Compiler Construction

Mini Project

D Type Reconstruction

The aim of this mini project is to reconstruct explicit type annotations from implicitly typed lambda-terms.

The project distribution includes abstract-syntax types for both an implicitly typed let-polymorphic lambda-calculus and System F. The former comes with a parser (`parse-hm`), the latter with a pretty printer (`pp-systemf`). Our objective is to implement a type inferencer for terms in the implicitly typed language that, as a side product, constructs corresponding System-F terms.

Implicitly Typed Language

Starting from a countable infinite set of variable symbols,

$$x \quad \in \quad \mathbf{Var} \qquad \text{variables,}$$

the terms of the implicitly typed language,

$$t \quad \in \quad \mathbf{Tm}_{\mathsf{HM}} \qquad \text{terms,}$$

are given by

$$t \quad ::= \quad x \mid \lambda x.\, t_1 \mid t_1\ t_2 \mid \mathbf{let}\ x = t_1\ \mathbf{in}\ t_2\ \mathbf{ni}.$$

Its type language is stratified into types and type schemes,

$$\begin{array}{lcll} \tau & \in & \mathbf{Ty}_{\mathsf{HM}} & \text{types} \\ \sigma & \in & \mathbf{TyScheme}_{\mathsf{HM}} & \text{type schemes,} \end{array}$$

and furthermore constructed from type variables,

$$\alpha \quad \in \quad \mathbf{TyVar} \qquad \text{type variables.}$$

Types and type schemes are then defined by

$$\begin{array}{lcl} \tau & ::= & \alpha \mid \tau_1 \to \tau_2 \\ \sigma & ::= & \tau \mid \forall \alpha.\, \sigma_1. \end{array}$$

Type environments,

$$\Gamma \quad \in \quad \mathbf{TyEnv}_{\mathsf{HM}} \qquad \text{type environments,}$$

map from variables to type schemes:

$$\Gamma \quad ::= \quad [\,] \mid \Gamma_1[x \mapsto \sigma].$$

We write $\Gamma(x) = \sigma$ to indicate that the rightmost binding for x in Γ maps x to σ. The set of free type variables of a type scheme or type environment are given by

$$\begin{array}{ll} \mathit{ftv} : (\mathbf{TyScheme}_{\mathsf{HM}} \cup \mathbf{TyEnv}_{\mathsf{HM}}) \to \mathcal{P}(\mathbf{TyVar}) & \\ \mathit{ftv}(\alpha) & = \{\alpha\} \end{array}$$

Typing $\boxed{\Gamma \vdash_{\mathsf{HM}} t : \sigma}$

$$\frac{\Gamma(x) = \sigma}{\Gamma \vdash_{\mathsf{HM}} x : \sigma}\ [\textit{hm-var}] \qquad \frac{\Gamma[x \mapsto \tau_1] \vdash_{\mathsf{HM}} t_1 : \tau_2}{\Gamma \vdash_{\mathsf{HM}} \lambda x.\, t_1 : \tau_1 \to \tau_2}\ [\textit{hm-lam}]$$

$$\frac{\Gamma \vdash_{\mathsf{HM}} t_1 : \tau_2 \to \tau \quad \Gamma \vdash_{\mathsf{HM}} t_2 : \tau_2}{\Gamma \vdash_{\mathsf{HM}} t_1\ t_2 : \tau}\ [\textit{hm-app}] \qquad \frac{\begin{array}{c}\Gamma \vdash_{\mathsf{HM}} t_1 : \sigma_1 \\ \Gamma[x \mapsto \sigma_1] \vdash_{\mathsf{HM}} t_2 : \tau\end{array}}{\Gamma \vdash_{\mathsf{HM}} \textbf{let}\ x = t_1\ \textbf{in}\ t_2\ \textbf{ni} : \tau}\ [\textit{hm-let}]$$

$$\frac{\Gamma \vdash_{\mathsf{HM}} t : \sigma_1 \quad \alpha \notin \mathit{ftv}(\Gamma)}{\Gamma \vdash_{\mathsf{HM}} t : \forall \alpha.\, \sigma_1}\ [\textit{hm-gen}] \qquad \frac{\Gamma \vdash_{\mathsf{HM}} t : \forall \alpha.\, \sigma_1}{\Gamma \vdash_{\mathsf{HM}} t : [\alpha \mapsto \tau_0]\sigma_1}\ [\textit{hm-inst}]$$

Figure 1: Typing rules for the implicitly typed language

$$\begin{array}{ll}
\mathit{ftv}(\tau_1 \to \tau_2) & = \mathit{ftv}(\tau_1) \cup \mathit{ftv}(\tau_2) \\
\mathit{ftv}(\forall \alpha.\, \sigma_1) & = \mathit{ftv}(\sigma_1) \backslash \{\alpha\} \\
\mathit{ftv}([\,]) & = \{\,\} \\
\mathit{ftv}(\Gamma_1[x \mapsto \sigma]) & = \mathit{ftv}(\Gamma_1) \cup \mathit{ftv}(\sigma).
\end{array}$$

The typing relation for the implicitly typed language, with judgements of the form

$$\Gamma \vdash_{\mathsf{HM}} t : \sigma,$$

is defined in Figure **??**.

