# 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.

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

# 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!"

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

# Motivation

- Worse are non-escaping local definitions. The canonical example from Mitchell & Runciman: