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

Category Theory Applied to Theorem Proving

Dmitriy Traytel Andrei Popescu Jasmin Christian Blanchette

November 13, 2015

Technische Universität München

Outline

Motivation

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

Motivation

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

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

Motivation

`datatype` α list $=$ Nil $|$ Cons α (α list)

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

Prove $$\frac{\varphi\ \text{Nil} \qquad \forall x\ xs.\ \varphi\ xs \Rightarrow \varphi\ (\mathit{Cons}\ x\ xs)}{\forall xs.\ \varphi\ xs}$$

Motivation

$\texttt{datatype}\ \alpha\ \mathsf{list} = \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list})$
$\texttt{codatatype}\ \ \alpha\ \mathsf{tree_l} = \mathsf{Node}\ (\mathsf{lab}\colon \alpha)\ (\mathsf{sub}\colon (\alpha\ \mathsf{tree_l})\ \mathsf{list})$

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

Prove $\quad \dfrac{\varphi\ \mathsf{Nil} \qquad \forall x\ xs.\ \varphi\ xs \Rightarrow \varphi\ (Cons\ x\ xs)}{\forall xs.\ \varphi\ xs}$

Motivation

`datatype` α list $=$ Nil $|$ Cons α (α list)
`codatatype` α tree_l $=$ Node (lab: α) (sub: (α tree_l) list)

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

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

Motivation

$\texttt{datatype}\ \alpha\ \mathsf{list} = \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list})$
$\texttt{codatatype}\ \ \alpha\ \mathsf{tree_l} = \mathsf{Node}\ (\mathsf{lab}\colon \alpha)\ (\mathsf{sub}\colon (\alpha\,\mathsf{tree_l})\ \mathsf{list})$

Resolve $\beta = \mathsf{unit} + \alpha \times \beta$ minimally
and $\gamma = \alpha \times \gamma\ \mathsf{list}$ maximally

Prove $$\frac{\varphi\ \mathsf{Nil} \qquad \forall x\ xs.\ \varphi\ xs \Rightarrow \varphi\ (Cons\ x\ xs)}{\forall xs.\ \varphi\ xs}$$

and $$\frac{\psi\ t_1\ t_2 \qquad \forall x\ y.\ \psi\ x\ y \Rightarrow \mathsf{lab}\ x = \mathsf{lab}\ y \wedge \mathsf{list\_pred}\ \psi\ (sub\ x)\ (sub\ y)}{t_1 = t_2}$$

Motivation

`datatype` α list $=$ Nil $|$ Cons α (α list)
`codatatype` α tree$_\mathsf{l}$ $=$ Node (lab: α) (sub: (α tree$_\mathsf{l}$) fset)

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

Prove
$$\frac{\varphi\ \mathsf{Nil} \qquad \forall x\ xs.\ \varphi\ xs \Rightarrow \varphi\ (Cons\ x\ xs)}{\forall xs.\ \varphi\ xs}$$

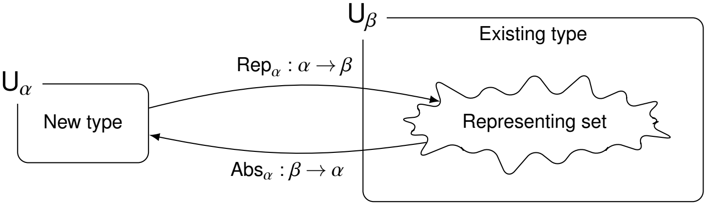
and
$$\frac{\psi\ t_1\ t_2 \qquad \forall x\ y.\ \psi\ x\ y \Rightarrow \mathsf{lab}\ x = \mathsf{lab}\ y \wedge \mathsf{fset\_pred}\ \psi\ (sub\ x)\ (sub\ y)}{t_1 = t_2}$$

Higher-Order Logic

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

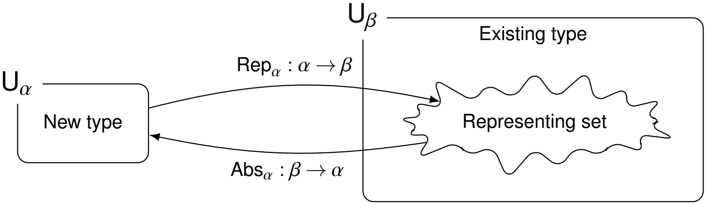
Higher-Order Logic

- 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, ...

