Higher-Ranked Exception Types

Ruud Koot

Utrecht University

May 13, 2014

Motivation

- Types should not lie; we would like to have *checked exceptions* in Haskell:

$$map :: (\alpha \to \beta) \to [\alpha] \to [\beta] \ \mathbf{throws}\ e$$

- What should be the correct value of e?

Motivation

Assigning accurate exception types is complicated by:

Higher-order functions Exceptions raised by higher-order functions depend on the exceptions raised by functional arguments.

$$map :: (\alpha \to \beta\ \mathbf{throws}\ e_1) \to [\alpha] \to [\beta]\ \mathbf{throws}\ (e_1\ \cup\ e_2)$$

Non-strict evaluation Exceptions are embedded inside values.

$$\begin{aligned} map :: {} & (\alpha\ \mathbf{throws}\ e_1 \to \beta)\ \mathbf{throws}\ e_2 \\ & \to [\alpha\ \mathbf{throws}\ e_3]\ \mathbf{throws}\ e_4 \to [\beta\ \mathbf{throws}\ e_5]\ \mathbf{throws}\ e_6 \end{aligned}$$

Motivation

- Instead of τ **throws** e, write τ^e for a type τ that can evaluate to $\bot_\chi$ for some $\chi \in e$.
- The fully annotated exception type for *map* would be:

$$
\begin{aligned}
&map :: (\alpha^{e_1} \to \beta^{(e_1 \,\cup\, e_2)})^{e_3} \to [\alpha^{e_1}]^{e_4} \to [\beta^{(e_1 \,\cup\, e_2 \,\cup\, e_3)}]^{e_4} \\
&map = \lambda f.\lambda xs.\ \mathbf{case}\ xs\ \mathbf{of} \\
&\qquad\qquad\quad \begin{array}{lcl} [\,] & \mapsto & [\,] \\ (y : ys) & \mapsto & f\ y : map\ f\ ys \end{array}
\end{aligned}
$$

Motivation

- Instead of τ **throws** e, write τ^e for a type τ that can evaluate to $\bot_\chi$ for some $\chi \in e$.
- The fully annotated exception type for *map* would be:

$$
\begin{aligned}
&\mathit{map} :: (\alpha^{e_1} \to \beta^{(e_1 \,\cup\, e_2)})^{e_3} \to [\alpha^{e_1}]^{e_4} \to [\beta^{(e_1 \,\cup\, e_2 \,\cup\, e_3)}]^{e_4} \\
&\mathit{map} = \lambda f.\lambda xs.\ \textbf{case}\ xs\ \textbf{of} \\
&\qquad\qquad [\,] \qquad\;\; \mapsto [\,] \\
&\qquad\qquad (y : ys) \mapsto f\ y : \mathit{map}\ f\ ys
\end{aligned}
$$

- If you want to be pedantic:

$$
\begin{aligned}
&\mathit{map} :: \forall \alpha\ \beta\ e_1\ e_2\ e_3\ e_4. \\
&\quad ((\alpha^{e_1} \to \beta^{(e_1 \,\cup\, e_2)})^{e_3} \to ([\alpha^{e_1}]^{e_4} \to [\beta^{(e_1 \,\cup\, e_2 \,\cup\, e_3)}]^{e_4})^{\emptyset})^{\emptyset}
\end{aligned}
$$

Motivation

- Instead of τ **throws** e, write τ^e for a type τ that can evaluate to $\bot_\chi$ for some $\chi \in e$.
- The fully annotated exception type for *map* would be:

$$
\begin{aligned}
&map :: (\alpha^{e_1} \xrightarrow{e_3} \beta^{(e_1 \cup e_2)}) \to [\alpha^{e_1}]^{e_4} \to [\beta^{(e_1 \cup e_2 \cup e_3)}]^{e_4} \\
&map = \lambda f.\lambda xs.\ \textbf{case}\ xs\ \textbf{of} \\
&\qquad\qquad [\,] \qquad\ \mapsto [\,] \\
&\qquad\qquad (y : ys) \mapsto f\ y : map\ f\ ys
\end{aligned}
$$

- If you want to be pedantic:

$$
\begin{aligned}
&map :: \forall \alpha\ \beta\ e_1\ e_2\ e_3\ e_4. \\
&\quad (\alpha^{e_1} \xrightarrow{e_3} \beta^{e_1 \cup e_2}) \xrightarrow{\cdot} \cdot)\ ([\alpha^{e_1}]^{e_4} \xrightarrow{\emptyset} [\beta^{(e_1 \cup e_2 \cup e_3)}]^{e_4})\ \emptyset
\end{aligned}
$$

