Elimination Principles for Initial Dialgebras

Fredrik Nordvall Forsberg

Swansea University, UK
csfnf@swansea.ac.uk

TYPES 08/09/2011, Bergen

Joint work with Thorsten Altenkirch, Peter Morris, Anton Setzer



Question

How do we reason about $\mathbb{N}$, declared by

```
data ℕ : Set where
  zero : ℕ
  suc  : ℕ → ℕ,
```

and similar data types?

Elegant answer (known since the 70's at least)

Model $\mathbb{N}$ as the initial F-algebra, where $F : \mathbf{Set} \to \mathbf{Set}$ is the functor $F(X) = \mathbf{1} + X$.

F-algebras

- Let $F : \mathbb{C} \to \mathbb{C}$ be a functor. Recall that an F-algebra is a pair (X, f) where $X \in \mathbb{C}$ and $f : F(X) \to X$.
- X carrier, $f : F(X) \to X$ constructor of the data type.
- Constructor for natural numbers: $[\texttt{zero}, \texttt{suc}] : \mathbf{1} + \mathbb{N} \to \mathbb{N}$.

F-algebras

- Let $F : \mathbb{C} \to \mathbb{C}$ be a functor. Recall that an F-algebra is a pair (X, f) where $X \in \mathbb{C}$ and $f : F(X) \to X$.
- X carrier, $f : F(X) \to X$ constructor of the data type.
- Constructor for natural numbers: `[zero, suc]` $: \mathbf{1} + \mathbb{N} \to \mathbb{N}$.
- A morphism $\alpha : (X, f) \to (Y, g)$ between F-algebras is a morphism $\alpha : X \to Y$ such that

$$\begin{array}{ccc} F(X) & \xrightarrow{f} & X \\ {\scriptstyle F(\alpha)}\downarrow & & \downarrow{\scriptstyle \alpha} \\ F(Y) & \xrightarrow[g]{} & Y \end{array}$$

Question

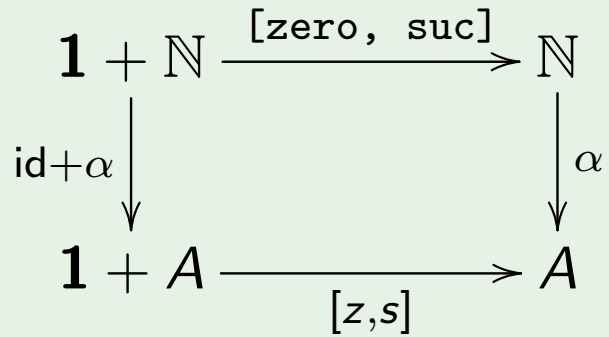
How do we write functions $\mathbb{N} \to A$ for some other type A?

Answer

Use the initiality!

- Give an element $z : A$ which `zero` should be sent to.
- Give a function $s : A \to A$, used for successors.

Combine into F-algebra $[z, s] : \mathbf{1} + A \to A$ and get function $\alpha : \mathbb{N} \to A$ such that



Question

What if I want a dependent function $(x : \mathbb{N}) \to P(x)$? I want something like an induction (or elimination) principle

$$\frac{x : \mathbb{N} \vdash P(x) : \mathsf{Set} \quad n : \mathbb{N}, \widetilde{n} : P(n) \vdash \mathsf{step}_{\mathtt{suc}}(n, \widetilde{n}) : P(\mathtt{suc}(n)) \qquad \mathsf{step}_{\mathtt{zero}} : P(\mathtt{zero})}{\mathsf{elim}(P, \mathsf{step}_{\mathtt{zero}}, \mathsf{step}_{\mathtt{suc}}) : (x : \mathbb{N}) \to P(x)}$$

Clever answer (Hermida and Jacobs, Ghani et al., . . .)

That's fine! Such a term can be defined for initial algebras. Main idea:

- Make $\Sigma n{:}\mathbb{N}.P(n)$ into an F-algebra.

Question

What if I want a dependent function $(x : \mathbb{N}) \to P(x)$? I want something like an induction (or elimination) principle

$$\frac{x : \mathbb{N} \vdash P(x) : \mathsf{Set} \quad n : \mathbb{N}, \widetilde{n} : P(n) \vdash \mathsf{step}_{\mathtt{suc}}(n, \widetilde{n}) : P(\mathtt{suc}(n)) \qquad \mathsf{step}_{\mathtt{zero}} : P(\mathtt{zero})}{\mathsf{elim}(P, \mathsf{step}_{\mathtt{zero}}, \mathsf{step}_{\mathtt{suc}}) : (x : \mathbb{N}) \to P(x)}$$

Clever answer (Hermida and Jacobs, Ghani et al., . . .)

