

# QuB

## A Resource Aware Functional Programming Language

Apoorv Ingle

The University of Kansas

# Table of Contents

- 1 Introduction and Motivation
- 2 Background Work
- 3 QuB
- 4 Examples And Extensions
- 5 Conclusion and Future Work

Hard problems in programming

Naming variables

Hard problems in programming

**Resource management**

in evolving production code

Resources: Files, database connections, entity with a shared state

- 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
```

- File Handling in Haskell

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

- 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



- 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 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

- `MonadError`[4] 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

- Using 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")
```

- 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

- Design and implement QuB type system
  - Resources as first class citizens
  - Program objects are restricted or unrestricted
  - Functions that share resources with their arguments or are separate.
- Formalizing and proving important properties of QuB
- QuB is logic of **BI** with steroids
  - Environments as graphs
- Working examples

# Background Work: Simply Typed Lambda Calculus (STLC)

$$\begin{array}{l} \lambda x.M \left\{ \begin{array}{l} \text{Abstract over computation} \\ \text{Define functions} \end{array} \right. \\ MN \left\{ \begin{array}{l} \text{Do the computation} \\ \text{Use functions} \end{array} \right. \end{array}$$



# Background Work: Simply Typed Lambda Calculus (STLC)

$\lambda x.M$   $\left\{ \begin{array}{l} \text{Abstract over computation} \\ \text{Define functions} \end{array} \right.$

$MN$   $\left\{ \begin{array}{l} \text{Do the computation} \\ \text{Use functions} \end{array} \right.$

$$\frac{\Gamma_{x, x:\tau} \vdash M : \tau'}{\Gamma \vdash \lambda x.M : \tau \rightarrow \tau'} [\rightarrow I]$$

$$\frac{\Gamma \vdash M : \tau \rightarrow \tau' \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \tau'} [\rightarrow E]$$

# Background Work: Simply Typed Lambda Calculus (STLC)

$\lambda x.M$   $\left\{ \begin{array}{l} \text{Abstract over computation} \\ \text{Define functions} \end{array} \right.$

$MN$   $\left\{ \begin{array}{l} \text{Do the computation} \\ \text{Use functions} \end{array} \right.$

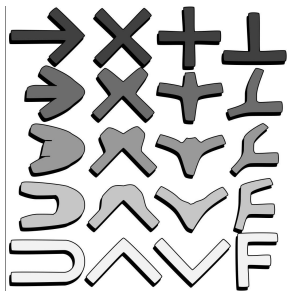
$$\frac{\Gamma_{x, x:\tau} \vdash M : \tau'}{\Gamma \vdash \lambda x.M : \tau \rightarrow \tau'} [\rightarrow I] \qquad \frac{\Gamma \vdash M : \tau \rightarrow \tau' \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \tau'} [\rightarrow E]$$

Hinley-Milner (**HM**) type system ensures sane programs

# Background Work: Curry-Howard Correspondence

- Types are Propositions
- Programs are Proofs

**HM** type system  $\equiv$  Second Order Intuitionistic Propositional Logic



LC90

*The Curry-Howard homomorphism*

Source: <http://lucacardelli.name/Artifacts/Drawings/CurryHoward/CurryHoward.pdf>

# Background Work: Second Order Intuitionistic Propositional Logic

## Language

Propositions & connectives  $A, B, C ::= x \mid A \supset B \mid \forall x.B \mid \dots$

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

## Logic Rules

$$\frac{}{A \vdash A} [\text{Ax}]$$

$$\frac{\Gamma \vdash B \quad x \notin \Gamma}{\forall x.B} [\forall I]$$

$$\frac{\Gamma \vdash \forall x.B \quad \Gamma \vdash A}{B[x/A]} [\forall E]$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} [\supset I]$$

$$\frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B} [\supset E]$$

# Background Work: Second Order Intuitionistic Propositional Logic

## Propostions are truth values not resources

Language

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

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

Logic Rules

$$\frac{}{A \vdash A} [Ax]$$

$$\frac{\Gamma \vdash B \quad x \notin \Gamma}{\forall x.B} [\forall I]$$

$$\frac{\Gamma \vdash \forall x.B \quad \Gamma \vdash A}{B[x/A]} [\forall E]$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} [\supset I]$$

$$\frac{\Gamma \vdash A \supset B \quad \Gamma \vdash A}{\Gamma \vdash B} [\supset E]$$

