QuB

A Resource Aware Functional Programming Language

Apoorv Ingle

The University of Kansas

Hard problems in programming

Naming variables

Hard problems in programming

Resource management

in evolving production code

Resources: Files, database connections, shared mutable state

Resource Management: File Handling

- Modified File Handling API in Haskell

```
openFile  :: FilePath   → IO FileHandle
closeFile :: FileHandle → IO ()
readLine  :: FileHandle → IO (String, FileHandle)
writeFile :: String     → FileHandle
                        → IO FileHandle

upper     :: String     → String

(>>=) :: IO FileHandle → (FileHandle → IO b) → IO b
```

Resource Management: File Handling

- File Handling in Haskell

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

Resource Management: File Handling

- File Handling in Haskell 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

- File Handling in Haskell 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

- File Handling in Haskell 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

- File Handling in Haskell 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

- $\texttt{MonadError}$[1] type class in Haskell

$$\begin{aligned} &\texttt{class Monad m} \Rightarrow \texttt{MonadError e m} \mid \texttt{m} \to \texttt{e where} \\ &\quad \texttt{throwError} :: \texttt{e} \to \texttt{m a} \\ &\quad \texttt{catchError} :: \texttt{m a} \to (\texttt{e} \to \texttt{m a}) \to \texttt{m a} \end{aligned}$$

- $\texttt{MonadError}$ instance with $\texttt{IO}$ and $\texttt{Exception}$

$$\begin{aligned} &\texttt{throwError} :: \texttt{Exception} \to \texttt{IO a} \\ &\texttt{catchError} :: \texttt{IO a} \to (\texttt{Exception} \to \texttt{IO a}) \to \texttt{IO a} \end{aligned}$$

- $\texttt{throwError}$ start exception processing
- $\texttt{catchError}$ exception handler

[1] (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. New York, NY, USA: ACM, 1995, pp. 333–343)

Resource Management: Exception Handling

- Using `MonadError` in Haskell

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

- Exception may cause memory leak

Well typed programs do not go wrong.

— R. Milner

Well typed programs do not go wrong.

— R. Milner

~~Lights~~Types will guide you home...

— Coldplay

Contributions revisited

- Language design
  - Resources are "first class citizens"
  - Resources(variables) can be in sharing or separated
- QuB is logic of ***BI*** on steroids
  - Typing Environments as graphs
- Formalizing and proving important properties of QuB
  - Type system
  - Syntax directed type system (sound and complete)
  - Type inference algorithm $\mathcal{M}$ (sound)
- Working examples

Background Work: STLC and ***HM*** type system

$$\lambda x.M \begin{cases} \text{Abstract over computation} \\ \text{Define functions} \end{cases}$$

$$MN \begin{cases} \text{Do the computation} \\ \text{Use functions} \end{cases}$$

Background Work: STLC and *HM* type system

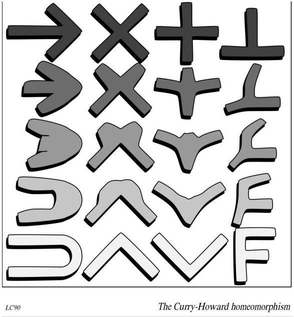
$$\lambda x.M \begin{cases} \text{Abstract over computation} \\ \text{Define functions} \end{cases} \qquad \lambda x.M : \tau \rightarrow \tau' \begin{cases} x : \tau \\ M : \tau' \end{cases}$$

$$MN \begin{cases} \text{Do the computation} \\ \text{Use functions} \end{cases} \qquad MN : \tau' \begin{cases} M : \tau \rightarrow \tau' \\ N : \tau \end{cases}$$

Background Work: Curry-Howard Correspondence

- Types are Propostions
- Programs are Proofs

HM type system ≡ Second Order Intuitionistic Propositional Logic



Source:

`http://lucacardelli.name/Artifacts/Drawings/CurryHoward/CurryHoward.pdf`

Background Work: Second Order Intuitionistic Propositional Logic

Language

Propostions & connectives $\quad A, B, C ::= x \mid A \supset B \mid \forall x.B \mid A \vee B \mid A \wedge B$

