Foundational, Compositional (Co)datatypes for Higher-Order Logic
Category Theory Applied to Theorem Proving

Dmitriy Traytel  Andrei Popescu  Jasmin Blanchette

Technische Universität München
Outline

Datatypes in HOL—State of the Art

Bounded Natural Functors

(Co)datatypes

(Co)nclusion
Outline

Datatypes in HOL—State of the Art

Bounded Natural Functors

(Co)datatypes

(Co)nclusion
LCF philosophy
LCF philosophy

*Small inference kernel*
Isabelle/HOL

- LCF philosophy
  *Small inference kernel*
- Foundational approach
Isabelle/HOL

- LCF philosophy
  - Small inference kernel
- Foundational approach
  - Reduce high-level specifications to primitive mechanisms
Isabelle/HOL

- LCF philosophy
  *Small inference kernel*
- Foundational approach
  *Reduce high-level specifications to primitive mechanisms*
- HOL = simply typed set theory with ML-style polymorphism
Isabelle/HOL

- LCF philosophy
  *Small inference kernel*
- Foundational approach
  *Reduce high-level specifications to primitive mechanisms*
- HOL = simply typed set theory with ML-style polymorphism
  *Restrictive logic*
- LCF philosophy
  *Small inference kernel*

- Foundational approach
  *Reduce high-level specifications to primitive mechanisms*

- HOL = simply typed set theory with ML-style polymorphism
  *Restrictive logic*
  *Weaker than ZF*
LCF philosophy
   Small inference kernel

Foundational approach
   Reduce high-level specifications to primitive mechanisms

HOL = simply typed set theory with ML-style polymorphism
   Restrictive logic
   Weaker than ZF
Datatype specification

datatype $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)
datatype $\alpha$ tree = Node $\alpha$ ($\alpha$ tree list)
Datatype specification

\[
\text{datatype } \alpha \text{ list } = \text{Nil} \mid \text{Cons } \alpha (\alpha \text{ list}) \\
\text{datatype } \alpha \text{ tree } = \text{Node } \alpha (\alpha \text{ tree list})
\]

Primitive type definitions
The traditional approach
Melham 1989, Gunter 1994

- Fragment of ML (non-co)datatypes
The traditional approach
Melham 1989, Gunter 1994

- Fragment of ML (non-co)datatypes
- Fixed universe for recursive types
The traditional approach
Melham 1989, Gunter 1994

- Fragment of ML (non-co)datatypes
- Fixed universe for recursive types
- Simulate nested recursion by mutual recursion

\[
\text{datatype } \alpha\text{ list } = \text{Nil} \mid \text{Cons } \alpha (\alpha\text{ list})
\]
\[
\text{datatype } \alpha\text{ tree } = \text{Node } \alpha (\alpha\text{ tree list})
\]
The traditional approach
Melham 1989, Gunter 1994

- Fragment of ML (non-co)datatypes
- Fixed universe for recursive types
- Simulate nested recursion by mutual recursion

\[
\begin{align*}
\text{datatype } \alpha \text{ list} & \quad = \quad \text{Nil} \mid \text{Cons } \alpha (\alpha \text{ list}) \\
\text{datatype } \alpha \text{ tree} & \quad = \quad \text{Node } \alpha (\alpha \text{ tree_list}) \\
\text{and } \alpha \text{ tree_list} & \quad = \quad \text{Nil} \mid \text{Cons } (\alpha \text{ tree}) (\alpha \text{ tree_list})
\end{align*}
\]
The traditional approach
Melham 1989, Gunter 1994

- Fragment of ML (non-co)datatypes
- Fixed universe for recursive types
- Simulate nested recursion by mutual recursion

\[
\begin{align*}
\text{datatype } \alpha \text{ list} & = \text{Nil} | \text{Cons } \alpha (\alpha \text{ list}) \\
\text{datatype } \alpha \text{ tree} & = \text{Node } \alpha (\alpha \text{ tree}_\text{list}) \\
\text{and } \alpha \text{ tree}_\text{list} & = \text{Nil} | \text{Cons } (\alpha \text{ tree}) (\alpha \text{ tree}_\text{list})
\end{align*}
\]

- Implemented in Isabelle by Berghofer & Wenzel 1999
Limitations
Berghofer & Wenzel 1999

1. noncompositionality
2. no codatatypes
3. no non-free structures
Limitations
LICS 2012

1. noncompositionality
2. no codatatypes
3. no non-free structures
Limitations
LICS 2012

1. noncompositionality
2. no codatatypes
3. no non-free structures
Limitations

LICS 2012

1. noncompositionality
2. no codatatypes
3. no non-free structures
Limitations

LICS 2012

1. noncompositionality
2. no codatatypes
3. no non-free structures
Outline

Datatypes in HOL—State of the Art

Bounded Natural Functors

(Co)datatypes

(Co)nclusion
\[
\text{datatype } \alpha \text{ list } = \text{Nil} | \text{Cons } \alpha (\alpha \text{ list}) \\
\text{codatatype } \alpha \text{ tree } = \text{Node } \alpha (\alpha \text{ tree list})
\]
datatype \( \alpha \) list = Nil | Cons \( \alpha \) (\( \alpha \) list)

codatatype \( \alpha \) tree = Node \( \alpha \) (\( \alpha \) tree list)

\[\begin{array}{c}
P \ n = \text{print} \ n; \text{for } i = 1 \text{ to } n \text{ do } P \ (n + i);
\end{array}\]
\[
\text{datatype } \alpha \text{ list } = \text{Nil} \mid \text{Cons } \alpha (\alpha \text{ list}) \\
\text{codatatype } \alpha \text{ tree } = \text{Node } \alpha (\alpha \text{ tree list})
\]

