Hasochism

The pleasure and pain of dependently typed Haskell programming

Sam Lindley

Laboratory for foundations of computer science
The University of Edinburgh

Sam.Lindley@ed.ac.uk

23rd September 2013

(joint work with Conor McBride)

Promotion

$$\textbf{data}\ \mathsf{Nat} = \mathsf{Z} \mid \mathsf{S}\ \mathsf{Nat}\ \textbf{deriving}\ (\mathsf{Show}, \mathsf{Eq}, \mathsf{Ord})$$

$$\begin{array}{l}
\textbf{data}\ \mathsf{Vec} :: \mathsf{Nat} \to \star \to \star\ \textbf{where} \\
\quad \begin{array}{lll}
\mathsf{V0} & :: & \qquad\qquad\qquad\qquad\quad \mathsf{Vec}\ \mathsf{Z}\ x \\
(:>) & :: & x \to \mathsf{Vec}\ n\ x \to \mathsf{Vec}\ (\mathsf{S}\ n)\ x
\end{array}
\end{array}$$

$$\begin{array}{l}
\textbf{type family}\ (m :: \mathsf{Nat})\ {:}{+}\ (n :: \mathsf{Nat}) :: \mathsf{Nat} \\
\textbf{type instance}\ \mathsf{Z} \quad\ \ {:}{+}\ \ n\ \ = n \\
\textbf{type instance}\ \mathsf{S}\ m\ {:}{+}\ \ n\ \ = \mathsf{S}\ (m\ {:}{+}\ n)
\end{array}$$

$$\begin{array}{l}
\mathsf{vappend} :: \mathsf{Vec}\ m\ x \to \mathsf{Vec}\ n\ x \to \mathsf{Vec}\ (m\ {:}{+}\ n)\ x \\
\mathsf{vappend}\ \mathsf{V0} \qquad\qquad ys\ \ = ys \\
\mathsf{vappend}\ (x :> xs)\ \ ys\ \ = x :> \mathsf{vappend}\ xs\ ys
\end{array}$$

Singletons

Inverting vappend:

$$\text{vchop} :: \text{Vec}\ (m :+ n)\ x \to (\text{Vec}\ m\ x, \text{Vec}\ n\ x) \quad \text{-- } (\times)$$

We cannot write vchop as m is not available at run time.

Singleton GADT for naturals:

$$\begin{array}{l}
\textbf{data}\ \text{Natty} :: \text{Nat} \to \star\ \textbf{where} \\
\quad \text{Zy} :: \text{Natty Z} \\
\quad \text{Sy} :: \text{Natty}\ n \to \text{Natty}\ (\text{S}\ n)
\end{array}$$

If we pass in a Natty m, then we can compute over it at run time:

$$\begin{array}{l}
\text{vchop} :: \text{Natty}\ m \to \text{Vec}\ (m :+ n)\ x \to (\text{Vec}\ m\ x, \text{Vec}\ n\ x) \\
\text{vchop Zy} \qquad xs \qquad\quad = (\text{V0}, \qquad xs) \\
\text{vchop}\ (\text{Sy}\ m)\ (x :> xs) = (x :> ys, zs) \\
\quad \textbf{where}\ (ys, zs) = \text{vchop}\ m\ xs
\end{array}$$

Proxies

Compute the first component of vchop:

$$\begin{array}{l} \mathsf{vtake} :: \mathsf{Natty}\ m \to \mathsf{Vec}\ (m :+ n)\ x \to \mathsf{Vec}\ m\ x \quad \text{-- } (\times) \\ \mathsf{vtake}\ \mathsf{Zy}\qquad xs \qquad\quad = \mathsf{V0} \\ \mathsf{vtake}\ (\mathsf{Sy}\ m)\ (x :> xs) = x :> \mathsf{vtake}\ m\ xs \end{array}$$

Doesn't type check because GHC does not know how to instantiate n in the recursive call.

