The Groupoid-syntax of Type Theory is a Set

Thorsten Altenkirch ✉ 🄳
University of Nottingham, United Kingdom

Ambrus Kaposi ✉ 🄳
Eötvös Loránd University (ELTE), Budapest, Hungary

Szumi Xie ✉ 🄳
Eötvös Loránd University (ELTE), Budapest, Hungary

───── **Abstract** ───

Categories with families (CwFs) have been used to define the semantics of type theory in type theory. In the setting of Homotopy Type Theory (HoTT), one of the limitations of the traditional notion of CwFs is the requirement to set-truncate types, which excludes models based on univalent categories, such as the standard set model. To address this limitation, we introduce the concept of a *Groupoid Category with Families (GCwF)*. This framework truncates types at the groupoid level and incorporates coherence equations, providing a natural extension of the CwF framework when starting from a 1-category.

We demonstrate that the initial GCwF for a type theory with a base family of sets and Π-types (groupoid-syntax) is set-truncated. Consequently, this allows us to utilize the conventional intrinsic syntax of type theory while enabling interpretations in semantically richer and more natural models. All constructions in this paper were formalised in Cubical Agda.

1 Introduction

In [6], an *intrinsically typed syntax* for basic type theory using a *Quotient-Inductive-Inductive Type (QIIT)* was introduced. By intrinsically typed, we mean that the syntax directly enforces typing constraints, eliminating the need for separate untyped preterms. The equational theory is integrated naturally using *path constructors* from Homotopy Type Theory (HoTT), while set-truncation ensures that types behave as sets.

QIITs are a special case of *Higher Inductive-Inductive Types (HIITs)* where all types are truncated to sets by adding a higher path constructor. The term *inductive-inductive* signals that constructors can reference other constructors in their types. In essence, [6] defined the syntax of type theory as the *initial Category with Families (CwF)* with Π-types and an uninterpreted base family. This allowed the syntax to be interpreted in any CwF with the necessary structure and served as the foundation for a proof of normalisation using *Normalisation by Evaluation (NbE)* [7].

However, this approach had a significant limitation: the syntax could not be interpreted in the *intended model* where types are sets. This issue arose due to the use of set truncation, which enforced types to be sets but precluded a univalent semantics, such as Set. To work around this, inductive-recursive universes were used. While effective, this approach was unsatisfactory as it excluded univalent models, which are natural semantics for type theory.

Simply omitting set truncation is not a solution. Without truncation: (i) we cannot prove necessary equations in the syntax; (ii) the syntax itself is no longer a set, which e.g. makes equality in the syntax undecidable. A fully principled solution would require adding

all higher coherences. However, this is both technically complex and generally believed to require a *2-level type theory* rather than plain HoTT [29].

In this paper, we propose a middle ground: we lift the truncation level to *groupoids* and add a minimal set of coherence equations. This enables interpreting the syntax into the set model and other univalent category-based models. This compromise aligns naturally with the structure of categories in HoTT [35], where *objects* are groupoids with no truncation restriction, while *hom-sets* remain sets, as their name implies. Actually, we only restrict types to be groupoids, then we can prove that contexts in the syntax also form a groupoid.

At first glance, this raises a new concern: does lifting to groupoids and adding coherence equations require redefining the syntax? Do we lose decidability of equality? Our *main result* resolves this concern:

> **The groupoid-syntax of type theory with Π-types and a base family has types and contexts that are sets.**

In essence, we retain the *set-truncated syntax of type theory* while enabling evaluation in *groupoid-level models*. This allows us to interpret the set-truncated syntax into univalent models, such as Set or the *container model* [8]. However, we note that univalence for types cannot be assumed as a principle at the judgmental level—doing so would mean that types are not a set anymore.

Contributions.   The main contributions of this paper are as follows:

- We introduce the notion of a *Groupoid Category with Families (GCwF)* with Π-types and a base family.
- We show that the initial GCwF with Π-types and a base family is *set-truncated*.
- We establish the above proof using an *α-normalisation* construction.
- As a result, we enable the definition of the univalent *set model* and other univalent category-based models for the set-truncated syntax.

All results are formalised in *Cubical Agda*.

Structure of the paper.   After listing related work, we describe our metatheory and notation in Section 2. In Section 3, we define various syntaxes as HIITs and describe the problem of interpreting the set-truncated syntax in sets. In Section 4 we show that the groupoid-syntax is a set. We use this fact in Section 5 to fix the above problem. We conclude in Section 6.

Related work.   This paper is a continuation of the series of papers internalising the intrinsic syntax of type theory in type theory [20, 16] and in homotopy type theory [33, 6]. Intrinsic syntax means that there are only well-formed, well-scoped, well-typed terms which are quotiented by conversion. This is in contrast with extrinsic style formalisations [1, 2]. We use a variant of Dybjer's CwFs [22] introduced by Ehrhard [23, 18].

Infinite-dimensional versions of our 1-dimensional notion of model are given by Kraus and Uemura. Kraus defines a notion of ∞-CwF [29] inside an extension of type theory with a strict equality (two-level type theory, [4, 10, 11]). He conjectures that the set-level (0-level) syntax is initial for his ∞-model. Uemura [34] proves normalisation for an ∞-dimensional presentation of type theory, however his work is not formalised in intensional type theory.

Our theorem that the initial GCwF with certain type formers is set-truncated can be seen as a simple coherence theorem analogous to that of monoidal categories. Coherence for monoidal categories says that in the free monoidal category over a set of objects, morphisms form a set. Our coherence theorem is for types rather than morphisms (substitutions), and we

generate the types from a set-valued family using Π and instantiation. Coherence for monoidal groupoids was proven in HoTT by Piceghello [32], where he also used groupoid-truncated HITs to define the free monoidal groupoid.

In HoTT, the ideal solution for coherence problems is to find finite descriptions which imply all the infinitely many coherences. For example, usability of integers defined as set-quotients is limited, but there is a way to define their ∞-version without truncation [9]. Free groups can defined without truncation [30], however originally groupoid-truncation was needed to prove the free group over a set is a set. The general case was resolved by Wärn [37]. The Symmetry book [12] contains several similar examples.

There are notions of model of type theory weaker than CwFs where e.g. substitution is only functorial up to isomorphism [24, 31, 13]. Formulating the weaker notion of model in a 2-categorical setting clarifies and simpifies the situation. This has been used by Dybjer and Clairambault [17] to prove the 2-equivalence of locally cartesian closed categories and Martin-Löf type theories, and by Van der Weide to describe comprehension categories in a univalent setting [3].

Higher inductive-inductive types (HIITs) have been used before to describe free algebraic structures such as real numbers [35], the partiality monad [5], or hybrid semantics [21], but all of these are set-truncated HIITs, unlike our groupoid-syntax. Cubical type theory supports HITs [19, 15], and there is a scheme for describing HIITs [26] which covers our usages.

2 Metatheory and formalisation

Everything in this paper is formalised in Cubical Agda [36], the formalisation is available online in an anonymised form: the zip file of all source code is available; next to definitions/-theorems/etc., ↻ icons point to the corresponding part in the HTML version of the source code. In the paper text, we use informal cubical type theory: this means that we don't refer to the interval and instead of using 3-dimensional cubes, we only compose and fill larger 2-dimensional shapes.

We write $\equiv$ for equations holding definitionally, $:\equiv$ denotes definitions. Dependent function space is written $(x : A) \to B$ or $\forall x.\, B$, we also use implicit quantifications. We write dependent pairs as $(x : A) \times B$, the empty type as $\bot$, the unit type as $\top$. The universe of types is Type, we also use the universe of h-sets Set and h-groupoids $\mathsf{Groupoid}$. We have a predicative universe hierarchy, but we don't write levels for readability. The identity (path, equality) type is written $a =_A b$ for $a, b : A$, where the subscript $_A$ is usually ommitted. The dependent path type (PathP, heterogeneous equality) is written $b_0 =_B^e b_1$ for $e : a_0 = a_1$ and $b_i : B\, a_i$, sometimes the subscript $_B$ is ommitted. We overload functions and their congruence (ap operator), e.g. $f\, e : f\, a = f\, b$ where $e : a = b$, and we omit symmetries as well. Transport is written $e_*\, b_0 : B\, a_1$ for $e : a_0 = a_1$ and $b_0 : B\, a_0$, we tend to give a separate name for operations using transport (e.g. $-[-]^{\mathsf{U}}$ is a transported version of $-[-]$). The obvious element of the heterogeneous equality $b_0 =_B^e e_*\, b_0$ is called transportFiller. The composition operator of cubical type theory is the generalisation of transitivity as depicted below, it also comes with a filler operation.
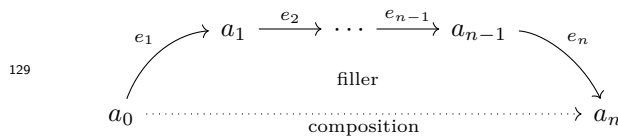
For the composite equality e, we denote the filler by fillerOf e. Some of these composition

and filling operations are primitive in cubical type theory, but they are also definable via the eliminator of the identity type (J). In this paper we abstract over these differences.

We write compositions with equational reasoning by $a_0 \overset{e_1}{=} a_1 \overset{e_2}{=} \ldots \overset{e_n}{=} a_n$, or its multi-line variant (left, below). Composition also works for heterogeneous equalities, in this case we write the base equalities in superscripts (right, below).

$$a_0 \;=(e_1) \qquad\qquad\qquad\qquad b_0 \;=^{e_1}(e_1')$$
$$\ldots \qquad\qquad\qquad\qquad\qquad\quad \ldots$$
$$a_{n-1}=(e_n) \qquad\qquad\qquad\quad b_{n-1}=^{e_n}(e_n')$$
$$a_n \qquad\qquad\qquad\qquad\qquad\quad b_n$$

Here $b_i : B\,a_i$, $e_i : a_{i-1} = a_i$ and $e_i' : b_{i-1} =^{e_i}_B b_i$, and the resulting heterogeneous equality is $b_0 =^{\text{composite of the } e_i\text{s}}_B b_n$. We denote heterogeneous composition and filling of shapes by drawing the base diagram below the dependent diagram. We say that the right diagram is *over* the left one: in this case the dotted composition line has type $b_0 =^{e_0}_B b_n$.
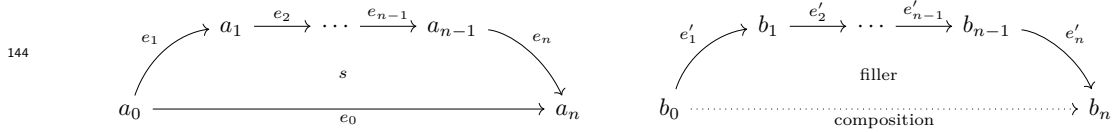
Some squares can be filled by the fact that every parameterised path is natural. We denote

these naturality squares by writing nat in the center:

There are some technical limitations of Cubical Agda that we have to circumvent in the formalisation, but are not visible in the text of this paper. We summerise these below.

- Interleaved constructors of (higher) inductive-inductive datatypes are not allowed in Cubical Agda. For example, this fragment of a syntax of a type theory is not allowed:

$$\mathsf{Con} : \mathsf{Type} \qquad\qquad\qquad - \rhd - : (\varGamma : \mathsf{Con}) \to \mathsf{Ty}\,\varGamma \to \mathsf{Con}$$
$$\mathsf{Ty} \; : \mathsf{Con} \to \mathsf{Type} \qquad\qquad \Sigma \quad : (A : \mathsf{Ty}\,\varGamma) \to \mathsf{Ty}\,(\varGamma \rhd A) \to \mathsf{Ty}\,\varGamma$$
$$\mathsf{eq} \quad : \varGamma \rhd \Sigma\,A\,B =_{\mathsf{Con}} \varGamma \rhd A \rhd B$$

Here every later constructor depends on all the previous constructors, the order can't be modified, and first we have a Con-constructor, then a Ty-constructor, then another Con-constructor. We solve this via the encoding proposed in [25], which uses the same idea as encoding mutual inductive types as an indexed family [27]: we introduce a sort of codes Code and a family of elements EL, and then all constructors are in the same sort:

$$\mathsf{Code} : \mathsf{Type} \qquad\qquad\qquad - \rhd - : (\varGamma : \mathsf{EL}\,\mathsf{Con}) \to \mathsf{EL}\,(\mathsf{Ty}\,\varGamma) \to \mathsf{EL}\,\mathsf{Con}$$
$$\mathsf{EL} \quad : \mathsf{Code} \to \mathsf{Type} \qquad\quad \Sigma \quad : (A : \mathsf{EL}\,(\mathsf{Ty}\,\varGamma)) \to \mathsf{EL}\,(\mathsf{Ty}\,(\varGamma \rhd A)) \to \mathsf{EL}\,(\mathsf{Ty}\,\varGamma)$$
$$\mathsf{Con} \; : \mathsf{Code} \qquad\qquad\qquad \mathsf{eq} \quad : \varGamma \rhd \Sigma\,A\,B =_{\mathsf{EL}\,\mathsf{Con}} \varGamma \rhd A \rhd B$$
$$\mathsf{Ty} \quad : \mathsf{EL}\,\mathsf{Con} \to \mathsf{Code}$$

We use the same technique when defining our syntaxes (Definitions 2, 6, 7).

- When we describe HIITs, we use transport and composition, but in the formalisation, we avoid them (we still use composition operators in some 2-paths). The reason is twofold:

166 (i) Agda does not see that these operations preserve strict positivity; (ii) as the β rule
167 for transport is not definitional, it makes it difficult to formalise strict models such as
168 the Type-interpretation. Instead, we make sure that all transports appear outermost
169 and then can be encoded via dependent paths (a dependent path on refl computes to a
170 nondependent one). When it is not possible to do this, we add extra constructors together
171 with equations which singleton contract them. For example, in the text of the paper we
172 write the substitution law for El using a transport: $(\mathsf{El}\,\hat{A})[\gamma] = \mathsf{El}\,((\mathsf{U}[]\,\gamma)_*\,(\hat{A}[\gamma]))$. In
173 the formalisation, we introduce a new constructor $-[-]^{\mathsf{U}} : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta\,\mathsf{U}$
174 together with the contracting equation $\hat{A}[\gamma] =^{\mathsf{U}[]\,\gamma}\,\hat{A}[\gamma]^{\mathsf{U}}$, and then use this new constructor
175 when describing El[].
176 ▬ When characterising the equality of normal types, in the formalisation we use the
177 inductively defined Martin-Löf identity type instead of the built-in path type (note
178 that they are equivalent). This is convenient because J computes definitionally on its
179 constructor refl. In the text of the paper we abstract over this.

3 Variants of the syntax and the set interpretation

181 In this section we define three different variants of the syntax of a type theory with dependent
182 function space and a base family: the wild syntax, the set-truncated and the groupoid-
183 truncated syntax. We show that types in the wild syntax don't form a set, so in particular
184 they cannot have decidable equality. The set-syntax cannot be interpreted into the set model
185 directly, while the groupoid-syntax can.

186 ▶ **Parameter 1.** Everything in this section is parameterised by an $X : \mathsf{Set}$ and a $Y : X \to \mathsf{Set}$.

187 ▶ **Definition 2** (Wild syntax ♺)**.** We define a higher inductive-inductive type with four
188 sorts. It starts with a category with a terminal object. Objects are called contexts and
189 morphisms are called substitutions, the terminal object is called the empty context. Note
190 that composition $-\circ-$ takes the Γ, Δ and Θ arguments implicitly, and similarly for all the
191 forthcoming operations and equations.

Con	: Type	idl	$: \forall\gamma.\,\mathsf{id}\circ\gamma = \gamma$
Sub	: Con → Con → Type	idr	$: \forall\gamma.\,\gamma\circ\mathsf{id} = \gamma$
$-\circ-$	$: \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Theta\,\Gamma$	◇	: Con
ass	$: \forall\gamma\,\delta\,\theta.\,\gamma\circ(\delta\circ\theta) = (\gamma\circ\delta)\circ\theta$	ε	$: \mathsf{Sub}\,\Gamma\,\diamond$
id	$: \mathsf{Sub}\,\Gamma\,\Gamma$	◇η	$: (\sigma : \mathsf{Sub}\,\Gamma\,\diamond) \to \sigma = \varepsilon$

197 Types form a presheaf over the category of contexts and substitutions. The action on
198 morphisms is called instantiation, it uses a flipped notation because of contravariance.

Ty	: Con → Type	[∘]	$: \forall A\,\gamma\,\delta.\,A[\gamma\circ\delta] = A[\gamma][\delta]$
$-[-]$	$: \mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta$	[id]	$: \forall A.\,A[\mathsf{id}] = A$

201 Terms form a dependent presheaf over types. The instantiation operation is overloaded. Note
202 that the functor laws are paths dependent over the functor laws for Ty.

Tm	$: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Type}$	[∘]	$: \forall a\,\gamma\,\delta.\,a[\gamma\circ\delta] =^{[\circ]\,A\,\gamma\,\delta}_{\mathsf{Tm}\,\Theta}\,a[\gamma][\delta]$
$-[-]$	$: \mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma])$	[id]	$: \forall a.\,a[\mathsf{id}] =^{[\mathsf{id}]\,A}_{\mathsf{Tm}\,\Gamma}\,a$

In addition to context extension (infix triangle), we have lifting of substitutions which is its functorial action on morphisms. The functor laws again depend on those for Ty.

$$- \triangleright - : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con} \qquad \circ^+ : \forall \gamma\,\delta\,.\,(\gamma \circ \delta)^+ =^{[\circ]\,A\,\gamma\,\delta} \gamma^+ \circ \delta^+$$

$$-^+ \quad : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Sub}\,(\Delta \triangleright A[\gamma])\,(\Gamma \triangleright A) \qquad \mathsf{id}^+ : \mathsf{id}^+ =^{[\mathsf{id}]\,A} \mathsf{id}$$

We have weakening p (or first projection), and zero De Bruijn index q (second projection). We explain how to compose either with lifted substitutions.

$$\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma \quad \mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}]) \quad \mathsf{p} \circ^+ : \forall \gamma.\,\mathsf{p} \circ \gamma^+ = \gamma \circ \mathsf{p} \quad \mathsf{q}[^+] : \forall \gamma.\,\mathsf{q}[\gamma^+] =^e \mathsf{q}$$

The last equation is heterogeneous over the previous one, e abbreviates the following composite path in $\mathsf{Ty}\,(\Delta \triangleright A[\gamma])$: $A[\mathsf{p}][\gamma^+] \overset{[\circ]\,A\,\mathsf{p}\,\gamma^+}{=} A[\mathsf{p} \circ \gamma^+] \overset{\mathsf{p} \circ^+ \gamma}{=} A[\gamma \circ \mathsf{p}] \overset{[\circ]\,A\,\gamma\,\mathsf{p}}{=} A[\gamma][\mathsf{p}]$.

So far we have all weakenings and variables, for example De Bruijn index 3 is given by $\mathsf{q}[\mathsf{p}][\mathsf{p}][\mathsf{p}]$. Now we introduce single substitutions via $\langle a \rangle$ which instantiates the last variable in the context by a, and leaves the rest. It commutes with any substitution, and we explain how to compose p and q with single substitutions.

$$\langle - \rangle : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Sub}\,\Gamma\,(\Gamma \triangleright A) \qquad \mathsf{p} \circ \langle \rangle : \forall a.\,\mathsf{p} \circ \langle a \rangle = \mathsf{id}$$

$$\langle \rangle \circ : \forall a\,\gamma.\,\langle a \rangle \circ \gamma = \gamma^+ \circ \langle a[\gamma] \rangle \qquad \mathsf{q}[\langle \rangle] : \forall a.\,\mathsf{q}[\langle a \rangle] =^e a$$

Again, the last equation is heterogeneous over the previous one, where e abbreviates the following path in $\mathsf{Ty}\,\Gamma$: $A[\mathsf{p}][\langle a \rangle] \overset{[\circ]\,A\,\mathsf{p}\,\langle a \rangle}{=} A[\mathsf{p} \circ \langle a \rangle] \overset{\mathsf{p} \circ \langle \rangle}{=} A[\mathsf{id}] \overset{[\mathsf{id}]\,A}{=} A$.

The last equation for the substitution calculus is an η law explaining that an identity substitution on an extended context is given by p and q.

$$\triangleright \eta : \mathsf{id} = \mathsf{p}^+ \circ \langle \mathsf{q} \rangle$$

We have a base type and a family over it, and elements of these coming from the parameters.

$$\mathsf{U} : \mathsf{Ty}\,\Gamma \qquad \mathsf{El} : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Ty}\,\Gamma \qquad \mathsf{inU} : X \to \mathsf{Tm}\,\diamond\,\mathsf{U} \qquad \mathsf{inEl} : Y\,x \to \mathsf{Tm}\,\diamond\,(\mathsf{El}\,(\mathsf{inU}\,x))$$

The substitution law for U is easy. To express $\mathsf{El}[]$, we introduce notation for the instantiation operation of terms of type U, which is just a transported version of ordinary instantiation.

$$\mathsf{U}[] : \forall \gamma.\,\mathsf{U}[\gamma] = \mathsf{U} \qquad\qquad -[-]^{\mathsf{U}} : \mathsf{Tm}\,\Gamma\,\mathsf{U} \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Tm}\,\Delta\,\mathsf{U}$$

$$\mathsf{El}[] : \forall \gamma.\,(\mathsf{El}\,\hat{A})[\gamma] = \mathsf{El}\,(\hat{A}[\gamma]^{\mathsf{U}}) \qquad\qquad \hat{A}[\gamma]^{\mathsf{U}} :\equiv (\mathsf{U}[]\,\gamma)_* \,\hat{A}[\gamma]$$

We introduce a transport-filler heterogeneous equality for each $\hat{A}$ and γ that we will make use of later: $\hat{A}[\gamma]^{\mathsf{U}}\mathsf{filler} : \hat{A}[\gamma] =^{\mathsf{U}[]\,\gamma} \hat{A}[\gamma]^{\mathsf{U}}$.

Dependent function space with β, η laws is defined by the isomorphism $\mathsf{Tm}\,(\Gamma \triangleright A)\,B \cong \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$ natural in Γ. It is enough to state naturality in one direction.

$$\Pi \quad : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma \qquad \Pi\beta \quad : \forall b.\,\mathsf{app}\,(\mathsf{lam}\,b) = b$$

$$\Pi[] \quad : \forall A\,B\,\gamma.\,(\Pi\,A\,B)[\gamma] = \Pi\,(A[\gamma])\,(B[\gamma^+]) \quad \Pi\eta \quad : \forall f.\,\mathsf{lam}\,(\mathsf{app}\,f) = f$$

$$\mathsf{lam} : \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \qquad \mathsf{lam}[] : \forall b\,\gamma.\,(\mathsf{lam}\,b)[\gamma] =^{\Pi[]\,A\,B\,\gamma} \mathsf{lam}\,(b[\gamma^+])$$

$$\mathsf{app} : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to \mathsf{Tm}\,(\Gamma \triangleright A)\,B$$

This concludes the definition of the wild syntax.

We defined the substitution calculus in Ehrhard's style [23, 18] instead of the more usual category with families (CwF) [22, 14]. These two presentations of the substitution calculus are isomorphic. In the above syntax, substitution extension $-, - : (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma]) \to \mathsf{Sub}\,\Delta\,(\Gamma \rhd A)$ is defined as $(\gamma, a) :\equiv \gamma^+ \circ \langle a \rangle$. In the other direction, $\gamma^+ :\equiv \big(\gamma \circ \mathsf{p}, ([\circ]\,A\,\gamma\,\mathsf{p})_* \,\mathsf{q}\big)$ and $\langle a \rangle :\equiv (\mathsf{id}, ([\mathsf{id}]\,A)_* \,a)$.

Although CwFs have one less operation and fewer equations, we chose the Ehrhard style syntax as there is no need to use the transport operation when specifying the equations. In CwFs, the naturality of substitution extension needs a transport in the middle: $(\gamma, a) \circ \delta = \big(\gamma \circ \delta, ([\circ]\,A\,\gamma\,\delta)_* \,(a[\delta])\big)$ In our syntax, all the transports are outermost, hence can be encoded by dependent paths.

▶ **Example 3** (Using the wild syntax ☿). We derive the other direction of naturality for the Π-isomorphism: this is the substitution law for app called app[].

$$(\mathsf{app}\,t)[\gamma^+] \qquad\qquad\qquad =(\Pi\beta\,t)$$

$$\mathsf{app}\,\Big(\mathsf{lam}\,\big((\mathsf{app}\,t)[\gamma^+]\big)\Big) \qquad\qquad =(\mathsf{lam}[]\,(\mathsf{app}\,t)\,\gamma^+)$$

$$\mathsf{app}\,\Big((\Pi[]\,A\,B\,\gamma)_* \,\big((\mathsf{lam}\,(\mathsf{app}\,t))[\gamma]\big)\Big) =(\Pi\eta\,t)$$

$$\mathsf{app}\,\big((\Pi[]\,A\,B\,\gamma)_* \,(t[\gamma])\big)$$

Nondependent function space is encoded as $A \Rightarrow B :\equiv \Pi\,A\,(B[\mathsf{p}])$.

The identity function for the family U, El is defined as

$$\mathsf{ID} : \mathsf{Tm}\,\diamond\,(\Pi\,\mathsf{U}\,(\mathsf{El}\,((\mathsf{U}[]\,\mathsf{p})_* \,\mathsf{q}) \Rightarrow \mathsf{El}\,((\mathsf{U}[]\,\mathsf{p})_* \,\mathsf{q}))) \qquad\qquad \mathsf{ID} :\equiv \mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})$$

Note that we had to transport the zero De Bruijn index $\mathsf{q} : \mathsf{Tm}\,(\diamond \rhd \mathsf{U})\,(\mathsf{U}[\mathsf{p}])$ so that we can apply El to it: $(\mathsf{U}[]\,\mathsf{p})_* \,\mathsf{q} : \mathsf{Tm}\,(\diamond \rhd \mathsf{U})\,\mathsf{U}$.

In the syntax, we have the categorical application operation for Π. Ordinary application is given by $- \cdot - : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\langle a \rangle])$ defined as $t \cdot a :\equiv (\mathsf{app}\,t)[\langle a \rangle]$. It is easy to prove its β law $(\mathsf{lam}\,t) \cdot a \equiv \mathsf{app}\,(\mathsf{lam}\,t)[\langle a \rangle] \overset{\Pi\beta\,t}{=} t[\langle a \rangle]$, but the η law is more involved as it needs several transports. We prove it via heterogeneous equality reasoning, where the proof of the equality of the types is written in the superscript of the equality sign.

$$f \qquad\qquad\qquad =(\Pi\eta\,f)$$

$$\mathsf{lam}\,(\mathsf{app}\,f) \qquad\qquad =^{[\mathsf{id}]\,B}([\mathsf{id}]\,(\mathsf{app}\,f))$$

$$\mathsf{lam}\,((\mathsf{app}\,f)[\mathsf{id}]) \qquad\qquad =^{\rhd\eta}(\rhd\eta)$$

$$\mathsf{lam}\,((\mathsf{app}\,f)[\mathsf{p}^+ \circ \langle \mathsf{q} \rangle]) \qquad\qquad =^{[\circ]\,B\,\mathsf{p}^+\,\langle \mathsf{q} \rangle}([\circ]\,(\mathsf{app}\,f)\,\mathsf{p}^+\,\langle \mathsf{q} \rangle)$$

$$\mathsf{lam}\,((\mathsf{app}\,f)[\mathsf{p}^+][\langle \mathsf{q} \rangle]) \qquad\qquad =(\mathsf{app}[]\,t\,\mathsf{p})$$

$$\mathsf{lam}\,\Big(\mathsf{app}\,\big((\Pi[]\,A\,B\,\mathsf{p})_* \,(f[\mathsf{p}])\big)[\langle \mathsf{q} \rangle]\Big) \equiv$$

$$\mathsf{lam}\,\big((\Pi[]\,A\,B\,\mathsf{p})_* \,(f[\mathsf{p}]) \cdot \mathsf{q}\big)$$

The type of the above heterogeneous equality is $f =^e_{\mathsf{Tm}\,\Gamma\,(\Pi\,A\,-)} \mathsf{lam}\,\big((\Pi[]\,A\,B\,\mathsf{p})_* \,(f[\mathsf{p}]) \cdot \mathsf{q}\big)$, where e is the following composite of the three heterogeneous steps in the above equality reasoning: $B \overset{[\mathsf{id}]\,B}{=} B[\mathsf{id}] \overset{\rhd\eta}{=} B[\mathsf{p}^+ \circ \langle \mathsf{q} \rangle] \overset{[\circ]\,B\,\mathsf{p}^+\,\langle \mathsf{q} \rangle}{=} B[\mathsf{p}^+][\langle \mathsf{q} \rangle]$.

▶ **Problem 4** (Type interpretation of the wild syntax ☿). *As a sanity check for our wild syntax, we define its type (standard, metacircular) interpretation.*

Construction. We define the following four recursive-recursive functions by pattern matching on the constructors of the higher inductive-inductive type.

$$[\![-]\!] : \mathsf{Con} \to \mathsf{Type} \qquad\qquad [\![-]\!] : \mathsf{Ty}\,\Gamma \to [\![\Gamma]\!] \to \mathsf{Type}$$

$$[\![-]\!] : \mathsf{Sub}\,\Delta\,\Gamma \to [\![\Delta]\!] \to [\![\Gamma]\!] \qquad\qquad [\![-]\!] : \mathsf{Tm}\,\Gamma\,A \to (\gamma : [\![\Gamma]\!]) \to [\![A]\!]\,\gamma$$

Composition is function composition ($[\![\gamma \circ \delta]\!]\,\bar{\theta} :\equiv [\![\gamma]\!]\,([\![\delta]\!]\,\bar{\theta})$), identity is identity ($[\![\mathsf{id}]\!]\,\bar{\gamma} :\equiv \bar{\gamma}$), instantiation is composition ($[\![A[\gamma]]\!]\,\bar{\delta} :\equiv [\![A]\!]\,([\![\gamma]\!]\,\bar{\delta})$), context extension is dependent sum ($[\![\Gamma \triangleright A]\!] :\equiv (\bar{\gamma} : [\![\Gamma]\!]) \times [\![A]\!]\,\bar{\gamma}$), lifting is $[\![\gamma^+]\!]\,(\bar{\delta},\bar{a}) :\equiv ([\![\gamma]\!]\,\bar{\delta},\bar{a})$, p and q are first and second projections, single substitution is $[\![\langle a \rangle]\!]\,\bar{\gamma} :\equiv (\bar{\gamma}, [\![a]\!]\,\bar{\gamma})$. Function space is interpreted by metatheoretic functions ($[\![\Pi\,A\,B]\!]\,\bar{\gamma} :\equiv (\bar{a} : [\![A]\!]) \to [\![B]\!]\,(\bar{\gamma},\bar{a})$). U and El are interpreted by X and Y, inU and inEl simply return their arguments. All the equations are refl.   ◀

The standard interpretation shows that our theory is consistent, that is, not all types are inhabited: $\mathsf{Tm}\,\diamond\,\mathsf{U}$ is interpreted by $\top \to X$ so it is inhabited if and only if X is.

▶ **Proposition 5** (↻). *Types in the wild syntax do not form a set.*

Proof. Every higher inductive type, including our Definition 2 can be interpreted into the unit type where all paths are interpreted by refl. We use a variant of this where every sort is interpreted by $\top$ except $\mathsf{Ty}\,\Gamma$ is interpreted by the circle S^1. Π, U and El are constant base, $A[\gamma]$ is interpreted by the interpretation of A. All equations are interpreted by refl, except $\mathsf{U}[]$ which is interpreted by loop. The two different proofs of $\mathsf{U}[\mathsf{id}] = \mathsf{U}$, namely $[\mathsf{id}]\,\mathsf{U}$ and $\mathsf{U}[]\,\mathsf{id}$ are interpreted by refl and loop, respectively.   ◀

When using the wild syntax, this is a practical problem: it can happen that we need a term of type $\mathsf{El}\,((\mathsf{U}[]\,\mathsf{id})_*\,a)$, but we only have a term of type $\mathsf{El}\,(([\mathsf{id}]\,\mathsf{U})_*\,a)$ available. From a broader perspective, Hedberg's theorem [35, Theorem 7.2.5] implies that we cannot prove normalisation for the wild syntax. In principle, there could be a clever way of defining the equations in the syntax such that there is only one proof for each equation. It is not known whether this is possible [33]. Instead, we make all the equations equal by force.

▶ **Definition 6** (Set-syntax ↻). The set-based syntax is the wild syntax (Definition 2) extended with the following three higher equality constructors. They truncate substitutions, types and terms to sets.
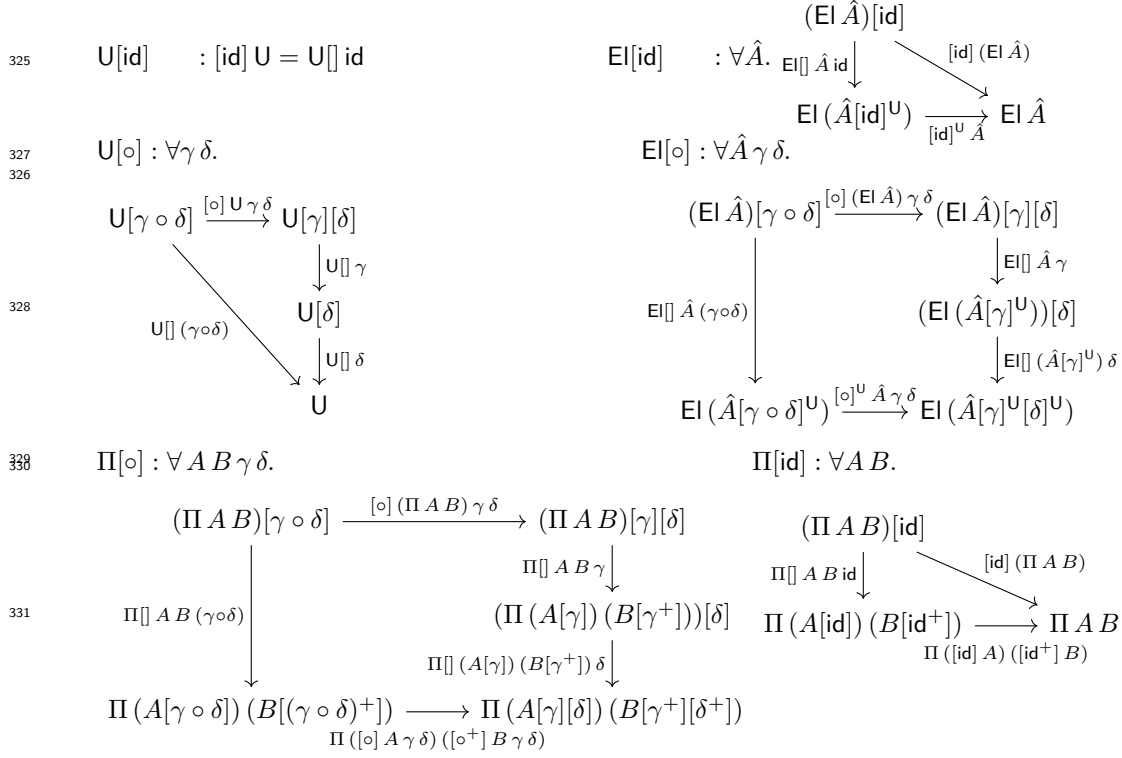
$$\mathsf{isSetTy}\ : (e\,e' : A_0 =_{\mathsf{Ty}\,\Gamma} A_1) \to e = e'$$

$$\mathsf{isSetSub} : (e\,e' : \gamma_0 =_{\mathsf{Sub}\,\Delta\,\Gamma} \gamma_1) \to e = e' \qquad \mathsf{isSetTm} : (e\,e' : a_0 =_{\mathsf{Tm}\,\Gamma\,A} a_1) \to e = e'$$

We don't add that contexts form a set as it is provable by induction on the context (↻).

Now we can hope for normalisation for this syntax, but the standard interpretation does not work anymore: the interpretation of $\mathsf{Ty}\,\Gamma$ would be $[\![\Gamma]\!] \to \mathsf{Type}$, but then we cannot interpret $\mathsf{isSetTy}$, as Type does not form a set. We have to limit ourselves to interpreting $\mathsf{Ty}\,\Gamma$ by $[\![\Gamma]\!] \to \mathsf{Prop}$ where Prop is defined as $(A : \mathsf{Type}) \times \big((x\,y : A) \to x = y\big)$. Alternatively, we can interpret Ty into an inductive-recursive universe as in [6, Section 6], but we cannot interpret the set-syntax in a univalent model. To fix this, we introduce a syntax where substitutions and terms are truncated to be sets, but types are only groupoid-truncated. To make types well-behaved, we add coherence laws which are equations between equations between types. These express that the substitution laws $\mathsf{U}[]$, $\mathsf{El}[]$ and $\Pi[]$ commute with the functoriality laws $[\circ]$, $[\mathsf{id}]$. In the diagrams below, the vertical directions are the substitution laws and the horizontal directions are the functoriality laws.

▶ **Definition 7** (Groupoid-syntax ♺). The groupoid-based syntax is the wild syntax (Definition 2) extended with the following higher equality constructors. Some of them are drawn as commutative diagrams.

$$\mathsf{isGrpdTy} : (w\,w' : e =_{A_0 =_{\mathsf{Ty}\,\Gamma} A_1} e') \to w = w'$$

$$\mathsf{isSetSub} : (e\,e' : \gamma_0 =_{\mathsf{Sub}\,\Delta\,\Gamma} \gamma_1) \to e = e' \qquad \mathsf{isSetTm} : (e\,e' : a_0 =_{\mathsf{Tm}\,\Gamma\,A} a_1) \to e = e'$$

$\mathsf{U[id]} \qquad : [\mathsf{id}]\,\mathsf{U} = \mathsf{U[]}\,\mathsf{id}$

$\mathsf{EI[id]} \qquad : \forall \hat{A}.$

$\mathsf{U[\circ]} : \forall \gamma\,\delta.$

$\mathsf{EI[\circ]} : \forall \hat{A}\,\gamma\,\delta.$

$\Pi[\circ] : \forall A\,B\,\gamma\,\delta.$

$\Pi[\mathsf{id}] : \forall A\,B.$

In the types of $\mathsf{U[\circ]}$ and $\mathsf{U[id]}$ above, $[\circ]^{\mathsf{U}}$ and $[\mathsf{id}]^{\mathsf{U}}$ abbreviate the following equality proofs. $[\circ]^{\mathsf{U}}$ is the dotted line in the left dependent square which is over the right square. $[\mathsf{id}]^{\mathsf{U}}$ is the dotted line in the upper dependent triangle which is over the lower triangle. As the bottom lines in the base square/triangle are reflexivities, $[\circ]^{\mathsf{U}}$ and $[\mathsf{id}]^{\mathsf{U}}$ are homogeneous equalities, but all the other lines in the upper shapes are heterogeneous. Fillers of the base shapes are written in their center, they are operations of the groupoid-syntax defined before. In Cubical Agda, the dotted lines are defined via heterogeneous composition. The $-[-]^{\mathsf{U}}\mathsf{filler}$ operation is part of Definition 2.
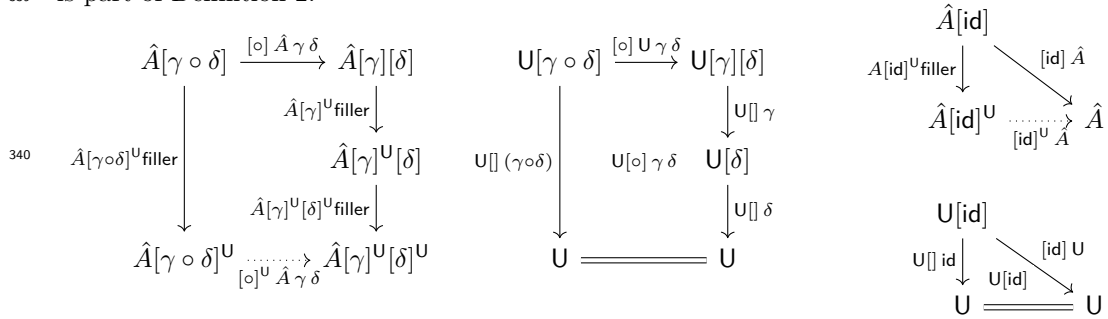
In the types of $\Pi[\circ]$ and $\Pi[\mathsf{id}]$ above, we used the following abbreviations of paths. $[\circ^+]$ and $[\mathsf{id}^+]$ are the dotted lines in the upper triangles, which are over the lower triangles. The dotted lines are defined by composition. We also give names to the fillers of the upper

344 triangles which will be used in Figures 2 and 3, respectively:

345
$$B[(\gamma \circ \delta)^+] \xrightarrow{\circ^+ \gamma \delta} B[\gamma^+ \circ \delta^+] \qquad A[\gamma \circ \delta] \xrightarrow{[\circ] A \gamma \delta} A[\gamma][\delta] \quad B[\mathsf{id}^+] \xrightarrow{\mathsf{id}^+} B[\mathsf{id}] \qquad A[\mathsf{id}] \xrightarrow{[\mathsf{id}] A} A$$

$$[\circ^+]\mathsf{filler}\, B\,\gamma\,\delta \Big\downarrow [\circ]\, B\,\gamma^+\,\delta^+ \qquad\qquad \Big\| \qquad\qquad [\mathsf{id}^+]\mathsf{filler}\, B \Big\downarrow [\mathsf{id}]\, B \qquad\qquad \Big\|$$

$$[\circ^+] B\,\gamma\,\delta \dashrightarrow B[\gamma^+][\delta^+] \qquad [\circ] A\,\gamma\,\delta \searrow A[\gamma][\delta] \quad [\mathsf{id}^+]\, B \dashrightarrow B \qquad [\mathsf{id}]\, A \rightarrow A$$

346 This concludes the definition of the groupoid-syntax.

347 ▶ **Notation 8.** We denote the components of the set-syntax by $_\mathsf{S}$ and the groupoid-syntax
348 by $_\mathsf{G}$ subscripts, e.g. $\mathsf{Con_S}$ and $\mathsf{Con_G}$.

349 We cannot redo the interpretation of Problem 4 because Type is not a groupoid, but we can
350 refine it by interpreting types into Set.

351 ▶ **Construction 9** (Set interpretation of the groupoid-syntax ↻). We define the following
352 functions mutually by pattern matching on the groupoid-syntax where $\mathsf{Set} :\equiv (X : \mathsf{Type}) \times$
353 $((e\,e' : x_0 =_X x_1) \to e = e')$.

354 $\quad [\![-]\!] : \mathsf{Con_G} \to \mathsf{Set} \qquad\qquad\qquad\qquad [\![-]\!] : \mathsf{Ty_G}\, \Gamma \to [\![\Gamma]\!]_{.1} \to \mathsf{Set}$

355 $\quad [\![-]\!] : \mathsf{Sub_G}\, \Delta\, \Gamma \to [\![\Delta]\!]_{.1} \to [\![\Gamma]\!]_{.1} \qquad [\![-]\!] : \mathsf{Tm_G}\, \Gamma\, A \to (\gamma : [\![\Gamma]\!]_{.1}) \to ([\![A]\!]\, \gamma)_{.1}$

356 The cases for the constructors are analogous to the ones in Problem 4, with additional proofs
357 of truncation-preservation: e.g. the empty context needs that $\top$ is a set, context extension
358 needs that Σ preserves set-truncation. U is interpreted by X, El by Y. We interpret the extra
359 truncation constructors as follows: we prove $\mathsf{isGrpdTy}$ by the fact that Set forms a groupoid,
360 while functions between sets are sets, which proves $\mathsf{isSetSub}$ and $\mathsf{isSetTm}$. All 1-dimensional
361 equalities and the 2-equalities $\mathsf{U[id]}$, $\mathsf{El[id]}$, $\Pi[\mathsf{id}]$ are interpreted by refl, while the 2-equalities
362 $\mathsf{U[\circ]}$, $\mathsf{El[\circ]}$, $\Pi[\circ]$ use cubical filling because these include compositions in the formalisation
363 (this could be avoided using the technique explained in Section 2).

364 The groupoid-syntax can be trivially interpreted into the set-syntax:

365 ▶ **Construction 10** (Set-syntax interpretation of the groupoid-syntax ↻). By pattern matching:

366 $\quad [\![-]\!] : \mathsf{Con_G} \to \mathsf{Con_S} \qquad\qquad\qquad\qquad [\![-]\!] : \mathsf{Ty_G}\, \Gamma \to \mathsf{Ty_S}\, [\![\Gamma]\!]$

367 $\quad [\![-]\!] : \mathsf{Sub_G}\, \Delta\, \Gamma \to \mathsf{Sub_S}\, [\![\Delta]\!]\, [\![\Gamma]\!] \qquad [\![-]\!] : \mathsf{Tm_S}\, \Gamma\, A \to \mathsf{Tm_S}\, [\![\Gamma]\!]\, [\![A]\!]$

368 Everything is interpreted by the corresponding component in the set-syntax, except (i)
369 $\mathsf{isGrpdTy_G}$ is interpreted by applying cumulativity of truncation levels to $\mathsf{isSetTy_S}$; (ii) the
370 higher equalities $\mathsf{U[\circ]}, \ldots, \Pi[\mathsf{id}]$ are interpreted by $\mathsf{isSetTy_S}$.

4 α-normalisation for the groupoid-syntax

372 In this section we prove that although elements of $\mathsf{Ty_G}$ in the groupoid-syntax are only
373 groupoid-truncated, they form a set. We define the set of α-normal forms for $\mathsf{Ty_G}$, and then
374 we show that every $\mathsf{Ty_G}$ is a retract of its α-normal forms. α-normalisation is the process of
375 eliminating explicit instantiations from types along the substitution laws for types.

4.1 α-normal forms

377 ▶ **Definition 11** (α-normal forms ↻). α-normal forms are given by the inductive family NTy
378 which is defined mutually with the quote function $\ulcorner - \urcorner$. We overload constructor names and

379   metavariables, but use brick red colour for disambiguation.

380   $\mathsf{NTy} : \mathsf{Con_G} \to \mathsf{Type}$ $\ulcorner \_ \urcorner$ $: \mathsf{NTy}\,\Gamma \to \mathsf{Ty_G}\,\Gamma$

381   U $: \mathsf{NTy}\,\Gamma$ $\ulcorner\mathsf{U}\urcorner$ $:\equiv \mathsf{U_G}$

382   El $: \mathsf{Tm_G}\,\Gamma\,\mathsf{U_G} \to \mathsf{NTy}\,\Gamma$ $\ulcorner\mathsf{El}\,\hat{A}\urcorner$ $:\equiv \mathsf{El_G}\,\hat{A}$

383   Π $: (A : \mathsf{NTy}\,\Gamma) \to \mathsf{NTy}\,(\Gamma \triangleright_{\mathsf{G}} \ulcorner A \urcorner) \to \mathsf{NTy}\,\Gamma$ $\ulcorner\Pi\,A\,B\urcorner :\equiv \Pi_{\mathsf{G}}\,\ulcorner A\urcorner\,\ulcorner B\urcorner$

384   It is not obvious that α-normal forms are a set because NTy is indexed by $\mathsf{Con_G}$ which
385   contains elements of $\mathsf{Ty_G}$ for which at this point we don't know that it forms a set. NTy also
386   includes non-normal terms (via El), hence we cannot rely on decidability of equality and
387   Hedberg's theorem [35, Section 7.2]. However, we can still show the following.

388   ▶ **Lemma 12** (♺). $\mathsf{NTy}\,\Gamma$ *forms a set.*

389   **Proof.** We use the encode-decode method [35] to characterise equality of NTy. The cover (or
390   code) relation is defined by double-recursion on NTy, mutually with the decode function.

391   $\mathsf{Cover} : \mathsf{NTy}\,\Gamma \to \mathsf{NTy}\,\Gamma \to \mathsf{Type}$ $\mathsf{decode} : \mathsf{Cover}\,A_0\,A_1 \to A_0 = A_1$

392   $\mathsf{Cover}\,\mathsf{U}$ U $:\equiv \top$

393   $\mathsf{Cover}\,(\mathsf{El}\,\hat{A}_0)$ $(\mathsf{El}\,\hat{A}_1)$ $:\equiv \hat{A}_0 = \hat{A}_1$

394   $\mathsf{Cover}\,(\Pi\,A_0\,B_0)\,(\Pi\,A_1\,B_1) :\equiv (A_2 : \mathsf{Cover}\,A_0\,A_1) \times \mathsf{Cover}\,((\mathsf{decode}\,A_2)_*\,B_0)\,B_1$

395   $\mathsf{Cover}\,\_$ $\_$ $:\equiv \bot$

396   The decode function is defined by double-induction on A_0 and A_1. Again, by double induction
397   on NTy, we prove that Cover is a proposition. By mutual induction on NTy, we prove that
398   Cover is reflexive and decoding this reflexivity proof gives an identity (reflexivity) path.

