# Rethinking Inheritance with Algebraic Ornaments

Dai

MIT

Formal Methods Seminar Series
April 2015

# Rethinking Inheritance

- Inheritance allows building more complex types from simpler ones
- In OO, functions are bundled with data, so we also extend functions

# Rethinking Inheritance

- Inheritance allows building more complex types from simpler ones
- In OO, functions are bundled with data, so we also extend functions
- A subtype $A$ of $B$ allows the substitution of $A$ whenever a $B$ is needed.

# Rethinking Inheritance

- Inheritance allows building more complex types from simpler ones
- In OO, functions are bundled with data, so we also extend functions
- A subtype $A$ of $B$ allows the substitution of $A$ whenever a $B$ is needed.

Inheritance is a construction, subtyping is a property of the type system!

# Simple Data

```
data FooBar = Foo Int Double | Bar String
```

---

```
datatype foobar = | Foo of (int,double)
                  | Bar of string
```

---

```
sealed abstract class FooBar
final case class Foo(foo1: Int, foo2: Double) exten
final case class Bar(bar1: String) extends FooBar
```

# Simple Data Cont.

```
struct foo {int foo1; double foo2;}
struct bar {char *bar1;}
union foo_bar_untagged {foo f; bar b;}
enum {FOO,BAR} foo_bar_tag;
struct foo_bar {foo_bar_tag tag; foo_bar_untagged v
```

# Algebraic Data

We'll focus on algebraic data for its nice properties and powerful type theory.

# Algebraic Data

We'll focus on algebraic data for its nice properties and powerful type theory.

- $\mathit{Void} \cong 0$
- $() \cong 1$
- Either $ab \cong a + b$
- $(a, b) \cong a * b$
- $(a \rightarrow b) \cong b^a$

# Algebraic Data

We'll focus on algebraic data for its nice properties and powerful type theory.

- $\mathit{Void} \cong 0$
- $() \cong 1$
- Either $ab \cong a + b$
- $(a, b) \cong a * b$
- $(a \to b) \cong b^a$

---

In fact, calculus, generating functionology, and nearly anything that works with complex number expressions works here.

# Algebraic Examples

- data Maybe a = Nothing | Just a $\cong 1 + a$

# Algebraic Examples

- data Maybe a = Nothing | Just a $\cong 1 + a$
- data Nat = Z | S Nat $\cong \mu x.1 + x$
- data Fix f = Fix (f (Fix f)) $\cong \mu x.fx$
- Nat $\cong$ Fix Maybe

# Algebraic Examples

- `data Maybe a = Nothing | Just a` $\cong 1 + a$
- `data Nat = Z | S Nat` $\cong \mu x.1 + x$
- `data Fix f = Fix (f (Fix f))` $\cong \mu x.fx$
- Nat $\cong$ Fix Maybe
- `data List a = Nil | Cons a (List a)` $\cong \mu x.1 + a * x$
- `data Cell a x = CNil | Cell a x` $\cong 1 + a * x$
- List $a \cong$ Fix (Cell a)

# Algebraic Examples

- data Maybe a = Nothing | Just a $\cong 1 + a$
- data Nat = Z | S Nat $\cong \mu x.1 + x$
- data Fix f = Fix (f (Fix f)) $\cong \mu x.fx$
- Nat $\cong$ Fix Maybe
- data List a = Nil | Cons a (List a) $\cong \mu x.1 + a * x$
- data Cell a x = CNil | Cell a x $\cong 1 + a * x$
- List $a \cong$ Fix (Cell a)

### Definition (Description)

Every recursive type is the fixed point of some “base” polynomial.

# An Extended Type Theory

$\hat{I}$, Types indexed by $I$:

$$(i : I) \vdash X_i \quad \text{or} \quad (i : I) \vdash f(i)$$

# An Extended Type Theory

$\hat{I}$, Types indexed by $I$:

$$(i : I) \vdash X_i \quad \text{or} \quad (i : I) \vdash f(i)$$

The dependent sum (pair):

$$\sum_{a:A} f(a) \quad \text{or} \quad (a : A, f(a))$$

# An Extended Type Theory

$\hat{I}$, Types indexed by $I$:

$$(i : I) \vdash X_i \quad \text{or} \quad (i : I) \vdash f(i)$$

The dependent sum (pair):

$$\sum_{a:A} f(a) \quad \text{or} \quad (a : A, f(a))$$

The dependent sum (pair):

$$\prod_{a:A} \quad \text{or} \quad (a : A) \to f(a)$$

# What's in a datatype?

```
data ListA_ x = Nil | Cons A x
type ListA = Fix ListA_
```

# What's in a datatype?

```
data ListA_ x = Nil | Cons A x
type ListA = Fix ListA_
```

The Description ListA_ is made of:

- A set of shapes/constructors $\int S : \texttt{Set}$

  $\{\text{Nil}\} \cup \{\text{Cons } a \mid a \in A\}$

# What's in a datatype?

```
data ListA_ x = Nil | Cons A x
type ListA = Fix ListA_
```

The Description ListA_ is made of:

- A set of shapes/constructors $\int S$ : Set

  $\{\text{Nil}\} \cup \{\text{Cons } a \mid a \in A\}$

- A function $\int S \xrightarrow{P} \texttt{Set}$ giving recursive positions to shapes

  $\text{Nil} \overset{P}{\mapsto} \emptyset$

  $\text{Cons } a \overset{P}{\mapsto} \{\bullet\}$

# What's in a datatype?

```
data ListA_ x = Nil | Cons A x
type ListA = Fix ListA_
```

The Description ListA_ is made of:

- A set of shapes/constructors $\int S : \mathtt{Set}$

  $\{\mathsf{Nil}\} \cup \{\mathsf{Cons}\ a \mid a \in A\}$

- A function $\int S \xrightarrow{P} \mathtt{Set}$ giving recursive positions to shapes

  $\mathsf{Nil} \overset{P}{\mapsto} \emptyset$
  $\mathsf{Cons}\ a \overset{P}{\mapsto} \{\bullet\}$

## Interpretations:

- A map of dependent types $X \mapsto (s : \int S, (p : P\ s) \to X)$
- A polynomial $\sum_{(s:\int S)} \prod_{(p:P\ s)} X$
- A forest of forks

# Indexed Data

Example (Packed Data)

data Packed a = Array (Array a) | Bytes ByteString

---

But *a* is free in Bytes :: ByteString $\to$ Packed a

# Indexed Data

**Example (Packed Data)**

data Packed a = Array (Array a) | Bytes ByteString

---

But *a* is free in Bytes :: ByteString → Packed a

We want to control the input and output index using *Generalized* ADT (GADTs):

```
data Packed a where
    Array :: Array a → Packed a
    Bytes :: ByteString → Packed Char
```

# Indexed Data

## Example (Packed Data)

data Packed a = Array (Array a) | Bytes ByteString

---

But *a* is free in Bytes :: ByteString → Packed a

We want to control the input and output index using *Generalized* ADT (GADTs):

```
data Packed a where
    Array :: Array a → Packed a
    Bytes :: ByteString → Packed Char
```

## Example (Length-indexed vectors)

```
data VecA :: Nat → * where
    VNil :: VecA Z
    VCons :: A → VecA n → VecA (S n)
```

# Indexed Data

Example (Packed Data)

data Packed a = Array (Array a) | Bytes ByteString

---

But *a* is free in Bytes :: ByteString → Packed a
We want to control the input and output index using *Generalized* ADT (GADTs):

```
data Packed a where
    Array :: Array a → Packed a
    Bytes :: ByteString → Packed Char
```

Example (Length-indexed vectors)

```
data VecA :: Nat → * where
    VNil :: VecA Z
    VCons :: A → VecA n → VecA (S n)
```

Best understood as labeled forks.

# What's in a datatype? Redux

### Definition (Description)

A Data Description $S \triangleleft^n P$ from from index set $I$ to $J$ is made of:

$S : J \to \texttt{Set}$, A family of shapes/constructors indexed by $J$

$P : \int S \to \texttt{Set}$, A family of positions indexed by shapes

$n : \int P \to I$, A next/recursive index for each position

Where $\int S = (j : J, S\ j) \quad \int P = (s : \int S, P\ s)$

# What's in a datatype? Redux

**Definition (Description)**

A Data Description $S \lhd^n P$ from from index set $I$ to $J$ is made of:

$S : J \to \mathtt{Set}$, A family of shapes/constructors indexed by $J$

$P : \int S \to \mathtt{Set}$, A family of positions indexed by shapes

$n : \int P \to I$, A next/recursive index for each position

Where $\int S = (j : J, S\ j) \quad \int P = (s : \int S, P\ s)$

$$I \xleftarrow{n} \int P \xrightarrow{P^{-1}} \int S \xrightarrow{S^{-1}} J$$

# Interpretations

$$\hat{I} \xrightarrow{\Delta_n} \int^{\hat{}} P \xrightarrow{\Pi_P} \int^{\hat{}} S \xrightarrow{\Sigma_S} \hat{J}$$

$$\frac{(i : I) \vdash X_i}{(j : J) \vdash (s : S\ j, (p : P\ s) \to X_{n(p)})} \Sigma_S \Pi_P \Delta_n$$

# Interpretations

$$\hat{I} \xrightarrow{\Delta_n} \int^{\hat{}} P \xrightarrow{\Pi_P} \int^{\hat{}} S \xrightarrow{\Sigma_S} \hat{J}$$

$$\frac{(i : I) \vdash X_i}{(j : J) \vdash (s : S\ j, (p : P\ s) \to X_{n(p)})}\Sigma_S \Pi_P \Delta_n$$

---

$$\frac{(i : I) \vdash X_i}{(p : \int P) \vdash X_{n(p)}}\Delta_n$$

$$\frac{(s : \int S) \vdash X_s}{(j : J) \vdash (s : S\ j, X_s)}\Sigma_S \qquad \frac{(p : \int P) \vdash X_p}{(s : \int S) \vdash (p : P\ s) \to X_p}\Pi_P$$

# Example: Constant Maybe

$$\texttt{data}\ \mathsf{MaybeA} = \texttt{Nothing}\ \mid\ \texttt{Just}\ \mathsf{A}$$

# Example: Constant Maybe

```
data MaybeA = Nothing | Just A
```

$1 \xrightarrow{Shape} \mathtt{Set}$

$\bullet \mapsto$

$\{Nothing, Just\ a\}$

$\int Shape \xrightarrow{Pos} \mathtt{Set}$

$s \mapsto \emptyset$

$\int Pos \xrightarrow{next} 1$

$p \mapsto 1$

# Example: Constant Maybe

```
data MaybeA = Nothing | Just A
```

$1 \xrightarrow{Shape} \mathtt{Set}$

$\bullet \mapsto$

$\{Nothing, Just\ a\}$

$\int Shape \xrightarrow{Pos} \mathtt{Set}$

$s \mapsto \emptyset$

$\int Pos \xrightarrow{next} 1$

$p \mapsto 1$

$$1 \longleftarrow \emptyset\bot \xrightarrow{\ \bot\ } 1 + A \xrightarrow{\ !\ } 1$$