That's fine! Such a term can be defined for initial algebras. Main idea:

- Make $\Sigma n{:}\mathbb{N}.P(n)$ into an F-algebra.
- Initiality gives function $\alpha : \mathbb{N} \to \Sigma n{:}\mathbb{N}.P(n)$.

Question

What if I want a dependent function $(x : \mathbb{N}) \to P(x)$? I want something like an induction (or elimination) principle

$$\frac{x : \mathbb{N} \vdash P(x) : \mathsf{Set} \quad \begin{array}{c} \mathsf{step}_{\mathtt{zero}} : P(\mathtt{zero}) \\ n : \mathbb{N}, \widetilde{n} : P(n) \vdash \mathsf{step}_{\mathtt{suc}}(n, \widetilde{n}) : P(\mathtt{suc}(n)) \end{array}}{\mathsf{elim}(P, \mathsf{step}_{\mathtt{zero}}, \mathsf{step}_{\mathtt{suc}}) : (x : \mathbb{N}) \to P(x)}$$

Clever answer (Hermida and Jacobs, Ghani et al., . . .)

That's fine! Such a term can be defined for initial algebras. Main idea:

- Make $\Sigma n{:}\mathbb{N}.P(n)$ into an F-algebra.
- Initiality gives function $\alpha : \mathbb{N} \to \Sigma n{:}\mathbb{N}.P(n)$.
- Prove that $\pi_0 \circ \alpha = \mathsf{id} : \mathbb{N} \to \mathbb{N}$ using uniqueness.

Question

What if I want a dependent function $(x : \mathbb{N}) \to P(x)$? I want something like an induction (or elimination) principle

$$\frac{x : \mathbb{N} \vdash P(x) : \mathsf{Set} \quad n : \mathbb{N}, \widetilde{n} : P(n) \vdash \mathsf{step}_{\mathtt{suc}}(n, \widetilde{n}) : P(\mathtt{suc}(n)) \qquad \mathsf{step}_{\mathtt{zero}} : P(\mathtt{zero})}{\mathsf{elim}(P, \mathsf{step}_{\mathtt{zero}}, \mathsf{step}_{\mathtt{suc}}) : (x : \mathbb{N}) \to P(x)}$$

Clever answer (Hermida and Jacobs, Ghani et al., . . .)

That's fine! Such a term can be defined for initial algebras. Main idea:

- Make $\Sigma n{:}\mathbb{N}.P(n)$ into an F-algebra.
- Initiality gives function $\alpha : \mathbb{N} \to \Sigma n{:}\mathbb{N}.P(n)$.
- Prove that $\pi_0 \circ \alpha = \mathsf{id} : \mathbb{N} \to \mathbb{N}$ using uniqueness.
- Hence $\pi_1 \circ \alpha : (x : \mathbb{N}) \to P(x)$ is desired eliminator.

Elimination rules for an arbitrary inductive data type

It might be worth pointing out the form of the elimination principle for an arbitrary inductive type μF with constructor $\mathsf{in} : F(\mu F) \to \mu F$:

$$\frac{x : \mu F \vdash P(x) : \mathsf{Set} \qquad x : F(\mu F), \widetilde{x} : \Box_F(P, x) \vdash \mathsf{step}(x, \widetilde{x}) : P(\mathsf{in}(x))}{\mathsf{elim}(P, \mathsf{step}) : (x : \mu F) \to P(x)}$$

- Here $\Box_F(P, x)$ is the "induction hypothesis" – the set of proofs that P holds for the "pieces" of x.
- Should satisfy a computation rule (omitted here).

Question

What if I have a more exotic data type, such as

1. an indexed inductive definition (such as lists of a certain lengths),
2. an inductive-recursive definition (such as a type-theoretic universe), or
3. an inductive-inductive definition (such as the well-formed syntax of dependent type theory)?

Answer

1 and 2 fit into the framework (see recent work by Ghani et al., and work by Dybjer and Setzer respectively) , but 3 seems not to.

We should generalise the framework so that also 3 is covered!

Side remark: inductive-inductive definitions

- An inductive-inductive definition consists of a data type $A : \mathrm{Set}$, defined mutually with an A-indexed data type $B : A \to \mathrm{Set}$.
- Both defined inductively.
- Constructors for A can refer to B and vice versa.
- In addition, constructors for B can refer to *constructors for* A.

