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 $\lambda^\rightarrow$.

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

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

b) Considering type inference, what is the problematic property of this rule compared to the rule for let?
Exercise 3 (Type Inference in Haskell (2))

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

Let

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

and

\[ \mathcal{E} = \text{thisisafixedpointcombinator} \].

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 \( \tau \):

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

Homework 6 (Towards Syntax-Directed let-Polymorphism)

In the lecture, it was claimed that the systems \( \text{DM} \) and \( \text{DM}' \), which, in contrast to \( \text{DM} \), has explicit rules \( \forall \text{Intro} \) and \( \forall \text{Elim} \), are essentially equivalent. More specifically, it was claimed that

\[
\Gamma \vdash_{\text{DM}} t : \sigma \Rightarrow \exists \tau. \quad \Gamma \vdash_{\text{DM}'} t : \tau \wedge \text{gen}(\Gamma, \tau) \preceq \sigma.
\]

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

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

\[
\frac{\frac{\frac{\vdots \quad \text{Var}}{\Gamma \vdash x : \forall \alpha_1, \ldots, \alpha_n. \tau} \quad \forall \text{Elim}}{\vdash x : \forall \alpha_1, \ldots, \alpha_n. \tau} \quad \forall \text{Elim}}{\vdash x : \tau} \quad \forall \text{Elim} \]

b) Any derivation \( \Gamma \vdash_{\text{DM}} t : \sigma \) 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.

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