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

Category Theory Applied to Theorem Proving

Dmitriy Traytel Andrei Popescu Jasmin Blanchette

Technische Universität München

Outline

Outline

Isabelle/HOL

- LCF philosophy

Isabelle/HOL

- 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

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

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

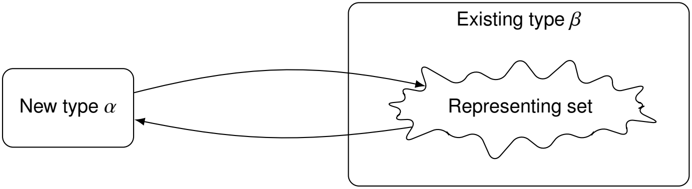
- Datatype specification

$$\begin{aligned}\texttt{datatype}\ \alpha\ \mathsf{list} &= \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list}) \\ \texttt{datatype}\ \alpha\ \mathsf{tree} &= \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree}\ \mathsf{list})\end{aligned}$$

- Datatype specification

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

- 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

$$\begin{aligned}\texttt{datatype}\ \alpha\ \mathsf{list} &= \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list}) \\ \texttt{datatype}\ \alpha\ \mathsf{tree} &= \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree}\ \mathsf{list})\end{aligned}$$

The traditional approach

Melham 1989, Gunter 1994

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

$$\begin{array}{lcl} \texttt{datatype}\ \alpha\ \mathsf{list} & = & \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list}) \\ \texttt{datatype}\ \alpha\ \mathsf{tree} & = & \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree\_list}) \\ \texttt{and}\quad \alpha\ \mathsf{tree\_list} & = & \mathsf{Nil} \mid \mathsf{Cons}\ (\alpha\ \mathsf{tree})\ (\alpha\ \mathsf{tree\_list}) \end{array}$$

The traditional approach

Melham 1989, Gunter 1994

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

$$\begin{array}{lll} \texttt{datatype}\ \alpha\ \mathsf{list} & = & \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list}) \\ \texttt{datatype}\ \alpha\ \mathsf{tree} & = & \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree\_list}) \\ \texttt{and}\quad \alpha\ \mathsf{tree\_list} & = & \mathsf{Nil} \mid \mathsf{Cons}\ (\alpha\ \mathsf{tree})\ (\alpha\ \mathsf{tree\_list}) \end{array}$$

- 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. ~~non~~compositionality ✓
2. no codatatypes
3. no non-free structures

~~Limitations~~

LICS 2012

1. ~~non~~compositionality ✔
2. ~~no~~ codatatypes ✔
3. no non-free structures

~~Limitations~~

LICS 2012

1. ~~non~~compositionality ✔
2. ~~no~~ codatatypes ✔
3. ~~no~~ non-free structures ✔

Outline

$$
\begin{aligned}
\texttt{datatype}\ \alpha\ \mathsf{list} &= \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list}) \\
\texttt{codatatype}\ \alpha\ \mathsf{tree} &= \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree}\ \mathsf{list})
\end{aligned}
$$

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

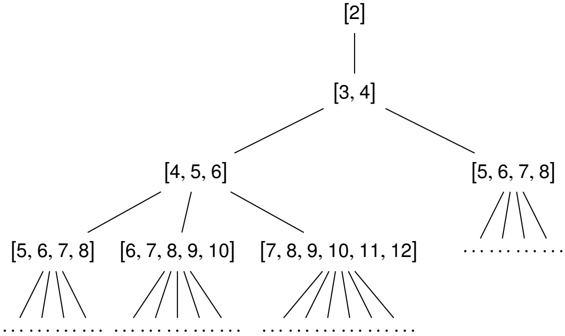
$$\texttt{codatatype}\ \alpha\ \mathsf{tree} \quad = \quad \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree}\ \mathsf{list})$$

- `P n = print n; for i = 1 to n do P (n + i);`

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

$$\texttt{codatatype}\ \alpha\ \text{tree} = \text{Node}\ \alpha\ (\alpha\ \text{tree list})$$

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



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

$$\texttt{codatatype}\ \alpha\ \mathsf{tree} \quad = \quad \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree}\ \mathsf{list})$$

- Compositionality = no unfolding

$$\texttt{datatype}\ \alpha\ \mathsf{list} \quad = \quad \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list})$$
$$\texttt{codatatype}\ \alpha\ \mathsf{tree} \quad = \quad \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree}\ \mathsf{fset})$$

