

Towards Computational UIP in Cubical Agda

Agda Implementor's Meeting XLI (Angers, France)

Advisors: Andreas Nuyts, Dominique Devriese



Yee-Jian Tan

24 November, 2025

KU Leuven, IP Paris (École Polytechnique)

Introduction

A (Pictorial) Crash Course on Cubical Type Theory

Implementation: Cubical Agda without Glue

The Tale of Two Square-Fills

Conclusion and Future Work

Introduction

Uniqueness of Identity Proofs (UIP)

Uniqueness of Identity Proofs

Uniqueness of Identity Proofs (UIP) [HS94] postulate: are proofs of equality in Type Theory unique?

- Setoid model [Hof97] supports UIP
- Groupoid model [HS94] refutes UIP

hence UIP does not necessarily hold for every type theory.

Homotopy Type Theory (HoTT) [Uni13]

Vastly generalises the non-uniqueness of identity proofs.

Slogan: “propositions as types, proofs as programs, equalities as paths”

Univalence axiom: $(A \equiv_{\text{Type}} B) \simeq (A \simeq B)$ **incompatible** with UIP.

Cubical Type Theory (CubTT)

A flavour of HoTT [Coh+17] implemented by Cubical Agda [VMA21].

Some **advantages** of CubTT over Dependent Type Theory:

- Functional extensionality (pointwise equal functions are equal)
- Quotient Inductive Types (QITs) as an instance of Higher Inductive Types (HITs)

Cubical Type Theory & Cubical Agda

Cubical Agda and... UIP?

Reasons to have UIP in Cubical Agda include:

- one equality level instead of *infinitely* many (Set-level Type Theory),
- still have funext and Quotient Types,
- researchers would like to work in [Coc19, Pit20, Shu17]

Cubical Type Theory & Cubical Agda

Cubical Agda and... UIP?

Reasons to have UIP in Cubical Agda include:

- one equality level instead of *infinitely* many (Set-level Type Theory),
- still have funext and Quotient Types,
- researchers would like to work in [Coc19, Pit20, Shu17]

But ...

- Univalence is incompatible with UIP,
- naïvely postulating UIP in Cubical Agda **blocks** computation!

$\dots(\text{UIP } (A \times A) a b p q)\dots \rightarrow_{\beta} \dots(\text{UIP } (A \times A) a b p q)\dots$

$\rightarrow_{\beta}^* \dots(\text{UIP } (A \times A) a b p q)\dots$ never reduces!

- **Glue Types** is the only source of univalence in **CubTT**
- if we have a Cubical Agda variant **without Glue**, then one can safely postulate **UIP** as an axiom (consistent by a set model)...
- where functional extensionality holds, QITs too...
- What if: instead of a computation blocking axiom, we can have UIP that **computes**?

Computational UIP in Cubical Agda

Our plan for computational behaviour for UIP in Cubical Agda:

1. A variant of Cubical Agda **without** Glue Types (hence without univalence) to ensure UIP compatibility.
2. The proofs of UIP compute automatically based on their type derivation.

...($\text{UIP } (A \times A) a b p q$)...

\rightarrow_{β} ...[UIP-product ($\text{UIP } A (\pi_1 a) (\pi_1 b)$)...] ($\text{UIP } A (\pi_2 a) (\pi_2 b)$)...]... inductive case


\rightarrow_{β} ... computes away

and so on, reaching base types (such as $\mathbb{0}$, $\mathbb{1}$) or quotient inductive types. Essentially a **proof by induction** on type derivation!

3. It remains to detail all computation rules (e.g. inductive cases: preservation by type formers) in a suitable UIP formulation.

Contributions

In this internship, I

1. extended Cubical Agda (which implements CubTT) with a `--cubical=no-glue` **variant** (<https://github.com/agda/agda/pull/7861>)
2. propose implementing **computational UIP as an induction** on type derivation
 - base cases = base types + (possibly higher) inductive types
 - inductive cases = preservation by type formers
3. propose **homogeneous** `SqFill` and **heterogeneous** `SqPFill` as equivalent generalisations of UIP
4. prove the preservation of `SqFill` and `SqPFill`  by 4 type formers:
 - Π (dependent functions)
 - Σ (dependent products),
 - Coproducts (disjoint sum), and
 - Path types (equality types).

A (Pictorial) Crash Course on Cubical Type Theory

Cubical Type Theory (Simplified)

- CubTT takes the “equalities as **paths**” slogan literally:
 - **paths** (just like in topology) are functions from the interval type $p : I \rightarrow A$
 $p(0) = a, p(1) = b$, then $p : a \equiv b$
 - points (A), line ($I \rightarrow A$), squares ($I^2 \rightarrow A$), cubes ($I^3 \rightarrow A$), ...

Cubical Type Theory (Simplified)

- CubTT takes the “equalities as **paths**” slogan literally:
 - **paths** (just like in topology) are functions from the interval type $p : I \rightarrow A$
 $p(0) = a, p(1) = b$, then $p : a \equiv b$
 - points (A), line ($I \rightarrow A$), squares ($I^2 \rightarrow A$), cubes ($I^3 \rightarrow A$), ...

Cubical Type Theory (Simplified)

- Two primitive structures:
 1. **Interval type I** (for paths): a de Morgan algebra (i.e. bounded distributed lattice $(0, 1, \wedge, \vee)$ + de Morgan involution \sim)
 - de Morgan laws (e.g. $\sim (a \wedge b) = \sim a \vee \sim b$)
 - distributivity laws (e.g. $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$)
 - but **NO** LEM ($a \vee \sim a = 1$) nor absurdity ($a \wedge \sim a = 0$)
 2. Glue Types (for univalence)
- Two primitive operations in Cubical Agda (Kan operations):
 1. **hcomp** (composition) and
 2. **transp** (transport)

Pictorial interpretation (node = object, edge = path)

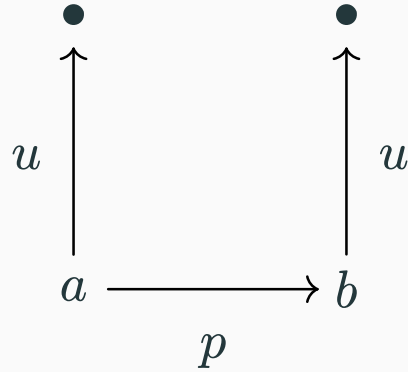
- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.

$$a \xrightarrow[p]{} b$$

hcomp of a homogeneous path p along partial sides u .

Pictorial interpretation (node = object, edge = path)

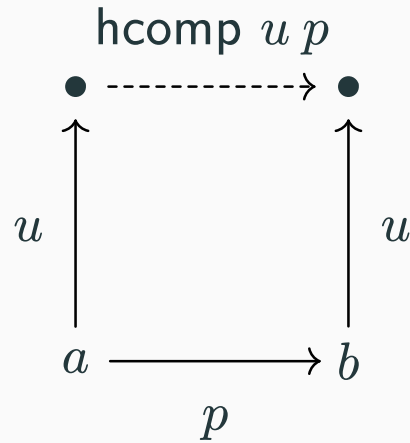
- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.



hcomp of a homogeneous path p along partial sides u .

Pictorial interpretation (node = object, edge = path)

- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.



hcomp of a homogeneous path p along partial sides u .

Pictorial interpretation (node = object, edge = path)

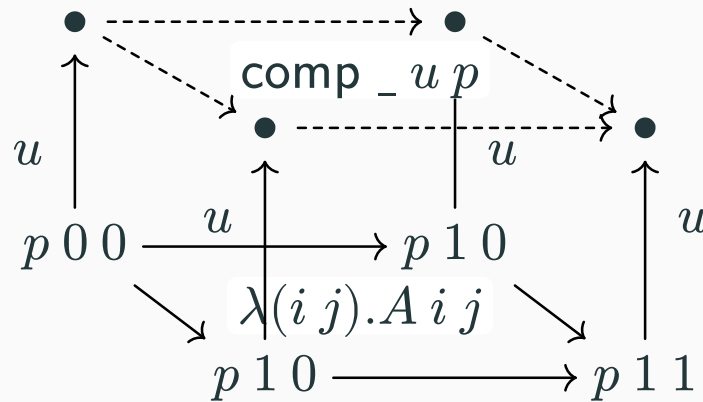
- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.
- **comp**: *heterogeneously* composing a path with partial sides to get a “lid”.

$$\begin{array}{ccc} p\ 0\ 0 & \longrightarrow & p\ 1\ 0 \\ & \searrow & \swarrow \\ & \lambda(i\ j).A\ i\ j & \\ & p\ 1\ 0 & \longrightarrow & p\ 1\ 1 \end{array}$$

comp of a heterogeneous square p along partial sides u .

Pictorial interpretation (node = object, edge = path)

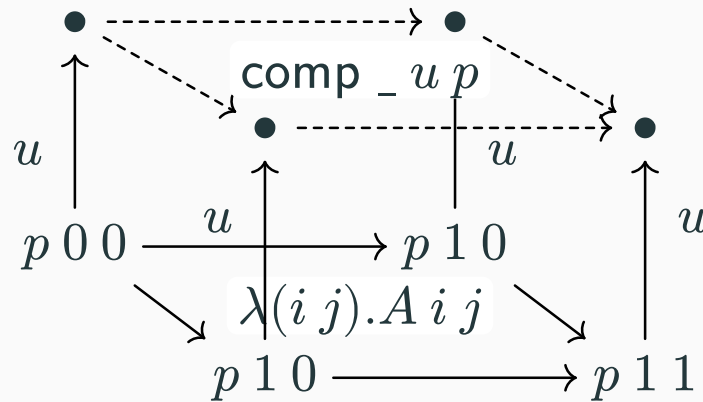
- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.
- **comp**: *heterogeneously* composing a path with partial sides to get a “lid”.



comp of a heterogeneous square p along partial sides u .

Pictorial interpretation (node = object, edge = path)

- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.
- **comp**: *heterogeneously* composing a path with partial sides to get a “lid”.



comp of a heterogeneous square p along partial sides u .

- Glue types: comp but for types, where u can be type equivalences (instead of paths)

Pictorial interpretation (node = object, edge = path)

- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.
- **comp**: *heterogeneously* composing a path with partial sides to get a “lid”.
- Glue types: **comp** but for types, where u can be type equivalences (instead of paths)
- **transport** : $\forall (A : I \rightarrow \text{Type}). A\ 0 \rightarrow A\ 1.$

$a : A\ 0$

Transporting along the line of types A .

Pictorial interpretation (node = object, edge = path)

- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.
- **comp**: *heterogeneously* composing a path with partial sides to get a “lid”.
- Glue types: **comp** but for types, where u can be type equivalences (instead of paths)
- **transport** : $\forall (A : I \rightarrow \text{Type}). A\ 0 \rightarrow A\ 1$.

transport $A\ a : A\ 1$

$a : A\ 0$

Transporting along the line of types A .

Pictorial interpretation (node = object, edge = path)

- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.
- **comp**: *heterogeneously* composing a path with partial sides to get a “lid”.
- Glue types: **comp** but for types, where u can be type equivalences (instead of paths)
- **transport** : $\forall (A : I \rightarrow \text{Type}). A\ 0 \rightarrow A\ 1$.
 - “transportee” and the transported target are always propositionally equal.

$$\begin{array}{c} \text{transport } A\ a : A\ 1 \\ \uparrow \\ \text{transport-fill } A\ (p\ 0) \\ \uparrow \\ a : A\ 0 \end{array}$$

Transporting along the line of types A .

Pictorial interpretation (node = object, edge = path)

- **hcomp**: *homogeneously* composing a path with partial sides to get a “lid”.
- **comp**: *heterogeneously* composing a path with partial sides to get a “lid”.
- Glue types: **comp** but for types, where u can be type equivalences (instead of paths)
- **transport** : $\forall (A : I \rightarrow \text{Type}). A\ 0 \rightarrow A\ 1$.
 - ▶ “transportee” and the transported target are always propositionally equal.
 - ▶ In a square of types $A : I \rightarrow I \rightarrow \text{Type}$, any two types $A\ i\ j$ and $A\ i'\ j'$ has a path between them: $\lambda(k : I). A\ \text{coe}_k(i, i')\ \text{coe}_k(j, j')$.

$$\begin{array}{c} \text{transport } A\ a : A\ 1 \\ \uparrow \\ \text{transport-fill } A\ (p\ 0) \\ \uparrow \\ a : A\ 0 \end{array}$$

Transporting along the line of types A .

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity?

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)
- symmetry?

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)
- symmetry? negating path variables (e.g. $\text{sym } p := \lambda(i : I).p (\sim i)$)

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)
- symmetry? negating path variables (e.g. $\text{sym } p := \lambda(i : I).p (\sim i)$)
- transitivity?

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)
- symmetry? negating path variables (e.g. $\text{sym } p := \lambda(i : I).p (\sim i)$)
- transitivity? by **hcomp**:

$$a \xrightarrow{p} b$$

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)
- symmetry? negating path variables (e.g. $\text{sym } p := \lambda(i : I).p (\sim i)$)
- transitivity? by **hcomp**:

$$\begin{array}{ccc} & & c \\ & & \uparrow q \\ a & \longrightarrow & b \\ & & p \end{array}$$

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)
- symmetry? negating path variables (e.g. $\text{sym } p := \lambda(i : I).p (\sim i)$)
- transitivity? by **hcomp**:

$$\begin{array}{ccc} & a & c \\ \text{refl} & \parallel & \uparrow q \\ & a & \longrightarrow b \\ & & p \end{array}$$

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)
- symmetry? negating path variables (e.g. $\text{sym } p := \lambda(i : I).p (\sim i)$)
- transitivity? by **hcomp**:

$$\begin{array}{ccc} & & \text{hcomp (refl, } q) p \\ & a \text{ } \dashrightarrow & c \\ \text{refl} \parallel & & \uparrow q \\ & a \text{ } \longrightarrow & b \\ & & p \end{array}$$

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)
- symmetry? negating path variables (e.g. $\text{sym } p := \lambda(i : I).p (\sim i)$)
- transitivity? by **hcomp**:

$$\begin{array}{ccc} & \text{hcomp } (\text{refl}, q) p & \\ & a \dashrightarrow c & \\ \text{refl} \parallel & & \uparrow q \\ & a \longrightarrow b & \\ & p & \end{array}$$

- substitutivity ($\forall (P : A \rightarrow \text{Type}).(x y : A).(p : x = y).Px \rightarrow Py$)?

Properties of Equality

Let's show reflexivity, symmetry, transitivity, and substitutivity (Leibniz's law).

- reflexivity? constant functions from I (e.g. $\text{refl } \{a\} := \lambda(i : I).a : I \rightarrow A$)
- symmetry? negating path variables (e.g. $\text{sym } p := \lambda(i : I).p (\sim i)$)
- transitivity? by **hcomp**:

$$\begin{array}{ccc} & \text{hcomp (refl, } q) p & \\ & a \dashrightarrow c & \\ \text{refl} \parallel & & \uparrow q \\ & a \longrightarrow b & \\ & p & \end{array}$$

- substitutivity ($\forall (P : A \rightarrow \text{Type}).(x y : A).(p : x = y).Px \rightarrow Py$)?

transport along the line of types $\lambda(i : I).P(p i)$.

Implementation: Cubical Agda without Glue

Cubical Agda without Glue

... is a Cubical Agda variant designed to be *compatible* with UIP.

Three Cubical-related variants already exist:

1. Full Cubical `--cubical[=full]` and
2. Cubical with Erased Glue `--cubical=erased`
3. Cubical-Compatible `--cubical=compatible`

Two key safety features: `--cubical=no-glue` should have...

1. **NO** primitive Glue types and terms

Cubical Agda without Glue

... is a Cubical Agda variant designed to be *compatible* with UIP.

Three Cubical-related variants already exist:

1. Full Cubical `--cubical[=full]` and
2. Cubical with Erased Glue `--cubical=erased`
3. Cubical-Compatible `--cubical=compatible`

Two key safety features: `--cubical=no-glue` should have...

1. **NO** primitive Glue types and terms: `requireCubical <variant>` in the TCM monad
2. **NO** imported modules can contain Glue

Cubical Agda without Glue

... is a Cubical Agda variant designed to be *compatible* with UIP.

Three Cubical-related variants already exist:

1. Full Cubical `--cubical[=full]` and
2. Cubical with Erased Glue `--cubical=erased`
3. Cubical-Compatible `--cubical=compatible`

Two key safety features: `--cubical=no-glue` should have...

1. **NO** primitive Glue types and terms: `requireCubical <variant>` in the TCM monad
2. **NO** imported modules can contain Glue:
 - total order on variants: `--cubical={full,erased,no-glue,compatible}`
 - require dependent modules to enable Cubical (“infective option”)

Current Status of `--cubical=no-glue`

- ~~<https://github.com/agda/agda/pull/7861>~~ merged! (16 Oct 2025)

Cubical Agda without Glue

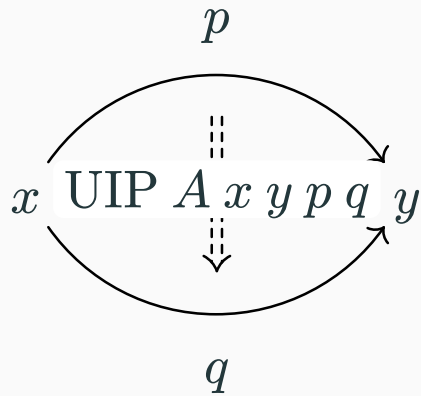
Current Status of `--cubical=no-glue`

- ~~<https://github.com/agda/agda/pull/7861>~~ merged! (16 Oct 2025)
- Type checks the Glue-less parts of the Cubical Library [Agd25]!
- Type checks our **SqFill**, **SqPFill** preservation proofs! (which should definitely *not* use Glue)

The Tale of Two Square-Fills

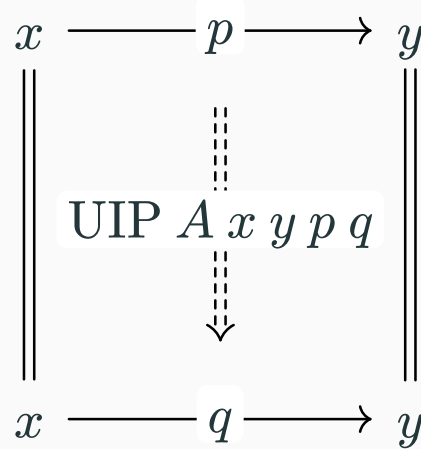
Generalising UIP: SqFill

- $\text{UIP } A$: for any two proofs of identity $p q : x \equiv_A y$, we have $p \equiv_{x \equiv_A y} q$.



Generalising UIP: SqFill

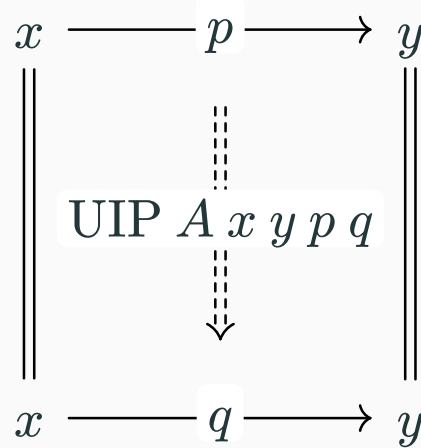
- **UIP A**: for any square with two (opposing) reflexive sides, we have a filling.



Square with reflexive sides.

Generalising UIP: SqFill

- **UIP** \mathcal{A} : for any square with two (opposing) reflexive sides, we have a filling.



Square with reflexive sides.

- **SqFill** \mathcal{A} : Any hollow square in \mathcal{A} has a filling.

