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

$$
\begin{aligned}
&\texttt{openFile} &&:: \texttt{FilePath} &&\to \texttt{IO FileHandle} \\
&\texttt{closeFile} &&:: \texttt{FileHandle} &&\to \texttt{IO } () \\
&\texttt{readLine} &&:: \texttt{FileHandle} &&\to \texttt{IO } (\texttt{String}, \texttt{FileHandle}) \\
&\texttt{writeLine} &&:: \texttt{FileHandle} &&\to \texttt{String} \\
& && &&\to \texttt{IO FileHandle}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{upper} &&:: \texttt{String} &&\to \texttt{String}
\end{aligned}
$$

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

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

- 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

- `MonadError`[1] type class in Haskell

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

- `MonadError` instance with `IO` and `Exception`

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

- `throwError` start exception processing
- `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

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

Bootstrapping: STLC and $\boldsymbol{HM}$ type system

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

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

Bootstrapping: STLC and *HM* type system

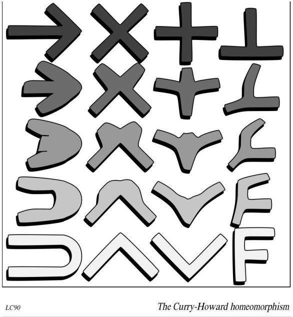
$$\lambda x.M \begin{cases} \text{Abstract over computation} \\ \text{Define functions} \end{cases} \qquad \lambda x.M : \tau \to \tau' \begin{cases} x : \tau \\ M : \tau' \end{cases}$$

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

Bootstrapping: Curry-Howard Correspondence

HM type system ≡ Second Order Intuitionistic Propositional Logic

- Types are Propostions
- Programs are Proofs



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

Bootstrapping: S O Intuitionistic Propositional Logic

$\boxed{\text{Language}}$

Propositions and Connectives $\quad A, B, C ::= x \mid A \supset B \mid \forall x.B \mid A \vee B \mid A \wedge B$

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

$\boxed{\text{Implicit Structural Rules}}$

$$\begin{array}{cc} A, B \vdash A \quad A, B \vdash B & \text{Weakening} \\ A \vdash A \wedge A & \text{Contraction} \\ A, B \vdash B, A & \text{Exchange} \end{array}$$

Propositions are truth values not resources

Language

Propositions and Connectives $A, B, C ::= x \mid A \supset B \mid \forall x.B \mid A \vee B \mid A \wedge B$

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

Implicit Structural Rules

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

Bootstrapping: 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

,

$\vdash$

Bootstrapping: Logic of ***BI***

Coffee Shop

1 cup coffee costs $2

,


$\vdash$


,


$\nvdash$


two separate dollar bills necessary

Bootstrapping: Logic of $\boldsymbol{BI}$

- 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

Bootstrapping: 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

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

Bootstrapping: Logic of ***BI***

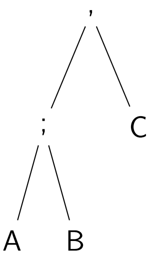
- Conjunction ($\wedge$) split into two flavors

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

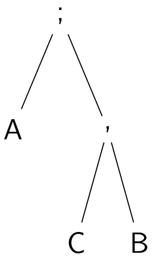
- ***BI*** contexts sensitive to different conjunction

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

- Contexts form trees, called bunches



$(A; B), C$



$A; (B, C)$

Bootstrapping: Logic of *BI*

Context connectives guide structural rules

- Contraction

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

- Weakening

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

Coffee Shop (Revisited)

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

 $\vdash$  $-\!\!*$ 

 $\vdash$  $\twoheadrightarrow$ 

 $\nvdash$  $\twoheadrightarrow$ 

Bootstrapping: Logic of ***BI***

Implications get corresponding flavors

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

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

QuB: Curry-Howard interpretation of logic of $\boldsymbol{BI}$

QuB: Types

$$\begin{aligned} \text{Types} \quad \tau, \upsilon, \phi &::= \boldsymbol{t} \mid \iota \mid \tau \to \tau \\ \text{where} \qquad &\to \in \{ -\!\!*, \twoheadrightarrow \} \end{aligned}$$