Main example (Dybjer, Danielsson, Chapman)

Well-formed syntax of dependent type theory.

$$\begin{aligned} \mathrm{Context} &: \mathrm{Set} \\ \mathrm{Type} &: \mathrm{Context} \to \mathrm{Set} \\ &\vdots \end{aligned}$$

Side remark: inductive-inductive definitions

- An inductive-inductive definition consists of a data type $A : \mathrm{Set}$, defined mutually with an A-indexed data type $B : A \to \mathrm{Set}$.
- Both defined inductively.
- Constructors for A can refer to B and vice versa.
- In addition, constructors for B can refer to *constructors for* A.

Main example (Dybjer, Danielsson, Chapman)

Well-formed syntax of dependent type theory.

$$\begin{aligned} \mathrm{Context} &: \mathrm{Set} \\ \mathrm{Type} &: \mathrm{Context} \to \mathrm{Set} \\ &\vdots \end{aligned}$$

Problem

Can no longer describe constructors by endofunctors.

Generalisation

Describe the constructors by functors $F : \mathbb{C} \to \mathbb{D}$, not only $\mathbb{C} \to \mathbb{C}$.

This means that

$$\begin{array}{ccc} & \mathbb{D} & \mathbb{C} \\ & \Downarrow & \Downarrow \\ f : & F(X) & \longrightarrow X \end{array}$$

is no longer type correct.

Let $F, G : \mathbb{C} \to \mathbb{D}$ and consider "constructors" $f : F(X) \to G(X)$.

We will mostly be interested in the case when G is "almost" the identity, such as e.g. a forgetful functor $G : T\text{-Alg} \to \mathbb{C}$,

$$\begin{aligned} G(X, f) &= X \\ G(\alpha) &= \alpha \ . \end{aligned}$$

Dialgebras

Such structures were called dialgebras by Hagino in his thesis:

Definition

Let $F, G : \mathbb{C} \to \mathbb{D}$ be functors. An (F, G)-dialgebra (X, f) consists of $X \in \mathbb{C}$ and $f : F(X) \to G(X)$. A morphism between dialgebras (X, f) and (Y, g) is a morphism $\alpha : X \to Y$ in $\mathbb{C}$ such that

$$\begin{array}{ccc} F(X) & \xrightarrow{f} & G(X) \\ {\scriptstyle F(\alpha)}\downarrow & & \downarrow{\scriptstyle G(\alpha)} \\ F(Y) & \xrightarrow[g]{} & G(Y) \end{array}$$

Write Dialg(F, G) for the category of (F, G)-dialgebras.

Of course, $G = \mathrm{id} : \mathbb{C} \to \mathbb{C}$ gives ordinary F-algebras as a special case.

Elimination rules for dialgebras

Goal

Show that one can define an eliminator for every initial dialgebra.

First problem

What *is* an eliminator for a dialgebra?

To describe this, we require some extra structure from $\mathbb{C}$ and $\mathbb{D}$, namely that they are *categories with families*.

Categories with Families

Introduced by Dybjer as "an uncategorical categorical notion of model of type theory", with goal of being as close to the syntax as possible.

For us, an abstract notion of family of objects (property under propositions-as-types), dependent functions and dependent pairs.

Definition (Category with families)

A *category with families* is given by

- A category $\mathbb{C}$ – the category of contexts.
- A functor $F = (\mathsf{Ty}, \mathsf{Tm}) : \mathbb{C}^{\mathrm{op}} \to \mathbf{Fam}(\mathbf{Set})$ – to model types, terms (and substitutions of both).
- Extra structure to model the construction of contexts as lists of types.

Fam(Set)

Definition

The category **Fam**(**Set**) has

- **objects** pairs (I, X) where I is a set (the *index set*) and $X : I \to \mathrm{Set}$ is a function.
- **morphisms** $(I, X) \to (J, Y)$ are pairs (f, g) where $f : I \to J$ and $g : (i : I) \to X(i) \to Y(f(i))$ (i.e. a natural transformation $g : X \to Y \circ f$).

Fam(**Set**) is equivalent to $\mathrm{Set}^{\to}$.

Categories with families as models of type theory

- Think of the base category $\mathbb{C}$ as a category of contexts (and context substitutions).

Categories with families as models of type theory

- Think of the base category $\mathbb{C}$ as a category of contexts (and context substitutions).
- The object part of the functor $(\mathsf{Ty}, \mathsf{Tm}) : \mathbb{C}^{\mathrm{op}} \to \mathbf{Fam}(\mathbf{Set})$ maps a context to family of terms indexed by types in that context.
  - That is: for each $\Gamma \in \mathbb{C}$, we have a set of types $\mathsf{Ty}(\Gamma)$ in context Γ.
  - For each context $\Gamma \in \mathbb{C}$ and type $\sigma \in \mathsf{Ty}(\Gamma)$, we have a set of terms $\mathsf{Tm}(\Gamma, \sigma)$ of type σ in context Γ.