A proxy is an explicit representation of a type:

$$\begin{array}{l} \mathbf{data}\ \mathsf{Proxy} :: \kappa \to \star\ \mathbf{where} \\ \quad \mathsf{Proxy} :: \mathsf{Proxy}\ i \end{array}$$

This type checks:

$$\begin{array}{l} \mathsf{vtake} :: \mathsf{Natty}\ m \to \mathsf{Proxy}\ n \to \mathsf{Vec}\ (m :+ n)\ x \to \mathsf{Vec}\ m\ x \\ \mathsf{vtake}\ \mathsf{Zy}\qquad n\ xs \qquad\quad = \mathsf{V0} \\ \mathsf{vtake}\ (\mathsf{Sy}\ m)\ n\ (x :> xs) = x :> \mathsf{vtake}\ m\ n\ xs \end{array}$$

Generate a proxy from existing data:

$$\begin{array}{l} \mathsf{proxy} :: f\ \ i \to \mathsf{Proxy}\ i \\ \mathsf{proxy}\ \_ = \mathsf{Proxy} \end{array}$$

Implicit singletons

$$
\begin{array}{l}
\textbf{class}\ \mathsf{NATTY}\ (n :: \mathsf{Nat})\ \ \textbf{where} \\
\quad \mathsf{natty} :: \mathsf{Natty}\ n \\
\textbf{instance}\ \mathsf{NATTY}\ \mathsf{Z}\ \ \textbf{where} \\
\quad \mathsf{natty} = \mathsf{Zy} \\
\textbf{instance}\ \mathsf{NATTY}\ n \Rightarrow \mathsf{NATTY}\ (\mathsf{S}\ n)\ \ \textbf{where} \\
\quad \mathsf{natty} = \mathsf{Sy}\ \mathsf{natty}
\end{array}
$$

An implicit version of vtake that infers the required length from the result type:

$$
\begin{array}{l}
\mathsf{vtrunc} :: \mathsf{NATTY}\ m \Rightarrow \mathsf{Proxy}\ n \to \mathsf{Vec}\ (m :\!+\ n)\ x \to \mathsf{Vec}\ m\ x \\
\mathsf{vtrunc} = \mathsf{vtake}\ \mathsf{natty}
\end{array}
$$

$$
\begin{array}{l}
> \mathsf{vtrunc}\ \mathsf{Proxy}\ (1 :> 2 :> 3 :> 4 :> \mathsf{V0}) :: \mathsf{Vec}\ (\mathsf{S}\ (\mathsf{S}\ \mathsf{Z}))\ \mathsf{Int} \\
1 :> 2 :> \mathsf{V0}
\end{array}
$$

Four kinds of quantifier

	implicit	**explicit**
static	$\forall(n :: \mathsf{Nat}).$	$\forall(n :: \mathsf{Nat}).\mathsf{Proxy}\ n \rightarrow$
dynamic	$\forall n.\mathsf{NATTY}\ n \Rightarrow$	$\forall n.\mathsf{Natty}\ n \rightarrow$

Instances for indexed types

$$
\begin{array}{l}
\textbf{instance}\ \mathsf{NATTY}\ n \Rightarrow \mathsf{Applicative}\ (\mathsf{Vec}\ n)\ \ \textbf{where} \\
\quad \mathsf{pure} \quad\ \ = \mathsf{vcopies\ natty} \\
\quad (<\star>) = \mathsf{vapp}
\end{array}
$$

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{vcopies} :: \forall n\ x.\mathsf{Natty}\ n \to x \to \mathsf{Vec}\ n\ x} \\
\mathsf{vcopies} & \mathsf{Zy} & x = \mathsf{V0} \\
\mathsf{vcopies} & (\mathsf{Sy}\ n) & x = x :> \mathsf{vcopies}\ n\ x
\end{array}
$$

$$
\begin{array}{llll}
\multicolumn{4}{l}{\mathsf{vapp} :: \forall n\ s\ t.\mathsf{Vec}\ n\ (s \to t) \to \mathsf{Vec}\ n\ s \to \mathsf{Vec}\ n\ t} \\
\mathsf{vapp} & \mathsf{V0} & \mathsf{V0} & = \mathsf{V0} \\
\mathsf{vapp} & (f :> \mathit{fs}) & (s :> \mathit{ss}) & = f\ s :> \mathsf{vapp}\ \mathit{fs}\ \mathit{ss}
\end{array}
$$

Converting between implicit and explict singletons

Implicit to explicit:

$$\mathsf{natty} :: \mathsf{NATTY}\ n \Rightarrow \mathsf{Natty}\ n$$

Explicit to implicit:

$$\begin{array}{l}
\mathsf{natter} :: \mathsf{Natty}\ n \to (\mathsf{NATTY}\ n \Rightarrow t) \to t \\
\mathsf{natter}\ \mathsf{Zy} \qquad\ \ t = t \\
\mathsf{natter}\ (\mathsf{Sy}\ n)\ t = \mathsf{natter}\ n\ t
\end{array}$$

Proof objects

$$
\begin{array}{l}
\textbf{data}\ \mathsf{Cmp} :: \mathsf{Nat} \to \mathsf{Nat} \to \star\ \textbf{where} \\
\quad \mathsf{LTNat}\ :: \mathsf{Natty}\ z \to \mathsf{Cmp}\ m\ (m :\!+\ \mathsf{S}\ z) \\
\quad \mathsf{EQNat} :: \qquad\qquad\quad\ \mathsf{Cmp}\ m\ m \\
\quad \mathsf{GTNat} :: \mathsf{Natty}\ z \to \mathsf{Cmp}\ (n :\!+\ \mathsf{S}\ z)\ n
\end{array}
$$

$$
\begin{array}{l}
\mathsf{cmp} :: \mathsf{Natty}\ m \to \mathsf{Natty}\ n \to \mathsf{Cmp}\ m\ n \\
\mathsf{cmp}\ \mathsf{Zy}\qquad\quad\ \mathsf{Zy}\qquad\ = \mathsf{EQNat} \\
\mathsf{cmp}\ \mathsf{Zy}\qquad\quad\ (\mathsf{Sy}\ n) = \mathsf{LTNat}\ n \\
\mathsf{cmp}\ (\mathsf{Sy}\ m)\ \mathsf{Zy}\qquad\ = \mathsf{GTNat}\ m \\
\mathsf{cmp}\ (\mathsf{Sy}\ m)\ (\mathsf{Sy}\ n) = \textbf{case}\ \mathsf{cmp}\ m\ n\ \textbf{of} \\
\quad \mathsf{LTNat}\ z\ \to \mathsf{LTNat}\ z \\
\quad \mathsf{EQNat}\quad \to \mathsf{EQNat} \\
\quad \mathsf{GTNat}\ z \to \mathsf{GTNat}\ z
\end{array}
$$

The Procrustes function

Fit a vector of length m into a vector of length n by padding or trimming as necessary:

$$
\begin{array}{l}
\mathit{procrustes} :: a \rightarrow \mathrm{Natty}\ m \rightarrow \mathrm{Natty}\ n \rightarrow \mathrm{Vec}\ m\ a \rightarrow \mathrm{Vec}\ n\ a \\
\mathit{procrustes}\ p\ m\ n\ \mathit{xs} = \textbf{case}\ \mathrm{cmp}\ m\ n\ \textbf{of} \\
\quad \mathrm{LTNat}\ z \ \rightarrow \mathrm{vappend}\ \mathit{xs}\ (\mathrm{vcopies}\ (\mathrm{Sy}\ z)\ p) \\
\quad \mathrm{EQNat} \quad \rightarrow \mathit{xs} \\
\quad \mathrm{GTNat}\ z \rightarrow \mathrm{vtake}\ n\ (\mathrm{proxy}\ (\mathrm{Sy}\ z))\ \mathit{xs}
\end{array}
$$

Boxes

$$
\begin{array}{l}
\textbf{data}\ \mathrm{Box} :: ((\mathrm{Nat}, \mathrm{Nat}) \to \star) \to (\mathrm{Nat}, \mathrm{Nat}) \to \star\ \textbf{where} \\
\quad \begin{array}{lll}
\mathrm{Stuff} & :: & p\ wh \to \mathrm{Box}\ p\ wh \\
\mathrm{Clear} & :: & \mathrm{Box}\ p\ wh \\
\mathrm{Hor} & :: & \mathrm{Natty}\ w_1 \to \mathrm{Box}\ p\ '(w_1, h) \to \\
 & & \mathrm{Natty}\ w_2 \to \mathrm{Box}\ p\ '(w_2, h) \to \mathrm{Box}\ p\ '(w_1 :+ w_2, h) \\
\mathrm{Ver} & :: & \mathrm{Natty}\ h_1 \to \mathrm{Box}\ p\ '(w, h_1) \to \\
 & & \mathrm{Natty}\ h_2 \to \mathrm{Box}\ p\ '(w, h_2) \to \mathrm{Box}\ p\ '(w, h_1 :+ h_2)
\end{array}
\end{array}
$$

