The Groupoid CwF of Containers

Stefania Damato

j.w.w. Thorsten Altenkirch

University of Nottingham, UK

HoTTEST

6th November 2025

Overview

CwFs in Intensional Type Theory

What are CwFs?

A type theory is a formal system in which we can derive certain kinds of judgments.

A model constitutes a **sound semantics** for a type theory.

What are CwFs?

A type theory is a formal system in which we can derive certain kinds of judgments.

A model constitutes a **sound semantics** for a type theory.

Categories with families (CwFs) are one way to model dependent type theory.

If we write down the intrinsic syntax of dependent type theory as a quotient inductive-inductive type (QIIT), algebras of this signature correspond to CwFs.

Categories with families

A **category with families (CwF)** [Dybjer, 1996] consists of:

- A category **C** of contexts $\Gamma, \Delta, \ldots$ and substitutions $\Delta \xrightarrow{\gamma} \Gamma, \ldots$.

Categories with families

A **category with families (CwF)** [Dybjer, 1996] consists of:

- A category $\mathbf{C}$ of contexts $\Gamma, \Delta, \ldots$ and substitutions $\Delta \xrightarrow{\gamma} \Gamma, \ldots$.
- A presheaf

$$\mathrm{Ty} \colon \mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$$

  of types $A : \mathrm{Ty}\ \Gamma, \ldots$ and type substitutions $A[\gamma] : \mathrm{Ty}\ \Delta, \ldots$.

Categories with families

A **category with families (CwF)** [Dybjer, 1996] consists of:

- A category $\mathbf{C}$ of contexts $\Gamma, \Delta, \ldots$ and substitutions $\Delta \xrightarrow{\gamma} \Gamma, \ldots$.
- A presheaf
$$\mathrm{Ty} \colon \mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$$
of types $A : \mathrm{Ty}\ \Gamma, \ldots$ and type substitutions $A[\gamma] : \mathrm{Ty}\ \Delta, \ldots$.
- A presheaf
$$\mathrm{Tm} \colon (\textstyle\int \mathrm{Ty})^{\mathrm{op}} \to \mathbf{Set}$$
of terms $a : \mathrm{Tm}\,(\Gamma, A), \ldots$ and term substitutions $a[\gamma] : \mathrm{Tm}\,(\Delta, A[\gamma]), \ldots$.

Categories with families

A **category with families (CwF)** [Dybjer, 1996] consists of:

- A category **C** of contexts $\Gamma, \Delta, \ldots$ and substitutions $\Delta \xrightarrow{\gamma} \Gamma, \ldots$.
- A presheaf
$$\mathrm{Ty} \colon \mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$$
of types $A : \mathrm{Ty}\ \Gamma, \ldots$ and type substitutions $A[\gamma] : \mathrm{Ty}\ \Delta, \ldots$.
- A presheaf
$$\mathrm{Tm} \colon (\textstyle\int \mathrm{Ty})^{\mathrm{op}} \to \mathbf{Set}$$
of terms $a : \mathrm{Tm}\,(\Gamma, A), \ldots$ and term substitutions $a[\gamma] : \mathrm{Tm}\,(\Delta, A[\gamma]), \ldots$.
- A context extension operation $\Gamma.A$ for $\Gamma : |\mathbf{C}|$ and $A : \mathrm{Ty}\ \Gamma$ such that
$$\Delta \to \Gamma.A \cong \sum_{\gamma \colon \Delta \to \Gamma} \mathrm{Tm}\,(\Delta, A[\gamma]).$$

Categories with families

A **category with families (CwF)** [Dybjer, 1996] consists of:

- A category $\mathbf{C}$ of contexts $\Gamma, \Delta, \ldots$ and substitutions $\Delta \xrightarrow{\gamma} \Gamma, \ldots$.
- A presheaf
$$\mathrm{Ty} \colon \mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$$
of types $A : \mathrm{Ty}\ \Gamma, \ldots$ and type substitutions $A[\gamma] : \mathrm{Ty}\ \Delta, \ldots$.
- A presheaf
$$\mathrm{Tm} \colon (\textstyle\int \mathrm{Ty})^{\mathrm{op}} \to \mathbf{Set}$$
of terms $a : \mathrm{Tm}\,(\Gamma, A), \ldots$ and term substitutions $a[\gamma] : \mathrm{Tm}\,(\Delta, A[\gamma]), \ldots$
- A context extension operation $\Gamma.A$ for $\Gamma : |\mathbf{C}|$ and $A : \mathrm{Ty}\ \Gamma$ such that
$$\Delta \to \Gamma.A \cong \sum_{\gamma \colon \Delta \to \Gamma} \mathrm{Tm}\,(\Delta, A[\gamma]).$$

Coherence issues in ITT

Example

In the standard model/set model,

$$\mathsf{Ty}\,(\Gamma : \mathsf{Set}) :\equiv \Gamma \to \mathsf{Set}.$$

Coherence issues in ITT

Example

In the standard model/set model,

$$\mathrm{Ty}\,(\Gamma : \mathrm{Set}) :\equiv \Gamma \to \mathrm{Set}.$$

Example

In the presheaf model over a category **C**,

$$\mathrm{Ty}\,(\Gamma : \mathbf{C}^{\mathrm{op}} \to \mathbf{Set}) :\equiv \prod_{X : |\mathbf{C}|} \Gamma\, X \to \mathrm{Set}.$$

Coherence issues in ITT

Example

In the standard model/set model,

$$\mathrm{Ty}\,(\Gamma : \mathrm{Set}) :\equiv \Gamma \to \mathrm{Set}.$$

Example

In the presheaf model over a category $\mathbf{C}$,

$$\mathrm{Ty}\,(\Gamma : \mathbf{C}^{\mathrm{op}} \to \mathbf{Set}) :\equiv \prod_{X : |\mathbf{C}|} \Gamma\, X \to \mathrm{Set}.$$

When working in intensional type theory (ITT) i.e. no UIP, in both cases, $\mathrm{Ty}\,\Gamma$ forms a **groupoid** not a **set**.

How do we solve this?

Force models to fit original CwF definition

↓

Strictification

Define a ‘higher’ CwF e.g. ∞-CwF, groupoid CwF

↓

Prove extra coherence laws

How do we solve this?

Force models to fit original CwF definition

Strictification

CwF of strictified containers

Define a ‘higher’ CwF
e.g. ∞-CwF, groupoid CwF

Prove extra coherence laws

Groupoid CwF of containers

we do both

Related work on higher CwFs

- [Kraus, 2021]: Develops the notion of an ∞-CwF, and discusses the ‘coherence problem’: **is the initial model/syntax of an ∞-CwF set-truncated?**

Related work on higher CwFs

- [Kraus, 2021]: Develops the notion of an ∞-CwF, and discusses the ‘coherence problem’: **is the initial model/syntax of an ∞-CwF set-truncated?**
- [Uemura, 2022]: Proves that the initial ∞-natural model is set-truncated (caveat: not in type theory).

Related work on higher CwFs

- [Kraus, 2021]: Develops the notion of an ∞-CwF, and discusses the ‘coherence problem’: **is the initial model/syntax of an ∞-CwF set-truncated?**
- [Uemura, 2022]: Proves that the initial ∞-natural model is set-truncated (caveat: not in type theory).
- [Altenkirch et al., 2025] Define groupoid CwFs (a.k.a. 2-CwFs), and show that **the initial model is set-truncated**.

Related work on higher CwFs

- [Kraus, 2021]: Develops the notion of an ∞-CwF, and discusses the ‘coherence problem’: **is the initial model/syntax of an ∞-CwF set-truncated?**
- [Uemura, 2022]: Proves that the initial ∞-natural model is set-truncated (caveat: not in type theory).
- [Altenkirch et al., 2025] Define groupoid CwFs (a.k.a. 2-CwFs), and show that **the initial model is set-truncated**.
- [Chen, 2025] Studies wild CwFs, where context substitutions need not form a set. (Groupoid CwFs are simple examples of 2-coherent wild CwFs.)

Related work on higher CwFs

- [Kraus, 2021]: Develops the notion of an ∞-CwF, and discusses the 'coherence problem': **is the initial model/syntax of an ∞-CwF set-truncated?**
- [Uemura, 2022]: Proves that the initial ∞-natural model is set-truncated (caveat: not in type theory).
- [Altenkirch et al., 2025] Define groupoid CwFs (a.k.a. 2-CwFs), and show that **the initial model is set-truncated**
- [Chen, 2025] Studies wild CwFs, where context substitutions need not form a set. (Groupoid CwFs are simple examples of 2-coherent wild CwFs.)

  We focus on groupoid CwFs.

Groupoid CwFs (GCwFs)

In a groupoid CwF (GCwF),

$$\mathrm{Ty} \colon \mathbf{C}^{\mathrm{op}} \to \mathbf{Gpd}$$

is now a pseudofunctor from a 1-category $\mathbf{C}^{\mathrm{op}}$ to the bicategory **Gpd** (see [Ahrens et al., 2019]).

Groupoid CwFs (GCwFs)

In a groupoid CwF (GCwF),

$$\mathrm{Ty} \colon \mathbf{C}^{\mathrm{op}} \to \mathbf{Gpd}$$

is now a pseudofunctor from a 1-category $\mathbf{C}^{\mathrm{op}}$ to the bicategory **Gpd** (see [Ahrens et al., 2019]).

Additional coherence laws on types need to checked.

$$\begin{tikzcd}
 & & A[\sigma][\delta][\gamma] \arrow[dll, Rightarrow, "{[\circ]\,\mathrm{Ty}}"'] \arrow[drr, Rightarrow, "{\mathrm{ap}_{-[\delta]}[\circ]\,\mathrm{Ty}}"] & & \\
A[\sigma][\delta \circ \gamma] \arrow[dr, Rightarrow, "{[\circ]\,\mathrm{Ty}}"'] & & & & A[\sigma \circ \delta][\gamma] \arrow[dl, Rightarrow, "{[\circ]\,\mathrm{Ty}}"] \\
 & A[\sigma \circ (\delta \circ \gamma)] \arrow[rr, Rightarrow, "{\mathrm{ap}_{A[-]}\,\mathrm{assoc}_{\mathbf{C}}^{-1}}"'] & & A[(\sigma \circ \delta) \circ \gamma] &
\end{tikzcd}$$

$$\begin{tikzcd}
 & A[\mathrm{id}_\Gamma][\gamma] \arrow[dl, Rightarrow, "{[\circ]\,\mathrm{Ty}}"'] \arrow[dr, Rightarrow, "{\mathrm{ap}_{-[\gamma]}[\mathrm{id}]\,\mathrm{Ty}}"] & \\
A[\mathrm{id}_\Gamma \circ \gamma] \arrow[rr, Rightarrow, "{\mathrm{ap}_{A[-]}\,\mathrm{idl}_{\mathbf{C}}}"'] & & A[\gamma]
\end{tikzcd}$$

$$\begin{tikzcd}
 & A[\gamma][\mathrm{id}_\Gamma] \arrow[dl, Rightarrow, "{[\circ]\,\mathrm{Ty}}"'] \arrow[dr, Rightarrow, "{[\mathrm{id}]\,\mathrm{Ty}}"] & \\
A[\gamma \circ \mathrm{id}_\Gamma] \arrow[rr, Rightarrow, "{\mathrm{ap}_{A[-]}\,\mathrm{idr}_{\mathbf{C}}}"'] & & A[\gamma]
\end{tikzcd}$$

A Groupoid CwF of Containers

Containers (a.k.a. polynomial functors)

Definition

A **(set)-container** is a pair $S : \mathrm{Set}, P : S \to \mathrm{Set}$ written $S \triangleleft P$.
Every container has a functor representation

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

Containers (a.k.a. polynomial functors)

Definition

A **(set)-container** is a pair S : Set, $P\colon S \to$ Set written $S \triangleleft P$. Every container has a functor representation

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

Definition

A **generalised container** over a category $\mathbf{C}$ is a pair S : Set, $P\colon S \to |\mathbf{C}|$, written $S \triangleleft^G P$. Every generalised container has a functor representation

$$[\![S \triangleleft^G P]\!]^G : \mathbf{C} \to \mathbf{Set}.$$

Containers (a.k.a. polynomial functors)

Definition

A **(set)-container** is a pair $S : \mathrm{Set}, P : S \to \mathrm{Set}$ written $S \triangleleft P$. Every container has a functor representation

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

Definition

A **generalised container** over a category $\mathbf{C}$ is a pair $S : \mathrm{Set}$, $P : S \to |\mathbf{C}|$, written $S \triangleleft^G P$. Every generalised container has a functor representation

$$[\![S \triangleleft^G P]\!]^G : \mathbf{C} \to \mathbf{Set}.$$

We are interested in a container model of type theory for reasons to do with semantics of inductive types.

The container model (as outlined in [Altenkirch and Kaposi, 2021])

- The category of contexts and substitutions is the category of set-containers $S_\Gamma : \mathrm{Set} \triangleleft P_\Gamma : S_\Gamma \to \mathrm{Set}$ and their morphisms. Set-containers have functors

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

The container model (as outlined in [Altenkirch and Kaposi, 2021])

- The category of contexts and substitutions is the category of set-containers $S_\Gamma : \mathrm{Set} \triangleleft P_\Gamma : S_\Gamma \to \mathrm{Set}$ and their morphisms. Set-containers have functors

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

- Types in context $\Gamma = S_\Gamma \triangleleft P_\Gamma$ are generalised containers over $\int[\![\Gamma]\!]$, $S_A : \mathrm{Set} \triangleleft^G P_A : S_A \to |\int[\![\Gamma]\!]|$, having functors

$$[\![S_A \triangleleft^G P_A]\!]^G : (\textstyle\int[\![\Gamma]\!]) \to \mathbf{Set}$$

The container model (as outlined in [Altenkirch and Kaposi, 2021])

- The category of contexts and substitutions is the category of set-containers $S_\Gamma : \mathrm{Set} \triangleleft P_\Gamma : S_\Gamma \to \mathrm{Set}$ and their morphisms. Set-containers have functors

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

- Types in context $\Gamma = S_\Gamma \triangleleft P_\Gamma$ are generalised containers over $\int[\![\Gamma]\!]$, $S_A : \mathrm{Set} \triangleleft^G P_A : S_A \to |\int[\![\Gamma]\!]|$, having functors

$$[\![S_A \triangleleft^G P_A]\!]^G : (\textstyle\int[\![\Gamma]\!]) \to \mathbf{Set}$$

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

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

The container model (as outlined in [Altenkirch and Kaposi, 2021])

- The category of contexts and substitutions is the category of set-containers $S_\Gamma : \mathrm{Set} \triangleleft P_\Gamma : S_\Gamma \to \mathrm{Set}$ and their morphisms. Set-containers have functors

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

- Types in context $\Gamma = S_\Gamma \triangleleft P_\Gamma$ are generalised containers over $\int [\![\Gamma]\!]$, $S_A : \mathrm{Set} \triangleleft^G P_A : S_A \to |\int [\![\Gamma]\!]|$, having functors

$$[\![S_A \triangleleft^G P_A]\!]^G : (\textstyle\int [\![\Gamma]\!]) \to \mathbf{Set}$$