# Example: Constant Maybe

```
data MaybeA = Nothing | Just A
```

$1 \xrightarrow{Shape} \mathtt{Set}$
$\bullet \mapsto$
$\{Nothing, Just\ a\}$

$\int Shape \xrightarrow{Pos} \mathtt{Set}$
$s \mapsto \emptyset$

$\int Pos \xrightarrow{next} 1$
$p \mapsto 1$

$$1 \longleftarrow \emptyset\perp \xrightarrow{\perp} 1 + A \xrightarrow{!} 1$$

$$\frac{\bullet : 1 \vdash X}{\bullet : 1 \vdash (Nothing, \bullet) + (Just\ a_1, \bullet) + (Just\ a_2, \bullet) + \ldots}$$

# Example: Maybe

```
data Maybe a = Nothing | Just a   ???
```

# Example: Maybe

```
data Maybe a = Nothing | Just a   ???
```

$$\mathtt{Type} \xrightarrow{Shape} \mathtt{Set}$$
$$a \mapsto \{Nothing, Just\}$$

$$\int Shape \xrightarrow{Pos} \mathtt{Set}$$
$$(a, Nothing) \mapsto \emptyset$$
$$(\text{a,Just}) \mapsto \{\bullet\}$$

$$\int Pos \xrightarrow{next} \mathtt{Type}$$
$$(a, Just, \bullet) \mapsto a$$

# Example: Maybe

```
data Maybe a = Nothing | Just a   ???
```

$$\mathtt{Type} \xrightarrow{Shape} \mathtt{Set}$$

$$a \mapsto \{Nothing, Just\}$$

$$\textstyle\int Shape \xrightarrow{Pos} \mathtt{Set}$$

$$(a, Nothing) \mapsto \emptyset$$

$$(\mathrm{a{,}Just}) \mapsto \{\bullet\}$$

$$\textstyle\int Pos \xrightarrow{next} \mathtt{Type}$$

$$(a, Just, \bullet) \mapsto a$$

$$\mathtt{Type} = \mathtt{Type} \hookrightarrow 2 * \mathtt{Type} \xrightarrow{\pi_2} \mathtt{Type}$$

# Example: Maybe

```
data Maybe a = Nothing | Just a    ???
```

$$\mathtt{Type} \xrightarrow{Shape} \mathtt{Set}$$

$$a \mapsto \{Nothing, Just\}$$

$$\int Shape \xrightarrow{Pos} \mathtt{Set}$$

$$(a, Nothing) \mapsto \emptyset$$

$$(\mathrm{a,Just}) \mapsto \{\bullet\}$$

$$\int Pos \xrightarrow{next} \mathtt{Type}$$

$$(a, Just, \bullet) \mapsto a$$

$$\mathtt{Type} =\!=\!= \mathtt{Type} \hookrightarrow 2 * \mathtt{Type} \xrightarrow{\pi_2} \mathtt{Type}$$

$$\frac{a : \mathtt{Type} \vdash X_a}{a : \mathtt{Type} \vdash (Nothing, \bullet) + (Just, X_a)}$$

# Example: Maybe

```
data Maybe a = Nothing | Just a   ???
```

$$\mathtt{Type} \xrightarrow{\mathit{Shape}} \mathtt{Set}$$
$$a \mapsto \{\mathit{Nothing}, \mathit{Just}\}$$

$$\textstyle\int \mathit{Shape} \xrightarrow{\mathit{Pos}} \mathtt{Set}$$
$$(a, \mathit{Nothing}) \mapsto \emptyset$$
$$(\mathrm{a}, \mathrm{Just}) \mapsto \{\bullet\}$$

$$\textstyle\int \mathit{Pos} \xrightarrow{\mathit{next}} \mathtt{Type}$$
$$(a, \mathit{Just}, \bullet) \mapsto a$$

$$\mathtt{Type} = \mathtt{Type} \hookrightarrow 2 * \mathtt{Type} \xrightarrow{\pi_2} \mathtt{Type}$$

$$\frac{a : \mathtt{Type} \vdash X_a}{a : \mathtt{Type} \vdash (\mathit{Nothing}, \bullet) + (\mathit{Just}, X_a)}$$

Not what we expect!

data Maybe2 (f :: $* \to *$) a = Nothing | Just (f a)

