Revisiting Containers
in Cubical Agda

Stefania Damato   Thorsten Altenkirch

University of Nottingham, UK

TYPES Conference

12th June 2023

WHY do we need containers (a.k.a. polynomial functors)?

Strict positivity!

WHY do we need containers (a.k.a. polynomial functors)?

Strict positivity!

```
data Contra : Set where
    c : ((Contra → Bool) → Bool) → Contra
```

👎

WHY do we need containers (a.k.a. polynomial functors)?

Strict positivity!

```
data Contra : Set where
    c : ((Contra → Bool) → Bool) → Contra
```

```
data ∞Tree : Set where
    leaf : ∞Tree
    node : (ℕ → ∞Tree) → ∞Tree
```

HOW do we approach strict positivity?

1. Syntactically

HOW do we approach strict positivity?

1. Syntactically [Abel and Altenkirch, 2000]

> "We define the set of types in which the variables $\boldsymbol{X}$ occur at most strictly positive $\mathsf{Ty}(\boldsymbol{X})$ inductively by the following rules:
>
> $$\frac{}{0, 1 \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Const}) \qquad \frac{}{X_i \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Var}) \qquad \frac{\sigma \in \mathsf{Ty}() \qquad \tau \in \mathsf{Ty}(\boldsymbol{X})}{\sigma \to \tau \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Arr})$$
>
> $$\frac{\sigma, \tau \in \mathsf{Ty}(\boldsymbol{X})}{\sigma + \tau, \sigma \times \tau \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Sum}),(\mathrm{Prod}) \qquad \frac{\sigma \in \mathsf{Ty}(\boldsymbol{X}, Y)}{\mu Y.\sigma \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Mu})$$"

HOW do we approach strict positivity?

1. Syntactically [Abel and Altenkirch, 2000]

> “ We define the set of types in which the variables $\boldsymbol{X}$ occur at most strictly positive $\mathsf{Ty}(\boldsymbol{X})$ inductively by the following rules:
>
> $$\frac{}{0, 1 \in \mathsf{Ty}(\boldsymbol{X})}\,(\text{Const}) \qquad \frac{}{X_i \in \mathsf{Ty}(\boldsymbol{X})}\,(\text{Var}) \qquad \frac{\sigma \in \mathsf{Ty}() \qquad \tau \in \mathsf{Ty}(\boldsymbol{X})}{\sigma \to \tau \in \mathsf{Ty}(\boldsymbol{X})}\,(\text{Arr})$$
>
> $$\frac{\sigma, \tau \in \mathsf{Ty}(\boldsymbol{X})}{\sigma + \tau, \sigma \times \tau \in \mathsf{Ty}(\boldsymbol{X})}\,(\text{Sum}),(\text{Prod}) \qquad \frac{\sigma \in \mathsf{Ty}(\boldsymbol{X}, Y)}{\mu Y.\sigma \in \mathsf{Ty}(\boldsymbol{X})}\,(\text{Mu})$$ ”

2. Semantically

HOW do we approach strict positivity?

1. Syntactically [Abel and Altenkirch, 2000]

> We define the set of types in which the variables $\boldsymbol{X}$ occur at most strictly positive $\mathsf{Ty}(\boldsymbol{X})$ inductively by the following rules:
>
> $$\frac{}{0, 1 \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Const}) \qquad \frac{}{X_i \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Var}) \qquad \frac{\sigma \in \mathsf{Ty}() \qquad \tau \in \mathsf{Ty}(\boldsymbol{X})}{\sigma \to \tau \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Arr})$$
>
> $$\frac{\sigma, \tau \in \mathsf{Ty}(\boldsymbol{X})}{\sigma + \tau, \sigma \times \tau \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Sum}),(\mathrm{Prod}) \qquad \frac{\sigma \in \mathsf{Ty}(\boldsymbol{X}, Y)}{\mu Y.\sigma \in \mathsf{Ty}(\boldsymbol{X})}\,(\mathrm{Mu})$$

2. Semantically

   Use **containers** to provide a **categorical semantics** for strictly positive types.

WHAT are containers?

