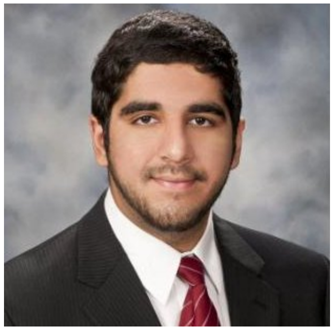
Visible Type Application

Stephanie Weirich

joint work with Richard Eisenberg and Hamidhasan Ahmed

November 2015

Penn



Visible Type Application in Haskell

Given a polymorphic function

```
id :: ∀a. a -> a
id x = x
```

we want to be able to *explicitly* control type instantiation.

```
id @Int  -- has type Int -> Int
```

How to control instantiation in Haskell (now)

- Type signatures:

```
(id :: Int -> Int)
```

More verbose

- Phantom type (aka Proxy):

```
data Proxy a = Proxy
```

Requires planning by library author

```
pid :: Proxy a -> a -> a
pid (Proxy :: Proxy Int)
```

Why? Type class ambiguity

- Suppose we had

```
normalize :: String -> String
normalize x = show ((read :: String -> Expr) x)
```

- What if we want to make it polymorphic?

```
normP :: ∀a. (Show a, Read a) => String -> String
normP x = show (read @a x)
```

- With VTA can use ambiguous types (and simplify code)

```
normP @Expr “1+2+3+4”
```

Why? Type Families

- Another ambiguous type:

```
type family F a where
   F Int = Bool

g :: F a -> F a
```

- GHC cannot determine `a` by unification, so this type is also ambiguous
- More realistic examples common with dependently-typed programming patterns

How hard could it be?

- Version 1: Undergraduate research project, Summer 2014

- Allow VTA at uses of variables:

```
f @Int @Bool
```

Gather all type arguments, lookup f's polymorphic type and instantiate it

Hindley-Milner Algorithm

$$\boxed{\Gamma \vdash_{\mathsf{c}} e : \tau}$$

$$\boxed{\frac{x{:}\forall\{\overline{a}\}.\,\tau \in \Gamma}{\Gamma \vdash_{\mathsf{c}} x : \tau[\overline{\tau}/\overline{a}]}\,\text{C\_VAR}}$$

$$\frac{\Gamma, x{:}\tau_1 \vdash_{\mathsf{c}} e : \tau_2}{\Gamma \vdash_{\mathsf{c}} \lambda x.\, e : \tau_1 \to \tau_2}\,\text{C\_ABS}$$

$$\frac{\Gamma \vdash_{\mathsf{c}} e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash_{\mathsf{c}} e_2 : \tau_1}{\Gamma \vdash_{\mathsf{c}} e_1\; e_2 : \tau_2}\,\text{C\_APP}$$

$$\frac{}{\Gamma \vdash_{\mathsf{c}} n : \mathit{Int}}\,\text{C\_INT}$$

$$\frac{\Gamma \vdash_{\mathsf{c}}^{gen} e_1 : \sigma \qquad \Gamma, x{:}\sigma \vdash_{\mathsf{c}} e_2 : \tau_2}{\Gamma \vdash_{\mathsf{c}} \textbf{let}\; x = e_1\; \textbf{in}\; e_2 : \tau_2}\,\text{C\_LET}$$

$$\boxed{\Gamma \vdash_{\mathsf{c}}^{gen} e : \sigma}$$

$$\frac{\overline{a} = \mathit{ftv}(\tau) \setminus \mathit{ftv}(\Gamma) \qquad \Gamma \vdash_{\mathsf{c}} e : \tau}{\Gamma \vdash_{\mathsf{c}}^{gen} e : \forall\{\overline{a}\}.\,\tau}\,\text{C\_GEN}$$

Did it work?

- Not too difficult to get something that works for many examples
- But how did we know we were doing it "right"?
- And, how should it interact with other features of GHC?

```
pair :: ∀a. a -> ∀b. b -> (a,b)
pair @Int @Bool 3 True
```

Properties?

- The HM type system has strong properties leading to a *predictable* type system
- Did they still hold after this extension?

NO!

