Mendler-style Recursion Schemes for Mixed-Variant Datatypes

Ki Yung Ahn  Tim Sheard  Marcelo P. Fiore

kya@cs.pdx.edu  sheard@cs.pdx.edu  Marcelo.Fiore@cl.cam.ac.uk
Outline

Mendler-style Recursion Schemes
  mit (iteration)
  msfit (iteration with syntactic Inverse)

Untyped HOAS (Regular Mixed-Variant Datatype)
Formatting Untyped HOAS (using msfit at kind \(*\))

Simply-Typed HOAS (Indexed Mixed-Variant Datatype)
Formatting Untyped HOAS (using msfit at kind \(* \rightarrow *\))

The Nax language

Need for yet another Mendler-style Recursion Scheme
(iso-) Recursive Types in Functional Languages

kinding: \( (\mu\text{-form}) \frac{\Gamma \vdash F : * \rightarrow *}{\Gamma \vdash \mu F : *} \)

typing: \( (\mu\text{-intro}) \frac{\Gamma \vdash t : F(\mu F')}{\Gamma \vdash \text{ln} t : \mu F} \) \( (\mu\text{-elim}) \frac{\Gamma \vdash t : \mu F}{\Gamma \vdash \text{unln} t : F(\mu F)} \)

reduction: \( (\text{unln-ln}) \frac{\text{unln} (\text{ln} t) \sim t}{\text{unln} (\text{ln} t) \sim t} \)

already inconsistent as a logic (via Curry--Howard correspondence) without even having to introduce any term-level recursion since we can already define general recursion using this
Having both unrestricted formation and unrestricted elimination of $\mu$ leads to inconsistency

data $T \ a \ r = C \ (r \rightarrow a)$

$w : \mu \ (T \ a) \rightarrow a$ -- encoding of $(\lambda x. \ x \ x)$ in the untyped $\lambda$-calc

$w = \lambda v. \ \text{case} \ (\text{unIn} \ v) \ \text{of} \ (C \ x) \rightarrow f \ (C \ x)$

-- $w \ (C \ w)$ amounts to a well-known diverging term in $\lambda$-calculus

$\text{fix} : (a \rightarrow a) \rightarrow a$ -- the $Y$ combinator $(\lambda f. (\lambda x. f(xx))(\lambda x. f(xx)))$

$\text{fix} = \lambda f. \ (\lambda x. f \ (w \ x)) \ (\text{In} \ (C \ (\lambda x. f \ (w \ x))))$

To recover consistency, one could

either restrict formation (conventional approach)
or restrict elimination (Mendler-style approach)
Conventional Iteration over Recursive Types

kinding: \((\mu\text{-form}^+)\) \(\frac{\Gamma \vdash F : * \rightarrow * \quad \text{positive}(F)}{\Gamma \vdash \mu F : *}\)

allow formation of only positive recursive types (i.e., when \(F\) has a map)

typing: \((\mu\text{-intro})\) and \((\mu\text{-elim})\) same as functional language

\[(\text{It})\] \(\frac{\Gamma \vdash t : \mu F \quad \Gamma \vdash \varphi : FA \rightarrow A}{\Gamma \vdash \text{It} \ \varphi \ t : A}\)

freely eliminate (i.e. use unIn) recursive values

reduction: \((\text{unIn-ln})\) same as functional language

\[(\text{It-ln})\] \(\frac{\text{It} \ \varphi \ (\text{ln} \ t) \leadsto \varphi \ (\text{map}_F \ (\text{It} \ \varphi) \ t)}{}\)
Mendler-style Iteration over Recursive Types

kinding: (\(\mu\)-form) same as functional language

\[ \text{freely form ANY recursive type!!} \]

typing: (\(\mu\)-intro) same as functional language

\[
\frac{\Gamma \vdash t : \mu F \quad \Gamma \vdash \varphi : \forall X.(X \rightarrow A) \rightarrow FX \rightarrow A} {\Gamma \vdash \text{mit } \varphi t : A}
\]