Generalising UIP: SqFill

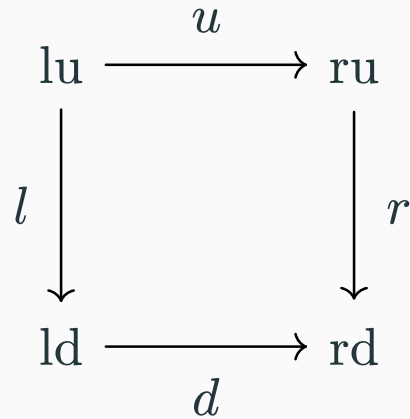
- **UIP** \mathcal{A} : for any square with two (opposing) reflexive sides, we have a filling.
- **SqFill** \mathcal{A} : Any hollow square in \mathcal{A} has a filling.

lu ru

ld rd

Generalising UIP: SqFill

- **UIP** \mathcal{A} : for any square with two (opposing) reflexive sides, we have a filling.
- **SqFill** \mathcal{A} : Any hollow square in \mathcal{A} has a filling.



Generalising UIP: SqFill

- **UIP** \mathcal{A} : for any square with two (opposing) reflexive sides, we have a filling.
- **SqFill** \mathcal{A} : Any hollow square in \mathcal{A} has a filling.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{ru} \\ \downarrow l & \text{SqFill } \mathcal{A} & \downarrow r \\ & \dashrightarrow & \\ \text{ld} & \xrightarrow{d} & \text{rd} \end{array}$$

- Surprisingly, **SqFill** \leftrightarrow **UIP**, even though easier to use!

Generalising UIP: SqFill

- **UIP** \mathcal{A} : for any square with two (opposing) reflexive sides, we have a filling.
- **SqFill** \mathcal{A} : Any hollow square in \mathcal{A} has a filling.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{ru} \\ \downarrow l & \text{SqFill } \mathcal{A} & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{rd} \end{array}$$

- Surprisingly, **SqFill** \leftrightarrow **UIP**, even though easier to use!

Generalising SqFill: SqPFill

A square of types $A : I \rightarrow I \rightarrow \text{Type}$ has the **heterogeneous square-filling property** $\text{SqPFill } A$ if the following holds:

For any hollow square in $A : I \rightarrow I \rightarrow \text{Type}$, that is

- four corners $lu : A\ 0\ 0$, $ru : A\ 1\ 0$, $ld : A\ 0\ 1$, $rd : A\ 1\ 1$,
and
- four sides connecting the four corners, namely
 1. $l : \text{PathP } (\lambda j \rightarrow A\ 0\ j) lu\ ld$
 2. $r : \text{PathP } (\lambda j \rightarrow A\ 1\ j) ru\ rd$
 3. $u : \text{PathP } (\lambda i \rightarrow A\ i\ 0) lu\ ru$
 4. $d : \text{PathP } (\lambda i \rightarrow A\ i\ 1) ld\ rd$

$$\begin{array}{ccc} lu : A\ 0\ 0 & \xrightarrow{u} & ru : A\ 1\ 0 \\ \downarrow l & \text{SqPFill } A & \downarrow r \\ ld : A\ 0\ 1 & \xrightarrow{d} & rd : A\ 1\ 1 \end{array}$$

then the square has a filling: $\text{PathP } (\lambda(i : I) \rightarrow \text{PathP } (\lambda(j : I).A\ i\ j) (u\ i) (d\ i)) l\ r$.

Reminder: How to have Computational SqFill?

...($p : \text{SqFill } (A \times B)$)...

\rightarrow_{β} ...(**theSqFill-Sigma** (SqFill A) (SqFill B))... inductive case

\rightarrow_{β} ... computes away

...($p : \text{SqFill } (\Pi(a : A).B a)$)...

\rightarrow_{β} ...(**theSqFill-Pi** (SqFill B))... inductive case

\rightarrow_{β} ... computes away

Base cases:

- $\text{SqFill } \mathbb{0}$, $\text{SqFill } \mathbb{1}$, $\text{SqFill } \mathbb{2}$
- (higher) inductive types (W -types, adding a `sqFill` constructor)

SqFill: Summary

	SqFill (<i>homogeneous</i> Square-Filling)	SqPFill (<i>heterogeneous</i> Square-Filling)
Pi	Trivial (no Kan operations)	Complicated : transport-fill-align
Sigma	Complicated : transport-fill-align	Trivial (no Kan operations)
Coproducts	Standard encode-decode proof (J , hcomp)	Standard encode-decode proof (J , comp)
Path Types	Simple (a single hcomp)	Complicated* : transport-fill-align

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{ru} \\ l \downarrow & \Pi(a : A).B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{rd} \end{array}$$

Hollow square of functions given.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{ru} \\ l \downarrow & \Pi(a : A).B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{rd} \end{array}$$

Hollow square of functions given.

$$\begin{array}{ccc} \text{lub} & \xrightarrow{\text{ub}} & \text{rub} \\ \text{lb} \downarrow & \text{:: SqFill } (B a) \Rightarrow & \downarrow \text{rb} \\ \text{ldb} & \xrightarrow{\text{db}} & \text{rdb} \end{array}$$

Applying the hollow square to a , and fill.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**

`SqFillPiAB : SqFill ((a : A) → B a)`

`SqFillPiAB l r u d i j a = SqFillB a (λ i → l i a) (λ i → r i a) (λ i → u i a) (λ i → d i a) i j`

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A)B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

Hollow square in $\Sigma(a : A)B a$.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A)B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

Hollow square in $\Sigma(a : A)B a$.

$$\begin{array}{ccc} \pi_1(\text{lu}) & \xrightarrow{\pi_1(u)} & \pi_1(\text{rd}) \\ \pi_1(l) \downarrow & & \downarrow \pi_1(r) \\ \pi_1(\text{ld}) & \xrightarrow{\pi_1(d)} & \pi_1(\text{ru}) \end{array}$$

First projection in A .

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A)B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

Hollow square in $\Sigma(a : A)B a$.

$$\begin{array}{ccc} \pi_1(\text{lu}) & \xrightarrow{\pi_1(u)} & \pi_1(\text{rd}) \\ \pi_1(l) \downarrow & \text{SqFill } A & \downarrow \pi_1(r) \\ \pi_1(\text{ld}) & \xrightarrow{\pi_1(d)} & \pi_1(\text{ru}) \end{array}$$

First projection in A .

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A)B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

Hollow square in $\Sigma(a : A)B a$.

$$\begin{array}{ccc} \pi_2(\text{lu}) & \xrightarrow{\pi_2(u)} & \pi_2(\text{rd}) \\ \pi_2(l) \downarrow & \lambda(i j : I).B(\text{sqa } i j) & \downarrow \pi_2(r) \\ \pi_2(\text{ld}) & \xrightarrow{\pi_2(d)} & \pi_2(\text{ru}) \end{array}$$

Second projection

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A)B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

Hollow square in $\Sigma(a : A)B a$.

$$\begin{array}{ccc} \pi_2(\text{lu}) & \xrightarrow{\pi_2(u)} & \pi_2(\text{rd}) \\ \pi_2(l) \downarrow & \lambda(i j : I).B(\text{sqa } i j) & \downarrow \pi_2(r) \\ \pi_2(\text{ld}) & \xrightarrow{\pi_2(d)} & \pi_2(\text{ru}) \end{array}$$

Second projection is **heterogeneous!**

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A)B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

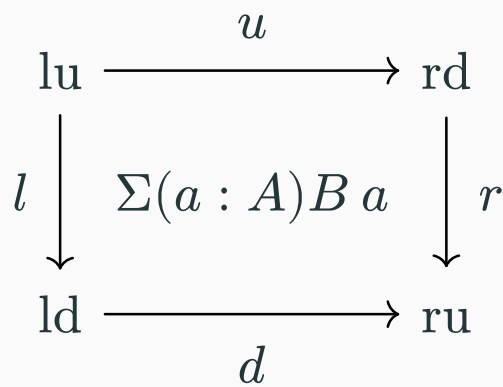
Hollow square in $\Sigma(a : A)B a$.

$$\begin{array}{ccc} \pi_2(\text{lu}) & \xrightarrow{\quad} & \pi_2(\text{ru}) \\ & \searrow & \searrow \\ & \pi_2(\text{ld}) & \xrightarrow{\quad} \pi_2(\text{rd}) \end{array}$$

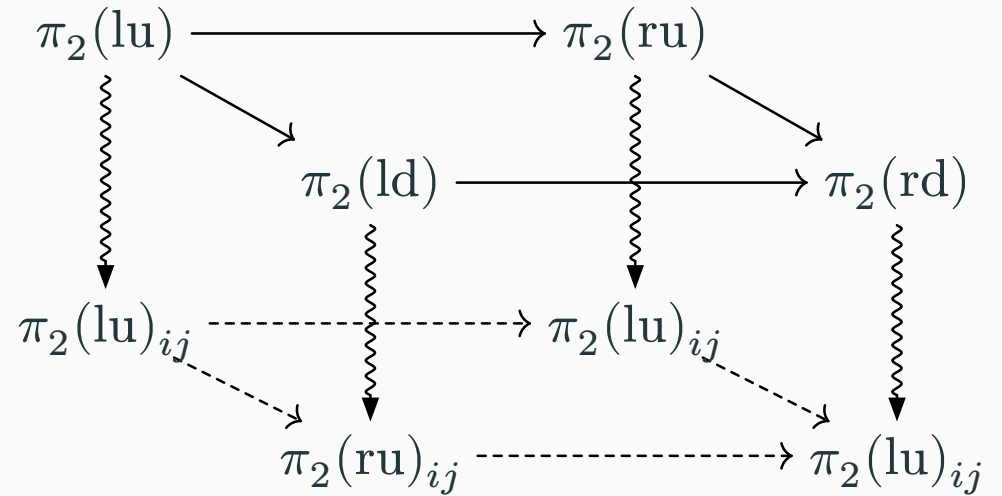
Transporting the second projection down to a fixed B (sq*a i j*).

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.



Hollow square in $\Sigma(a : A) B a$.



Transporting the second projection down to a fixed B (sq*a i j*).

