Formalising coinductive containers is hard!

Stefania Damato

University of Nottingham

joint work with Thorsten Altenkirch & Axel Ljungström

Tallinn Theory Seminar, 13 March 2025

Once upon a time ...

Available online at www.sciencedirect.com
SCIENCE @ DIRECT®

Theoretical Computer Science

Theoretical Computer Science 342 (2005) 3–27

www.elsevier.com/locate/tcs

ELSEVIER

Containers: Constructing strictly positive types

Michael Abbott[a], Thorsten Altenkirch[b,*], Neil Ghani[c]

[a] *Diamond Light Source, Rutherford Appleton Laboratory, UK*
[b] *School of Computer Science and Information Technology, Nottingham University, UK*
[c] *Department of Mathematics and Computer Science, University of Leicester, UK*

Abstract

We introduce the notion of a *Martin-Löf category*—a locally cartesian closed category with disjoint coproducts and initial algebras of container functors (the categorical analogue of W-types)—and then establish that nested strictly positive inductive and coinductive types, which we call *strictly positive types*, exist in any Martin-Löf category.

Central to our development are the notions of *containers* and *container functors*. These provide a new conceptual analysis of data structures and polymorphic functions by exploiting dependent type theory as a convenient way to define constructions in Martin-Löf categories. We also show that morphisms between containers can be full and faithfully interpreted as polymorphic functions (i.e. natural transformations) and that, in the presence of W-types, all strictly positive types (including nested inductive and coinductive types) give rise to containers.

Keywords: Type theory; Category theory; Container functors; W-Types; Induction; Coinduction; Initial algebras; Final coalgebras

2. Background

2.1. The categorical semantics of dependent types

This paper can be read in two ways (see Proposition 2.5):

(1) as a construction within the extensional type theory **MLW**$^{\mathbf{ext}}$ (see [8]) with finite types, W-types, a proof of true $\neq$ false and no universes;

(2) as a construction in the internal language of locally cartesian closed categories with disjoint coproducts and initial algebras of container functors in one variable—we call these **Martin-Löf categories**.

Once upon a time . . .

Proposition 5.3. *Given a container* $F \equiv (S \rhd P, Q) \in \mathcal{G}_{I+1}$ *then*

$$[\![\mathsf{W}_S Q \rhd \mathrm{Pos}_{P,\mathsf{sup}^\mu}]\!] X \cong \mu Y.\ [\![F]\!](X, Y);$$

writing $\mu F \equiv (\mathsf{W}_S Q \rhd \mathrm{Pos}_{P,\mathsf{sup}^\mu})$ *we can conclude that* $[\![\mu F]\!] \cong \mu [\![F[-]]\!]$.

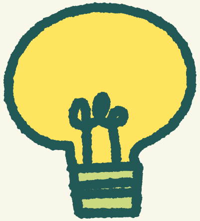
Easy enough.

Proposition 5.4. *Given a container* $F \equiv (S \rhd P, Q) \in \mathcal{G}_{I+1}$ *then*

$$\left[\mathsf{M}_S Q \rhd \mathrm{Pos}_{P,\mathsf{sup}^\nu} \right] X \cong \nu Y.\ [\![F]\!](X, Y);$$

writing $\nu F \equiv (\mathsf{M}_S Q \rhd \mathrm{Pos}_{P,\mathsf{sup}^\nu})$ *we have* $[\![\nu F]\!] \cong \nu [\![F[-]]\!]$.

Confusing.



I should formalise!

The statement (Prop. 5.4)

If $[\![S \triangleleft P]\!] : \mathrm{Set}^{I+1} \to \mathrm{Set}$ is a container functor, then for $X : \mathrm{Set}^{I}$, we know the terminal coalgebra of $[\![S \triangleleft P]\!]\, X : \mathrm{Set} \to \mathrm{Set}$, and its carrier set is some $[\![T \triangleleft Q]\!]\, X$.

'Containers are closed under terminal coalgebras'.

Background : Containers

A container is given by a pair $S : \mathrm{Set}$, $P : S \to \mathrm{Set}$, written $S \lhd P$.

Containers carve out a class of strictly positive types.

✓ $c : (\mathbb{N} \to X) \to X$

✗ $d : (X \to \mathbb{N}) \to X$

E.g. Container representation of list is $\mathbb{N} \lhd \mathrm{Fin}$.

Background: Containers

Containers have a functorial interpretation.

The container functor $[\![S \triangleleft P]\!] : \mathrm{Set} \to \mathrm{Set}$ is defined as:

- $[\![S \triangleleft P]\!] X := \sum_{s:S} (P_s \to X)$
- $[\![S \triangleleft P]\!] f \ (s, g) := (s, f \circ g)$.

Background: Containers

For data types parameterised by 1 or more types, we have I-ary containers given by

$S : \mathrm{Set}$, $\underline{P} : I \to S \to \mathrm{Set}$ for some indexing type I.

