Containers in Homotopy Type Theory

Stefania Damato

based on joint work with Thorsten Altenkirch

HoTT/UF 2026

Aarhus, Denmark

Outline

Motivation: Semantics for inductive and coinductive types

Simple inductive types

Data type

$$\begin{aligned}&\mathsf{data}\ \mathbb{N} : \mathsf{Set}\ \mathsf{where}\\ &\quad \mathsf{zero} : \mathbb{N}\\ &\quad \mathsf{suc} : \mathbb{N} \to \mathbb{N}\end{aligned}$$

Semantics: Initial algebra of the signature functor $F_{\mathbb{N}}$

$$\begin{aligned}F_{\mathbb{N}} &: \mathsf{Set} \to \mathsf{Set}\\ F_{\mathbb{N}}\ X &:= \top + X\end{aligned}$$

Normal form: Set containers

Also: Simple coinductive types

Data type

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

Semantics: Terminal coalgebra of the signature functor $F_{\mathbb{N}}$

$$F_{\mathbb{N}} : \mathrm{Set} \to \mathrm{Set}$$
$$F_{\mathbb{N}}\ X := \top + X$$

Normal form: Set containers

Simple parameterised inductive types

Data type

$$\begin{aligned}&\mathsf{data}\ \mathsf{List}\ (A : \mathsf{Set}) : \mathsf{Set}\ \mathsf{where}\\ &\quad [] : \mathsf{List}\ A\\ &\quad \_::\_ : A \to \mathsf{List}\ A \to \mathsf{List}\ A\end{aligned}$$

Semantics: Initial algebra of the signature functor F_{List}

$$F_{\mathsf{List}} : \mathsf{Set}^2 \to \mathsf{Set}$$
$$F_{\mathsf{List}}\ (A, X) :\equiv \top + A \times X$$

Normal form: n-ary containers

Dependent types (a.k.a. Inductive families a.k.a. Indexed-inductive types)

Data type

$$\begin{aligned}&\mathsf{data}\ \mathsf{Vec}\ (A : \mathsf{Set}) : \mathbb{N} \to \mathsf{Set}\ \mathsf{where}\\ &\quad [] : \mathsf{Vec}\ A\ \mathsf{zero}\\ &\quad \_::\_ : \{n : \mathbb{N}\} \to A \to \mathsf{Vec}\ A\ n \to \mathsf{Vec}\ A\ (\mathsf{suc}\ n)\end{aligned}$$

Semantics: Inital algebra of signature functor F_{Vec}

$$F_{\mathsf{Vec}} : \mathsf{Set} \to (\mathbb{N} \to \mathsf{Set}) \to (\mathbb{N} \to \mathsf{Set})$$

$$F_{\mathsf{Vec}}\ A\ X\ m := (m \equiv \mathsf{zero}) + \sum_{n:\mathbb{N}} (m \equiv \mathsf{suc}\ n) \times A \times X\ n$$

Normal form: Indexed containers

Inductive-inductive types (IITs)

Data type

```
data Con : Set
data Ty : Con → Set

data Con where
  ◇ : Con
  _,_ : (Γ : Con) → Ty Γ → Con

data Ty where
  ι : (Γ : Con) → Ty Γ
  σ : (Γ : Con) (A : Ty Γ) → Ty (Γ , A) → Ty Γ
```

Semantics: Initial algebra in a category of dialgebras constructed iteratively [ACD+18]

Normal form: ?

Quotient inductive-inductive types (QIITs)

Data type (in pseudo-Agda code)

```
data Con : Set
data Ty : Con → Set

data Con where
  ◇ : Con
  _,_ : (Γ : Con) → Ty Γ → Con
  eq : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) →
       ((Γ , A) , B) ≡ (Γ , ( σ Γ A B))

data Ty where
  ι : (Γ : Con) → Ty Γ
  σ : (Γ : Con) (A : Ty Γ) → Ty (Γ , A) → Ty Γ
```

Semantics: Initial algebra in a category of dialgebras constructed iteratively [ACD+18]

Normal form: ? [KKA19]

Higher inductive types (HITs)

Data type

```
data S¹ : Type where
  base : S¹
  loop : base ≡ base
```

Semantics: ? [CHM18, LS19, CH19]

Normal form: ?