Context $\quad \Gamma, \Delta ::= \epsilon \mid \Gamma, A$

Implicit Structural Rules

$A, B \vdash A \quad A, B \vdash B$	Contraction
$A \vdash A \wedge A$	Weakening
$A, B \vdash B, A$	Exchange

Background Work: Second Order Intuitionistic Propositional Logic

Propositions are truth values not resources

Language

Propostions & connectives $\quad A, B, C ::= x \mid A \supset B \mid \forall x.B \mid A \vee B \mid A \wedge B$

Context $\quad \Gamma, \Delta ::= \epsilon \mid \Gamma, A$

Implicit Structural Rules

$A, B \vdash A \quad A, B \vdash B$	Contraction
$A \vdash A \wedge A$	Weakening
$A, B \vdash B, A$	Exchange

Background Work: Substructural Logic

System	Who	Year	Control
Revelance Logic[2]	Orlev	1928	[WKN]
Lambek Logic[3]	Lambek	1958	[EXCH]
Affine Logic[4]	Grishin	1974	[CTR]
Linear Logic[5]	Girard	1987	[WKN] [CTR]
Logic of Bunched Implications[6]	O'Hearn and Pym	1999	[WKN] [CTR]
Separation Logic [7]	Reynolds	2002	[WKN] [CTR]
⋮	⋮	⋮	⋮

[2] Ivan Orlov. 'The Logic of Compatibility of Propositions'. In: *Matematicheskii Sbornik* (1928)

[3] Joachim Lambek. 'The Mathematics of Sentence Structure'. In: *The American Mathematical Monthly* 65.3 (1958), pp. 154–170

[4] V Grishin. 'A nonstandard logic and its application to set theory'. Russian. In: *Studies in Formalized Languages and Nonclassical Logics* (1974)

[5] Jean-Yves Girard. 'Linear logic'. In: *Theoretical Computer Science* 50.1 (1987), pp. 1–101

[6] Peter W. O'Hearn and David J. Pym. 'The Logic of Bunched Implications'. In: *The Bulletin of Symbolic Logic* 5.2 (1999), pp. 215–244

[7] John C. Reynolds. 'Separation Logic: A Logic for Shared Mutable Data Structures'. In: *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*. 2002

Coffee Shop

1 cup coffee costs \$2

,

$\vdash$

Coffee Shop

1 cup coffee costs \$2

,


$\vdash$


,


$\nvdash$


two separate dollar bills necessary

Background Work: Logic of ***BI***

- Conjunction ($\wedge$) split into two flavors

$A \otimes B$	A is separate from B
$A \,\&\, B$	A is a different view of B or A shares with B

Background Work: Logic of ***BI***

- Conjunction ($\wedge$) split into two flavors

$A \otimes B$	A is separate from B
$A \mathbin{\&} B$	A is a different view of B or A shares with B

- ***BI*** contexts sensitive to different conjunction

$A, B \vdash A \otimes B$ $\qquad\qquad$ $A; B \vdash A \mathbin{\&} B$

Background Work: Logic of ***BI***

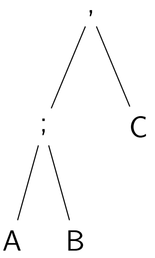
- Conjunction ($\wedge$) split into two flavors

$A \otimes B$	A is separate from B
$A \,\&\, B$	A is a different view of B or A shares with B

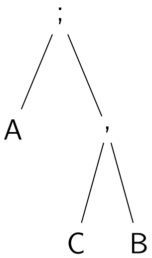
- ***BI*** contexts sensitive to different conjunction

$A, B \vdash A \otimes B$ $\qquad\qquad$ $A; B \vdash A \,\&\, B$

- Contexts form trees, called bunches



$(A; B), C$



$A; (B, C)$

Background Work: Logic of ***BI***

Context connectives guide structural rules

- Contraction

$$A \vdash A; A$$
$$A \nvdash A, A$$

- Weakening

$$A; A \vdash A \qquad A; B \vdash B \qquad A; B \vdash A$$
$$A, B \nvdash A \qquad A, B \nvdash B$$

Background Work: Logic of *BI*

Implications naturally get corresponding flavors

$$A \otimes B \vdash C \text{ iff } A \vdash B \twoheadrightarrow C$$

$$A \mathbin{\&} B \vdash C \text{ iff } A \vdash B \twoheadrightarrow C$$

Background work: Logic of ***BI***

Coffee Shop (Revisited)