Context Generalization

Theorem

If an HM program type checks using $x :: \sigma$ then it will still type check if we replace x's type with a *more general* type.

Counterexample

- Given `x :: ∀a b. (a,b) -> (a,b)`
- The program `x @Int @Bool` type checks
- If we update `x`'s type to `∀a. a -> a`, then type checking fails

Quiz

What is the *principal* type of

```
swap (x,y) = (y,x)
```

a) `∀a b. (a,b) -> (b,a)`
b) `∀a b. (b,a) -> (a,b)`
c) `∀a b c. (a,b) -> (b,a)`
d) all of the above

All of these types are equivalent. Worry: compiler updates can invalidate programs!

Two Part Solution

1) Differentiate in the type system between "generalized" and "specified" type variables

$\tau ::= \text{Int} \mid a \mid \tau \to \tau$	monotype
$\upsilon ::= \forall \bar{a}.\ \tau$	specified polytype *(from user annotations)*
$\sigma ::= \forall \{\bar{a}\}.\ \upsilon$	type scheme

2) Be a little more principled about type system design…

Hindley-Milner Type System

$\boxed{\Gamma \vdash_{hm} e : \sigma}$ Typing rules for System HM

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash_{hm} x : \sigma}\text{HM\_VAR} \qquad \frac{\Gamma, x{:}\tau_1 \vdash_{hm} e : \tau_2}{\Gamma \vdash_{hm} \lambda x.\, e : \tau_1 \to \tau_2}\text{HM\_ABS}$$

$$\frac{\Gamma \vdash_{hm} e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash_{hm} e_2 : \tau_1}{\Gamma \vdash_{hm} e_1\ e_2 : \tau_2}\text{HM\_APP} \qquad \frac{}{\Gamma \vdash_{hm} n : \mathit{Int}}\text{HM\_INT}$$

$$\frac{\Gamma \vdash_{hm} e_1 : \sigma_1 \qquad \Gamma, x{:}\sigma_1 \vdash_{hm} e_2 : \sigma_2}{\Gamma \vdash_{hm} \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \sigma_2}\text{HM\_LET}$$

$$\frac{\Gamma \vdash_{hm} e : \sigma \qquad a \notin ftv(\Gamma)}{\Gamma \vdash_{hm} e : \forall\{a\}.\,\sigma}\text{HM\_GEN} \qquad \frac{\Gamma \vdash_{hm} e : \sigma_1 \qquad \sigma_1 \leq_{hm} \sigma_2}{\Gamma \vdash_{hm} e : \sigma_2}\text{HM\_SUB}$$

Type Instantiation is just subsumption
to a less general (i.e. more specific) type.
Can happen ANYWHERE in the derivation.

HM + Type Application

$\boxed{\Gamma \vdash_{hm} e : \sigma}$ Typing rules for System HM

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash_{hm} x : \sigma}\text{HM\_VAR} \qquad \frac{\Gamma, x{:}\tau_1 \vdash_{hm} e : \tau_2}{\Gamma \vdash_{hm} \lambda x.\, e : \tau_1 \to \tau_2}\text{HM\_ABS}$$

$$\frac{\Gamma \vdash_{hm} e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash_{hm} e_2 : \tau_1}{\Gamma \vdash_{hm} e_1\ e_2 : \tau_2}\text{HM\_APP} \qquad \frac{}{\Gamma \vdash_{hm} n : \mathit{Int}}\text{HM\_INT}$$

$$\frac{\Gamma \vdash_{hm} e_1 : \sigma_1 \qquad \Gamma, x{:}\sigma_1 \vdash_{hm} e_2 : \sigma_2}{\Gamma \vdash_{hm} \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 : \sigma_2}\text{HM\_LET}$$

$$\frac{\Gamma \vdash_{hm} e : \sigma \qquad a \notin \mathit{ftv}(\Gamma)}{\Gamma \vdash_{hm} e : \forall\{a\}.\, \sigma}\text{HM\_GEN} \qquad \frac{\Gamma \vdash_{hm} e : \sigma_1 \qquad \boxed{\sigma_1 \leq_{hm} \sigma_2}}{\Gamma \vdash_{hm} e : \sigma_2}\text{HM\_SUB}$$

$$\frac{\begin{array}{l}\Gamma \vdash \tau \\ \Gamma \vdash_{hmv} e : \forall a.\, \upsilon\end{array}}{\Gamma \vdash_{hmv} e\,@\tau : \upsilon[\tau/a]}\text{HMV\_TAPP}$$

Subsumption for specified polymorphism?

