CLASE

Cursor Library for A Structured Editor

“Tool Demo”

Tristan Allwood
(tora@doc.ic.ac.uk)

Susan Eisenbach
(s.eisenbach@imperial.ac.uk)

Zip! Photo from
http://www.flickr.com/photos/sarmax/109561164/

Polite Notice

This talk will feature code snippets!

Code a user has to write	Code that is in the CLASE library	Code that can be autogenerated with T.H. scripts
"Blue User"	"Green Library"	"Generated Orange"

http://www.flickr.com/photos/alkalinezoo/2374201026/

http://www.flickr.com/photos/cambridgelib/2343211287/

http://www.flickr.com/photos/webel/76665500/

A Sample Language

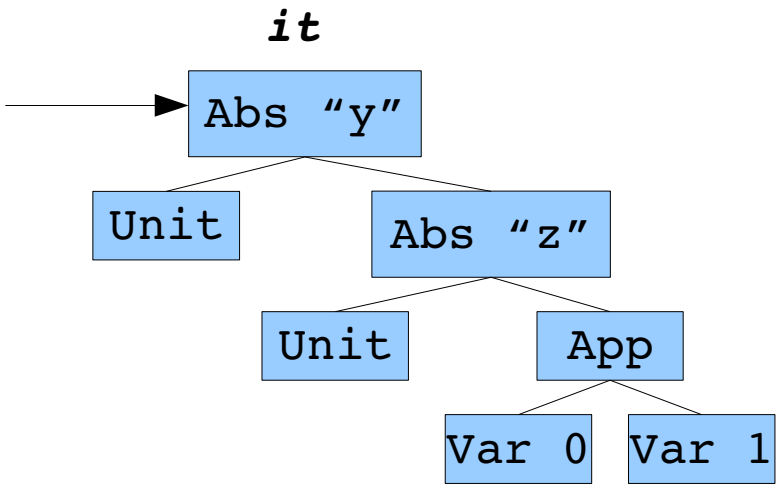
```
module Lam.Lam where

data Lam
  = Lam Exp

data Exp
  = Abs String Type Exp
  | App Exp Exp
  | Var Integer
  | NoExp

data Type
  = Unit
  | Arr Type Type
  | NoType
```