1 cup coffee costs \$2   1 cookie costs \$1

$\vdash$

$-\!\!*$

$\vdash$

$\twoheadrightarrow$

$\nvdash$

$\twoheadrightarrow$

QuB: Curry-Howard interpretation of logic of BI

QuB: Types

$$\begin{array}{lrl} \text{Types} & \tau, \upsilon, \phi ::= & \textit{\textbf{t}} \mid \iota \mid \tau \to \tau \\ & \text{where} & \to \in \{\twoheadrightarrow\!\!\!\!*, \twoheadrightarrow\} \end{array}$$

- $-\!\!*$: Function type that is separate from its argument
- $\twoheadrightarrow$: Function type that is in sharing with its argument

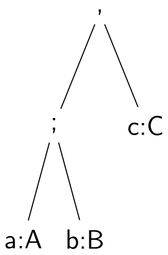
QuB: Expression Language

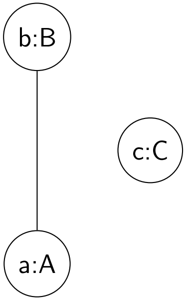
$$
\begin{aligned}
\text{Term Variables} \quad & x, y, z \in \mathsf{Var} \\
\text{Expressions} \quad & M, N ::= x \\
& \qquad\quad \mid \lambda^{-\!*} x.M \mid \lambda^{\twoheadrightarrow} x.M \\
& \qquad\quad \mid MN
\end{aligned}
$$

- $\lambda^{-\!*} x.M$: Argument x separate from M
- $\lambda^{\twoheadrightarrow} x.N$: Argument x sharing with M

QuB: Typing Environment

- Logic of ***BI***: Contexts are trees
- QuB: Contexts generalized to graphs





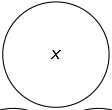
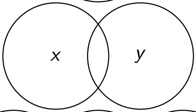
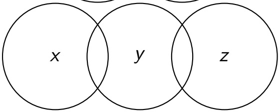
- Nodes are program objects
- (No) Edges represent (no) sharing

QuB: Typing Environment

Trees need explicit transformations

Graphs internalize the transformation

Sharing relation Ψ

reflexive	$\forall x.\ x \,\Psi\, x$	
symmetric	$\forall x, y.\ x \,\Psi\, y \Rightarrow y \,\Psi\, x$	
non-transitive	$\forall x, y, z.\ x \,\Psi\, y \wedge y \,\Psi\, z \not\Rightarrow x \,\Psi\, z$	

QuB: Typing Environment

Adjacency lists for sharing graphs

"x of type σ is in sharing with $\vec{y}$"

$$(x, \sigma, \vec{y}) \in \Gamma$$

Typing Context $\quad \Gamma, \Delta ::= \epsilon \mid \Gamma, x^{\vec{y}} : \sigma$

Examples: Basic Data Structures

- Multiplicative Product (Separating Pair)

$$\tau \otimes \tau' = \tau \twoheadrightarrow\!\!\!\!* \ \tau' \twoheadrightarrow\!\!\!\!* \ (\tau \twoheadrightarrow\!\!\!\!* \ \tau' \twoheadrightarrow\!\!\!\!* \ \upsilon) \twoheadrightarrow\!\!\!\!* \ \upsilon$$
$$\langle x, y \rangle = \lambda^{\twoheadrightarrow\!\!\!*} x.\lambda^{\twoheadrightarrow\!\!\!*} y.\lambda^{\twoheadrightarrow\!\!\!*} f.fxy$$

- Additive Product (Sharing Pair)

$$\tau \,\&\, \tau' = \tau \twoheadrightarrow\!\!\!\!* \ \tau' \twoheadrightarrow (\tau \twoheadrightarrow\!\!\!\!* \ \tau' \twoheadrightarrow \upsilon) \twoheadrightarrow \upsilon$$
$$\langle x; y \rangle = \lambda^{\twoheadrightarrow\!\!\!*} x.\lambda^{\twoheadrightarrow} y.\lambda^{\twoheadrightarrow} f.fxy$$