- Compositionality = no unfolding
- Need abstract interface

$$\texttt{datatype}\ \alpha\ \mathsf{list} \quad = \quad \mathsf{Nil} \mid \mathsf{Cons}\ \alpha\ (\alpha\ \mathsf{list})$$
$$\texttt{codatatype}\ \alpha\ \mathsf{tree} \quad = \quad \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree}\ \mathsf{fset})$$

- 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**

type constructor F	}	functor
Fmap		

The interface: **bounded natural functor**

type constructor F	}	functor
Fmap		
Fset	}	natural transformation

The interface: **bounded natural functor**

type constructor F	}	functor
Fmap		
Fset	}	natural transformation
Fbd	}	infinite cardinal

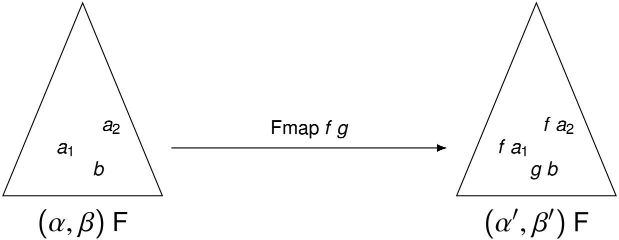
The interface: **bounded natural functor**

type constructor F Fmap	functor
Fset	natural transformation
Fbd	infinite cardinal

BNF = type constructor + polymorphic constrants + assumptions

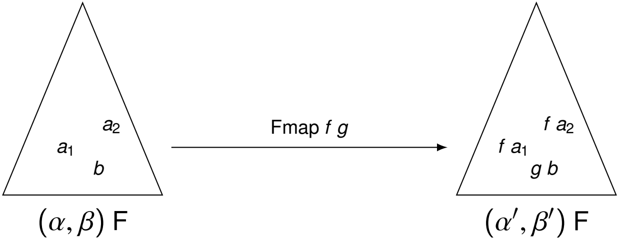
Type constructors are functors

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



Type constructors are functors

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

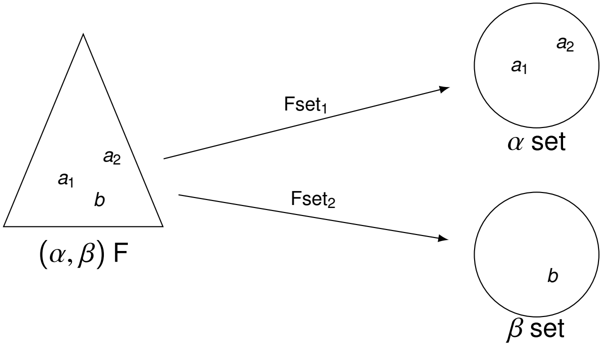


$$\begin{aligned} \text{Fmap id id} &= \text{id} \\ \text{Fmap}\ f_1\ f_2 \circ \text{Fmap}\ g_1\ g_2 &= \text{Fmap}\ (f_1 \circ g_1)\ (f_2 \circ g_2) \end{aligned}$$

Type constructors are containers

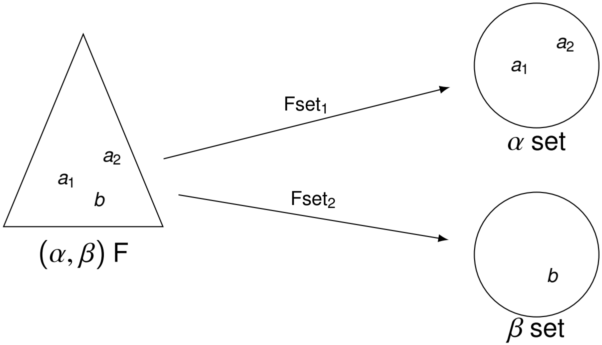
$$\mathsf{Fset}_1 : (\alpha, \beta)\ \mathsf{F} \to \alpha\ \mathsf{set}$$

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



Type constructors are containers

$$\mathsf{Fset}_1 : (\alpha, \beta)\ \mathsf{F} \to \alpha\ \mathsf{set}$$
$$\mathsf{Fset}_2 : (\alpha, \beta)\ \mathsf{F} \to \beta\ \mathsf{set}$$



$$\mathsf{Fset}_1 \circ \mathsf{Fmap}\ f_1\ f_2 = \mathsf{image}\ f_1 \circ \mathsf{Fset}_1$$
$$\mathsf{Fset}_2 \circ \mathsf{Fmap}\ f_1\ f_2 = \mathsf{image}\ f_2 \circ \mathsf{Fset}_2$$

Further BNF assumptions

$$\left.\begin{array}{l} \forall x \in \mathrm{Fset}_1\ z.\ f_1\ x = g_1\ x \\ \forall x \in \mathrm{Fset}_2\ z.\ f_2\ x = g_2\ x \end{array}\right\} \quad \Rightarrow \quad \mathrm{Fmap}\ f_1\ f_2\ z = \mathrm{Fmap}\ g_1\ g_2\ z$$

Further BNF assumptions

$$\left.\begin{array}{l}\forall x \in \mathsf{Fset}_1\ z.\ f_1\ x = g_1\ x \\ \forall x \in \mathsf{Fset}_2\ z.\ f_2\ x = g_2\ x\end{array}\right\} \quad \Rightarrow \quad \mathsf{Fmap}\ f_1\ f_2\ z = \mathsf{Fmap}\ g_1\ g_2\ z$$

$$\aleph_0 \quad \leq \quad \mathsf{Fbd}$$

Further BNF assumptions

$$\left.\begin{array}{l}\forall x \in \mathrm{Fset}_1\ z.\ f_1\ x = g_1\ x \\ \forall x \in \mathrm{Fset}_2\ z.\ f_2\ x = g_2\ x\end{array}\right\} \quad \Rightarrow \quad \mathrm{Fmap}\ f_1\ f_2\ z = \mathrm{Fmap}\ g_1\ g_2\ z$$

$$\aleph_0 \quad \leq \quad \mathrm{Fbd}$$

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

Further BNF assumptions

$$\left.\begin{array}{l}\forall x \in \mathsf{Fset}_1\ z.\ f_1\ x = g_1\ x \\ \forall x \in \mathsf{Fset}_2\ z.\ f_2\ x = g_2\ x\end{array}\right\} \Rightarrow \mathsf{Fmap}\ f_1\ f_2\ z = \mathsf{Fmap}\ g_1\ g_2\ z$$

$$\aleph_0 \leq \mathsf{Fbd}$$

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

$$|(\alpha_1, \alpha_2)\ \mathsf{F}| \leq (|\alpha_1| + |\alpha_2|)^{\mathsf{Fbd}}$$

Further BNF assumptions

$$\left.\begin{array}{l}\forall x \in \mathsf{Fset}_1\ z.\ f_1\ x = g_1\ x \\ \forall x \in \mathsf{Fset}_2\ z.\ f_2\ x = g_2\ x\end{array}\right\} \quad \Rightarrow \quad \mathsf{Fmap}\ f_1\ f_2\ z = \mathsf{Fmap}\ g_1\ g_2\ z$$

$$\aleph_0 \quad \leq \quad \mathsf{Fbd}$$

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

$$|(\alpha_1, \alpha_2)\ \mathsf{F}| \quad \leq \quad (|\alpha_1| + |\alpha_2|)^{\mathsf{Fbd}}$$

(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. $+$, $\times$, unit, and $\alpha \to \beta$ for fixed α)

What are bounded natural functors good for?

BNFs ...

- 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 bounded natural functors good for?

BNFs ...

- 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 bounded natural functors good for?

BNFs ...

- 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 (datatypes)

What are bounded natural functors good for?

BNFs ...

- 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 (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 α)
- 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 α)
- 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

From user specifications to (co)datatypes

Given

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

From user specifications to (co)datatypes

Given

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

1. Abstract to $\beta = \mathsf{unit} + \alpha \times \beta$

From user specifications to (co)datatypes

Given

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

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

From user specifications to (co)datatypes

Given

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

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

From user specifications to (co)datatypes

Given

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

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

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

From user specifications to (co)datatypes

Given

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

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

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

5. Define iterator

$$\mathsf{iter} : (\mathsf{unit} + \alpha \times \alpha\ \mathsf{list} \to \beta) \to \alpha\ \mathsf{list} \to \beta$$

From user specifications to (co)datatypes

Given

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

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

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

5. Define iterator

   $$\mathsf{iter} : (\mathsf{unit} + \alpha \times \alpha\ \mathsf{list} \to \beta) \to \alpha\ \mathsf{list} \to \beta$$

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

From user specifications to (co)datatypes

Given

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

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

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

5. Define iterator

$$\mathsf{iter} : (\mathsf{unit} + \alpha \times \alpha\ \mathsf{list} \to \beta) \to \alpha\ \mathsf{list} \to \beta$$