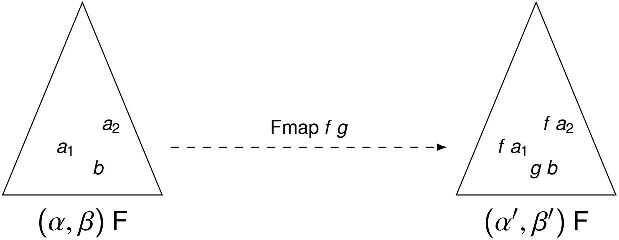
Beyond Melham–Gunter

- Codatatypes
- Mixture of codatatypes and datatypes
- Non-free structures (e.g. fset)
- “Real” nested recursion

Type constructors are not just operators on types!

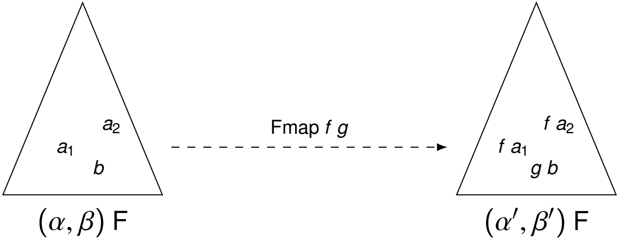
Type Constructors are Functors

$$\mathsf{Fmap} : (\alpha \to \alpha') \to (\beta \to \beta') \to (\alpha, \beta)\ \mathsf{F} \to (\alpha', \beta')\ \mathsf{F}$$



Type Constructors are Functors

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

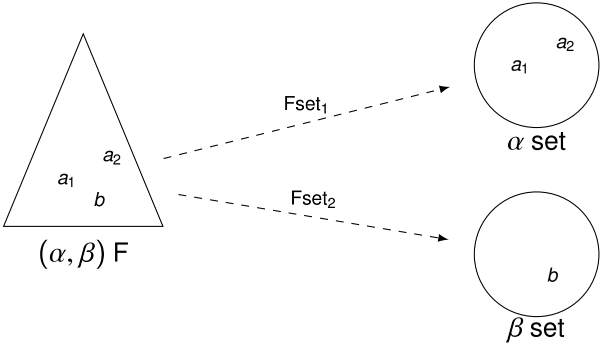


$$\begin{aligned} \text{Fmap id id} &= \text{id} \\ \text{Fmap } f_1\ f_2 \circ \text{Fmap } g_1\ g_2 &= \text{Fmap } (f_1 \circ g_2)\ (f_2 \circ g_2) \end{aligned}$$

Type Constructors are Containers

$$\mathsf{Fset}_1 : (\alpha, \beta)\ \mathsf{F} \to \alpha\ \mathsf{set}$$

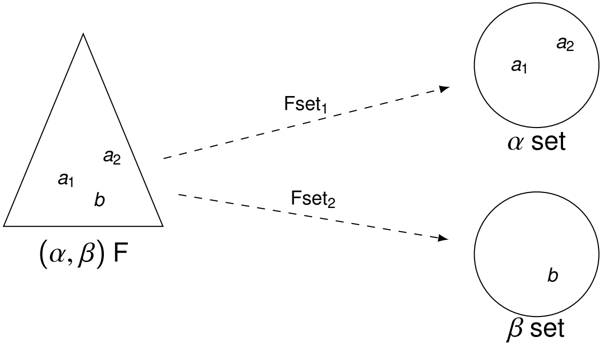
$$\mathsf{Fset}_2 : (\alpha, \beta)\ \mathsf{F} \to \beta\ \mathsf{set}$$