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A)B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

Hollow square in $\Sigma(a : A)B a$.

$$\begin{array}{ccccc} \pi_2(\text{lu})_{ij} & \xrightarrow{\quad} & \pi_2(\text{lu})_{ij} & & \\ & \searrow & & \searrow & \\ & & \pi_2(\text{ru})_{ij} & \xrightarrow{\quad} & \pi_2(\text{lu})_{ij} \end{array}$$

Transporting the second projection down to a fixed B (sq*a i j*).

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A)B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

Hollow square in $\Sigma(a : A)B a$.

$$\begin{array}{ccccc} \pi_2(\text{lu})_{ij} & \xrightarrow{\quad} & \pi_2(\text{lu})_{ij} & & \\ & \searrow & \text{SqFill } (B (\text{sqa } i j)) & \searrow & \\ & & \pi_2(\text{ru})_{ij} & \xrightarrow{\quad} & \pi_2(\text{ru})_{ij} \end{array}$$

Transporting the second projection down to a fixed $B (\text{sqa } i j)$.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A) B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

Hollow square in $\Sigma(a : A) B a$.

$$\begin{array}{ccccc} \pi_2(\text{lu})_{ij} & \xrightarrow{\quad} & \pi_2(\text{lu})_{ij} & & \\ & \searrow & \text{SqFill } (B (\text{sqa } i j)) & \searrow & \\ & & \bullet & & \\ & & \pi_2(\text{ru})_{ij} & \xrightarrow{\quad} & \pi_2(\text{ru})_{ij} \end{array}$$

Take the (i, j) -th coordinate from the fill.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.

$$\begin{array}{ccc} \text{lu} & \xrightarrow{u} & \text{rd} \\ l \downarrow & \Sigma(a : A)B a & \downarrow r \\ \text{ld} & \xrightarrow{d} & \text{ru} \end{array}$$

Hollow square in $\Sigma(a : A)B a$.

$$\begin{array}{ccc} \pi_2(\text{lu})_{00} & \xrightarrow{\quad} & \pi_2(\text{ru})_{10} \\ & \searrow & \swarrow \\ & \pi_2(\text{ld})_{01} & \xrightarrow{\quad} & \pi_2(\text{rd})_{11} \end{array}$$

Collect all the (i, j) -th coordinates to form a **heterogeneous** square.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.

$$\begin{array}{ccc}
 \text{lu} & \xrightarrow{u} & \text{rd} \\
 l \downarrow & \Sigma(a : A) B a & \downarrow r \\
 \text{ld} & \xrightarrow{d} & \text{ru}
 \end{array}$$

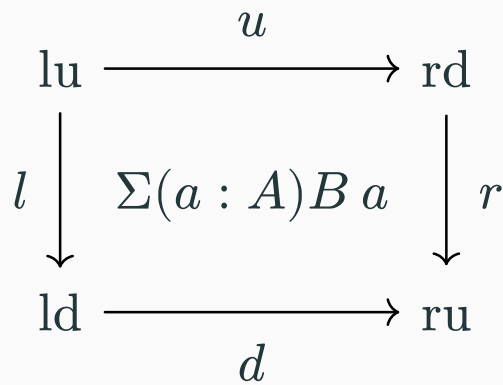
Hollow square in $\Sigma(a : A) B a$.

$$\begin{array}{ccc}
 \pi_2(\text{lu}) & \dashrightarrow & \pi_2(\text{ru}) \\
 \swarrow & & \searrow \\
 & \pi_2(\text{ld}) \dashrightarrow & \pi_2(\text{rd}) \\
 \\
 \pi_2(\text{lu})_{00} & \longrightarrow & \pi_2(\text{ru})_{10} \\
 \searrow & & \searrow \\
 & \pi_2(\text{ld})_{01} \longrightarrow & \pi_2(\text{rd})_{11}
 \end{array}$$

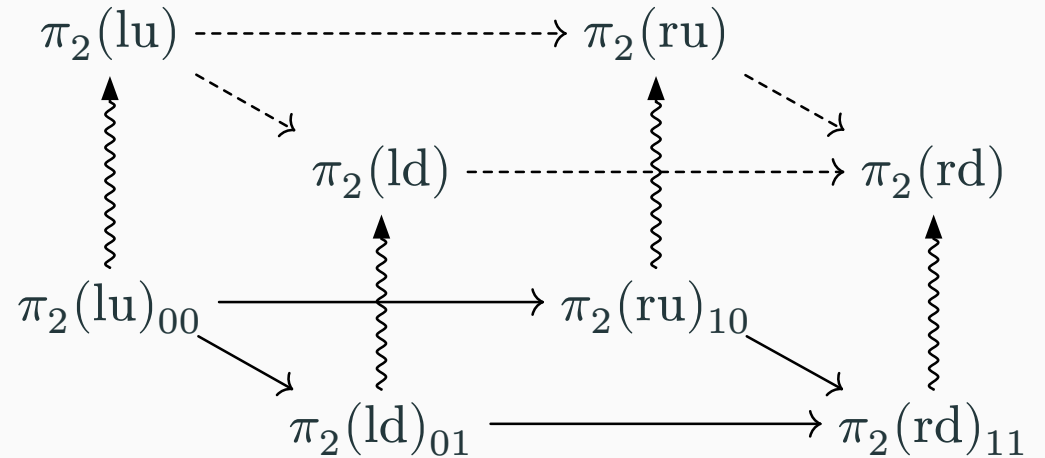
Aligning the sides of the heterogeneous square in $\Pi(i' j' : I).B (\text{sq} a i' j')$ by **comp**.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial.



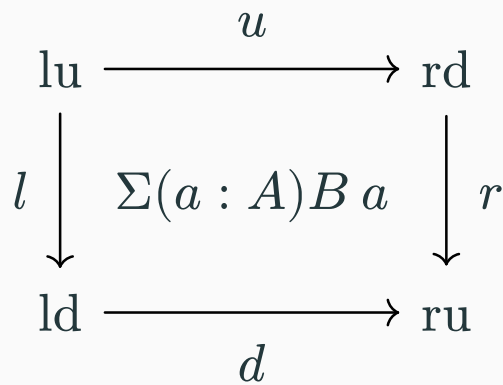
Hollow square in $\Sigma(a : A) B a$.



Aligning the sides of the heterogeneous square in $\Pi(i' j' : I).B (\text{sqa } i' j')$ by **comp.**

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial. Second projection is **complicated**.



Hollow square in $\Sigma(a : A) B a$.



Aligning the sides of the heterogeneous square in $\Pi(i' j' : I).B(\text{sqa } i' j')$ by **comp**.

