Type-based Exception Analysis

for Non-strict Higher-order Functional Languages with Imprecise Exception Semantics

Ruud Koot   Jurriaan Hage

Department of Information and Computing Sciences
Utrecht University

January 14, 2015

Motivation

- "Well-typed programs do not go wrong."

Motivation

- "Well-typed programs do not go wrong."
- Except:
  - $reciprocal\ x = 1 / x$
  - $head\ (x :: xs) = x$
  - ...
- Practical programming languages allow functions to be *partial*.

Motivation

- Requiring all functions to be total may be undesirable.
  - Dependent types are heavy-weight.
  - Running everything in the *Maybe* monad does not solve the problem, only moves it.
  - Some partial functions are *benign*.
- We do want to warn the programmer something may go wrong at run-time.

Motivation

- Currently compilers do a local and syntactic analysis.

$$\begin{aligned}&\mathit{head} :: [\,\alpha\,] \to \alpha \\ &\mathit{head}\ \mathit{xs} = \textbf{case}\ \mathit{xs}\ \textbf{of}\ \{\,(y : \mathit{ys}) \to y\,\}\end{aligned}$$

Motivation

- Currently compilers do a local and syntactic analysis.

  $head :: [\alpha] \to \alpha$
  $head\ xs = \mathbf{case}\ xs\ \mathbf{of}\ \{\,(y : ys) \to y\,\}$

- "The problem is in *head* and *every* place you call it!"

  $main = head\ [1, 2, 3]$

- Worse are non-escaping local definitions.

Motivation

- The canonical example by Mitchell & Runciman (2008):

$$
\begin{array}{l}
\mathit{risers} \;::\; \mathit{Ord}\ \alpha \Rightarrow [\alpha] \rightarrow [[\alpha]] \\
\mathit{risers}\ [\,] \qquad\qquad\quad = [\,] \\
\mathit{risers}\ [x] \qquad\qquad\quad = [[x]] \\
\mathit{risers}\ (x_1 : x_2 : xs) = \\
\quad \textbf{if}\ x_1 \leqslant x_2\ \textbf{then}\ (x_1 : y) : ys\ \textbf{else}\ [x_1] : (y : ys) \\
\qquad \textbf{where}\ (y : ys) = \mathit{risers}\ (x_2 : xs)
\end{array}
$$

- Program invariants can ensure incomplete pattern matches never fail.

Motivation

- Instead use a semantic approach: "where can exceptions flow to?"

$$\mathit{head} :: [\,\alpha\,] \to \alpha$$
$$\mathit{head}\ \mathit{xs} = \mathbf{case}\ \mathit{xs}\ \mathbf{of}\ \{\,[\,] \to \lightning\,;(y : ys) \to y\,\}$$

- Simultaneously need to track data flow to determine which branches are not taken.
- Using a type-and-effect system, the analysis is still modular.

Basic idea: data flow

- We can then assign to each of the three individual branches of *risers* the following types:

$$\begin{array}{ll} \mathit{risers}_1 & :: \forall\alpha.\mathit{Ord}\ \alpha \Rightarrow [\alpha]^{\mathbf{N}} \to [[\alpha]^{\mathbf{N}\sqcup\mathbf{C}}]^{\mathbf{N}} \\ \mathit{risers}_2, \mathit{risers}_3 & :: \forall\alpha.\mathit{Ord}\ \alpha \Rightarrow [\alpha]^{\mathbf{C}} \to [[\alpha]^{\mathbf{N}\sqcup\mathbf{C}}]^{\mathbf{C}} \end{array}$$

- From the three individual branches we may infer:

$$\mathit{risers} :: \forall\alpha.\mathit{Ord}\ \alpha \Rightarrow [\alpha]^{\mathbf{N}\sqcup\mathbf{C}} \to [[\alpha]^{\mathbf{N}\sqcup\mathbf{C}}]^{\mathbf{N}\sqcup\mathbf{C}}$$

- Adding *polyvariance* gives us a more precise result:

$$\mathit{risers} :: \forall\alpha\beta.\mathit{Ord}\ \alpha \Rightarrow [\alpha]^{\beta} \to [[\alpha]^{\mathbf{N}\sqcup\mathbf{C}}]^{\beta}$$

Basic idea: exception flow

- A compiler will translate the partial function *head* into:

$$\begin{aligned} \mathit{head}\ \mathit{xs} = \mathbf{case}\ \mathit{xs}\ \mathbf{of}\ & \\ & [\,] \mapsto \mathit{crash}\ \textbf{pattern-match-failure} \\ & y : \mathit{ys} \mapsto y \end{aligned}$$

- which can be assigned the exception type:

$$\mathit{head} :: \forall \tau \alpha \beta . [\tau^{\alpha}]^{\beta} \xrightarrow{\emptyset} \tau^{\alpha \sqcup \beta \sqcup \textbf{pattern-match-failure}}$$

- This type tells us that *head* might always raise a **pattern-match-failure** exception!
- Introduce a dependency of the exception flow on the data flow of the program:

$$\begin{aligned} \mathit{head} :: \forall \tau \alpha \beta \gamma . [\tau^{\alpha}]^{\beta} \xrightarrow{\emptyset} \tau^{\alpha \sqcup \beta \sqcup \gamma} & \\ \mathbf{with}\ \{\mathbf{N} \sqsubseteq \beta \Rightarrow \textbf{pattern-match-failure} \sqsubseteq \gamma\} & \end{aligned}$$

Imprecise exception semantics

- Non-strict languages can have an *imprecise exception semantics* (Peyton Jones *et al*, 1999).
  - Can non-deterministically raise one from a set of exceptions.
  - Necessary for the soundness of certain program transformations, e.g. the case-switching transformation:

$$
\begin{array}{l}
\forall e_i.\mathbf{if}\ e_1\ \mathbf{then} \\
\quad\quad \mathbf{if}\ e_2\ \mathbf{then}\ e_3\ \mathbf{else}\ e_4 \\
\quad \mathbf{else} \\
\quad\quad \mathbf{if}\ e_2\ \mathbf{then}\ e_5\ \mathbf{else}\ e_6
\end{array}
=
\begin{array}{l}
\mathbf{if}\ e_2\ \mathbf{then} \\
\quad \mathbf{if}\ e_1\ \mathbf{then}\ e_3\ \mathbf{else}\ e_5 \\
\mathbf{else} \\
\quad \mathbf{if}\ e_1\ \mathbf{then}\ e_4\ \mathbf{else}\ e_6
\end{array}
$$

Imprecise exception semantics

- If the scrutinee of a ... exception-finding mode
- implication for the analysis: cannot separate data and exception flow phases

Algorithm

- Assumes program is well-typed in underlying type system.
- Algorithm $\mathcal{W}$-like constraint generation phase.
- Worklist/fixpoint-based constraint solver.

Constraint language

- *Conditional constraints* and *indices* model dependence between data flow and exception flow
- Asymmetric to keep solving tractable

$$
\begin{array}{rcl}
c & ::= & g \Rightarrow r \\
g & ::= & \Lambda_\iota \sqsubseteq_\iota \alpha \quad | \quad \exists_\iota \alpha \quad | \quad g_1 \vee g_2 \quad | \quad \mathbf{true} \\
r & ::= & \Lambda_\iota \sqsubseteq_\iota \alpha \quad | \quad \alpha_1 \sqsubseteq_\iota \alpha_2 \quad | \quad \tau_1 \leqslant_\iota \tau_2
\end{array}
$$

A type rule (case-expressions)

$$\frac{\begin{array}{c} C;\Gamma \vdash e_1 : [\tau_1]^{\alpha_1} \quad C;\Gamma \vdash e_2 : \tau_2 \\ C;\Gamma, x_1 : \tau_1, x_2 : [\tau_1]^{\beta} \vdash e_3 : \tau_3 \\ C \Vdash \mathbf{N} \sqsubseteq_{\delta} \alpha_1 \vee \exists_{\chi}\alpha_1 \Rightarrow \tau_2 \leqslant_{\delta\chi} \tau \\ C \Vdash \mathbf{C} \sqsubseteq_{\delta} \alpha_1 \vee \exists_{\chi}\alpha_1 \Rightarrow \tau_3 \leqslant_{\delta\chi} \tau \\ C \Vdash \alpha_1 \sqsubseteq_{\chi} \lceil\tau\rceil \quad C \Vdash \mathbf{N} \sqcup \mathbf{C} \sqsubseteq_{\delta} \beta \quad C \Vdash \alpha_1 \sqsubseteq_{\chi} \beta \end{array}}{C;\Gamma \vdash \mathbf{case}\ e_1\ \mathbf{of}\ \{[] \mapsto e_2; x_1 :: x_2 \mapsto e_3\} : \tau} \text{[T-C\textsc{ase}]}$$

Questions?