**Technische Universität München**
**Institut für Informatik**
Prof. Tobias Nipkow, Ph.D.
Lukas Stevens

**Lambda Calculus**
**Winter Term 2022/23**
Solutions to Exercise Sheet 10

## Exercise 1 (Example of Type Inference for let)

Consider the typing problem

$$x\colon \alpha \vdash \mathsf{let}\ y = \lambda z.\ z\ x\ \mathsf{in}\ y\ (\lambda v.\ x)\ :\ ?\tau$$

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

$$y\colon \sigma, x\colon \alpha \vdash y\ (\lambda v.\ x)\ :\ ?\tau$$

with the correct type for $?\tau$.

## Solution

a) $\sigma = \forall\beta.\ (\alpha \to \beta) \to \beta$. Typing derivation:

$$\dfrac{\dfrac{\dfrac{\text{Var}\ \dfrac{}{z\colon \alpha \to \beta, x\colon \alpha \vdash z\ :\ \alpha \to \beta} \qquad \dfrac{}{z\colon \alpha \to \beta, x\colon \alpha \vdash x\ :\ \alpha}\ \text{Var}}{z\colon \alpha \to \beta, x\colon \alpha \vdash z\ x\ :\ \beta}\ \text{App}}{x\colon \alpha \vdash \lambda z.\ z\ x\ :\ (\alpha \to \beta) \to \beta}\ \text{Abs}}{x\colon \alpha \vdash \lambda z.\ z\ x\ :\ \forall\beta.\ (\alpha \to \beta) \to \beta}\ \forall\text{Intro}$$

b) Typing derivation:

$$\dfrac{\forall\text{Elim}\ \dfrac{\text{Var}\ \dfrac{}{y\colon \sigma, x\colon \alpha \vdash y\ :\ \forall\beta.\ (\alpha \to \beta) \to \beta}}{y\colon \sigma, x\colon \alpha \vdash y\ :\ (\alpha \to \alpha) \to \alpha} \qquad \dfrac{\dfrac{}{v\colon \gamma, y\colon \sigma, x\colon \alpha \vdash x\ :\ \alpha}\ \text{Var}}{y\colon \sigma, x\colon \alpha \vdash (\lambda v.\ x)\ :\ \alpha \to \alpha}\ \text{Abs}}{y\colon \sigma, x\colon \alpha \vdash y\ (\lambda v.\ x)\ :\ \alpha}\ \text{App}$$

## Exercise 2 (Recursive `let`)

Recursive `let` expressions are one way (besides $Y$-combinators) to add recursion to $\lambda^{\to}$.

$$t ::= x \mid (t_1\ t_2) \mid (\lambda x.\ \ t) \mid \texttt{letrec}\ x = t_1\ \texttt{in}\ t_2$$

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`?

## Solution

a) The rule for `letrec` is like the rule for `let`, but we also add $x$ to $\Gamma$ when checking $t_1$.

$$\frac{\Gamma[x\colon \sigma_1] \vdash t_1 \colon \sigma_1 \qquad \Gamma[x\colon \sigma_1] \vdash t_2 \colon \sigma_2}{\Gamma \vdash (\texttt{letrec}\ x = t_1\ \texttt{in}\ t_2) \colon \sigma_2}\ \textsc{LetRec}$$

Alternatively, we can combine this rule with the $\forall$-intro typing rule:

$$\frac{\{\alpha_1 \dots \alpha_n\} = FV(\tau) \setminus FV(\Gamma) \\ \Gamma[x\colon \forall\alpha_1 \dots \alpha_n.\ \tau] \vdash t_1 \colon \tau \qquad \Gamma[x\colon \forall\alpha_1 \dots \alpha_n.\ \tau] \vdash t_2 \colon \tau_2}{\Gamma \vdash \texttt{letrec}\ x = t_1\ \texttt{in}\ t_2 \colon \tau_2}\ \textsc{LetRec'}$$

b) The interesting property of this new typing rule is that we cannot know which $\alpha_1 \dots \alpha_n$ we need to generalize $\tau$ over before we have inferred $\tau$ (the type of $t_1$). Thus, typical compilers will only allow $x$ to be used monomorphically in $t_1$. Alternatively, the user can explicitly specify a type schema for $x$, so that it can be used polymorphically.

## Exercise 3 (Type Inference in Haskell (2))

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

## Solution

See *type_inference_let.hs*.

**Homework 4 (Fixed-point combinator)**

Let

$$\$ = \lambda abcdefghijklmnopqstuvwxyzr.\ r(thisisafixedpointcombinator)$$

and

$$€ = \$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\$\ .$$

Show that € is a fixed-point combinator.

**Homework 5 (`let`-Polymorphism)**

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

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

**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 ∀Intro and ∀Elim, are essentially equivalent. More specifically, it was claimed that

$$\Gamma \vdash_{DM} t\ :\ \sigma \Longrightarrow \exists\tau.\ \ \Gamma \vdash_{DM'} t\ :\ \tau \wedge \mathsf{gen}(\Gamma,\tau) \preceq \sigma.$$

As a step towards proving this result, we want to rearrange 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 ∀Elim only occur in a chain below the Var rule, i.e.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\Gamma \vdash x\colon \forall\alpha_1,\dots,\alpha_n.\ \tau}\ \text{Var}}{\vdots}\ \forall\text{Elim}}{\Gamma \vdash x\colon \forall\alpha_n.\ \tau}\ \forall\text{Elim}}{\Gamma \vdash x\colon \tau}\ \forall\text{Elim}$$
$$\vdots$$

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

$$\cfrac{\forall\text{Intro}\ \cfrac{\forall\text{Intro}\ \cfrac{\forall\text{Intro}\ \cfrac{\vdots}{\Gamma \vdash t_1\colon \tau}}{\Gamma \vdash t_1\colon \forall\alpha_n.\ \tau}}{\cfrac{\vdots}{\Gamma \vdash t_1\colon \forall\alpha_1,\dots,\alpha_n.\ \tau}} \qquad \cfrac{\vdots}{\Gamma[x\colon \forall\alpha_1,\dots,\alpha_n.\ \tau] \vdash t_2\colon \sigma}}{\Gamma \vdash \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2\colon \sigma}\ \text{Let}$$

or

$$
\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash t_1 : \tau}}{\Gamma \vdash t_1 : \forall \alpha_n .\ \tau}\ \forall\text{Intro}}{\dfrac{\vdots}{\Gamma \vdash t_1 : \forall \alpha_1, \ldots, \alpha_n .\ \tau}\ \forall\text{Intro}}\ \forall\text{Intro}
$$