Type Constructors are Containers

$$\mathsf{Fset}_1 : (\alpha, \beta)\ \mathsf{F} \to \alpha\ \mathsf{set}$$

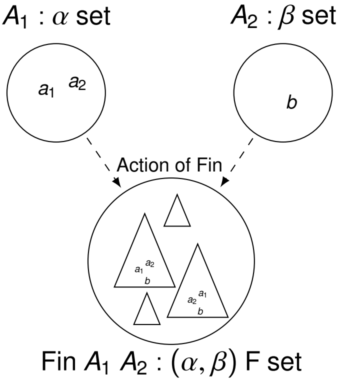
$$\mathsf{Fset}_2 : (\alpha, \beta)\ \mathsf{F} \to \beta\ \mathsf{set}$$



$$\mathsf{Fset}_i \circ \mathsf{Fmap}\ f_1\ f_2 \quad = \quad \mathsf{image}\ f_i \circ \mathsf{Fset}_i$$

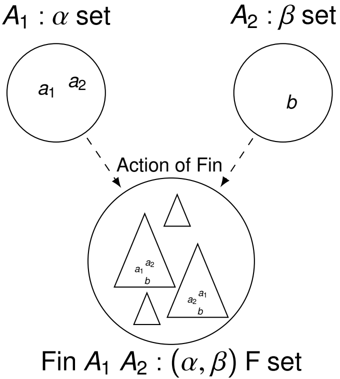
Type Constructors Act on Sets

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



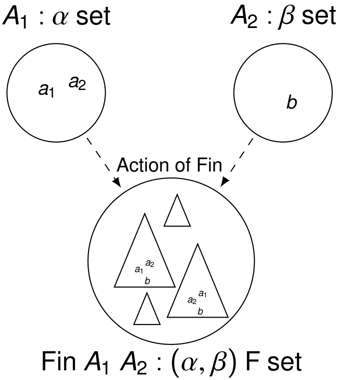
Type Constructors Act on Sets

$$\mathsf{Fin}\ A_1\ A_2 = \{z \mid \mathsf{Fset}_1\ z \subseteq A_1 \wedge \mathsf{Fset}_2\ z \subseteq A_2\}$$



Type Constructors Act on Sets

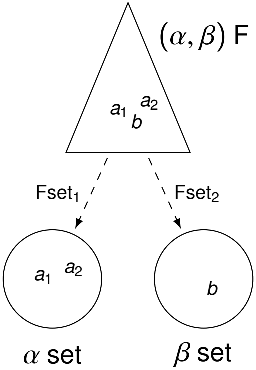
$$\mathrm{Fin}\ A_1\ A_2 = \{z \mid \mathrm{Fset}_1\ z \subseteq A_1 \wedge \mathrm{Fset}_2\ z \subseteq A_2\}$$



$$\forall i \in \{1, 2\}.\ \forall x \in \mathrm{Fset}_i\ z.\ f_i\ x = g_i\ x \quad \Rightarrow \quad \mathrm{Fmap}\ f_1\ f_2\ z = \mathrm{Fmap}\ g_1\ g_2\ z$$