- Structural rules implicit in intuitionistic propositional logics

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B} \text{ [WKN]}$$

$$\frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} \text{ [CTR]}$$

$$\frac{\Gamma, \Delta \vdash B}{\Delta, \Gamma \vdash B} \text{ [EXCH]}$$

- Structural rules implicit in intuitionistic propositional logics

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B} [\text{WKN}] \quad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B} [\text{CTR}] \quad \frac{\Gamma, \Delta \vdash B}{\Delta, \Gamma \vdash B} [\text{EXCH}]$$

- Control the use of [WKN] and [CTR]

Propositions now behave like resources

# Background Work: Substructural Logic

System	Who	Restrictions
Linear Logic[1]	Girard	[WKN] [CTRN]
Lambek Logic[3]	Lambek	[EXCH]
Logic of Bunched Implications[7]	O'Hearn and Pym	[WKN] [CTRN]
⋮	⋮	⋮



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

“Type of  $M$  is  $\sigma$   
when predicates in  $P$  are satisfied  
and  $\Gamma$  specifies the free variables in  $M$ ”[2]

General framework, incorporates predicates into type language

# Background Work: Quill

Quill[5]: Qualified types + linear logic

Predicates:

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

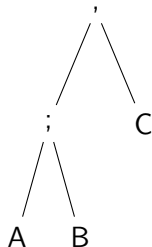
Quill[5]: Qualified types + linear logic

Qualifying Types:

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

# Background Work: Logic of Bunched Implications (*BI*)

- Contexts as trees, called bunches[7]
- Two connective used to combine contexts:  $A; B$  or  $A, B$



# Background Work: Logic of *BI*

- Context connective guides choice of implication

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} [\multimap I]$$

$$\frac{\Gamma; A \vdash B}{\Gamma \vdash A \multimap B} [\multimap I]$$

- Contraction

$$\begin{array}{ll} A; B \vdash A & A; B \vdash B \\ A, B \nvdash A & A, B \nvdash B \end{array}$$

- Weakening

$$\begin{array}{ll} A; B \vdash A; B; B & A; B \vdash A; A; B \\ A, B \nvdash A, B, B & A, B \nvdash A, A, B \end{array}$$

# Background Work: Logic of *BI*

# Background Work: $\alpha\lambda$ -calculus

Curry-Howard interpretation of logic of **BI**[6]

Language

Context  $\Gamma, \Delta ::= \{\}_m \mid \{\}_a \mid x : \tau \mid \Gamma, \Delta \mid \Gamma; \Delta$

Types  $\tau, \upsilon ::= t \mid \iota \mid \tau \rightarrow \tau \mid \tau \multimap \tau$

Expressions  $M, N ::= x \mid \lambda x. M \mid \alpha x. M \mid MN$

Typing rules

$$\begin{array}{c} \frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau} [\text{VAR}] \quad \frac{\Gamma; \Gamma \vdash M : \tau}{\Gamma \vdash M : \tau} [\text{CTR}] \quad \frac{\Gamma \vdash M : \tau}{\Gamma; \Delta \vdash M : \tau} [\text{WKN}] \\[10pt] \frac{\Gamma_x, x : \tau \vdash M : \tau'}{\Gamma \vdash \lambda x. M : \tau \multimap \tau'} [*I] \quad \frac{\Gamma \vdash M : \tau \multimap \tau' \quad \Delta \vdash N : \tau}{\Gamma, \Delta \vdash MN : \tau'} [*E] \\[10pt] \frac{\Gamma_x; x : \tau \vdash M : \tau'}{\Gamma \vdash \alpha x. M : \tau \rightarrow \tau'} [\rightarrow I] \quad \frac{\Gamma \vdash M : \tau \rightarrow \tau' \quad \Delta \vdash N : \tau}{\Gamma; \Delta \vdash MN : \tau'} [\rightarrow E] \end{array}$$