\(\text{note, no (\(\mu\)-elim) rule}\)

elimination is possible only through mit

reduction: (\text{mit-ln})

\[
\text{mit } \varphi (\text{ln } t) \leadsto \varphi (\text{mit } \varphi) t
\]
Mendler-style Iteration

-- Mu at different kinds (there are many more, one at each kind)
\[
\text{data } \text{Mu0 } (f :: * \rightarrow *) \quad = \text{In0 } (f \ (\text{Mu0 } f )) \\
\text{data } \text{Mu1 } (f :: (*\rightarrow *)\rightarrow(*\rightarrow *)) \ i = \text{In1 } (f \ (\text{Mu1 } f ) \ i)
\]

-- Mendler-style iterators at different kinds
\[
\text{mit0 } :: \Phi0 \ f \ a \rightarrow \text{Mu } f \rightarrow a \\
\text{mit0 } \phi i (\text{In0 } x) = \phi i (\text{mit0 } \phi i) \ x
\]
\[
\text{mit1 } :: \Phi1 \ f \ a \ i \rightarrow \text{Mu } f \ i \rightarrow a \ i \\
\text{mit1 } \phi i (\text{In1 } x) = \phi i (\text{mit1 } \phi i) \ x
\]

\[
\begin{align*}
\text{type } \Phi0 &= \forall r. \ (r \rightarrow a ) \rightarrow \ (f \ r \rightarrow a ) \\
\text{type } \Phi1 &= \forall r. \ (\forall i. \ r \ i \rightarrow a \ i) \rightarrow \ (\forall i. \ f \ r \ i \rightarrow a \ i)
\end{align*}
\]

✔ Uniformly defined at different kinds (can handle non-regular datatypes quite the same way as handling regular datatypes)

✔ Terminates for ANY datatype (can embed Mu and mit into Fw. see Abel, Matthes and Uustalu TCS'04)

✔ However, not very useful for mixed-variant datatypes

\[
\text{eliminating In0 or In1 by pattern matching is only allowed in Mendler-style iterators}
\]
Mendler-style Iteration
with a syntactic inverse

-- Mu' at different kinds (we only use two of them in this talk)

\[
\text{data } \text{Mu}'0 \ (f :: \star \rightarrow \star) \quad \text{a} \quad = \text{In}'0 \ (f \ (\text{Mu}'0 \ f \ a)) \mid \text{Inverse0} \ a \\
\text{data } \text{Mu}'1 \ (f :: (\star \rightarrow \star) \rightarrow (\star \rightarrow \star)) \quad \text{a i} \quad = \text{In}'1 \ (f \ (\text{Mu}'1 \ f \ a \ i)) \mid \text{Inverse1} \ (\text{a i})
\]

-- msfit at different kinds

\[
\text{msfit0} :: \text{Phi}'0 \ f \ a \rightarrow (\forall \text{a}. \ \text{Mu}'0 \ f \ a) \rightarrow \text{a} \\
\text{msfit0 phi} \ (\text{In}'0 \ x) = \phi \text{Inverse0} \ (\text{msfit0 phi}) \ x
\]

\[
\text{msfit1} :: \text{Phi}'1 \ f \ a \ i \rightarrow (\forall \text{a}. \ \text{Mu}'1 \ f \ a \ i) \rightarrow \text{a i} \\
\text{msfit1 phi} \ (\text{In}'1 \ x) = \phi \text{Inverse1} \ (\text{msfit1 phi}) \ x
\]

✔ Terminates for ANY datatype
(can embed Mu and msfit into Fw. see Ahn and Sheard ICFP'11)

✔ msfit is quite useful for
mixed-variant datatypes,
due to "inverse" operation in
addition to "recursive call"

inverse

\[
\text{type } \text{Phi}'0 \ f \ a = \forall \text{r}. \quad (\text{a} \rightarrow \text{r a a}) \rightarrow \quad (\text{r a a} \rightarrow \text{a}) \rightarrow \quad (f \text{r a a} \rightarrow \text{a})
\]

recursion

\[
\text{type } \text{Phi}'1 \ f \ a = \forall \text{r}. \quad (\forall \text{i. a i a i} \rightarrow \text{r a i}) \rightarrow \quad (\forall \text{i. r a i a i} \rightarrow \text{a i}) \rightarrow \quad (\forall \text{i. f (r a i) a i a i})
\]

eliminating In'0 or In'1 by pattern matching
is only allowed in Mendler-style iterators
Untyped HOAS
(a regular mixed-variant datatype)