▷ \( \text{P } n = \text{print } n; \text{ for } i = 1 \text{ to } n \text{ do } \text{P} (n + i); \)

▷ \text{evaluation tree for } P \ 2

\[
\begin{array}{c}
[2] \\
[3, 4] \\
[4, 5, 6] \\
[5, 6, 7, 8] \\
[5, 6, 7, 8] \\
[6, 7, 8, 9, 10] \\
[7, 8, 9, 10, 11, 12] \\
\end{array}
\]
datatype $\alpha$ list $=$ Nil | Cons $\alpha$ ($\alpha$ list)
codatatype $\alpha$ tree $=$ Node $\alpha$ ($\alpha$ tree list)

- Compositionality = no unfolding
datatype \( \alpha \text{ list} \) = Nil | Cons \( \alpha \) (\( \alpha \text{ list} \))

codatatype \( \alpha \text{ tree} \) = Node \( \alpha \) (\( \alpha \text{ tree} \) fset)

- Compositionality = no unfolding
- Need abstract interface
datatype \( \alpha \) list = Nil | Cons \( \alpha \) (\( \alpha \) list)

codatatype \( \alpha \) tree = Node \( \alpha \) (\( \alpha \) tree fset)

▶ Compositionality = no unfolding
▶ Need abstract interface
▶ What interface?
Type constructors are not just operators on types!
The interface: **bounded natural functor**

type constructor $F$
The interface: bounded natural functor

\[ \text{type constructor } F \quad \text{functor} \]

\[ \text{Fmap} \]
The interface: **bounded natural functor**

- type constructor \( F \)
- \( F_{\text{map}} \)
- \( F_{\text{set}} \)

\( F \) is a functor, \( F_{\text{map}} \) is a functor, and \( F_{\text{set}} \) is a natural transformation.
The interface: **bounded natural functor**

- **type constructor** $F$
- **Fmap**
- **Fset**
- **Fbd**

$F$ is a functor

$Fset$ is a natural transformation

$Fbd$ is an infinite cardinal
The interface: **bounded natural functor**

- type constructor $F$
- $\text{Fmap}$
- $\text{Fset}$
- $\text{Fbd}$

$\{ \text{functor} \}$

$\{ \text{natural transformation} \}$

$\{ \text{infinite cardinal} \}$

$\text{BNF} = \text{type constructor} + \text{polymorphic constraints} + \text{assumptions}$
Type constructors are functors

\[ \text{Fmap} : (\alpha \to \alpha') \to (\beta \to \beta') \to (\alpha, \beta) \text{ F} \to (\alpha', \beta') \text{ F} \]
Type constructors are functors

\[ \text{Fmap : } (\alpha \to \alpha') \to (\beta \to \beta') \to (\alpha, \beta) \text{ F} \to (\alpha', \beta') \text{ F} \]

\[ \text{Fmap id id } = \text{id} \]

\[ \text{Fmap } f_1 f_2 \circ \text{Fmap } g_1 g_2 = \text{Fmap } (f_1 \circ g_1) (f_2 \circ g_2) \]
Type constructors are containers

\[ \text{Fset}_1 : (\alpha, \beta) \ F \rightarrow \alpha \ \text{set} \]
\[ \text{Fset}_2 : (\alpha, \beta) \ F \rightarrow \beta \ \text{set} \]
Type constructors are containers

$$\text{Fset}_1 : (\alpha, \beta) \ F \rightarrow \alpha \ 	ext{set}$$
$$\text{Fset}_2 : (\alpha, \beta) \ F \rightarrow \beta \ 	ext{set}$$

$$\text{Fset}_1 \circ \text{Fmap} \ f_1 \ f_2 = \text{image} \ f_1 \circ \text{Fset}_1$$
$$\text{Fset}_2 \circ \text{Fmap} \ f_1 \ f_2 = \text{image} \ f_2 \circ \text{Fset}_2$$
Further BNF assumptions

$$\forall x \in \text{Fset}_1 \ z. \ f_1 \ x = g_1 \ x$$
$$\forall x \in \text{Fset}_2 \ z. \ f_2 \ x = g_2 \ x$$

$$\Rightarrow$$

$$\text{Fmap} \ f_1 \ f_2 \ z = \text{Fmap} \ g_1 \ g_2 \ z$$
Further BNF assumptions

\[
\forall x \in \text{Fset}_1 \ z. \ f_1 x = g_1 x \\
\forall x \in \text{Fset}_2 \ z. \ f_2 x = g_2 x
\] \Rightarrow \text{Fmap} \ f_1 \ f_2 \ z = \text{Fmap} \ g_1 \ g_2 \ z