- Quill: Qualified types for type system based on linear logic
- QuB: Qualified types for type system based on logic of ***BI***



# QuB: Types and Predicates

Types  $\tau, v, \phi ::= t \mid \iota \mid \tau \rightarrow \tau$

where  $\rightarrow \in \{ \overset{!}{\rightarrow}, \multimap, \multimap, \multimap, \multimap \}$

Predicates  $\pi, \omega ::= \text{Un } \tau \mid \text{SeFun } \tau \mid \text{ShFun } \tau \mid \tau \geq \tau'$

Qualified Types  $\rho ::= \tau \mid \pi \Rightarrow \rho$

Type schemes  $\sigma ::= \rho \mid \forall t. \sigma$

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

# QuB: Types and Predicates

Types  $\tau, v, \phi ::= t \mid \iota \mid \tau \rightarrow \tau$

where  $\rightarrow \in \{ \overset{!}{\rightarrow}*, \rightarrow*, \overset{!}{\rightarrow}\!\!\rightarrow, \rightarrow\!\!\rightarrow \}$

Predicates  $\pi, \omega ::= \text{Un } \tau \mid \text{SeFun } \tau \mid \text{ShFun } \tau \mid \tau \geq \tau'$

Qualified Types  $\rho ::= \tau \mid \pi \Rightarrow \rho$

Type schemes  $\sigma ::= \rho \mid \forall t. \sigma$

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

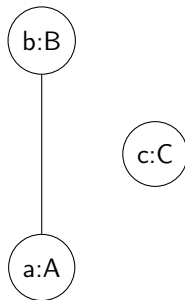
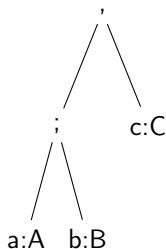
Term Variables  $x, y, z \in \text{Var}$

Expressions  $M, N ::= x \mid \lambda^* x.M \mid \lambda^{\rightarrow} x.M \mid MN \mid \text{let } x = M \text{ in } N$

- $\lambda^* x.M$ : Introduces  $\rightarrow^*$  function
- $\lambda^{\rightarrow} x.N$ : Introduces  $\rightarrow$  function

# QuB: Typing Environment

- Logic of **BI**: Contexts are trees
- QuB: Contexts are flattened sharing graphs



- Sharing relation  $\Psi$

$$\forall_{x \in \text{dom}(\Psi)} x \in \Psi(x) \quad (\text{reflexive})$$

$$\forall_{x, y \in \text{dom}(\Psi)} \text{ if } y \in \Psi(x) \text{ then } x \in \Psi(y) \quad (\text{symmetric})$$

$$\forall_{x, y, z \in \text{dom}(\Psi)} \text{ if } y \in \Psi(x) \text{ and } z \in \Psi(y) \not\Rightarrow z \in \Psi(x) \quad (\text{non-transitive})$$

# QuB: Typing Environment

“ $x$  of type  $\sigma$  is in sharing with  $\vec{y}$ ”

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

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

$$\text{Vars}(\Gamma, x^{\vec{y}} : \tau) = \text{Vars}(\Gamma) \cup \{x\}$$

$$\text{Shared}(\Gamma, x^{\vec{y}} : \tau) = \text{Shared}(\Gamma) \cup \{\vec{y}\}$$

$$\text{Used}(\Gamma) = \text{Vars}(\Gamma) \cup \text{Shared}(\Gamma)$$

$$(\Gamma, x^{\vec{y}} : \tau)^{[a \mapsto \vec{b}]} = \begin{cases} a \notin \vec{y} & (\Gamma^{[a \mapsto \vec{b}]}, x^{\vec{y}} : \tau) \\ a \in \vec{y} & (\Gamma^{[a \mapsto \vec{b}]}, x^{(\vec{y} \setminus a) \cup \vec{b}} : \tau) \end{cases}$$