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. First projection is trivial. Second projection is **complicated**.

```
if_then_else_end : I → I → I → I
if i then j else k end = (k ∧ (~ i ∨ j)) ∨ ((i ∨ k) ∧ j)

{-# INLINE if_then_else_end #-}

SqFillSigmaAB : SqFill (λ [a ∈ A] B a)
SqFillSigmaAB l r u d i j .fst = SqFillA (cong fst l) (cong fst r) (cong fst u) (cong fst d) i j
SqFillSigmaAB {lu} {ld} l {ru} {rd} r u d i j .snd = outS (sqb i j)
  where
    sq = Square (cong fst l) (cong fst r) (cong fst u) (cong fst d)
    sq = SqFillA (cong fst l) (cong fst r) (cong fst u) (cong fst d)

spread : (i j i' j' : I) → sq a i j = sq a i' j'
spread i j i' j' k = sq (if k then i' else i end) (if k then j' else j end)

lub : B (sq a i0 i0)
lub = snd lu
lub' : B (sq a i j)
lub' = transport (λ k → B (spread i0 i0 i j k)) lub
LemmaLU : PathP (λ k → B (spread i0 i0 i j k)) lub lub'
LemmaLU k = transp (λ l → B (spread i0 i0 i j (k A l))) (~ k) lub

ldb : B (fst ld)
ldb = snd ld
ldb' : B (sq a i j)
ldb' = transport (λ k → B (spread i0 i1 i j k)) ldb
LemmaLD : PathP (λ k → B (spread i0 i1 i j k)) ldb ldb'
LemmaLD k = transp (λ l → B (spread i0 i1 i j (k A l))) (~ k) ldb

lb : PathP (λ k → B (spread i0 i0 i1 i k)) lub ldb
lb = cong snd l
lb' : PathP (λ k → B (spread i j i j k)) lub' ldb'
lb' j' = comp (λ k → B (spread (k A i) (k A j) (k A i) (~ k ∨ j) j'))
  (λ where
    k (j' = i0) → LemmaLU k
    k (j' = i1) → LemmaLD k) (lb j')
LemmaL : PathP (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (k' A i) (~ k' ∨ j) k)) (LemmaLU k')
(LemmaLD k')) lb lb'
LemmaL k' = transport-filler (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (k' A i) (~ k' ∨ j) k))
(LemmaLU k') (LemmaLD k')) lb k'

(SqFillSigmaAB {lu} {ld} l {ru} {rd} r u d i j .snd = outS (sqb i j)
  where
    sq = Square (cong fst l) (cong fst r) (cong fst u) (cong fst d)
    sq = SqFillA (cong fst l) (cong fst r) (cong fst u) (cong fst d)

spread : (i j i' j' : I) → sq a i j = sq a i' j'
spread i j i' j' k = sq (if k then i' else i end) (if k then j' else j end)

lub : B (sq a i0 i0)
lub = snd lu
lub' : B (sq a i j)
lub' = transport (λ k → B (spread i0 i0 i j k)) lub
LemmaLU : PathP (λ k → B (spread i0 i0 i j k)) lub lub'
LemmaLU k = transp (λ l → B (spread i0 i0 i j (k A l))) (~ k) lub

ldb : B (fst ld)
ldb = snd ld
ldb' : B (sq a i j)
ldb' = transport (λ k → B (spread i0 i1 i j k)) ldb
LemmaLD : PathP (λ k → B (spread i0 i1 i j k)) ldb ldb'
LemmaLD k = transp (λ l → B (spread i0 i1 i j (k A l))) (~ k) ldb

lb : PathP (λ k → B (spread i0 i0 i1 i k)) lub ldb
lb = cong snd l
lb' : PathP (λ k → B (spread i j i j k)) lub' ldb'
lb' j' = comp (λ k → B (spread (k A i) (k A j) (k A i) (~ k ∨ j) j'))
  (λ where
    k (j' = i0) → LemmaLU k
    k (j' = i1) → LemmaLD k) (lb j')
LemmaL : PathP (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (k' A i) (~ k' ∨ j) k)) (LemmaLU k')
(LemmaLD k')) lb lb'
LemmaL k' = transport-filler (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (k' A i) (~ k' ∨ j) k))
(LemmaLU k') (LemmaLD k')) lb k'
```

```
rub : B (fst ru)
rub = snd ru
rub' : B (sq a i j)
rub' = transport (λ k → B (spread i1 i0 i j k)) rub
LemmaRU : PathP (λ k → B (spread i1 i0 i j k)) rub rub'
LemmaRU k = transp (λ l → B (spread i1 i0 i j (k A l))) (~ k) rub

rdb : B (fst rd)
rdb = snd rd
rdb' : B (sq a i j)
rdb' = transport (λ k → B (spread i1 i1 i j k)) rdb
LemmaRD : PathP (λ k → B (spread i1 i1 i j k)) rdb rdb'
LemmaRD k = transp (λ l → B (spread i1 i1 i j (k A l))) (~ k) rdb

rb : PathP (λ j → B (sq a i1 j)) rub rdb
rb = cong snd r
rb' : rub' = rdb'
rb' j' = comp (λ k → B (spread (~ k ∨ i) (k A j) (~ k ∨ i) (~ k ∨ j) j'))
  (λ where
    k (j' = i0) → LemmaRU k
    k (j' = i1) → LemmaRD k) (rb j')
LemmaR : PathP (λ k' → PathP (λ k → B (spread (~ k' ∨ i) (k' A j) (~ k' ∨ i) (~ k' ∨ j) k)) (LemmaRU k')
(LemmaRD k')) rb rb'
LemmaR k' = transport-filler (λ k' → PathP (λ k → B (spread (~ k' ∨ i) (k' A j) (~ k' ∨ i) (~ k' ∨ j) k))
(LemmaRU k') (LemmaRD k')) rb k'

ub : PathP (λ i → B (sq a i i0)) lub rub
ub = cong snd u
ub' : lub' = rub'
ub' i' = comp (λ k → B (spread (k A i) (k A j) (~ k ∨ i) (k A j) i'))
  (λ where
    k (i' = i0) → LemmaLU k
    k (i' = i1) → LemmaRU k) (ub i')
LemmaU : PathP (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (~ k' ∨ i) (k' A j) k)) (LemmaLU k')
(LemmaRU k')) ub ub'
LemmaU k' = transport-filler (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (~ k' ∨ i) (k' A j) k))
(LemmaLU k') (LemmaRU k')) ub k'

db : PathP (λ i → B (sq a i i1)) ldb rdb
db = cong snd d
db' : ldb' = rdb'
```

```
db' i' = comp (λ k → B (spread (k A i) (~ k ∨ j) (~ k ∨ i) (~ k ∨ j) i'))
  (λ where
    k (i' = i0) → LemmaLD k
    k (i' = i1) → LemmaRD k) (db i')
LemmaD : PathP (λ k' → PathP (λ k → B (spread (k' A i) (~ k' ∨ j) (~ k' ∨ i) (~ k' ∨ j) k)) (LemmaLD k')
(LemmaRD k')) db db'
LemmaD k' = transport-filler (λ k' → PathP (λ k → B (spread (k' A i) (~ k' ∨ j) (~ k' ∨ i) (~ k' ∨ j) k))
(LemmaLD k') (LemmaRD k')) db k'

sqb-hollow : (i' j' : I) → Partial (i' ∨ j' ∨ ~ i' ∨ ~ j') (B (sq a i' j'))
sqb-hollow i' j' (i' = i0) = l j' .snd
sqb-hollow i' j' (i' = i1) = r j' .snd
sqb-hollow i' j' (j' = i0) = u i' .snd
sqb-hollow i' j' (j' = i1) = d i' .snd

sqb'-hollow : (i' j' : I) → Partial (i' ∨ j' ∨ ~ i' ∨ ~ j') (B (sq a i j))
sqb'-hollow i' j' (i' = i0) = lb' j'
sqb'-hollow i' j' (i' = i1) = rb' j'
sqb'-hollow i' j' (j' = i0) = ub' i'
sqb'-hollow i' j' (j' = i1) = db' i'

sqb' : (i' j' : I) → (B (sq a i j)) [ (i' ∨ j' ∨ ~ i' ∨ ~ j') ⇒ sqb'-hollow i' j' ]
sqb' i' j' = inS (SqFillB (sq a i j) lb' rb' ub' db' i' j')
```

```
sqb : (i' j' : I) → (B (sq a i' j')) [ (i' ∨ j' ∨ ~ i' ∨ ~ j') ⇒ sqb-hollow i' j' ]
sqb i' j' = inS (comp (λ k → B (spread i j i' j' k)) (
  λ where
    k (i' = i0) → LemmaL (~ k) j'
    k (i' = i1) → LemmaR (~ k) j'
    k (j' = i0) → LemmaU (~ k) i'
    k (j' = i1) → LemmaD (~ k) i') (outS (sqb' i' j')))
```

How did the SqFill proofs go?

- Theorem (SqFill-Pi): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Pi(a : A).B a$. **Trivial!**
- Theorem (SqFill-Sigma): If $A : \text{Type}$ and $B : A \rightarrow \text{Type}$ has the SqFill property, then so does $\Sigma(a : A).B a$. **Very complicated.**

```

if_then_else_end : I → I → I → I
if i then j else k end = (k ∧ (~ i ∨ j)) ∨ ((i ∨ k) ∧ j)

{-# INLINE if_then_else_end #-}

SqFillSigmaAB : SqFill (λ [a ∈ A] B a)
SqFillSigmaAB l r u d i j .fst = SqFillA (cong fst l) (cong fst r) (cong fst u) (cong fst d) i j
SqFillSigmaAB {lu} {ld} l {ru} {rd} r u d i j .snd = outS (sqb i j)
  where
    sq = Square (cong fst l) (cong fst r) (cong fst u) (cong fst d)
    sq = SqFillA (cong fst l) (cong fst r) (cong fst u) (cong fst d)

spread : (i j i' j' : I) → sq a i j = sq a i' j'
spread i j i' j' k = sq (if k then i' else i end) (if k then j' else j end)

lub : B (sq a i0 i0)
lub = snd lu
lub' : B (sq a i j)
lub' = transport (λ k → B (spread i0 i0 i j k)) lub
LemmaLU : PathP (λ k → B (spread i0 i0 i j k)) lub lub'
LemmaLU k = transp (λ l → B (spread i0 i0 i j (k A l))) (~ k) lub

ldb : B (fst ld)
ldb = snd ld
ldb' : B (sq a i j)
ldb' = transport (λ k → B (spread i0 i1 i j k)) ldb
LemmaLD : PathP (λ k → B (spread i0 i1 i j k)) ldb ldb'
LemmaLD k = transp (λ l → B (spread i0 i1 i j (k A l))) (~ k) ldb

lb : PathP (λ k → B (spread i0 i0 i1 i k)) lub ldb
lb = cong snd l
lb' : PathP (λ k → B (spread i j i j k)) lub' ldb'
lb' j' = comp (λ k → B (spread (k A i) (k A j) (k A i) (~ k ∨ j) j'))
  (λ where
    k (j' = i0) → LemmaLU k
    k (j' = i1) → LemmaLD k) (lb j')
LemmaL : PathP (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (k' A i) (~ k' ∨ j) k)) (LemmaLU k')
(LemmaLD k')) lb lb'
LemmaL k' = transport-filler (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (k' A i) (~ k' ∨ j) k))
(LemmaLU k') (LemmaLD k')) lb k'


```

```

rub : B (fst ru)
rub = snd ru
rub' : B (sq a i j)
rub' = transport (λ k → B (spread i1 i0 i j k)) rub
LemmaRU : PathP (λ k → B (spread i1 i0 i j k)) rub rub'
LemmaRU k = transp (λ l → B (spread i1 i0 i j (k A l))) (~ k) rub

rdb : B (fst rd)
rdb = snd rd
rdb' : B (sq a i j)
rdb' = transport (λ k → B (spread i1 i1 i j k)) rdb
LemmaRD : PathP (λ k → B (spread i1 i1 i j k)) rdb rdb'
LemmaRD k = transp (λ l → B (spread i1 i1 i j (k A l))) (~ k) rdb

rb : PathP (λ j → B (sq a i1 j)) rub rdb
rb = cong snd r
rb' : rub' = rdb'
rb' j' = comp (λ k → B (spread (~ k ∨ i) (k A j) (~ k ∨ i) (~ k ∨ j) j'))
  (λ where
    k (j' = i0) → LemmaRU k
    k (j' = i1) → LemmaRD k) (rb j')
LemmaR : PathP (λ k' → PathP (λ k → B (spread (~ k' ∨ i) (k' A j) (~ k' ∨ i) (~ k' ∨ j) k)) (LemmaRU k')
(LemmaRD k')) rb rb'
LemmaR k' = transport-filler (λ k' → PathP (λ k → B (spread (~ k' ∨ i) (k' A j) (~ k' ∨ i) (~ k' ∨ j) k))
(LemmaRU k') (LemmaRD k')) rb k'

ub : PathP (λ i → B (sq a i i0)) lub rub
ub = cong snd u
ub' : lub' = rub'
ub' i' = comp (λ k → B (spread (k A i) (k A j) (~ k ∨ i) (k A j) i'))
  (λ where
    k (i' = i0) → LemmaLU k
    k (i' = i1) → LemmaRU k) (ub i')
LemmaU : PathP (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (~ k' ∨ i) (k' A j) k)) (LemmaLU k')
(LemmaRU k')) ub ub'
LemmaU k' = transport-filler (λ k' → PathP (λ k → B (spread (k' A i) (k' A j) (~ k' ∨ i) (k' A j) k))
(LemmaLU k') (LemmaRU k')) ub k'

db : PathP (λ i → B (sq a i i1)) ldb rdb
db = cong snd d
db' : ldb' = rdb'