Categories with families as models of type theory

- Think of the base category $\mathbb{C}$ as a category of contexts (and context substitutions).
- The object part of the functor $(\mathsf{Ty}, \mathsf{Tm}) : \mathbb{C}^{\mathrm{op}} \to \mathbf{Fam}(\mathbf{Set})$ maps a context to family of terms indexed by types in that context.
  - That is: for each $\Gamma \in \mathbb{C}$, we have a set of types $\mathsf{Ty}(\Gamma)$ in context Γ.
  - For each context $\Gamma \in \mathbb{C}$ and type $\sigma \in \mathsf{Ty}(\Gamma)$, we have a set of terms $\mathsf{Tm}(\Gamma, \sigma)$ of type σ in context Γ.
- The morphism part of the functor performs substitution in types and terms. We write $\_\{\cdot\}$ for the action on both the index set and the indexed family, ie. if $f : \Delta \to \Gamma$, then

$$\_\{f\} : \mathsf{Ty}(\Gamma) \to \mathsf{Ty}(\Delta)$$
$$\_\{f\} : \mathsf{Tm}(\Gamma, \sigma) \to \mathsf{Tm}(\Delta, \sigma\{f\})$$

  for every $\sigma \in \mathsf{Ty}(\Gamma)$.

Categories with families as models of type theory

- Think of the base category $\mathbb{C}$ as a category of contexts (and context substitutions).
- The object part of the functor $(\mathsf{Ty}, \mathsf{Tm}) : \mathbb{C}^{\mathrm{op}} \to \mathbf{Fam}(\mathbf{Set})$ maps a context to family of terms indexed by types in that context.
  - That is: for each $\Gamma \in \mathbb{C}$, we have a set of types $\mathsf{Ty}(\Gamma)$ in context Γ.
  - For each context $\Gamma \in \mathbb{C}$ and type $\sigma \in \mathsf{Ty}(\Gamma)$, we have a set of terms $\mathsf{Tm}(\Gamma, \sigma)$ of type σ in context Γ.
- The morphism part of the functor performs substitution in types and terms. We write $\_\{\cdot\}$ for the action on both the index set and the indexed family, ie. if $f : \Delta \to \Gamma$, then

$$\_\{f\} : \mathsf{Ty}(\Gamma) \to \mathsf{Ty}(\Delta)$$
$$\_\{f\} : \mathsf{Tm}(\Gamma, \sigma) \to \mathsf{Tm}(\Delta, \sigma\{f\})$$

  for every $\sigma \in \mathsf{Ty}(\Gamma)$.
- Contravariant, because that is how substitution works.

Extra structure

- $\mathbb{C}$ should have a terminal object (the *empty context*).

Extra structure

- $\mathbb{C}$ should have a terminal object (the *empty context*).
- For each $\Gamma \in \mathbb{C}$ and $\sigma \in \mathsf{Ty}(\Gamma)$, there is
  - an object $\Gamma \cdot \sigma$ (the *context comprehension*),
- Corresponds to extending a context Γ with a fresh variable x of type σ to form the context $\Gamma, x : \sigma$.

Extra structure

- $\mathbb{C}$ should have a terminal object (the *empty context*).
- For each $\Gamma \in \mathbb{C}$ and $\sigma \in \mathsf{Ty}(\Gamma)$, there is
  - an object $\Gamma \cdot \sigma$ (the *context comprehension*),
  - a morphism $\mathbf{p}_\sigma : \Gamma \cdot \sigma \to \Gamma$ (the *first projection*),
- Corresponds to extending a context Γ with a fresh variable x of type σ to form the context $\Gamma, x : \sigma$.

Extra structure

- $\mathbb{C}$ should have a terminal object (the *empty context*).
- For each $\Gamma \in \mathbb{C}$ and $\sigma \in \mathsf{Ty}(\Gamma)$, there is
  - an object $\Gamma \cdot \sigma$ (the *context comprehension*),
  - a morphism $\mathbf{p}_\sigma : \Gamma \cdot \sigma \to \Gamma$ (the *first projection*),
  - a term $\mathbf{v}_\sigma \in \mathsf{Tm}(\Gamma \cdot \sigma, \sigma\{\mathbf{p}\})$ (the *second projection*),
- Corresponds to extending a context Γ with a fresh variable x of type σ to form the context $\Gamma, x : \sigma$.

