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
  - Exceptions 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:

```
map : ∀α β e1 e2 e3 e4. 
(αe1 ⊅ β(e1 ∪ e2)) ⊇ [αe1]e4 ⊇ [β(e1 ∪ e2 ∪ e3)]e4
map = λf.λxs. case xs of 
  [] → [] 
  (y : ys) → f y : map f ys
```

Precise Exception Types

- The exception type above is not a precise as we would like
  - map id : [αe1]e4 → [αe1]e4
  - map (const ⊥E) : [αe1]e4 → [β(e1 ∪ {E})]e4

  - A more appropriate type for map (const ⊥E) would be:

```
map (const ⊥E) : [αe1]e4 → [β{E}]e4
```

- 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 αe1 ⊅ β(e1 ∪ e2)
- It always propagates exceptional values from the input to the output
- The solution is to move from Hindley–Milner to System Fω, introducing higher-ranked exception types and exception operators

```
map : ∀e2 e3. (∀e1. αe1 ⊅ β(e1 ∪ e2)) → (∀e4 e5. [αe4]e5 → [β(e1 ∪ e2 ∪ e3)]e5)

id : ∀e.e ⊇ αe

case xs of 
  [] → [] 
  (y : ys) → f y : map f ys
```

Exception Type Inference

- Higher-ranked exception types are syntactically heavy; we need type inference
- Type inference is undecidable in System Fω, 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 AC1 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