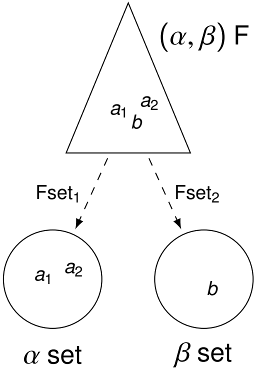
Type Constructors are Bounded

Fbd: infinite cardinal



Type Constructors are Bounded

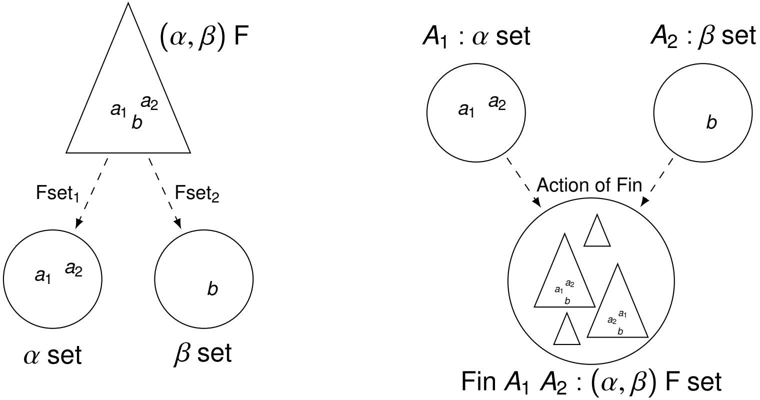
Fbd: infinite cardinal



$$|\mathsf{Fset}_i\ z| \quad \leq \quad \mathsf{Fbd}$$

Type Constructors are Bounded

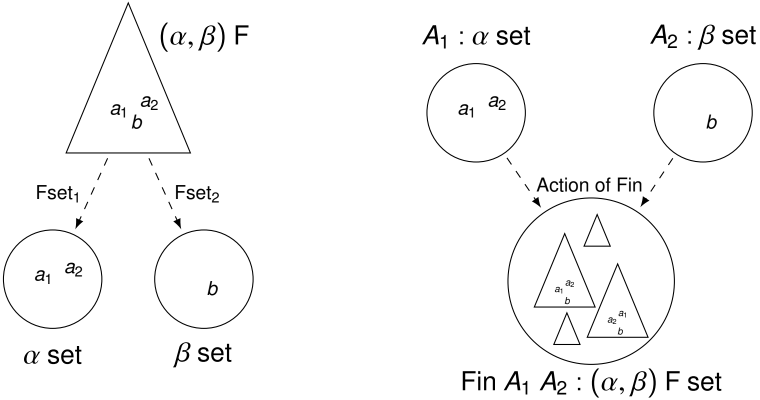
Fbd: infinite cardinal



$$|\mathsf{Fset}_i\ z| \quad \leq \quad \mathsf{Fbd}$$

Type Constructors are Bounded

Fbd: infinite cardinal



$$
\begin{aligned}
|\mathrm{Fset}_i\ z| &\leq \mathrm{Fbd} \\
|\mathrm{Fin}\ A_1\ A_2| &\leq (|A_1| + |A_2| + 2)^{\mathrm{Fbd}}
\end{aligned}
$$

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, Fmap, Fset, Fbd) satisfying

- (F, Fmap) is a binary functor.
- For all α_1, Fset_1 is a natural transformation between $((\alpha_1, \_)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- For all α_2, Fset_2 is a natural transformation between $((\_, \alpha_2)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- If $\forall a \in \mathsf{Fset}_i\ x.\ f_i\ a = g_i\ a$ for all $i \in \{1, 2\}$, then $\mathsf{Fmap}\ f_1\ f_2\ x = \mathsf{Fmap}\ g_1\ g_2\ x$.
- The following cardinal-bound conditions hold:
  a. $\forall x : (\alpha_1, \alpha_2)\ \mathsf{F}.\ \ |\mathsf{Fset}_i\ x| \leq \mathsf{Fbd}$ for $i \in \{1,2\}$;
  b. $|\mathsf{Fin}\ A_1\ A_2| \leq (|A_1| + |A_2| + 2)^{\mathsf{Fbd}}$.
- (F, Fmap) preserves weak pullbacks.