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

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

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

Presheaf model vs. Container model

	Presheaf model	Container model
Contexts	$\mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$	$\mathbf{Set} \to \mathbf{Set}$

Presheaf model vs. Container model

	Presheaf model	Container model
Contexts	$\mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$	$\mathbf{Set} \to \mathbf{Set}$
Substitutions	natural transformations	container morphisms

Presheaf model vs. Container model

	Presheaf model	Container model
Contexts	$\mathbf{C}^{\mathrm{op}} \to \mathbf{Set}$	$\mathbf{Set} \to \mathbf{Set}$
Substitutions	natural transformations	container morphisms
Types	$(\int \Gamma)^{\mathrm{op}} \to \mathbf{Set}$	$(\int [\![\Gamma]\!]) \to \mathbf{Set}$

Presheaf model vs. Container model

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

Presheaf model vs. Container model

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

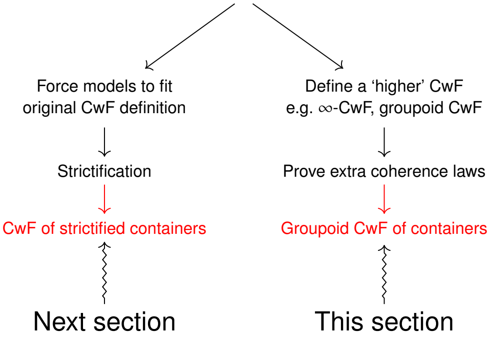
The container model — coherence issues

The container model of [Altenkirch and Kaposi, 2021] suffers from the same coherence issues as the set and presheaf models: $\mathsf{Ty}\ \Gamma$ is a groupoid, not a set.

The container model — coherence issues

The container model of [Altenkirch and Kaposi, 2021] suffers from the same coherence issues as the set and presheaf models: $\mathsf{Ty}\ \Gamma$ is a groupoid, not a set.

Recall . . .



The container model — contexts & types

Contexts If Γ is a context, then $\Gamma = S_\Gamma \triangleleft P_\Gamma$ is a **set-container**.

A substitution $\Delta \xrightarrow{\gamma} \Gamma$ is a container morphism:

$$\gamma_s : S_\Delta \to S_\Gamma$$
$$\gamma_p : \prod_{s_\Delta : S_\Delta} P_\Gamma \, (\gamma_s \, s_\Delta) \to P_\Delta \, s_\Delta$$

The container model — contexts & types

Contexts If Γ is a context, then $\Gamma = S_\Gamma \triangleleft P_\Gamma$ is a **set-container**.
A substitution $\Delta \xrightarrow{\gamma} \Gamma$ is a container morphism:

$$\gamma_s : S_\Delta \to S_\Gamma$$
$$\gamma_p : \prod_{s_\Delta : S_\Delta} P_\Gamma\,(\gamma_s\, s_\Delta) \to P_\Delta\, s_\Delta$$

Types If A : Ty Γ, then $A = S_A \triangleleft^G P_A$ is a **generalised container**, with S_A : Set, $P_A : S_A \to |\int [\![\Gamma]\!]|$, where we can break apart P_A into 3 components.

$$S_A : \mathrm{Set}$$
$$P_A^X : S_A \to \mathrm{Set}$$
$$P_A^s : S_A \to S_\Gamma$$
$$P_A^f : \prod_{s : S_A} P_\Gamma\,(P_A^s\, s) \to P_A^X\, s$$

The container model — type substitution

If $\Delta \xrightarrow{\gamma} \Gamma$ is a container morphism, then:

$$\begin{aligned} \gamma_s &: S_\Delta \to S_\Gamma \\ \gamma_p &: \prod_{s_\Delta : S_\Delta} P_\Gamma\,(\gamma_s\, s_\Delta) \to P_\Delta\, s_\Delta \end{aligned}$$

If A : Ty Γ, then:

$$\begin{aligned} S_A &: \mathrm{Set} \\ P_A^X &: S_A \to \mathrm{Set} \\ P_A^s &: S_A \to S_\Gamma \\ P_A^f &: \prod_{s : S_A} P_\Gamma\,(P_A^s\, s) \to P_A^X\, s \end{aligned}$$

Given a container morphism $\Delta \xrightarrow{\gamma} \Gamma$, we (roughly) define $A[\gamma]$ as:

$$\begin{tikzcd} S_{A[\gamma]} \arrow[r, "s_A"] \arrow[d, "s_\Delta"'] \arrow[dr, phantom, "\lrcorner", very near start] & S_A \arrow[d, "P_A^s"] & \triangleleft^G & P_\Gamma\,(\gamma_s\, s_\Delta) \arrow[r, "(P_A^f\, s_A)"] \arrow[d, "\gamma_p\, s_\Delta"'] & P_A^X\, s_A \arrow[d, "\mathrm{inr}"] \\ S_\Delta \arrow[r, "\gamma_s"'] & S_\Gamma & & P_\Delta\, s_\Delta \arrow[r, "\mathrm{inl}"'] \arrow[ur, phantom, "\ulcorner", very near end] & P_{A[\gamma]}^X\, s_{A[\gamma]} \end{tikzcd}$$

A trick for pullback in type theory

$$\begin{tikzcd} A \arrow[d, "f"] \\ B \end{tikzcd}$$

can be written as

$$F : B \to \mathcal{U}$$
$$F\,b = \sum_{a:A} f\,a \equiv b.$$

A trick for pullback in type theory

$$\begin{tikzcd} A \arrow[d, "f"] \\ B \end{tikzcd}$$ can be written as $$\begin{aligned} &F : B \to \mathcal{U} \\ &F\, b = \sum_{a:A} f\, a \equiv b. \end{aligned}$$

Then $$\begin{tikzcd} PB \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "f"] \\ C \arrow[r, "g"'] & B \end{tikzcd}$$ can be written as $$\begin{aligned} &H : C \to \mathcal{U} \\ &H\, c = F\,(g\, c) \\ &\quad = \sum_{a:A} f\, a \equiv g\, c. \end{aligned}$$

A trick for pullback in type theory

$$\begin{array}{c} A \\ \big\downarrow f \\ B \end{array}$$ can be written as $$\begin{aligned} &F : B \to \mathcal{U} \\ &F\, b = \sum_{a:A} f\, a \equiv b. \end{aligned}$$

Then
$$\begin{tikzcd} PB \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "f"] \\ C \arrow[r, "g"'] & B \end{tikzcd}$$
can be written as
$$\begin{aligned} H &: C \to \mathcal{U} \\ H\, c &= F\,(g\, c) \\ &= \sum_{a:A} f\, a \equiv g\, c. \end{aligned}$$

So we represent pullbacks as families.

We don't have a trick for pushouts

$$\begin{tikzcd} X \arrow[r, "g"] \arrow[d, "f"'] & Y \arrow[d] \\ Z \arrow[r] \arrow[ur, phantom, "\ulcorner", very near end] & PO \end{tikzcd}$$

is written as $\|\text{Pushout}\, f\, g\|_0$, where

$$\begin{aligned} \mathsf{inl} &: Z \to \text{Pushout}\, f\, g \\ \mathsf{inr} &: Y \to \text{Pushout}\, f\, g \\ \mathsf{push} &: \prod_{x:X} \mathsf{inl}\,(f\,x) \equiv \mathsf{inr}\,(g\,x) \end{aligned}$$

and

$$\begin{aligned} |\_|_0 &: A \to \|A\|_0 \\ \mathsf{squash}_0 &: \prod_{x,y:\|A\|_0} \prod_{p,q:x\equiv y} p \equiv q. \end{aligned}$$

The container model — coherences

Triangulators (the identity coherence laws)

$$\begin{tikzcd} & A[\mathsf{id}_\Gamma][\gamma] \arrow[dl, Rightarrow, "{[\circ]\,\mathsf{Ty}}"'] \arrow[dr, Rightarrow, "{\mathsf{ap}_{-[\gamma]}[\mathsf{id}]\,\mathsf{Ty}}"] & \\ A[\mathsf{id}_\Gamma \circ \gamma] \arrow[rr, Rightarrow, "{\mathsf{ap}_{A[-]}\,\mathsf{idl}_{\mathbf{C}}}"'] & & A[\gamma] \end{tikzcd} \qquad \begin{tikzcd} & A[\gamma][\mathsf{id}_\Gamma] \arrow[dl, Rightarrow, "{[\circ]\,\mathsf{Ty}}"'] \arrow[dr, Rightarrow, "{[\mathsf{id}]\,\mathsf{Ty}}"] & \\ A[\gamma \circ \mathsf{id}_\Gamma] \arrow[rr, Rightarrow, "{\mathsf{ap}_{A[-]}\,\mathsf{idr}_{\mathbf{C}}}"'] & & A[\gamma] \end{tikzcd}$$

where

$$[\mathsf{id}]\,\mathsf{Ty} : A[\mathsf{id}] \equiv A$$
$$[\circ]\,\mathsf{Ty} : A[\theta][\delta] \equiv A[\theta \circ \delta].$$

The container model — coherences

Triangulators (the identity coherence laws)

$$\begin{tikzcd}
 & A[\mathrm{id}_\Gamma][\gamma] \arrow[dl, Rightarrow, "{[\circ]\,\mathrm{Ty}}"'] \arrow[dr, Rightarrow, "{\mathrm{ap}_{-[\gamma]}\,[\mathrm{id}]\,\mathrm{Ty}}"] & \\
A[\mathrm{id}_\Gamma \circ \gamma] \arrow[rr, Rightarrow, "{\mathrm{ap}_{A[-]}\,\mathrm{idl}_{\mathbf{C}}}"'] & & A[\gamma]
\end{tikzcd}
\qquad
\begin{tikzcd}
 & A[\gamma][\mathrm{id}_\Gamma] \arrow[dl, Rightarrow, "{[\circ]\,\mathrm{Ty}}"'] \arrow[dr, Rightarrow, "{[\mathrm{id}]\,\mathrm{Ty}}"] & \\
A[\gamma \circ \mathrm{id}_\Gamma] \arrow[rr, Rightarrow, "{\mathrm{ap}_{A[-]}\,\mathrm{idr}_{\mathbf{C}}}"'] & & A[\gamma]
\end{tikzcd}$$

where

$$[\mathrm{id}]\,\mathrm{Ty} : A[\mathrm{id}] \equiv A$$
$$[\circ]\,\mathrm{Ty} : A[\theta][\delta] \equiv A[\theta \circ \delta].$$

I will talk about the left coherence law.

Left identity coherence law, in `Cubical Agda`

To prove:

$$
\begin{tikzcd}
 & A[\mathsf{id}_\Gamma][\gamma] \arrow[dl, Rightarrow, "{[\circ]\ \mathsf{Ty}}"'] \arrow[dr, Rightarrow, "{\mathsf{ap}_{-[\gamma]}[\mathsf{id}]\ \mathsf{Ty}}"] & \\
A[\mathsf{id}_\Gamma \circ \gamma] \arrow[rr, Rightarrow, "{\mathsf{ap}_{A[-]}\ \mathsf{idl}_{\mathbf{C}}}"'] & & A[\gamma]
\end{tikzcd}
$$

Left identity coherence law, in `Cubical Agda`

To prove:

$$\begin{tikzcd} & A[\mathrm{id}_\Gamma][\gamma] \arrow[dl, Rightarrow, "{[\circ]\ \mathsf{Ty}}"'] \arrow[dr, Rightarrow, "{\mathsf{ap}_{-[\gamma]}\,[\mathrm{id}]\ \mathsf{Ty}}"] & \\ A[\mathrm{id}_\Gamma \circ \gamma] \arrow[rr, Rightarrow, "{\mathsf{ap}_{A[-]}\ \mathsf{idl}_{\mathbf{C}}}"'] & & A[\gamma] \end{tikzcd}$$

Proof sketch: Thanks to Axel's help!
We write the above as a square of generalised containers

$$\begin{tikzcd} A[\mathrm{id}_\Gamma \circ \gamma] \arrow[r, Rightarrow, "{\mathsf{ap}_{A[-]}\ \mathsf{idl}_{\mathbf{C}}}"] & A[\gamma] \\ A[\mathrm{id}_\Gamma][\gamma] \arrow[u, Rightarrow, "{[\circ]\ \mathsf{Ty}}"] \arrow[r, Rightarrow, "{\mathsf{ap}_{-[\gamma]}\,[\mathrm{id}]\ \mathsf{Ty}}"'] & A[\gamma] \arrow[u, Rightarrow, "\mathsf{refl}"'] \end{tikzcd}$$

Left identity coherence law, in `Cubical Agda`

$$
\begin{tikzcd}
A[\mathrm{id}_\Gamma \circ \gamma] \arrow[r, Rightarrow, "\mathrm{ap}_{A[-]}\ \mathrm{idl}_{\mathbf{C}}"] & A[\gamma] \\
A[\mathrm{id}_\Gamma][\gamma] \arrow[u, Rightarrow, "{[\circ]\ \mathrm{Ty}}"] \arrow[r, Rightarrow, "\mathrm{ap}_{-[\gamma]}\,[\mathrm{id}]\ \mathrm{Ty}"'] & A[\gamma] \arrow[u, Rightarrow, "\mathrm{refl}"']
\end{tikzcd}
$$

which we can rewrite as

$$
\begin{tikzcd}
A[\gamma] \arrow[r, Rightarrow, "\mathrm{uaGenCon}\ \ \mathrm{id}_{\simeq\mathrm{GenCon}}"] & A[\gamma] \\
A[\mathrm{id}_\Gamma][\gamma] \arrow[u, Rightarrow, "\mathrm{uaGenCon}\ (...[\circ]\ \mathrm{Ty}\text{-eq})"] \arrow[r, Rightarrow, "\mathrm{uaGenCon}\ (...[\mathrm{id}]\ \mathrm{Ty}\text{-eq})"'] & A[\gamma] \arrow[u, Rightarrow, "\mathrm{uaGenCon}\ \ \mathrm{id}_{\simeq\mathrm{GenCon}}"']
\end{tikzcd}
$$

where uaGenCon : $A \simeq_{\mathrm{GenCon}} B \to A \equiv B$.

Left identity coherence law, in `Cubical Agda`

$$\begin{tikzcd}
A[\gamma] \arrow[rr, Rightarrow, "\mathsf{uaGenCon}\ \mathrm{id}_{\simeq\mathrm{GenCon}}"] & & A[\gamma] \\
A[\mathrm{id}_\Gamma][\gamma] \arrow[u, Rightarrow, "\mathsf{uaGenCon}\ (...[\circ]\ \mathrm{Ty}\text{-eq})"] \arrow[rr, Rightarrow, "\mathsf{uaGenCon}\ (...[\mathrm{id}]\ \mathrm{Ty}\text{-eq})"'] & & A[\gamma] \arrow[u, Rightarrow, "\mathsf{uaGenCon}\ \mathrm{id}_{\simeq\mathrm{GenCon}}"']
\end{tikzcd}$$

where uaGenCon : $A \simeq_{\mathrm{GenCon}} B \to A \equiv B$.

Left identity coherence law, in `Cubical Agda`

$$
\begin{tikzcd}
A[\gamma] \arrow[rr, Rightarrow, "\mathsf{uaGenCon}\ \mathsf{id}_{\simeq\mathsf{GenCon}}"] & & A[\gamma] \\
A[\mathsf{id}_\Gamma][\gamma] \arrow[u, Rightarrow, "\mathsf{uaGenCon}\ (\ldots[\circ]\ \mathsf{Ty}\text{-}\mathsf{eq})"] \arrow[rr, Rightarrow, "\mathsf{uaGenCon}\ (\ldots[\mathsf{id}]\ \mathsf{Ty}\text{-}\mathsf{eq})"'] & & A[\gamma] \arrow[u, Rightarrow, "\mathsf{uaGenCon}\ \mathsf{id}_{\simeq\mathsf{GenCon}}"']
\end{tikzcd}
$$

where uaGenCon : $A \simeq_{\mathsf{GenCon}} B \to A \equiv B$.

We show that this square commutes by giving equalities for each of the generalised container components.

Left identity coherence law, in `Cubical Agda`

$$\begin{tikzcd}
A[\gamma] \arrow[r, Rightarrow, "\text{uaGenCon } \mathrm{id}_{\simeq\text{GenCon}}"] & A[\gamma] \\
A[\mathrm{id}_\Gamma][\gamma] \arrow[u, Rightarrow, "\text{uaGenCon } (\ldots[\circ]\ \mathrm{Ty}\text{-eq})"] \arrow[r, Rightarrow, "\text{uaGenCon } (\ldots[\mathrm{id}]\ \mathrm{Ty}\text{-eq})"'] & A[\gamma] \arrow[u, Rightarrow, "\text{uaGenCon } \mathrm{id}_{\simeq\text{GenCon}}"']
\end{tikzcd}$$

where uaGenCon : $A \simeq_{\text{GenCon}} B \to A \equiv B$.

We show that this square commutes by giving equalities for each of the generalised container components.

Finally, to get the original square, we massage our equalities to match those we need via some lemmas.
E.g. one lemma is uaGenCon $\mathrm{id}_{\simeq\text{GenCon}} \equiv \text{refl}$. □

Progress so far

- Formalised left and right identity coherence laws in `Cubical Agda` i.e. the **triangulators**

Progress so far

- Formalised left and right identity coherence laws in `Cubical Agda` i.e. the **triangulators**
- To do: associativity coherence law i.e. the **pentagonator**

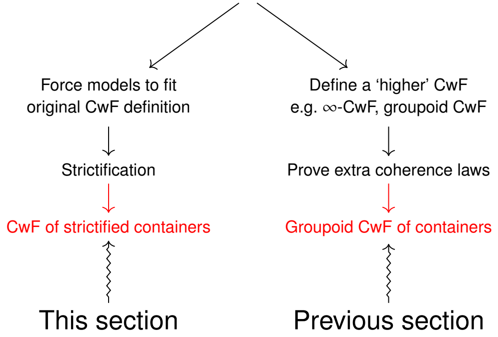
$$
\begin{tikzcd}
 & & A[\sigma][\delta][\gamma] \arrow[dll, Rightarrow, "{[\circ]\ \mathsf{Ty}}"'] \arrow[drr, Rightarrow, "{\mathsf{ap}_{-[\delta]}\,[\circ]\ \mathsf{Ty}}"] & & \\
A[\sigma][\delta \circ \gamma] \arrow[dr, Rightarrow, "{[\circ]\ \mathsf{Ty}}"'] & & & & A[\sigma \circ \delta][\gamma] \arrow[dl, Rightarrow, "{[\circ]\ \mathsf{Ty}}"] \\
 & A[\sigma \circ (\delta \circ \gamma)] \arrow[rr, Rightarrow, "{\mathsf{ap}_{A[-]}\ \mathsf{assoc}_{\mathbf{C}}^{-1}}"'] & & A[(\sigma \circ \delta) \circ \gamma] &
\end{tikzcd}
$$

A CwF of
Strictified Containers

Another way to solve the coherence issues

This section illustrates another way to deal with the coherence issues in the model given by [Altenkirch and Kaposi, 2021].

Recall . . .



The strictified container model

We use an inductive-recursive universe $\mathsf{U} : \mathsf{Set}, \mathsf{El} : \mathsf{U} \to \mathsf{Set}$.

The strictified container model

We use an inductive-recursive universe $\mathsf{U} : \mathsf{Set}, \mathsf{El} : \mathsf{U} \to \mathsf{Set}$.

- The category of <u>contexts</u> and <u>substitutions</u> is the category of **codes** for set-containers and codes for their morphisms:

$$\hat{\Gamma} = \hat{S}_\Gamma : \mathsf{U} \triangleleft \hat{P}_\Gamma : \mathsf{El}\ \hat{S}_\Gamma \to \mathsf{U}$$

such that $\Gamma = S_\Gamma \triangleleft P_\Gamma = \mathsf{El}\ \hat{S}_\Gamma \triangleleft (\mathsf{El} \circ \hat{P}_\Gamma)$.

The strictified container model

We use an inductive-recursive universe $\mathrm{U} : \mathrm{Set}, \mathrm{El} : \mathrm{U} \to \mathrm{Set}$.