Extra structure

- $\mathbb{C}$ should have a terminal object (the *empty context*).
- For each $\Gamma \in \mathbb{C}$ and $\sigma \in \mathsf{Ty}(\Gamma)$, there is
  - an object $\Gamma \cdot \sigma$ (the *context comprehension*),
  - a morphism $\mathbf{p}_\sigma : \Gamma \cdot \sigma \to \Gamma$ (the *first projection*),
  - a term $\mathbf{v}_\sigma \in \mathsf{Tm}(\Gamma \cdot \sigma, \sigma\{\mathbf{p}\})$ (the *second projection*),

  satisfying a universal property (omitted here).
- Corresponds to extending a context Γ with a fresh variable x of type σ to form the context $\Gamma, x : \sigma$.

Extra structure

- $\mathbb{C}$ should have a terminal object (the *empty context*).
- For each $\Gamma \in \mathbb{C}$ and $\sigma \in \mathsf{Ty}(\Gamma)$, there is
  - an object $\Gamma \cdot \sigma$ (the *context comprehension*),
  - a morphism $\mathbf{p}_\sigma : \Gamma \cdot \sigma \to \Gamma$ (the *first projection*),
  - a term $\mathbf{v}_\sigma \in \mathsf{Tm}(\Gamma \cdot \sigma, \sigma\{\mathbf{p}\})$ (the *second projection*),

  satisfying a universal property (omitted here).
- Corresponds to extending a context Γ with a fresh variable x of type σ to form the context $\Gamma, x : \sigma$.

Fact

There is a sound interpretation of (the structural rules of) type theory in any Category with Families.

Set as a category with families

$$
\begin{array}{lcll}
\mathsf{Ty}(\Gamma) & = & \{A \mid A : \Gamma \to \mathsf{Set} \text{ is a } \Gamma\text{-indexed family of (small) sets}\} & \\
A\{f\} & = & A \circ f \in \mathsf{Ty}(\Delta) & (f : \Delta \to \Gamma) \\
\mathsf{Tm}(\Gamma, A) & = & (x : \Gamma) \to A(x) & \\
a\{f\} & = & a \circ f \in \mathsf{Tm}(\Delta, A\{f\}) & (f : \Delta \to \Gamma) \\
\Gamma \cdot A & = & (\Sigma x : \Gamma) A(x) & \\
\mathbf{p}_A(x, y) & = & x & \\
\mathbf{v}_A(x, y) & = & y &
\end{array}
$$

Set as a category with families

$$
\begin{array}{lcll}
\mathsf{Ty}(\Gamma) & = & \{A \mid A : \Gamma \to \mathsf{Set} \text{ is a } \Gamma\text{-indexed family of (small) sets}\} & \\
A\{f\} & = & A \circ f \in \mathsf{Ty}(\Delta) & (f : \Delta \to \Gamma) \\
\mathsf{Tm}(\Gamma, A) & = & (x : \Gamma) \to A(x) & \\
a\{f\} & = & a \circ f \in \mathsf{Tm}(\Delta, A\{f\}) & (f : \Delta \to \Gamma) \\
\Gamma \cdot A & = & (\Sigma x : \Gamma) A(x) & \\
\mathbf{p}_A(x, y) & = & x & \\
\mathbf{v}_A(x, y) & = & y &
\end{array}
$$

Categories with families as abstract dependencies

Think of

- $\mathsf{Ty}(\Gamma)$ as properties of Γ,
- $\mathsf{Tm}(\Gamma, \sigma)$ as a dependent function space,
- $\Gamma \cdot \sigma$ as a dependent pair (with projections **p** and **v**).

Elimination rules for F-algebras in arbitrary CwFs

With this in mind, let us look at the ordinary elimination rule again:

$$\frac{P : \mu F \to \mathsf{Set} \qquad \mathsf{step} : (x : F(\mu F))(\widetilde{x} : \square_F(P, x)) \to P(\mathsf{in}(x))}{\mathsf{elim}(P, \mathsf{step}) : (x : \mu F) \to P(x)}$$

Generalising to an arbitrary category with families, we get

$$\frac{P \in \mathsf{Ty}(\mu F) \qquad \mathsf{step} \in \mathsf{Tm}(F(\mu F) \cdot \square_F(P), P\{\mathsf{in} \circ \mathbf{p}\})}{\mathsf{elim}(P, \mathsf{step}) \in \mathsf{Tm}(\mu F, P)}$$

This is still only for an F-algebra, not for dialgebras!

Problem

What is $\square_F$?

What is $\Box_F$?

$\Box_F$ should lift properties on X to properties on $F(X)$: If $P : X \to \mathrm{Set}$, then $\Box_F(P) : F(X) \to \mathrm{Set}$.