System F

The terms of System F,

$$t \quad \in \quad \mathbf{Tm}_{\mathsf{F}} \qquad \text{terms},$$

containing explicit type terms,

$$\tau \quad \in \quad \mathbf{Ty}_{\mathsf{F}} \qquad \text{types},$$

are given by

$$\begin{array}{lll}
\tau & ::= & \alpha \mid \tau_1 \to \tau_2 \mid \forall \alpha.\, \tau_1 \\
t & ::= & x \mid \lambda x : \tau.\, t_1 \mid t_1\ t_2 \mid \Lambda \alpha.\, t_1 \mid t_1\ [\tau].
\end{array}$$

Here, type environments

$$\Gamma \quad \in \quad \mathbf{TyEnv}_{\mathsf{F}} \qquad \text{type environments},$$

map from variables to types:

$$\Gamma \quad ::= \quad [\,] \mid \Gamma_1[x \mapsto \tau].$$

The typing rules for System F, with judgements of the form

$$\Gamma \vdash_{\mathsf{F}} t : \tau,$$

are given in Figure **??**.

Typing $\boxed{\Gamma \vdash_{\mathsf{F}} t : \tau}$

$$\frac{\Gamma(x) = \tau}{\Gamma \vdash_{\mathsf{F}} x : \tau}\ [\textit{f-var}]$$

$$\frac{\Gamma[x \mapsto \tau_1] \vdash_{\mathsf{F}} t_1 : \tau_2}{\Gamma \vdash_{\mathsf{F}} \lambda x : \tau_1.\, t_1 : \tau_1 \to \tau_2}\ [\textit{f-lam}] \qquad \frac{\Gamma \vdash_{\mathsf{F}} t_1 : \tau_2 \to \tau \quad \Gamma \vdash_{\mathsf{F}} t_2 : \tau_2}{\Gamma \vdash_{\mathsf{F}} t_1\ t_2 : \tau}\ [\textit{f-app}]$$

$$\frac{\Gamma \vdash_{\mathsf{F}} t_1 : \tau_1}{\Gamma \vdash_{\mathsf{F}} \Lambda\alpha.\, t_1 : \forall\alpha.\, \tau_1}\ [\textit{f-tylam}] \qquad \frac{\Gamma \vdash_{\mathsf{F}} t_1 : \forall\alpha.\, \tau_1}{\Gamma \vdash_{\mathsf{F}} t_1\ [\tau_0] : [\alpha \mapsto \tau_0]\tau_1}\ [\textit{f-tyapp}]$$

Figure 2: Typing rules for System F

What to Do

Note that $\mathbf{TyScheme}_{\mathsf{HM}} \subseteq \mathbf{Ty}_{\mathsf{F}}$.

Assuming conventional dynamic semantics for the implicitly typed langauge and System F, implement a program `hm2systemf` that takes as input an ATerm representation (as produced by `parse-hm`) for a term $t \in \mathbf{Tm}_{\mathsf{HM}}$ and produces as output an ATerm representation (as consumed by `pp-systemf`) for an *operationally equivalent* term $t' \in \mathbf{Tm}_{\mathsf{F}}$, such that if

$$[\,] \vdash_{\mathsf{HM}} t : \sigma$$

with σ a *principal type* for t in $[\,]$, then

$$[\,] \vdash_{\mathsf{F}} t' : \sigma.$$

If t is ill-typed in [], then an appropriate type-error message should be issued.

As an example, consider the implicitly typed term

$$\lambda f.\, \lambda x.\, \mathbf{let}\ \mathit{id} = \lambda y.\, y\ \mathbf{in}\ (\mathit{id}\ f)\ (\mathit{id}\ x)\ \mathbf{ni}$$

and its possible System-F reconstruction

$$\begin{array}{l}\Lambda\alpha.\, \Lambda\beta.\, \lambda f : \alpha \to \beta.\, \lambda x : \alpha.\\ \quad (\lambda \mathit{id} : \forall\gamma.\, \gamma \to \gamma.\, (\mathit{id}\ [\alpha \to \beta]\ f)\ (\mathit{id}\ [\alpha]\ x))\ (\Lambda\gamma.\, \lambda y : \gamma.\, y).\end{array}$$

For the More Ambitious

A naïve implementation, based on, for example, Algorithm W, typically produces System-F terms that may contain "unnecessary" type abstractions and type applications. Consider, for instance, the term

$$\lambda f.\, \lambda x.\, \mathbf{let}\ \mathit{id} = \lambda y.\, y\ \mathbf{in}\ f\ (\mathit{id}\ x)\ (\mathit{id}\ x)\ \mathbf{ni}$$

and its naïve System-F reconstruction

$$\begin{array}{l}\Lambda\alpha.\, \Lambda\beta.\, \lambda f : \alpha \to \alpha \to \beta.\, \lambda x : \alpha.\\ \quad (\lambda \mathit{id} : \forall\gamma.\, \gamma \to \gamma.\, f\ (\mathit{id}\ [\alpha]\ x)\ (\mathit{id}\ [\alpha]\ x))\ (\Lambda\gamma.\, \lambda y : \gamma.\, y).\end{array}$$

Here, the lambda-bound identity function is applied to the type argument α exclusively. This suggests a more "direct" reconstruction:

$$\begin{aligned}&\Lambda\alpha.\,\Lambda\beta.\,\lambda f : \alpha \to \alpha \to \beta.\,\lambda x : \alpha.\\ &\quad(\lambda \mathit{id} : \alpha \to \alpha.\, f\ (\mathit{id}\ x)\ (\mathit{id}\ x))\ (\lambda y : \alpha.\, y).\end{aligned}$$

Can you adapt the implementation so that unnecessary type abstractions and type applications are avoided?

Submitting

The source code of your implementation should be handed in according to the submission instructions on the website of this course.

Submit the source code of your implementation (including both UUAG sources and Haskell sources *not* generated by the UUAG system).

Include in your submission a number of example programs that illustrate the capabilities of your implementation.