6. Prove characteristic theorems (e.g. induction)
7. Prove that list is a BNF

From user specifications to (co)datatypes

Given

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

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

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

5. Define iterator

$$\mathsf{iter} : (\mathsf{unit} + \alpha \times \alpha\ \mathsf{list} \to \beta) \to \alpha\ \mathsf{list} \to \beta$$

6. Prove characteristic theorems (e.g. induction)
7. Prove that list is a BNF (enables nested recursion)

From user specifications to (co)datatypes

Given

$$\texttt{codatatype}\ \alpha\ \mathsf{llist} = \mathsf{LNil} \mid \mathsf{LCons}\ \alpha\ (\alpha\ \mathsf{llist})$$

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

$$(\alpha\ \mathsf{llist}, \mathsf{unf} : \alpha\ \mathsf{llist} \to \mathsf{unit} + \alpha \times \alpha\ \mathsf{llist})$$

5. Define coiterator

$$\mathsf{coiter} : (\beta \to \mathsf{unit} + \alpha \times \alpha\ \mathsf{llist}) \to \beta \to \alpha\ \mathsf{llist}$$

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

Induction

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

- Given φ : α IF $\rightarrow$ bool

Induction

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

- 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

$\beta = \mathsf{unit} + \alpha \times \beta$

- 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 $\varphi : \alpha\ \mathsf{list} \to \mathsf{bool}$
- Case distinction on z

$$\frac{\begin{array}{r} (\forall ys \in \emptyset.\ \varphi\ ys)\ \Rightarrow\ \varphi\ (\mathsf{fld}\ (\mathsf{Inl}\ ())) \\ \forall x\ xs.\ (\forall ys \in \{xs\}.\ \varphi\ ys)\ \Rightarrow\ \varphi\ (\mathsf{fld}\ (\mathsf{Inr}\ (x, xs))) \end{array}}{\forall xs.\ \varphi\ xs}$$

Induction

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

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

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

- Given $\varphi : \alpha \text{ list} \to \text{bool}$
- Concrete induction principle

$$\frac{\begin{array}{lcl} & & \varphi\ (\text{fld}\ (\text{Inl}\ ())) \\ \forall x\ xs. & \varphi\ xs \Rightarrow & \varphi\ (\text{fld}\ (\text{Inr}\ (x, xs))) \end{array}}{\forall xs.\ \varphi\ xs}$$

Induction

$\beta = \mathsf{unit} + \alpha \times \beta$

- 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 $\varphi : \alpha\ \mathsf{list} \to \mathsf{bool}$
- In constructor notation

$$\frac{\begin{array}{lr} & \varphi\ \mathsf{Nil} \\ \forall x\ xs. \quad \varphi\ xs & \Rightarrow\ \varphi\ (\mathsf{Cons}\ x\ xs) \end{array}}{\forall xs.\ \varphi\ xs}$$

Induction & Coinduction

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

- Given $\varphi : \alpha$ IF $\rightarrow$ 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 $\psi : \alpha$ JF $\rightarrow \alpha$ JF $\rightarrow$ 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

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

Example

`codatatype` α tree $=$ Node (lab: α) (sub: α tree fset)

Example

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

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

$$\begin{aligned} \text{lab}\ (\text{tmap}\ f\ t) &= f\ (\text{lab}\ t) \\ \text{sub}\ (\text{tmap}\ f\ t) &= \text{image}\ (\text{tmap}\ f)\ (\text{sub}\ t) \end{aligned}$$

Example

`codatatype` α tree $=$ Node (lab: α) (sub: α tree fset)

`corec` tmap : $(\alpha \rightarrow \beta) \rightarrow \alpha$ tree $\rightarrow \beta$ tree `where`

$$\begin{aligned} \text{lab}\ (\text{tmap}\ f\ t) &= f\ (\text{lab}\ t) \\ \text{sub}\ (\text{tmap}\ f\ t) &= \text{image}\ (\text{tmap}\ f)\ (\text{sub}\ t) \end{aligned}$$

`lemma` tmap $(f \circ g)\ t =$ tmap f (tmap g t)

Example

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

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

$$\begin{aligned} \text{lab}\ (\text{tmap}\ f\ t) &= f\ (\text{lab}\ t) \\ \text{sub}\ (\text{tmap}\ f\ t) &= \text{image}\ (\text{tmap}\ f)\ (\text{sub}\ t) \end{aligned}$$