For ordinary F-algebras, there is an isomorphism

$$\varphi_F : F((\Sigma x : X)P(x)) \to (\Sigma x : F(X))\Box_F(P, x)$$

with $\pi_0 \circ \varphi_F = F(\pi_0)$.

$$\begin{array}{ccc} F((\Sigma x : X)P(x)) & \xrightarrow{\varphi_F} & (\Sigma x : F(X))\Box_F(P, x) \\ {\scriptstyle F(\pi_0)}\downarrow & \swarrow{\scriptstyle \pi_0} & \\ F(X) & & \end{array}$$

In fact, this determines $\Box_F$ up to isomorphism.

What $\Box_F$ is for $F : \mathbb{C} \to \mathbb{D}$

$\Box_F$ should lift properties on X to properties on $F(X)$: If $P : X \to \mathrm{Set}$, then $\Box_F(P) : F(X) \to \mathrm{Set}$.

For ordinary F-algebras, there is an isomorphism

$$\varphi_F : F((\Sigma x : X)P(x)) \to (\Sigma x : F(X))\Box_F(P, x)$$

with $\pi_0 \circ \varphi_F = F(\pi_0)$.

$$\begin{array}{ccc} F((\Sigma x : X)P(x)) & \xrightarrow{\varphi_F} & (\Sigma x : F(X))\Box_F(P, x) \\ {\scriptstyle F(\pi_0)}\downarrow & \swarrow{\scriptstyle \pi_0} & \\ F(X) & & \end{array}$$

In fact, this determines $\Box_F$ up to isomorphism.

What $\Box_F$ is for $F : \mathbb{C} \to \mathbb{D}$

$\Box_F$ should lift properties on X to properties on $F(X)$: If $P \in \mathrm{Ty}_{\mathbb{C}}(X)$, then $\Box_F(P) : F(X) \to \mathrm{Set}$.

For ordinary F-algebras, there is an isomorphism

$$\varphi_F : F((\Sigma x : X)P(x)) \to (\Sigma x : F(X))\Box_F(P, x)$$

with $\pi_0 \circ \varphi_F = F(\pi_0)$.

$$\begin{array}{ccc} F((\Sigma x : X)P(x)) & \xrightarrow{\varphi_F} & (\Sigma x : F(X))\Box_F(P, x) \\ {\scriptstyle F(\pi_0)}\downarrow & \swarrow {\scriptstyle \pi_0} & \\ F(X) & & \end{array}$$

In fact, this determines $\Box_F$ up to isomorphism.

What $\Box_F$ is for $F : \mathbb{C} \to \mathbb{D}$

$\Box_F$ should lift properties on X to properties on $F(X)$: If $P \in \mathsf{Ty}_{\mathbb{C}}(X)$, then $\Box_F(P) \in \mathsf{Ty}_{\mathbb{D}}(F(X))$.

For ordinary F-algebras, there is an isomorphism

$$\varphi_F : F((\Sigma x : X)P(x)) \to (\Sigma x : F(X))\Box_F(P, x)$$

with $\pi_0 \circ \varphi_F = F(\pi_0)$.

$$\begin{array}{ccc} F((\Sigma x : X)P(x)) & \xrightarrow{\varphi_F} & (\Sigma x : F(X))\Box_F(P, x) \\ {\scriptstyle F(\pi_0)}\downarrow & \swarrow {\scriptstyle \pi_0} & \\ F(X) & & \end{array}$$

In fact, this determines $\Box_F$ up to isomorphism.

What $\Box_F$ is for $F : \mathbb{C} \to \mathbb{D}$

$\Box_F$ should lift properties on X to properties on $F(X)$: If $P \in \mathrm{Ty}_{\mathbb{C}}(X)$, then $\Box_F(P) \in \mathrm{Ty}_{\mathbb{D}}(F(X))$.

There should be an isomorphism

$$\varphi_F : F(X \cdot P) \to (\Sigma x : F(X))\Box_F(P, x)$$

with $\pi_0 \circ \varphi_F = F(\pi_0)$.

$$\begin{array}{ccc} F((\Sigma x : X)P(x)) & \xrightarrow{\varphi_F} & (\Sigma x : F(X))\Box_F(P, x) \\ \Big\downarrow{\scriptstyle F(\pi_0)} & \swarrow{\scriptstyle \pi_0} & \\ F(X) & & \end{array}$$

In fact, this determines $\Box_F$ up to isomorphism.

What $\Box_F$ is for $F : \mathbb{C} \to \mathbb{D}$

$\Box_F$ should lift properties on X to properties on $F(X)$: If $P \in \mathrm{Ty}_{\mathbb{C}}(X)$, then $\Box_F(P) \in \mathrm{Ty}_{\mathbb{D}}(F(X))$.