$$\Gamma^{[\vec{a} \mapsto \vec{b}]} = (\dots ((\Gamma^{[a_1 \mapsto \vec{b}]})^{[a_2 \mapsto \vec{b}]}) \dots)^{[a_n \mapsto \vec{b}]}$$

$$\Gamma \otimes \Gamma' = \Gamma \sqcup \Gamma' \quad \text{if } \text{Vars}(\Gamma) \# \text{Used}(\Gamma') \wedge \text{Vars}(\Gamma') \# \text{Used}(\Gamma)$$

$$\Gamma \oplus \Gamma' = \Gamma \sqcup \Gamma' \quad \text{if } \text{Used}(\Gamma) = \text{Used}(\Gamma')$$

## Structural Rules

$$\frac{}{P \mid x^{\vec{y}} : \sigma \vdash x : \sigma} \text{[ID]}$$

$$\frac{P \mid \Gamma \oplus \Delta \oplus \Delta \vdash M : \sigma \quad P \vdash \Delta \text{ un}}{P \mid \Gamma \oplus \Delta \vdash M : \sigma} \text{[CTR-UN]} \quad \frac{P \mid \Gamma \oplus \Delta \oplus \Delta \vdash M : \sigma}{P \mid \Gamma \oplus \Delta \vdash M : \sigma} \text{[CTR-SH]}$$

$$\frac{P \mid \Gamma \vdash M : \sigma \quad P \vdash \Delta \text{ un}}{P \mid \Gamma \oplus \Delta \vdash M : \sigma} \text{[WKN-UN]} \quad \frac{P \mid \Gamma \vdash M : \sigma}{P \mid \Gamma \oplus \Delta \vdash M : \sigma} \text{[WKN-SH]}$$

## Connective Rules

$$\frac{P \mid \Gamma \vdash M : \sigma \quad P' \mid \Gamma'_x, x : \sigma \vdash N : \tau}{P \cup P' \mid \Gamma \sqcup \Gamma' \vdash (\text{let } x = M \text{ in } N) : \tau} [\text{LET}]$$

$$\frac{P \mid \Gamma \vdash M : \sigma \quad t \notin \text{fvs}(\Gamma) \cup \text{fvs}(P)}{P \mid \Gamma \vdash M : \forall t. \sigma} [\forall \text{ I}]$$

$$\frac{P \mid \Gamma \vdash M : \forall t. \sigma}{P \mid \Gamma \vdash M : [\tau \backslash t] \sigma} [\forall \text{ E}]$$

$$\frac{P, \pi \mid \Gamma \vdash M : \rho}{P \mid \Gamma \vdash M : \pi \Rightarrow \rho} [\Rightarrow \text{ I}]$$

$$\frac{P \mid \Gamma \vdash M : \pi \Rightarrow \rho \quad P \vdash \pi}{P \mid \Gamma \vdash M : \rho} [\Rightarrow \text{ E}]$$

$$\frac{P \Rightarrow \text{ShFun } \phi \quad P \vdash \Gamma \geq \phi \quad P \mid \Gamma[\emptyset \mapsto \{x\}], x^{\text{Vars}(\Gamma)} : \tau \vdash M : \tau'}{P \mid \Gamma \vdash \lambda^{\rightarrow} x. M : \phi \tau \tau'} [\rightarrow \text{ I}]$$

$$\frac{P \Rightarrow \text{ShFun } \phi \quad P \mid \Gamma \vdash M : \phi \tau \tau' \quad P \mid \Gamma' \vdash N : \tau}{P \mid \Gamma \oplus \Gamma' \vdash MN : \tau'} [\rightarrow \text{ E}]$$

$$\frac{P \Rightarrow \text{SeFun } \phi \quad P \vdash \Gamma \geq \phi \quad P \mid \Gamma, x^{\emptyset} : \tau \vdash M : \tau'}{P \mid \Gamma \vdash \lambda^* x. M : \phi \tau \tau'} [* \text{ I}]$$

$$\frac{P \Rightarrow \text{SeFun } \phi \quad P \mid \Gamma \vdash M : \phi \tau \tau' \quad P \mid \Gamma' \vdash N : \tau}{P \mid \Gamma \otimes \Gamma' \vdash MN : \tau'} [* \text{ E}]$$

