Generic Programming Within Dependently Typed Programming (Implementation)

Thorsten Altenkirch and Conor McBride

July 25, 2002

1 Introduction

This document contains the annotated Oleg code for our implementation of generic operators for concrete Haskell datatypes.

Oleg was never intended as a programming language, and the fact that we can use it as such is a testament to the flexibility of Randy Pollack's implementation. Before inductive datatypes were supported more directly, a simple mechanism was added whereby the user could declare formation, introduction and elimination rules, then equip the latter with computational behaviour. It is this support for ad hoc computation rules which enables us to write our programs, but it is somewhat rudimentary, for example requiring that all pattern variables be explicitly typed in advance. We wish to emphasize that the notational prolixity demanded by this mechanism is *not* indicative of what a purpose-built programming platform would require.

2 Preamble

We define constants required by Oleg's mechanism for inductive datatypes:

```
[SET=Type]
[TYPE=Type];
```

We declare a type used to mark up elimination rules for the `Elim` tactic:

```
$[Target : {A|Type}A->Type]
$[on : {A|Type}{a:A}Target a];
Label (!Target!) Target;
Label (!Target on!) on;
```

We declare the heterogeneous 'John Major' equality and prove some useful properties of it.

```
$[op== : {A,B|Type}A->B->Prop // JMr];
Configure Infix == right 2;

$[JMr : {A|Type}{x:A}x == x];

$[JM_elim : {A|Type}{a:A}{P:{b:A}(a == b)->Type}(P a (JMr a))->
{b:A}{q:a == b}{tg:Target q}P b q];

[[A:Type][a,b:A][P:{b:A}(a == b)->Type][p:P a (JMr a)][tg:Target (JMr a)]
  JM_elim a P p a (JMr a) tg ==> p
];

Label (!JM!) op==;
Label (!JM refl!) JMr;
Label (!JM elim!) JM_elim;

Goal JM_J : {A|Type}{a,b|A}{q:a == b}{tg:Target q}
            {Phi:A->Type}{phi:Phi a}Phi b;
Elim (!JM elim!) q;
  intros; Immed;
Save;
Label (!JM J!) JM_J;

Goal JM_Jsym : {A|Type}{b,a|A}{q:b == a}{tg:Target q}
               {Phi:A->Type}{phi:Phi a}Phi b;
Elim (!JM elim!) q;
  intros; Immed;
Save;
Label (!JM Jsym!) JM_Jsym;

Goal JM_K : {A|Type}{a|A}{q:a == a}{tg:Target q}
            {Phi:(a == a)->Type}{phi:Phi (JMr a)}Phi q;
Elim (!JM elim!) q;
  intros; Immed;
Save;
Label (!JM K!) JM_K;
```

We shall not discuss this definition here, save to observe that it allows elements of different types to *aspire* to equality, but only allows elements of the same type to be *treated* as equal (and thus substituted).

3 Natural Numbers, Finite Sets and Vectors

Natural numbers, the finite set family and the functional encoding of vectors are introduced as follows:

$$\text{data} \quad \frac{}{\mathbb{N} \ : \ \mathsf{Type}} \quad \text{where} \quad \frac{}{\mathsf{0} \ : \ \mathbb{N}} \quad \frac{n \ : \ \mathbb{N}}{\mathsf{s}n \ : \ \mathbb{N}}$$

$$\text{data} \quad \frac{n \,:\, \mathbb{N}}{\mathsf{Fin}\; n \,:\, \mathsf{Type}} \quad \text{where} \quad \frac{}{0_n \,:\, \mathsf{Fin}\,\mathsf{s}n} \quad \frac{i \,:\, \mathsf{Fin}\; n}{\mathsf{s}_n\; i \,:\, \mathsf{Fin}\,\mathsf{s}n}$$

$$\text{let} \quad \frac{A \,:\, \mathsf{Type} \quad n \,:\, \mathbb{N}}{A^n \,:\, \mathsf{Type}} \quad A^n \;\mapsto\; (\mathsf{Fin}\; n) \to A$$

```
Inductive [Nat : Type] Constructors
  [n0 : Nat]
  [nS : {n:Nat}Nat];

Inductive [Fin : Nat -> Type] Constructors
  [f0 : {n:Nat}Fin (nS n)]
  [fS : {n|Nat}{i:Fin n}Fin (nS n)];

[op&v
   [A:Type][n:Nat]
 = (Fin n) -> A
 : Type
];
Configure Infix &v left 4;
```

Given $\boldsymbol{a} \,:\, A^{\mathsf{s}n}$, its head $\mathbf{hd}\,\boldsymbol{a} \,:\, A$ is given by $\mathbf{hd}\,\boldsymbol{a} \;\mapsto\; \boldsymbol{a}\,0$ and its tail $\mathbf{tl}\,\boldsymbol{a} \,:\, A^n$ by $\mathbf{tl}\,\boldsymbol{a} \;\mapsto\; \boldsymbol{a} \circ \mathsf{s}$. We also define constant vectors, and application of a vector of functions to a vector of arguments. In the code (but not the paper), we give 'nil' and 'cons' operations too.

$$\text{let} \quad \frac{a \,:\, A}{a^n \,:\, A^n} \quad a^n\; x \;\mapsto\; a$$

$$\text{let} \quad \frac{\boldsymbol{f} \,:\, (A \to B)^n \quad \boldsymbol{a} \,:\, A^n}{\boldsymbol{f}\;\boldsymbol{a} \,:\, B^n} \quad \boldsymbol{f}\;\boldsymbol{a}\;x \;\mapsto\; (\boldsymbol{f}\;x)\,(\boldsymbol{a}\;x)$$

```
[vHd
   [A|Type][n|Nat][xs:A &v (nS n)]
 = xs (f0 n)
 : A
];

[vTl
   [A|Type][n|Nat][xs:A &v (nS n)]
 = [i:Fin n]xs (fS i)
 : A &v n
];

$[vNil : {A:Type}A &v n0];

$[vCons : {A|Type}{n|Nat}{x:A}{xs:A &v n}(A &v (nS n))];
[[A|Type][n|Nat][x:A][xs:A &v n][i:Fin n]
   vCons x xs  f0    ==> x
|| vCons x xs (fS i) ==> xs i
];
```

```
[op&c
   [A|Type][a:A][n:Nat]
 = [i:Fin n]a
 : A &v n
];
Configure Infix &c left 5;

[op&a
   [A,B|Type][n|Nat][fs:(A -> B) &v n][xs:A &v n]
 = [i:Fin n]fs i (xs i)
 : B &v n
];
Configure Infix &a left 4;
```