Summary

Class of types	**Category theory semantics**	**Type theoretic normal form**
simple inductive types e.g. $\mathbb{N}$: Set	initial algebras of functors Set → Set	containers
dependent types e.g. Vec : $\mathbb{N}$ → Set	initial algebras of functors $(I \to \text{Set}) \to (I \to \text{Set})$	indexed containers
QIITs e.g. Con : Set, Ty : Con → Set	initial objects in categories of dialgebras	?
HITs e.g. S^1 : Type	?	?

Container theory

The W-type: A universal type

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

The W-type: A universal type

$$\begin{aligned}&\mathsf{data}\ \mathsf{W}\ (S : \mathsf{Set})\ (P : S \to \mathsf{Set}) : \mathsf{Set}\ \mathsf{where}\\ &\quad \mathsf{sup} : (s : S) \to (P\ s \to \mathsf{W}\ S\ P) \to \mathsf{W}\ S\ P\end{aligned}$$

Example: $\mathbb{N}$ as a W-type

$$\begin{aligned}&\mathsf{data}\ \mathbb{N} : \mathsf{Set}\ \mathsf{where}\\ &\quad \mathsf{zero} : \mathbb{N}\\ &\quad \mathsf{suc} : \mathbb{N} \to \mathbb{N}\end{aligned}$$

$$S := \top + \top$$

$$\begin{aligned}P\ (\mathsf{inl}\ \mathsf{tt}) &:= \bot\\ P\ (\mathsf{inr}\ \mathsf{tt}) &:= \top\end{aligned}$$

The W-type: A universal type

$$\begin{aligned}&\mathsf{data}\ \mathsf{W}\ (S : \mathsf{Set})\ (P : S \to \mathsf{Set}) : \mathsf{Set}\ \mathsf{where}\\ &\quad \mathsf{sup} : (s : S) \to (P\ s \to \mathsf{W}\ S\ P) \to \mathsf{W}\ S\ P\end{aligned}$$

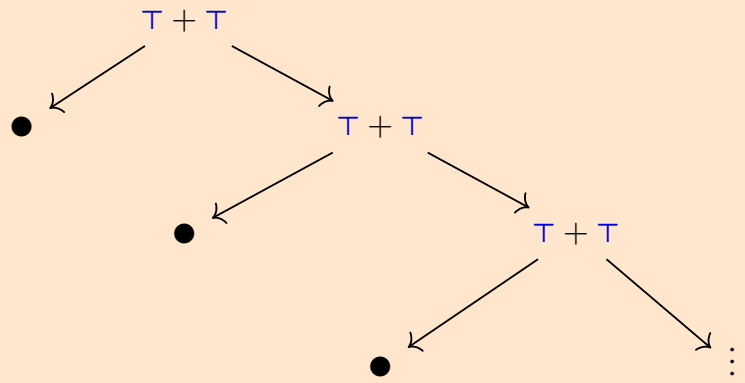
Example: $\mathbb{N}$ as a W-type

$$\begin{aligned}&\mathsf{data}\ \mathbb{N} : \mathsf{Set}\ \mathsf{where}\\ &\quad \mathsf{zero} : \mathbb{N}\\ &\quad \mathsf{suc} : \mathbb{N} \to \mathbb{N}\end{aligned}$$

$$S := \top + \top$$

$$\begin{aligned}P\ (\mathsf{inl}\ \mathsf{tt}) &:= \bot\\ P\ (\mathsf{inr}\ \mathsf{tt}) &:= \top\end{aligned}$$

Then $\mathbb{N} \cong \mathsf{W}\ S\ P$.



Set containers

Definition

A **set container** is a pair $S : \text{Set}, P : S \to \text{Set}$, written $S \triangleleft P$.

Set containers

Definition

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

Set containers form a category **Cont**.

Definition

A **set container morphism** $(S \triangleleft P) \to (T \triangleleft Q)$ is a pair

$$u : S \to T$$
$$f : (s : S) \to Q\,(u\,s) \to P\,s$$

Set containers

Definition

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

Set containers form a category **Cont**.