Type Constructors are Bounded Natural Functors

binary BNF is a tuple (F, Fmap, Fset, Fbd) satisfying

- (F, Fmap) is a binary functor.
- For all α_1, Fset_1 is a natural transformation between $((\alpha_1, \_)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- For all α_2, Fset_2 is a natural transformation between $((\_, \alpha_2)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- If $\forall a \in \mathsf{Fset}_i\ x.\ \ f_i\ a = g_i\ a$ for all $i \in \{1, 2\}$, then $\mathsf{Fmap}\ f_1\ f_2\ x = \mathsf{Fmap}\ g_1\ g_2\ x$.
- The following cardinal-bound conditions hold:
  a. $\forall x : (\alpha_1, \alpha_2)\ \mathsf{F}.\ \ |\mathsf{Fset}_i\ x| \leq \mathsf{Fbd}$ for $i \in \{1,2\}$;
  b. $|\mathsf{Fin}\ A_1\ A_2| \leq (|A_1| + |A_2| + 2)^{\mathsf{Fbd}}$.
- (F, Fmap) preserves weak pullbacks.

Type Constructors are Bounded Natural Functors

binary BNF is a tuple (F, Fmap, Fset, Fbd) satisfying

- (F, Fmap) is a binary functor.
- For all α_1, Fset_1 is a natural transformation between $((\alpha_1, \_)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- For all α_2, Fset_2 is a natural transformation between $((\_, \alpha_2)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- If $\forall a \in \mathsf{Fset}_i\ x.\ f_i\ a = g_i\ a$ for all $i \in \{1, 2\}$, then $\mathsf{Fmap}\ f_1\ f_2\ x = \mathsf{Fmap}\ g_1\ g_2\ x$.
- The following cardinal-bound conditions hold:
  a. $\forall x : (\alpha_1, \alpha_2)\ \mathsf{F}.\ \ |\mathsf{Fset}_i\ x| \leq \mathsf{Fbd}$ for $i \in \{1,2\}$;
  b. $|\mathsf{Fin}\ A_1\ A_2| \leq (|A_1| + |A_2| + 2)^{\mathsf{Fbd}}$.
- (F, Fmap) preserves weak pullbacks.

Type Constructors are Bounded Natural Functors

binary BNF is a tuple (F, Fmap, Fset, Fbd) satisfying

- (F, Fmap) is a binary functor.
- For all α_1, Fset_1 is a natural transformation between $((\alpha_1, \_)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- For all α_2, Fset_2 is a natural transformation between $((\_, \alpha_2)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- If $\forall a \in \mathsf{Fset}_i\ x.\ f_i\ a = g_i\ a$ for all $i \in \{1, 2\}$, then $\mathsf{Fmap}\ f_1\ f_2\ x = \mathsf{Fmap}\ g_1\ g_2\ x$.
- The following cardinal-bound conditions hold:
  a. $\forall x : (\alpha_1, \alpha_2)\ \mathsf{F}.\ \ |\mathsf{Fset}_i\ x| \leq \mathsf{Fbd}$ for $i \in \{1,2\}$;
  b. $|\mathsf{Fin}\ A_1\ A_2| \leq (|A_1| + |A_2| + 2)^{\mathsf{Fbd}}$.
- (F, Fmap) preserves weak pullbacks.

Type Constructors are Bounded Natural Functors

binary BNF is a tuple (F, Fmap, Fset, Fbd) satisfying

- (F, Fmap) is a binary functor.
- For all α_1, Fset_1 is a natural transformation between $((\alpha_1, \_)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- For all α_2, Fset_2 is a natural transformation between $((\_, \alpha_2)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).
- If $\forall a \in \mathsf{Fset}_i\ x.\ \ f_i\ a = g_i\ a$ for all $i \in \{1, 2\}$, then $\mathsf{Fmap}\ f_1\ f_2\ x = \mathsf{Fmap}\ g_1\ g_2\ x$.
- The following cardinal-bound conditions hold:
  a. $\forall x : (\alpha_1, \alpha_2)\ \mathsf{F}.\ \ |\mathsf{Fset}_i\ x| \leq \mathsf{Fbd}$ for $i \in \{1,2\}$;
  b. $|\mathsf{Fin}\ A_1\ A_2| \leq (|A_1| + |A_2| + 2)^{\mathsf{Fbd}}$.
- (F, 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 α)

What are BNFs good for?