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

$\texttt{by}\ (\texttt{intro}\ \text{tree\_coinduct}[\texttt{where}\ \psi = \lambda t_1\ t_2.\ \exists t.\ t_1 = \text{tmap}\ (f \circ g)\ t \land t_2 = \text{tmap}\ f\ (\text{tmap}\ g\ t)])\ \texttt{force+}$

Outline

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

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

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

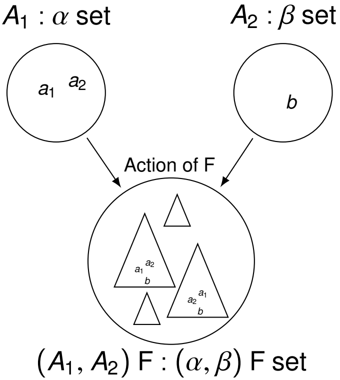
Technische Universität München

Outline

Backup slides

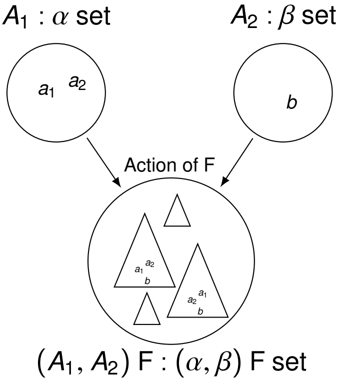
Type constructors act on sets

$$(A_1, A_2)\,\mathsf{F} = \{z \mid \mathsf{Fset}_1\ z \subseteq A_1 \wedge \mathsf{Fset}_2\ z \subseteq A_2\}$$



Type constructors act on sets

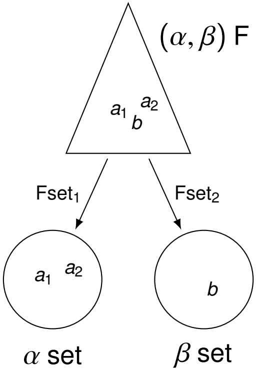
$$(A_1, A_2)\ \mathsf{F} = \{z \mid \mathsf{Fset}_1\ z \subseteq A_1 \wedge \mathsf{Fset}_2\ z \subseteq A_2\}$$