- What is the "more general than" relation when specified types are involved?

Don't want

```
∀a b. (a,b) -> (b,a) ≰ ∀b a. (a,b) -> (a,b)
∀a b. (a,b) -> (b,a) ≰ ∀a. a -> a
∀a b. (a,b) -> (b,a) ≰ ∀b. (Int,b) -> (Int,b)
```

Ok

```
∀a b. (a,b) -> (b,a) ≤ ∀{a b}. (a,b) -> (a,b)
∀a b. (a,b) -> (b,a) ≤ (Int, Bool) -> (Int, Bool)
∀a b. (a,b) -> (b,a) ≤ ∀a. (a,Bool) -> (a,Bool)
```

Subsumption for specified polymorphism

$\boxed{\sigma_1 \leq_{\mathsf{hm}} \sigma_2}$ HM subsumption

$$\frac{\tau_1[\overline{\tau}/\overline{a}_1] = \tau_2 \qquad \overline{a}_2 \notin \mathit{ftv}(\forall\{\overline{a}_1\}.\,\tau_1)}{\forall\{\overline{a}_1\}.\,\tau_1 \leq_{\mathsf{hm}} \forall\{\overline{a}_2\}.\,\tau_2}\text{HM\_InstG}$$

$\boxed{\sigma_1 \leq_{\mathsf{hmv}} \sigma_2}$ HMV subsumption

$$\frac{\tau_1[\overline{\tau}/\overline{b}] = \tau_2}{\forall\overline{a},\overline{b}.\,\tau_1 \leq_{\mathsf{hmv}} \forall\overline{a}.\,\tau_2}\text{HMV\_InstS}$$

$$\frac{\upsilon_1[\overline{\tau}/\overline{a}_1] \leq_{\mathsf{hmv}} \upsilon_2 \qquad \overline{a}_2 \notin \mathit{ftv}(\forall\{\overline{a}_1\}.\,\upsilon_1)}{\forall\{\overline{a}_1\}.\,\upsilon_1 \leq_{\mathsf{hmv}} \forall\{\overline{a}_2\}.\,\upsilon_2}\text{HMV\_InstG}$$

What is true about this system?

Lemma 14 (Context Generalization for HMV). *If* $\Gamma \vdash_{\mathsf{hmv}} e : \sigma$ *and* $\Gamma' \leq_{\mathsf{hmv}} \Gamma$*, then* $\Gamma' \vdash_{\mathsf{hmv}} e : \sigma$*.*

Lemma 2 (Extra knowledge is harmless). *If* $\Gamma, x{:}\forall\{\overline{a}\}.\,\tau \vdash_{\mathsf{hmv}} e : \sigma$*, then* $\Gamma, x{:}\forall\overline{a}.\,\tau \vdash_{\mathsf{hmv}} e : \sigma$*.*

Theorem 3 (Principal types for HMV). *For all terms* e *well-typed in a context* Γ*, there exists a type scheme* σ_{p} *such that* $\Gamma \vdash_{\mathsf{hmv}} e : \sigma_{\mathrm{p}}$ *and, for all* σ *such that* $\Gamma \vdash_{\mathsf{hmv}} e : \sigma$*,* $\sigma_{\mathrm{p}} \leq_{\mathsf{hmv}} \sigma$*.*

Algorithm – System V

- How do we implement this specification?

$$\frac{\Gamma \vdash \tau \qquad \Gamma \vdash_{\mathsf{hmv}} e : \forall a.\, \upsilon}{\Gamma \vdash_{\mathsf{hmv}} e\,@\tau : \upsilon[\tau/a]}\ \text{HMV\_TApp}$$

- Key idea: *Lazy instantiation*

  If a term has a specified polymorphic type, don't instantiate it until absolutely necessary