# QuB: Typing Terms

$$\begin{array}{c}
 \frac{}{\emptyset \mid y^{\emptyset} : \tau' \vdash y : \tau'} \text{[ID]} \quad \frac{}{\emptyset \mid x^{\emptyset} : \tau \vdash x : \tau} \text{[ID]} \quad \frac{}{\emptyset \mid f^{\emptyset} : \tau \multimap \tau' \multimap v \vdash f : \tau \multimap \tau' \multimap v} \text{[ID]} \\
 \hline
 \frac{}{\emptyset \mid x^{\emptyset} : \tau \oplus f^{\emptyset} : \tau \multimap \tau' \multimap v \vdash fx : (\tau' \multimap v)} \text{[*E]} \\
 \hline
 \frac{}{\emptyset \mid y^{\emptyset} : \tau' \oplus x^{\emptyset} : \tau \oplus f^{\emptyset} : \tau \multimap \tau' \multimap v \vdash fxy : v} \text{[*E]} \\
 \hline
 \frac{}{\emptyset \mid x^{\emptyset} : \tau \oplus y^{\emptyset} : \tau' \oplus f^{\emptyset} : \tau \multimap \tau' \multimap v \vdash fxy : v} \text{[EXCH]} \\
 \hline
 \frac{}{\emptyset \mid x^{\emptyset} : \tau \oplus y^{\emptyset} : \tau' \vdash \lambda^* f. fxy : (\tau \multimap \tau' \multimap v) \multimap v} \text{[* I]} \\
 \hline
 \frac{}{\emptyset \mid x^{\emptyset} : \tau \vdash \lambda^* y. \lambda^* f. fxy : \tau' \multimap (\tau \multimap \tau' \multimap v) \multimap v} \text{[* I]} \\
 \hline
 \frac{}{\emptyset \mid I \vdash \lambda^* x. \lambda^* y. \lambda^* f. fxy : \tau \multimap \tau' \multimap (\tau \multimap \tau' \multimap v) \multimap v} \text{[=]}
 \end{array}$$



# QuB: Typing Terms

$$\begin{array}{c}
 \frac{}{\emptyset \mid y^{xf} : \tau' \vdash y : \tau'} \text{[ID]} \quad \frac{}{\emptyset \mid x^{fy} : \tau \vdash x : \tau} \text{[ID]} \quad \frac{}{\emptyset \mid f^{xy} : \tau \rightarrow \tau' \rightarrow v \vdash f : \tau \rightarrow \tau' \rightarrow v} \text{[ID]} \\
 \hline
 \frac{}{\emptyset \mid x^{xy} : \tau \oplus f^{xy} : \tau \rightarrow \tau' \rightarrow v \vdash fx : (\tau' \rightarrow v)} \text{[* E]} \\
 \hline
 \frac{}{\emptyset \mid y^{xf} : \tau' \oplus x^{yf} : \tau \oplus f^{xy} : \tau \rightarrow \tau' \rightarrow v \vdash fxy : v} \text{[EXCH]} \\
 \frac{}{\emptyset \mid x^{yf} : \tau \oplus y^{xf} : \tau' \oplus f^{xy} : \tau \rightarrow \tau' \rightarrow v \vdash fxy : v} \text{[→ I]} \\
 \frac{}{\emptyset \mid x^y : \tau \oplus y^x : \tau' \vdash \lambda^{\rightarrow} f. fxy : (\tau \rightarrow \tau' \rightarrow v) \rightarrow v} \text{[→ I]} \\
 \frac{}{\emptyset \mid x^{\emptyset} : \tau \vdash \lambda^{\rightarrow} y. \lambda^{\rightarrow} f. fxy : \tau' \rightarrow (\tau \rightarrow \tau' \rightarrow v) \rightarrow v} \text{[* I]} \\
 \hline
 \emptyset \mid I \vdash \lambda^* x. \lambda^{\rightarrow} y. \lambda^{\rightarrow} f. fxy : \tau * \tau' \rightarrow (\tau \rightarrow \tau' \rightarrow v) \rightarrow v
 \end{array}$$

