# Containers form a Groupoid CwF

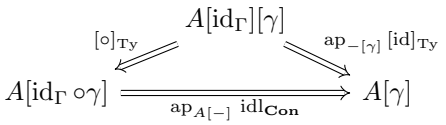
Stefania Damato[1,2 *] and Thorsten Altenkirch[1]

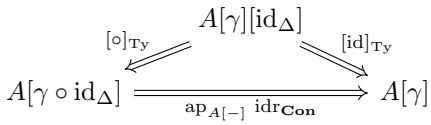
[1] University of Nottingham, Nottingham, UK
[2] Eötvös Loránd University, Budapest, Hungary

At TYPES 2021, Altenkirch and Kaposi proposed a model of type theory as a category with families (CwF) where types are interpreted as containers [2]. They wrote that their construction has the issue that types are not h-sets (or 0-types), therefore in a setting without uniqueness of identity proofs (UIP), their model does not precisely fit the original definition of a CwF. They conjectured that this model is an instance of a generalisation of CwFs where types can be h-groupoids (or 1-types), but that for this to be made precise the coherence issues associated to the h-levels of types need to be addressed. More recently, Altenkirch, Kaposi, and Xie specified such a generalisation of CwFs, called groupoid CwFs (GCwFs), and showed that the initial GCwF for a type theory with a base family and Π-types forms an h-set [3]. In this talk, we present recent work on showing that the container model proposed by Altenkirch and Kaposi does in fact form such a GCwF. We work in cubical type theory, and our construction is fully formalised in `Cubical Agda` and can be found at [5].

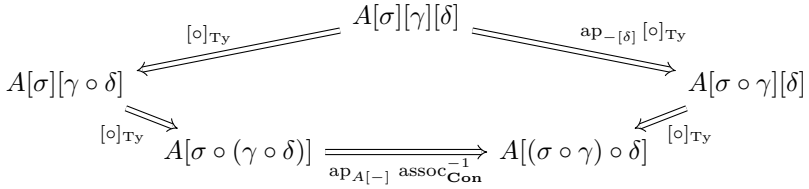
**Groupoid CwFs** The main distinction between CwFs and GCwFs is the structure involved to represent types. In a CwF, types are encoded as a presheaf on the category **Con** of contexts, $\mathrm{Ty}\colon \mathbf{Con}^{\mathrm{op}} \to \mathbf{Set}$. In a GCwF, we require $\mathrm{Ty}\colon \mathbf{Con}^{\mathrm{op}} \to \mathbf{Gpd}$ to be a pseudofunctor in the sense of Ahrens et. al [1], in the special case where the source is the 1-category of contexts **Con** instead of a bicategory. Concretely, the structure we want Ty to have is as follows, where we interpret **Gpd** synthetically i.e. we mean **hGpd**.

- A map on objects $\mathrm{Ty}\colon \mathbf{Con} \to \mathbf{Gpd}$.
- A contravariant map on morphisms $-[-]_{\mathrm{Ty}}\colon \mathrm{Ty}\ \Delta \to \mathbf{Con}(\Gamma, \Delta) \to \mathrm{Ty}\ \Gamma$.
- For $\Gamma$ in **Con** and $A : \mathrm{Ty}\ \Gamma$, an equality $[\mathrm{id}]_{\mathrm{Ty}}\colon A[\mathrm{id}_\Gamma] \equiv_{\mathrm{Ty}\ \Gamma} A$.
- For $\Delta \xrightarrow{\gamma} \Gamma \xrightarrow{\sigma} \Sigma$ in **Con** and $A : \mathrm{Ty}\ \Sigma$, an equality $[\circ]_{\mathrm{Ty}}\colon A[\sigma][\gamma] \equiv_{\mathrm{Ty}\ \Delta} A[\sigma \circ \gamma]$.
- For $\gamma\colon \Delta \to \Gamma$ in **Con** and $A : \mathrm{Ty}\ \Gamma$, the following commutative diagrams on equalities, which we call $[\mathrm{id}]_{\mathrm{Ty}}$-cohL and $[\mathrm{id}]_{\mathrm{Ty}}$-cohR respectively, known as the **triangulators**.