\aleph_0 \leq \text{Fbd}
Further BNF assumptions

\[ \forall x \in \text{Fset}_1 \ z. \ f_1 x = g_1 x \quad \forall x \in \text{Fset}_2 \ z. \ f_2 x = g_2 x \]  \Rightarrow \quad \text{Fmap} \ f_1 \ f_2 \ z = \text{Fmap} \ g_1 \ g_2 \ z \]

\[ \aleph_0 \leq \text{Fbd} \]

\[ |\text{Fset}_i \ z| \leq \text{Fbd} \]
Further BNF assumptions

\[ \forall x \in \text{Fset}_1 \ z. \ f_1 \ x = g_1 \ x \]
\[ \forall x \in \text{Fset}_2 \ z. \ f_2 \ x = g_2 \ x \]  \[\Rightarrow\]  \[F\text{map} \ f_1 \ f_2 \ z = F\text{map} \ g_1 \ g_2 \ z\]

\[ \aleph_0 \leq \text{Fbd} \]

\[ |\text{Fset}_i \ z| \leq \text{Fbd} \]

\[ |(\alpha_1, \alpha_2) \ F| \leq (|\alpha_1| + |\alpha_2|)^{\text{Fbd}} \]
Further BNF assumptions

\[\forall x \in \text{Fset}_1 \ z. \ f_1 x = g_1 x \quad \forall x \in \text{Fset}_2 \ z. \ f_2 x = g_2 x \quad \Rightarrow \quad \text{Fmap} \ f_1 \ f_2 \ z = \text{Fmap} \ g_1 \ g_2 \ z\]

\[\aleph_0 \leq \text{Fbd} \]

\[|\text{Fset}_i \ z| \leq \text{Fbd}\]

\[|\alpha_1, \alpha_2\ F| \leq (|\alpha_1| + |\alpha_2|)^\text{Fbd}\]

(F, Fmap) preserves weak pullbacks
What are bounded natural functors good for?

BNFs ...
What are bounded natural functors good for?

BNFs ...

- cover basic type constructors (e.g. +, ×, unit, and $\alpha \rightarrow \beta$ for fixed $\alpha$)
- cover non-free type constructors (e.g. fset, cset)
- are closed under composition
- admit initial algebras (datatypes)
- admit final coalgebras (codatatypes)
- are closed under initial algebras and final coalgebras
- make initial algebras and final coalgebras expressible in HOL
What are bounded natural functors good for?

BNFs ...

- cover basic type constructors (e.g. $+$, $\times$, unit, and $\alpha \rightarrow \beta$ for fixed $\alpha$)
- cover non-free type constructors (e.g. fset, cset)
What are bounded natural functors good for?

BNFs ...

- cover basic type constructors (e.g. +, ×, unit, and \( \alpha \rightarrow \beta \) for fixed \( \alpha \))
- cover non-free type constructors (e.g. fset, cset)
- are closed under composition
What are bounded natural functors good for?

BNFs ...

- cover basic type constructors (e.g. +, ×, unit, and \( \alpha \to \beta \) for fixed \( \alpha \))
- cover non-free type constructors (e.g. fset, cset)
- are closed under composition
- admit initial algebras (datatypes)
What are bounded natural functors good for?

BNFs ...

- cover basic type constructors (e.g. $+$, $\times$, unit, and $\alpha \rightarrow \beta$ for fixed $\alpha$)
- cover non-free type constructors (e.g. fset, cset)
- are closed under composition
- admit initial algebras (datatypes)
- admit final coalgebras (codatatypes)
What are bounded natural functors good for?

BNFs ...

- cover basic type constructors (e.g. $+$, $\times$, unit, and $\alpha \rightarrow \beta$ for fixed $\alpha$)
- cover non-free type constructors (e.g. fset, cset)
- are closed under composition
- admit initial algebras (datatypes)
- admit final coalgebras (codatatypes)
- are closed under initial algebras and final coalgebras
What are bounded natural functors good for?

BNFs ...
- cover basic type constructors (e.g. $+$, $\times$, unit, and $\alpha \to \beta$ for fixed $\alpha$)
- cover non-free type constructors (e.g. fset, cset)
- are closed under composition
- admit initial algebras (datatypes)
- admit final coalgebras (codatatypes)
- are closed under initial algebras and final coalgebras
- make initial algebras and final coalgebras expressible in HOL
Outline

Datatypes in HOL—State of the Art

Bounded Natural Functors

(Co)datatypes

(Co)nclusion
From user specifications to (co)datatypes

Given

```plaintext
datatype \( \alpha \) list = Nil | Cons \( \alpha \) (\( \alpha \) list)
```

From user specifications to (co)datatypes

Given

\[ \text{datatype } \alpha \text{ list } = \text{Nil} | \text{Cons } \alpha (\alpha \text{ list}) \]

1. Abstract to \( \beta = \text{unit} + \alpha \times \beta \)
From user specifications to (co)datatypes

Given

datatype \( \alpha \) list = Nil \mid Cons \( \alpha \) (\( \alpha \) list)