399   $\mathsf{reflCode} : (A : \mathsf{NTy}\,\Gamma) \to \mathsf{Cover}\,A\,A$ $\mathsf{decRefl}: (A : \mathsf{NTy}\,\Gamma) \to \mathsf{decode}\,(\mathsf{reflCode}\,A) = \mathsf{refl}$

400   We use these and J to define encode and prove that decode is a retraction:

401   $\mathsf{encode} : A_0 = A_1 \to \mathsf{Cover}\,A_0\,A_1$ $\mathsf{decEnc}: (A_2 : A_0 = A_1) \to \mathsf{decode}\,(\mathsf{encode}\,A_2) = A_2$

402   As retractions preserve homotopy levels, from $\mathsf{Cover}\,A_0\,A_1$ being a proposition, we obtain
403   that $A_0 = A_1$ is a proposition, hence $\mathsf{NTy}\,\Gamma$ is a set.   ◀

404   4.2 α-**normalisation**

405   We want to show that $\ulcorner \_ \urcorner : \mathsf{NTy}\,\Gamma \to \mathsf{Ty_G}\,\Gamma$ is a retraction, which will imply that $\mathsf{Ty_G}\,\Gamma$ is
406   a set. For this, we define the other direction which is the normalisation function and its
407   completeness.

408   ▶ **Notation 13.** For the rest of this section, as we only talk about the groupoid-syntax, we
409   don't write the G subscripts, so Ty means $\mathsf{Ty_G}$, U means $\mathsf{U_G}$, and so on.

410   ▶ **Problem 14** (α-normalisation ♺). *We define the following two functions by mutual induction*
411   *on the groupoid-syntax.*

412   $\mathsf{norm} : \mathsf{Ty}\,\Gamma \to \mathsf{NTy}\,\Gamma$ $\mathsf{compl}: (A : \mathsf{Ty}\,\Gamma) \to \ulcorner\mathsf{norm}\,A\urcorner = A$

₄₁₃ **Construction.** On U and El, the construction is trivial.

₄₁₄ $\mathsf{norm}\,\mathsf{U} :\equiv \mathsf{U}$ $\mathsf{norm}\,(\mathsf{El}\,\hat{A}) :\equiv \mathsf{El}\,\hat{A}$ $\mathsf{compl}\,\mathsf{U} :\equiv \mathsf{refl}$ $\mathsf{compl}\,(\mathsf{El}\,\hat{A}) :\equiv \mathsf{refl}$

₄₁₅ On Π, we normalise recursively, but as $\mathsf{norm}\,B : \mathsf{NTy}\,(\Gamma \rhd A)$, we need to transport it over
₄₁₆ completeness of A to obtain an $\mathsf{NTy}\,(\Gamma \rhd \ulcorner \mathsf{norm}\,A \urcorner)$:

₄₁₇ $\mathsf{norm}\,(\Pi\,A\,B) :\equiv \Pi\,(\mathsf{norm}\,A)\,\big((\mathsf{compl}\,A)_* \,\mathsf{norm}\,B\big)$

₄₁₈ $\mathsf{compl}\,(\Pi\,A\,B) : \ulcorner \mathsf{norm}\,(\Pi\,A\,B) \urcorner \equiv$

₄₁₉ $\Pi\,\ulcorner \mathsf{norm}\,A \urcorner \ulcorner (\mathsf{compl}\,A)_* \,(\mathsf{norm}\,B) \urcorner \stackrel{\mathsf{compl}\,A}{=} \Pi\,A\,\ulcorner \mathsf{norm}\,B \urcorner \stackrel{\mathsf{compl}\,B}{=} \Pi\,A\,B$

₄₂₀ To define norm on instantiated types, we need to instantiate normal forms. For this, we first
₄₂₁ show the following.

₄₂₂ ▶ **Problem 15** (♻). NTy *can be equipped with an instantiation operation* $-[-]$ *which is*
₄₂₃ *functorial, and* $\ulcorner - \urcorner$ *is a 2-natural transformation into* Ty*, as follows (note the difference in*
₄₂₄ *colours for the overloaded names).*

₄₂₅ $-[-] : \mathsf{NTy}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{NTy}\,\Delta$ $[\circ] : \forall A\,\gamma\,\delta.\,A[\gamma \circ \delta] = A[\gamma][\delta]$ $[\mathsf{id}] : \forall A.\,A[\mathsf{id}] = A$
₄₂₆ $\ulcorner \urcorner [] : \forall A\,\gamma.\,\ulcorner A \urcorner [\gamma] = \ulcorner A[\gamma] \urcorner$

₄₂₇ $\ulcorner \urcorner [\circ] : \forall A\,\gamma\,\delta.\,{}^{\ulcorner \urcorner [] \,A\,(\gamma \circ \delta)}$

$$\begin{array}{ccc}
\ulcorner A \urcorner [\gamma \circ \delta] \xrightarrow{[\circ]\,\ulcorner A \urcorner \gamma\,\delta} \ulcorner A \urcorner [\gamma][\delta] & & \ulcorner A \urcorner [\mathsf{id}] \\
\downarrow & \downarrow {\scriptstyle \ulcorner \urcorner [] \,A\,\gamma} & \\
& \ulcorner A[\gamma] \urcorner [\delta] & \ulcorner \urcorner [\mathsf{id}] : \forall A.\,{}_{\ulcorner \urcorner [] \,A\,\mathsf{id}} \downarrow \quad \searrow^{[\mathsf{id}]\,\ulcorner A \urcorner} \\
& \downarrow {\scriptstyle \ulcorner \urcorner [] \,(A[\gamma])\,\delta} & \ulcorner A[\mathsf{id}] \urcorner \xrightarrow[{[\mathsf{id}]\,A}]{} \ulcorner A \urcorner \\
\ulcorner A[\gamma \circ \delta] \urcorner \xrightarrow[{[\circ]\,A\,\gamma\,\delta}]{} \ulcorner A[\gamma][\delta] \urcorner & &
\end{array}$$

₄₂₈ **Construction for Problem 15.** Instantiation of normal types is by mutual induction with
₄₂₉ naturality of $\ulcorner - \urcorner$. Instantiating U just changes the implicit context arguments, instantiating
₄₃₀ El means instantiating the term (which is an ordinary $\mathsf{Tm_G}$ term, and is not normal),
₄₃₁ instantiating Π is recursive:

₄₃₂ $\mathsf{U}[\gamma] :\equiv \mathsf{U}$ $(\mathsf{El}\,\hat{A})[\gamma] :\equiv \mathsf{El}\,(\hat{A}[\gamma]^{\mathsf{U}})$ $(\Pi\,A\,B)[\gamma] :\equiv \Pi\,(A[\gamma])\,(B[\gamma^{\ulcorner + \urcorner}])$

₄₃₃ The operation $-[-^{\ulcorner + \urcorner}]$ used in the codomain of Π is defined as follows. It also comes with a
₄₃₄ filler equation.

₄₃₅ $-[-^{\ulcorner + \urcorner}] : \mathsf{NTy}\,(\Gamma \rhd \ulcorner A \urcorner) \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{NTy}\,(\Delta \rhd \ulcorner A[\gamma] \urcorner)$
₄₃₆ $B[\gamma^{\ulcorner + \urcorner}] :\equiv (\ulcorner \urcorner [] \,A\,\gamma)_* \,(B[\gamma^+])$
₄₃₇ $B[\gamma^{\ulcorner + \urcorner}]\mathsf{filler} : B[\gamma^+] =^{\ulcorner \urcorner [] \,A\,\gamma} B[\gamma^{\ulcorner + \urcorner}]$

₄₃₈ Analogously to $[\circ^+]$ and $[\mathsf{id}^+]$ of Definition 7, we define their "normal substitution" versions
₄₃₉ $[\circ^+]$ and $[\mathsf{id}^+]$. Naturality is reusing the substitution law of the corresponding syntactic
₄₄₀ operation, and in the case of $\Pi\,A\,B$, naturality for A and B are used (in the codomain of Π,
₄₄₁ both $-[-^{\ulcorner + \urcorner}]$ and its filler are used):

₄₄₂ $\ulcorner \urcorner [] \,\mathsf{U}\,\gamma$ $: \ulcorner \mathsf{U} \urcorner [\gamma] \equiv \mathsf{U}[\gamma] \stackrel{\mathsf{U}[]\,\gamma}{=} \mathsf{U} \equiv \ulcorner \mathsf{U}[\gamma] \urcorner$

₄₄₃ $\ulcorner \urcorner [] \,(\mathsf{El}\,\hat{A})\,\gamma$ $: \ulcorner \mathsf{El}\,\hat{A} \urcorner [\gamma] \equiv (\mathsf{El}\,\hat{A})[\gamma] \stackrel{\mathsf{El}[]\,\hat{A}\,\gamma}{=} \mathsf{El}\,(\hat{A}[\gamma]^{\mathsf{U}}) \equiv \ulcorner (\mathsf{El}\,\hat{A})[\gamma] \urcorner$

₄₄₄ $\ulcorner \urcorner [] \,(\Pi\,A\,B)\,\gamma : \ulcorner \Pi\,A\,B \urcorner [\gamma] \equiv (\Pi\,\ulcorner A \urcorner \ulcorner B \urcorner)[\gamma] \stackrel{\Pi[]\,\ulcorner A \urcorner \ulcorner B \urcorner \gamma}{=} \Pi\,(\ulcorner A \urcorner [\gamma])\,(\ulcorner B \urcorner [\gamma^+]) \stackrel{\ulcorner \urcorner [] \,B\,\gamma^+}{=}$

₄₄₅ $\Pi\,(\ulcorner A \urcorner [\gamma])\,\ulcorner B[\gamma^+] \urcorner \stackrel{\Pi\,(\ulcorner \urcorner [] \,A\,\gamma)\,\ulcorner B[\gamma^{\ulcorner + \urcorner}]\mathsf{filler} \urcorner}{=} \Pi\,\ulcorner A[\gamma] \urcorner \ulcorner B[\gamma^{\ulcorner + \urcorner}] \urcorner \equiv \ulcorner (\Pi\,A\,B)[\gamma] \urcorner$

The functoriality equation $[\circ]$ and the 2-naturality square $\ulcorner\urcorner[\circ]$ are proven mutually by induction on NTy. The composition functor law for U is definitional, for El it reuses the functor law for terms of type U, for Π it is recursive:

$[\circ]\,\mathsf{U}\,\gamma\,\delta \qquad : \mathsf{U}[\gamma\circ\delta] \equiv \mathsf{U} \equiv \mathsf{U}[\gamma][\delta]$

$[\circ]\,(\mathsf{El}\,\hat{A})\,\gamma\,\delta \quad : (\mathsf{El}\,\hat{A})[\gamma\circ\delta] \equiv \mathsf{El}\,(\hat{A}[\gamma\circ\delta]^{\mathsf{U}}) \overset{[\circ]^{\mathsf{U}}\,\hat{A}\,\gamma\,\delta}{\equiv} \mathsf{El}\,(\hat{A}[\gamma]^{\mathsf{U}}[\delta]^{\mathsf{U}}) \equiv (\mathsf{El}\,\hat{A})[\gamma][\delta]$

$[\circ]\,(\Pi\,A\,B)\,\gamma\,\delta : (\Pi\,A\,B)[\gamma\circ\delta] \equiv \Pi\,(A[\gamma\circ\delta])\,(B[(\gamma\circ\delta)^{\ulcorner+\urcorner}]) \overset{\Pi\,([\circ]\,A\,\gamma\,\delta)\,([\circ^{\ulcorner+\urcorner}]\,B\,\gamma\,\delta)}{\equiv}$

$\qquad\qquad \Pi\,(A[\gamma][\delta])\,(B[\gamma^{\ulcorner+\urcorner}][\delta^{\ulcorner+\urcorner}]) \equiv (\Pi\,A\,B)[\gamma][\delta]$

In the codomain part of the proof for Π above, we used functoriality of the $-[-^{\ulcorner+\urcorner}]$ operation which is defined by the dotted line (given by composition) in the following left square which is over the right square. We also give name to the filler of the left square.