Term structure  $\leftrightarrow$  Typing Rule

# QuB: Syntax Directed Typing Rules

$$\frac{P \vdash \Gamma_{\bar{y}} \text{ un} \quad (P \Rightarrow \tau) \sqsubseteq \sigma}{P \mid \Gamma, x^{\bar{y}} : \sigma \vdash^s x : \tau} [\text{VAR}^s]$$

$$\frac{Q \mid (\Gamma'_x \oplus \Gamma''_x) \otimes \Delta \vdash^s M : v \quad P \vdash \Delta \text{ un} \quad P \mid (\Gamma_x, x^{\emptyset} : \sigma) \oplus \Gamma''_x \otimes \Delta \vdash^s N : \tau \quad \sigma = \text{Gen}(\{\Gamma' \oplus \Gamma''_x \otimes \Delta\}, Q \Rightarrow v)}{P, Q \mid (\Gamma \otimes \Gamma') \oplus \Gamma'' \otimes \Delta \vdash^s (\text{let } x = M \text{ in } N) : \tau} [\text{Let}^s]$$

$$[\text{VAR}^s] \equiv [\text{ID}] + [\forall E] + [\Rightarrow E]$$

$$[\text{LET}^s] \equiv [\text{LET}] + [\forall I] + [\Rightarrow I]$$

# QuB: Syntax Directed Typing Rules

$$\frac{P \Rightarrow \text{SeFun } \phi \quad P \vdash \Gamma \geq \phi \quad P \mid \Gamma \otimes x^{\emptyset} : \tau \vdash^s M : v}{P \mid \Gamma \vdash^s \lambda^{*} x. M : \phi \tau v} \quad [*I^s] \quad \frac{P \Rightarrow \text{ShFun } \phi \quad P \vdash \Gamma \geq \phi \quad P \mid \Gamma[\emptyset \mapsto \{x\}] \oplus x^{\text{Vars}(\Gamma)} : \tau \vdash^s M : v}{P \mid \Gamma \vdash^s \lambda^{\rightarrow} x. M : \phi \tau v} \quad [\rightarrow I^s]$$

$$\frac{P \mid \Gamma \otimes \Delta \vdash^s M : \phi v \tau \quad P \mid \Gamma' \otimes \Delta \vdash^s N : v \quad P \vdash \Delta \text{ un} \quad (\Gamma \tilde{\otimes} \Gamma' \wedge (P \Rightarrow \text{ShFun } \phi)) \vee (\Gamma \tilde{\otimes} \Gamma' \wedge (P \Rightarrow \text{SeFun } \phi))}{P \mid \Gamma \sqcup \Gamma' \otimes \Delta \vdash^s MN : \tau} \quad [\text{App}^s]$$

Context Operations

$$\Gamma \tilde{\otimes} \Delta = \text{Used}(\Gamma) \# \text{Vars}(\Delta) \wedge \text{Used}(\Delta) \# \text{Vars}(\Gamma)$$

$$\Gamma \tilde{\oplus} \Delta = \text{Used}(\Gamma) = \text{Used}(\Delta)$$

$$[\rightarrow I] \equiv [\rightarrow I] \quad [*I] \equiv [*I]$$

$$[\text{APP}^s] \equiv [\rightarrow E] + [*E]$$

## Theorem (Soundness of $\vdash^s$ )

*If  $P \mid \Gamma \vdash^s M : \tau$  then  $P \mid \Gamma \vdash M : \tau$*

## Theorem (Completeness of $\vdash^s$ )

*If  $P \mid \Gamma \vdash M : \sigma$  then  $\exists Q, \tau$  such that  $Q \mid \Gamma \vdash^s M : \tau$  and  $(P \mid \sigma) \sqsubseteq \text{Gen}(\Gamma, Q \Rightarrow \tau)$*