1. Abstract to \( \beta = \text{unit} + \alpha \times \beta \)

2. Prove that \( (\alpha, \beta) F = \text{unit} + \alpha \times \beta \) is a BNF

3. Define \( \text{F-algebras} \)

4. Construct initial algebra \( (\alpha \) list, \( \text{fld: unit} + \alpha \times \alpha \) list \( \rightarrow \alpha \) list) \)

5. Define iterator \( \text{iter: (unit} + \alpha \times \alpha \) list \( \rightarrow \beta \rightarrow \alpha \) list} \)

6. Prove characteristic theorems (e.g. induction)

7. Prove that list is a BNF (enables nested recursion)
From user specifications to (co)datatypes

Given

datatype $\alpha\text{ list} = \text{Nil} | \text{Cons } \alpha (\alpha\text{ list})$

1. Abstract to $\beta = \text{unit} + \alpha \times \beta$
2. Prove that $(\alpha, \beta) F = \text{unit} + \alpha \times \beta$ is a BNF
3. Define F-algebras
From user specifications to (co)datatypes

Given

datatype \( \alpha \) list = Nil | Cons \( \alpha \) (\( \alpha \) list)

1. Abstract to \( \beta = \text{unit} + \alpha \times \beta \)
2. Prove that \((\alpha, \beta) \mathcal{F} = \text{unit} + \alpha \times \beta\) is a BNF
3. Define \(\mathcal{F}\)-algebras
4. Construct initial algebra

\((\alpha \) list, fld : unit + \( \alpha \) \times \( \alpha \) list \rightarrow \( \alpha \) list)\)
From user specifications to (co)datatypes

Given

datatype \( \alpha \text{ list} = \text{Nil} \mid \text{Cons } \alpha (\alpha \text{ list}) \)

1. Abstract to \( \beta = \text{unit} + \alpha \times \beta \)
2. Prove that \((\alpha, \beta) F = \text{unit} + \alpha \times \beta \) is a BNF
3. Define F-algebras
4. Construct initial algebra

\((\alpha \text{ list}, \text{fld} : \text{unit} + \alpha \times \alpha \text{ list} \to \alpha \text{ list})\)

5. Define iterator

\(\text{iter} : (\text{unit} + \alpha \times \alpha \text{ list} \to \beta) \to \alpha \text{ list} \to \beta\)
From user specifications to (co)datatypes

Given

\[
datatype \alpha \text{ list } = \text{Nil} \mid \text{Cons } \alpha (\alpha \text{ list})
\]

1. Abstract to \( \beta = \text{unit} + \alpha \times \beta \)

2. Prove that \((\alpha, \beta) F = \text{unit} + \alpha \times \beta\) is a BNF

3. Define F-algebras

4. Construct initial algebra

\[
(\alpha \text{ list}, \text{fld} : \text{unit} + \alpha \times \alpha \text{ list} \to \alpha \text{ list})
\]

5. Define iterator

\[
\text{iter} : (\text{unit} + \alpha \times \alpha \text{ list} \to \beta) \to \alpha \text{ list} \to \beta
\]

6. Prove characteristic theorems (e.g. induction)
From user specifications to (co)datatypes

Given

$$\text{datatype } \alpha \text{ list } = \text{Nil} \mid \text{Cons } \alpha (\alpha \text{ list})$$

1. Abstract to $$\beta = \text{unit} + \alpha \times \beta$$
2. Prove that $$(\alpha, \beta) F = \text{unit} + \alpha \times \beta$$ is a BNF
3. Define F-algebras
4. Construct initial algebra

$$(\alpha \text{ list}, \text{fld} : \text{unit} + \alpha \times \alpha \text{ list} \rightarrow \alpha \text{ list})$$

5. Define iterator

$$\text{iter} : (\text{unit} + \alpha \times \alpha \text{ list} \rightarrow \beta) \rightarrow \alpha \text{ list} \rightarrow \beta$$

6. Prove characteristic theorems (e.g. induction)
7. Prove that list is a BNF
From user specifications to (co)datatypes

Given

\[
\text{datatype } \alpha \text{ list } = \text{Nil} \mid \text{Cons } \alpha (\alpha \text{ list})
\]

1. Abstract to \( \beta = \text{unit} + \alpha \times \beta \)
2. Prove that \((\alpha, \beta) F = \text{unit} + \alpha \times \beta\) is a BNF
3. Define F-algebras
4. Construct initial algebra

\[
(\alpha \text{ list}, \text{fld} : \text{unit} + \alpha \times \alpha \text{ list} \rightarrow \alpha \text{ list})
\]

5. Define iterator

\[
\text{iter} : (\text{unit} + \alpha \times \alpha \text{ list} \rightarrow \beta) \rightarrow \alpha \text{ list} \rightarrow \beta
\]

6. Prove characteristic theorems (e.g. induction)
7. Prove that list is a BNF (enables nested recursion)
From user specifications to (co)datatypes

Given

\[
\text{codatatype } \alpha \text{llist} = \text{LNil} | \text{LCons } \alpha (\alpha \text{llist})
\]

