Higher-Ranked Exception Types

(Extended abstract of work-in-progress)

Ruud Koot *

Department of Computing and Information Sciences
Princetonplein 5, 3584 CC, Utrecht
Utrecht University
r.koot@uu.nl

1. Motivation

An important design decision underlying the Haskell programming languages is that "types should not lie": functions should behave as if they where mathematical functions on the domain and range stated by their type; any side-effects the function wishes to perform must be made explicit by giving it an appropriate monadic type. Surprisingly to some, then, Haskell does not feature *checked exceptions*. Any exception that may be raised—whether explicitly by a call to the the *error* function, or implicitly by a pattern match failure—is completely invisible in the type. This is unfortunate as during program verification such a type would give the programmer an explicit list of proof obligations he must fulfill: for each exception it must be shown that either the exception can never occur or only occur in truly exceptional circumstances.

While some programming languages with more mundane type systems do state the exceptions a function may raise in its type signature, assigning accurate exception types to functions in a Haskell program is complicated by:

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.

Writing the set of exceptions that may be raised when evaluating an expression to weak head normal form as a superscript on it type, a fully annotated exception type for *map* would be:

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

However, this exception type is not as accurate as we would like it to be. Consider the applications:

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

While the exception type for $\mathit{map}\ \mathit{id}$ is as expected, a more appropriate type for $\mathit{map}\ (\mathit{const}\ \bot_{\mathbf{E}})$ would be

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

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

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 some additional exceptional values. This is a worst-case scenario: it is sound but inaccurate.

The solution is to move from Hindley–Milner to System F_ω, introducing *higher-ranked exception types* and *exception operators*. This gives us the expressiveness to state the exception type of *map* as:

$$\begin{aligned}&\forall e_2\ e_3.(\forall e_1.\alpha^{e_1} \xrightarrow{e_3} \beta^{(e_2\ e_1)})\\ &\quad \to (\forall e_4\ e_5.[\,\alpha^{e_4}\,]^{e_5} \to [\,\beta^{(e_2\ e_4\ \cup\ e_3)}\,]^{e_5})\end{aligned}$$

Note that e_2 has kind $\textsc{exn} \to \textsc{exn}$. Given the following functions:

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

Applying *map* to *id* or $\mathit{const}\ \bot_{\mathbf{E}}$ 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:

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

as desired.

2. Related Work

3. Approach

4. Results

References

V. Breazu-Tannen. Combining algebra and higher-order types. In *Logic in Computer Science, 1988. LICS '88., Proceedings of the Third Annual Symposium on*, pages 82–90, 1988. doi: 10.1109/LICS.1988.5103.

S. Holdermans and J. Hage. Polyvariant flow analysis with higher-ranked polymorphic types and higher-order effect operators. In *Proceedings of the 15th ACM SIGPLAN International Conference on Functional Programming*, ICFP '10, pages 63–74, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-794-3. doi: 10.1145/1863543.1863554. URL `http://doi.acm.org/10.1145/1863543.1863554`.

A. J. Kennedy. Type inference and equational theories. Technical Report LIX-RR-96-09, Laboratoire D'Informatique, École Polytechnique, 1996.

* This material is based upon work supported by the Netherlands Organisation for Scientific Research (NWO).

D. A. Wright. A new technique for strictness analysis. In S. Abramsky and T. Maibaum, editors, *TAPSOFT '91*, volume 494 of *Lecture Notes in Computer Science*, pages 235–258. Springer Berlin Heidelberg, 1991. ISBN 978-3-540-53981-0.