Higher-Ranked Exception Types

Ruud Koot
Department of Computing and Information Sciences, Utrecht University
September 2014

Exception Types

- In Haskell "types do not lie":
  - Functions behave as mathematical function on the domain and range given by their type
  - Side-effects are made explicit by monadic types
- Exceptions that may be raised are *not* captured in the type
  - We would like them to be during program verification
  - Adding exception types to Haskell is more complicated than in a strict first-order language

Exception Types in Haskell

- Exception types in Haskell can get complicated because of:

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

Non-strict evaluation Exceptions are not a form of control flow, but are values that can be embedded inside other values.

- An exception-annotated type for *map* would be:

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

$$
\begin{aligned}
map = \lambda f.\lambda xs.\ &\mathbf{case}\ xs\ \mathbf{of} \\
&[\,] \mapsto [\,] \\
&(y : ys) \mapsto f\ y : map\ f\ ys
\end{aligned}
$$

Precise Exception Types

- The exception type above is not a precise as we would like

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

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

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

- Exceptional elements in the input list cannot be propagated to the output.

Higher-Ranked Exception Types

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

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

- It always propagates exceptional values from the input to the output
- The solution is to move from Hindley–Milner to System F_ω, introducing *higher-ranked exception types* and *exception operators*

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

- This gives us the desired exception types:

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

Exception Type Inference

- Higher-ranked exception types are syntactically heavy; we need type inference
- Type inference is undecidable in System F_ω, but exception types piggyback on an underlying type
- Holdermans and Hage (2010) show that type inference is decidable for a similar higher-ranked annotated type system with type operators

Work-in-Progress

Imprecise exception semantics Haskell has an *imprecise exception semantics*
  - Needed for soundness of various program transformations in an optimizing compiler
  - Not adequately captured by ACI1 constraints; attempt to use equational unification in Boolean rings instead

Metatheory Is the combined rewrite system of STLC and BR still confluent and normalizing?
  - Needed for decidable exception type equality
  - Hope to use a general result by Breazu-Tannen