Containers

Constructing Strictly Positive Types

Felix Rech
Advisor: Steven Schäfer

December 9, 2016

(Co-)Inductive Types in Coq

- Coq doesn't always generate a useful induction principle:

```
Inductive Tree := node : List Tree -> Tree
```

- Equality on co-inductive types is to weak:

```
CoFixpoint ones : stream nat := Cons 1 ones.

CoFixpoint zeroes : stream nat := Cons 0 zeroes.
Definition ones' := map S zeroes.
```

- Syntactic conditions for (co-)inductive and (co-)recursive definitions are hard to justify.
- Functions like $size$ and map have to be rewritten for every (co-)inductive definition.

We overcome those problems by a construction of types and type constructors inside our type theory.

Type Equivalence

Definition (Equivalence)

Two types A and B are equivalent ($A \simeq B$) iff there is an isomorphism from A to B.

Examples

- $\text{Unit} + \text{Unit} \simeq \text{Bool}$
- $A \times B \to C \simeq A \to B \to C$

Axiom (Univalence)

Equivalence is equivalent to equality between two types

Proposition

$$Univalence \to Funext$$

Functor

A container type

Examples

- List
- Option
- Tree

Definition

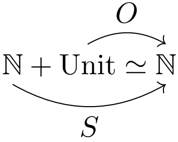
A Functor consists of functions $F : \mathrm{Type} \to \mathrm{Type}$ and $\mathrm{map} : (A \to B) \to F\ A \to F\ B$, that obey two rules:

1. $\mathrm{map}\ \ \mathrm{id} = \mathrm{id}$
2. $\mathrm{map}\ (f \circ g) = \mathrm{map}\ \ f \circ \mathrm{map}\ \ g$

Inductive Types are Fixed Points

Every inductive type is fixed point of some non-trivial functor.

Example (Natural Numbers)



$$F_{\mathbb{N}}\ X :\equiv X + \mathrm{Unit}$$

Example (Binary Trees)

$$A + (A \times T \times T) \simeq T$$

Algebra

A type with a constructor

Definition (Algebra)

An algebra over a functor F consists of

- A type A (the carrier)
- A function $\alpha : F\ A \to A$

Example (Natural Numbers)

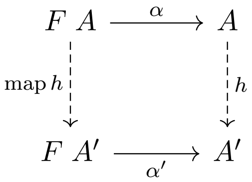
$$\alpha_{\mathbb{N}} : \mathbb{N} + \mathrm{Unit} \to \mathbb{N}$$
$$\alpha_{\mathbb{N}}\ (\mathrm{inl}\ \ n) :\equiv n + 1$$
$$\alpha_{\mathbb{N}}\ (\mathrm{inr}\ \ tt) :\equiv 0$$

Initial Algebra

A type with a constructor an a unique recursion function

Definition (Initial Algebra)

An F-algebra (A, α) is initial iff for every F-algebra (A', α') there is exactly one function $h : A \to A'$ with



Example (Natural Numbers)

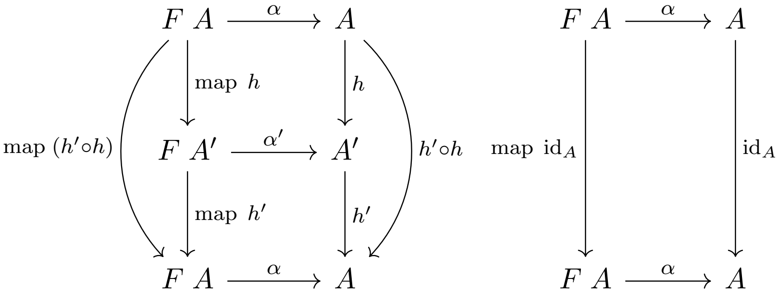
For $A' : \mathrm{Type}$ and $\alpha' : A' + \mathrm{Unit} \to A'$ we define:

$$h : \mathbb{N} \to A'$$
$$h\ 0 :\equiv \alpha'\ (\mathrm{inr}\ \ tt)$$
$$h\ (n+1) :\equiv \alpha'\ (\mathrm{inl}\ (h\ n))$$

Initial Algebras are Unique

Proof Sketch

Fix two initial F-algebras A and A'.