1. Abstract to \( \beta = \text{unit} + \alpha \times \beta \)
2. Prove that \((\alpha, \beta) F = \text{unit} + \alpha \times \beta\) is a BNF
3. Define F-coalgebras
4. Construct final coalgebra

\[
(\alpha \text{llist}, \text{unf} : \alpha \text{llist} \rightarrow \text{unit} + \alpha \times \alpha \text{llist})
\]

5. Define coiterator

\[
\text{coiter} : (\beta \rightarrow \text{unit} + \alpha \times \alpha \text{llist}) \rightarrow \beta \rightarrow \alpha \text{llist}
\]

6. Prove characteristic theorems (e.g. coinduction)
7. Prove that llist is a BNF (enables nested corecursion)
Induction

\[ \beta = (\alpha, \beta) F \]

- Given \( \varphi : \alpha \text{IF} \rightarrow \text{bool} \)
Induction

\[ \beta = (\alpha, \beta) F \]

- Given \( \varphi : \alpha \text{IF} \to \text{bool} \)
- Abstract induction principle

\[ \forall z. \ (\forall x \in \text{Fset}_2 z. \ \varphi x) \Rightarrow \varphi (\text{fld} z) \]

\[ \forall x. \ \varphi x \]
Induction

\[ \beta = \text{unit} + \alpha \times \beta \]

- Given \( \varphi : \alpha \text{ IF} \rightarrow \text{bool} \)
- Abstract induction principle

\[
\forall z. \left( \forall x \in \text{Fset}_2 \ z. \varphi x \right) \Rightarrow \varphi (\text{fld} \ z) \\
\forall x. \varphi x
\]

- Given \( \varphi : \alpha \text{ list} \rightarrow \text{bool} \)
- Case distinction on \( z \)

\[
(\forall ys \in \emptyset. \varphi ys) \Rightarrow \varphi (\text{fld} (\text{Inl} ())) \\
\forall x \ \forall xs. (\forall ys \in \{xs\}. \varphi ys) \Rightarrow \varphi (\text{fld} (\text{Inr} (x, xs))) \\
\forall xs. \varphi xs
\]
Induction

\[ \beta = \text{unit} + \alpha \times \beta \]

- Given \( \varphi : \alpha \text{ IF} \rightarrow \text{bool} \)
  - Abstract induction principle

\[ \forall z. (\forall x \in \text{Fset}_2 z. \varphi x) \Rightarrow \varphi (\text{fld} z) \]

\[ \forall x. \varphi x \]

- Given \( \varphi : \alpha \text{ list} \rightarrow \text{bool} \)
  - Concrete induction principle

\[ \forall x \; \varphi x \Rightarrow \varphi (\text{fld} (\text{Inl} ())) \]

\[ \forall x. \varphi xs \Rightarrow \varphi (\text{fld} (\text{Inr} (x, xs))) \]

\[ \forall xs. \varphi xs \]
Induction

\[ \beta = \text{unit} + \alpha \times \beta \]

- Given \( \varphi : \alpha \text{ IF} \rightarrow \text{bool} \)
- Abstract induction principle

\[
\forall z. \left( \forall x \in \text{Fset}_2 z. \varphi x \right) \Rightarrow \varphi (\text{fld } z)
\]

\[
\forall x. \varphi x
\]

- Given \( \varphi : \alpha \text{ list} \rightarrow \text{bool} \)
- In constructor notation

\[
\forall xzs. \varphi xsz \Rightarrow \varphi (\text{Cons } xzs)
\]

\[
\forall xzs. \varphi xzs
\]
Induction & Coinduction

\[ \beta = (\alpha, \beta) \text{ F} \]

- Given \( \varphi : \alpha \text{ IF} \rightarrow \text{ bool} \)
- Abstract induction principle

\[ \forall z. \left( \forall x \in \text{Fset}_2 z. \varphi x \right) \Rightarrow \varphi (\text{fld} z) \]
\[ \forall x. \varphi x \]

- Given \( \psi : \alpha \text{ JF} \rightarrow \alpha \text{ JF} \rightarrow \text{ bool} \)
Induction & Coinduction

\[ \beta = (\alpha, \beta) \]

- Given \( \varphi : \alpha \text{IF} \rightarrow \text{bool} \)

- Abstract induction principle

\[
\forall z. \left( \forall x \in \text{Fset}_2 z. \varphi x \right) \Rightarrow \varphi (\text{fld } z)
\]

\[
\forall x. \varphi x
\]

- Given \( \psi : \alpha \text{JF} \rightarrow \alpha \text{JF} \rightarrow \text{bool} \)