- Sums

$$\tau \oplus \tau' = (\tau \twoheadrightarrow\!\!\!\!* \ \upsilon) \rightarrow (\tau' \twoheadrightarrow \upsilon) \twoheadrightarrow \upsilon$$
$$\texttt{case } c \texttt{ of } \{f; g\} = \lambda^{\twoheadrightarrow\!\!\!*} c.\lambda^{\twoheadrightarrow} f.\lambda^{\twoheadrightarrow} g.cfg$$

$$\texttt{inl} : \tau \twoheadrightarrow\!\!\!\!* \ (\tau \oplus \tau') \qquad \texttt{inr} : \tau' \twoheadrightarrow\!\!\!\!* \ (\tau \oplus \tau')$$
$$\texttt{inl} = \lambda^{\twoheadrightarrow\!\!\!*} x.\lambda^{\twoheadrightarrow} f.\lambda^{\twoheadrightarrow} g.fx \qquad \texttt{inr} = \lambda^{\twoheadrightarrow\!\!\!*} y.\lambda^{\twoheadrightarrow} f.\lambda^{\twoheadrightarrow} g.gy$$

Programmer Friendly

- Define custom types

```
data Bool   = True | False
data List a = Nil | Cons a a
data Tree a = Leaf | Node a a
```

$$\vdots$$

Programmer Friendly

- Define custom types

```
data Bool   = True | False
data List a = Nil | Cons a a
data Tree a = Leaf | Node a a

        ⋮
```

- Type classes, functional dependencies

```
class Monad m a | a → m where
    return :: a → m a
    (>>=) :: a → (a → m b) → m b

        ⋮
```

More Background Work: Qualified Types

$$\Gamma \vdash M : \sigma$$

"Type of M is σ
and Γ specifies the free variables in M"

$$P \mid \Gamma \vdash M : \sigma$$

[89] "Type of M is σ
when predicates in P are satisfied
and Γ specifies the free variables in M"

Incorporate predicates into type language for finer grained polymorphism

More Background Work: Qualified Types

$$P \mid \Gamma \vdash M : \sigma$$

"Type of M is σ
when predicates in P are satisfied
and Γ specifies the free variables in M"[10]

Incorporate predicates into type language for finer grained polymorphism

$$(P \mid \sigma)$$

Instances of σ that satisfy P

More Background Work: Quill

Quill[11]: Qualified types + linear logic

Predicates:

- `Un` τ If τ does not have resources or can be copied or dropped easily.
- `Fun` τ If τ is a function type
- $\tau \geq \tau'$ If τ less restricting than τ'

More Background Work: Quill

Quill[12]: Qualified types + linear logic

Qualifying Types:

- Unrestricted Types: `Un Int`, `Un` Bool
- Restricted or Linear Types: FileHandle
- Function Types: `Fun (Int → Int)`, `Fun (String → String)`

QuB: Types and Predicates

$$
\begin{array}{rl}
\text{Types} & \tau, \upsilon, \phi ::= t \mid \iota \mid \tau \rightarrow \tau \\
& \text{where} \qquad \rightarrow \in \{\overset{!}{-\!\!\ast}, -\!\!\ast, \overset{!}{\twoheadrightarrow}, \twoheadrightarrow\} \\
\text{Predicates} & \pi, \omega ::= \texttt{Un}\ \tau \mid \texttt{ShFun}\ \phi \mid \texttt{SeFun}\ \phi \mid \tau \geq \tau'
\end{array}
$$

- $\texttt{SeFun}\ \phi$: ϕ is a function that is separate from its argument
- $\texttt{ShFun}\ \phi$: ϕ is a function that is in sharing with its argument
- $\texttt{Un}\ \tau$: τ does not have resources or they can be copied/dropped easily

QuB: Types and Predicates

$$
\begin{array}{lrl}
\text{Types} & \tau, \upsilon, \phi ::= & t \mid \iota \mid \tau \to \tau \\
 & \text{where} & \to \in \{ -\!\!*, \twoheadrightarrow \} \\
\text{Predicates} & \pi, \omega ::= & \mathtt{Un}\ \tau \mid \mathtt{ShFun}\ \phi \mid \mathtt{SeFun}\ \phi \mid \tau \geq \tau'
\end{array}
$$

