):
\[ [z: \tau_0] \vdash \text{let } x = \lambda y z. z y y \text{ in } x (x z) : \tau \]

Homework 6 (Towards Syntax-Directed let-Polymorphism)

In the lecture, it was claimed that the systems \( DM \) and \( DM' \), which, in contrast to \( DM \), has explicit rules \( \forall \text{Intro} \) and \( \forall \text{Elim} \), are essentially equivalent. More specifically, it was claimed that
\[ \Gamma \vdash_{DM} t : \sigma \Longrightarrow \exists \tau. \quad \Gamma \vdash_{DM'} t : \tau \land \text{gen}(\Gamma, \tau) \preceq \sigma. \]

As a step towards proving this result, we want to rearrangement derivations in \( DM \) such that they resemble derivations in \( DM' \). In particular, prove that

a) Any derivation \( \Gamma \vdash_{DM} t : \sigma \) can be transformed such that \( \forall \text{Elim} \) only occur in a chain below the \( \text{Var} \) rule, i.e.

\[ \begin{array}{c}
\vdash_{Var} x : \forall \alpha_1, \ldots, \alpha_n. \tau \\
\vdash_{\forall \text{Elim}} x : \tau \\
\vdash_{\forall \text{Elim}} \vdash x : \tau
\end{array} \]

b) Any derivation \( \Gamma \vdash_{DM} t : \sigma \) can be transformed such that \( \forall \text{Intro} \) only occur in a chain that is terminated by an application of the \( \text{Let} \) rule or by the end of the proof, i.e.

\[ \begin{array}{c}
\vdash_{\forall \text{Intro}} \vdash l_1 : \tau \\
\vdash_{\forall \text{Intro}} \vdash l_1 : \forall \alpha_n. \tau \\
\vdash_{\forall \text{Intro}} \vdash l_1 : \forall \alpha_1, \ldots, \alpha_n. \tau \\
\vdash_{\forall \text{Intro}} \vdash l_1 : \forall \alpha_1, \ldots, \alpha_n. \tau \\
\vdash_{\text{Let}} \vdash \text{let } x = l_1 \text{ in } t_2 : \sigma
\end{array} \]
\[\vdash t_1 : \tau \quad \forall \text{Intro} \]
\[\vdash t_1 : \forall \alpha_n. \tau \quad \forall \text{Intro} \]
\[\vdash t_1 : \forall \alpha_1, \ldots, \alpha_n. \tau \quad \forall \text{Intro} \]