They ...

- cover basic type constructors (e.g. $+$, $\times$, unit, and $\alpha \to \beta$ for fixed α)
- cover non-free type constructors (e.g. fset, cset)

What are BNFs good for?

They ...

- cover basic type constructors (e.g. $+$, $\times$, unit, and $\alpha \to \beta$ for fixed α)
- 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 $\alpha \to \beta$ for fixed α)
- 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 $\alpha \to \beta$ for fixed α)
- 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 α)
- 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` α list $=$ Nil $|$ Cons α (α list)
- Abstract to $\beta = \text{unit} + \alpha \times \beta$
- Prove (α, β) F $= \text{unit} + \alpha \times \beta$ is BNF
- Define F-algebras
- Construct initial algebra (α IF, fld)
- Define iterator iter
- Prove characteristic theorems
- Prove that IF is a BNF

From user specifications to (co)datatypes

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

Algebras, Coalgebras & Morphisms

In category theory:

$$\begin{tikzcd} \mathsf{F}A \arrow[d, dashed, "s"] \\ A \end{tikzcd}$$

Algebras, Coalgebras & Morphisms

In category theory:

$$\begin{tikzcd} \mathrm{F}A \arrow[d, dashed, "s"] \\ A \end{tikzcd}$$

$$\begin{tikzcd} \mathrm{F}A \arrow[r, dashed, "\mathrm{F}(f)"] \arrow[d, dashed, "s_A"'] & \mathrm{F}B \arrow[d, dashed, "s_B"] \\ A \arrow[r, dashed, "f"] & B \end{tikzcd}$$

Algebras, Coalgebras & Morphisms

In category theory:

$$\begin{tikzcd} \mathrm{F}A \arrow[d, dashed, "s"] \\ A \end{tikzcd} \qquad\qquad \begin{tikzcd} A \arrow[d, dashed, "s"] \\ \mathrm{F}A \end{tikzcd}$$

$$\begin{tikzcd} \mathrm{F}A \arrow[r, dashed, "\mathrm{F}(f)"] \arrow[d, dashed, "s_A"'] & \mathrm{F}B \arrow[d, dashed, "s_B"] \\ A \arrow[r, dashed, "f"] & B \end{tikzcd}$$

Algebras, Coalgebras & Morphisms

In category theory:

$$\begin{tikzcd} \mathsf{F}A \arrow[d, dashed, "s"] \\ A \end{tikzcd} \qquad \begin{tikzcd} A \arrow[d, dashed, "s"] \\ \mathsf{F}A \end{tikzcd}$$

$$\begin{tikzcd} \mathsf{F}A \arrow[r, dashed, "\mathsf{F}(f)"] \arrow[d, dashed, "s_A"'] & \mathsf{F}B \arrow[d, dashed, "s_B"] \\ A \arrow[r, dashed, "f"] & B \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, dashed, "f"] \arrow[d, dashed, "s_A"'] & B \arrow[d, dashed, "s_B"] \\ \mathsf{F}A \arrow[r, dashed, "\mathsf{F}(f)"] & \mathsf{F}B \end{tikzcd}$$