- Syntax-directed system has three judgments:

$$\boxed{\Gamma \vdash_{\mathsf{v}} e : \tau} \qquad \boxed{\Gamma \vdash^{*}_{\mathsf{v}} e : \upsilon} \qquad \boxed{\Gamma \vdash^{gen}_{\mathsf{v}} e : \sigma}$$

Syntax-directed Algorithm

$\boxed{\Gamma \vdash_{\mathsf{v}} e : \tau}$ Monotype checking for System V

$$\frac{\Gamma, x{:}\tau_1 \vdash_{\mathsf{v}} e : \tau_2}{\Gamma \vdash_{\mathsf{v}} \lambda x.\, e : \tau_1 \to \tau_2}\text{V\_ABS} \qquad \frac{\Gamma \vdash_{\mathsf{v}} e_1 : \tau_1 \to \tau_2 \qquad \Gamma \vdash_{\mathsf{v}} e_2 : \tau_1}{\Gamma \vdash_{\mathsf{v}} e_1\ e_2 : \tau_2}\text{V\_APP}$$

$$\frac{}{\Gamma \vdash_{\mathsf{v}} n : \textit{\textbf{Int}}}\text{V\_INT} \qquad \frac{\begin{array}{l}\Gamma \vdash^{*}_{\mathsf{v}} e : \forall \overline{a}.\, \tau \\ \text{no other rule matches}\end{array}}{\Gamma \vdash_{\mathsf{v}} e : \tau[\overline{\tau}/\overline{a}]}\text{V\_INSTS}$$

$\boxed{\Gamma \vdash^{*}_{\mathsf{v}} e : \upsilon}$ Specified polytype checking for System V

$$\frac{x{:}\forall\{\overline{a}\}.\, \upsilon \in \Gamma}{\Gamma \vdash^{*}_{\mathsf{v}} x : \upsilon[\overline{\tau}/\overline{a}]}\text{V\_VAR} \qquad \frac{\Gamma \vdash^{gen}_{\mathsf{v}} e_1 : \sigma_1 \qquad \Gamma, x{:}\sigma_1 \vdash^{*}_{\mathsf{v}} e_2 : \upsilon_2}{\Gamma \vdash^{*}_{\mathsf{v}} \textbf{let}\ x = e_1\ \textbf{in}\ e_2 : \upsilon_2}\text{V\_LET}$$

$$\frac{\begin{array}{l}\Gamma \vdash \tau \\ \Gamma \vdash^{*}_{\mathsf{v}} e : \forall a.\, \upsilon\end{array}}{\Gamma \vdash^{*}_{\mathsf{v}} e\ @\tau : \upsilon[\tau/a]}\text{V\_TAPP} \qquad \frac{\begin{array}{l}\Gamma \vdash_{\mathsf{v}} e : \tau \\ \text{no other rule matches}\end{array}}{\Gamma \vdash^{*}_{\mathsf{v}} e : \tau}\text{V\_MONO}$$

Payoff

- Easy extension to GHC's bidirectional, higher-rank system

```
runST :: (forall s. ST s a) -> a
pair :: ∀a. a -> ∀b. b -> (a,b)
pair @Int 3 @Bool True
```

- Distinction between specified/generalized types makes sense there too:

  All "higher-rank" types must be specified

- New *declarative* higher-rank type system

$\boxed{\Gamma \vdash_{\mathsf{b}} e \Rightarrow \sigma}$ Synthesis rules for System B

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash_{\mathsf{b}} x \Rightarrow \sigma}\text{B\_VAR} \qquad \frac{\Gamma, x{:}\tau \vdash_{\mathsf{b}} e \Rightarrow \upsilon}{\Gamma \vdash_{\mathsf{b}} \lambda x.\, e \Rightarrow \tau \to \upsilon}\text{B\_ABS}$$

$$\frac{\Gamma \vdash_{\mathsf{b}} e_1 \Rightarrow \upsilon_1 \to \upsilon_2 \qquad \Gamma \vdash_{\mathsf{b}} e_2 \Leftarrow \upsilon_1}{\Gamma \vdash_{\mathsf{b}} e_1\ e_2 \Rightarrow \upsilon_2}\text{B\_APP} \qquad \frac{}{\Gamma \vdash_{\mathsf{b}} n \Rightarrow \mathit{Int}}\text{B\_INT}$$