Then $[\![S \triangleleft \underline{P}]\!] : \mathrm{Set}^I \to \mathrm{Set}$.

Background: Coinductive types

- Destructors vs constructors
- Copattern matching vs pattern matching

$$
\begin{array}{l}
\textsf{record Stream } (A : \textsf{Type}) : \textsf{Type where} \\
\quad \textsf{coinductive} \\
\quad \textsf{field} \\
\qquad \textsf{hd} : A \\
\qquad \textsf{tl} : \textsf{Stream } A
\end{array}
$$

$$
\begin{array}{l}
\textsf{from} : \mathbb{N} \to \textsf{Stream } \mathbb{N} \\
\textsf{hd } (\textsf{from } n) = n \\
\textsf{tl } (\textsf{from } n) = \textsf{from } (\textsf{suc } n)
\end{array}
$$

Background : The M-type

M is the type of non-wellfounded labelled trees. A tree of type M can have both finite and infinite paths.

```
record M (S : Type) (P : S → Type) : Type where
  coinductive
  field
    shape : S
    pos : P shape → M S P
```

M is the universal type of strictly positive coinductive types (dual to W).

Background : M-type example

To encode the conatural numbers $\mathbb{N}\infty$ via M, we define S & P:

```
record ℕ∞ : Type where
  coinductive
  field
    pred∞ : Maybe ℕ∞
```

$$S = \top \uplus \top$$
$$P\,(\mathsf{inl}\ \_) = \bot$$
$$P\,(\mathsf{inr}\ \_) = \top,$$

So $M\,S\,P \cong \mathbb{N}\infty$.

tree representations	inl tt	inr tt → inl tt	inr tt ↻	
conaturals	zero	succ zero	∞	. . .

Background: Finite paths through an M tree

```
data Pos : M S P → Type where
  here : {m : M S P} → P (shape m) → Pos m
  below : {m : M S P} (p : P (shape m)) → Pos ((pos m) p) → Pos m
```

tree representations	inl tt	inr tt ↘ inl tt	inr tt ↘ inr tt ↘ inl tt
conaturals	zero	succ zero	succ (succ zero)
	↓	↓	↓
	Pos = ∅	here tt : Pos	here tt, below tt (here tt) : Pos

Background: Coinduction in Agda

In vanilla Agda (without postulates):

✓ copattern matching

✓ guarded corecursion

✗ not enough extensionality e.g. no function extensionality

Background : Cubical Agda

Extends Agda with primitives from cubical type theory.

Equality on a type A is now a function of the form $e : I \to A$, where I is the interval (pre-) type.

Has more extensional properties than Agda:

$$\mathsf{funExt} : ((x : A) \to f\ x \equiv g\ x) \to f \equiv g$$
$$\mathsf{funExt}\ p\ i\ x = p\ x\ i$$

The statement, more precisely

For $[\![S \triangleleft \underline{P}, Q]\!] : \mathrm{Set}^{I+1} \to \mathrm{Set}$, and for $\underline{X} : \mathrm{Set}^{I}$,

$$\left([\![M\, S\, Q \triangleleft \mathrm{Pos}]\!]\, \underline{X} \,,\, \bullet \right)$$

is the terminal $[\![S \triangleleft \underline{P}, Q]\!] (\underline{X}, -)$-coalgebra.

Special case: $([\![\mathbb{N}\infty \triangleleft \mathrm{Fin}\infty]\!]\, A, \bullet)$ is the terminal coalgebra of $[\![L]\!] (A, X) := 1 + A \times X$.

What we need to show (modulo indices)

$(\llbracket M\,S\,Q \triangleleft Pos \rrbracket\, X \,,\, out)$ is the terminal

$\llbracket \underbrace{S \triangleleft P,\, Q}_{=:F} \rrbracket\, (X, -)$ coalgebra.

$$\begin{tikzcd}
Y \arrow[rr, "\beta"] \arrow[d, "\bar{\beta}"'] & & \llbracket F \rrbracket (X, Y) \arrow[d, "\llbracket F \rrbracket (X, \bar{\beta})"] \\
\llbracket M\,S\,Q \triangleleft Pos \rrbracket\, X \arrow[rr, "out"'] & & \llbracket F \rrbracket (X, \llbracket M\,S\,Q \triangleleft Pos \rrbracket\, X)
\end{tikzcd}$$