-- using general recursion at type level
data Exp = Lam (Exp → Exp) | App Exp Exp

-- using fixpoint (Mu'0) over non-recursive base structure (ExpF)
data ExpF r = Lam (r → r) | App r r
type Exp' a = Mu'0 ExpF a -- (Exp' a) may contain Inverse
type Exp = ∀ a . Exp' a -- Exp does not contain Inverse

lam :: ( ∀ a . Exp' a → Exp' a ) → Exp -- it's not (Exp → Exp) → Exp
lam f = In'0 (Lam f ) -- f can handle Inverse containing values
-- but can never examine its content

app :: Exp → Exp → Exp
app e₁ e₂ = In'0 (App e₁ e₂)
Formatting Untyped HOAS using msfit0 at kind *

showExp :: Exp → String
showExp e = msfit0 phi e vars where
  phi :: Phi'0 ExpF ([String] → String)
  phi inv showE (Lam z) =
    λ(v:vs) → "(λ"++v++"→"++ showE (z (inv (const v))) vs ++")"
  phi inv showE (App x y) =
    λvs → "("++ showE x vs ++" "++ showE y vs ++")"

inverse recursive call

type Phi0 f a = ∀ r. (a → r a) → (r a → a) → (f r a → a)

vars = [ "x"++show n | n<-[0..] ] :: [String]
Simply-Typed HOAS
(a non-regular mixed-variant datatype)

-- using general recursion at type level
data Exp t where -- Exp :: * → *
   Lam :: (Exp t₁ → Exp t₂) → Exp (t₁ → t₂)
   App :: Exp (t₁ → t₂) → Exp t₁ → Exp t₂

-- using fixpoint (Mu'1) over non-recursive base structure (ExpF)
data ExpF r t where -- ExpF :: (* → *) → (* → *)
   Lam :: (r t₁ → r t₂) → ExpF r (t₁ → t₂)
   App :: r (t₁ → t₂) → r t₁ → ExpF r t₂

type Exp' a t = Mu'1 ExpF a t -- (Exp' a t) might contain Inverse
type Exp t = ∀ a . Exp' a t -- (Exp t) does not contain Inverse

lam :: (∀ a . Exp' a t₁ → Exp' a t₂) → Exp (t₁ → t₂)
lam f = In'1 (Lam f) -- f can handle Inverse containing values
   -- but can never examine its content

app :: Exp (t₁ → t₂) → Exp t₁ → Exp t₂
app e₁ e₂ = In'1 (App e₁ e₂)
Evaluating Simply-Typed HOAS using msfit1 at kind $* \rightarrow *$

newtype Id a = MkId { unId :: a }

evalHOAS :: Exp t → Id t
evalHOAS e = msfit1 phi e where
  phi :: Phi'1 ExpF Id
  phi inv ev (Lam f) = MkId(λv → unId(ev (f (inv (MkId v)))))
  phi inv ev (App e₁ e₂) = MkId(unId(ev e₁) (unId(ev e₂)))
  inverse
  recursive call

type Phi'1 f a = ∀ r. (∀ i. a i → r a i) → (∀ i. r a i → a i) → (∀ i. f (r a) i → a i)

This is example is a super awesome coooool discovery that System Fw can express Simply-Typed HOAS evaluation!!!
The Nax language

✔ Built upon the idea of Mendler-style recursion schemes
✔ Supports an extension of Hindley--Milner type inference to make it easier to use Mendler-style recursion schemes and indexed datatypes
✔ Handling different notions of recursive type operators (Mu, Mu') still needs more rigorous clarification in the theory, but intuitively,

\[
\begin{align*}
\text{Mu0 } f & = \forall a. \text{Mu'0 } f \, a \\
\text{Mu1 } f & = \forall a. \text{Mu'1 } f \, a \, i \\
& \ldots
\end{align*}
\]

So, we hide Mu' from users as if there is one kind of Mu and Mu' is only used during the computation of msfit