4 Kinds, Signatures and Variables

First we introduce $\Box$ as a representation of Haskell kinds. Signatures $\mathsf{Sig}\ :\ \mathsf{Type}$ are sequences of kinds. As we often need access to the argument list of a kind, we shall identify $\mathsf{Sig}\ \mapsto\ \Box$.

$$\text{data}\quad \frac{}{\Box\ :\ \mathsf{Type}}\quad \text{where}\quad \frac{}{\star\ :\ \Box}\quad \frac{J, K\ :\ \Box}{J \Rightarrow K\ :\ \Box}$$

```
Inductive [Kind : Type] Constructors
  [STAR : Kind]
  [op>> : {S,T:Kind}Kind];
Configure Infix >> right 3;

[Sig = Kind];
```

We will use typed de Bruijn variables for our datatype declarations. Given a signature Σ and a kind K we define the type of variables of that kind as

$$\text{data}\quad \frac{\Sigma\ :\ \mathsf{Sig}\quad K\ :\ \Box}{\mathsf{Var}\,\Sigma\,K\ :\ \mathsf{Type}}\quad \text{where}\quad \frac{}{0\ :\ \mathsf{Var}\,(K;\Sigma)\,K}$$

$$\frac{v\ :\ \mathsf{Var}\,\Sigma\,K}{\mathsf{s}v\ :\ \mathsf{Var}\,(J;\Sigma)\,K}$$

```
Inductive [Var : Sig -> Kind -> Type] Constructors
  [v0 : {K:Kind}{S:Sig}Var (K >> S) K]
  [vS : {J,K|Kind}{S|Sig}{v:Var S K}Var (J >> S) K];
```

5 Type Expressions

We will now introduce the type Ty of polynomial type expressions representing the right hand side of a datatype declaration. Ty is indexed by two signatures Δ for mutually defined datatypes and Λ for parameter kinds.

$$\text{data} \quad \frac{\Delta, \Lambda \;:\; \mathsf{Sig} \qquad K \;:\; \Box}{\mathsf{Ty}\;\Delta\;\Lambda\;K \;:\; \mathsf{Type}}$$

$$\text{where} \quad \frac{v \;:\; \mathsf{Var}\;\Delta\;K}{\mathsf{D}\;v \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;K} \qquad \frac{v \;:\; \mathsf{Var}\;\Lambda\;K}{\mathsf{V}\;v \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;K}$$

$$\frac{F \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;(J \Rightarrow K) \qquad X \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;J}{F \cdot X \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;K}$$

$$\frac{}{\mathsf{0} \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;\star} \qquad \frac{S, T \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;\star}{S + T \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;\star}$$

$$\frac{}{\mathsf{1} \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;\star} \qquad \frac{S, T \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;\star}{S \times T \;:\; \mathsf{Ty}\;\Delta\;\Lambda\;\star}$$

OLEG permits us to declare the signatures globally. We discharge them when we have finished defining everything parametrized by them.

```
[Del,Lam ? Sig];

Inductive [Ty : Kind -> Type] Constructors
  [D : {K|Kind}{v:Var Del K}Ty K]
  [V : {K|Kind}{v:Var Lam K}Ty K]
  [op/ : {J,K|Kind}{F:Ty (J >> K)}{X:Ty J}Ty K]
  [Zero,One : Ty STAR]
  [op+,op* : {S,T:Ty STAR}Ty STAR];
Configure Infix + right 4;
Configure Infix * right 5;
Configure Infix / left 6;

Discharge Lam;

[TY = Ty STAR];
```

Here, we have discharged `Lam`, but `Del` is still global.

6 Argument Stacks, Application and Substitution

To define the interpretation of types, we introduce an iterated application operator and substitution on type expressions. Both are parametrized by an argument stack of closed types, represented via a function space, as follows:

$$\text{let} \quad \frac{\Delta \;:\; \mathsf{Sig} \qquad K \;:\; \Box}{\mathbf{Args}\;\Delta\;K \;:\; \mathsf{Type}}$$

$$\mathbf{Args}\;\Delta\;K \;\mapsto\; \forall_{J:\Box}.\;\mathsf{Var}\;\langle K\rangle\;J \;\rightarrow\; \mathsf{Ty}_{\Delta}\;J$$

```
[Args
   [K:Kind]
 = {J|Kind}(Var K J) -> TY J
 : Type
];
```

As with vectors, such functions (nonempty signatures) admit head and tail operations, by application to $\mathsf{0}$ and composition with s, respectively.

$$\text{let}\quad \frac{\vec{f}\ :\ \forall_{J:\Box}.\,\mathsf{Var}\,(K;\Sigma)\,J\ \to\ F\,J}{\mathbf{hd}\,\vec{f}\ :\ F\,K}\qquad \mathbf{hd}\,\vec{f}\ \mapsto\ \vec{f}\,\mathsf{0}$$

$$\text{let}\quad \frac{\vec{f}\ :\ \forall_{J:\Box}.\,\mathsf{Var}\,(K;\Sigma)\,J\ \to\ F\,J}{\mathbf{tl}\,\vec{f}\ :\ \forall_{J:\Box}.\,\mathsf{Var}\,\Sigma\,J\ \to\ F\,J}\qquad \mathbf{tl}\,\vec{f}\ \mapsto\ \vec{f}\circ\mathsf{s}$$

We also give 'nil' and 'cons'.

```
[aHd
   [J,K|Kind][Xs:Args (J >> K)]
 = Xs v0
 : TY J
];

[aTl
   [J,K|Kind][Xs:Args (J >> K)]
 = [L|Kind][v:Var K L]Xs (vS v)
 : Args K
];

$[aNil : Args STAR];

$[aCons : {J,K|Kind}{X:TY J}{Xs:Args K}Args (J >> K)];
[[J,K,L|Kind][X:TY J][Xs:Args K][v:Var K L]
   aCons X Xs  v0    ==> X
|| aCons X Xs (vS v) ==> Xs v
];
```