Boxes are monads in a category of indexed types

Morphisms:

$$\textbf{type}\ s :\to t = \forall i. s\ i \to t\ i$$

Monads over indexed types:

$$
\begin{array}{l}
\textbf{class}\ \mathsf{MonadIx}\ (m :: (\kappa \to \star) \to (\kappa \to \star))\ \ \textbf{where} \\
\quad \mathsf{returnIx}\ :: a :\to m\ a \\
\quad \mathsf{extendIx} :: (a :\to m\ b) \to (m\ a :\to m\ b)
\end{array}
$$

Box is a MonadIx:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\textbf{instance}\ \mathsf{MonadIx}\ \mathsf{Box}\ \ \textbf{where}} \\
\quad \mathsf{returnIx} & = \mathsf{Stuff} \\
\quad \mathsf{extendIx}\ f\ (\mathsf{Stuff}\ c) & = f\ c \\
\quad \mathsf{extendIx}\ f\ \mathsf{Clear} & = \mathsf{Clear} \\
\quad \mathsf{extendIx}\ f\ (\mathsf{Hor}\ w_1\ b_1\ w_2\ b_2) = & \\
\quad\quad \mathsf{Hor}\ w_1\ (\mathsf{extendIx}\ f\ b_1)\ w_2\ (\mathsf{extendIx}\ f\ b_2) & \\
\quad \mathsf{extendIx}\ f\ (\mathsf{Ver}\ h_1\ b_1\ h_2\ b_2)\ = & \\
\quad\quad \mathsf{Ver}\ h_1\ (\mathsf{extendIx}\ f\ b_1)\ h_2\ (\mathsf{extendIx}\ f\ b_2) &
\end{array}
$$

Two flavours of conjunction for indexed things

Separating conjunction:

$$
\begin{array}{l}
\textbf{data}\ (p :: \iota \to \star)\ {:}{*}{*}{:}\ (q :: \kappa \to \star) :: (\iota, \kappa) \to \star\ \textbf{where} \\
\quad ({:}\&\&{:}) :: p\ \iota \to q\ \kappa \to (p\ {:}{*}{*}{:}\ q)\ '(\iota, \kappa)
\end{array}
$$

Non-separating conjunction:

$$
\begin{array}{l}
\textbf{data}\ (p :: \kappa \to \star)\ {:}{*}{:}\ (q :: \kappa \to \star) :: \kappa \to \star\ \textbf{where} \\
\quad ({:}\&{:}) :: p\ \kappa \to q\ \kappa \to (p\ {:}{*}{:}\ q)\ k
\end{array}
$$

Juxtaposition

$$\begin{array}{l}\mathsf{juxH} :: \mathsf{Size}\ '(w_1, h_1) \to \mathsf{Size}\ '(w_2, h_2) \to \\ \quad \mathsf{Box}\ p\ '(w_1, h_1) \to \mathsf{Box}\ p\ '(w_2, h_2) \to \\ \qquad \mathsf{Box}\ p\ '(w_1 :+ w_2, \mathsf{Max}\ h_1\ h_2)\end{array}$$

$$\textbf{type}\ \mathsf{Size} = \mathsf{Natty}\ {:}{**}{:}\ \mathsf{Natty}$$

$$\begin{array}{l}\textbf{type family}\ \mathsf{Max}\ (m :: \mathsf{Nat})\ (n :: \mathsf{Nat}) :: \mathsf{Nat} \\ \textbf{type instance}\ \mathsf{Max}\ \mathsf{Z}\qquad\ \ n\qquad = n \\ \textbf{type instance}\ \mathsf{Max}\ (\mathsf{S}\ m)\ \mathsf{Z}\qquad = \mathsf{S}\ m \\ \textbf{type instance}\ \mathsf{Max}\ (\mathsf{S}\ m)\ (\mathsf{S}\ n) = \mathsf{S}\ (\mathsf{Max}\ m\ n)\end{array}$$

