Strictly Positive Types in Homotopy Type Theory

Final Bachelor Talk

Felix Rech
Advisor: Steven Schäfer

May 12, 2017

Introduction

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

- Construction of nested inductive and coinductive types in a small type theory
- Reduction of coinductive to inductive types
- **But:** They require uniqueness of identity proofs.

Our Contribution

- The same in homotopy type theory (HoTT)
- Reduction of inductive types to natural numbers
- Good conversion behavior in some cases

Motivation

- Simplified models for HoTT
- Smaller trusted core for proof checkers
- Generic programming (and proving)
- In Coq: Real coinductive types without additional axioms

Starting Point

We work in a subset of the type theory from the HoTT book.

- Dependent functions (Π)
- Dependent pairs (Σ)
- Natural numbers ($\mathbb{N}$)
- Equality ($=$)
- Universes ($\mathcal{U}$)
- Univalence
- Propositional Resizing

Univalence

Definition (Equivalence)

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

Examples

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

Axiom (Univalence)

Equivalence is equivalent to equality between two types.

Proposition

$$Univalence \to Funext$$

Propositional Resizing

Definition (Mere Proposition)

A type P is a mere proposition if it has at most one inhabitant.

Examples

- $A \to \mathrm{Unit}$
- $\sum_{x:A} a = x$

Universe Hierarchy

$$\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \cdots$$

One consequence: A function cannot take its own type as argument.

Axiom (Propositional Resizing)

Every mere proposition inhabits the smallest universe.

Construction of Basic Types

Empty Type

$$\mathbf{0} :\equiv 0 = 1$$

Unit

$$\mathbf{1} :\equiv \sum_{n:\mathbb{N}} 0 = n$$

Bool

$$\mathbf{2} :\equiv \sum_{n:\mathbb{N}} n < 2$$

Coproduct

$$A + B :\equiv \sum_{b:\mathbf{2}} \textbf{if } b \textbf{ then } A \textbf{ else } B$$

We get the right conversion behavior!

Propositional Truncation

Given a type A, we want a propositional truncation $\|A\|$ of A with the following properties.

- It is mere proposition.
- For all $x : A$, there is an inhabitant $|x| : \|A\|$.
- For all mere propositions P and functions $f : A \to P$, there is a function $\bar{f} : \|A\| \to P$ such that $\bar{f}(|x|) \equiv f(x)$ for all $x : A$.

Definition

We define the propositional truncation by

$$\|A\| :\equiv \prod_{P:\mathsf{Prop}} (A \to P) \to P.$$

With propositional resizing, the universe of P doesn't matter.

Inductive and Coinductive Types

Inductive (W)

- Intuition: well-founded trees
- Has a constructor
- Has a unique recursor

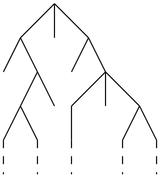
Coinductive (M)

- Intuition: non-well-founded trees
- Has a destructor
- Has a unique corecursor

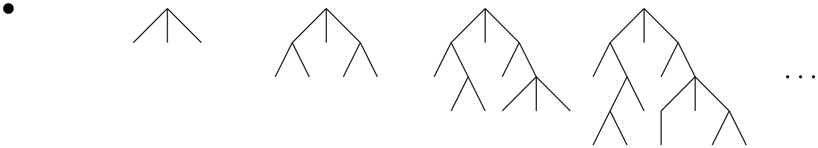
Both are uniquely determined.

Construction of Coinductive Types

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



Representation as sequence of approximations:



No inductive types necessary!

Conversion for Coinductive Types

The Computation Rule

$$\mathsf{destr}(\mathsf{corec}(X, step, x)) = \phi(X, step, x)$$

We want this by definition.

The Solution

$$\mathsf{M}' :\equiv \sum_{(m:\mathsf{M})} \left\| \sum_{(X, step, x)} \big(\mathsf{corec}(X, step, x) = m\big) \right\|$$

$$\cancel{\mathsf{corec}'(X, step, x) :\equiv \big(\mathsf{corec}(X, step, x), X, step, x, \mathsf{refl}\big)}$$

$$\cancel{\mathsf{destr}'\big((m, X, step, x)\big) :\equiv \phi(X, step, x)}$$

The propositional truncation turns M' into a subtype and propositional resizing allows us to use X from an arbitrary universe.

We need to eliminate the propositional truncation.

Construction of Inductive Types

$$\mathsf{W} :\equiv \text{well-founded elements of } \mathsf{M}$$

How do we define well-foundedness?

A tree is well-founded iff it satisfies the <u>induction principle</u> for W.

How do we get the computation rule by conversion?

Every element contains its own recursor:

$$\mathsf{W}' :\equiv \sum_{w:\mathsf{W}} \sum_{r} \mathsf{rec}(-, -, w) = r.$$

Strictly Positive Types

Nested inductive and coinductive types with variables

$$A, B ::= K \mid x \mid A \times B \mid A + B \mid K \to A \mid \mu\, x.A \mid \nu\, x.B$$

where K is a constant type and x a variable.

- The expression $\mu\, x.A$ stands for the inductive type X with a constructor of type $A[X/x] \to X$.
  Example: $\mu\, x.\mathbf{1} + (\mathbb{N} \times x)$ stands for lists of natural numbers.
- The expression $\nu\, x.A$ stands for the coinductive type X with a destructor of type $X \to A[X/x]$.
  Example: $\nu\, x.\mathbf{1} + (\mathbb{N} \times x)$ stands for potentially infinite lists of natural numbers.

The existence and uniqueness of those types is not obvious!

Containers

A polynomial-like normal form for strictly positive types

Example (List)

$$\mathrm{List}(A) \simeq \sum_{n:\boxed{\mathbb{N}}} \boxed{\mathrm{Fin}}(n) \to A$$

In general

A container consists of:

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

Semantics:

$$[\![S \triangleright P]\!]\, A :\equiv \sum_{s:S} P(s) \to A$$

This generalizes to multiple variables.

Construction of Strictly Positive Types

Theorem

Container types are closed under all strictly positive type formers.

Example (Function Type — Simplified)

$$
\begin{aligned}
& K \to [\![S \rhd P]\!]\, A \\
& \qquad \equiv K \to \sum_{s:S} P(s) \to A \\
& \qquad \simeq \sum_{(f:K\to S)} \prod_{(k:K)} P\big(f(k)\big) \to A \\
& \qquad \simeq \sum_{f:K\to A} \left(\sum_{k:K} P\big(f(k)\big) \right) \to A \\
& \qquad \equiv [\![K \to S \rhd \lambda\, f.\, \sum_{k:K} P\big(f(k)\big)]\!]\, A
\end{aligned}
$$

Conclusion

What We Did

1. Construct coinductive types from natural numbers
2. Construct inductive types from coinductive types
3. Obtain some computation rules by conversion
4. Construct all strictly positive types
5. Everything is checked in Coq except that we use built-in types instead of our own constructions in many places.

Possible Next Steps

- Inductive and coinductive families (Example: Vec)
- Conversion without propositional resizing
- Construction of higher inductive types
- 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).

The Univalent Foundations Program. Homotopy Type Theory: Univalent Foundations of Mathematics. Institute for Advanced Study: https://homotopytypetheory.org/book, 2013.