# Example: Polymorphic Lists

```
data List_ a x where
    Nil  :: List_ a x
    Cons :: a → x a → List_ a x
```

# Example: Polymorphic Lists

```
data List_ a x where
    Nil :: List_ a x
    Cons :: a → x a → List_ a x
```

$\mathtt{Type} \xrightarrow{\mathit{Shape}} \mathtt{Set}$

$t \mapsto \{\text{Nil}\ , \text{Cons}\ (a :: t)\}$

$\int \mathit{Shape} \xrightarrow{\mathit{Pos}} \mathtt{Set}$

$(t, \text{Nil}\ ) \mapsto \emptyset$

$(\text{t,Cons (a :: t)}) \mapsto \{\bullet\}$

$\int \mathit{Pos} \xrightarrow{\mathit{next}} \mathbb{N}$

$(t, \text{Cons}\ (a : t), \bullet) \mapsto t$

# Example: Polymorphic Lists

```
data List_ a x where
    Nil  :: List_ a x
    Cons :: a → x a → List_ a x
```

$\mathtt{Type} \xrightarrow{\mathit{Shape}} \mathtt{Set}$

$t \mapsto \{\mathrm{Nil}\ , \mathrm{Cons}\ (a :: t)\}$

$\int \mathit{Shape} \xrightarrow{\mathit{Pos}} \mathtt{Set}$

$(t, \mathrm{Nil}\ ) \mapsto \emptyset$

$(\mathrm{t,Cons}\ (\mathrm{a} :: \mathrm{t}))$
$\mapsto \{\bullet\}$

$\int \mathit{Pos} \xrightarrow{\mathit{next}} \mathbb{N}$

$(t, \mathrm{Cons}\ (a : t), \bullet) \mapsto t$

$$\mathtt{Type} \xleftarrow{\pi_1} \mathtt{Type} * a \hookrightarrow 1 + \mathtt{Type} * (1 + a) \xrightarrow{\pi_1} \mathtt{Type}$$

# Example: Polymorphic Lists

```
data List_ a x where
    Nil :: List_ a x
    Cons :: a → x a → List_ a x
```

$$\mathrm{Type} \xrightarrow{Shape} \mathrm{Set}$$

$t \mapsto \{\mathrm{Nil}\ , \mathrm{Cons}\ (a :: t)\}$

$$\int Shape \xrightarrow{Pos} \mathrm{Set}$$

$(t, \mathrm{Nil}\ ) \mapsto \emptyset$

$(\mathrm{t,Cons}\ (\mathrm{a} :: \mathrm{t}))$

$\mapsto \{\bullet\}$

$$\int Pos \xrightarrow{next} \mathbb{N}$$

$(t, \mathrm{Cons}\ (a : t), \bullet) \mapsto t$

$$\mathtt{Type} \xleftarrow{\pi_1} \mathtt{Type} * a \hookrightarrow 1 + \mathtt{Type} * (1 + a) \xrightarrow{\pi_1} \mathtt{Type}$$

$$\frac{t : \mathtt{Type} \vdash X_t}{t : \mathtt{Type} \vdash (\mathrm{Nil}\ , \bullet) + (Cons(a :: t), \bullet \to X_t)}$$

# Example: Monomorphic Vectors

```
data AVec_ n x where
    VNilA  :: AVec_ 0 x
    VConsA :: A → x n → AVec_ (n+1) x
```

# Example: Monomorphic Vectors

```
data AVec_ n x where
    VNilA  :: AVec_ 0 x
    VConsA :: A → x n → AVec_ (n+1) x
```

$\mathbb{N} \xrightarrow{Shape} \texttt{Set}$

$A \mapsto \{VNilA\}$

$\mathrm{S\ n} \mapsto \mathrm{Cons}\ a$

$\int Shape \xrightarrow{Pos} \texttt{Set}$

$(Z, VNilA) \mapsto \emptyset$

$(\mathrm{S\ n}, \mathrm{VConsA\ a})$

$\mapsto \{\bullet\}$

$\int Pos \xrightarrow{next} \mathbb{N}$

$(S\ \ n, VConsAa, \bullet) \mapsto$

$n$

# Example: Monomorphic Vectors

```
data AVec_ n x where
  VNilA  :: AVec_ 0 x
  VConsA :: A → x n → AVec_ (n+1) x
```

$\mathbb{N} \xrightarrow{\mathit{Shape}} \mathtt{Set}$
$A \mapsto \{\mathit{VNilA}\}$
S n $\mapsto$ Cons $a$

$\int \mathit{Shape} \xrightarrow{\mathit{Pos}} \mathtt{Set}$
$(Z, \mathit{VNilA}) \mapsto \emptyset$
(S n,VConsA a)
$\mapsto \{\bullet\}$

$\int \mathit{Pos} \xrightarrow{\mathit{next}} \mathbb{N}$
$(S\ \ n, \mathit{VConsAa}, \bullet) \mapsto$
$n$

$$\mathbb{N} \xleftarrow[\pi_2]{} A * \mathbb{N}^+ \hookrightarrow 1 + A * \mathbb{N}^+ \xrightarrow{Z\nabla \mathit{snd}} \mathbb{N}$$

# Example: Monomorphic Vectors

```
data AVec_ n x where
    VNilA  :: AVec_ 0 x
    VConsA :: A → x n → AVec_ (n+1) x
```

$\mathbb{N} \xrightarrow{Shape} \mathtt{Set}$

$A \mapsto \{VNilA\}$

$\mathrm{S\ n} \mapsto \mathrm{Cons}\ a$

$\int Shape \xrightarrow{Pos} \mathtt{Set}$

$(Z, VNilA) \mapsto \emptyset$

$(\mathrm{S\ n}, \mathrm{VConsA\ a})$

$\mapsto \{\bullet\}$

$\int Pos \xrightarrow{next} \mathbb{N}$

$(S\ \ n, VConsAa, \bullet) \mapsto$

$n$

$$\mathbb{N} \xleftarrow[\pi_2]{} A * \mathbb{N}^+ \hookrightarrow 1 + A * \mathbb{N}^+ \xrightarrow{Z \nabla snd} \mathbb{N}$$

$$\frac{n : \mathbb{N} \vdash X_n}{n : \mathbb{N} \vdash (c : Shape(n), p : Pos(c) \to X_{n-1})}$$

# Ornaments

An Ornament from $(S \triangleleft^{n} P)$ to $(S' \triangleleft^{n'} P')$ is a Morphism of Containers:

$\alpha \overset{v}{\underset{u}{\blacktriangleleft}} \omega : (S' \triangleleft^{n'} P') \overset{v}{\underset{u}{\Rrightarrow}} (S \triangleleft^{n} P)$:

$$\begin{tikzcd}
K \arrow[d, "u"] & \int P' \arrow[l, "n'"'] \arrow[r, "P'^{-1}"] & \int S' \arrow[r, "S'^{-1}"] \arrow[dd, "\alpha"] & L \arrow[dd, "v"] \\
 & \int P \times_{\int S} \int S' \arrow[u, "\omega"] \arrow[ur, "\pi_2"'] \arrow[d, "\pi_1"] & & \\
I & \int P \arrow[l, "n"] \arrow[r, "P^{-1}"'] & \int S \arrow[r, "S^{-1}"'] & J
\end{tikzcd}$$

Explicitly:

$\alpha$ : (l : L,S' l) $\rightarrow$ S (v l)

$\omega$ : (sh' : $\int$ S', P (A sh')) $\rightarrow$ P' sh'

q : (l : L, sh' : S' l, pos : P (A sh')) $\rightarrow$ u (n'($\omega$ pos)) $=$ n pos

# Lists from Naturals

```
data Nat = Z | S Nat
```

$\Downarrow$

```
data ListA = Nil | Cons A
```

# Vectors from Lists

```
data ListA = Nil | Cons A
⇓
data VecA :: Nat → * where
    VNil :: VecA Z
    VCons :: A → VecA n → VecA (S n)
```

# Red-Black Trees from Trees

```
data Tree = Leaf | Branch Tree Tree
⇓
data RB = R | B
data RBTreeA :: RB → * where
Leaf :: RBTree rb
RBranch :: RBTreeA B → A → RBTreeA B → RBTreeA R
BBranch :: RBTreeA R → A → RBTreeA R → RBTreeA B
```

# Singleton Ornaments

```
data Nat = Z | S
⇓
data SNat :: Nat → * where
    SZ :: SNat Z
    SS :: (n :: Nat) → SNat (S n)
```

# Combining Ornaments

## Definition

The Parallel Composition of ornaments $A \overset{F}{\Rightarrow} B$ and $A \xrightarrow{C}$ is a new ornament $A \xrightarrow{F \otimes G} B \times_A C$: the most general unifier of both enhancements

# Combining Ornaments

### Definition

The Parallel Composition of ornaments $A \stackrel{F}{\Rightarrow} B$ and $A \xrightarrow{C}$ is a new ornament $A \xrightarrow{F \otimes G} B \times_A C$: the most general unifier of both enhancements

### Example (Vectors)

$(\text{List} \Rightarrow \text{Vector}) \cong (\text{Singleton}_{\mathbb{N}}) \otimes (\mathbb{N} \Rightarrow \text{List})$

# Combining Ornaments

**Definition**

The Parallel Composition of ornaments $A \overset{F}{\Rightarrow} B$ and $A \xrightarrow{C}$ is a new ornament $A \xrightarrow{F \otimes G} B \times_A C$: the most general unifier of both enhancements

**Example (Vectors)**

$(\text{List} \Rightarrow \text{Vector}) \cong (\text{Singleton}_{\mathbb{N}}) \otimes (\mathbb{N} \Rightarrow \text{List})$

**Definition**

The Optimized Predicate of an ornament $A \overset{F}{\Rightarrow} B$ is the parallel composition $F \otimes$ Singleton

**Example (Optimized Maybe)**

```
data IMaybeA :: Bool → * where
    INothing :: IMaybeA False
    IJust :: A → IMaybeA True
```

# Re-Rethinking Inheritance

- Inheritance allows code reuse by extending data and functions
- Subtyping $A < B$ allows the complete substitution of $A$ for whenever a $B$ is needed. *All methods defined on $B$ are defined on $A$*

# Re-Rethinking Inheritance

- Inheritance allows code reuse by extending data and functions
- Subtyping $A < B$ allows the complete substitution of $A$ for whenever a $B$ is needed. *All methods defined on $B$ are defined on $A$*

How to generalize to a functional setting?

# Transporting Functions Across Ornaments

Notice the similarity:

$\mathbb{N} + \mathbb{N} : \mathbb{N}$
$\{Z, \bullet\} + m \mapsto m$
$\{\text{Suc, n}\} + \text{m} \mapsto \{Suc, \lambda \bullet .m(\bullet) + n(\bullet)\}$

$List\ t +\!\!+ List\ t : List\ t$
$\{Nil_t, \bullet\} +\!\!+ ys \mapsto ys$
$\{\text{Cons (a :: t), xs}\} +\!\!+ \text{ys} \mapsto \{Cons\ (a :: t), \lambda \bullet .xs(\bullet) +\!\!+ ys(\bullet)\}$

# Transporting Functions Across Ornaments

Notice the similarity:

$\mathbb{N} + \mathbb{N} : \mathbb{N}$
$\{Z, \bullet\} + m \mapsto m$
$\{\text{Suc, n}\} + \text{m} \mapsto \{Suc, \lambda \bullet .m(\bullet) + n(\bullet)\}$

$List\ t +\!\!+\ List\ t : List\ t$
$\{Nil_t, \bullet\} +\!\!+\ ys \mapsto ys$
$\{\text{Cons (a :: t), xs}\} +\!\!+ \text{ ys} \mapsto \{Cons\ (a :: t), \lambda \bullet .xs(\bullet) +\!\!+\ ys(\bullet)\}$

Look at what happens to the trees.