$\Rightarrow h' \circ h = \mathrm{id}_A$

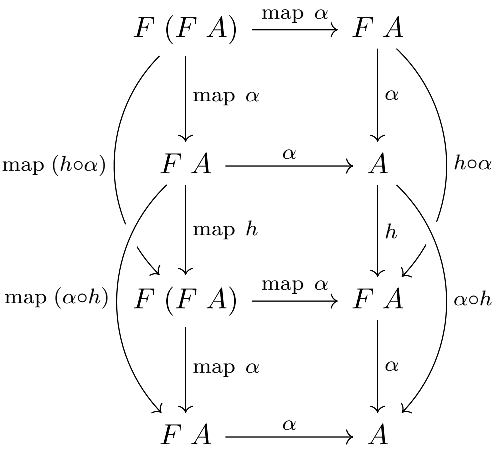
$h \circ h' = \mathrm{id}_{A'}$ follows in the same way.

$\Rightarrow$ h is an equivalence.

Initial Algebras are Fixed Points (Lambek's theorem)

Proof Sketch

Fix an initial F-algebra A.



$\Rightarrow h \circ \alpha = \mathrm{id}_{F\ A}$

$\Rightarrow \alpha \circ h = \mathrm{id}_A$

$\Rightarrow \alpha$ is an equivalence.

Initial Algebra – Induction (On Natural Numbers)

Proof Sketch

We have $P : \mathbb{N} \to Type$, $s : P\ 0$ and $f : \prod_n P\ n \to P\ (n+1)$.
We want to obtain a function $ind : \prod_n P\ n$ just from initiality of $\mathbb{N}$.

Outline

1. Construct a recursive function $h : \mathbb{N} \to \sum_n P\ n$
2. Show $\pi_1 \circ h = \mathrm{id}_{\mathbb{N}}$ to obtain a function $ind : \prod_n P\ n$
3. Prove β-law for ind

$$\alpha'\ (\mathrm{inr}\ \ tt) \equiv (0, s)$$
$$\alpha'\ (\mathrm{inl}\ (n, x)) \equiv (n+1, f_n\ x)$$

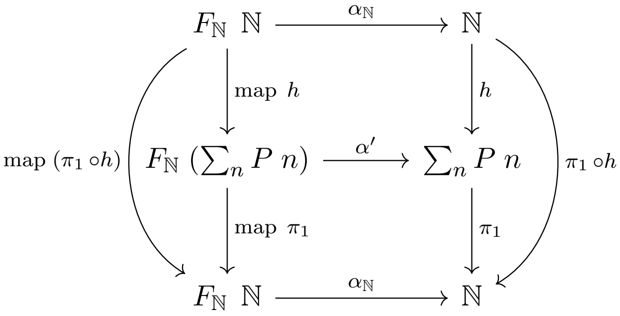
Initial Algebra – Induction (On Natural Numbers)

Proof Sketch

We have $P : \mathbb{N} \to Type$, $s : P\ 0$ and $f : \prod_n P\ n \to P\ (n+1)$.
We want to obtain a function $ind : \prod_n P\ n$ just from initiality of $\mathbb{N}$.

Outline

1. Construct a recursive function $h : \mathbb{N} \to \sum_n P\ n$
2. Show $\pi_1 \circ h = \mathrm{id}_{\mathbb{N}}$ to obtain a function $ind : \prod_n P\ n$
3. Prove β-law for ind



Unary Container

A polynomial-like normal form for strictly positive functors

Example (List)

$$\text{List}\ A \simeq \sum_{n:\mathbb{N}} \text{Fin}\ n \to A \equiv (\!|\ \mathbb{N} \blacktriangleright \text{Fin}\ |\!)\ A$$

In general

A unary container consists of:

- A type of shapes S
- A function $P : S \to \text{Type}$

Semantics:

$$(\!|\ S \blacktriangleright P\ |\!)\ A :\equiv \sum_{s:S} P\ s \to A$$

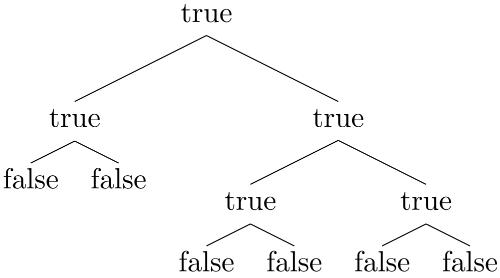
W-Types

Type of well-founded trees

```
Inductive W A (B : A -> Type) :=
sup (label : A) (subtrees : B label -> W A B) : W A B.
```

Example

$$BTree \simeq W\ \ \mathrm{Bool}\ (\lambda b \Rightarrow \mathit{if}\ \ b\ \mathit{then}\ \ \mathrm{Bool}\ \ \mathit{else}\ \ \mathrm{Empty})$$



Lemma

$W\ \ A\ B$ is the initial algebra for $(\!|\ A \blacktriangleright B\ |\!)$.

Parameterized Initial Algebra

A Functor that produces initial algebras

Example (List)

For all A, $\mathrm{List}\ A$ is initial algebra of $\lambda X.\,(A \times X) + \mathrm{Unit}$.

In general

Fix a multi-functor $F : (\mathrm{Option}\, I \to \mathrm{Type}) \to \mathrm{Type}$.
Define $F_\Gamma :\equiv \lambda A.\, F\ (\Gamma; ; A)$ as the partial application of F to $\Gamma : I \to \mathrm{Type}$.
A parameterized initial algebra of F is a multi-functor $G : (I \to \mathrm{Type}) \to \mathrm{Type}$ such that $G\ \Gamma$ is initial algebra of F_Γ for all Γ.

Indexed Containers

Polynomial functors with multiple arguments

Example (Sum)

$$A + B \simeq \sum_{b:\mathrm{Bool}} (b = \mathit{true} \to A) * (b = \mathit{false} \to B)$$

In general

An I-indexed container for $I : \mathrm{Type}$ consists of:

- A type of shapes S
- A function $P : I \to S \to \mathrm{Type}$

Semantics:

$$[\![S \rhd P]\!]\ \Gamma :\equiv \sum_{s:S} \prod_{i:I} P\ i\ s \to \Gamma\ i$$

μ-Containers

Containers that produce initial algebras

We have an $\mathrm{Option}\, I$-indexed container $S \rhd P$.
We want an I-indexed container c_μ such that $[\![c_\mu]\!]\ \Gamma$ is the initial algebra of $[\![S \rhd P]\!]_\Gamma$ for all environments Γ.

Outline

1. Fix Γ and transform $[\![S \rhd P]\!]_\Gamma$ into polynomial form
2. Obtain the initial algebra as W-type
3. Transform the W-type into a polynomial in Γ

μ-Containers

Outline

1. Fix Γ and transform $[\![S \triangleright P]\!]_\Gamma$ into polynomial form
2. Obtain the initial algebra as W-type
3. Transform the W-type into a polynomial in Γ

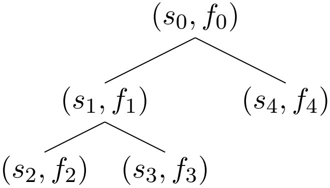
$$
\begin{aligned}
[\![S \triangleright P]\!]_\Gamma\ X &\equiv \sum_{s:S} \prod_{i:\text{Option}\ I} P\ i\ s \to (\Gamma;;X)\ i \\
&\simeq \sum_{s:S} (\prod_{i:I} P\ (\text{some}\ \ i)\ s \to \Gamma\ i) \times P\ \ \text{none}\ \ s \to X \\
&\simeq \sum_{s':\sum_s \prod_i P\ (\text{some}\ \ i)\ s \to \Gamma\ i} P\ \ \text{none}\ (\pi_1\ \ s') \to X
\end{aligned}
$$