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

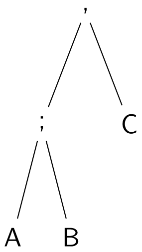
QuB: Expression Language

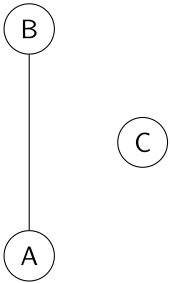
$$
\begin{array}{rrl}
\text{Term Variables} & x, y, z & \in \text{Variables} \\
\text{Expressions} & M, N & ::= x \\
 & & \mid \lambda^{-\!*} x.M \mid \lambda^{\twoheadrightarrow} x.M \\
 & & \mid MN
\end{array}
$$

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

QuB: Typing Environment

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



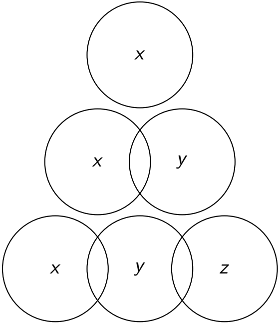


- Nodes are program objects
- (No) Edges represent (no) sharing

QuB: Typing Environment

Sharing relation Ψ

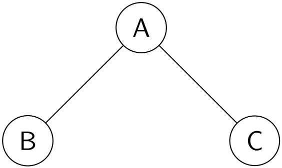
reflexive	$\forall x.\ x \,\Psi\, x$	
symmetric	$\forall x, y.\ x \,\Psi\, y \Rightarrow y \,\Psi\, x$	
non-transitive	$\forall x, y, z.\ x \,\Psi\, y \wedge y \,\Psi\, z \not\Rightarrow x \,\Psi\, z$	



- ***BI*** bunches need explicit transformations

$$A; (B, C) \equiv (A; B), (A; C)$$

- QuB sharing graphs internalize the transformation



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: λ-encoding standard structures

- Multiplicative Product (Separating Pair)

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

- Additive Product (Sharing Pair)

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

- Sums

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

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

QuB: Extension

Towards programmer friendly

- Define custom types

$$
\begin{aligned}
&\texttt{data Bool} \;\; = \texttt{True} \mid \texttt{False} \\
&\texttt{data List a} = \texttt{Nil} \mid \texttt{Cons a a} \\
&\texttt{data Tree a} = \texttt{Leaf} \mid \texttt{Node a a} \\
&\qquad\qquad\vdots
\end{aligned}
$$

QuB: Extension

Towards programmer friendly

- Define custom types

$$
\begin{aligned}
&\texttt{data Bool} \quad = \texttt{True} \mid \texttt{False}\\
&\texttt{data List a} = \texttt{Nil} \mid \texttt{Cons a a}\\
&\texttt{data Tree a} = \texttt{Leaf} \mid \texttt{Node a a}\\
&\qquad\qquad\vdots
\end{aligned}
$$

- Type classes, functional dependencies

$$
\begin{aligned}
&\texttt{class Monad m a where}\\
&\qquad \texttt{return} \quad :: \ \texttt{a} \to \texttt{m a}\\
&\qquad (\gg=) :: \texttt{a} \to (\texttt{a} \to \texttt{m b}) \to \texttt{m b}
\end{aligned}
$$

$$
\begin{aligned}
&\texttt{class Collection e co} \mid \texttt{co} \to \texttt{e where}\\
&\qquad \texttt{empty} :: \ \texttt{co}\\
&\qquad \texttt{insert} :: \ \texttt{e} \to \texttt{co} \to \texttt{co}\\
&\qquad \texttt{member} :: \ \texttt{e} \to \texttt{co} \to \texttt{Bool}\\
&\qquad\qquad \vdots
\end{aligned}
$$

