Coherence for Segal types via the Leibniz adjunction

Tom de Jong, Nicolai Kraus, Axel Ljungström

University of Nottingham, UK

Dagstuhl Seminar 26121

Proof Systems in Actual Practice: Reasoning and Computation

20 March 2026

Introduction & first main result (informally)

- Riehl and Shulman introduced simplicial type theory (STT) as a synthetic framework for higher categories.[1]
- In homotopy type theory (HoTT): types behave like ∞-groupoids. Therefore, STT additionally equips types with a notion of directed morphisms, via an interval.

[1] Emily Riehl and Michael Shulman (2017). 'A type theory for synthetic ∞-categories'. In: *Higher Structures* 1.1, pp. 147–224. DOI: 10.21136/hs.2017.06.

Introduction & first main result (informally)

- Riehl and Shulman introduced simplicial type theory (STT) as a synthetic framework for higher categories.[1]
- In homotopy type theory (HoTT): types behave like ∞-groupoids. Therefore, STT additionally equips types with a notion of directed morphisms, via an interval.
- Segal types are those types in which such morphisms compose.
- Composition should be associative, identities should be neutral w.r.t. composition, and higher coherences should be satisfied, e.g. the different ways of rewriting $\mathrm{id} \circ (f \circ \mathrm{id})$ to f should be the same, up to a higher morphism.

[1] Emily Riehl and Michael Shulman (2017). 'A type theory for synthetic ∞-categories'. In: *Higher Structures* 1.1, pp. 147–224. DOI: 10.21136/hs.2017.06.

Introduction & first main result (informally)

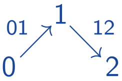
- Riehl and Shulman introduced simplicial type theory (STT) as a synthetic framework for higher categories.[1]
- In homotopy type theory (HoTT): types behave like ∞-groupoids. Therefore, STT additionally equips types with a notion of directed morphisms, via an interval.
- Segal types are those types in which such morphisms compose.
- Composition should be associative, identities should be neutral w.r.t. composition, and higher coherences should be satisfied, e.g. the different ways of rewriting $\mathrm{id} \circ (f \circ \mathrm{id})$ to f should be the same, up to a higher morphism.

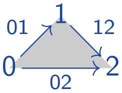
🏁 **For Segal types, all higher coherences can be derived.**

[1] Emily Riehl and Michael Shulman (2017). 'A type theory for synthetic ∞-categories'. In: *Higher Structures* 1.1, pp. 147–224. DOI: 10.21136/hs.2017.06.

First main result (more precisely)

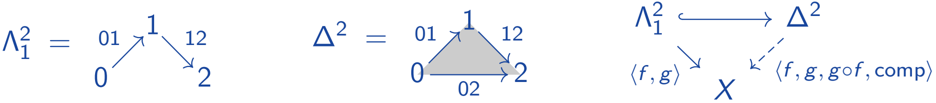
- Simplicial combinatorics: composition in X is given by extending along the horn inclusion $\Lambda^2_1 \hookrightarrow \Delta^2$.

$\Lambda^2_1 =$ 

$\Delta^2 =$ 

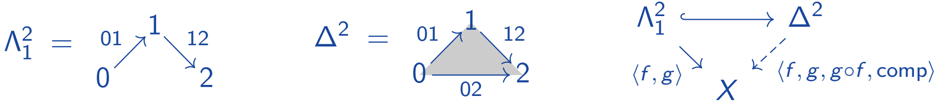
First main result (more precisely)

- Simplicial combinatorics: composition in X is given by extending along the horn inclusion $\Lambda^2_1 \hookrightarrow \Delta^2$.



First main result (more precisely)

- Simplicial combinatorics: composition in X is given by extending along the horn inclusion $\Lambda^2_1 \hookrightarrow \Delta^2$.

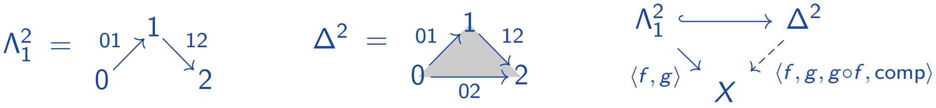


Def. A type X is a Segal type if every map $\Lambda^2_1 \to X$ has a unique (in the contractible/HoTT sense) extension to a map $\Delta^2 \to X$.

We also say that a Segal type has unique fillers for (2,1)-horns.

First main result (more precisely)

- Simplicial combinatorics: composition in X is given by extending along the horn inclusion $\Lambda_1^2 \hookrightarrow \Delta^2$.



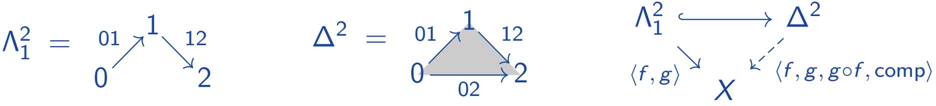
Def. A type X is a Segal type if every map $\Lambda_1^2 \to X$ has a unique (in the contractible/HoTT sense) extension to a map $\Delta^2 \to X$.

We also say that a Segal type has unique fillers for (2,1)-horns.

- We can similarly consider higher dimensional simplices Δ^n and horns Λ_k^n which encode higher coherences.

First main result (more precisely)

- Simplicial combinatorics: composition in X is given by extending along the horn inclusion $\Lambda^2_1 \hookrightarrow \Delta^2$.



Def. A type X is a Segal type if every map $\Lambda^2_1 \to X$ has a unique (in the contractible/HoTT sense) extension to a map $\Delta^2 \to X$.

We also say that a Segal type has unique fillers for (2,1)-horns.

- We can similarly consider higher dimensional simplices Δ^n and horns Λ^n_k which encode higher coherences.

Thm. Any Segal type has unique fillers for all (n,k)-horns where $0 < k < n$.

- This generalizes a result of Riehl & Shulman for $n = 3$ and $k \in \{1, 2\}$, and is a STT version of a result by Lurie.

Foundational setup

- Instead of simplicial type theory as introduced by Riehl & Shulman, we follow Gratzer, Weinberger and Buchholtz[2] and work in an axiomatic extension of HoTT:

$$\text{HoTT} + \text{bounded distributive lattice } (\mathbb{I}, 0, 1, \wedge, \vee).$$

[2] Daniel Gratzer, Jonathan Weinberger and Ulrik Buchholtz (2024). 'Directed univalence in simplicial homotopy type theory'. arXiv: 2407.09146 [cs.LO].

Foundational setup

- Instead of simplicial type theory as introduced by Riehl & Shulman, we follow Gratzer, Weinberger and Buchholtz[2] and work in an axiomatic extension of HoTT:

$$\text{HoTT} + \text{bounded distributive lattice } (I, 0, 1, \wedge, \vee).$$

- Use I to construct the horns Λ^n_k and simplices Δ^n.

[2] Daniel Gratzer, Jonathan Weinberger and Ulrik Buchholtz (2024). 'Directed univalence in simplicial homotopy type theory'. arXiv: 2407.09146 [cs.LO].

Foundational setup

- Instead of simplicial type theory as introduced by Riehl & Shulman, we follow Gratzer, Weinberger and Buchholtz[2] and work in an axiomatic extension of HoTT:

$$\text{HoTT} + \text{bounded distributive lattice } (I, 0, 1, \wedge, \vee).$$

- Use I to construct the horns Λ^n_k and simplices Δ^n.

- David Wärn (Feb. 2026): any semi-lattice is automatically a set!

  https://martinescardo.github.io/TypeTopology/gist.ThereAreNoHigherSemilattices2.html

[2] Daniel Gratzer, Jonathan Weinberger and Ulrik Buchholtz (2024). 'Directed univalence in simplicial homotopy type theory'. arXiv: 2407.09146 [cs.LO].

Reduction to the Leibniz adjunction

- Segal types can be characterized by orthogonality

$$\begin{array}{ccc} A & \longrightarrow & X \\ {\scriptstyle f}\downarrow & \overset{\exists!}{\nearrow} & \downarrow{\scriptstyle g} \\ B & \longrightarrow & Y \end{array} \qquad (f \perp g)$$

Reduction to the Leibniz adjunction

- Segal types can be characterized by orthogonality

$$\begin{array}{ccc} A & \longrightarrow & X \\ {\scriptstyle f}\downarrow & \overset{\exists!}{\nearrow} & \downarrow{\scriptstyle g} \\ B & \longrightarrow & Y \end{array} \quad (f \perp g) \qquad \rightsquigarrow \qquad \begin{array}{ccc} \Lambda^2_1 & \longrightarrow & X \\ \hookdownarrow{\scriptstyle \iota} & \overset{\exists!}{\nearrow} & \downarrow{\scriptstyle t_X} \\ \Delta^2 & \longrightarrow & \mathbf{1} \end{array} \quad (X \text{ Segal})$$

Reduction to the Leibniz adjunction

- Segal types can be characterized by orthogonality

$$
\begin{array}{ccc}
A & \longrightarrow & X \\
{\scriptstyle f}\downarrow & \overset{\exists!}{\nearrow} & \downarrow{\scriptstyle g} \\
B & \longrightarrow & Y
\end{array}
\quad (f \perp g)
\qquad \rightsquigarrow \qquad
\begin{array}{ccc}
\Lambda^2_1 & \longrightarrow & X \\
\downarrow{\scriptstyle \iota} & \overset{\exists!}{\nearrow} & \downarrow{\scriptstyle t_X} \\
\Delta^2 & \longrightarrow & \mathbf{1}
\end{array}
\quad (X \text{ Segal})
$$

- General fact: $f \perp g$ if and only if their pullback-hom $f \pitchfork g$ is an equivalence. (Definition on the next slide.)

Reduction to the Leibniz adjunction

- Segal types can be characterized by orthogonality

$$\begin{array}{ccc} A & \longrightarrow & X \\ {\scriptstyle f}\downarrow & \exists! \nearrow & \downarrow{\scriptstyle g} \\ B & \longrightarrow & Y \end{array} \quad (f \perp g) \qquad \rightsquigarrow \qquad \begin{array}{ccc} \Lambda^2_1 & \longrightarrow & X \\ \downarrow{\scriptstyle \iota} & \exists! \nearrow & \downarrow{\scriptstyle t_X} \\ \Delta^2 & \longrightarrow & \mathbf{1} \end{array} \quad (X \text{ Segal})$$

- General fact: $f \perp g$ if and only if their pullback-hom $f \pitchfork g$ is an equivalence. (Definition on the next slide.)
- For 1-categories: the pullback-hom is right adjoint to the pushout-product $f \mathbin{\widehat{\times}} g$. Sometimes known as the Leibniz adjunction.

Reduction to the Leibniz adjunction

- Segal types can be characterized by orthogonality

$$\begin{array}{ccc} A & \longrightarrow & X \\ {\scriptstyle f}\downarrow & \exists! \nearrow & \downarrow{\scriptstyle g} \\ B & \longrightarrow & Y \end{array} \quad (f \perp g) \qquad \rightsquigarrow \qquad \begin{array}{ccc} \Lambda^2_1 & \longrightarrow & X \\ \downarrow{\scriptstyle \iota} & \exists! \nearrow & \downarrow{\scriptstyle t_X} \\ \Delta^2 & \longrightarrow & \mathbf{1} \end{array} \quad (X \text{ Segal})$$

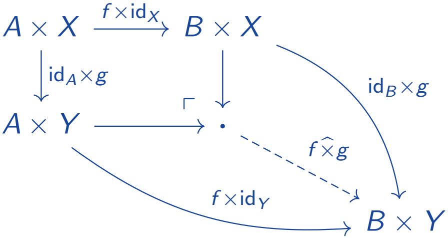
- General fact: $f \perp g$ if and only if their pullback-hom $f \pitchfork g$ is an equivalence.
  (Definition on the next slide.)
- For 1-categories: the pullback-hom is right adjoint to the pushout-product $f \mathbin{\widehat{\times}} g$.
  Sometimes known as the Leibniz adjunction.

What we did:

- Reduce simplicial comb. to formal arguments about $\pitchfork$ and $\widehat{\times}$ as much as possible.
- Prove the Leibniz adjunction for maps in HoTT.

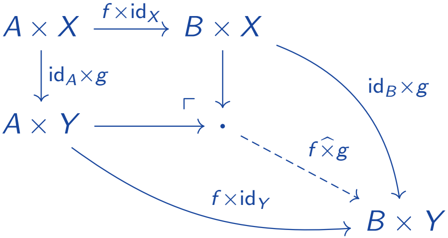
Pushout-product and pullback-hom

Given $f : A \to B$ and $g : X \to Y$, we define

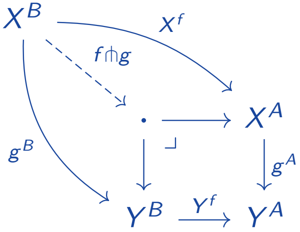


Pushout-product and pullback-hom

Given $f : A \to B$ and $g : X \to Y$, we define



and



Second main result: the Leibniz adjunction in HoTT

Def. For $f : A \to B$ and $g : X \to Y$, write

$$\mathrm{Map}(f,g) :\equiv \left\{ \begin{array}{ccc} A & \xrightarrow{u} & X \\ {\scriptstyle f}\downarrow & \overset{\alpha}{\swarrow} & \downarrow{\scriptstyle g} \\ B & \xrightarrow{v} & Y \end{array} \right\}$$

for the type of commutative squares with f on the left and g on the right.

