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

Lambda Calculus
Winter Term 2021/22
Solutions to Exercise Sheet 09

Exercise 1 (Recursive `let`)

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

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

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 Γ when checking t_1.

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

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

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

b) The interesting property of this new typing rule is that we cannot know which $\alpha_1 \dots \alpha_n$ we need to generalize τ over before we have inferred τ (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 2 (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*.

Exercise 3 (Peirce's Law in Intuitionistic Logic)

Prove the following variant of Peirce's Law in inuitionistic logic:

$$((((P \to Q) \to P) \to P) \to Q) \to Q$$

Solution

Let $A_3 = (P \to Q) \to P$, $A_2 = A_3 \to P$, and $A_1 = A_2 \to Q$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A_1, A_3, P \vdash A_1 \qquad \dfrac{A_1, A_3, P \vdash P}{A_1, A_3, P \vdash A_2}\,{\to}\mathrm{I}}{A_1, A_3, P \vdash Q}\,{\to}\mathrm{E}}{A_1, A_3 \vdash P \to Q}\,{\to}\mathrm{I} \qquad A_1, A_3 \vdash A_3}{A_1, A_3 \vdash P}\,{\to}\mathrm{E}}{A_1 \vdash A_2}\,{\to}\mathrm{I} \qquad A_1 \vdash A_1}{A_1 \vdash Q}\,{\to}\mathrm{E}}{\vdash ((((P \to Q) \to P) \to P) \to Q) \to Q}\,{\to}\mathrm{I}$$

Homework 4 (Fixed-point combinator)

Let

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

and

$$\text{€} = \$\ .$$

Show that € is a fixed-point combinator.

Homework 5 (`let`-Polymorphism)

Give a derivation tree for the following statement, and so determine the type τ:

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

Homework 6 (Constructive Logic)

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

$$((c \to b) \to b) \to (c \to a) \to ((a \to b) \to b)$$

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 $\lambda^{\to}$ with the type

$$((\gamma \to \beta) \to \beta) \to (\gamma \to \alpha) \to ((\alpha \to \beta) \to \beta)$$

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