The $\ulcorner\urcorner[\circ]$-squares for U and El are definitionally the same as $\mathsf{U}[\circ]$ and $\mathsf{El}[\circ]$, respectively. We present the diagrammatic proof of $\mathsf{U}[\circ]$ for clarity, where double line means definitional equality. In this diagram, the inner and outer squares are definitionally equal. The square for Π is more involved, we present it in Figure 2 in the Appendix.

$\ulcorner\urcorner[\circ]\,\mathsf{U}\,\gamma\,\delta :\equiv \mathsf{U}[\circ]\,\gamma\,\delta$

$\ulcorner\urcorner[\circ]\,(\mathsf{El}\,\hat{A})\,\gamma\,\delta :\equiv \mathsf{El}[\circ]\,\hat{A}\,\gamma\,\delta$, see also Figure 1

$\ulcorner\urcorner[\circ]\,(\Pi\,A\,B)\,\gamma\,\delta :$ see Figure 2

The functoriality equation $[\mathsf{id}]$ is proven by mutual induction on NTy.

$[\mathsf{id}]\,\mathsf{U} : \mathsf{U}[\mathsf{id}] \equiv \mathsf{U}$ $\qquad\qquad\qquad\qquad$ $[\mathsf{id}]\,(\mathsf{El}\,\hat{A}) : (\mathsf{El}\,\hat{A})[\mathsf{id}] \equiv \mathsf{El}\,(\hat{A}[\mathsf{id}]^{\mathsf{U}}) \overset{[\mathsf{id}]^{\mathsf{U}}\,\hat{A}}{\equiv} \mathsf{El}\,\hat{A}$

$[\mathsf{id}]\,(\Pi\,A\,B) : (\Pi\,A\,B)[\mathsf{id}] \equiv \Pi\,(A[\mathsf{id}])\,(B[\mathsf{id}^{\ulcorner+\urcorner}]) \overset{\Pi\,([\mathsf{id}]\,A)\,([\mathsf{id}^{\ulcorner+\urcorner}]\,B)}{\equiv} \Pi\,A\,B$

In the codomain part of the proof for Π above, we used functoriality of the $-[-^{\ulcorner+\urcorner}]$ operation which is defined by the dotted line in the following upper triangle which is over the lower

469   triangle. We also give a name to the filler of the upper triangle.

471   The 2-naturality triangle $\ulcorner\urcorner[\mathsf{id}]$ is proven by induction on NTy as follows:

472      $\ulcorner\urcorner[\mathsf{id}]\ \mathsf{U} :\equiv \mathsf{U}[\mathsf{id}]$ $\ulcorner\urcorner[\mathsf{id}]\ (\mathsf{El}\ \hat{A}):\equiv \mathsf{El}[\mathsf{id}]\ \hat{A}$ $\ulcorner\urcorner[\mathsf{id}]\ (\Pi\ A\ B)$: see Figure 3

473   This finishes the construction for Problem 15.   ◀

474   So far, we defined norm and compl on U, El and Π. On substituted types, we define
475   normalisation and its completeness as follows.

476      $\mathsf{norm}\ (A[\gamma]) :\equiv (\mathsf{norm}\ A)[\gamma]$

477      $\mathsf{compl}\ (A[\gamma]) : \ulcorner\mathsf{norm}\ (A[\gamma])\urcorner \equiv \ulcorner(\mathsf{norm}\ A)[\gamma]\urcorner \overset{\ulcorner\urcorner[] \ (\mathsf{norm}\ A)\ \gamma}{=} \ulcorner\mathsf{norm}\ A\urcorner[\gamma] \overset{\mathsf{compl}\ A}{=} A[\gamma]$

478   The action of norm on the functor laws is the corresponding functor law for instantiation
479   of normal types, i.e. $\mathsf{norm}\ ([\circ]\ A\ \gamma\ \delta) :\equiv [\circ]\ (\mathsf{norm}\ A)\ \gamma\ \delta$ and $\mathsf{norm}\ ([\mathsf{id}]\ A) :\equiv [\mathsf{id}]\ (\mathsf{norm}\ A)$.
480   Completeness for the functor laws is the filling of the following two squares:

483   The action of norm on the substitution laws for U and El is given by refl, and compl is given
484   by trivial fillers for degenerate squares. The actions of norm and compl on $\Pi[]\ A\ B\ \gamma$ only
485   involve naturality squares and fillers, they are presented in Figures 4 and 5 in the appendix.
486   The rest of the Ty-paths that norm and compl have to preserve are the 2-paths $\mathsf{U}[\circ]$, $\mathsf{U}[\mathsf{id}]$,
487    $\mathsf{El}[\circ]$, $\mathsf{El}[\mathsf{id}]$, $\Pi[\circ]$, $\Pi[\mathsf{id}]$. As norm returns in a set, these are all trivial. The function compl
488   produces an equality between elements of Ty, and as Ty is a groupoid, it trivially preserves
489   2-paths. Having defined norm and compl, we finished the construction for Problem 14.   ◀

490   ▶ **Theorem 16** (♡). $\mathsf{Ty_G}$ *is a set.*

491   **Proof.** Together, norm and compl witness that $\ulcorner - \urcorner$ is a retraction, which preserves h-levels:
492   as NTy is a set, so is Ty.   ◀

493   ▶ Remark 1. We also have stability of normalisation, but we don't need it in this paper.

5   Reaping the fruits

495   ▶ **Problem 17** (♡). *The set-syntax is isomorphic to the groupoid-syntax.*

Construction. In Construction 10, we defined the map from the groupoid-syntax to the set-syntax. Now we define the opposite direction using that $\mathsf{Ty_G}$ is a set. The roundtrips are proven by two simple inductions. ◀

▶ **Construction 18** (Set interpretation of the set-syntax ↺)**.** We compose the groupoid-interpretation of the set syntax (Problem 17) and the set interpretation of the groupoid-syntax (Construction 9).

Groupoid CwFs are essentially algebras of the substitution calculus part of the groupoid-syntax (Definition 7), but we also include three coherence laws for types (the pentagon law [ass] and two identity triangles).

▶ **Definition 19** (Groupoid CwF, GCwF ↺)**.** An Ehrhard-style groupoid CwF is a 1-category (objects named $\mathsf{Con} : \mathsf{Type}$, morphisms $\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$), a 2-presheaf of types (given by $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Groupoid}$, $-[-] : \mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta$, $[\circ] : A[\gamma \circ \delta] = A[\gamma][\delta]$, $[\mathsf{id}] : A[\mathsf{id}] = A$, $[\mathsf{ass}]$, $[\mathsf{idl}]$, $[\mathsf{idr}]$ as depicted below), a dependent presheaf of terms over types ($\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}$, with instantiation and functor laws), with Ehrhard-style comprehension (operations $- \triangleright -$, $-^+$, p, q, $\langle - \rangle$ with 8 equations as in Definition 2).

$[\mathsf{ass}] : \forall A\,\gamma\,\delta\,\theta.$ $[\mathsf{idl}] : \forall A\,\gamma.$ $[\mathsf{idr}] : \forall A\,\gamma.$

$$
\begin{array}{ccc}
A[\gamma \circ (\delta \circ \theta)] & \xrightarrow{\mathsf{ass}\,\gamma\,\delta\,\theta} & A[(\gamma \circ \delta) \circ \theta] \\
{\scriptstyle [\circ]\,A\,\gamma\,(\delta\circ\theta)} \Big\downarrow & & \Big\downarrow {\scriptstyle [\circ]\,A\,(\gamma\circ\delta)\,\theta} \\
& & A[\gamma \circ \delta][\theta] \\
& & \Big\downarrow {\scriptstyle [\circ]\,A\,\gamma\,\delta} \\
A[\gamma][\delta \circ \theta] & \xrightarrow[{\scriptstyle [\circ]\,(A[\gamma])\,\delta\,\theta}]{} & A[\gamma][\delta][\theta]
\end{array}
$$

$$
\begin{array}{ccc}
A[\mathsf{id} \circ \gamma] & \xrightarrow{\mathsf{idl}\,\gamma} & \\
{\scriptstyle [\circ]\,A\,\mathsf{id}\,\gamma}\Big\downarrow & \searrow & \\
A[\mathsf{id}][\gamma] & \xrightarrow[{\scriptstyle [\mathsf{id}]\,A}]{} & A[\gamma]
\end{array}
\qquad
\begin{array}{ccc}
A[\gamma \circ \mathsf{id}] & \xrightarrow{\mathsf{idr}\,\gamma} & \\
{\scriptstyle [\circ]\,A\,\gamma\,\mathsf{id}}\Big\downarrow & \searrow & \\
A[\gamma][\mathsf{id}] & \xrightarrow[{\scriptstyle [\mathsf{id}]\,(A[\gamma])}]{} & A[\gamma]
\end{array}
$$

▶ **Remark 2** (↺)**.** In any groupoid CwF, [idl] and [idr] are interderivable. The direction [idl] $\to$ [idr] is described in Figure 6 in the appendix. The same proof in the context of monoidal categories appears in [28, Theorem 7].

▶ **Proposition 20** (↺)**.** *In the groupoid-syntax (Definition 7), the laws* [ass]*,* [idl] *and* [idr] *are admissible.*

Proof. Direct consequence of Theorem 16. ◀

6   Conclusions

We have presented a basic coherence theorem for **GCwF**, enabling the interpretation of the usual decidable intrinsic syntax of type theory within models based on categories where the objects do not form a set, such as the set model. Notably, we have achieved this without relying on normalisation for the groupoid syntax or invoking Hedberg's theorem. Furthermore, our method is adaptable, in principle, to type theories without decidable equality. An interesting feature of our approach is that it eliminates the need to explicitly incorporate the usual coherence laws for 2-categories (such as the pentagon law) into the syntax; these laws are admissible in our groupoid-syntax.

Despite these advancements, several significant challenges remain. For instance, we aim to extend this framework to include a univalent universe of propositions (i.e. Prop with propositional extensionality). We also seek to address univalence for types without

introducing an additional universe, thereby demonstrating that univalence can be soundly supported in this setting.

The addition of universes, even a minimal one such as a universe of Booleans with large eliminations, would require a shift in our methodology and might necessitate term normalisation. Extending the framework to accommodate multiple universes would inevitably demand a move to higher dimensions, introducing further complexity.

Our groupoid-syntax can be seen as the GCwF with Π freely generated from a set and a family over it. We would like to support more interesting generating data, i.e. generating data which can refer to the GCwF structure while being defined.

Finally, we would like to revisit the longstanding problem of modeling semi-simplicial types within this context. One potential direction is to extend our current recursive treatment of substitution and substitution-related coherence laws, using these as a foundation to systematically derive higher coherence conditions.

References

1 Andreas Abel, Joakim Öhman, and Andrea Vezzosi. Decidability of conversion for type theory in type theory. *Proc. ACM Program. Lang.*, 2(POPL):23:1–23:29, 2018. `doi:10.1145/3158111`.

2 Arthur Adjedj, Meven Lennon-Bertrand, Kenji Maillard, Pierre-Marie Pédrot, and Loïc Pujet. Martin-Löf à la Coq. In Amin Timany, Dmitriy Traytel, Brigitte Pientka, and Sandrine Blazy, editors, *Proceedings of the 13th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2024, London, UK, January 15-16, 2024*, pages 230–245. ACM, 2024. `doi:10.1145/3636501.3636951`.

3 Benedikt Ahrens, Paige Randall North, and Niels van der Weide. Bicategorical type theory: semantics and syntax. *Math. Struct. Comput. Sci.*, 33(10):868–912, 2023. `doi:10.1017/S0960129523000312`.

4 Thorsten Altenkirch, Paolo Capriotti, and Nicolai Kraus. Extending homotopy type theory with strict equality. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic, CSL 2016, August 29 - September 1, 2016, Marseille, France*, volume 62 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. `doi:10.4230/LIPICS.CSL.2016.21`.

5 Thorsten Altenkirch, Nils Anders Danielsson, and Nicolai Kraus. Partiality, revisited - the partiality monad as a quotient inductive-inductive type. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 534–549, 2017. `doi:10.1007/978-3-662-54458-7\_31`.

6 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 18–29. ACM, 2016. `doi:10.1145/2837614.2837638`.

7 Thorsten Altenkirch and Ambrus Kaposi. Normalisation by evaluation for type theory, in type theory. *Logical methods in computer science*, 13, 2017.

8 Thorsten Altenkirch and Ambrus Kaposi. A container model of type theory. In Henning Basold, editor, *27th International Conference on Types for Proofs and Programs, TYPES 2021*. Universiteit Leiden, 2021. URL: `https://types21.liacs.nl/download/a-container-model-of-type-theory/`.

9 Thorsten Altenkirch and Luis Scoccola. The integers as a higher inductive type. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual*

ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020, pages 67–73. ACM, 2020. `doi:10.1145/3373718.3394760`.

10 Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *Math. Struct. Comput. Sci.*, 33(8):688–743, 2023. `doi:10.1017/S0960129523000130`.

11 Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications - ERRATUM. *Math. Struct. Comput. Sci.*, 34(1):80, 2024. `doi:10.1017/S096012952300021X`.

12 Marc Bezem, Ulrik Buchholtz, Pierre Cagne, Bjørn Ian Dundas, and Daniel R. Grayson. Symmetry. `https://github.com/UniMath/SymmetryBook`. Commit: `ec9be72`.

13 Rafaël Bocquet. Strictification of weakly stable type-theoretic structures using generic contexts. In Henning Basold, Jesper Cockx, and Silvia Ghilezan, editors, *27th International Conference on Types for Proofs and Programs, TYPES 2021, June 14-18, 2021, Leiden, The Netherlands (Virtual Conference)*, volume 239 of *LIPIcs*, pages 3:1–3:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPICS.TYPES.2021.3`.

14 Simon Castellan, Pierre Clairambault, and Peter Dybjer. *Categories with Families: Unityped, Simply Typed, and Dependently Typed*, pages 135–180. Springer International Publishing, Cham, 2021. `doi:10.1007/978-3-030-66545-6_5`.

15 Evan Cavallo and Robert Harper. Higher inductive types in cubical computational type theory. *Proc. ACM Program. Lang.*, 3(POPL):1:1–1:27, 2019. `doi:10.1145/3290314`.

16 James Chapman. Type theory should eat itself. In Andreas Abel and Christian Urban, editors, *Proceedings of the International Workshop on Logical Frameworks and Metalanguages: Theory and Practice, LFMTP@LICS 2008, Pittsburgh, PA, USA, June 23, 2008*, volume 228 of *Electronic Notes in Theoretical Computer Science*, pages 21–36. Elsevier, 2008. `doi:10.1016/J.ENTCS.2008.12.114`.

17 Pierre Clairambault and Peter Dybjer. The biequivalence of locally cartesian closed categories and martin-löf type theories. *Math. Struct. Comput. Sci.*, 24(6), 2014. `doi:10.1017/S0960129513000881`.

18 Thierry Coquand. Generalised algebraic presentation of type theory. `https://www.cse.chalmers.se/~coquand/cwf2.pdf`, 2020.

19 Thierry Coquand, Simon Huber, and Anders Mörtberg. On higher inductive types in cubical type theory. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 255–264. ACM, 2018. `doi:10.1145/3209108.3209197`.

20 Nils Anders Danielsson. A formalisation of a dependently typed language as an inductive-recursive family. In Thorsten Altenkirch and Conor McBride, editors, *Types for Proofs and Programs, International Workshop, TYPES 2006, Nottingham, UK, April 18-21, 2006, Revised Selected Papers*, volume 4502 of *Lecture Notes in Computer Science*, pages 93–109. Springer, 2006. `doi:10.1007/978-3-540-74464-1\_7`.

21 Tim Lukas Diezel and Sergey Goncharov. Towards constructive hybrid semantics. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 24:1–24:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPICS.FSCD.2020.24`.

22 Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*, volume 1158 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 1995. `doi:10.1007/3-540-61780-9\_66`.

23 Thomas Ehrhard. *Une sémantique catégorique des types dépendents*. PhD thesis, Université Paris VII, 1988.

24 Martin Hofmann. On the interpretation of type theory in locally cartesian closed categories. In Leszek Pacholski and Jerzy Tiuryn, editors, *Computer Science Logic, 8th International*

632   *Workshop, CSL '94, Kazimierz, Poland, September 25-30, 1994, Selected Papers*, volume 933 of
633   *Lecture Notes in Computer Science*, pages 427–441. Springer, 1994. `doi:10.1007/BFB0022273`.

25   Ambrus Kaposi. Re: Separate definition of constructors?, May 2019. Email message to the
635   Agda mailing list. URL: `https://lists.chalmers.se/pipermail/agda/2019/011176.html`.

26   Ambrus Kaposi and András Kovács. Signatures and induction principles for higher inductive-
637   inductive types. *Log. Methods Comput. Sci.*, 16(1), 2020. `doi:10.23638/LMCS-16(1:10)2020`.

27   Ambrus Kaposi and Jakob von Raumer. A syntax for mutual inductive families. In Zena M.
639   Ariola, editor, *5th International Conference on Formal Structures for Computation and
640   Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume
641   167 of *LIPIcs*, pages 23:1–23:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
642   `doi:10.4230/LIPICS.FSCD.2020.23`.

28   G. M. Kelly. On MacLane's conditions for coherence of natural associativities, commutativities,
644   etc. *Journal of Algebra*, 1(4):397–402, 1964. `doi:10.1016/0021-8693(64)90018-3`.

29   Nicolai Kraus. Internal ∞-categorical models of dependent type theory: Towards 2LTT eating
646   HoTT. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021,
647   Rome, Italy, June 29 - July 2, 2021*, pages 1–14. IEEE, 2021. `doi:10.1109/LICS52264.2021.`
648   `9470667`.

30   Nicolai Kraus and Thorsten Altenkirch. Free higher groups in homotopy type theory. In Anuj
650   Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on
651   Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 599–608. ACM,
652   2018. `doi:10.1145/3209108.3209183`.

31   Peter LeFanu Lumsdaine and Michael A. Warren. The local universes model: An overlooked
654   coherence construction for dependent type theories. *ACM Trans. Comput. Log.*, 16(3):23:1–
655   23:31, 2015. `doi:10.1145/2754931`.

32   Stefano Piceghello. Coherence for Monoidal Groupoids in HoTT. In Marc Bezem and As-
657   sia Mahboubi, editors, *25th International Conference on Types for Proofs and Programs
658   (TYPES 2019)*, volume 175 of *Leibniz International Proceedings in Informatics (LIPIcs)*,
659   pages 8:1–8:20, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Inform-
660   atik. URL: `https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.TYPES.2019.`
661   `8`, `doi:10.4230/LIPIcs.TYPES.2019.8`.

33   Mike Shulman. Homotopy type theory should eat itself (but so far, it's too big to swallow),
663   2014. Blog post on the Homotopy Type Theory website. URL: `https://homotopytypetheory.`
664   `org/2014/03/03/hott-should-eat-itself/`.

34   Taichi Uemura. Normalization and coherence for ∞-type theories. *CoRR*, abs/2212.11764,
666   2022. `arXiv:2212.11764`, `doi:10.48550/ARXIV.2212.11764`.

35   The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of
668   Mathematics*. `https://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

36   Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical Agda: A dependently typed
670   programming language with univalence and higher inductive types. *J. Funct. Program.*, 31:e8,
671   2021. `doi:10.1017/S0956796821000034`.

37   David Wärn. Path spaces of pushouts, 2024. URL: `https://arxiv.org/abs/2402.12339`,
673   `arXiv:2402.12339`.

A   More diagrams
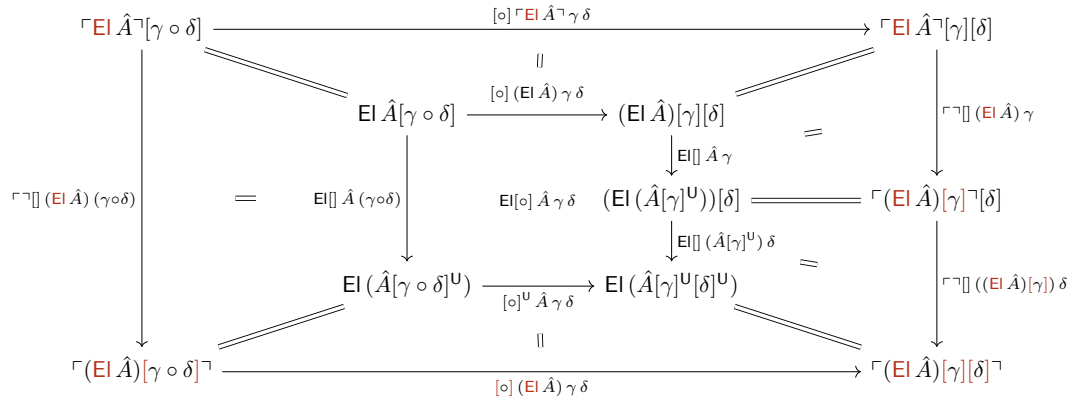
Figure 1 This diagram is the proof $\ulcorner\urcorner[\circ]\,(\mathsf{El}\,\hat{A})\,\gamma\,\delta$, which is the outer square. Double lines mean definitional equality. The boundaries of the outer square are definitionally equal to the boundaries of the inner square, which we fill by $\mathsf{El}[\circ]\,\hat{A}\,\gamma\,\delta$.
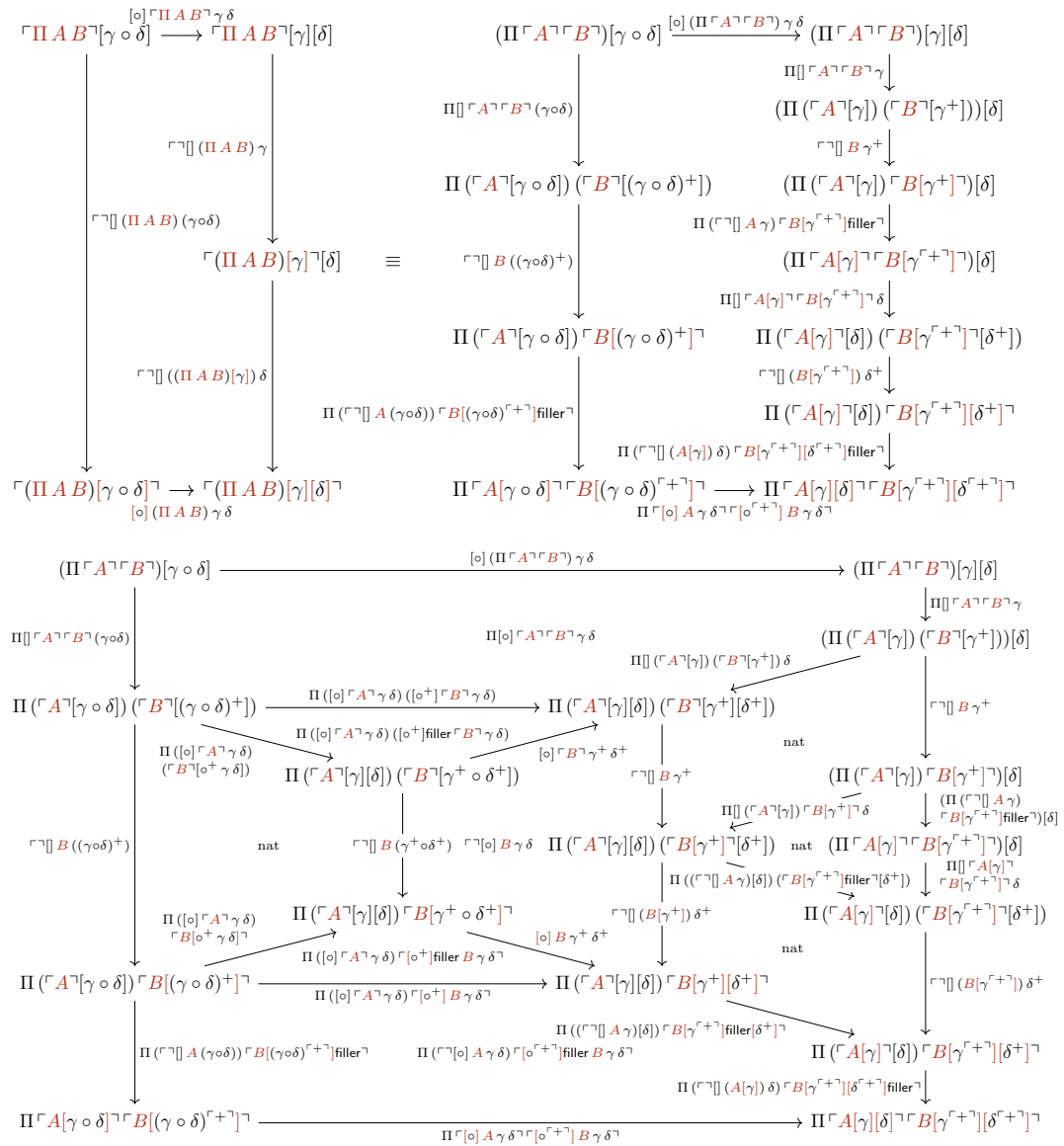
Figure 2 This diagram is the proof $\ulcorner\urcorner[\circ]\,(\Pi\,A\,B)\,\gamma\,\delta$. In the upper part, we compute the square to be filled: the left hand side square is definitionally equal to the right hand side one. Then, we fill the right hand side square in the lower diagram, where the boundary of the square is the same as the upper right hand side square.
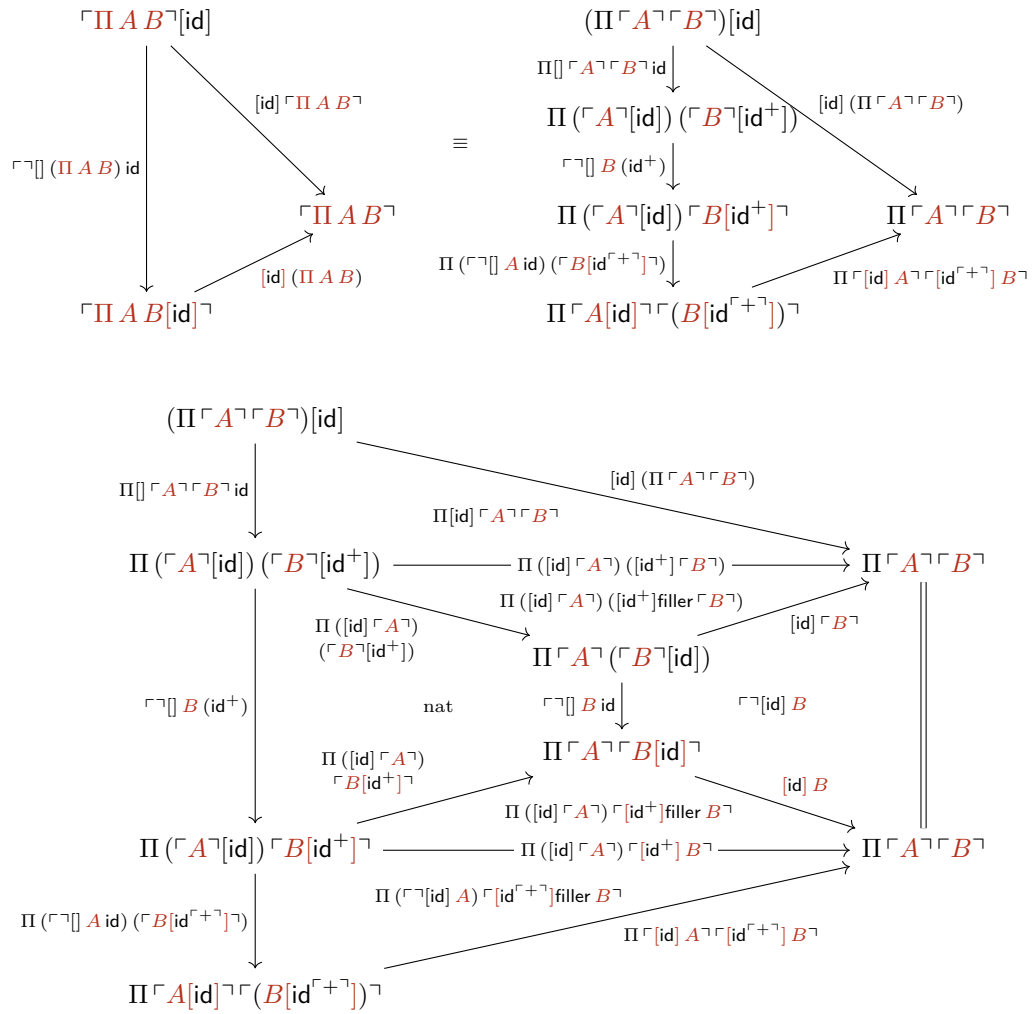
Figure 3 This diagram is the proof $\ulcorner\urcorner[\text{id}]\,(\Pi\,A\,B)$. In the upper part, we compute the triangle to be filled: the left hand side triangle is definitionally equal to the right hand side one. Then, we fill the right hand side triangle in the lower diagram, where we duplicate the vertex $\Pi\ulcorner A\urcorner\ulcorner B\urcorner$ for readability.