Juxtaposition (broken)

$$
\begin{array}{l}
\text{juxH}\ (w_1\ \text{:\&\&:}\ h_1)\ (w_2\ \text{:\&\&:}\ h_2)\ b_1\ b_2 = \\
\quad \textbf{case}\ \text{cmp}\ h_1\ h_2\ \textbf{of} \\
\qquad \begin{array}{lll}
\text{LTNat}\ n & \rightarrow \text{Hor}\ w_1\ (\text{Ver}\ h_1\ b_1\ (\text{Sy}\ n)\ \text{Clear})\ w_2\ b_2 & \text{-- }(\times) \\
\text{EQNat} & \rightarrow \text{Hor}\ w_1\ b_1\ w_2\ b_2 & \text{-- }(\times) \\
\text{GTNat}\ n & \rightarrow \text{Hor}\ w_1\ b_1\ w_2\ (\text{Ver}\ h_2\ b_2\ (\text{Sy}\ n)\ \text{Clear}) & \text{-- }(\times)
\end{array}
\end{array}
$$

Doesn't type check because GHC has no way of knowing that the height of the resulting box is the maximum of the heights of the component boxes.

Pain

$$
\begin{array}{l}
\mathsf{juxH}\ (w_1\ {:}\&\&{:}\ h_1)\ (w_2\ {:}\&\&{:}\ h_2)\ b_1\ b_2 = \\
\quad \textbf{case}\ \mathsf{cmp}\ h_1\ h_2\ \textbf{of} \\
\qquad \mathsf{LTNat}\ z \quad \to \mathsf{maxLT}\ h_1\ z\ \$\ \mathsf{Hor}\ w_1\ (\mathsf{Ver}\ h_1\ b_1\ (\mathsf{Sy}\ z)\ \mathsf{Clear})\ w_2\ b_2 \\
\qquad \mathsf{EQNat} \quad \to \mathsf{maxEQ}\ h_1\ \$\ \mathsf{Hor}\ w_1\ b_1\ w_2\ b_2 \\
\qquad \mathsf{GTNat}\ z \quad \to \mathsf{maxGT}\ h_2\ z\ \$\ \mathsf{Hor}\ w_1\ b_1\ w_2\ (\mathsf{Ver}\ h_2\ b_2\ (\mathsf{Sy}\ z)\ \mathsf{Clear})
\end{array}
$$

$$
\begin{array}{l}
\mathsf{maxLT} :: \forall m\ z\ t. \mathsf{Natty}\ m \to \mathsf{Natty}\ z \to \\
\qquad\qquad ((\mathsf{Max}\ m\ (m\ {:}{+}\ \mathsf{S}\ z) \sim (m\ {:}{+}\ \mathsf{S}\ z)) \Rightarrow t) \to t \\
\mathsf{maxLT}\ \mathsf{Zy}\ \quad z\ t \quad = t \\
\mathsf{maxLT}\ (\mathsf{Sy}\ m)\ z\ t \quad = \mathsf{maxLT}\ m\ z\ t
\end{array}
$$

$$
\begin{array}{l}
\mathsf{maxEQ} :: \forall m\ t. \mathsf{Natty}\ m \to ((\mathsf{Max}\ m\ m \sim m) \Rightarrow t) \to t \\
\mathsf{maxEQ}\ \mathsf{Zy}\ \quad t \quad = t \\
\mathsf{maxEQ}\ (\mathsf{Sy}\ m)\ t \quad = \mathsf{maxEQ}\ m\ t
\end{array}
$$

$$
\begin{array}{l}
\mathsf{maxGT} :: \forall n\ z\ t. \mathsf{Natty}\ n \to \mathsf{Natty}\ z \to \\
\qquad\qquad ((\mathsf{Max}\ (n\ {:}{+}\ \mathsf{S}\ z)\ n \sim (n\ {:}{+}\ \mathsf{S}\ z)) \Rightarrow t) \to t \\
\mathsf{maxGT}\ \mathsf{Zy}\ \quad z\ t \quad = t \\
\mathsf{maxGT}\ (\mathsf{Sy}\ n)\ z\ t \quad = \mathsf{maxGT}\ n\ z\ t
\end{array}
$$