- For $\Xi \xrightarrow{\delta} \Delta \xrightarrow{\gamma} \Gamma \xrightarrow{\sigma} \Sigma$ in **Con** and $A : \mathrm{Ty}\ \Sigma$, the following commutative diagram on equalities, called $[\circ]_{\mathrm{Ty}}$-coh, known as the **pentagonator**.




* Supported by ERC grant 101170308.

**The container model** We briefly outline the container model of type theory proposed by Altenkirch and Kaposi. This model can be seen as a restriction of the presheaf model, where we only consider endofunctors on **Set** that are container functors. *Contexts* are set-containers, i.e. pairs of shapes $S_\Gamma : \mathsf{Set}$ and positions $P_\Gamma : S_\Gamma \to \mathsf{Set}$, written $S_\Gamma \triangleleft P_\Gamma$. *Substitutions* $\delta : S_\Gamma \triangleleft P_\Gamma \to S_\Delta \triangleleft P_\Delta$ are container morphisms, i.e. pairs $\delta_s : S_\Gamma \to S_\Delta, \delta_p : (s : S_\Gamma) \to P_\Delta\,(\delta_s\, s) \to P_\Gamma\, s$. Every container $S_\Gamma \triangleleft P_\Gamma$ gives rise to an endofunctor $[\![S_\Gamma \triangleleft P_\Gamma]\!] : \mathbf{Set} \to \mathbf{Set}$. A *type* over context $\Gamma$ is given by a generalised container $A$, i.e. a pair $S_A : \mathsf{Set}$, $P_A : S_A \to \int [\![\Gamma]\!]$, written $S_A \triangleleft_{\int [\![\Gamma]\!]} P_A$, which gives rise to a functor $[\![S_A \triangleleft_{\int [\![\Gamma]\!]} P_A]\!] : \int [\![\Gamma]\!] \to \mathbf{Set}$. A *term* of type $A$ in context $\Gamma$ is given by a 'dependent natural transformation' from $[\![\Gamma]\!]$ to $[\![A]\!]$. *Context extension* $\Gamma.A$ is defined using components of $A$.

**Proofs of the triangulators and pentagonator** Our main contribution is proving the necessary coherence laws, namely $[\mathrm{id}]_{\mathrm{Ty}}$-cohL, $[\mathrm{id}]_{\mathrm{Ty}}$-cohR, and $[\circ]_{\mathrm{Ty}}$-coh. These are all higher equalities between equalities stating rules on type substitution. In the container model, type substitution $A[\gamma]$ is defined as a generalised container whose shape component is a pullback and whose position component is a set-pushout. We encode pullbacks as families via the indexed-fibred translation: the pullback $\begin{array}{ccc} P & \longrightarrow & A \\ \downarrow & \lrcorner & \downarrow f \\ C & \underset{g}{\longrightarrow} & B \end{array}$ can be expressed as the family $\begin{array}{l} H : C \to \mathcal{U} \\ H\, c = \sum_{a:A} f\, a \equiv_B g\, c. \end{array}$

We represent set-pushouts as set-truncated higher inductive types (HITs).

We sketch the proof of $[\mathrm{id}]_{\mathrm{Ty}}$-cohL to give an intuitive idea of the proofs involved. Firstly, since we work in a cubical setting, we rewrite the triangle $[\mathrm{id}]_{\mathrm{Ty}}$-cohL as a square with one side being refl, resulting in the left square below. The proofs of $[\mathrm{id}]_{\mathrm{Ty}}$ and $[\circ]_{\mathrm{Ty}}$ are of the form $\mathbf{uaGenCon}\, e$, where $\mathbf{uaGenCon} : A \simeq_{\mathrm{GenCon}} B \to A \equiv B$ is a univalence principle specialised to generalised containers, and $e$ is an equivalence of the required type. Equivalence between two generalised containers $\simeq_{\mathrm{GenCon}}$ is defined as component-wise equivalence. We can then express this square as the equivalent square on the right.