- Abstract coinduction principle

\[
\forall x y. \psi x y \Rightarrow \text{Fpred Eq } \psi (\text{unf } x) (\text{unf } y)
\]

\[
\forall x y. \psi x y \Rightarrow x = y
\]
Example

codatatype α tree = Node (lab: α) (sub: α tree fset)
Example

codatatype \( \alpha \) tree = Node (lab: \( \alpha \)) (sub: \( \alpha \) tree fset)

corec tmap : (\( \alpha \to \beta \)) \to \alpha \) tree \to \beta \) tree where
lab (tmap f t) = f (lab t)
sub (tmap f t) = image (tmap f) (sub t)
Example

codatatype \( \alpha \text{ tree} = \text{Node} (\text{lab: } \alpha) (\text{sub: } \alpha \text{ tree fset}) \)

corec \( \text{tmap} : (\alpha \to \beta) \to \alpha \text{ tree} \to \beta \text{ tree} \) where

\[
\begin{align*}
\text{lab (tmap } f \ t) &= f (\text{lab } t) \\
\text{sub (tmap } f \ t) &= \text{image (tmap } f \text{) (sub } t)
\end{align*}
\]

lemma \( \text{tmap } (f \circ g) \ t = \text{tmap } f \text{ (tmap } g \ t) \)
Example

codatatype \( \alpha \) tree = Node (lab: \( \alpha \)) (sub: \( \alpha \) tree fset)

corec tmap : (\( \alpha \rightarrow \beta \)) \rightarrow \alpha \) tree \rightarrow \beta \) tree where
lab (tmap f t) = f (lab t)
sub (tmap f t) = image (tmap f) (sub t)

lemma tmap (f \circ g) t = tmap f (tmap g t)
by (intro tree_coinduct[where \( \psi = \lambda t_1 \ t_2. \exists t. t_1 = tmap (f \circ g) t \land t_2 = tmap f (tmap g t)])) force+
Outline

Datatypes in HOL—State of the Art

Bounded Natural Functors

(Co)datatypes

(Co)nclusion
Foundational, Compositional (Co)datatypes for Higher-Order Logic

Category Theory Applied to Theorem Proving
Foundational, Compositional (Co)datatypes for Higher-Order Logic
Category Theory Applied to Theorem Proving

- Framework for defining types in HOL
Foundational, Compositional (Co)datatypes for Higher-Order Logic

Category Theory Applied to Theorem Proving

- Framework for defining types in HOL
- Characteristic theorems are derived, not stated as axioms
Foundational, Compositional (Co)datatypes for Higher-Order Logic
Category Theory Applied to Theorem Proving

- Framework for defining types in HOL
- Characteristic theorems are derived, not stated as axioms
- Mutual and nested combinations of (co)datatypes and custom BNFs

Thank you for your attention!
Framework for defining types in HOL
- Characteristic theorems are derived, not stated as axioms
- Mutual and nested combinations of (co)datatypes and custom BNFs
- Adapt insights from category theory to HOL’s restrictive type system
Framework for defining types in HOL

Characteristic theorems are derived, not stated as axioms

Mutual and nested combinations of (co)datatypes and custom BNFs

Adapt insights from category theory to HOL’s restrictive type system

Formalized & implemented in Isabelle/HOL
Foundational, Compositional (Co)datatypes for Higher-Order Logic
Category Theory Applied to Theorem Proving

- Framework for defining types in HOL
- Characteristic theorems are derived, not stated as axioms
- Mutual and nested combinations of (co)datatypes and custom BNFs
- Adapt insights from category theory to HOL’s restrictive type system
- Formalized & implemented in Isabelle/HOL

Thank you for your attention!
Foundational, Compositional (Co)datatypes for
Higher-Order Logic
Category Theory Applied to Theorem Proving

Dmitriy Traytel    Andrei Popescu    Jasmin Blanchette

Technische Universität München
Outline

Backup slides
Type constructors act on sets

\[(A_1, A_2) \ F = \{ z \mid \text{Fset}_1 \ z \subseteq A_1 \land \text{Fset}_2 \ z \subseteq A_2 \}\]
Type constructors act on sets

\[(A_1, A_2) F = \{ z \mid \text{Fset}_1 \ z \subseteq A_1 \land \text{Fset}_2 \ z \subseteq A_2 \}\]

\[(A_1, A_2) F : (\alpha, \beta) \text{ F set} \]

\[(\forall i \in \{1, 2\}. \ \forall x \in \text{Fset}_i \ z. \ \ f_i x = g_i x \) \implies \text{Fmap} \ f_1 \ f_2 \ z = \text{Fmap} \ g_1 \ g_2 \ z\]
Type constructors are bounded

Fbd: infinite cardinal

\[ (\alpha, \beta) \quad F \]

\[ Fset_1 \rightarrow a_1 \quad a_2 \rightarrow Fset_2 \]

\[ \alpha \text{ set} \quad \beta \text{ set} \]
Type constructors are bounded

$F_{bd}$: infinite cardinal

$$\alpha, \beta \in F$$

$F_{set_1}$ and $F_{set_2}$

$|F_{set_i} z| \leq F_{bd}$
Type constructors are bounded

Fbd: infinite cardinal

\[ F^{(\alpha, \beta)} : \text{set} \]

\[ F_{\alpha} \setminus F_{\beta} \]

| Fset \_ z | \leq Fbd
Type constructors are bounded

Fbd: infinite cardinal

\[ (\alpha, \beta) \ F \]