Hence we may now define the application operator.

$$\text{let}\quad \frac{X\ :\ \mathsf{Ty}_\Delta\,K\qquad \vec{Y}\ :\ \mathbf{Args}\,\Delta\,K}{X@_K\,\vec{Y}\ :\ \mathsf{Ty}_\Delta\,\star}$$

$$\begin{array}{lcl} T\ \ @_\star\ \ \vec{Y} & \mapsto & T \\ F\ @_{J\Rightarrow K}\ \vec{Y} & \mapsto & (F\cdot\mathbf{hd}\,\vec{Y})\,@_K\,(\mathbf{tl}\,\vec{Y}) \end{array}$$

```
$[op@a : {K|Kind}{X:TY K}{Ys:Args K}TY STAR];
Configure Infix @a left 6;
[[J,K|Kind][X:TY STAR][Ys0:Args STAR][F:TY (J >> K)][YsS:Args (J >> K)]
   op@a|STAR     X Ys0 ==> X
|| op@a|(J >> K) F YsS ==> F / aHd YsS @a aTl YsS
];
```

We define substitution by recursion over the structure of type expressions.

$$
\text{let}\quad \frac{X\ :\ \mathsf{Ty}\,\Delta\,\langle J\rangle\,K \qquad \vec{Y}\ :\ \mathbf{Args}\,\Delta\,J}{X[\vec{Y}]\ :\ \mathsf{Ty}_{\Delta}\,K}
\qquad
\begin{array}{ll}
(\mathsf{V}\,v)[\vec{Y}] & \mapsto \vec{Y}\,v \\
(\mathsf{D}\,v)[\vec{Y}] & \mapsto \mathsf{D}\,v \\
(F\cdot X)[\vec{Y}] & \mapsto F[\vec{Y}]\cdot X[\vec{Y}] \\
\mathsf{0}[\vec{Y}] & \mapsto \mathsf{0} \\
\mathsf{1}[\vec{Y}] & \mapsto \mathsf{1} \\
(S+T)[\vec{Y}] & \mapsto S[\vec{Y}]+T[\vec{Y}] \\
(S\times T)[\vec{Y}] & \mapsto S[\vec{Y}]\times T[\vec{Y}]
\end{array}
$$

```
$[op<< : {L,K|Kind}(Ty L K) -> (Args L) -> TY K];
Configure Infix << left 1;
[[J,K,L|Kind][v:Var L K][d:Var Del K][Ys:Args L]
 [F:Ty L (J >> K)][X:Ty L J]
 [S,T:Ty L STAR]
   V v    << Ys ==> Ys v
|| D L d  << Ys ==> D STAR d
|| F / X  << Ys ==> (F << Ys) / (X << Ys)
|| Zero L << Ys ==> Zero STAR
|| One L  << Ys ==> One STAR
|| S + T  << Ys ==> (S << Ys) + (T << Ys)
|| S * T  << Ys ==> (S << Ys) * (T << Ys)
];
```

7 Environments and Data

We define the meaning of a signature $[\![\Delta]\!]$ as the type of functions which assign to each variable a datatype declaration of the appropriate type:

$$
\text{let}\quad \frac{\Delta\ :\ \mathsf{Sig}}{[\![\Delta]\!]\ :\ \mathsf{Type}} \qquad [\![\Delta]\!]\ \mapsto\ \forall_{K\,:\,\Box}.(\mathsf{Var}\ \Delta\ K) \to \mathsf{Ty}\ \Delta\ \langle K\rangle\ \star
$$

Here, we just define the type of environments inhabiting the global `Del`, then take a further global assumption that there is such an environment—this too will be discharged in due course.

```
[Env = {K|Kind}(Var Del K) -> (Ty K STAR)];

[del ? Env];
```

We are now able to define the interpretation of types $[\![T]\!]_\delta$.

$$\text{data}\quad \frac{T \ : \ \mathsf{Ty}_{\Delta}\,\star}{[\![T]\!]_\delta \ : \ \mathsf{Type}}$$

$$\text{where}\quad \frac{t \ : \ [\![(\delta\ v)[\vec{X}]]\!]_\delta}{\mathsf{con}_v\ t \ : \ [\![\mathsf{D}\ v@\vec{X}]\!]_\delta}$$

$$\frac{s \ : \ [\![S]\!]_\delta}{\mathsf{inl}\ s \ : \ [\![S+T]\!]_\delta} \qquad \frac{t \ : \ [\![T]\!]_\delta}{\mathsf{inr}\ t \ : \ [\![S+T]\!]_\delta}$$

$$\frac{}{\mathsf{void} \ : \ [\![1]\!]_\delta} \qquad \frac{s \ : \ [\![S]\!]_\delta \quad t \ : \ [\![T]\!]_\delta}{\mathsf{pair}\ s\ t \ : \ [\![S\times T]\!]_\delta}$$

```
Inductive [Data : {T:TY STAR}Type] Constructors
  [con : {K|Kind}{Xs|Args K}{d:Var Del K}
         {t:Data (del d << Xs)}Data (D STAR d @a Xs)]
  [inl : {S,T|TY STAR}{s:Data S}Data (S + T)]
  [inr : {S,T|TY STAR}{t:Data T}Data (S + T)]
  [void : Data One]
  [pair : {S,T|TY STAR}{s:Data S}{t:Data T}Data (S * T)];
```

8 The Fold Operator

We now present the code for the fold operator. Where, in the paper, each fold is parametrized by some Φ taking the source type and a vector of auxiliary types,

$$\frac{S \ : \ \mathsf{Ty}_{\Delta}\,\star \qquad \boldsymbol{T} \ : \ (\mathsf{Ty}_{\Delta}\star)^n}{\Phi\ S\ \boldsymbol{T} \ : \ \mathsf{Type}}$$

here we just require a nonempty vector of types and take the head type as the source type for the fold. The length of the vector and the choice of `Phi` are parametric to the whole development, so once again we assume them globally and will discharge them later.

```
[n|Nat]
[Phi : (TY STAR &v (nS n)) -> Type];
```