A CLASE Cursor

```
(λ x : ? . x) >(λ y : τ . λ z : τ . (z y))<
```

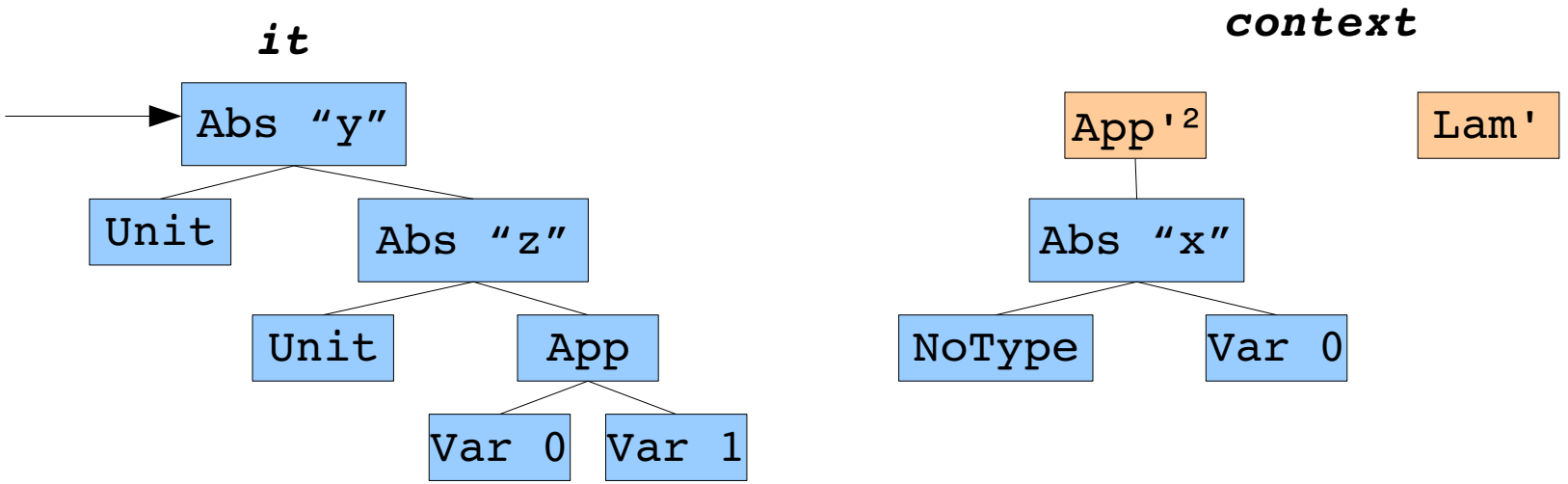
A CLASE Cursor

(λ x : ? . x) >(λ y : τ . λ z : τ . (z y))<



A CLASE Cursor

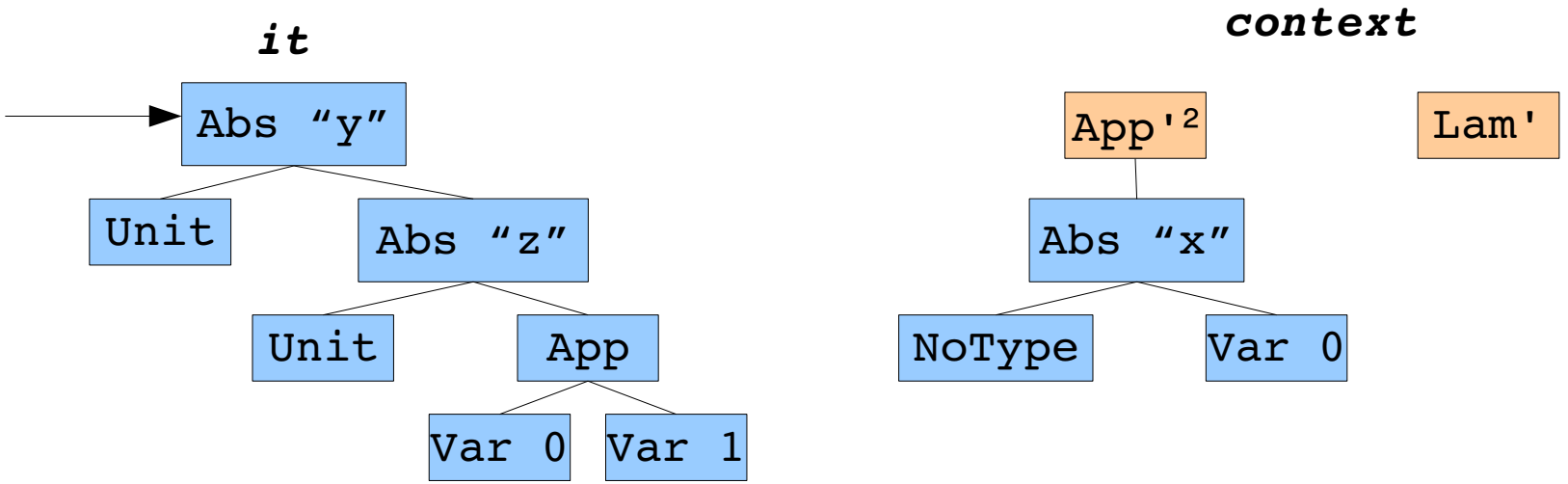
(λ x : ? . x) >(λ y : τ . λ z : τ . (z y))<



A CLASE Cursor

```
data Cursor l x a = (Reify l a) => Cursor {
    it :: a,
    ctx :: Path l (Context l) a l,
    log :: Route l a x
  }
```

(λ x : ? . x) >(λ y : τ . λ z : τ . (z y))<



Generating Boilerplate

```
{-# LANGUAGE TemplateHaskell #-}
module Lam.Gen where
import Lam.Lam
import Data.Cursor.CLASE.Gen.Adapters
import Data.Cursor.CLASE.Gen.Language
import Data.Cursor.CLASE.Gen.Persistence

$(languageGen ["Lam", "Language"] ''Lam
              [''Lam, ''Exp, ''Type])

$(adapterGen  ["Lam", "Adapters"] ''Lam
              [''Lam, ''Exp, ''Type] "Lam.Language")

$(persistenceGen ["Lam", "Persistence"] ''Lam
                 [''Lam, ''Exp, ''Type] "Lam.Language")

main :: IO ()
main = return ()
```