# Indexed Transport

```
data HList (ts :: [*]) where
    HNil :: HList []
    HCons :: t → HList ts → HList (t:ts)

reverse :: List a → List a
reverse Nil = Nil
reverse (Cons a as) = reverse as ++ (Cons a Nil)

hReverse :: HList xs → HList (reverse xs)
hReverse HNil = HNil
hReverse (HCons a as) = hReverse as ++ (HCons a HNi
```

# Coherence Concerns

```
(<) :: ℕ → ℕ → Bool
n < Z = False
Z < S m = True
S n < m = n < m
lookup :: ℕ → ListA → MaybeA
lookup n Nil = Nothing
lookup Z (Cons a xs) = Just a
lookup (S n) xs = lookup n xs
```

# Coherence Concerns

```
(<) :: ℕ → ℕ → Bool
n < Z = False
Z < S m = True
S n < m = n < m
lookup :: ℕ → ListA → MaybeA
lookup n Nil = Nothing
lookup Z (Cons a xs) = Just a
lookup (S n) xs = lookup n xs
```

Coherence: isJust . lookup n == (n <) . length

# Coherence Concerns

```
(<) :: ℕ → ℕ → Bool
n < Z = False
Z < S m = True
S n < m = n < m
lookup :: ℕ → ListA → MaybeA
lookup n Nil = Nothing
lookup Z (Cons a xs) = Just a
lookup (S n) xs = lookup n xs
```

Coherence: isJust . lookup n == (n <) . length

```
-- The optimized predicate MaybeA ⊗ Singleton_Bool
data IMaybeA :: Bool → * where
     INothing :: IMaybeA False
     IJust :: A → IMaybeA True
```

Lift lookup to opimized predicates VecA and IMaybeA, with (¡) on indicies. Coherence for free!

# Recap

- Polynomials allow first class data representation with nice algebraic properties
- Ornaments let us build more complex types from simpler ones, and allows *ad-hoc* extension
- Transport of functions across ornaments allow inheritance
- Coherent liftings allow subtypeing

# Questions?

# Categories

Categories capture the essence of composition and modularity.

# Categories

Categories capture the essence of composition and modularity.

## Definition (Category)

A category $C$ has:

- A collection of objects $c : C$
- A collection of morphisms (arrows) between (indexed by) pairs of objects $c \xrightarrow{f} c'$
- Arrows compose: For every pair of arrows $a \xrightarrow{f} b \xrightarrow{g} c$, their composition $a \xrightarrow{g \circ f} c$
- Every object $a$ has an identity arrow $a \xrightarrow{1_a} a$

# Why Categories?

# Universal Constructions

Universal objects are the "most general" of its kind, and are "Unique up to unique isomorphism" The action of any other object is determined by factoring through the universal one.

# Universal Constructions

Universal objects are the "most general" of its kind, and are "Unique up to unique isomorphism" The action of any other object is determined by factoring through the universal one.

Example ((Co)Universal Constructions)

- Products

# Universal Constructions

Universal objects are the "most general" of its kind, and are "Unique up to unique isomorphism" The action of any other object is determined by factoring through the universal one.

Example ((Co)Universal Constructions)

- Products
- Coproducts (Sums)

# Universal Constructions

Universal objects are the "most general" of its kind, and are "Unique up to unique isomorphism" The action of any other object is determined by factoring through the universal one.

## Example ((Co)Universal Constructions)

- Products
- Coproducts (Sums)
- Pullbacks (Fiber Products)

# Universal Constructions

Universal objects are the "most general" of its kind, and are "Unique up to unique isomorphism" The action of any other object is determined by factoring through the universal one.

### Example ((Co)Universal Constructions)

- Products
- Coproducts (Sums)
- Pullbacks (Fiber Products)
- Initial/Terminal Objects

# Universal Constructions

Universal objects are the "most general" of its kind, and are "Unique up to unique isomorphism" The action of any other object is determined by factoring through the universal one.

Example ((Co)Universal Constructions)

- Products
- Coproducts (Sums)
- Pullbacks (Fiber Products)
- Initial/Terminal Objects

Universal properties allow encapsulation: Even if the object construction is messy (or unknown), its interaction is completely determined by the universal property. They are a bridge between abstract interfaces and concrete representations.

# Functors

- Functors are arrows between categories.
- $A \xrightarrow{F} B$ sends objects $a : A$ to objects $b : B$, and arrows $a \to a'$ to arrows $F(a) \to F(a')$
- "Functors" in Haskell are actually functors **Hask** $\to$ **Hask**

# Functors

- Functors are arrows between categories.
- $A \xrightarrow{F} B$ sends objects $a : A$ to objects $b : B$, and arrows $a \to a'$ to arrows $F(a) \to F(a')$
- "Functors" in Haskell are actually functors **Hask** $\to$ **Hask**

## Example

data FunBox a = F a deriving (Functor)

- The constructor (F :: a → Funbox a) is the object component of the functor
- fmap :: (a → b) → (Funbox a → Funbox b) is the arrow component of the functor

# (Co)Limits

## Definition (Diagrams)

A *J-shaped diagram* in a category $C$ is any functor $J \xrightarrow{F} C$

We draw them as collection of objects and arrows in $C$, leaving $J$ implicit, because only the shape matters.

