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
Isabelle/HOL

- LCF philosophy
LCF philosophy

*Small inference kernel*
LCF philosophy

*Small inference kernel*

Foundational approach
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
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*
  *Weaker than ZF*
Datatype specification

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

datatype \( \alpha \text{ tree} \) = Node \( \alpha \) (\( \alpha \text{ tree list} \))
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

Diagram:

- New type $\alpha$
- Existing type $\beta$
- Representing set
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

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

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

```
datatype α list = Nil | Cons α (α list)
datatype α tree = Node α (α tree_list)
and α tree_list = Nil | Cons (α tree) (α tree_list)
```

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

codatatype \( \alpha \) tree  =  Node \( \alpha \) (\( \alpha \) tree list)
datatype \( \alpha \) list = Nil | Cons \( \alpha \) (\( \alpha \) list)

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

\( \text{P n = print n; for i = 1 to n do P (n + i);} \)
datatype α list = Nil | Cons α (α list)
codatatype α tree = Node α (α tree list)

▷ P n = print n; for i = 1 to n do P (n + i);
▷ evaluation tree for P 2

```
[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]
```

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

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

▶ Compositionality = no unfolding
datatype α list = Nil | Cons α (α list)
codatatype α tree = Node α (α tree fset)

- Compositionality = no unfolding
- Need abstract interface
datatype $\alpha$ list $\equiv$ Nil $|$ Cons $\alpha$ ($\alpha$ list)
codatatype $\alpha$ tree $\equiv$ 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**

\[
\text{type constructor } F \\
F\text{map}
\]
The interface: **bounded natural functor**

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

functor

natural transformation

Infinite cardinal $\aleph_0$
The interface: **bounded natural functor**

- Type constructor \( F \)
- \( \text{Fmap} \)
- \( \text{Fset} \)
- \( \text{Fbd} \)

functor

natural transformation

infinite cardinal
The interface: **bounded natural functor**

\[\text{type constructor } F \{ \begin{align*}
Fmap & \quad \text{functor} \\
Fset & \quad \text{natural transformation} \\
Fbd & \quad \text{infinite cardinal}
\end{align*} \}\]

**BNF** = type constructor + polymorphic constraints + assumptions
Type constructors are functors

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

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

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

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

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

Fset₁ : \((\alpha, \beta) \overset{F}{\to} \alpha\) set
Fset₂ : \((\alpha, \beta) \overset{F}{\to} \beta\) set

Fset₁ \circ Fmap \ f₁ \ f₂ = \text{image } f₁ \circ Fset₁
Fset₂ \circ Fmap \ f₁ \ f₂ = \text{image } f₂ \circ Fset₂
Further BNF assumptions

\[
\forall x \in Fset_1 \ z. \ f_1 x = g_1 x \\
\forall x \in Fset_2 \ z. \ f_2 x = g_2 x
\]

⇒ \ Fmap f_1 f_2 z = Fmap g_1 g_2 z
Further BNF assumptions

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

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

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

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

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

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

\[ \aleph_0 \leq \ F\text{bd} \]

\[ |\text{Fset}_i \, z| \leq \ F\text{bd} \]

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

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

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

(F, \text{Fmap}) \text{ 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. $+$, $\times$, 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

- 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 \(\alpha \to \beta\) 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. $+$, $\times$, 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
- 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 $\alpha \to \beta$ 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 $\alpha \to \beta$ 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
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
Outline

Datatypes in HOL—State of the Art

Bounded Natural Functors

(Co)datatypes

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

Given

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

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

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

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

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

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 $\alpha$ 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

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

5. Define iterator

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

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

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

5. Define iterator

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

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

Given

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

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

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

5. Define iterator

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

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

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

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

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

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

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

\[
\forall x. \phi x
\]
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 \( \varphi : \alpha \text{ list} \rightarrow \text{bool} \)
- Case distinction on \( z \)

\[
(\forall y_s \in \emptyset. \varphi y_s) \Rightarrow \varphi (\text{fld (Inl ())})
\]

\[
\forall x \; x_s. (\forall y_s \in \{x_s\}. \varphi y_s) \Rightarrow \varphi (\text{fld (Inr (x, x_s)))}
\]

\[
\forall x_s. \varphi x_s
\]
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 \( \varphi : \alpha \text{list} \rightarrow \text{bool} \)
- Concrete induction principle