There should be an isomorphism

$$\varphi_F : F(X \cdot P) \to F(X) \cdot \Box_F(P)$$

with $\pi_0 \circ \varphi_F = F(\pi_0)$.

$$\begin{array}{ccc} F((\Sigma x : X) P(x)) & \xrightarrow{\varphi_F} & (\Sigma x : F(X)) \Box_F(P, x) \\ \Big\downarrow{\scriptstyle F(\pi_0)} & \swarrow{\scriptstyle \pi_0} & \\ F(X) & & \end{array}$$

In fact, this determines $\Box_F$ up to isomorphism.

What $\Box_F$ is for $F : \mathbb{C} \to \mathbb{D}$

$\Box_F$ should lift properties on X to properties on $F(X)$: If $P \in \mathrm{Ty}_{\mathbb{C}}(X)$, then $\Box_F(P) \in \mathrm{Ty}_{\mathbb{D}}(F(X))$.

There should be an isomorphism

$$\varphi_F : F(X \cdot P) \to F(X) \cdot \Box_F(P)$$

with $\mathbf{p}_{\Box_F(P)} \circ \varphi_F = F(\mathbf{p}_P)$.

$$\begin{array}{ccc} F((\Sigma x : X) P(x)) & \xrightarrow{\varphi_F} & (\Sigma x : F(X)) \Box_F(P, x) \\ {\scriptstyle F(\pi_0)}\downarrow & \swarrow {\scriptstyle \pi_0} & \\ F(X) & & \end{array}$$

In fact, this determines $\Box_F$ up to isomorphism.

What $\Box_F$ is for $F : \mathbb{C} \to \mathbb{D}$

$\Box_F$ should lift properties on X to properties on $F(X)$: If $P \in \mathrm{Ty}_{\mathbb{C}}(X)$, then $\Box_F(P) \in \mathrm{Ty}_{\mathbb{D}}(F(X))$.

There should be an isomorphism

$$\varphi_F : F(X \cdot P) \to F(X) \cdot \Box_F(P)$$

with $\mathbf{p}_{\Box_F(P)} \circ \varphi_F = F(\mathbf{p}_P)$.

$$\begin{array}{ccc} F(X \cdot P) & \xrightarrow{\varphi_F} & F(X) \cdot \Box_F(P) \\ {\scriptstyle F(\mathbf{p}_P)}\downarrow & \swarrow_{\mathbf{p}_{\Box_F(P)}} & \\ F(X) & & \end{array}$$

In fact, this determines $\Box_F$ up to isomorphism.

What $\Box_F$ is for $F : \mathbb{C} \to \mathbb{D}$

$\Box_F$ should lift properties on X to properties on $F(X)$: If $P \in \mathrm{Ty}_{\mathbb{C}}(X)$, then $\Box_F(P) \in \mathrm{Ty}_{\mathbb{D}}(F(X))$.

There should be an isomorphism

$$\varphi_F : F(X \cdot P) \to F(X) \cdot \Box_F(P)$$

with $\mathbf{p}_{\Box_F(P)} \circ \varphi_F = F(\mathbf{p}_P)$.

$$\begin{array}{ccc} F(X \cdot P) & \xrightarrow{\varphi_F} & F(X) \cdot \Box_F(P) \\ {\scriptstyle F(\mathbf{p}_P)}\downarrow & \swarrow_{\mathbf{p}_{\Box_F(P)}} & \\ F(X) & & \end{array}$$

This determines $\Box_F$ up to isomorphism.

$\Box_F$ can also be explicitly constructed if the CwF $\mathbb{D}$ supports sigma types, extensional identity types and "constant families".

Elimination rules for (F, G)-dialgebras

The elimination rule for an (F, G)-dialgebra is

$$\frac{P \in \mathsf{Ty}(X) \qquad \mathsf{step} \in \mathsf{Tm}(F(X) \cdot \Box_F(P), \Box_G(P)\{\mathsf{in} \circ \mathbf{p}\})}{\mathsf{elim}(P, \mathsf{step}) \in \mathsf{Tm}(G(X), \Box_G(P))}$$

If G is "almost" the identity, then so is $\Box_G$.

(This justifies the occurrence of $\Box_G(P)$ where perhaps P was expected.)

Must be this way to make computation rules of the form

$$\mathsf{elim}(P, \mathsf{step})\{\mathsf{in}\} = \mathsf{step}\{\ldots\}$$

typecheck.

Initial dialgebras have eliminators

The proof that an initial dialgebra (X, in) have an eliminator can now be lifted from the proof for algebras in **Set**:

- Make $\Sigma n{:}X.P(n)$ into an F-algebra.
- Initiality gives function $\alpha : X \to \Sigma n{:}X.P(n)$.
- Prove that $\pi_0 \circ \alpha = \mathsf{id} : X \to X$ using uniqueness.
- Hence $\pi_1 \circ \alpha \ : (x : X) \to P(x)$ is desired eliminator.
- Verify computation rules.