Algebras, Coalgebras & Morphisms

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

In HOL:

$$\begin{tikzcd} \mathrm{Fin}\ \mathrm{U}_\alpha\ A \arrow[d, dashed, "s"] \\ A \end{tikzcd} \qquad \begin{tikzcd} A \arrow[d, dashed, "s"] \\ \mathrm{Fin}\ \mathrm{U}_\alpha\ A \end{tikzcd}$$

$$\begin{tikzcd} \mathrm{Fin}\ \mathrm{U}_\alpha\ A \arrow[r, dashed, "\mathrm{Fmap\ id}\ f"] \arrow[d, dashed, "s_A"'] & \mathrm{Fin}\ \mathrm{U}_\alpha\ B \arrow[d, dashed, "s_B"] \\ A \arrow[r, dashed, "f"] & B \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, dashed, "f"] \arrow[d, dashed, "s_A"'] & B \arrow[d, dashed, "s_B"] \\ \mathrm{Fin}\ \mathrm{U}_\alpha\ A \arrow[r, dashed, "\mathrm{Fmap\ id}\ f"] & \mathrm{Fin}\ \mathrm{U}_\alpha\ B \end{tikzcd}$$

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

$$\beta = (\alpha, \beta)\ \mathsf{F}$$

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

$$\beta = (\alpha, \beta)\ \mathsf{F}$$

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

$$\Rightarrow (\alpha\ \mathsf{IF}, \mathsf{fld} : (\alpha, \alpha\ \mathsf{IF})\ \mathsf{F} \to \alpha\ \mathsf{IF})$$

Initial Algebras & Final Coalgebras

$$\beta = (\alpha, \beta)\ \mathsf{F}$$

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

$$\Rightarrow (\alpha\ \mathsf{IF}, \mathsf{fld} : (\alpha, \alpha\ \mathsf{IF})\ \mathsf{F} \to \alpha\ \mathsf{IF})$$

- 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

Initial Algebras & Final Coalgebras

$$\beta = (\alpha, \beta)\ \mathsf{F}$$

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

$$\Rightarrow (\alpha\ \mathsf{IF},\ \mathsf{fld} : (\alpha, \alpha\ \mathsf{IF})\ \mathsf{F} \to \alpha\ \mathsf{IF})$$

- 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

$$\Rightarrow (\alpha\ \mathsf{JF},\ \mathsf{unf} : \alpha\ \mathsf{JF} \to (\alpha, \alpha\ \mathsf{JF})\ \mathsf{F})$$

Iteration & Coiteration

$\beta = (\alpha, \beta)\ \mathsf{F}$

- Given $s : (\alpha, \beta)\ \mathsf{F} \to \beta$

Iteration & Coiteration

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

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

$$\begin{tikzcd}
(\alpha, \alpha \text{ IF}) \text{ F} \arrow[r, dashed, "\text{Fmap id (iter } s)"] \arrow[d, dashed, "\text{fld}"'] & (\alpha, \beta) \text{ F} \arrow[d, dashed, "s"] \\
\alpha \text{ IF} \arrow[r, dashed, "\text{iter } s"] & \beta
\end{tikzcd}$$

Iteration & Coiteration

$\beta = (\alpha, \beta)\ \mathsf{F}$

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

$$
\begin{tikzcd}
(\alpha, \alpha\ \mathsf{IF})\ \mathsf{F} \arrow[r, dashed, "\mathsf{Fmap\ id\ (iter}\ s)"] \arrow[d, dashed, "\mathsf{fld}"'] & (\alpha, \beta)\ \mathsf{F} \arrow[d, dashed, "s"] \\
\alpha\ \mathsf{IF} \arrow[r, dashed, "\mathsf{iter}\ s"] & \beta
\end{tikzcd}
$$

- Given $s : \beta \to (\alpha, \beta)\ \mathsf{F}$

Iteration & Coiteration

$\beta = (\alpha, \beta)\ \mathsf{F}$

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

$$\begin{tikzcd}
(\alpha, \alpha\ \mathsf{IF})\ \mathsf{F} \arrow[r, dashed, "\mathsf{Fmap\ id\ (iter}\ s)"] \arrow[d, dashed, "\mathsf{fld}"'] & (\alpha, \beta)\ \mathsf{F} \arrow[d, dashed, "s"] \\
\alpha\ \mathsf{IF} \arrow[r, dashed, "\mathsf{iter}\ s"] & \beta
\end{tikzcd}$$