Problem 1: Dependencies between diagrams

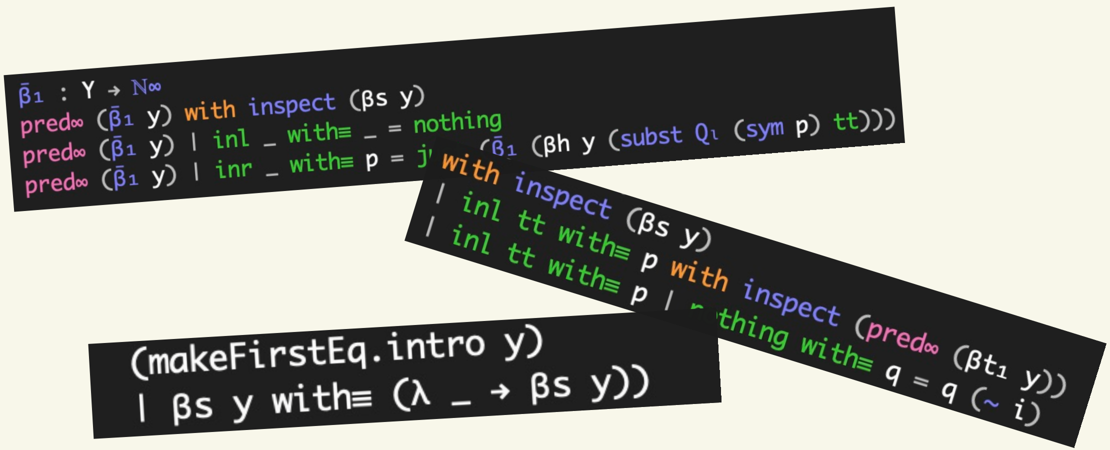
$$\begin{tikzcd}
Y \arrow[r, "{\beta_s,\, \beta_h}"] \arrow[d, "\bar{\beta}_1"'] \arrow[dr, phantom, "1"] & \sum_{s:S} (Q\, s \to Y) \arrow[d, "{(\mathrm{id},\, \bar{\beta}_1 \circ -)}"] \\
M\, S\, Q \arrow[r, "\mathrm{out}"'] & \sum_{s:S} (f : Q\, s \to M\, S\, Q)
\end{tikzcd}$$

2 depends on 1!

$$\begin{tikzcd}
y : Y \arrow[r, "\beta_g"] \arrow[d, "\bar{\beta}_2"'] \arrow[dr, phantom, "2"] & P\, (\beta_s\, y) \to X \arrow[d, "{(\mathrm{id},\, \bar{\beta}_2 \circ -)}"] \\
\mathrm{Pos}\, (\bar{\beta}_1\, y) \to X \arrow[r, "\mathrm{out}"'] & (P\, (\beta_s\, y) \to X) \times ((q : Q\, s) \to \mathrm{Pos}\, (f\, q) \to X)
\end{tikzcd}$$

Problem 2 : Checking intermediate computations

We frequently had to check intermediate computations. Use 'with' abstraction ?



But there are workarounds.

Problem 3: Agda's termination checker

Definitions that should have been accepted raised termination issues, so we had to find workarounds.

```
preFstEq : (y : Y) → β̃₁ y ≡ β̄₁ βs βh y
shape (preFstEq y i) = comm1 i y
pos (preFstEq y i) q =
  hcomp (λ j → λ { (i = i0) → pos (β̃₁ y) q ;
                   (i = i1) → preFstEq (βh y q) j })
        (comm2 y i q)
```

This is an issue with the termination checker: github.com/agda/agda/issues/4740

Lesson 1: Generalised elimination principle

We were getting stuck with the standard elimination principle for Pos. (This is analogous to how path induction does not apply to paths with fixed endpoints.)

Solution: Formulate a 'generalised' elimination principle.

Lesson 2: Proof does not require UIP

Using UIP was tempting and it would have made our lives easier, but we did not use it in our proof. This generalises the original result:

"For $[\![S \lhd \underline{P}, Q]\!] : \overset{\text{Type}}{\cancel{\text{Set}}}^{I+1} \to \overset{\text{Type}}{\cancel{\text{Set}}}$, and for $\underline{X} : \overset{\text{Type}}{\cancel{\text{Set}}}^{I} \to \overset{\text{Type}}{\cancel{\text{Set}}}$,

$$([\![M\, S\, Q \lhd Pos]\!]\, \underline{X}, -)$$

is the terminal $[\![S \lhd \underline{P}, Q]\!] (\underline{X}, -)$-coalgebra. "

Type = wild cat. of types

Lesson 3 : Dealing with the termination checker

- Avoid using 'with' especially in definitions that will later be involved in proofs. Use auxiliary functions instead.

- Use elimination principles, but this might lead to coherences that have to be proved.

Lesson 4: General case is easier?

Initially, we tried a special case of Prop. 5.4: showing that $(\llbracket \mathbb{N}\infty \triangleleft \mathrm{Fin}\,\infty \rrbracket A, -)$ is the terminal $\llbracket L \rrbracket(A, X)$-coalgebra, for

$$\llbracket L \rrbracket (A, X) := 1 + A \times X.$$

In practice, the general formulation is more adapted to the kinds of proofs we are doing.

Conclusion

We formalised 'containers are closed under initial algebras & terminal coalgebras' and did so without UIP, generalising the original results.

Preprint: arxiv.org/abs/2409.02603

Thank you!