$$\begin{array}{ccc} A[\mathrm{id}_\Gamma \circ \gamma] & \xRightarrow{\mathrm{ap}_{A[-]}\ \mathrm{idl}_{\mathbf{Con}}} & A[\gamma] \\ {\scriptstyle [\circ]_{\mathrm{Ty}}}\uparrow\uparrow & & \uparrow\uparrow{\scriptstyle \mathrm{refl}} \\ A[\mathrm{id}_\Gamma][\gamma] & \xRightarrow[\mathrm{ap}_{-[\gamma]}\,[\mathrm{id}]_{\mathrm{Ty}}]{} & A[\gamma] \end{array} \qquad \begin{array}{ccc} A[\gamma] & \xRightarrow{\mathbf{uaGenCon}\ \mathrm{id}_{\simeq_{\mathrm{GenCon}}}} & A[\gamma] \\ {\scriptstyle \mathbf{uaGenCon}\atop\scriptstyle (\ldots[\circ]_{\mathrm{Ty}}\text{-eq})}\uparrow\uparrow & & \uparrow\uparrow{\scriptstyle \mathbf{uaGenCon}\atop\scriptstyle \mathrm{id}_{\simeq_{\mathrm{GenCon}}}} \\ A[\mathrm{id}_\Gamma][\gamma] & \xRightarrow[\mathbf{uaGenCon}\ (\ldots[\mathrm{id}]_{\mathrm{Ty}}\text{-eq})]{} & A[\gamma] \end{array}$$

At this point, we have a square whose edges are all of the form $\mathbf{uaGenCon}\ x$ for some equivalence $x$. We can now factor out the univalence from our proof goal, and focus on proving commutativity of a square of equivalences of generalised containers; we call such a square a $\simeq$GenConSquare, which is defined as component-wise equalities or paths of generalised container components. Constructing such a $\simeq$GenConSquare involves only one non-trivial equality proof, that of two set-pushouts, which we prove by using the set-pushout's eliminator and some path algebra. What remains to show is that the square on the right is equal to our original goal on the left. This requires us to prove and apply some lemmas on the behaviour of $\simeq_{\mathrm{GenCon}}$ and **uaGenCon**. For example, we need that two equivalent generalised containers remain so after the same substitution is applied to both: $A \simeq_{\mathrm{GenCon}} B \to (A[\gamma]_{\mathrm{Ty}}) \simeq_{\mathrm{GenCon}} (B[\gamma]_{\mathrm{Ty}})$, and that **uaGenCon** on the identity equivalence is refl: $\mathbf{uaGenCon}\,\mathrm{id}_{\simeq_{\mathrm{GenCon}}} \equiv \mathrm{refl}$.

**Further work** We have worked out that this model admits $\Sigma$-types but we still do not know about $\Pi$-types—we sketch the difficulties that arise with $\Pi$-types in our talk. Von Glehn's polynomial functor model of type theory [6], which has the same contexts and substitutions as ours but different types and terms, was shown to admit $\Pi$-types but refute function extensionality [4]. In our case this remains an open problem.

# References

[1] Benedikt Ahrens, Dan Frumin, Marco Maggesi, Niccolò Veltri, and Niels van der Weide. Bicategories in univalent foundations. *Mathematical Structures in Computer Science*, 31(10):1232–1269, 2021.

[2] Thorsten Altenkirch and Ambrus Kaposi. A container model of type theory, 2021. TYPES 2021 abstract, available at https://types21.liacs.nl/download/a-container-model-of-type-theory/.

[3] Thorsten Altenkirch, Ambrus Kaposi, and Szumi Xie. The Groupoid-Syntax of Type Theory Is a Set. In Stefano Guerrini and Barbara König, editors, *34th EACSL Annual Conference on Computer Science Logic (CSL 2026)*, volume 363 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 40:1–40:23, Dagstuhl, Germany, 2026. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[4] Robert Atkey. Interpreting dependent types with containers. Talk at the MSP101 seminar, University of Strathclyde, slides at https://bentnib.org/docs/tt-in-containers.pdf, code at https://gist.github.com/bobatkey/0d1f04057939905d35699f1b1c323736, 2020.

[5] Stefania Damato. The groupoid category with families of containers, December 2025. Accompanying formalisation, available at https://github.com/stefaniatadama/container-gcwf.

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