8.1 The Type of Uncurried Fold

We define the type of uncurried folds, making use of OLEG's native Σ-types, rather than defining pairs inductively (which would also work). Any inhabited type will serve as 'unit': we use `Prop`.

$$\text{let} \quad \frac{\Phi \;:\; \mathsf{Ty}_\Delta\,\star \;\to\; (\mathsf{Ty}_\Delta\,\star)^n \;\to\; \mathsf{Type} \qquad X \;:\; \mathsf{Ty}_\Delta\,K \quad \boldsymbol{Y} \;:\; (\mathsf{Ty}_\Delta\,K)^n}{\mathbf{UFold}_K\;\Phi\;X\;\boldsymbol{Y} \;:\; \mathsf{Type}}$$

$$\frac{\Phi \;:\; \mathsf{Ty}_\Delta\,\star \;\to\; (\mathsf{Ty}_\Delta\,\star)^n \;\to\; \mathsf{Type} \qquad \vec{W} \;:\; \mathbf{Args}\,\Delta\,K \quad \vec{\boldsymbol{Z}} \;:\; (\mathbf{Args}\,\Delta\,K)^n}{\mathbf{UFolds}_K\;\Phi\;\vec{W}\;\vec{\boldsymbol{Z}} \;:\; \mathsf{Type}}$$

$$\begin{array}{l}
\mathbf{UFold}_K\;\Phi\;X\;\boldsymbol{Y} \;\mapsto \\
\quad \forall_{\vec{W}:\mathbf{Args}\,\Delta\,K}.\,\forall_{\vec{\boldsymbol{Z}}:(\mathbf{Args}\,\Delta\,K)^n}.\,\mathbf{UFolds}_K\;\Phi\;\vec{W}\;\vec{\boldsymbol{Z}} \;\to \\
\qquad [\![X @ \vec{W}]\!]_\delta \;\to\; \Phi\,(X @ \vec{W})\,(\boldsymbol{Y} @^n \vec{\boldsymbol{Z}})
\end{array}$$

$$\begin{array}{l}
\mathbf{UFolds}_\star\;\Phi\;\vec{W}\;\vec{\boldsymbol{Z}} \;\mapsto\; \mathbf{1} \\
\mathbf{UFolds}_{J\Rightarrow K}\;\Phi\;\vec{W}\;\vec{\boldsymbol{Z}} \;\mapsto\; \mathbf{UFold}_J\;\Phi\,(\mathbf{hd}\;\vec{W})\,(\mathbf{hd}^n\;\vec{\boldsymbol{Z}}) \times \\
\qquad\qquad\qquad\qquad\qquad\; \mathbf{UFolds}_K\;\Phi\,(\mathbf{tl}\;\vec{W})\,(\mathbf{tl}^n\;\vec{\boldsymbol{Z}})
\end{array}$$

A little η-expansion is necessary, as OLEG uses only β-equality in typechecking. Recent experimental systems now implement η-normalization.

```
$[UFolds : {K|Kind}(Args K &v nS n)->Type // del Phi];
[UFold
   [K|Kind][Xs:TY K &v nS n]
 = {Yss|Args K &v nS n}[Yss'=[i:Fin (nS n)][J|Kind][v:Var K J]Yss i v]
   (UFolds Yss') ->
   (Data (vHd Xs @a vHd Yss')) ->
   Phi ((op@a|K) &c nS n &a Xs &a Yss')
 : Type
];
[[J,K|Kind][Yss0:Args STAR &v nS n][Yss1:Args (J >> K) &v nS n]
   UFolds|STAR     Yss0 ==> Prop
|| UFolds|(J >> K) Yss1 ==>
     (UFold|J  ((aHd|J|K) &c nS n &a Yss1)) #
     (UFolds|K ((aTl|J|K) &c nS n &a Yss1))
];
```

We define the 'transpose' operation for vectors of argument stacks:

$$\text{let} \quad \frac{\vec{\boldsymbol{Z}} \;:\; (\mathbf{Args}\,\Delta\,K)^n}{\vec{\boldsymbol{Z}}^\top \;:\; \forall_{J:\square}.\,\mathsf{Var}\,\langle K\rangle\,J \;\to\; (\mathsf{Ty}_\Delta\,J)^n} \qquad \vec{\boldsymbol{Z}}^\top\,v\,i \;\mapsto\; \vec{\boldsymbol{Z}}\,i\,v$$

```
[Tr
   [n|Nat][K|Kind][Xss:Args K &v n]
 = [J|Kind][v:Var K J][i:Fin n]Xss i v
 : {J|Kind}(Var K J) -> (TY J &v n)
];
```

We may thus define projection from an environment of folds:

$$\text{let}\quad \frac{\mathit{fs}\ :\ \mathbf{UFolds}_K\ \Phi\ \vec{W}\ \vec{\boldsymbol{Z}} \qquad v\ :\ \mathsf{Var}\ \langle K\rangle\ J}{\mathit{fs}.v\ :\ \mathbf{UFold}_J\ \Phi\ (\vec{W}\ v)\ (\vec{\boldsymbol{Z}}^\top\ v)}$$

$$(f, \mathit{fs}).0 \mapsto f$$
$$(f, \mathit{fs}).(\mathsf{s}\ v) \mapsto \mathit{fs}.v$$

```
$[UFproj : {K,J|Kind}{Yss|Args K &v nS n}
           {fs:UFolds Yss}{v:Var K J}
           UFold (Tr Yss v)];
[[J,K,L|Kind][Yss|Args (K >> L) &v nS n][fs:UFolds Yss]
 [v:Var L J]
   UFproj fs  v0    ==> fs.1
|| UFproj fs (vS v) ==> UFproj (fs.2) v
];
```