- $-\!\!*$: Function type that is separate from its argument
- $\twoheadrightarrow$: Function type that is in sharing with its argument
- $\overset{!}{-\!\!*}$, $\overset{!}{\twoheadrightarrow}$: Unrestricted versions of $-\!\!*$ and $\twoheadrightarrow$

- Kind System with type constructors and qualified types[10, 11]

$$
\begin{array}{rl}
 & t, u \in \text{Type Variables} \\
\text{Kinds} & \kappa ::= \star \mid \kappa' \to \kappa \\
\text{Types} & \tau^{\kappa}, \phi^{\kappa} ::= t^{\kappa} \mid T^{\kappa} \mid \tau^{\kappa' \to \kappa} \tau^{\kappa'} \\
\text{Type Constructors} & T^{\kappa} \in \mathcal{T}^{\kappa} \\
\text{where} & \{\otimes, \&, \oplus, \overset{!}{\twoheadrightarrow}, \twoheadrightarrow, \overset{!}{\twoheadrightarrow}, \twoheadrightarrow\} \subseteq \mathcal{T}^{\star \to \star \to \star} \\
\text{Predicates} & \pi, \omega ::= \texttt{Un}\ \tau \mid \texttt{SeFun}\ \phi \mid \texttt{ShFun}\ \phi \mid \tau \geq \tau'
\end{array}
$$

QuB: Extension

- Sharing Pair

```
data ShPair a b = ShP a b

fst :: ShPair a b ↠ a
fst (ShP a b) = a              {- Succeeds typecheck -}

snd :: ShPair a b ↠ b
snd (ShP a b) = b              {- Succeeds typecheck -}
```

QuB: Extension

- Sharing Pair

```
data ShPair a b = ShP a b

fst :: ShPair a b ↠ a
fst (ShP a b) = a            {- Succeeds typecheck -}

snd :: ShPair a b ↠ b
snd (ShP a b) = b            {- Succeeds typecheck -}
```

- Separating Pair

```
data SePair a b = SeP a b

fst :: SePair a b ⇸ a
fst (SeP a b) = a            {- Fails typecheck -}

swap :: SePair a b ⇸ SePair b a
swap (SeP a b) = SePair b a  {- Succeeds typecheck -}
```

Motivation Revisited

What about filehandles, exceptions and memory leaks and runtime crashes?

Filehandles Revisited

File Handling API in QuB

$$\texttt{openFile} :: \texttt{FilePath} \multimap \texttt{IO FileHandle}$$

$$\texttt{closeFile} :: \texttt{FileHandle} \multimap \texttt{IO}\ ()$$

$$\texttt{readLine} :: \texttt{FileHandle} \multimap \texttt{IO}\ (\texttt{String}, \texttt{FileHandle})$$

$$\begin{aligned}\texttt{writeFile} :: \texttt{String} &\multimap \texttt{FileHandle} \\ &\multimap \texttt{IO}\ ((), \texttt{FileHandle})\end{aligned}$$

$$(\gg=) :: \texttt{IO a} \multimap (\texttt{a} \multimap \texttt{IO b}) \multimap \texttt{IO b}$$

Filehandles Revisited

```
do f  ← openFile "sample.txt"
   (s, f)  ← readLine f
   () ← closeFile f
   () ← closeFile f

(>>=) (openFile "sample.txt") (\ f →

(>>=) (readLine f) (\ (s, f) →

(>>=) (closeFile f) (\ _ → closeFile f)
```

Filehandles Revisited

```
do f  ← openFile "sample.txt"
   (s, f)  ← readLine f
   ()  ← closeFile f
   ()  ← closeFile f

(>>=) (openFile "sample.txt") (\ f →

(>>=) (readLine f) (\ (s, f) →

(>>=) (closeFile f) (\ _ → closeFile f)
```

Filehandles Revisited

Fails Typecheck!

```
do f      ← openFile "sample.txt"
   (s, f) ← readLine f
   ()     ← closeFile f
   ()     ← closeFile f
```

```
{- (>>=) :: IO a ⊸ (a ⊸ IO b) ⊸ IO b -}
(>>=) (openFile "sample.txt") (\ f →
{- (>>=) :: IO a ⊸ (a ⊸ IO b) ⊸ IO b -}
(>>=) (readLine f) (\ (s, f) →
{- (>>=) :: IO a ⊸ (a ⊸ IO b) ↠ IO b -}
(>>=) (closeFile f) (\ _ → closeFile f)
```