Second main result: the Leibniz adjunction in HoTT

Def. For $f : A \to B$ and $g : X \to Y$, write

$$\mathrm{Map}(f, g) :\equiv \left\{ \begin{array}{ccc} A & \xrightarrow{u} & X \\ {\scriptstyle f}\downarrow & \overset{\alpha}{\swarrow} & \downarrow{\scriptstyle g} \\ B & \xrightarrow[v]{} & Y \end{array} \right\}$$

for the type of commutative squares with f on the left and g on the right.

The commutativity is *data* in HoTT!

Second main result: the Leibniz adjunction in HoTT

Def. For $f : A \to B$ and $g : X \to Y$, write

$$\mathrm{Map}(f, g) :\equiv \left\{ \begin{array}{ccc} A & \xrightarrow{u} & X \\ {\scriptstyle f}\downarrow & \overset{\alpha}{\swarrow} & \downarrow{\scriptstyle g} \\ B & \xrightarrow{v} & Y \end{array} \right\}$$

for the type of commutative squares with f on the left and g on the right.

⚠ The commutativity is *data* in HoTT!

Thm. We have a natural equivalence

$$\mathrm{Map}(f \mathbin{\widehat{\times}} g, h) \simeq \mathrm{Map}(f, g \pitchfork h).$$

Towards a proof of the Leibniz adjunction

- Unfolding definitions, an element of $\mathrm{Map}(f \mathbin{\widehat{\times}} g, h)$ is a 7-tuple: three maps, three equations between maps and one additional coherence.

  Similarly for $\mathrm{Map}(f, g \pitchfork h)$.

Towards a proof of the Leibniz adjunction

- Unfolding definitions, an element of $\mathrm{Map}(f \mathbin{\widehat{\times}} g, h)$ is a 7-tuple: three maps, three equations between maps and one additional coherence.

  Similarly for $\mathrm{Map}(f, g \pitchfork h)$.

- With some squinting, a little hand-waving and enough whiteboard space, you can convince yourself that you can map such tuples across to get an equivalence.

Towards a proof of the Leibniz adjunction

- Unfolding definitions, an element of $\mathrm{Map}(f \mathbin{\widehat{\times}} g, h)$ is a 7-tuple: three maps, three equations between maps and one additional coherence.

  Similarly for $\mathrm{Map}(f, g \pitchfork h)$.

- With some squinting, a little hand-waving and enough whiteboard space, you can convince yourself that you can map such tuples across to get an equivalence.

- *But the details are annoying...*

Towards a proof of the Leibniz adjunction

- Unfolding definitions, an element of $\mathrm{Map}(f \mathbin{\widehat{\times}} g, h)$ is a 7-tuple: three maps, three equations between maps and one additional coherence.

  Similarly for $\mathrm{Map}(f, g \pitchfork h)$.

- With some squinting, a little hand-waving and enough whiteboard space, you can convince yourself that you can map such tuples across to get an equivalence.

- *But the details are annoying...*

  Use a proof assistant to help us!

Towards a proof of the Leibniz adjunction

- Unfolding definitions, an element of $\mathrm{Map}(f \mathbin{\widehat{\times}} g, h)$ is a 7-tuple: three maps, three equations between maps and one additional coherence.

  Similarly for $\mathrm{Map}(f, g \pitchfork h)$.

- With some squinting, a little hand-waving and enough whiteboard space, you can convince yourself that you can map such tuples across to get an equivalence.

- *But the details are annoying...*

  💡 Use a proof assistant to help us!

  But the details are annoying...

Axel goes to Nottingham

- Axel Ljungström suggested to use families instead of maps.

