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

Category Theory Applied to Theorem Proving

Dmitriy Traytel  Andrei Popescu  Jasmin Christian Blanchette

November 13, 2015
Outline

Introduction

Bounded Natural Functors

(Co)datatype (Co)struction

Conclusion
Motivation

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

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

Resolve $\beta = \text{unit} + \alpha \times \beta$ minimally
Motivation

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

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

\[\text{Prove } \begin{array}{c} \varphi \text{ Nil} \\ \forall x \ xs. \varphi \ xs \Rightarrow \varphi (\text{Cons } x \ xs) \end{array} \quad \forall x \ xs. \varphi \ xs\]
**Motivation**

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

codatatype $\alpha$ treeI = Node (lab: $\alpha$) (sub: ($\alpha$ treeI) list)

Resolve $\beta = \text{unit} + \alpha \times \beta$ minimally

Prove $\phi$ Nil \quad \forall x xs. \phi xs \Rightarrow \phi (Cons x xs) \\
\forall xs. \phi xs
Motivation

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

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

Resolve $\beta = \text{unit} + \alpha \times \beta$ minimally

and $\gamma = \alpha \times \gamma$ list maximally

Prove $\forall \alpha \text{ Nil}. \forall x \, \text{xs}. \, \varphi \, \text{xs} \Rightarrow \varphi \, (\text{Cons} \, x \, \text{xs})$

$\forall \alpha \, \text{xs}. \, \varphi \, \text{xs}$
Motivation

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

codatatype $\alpha \text{ tree}_{\text{I}} = \text{Node} \ (\text{lab}: \alpha) \ (\text{sub}: (\alpha \text{ tree}_{\text{I}}) \text{ list})$

Resolve $\beta = \text{unit} + \alpha \times \beta$ minimally

and $\gamma = \alpha \times \gamma \text{ list}$ maximally

Prove $\phi \ \text{Nil} \quad \forall x \ xs. \phi xs \Rightarrow \phi (\text{Cons} x xs)$

$\forall xs. \phi xs$

$\psi \ t_1 \ t_2$

and $\forall x \ y. \psi x y \Rightarrow \text{lab} x = \text{lab} y \land \text{list_pred} \psi (\text{sub} x) (\text{sub} y)$

$t_1 = t_2$
Motivation

datatype $\alpha$ list = Nil | Cons $\alpha$ ($\alpha$ list)
codatatype $\alpha$ tree = Node (lab: $\alpha$) (sub: ($\alpha$ tree) fset)

Resolve $\beta = \text{unit} + \alpha \times \beta$ minimally
and $\gamma = \alpha \times \gamma \text{fset}$ maximally

Prove $\varphi$ Nil
$\forall x \hspace{0.5em} xs. \hspace{0.5em} \varphi xs \Rightarrow \varphi\hspace{0.5em} (\text{Cons} x \hspace{0.5em} xs)$
$\forall xs. \hspace{0.5em} \varphi xs$

and $\forall x \hspace{0.5em} y. \hspace{0.5em} \psi x \hspace{0.5em} y \Rightarrow \text{lab} x = \text{lab} y \land \text{fset_pred} \hspace{0.5em} \psi \hspace{0.5em} (\text{sub} x) \hspace{0.5em} (\text{sub} y)$
$t_1 = t_2$
Higher-Order Logic

- Simply typed set theory with ML-style polymorphism
- Cannot handle proper classes
Higher-Order Logic

- Simply typed set theory with ML-style polymorphism
- Cannot handle proper classes
- Primitive type definitions
Higher-Order Logic

- Simply typed set theory with ML-style polymorphism
- Cannot handle proper classes
- Primitive type definitions

Goal: Reduce (co)datatype specification to primitive type definitions
(Co)datatypes in interactive theorem provers

- PVS: axiomatic, monolithic (co)datatypes
- Agda, Coq: built-in (co)datatypes
- HOL based provers: definitional datatypes
(Co)datatypes in interactive theorem provers

- PVS: axiomatic, monolithic (co)datatypes
- Agda, Coq: built-in (co)datatypes
- HOL based provers: definitional datatypes
  - Melham–Gunter approach
    - Fixed universe for recursive, freely generated datatypes
    - Simulates nested recursion by mutual recursion
    - Used in HOL4, HOL Light, Isabelle/HOL, ...
Beyond Melham–Gunter

- Codatatypes
- Mixture of codatatypes and datatypes
- Non-free structures (e.g. fset)
- “Real” nested recursion
Type constructors are not just operators on types!
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

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

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

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

\[ Fset_1 : (\alpha, \beta) \ F \rightarrow \alpha \ set \]
\[ Fset_2 : (\alpha, \beta) \ F \rightarrow \beta \ set \]
Type Constructors are Containers

\[ Fset_1 : (\alpha, \beta) \ F \rightarrow \alpha \ set \]
\[ Fset_2 : (\alpha, \beta) \ F \rightarrow \beta \ set \]

\[ Fset_i \circ \text{Fmap } f_1 \ f_2 = \text{image } f_i \circ \text{Fset}_i \]
Type Constructors Act on Sets

$$\text{Fin } A_1 A_2 = \{ z \mid \text{Fset}_1 \ z \subseteq A_1 \land \text{Fset}_2 \ z \subseteq A_2 \}$$

$$A_1 : \alpha \text{ set} \quad A_2 : \beta \text{ set}$$

$$\text{Fin } A_1 A_2 : (\alpha, \beta) \text{ F set}$$
Type Constructors Act on Sets

\[ \text{Fin } A_1 A_2 = \{ z \mid \text{Fset}_1 z \subseteq A_1 \land \text{Fset}_2 z \subseteq A_2 \}\]
Type Constructors Act on Sets

\[ \text{Fin } A_1 \ A_2 = \{ z \mid \text{Fset}_1 \ z \subseteq A_1 \land \text{Fset}_2 \ z \subseteq A_2 \} \]

\[ \forall i \in \{1, 2\}. \ \forall x \in \text{Fset}_i \ z. \ f_i \ x = g_i \ x \ \Rightarrow \ F\text{map} \ f_1 \ f_2 \ z = F\text{map} \ g_1 \ g_2 \ z \]
Type Constructors are Bounded

Fbd: infinite cardinal

Diagram:
- A triangle labeled \((\alpha, \beta) \ F\)
- One vertex labeled \(a_1\)
- Another vertex labeled \(a_2\)
- A vertex labeled \(b\)
- Fset_1 connected to \(a_1\) and \(a_2\)
- Fset_2 connected to \(b\)

\(a_1\) set \(\alpha\) set
\(a_2\) set \(\beta\) set

\(a_1, a_2\) and \(b\) connected by dashed lines.
Type Constructors are Bounded

Fbd: infinite cardinal

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

\[\begin{array}{c}
\alpha \text{ set} \\
\beta \text{ set}
\end{array}\]

\[|Fset_i z| \leq Fbd\]
Type Constructors are Bounded

Fbd: infinite cardinal

\[|Fset_i z| \leq \text{Fbd}\]
Type Constructors are Bounded

Fbd: infinite cardinal

| Fset$_i$ z | ≤ Fbd
| Fin $A_1 A_2$ | ≤ $(|A_1| + |A_2| + 2)^{Fbd}$
Type Constructors are Bounded Natural Functors

binary BNF is a tuple \((F, Fmap, Fset, Fbd)\) satisfying
Type Constructors are Bounded Natural Functors

binary BNF is a tuple \((F, \text{Fmap}, \text{Fset}, \text{Fbd})\) satisfying

- \((F, \text{Fmap})\) is a binary functor.
- For all \(\alpha_1\), \(\text{Fset}_1\) is a natural transformation between \(((\alpha_1, \_), F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- For all \(\alpha_2\), \(\text{Fset}_2\) is a natural transformation between \(((\_, \alpha_2), F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- If \(\forall a \in \text{Fset}_i \; x. \; f_i \; a = g_i \; a\) for all \(i \in \{1, 2\}\), then \(\text{Fmap} \; f_1 \; f_2 \; x = \text{Fmap} \; g_1 \; g_2 \; x\).
- The following cardinal-bound conditions hold:
  - a. \(\forall x : (\alpha_1, \alpha_2) \; F. \; |\text{Fset}_i \; x| \leq \text{Fbd}\) for \(i \in \{1, 2\}\);
  - b. \(|\text{Fin} \; A_1 \; A_2| \leq (|A_1| + |A_2| + 2)^\text{Fbd}\).
- \((F, \text{Fmap})\) preserves weak pullbacks.
Type Constructors are Bounded Natural Functors

binary BNF is a tuple \((F, \text{Fmap}, \text{Fset}, \text{Fbd})\) satisfying

- \((F, \text{Fmap})\) is a binary functor.
- For all \(\alpha_1\), \(\text{Fset}_1\) is a natural transformation between \(((\alpha_1, \_\_\) \(F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- For all \(\alpha_2\), \(\text{Fset}_2\) is a natural transformation between \((\_\_, \alpha_2 \) \(F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- If \(\forall a \in \text{Fset}_i \ x. \ f_i a = g_i a\) for all \(i \in \{1, 2\}\), then \(\text{Fmap} f_1 f_2 x = \text{Fmap} g_1 g_2 x\).
- The following cardinal-bound conditions hold:
  a. \(\forall x : (\alpha_1, \alpha_2) \ F. \ |\text{Fset}_i x| \leq \text{Fbd}\) for \(i \in \{1, 2\}\);  
  b. \(|\text{Fin} A_1 A_2| \leq (|A_1| + |A_2| + 2)^\text{Fbd}\).  
- \((F, \text{Fmap})\) preserves weak pullbacks.
Type Constructors are Bounded Natural Functors

binary BNF is a tuple \((F, \text{Fmap}, \text{Fset}, \text{Fbd})\) satisfying

- \((F, \text{Fmap})\) is a binary functor.
- For all \(\alpha_1\), \(\text{Fset}_1\) is a natural transformation between \(((\alpha_1, \_), F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- For all \(\alpha_2\), \(\text{Fset}_2\) is a natural transformation between \((\_, \alpha_2, F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- If \(\forall a \in \text{Fset}_i\ x\ .\ f_i a = g_i a\) for all \(i \in \{1, 2\}\), then 
  \(\text{Fmap} f_1 f_2 x = \text{Fmap} g_1 g_2 x\).
- The following cardinal-bound conditions hold:
  a. \(\forall x : (\alpha_1, \alpha_2) F, \ |\text{Fset}_i x| \leq \text{Fbd}\) for \(i \in \{1, 2\}\);
  b. \(|\text{Fin} A_1 A_2| \leq (|A_1| + |A_2| + 2)^{\text{Fbd}}\).
- \((F, \text{Fmap})\) preserves weak pullbacks.
Type Constructors are Bounded Natural Functors

binary BNF is a tuple \((F, \text{Fmap}, \text{Fset}, \text{Fbd})\) satisfying

- \((F, \text{Fmap})\) is a binary functor.
- For all \(\alpha_1\), \(\text{Fset}_1\) is a natural transformation between \(((\alpha_1, _) \ F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- For all \(\alpha_2\), \(\text{Fset}_2\) is a natural transformation between \(((_, \alpha_2) \ F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- If \(\forall a \in \text{Fset}_i \ x. \ f_i a = g_i a\) for all \(i \in \{1, 2\}\), then \(\text{Fmap} \ f_1 \ f_2 \ x = \text{Fmap} \ g_1 \ g_2 \ x\).
- The following cardinal-bound conditions hold:
  a. \(\forall x : (\alpha_1, \alpha_2) \ F. \ |\text{Fset}_i \ x| \leq \text{Fbd}\) for \(i \in \{1, 2\}\);
  b. \(|\text{Fin} \ A_1 \ A_2| \leq (|A_1| + |A_2| + 2)^\text{Fbd}\).
- \((F, \text{Fmap})\) preserves weak pullbacks.
Type Constructors are Bounded Natural Functors

binary BNF is a tuple \((F, \text{Fmap}, \text{Fset}, \text{Fbd})\) satisfying

- \((F, \text{Fmap})\) is a binary functor.
- For all \(\alpha_1\), \(\text{Fset}_1\) is a natural transformation between \(((\alpha_1, \_), F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- For all \(\alpha_2\), \(\text{Fset}_2\) is a natural transformation between \(((\_, \alpha_2), F, \text{Fmap})\) and \((\text{set}, \text{image})\).
- If \(\forall a \in \text{Fset}_i x. \ f_i a = g_i a\) for all \(i \in \{1, 2\}\), then \(\text{Fmap} f_1 f_2 x = \text{Fmap} g_1 g_2 x\).
- The following cardinal-bound conditions hold:
  a. \(\forall x : (\alpha_1, \alpha_2) F. \ |\text{Fset}_i x| \leq \text{Fbd}\) for \(i \in \{1, 2\}\);
  b. \(|\text{Fin } A_1 A_2| \leq (|A_1| + |A_2| + 2)^\text{Fbd}\).
- \((F, \text{Fmap})\) preserves weak pullbacks.
What are BNFs good for?

They ...
What are BNFs good for?

They ...

- cover basic type constructors (e.g. $+$, $\times$, unit, and $\alpha \to \beta$ for fixed $\alpha$)
What are BNFs good for?

They ...

- 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 BNFs good for?

They ...

- 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
What are BNFs good for?

They...

- 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 and final coalgebras
What are BNFs good for?

They ...

- 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 and final coalgebras
- are closed under initial algebras and final coalgebras
What are BNFs good for?

They ...

- 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 and final coalgebras
- are closed under initial algebras and final coalgebras
- make initial algebras and final coalgebras expressible in HOL
From user specifications to (co)datatypes

- datatype \( \alpha \text{ list} = \text{Nil} \mid \text{Cons } \alpha (\alpha \text{ list}) \)
- Abstract to \( \beta = \text{unit} + \alpha \times \beta \)
- Prove \( (\alpha, \beta) F = \text{unit} + \alpha \times \beta \) is BNF
- Define F-algebras
- Construct initial algebra \( (\alpha \ IF, \ fld) \)
- Define iterator iter
- Prove characteristic theorems
- Prove that IF is a BNF
From user specifications to (co)datatatypes

- codatatype $\alpha \text{llist} = \text{Nil} \mid \text{Cons } \alpha (\alpha \text{llist})$
- Abstract to $\beta = \text{unit} + \alpha \times \beta$
- Prove $(\alpha, \beta) \ F = \text{unit} + \alpha \times \beta$ is BNF
- Define F-coalgebras
- Construct final coalgebra $(\alpha \ JF, \ unf)$
- Define coiterator coiter
- Prove characteristic theorems
- Prove that $JF$ is a BNF
Algebras, Coalgebras & Morphisms

In category theory:

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

\[ F : A \to A \]

\[ \text{s} \]

\[ A \]
Algebras, Coalgebras & Morphisms

In category theory:

\[
\begin{align*}
\beta &= (\alpha, \beta) \\
F &= \text{Functor}
\end{align*}
\]

\[
\begin{align*}
F(A) &\xrightarrow{s} A \\
F(f) &\xrightarrow{S_A} S_A \\
A &\xrightarrow{f} B
\end{align*}
\]
Algebras, Coalgebras & Morphisms

In category theory:

\[ F(A) \xrightarrow{s} A \]

\[ A \xrightarrow{s} F(A) \]

\[ F(f) \quad \text{and} \quad A \xrightarrow{f} B \]

\[ F(A) \xrightarrow{S_A} A \quad \text{and} \quad F(B) \xrightarrow{S_B} B \]
Algebras, Coalgebras & Morphisms

In category theory:

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

\[
\text{In category theory:} \quad FA \quad A \\
\quad \downarrow s \quad \downarrow s \\
\quad A \quad FA \\
\]

\[
F(f) \quad F(f) \\
\downarrow f \quad \downarrow f \\
A \quad B \\
\]

\[
A \quad B \\
\downarrow s \quad \downarrow s \\
FA \quad FB \\
\]

\[
A \quad f \\
\downarrow s \quad \downarrow s \\
FA \quad FB \\
\]
Algebras, Coalgebras & Morphisms

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

In HOL:

\[
\begin{align*}
\text{Fin } U_\alpha A &\quad \text{Fmap id } f \\
S_A &\quad f \\
A &\quad B
\end{align*}
\]

\[
\begin{align*}
A &\quad f \\
S_A &\quad S_B \\
\text{Fin } U_\alpha A &\quad \text{Fin } U_\alpha B
\end{align*}
\]
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
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
  $$\Rightarrow$$ 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

⇒ 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

⇒ \((\alpha \text{ JF}, \text{unf} : \alpha \text{ JF} \rightarrow (\alpha, \alpha \text{ JF}) F)\)
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

\[\Rightarrow\] Have a bound for its cardinality

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

- 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)

\[\Rightarrow\] Have a bound for its cardinality

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

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

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

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

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

\[ (\alpha, \alpha \mathcal{F}) \mathcal{F} \xrightarrow{\text{Fmap id } (\text{iter } s)} (\alpha, \beta) \mathcal{F} \]

\[ \alpha \mathcal{F} \xrightarrow{\text{iter } s} \beta \]
Iteration & Coiteration

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

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

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

- Given \( s : \beta \to (\alpha, \beta) \mathcal{F} \)
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 $$(\alpha \text{ JF}, \text{unf})$$
Induction & Coinduction

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

- Given \( \phi : \alpha \text{IF} \rightarrow \text{bool} \)
Induction & Coinduction

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

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

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

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

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

- 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

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

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

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

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

\[ \forall x y. \psi x y \Rightarrow \text{Fpred Eq } \psi (\text{unf } x) (\text{unf } y) \]
\[ \forall x y. \psi x y \Rightarrow x = y \]
Preservation of BNF Properties

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

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

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

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

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

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

**Theorem**

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

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

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

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

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

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

**Theorem**

\((\text{IF, IFmap, IFset, } 2^{\text{Fbd}}) \text{ is an BNF}\)
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

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

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

Theorem
\( (\text{JF}, \text{JFmap}, \text{JFset}, \text{Fbd}^{\text{Fbd}}) \) is an BNF
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 (co)recursion involving arbitrary combinations of datatypes, codatatypes, 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 (co)recursion involving arbitrary combinations of datatypes, codatatypes, and custom BNFs.
- Adapt insights from category theory in HOL’s restrictive type system
Thank you for your attention!
Questions?
Foundational, Compositional (Co)datatypes for Higher-Order Logic
Category Theory Applied to Theorem Proving

Dmitriy Traytel    Andrei Popescu    Jasmin Christian Blanchette

November 13, 2015