Exceptions Revisited

$$\begin{array}{l}
\texttt{openFile} :: \texttt{FilePath} \multimap \texttt{IO FileHandle} \\
\texttt{closeFile} :: \texttt{FileHandle} \multimap \texttt{IO} \; () \\
\texttt{readFile} :: \texttt{FileHandle} \multimap \texttt{IOF} \; (\texttt{String}, \texttt{FileHandle}) \\
\texttt{writeFile} :: \texttt{String} \multimap \texttt{FileHandle} \multimap \texttt{IOF} \; ((), \texttt{FileHandle})
\end{array}$$

$$\begin{array}{l}
\texttt{throw} :: \; \texttt{Exception} \multimap \texttt{IOF a} \\
\texttt{catch} :: \; \texttt{IOF a} \multimap (\texttt{Exception} \multimap \texttt{IO a}) \twoheadrightarrow \texttt{IO a}
\end{array}$$

- May not fail `IO a`
- May fail `IOF a`

Exceptions Revisited

```
readFromFile :: FilePath ⇾ IO (Either String String)
readFromFile fpath =
do fh  ← openFile fpath
   ((s, fh)  ← readLine fh
   let l = caps s
   () ← closeFile fh
   return $ Right l) `catch`
           (\e → do closeFile fh
           return $ Left "Could not read file")
```

- Filehandle fh is shared between the catch arguments
- Avoids memory leak

Contributions revisited

- Language design
  - Resources are "first class citizens"
  - Resources(variables) can be in sharing or separated
- QuB is logic of ***BI*** on steroids
  - Typing Environments as graphs
- Formalizing and proving important properties of QuB
  - Type system
  - Syntax directed type system (sound and complete)
  - Type inference algorithm $\mathcal{M}$ (sound)
- Working examples

Future Work

- Type inference algorithm $\mathcal{M}$ is incomplete. Terms can have two types.
  - $\{\mathtt{Un}\ A\} \mid \varnothing \vdash \lambda^{\to\!\!\!\ast} f.\lambda^{\to\!\!\!\ast} x.fxx : (A \to\!\!\!\!\ast\ A \to\!\!\!\!\ast\ B) \to\!\!\!\!\ast\ A \to\!\!\!\!\ast\ B$
  - $\varnothing \mid \varnothing \vdash \lambda^{\to\!\!\!\ast} f.\lambda^{\to\!\!\!\ast} x.fxx : (A \to\!\!\!\!\ast\ A \twoheadrightarrow B) \to\!\!\!\!\ast\ A \twoheadrightarrow B$
- Current semantics: call-by-value.

  Formalize resource correctness.

Thank You!

Q & A

References I

- 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. New York, NY, USA: ACM, 1995, pp. 333–343.
- Ivan Orlov. 'The Logic of Compatibility of Propositions'. In: *Matematicheskii Sbornik* (1928).
- Joachim Lambek. 'The Mathematics of Sentence Structure'. In: *The American Mathematical Monthly* 65.3 (1958), pp. 154–170.
- V Grishin. 'A nonstandard logic and its application to set theory'. Russian. In: *Studies in Formalized Languages and Nonclassical Logics* (1974).
- Jean-Yves Girard. 'Linear logic'. In: *Theoretical Computer Science* 50.1 (1987), pp. 1–101.

References II

Peter W. O'Hearn and David J. Pym. 'The Logic of Bunched Implications'. In: *The Bulletin of Symbolic Logic* 5.2 (1999), pp. 215–244.

John C. Reynolds. 'Separation Logic: A Logic for Shared Mutable Data Structures'. In: *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*. 2002.

Mark P. Jones. 'A theory of qualified types'. In: *Science of Computer Programming* 22.3 (1994), pp. 231 –256.

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. New York, NY, USA: ACM, 2016, pp. 448–461.

Henk Barendregt. 'Introduction to generalized type systems'. In: *Journal of Functional Programming* 1.2 (1991), 125–154.

References III

Mark P. Jones. 'Type Classes with Functional Dependencies'. In: *Proceedings of the 9th European Symposium on Programming*. ESOP 2000. Berlin, Germany: Springer-Verlag LNCS 1782, 2000.