Axel goes to Nottingham

- Axel Ljungström suggested to use families instead of maps.
- By univalence, the type of (small) maps
$$\textstyle\sum (A : \mathcal{U}) \sum (B : \mathcal{U}) \, (A \to B)$$
is equivalent to the type of type families
$$\textstyle\sum (X : \mathcal{U}) \, (X \to \mathcal{U}).$$

Axel goes to Nottingham

- Axel Ljungström suggested to use families instead of maps.
- By univalence, the type of (small) maps

$$\sum(A : \mathcal{U}) \sum(B : \mathcal{U}) (A \to B)$$

is equivalent to the type of type families

$$\sum(X : \mathcal{U}) (X \to \mathcal{U}).$$

A map $f : A \to B$ is sent to the family $b \mapsto \mathrm{fib}_f \ b :\equiv \sum(a : A) \, f \, a = b$.
Conversely, a family $Y \to \mathcal{U}$ is sent to the projection $\mathrm{pr}_1 : (\sum(x : X) \, Y \, x) \to X$.

Axel goes to Nottingham

- Axel Ljungström suggested to use families instead of maps.
- By univalence, the type of (small) maps
$$\textstyle\sum(A : \mathcal{U}) \sum(B : \mathcal{U}) (A \to B)$$
is equivalent to the type of type families
$$\textstyle\sum(X : \mathcal{U}) (X \to \mathcal{U}).$$
A map $f : A \to B$ is sent to the family $b \mapsto \mathrm{fib}_f\ b :\equiv \sum(a : A)\, f\, a = b$.
Conversely, a family $Y \to \mathcal{U}$ is sent to the projection $\mathrm{pr}_1 : (\sum(x : X)\, Y\, x) \to X$.
- Given $Y : X \to \mathcal{U}$ and $Y' : X' \to \mathcal{U}$, write
$$\textstyle\mathrm{Fam}((X, Y); (X', Y')) :\equiv \sum(m : X \to X') \prod(x : X) (Y\, x \to Y'(m\, x)).$$

Axel goes to Nottingham

- Axel Ljungström suggested to use families instead of maps.
- By univalence, the type of (small) maps
$$\sum(A : \mathcal{U}) \sum(B : \mathcal{U}) (A \to B)$$
is equivalent to the type of type families
$$\sum(X : \mathcal{U}) (X \to \mathcal{U}).$$
A map $f : A \to B$ is sent to the family $b \mapsto \mathsf{fib}_f\ b :\equiv \sum(a : A)\, f\, a = b$.
Conversely, a family $Y \to \mathcal{U}$ is sent to the projection $\mathsf{pr}_1 : (\sum(x : X)\, Y\, x) \to X$.
- Given $Y : X \to \mathcal{U}$ and $Y' : X' \to \mathcal{U}$, write
$$\mathsf{Fam}((X, Y); (X', Y')) :\equiv \sum(m : X \to X') \prod(x : X) (Y\, x \to Y'(m\, x)).$$
- The above equivalence χ extends to an equivalence
$$\mathsf{Map}(f, g) \simeq \mathsf{Fam}(\chi\, f, \chi\, g).$$

The merits of Fam

- Associativity of composition in Map requires some path algebra.
  In contrast, composition in Fam is *definitionally* associative.

The merits of Fam

- Associativity of composition in Map requires some path algebra.
  In contrast, composition in Fam is *definitionally* associative.

The merits of Fam

- Associativity of composition in Map requires some path algebra.
  In contrast, composition in Fam is *definitionally* associative.

1. Calculate what $\pitchfork$ and $\widehat{\times}$ should be for families.
   Calculation is guided by the desired equations
   $\chi(f \mathbin{\widehat{\times}} g) = \chi f \mathbin{\widehat{\times}} \chi g$ and $\chi(f \pitchfork g) = \chi f \pitchfork \chi g$.

The merits of Fam

- Associativity of composition in Map requires some path algebra.
  In contrast, composition in Fam is *definitionally* associative.

1. Calculate what $\pitchfork$ and $\widehat{\times}$ should be for families.
   Calculation is guided by the desired equations
   $\chi(f \mathbin{\widehat{\times}} g) = \chi f \mathbin{\widehat{\times}} \chi g$ and $\chi(f \pitchfork g) = \chi f \pitchfork \chi g$.
2. Construct the adjunction in Fam.

The merits of Fam

- Associativity of composition in Map requires some path algebra.
  In contrast, composition in Fam is *definitionally* associative.