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

\[
\forall x \ x_\text{s}. \varphi x_\text{s}
\]
Induction

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

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

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

\[
\forall x. \varphi x
\]

\[
\forall x \hspace{0.1cm} \forall xs. \varphi x \hspace{0.1cm} \Rightarrow \hspace{0.1cm} \varphi (\text{Cons } x \hspace{0.1cm} xs)
\]

\[
\forall xs. \varphi xs
\]

\[
\varphi \text{ Nil}
\]

\[
\forall x \hspace{0.1cm} \forall xs. \varphi x \hspace{0.1cm} \Rightarrow \hspace{0.1cm} \varphi (\text{Cons } x \hspace{0.1cm} xs)
\]

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

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

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

- 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

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

\[ \forall x. \varphi x \]

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 $\alpha$ tree = Node (lab: $\alpha$) (sub: $\alpha$ tree fset)
Example

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

corec $\text{tmap} : (\alpha \rightarrow \beta) \rightarrow \alpha$ tree $\rightarrow \beta$ tree where
lab ($\text{tmap}$ f t) = f (lab t)
sub ($\text{tmap}$ f t) = image ($\text{tmap}$ f) (sub t)
Example

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

corec $\text{tmap} : (\alpha \to \beta) \to \alpha$ tree $\to \beta$ tree where
$\text{lab} (\text{tmap } f \ t) = f (\text{lab } t)$
$\text{sub} (\text{tmap } f \ t) = \text{image} (\text{tmap } f) (\text{sub } t)$

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_1 t_2=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

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

- Framework for defining types in HOL

Thank you for your attention!
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

Thank you for your attention!
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
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

Thank you for your attention!
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
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
Outline

Backup slides
Type constructors act on sets

\[(A_1, A_2) \ F = \{ z \mid Fset_1 \ z \subseteq A_1 \land Fset_2 \ z \subseteq A_2 \}\]
Type constructors act on sets

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

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

Fbd: infinite cardinal

\[ F_{\text{set}} \leq \text{Fbd} \]
Type constructors are bounded

Fbd: infinite cardinal

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

\[a_1, a_2 \quad b\]

\[F_{set_1}\]

\[F_{set_2}\]

\[\alpha \ set \quad \beta \ set\]

\[|F_{set_i} z| \leq \ Fbd\]
Type constructors are bounded

\[ \text{Fbd: infinite cardinal} \]

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

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

\[ |F_{seti} z| \leq \text{Fbd} \]
Type constructors are bounded

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

\[ F_{set_1} \]
\[ a_1 \ a_2 \]
\[ \alpha \ set \]

\[ F_{set_2} \]
\[ b \]
\[ \beta \ set \]

\[ A_1 : \alpha \ set \]
\[ a_1 \ a_2 \]

\[ A_2 : \beta \ set \]
\[ b \]

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

\[ |F_{set_j} z| \leq \ F_{bd} \]

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

\[ \beta = (\alpha, \beta) F \]
Algebras, Coalgebras & Morphisms

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

\[ \beta = (\alpha, \beta)F \]
Algebras, Coalgebras & Morphisms

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

- 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
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

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

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

- weakly initial: exists morphism to any other algebra
- initial: exists \textit{unique} morphism to any other algebra
- weakly final: exists morphism from any other coalgebra
- final: exists \textit{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

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

\[ \Rightarrow (\alpha \text{ IF}, \text{fld : } (\alpha, \alpha \text{ IF}) F \rightarrow \alpha \text{ IF}) \]
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
  \[ \Rightarrow \text{ Have a bound for its cardinality} \]
  \[ \Rightarrow (\alpha \text{ IF}, \text{fld} : (\alpha, \alpha \text{ IF}) F \to \alpha \text{ IF}) \]

- 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)
  \[ \Rightarrow \text{ Have a bound for its cardinality} \]
  \[ \Rightarrow (\alpha \text{ JF}, \text{unf} : \alpha \text{ JF} \to (\alpha, \alpha \text{ JF}) F) \]
Iteration & Coiteration

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

- Given \( s : (\alpha, \beta) \text{ F} \to \beta \)
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 \((U\beta, s)\)

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

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

- Given $s : \beta \rightarrow (\alpha, \beta) \ F$

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

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

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

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

$$\text{collect } z = Fset_1 z \cup \bigcup y \in Fset_2 (\text{unf } x)$$
Preservation of BNF Properties

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

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

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

Theorem

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

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

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

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

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

\[
\text{collect}_0 x = \emptyset
\]

\[
\text{collect}_{i+1} x = Fset_1 (\text{unf } x) \cup \bigcup \text{collect}_i y
\]

\[
y \in Fset_2 (\text{unf } x)
\]

**Theorem**

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

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

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

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

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

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

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

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