CST II 2001, paper 7, question ?13 AMP

Types

A type scheme $\forall\{\alpha_1,\dots,\alpha_n\}(\tau')$ generalises a type τ if τ can be obtained from τ' by substituting types for the type variables $\alpha_1,\dots,\alpha_n$: $\tau = \tau'[\tau_1/\alpha_1,\dots,\tau_n/\alpha_n]$.

1

For the given $\sigma_1, \sigma_2, \tau_1, \tau_2$ we have:

1 $\sigma_1 > \tau_1$, using $\alpha \mapsto (\alpha \to \beta)$, $\beta \mapsto \alpha$

1 $\sigma_1 > \tau_2$, using $\alpha \mapsto (\beta \to \alpha)$, $\beta \mapsto \beta$

1 $\sigma_2 \not> \tau_1$, because if $\sigma_2 > \tau$ then τ must be of the form $? \to \beta$ (and $\beta \neq \alpha$)

5 1 $\sigma_2 > \tau_2$, using $\alpha \mapsto (\beta \to \alpha)$.

Write "Γ ok" to mean the free type variables in Γ_{ta} are in Γ_{tv}.

(1) Axiom for variables:

1 $\Gamma \vdash x : \tau$ if Γ ok and $(x,\sigma) \in \Gamma_{ta}$ with $\sigma > \tau$ and type vars of τ in Γ_{tv}

(2) Axiom for boolean values $b ::= true \mid false$:

1 $\Gamma \vdash b : bool$ if Γ ok

(3) Rule for conditionals:

1
$$\frac{\Gamma \vdash M_1 : bool \quad \Gamma \vdash M_2 : \tau \quad \Gamma \vdash M_3 : \tau}{\Gamma \vdash \text{if } M_1 \text{ then } M_2 \text{ else } M_3 : \tau}$$

(4) Rule for function abstractions:

$$\frac{\Gamma, x : \tau_1 \vdash M : \tau_2}{\Gamma \vdash \lambda x(M) : \tau_1 \to \tau_2} \quad \text{if } x \notin dom(\Gamma_{ta})$$

where $\begin{cases} (\Gamma, x : \tau_1)_{tv} \triangleq \Gamma_{tv} \\ (\Gamma, x : \tau_1)_{ta} \triangleq \Gamma_{ta}[x \mapsto \forall \emptyset(\tau_1)] \end{cases}$

(5) Rule for function applications:

$$\frac{\Gamma \vdash M_1 : \tau_1 \to \tau_2 \quad \Gamma \vdash M_2 : \tau_1}{\Gamma \vdash M_1 M_2 : \tau_2}$$

(6) Rule for let-expressions:

$$\frac{A, \Gamma \vdash M_1 : \tau_1 \quad \Gamma, x : \forall A(\tau_1) \vdash M_2 : \tau_2}{\Gamma \vdash \text{let } x = M_1 \text{ in } M_2 : \tau}$$

if $A \cap \Gamma_{tv} = \emptyset$ and $x \notin dom(\Gamma_{ta})$

where $\begin{cases} (A, \Gamma)_{tv} \triangleq A \cup \Gamma_{tv} \\ (A, \Gamma)_{ta} \triangleq \Gamma_{ta} \end{cases}$

and $\Gamma, x : \forall A(\tau_1)$ is as above.

8

The ML type system allows let-bound variables to be used polymorphically in the body of the let-expression; the same is not true for λ-bound variables — all occurrences of a λ-bound variable in the body of a λ-abstraction must have the same implicit type. For example we have

$$\{\alpha\}, \emptyset \vdash \mathrm{let}\ x = \lambda y(y)\ \mathrm{in}\ \underbrace{x}_{\substack{\text{implicitly of}\\ \text{type } (\alpha\to\alpha)\to(\alpha\to\alpha)}}\ \underbrace{x}_{\substack{\text{implicitly of}\\ \text{type } \alpha\to\alpha}} : \alpha \to \alpha$$

as witnessed by proof:

$$\dfrac{\dfrac{\dfrac{}{\{\alpha,\beta\},\{y:\beta\}\vdash y:\beta}\,(1)}{\{\alpha,\beta\},\emptyset\vdash \lambda y(y):\beta\to\beta}\,(4) \qquad \dfrac{\dfrac{}{\Gamma\vdash x:(\alpha\to\alpha)\to(\alpha\to\alpha)}\,(1) \quad \dfrac{}{\Gamma\vdash x:\alpha\to\alpha}\,(1)}{\overbrace{\{\alpha\},\{x:\forall\beta(\beta\to\beta)\}}^{\Gamma}\vdash xx:\alpha\to\alpha}\,(5)}{\{\alpha\},\emptyset\vdash \mathrm{let}\ x=\lambda y(y)\ \mathrm{in}\ xx:\alpha\to\alpha}\,(6)$$

whereas $\Gamma \vdash (\lambda x(xx))(\lambda y(y)) : \tau$
holds for <u>no</u> τ. For if there were such a τ, the proof of typing would have to have the following structure

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x:\tau_3\vdash x:\tau_5}\,(1)\quad \dfrac{}{\Gamma, x:\tau_3\vdash x:\tau_6}\,(1)}{\Gamma, x:\tau_3\vdash xx:\tau_4}\,(5)}{\Gamma\vdash \lambda x(xx):\tau_1}\,(4) \qquad \dfrac{?}{\Gamma\vdash\lambda y(y):\tau_2}}{\Gamma\vdash(\lambda x(xx))(\lambda y(y)):\tau}\,(5)$$

$$\left.\begin{array}{ll} \text{with} & \forall\emptyset(\tau_3) \succ \tau_5 \ \text{— so}\ \tau_3 = \tau_5 \\ & \forall\emptyset(\tau_3) \succ \tau_6 \ \text{— so}\ \tau_3 = \tau_6 \\ \text{and} & \tau_5 = \tau_6 \to \tau_4 \end{array}\right\} \text{hence we}$$

would have $\tau_3 = \tau_3 \to \tau_4$ & no such τ_3 can exist (by counting # of $\to$ symbols on LHS & RHS).

(7)