1. Calculate what $\pitchfork$ and $\widehat{\times}$ should be for families.
   Calculation is guided by the desired equations
   $\chi(f \mathbin{\widehat{\times}} g) = \chi f \mathbin{\widehat{\times}} \chi g$ and $\chi(f \pitchfork g) = \chi f \pitchfork \chi g$.
2. Construct the adjunction in Fam.
3. Transfer the adjunction back to Map using
   - $\mathsf{Map}(f, g) \simeq \mathsf{Fam}(\chi f, \chi g)$,
   - that χ and its inverse preserve $\widehat{\times}$ and $\pitchfork$.

The merits of Fam

- Associativity of composition in Map requires some path algebra.
  In contrast, composition in Fam is *definitionally* associative.

1. Calculate what $\pitchfork$ and $\widehat{\times}$ should be for families.
   Calculation is guided by the desired equations
   $\chi(f \mathbin{\widehat{\times}} g) = \chi f \mathbin{\widehat{\times}} \chi g$ and $\chi(f \pitchfork g) = \chi f \pitchfork \chi g$.
2. Construct the adjunction in Fam.
3. Transfer the adjunction back to Map using
   - $\mathsf{Map}(f, g) \simeq \mathsf{Fam}(\chi f, \chi g)$,
   - that χ and its inverse preserve $\widehat{\times}$ and $\pitchfork$.

- Steps (1)–(3) turn out to be relatively easy and natural. ☺

Pushout-products and pullback-homs of families

- As observed in HoTT by Rijke,[3] the pushout-product $f \mathbin{\widehat{\times}} g$ of two maps is the fiberwise join, a certain pushout denoted by $*$.

[3] Egbert Rijke (2017). 'The join construction'. arXiv: 1701.07538 [math.CT].

Pushout-products and pullback-homs of families

- As observed in HoTT by Rijke,[3] the pushout-product $f \mathbin{\widehat{\times}} g$ of two maps is the fiberwise join, a certain pushout denoted by $*$.
- For families $B : A \to \mathcal{U}$ and $Y : X \to \mathcal{U}$, this *immediately* suggests:

$$(A, B) \mathbin{\widehat{\times}} (X, Y) :\equiv (A \times X, (a, x) \mapsto B\, a * Y\, x).$$

[3] Egbert Rijke (2017). 'The join construction'. arXiv: 1701.07538 [math.CT].

Pushout-products and pullback-homs of families

- As observed in HoTT by Rijke,[3] the pushout-product $f \mathbin{\widehat{\times}} g$ of two maps is the fiberwise join, a certain pushout denoted by $*$.

- For families $B : A \to \mathcal{U}$ and $Y : X \to \mathcal{U}$, this *immediately* suggests:

$$(A, B) \mathbin{\widehat{\times}} (X, Y) :\equiv (A \times X, (a, x) \mapsto B\, a * Y\, x).$$

- The pullback-hom is slightly more involved:

[3] Egbert Rijke (2017). 'The join construction'. arXiv: 1701.07538 [math.CT].

Pushout-products and pullback-homs of families

- As observed in HoTT by Rijke,[3] the pushout-product $f \mathbin{\widehat{\times}} g$ of two maps is the fiberwise join, a certain pushout denoted by $*$.

- For families $B : A \to \mathcal{U}$ and $Y : X \to \mathcal{U}$, this *immediately* suggests:

$$(A, B) \mathbin{\widehat{\times}} (X, Y) :\equiv (A \times X, (a, x) \mapsto B\, a * Y\, x).$$

- The pullback-hom is slightly more involved:

$$(A, B) \pitchfork (X, Y) :\equiv (\mathsf{Fam}((A, B); (X, Y)), (m, \mu) \mapsto \textstyle\prod(a : A)\, \mathsf{const}(\mu\, a))$$

[3] Egbert Rijke (2017). 'The join construction'. arXiv: 1701.07538 [math.CT].

Pushout-products and pullback-homs of families

- As observed in HoTT by Rijke,[3] the pushout-product $f \mathbin{\widehat{\times}} g$ of two maps is the fiberwise join, a certain pushout denoted by $*$.

- For families $B : A \to \mathcal{U}$ and $Y : X \to \mathcal{U}$, this *immediately* suggests:

$$(A, B) \mathbin{\widehat{\times}} (X, Y) :\equiv (A \times X, (a, x) \mapsto B\, a * Y\, x).$$

- The pullback-hom is slightly more involved:

$$(A, B) \pitchfork (X, Y) :\equiv (\mathsf{Fam}((A, B); (X, Y)), (m, \mu) \mapsto \textstyle\prod(a : A)\, \mathsf{const}(\mu\, a))$$

where

$$m : A \to X$$
$$\mu : \textstyle\prod(a : A)\, (B\, a \to Y(m\, a))$$

[3] Egbert Rijke (2017). 'The join construction'. arXiv: 1701.07538 [math.CT].