$$\frac{\Gamma \vdash_{\mathsf{b}} e_1 \Rightarrow \sigma_1 \qquad \Gamma, x{:}\sigma_1 \vdash_{\mathsf{b}} e_2 \Rightarrow \sigma}{\Gamma \vdash_{\mathsf{b}} \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \Rightarrow \sigma}\text{B\_LET}$$

$$\frac{\Gamma \vdash_{\mathsf{b}} e \Rightarrow \sigma \qquad a \notin \mathit{ftv}(\Gamma)}{\Gamma \vdash_{\mathsf{b}} e \Rightarrow \forall \{a\}.\, \sigma}\text{B\_GEN} \qquad \frac{\Gamma \vdash_{\mathsf{b}} e \Rightarrow \sigma_1 \qquad \sigma_1 \leq_{\mathsf{b}} \sigma_2}{\Gamma \vdash_{\mathsf{b}} e \Rightarrow \sigma_2}\text{B\_SUB}$$

$$\frac{\begin{array}{l}\Gamma \vdash \tau \\ \Gamma \vdash_{\mathsf{b}} e \Rightarrow \forall a.\, \upsilon\end{array}}{\Gamma \vdash_{\mathsf{b}} e\ @\tau \Rightarrow \upsilon[\tau/a]}\text{B\_TAPP} \qquad \frac{\begin{array}{l}\Gamma \vdash \upsilon \qquad \upsilon = \forall \overline{a}, \overline{b}.\, \phi \\ \Gamma, \overline{a} \vdash_{\mathsf{b}} e \Leftarrow \phi\end{array}}{\Gamma \vdash_{\mathsf{b}} (\Lambda \overline{a}.e : \upsilon) \Rightarrow \upsilon}\text{B\_ANNOT}$$

$\boxed{\Gamma \vdash_{\mathsf{b}} e \Leftarrow \upsilon}$ Checking rules for System B

$$\frac{\Gamma, x{:}\upsilon_1 \vdash_{\mathsf{b}} e \Leftarrow \upsilon_2}{\Gamma \vdash_{\mathsf{b}} \lambda x.\, e \Leftarrow \upsilon_1 \to \upsilon_2}\text{B\_DABS} \qquad \frac{\begin{array}{l}\Gamma \vdash_{\mathsf{b}} e_1 \Rightarrow \sigma_1 \\ \Gamma, x{:}\sigma_1 \vdash_{\mathsf{b}} e_2 \Leftarrow \upsilon\end{array}}{\Gamma \vdash_{\mathsf{b}} \textbf{let}\ x = e_1\ \textbf{in}\ e_2 \Leftarrow \upsilon}\text{B\_DLET}$$

$$\frac{\Gamma \vdash_{\mathsf{b}} e \Leftarrow \upsilon \qquad a \notin \mathit{ftv}(\Gamma)}{\Gamma \vdash_{\mathsf{b}} e \Leftarrow \forall a.\, \upsilon}\text{B\_SKOL} \qquad \frac{\Gamma \vdash_{\mathsf{b}} e \Rightarrow \sigma_1 \qquad \sigma_1 \leq_{\mathsf{dsk}} \upsilon_2}{\Gamma \vdash_{\mathsf{b}} e \Leftarrow \upsilon_2}\text{B\_INFER}$$

Related Work

- Explicit form of type application found in many explicit/semi-explicit type systems
  - System F
  - Coq / Agda
- Lazy instantiation seems to be a new algorithm for the Hindley-Milner type system
- Bidirectional, higher-rank polymorphism
  - Peyton Jones, Vytiniotis, Weirich, and Shields. *Practical type inference for arbitrary-rank types*. JFP 2007.
  - Dunfield and Krishnaswami. *Complete and Easy Bidirectional Typechecking for Higher-Rank Polymorphism*. ICFP 2013.

Conclusion

- Type system additions should be compositional
- Even if something seems “easy” it is important to do it well; we should have thought about compositionality from the beginning
- VTA release planned soon, Richard is merging into HEAD