Type equations for free

$$\begin{aligned}
&\textbf{data}\ \mathsf{Cmp} :: \mathsf{Nat} \to \mathsf{Nat} \to \star\ \textbf{where}\\
&\quad \mathsf{LTNat} :: \mathsf{Natty}\ z \to \mathsf{Cmp}\ m\ (m :+ \mathsf{S}\ z)\\
&\quad \mathsf{EQNat} :: \qquad\qquad\quad \mathsf{Cmp}\ m\ m\\
&\quad \mathsf{GTNat} :: \mathsf{Natty}\ z \to \mathsf{Cmp}\ (n :+ \mathsf{S}\ z)\ n
\end{aligned}$$

Make GADT type equations explicit:

$$\begin{aligned}
&\textbf{data}\ \mathsf{Cmp} :: \mathsf{Nat} \to \mathsf{Nat} \to \star\ \textbf{where}\\
&\quad \mathsf{LTNat} :: ((m :+ \mathsf{S}\ z) \sim n) \Rightarrow \mathsf{Natty}\ z \to \mathsf{Cmp}\ m\ n\\
&\quad \mathsf{EQNat} :: (m \sim n) \Rightarrow \mathsf{Cmp}\ m\ n\\
&\quad \mathsf{GTNat} :: (m \sim (n :+ \mathsf{S}\ z)) \Rightarrow \mathsf{Natty}\ z \to \mathsf{Cmp}\ m\ n
\end{aligned}$$

Add more type equations:

$$\begin{aligned}
&\textbf{data}\ \mathsf{Cmp} :: \mathsf{Nat} \to \mathsf{Nat} \to \star\ \textbf{where}\\
&\quad \mathsf{LTNat} :: ((m :+ \mathsf{S}\ z) \sim n, \mathsf{Max}\ m\ n \sim n) \Rightarrow \mathsf{Natty}\ z \to \mathsf{Cmp}\ m\ n\\
&\quad \mathsf{EQNat} :: (m \sim n, \mathsf{Max}\ m\ n \sim m) \Rightarrow \mathsf{Cmp}\ m\ n\\
&\quad \mathsf{GTNat} :: (m \sim (n :+ \mathsf{S}\ z), \mathsf{Max}\ m\ n \sim m) \Rightarrow \mathsf{Natty}\ z \to \mathsf{Cmp}\ m\ n
\end{aligned}$$

Pleasure

$$
\begin{aligned}
&\mathrm{juxH} :: \mathrm{Size}\ '(w_1, h_1) \to \mathrm{Size}\ '(w_2, h_2) \to \\
&\qquad\qquad \mathrm{Box}\ p\ '(w_1, h_1) \to \mathrm{Box}\ p\ '(w_2, h_2) \to \\
&\qquad\qquad\quad \mathrm{Box}\ p\ '(w_1 :\!+\ w_2, \mathrm{Max}\ h_1\ h_2) \\
&\mathrm{juxH}\ (w_1\ :\&\&:\ h_1)\ (w_2\ :\&\&:\ h_2)\ b_1\ b_2 = \\
&\quad \mathbf{case}\ \mathrm{cmp}\ h_1\ h_2\ \mathbf{of} \\
&\qquad \mathrm{LTNat}\ z\ \to \mathrm{Hor}\ w_1\ (\mathrm{Ver}\ h_1\ b_1\ (\mathrm{Sy}\ z)\ \mathrm{Clear})\ w_2\ b_2 \\
&\qquad \mathrm{EQNat}\quad\ \to \mathrm{Hor}\ w_1\ b_1\ w_2\ b_2 \\
&\qquad \mathrm{GTNat}\ z \to \mathrm{Hor}\ w_1\ b_1\ w_2\ (\mathrm{Ver}\ h_2\ b_2\ (\mathrm{Sy}\ z)\ \mathrm{Clear})
\end{aligned}
$$

The required properties of maximum are available as type equations in the proof object.

Scalability

- Can add extra type equations to Cmp by hand.
- Seems difficult to be more modular without higher-order constraints.
- Works for other equations concerning properties of functions such as subtraction that are defined inductively on the structure of natural numbers.