Need for yet another Mendler-style recursion scheme

There are more recursion schemes other than \texttt{mit} and \texttt{msfit}

E.g., Mendler-style primitive recursion (\texttt{mpr}) can \texttt{cast} from abstract recursive type (\texttt{r}) to concrete recursive type (\texttt{Mu f}).

\[
\text{Phi0 } f \ a = \forall r. \quad (r \rightarrow Mu \ f) \rightarrow (r \rightarrow a) \rightarrow (fr \rightarrow a)
\]
\[
\text{Phi1 } f \ a = \forall r. (\forall i. \ r \ i \rightarrow Mu \ f \ i) \rightarrow (\forall i. \ r \ i \rightarrow a \ i) \rightarrow (\forall i. \ f \ r \ i \rightarrow a \ i)
\]

With \texttt{mpr}, one can write constant time predecessor for natural numbers and tail function for lists by using the cast operation.

Next example motivates an extension to \texttt{mpr} that can \texttt{uncast} from concrete recursive type (\texttt{Mu f i}) to abstract recursive type (\texttt{r i})

\[
\text{Phi1 } f \ a = \forall r. (\forall i. \ Mu \ f \ i \rightarrow r \ i) \rightarrow
\]
\[
(\forall i. \ r \ i \rightarrow Mu \ f \ i) \rightarrow (\forall i. \ r \ i \rightarrow a \ i) \rightarrow (\forall i. \ f \ r \ i \rightarrow a \ i)
\]

Recursion scheme with above Phi1 type does not terminate for mixed-variant datatypes -- needs some additional restriction on uncast.
Evaluating Simply-Typed HOAS into a user defined value domain

data V r t where V :: (r t₁ → r t₂) → V r (t₁ → t₂)
type Val t = Mu V t
val f = In1 (V f)

evalHOAS :: Exp t → Val t
evalHOAS e = msfit1 phi e where
  phi :: Phi'1 ExpF (Mu1 V) -- f :: r Val t₁ -> r Val t₂ , v :: Val t₁
  phi inv ev (Lam f) = val(λv → ev (f (inv v)))
  phi inv ev (App e₁ e₂) = unVal (ev e₁) (ev e₂)

Only if we had unVal :: Val (t₁ → t₂) → (Val t₁ → Val t₂) it would be possible to write this, but unVal does not seem to be definable using any known Mendler-style recursion scheme.
New recursion scheme to write unVal

data V r t where V :: (r t₁ → r t₂) → V r (t₁ → t₂)
type Val t = Mu V t
val f = In1 (V f)
unVal v = unId(mprsi phi v) where
  phi :: Phi1 V Id
  phi uncast cast (V f) = Id(λv. cast (f (uncast v)))

mprsi :: Phi1 f a → Mu f i → a i
mprsi phi (In1 x) = phi id id (mprsi phi) x
-- size restriction over indices in both uncast and cast

Preliminary idea that still need further studies for the termination proof

\[
\text{type } \Phi_1 f a = \forall r j. \quad \begin{align*}
(\forall i. (i < j) \Rightarrow Mu f i & \rightarrow r i) \rightarrow \quad & \text{-- uncast} \\
(\forall i. (i < j) \Rightarrow r i & \rightarrow Mu f i) \rightarrow \quad & \text{-- cast} \\
(\forall i. r i & \rightarrow a i) \rightarrow (r j & \rightarrow a j)
\end{align*}
\]

-- or, maybe without the size restriction over indices in cast

\[
\text{type } \Phi_1 f a = \forall r j. \quad \begin{align*}
(\forall i. (i < j) \Rightarrow Mu f i & \rightarrow r i) \rightarrow \quad & \text{-- uncast} \\
(\forall i. r i & \rightarrow Mu f i) \rightarrow \quad & \text{-- cast} \\
(\forall i. r i & \rightarrow a i) \rightarrow (f r j & \rightarrow a j)
\end{align*}
\]
Questions or Suggestions?

Thanks for listening.

Ki Yung Ahn <kya@cs.pdx.edu> is graduating soon this summer and openly looking for research positions worldwide.