Original Type System  $\equiv$  Syntax Directed Typing Rules

Proof in Original Type System  $\equiv$  Proof in Syntax Directed Typing Rules

$$\boxed{\mathcal{M}(S, \Psi, \Gamma \vdash M : \tau) = P, S', \Sigma, \Psi'}$$

$$\begin{aligned} \mathcal{M}(S, \Psi, \Gamma \vdash x : \tau) &= ([\vec{u}/\vec{t}]P), S' \circ S, \{x\}, \Psi \\ \text{where} \quad (x : \forall \vec{t}. P \Rightarrow v) &\in S\Gamma \\ S' &= \mathcal{U}([\vec{u}/\vec{t}]v, S\tau) \end{aligned}$$

$$\begin{aligned} \mathcal{M}(S, \Psi, \Gamma \vdash \lambda^{\rightarrow} x. M : \tau) &= \{P \cup Q\}, S', \Sigma \setminus x, \Psi'' \\ \text{where} \quad P; S'; \Sigma; \Psi' &= \mathcal{M}(\mathcal{U}(\tau, u_1 u_2 u_3) \circ S, \Psi, \Gamma, x : u_2 \vdash M : u_3) \\ \Psi'' &= \{\forall_{y \in \text{dom}(\Psi')}. \Psi'(y) + x\} \cup \{(x, \{y \mid y \in \text{dom}(\Gamma)\})\} \\ Q &= \{\text{ShFun } u_1\} \cup \text{Leq}(u_1, \Gamma \upharpoonright_{\Sigma}) \cup \text{Weaken}(x, u_2, \Sigma, \Psi'') \end{aligned}$$

$$\begin{aligned} \mathcal{M}(S, \Psi, \Gamma \vdash \lambda^* x. M : \tau) &= \{P \cup Q\}, S', \Sigma \setminus x, \Psi'' \\ \text{where} \quad P; S'; \Sigma; \Psi' &= \mathcal{M}(\mathcal{U}(\tau, u_1 u_2 u_3) \circ S, X; \Gamma, x : u_2 \vdash M : u_3) \\ \Psi'' &= \Psi' \cup \{(x, \{x\})\} \\ Q &= \{\text{SeFun } u_1\} \cup \text{Leq}(u_1, \Gamma \upharpoonright_{\Sigma}) \cup \text{Weaken}(x, u_2, \Sigma, \Psi'') \end{aligned}$$

$$\mathcal{M}(S, \Psi, \Gamma \vdash M : \tau) = P, S', \Sigma, \Psi'$$

$$\mathcal{M}(S, \Psi, \Gamma \vdash MN : \tau) = \{P \cup P' \cup Q\}, R', \Sigma \cup \Sigma', \Psi''$$

where  $P; R; \Sigma; \Psi' = \mathcal{M}(S, \Psi, \Gamma \vdash M : u_1 u_2 \tau)$

$$P'; R'; \Sigma'; \Psi'' = \mathcal{M}(SR, \Psi', S\Gamma \vdash N : u_2)$$

if  $\mathcal{C}(\Gamma, \Psi'', \Sigma) = \mathcal{C}(\Gamma, \Psi'', \Sigma')$

then  $Q = \{\text{ShFun } u_1\}$

else if  $(\Sigma \# \mathcal{C}(R\Gamma, \Psi'', \Sigma'))$  and  $\Sigma' \# \mathcal{C}(R\Gamma, \Psi'', \Sigma)$

then  $Q = \{\text{SeFun } u_1\}$

$$\mathcal{M}(S, \Psi, \Gamma \vdash \text{let } x = M \text{ in } N : \tau) = (P \cup Q), R', \Sigma \cup \{\Sigma' \setminus x\}, \Psi''$$

where  $P; R; \Sigma; \Psi' = \mathcal{M}(S, \Psi, \Gamma \vdash M : u_1)$

$$\sigma = \text{GenI}(R\Gamma; R(P \Rightarrow u_1))$$

$$P'; R'; \Sigma'; \Psi'' = \mathcal{M}(R, \Psi', \Gamma, x : \sigma \vdash N : \tau)$$

$$Q = \text{Un}(\Gamma|_{\Sigma \cap \Sigma'}) \cup \text{Weaken}(x, \sigma, \Sigma', \Psi'')$$