More Bootstrapping: Qualified Types and Kinds

- Incorporate predicates into type language for finer grained polymorphism

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

- Incorporate kinds, build hierarchy over types and generalize types to type constructors[8]

$$\text{Kinds} \quad \kappa ::= \star \mid \kappa' \to \kappa$$

$$\text{Types} \quad \tau^{\kappa}, \phi^{\kappa} ::= t^{\kappa} \mid T^{\kappa} \mid \tau^{\kappa' \to \kappa} \tau^{\kappa'}$$

$$\text{Type Constructors} \quad T^{\kappa} \in \mathcal{T}^{\kappa} \quad \text{where} \quad T^{\kappa} \subseteq \ldots$$

[8] Henk Barendregt. 'Introduction to generalized type systems'. In: *Journal of Functional Programming* 1.2 (1991), 125–154, Mark P. Jones. 'A System of Constructor Classes: Overloading and Implicit Higher-order Polymorphism'. In: *Proceedings of the Conference on Functional Programming Languages and Computer Architecture*. FPCA '93. New York, NY, USA: ACM, 1993, pp. 52–61

$$
\begin{array}{rrl}
\text{Types} & \tau, \upsilon, \phi ::= & t \mid \iota \mid \tau \to \tau \\
& \text{where} & \to \in \{ \overset{!}{\rightarrow\!\!\!\ast}, \rightarrow\!\!\!\ast, \overset{!}{\twoheadrightarrow}, \twoheadrightarrow \} \\
\text{Predicates} & \pi, \omega ::= & \mathtt{Un}\ \tau \mid \mathtt{ShFun}\ \phi \mid \mathtt{SeFun}\ \phi
\end{array}
$$

- $\mathtt{SeFun}\ \phi$: ϕ is a function that is separate from its argument
- $\mathtt{ShFun}\ \phi$: ϕ is a function that is in sharing with its argument
- $\mathtt{Un}\ \tau$: τ does not have resources or they can be copied/dropped easily[9]

[9] 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

$$
\begin{array}{lrl}
\text{Types} & \tau, \upsilon, \phi ::= & t \mid \iota \mid \tau \to \tau \\
& \text{where} & \to \in \{\overset{!}{-\!\!*}, -\!\!*, \overset{!}{\twoheadrightarrow}, \twoheadrightarrow\} \\
\text{Predicates} & \pi, \omega ::= & \mathtt{Un}\ \tau \mid \mathtt{ShFun}\ \phi \mid \mathtt{SeFun}\ \phi
\end{array}
$$

- $-\!\!*$: 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 $\twoheadrightarrow$

QuB: Datatypes with Sharing and Separation

Sharing Pair

```
data ShPair a b = ShP a; b     {- ; for sharing -}

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

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

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

QuB: Datatypes with Sharing and Separation

Separating Pair

```
data SePair a b = SeP a, b    {- , for separation -}

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

snd :: SePair a b -* b
snd (SeP a b) = b             {- Fails typecheck -}

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

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

File Handling API in QuB

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

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

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

$$\begin{aligned}\texttt{writeLine} :: \texttt{FileHandle} &\multimap \texttt{String}\\ &\multimap \texttt{IO FileHandle}\end{aligned}$$

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

Filehandles Revisited

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

$\rightsquigarrow$ $\rightsquigarrow$ $\rightsquigarrow$

```
(>>=) (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)
```

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

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

- 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

```
catch :: IOF a ⊸ (Exception ⊸ IO a) ↠ IO a
```

- Avoids memory leak

Contributions revisited

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

Future Work

- Type inference algorithm $\mathcal{M}$ is incomplete

  Terms can have two types

  - $\{\mathtt{Un}\ A\} \mid \varnothing \vdash \lambda^{-\!*} f.\lambda^{-\!*} x.fxx : (A \mathrel{-\!\!*} A \mathrel{-\!\!*} B) \mathrel{-\!\!*} A \mathrel{-\!\!*} B$
  - $\varnothing \mid \varnothing \vdash \lambda^{-\!*} f.\lambda^{-\!*} x.fxx : (A \mathrel{-\!\!*} A \twoheadrightarrow B) \mathrel{-\!\!*} A \twoheadrightarrow B$

- Current semantics: call-by-value assumed

  Formalize resource correctness

Thank You!

Q & A