\( \alpha \) set

\( \alpha \) set

\( \beta \) set

\( \beta \) set

\( (A_1, A_2) \ F : (\alpha, \beta) \ F \ set \)

\[ |Fset_i \ z| \leq \ Fbd \]

\[ |(A_1, A_2) \ F| \leq (|A_1| + |A_2| + 2)^{Fbd} \]
Algebras, Coalgebras & Morphisms

\[ \beta = (\alpha, \beta) \mathcal{F} \]

\[
\begin{align*}
(\alpha, A) \mathcal{F} \\
\downarrow s \\
A
\end{align*}
\]
Algebras, Coalgebras & Morphisms

\[ \beta = (\alpha, \beta) F \]
$\beta = (\alpha, \beta) F$

```
(\alpha, A) F
  \downarrow s
  \downarrow
  A

(\alpha, A) F \xrightarrow{F \text{map id } f} (\alpha, B) F

(\alpha, A) F
  \downarrow S_A
  \downarrow
  A

\text{map id } f

A
  \downarrow s
  \downarrow
(\alpha, A) F

A
  \downarrow s
  \downarrow
(\alpha, A) F
```
Algebras, Coalgebras & Morphisms

\[ \beta = (\alpha, \beta) \mathcal{F} \]
Initial Algebras & Final Coalgebras

\[ \beta = (\alpha, \beta) \]

- **weakly initial**: exists morphism to any other algebra
- **initial**: exists *unique* morphism to any other algebra
- **weakly final**: exists morphism from any other coalgebra
- **final**: exists *unique* morphism from any other coalgebra
Initial Algebras & Final Coalgebras

\[ \beta = (\alpha, \beta)_F \]

weakly initial: exists morphism to any other algebra

initial: exists *unique* morphism to any other algebra

weakly final: exists morphism from any other coalgebra

final: exists *unique* morphism from any other coalgebra

- Product of all algebras is weakly initial
- Suffices to consider algebras over types of certain cardinality
- Minimal subalgebra of weakly initial algebra is initial
Initial Algebras & Final Coalgebras

\[ \beta = (\alpha, \beta) \mathcal{F} \]

- **weakly initial:** exists morphism to any other algebra
- **initial:** exists *unique* morphism to any other algebra
- **weakly final:** exists morphism from any other coalgebra
- **final:** exists *unique* morphism from any other coalgebra

- Product of all algebras is weakly initial
- Suffices to consider algebras over types of certain cardinality
- Minimal subalgebra of weakly initial algebra is initial
- Construct minimal subalgebra from below by transfinite recursion

\[ \Rightarrow \text{Have a bound for its cardinality} \]

\[ \Rightarrow (\alpha\ \text{IF}, \text{fld} : (\alpha, \alpha\ \text{IF}) \mathcal{F} \rightarrow \alpha\ \text{IF}) \]
Initial Algebras & Final Coalgebras

\[ \beta = (\alpha, \beta) \mathcal{F} \]

**weakly initial:** exists morphism to any other algebra

**initial:** exists *unique* morphism to any other algebra

**weakly final:** exists morphism from any other coalgebra

**final:** exists *unique* morphism from any other coalgebra

- Product of all algebras is weakly initial
- Suffices to consider algebras over types of certain cardinality
- Minimal subalgebra of weakly initial algebra is initial
- Construct minimal subalgebra from below by transfinite recursion
  - Have a bound for its cardinality

\[ \Rightarrow (\alpha \text{ IF}, \text{fld} : (\alpha, \alpha \text{ IF}) \mathcal{F} \rightarrow \alpha \text{ IF}) \]

- Sum of all coalgebras is weakly final
- Suffices to consider coalgebras over types of certain cardinality
- Quotient of weakly final coalgebra to the greatest bisimulation is final

\[ \Rightarrow (\alpha \text{ JF}, \text{unf} : \alpha \text{ JF} \rightarrow (\alpha, \alpha \text{ JF}) \mathcal{F}) \]
Initial Algebras & Final Coalgebras

\[ \beta = (\alpha, \beta) F \]

- **weakly initial**: exists morphism to any other algebra
- **initial**: exists *unique* morphism to any other algebra
- **weakly final**: exists morphism from any other coalgebra
- **final**: exists *unique* morphism from any other coalgebra

- Product of all algebras is weakly initial
- Suffices to consider algebras over types of certain cardinality
- Minimal subalgebra of weakly initial algebra is initial
- Construct minimal subalgebra from below by transfinite recursion
- Have a bound for its cardinality

\[ (\alpha \text{ IF}, \text{fld} : (\alpha, \alpha \text{ IF}) F \rightarrow \alpha \text{ IF}) \]

- Sum of all coalgebras is weakly final
- Suffices to consider coalgebras over types of certain cardinality
- Quotient of weakly final coalgebra to the greatest bisimulation is final
- Use concrete weakly final coalgebra (elements are tree-like structures)
- Have a bound for its cardinality

\[ (\alpha \text{ JF}, \text{unf} : \alpha \text{ JF} \rightarrow (\alpha, \alpha \text{ JF}) F) \]
Iteration & Coiteration

\[ \beta = (\alpha, \beta) \mathcal{F} \]

- Given \( s : (\alpha, \beta) \mathcal{F} \rightarrow \beta \)
Iteration & Coiteration

\[ \beta = (\alpha, \beta) F \]