Instantiate the proof in **Set** with $G = \mathsf{id}$, and the original proof falls out!

Initial dialgebras have eliminators

The proof that an initial dialgebra (X, in) have an eliminator can now be lifted from the proof for algebras in **Set**:

- Make $X \cdot P$ into an (F, G)-dialgebra.
- Initiality gives function $\alpha : X \to \Sigma n{:}X.P(n)$.
- Prove that $\pi_0 \circ \alpha = \mathsf{id} : X \to X$ using uniqueness.
- Hence $\pi_1 \circ \alpha \ : (x : X) \to P(x)$ is desired eliminator.
- Verify computation rules.

Instantiate the proof in **Set** with $G = \mathsf{id}$, and the original proof falls out!

Initial dialgebras have eliminators

The proof that an initial dialgebra (X, in) have an eliminator can now be lifted from the proof for algebras in **Set**:

- Make $X \cdot P$ into an (F, G)-dialgebra.
- Initiality gives function $\alpha : X \to X \cdot P$.
- Prove that $\pi_0 \circ \alpha = \mathsf{id} : X \to X$ using uniqueness.
- Hence $\pi_1 \circ \alpha \ : (x : X) \to P(x)$ is desired eliminator.
- Verify computation rules.

Instantiate the proof in **Set** with $G = \mathsf{id}$, and the original proof falls out!

Initial dialgebras have eliminators

The proof that an initial dialgebra (X, in) have an eliminator can now be lifted from the proof for algebras in **Set**:

- Make $X \cdot P$ into an (F, G)-dialgebra.
- Initiality gives function $\alpha : X \to X \cdot P$.
- Prove that $\mathbf{p} \circ \alpha = \mathsf{id} : X \to X$ using uniqueness.
- Hence $\pi_1 \circ \alpha \ : (x : X) \to P(x)$ is desired eliminator.
- Verify computation rules.

Instantiate the proof in **Set** with $G = \mathsf{id}$, and the original proof falls out!

Initial dialgebras have eliminators

The proof that an initial dialgebra (X, in) have an eliminator can now be lifted from the proof for algebras in **Set**:

- Make $X \cdot P$ into an (F, G)-dialgebra.
- Initiality gives function $\alpha : X \to X \cdot P$.
- Prove that $\mathbf{p} \circ \alpha = \mathsf{id} : X \to X$ using uniqueness.
- Hence $\mathbf{v}\{\varphi_G \circ G(\alpha)\} : (x : X) \to P(x)$ is desired eliminator.
- Verify computation rules.

Instantiate the proof in **Set** with $G = \mathsf{id}$, and the original proof falls out!

Initial dialgebras have eliminators

The proof that an initial dialgebra (X, in) have an eliminator can now be lifted from the proof for algebras in **Set**:

- Make $X \cdot P$ into an (F, G)-dialgebra.
- Initiality gives function $\alpha : X \to X \cdot P$.
- Prove that $\mathbf{p} \circ \alpha = \mathsf{id} : X \to X$ using uniqueness.
- Hence $\mathbf{v}\{\varphi_G \circ G(\alpha)\} \in \mathsf{Tm}(G(X), \Box_G(P))$ is desired eliminator.
- Verify computation rules.

Instantiate the proof in **Set** with $G = \mathsf{id}$, and the original proof falls out!

Some examples

Induction principle	$\mathbb{C}$	$\mathbb{D}$	G
Ordinary induction	**Set**	–	id
Indexed induction	$\mathbf{Set}^I$	–	id
Induction-recursion	$\mathbf{Type}/D$	–	id
induction-induction*	$\mathrm{Dialg}(F_0, U)$	**Fam**(**Set**)	V

* $U : \mathbf{Fam}(\mathbf{Set}) \to \mathrm{Set}$ and $V : \mathrm{Dialg}(F_0, U) \to \mathbf{Fam}(\mathbf{Set})$ are forgetful functors.
[$U(I, X) = I \quad V(X, f) = X$]

Summary

- **Dialgebras** instead of algebras (to include **induction-induction**).
- **Generic elimination principle** for any functors $F, G : \mathbb{C} \to \mathbb{D}$.
- Based on $\mathbb{C}$ and $\mathbb{D}$ being **categories with families**.
- Initial dialgebras have eliminators.

Summary

- **Dialgebr** **on**).
- **Generic**).
- Based on
- Initial dia

Thanks!