```

```

db' i' = comp (λ k → B (spread (k A i) (~ k ∨ j) (~ k ∨ i) (~ k ∨ j) i'))
  (λ where
    k (i' = i0) → LemmaLD k
    k (i' = i1) → LemmaRD k) (db i')
LemmaD : PathP (λ k' → PathP (λ k → B (spread (k' A i) (~ k' ∨ j) (~ k' ∨ i) (~ k' ∨ j) k)) (LemmaLD k')
(LemmaRD k')) db db'
LemmaD k' = transport-filler (λ k' → PathP (λ k → B (spread (k' A i) (~ k' ∨ j) (~ k' ∨ i) (~ k' ∨ j) k))
(LemmaLD k') (LemmaRD k')) db k'

sqb-hollow : (i' j' : I) → Partial (i' ∨ j' ∨ ~ i' ∨ ~ j') (B (sq a i' j'))
sqb-hollow i' j' (i' = i0) = l j' .snd
sqb-hollow i' j' (i' = i1) = r j' .snd
sqb-hollow i' j' (j' = i0) = u i' .snd
sqb-hollow i' j' (j' = i1) = d i' .snd

sqb'-hollow : (i' j' : I) → Partial (i' ∨ j' ∨ ~ i' ∨ ~ j') (B (sq a i j))
sqb'-hollow i' j' (i' = i0) = lb' j'
sqb'-hollow i' j' (i' = i1) = rb' j'
sqb'-hollow i' j' (j' = i0) = ub' i'
sqb'-hollow i' j' (j' = i1) = db' i'

sqb' : (i' j' : I) → (B (sq a i j)) [ (i' ∨ j' ∨ ~ i' ∨ ~ j') ⇒ sqb'-hollow i' j' ]
sqb' i' j' = inS (SqFillB (sq a i j) lb' rb' ub' db' i' j')

sqb : (i' j' : I) → (B (sq a i' j')) [ (i' ∨ j' ∨ ~ i' ∨ ~ j') ⇒ sqb-hollow i' j' ]
sqb i' j' = inS (comp (λ k → B (spread i j i' j' k)) (
  λ where
    k (i' = i0) → LemmaL (~ k) j'
    k (i' = i1) → LemmaR (~ k) j'
    k (j' = i0) → LemmaU (~ k) i'
    k (j' = i1) → LemmaD (~ k) i') (outS (sqb' i' j'))))

```

SqFill: Summary

	SqFill (<i>homogeneous</i> Square-Filling)
Pi	Trivial (no Kan operations)
Sigma	Complicated : transport-fill-align
Coproducts	Standard encode-decode proof (J , hcomp)
Path Types	Simple (a single hcomp)

SqFill-Sigma was complicated because the second projection (dependent) was **heterogeneous!**

How did the SqPFill proofs go?

- Bad news: SqPFill-Pi is now exceedingly hard (was trivial).
 - The inverse problem of SqFill-Sigma previously: we have a *homogeneous* square to fill from a **heterogeneous** SqPFill assumption...

How did the SqPFill proofs go?

- Bad news: SqPFill-Pi is now exceedingly hard (was trivial).
 - The inverse problem of SqFill-Sigma previously: we have a *homogeneous* square to fill from a **heterogeneous** SqPFill assumption...
- Good news: SqPFill-Sigma is **trivial**. (Great!)

How did the SqPFill proofs go?

- Bad news: SqPFill-Pi is now exceedingly hard (was trivial).
 - The inverse problem of SqFill-Sigma previously: we have a *homogeneous* square to fill from a **heterogeneous** SqPFill assumption...
- Good news: SqPFill-Sigma is **trivial**. (Great!)
- Okay news: SqPFill-Coproduct follows exactly the same way (encode-decode).

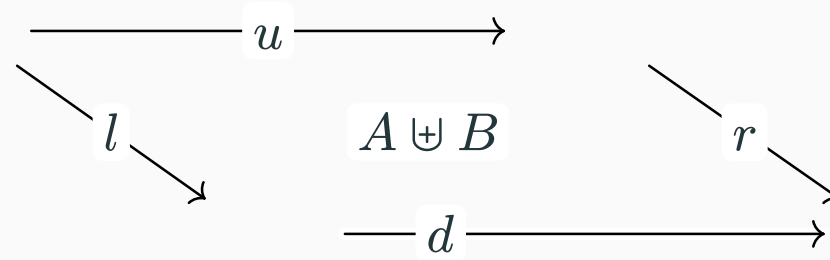
How did the SqPFill proofs go?

- Bad news: **SqPFill**-Pi is now exceedingly hard (was trivial).
 - ▶ The inverse problem of **SqFill**-Sigma previously: we have a *homogeneous* square to fill from a **heterogeneous** **SqPFill** assumption...
- Good news: **SqPFill**-Sigma is **trivial**. (Great!)
- Okay news: **SqPFill**-Coproduct follows exactly the same way (encode-decode).
- Bad news: **SqPFill**-Path is now very complicated (was just one **hcomp**)... or a more unconventional induction hypothesis (similar to course-of-values induction)

Full proofs: clickable HTML version of Agda  proofs + diagrams in report.

SqFill for Coproducts and Paths

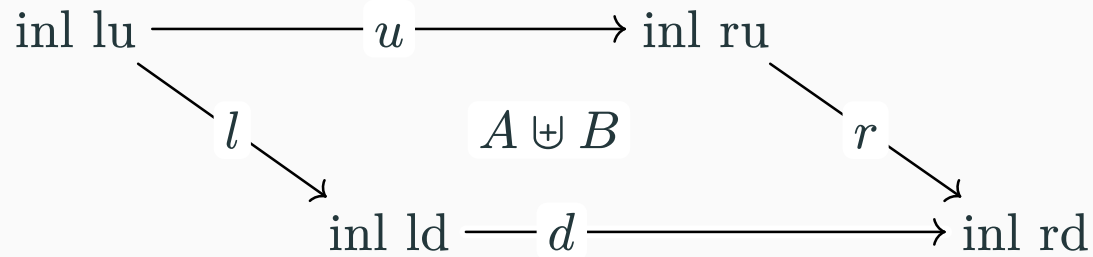
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Encoding an `inl` hollow square from $A \uplus B$ to A .

SqFill for Coproducts and Paths

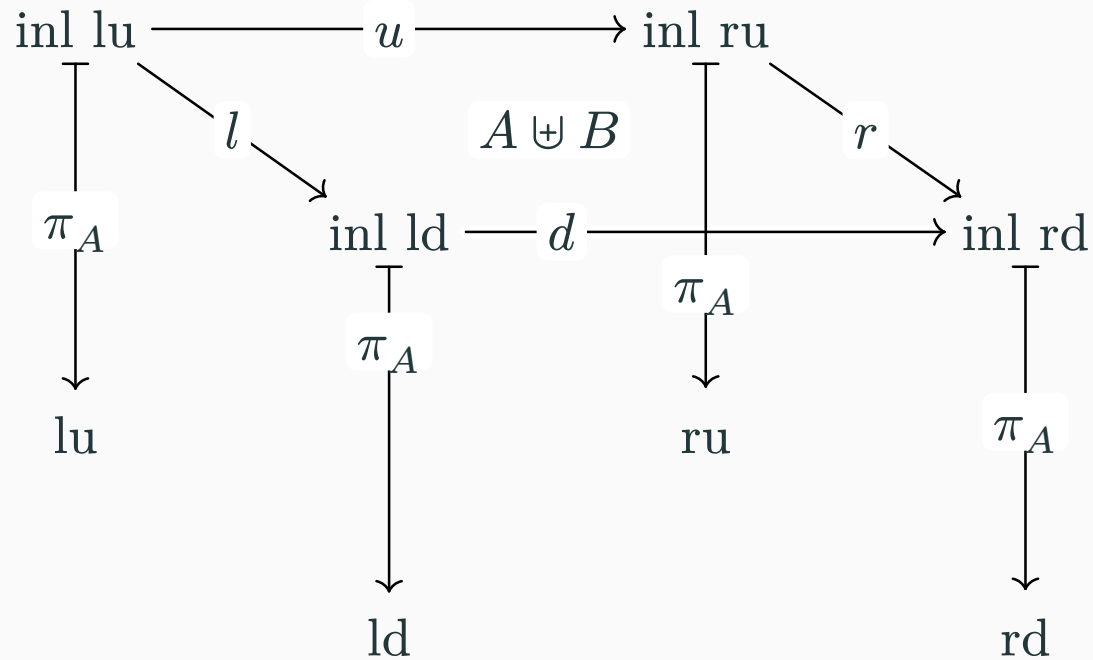
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Encoding an inl hollow square from $A \uplus B$ to A .