Generating Boilerplate

```
{-# LANGUAGE TemplateHaskell #-}
module Lam.Gen where
import Lam.Lam
import Data.Cursor.CLASE.Gen.Adapters
import Data.Cursor.CLASE.Gen.Language
import Data.Cursor.CLASE.Gen.Persistence

$(languageGen ["Lam", "Language"] ''Lam
              [''Lam, ''Exp, ''Type])

$(adapterGen  ["Lam", "Adapters"] ''Lam
              [''Lam, ''Exp, ''Type] "Lam.Language")

$(persistenceGen ["Lam", "Persistence"] ''Lam
                 [''Lam, ''Exp, ''Type] "Lam.Language")

main :: IO ()
main = return ()
```

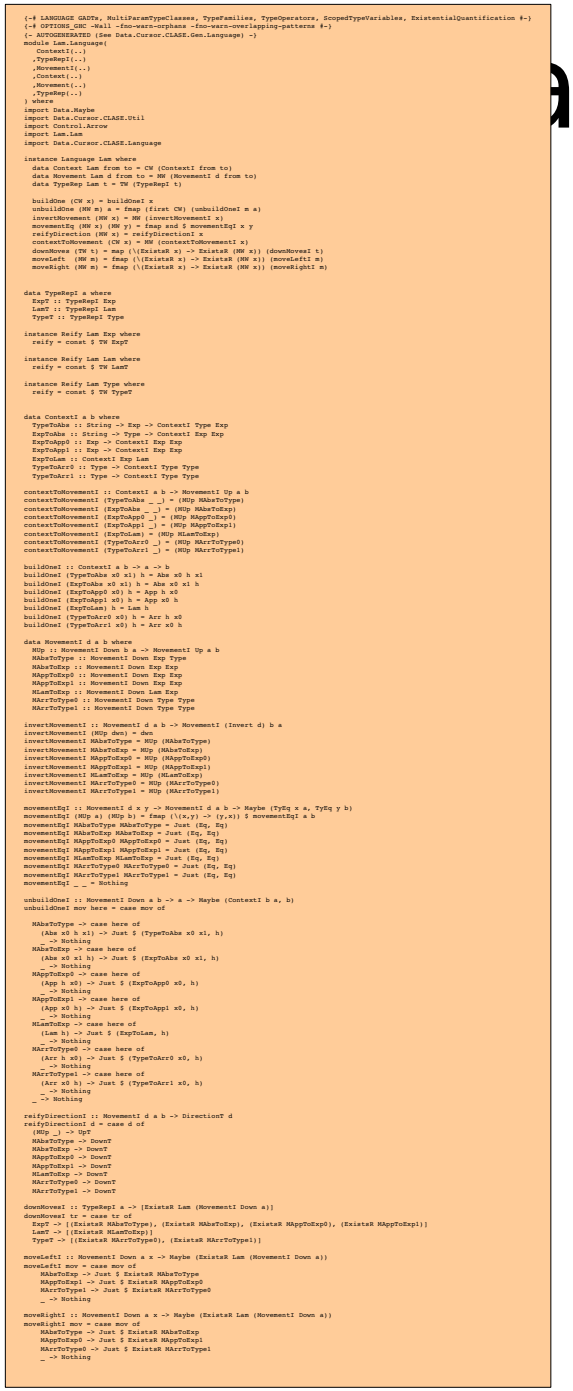
Generating Boilerplate

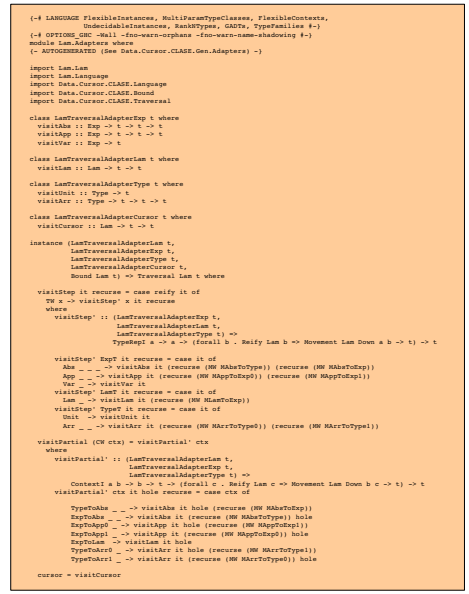
```
{-# LANGUAGE TemplateHaskell #-}
module Lam.Gen where
import Lam.Lam
import Data.Cursor.CLASE.Gen.Adapters
import Data.Cursor.CLASE.Gen.Language
import Data.Cursor.CLASE.Gen.Persistence

$(languageGen ["Lam", "Language"] ''Lam
              [''Lam, ''Exp, ''Type])

$(adapterGen  ["Lam", "Adapters"] ''Lam
              [''Lam, ''Exp, ''Type] "Lam.Language")

$(persistenceGen ["Lam", "Persistence"] ''Lam
                 [''Lam, ''Exp, ''Type] "Lam.Language")

main :: IO ()
main = return ()
```

Generating Boilerplate

```
{-# LANGUAGE TemplateHaskell #-}
module Lam.Gen where
import Lam.Lam
import Data.Cursor.CLASE.Gen.Adapters
import Data.Cursor.CLASE.Gen.Language
import Data.Cursor.CLASE.Gen.Persistence

$(languageGen ["Lam", "Language"] ''Lam
              [''Lam, ''Exp, ''Type])

$(adapterGen  ["Lam", "Adapters"] ''Lam
              [''Lam, ''Exp, ''Type] "Lam.Language")

$(persistenceGen ["Lam", "Persistence"] ''Lam
                 [''Lam, ''Exp, ''Type] "Lam.Language")

main :: IO ()
main = return ()
```

ating Boilerplate





Tristan Allwood, Susan Eisenbach

Rendering...

```
class LamTraversalAdapterExp t where
  visitAbs :: Exp → t → t → t
  visitApp :: Exp → t → t → t
  visitVar :: Exp → t

class LamTraversalAdapterLam t where
  visitLam :: Lam → t → t

class LamTraversalAdapterType t where
  visitUnit :: Type → t
  visitArr :: Type → t → t → t

class LamTraversalAdapterCursor t where
  visitCursor :: Lam → t → t
```

Rendering...

```
class (Bound l t) ⇒ Traversal l t where
   ...

completeTraversal :: (Traversal l t) ⇒ Cursor l x a → t
```

Rendering...

```
class (Bound l t) ⇒ Traversal l t where
   ...

completeTraversal :: (Traversal l t) ⇒ Cursor l x a → t
```

```
instance (LamTraversalAdapterLam t,
          LamTraversalAdapterExp t,
          LamTraversalAdapterType t,
          LamTraversalAdapterCursor t,
          Bound Lam t) ⇒ Traversal Lam t where
   ...
```

Rendering...

```
instance LamTraversalAdapterLam (LRM ()) where
  visitLam _ hole = hole

instance LamTraversalAdapterExp (LRM ()) where
  visitAbs (Abs name _ _) ty exp
    = out ("λ " ++ name ++ "::") >> ty >> out " . " >> exp
  visitApp _ l r
    = out "(" >> l >> out "∘" >> r >> out ")"
  visitVar (Var i)
    = (out . fromMaybe "Variable free!" =<< lookupBinding i) >>
      (out . subscript $ i)
  visitNoExp _ = out "?"

instance LamTraversalAdapterType (LRM ()) where
  visitUnit _ = out "τ"
  visitArr _ lhs rhs = out "(" >> lhs >> out " → " >> rhs >> out ")"
  visitNoType _ = out "?"

instance LamTraversalAdapterCursor (LRM ()) where
  visitCursor _ child = out "⟦" >> child >> out "⟧"
```

Rendering...

```
instance LamTraversalAdapterLam (LRM ()) where
  visitLam _ hole = hole

instance LamTraversalAdapterExp (LRM ()) where
  visitAbs (Abs name _ _) ty exp
    = out ("λ " ++ name ++ "::") >> ty >> out " . " >> exp
  visitApp _ l r
    = out "(" >> l >> out "∘" >> r >> out ")"
  visitVar (Var i)
    = (out . fromMaybe "Variable free!" =<< lookupBinding i) >>
      (out . subscript $ i)
  visitNoExp _ = out "?"

instance LamTraversalAdapterType (LRM ()) where
  visitUnit _ = out "τ"
  visitArr _ lhs rhs = out "(" >> lhs >> out " → " >> rhs >> out ")"
  visitNoType _ = out "?"

instance LamTraversalAdapterCursor (LRM ()) where
  visitCursor _ child = out "⟦" >> child >> out "⟧"
```

UI Edited Highlights - State

```
data CursorHolder where
  CH :: Cursor Lam a a → CursorHolder
```

```
data GuiState
  = GS { cursorBuffer :: TextBuffer
       , variableNameEntry :: Entry
       , variableNameDialog :: Dialog
       , whichVariableModel :: New.ListStore (Integer,
                                              String)
       , whichVariableDialog :: Dialog
       , cursorH :: IORef CursorHolder
       }
```

UI Edited Highlights: Movement

```
cursorKeyPress :: GuiState → Event → IO Bool
cursorKeyPress gs (Key { eventKeyChar  = Just char,
                         eventModifier = modifiers })
 | char == 'h'                 = moveCursor [gml, gmu]
 | char == 'j'                 = moveCursor [gmd, gmr]
 | char == 'k'                 = moveCursor [gmu, gml]
 | char == 'l'                 = moveCursor [gmr, gmd]
 ...
 where
   ref = cursorH gs
   gmd = fmap (\(CWM c _) → ExistsR c).genericMoveDown
   gmu = fmap (\(CWM c _) → ExistsR c).genericMoveUp
   gml = genericMoveLeft
   gmr = genericMoveRight
```

UI Edited Highlights: Movement

```
cursorKeyPress :: GuiState → Event → IO Bool
cursorKeyPress gs (Key { eventKeyChar  = Just char,
                         eventModifier = modifiers })
  | char == 'h'                 = moveCursor [gml, gmu]
  | char == 'j'                 = moveCursor [gmd, gmr]
  | char == 'k'                 = moveCursor [gmu, gml]
  | char == 'l'                 = moveCursor [gmr, gmd]
  ...
  where
    ref = cursorH gs
    gmd = fmap (\(CWM c _) → ExistsR c).genericMoveDown
    gmu = fmap (\(CWM c _) → ExistsR c).genericMoveUp
    gml = genericMoveLeft
    gmr = genericMoveRight
```

UI Edited Highlights: Movement

```
cursorKeyPress :: GuiState → Event → IO Bool
cursorKeyPress gs (Key { eventKeyChar  = Just char,
                         eventModifier = modifiers })
  | char == 'h'                 = moveCursor [gml, gmu]
  | char == 'j'                 = moveCursor [gmd, gmr]
  | char == 'k'                 = moveCursor [gmu, gml]
  | char == 'l'                 = moveCursor [gmr, gmd]
  ...
  where
    ref = cursorH gs
    gmd = fmap (\(CWM c _) → ExistsR c).genericMoveDown
    gmu = fmap (\(CWM c _) → ExistsR c).genericMoveUp
    gml = genericMoveLeft
    gmr = genericMoveRight
```

UI Edited Highlights: Movement

```
moveCursor :: (∀x a . [Cursor Lam x a →
   Maybe (ExistsR Lam (Cursor Lam x))]) → IO Bool
moveCursor movs = do
   CH (theCursor @ Cursor {}) ← readIORef ref

   maybe (return True)
         (\(ExistsR cursor') → do
           writeIORef ref $ CH (resetLog cursor')
           refreshAll gs
           return True
         )
         (msum $ map ($ theCursor) movs)
```

UI Edited Highlights: Movement

```
moveCursor :: (∀x a . [Cursor Lam x a →
   Maybe (ExistsR Lam (Cursor Lam x))]) → IO Bool
moveCursor movs = do
   CH (theCursor @ Cursor {}) ← readIORef ref

   maybe (return True)
         (\(ExistsR cursor') → do
           writeIORef ref $ CH (resetLog cursor')
           refreshAll gs
           return True
         )
         (msum $ map ($ theCursor) movs)
```

UI Edited Highlights: Movement

```
moveCursor :: (∀x a . [Cursor Lam x a →
   Maybe (ExistsR Lam (Cursor Lam x))]) → IO Bool
moveCursor movs = do
   CH (theCursor @ Cursor {}) ← readIORef ref

   maybe (return True)
         (\(ExistsR cursor') → do
           writeIORef ref $ CH (resetLog cursor')
           refreshAll gs
           return True
         )
         (msum $ map ($ theCursor) movs)
```

UI Edited Highlights: Movement

```
moveCursor :: (∀x a . [Cursor Lam x a →
   Maybe (ExistsR Lam (Cursor Lam x))]) → IO Bool
moveCursor movs = do
   CH (theCursor @ Cursor {}) ← readIORef ref

   maybe (return True)
         (\(ExistsR cursor') → do
           writeIORef ref $ CH (resetLog cursor')
           refreshAll gs
           return True
         )
         (msum $ map ($ theCursor) movs)
```