8.2 Uncurried Fold

We now take the constructor methods as global parameters of the fold:

$$\frac{\phi\ :\ \Phi\ ((\delta\ v)[\vec{W}])\ ((\delta\ v)^n[\vec{\boldsymbol{Z}}]^n)}{\mathit{doCon}\ v\ \phi\ :\ \Phi\ (\mathsf{D}\ v @ \vec{W})\ ((\mathsf{D}\ v)^n @^n \vec{\boldsymbol{Z}})}$$

$$\frac{\phi_1\ :\ \Phi\ S_1\ \boldsymbol{T}_1}{\mathit{doInl}\ \phi_1\ :\ \Phi\ (S_1{+}S_2)\ (\boldsymbol{T}_1{+}^n\ \boldsymbol{T}_2)} \quad \frac{\phi_2\ :\ \Phi\ S_2\ \boldsymbol{T}_2}{\mathit{doInr}\ \phi_2\ :\ \Phi\ (S_1{+}S_2)\ (\boldsymbol{T}_1{+}^n\ \boldsymbol{T}_2)}$$

$$\frac{}{\mathit{doVoid}\ :\ \Phi\ 1\ 1^n} \qquad \frac{\phi_1\ :\ \Phi\ S_1\ \boldsymbol{T}_1 \qquad \phi_2\ :\ \Phi\ S_2\ \boldsymbol{T}_2}{\mathit{doPair}\ \phi_1\ \phi_2\ :\ \Phi\ (S_1 \times S_2)\ (\boldsymbol{T}_1 \times^n \boldsymbol{T}_2)}$$

```
[doCon : {K|Kind}{d:Var Del K}{Wss|Args K &v nS n}
         (Phi (op<< (del d) &c nS n &a Wss)) ->
         Phi (op@a (D STAR d) &c nS n &a Wss)];

[doInl : {Ss,Ts|TY STAR &v nS n}
         (Phi Ss) -> Phi ((op+|STAR) &c nS n &a Ss &a Ts)];

[doInr : {Ss,Ts|TY STAR &v nS n}
         (Phi Ts) -> Phi ((op+|STAR) &c nS n &a Ss &a Ts)];

[doVoid : Phi (One STAR &c nS n)];

[doPair : {Ss,Ts|TY STAR &v nS n}
          (Phi Ss) -> (Phi Ts) -> Phi ((op*|STAR) &c nS n &a Ss &a Ts)];
```

We may then implement the core of the fold operator, operating on any type expression under any substitution.

$$\text{let}\quad \frac{\{\mathit{do}\text{'s}\} \qquad \mathit{es}\ :\ \mathbf{UFolds}_\Lambda\ \Phi\ \vec{W}\ \vec{\boldsymbol{Z}} \qquad X\ :\ \mathsf{Ty}\ \Delta\ \Lambda\ K}{\mathbf{ufoldBody}\ \{\mathit{do}\text{'s}\}\ \mathit{es}\ X\ :\ \mathbf{UFold}_K\ \Phi\ (X[\vec{W}])\ (X^n[\vec{\boldsymbol{Z}}]^n)}$$

$$
\begin{array}{l}
\textbf{ufoldBody}\ \{do\text{'s}\}\ es\ \ (\mathsf{V}\ v)\ \ fs\ \ \ \ t\ \ \ \ \mapsto\ es.v\ fs\ t\\
\textbf{ufoldBody}\ \{do\text{'s}\}\ es\ \ (F\cdot A)\ \ fs\ \ \ \ t\ \ \ \ \mapsto\\
\quad\textbf{ufoldBody}\ \{do\text{'s}\}\ es\ F\ (\textbf{ufoldBody}\ \{do\text{'s}\}\ es\ A, fs)\ t\\
\textbf{ufoldBody}\ \{do\text{'s}\}\ es\ \ (\mathsf{D}\ v)\ \ fs\ \ (\mathsf{con}\ t)\ \mapsto\\
\quad doCon\ v\ (\textbf{ufoldBody}\ \{do\text{'s}\}\ fs\ (\delta v)\ ()\ t)\\
\textbf{ufoldBody}\ \{do\text{'s}\}\ es\ (S+T)\ ()\ \ (\mathsf{inl}\ s)\ \mapsto\\
\quad doInl\ (\textbf{ufoldBody}\ \{do\text{'s}\}\ fs\ S\ ()\ s)\\
\textbf{ufoldBody}\ \{do\text{'s}\}\ es\ (S+T)\ ()\ \ (\mathsf{inr}\ t)\ \mapsto\\
\quad doInr\ (\textbf{ufoldBody}\ \{do\text{'s}\}\ fs\ T\ ()\ t)\\
\textbf{ufoldBody}\ \{do\text{'s}\}\ es\ \ \ 1\ \ \ ()\ \ \ \mathsf{void}\ \ \mapsto\ doVoid\\
\textbf{ufoldBody}\ \{do\text{'s}\}\ es\ (S\times T)\ ()\ (\mathsf{pair}\ s\ t)\mapsto\\
\quad doPair\ (\textbf{ufoldBody}\ \{do\text{'s}\}\ fs\ S\ ()\ s)\\
\qquad\qquad (\textbf{ufoldBody}\ \{do\text{'s}\}\ fs\ T\ ()\ t)
\end{array}
$$

```
$[ufoldBody : {L|Kind}{Wss|Args L &v nS n}{wfs:UFolds Wss}
              {K|Kind}{X:Ty L K}
              UFold (op<< X &c nS n &a Wss)
              // doCon doInl doInr doVoid doPair];

(* ufoldBody
 : {L|Kind}{Wss|Args L &v nS n}{wfs:UFolds Wss}
   {K|Kind}{X:Ty L K}
   {Yss|Args K &v nS n}[Yss'=[i:Fin (nS n)][J|Kind][v:Var K J]Yss i v]
   {yfs:UFolds Yss'}
   (Data ((X << vHd Wss) @a vHd Yss')) ->
   Phi ((op@a|K &c nS n) &a (op<< X &c nS n &a Wss) &a Yss')
*)

[[J,K,L|Kind][Wss|Args L &v nS n][wfs:UFolds Wss]
 [Yss|Args K &v nS n]
 [yfs:[Yss'=[i:Fin (nS n)][J|Kind][v:Var K J]Yss i v]UFolds Yss']
 [v:Var L K]
 [tV:[Yss'=[i:Fin (nS n)][J|Kind][v:Var K J]Yss i v]
     Data (vHd Wss v @a vHd Yss')]
 [F:Ty L (J >> K)][X:Ty L J]
 [tA:[Yss'=[i:Fin (nS n)][J|Kind][v:Var K J]Yss i v]
     Data ((F << vHd Wss) / (X << vHd Wss) @a vHd Yss')]
 [d:Var Del K]
 [tD:[Yss'=[i:Fin (nS n)][J|Kind][v:Var K J]Yss i v]
     Data (del d << vHd Yss')]
 [S,T:Ty L STAR]
 [tS:Data (S << vHd Wss)]
 [tT:Data (T << vHd Wss)]
 [Ys0|Args STAR &v nS n]
 [fs0:[Ys0'=[i:Fin (nS n)][J|Kind][v:Var STAR J]Ys0 i v]UFolds Ys0']

   ufoldBody wfs (V v)   yfs tV ==> UFproj wfs v yfs tV
|| ufoldBody wfs (F / X) yfs tA ==>
    [Yss'=[i:Fin (nS n)][J|Kind][v:Var K J]Yss i v]
    ufoldBody wfs F|([i:Fin (nS n)]aCons (X << Wss i) (Yss' i))
       (ufoldBody wfs X,yfs) tA
|| ufoldBody wfs (D L d) yfs (con d tD) ==>
     doCon d (ufoldBody yfs (del d)|(aNil &c nS n) ({P:Prop}P->P) tD)
```

```
|| ufoldBody wfs (S + T) fs0 (inl tS) ==>
     doInl|?|(op<< T &c ? &a Wss) (ufoldBody wfs S fs0 tS)
|| ufoldBody wfs (S + T) fs0 (inr tT) ==>
     doInr|(op<< S &c ? &a Wss) (ufoldBody wfs T fs0 tT)
|| ufoldBody wfs (One L) fs0 void ==> doVoid
|| ufoldBody wfs (S * T) fs0 (pair tS tT) ==>
     doPair (ufoldBody wfs S fs0 tS) (ufoldBody wfs T fs0 tT)
];
```