SqFill for Coproducts and Paths

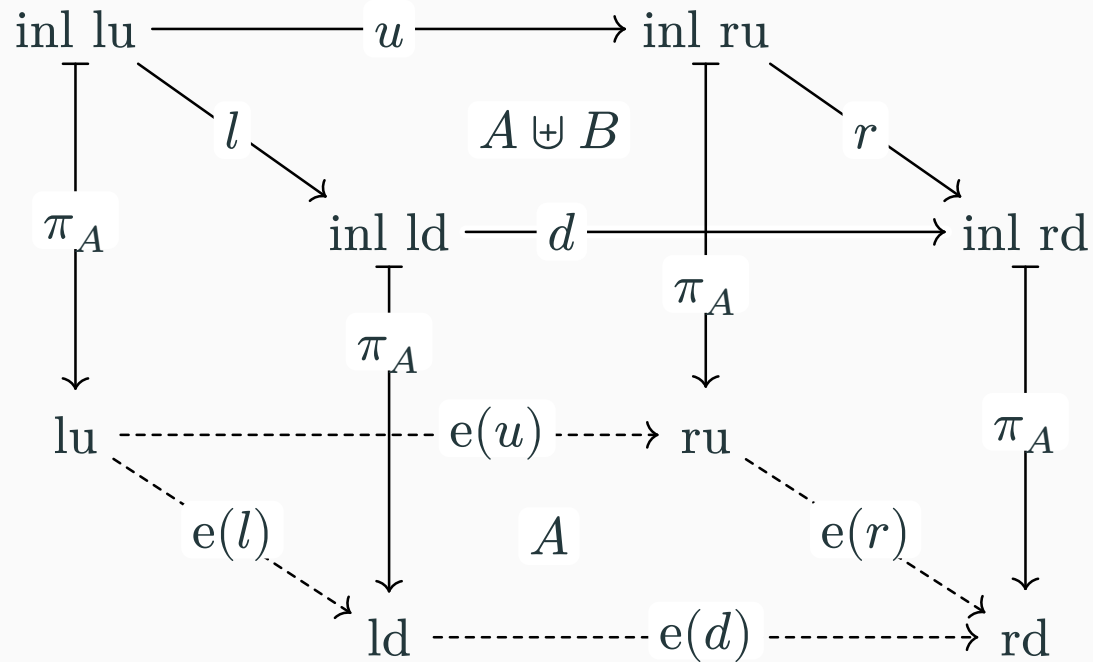
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Encoding an `inl` hollow square from $A \uplus B$ to A .

SqFill for Coproducts and Paths

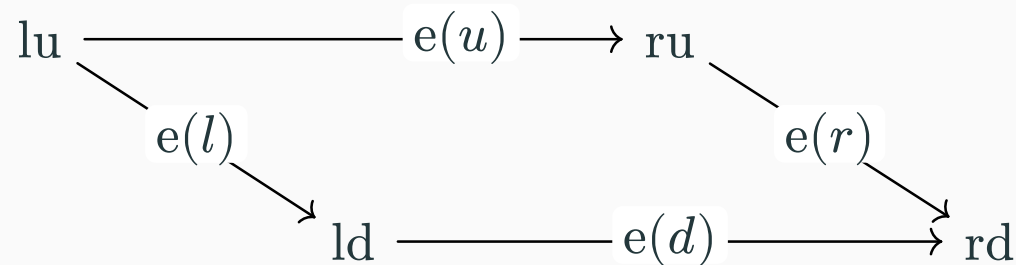
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Encoding an `inl` hollow square from $A \uplus B$ to A .

SqFill for Coproducts and Paths

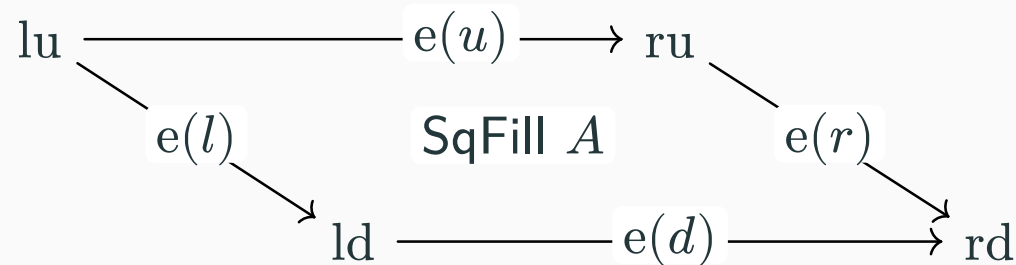
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Decoding (applying `inl` to) the filled square in A back to $A \uplus B$.

SqFill for Coproducts and Paths

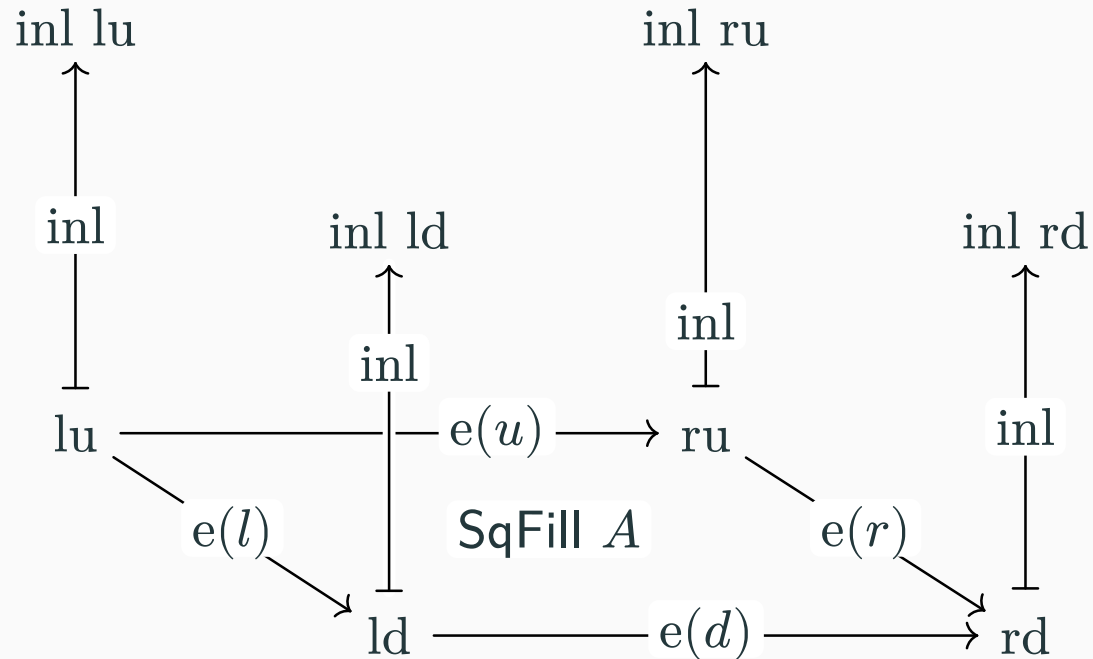
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Decoding (applying `inl` to) the filled square in A back to $A \uplus B$.

SqFill for Coproducts and Paths

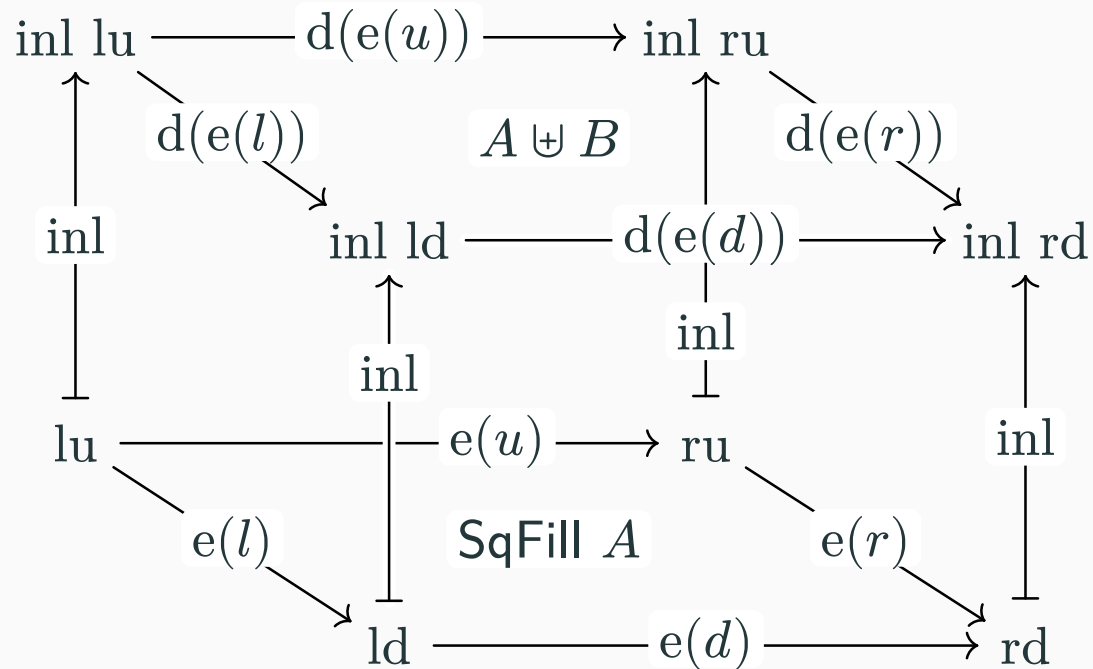
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Decoding (applying inl to) the filled square in A back to $A \uplus B$.

SqFill for Coproducts and Paths

- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Decoding (applying inl to) the filled square in A back to $A \uplus B$.

