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:
  - $divideByZero\ x = x\ /\ 0$
  - $head\ (x :: xs) = x$
  - ...
- 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.

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

Motivation

- Currently compilers do a local and syntactic analysis.

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

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

  $$\mathit{main} = \mathit{head}\ [1, 2, 3]$$

- Worse are non-escaping local definitions.

Motivation

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

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