UI Edited Highlights: Movement

```
moveCursor :: (∀x a . [Cursor Lam x a →
   Maybe (ExistsR Lam (Cursor Lam x))]) → IO Bool
moveCursor movs = do
   CH (theCursor @ Cursor {}) ← readIORef ref

   maybe (return True)
         (\(ExistsR cursor') → do
           writeIORef ref $ CH (resetLog cursor')
           refreshAll gs
           return True
         )
         (msum $ map ($ theCursor) movs)
```

UI Edited Highlights: Rendering

```
refreshAll :: GuiState → IO ()
refreshAll gs = do
  CH cursor@Cursor {} ← readIORef (cursorH gs)
  let cursorText = render cursor
  (cursorBuffer gs) `textBufferSetText` cursorText
```

UI Edited Highlights: Rendering

```
refreshAll :: GuiState → IO ()
refreshAll gs = do
  CH cursor@Cursor {} ← readIORef (cursorH gs)
  let cursorText = render cursor
  (cursorBuffer gs) `textBufferSetText` cursorText
```

UI Edited Highlights: Rendering

```
refreshAll :: GuiState → IO ()
refreshAll gs = do
  CH cursor@Cursor {} ← readIORef (cursorH gs)
  let cursorText = render cursor
  (cursorBuffer gs) `textBufferSetText` cursorText
```

UI Edited Highlights: Rendering

```
refreshAll :: GuiState → IO ()
refreshAll gs = do
  CH cursor@Cursor {} ← readIORef (cursorH gs)
  let cursorText = render cursor
  (cursorBuffer gs) `textBufferSetText` cursorText
```

UI Edited Highlights: Editing

```
insertAppOrArr :: Bool → TypeRepI a → a → a
insertAppOrArr True   ExpT e        = App NoExp e
insertAppOrArr False  ExpT e        = App e NoExp
insertAppOrArr True   LamT (Lam e) = Lam (App NoExp e)
insertAppOrArr False  LamT (Lam e) = Lam (App e NoExp)
insertAppOrArr True   TypeT t       = Arr NoType t
insertAppOrArr False  TypeT t       = Arr t NoType
```

UI Edited Highlights: Editing

```
insertAppOrArr :: Bool → TypeRepI a → a → a
insertAppOrArr True   ExpT e        = App NoExp e
insertAppOrArr False  ExpT e        = App e NoExp
insertAppOrArr True   LamT (Lam e) = Lam (App NoExp e)
insertAppOrArr False  LamT (Lam e) = Lam (App e NoExp)
insertAppOrArr True   TypeT t       = Arr NoType t
insertAppOrArr False  TypeT t       = Arr t NoType
```

UI Edited Highlights: Editing

```
insertAppOrArr :: Bool → TypeRepI a → a → a
insertAppOrArr True   ExpT e          = App NoExp e
insertAppOrArr False  ExpT e          = App e NoExp
insertAppOrArr True   LamT (Lam e) = Lam (App NoExp e)
insertAppOrArr False  LamT (Lam e) = Lam (App e NoExp)
insertAppOrArr True   TypeT t         = Arr NoType t
insertAppOrArr False  TypeT t         = Arr t NoType
```

```
data TypeRepI a where
  ExpT :: TypeRepI Exp
  LamT :: TypeRepI Lam
  TypeT :: TypeRepI Type
```

UI Edited Highlights: Editing

```
insertAppOrArr :: Bool → TypeRepI a → a → a
insertAppOrArr True   ExpT e          = App NoExp e
insertAppOrArr False  ExpT e          = App e NoExp
insertAppOrArr True   LamT (Lam e) = Lam (App NoExp e)
insertAppOrArr False  LamT (Lam e) = Lam (App e NoExp)
insertAppOrArr True   TypeT t         = Arr NoType t
insertAppOrArr False  TypeT t         = Arr t NoType
```

```
data TypeRepI a where
  ExpT :: TypeRepI Exp
  LamT :: TypeRepI Lam
  TypeT :: TypeRepI Type
```

App Demo

Other Features

- Abstraction of Binding
- Routes
- Bookmarks
- Persistence

Thank you for listening

For more see
www.zonetora.co.uk/clase

Binding...

Binding...

```
class (Language l) ⇒ Bound l t where
  bindingHook :: Context l from to → t → t

...
```

Binding...

```
class (Language l) ⇒ Bound l t where
  bindingHook :: Context l from to → t → t

...
```

```
instance Bound Lam (M a) where
  bindingHook (ExpToAbs str _) hole
    = addBinding str hole
  bindingHook _ hole = hole

...
```