To restrict this to closed type expressions, we need the lemma

$$\text{let}\quad \frac{X\ :\ \mathsf{Ty}_\Delta\ K}{\textbf{idLemma}\, X\ :\ X[\iota] = X}$$

```
$[idLemma : {K|Kind}{X:TY K}(X << aNil) == X];
[[J,K|Kind][d:Var Del K]
 [F:TY (J >> K)][X:TY J][S,T:TY STAR]
   idLemma (D STAR d) ==> JMr (D STAR d)
|| idLemma (F / X) ==>
     JM_J ? (on (idLemma F))
     ([F':TY (J >> K)]((F << aNil) / (X << aNil)) == F' / X)
     (JM_J ? (on (idLemma X))
      ([X':TY J]((F << aNil) / (X << aNil)) == (F << aNil) / X')
      JMr)
|| idLemma (Zero STAR) ==> JMr (Zero STAR)
|| idLemma (One  STAR) ==> JMr (One  STAR)
|| idLemma (S + T) ==>
     JM_J ? (on (idLemma S))
     ([S':TY STAR]((S << aNil) + (T << aNil)) == S' + T)
     (JM_J ? (on (idLemma T))
      ([T':TY STAR]((S << aNil) + (T << aNil)) == (S << aNil) + T')
      JMr)
|| idLemma (S * T) ==>
     JM_J ? (on (idLemma S))
     ([S':TY STAR]((S << aNil) * (T << aNil)) == S' * T)
     (JM_J ? (on (idLemma T))
      ([T':TY STAR]((S << aNil) * (T << aNil)) == (S << aNil) * T')
      JMr)
];
```

We then have

$$\textbf{ufold}\,\{do\text{'s}\}\,X\ \mapsto\ \textbf{idLemma}\,X \triangleright \textbf{ufoldBody}\,\{do\text{'s}\}_{\iota\,\iota^n}\,()\,X$$

```
[ufold
   [K|Kind][X:TY K]
 = JM_J ? (on (idLemma X))
   ([X:TY K]UFold (X &c nS n))
   (ufoldBody|STAR|(aNil &c nS n) ({P:Prop}P->P) X)
];
```

8.3 Currying the Fold Operator

We really want a curried fold operator with this type:

$$\text{let}\quad \frac{X\ :\ \mathsf{Ty}_\Delta\ K \qquad \boldsymbol{Y}\ :\ (\mathsf{Ty}_\Delta\ K)^n}{\mathbf{Fold}_K\ \Phi\ X\ \boldsymbol{Y}\ :\ \mathsf{Type}}$$

$$\begin{array}{l} \mathbf{Fold}_\star \quad\ \Phi\ S\ \boldsymbol{T} \mapsto [\![S]\!]_\delta \to \Phi\ S\ \boldsymbol{T} \\ \mathbf{Fold}_{J\Rightarrow K}\ \Phi\ F\ \boldsymbol{G} \mapsto \forall_{X:\mathsf{Ty}_\Delta\ J}.\ \forall_{\boldsymbol{Y}:(\mathsf{Ty}_\Delta\ J)^n}. \\ \qquad\qquad\qquad\qquad \mathbf{Fold}_J\ \Phi\ X\ \boldsymbol{Y}\ \to\ \mathbf{Fold}_K\ \Phi\ (F\cdot X)\ (\boldsymbol{G}\cdot^n\ \boldsymbol{Y}) \end{array}$$

```
$[Fold : {K|Kind}{X:TY K &v nS n}Type // del Phi];
[[Ts : TY STAR &v nS n]
 [J,K|Kind][Fs : TY (J >> K) &v nS n]
   Fold|STAR     Ts ==> (Data (vHd Ts)) -> Phi ([i:Fin (nS n)]Ts i)
|| Fold|(J >> K) Fs ==>
     {Xs|TY J &v nS n}
     (Fold ([i:Fin (nS n)]Xs i)) ->
      Fold ((op/|STAR|J|K) &c nS n &a Fs &a Xs)
];
```