- Given \( s : (\alpha, \beta) F \rightarrow \beta \)
- Obtain unique morphism \( \text{iter } s \) from \((\alpha \text{ IF}, \text{fld})\) to \((U\beta, s)\)

\[
\begin{array}{c}
(\alpha, \alpha \text{ IF}) F \\
\downarrow \text{fld}
\end{array}
\xrightarrow{\text{Fmap id (iter } s\text{)}}
\begin{array}{c}
(\alpha, \beta) F \\
\downarrow s
\end{array}
\]

\[
\begin{array}{c}
\alpha \text{ IF} \\
\downarrow \text{iter } s
\end{array}
\xrightarrow{\text{iter } s}
\begin{array}{c}
\beta
\end{array}
\]

- Given \( s : (\alpha, \beta) F \rightarrow (\alpha, \beta) F \)
- Obtain unique morphism \( \text{coiter } s \) from \((U\beta, s)\) to \((\alpha \text{ IF}, \text{unf})\)
Iteration & Coiteration

\[ \beta = (\alpha, \beta) F \]

- Given \( s : (\alpha, \beta) F \to \beta \)
- Obtain unique morphism \( \text{iter} \ s \)
  from \( (\alpha \text{ IF}, \text{fld}) \) to \( (U_\beta, s) \)

\[
\begin{array}{ccc}
(\alpha, \alpha \text{ IF}) F & \xrightarrow{\text{Fmap id (iter s)}} & (\alpha, \beta) F \\
\text{fld} & & \downarrow s \\
\alpha \text{ IF} & \xleftarrow{\text{iter s}} & \beta
\end{array}
\]

- Given \( s : \beta \to (\alpha, \beta) F \)
Iteration & Coiteration

$$\beta = (\alpha, \beta) F$$

- Given $$s : (\alpha, \beta) F \rightarrow \beta$$
- Obtain unique morphism $$\text{iter } s$$ from $$(\alpha \text{ IF, fld)}$$ to $$(U\beta, s)$$

- Given $$s : \beta \rightarrow (\alpha, \beta) F$$
- Obtain unique morphism $$\text{coiter } s$$ from $$(U\beta, s)$$ to $$(\alpha \text{ JF, unf})$$
Preservation of BNF Properties

\[ \beta = (\alpha, \beta) F \]

- IFmap \( f = \text{iter} (\text{fld} \circ \text{Fmap } f \text{ id}) \)
- IFset = \text{iter } \text{collect}, \text{ where}

\[
\text{collect } z = \text{Fset}_1 z \cup \bigcup \text{Fset}_2 z
\]
Preservation of BNF Properties

\[ \beta = (\alpha, \beta) F \]

- IFmap \( f = \text{iter} (\text{fld} \circ \text{Fmap} \ f \ \text{id}) \)
- IFset = iter collect, where

\[ \text{collect } z = Fset_1 \ z \cup \bigcup Fset_2 \ z \]

Theorem

(\text{IF, IFmap, IFset, } 2^{Fbd}) \text{ is a BNF}
Preservation of BNF Properties

\[ \beta = (\alpha, \beta) F \]

- \( \text{IFmap } f = \text{iter } (\text{fld } \circ \text{Fmap } f \text{ id}) \)
- \( \text{IFset } = \text{iter } \text{collect}, \text{ where } \)
  \[
  \text{collect } z = \text{Fset}_1 z \cup \bigcup \text{Fset}_2 z
  \]
- \( \text{JFmap } f = \text{coiter } (\text{Fmap } f \text{ id } \circ \text{unf}) \)
- \( \text{JFset } x = \bigcup_{i \in \mathbb{N}} \text{collect}_i x, \text{ where } \)
  \[
  \text{collect}_0 x = \emptyset
  \]
  \[
  \text{collect}_{i+1} x = \text{Fset}_1 (\text{unf } x) \cup \bigcup \text{collect}_i y
  \]
  \[
  y \in \text{Fset}_2 (\text{unf } x)
  \]

Theorem

\((\text{IF, IFmap, IFset, } 2^{\text{Fbd}}) \text{ is a BNF}\)
Preservation of BNF Properties

\[ \beta = (\alpha, \beta)^F \]

- IFmap \( f = \text{iter (fld } \circ \text{ Fmap } f \text{ id)} \)
- IFset = \text{iter collect, where}

\[
\text{collect } z = \text{Fset}_1 \ z \cup \bigcup \text{Fset}_2 \ z
\]

Theorem
(IF, IFmap, IFset, 2^{Fbd}) is a BNF

- JFmap \( f = \text{coiter (Fmap } f \text{ id } \circ \text{ unf)} \)
- JFset \( x = \bigcup_{i \in \mathbb{N}} \text{collect}_i \ x \), where

\[
\text{collect}_0 \ x = \emptyset
\]
\[
\text{collect}_{i+1} \ x = \text{Fset}_1 (\text{unf} \ x) \cup \bigcup \text{collect}_i \ y
\]
\[
y \in \text{Fset}_2 (\text{unf} \ x)
\]

Theorem
(JF, JFmap, JFset, Fbd^{Fbd}) is a BNF