Definition

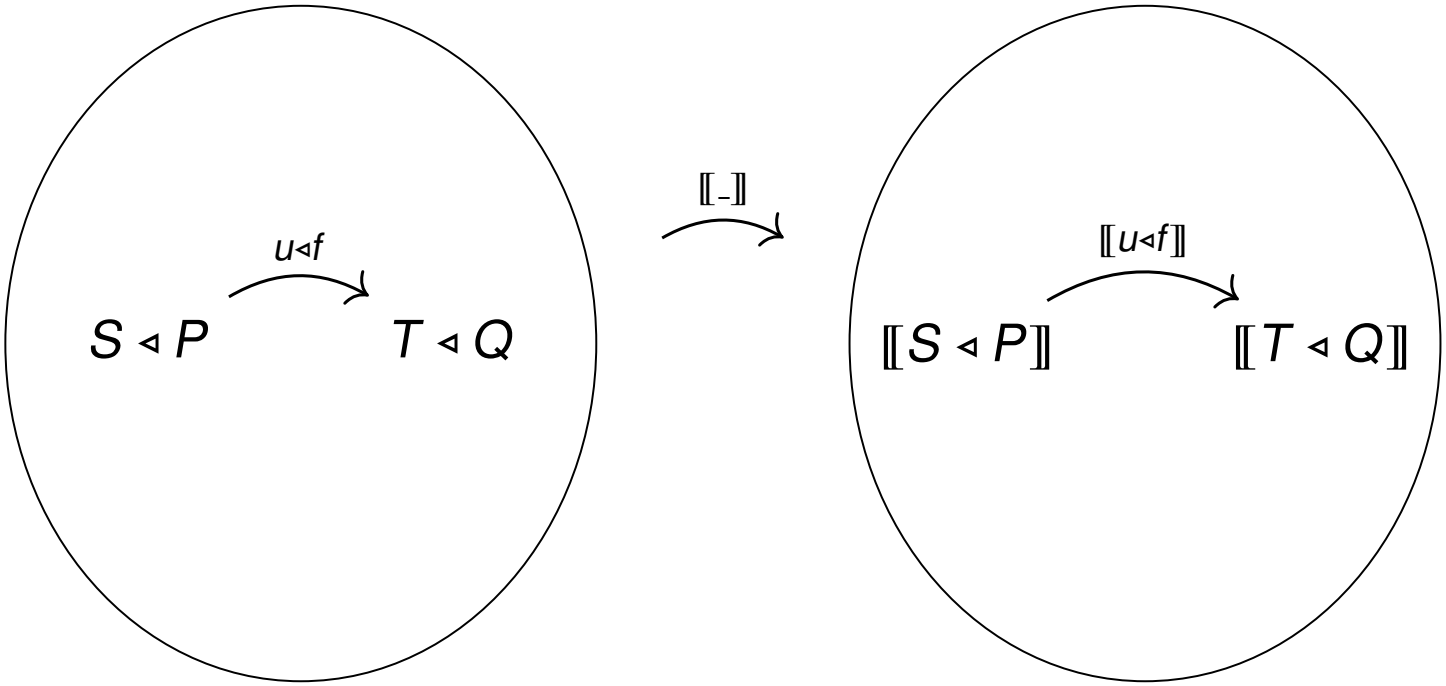
A **set container morphism** $(S \triangleleft P) \to (T \triangleleft Q)$ is a pair

$$u : S \to T$$
$$f : (s : S) \to Q\,(u\,s) \to P\,s$$

Cont is closed under products, coproducts, composition, and exponentiation.

Functorial interpretation

Set containers embed fully faithfully into **Set** endofunctors via $[\![\_]\!]$.



Cont $[\![\_]\!]$ $[\mathbf{Set}, \mathbf{Set}]$

Note: Set containers are polynomial functors on **Set**.

Functorial interpretation (cont.)

Definition (Container functor)

$$[\![S \triangleleft P]\!] : \mathbf{Set} \to \mathbf{Set}$$

$$[\![S \triangleleft P]\!]\, X := \sum_{s:S} (P\, s \to X)$$

Functorial interpretation (cont.)

Definition (**Container functor**)

$$[\![S \triangleleft P]\!] : \mathbf{Set} \to \mathbf{Set}$$

$$[\![S \triangleleft P]\!]\, X := \sum_{s:S} (P\, s \to X)$$

Example: $\mathbb{N}$'s functorial interpretation

$$F_{\mathbb{N}} : \mathrm{Set} \to \mathrm{Set}$$

$$F_{\mathbb{N}}\, X := \top + X$$

$$\cong \sum_{s:S} (P\, s \to X)$$

$$\cong [\![S \triangleleft P]\!]\, X$$

$$S := \top + \top$$

$$P\,(\mathrm{inl}\ \mathrm{tt}) := \bot$$

$$P\,(\mathrm{inr}\ \mathrm{tt}) := \top$$

Closure under taking fixed points

Container functors are closed under initial algebras and terminal coalgebras [AAG05]:

If $[\![S \triangleleft P]\!] : \mathbf{Set}^{I+1} \to \mathbf{Set}$, then for $X : \mathbf{Set}^I$, $Y : \mathbf{Set}$,

$$\mu Y.[\![S \triangleleft P]\!]\,(X, Y) : \mathbf{Set}^I \to \mathbf{Set}$$

$$\nu Y.[\![S \triangleleft P]\!]\,(X, Y) : \mathbf{Set}^I \to \mathbf{Set}$$

are also container functors.