SqFill for Coproducts and Paths

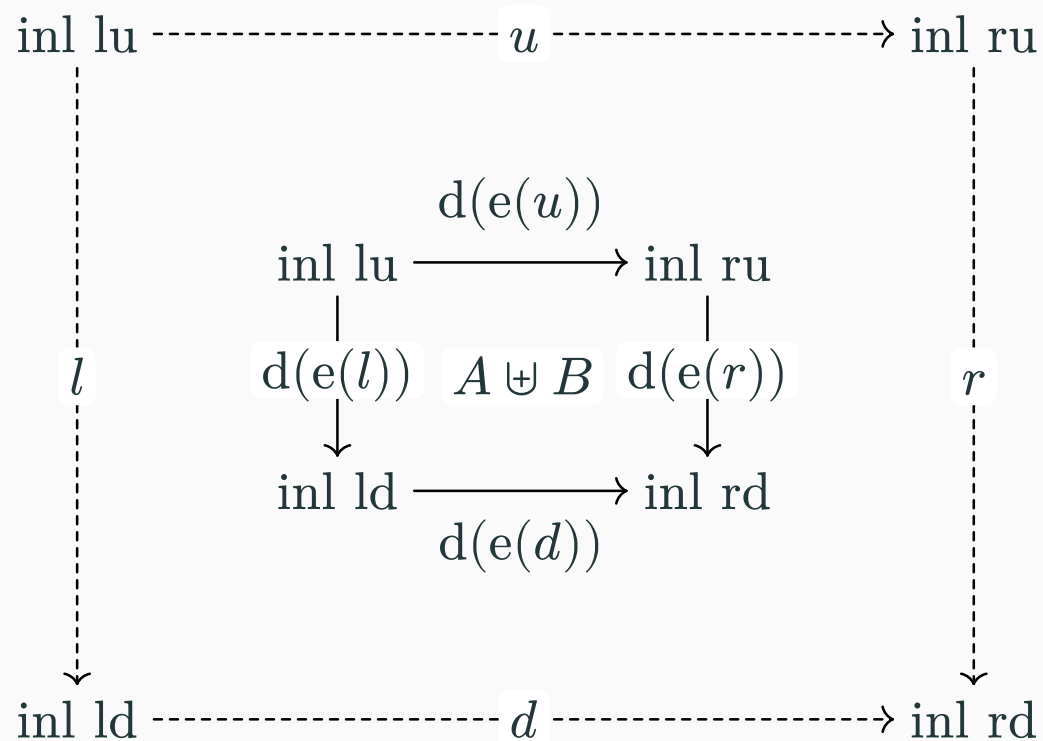
- Theorem (SqFill-Coproduct): If A B : Type have the SqFill property, then so does $A \uplus B$.

$$\begin{array}{ccc} & d(e(u)) & \\ & \longrightarrow & \\ \text{inl } lu & & \text{inl } ru \\ | & & | \\ d(e(l)) & A \uplus B & d(e(r)) \\ \downarrow & & \downarrow \\ \text{inl } ld & \longrightarrow & \text{inl } rd \\ & d(e(d)) & \end{array}$$

Aligning by hcomp along $d(e(p)) \equiv p$.

SqFill for Coproducts and Paths

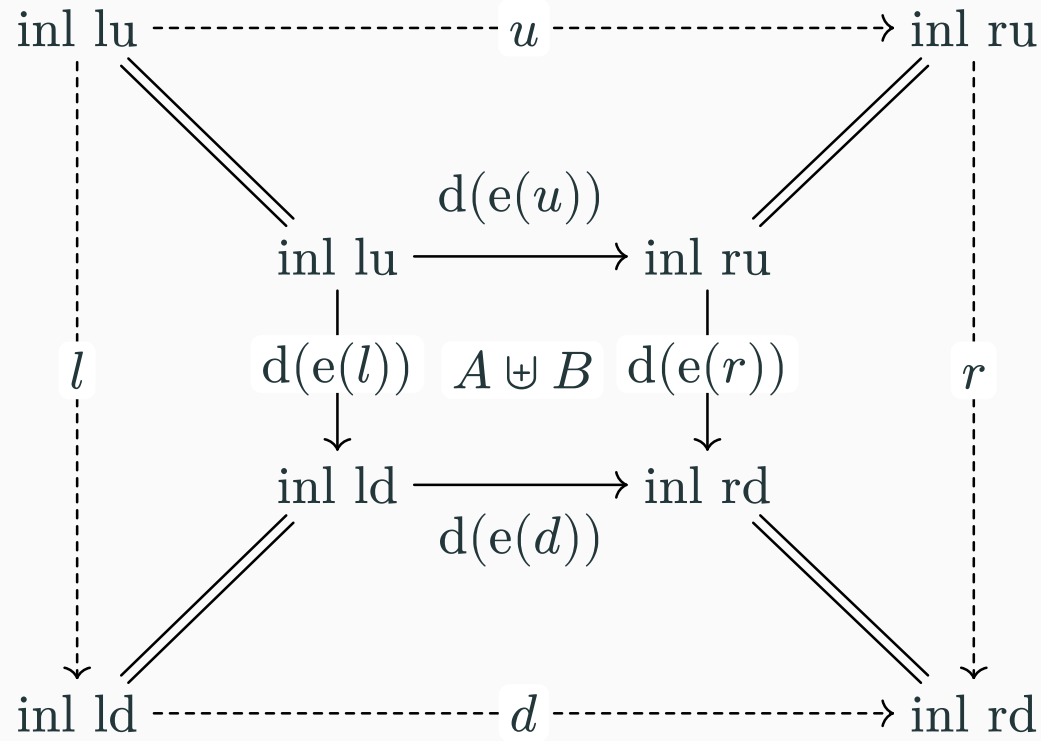
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Aligning by hcomp along $d(e(p)) \equiv p$.

SqFill for Coproducts and Paths

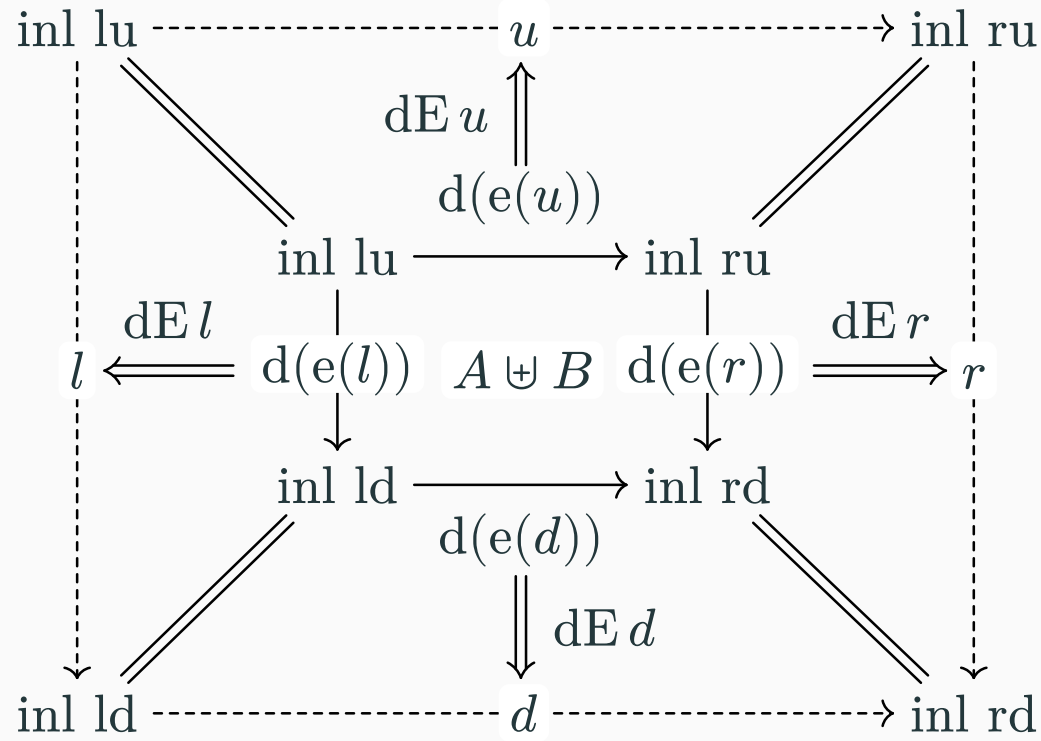
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Aligning by hcomp along $d(e(p)) \equiv p$.

SqFill for Coproducts and Paths

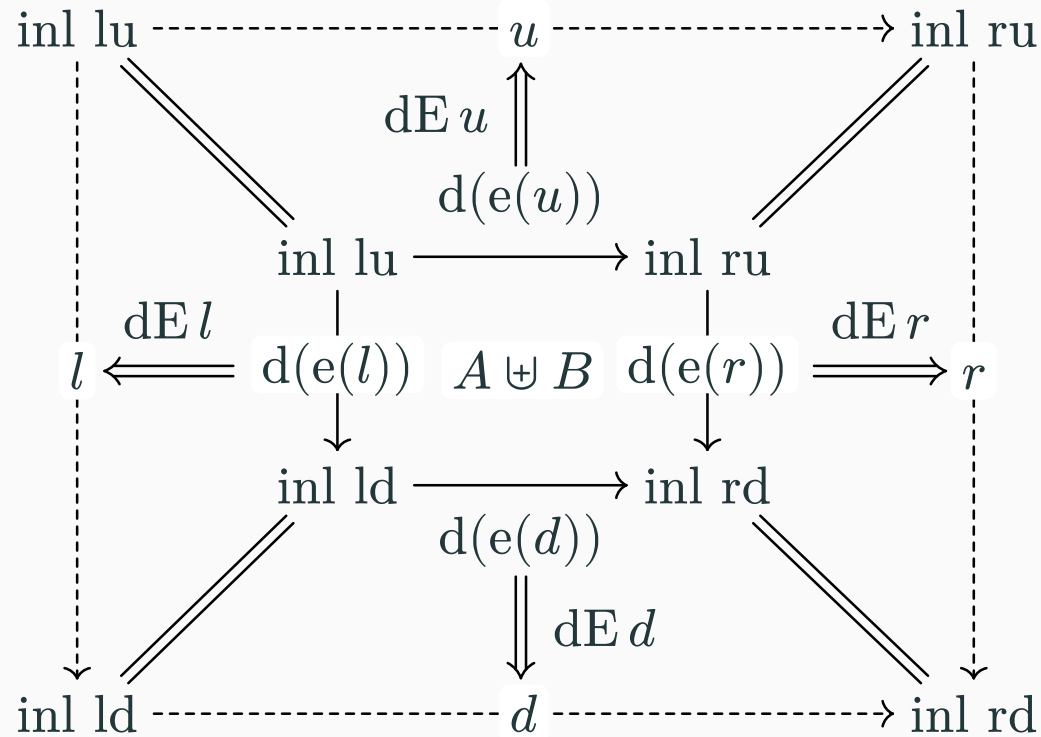
- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$.



Aligning by hcomp along $d(e(p)) \equiv p$.

SqFill for Coproducts and Paths

- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$. Classic **encode-decode** proof [MGM04, Uni13].



Aligning by **hcomp** along $d(e(p)) \equiv p$.

SqFill for Coproducts and Paths

- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$. Classic **encode-decode** proof [MGM04, Uni13].

```
Cover : {A B : Type} (c c' : A + B) → Type
```

```
Cover (inl x) (inl y) = x ