

# Eliminators in Agda and in general,

by Mathijs Swint and Tom Lokhorst  
on the 2nd of October 2008 for the  
course Dependently Typed Programming.

Eliminators are like folds, but cooler.

```
foldList : forall {A} {B : Set}
          -> B
          -> (A -> B -> B)
          -> List A
          -> B
```

```
foldList n c [] = n
```

```
foldList n c (x :: xs) = c x (foldList n c xs)
```

```

elimVec : {A : Set} {n : ℕ}
  -> (m : {n : ℕ} -> Vec A n -> Set)
  -> m []
  -> (forall {n} {xs : Vec A n}
      -> (x : A)
      -> m xs
      -> m (x :: xs))
  -> (v : Vec A n)
  -> m v

```

```

elimVec m n c [] = n

```

```

elimVec m n c (x :: xs) = c x (elimVec m n c xs)

```

# Why would we want them?

- For the same reasons you'd want folds.
- But eliminators can work on dependent types.

When do we need them?

# When do we need them?

`replicateList : {A : Set} -> A -> ℕ -> List A`

# When do we need them?

```
replicateList : {A : Set} -> A -> ℕ -> List A  
replicateList x zero      = []
```

# When do we need them?

`replicateList` : {A : Set} -> A ->  $\mathbb{N}$  -> List A

`replicateList` x zero = []

`replicateList` x (suc n) = x :: replicateList x n

# When do we need them?

`replicateList` : {A : Set} -> A -> ℕ -> List A

`replicateList` x zero = []

`replicateList` x (suc n) = x :: replicateList x n

`replicateList'` : {A : Set} -> A -> ℕ -> List A

# When do we need them?

`replicateList` : {A : Set} -> A -> ℕ -> List A

`replicateList` x zero = []

`replicateList` x (suc n) = x :: replicateList x n

`replicateList'` : {A : Set} -> A -> ℕ -> List A

`replicateList'` x n = foldℕ [] (\ys -> x :: ys) n

# When do we need them?

```
replicateList : {A : Set} -> A -> ℕ -> List A
replicateList x zero      = []
replicateList x (suc n) = x :: replicateList x n
```

```
replicateList' : {A : Set} -> A -> ℕ -> List A
replicateList' x n = foldℕ [] (\ys -> x :: ys) n
```

```
replicateVec : {A : Set} -> A -> (n : ℕ) -> Vec A n
```

# When do we need them?

```
replicateList : {A : Set} -> A -> ℕ -> List A
replicateList x zero      = []
replicateList x (suc n) = x :: replicateList x n
```

```
replicateList' : {A : Set} -> A -> ℕ -> List A
replicateList' x n = foldℕ [] (\ys -> x :: ys) n
```

```
replicateVec : {A : Set} -> A -> (n : ℕ) -> Vec A n
replicateVec {A} x n = ?
```

$\text{fold}_{\mathbb{N}} : \text{forall } \{A : \text{Set}\}$

$\rightarrow A$

$\rightarrow (A \rightarrow A)$

$\rightarrow \mathbb{N}$

$\rightarrow A$

$\text{fold}_{\mathbb{N}} \ z \ s \ \text{zero} \quad = \ z$

$\text{fold}_{\mathbb{N}} \ z \ s \ (\text{suc } n) \ = \ s \ (\text{fold}_{\mathbb{N}} \ z \ s \ n)$

```
fold $\mathbb{N}$  : forall {A : Set}
```

```
    -> A
```

```
    -> (A -> A)
```

```
    ->  $\mathbb{N}$ 
```

```
    -> A
```

```
fold $\mathbb{N}$  z s zero      = z
```

```
fold $\mathbb{N}$  z s (suc n) = s (fold $\mathbb{N}$  z s n)
```

```
data Parity : Set where
```

```
  odd  : Parity
```

```
  even : Parity
```

```
flip : Parity -> Parity
```

```
flip even = odd
```

```
flip odd  = even
```

```
parity :  $\mathbb{N}$  -> Parity
```

```
parity = fold $\mathbb{N}$  even flip
```

$\text{fold}_{\mathbb{N}} : \text{forall } \{A : \text{Set}\}$

$\rightarrow A$

$\rightarrow (A \rightarrow A)$

$\rightarrow \mathbb{N}$

$\rightarrow A$

$\text{fold}_{\mathbb{N}} \ z \ s \ \text{zero} \quad = \ z$

$\text{fold}_{\mathbb{N}} \ z \ s \ (\text{suc } n) = s \ (\text{fold}_{\mathbb{N}} \ z \ s \ n)$

$\text{foldN} : \text{forall } \{A : \text{Set}\}$

$\rightarrow A$

$\rightarrow (A \rightarrow A)$

$\rightarrow \mathbb{N}$

$\rightarrow A$

$\text{foldN } z \ s \ \text{zero} = z$

$\text{foldN } z \ s \ (\text{suc } n) = s \ (\text{foldN } z \ s \ n)$

$\text{elimN} : (m : \mathbb{N} \rightarrow \text{Set})$

$\rightarrow m \ \text{zero}$

$\rightarrow (\text{forall } \{k\} \rightarrow m \ k \rightarrow m \ (\text{suc } k))$

$\rightarrow (n : \mathbb{N})$

$\rightarrow m \ n$

$\text{elimN } m \ z \ s \ \text{zero} = z$

$\text{elimN } m \ z \ s \ (\text{suc } n) = s \ (\text{elimN } m \ z \ s \ n)$

```
parity' : ℕ -> Parity
parity' = elimℕ (\_ -> Parity) even flip
```

```
elimℕ : (m : ℕ -> Set)
        -> m zero
        -> (forall {k} -> m k -> m (suc k))
        -> (n : ℕ)
        -> m n
```

```
elimℕ m z s zero = z
```

```
elimℕ m z s (suc n) = s (elimℕ m z s n)
```

$\text{replicateVec} : \{A : \text{Set}\} \rightarrow A \rightarrow (n : \mathbb{N}) \rightarrow \text{Vec } A \ n$

$\text{replicateVec } \{A\} \ x \ n = \text{elim}\mathbb{N} \ (\text{Vec } A)$

$\square$

$(\backslash ys \rightarrow x :: ys)$

$n$

$\text{elim}\mathbb{N} : (m : \mathbb{N} \rightarrow \text{Set})$

$\rightarrow m \ \text{zero}$

$\rightarrow (\text{forall1 } \{k\} \rightarrow m \ k \rightarrow m \ (\text{suc } k))$

$\rightarrow (n : \mathbb{N})$

$\rightarrow m \ n$

$\text{elim}\mathbb{N} \ m \ z \ s \ \text{zero} = z$

$\text{elim}\mathbb{N} \ m \ z \ s \ (\text{suc } n) = s \ (\text{elim}\mathbb{N} \ m \ z \ s \ n)$

```
data Fin : ℕ -> Set where
```

```
  fzero : {n : ℕ} -> Fin (suc n)
```

```
  fsuc   : {n : ℕ} -> Fin n -> Fin (suc n)
```

```
elimFin : {n : ℕ}
```

```
  -> (m : forall {i} -> Fin i -> Set)
```

```
  -> (forall {i} -> m (fzero{i}))
```

```
  -> (forall {i} -> {f : Fin i}
```

```
      -> m f
```

```
      -> m (fsuc f))
```

```
  -> (f : Fin n)
```

```
  -> m f
```

```
elimFin m fz fs fzero      = fz
```

```
elimFin m fz fs (fsuc f) = fs (elimFin m fz fs f)
```

lookup : forall {A n} -> Vec A n -> Fin n -> A

lookup [] ()

lookup (x :: xs) fzero = x

lookup (x :: xs) (fsuc f) = lookup xs f

lookup : forall {A n} -> Vec A n -> Fin n -> A

lookup {A} {zero} [] ()

lookup {A} {suc n} (x :: xs) fzero = x

lookup {A} {suc n} (x :: xs) (fsuc f) = lookup xs f

```
data Vec (A : Set) : ℕ -> Set where
```

```
  [] : Vec A zero
```

```
  _::_ : {n : ℕ} -> A -> Vec A n -> Vec A (suc n)
```

```

data Vec (A : Set) : ℕ -> Set where
  [] : Vec A zero
  _::_ : {n : ℕ} -> A -> Vec A n -> Vec A (suc n)

```

```

elimVec : {A : Set} {n : ℕ}
  -> (m : {n : ℕ} -> Vec A n -> Set)
  -> m []
  -> (forall {n} {xs : Vec A n}
      -> (x : A)
      -> m xs
      -> m (x :: xs))
  -> (v : Vec A n)
  -> m v

```

```

elimVec m n c [] = n

```

```

elimVec m n c (x :: xs) = c x (elimVec m n c xs)

```

lookup : forall {A n} -> Vec A n -> Fin n -> A

lookup {A} {zero} [] ()

lookup {A} {suc n} (x :: xs) fzero = x

lookup {A} {suc n} (x :: xs) (fsuc f) = lookup xs f

lookup' : forall {A n} -> Vec A n -> Fin n -> A

lookup' {A} v f =

elimVec ?

?

?

v

lookup : forall {A n} -> Vec A n -> Fin n -> A

lookup {A} {zero} [] ()

lookup {A} {suc n} (x :: xs) fzero = x

lookup {A} {suc n} (x :: xs) (fsuc f) = lookup xs f

lookup' : forall {A n} -> Vec A n -> Fin n -> A

lookup' {A} v f =

elimVec (\{n} \_ -> Fin n -> A)

?

?

v

f

lookup : forall {A n} -> Vec A n -> Fin n -> A

lookup {A} {zero} [] ()

lookup {A} {suc n} (x :: xs) fzero = x

lookup {A} {suc n} (x :: xs) (fsuc f) = lookup xs f

lookup' : forall {A n} -> Vec A n -> Fin n -> A

lookup' {A} v f =

elimVec (\{n} \_ -> Fin n -> A)

(\f -> absurd f)

?

v

f

`absurd : {A : Set} -> Fin 0 -> A`

`absurd f = ?`

`absurd : {A : Set} -> Fin 0 -> A`

`absurd f = ?`

`type : forall {n} -> Fin n -> Set`

`type {zero} _ = ⊥`

`type {suc n} _ = ⊤`

`absurd : {A : Set} -> Fin 0 -> A`

`absurd f = ?`

`type : forall {n} -> Fin n -> Set`

`type {zero} _ = ⊥`

`type {suc n} _ = ⊤`

`type' : forall {n} -> Fin n -> Set`

`type' {n} = elim $\mathbb{N}$ 1 (\n -> Fin n -> Set)`

`(\fz -> ⊥)`

`(\_ -> \fs -> ⊤)`

`n`

absurd : {A : Set} -> Fin 0 -> A

absurd f = ?

type : forall {n} -> Fin n -> Set

type {zero} \_ = ⊥

type {suc n} \_ = ⊤

type' : forall {n} -> Fin n -> Set

type' {n} = elim $\mathbb{N}$ 1 (\n -> Fin n -> Set)

(\fz -> ⊥)

(\\_ -> \fs -> ⊤)

n

elim $\mathbb{N}$  : (m :  $\mathbb{N}$  -> Set)

-> m zero

-> (forall {k} -> m k -> m (suc k))

-> (n :  $\mathbb{N}$ )

-> m n

absurd : {A : Set} -> Fin 0 -> A

absurd f = ?

type : forall {n} -> Fin n -> Set

type {zero} \_ = ⊥

type {suc n} \_ = ⊤

type' : forall {n} -> Fin n -> Set

type' {n} = elim $\mathbb{N}1$  (\n -> Fin n -> Set)

(\fz -> ⊥)

(\\_ -> \fs -> ⊤)

n

elim $\mathbb{N}1$  : (m :  $\mathbb{N}$  -> Set1)

-> m zero

-> (forall {k} -> m k -> m (suc k))

-> (n :  $\mathbb{N}$ )

-> m n

$\text{absurd} : \{A : \text{Set}\} \rightarrow \text{Fin } 0 \rightarrow A$

$\text{absurd } f = ?$

$\text{type} : \text{forall } \{n\} \rightarrow \text{Fin } n \rightarrow \text{Set}$

$\text{type } \{\text{zero}\} \_ = \perp$

$\text{type } \{\text{suc } n\} \_ = \top$

$\text{absurd} : \{A : \text{Set}\} \rightarrow \text{Fin } 0 \rightarrow A$

$\text{absurd } f = ?$

$\text{type} : \text{forall } \{n\} \rightarrow \text{Fin } n \rightarrow \text{Set}$

$\text{type } \{\text{zero}\} \_ = \perp$

$\text{type } \{\text{suc } n\} \_ = \top$

$\text{fin0} : \text{Fin } 0 \rightarrow \perp$

$\text{fin0} = ?$

absurd : {A : Set} -> Fin 0 -> A

absurd f = ?

type : forall {n} -> Fin n -> Set

type {zero} \_ = ⊥

type {suc n} \_ = ⊤

fin0 : Fin 0 -> ⊥

fin0 = elimFin (\f -> type f)  
          tt  
          (\\_ -> tt)

absurd : {A : Set} -> Fin 0 -> A

absurd f = ?

type : forall {n} -> Fin n -> Set

type {zero} \_ = ⊥

type {suc n} \_ = ⊤

fin0 : Fin 0 -> ⊥

fin0 = elimFin (\f -> type f)

tt

(\\_ -> tt)

botA : {A : Set} -> ⊥ -> A

botA {A} b = elim⊥ (\\_ -> A) b

```
absurd : {A : Set} -> Fin 0 -> A
absurd f = botA (fin0 f)
```

```
type : forall {n} -> Fin n -> Set
type {zero} _ = ⊥
type {suc n} _ = ⊤
```

```
fin0 : Fin 0 -> ⊥
fin0 = elimFin (\f -> type f)
      tt
      (\_ -> tt)
```

```
botA : {A : Set} -> ⊥ -> A
botA {A} b = elim⊥ (\_ -> A) b
```

```

lookup : forall {A n} -> Vec A n -> Fin n -> A
lookup {A} {zero} [] ()
lookup {A} {suc n} (x :: xs) fzero = x
lookup {A} {suc n} (x :: xs) (fsuc f) = lookup xs f

```

```

lookup' : forall {A n} -> Vec A n -> Fin n -> A
lookup' {A} v f =
  elimVec (\{n} _ -> Fin n -> A)
    (\f -> absurd f)
    ?
    v
    f

```

lookup : forall {A n} -> Vec A n -> Fin n -> A

lookup {A} {zero} [] ()

lookup {A} {suc n} (x :: xs) fzero = x

lookup {A} {suc n} (x :: xs) (fsuc f) = lookup xs f

lookup' : forall {A n} -> Vec A n -> Fin n -> A

lookup' {A} v f =

elimVec (\{n} \_ -> Fin n -> A)

(\f -> absurd f)

(\x fc -> (\f -> ?))

v

f

```
fin : forall {n} {A : Set}
      -> Fin (suc n)
      -> A
      -> (Fin n -> A)
      -> A
```

```
fin fzero      x fc = x
fin (fsuc f) x fc = fc f
```

```
lookup' : forall {A n} -> Vec A n -> Fin n -> A
lookup' {A} v f =
  elimVec (\{n} _ -> Fin n -> A)
        (\f -> absurd f)
        (\x fc -> (\f -> fin f x fc))
        v
        f
```

Done.

$\text{elim}_{\top} : (m : \top \rightarrow \text{Set})$

$\rightarrow m \text{ tt}$

$\rightarrow (t : \top)$

$\rightarrow m t$

$\text{elim}_{\top} m t \text{ tt} = t$

$\text{elim}_{\perp} : (m : \perp \rightarrow \text{Set})$

$\rightarrow (b : \perp)$

$\rightarrow m b$

$\text{elim}_{\perp} m ()$

$\text{elim}_{\equiv} : \{A : \text{Set}\} \{x y : A\}$

$\rightarrow (m : \{z : A\} \rightarrow x \equiv z \rightarrow \text{Set})$

$\rightarrow m \text{ refl}$

$\rightarrow (e : x \equiv y)$

$\rightarrow m e$

$\text{elim}_{\equiv} m r \text{ refl} = r$