Closure under taking fixed points

Container functors are closed under initial algebras and terminal coalgebras [AAG05]:

If $[\![S \triangleleft P]\!] : \mathbf{Set}^{I+1} \to \mathbf{Set}$, then for $X : \mathbf{Set}^I$, $Y : \mathbf{Set}$,

$$\mu Y.[\![S \triangleleft P]\!]\,(X, Y) : \mathbf{Set}^I \to \mathbf{Set}$$

$$\nu Y.[\![S \triangleleft P]\!]\,(X, Y) : \mathbf{Set}^I \to \mathbf{Set}$$

are also container functors.

Our contribution: This also works if we replace **Set** by **Type**, the wild category of types [DAL25].

Indexed containers

Definition

An **indexed container** with index sets I, J is given by a pair $S : J \to \mathrm{Set}$ and $P : (j : J) \to S\, j \to I \to \mathrm{Set}$, which we write as $S \triangleleft^{\star} P$.

Indexed containers over index sets I, J embed fully faithfully into functors $(I \to \mathbf{Set}) \to (J \to \mathbf{Set})$

$$\mathbf{ICont}_{I,J} \xrightarrow{[\![\_]\!]} [\mathbf{Fam}_I, \mathbf{Fam}_J].$$

Generalised containers

Definition

A **generalised container** over some category $\mathbf{C}$ is given by a pair $S : \mathrm{Set}$ and $P : S \to \mathbf{C}$, which we write as $S \triangleleft_{\mathbf{C}} P$.

Generalised containers over $\mathbf{C}$ embed fully faithfully into functors $\mathbf{C} \to \mathbf{Set}$

$$\mathbf{GCont}_{\mathbf{C}} \xrightarrow{[\![-]\!]} [\mathbf{C}, \mathbf{Set}].$$

Summary

Container type	**Functorial interpretation**
set containers	$\mathbf{Set} \to \mathbf{Set}$
indexed containers	$(I \to \mathbf{Set}) \to (J \to \mathbf{Set})$
generalised containers	$\mathbf{C} \to \mathbf{Set}$

QIIT semantics

QIITs as initial objects in a category of dialgebras

It is unclear whether we can express (Q)IITs as endofunctors.

QIITs as initial objects in a category of dialgebras

It is unclear whether we can express (Q)IITs as endofunctors.

But! We can take the approach proposed by [ACD+18]:

1. Start with a category of sorts.
2. Encode the first constructor using a pair of functors L (arguments) and R (target).
3. Augment the type of objects of the category to allow for something of type "$((x : L) \to R\, x)$".
4. Iterate this process for all the constructors.
5. The QIIT corresponds to the initial object in the final category.

QIITs example

The category of the sorts and two constructors below

```
data Con : Set
data Ty : Con → Set

data Con where
  ◇ : Con
  _,_ : (Γ : Con) → Ty Γ → Con
```

has objects of type

$$\sum_{C:\mathrm{Set}} \sum_{T:\, C\to\mathrm{Set}} \sum_{\diamond:C} (ext : \underbrace{\sum_{\Gamma:C} (T\,\Gamma)}_{L} \to \underbrace{C}_{R}).$$

Containerification

This approach still allows non-strictly positive types: *if* a QIIT specification has an initial algebra, then we can describe it in this way.

It doesn't tell us *which* QIIT specifications have initial algebras.

Containerification

This approach still allows non-strictly positive types: *if* a QIIT specification has an initial algebra, then we can describe it in this way.

It doesn't tell us *which* QIIT specifications have initial algebras.

Our proposal: **containerification**. Restrict L and R functors to be generalised container functors (+ other restrictions), thereby only allowing strictly positive QIIT definitions.

But *can any expression in type theory be encoded as some kind of container*? This is the motivation for the container model.