- The category of <u>contexts</u> and <u>substitutions</u> is the category of **codes** for set-containers and codes for their morphisms:

$$\hat{\Gamma} = \hat{S}_\Gamma : \mathrm{U} \lhd \hat{P}_\Gamma : \mathrm{El}\ \hat{S}_\Gamma \to \mathrm{U}$$

  such that $\Gamma = S_\Gamma \lhd P_\Gamma = \mathrm{El}\ \hat{S}_\Gamma \lhd (\mathrm{El} \circ \hat{P}_\Gamma)$.

- <u>Types</u> in context $\hat{\Gamma} = \hat{S}_\Gamma \lhd \hat{P}_\Gamma$ are codes for generalised containers over $\int[\![\Delta]\!]$ for some context $\hat{\Delta}$, together with a substitution into $\hat{\Delta}$ — **we delay substitution**.

$$\begin{aligned} \hat{A} = (&\hat{\Delta} : |\mathbf{Con}|, \\ &\hat{\Gamma} \xrightarrow{\delta} \hat{\Delta}, \\ &\hat{S}_B : U \lhd \hat{P}_B : El\ \hat{S}_B \to |\textstyle\int[\![\Delta]\!]|^U) \end{aligned}$$

  Idea: $\hat{A}$ represents $\hat{B}[\delta]$.

The strictified container model

$$\hat{A} = (\hat{\Delta} : |\mathbf{Con}|, \\ \hat{\Gamma} \xrightarrow{\delta} \hat{\Delta}, \\ \hat{S}_B \lhd \hat{P}_B)$$

The strictified container model

$$\begin{aligned}\hat{A} = (\hat{\Delta} &: |\mathbf{Con}|,\\ \hat{\Gamma} &\xrightarrow{\delta} \hat{\Delta},\\ \hat{S}_B &\lhd \hat{P}_B)\end{aligned}$$

- Type substitution for $\gamma \colon \hat{\Theta} \to \hat{\Gamma}$, $\hat{A}[\gamma]$ can now be defined as

$$\hat{A}[\gamma] \coloneqq (\hat{\Theta}, \hat{\Theta} \xrightarrow{\delta \circ \gamma} \hat{\Delta}, \hat{S_B} \lhd \hat{P_B}).$$

The strictified container model

$$\hat{A} = (\hat{\Delta} : |\mathbf{Con}|,$$
$$\hat{\Gamma} \xrightarrow{\delta} \hat{\Delta},$$
$$\hat{S}_B \lhd \hat{P}_B)$$

- Type substitution for $\gamma\colon \hat{\Theta} \to \hat{\Gamma}$, $\hat{A}[\gamma]$ can now be defined as

$$\hat{A}[\gamma] \coloneqq (\hat{\Theta}, \hat{\Theta} \xrightarrow{\delta\circ\gamma} \hat{\Delta}, \hat{S_B} \lhd \hat{P_B}).$$

The collection of types is a set, since every component of a type $\hat{A}$ is of type U instead of Set. This fits the original CwF definition.

In conclusion

- In a setting without UIP (like in HoTT), CwFs raise coherence issues
- 2 ways to solve this: by defining 'higher' CwFs, or by strictifying
- We focus on a specific example: the container model
- 1^{st} approach is ongoing work: proving higher coherence laws for the container GCwF, formalised in `Cubical Agda`
- 2^{nd} approach involves using an inductive-recursive universe and delaying type substitution

In conclusion

- In a setting without UIP (like in HoTT), CwFs raise coherence issues
- 2 ways to solve this: by defining 'higher' CwFs, or by strictifying
- We focus on a specific example: the container model
- 1^{st} approach is ongoing work: proving higher coherence laws for the container GCwF, formalised in `Cubical Agda`
- 2^{nd} approach involves using an inductive-recursive universe and delaying type substitution

Thank you!

References I

Ahrens, B., Frumin, D., Maggesi, M., and van der Weide, N. (2019).
Bicategories in Univalent Foundations.
In Geuvers, H., editor, *FSCD 2019*, volume 131 of *LIPIcs*, pages 5:1–5:17, Dagstuhl, Germany. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Altenkirch, T. and Kaposi, A. (2021).
A container model of type theory.
In *TYPES 2021*.

Altenkirch, T., Kaposi, A., and Xie, S. (2025).
The groupoid-syntax of type theory is a set.

Cartmell, J. (1986).
Generalised algebraic theories and contextual categories.
Annals of Pure and Applied Logic, 32:209–243.

References II

Chen, J. (2025).
2-coherent internal models of homotopical type theory.

Dybjer, P. (1996).
Internal type theory.
In Berardi, S. and Coppo, M., editors, *Types for Proofs and Programs*, pages 120–134, Berlin, Heidelberg. Springer Berlin Heidelberg.

Jacobs, B. (1993).
Comprehension categories and the semantics of type dependency.
Theoretical Computer Science, 107(2):169–207.

References III

Kraus, N. (2021).
Internal ∞-categorical models of dependent type theory : Towards 2ltt eating hott.
LICS 2021, pages 1–14.

Uemura, T. (2022).
Normalization and coherence for ∞-type theories.