We thus need mutually defined 'curry' and 'uncurry' operators:

$$\text{let}\quad \frac{u\ :\ \mathbf{UFold}_K\ \Phi\ X\ \vec{Y}}{\mathbf{Curry}_K\ u\ :\ \mathbf{Fold}_K\ \Phi\ X\ \vec{Y}} \qquad \frac{f\ :\ \mathbf{Fold}_K\ \Phi\ X\ \vec{Y}}{\mathbf{Uncurry}_K\ f\ :\ \mathbf{UFold}_K\ \Phi\ X\ \vec{Y}}$$

$$\begin{array}{l} \mathbf{Curry}_\star \quad\ u \ \ \mapsto\ u() \\ \mathbf{Curry}_{J\Rightarrow K}\ u\ f \mapsto \\ \quad \mathbf{Curry}_K\ (\lambda_{\vec{W}}.\ \lambda_{\vec{Z}}.\ \lambda us.\ u\ (\mathbf{Uncurry}_J\ f, us)) \end{array}$$

$$\begin{array}{l} \mathbf{Uncurry}_\star \quad f \quad () \quad \mapsto f \\ \mathbf{Uncurry}_{J\Rightarrow K}\ f\ (u, us) \mapsto \\ \quad \mathbf{Uncurry}_K\ (f\ (\mathbf{Curry}_J\ u))\ us \end{array}$$

```
$[curry : {K|Kind}{Xs|TY K &v nS n}(UFold Xs)->Fold Xs];
$[uncurry : {K|Kind}{Xs|TY K &v nS n}(Fold Xs)->UFold Xs];

[[J,K|Kind]
 [Ts|TY STAR &v nS n][uTs:UFold Ts]
 [Fs|TY (J >> K) &v nS n][uFs:UFold Fs]
   curry|STAR     uTs ==> uTs|(aNil &c nS n) ({P:Prop}P->P)
|| curry|(J >> K) uFs ==>
     [Xs|TY J &v nS n][fXs:Fold ([i:Fin (nS n)]Xs i)]
     curry|K|((op/|STAR|J|K &c nS n) &a Fs &a Xs)
           ([Yss|Args K &v nS n]
            [usYss:UFolds ([i:Fin (nS n)][J|Kind][v:Var K J]Yss i v)]
            uFs|([i:Fin (nS n)]aCons (Xs i) (Yss i)) (uncurry fXs,usYss))
];
```

```
[[J,K|Kind]
 [Ts|TY STAR &v nS n][fTs:Fold Ts][Ys0:Args STAR &v nS n][usYs0:UFolds Ys0]
 [Fs|TY (J >> K) &v nS n][fFs:Fold Fs]
 [Yss:Args (J >> K) &v nS n][usYss:UFolds Yss]

   uncurry|STAR     fTs|Ys0 usYs0 ==> fTs
|| uncurry|(J >> K) fFs usYss ==>
     uncurry|K (fFs|([i:Fin (nS n)]Yss i (v0 J K)) (curry usYss.1))
            |([i:Fin (nS n)]aTl (Yss i)) usYss.2
];
```

Hence we may define

$$\begin{aligned}&\mathbf{fold}_X\ doCon\ doInl\ doInr\ doVoid\ doPair\ \mapsto\\ &\quad\mathbf{Curry}\ (\mathbf{ufold}\ doCon\ doInl\ doInr\ doVoid\ doPair\ X)\end{aligned}$$

```
[fold
   [K|Kind][X:TY K]
 = curry (ufold X)
 : Fold (X &c nS n)
];
```

Finally, we discharge the parameters `n`, `Phi` and the `do`'s, turning them into arguments of `fold`.

```
Discharge n;
```

8.4 Boolean Equality

$$\Phi^{=}\ S\ \boldsymbol{T}\ \mapsto\ [\![S]\!]_\delta \to \mathsf{Bool}$$

$$\begin{aligned}&doCon^{=}\ v\ \phi\,(\mathsf{con}\ x)\ \mapsto\ \phi\ x\\ &doInl^{=}\ \phi\,(\mathsf{inl}\ x)\ \mapsto\ \phi\ x\\ &doInl^{=}\ \phi\,(\mathsf{inr}\ y)\ \mapsto\ \mathsf{false}\\ &doInr^{=}\ \phi\,(\mathsf{inl}\ x)\ \mapsto\ \mathsf{false}\\ &doInr^{=}\ \phi\,(\mathsf{inr}\ y)\ \mapsto\ \phi\ y\\ &doVoid^{=}\ \mathsf{void}\ \mapsto\ \mathsf{true}\\ &doPair^{=}\ \phi_1\ \phi_2\,(\mathsf{pair}\ x\ y)\ \mapsto\ (\phi_1\ x)\wedge(\phi_2\ y)\end{aligned}$$

$$\text{let}\quad \frac{X\ :\ \mathsf{Ty}_\Delta\ K}{(==)_X\ :\ \mathbf{Fold}_K\ \Phi^{=}\ X\ X^0}$$

$$(==)_X\ \mapsto\ \mathbf{fold}_X\ doCon^{=}\ doInl^{=}\ doInr^{=}\ doVoid^{=}\ doPair^{=}$$

```
[Bool = One STAR + One STAR : TY STAR];

[True = inl|?|One void : Data Bool];
[False = inr|One void : Data Bool];

[PhiEq
```

```
   [Ts : TY STAR &v nS n0]
 = (Data (vHd Ts)) -> Data Bool
 : Type
];

$[conEq : {K|Kind}{d:Var Del K}{Wss|Args K &v nS n0}
          (PhiEq ((op<< (del d) &c nS n0) &a Wss))->
          PhiEq ((op@a (D STAR d) &c nS n0) &a Wss)];
[[K|Kind][d:Var Del K][Wss|Args K &v nS n0]
 [phi:PhiEq ((op<< (del d) &c nS n0) &a Wss)]
 [t:Data (del d << vHd Wss)]
   conEq d phi (con d t) ==> phi t
];

$[inlEq : {Ss,Ts|TY STAR &v nS n0}(PhiEq Ss)->
           PhiEq ((op+|STAR &c nS n0) &a Ss &a Ts)];
[[Ss,Ts|TY STAR &v nS n0][phi:PhiEq Ss][s:Data (vHd Ss)][t:Data (vHd Ts)]
   inlEq|Ss|Ts phi (inl s) ==> phi s
|| inlEq|Ss|Ts phi (inr t) ==> False
];

$[inrEq : {Ss,Ts|TY STAR &v nS n0}(PhiEq Ts)->
           PhiEq ((op+|STAR &c nS n0) &a Ss &a Ts)];
[[Ss,Ts|TY STAR &v nS n0][phi:PhiEq Ts][s:Data (vHd Ss)][t:Data (vHd Ts)]
   inrEq|Ss|Ts phi (inl s) ==> False
|| inrEq|Ss|Ts phi (inr t) ==> phi t
];

[voidEq
 = [_:Data One]True
 : PhiEq (One STAR &c nS n0)
];

$[and : {p,q:Data Bool}Data Bool];
[[b:Data Bool]
   and (inl|?|One void) b ==> b
|| and (inr|One void) b ==> False
];

$[pairEq : {Ss,Ts|TY STAR &v nS n0}(PhiEq Ss)->(PhiEq Ts)->
           PhiEq ((op*|STAR &c nS n0) &a Ss &a Ts)];
[[Ss,Ts|TY STAR &v nS n0][phis:PhiEq Ss][phit:PhiEq Ts]
 [s:Data (vHd Ss)][t:Data (vHd Ts)]
   pairEq|Ss|Ts phis phit (pair s t) ==> and (phis s) (phit t)
];

[eq = fold PhiEq conEq inlEq inrEq voidEq pairEq];
```