Motivation

- The exception type

$$map :: (\alpha^{e_1} \xrightarrow{e_3} \beta^{(e_1 \cup e_2)}) \to [\alpha^{e_1}]^{e_4} \to [\beta^{(e_1 \cup e_2 \cup e_3)}]^{e_4}$$

  is not as accurate as we would like.
- Consider the instantiations:

$$\begin{aligned} &map\ id && :: [\alpha^{e_1}]^{e_4} \to [\alpha^{e_1}]^{e_4} \\ &map\ (const\ \bot_{\mathbf{E}}) && :: [\alpha^{e_1}]^{e_4} \to [\beta^{(e_1 \cup \{\mathbf{E}\})}]^{e_4} \end{aligned}$$

- A more appropriate type for *map* $(const\ \bot_{\mathbf{E}})$ would be

$$map\ (const\ \bot_{\mathbf{E}}) :: [\alpha^{e_1}]^{e_4} \to [\beta^{\{\mathbf{E}\}}]^{e_4}$$

  as it cannot propagate exceptional elements inside the input list to the output list.

Motivation

- The problem is that we have already committed the first argument of *map* to be of type

  $$\alpha^{e_1} \xrightarrow{e_3} \beta^{(e_1 \cup e_2)},$$

  i.e. it propagates exceptional values from the its input to the output while possibly adding additional exceptional values.
- This is a worst-case scenario: it is sound but inaccurate.

Motivation

- The solution is to move from Hindley–Milner to F_ω, introducing *higher-ranked types* and *type operators*.
  - Recall that System F_ω replicates the *simply typed λ-calculus* on the type level.
- This gives us the expressiveness to state the exception type of *map* as:

$$\forall e_2\ e_3.(\forall e_1.\alpha^{e_1} \xrightarrow{e_3} \beta^{(e_2\ e_1)}) \\ \rightarrow (\forall e_4\ e_5.[\alpha^{e_4}]^{e_5} \rightarrow [\beta^{(e_2\ e_4\ \cup\ e_3)}]^{e_5})$$

- Note that e_2 is an *exception operator* of kind $\textsc{exn} \rightarrow \textsc{exn}$.

Motivation

- Given the following functions:

$$
\begin{aligned}
\mathit{map} \quad &:: \forall e_2\ e_3.(\forall e_1.\alpha^{e_1} \xrightarrow{e_3} \beta^{(e_2\ e_1)}) \\
&\quad \to (\forall e_4\ e_5.[\alpha^{e_4}]^{e_5} \to [\beta^{(e_2\ e_4\ \cup\ e_3)}]^{e_5}) \\
\mathit{id} \quad &:: \forall e.\alpha^{e} \xrightarrow{\emptyset} \alpha^{e} \\
\mathit{const}\ \bot_{\mathbf{E}} &:: \forall e.\alpha^{e} \xrightarrow{\emptyset} \beta^{\{\mathbf{E}\}}
\end{aligned}
$$

- Applying *id* or *const* $\bot_{\mathbf{E}}$ to *map* will give rise the the instantiations $e_2 \mapsto \lambda e.e$, respectively $e_2 \mapsto \lambda e.\{\mathbf{E}\}$.
- This gives us the exception types:

$$
\begin{aligned}
\mathit{map}\ \mathit{id} \quad &:: \forall e_4\ e_5.[\alpha^{e_4}]^{e_5} \to [\alpha^{e_4}]^{e_5} \\
\mathit{map}\ (\mathit{const}\ \bot_{\mathbf{E}}) &:: \forall e_4\ e_5.[\alpha^{e_4}]^{e_5} \to [\beta^{\{\mathbf{E}\}}]^{e_5}
\end{aligned}
$$

as desired.

Technicalities

- Due to their syntactic weight, higher-ranked exception type only seem useful if they can be infered automatically.
- Unlike for HM type inference is undecidable in F_ω.
- However, the exception types are annotations piggybacking on top of an underlying type system.
- Holdermans and Hage [HH10] showed type inference is decidable for a higher-ranked annotated type system with type operators performing control-flow analysis.

Technicalities

Stefan Holdermans and Jurriaan Hage, *Polyvariant flow analysis with higher-ranked polymorphic types and higher-order effect operators*, Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming (New York, NY, USA), ICFP '10, ACM, 2010, pp. 63–74.