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

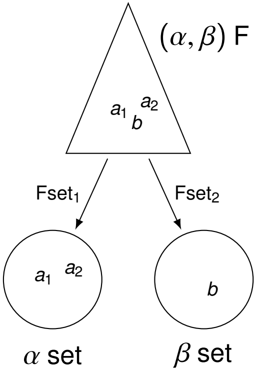
Type constructors are bounded

Fbd: infinite cardinal



Type constructors are bounded

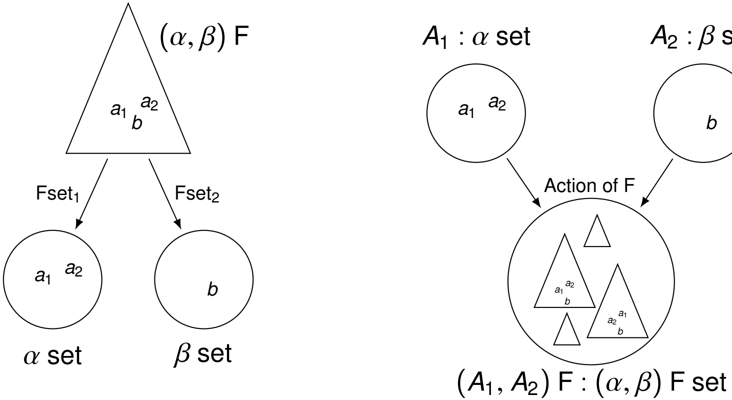
Fbd: infinite cardinal



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

Type constructors are bounded

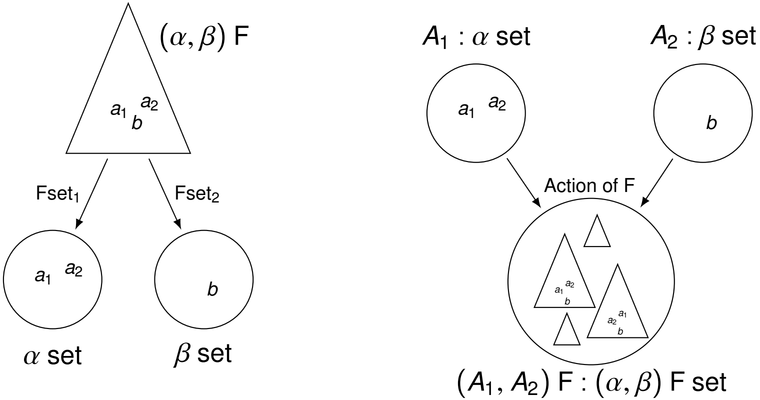
Fbd: infinite cardinal



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

Type constructors are bounded

Fbd: infinite cardinal



$$|\mathrm{Fset}_i\ z| \leq \mathrm{Fbd}$$

$$|(A_1, A_2)\ \mathrm{F}| \leq (|A_1| + |A_2| + 2)^{\mathrm{Fbd}}$$

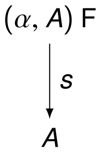
Algebras, Coalgebras & Morphisms

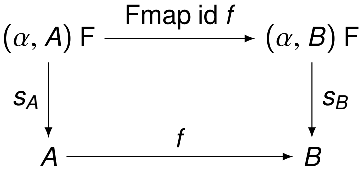
$\beta = (\alpha, \beta)\,\mathsf{F}$

$$\begin{array}{c} (\alpha, A)\,\mathsf{F} \\ \big\downarrow{\scriptstyle s} \\ A \end{array}$$

Algebras, Coalgebras & Morphisms

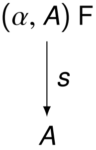
$\beta = (\alpha, \beta)$ F

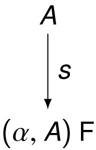


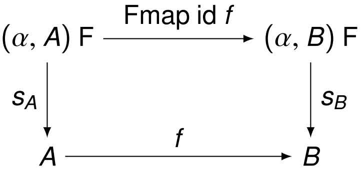


Algebras, Coalgebras & Morphisms

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

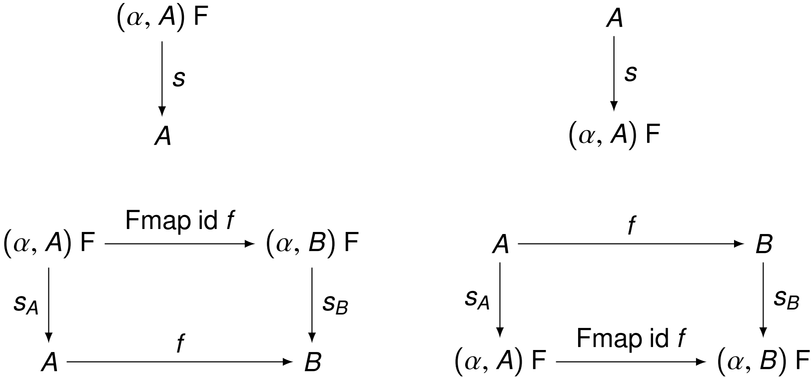






Algebras, Coalgebras & Morphisms

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

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

⇒ 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

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

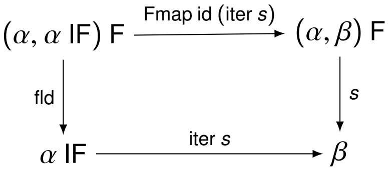
$\beta = (\alpha, \beta)\ \mathsf{F}$

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

Iteration & Coiteration

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

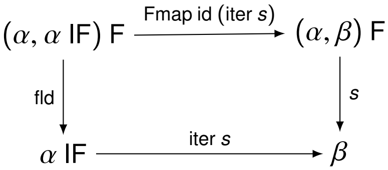
- Given $s : (\alpha, \beta)$ F $\to \beta$
- Obtain unique morphism iter s from (α IF, fld) to (U_β, s)



Iteration & Coiteration

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

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



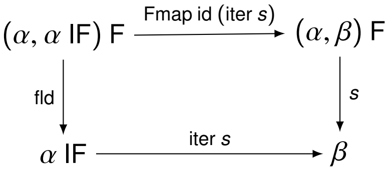
- Given $s : \beta \to (\alpha, \beta)$ F

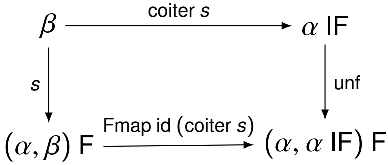
Iteration & Coiteration

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

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

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





Preservation of BNF Properties

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

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

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

Preservation of BNF Properties

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

- IFmap f = iter (fld ∘ Fmap f id)
- IFset = 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*

Preservation of BNF Properties

$\beta = (\alpha, \beta)$ 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 a 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 a BNF*

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