## Theorem (Soundness of $\mathcal{M}$ )

*if  $\mathcal{M}(S, \Psi, \Gamma \vdash M : \tau) = P, S', \Sigma, \Psi'$  then  $S'P \mid S'(\Gamma|_{\Sigma}) \vdash M : S'\tau$*

Algorithm  $\mathcal{M}$   $\rightarrow$  Type system



# Examples: Basic Structures

- Multiplicative Product

$$\begin{aligned}\tau \otimes \tau' &= \tau \multimap \tau' \multimap (\tau \multimap \tau' \multimap v) \multimap v \\ (,) &= \lambda^* x. \lambda^* y. \lambda^* f. fxy\end{aligned}$$

- Additive Product

$$\begin{aligned}\tau \& \tau' &= \tau \multimap \tau' \multimap (\tau \multimap \tau' \multimap v) \multimap v \\ (;) &= \lambda^* x. \lambda^{\multimap} y. \lambda^{\multimap} f. fxy\end{aligned}$$

- Sums

$$\begin{aligned}\tau \oplus \tau' &= (\tau \rightarrow v) \rightarrow (\tau' \rightarrow v) \rightarrow v \\ \text{case } c \text{ of } \{f; g\} &= \lambda^* c. \lambda^{\multimap} f. \lambda^{\multimap} g. c f g\end{aligned}$$

$$\text{inl} : \tau \multimap (\tau \oplus \tau')$$

$$\text{inr} : \tau' \multimap (\tau \oplus \tau')$$

$$\text{inl} = \lambda^* x. \lambda^{\multimap} f. \lambda^{\multimap} g. f x$$

$$\text{inr} = \lambda^* y. \lambda^{\multimap} f. \lambda^{\multimap} g. g y$$

- User defined types and type classes

- User defined types and type classes
- Kind System with type constructors

Type Variables  $t, u \in \text{Type Variables}$

Kinds  $\kappa ::= \star \mid \kappa' \rightarrow \kappa$

Types  $\tau^\kappa ::= t^\kappa \mid T^\kappa \mid \tau^{\kappa' \rightarrow \kappa} \tau^{\kappa'}$

Type Constructors  $T^\kappa \in \mathcal{T}^\kappa$  where  $\{\otimes, \&, \oplus, \frac{1}{\ast}, \ast, \multimap, \Rightarrow\} \subseteq \mathcal{T}^{\star \rightarrow \star \rightarrow}$

Predicates  $\pi, \omega ::= \text{Un } \tau \mid \text{SeFun } \tau \mid \text{ShFun } \tau \mid \tau \geq \tau'$

Qualified Types  $\rho ::= \tau^\star \mid \pi \Rightarrow \rho$

Type schemes  $\sigma ::= \rho \mid \forall t. \sigma$

# Conclusion and Future Work

What next?

Thank You!

Q & A

- [1] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.
- [2] Mark P. Jones. A theory of qualified types. *Science of Computer Programming*, 22(3):231 – 256, 1994.
- [3] Joachim Lambek. The mathematics of sentence structure. 65(3):154–170, 1958.
- [4] 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.
- [5] 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.

- [6] Peter W. O'Hearn. On bunched typing. *Journal of functional Programming*, 13(4):747–796, 2003.
- [7] Peter W. O'Hearn and David J. Pym. The logic of bunched implications. *The Bulletin of Symbolic Logic*, 5(2):215–244, 1999.