8.5 Map

$$\Phi^{\mathbf{map}}\, S\ \boldsymbol{T} \ \mapsto\ [\![\mathbf{hd}\ \boldsymbol{T}]\!]_\delta$$

$$
\begin{array}{l}
\mathit{doCon}^{\mathbf{map}}\ v\ \phi \ \mapsto\ \mathsf{con}\ \phi \\
\mathit{doInl}^{\mathbf{map}}\ \phi \ \mapsto\ \mathsf{inl}\ \phi \\
\mathit{doInr}^{\mathbf{map}}\ \phi \ \mapsto\ \mathsf{inr}\ \phi \\
\mathit{doVoid}^{\mathbf{map}} \ \mapsto\ \mathsf{void} \\
\mathit{doPair}^{\mathbf{map}}\ \phi_1\ \phi_2 \ \mapsto\ \mathsf{pair}\ \phi_1\ \phi_2
\end{array}
$$

$$
\text{let}\quad \frac{X\ :\ \mathsf{Ty}_\Delta\ K}{\mathbf{map}_X\ :\ \mathbf{Fold}_K\ \Phi^{\mathbf{map}}\ X\ X^1}
$$

$$
\begin{array}{l}
\mathbf{map}_X\ \mapsto\ \mathbf{fold}_X\ \mathit{doCon}^{\mathbf{map}}\ \mathit{doInl}^{\mathbf{map}}\ \mathit{doInr}^{\mathbf{map}} \\
\qquad\qquad\qquad \mathit{doVoid}^{\mathbf{map}}\ \mathit{doPair}^{\mathbf{map}}
\end{array}
$$

```
[PhiMap
   [Ts : TY STAR &v nS (nS n0)]
 = Data (vHd (vTl Ts))
 : Type
];

[conMap
   [K|Kind][d:Var Del K][Wss|Args K &v nS (nS n0)]
   [t:Data (del d << (vHd (vTl Wss)))]
 = con d t
 : Data (D STAR d @a (vHd (vTl Wss)))
];

[inlMap
   [Ss,Ts|TY STAR &v nS (nS n0)] = inl|(vHd (vTl Ss))|(vHd (vTl Ts))
];

[inrMap
   [Ss,Ts|TY STAR &v nS (nS n0)] = inr|(vHd (vTl Ss))|(vHd (vTl Ts))
];

[pairMap
   [Ss,Ts|TY STAR &v nS (nS n0)] = pair|(vHd (vTl Ss))|(vHd (vTl Ts))
];

[map = fold PhiMap conMap inlMap inrMap void pairMap];
```

9 Example Datatypes

Now that we have built the gadgets for parametric `Del` and `del`, let us at last discharge them, so they can be instantiated with examples.

```
Discharge Del;
```

Here, we just define natural numbers and lists. We build the constructors interactively:

```
[myDel = STAR >>              (* Nat *)
```

```
        (STAR >> STAR) >>    (* List *)
        STAR];

[myNat  [L:Sig] = D|myDel L v0      : Ty myDel L STAR];
[myList [L:Sig] = D|myDel L (vS v0) : Ty myDel L (STAR >> STAR)];

$[mydel : Env myDel];
[
|| mydel v0      ==> One ? STAR + myNat STAR
|| mydel (vS v0) ==> (One myDel (STAR >> STAR) + V ? v0 * myList / V ? v0
                     : Ty myDel (STAR >> STAR) STAR)
];

Goal my0 : Data mydel myNat;
  Refine con v0;
    Refine aNil;
    Refine inl;
      Refine void;
Save;

Goal myS : (Data mydel myNat) -> Data mydel myNat;
  intros n;
    Refine con v0;
      Refine aNil;
      Refine inr;
        Refine n;
Save;

Goal myNil : {X|TY myDel STAR}Data mydel (myList / X);
  intros X;
    Refine con|?|?|?|(aCons X aNil) (vS v0);
      Refine inl;
        Refine void;
Save;

Goal myCons : {X|TY myDel STAR}{x:Data mydel X}{xs:Data mydel (myList / X)}
              Data mydel (myList / X);
  intros X x xs;
    Refine con|?|?|?|(aCons X aNil) (vS v0);
      Refine inr;
        Refine pair;
          Refine x;
          Refine xs;
Save;
```

Now, a little experiment

```
[XXX =
   map mydel (myList STAR)
   |(vCons (myNat STAR) (vCons (myNat STAR) (vNil (Ty myDel STAR STAR))))
   myS
   (myCons my0 (myCons (myS my0) (myNil|(myNat STAR))))
];
```

```
[YYY = myCons (myS my0) (myCons (myS (myS my0)) myNil)];

[BBB = eq mydel (myList STAR)|(vCons (myNat STAR) (vNil (Ty myDel STAR STAR)))
       (eq mydel (myNat STAR))
       XXX YYY];

(Normal BBB);
```

And it's true!