Available online at www.sciencedirect.com
SCIENCE @ DIRECT®
ELSEVIER
Theoretical Computer Science 342 (2005) 3–27
Theoretical Computer Science
www.elsevier.com/locate/tcs

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

WHAT are containers?


Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

ELSEVIER

Theoretical Computer Science 342 (2005) 3–27

Theoretical Computer Science

www.elsevier.com/locate/tcs

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

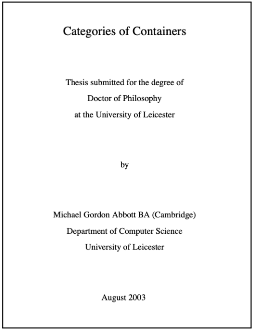
Abstract

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.

© 2005 Elsevier B.V. All rights reserved.

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


Categories of Containers

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Michael Gordon Abbott BA (Cambridge)
Department of Computer Science
University of Leicester

August 2003

WHAT are containers?

ELSEVIER

Available online at www.sciencedirect.com

SCIENCE @ DIRECT®

Theoretical Computer Science 342 (2005) 3–27

Theoretical Computer Science

www.elsevier.com/locate/tcs

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

Categories of Containers

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Michael Gordon Abbott BA (Cambridge)
Department of Computer Science
University of Leicester

August 2003

Higher Order Containers

Thorsten Altenkirch[1], Paul Levy[2], and Sam Staton[3]

[1] University of Nottingham
[2] University of Birmingham
[3] University of Cambridge

Abstract. Containers are a semantic way to talk about strictly positive types. In previous work it was shown that containers are closed under various constructions including products, coproducts, initial algebras and terminal coalgebras. In the present paper we show that, surprisingly, the category of containers is cartesian closed, giving rise to a full cartesian closed subcategory of endofunctors. The result has interesting applications in generic programming and representation of higher order abstract syntax. We also show that while the category of containers has finite limits, it is not locally cartesian closed.

WHAT are containers?

Available online at www.sciencedirect.com
SCIENCE @ DIRECT
ELSEVIER
Theoretical Computer Science 342 (2005) 3–27
Theoretical Computer Science
www.elsevier.com/locate/tcs

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

Categories of Containers

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Michael Gordon Abbott BA (Cambridge)
Department of Computer Science
University of Leicester

August 2003

Higher Order Containers

Thorsten Altenkirch[1], Paul Levy[2], and Sam Staton[3]

[1] University of Nottingham
[2] University of Birmingham
[3] University of Cambridge

Abstract. Containers are a semantic way to talk about strictly positive types. In previous work it was shown that containers are closed under various constructions including products, coproducts, initial algebras and terminal coalgebras. In the present paper we show that, surprisingly, the category of containers is cartesian closed, giving rise to a full cartesian closed subcategory of endofunctors. The result has interesting applications in generic programming and representation of higher order abstract syntax. We also show that while the category of containers has finite limits, it is not locally cartesian closed.

Indexed Containers

Thorsten Altenkirch Neil Ghani Peter Hancock
Conor McBride Peter Morris

May 12, 2014

Abstract

We show that the syntactically rich notion of strictly positive families can be reduced to a core type theory with a fixed number of type constructors exploiting the novel notion of indexed containers. As a result, we show indexed containers provide normal forms for strictly positive families in much the same way that containers provide normal forms for strictly positive types. Interestingly, this step from containers to indexed containers is achieved without having to extend the core type theory. Most of the construction presented here has been formalized using the Agda system – the missing bits are due to the current shortcomings of the Agda system.

WHAT are containers?

Available online at www.sciencedirect.com
SCIENCE @ DIRECT®
ELSEVIER
Theoretical Computer Science 342 (2005) 3–27
Theoretical Computer Science
www.elsevier.com/locate/tcs

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

Categories of Containers

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Michael Gordon Abbott BA (Cambridge)
Department of Computer Science
University of Leicester

August 2003

Higher Order Containers

Thorsten Altenkirch[1], Paul Levy[2], and Sam Staton[3]

[1] University of Nottingham
[2] University of Birmingham
[3] University of Cambridge

Abstract. Containers are a semantic way to talk about strictly positive types. In previous work it was shown that containers are closed under various constructions including products, coproducts, initial algebras and terminal coalgebras. In the present paper we show that, surprisingly, the category of containers is cartesian closed, giving rise to a full cartesian closed subcategory of endofunctors. The result has interesting applications in generic programming and representation of higher order abstract syntax. We also show that while the category of containers has finite limits, it is not locally cartesian closed.

