QuB

A Resource Aware Functional Programming Language

Apoorv Ingle

The University of Kansas

Table of Contents

Introduction and Motivation

- Hard problem: Resource Management in evolving production code
- Resources: Files, database connections, anything that represents a shared state in the program

Resource Management: File Handling in Haskell

- Modified File Handling API

```
openFile :: FilePath  → IO FileHandle

closeFile :: FileHandle → IO ()

readLine :: FileHandle → IO (String, FileHandle)

writeFile :: String    → FileHandle
                       → IO ((), FileHandle)


upper     :: String    → String
```

Resource Management: File Handling in Haskell

- File Handling

```
do f  ← openFile "sample.txt"
   (s, f)  ← readLine f
   let c = upper s
   ((), f) ← writeLine f c
           .
           .
           .
   ()  ← closeFile f
```

Resource Management: File Handling in Haskell

- File Handling Gone Wrong (Part I)

```
do f  ← openFile "sample.txt"
   (s, f)  ← readLine f
   let c = upper s
   ((), f) ← writeLine f c
        .
        .
        .
   () ← closeFile f
        .
        .
        .
   () ← closeFile f
   return c
```

Resource Management: File Handling in Haskell

- File Handling Gone Wrong (Part I)

```
do f  ← openFile "sample.txt"
   (s, f)  ← readLine f
   let c = upper s
   ((), f) ← writeLine f c
        .
        .
        .
  () ← closeFile f
        .
        .
        .
  () ← closeFile f
   return c
```

- File is closed twice: Run time crash

Resource Management: File Handling in Haskell

- File Handling Gone Wrong (Part II)

```
do f  ← openFile "sample.txt"
   (s, f)  ← readLine f
   let c = upper s
   ((), f) ← writeLine f c
       .
       .
       .
   return c
```

Resource Management: File Handling in Haskell

- File Handling Gone Wrong (Part II)

```
do f  ← openFile "sample.txt"
   (s, f)  ← readLine f
   let c = upper s
   ((), f) ← writeLine f c
       .
       .
       .
   return c /*File not closed!!*/
```

- File not closed: Memory leak

Resource Management: Exception Handling

- `MonadError`[5] in Haskell

```
class Monad m ⇒ MonadError e m | m → e where
    throwError :: e → m a
    catchError :: m a → (e → m a) → m a
```

- `throwError` starts exception processing
- `catchError` exception handler

Resource Management: Exception Handling

- `MonadError` in Haskell

```
do f ← openFile "sample.txt"
   ((s, f)  ← readLine f
   let c = upper s
   ()  ← closeFile f
   return $ Right c) `catchError` (\_ →
                        return $ Left "Error in reading file")
```

- Execution path for exception 👻 file not closed 👻 Memory leak

Introduction and Motivation

'Well typed programs do not go wrong.' Robin Milner

Introduction and Motivation

'Well typed programs do not go wrong.' Robin Milner

- Can we do better? Can types guide us?

Background Work

Hinley-Milner (**HM**) type system Algorithm $\mathcal{M}$[1] Algorithm $\mathcal{W}$[4]

Background Work

Curry Howard Correspondence:
HM type system is equivalent to second order propositional logic.
Propostions are not Resources.

Background Work

Substructural Logics: Make structural rules explicit

Background Work

Linear Logic[2, 8]
restrict weakening and contraction
Propostions act like resources

- Modality !
- Additive pair: &
- Multiplicative pair: $\otimes$ $\multimap$
- $\oplus$

Background Work

Theory of qualified types: Polymorphism using Predicates **HM**[3]

Background Work

Quill: Linear Logic using with Qualified Types[6]

Background Work

Logic of Bunched Implications (**BI**)[7]

- Sharing implication $\twoheadrightarrow$
- Separating implication $-\!\!*$

QuB Core Language: Syntax and Types

Multiplicative Products

Additive Products

Sums

Multilicative Implication

Additive Implication

Conclusion and Future Work

Thank You!

Q & A

References I

[1] Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '82, pages 207–212. ACM, 1982.

[2] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[3] Mark P. Jones. A theory of qualified types. *Science of Computer Programming*, 22(3):231 – 256, 1994.

[4] Oukseh Lee and Kwangkeun Yi. Proofs about a folklore let-polymorphic type inference algorithm. *ACM Trans. Program. Lang. Syst.*, 20(4):707–723, 1998.

[5] Sheng Liang, Paul Hudak, and Mark Jones. Monad transformers and modular interpreters. In *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '95, pages 333–343. ACM, 1995.

References II

[6] J. Garrett Morris. The best of both worlds: Linear functional programming without compromise. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP 2016, pages 448–461. ACM, 2016.

[7] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *The Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[8] Philip Wadler. A taste of linear logic. In Andrzej M. Borzyszkowski and Stefan Sokolowski, editors, *Mathematical Foundations of Computer Science 1993: 18th International Symposium, MFCS'93 Gdańsk, Poland, August 30–September 3, 1993 Proceedings*, pages 185–210. Springer Berlin Heidelberg, 1993.