$$((\text{compl } A)_* (\text{norm } B))[\gamma^+] \xrightarrow{\text{transportFiller}} (\text{norm } B)[\gamma^+]$$

$$((\text{compl } A)_* (\text{norm } B))[\gamma^{\ulcorner + \urcorner}]\text{filler} \Big\downarrow \qquad\qquad \text{fillerOf } e \qquad\qquad \Big\downarrow \text{transportFiller}$$

$$((\text{compl } A)_* (\text{norm } B))[\gamma^{\ulcorner + \urcorner}] \xdashrightarrow{\ e\ } (\text{compl } (A[\gamma]))_* ((\text{norm } B)[\gamma^+])$$

$$\ulcorner\text{norm } A\urcorner[\gamma] \xrightarrow{\qquad\text{compl } A\qquad} A[\gamma]$$

$$\ulcorner\urcorner[] \,(\text{norm } A)\, \gamma \Big\downarrow \qquad\qquad \text{fillerOf } (\text{compl } (A[\gamma])) \qquad\qquad \Big\downarrow \text{compl } (A[\gamma])$$

$$\ulcorner(\text{norm } A)[\gamma]\urcorner =\!=\!=\!=\!=\!=\!=\!= \ulcorner(\text{norm } A)[\gamma]\urcorner$$

■ **Figure 4** Normalisation on the substitution law for Π acts as follows: $\text{norm}\,(\Pi[]\,A\,B\,\gamma) :\equiv \Pi\,\text{refl}\,e$ where e is defined in the upper square in this diagram. The upper square is a dependent square over the lower one.

$$\Pi\,\ulcorner(\text{norm } A)[\gamma]\urcorner\,\ulcorner(((\text{compl } A)_* (\text{norm } B))[\gamma^{\ulcorner + \urcorner}])\urcorner \xrightarrow{\ e\ } \Pi\,\ulcorner\text{norm } A[\gamma]\urcorner\,\ulcorner(\text{compl } (A[\gamma]))_*(\text{norm } B[\gamma^+])\urcorner$$

$$\Pi\,(\ulcorner\urcorner[]\,(\text{norm } A)\,\gamma)$$
$$\ulcorner((\text{compl } A)_* (\text{norm } B))[\gamma^{\ulcorner + \urcorner}]\text{filler}\urcorner \Big\downarrow \qquad \Pi\,(\text{fillerOf } (\text{compl } (A[\gamma])))\,(\text{fillerOf } e) \qquad \Big\downarrow \Pi\,(\text{compl } (A[\gamma]))\,\text{transportFiller}$$

$$\Pi\,(\ulcorner\text{norm } A\urcorner[\gamma])\,\ulcorner((\text{compl } A)_* (\text{norm } B))[\gamma^+]\urcorner \xrightarrow{\Pi\,(\text{compl } A)\,\text{transportFiller}} \Pi\,(A[\gamma])\,\ulcorner(\text{norm } B)[\gamma^+]\urcorner$$

$$\ulcorner\urcorner[]\,((\text{compl } A)_* (\text{norm } B))\,\gamma^+ \Big\downarrow \qquad\qquad\qquad \text{nat}$$

$$\Pi\,(\ulcorner\text{norm } A\urcorner[\gamma])\,(\ulcorner(\text{compl } A)_* (\text{norm } B)\urcorner[\gamma^+]) \qquad\qquad \Big\downarrow \ulcorner\urcorner[]\,(\text{norm } B)\,\gamma^+$$

$$\Pi[]\,\ulcorner\text{norm } A\urcorner\,\ulcorner(\text{compl } A)_* (\text{norm } B)\urcorner\,\gamma \Big\downarrow \qquad\qquad \Pi\,(\text{compl } A)\,\text{transportFiller}$$

$$(\Pi\,\ulcorner\text{norm } A\urcorner\,\ulcorner(\text{compl } A)_* (\text{norm } B)\urcorner)[\gamma] \qquad \text{nat} \qquad \Pi\,(A[\gamma])\,(\ulcorner\text{norm } B\urcorner[\gamma^+])$$

$$\Pi\,(\text{compl } A)\,\text{transportFiller} \Big\downarrow \qquad\qquad \Pi[]\,A\,\ulcorner\text{norm } B\urcorner\,\gamma$$

$$(\Pi\,A\,\ulcorner\text{norm } B\urcorner)[\gamma] \qquad\qquad\qquad \Big\downarrow \text{compl } B$$

$$\text{compl } B \Big\downarrow \qquad\qquad \text{nat}$$

$$(\Pi\,A\,B)[\gamma] \xrightarrow{\qquad\Pi[]\,A\,B\,\gamma\qquad} \Pi\,(A[\gamma])\,(B[\gamma^+])$$

■ **Figure 5** This diagram is the proof $\text{compl}\,(\Pi[]\,A\,B\,\gamma)$. The line e is defined in Figure 4.
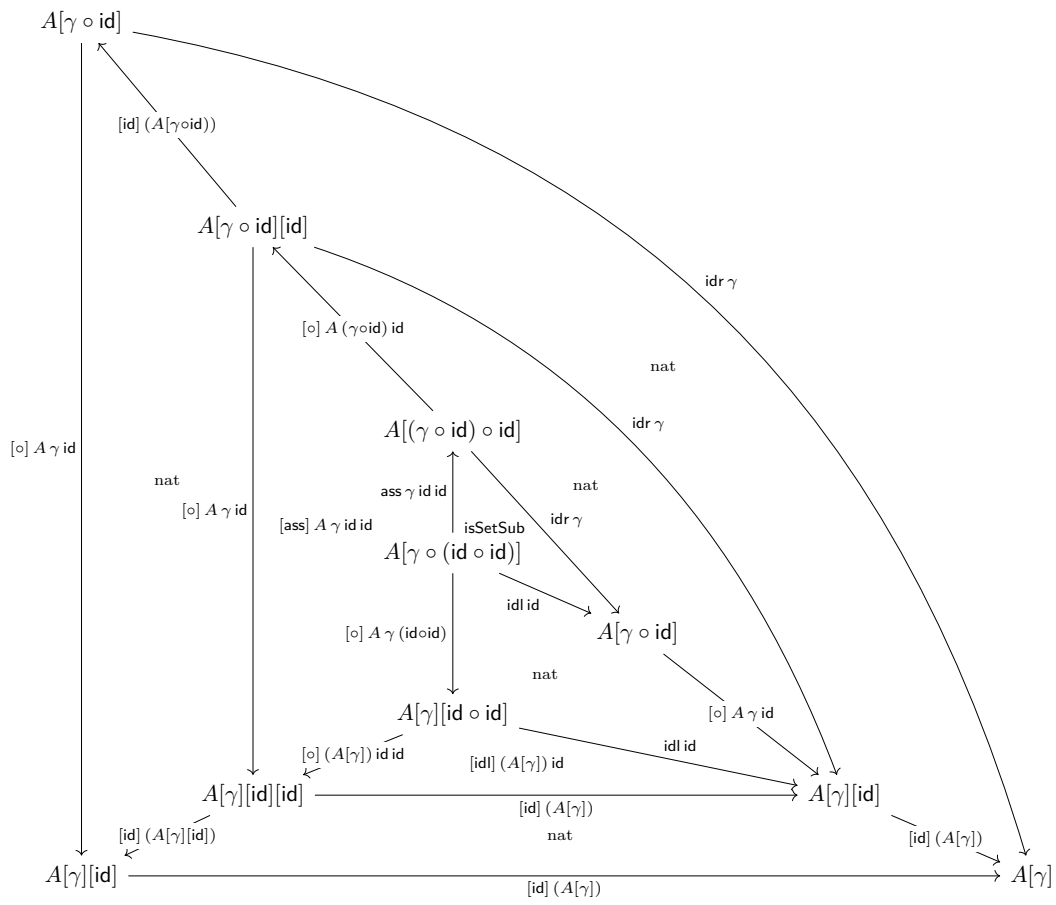
Figure 6 Proof that [idl] implies [idr] in any groupoid CwF (Definition 19).