# (Co)Limits

## Definition (Diagrams)

A *J-shaped diagram* in a category $C$ is any functor $J \xrightarrow{F} C$

We draw them as collection of objects and arrows in $C$, leaving $J$ implicit, because only the shape matters.

## Definition ((Co)Limit)

The Limit of a diagram (functor) in $C$ is a universal object $\mathrm{Lim}\, F : C$ with a unique arrow to every object in the diagram.

The interaction of the composition law and universality forces path equivalence

# Encoding Polynomials Categorically

$\mathsf{Packed}_a \cong \mathsf{Array}_a + \mathit{ByteString}$

$\texttt{Type} * \mathbb{N} \leftarrow B \rightarrow A \rightarrow \texttt{Type} * \mathbb{N}$

# Encoding Polynomials Categorically

$\mathsf{Packed}_a \cong \mathsf{Array}_a + \mathit{ByteString}$

$\mathtt{Type} * \mathbb{N} \leftarrow B \rightarrow A \rightarrow \mathtt{Type} * \mathbb{N}$

$$\mathtt{Set}/I \xrightarrow{\Delta_s} \mathtt{Set}/B \xrightarrow{\Pi_f} \mathtt{Set}/A \xrightarrow{\Sigma_t} \mathtt{Set}/J$$

$$\hat{I} \xrightarrow{\Delta_s} \hat{B} \xrightarrow{\Pi_f} \hat{A} \xrightarrow{\Sigma_t} \hat{J}$$

$$\begin{tikzcd}
& B \arrow[rr, "f"] \arrow[dl, "s"'] \arrow[dr, "X\circ s"'] \arrow[rr, phantom, yshift=-5ex, "\overset{\mathrm{Rex}_f(X\circ s)}{\Longrightarrow}"] & & A \arrow[dl] \arrow[dr, "t"] & \\
I \arrow[rr, "X"'] & & \mathtt{Set} & & J \arrow[ll, "\mathrm{Lex}_t(\mathrm{Rex}_f(X\circ s))"]
\end{tikzcd}$$

Three interpretations:

- $I, J$ are the type indicies, variable subscripts/letters, or incoming/outgoing branch labels
- $A$ are the constructor names, sum subscript, or outgoing edges.
- $B$ are the recursive positions, product subscript, or incoming

# Encoding Polynomials Categorically

$\mathsf{Packed}_a \cong \mathsf{Array}_a + \mathit{ByteString}$

$\mathtt{Type} * \mathbb{N} \leftarrow B \rightarrow A \rightarrow \mathtt{Type} * \mathbb{N}$

$$\mathtt{Set}/I \xrightarrow{\Delta_s} \mathtt{Set}/B \xrightarrow{\Pi_f} \mathtt{Set}/A \xrightarrow{\Sigma_t} \mathtt{Set}/J$$

$$\hat{I} \xrightarrow{\Delta_s} \hat{B} \xrightarrow{\Pi_f} \hat{A} \xrightarrow{\Sigma_t} \hat{J}$$

$$\begin{tikzcd}
B \arrow[rr, "f"] \arrow[d, "s"'] \arrow[dr, "X \circ s"'] & & A \arrow[dl] \arrow[d, "t"] \\
I \arrow[r, "X"'] & \mathtt{Set} & J \arrow[l, "\mathsf{Lex}_t(\mathsf{Rex}_f(X \circ s))"]
\end{tikzcd}$$

$\mathsf{Rex}_f(X \circ s)$

Three interpretations:

- $I, J$ are the type indicies, variable subscripts/letters, or incoming/outgoing branch labels
- $A$ are the constructor names, sum subscript, or outgoing edges.
- $B$ are the recursive positions, product subscript, or incoming

# Example: Rank-2 Maybe

data Maybe2 a (f :: $* \to *$) = Z | S (f a)

# Example: Rank-2 Maybe

```
data Maybe2 a (f :: * -> *) = Z | S (f a)
```

$$\mathtt{Type} \xrightarrow{Shape} \mathtt{Set}$$

$$a \mapsto \{\mathrm{Z}, \mathrm{S}\}$$

$$\int Shape \xrightarrow{Pos} \mathtt{Set}$$

$$(a, Z) \mapsto \emptyset$$

$$(\mathrm{a},\ \mathrm{S}) \mapsto \{\bullet\}$$

$$\int Pos \xrightarrow{next} \mathtt{Type}$$

$$(a, S, \bullet) \mapsto a$$

# Example: Rank-2 Maybe

```
data Maybe2 a (f :: * → *) = Z | S (f a)
```

$\mathtt{Type} \xrightarrow{Shape} \mathtt{Set}$

$a \mapsto \{\mathsf{Z}, \mathsf{S}\}$

$\int Shape \xrightarrow{Pos} \mathtt{Set}$

$(a, Z) \mapsto \emptyset$

$(\mathsf{a}, \mathsf{S}) \mapsto \{\bullet\}$

$\int Pos \xrightarrow{next} \mathtt{Type}$

$(a, S, \bullet) \mapsto a$

$$\begin{tikzcd}
X \arrow[r, equal] \arrow[d, "x"] & X \arrow[d, "x"] & \mathtt{Type} + X \arrow[d, "Id+x"'] \arrow[dr, "Id \nabla x"] & \\
\mathtt{Type} \arrow[r, equal] & \mathtt{Type} \arrow[r, "inR"] & \mathtt{Type} * 2 \arrow[r, "Id \nabla Id"] & \mathtt{Type}
\end{tikzcd}$$