Indexed Containers

Thorsten Altenkirch Neil Ghani Peter Hancock
Conor McBride Peter Morris

May 12, 2014

Abstract

We show that the syntactically rich notion of strictly positive families can be reduced to a core type theory with a fixed number of type constructors exploiting the novel notion of indexed containers. As a result, we show indexed containers provide normal forms for strictly positive families in much the same way that containers provide normal forms for strictly positive types. Interestingly, this step from containers to indexed containers is achieved without having to extend the core type theory. Most of the construction presented here has been formalized using the Agda system – the missing bits are due to the current shortcomings of the Agda system.

. . . and many more.

WHAT are containers? An overview

Class of types	**Functor type**	**Category theory semantics**	**Type theoretic normal form**	**Universal type**
ordinary inductive types e.g. $\mathbb{N}$: Set	**Set** → **Set**	initial algebras of endofunctors on **Set**	containers	W-type

WHAT are containers? An overview

Class of types	**Functor type**	**Category theory semantics**	**Type theoretic normal form**	**Universal type**
ordinary inductive types e.g. $\mathbb{N}$: Set	$\mathbf{Set} \to \mathbf{Set}$	initial algebras of endofunctors on **Set**	containers	W-type
inductive families e.g. $\texttt{Fin} : \mathbb{N} \to \text{Set}$	$(\mathbf{I} \to \mathbf{Set}) \to (\mathbf{I} \to \mathbf{Set})$	initial algebras of endofuntors on $\mathbf{Set}^I$	indexed containers	WI-type

WHAT are containers? An overview

Class of types	**Functor type**	**Category theory semantics**	**Type theoretic normal form**	**Universal type**
ordinary inductive types e.g. $\mathbb{N}$: Set	$\mathbf{Set} \to \mathbf{Set}$	initial algebras of endofunctors on **Set**	containers	W-type
inductive families e.g. Fin : $\mathbb{N} \to$ Set	$(\mathbf{I} \to \mathbf{Set}) \to (\mathbf{I} \to \mathbf{Set})$	initial algebras of endofuntors on $\mathbf{Set}^I$	indexed containers	WI-type
QIITs e.g. Con : Set, Ty : Con $\to$ Set	sequence of functors L_n and R_n and sequence of categories of dialgebras	initial object in last constructed category of dialgebras $\mathbf{A_n}$	representations constructed via generalised containers	?

WHAT are containers? An overview

Class of types	Functor type	Category theory semantics	Type theoretic normal form	Universal type
ordinary inductive types e.g. $\mathbb{N}$: Set	**Set** → **Set**	initial algebras of endofunctors on **Set**	containers	W-type

W-type

The type of **well-founded labelled trees**.

```
data W (S : Set) (P : S → Set) : Set where
    sup : (s : S) → (P s → W S P) → W S P
```

W-type

The type of **well-founded labelled trees**.

```
data W (S : Set) (P : S → Set) : Set where
    sup : (s : S) → (P s → W S P) → W S P
```

ℕ as a W-type

```
data ℕ : Set where
    zero : ℕ
    succ : ℕ → ℕ
```

W-type

The type of **well-founded labelled trees**.

```
data W (S : Set) (P : S → Set) : Set where
    sup : (s : S) → (P s → W S P) → W S P
```

ℕ as a W-type

```
data ℕ : Set where
    zero : ℕ
    succ : ℕ → ℕ
```

$S := \mathbf{1} + \mathbf{1}$

W-type

The type of **well-founded labelled trees**.

```
data W (S : Set) (P : S → Set) : Set where
    sup : (s : S) → (P s → W S P) → W S P
```