Pushout-products and pullback-homs of families

- As observed in HoTT by Rijke,[3] the pushout-product $f \mathbin{\widehat{\times}} g$ of two maps is the fiberwise join, a certain pushout denoted by $*$.

- For families $B : A \to \mathcal{U}$ and $Y : X \to \mathcal{U}$, this *immediately* suggests:
$$(A, B) \mathbin{\widehat{\times}} (X, Y) :\equiv (A \times X, (a, x) \mapsto B\, a * Y\, x).$$

- The pullback-hom is slightly more involved:
$$(A, B) \pitchfork (X, Y) :\equiv (\mathsf{Fam}((A, B); (X, Y)), (m, \mu) \mapsto \textstyle\prod(a : A)\, \mathsf{const}(\mu\, a))$$
where
$$m : A \to X$$
$$\mu : \textstyle\prod(a : A)\, (B\, a \to Y(m\, a))$$
and
$$\mathsf{const}(f : C \to D) :\equiv \textstyle\sum(d : D) \prod(c : C)\, f\, c = d.$$

[3] Egbert Rijke (2017). 'The join construction'. arXiv: 1701.07538 [math.CT].

The Leibniz adjunction for families

- Recall that the pushout-product of families involved joins ($*$) and that pullback-homs involved a constancy condition (const).

The Leibniz adjunction for families

- Recall that the pushout-product of families involved joins ($*$) and that pullback-homs involved a constancy condition (const).

- The Leibniz adjunction for families essentially follows from a small observation relating joins and constant maps.

The Leibniz adjunction for families

- Recall that the pushout-product of families involved joins ($*$) and that pullback-homs involved a constancy condition (const).
- The Leibniz adjunction for families essentially follows from a small observation relating joins and constant maps.

Lemma For types X, Y and Z, we have equivalences

$$\begin{aligned}(X * Y \to Z) &\simeq \sum (f : X \to Z)\,(Y \to \mathsf{const}\, f) \\ &\simeq \sum (g : Y \to Z)\,(X \to \mathsf{const}\, g).\end{aligned}$$

Conclusion

- Main results:
  1. Segal types have all higher coherences
  2. Leibniz adjunction in HoTT

Conclusion

- Main results:
  1. Segal types have all higher coherences
  2. Leibniz adjunction in HoTT
- The desire to formalize led to a much nicer proof.
- Families proved much more convenient than maps.
  Straightening vs unstraightening

Conclusion

- Main results:
  1. Segal types have all higher coherences
  2. Leibniz adjunction in HoTT
- The desire to formalize led to a much nicer proof.
- Families proved much more convenient than maps.
  Straightening vs unstraightening
- ? Other illustrative examples where this perspective helps?
- ? Precise connection to Riehl & Shulman's type theory?

Conclusion

- Main results:
    1. Segal types have all higher coherences
    2. Leibniz adjunction in HoTT
- The desire to formalize led to a much nicer proof.
- Families proved much more convenient than maps.
  Straightening vs unstraightening
- ? Other illustrative examples where this perspective helps?
- ? Precise connection to Riehl & Shulman's type theory?

The Leibniz adjunction in homotopy type theory, with an application to simplicial type theory. TdJ, Nicolai Kraus, Axel Ljungström. January 2026. `arXiv:2601.21843`.

Fully formalized in Cubical Agda.

References

Gratzer, Daniel, Jonathan Weinberger and Ulrik Buchholtz (2024). 'Directed univalence in simplicial homotopy type theory'. arXiv: 2407.09146 [cs.LO].

Lurie, Jacob (2009). *Higher Topos Theory*. Vol. 170. Annals of Mathematics Studies. Princeton University Press. DOI: 10.1515/9781400830558. arXiv: math/0608040.

Riehl, Emily and Michael Shulman (2017). 'A type theory for synthetic ∞-categories'. In: *Higher Structures* 1.1, pp. 147–224. DOI: 10.21136/hs.2017.06.

Rijke, Egbert (2017). 'The join construction'. arXiv: 1701.07538 [math.CT].