

# Dependent Types in Haskell: Theory and Practice

Richard A. Eisenberg
University of Pennsylvania
eir@cis.upenn.edu

Dissertation Proposal Defense Thursday, June 4, 2015

## Off to emacs...

## Other applications

- Typed reflection / Cloud Haskell [1]
- Embedded domain-specific languages
- Inferred algebraic effects [2]
- Type-safe database access [3]
- Machine-checked algorithms [4, 5]

- [1]: https://ghc.haskell.org/trac/ghc/wiki/DistributedHaskell
- [2]: E. Brady. "Programming and reasoning with algebraic effects and dependent types". ICFP '13
- [3]: N. Oury and W. Swierstra. "The Power of Pi". ICFP '08
- [4]: T. Altenkirch, C. McBride, and J. McKinna. "Why Dependent Types Matter". http://www.cs.nott.ac.uk/~txa/publ/ydtm.pdf
- [5]: A. Dergunov. "Generalized Algebraic Data Types in Haskell". In The Monad.Reader #22, 2013.

#### Type inference:

```
length :: [a] -> Int
length [] = 0
length (_ : xs) = 1 + length xs
```



[1]: D. Vytiniotis, S. Peyton Jones, T. Schrijvers, and M. Sulzmann. "OutsideIn(X): Modular Type Inference with Local Assumptions". JFP #21, 2011.

### Embracing partiality:

```
data StepResult :: Expr '[] ty -> * where
 Stepped :: pi (e' :: Expr '[] ty)
          -> ('eval e ~ 'eval e')
          => StepResult e
 Value :: pi (v :: Val ty)
          -> ('eval e ~ v)
                                 partial
          => StepResult e
                               (or, hard to
                               prove total)
             And: *
```

Two dependent quantifiers:

forall and pi

erased

not erased

#### Haskell is "real world"

- Industrial-strength optimizing compiler
- Some adoption in industry for Getting Work Done
- Broad ecosystem & user base

# Dependent Haskell

## It's All About the Quantifiers

- A quantifier introduces a function argument type.
- Today's Haskell has 3: forall, ->, and =>
- Questions about quantifiers:
  - Is the quantifiee dependent?
  - Is the quantifiee relevant?
  - Is the quantifiee visible?

# Dependency

- A quantifiee is dependent if it can be used later in a type.
- forall is dependent. -> and => are non-dependent.

foo :: forall a. a -> a

#### Relevance

- A quantifiee is relevant if it can be used in a relevant context or matched against.
- Almost, but not quite, the opposite of erasable.
- forall is irrelevant. -> and => are relevant.

## Visibility

- A quantifiee is visible if its value must be supplied by the programmer.
- -> is visible. forall and => are invisible.

# Quantifiers, Today

Quantifier	Dep?	Relevant?	Visible?
forall.	Yes	No	unification
->	No	Yes	Yes
=>	No	Yes	solving

# Quantifiers, Tomorrow

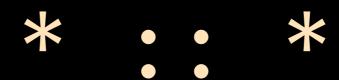
Quantifier	Dep?	Relevant?	Visible?
forall.	Yes	No	unification
forall->	Yes	No	Yes
pi.	Yes	Yes	unification
pi->	Yes	Yes	Yes
->	No	Yes	Yes
=>	No	Yes	solving

#### $\prod$

Pi-bound quantifiees are relevant and dependent.

```
data Vec :: * -> Nat -> * where
  Nil :: Vec a 'Zero
  (:>) :: a -> Vec a n -> Vec a ('Succ n)
replicate ::
 forall a. pi (n :: Nat) -> a -> Vec a n
                 = Nil
replicate Zero
replicate (Succ n') x
  = x :> replicate n' x
```

# Merging Types and Kinds



- All types are now kinds, too
- Promoting GADTs (previously unpromotable)
- Requires kind equalities [1]

#### Current Status

- Merged type/kind language complete
- Type inference for merged language complete, though not yet analyzed
- Implementation (without Π) working, nearly complete

# Remaining Challenges

- Type inference details, including proofs
  - ► Confident about inference with II due to experimentation with singletons [1]
- Implementation in GHC
- Expanding subset shared between types and terms
  - Possibly can promote lots of functions [2]
- Integration with roles [3]

<sup>[2]:</sup> R. A. Eisenberg and J. Stolarek. "Promoting functions to type families in Haskell". Haskell Symp. '14.

<sup>[3]:</sup> R. A. Eisenberg. "An overabundance of equality: Implementing kind equalities into Haskell". Submitted to Haskell Symp. '15

## Beyond Scope

- Promoting class constraints / dictionaries
- Termination checking [1]
- Pattern coverage checking [2]
- Laziness
- Higher-order unification

#### Timeline

2015

July Submit POPL paper

July Submit of L paper

Aug

Sep Merge w/ GHC master

Oct Develop new FC

Nov

Dec

Jan Finish Π

Feb Submit ICFP paper

2016

Mar

Apr Prove type inf.

May Finish writing

June Defend thesis



# Dependent Types in Haskell: Theory and Practice

Richard A. Eisenberg
University of Pennsylvania
eir@cis.upenn.edu

Dissertation Proposal Defense Thursday, June 4, 2015