ℕ as a W-type

```
data ℕ : Set where
    zero : ℕ
    succ : ℕ → ℕ
```

$S := \mathbf{1} + \mathbf{1}$

$P(\mathsf{inl}\,\star) := \mathbf{0}$

$P(\mathsf{inr}\,\star) := \mathbf{1}$

W-type

The type of **well-founded labelled trees**.

```
data W (S : Set) (P : S → Set) : Set where
    sup : (s : S) → (P s → W S P) → W S P
```

ℕ as a W-type

```
data ℕ : Set where
    zero : ℕ
    succ : ℕ → ℕ
```

$$S := 1 + 1$$

$$P(\mathsf{inl}\,\star) := 0$$

$$P(\mathsf{inr}\,\star) := 1$$

$\mathbb{N} \cong W\,S\,P$.

W-type

The type of **well-founded labelled trees**.

```
data W (S : Set) (P : S → Set) : Set where
    sup : (s : S) → (P s → W S P) → W S P
```

ℕ as a W-type

```
data ℕ : Set where
    zero : ℕ
    succ : ℕ → ℕ
```

$\mathtt{S} := \mathbf{1} + \mathbf{1}$

$\mathtt{P}(\mathsf{inl}\,\star) := \mathbf{0}$

$\mathtt{P}(\mathsf{inr}\,\star) := \mathbf{1}$

$\mathbb{N} \cong \mathtt{W\,S\,P}$.

$z : \mathtt{W\,S\,P}$

$z := \mathsf{sup}\,(\mathsf{inl}\,\star)\,(\lambda\,())$

$s : \mathtt{W\,S\,P} \rightarrow \mathtt{W\,S\,P}$

$s\,n := \mathsf{sup}\,(\mathsf{inr}\,\star)\,(\lambda\,\_.n)$

W-type

The type of **well-founded labelled trees**.

```
data W (S : Set) (P : S → Set) : Set where
    sup : (s : S) → (P s → W S P) → W S P
```

ℕ as a W-type

```
data ℕ : Set where
    zero : ℕ
    succ : ℕ → ℕ
```

$\mathbb{N} \cong \mathtt{W\,S\,P}$.

$$\mathtt{S} := \mathbf{1} + \mathbf{1}$$

$$\mathtt{P}(\mathsf{inl}\,\star) := \mathbf{0}$$

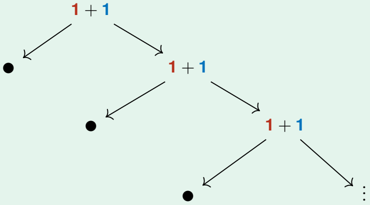
$$\mathtt{P}(\mathsf{inr}\,\star) := \mathbf{1}$$

$$z : \mathtt{W\,S\,P}$$

$$z := \mathsf{sup}\,(\mathsf{inl}\,\star)\,(\lambda\,())$$

$$s : \mathtt{W\,S\,P} \to \mathtt{W\,S\,P}$$

$$s\,n := \mathsf{sup}\,(\mathsf{inr}\,\star)\,(\lambda\,\_.n)$$



Containers

Definition

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

Containers

Definition

A *container* is a pair S : Set, $P : S \to$ Set, written as $S \lhd P$.

$\mathbb{N}$ as (the initial algebra of) an endofunctor

$$F_{\mathbb{N}} : \mathbf{Set} \to \mathbf{Set}$$
$$F_{\mathbb{N}}(X) := \mathbf{1} + X$$

Containers

Definition

A *container* is a pair S : Set, $P : S \to$ Set, written as $S \lhd P$.

$\mathbb{N}$ as (the initial algebra of) an endofunctor

$$F_{\mathbb{N}} : \mathbf{Set} \to \mathbf{Set}$$
$$F_{\mathbb{N}}(X) := \mathbf{1} + X$$

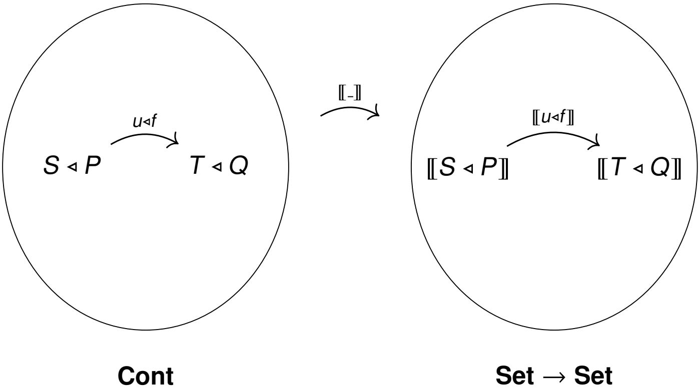
Definition

Extension functor $[\![S \lhd P]\!] : \mathbf{Set} \to \mathbf{Set}$ is defined on objects by $X \mapsto \sum(s : S)(P\,s \to X)$.

Categories of containers

Definition

Extension functor $[\![S \triangleleft P]\!] : \mathbf{Set} \to \mathbf{Set}$ is defined on objects by $X \mapsto \sum(s : S)(P\,s \to X)$.



Contributions

- Formalisation in Cubical Agda of
  - generalised containers
  - category **Cont**
  - functor $[\![\_]\!] : \mathbf{Cont} \to (\mathbf{Set} \to \mathbf{Set})$
  - proof that $[\![\_]\!]$ is full and faithful (NEW presentation)
  - (WIP) proofs that ordinary container functors are closed under fixed points
  - proof that indexed containers are a special case of generalised containers.

  https://github.com/stefaniatadama/TYPES-23

- (WIP) Updated, type-theoretic review paper on containers, including discussion on generalised containers.

$[\![\_]\!]$ is full and faithful, using Yoneda

Given $\alpha \colon [\![S \triangleleft P]\!] \to [\![T \triangleleft Q]\!]$, we obtain a container morphism.

$$\int_{X:\mathrm{Set}} ([\![S \triangleleft P]\!]\, X \to [\![T \triangleleft Q]\!]\, X)$$

$$= \int_{X:\mathrm{Set}} \Big(\sum_{s:S} (P\,s \to X)\Big) \to [\![T \triangleleft Q]\!] X\Big)$$ expanding definition of $[\![S \triangleleft P]\!]\, X$

$$\cong \int_{X:\mathrm{Set}} \prod_{s:S} ((P\,s \to X) \to [\![T \triangleleft Q]\!] X)$$ currying in **Set**: $\Pi\,((\Sigma\,A\,B)\,C) \cong \Pi\,(A\,(\Pi\,B\,C))$

$$\cong \prod_{s:S} \int_{X:\mathrm{Set}} ((P\,s \to X) \to [\![T \triangleleft Q]\!] X)$$ $\int$ and Π commute

$$\cong \prod_{s:S} [\![T \triangleleft Q]\!]\,(P\,s)$$ covariant Yoneda lemma: for $F \colon \mathbf{C} \to \mathbf{Set}$, $A : |\mathbf{C}|$, $\int_{X:|\mathbf{C}|} (\mathbf{C}(A, X), F\,X) \cong F\,A$

$$= \prod_{s:S} \sum_{t:T} (Q\,t \to P\,s)$$ expanding definition of $[\![T \triangleleft Q]\!]\, X$

$$\cong \sum (u \colon S \to T) \Big(\prod_{s:S} Q(u\,s) \to P\,s\Big)$$ type theoretic axiom of choice

$$= (S \triangleleft P) \to (T \triangleleft Q)$$ definition of container morphism

$\llbracket \_ \rrbracket$ is full and faithful, using Yoneda

Given $\alpha \colon \llbracket S \triangleleft P \rrbracket \to \llbracket T \triangleleft Q \rrbracket$, we obtain a container morphism.

$$\int_{X:\mathrm{Set}} (\llbracket S \triangleleft P \rrbracket\, X \to \llbracket T \triangleleft Q \rrbracket\, X)$$

$$= \int_{X:\mathrm{Set}} \Big(\sum_{s:S} (P\, s \to X)) \to \llbracket T \triangleleft Q \rrbracket X \Big)$$

expanding definition of $\llbracket S \triangleleft P \rrbracket\, X$

$$\cong \int_{X:\mathrm{Set}} \prod_{s:S} ((P\, s \to X) \to \llbracket T \triangleleft Q \rrbracket X)$$

currying in **Set**:
$\Pi\,((\Sigma\, A\, B)\, C) \cong \Pi\,(A\,(\Pi\, B\, C))$

$$\cong \prod_{s:S} \int_{X:\mathrm{Set}} ((P\, s \to X) \to \llbracket T \triangleleft Q \rrbracket X)$$

$\int$ and Π commute

$$\cong \prod_{s:S} \llbracket T \triangleleft Q \rrbracket\, (P\, s)$$

covariant Yoneda lemma:
for $F \colon \mathbf{C} \to \mathbf{Set}, A : |\mathbf{C}|$,
$\int_{X:|\mathbf{C}|} (\mathbf{C}(A, X), F\, X) \cong F\, A$

$$= \prod_{s:S} \sum_{t:T} (Q\, t \to P\, s)$$

expanding definition of $\llbracket T \triangleleft Q \rrbracket\, X$

$$\cong \sum (u \colon S \to T) \Big(\prod_{s:S} Q(u\, s) \to P\, s \Big)$$

type theoretic axiom of choice

$$= (S \triangleleft P) \to (T \triangleleft Q)$$

definition of container morphism

Generalised containers

Definition

Given category **C**, a *generalised container* is a pair $S : Set, P : S \to |\mathbf{C}|$.

The *extension functor* $[\![S \triangleleft P]\!] : \mathbf{C} \to \mathbf{Set}$ is defined on objects by $X \mapsto \sum(s : S)(\mathbf{C}(P\,s, X))$.

Generalised containers

Definition

Given category **C**, a *generalised container* is a pair $S : Set, P : S \to |\mathbf{C}|$.

The *extension functor* $[\![S \triangleleft P]\!] : \mathbf{C} \to \mathbf{Set}$ is defined on objects by $X \mapsto \sum(s : S)(\mathbf{C}(P\,s, X))$.

Useful for:

- strictly positive QIITs.
- container model of type theory.

Conclusion

- Containers: a semantic way to talk about **strictly positive types**.
- They form a category **Cont** which is Cartesian closed.
- They are a **normal form** for strictly positive types.
  - Unique representation as containers.
  - Polymorphic functions on strictly positive types have a unique representation as container morphisms.

Conclusion

- Containers: a semantic way to talk about **strictly positive types**.
- They form a category **Cont** which is Cartesian closed.
- They are a **normal form** for strictly positive types.
  - Unique representation as containers.
  - Polymorphic functions on strictly positive types have a unique representation as container morphisms.

Thank you!

References I

Abbott, M., Altenkirch, T., and Ghani, N. (2005).
Containers: Constructing strictly positive types.
Theoretical Computer Science, 342(1):3–27.

Abbott, M. G. (2003).
Categories of Containers.
PhD thesis, University of Leicester.

Abel, A. and Altenkirch, T. (2000).
A predicative strong normalisation proof for a λcalculus with interleaving inductive types.
In *Types for Proofs and Programs*, pages 21–40, Berlin, Heidelberg. Springer Berlin Heidelberg.

References II

Altenkirch, T., Ghani, N., Hancock, P., Mcbride, C., and Morris, P. (2015).

Indexed containers.

Journal of Functional Programming, 25.

Altenkirch, T. and Kaposi, A. (2021).

A container model of type theory.

In *TYPES 2021*.

Altenkirch, T., Levy, P., and Staton, S. (2010).

Higher-order containers.

In *Programs, Proofs, Processes*, pages 11–20, Berlin, Heidelberg. Springer Berlin Heidelberg.

References III

Gambino, N. and Hyland, M. (2004).
Wellfounded trees and dependent polynomial functors.
In *Types for Proofs and Programs*, pages 210–225, Berlin, Heidelberg. Springer Berlin Heidelberg.