μ-Containers

Outline

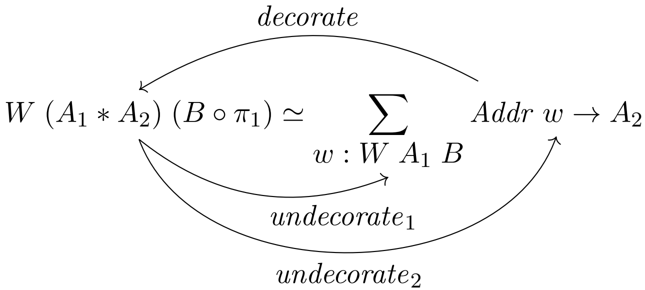
1. Fix Γ and transform $[\![S \rhd P]\!]_\Gamma$ into polynomial form
2. Obtain the initial algebra as W-type
3. Transform the W-type into a polynomial in Γ

$$W \left(\sum_{s:S} \prod_{i:I} P \ (\text{some} \ \ i) \ s \to \Gamma \ i \right) \ (P \ \ \text{none} \circ \pi_1)$$



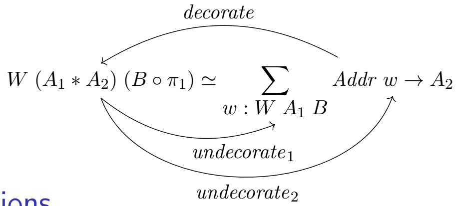
Tree Splitting

Given types A_1 and A_2 and a function $B : A_1 \to \mathrm{Type}$ we want to show:



Here $Addr\ w$ is the inductively defined type of addresses in the tree w.

Tree Splitting



Proof Obligations

- $\prod_w \mathit{decorate}\ (\mathit{undecorate}\ w) = w$ (by induction)
- $p : \prod_{w,f} \mathit{undecorate}_1\ (\mathit{decorate}\ (w, f)) = w$ (by induction)
- $$\prod_{w,f} p_{w,f} \# (\mathit{undecorate}_2\ (\mathit{decorate}\ (w, f))) = f$$
  $$\leftrightarrow \prod_{w,f} (\mathit{undecorate}_2\ (\mathit{decorate}\ (w, f))) = p_{w,f}^{-1} \# f$$
  $$\leftrightarrow \prod_{w,f,\mathit{addr}} (\mathit{undecorate}_2\ (\mathit{decorate}\ (w, f)))\ \mathit{addr} = f\ (p_{w,f} \# \mathit{addr})$$
  (by induction with a recursive description of $p_{w,f} \# \mathit{addr}$)

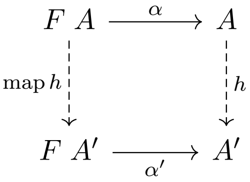
Co-Inductive Types

The description of co-inductive types is dual to initial algebras.

Inductive

Algebras Type with constructor

Initial Algebra

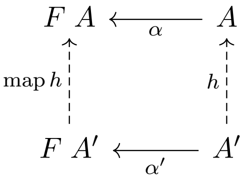


W-Types Well-founded trees

Coinductive

Coalgebra Type with destructor

Final Coalgebra



M-Types Potentially infinite trees

The proofs for uniqueness and the fixed point property are dual.
Instead of induction we have co-induction.

Conclusion

What you saw

- A general description of (co-)inductive types
- Construction of (co-)inductive types with containers

Next steps

- Indexed containers
- Construction of W-types from $\mathbb{N}$
- Rational fixed points

Conclusion

What you saw

- A general description of (co-)inductive types
- Construction of (co-)inductive types with containers

Next steps

- Indexed containers
- Construction of W-types from $\mathbb{N}$
- Rational fixed points

Thank you!

References

Michael Abbott, Thorsten Altenkirch, and Neil Ghani. "Containers: constructing strictly positive types". In: *Theoretical Computer Science* 342.1 (2005), pp. 3–27.

Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. "Non-wellfounded trees in homotopy type theory". In: *arXiv preprint arXiv:1504.02949* (2015).