# Example: Rank-2 Maybe

```
data Maybe2 a (f :: * → *) = Z | S (f a)
```

$\mathtt{Type} \xrightarrow{Shape} \mathtt{Set}$

$a \mapsto \{\mathsf{Z}, \mathsf{S}\}$

$\int Shape \xrightarrow{Pos} \mathtt{Set}$

$(a, Z) \mapsto \emptyset$

$(\mathrm{a},\ \mathrm{S}) \mapsto \{\bullet\}$

$\int Pos \xrightarrow{next} \mathtt{Type}$

$(a, S, \bullet) \mapsto a$

$$\begin{tikzcd}
X \arrow[r, equal] \arrow[d, "x"] & X \arrow[d, "x"] \arrow[dl, phantom, "\llcorner", very near start] & \mathtt{Type} + X \arrow[d, "Id+x"'] \arrow[dr, "Id\nabla x"] & \\
\mathtt{Type} \arrow[r, equal] & \mathtt{Type} \arrow[r, "inR"] & \mathtt{Type} * 2 \arrow[r, "Id\nabla Id"] & \mathtt{Type}
\end{tikzcd}$$

$\Pi_{inR} X \equiv$

(a,b) : $\mathtt{Type}$ * Bool $\vdash i : \mathrm{inL}^{-1}(a, b) \to X(a)$

$\cong\ Type + X$

# Example: Monomorphic Lists

$\textbf{data}\ \mathsf{AList}_{\_}\ \mathsf{x} = \mathsf{Z} \mid SA_1\ \mathsf{x} \mid SA_2\ \mathsf{x} \mid \ldots \cong$
$\textbf{data}\ \mathsf{AList}_{\_}\ \mathsf{x} = \mathsf{Z} \mid \mathsf{S}\ \mathsf{A}\ \mathsf{x}$

# Example: Monomorphic Lists

$$\texttt{data}\ \mathsf{AList\_}\ \mathsf{x} = \mathsf{Z} \mid SA_1\ \mathsf{x} \mid SA_2\ \mathsf{x} \mid \ldots \cong$$
$$\texttt{data}\ \mathsf{AList\_}\ \mathsf{x} = \mathsf{Z} \mid \mathsf{S}\ \mathsf{A}\ \mathsf{x}$$

$1 \xrightarrow{Shape} \mathtt{Set}$

$\bullet \mapsto \{\mathsf{Z}, \mathsf{S}\}$

$\int Shape \xrightarrow{Pos} \mathtt{Set}$

$(\bullet, Z) \mapsto \emptyset$

$(\bullet, Sa) \mapsto \{\bullet\}$

$\int Pos \xrightarrow{next} 1$

$(\bullet, Sa, \bullet) \mapsto a$

# Example: Monomorphic Lists

$\mathtt{data}$ $\mathsf{AList\_}\ x = Z \mid SA_1\ x \mid SA_2\ x \mid \ldots \cong$
$\mathtt{data}$ $\mathsf{AList\_}\ x = Z \mid S\ A\ x$

$$1 \xrightarrow{Shape} \mathtt{Set}$$
$$\bullet \mapsto \{Z, S\}$$

$$\int Shape \xrightarrow{Pos} \mathtt{Set}$$
$$(\bullet, Z) \mapsto \emptyset$$
$$(\bullet, Sa) \mapsto \{\bullet\}$$

$$\int Pos \xrightarrow{next} 1$$
$$(\bullet, Sa, \bullet) \mapsto a$$

$$\begin{tikzcd}
X \arrow[r, equal] \arrow[d, "x"] \arrow[dr, phantom, "\llcorner" very near end] & X \arrow[d, "x"] & 1+X \arrow[d, "Id+x"'] \arrow[dr, "Id\nabla x"] & \\
1 \arrow[r, equal] & 1 \arrow[r, "inR"] & 2 \arrow[r, "!"] & 1
\end{tikzcd}$$

# Example: Monomorphic Lists

$\mathtt{data}\ \mathrm{AList\_}\ x = Z \mid SA_1\ x \mid SA_2\ x \mid \ldots \cong$
$\mathtt{data}\ \mathrm{AList\_}\ x = Z \mid \mathrm{S}\ \mathrm{A}\ x$

$$1 \xrightarrow{Shape} \mathtt{Set}$$
$$\bullet \mapsto \{\mathrm{Z}, \mathrm{S}\}$$

$$\int Shape \xrightarrow{Pos} \mathtt{Set}$$
$$(\bullet, Z) \mapsto \emptyset$$
$$(\bullet, Sa) \mapsto \{\bullet\}$$

$$\int Pos \xrightarrow{next} 1$$
$$(\bullet, Sa, \bullet) \mapsto a$$

$$\begin{tikzcd}
X \arrow[r, equal] \arrow[d, "x"] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[d, "x"] & 1+X \arrow[d, "Id+x"'] \arrow[dr, "Id\nabla x"] & \\
1 \arrow[r, equal] & 1 \arrow[r, "inR"] & 2 \arrow[r, "!"] & 1
\end{tikzcd}$$

$\Pi_{inL} X \equiv$
$\mathrm{b} : \mathrm{Bool} \vdash (i : \mathrm{inL}^{-1}\ b) \to X$
$\cong b : \mathrm{Bool} \vdash i : (b == true) \to X$
$\cong 1 + X$