- Given $s : \beta \to (\alpha, \beta)\ \mathsf{F}$
- Obtain unique morphism coiter s from (U_β, s) to $(\alpha\ \mathsf{JF}, \mathsf{unf})$

$$\begin{tikzcd}
\beta \arrow[r, dashed, "\mathsf{coiter}\ s"] \arrow[d, dashed, "s"'] & \alpha\ \mathsf{IF} \arrow[d, dashed, "\mathsf{unf}"] \\
(\alpha, \beta)\ \mathsf{F} \arrow[r, dashed, "\mathsf{Fmap\ id\ (coiter}\ s)"] & (\alpha, \alpha\ \mathsf{IF})\ \mathsf{F}
\end{tikzcd}$$

Induction & Coinduction

$$\beta = (\alpha, \beta)\ \mathsf{F}$$

- Given $\varphi : \alpha\ \mathsf{IF} \rightarrow \mathsf{bool}$

Induction & Coinduction

$\beta = (\alpha, \beta)\ \mathsf{F}$

- Given $\varphi : \alpha\ \mathsf{IF} \to \mathsf{bool}$
- Abstract induction principle

$$\frac{\forall z.\ (\forall x \in \mathsf{Fset}_2\ z.\ \varphi\ x) \Rightarrow \varphi\ (\mathsf{fld}\ z)}{\forall x.\ \varphi\ x}$$

Induction & Coinduction

$\beta = (\alpha, \beta)\ \mathsf{F}$

- Given $\varphi : \alpha\ \mathsf{IF} \to \mathsf{bool}$
- Abstract induction principle

$$\frac{\forall z.\ (\forall x \in \mathsf{Fset}_2\ z.\ \varphi\ x) \Rightarrow \varphi\ (\mathsf{fld}\ z)}{\forall x.\ \varphi\ x}$$

- Given $\psi : \alpha\ \mathsf{JF} \to \alpha\ \mathsf{JF} \to \mathsf{bool}$

Induction & Coinduction

$\beta = (\alpha, \beta)\ \mathsf{F}$

- Given $\varphi : \alpha\ \mathsf{IF} \to \mathsf{bool}$
- Abstract induction principle

$$\frac{\forall z.\ (\forall x \in \mathsf{Fset}_2\ z.\ \varphi\ x) \Rightarrow \varphi\ (\mathsf{fld}\ z)}{\forall x.\ \varphi\ x}$$

- Given $\psi : \alpha\ \mathsf{JF} \to \alpha\ \mathsf{JF} \to \mathsf{bool}$
- Abstract coinduction principle

$$\frac{\forall x\ y.\ \psi\ x\ y \Rightarrow \mathsf{Fpred}\ \mathsf{Eq}\ \psi\ (\mathsf{unf}\ x)\ (\mathsf{unf}\ y)}{\forall x\ y.\ \psi\ x\ y \Rightarrow x = y}$$

Preservation of BNF Properties

$\beta = (\alpha, \beta)\ \mathsf{F}$

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

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

Preservation of BNF Properties

$\beta = (\alpha, \beta)\ \mathsf{F}$

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

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

Theorem

$(\mathsf{IF}, \mathsf{IFmap}, \mathsf{IFset}, 2^{\mathsf{Fbd}})$ *is an BNF*

Preservation of BNF Properties

$\beta = (\alpha, \beta)\ \mathsf{F}$

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

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

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

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

Theorem

$(\mathsf{IF}, \mathsf{IFmap}, \mathsf{IFset}, 2^{\mathsf{Fbd}})$ *is an BNF*

Preservation of BNF Properties

$\beta = (\alpha, \beta)\ \mathsf{F}$

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

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

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

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

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

Theorem
$(\mathsf{JF}, \mathsf{JFmap}, \mathsf{JFset}, \mathsf{Fbd}^{\mathsf{Fbd}})$ *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

Technische Universität München