The container model

The container model as a restriction of the presheaf model

Presheaf model

Contexts	$\Gamma : \mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$
Substitutions	natural transformations
Types	$A : (\int \Gamma)^{\mathrm{op}} \to \mathbf{Set}$
Terms	$\int_{X:\lvert\mathbf{C}\rvert} (\gamma : \Gamma\, X) \to A(X, \gamma)$
Context extension	$\Gamma.A\, X = \sum_{\rho:\Gamma\, X} (A\, (X, \rho))$

The container model as a restriction of the presheaf model

	Presheaf model	Container model
Contexts	$\Gamma : \mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$	$[\![\Gamma]\!] : \mathbf{Set} \to \mathbf{Set}$
Substitutions	natural transformations	container morphisms
Types	$A : (\int \Gamma)^{\mathrm{op}} \to \mathbf{Set}$	$[\![A]\!] : (\int [\![\Gamma]\!]) \to \mathbf{Set}$
Terms	$\int_{X:\lvert\mathbf{C}\rvert} (\gamma : \Gamma\, X) \to A(X, \gamma)$	$\int_{X:\mathrm{Set}} (\gamma : [\![\Gamma]\!]\, X) \to [\![A]\!]\,(X, \gamma)$
Context extension	$\Gamma.A\, X = \sum_{\gamma:\Gamma\, X} (A\,(X, \gamma))$	$[\![\Gamma.A]\!]\, X = \sum_{\gamma:[\![\Gamma]\!]\, X} ([\![A]\!]\,(X, \gamma))$

The container model (as outlined in [AK21])

- The category of contexts and substitutions is the category of set containers **Cont**. A context Γ in **Cont** has corresponding functor

$$[\![S_\Gamma \triangleleft P_\Gamma]\!] : \mathbf{Set} \to \mathbf{Set}$$

The container model (as outlined in [AK21])

- The category of contexts and substitutions is the category of set containers **Cont**. A context Γ in **Cont** has corresponding functor

$$[\![S_\Gamma \triangleleft P_\Gamma]\!] : \mathbf{Set} \to \mathbf{Set}$$

- The empty context is $\top \triangleleft \bot$.

The container model (as outlined in [AK21])

- The category of contexts and substitutions is the category of set containers **Cont**. A context Γ in **Cont** has corresponding functor

$$[\![S_\Gamma \triangleleft P_\Gamma]\!] : \mathbf{Set} \to \mathbf{Set}$$

- The empty context is $\top \triangleleft \bot$.
- Types in context $\Gamma = S_\Gamma \triangleleft P_\Gamma$ are generalised containers over $\int [\![\Gamma]\!]$. A type A has corresponding functor

$$[\![S_A \triangleleft_{\int [\![\Gamma]\!]} P_A]\!] : \left(\int [\![\Gamma]\!]\right) \to \mathbf{Set}$$

The container model (as outlined in [AK21])

- The category of contexts and substitutions is the category of set containers **Cont**. A context Γ in **Cont** has corresponding functor

$$[\![S_\Gamma \triangleleft P_\Gamma]\!] : \mathbf{Set} \to \mathbf{Set}$$

- The empty context is $\top \triangleleft \bot$.
- Types in context $\Gamma = S_\Gamma \triangleleft P_\Gamma$ are generalised containers over $\int [\![\Gamma]\!]$. A type A has corresponding functor

$$[\![S_A \triangleleft_{\int [\![\Gamma]\!]} P_A]\!] : \left(\int [\![\Gamma]\!]\right) \to \mathbf{Set}$$

- Terms of type A in context Γ are dependent natural transformations from $[\![\Gamma]\!]$ to $[\![A]\!]$:

$$\int_{X:\mathrm{Set}} (\gamma : [\![\Gamma]\!]\, X) \to [\![A]\!]\, (X, \gamma)$$

The container model (as outlined in [AK21])

- The category of contexts and substitutions is the category of set containers **Cont**. A context Γ in **Cont** has corresponding functor

$$[\![S_\Gamma \triangleleft P_\Gamma]\!] : \mathbf{Set} \to \mathbf{Set}$$

- The empty context is $\top \triangleleft \bot$.
- Types in context $\Gamma = S_\Gamma \triangleleft P_\Gamma$ are generalised containers over $\int [\![\Gamma]\!]$. A type A has corresponding functor

$$[\![S_A \triangleleft_{\int [\![\Gamma]\!]} P_A]\!] : (\int [\![\Gamma]\!]) \to \mathbf{Set}$$

- Terms of type A in context Γ are dependent natural transformations from $[\![\Gamma]\!]$ to $[\![A]\!]$:

$$\int_{X:\mathrm{Set}} (\gamma : [\![\Gamma]\!]\, X) \to [\![A]\!]\, (X, \gamma)$$

- Context extension is given by $\Gamma.A = S_A \triangleleft P_A^X$

The container model (cont.)

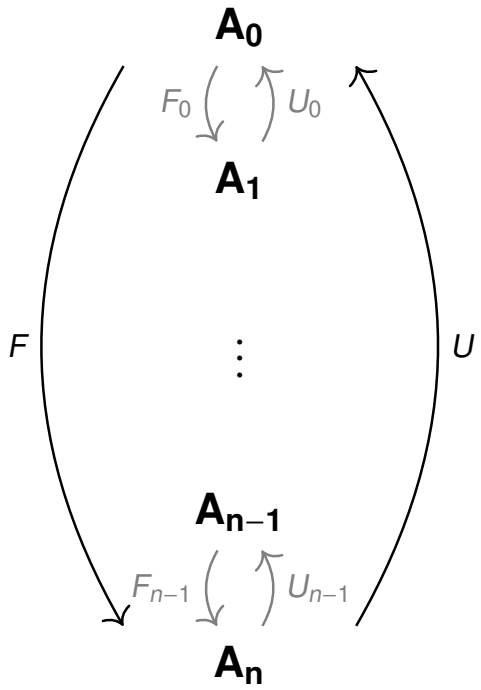
- We showed that the container model is a *groupoid* category with families (GCwF), where types are encoded as a presheaf from contexts into **h-groupoids** instead of **h-sets**. This introduces coherences to be checked.

The container model (cont.)

- We showed that the container model is a *groupoid* category with families (GCwF), where types are encoded as a presheaf from contexts into **h-groupoids** instead of **h-sets**. This introduces coherences to be checked.
- Type formers and the universe:
  - We can derive Σ-types in a similar way as for presheaves.
  - It looks like in general we do not have Π-types. (Von Glehn's polynomial model has Π-types and refutes function extensionality [vG15].)
  - It also looks like we don't have a universe $\mathcal{U}$.
  - Conjecture: We might get Π and $\mathcal{U}$ if we generalise our model. E.g. if contexts have type $\sum_{\mathbf{C}:Cat} \mathbf{GCont}_{\mathbf{C}}$.

Future research directions

- (Container model) Can we construct a more general container model with the desired type formers?
- (Containerification of QIIT semantics) We would like to have an existence result for the initial object in $\mathbf{A}_n$, provided that all the constructors added along the way were strictly positive, i.e. all the L_n's and R_n's were generalised container functors.



- (Container model) Can we construct a more general container model with the desired type formers?
- (Containerification of QIIT semantics) We would like to have an existence result for the initial object in $\mathbf{A}_n$, provided that all the constructors added along the way were strictly positive, i.e. all the L_n's and R_n's were generalised container functors.

Thank you!

More details in my thesis: `stefaniatadama.com`

Container GCwF formalisation:
`github.com/stefaniatadama/container-gcwf`

References I


Michael Abbott, Thorsten Altenkirch, and Neil Ghani.
Containers: Constructing strictly positive types.
Theoretical Computer Science, 342(1):3–27, 2005.
Applied Semantics: Selected Topics.


Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus, and Fredrik Nordvall Forsberg.
Quotient Inductive-Inductive Types, pages 293–310.
Springer International Publishing, 2018.


Thorsten Altenkirch and Ambrus Kaposi.
A container model of type theory.
In *TYPES 2021*, 2021.


Evan Cavallo and Robert Harper.
Higher inductive types in cubical computational type theory.
Proc. ACM Program. Lang., 3(POPL), January 2019.

References II


Thierry Coquand, Simon Huber, and Anders Mörtberg.
On higher inductive types in cubical type theory.
In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 255–264, New York, NY, USA, 2018. Association for Computing Machinery.


Stefania Damato, Thorsten Altenkirch, and Axel Ljungström.
Formalising Inductive and Coinductive Containers.
In Yannick Forster and Chantal Keller, editors, *16th International Conference on Interactive Theorem Proving (ITP 2025)*, volume 352 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17:1–17:20, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

References III


Ambrus Kaposi, András Kovács, and Thorsten Altenkirch.
Constructing quotient inductive-inductive types.
Proc. ACM Program. Lang., 3(POPL), 2019.


Peter LeFanu Lumsdaine and Michael Shulman.
Semantics of higher inductive types.
Mathematical Proceedings of the Cambridge Philosophical Society, 169(1):159–208, 6 2019.


Tamara von Glehn.
Polynomials and models of type theory.
PhD thesis, University of Cambridge, 2015.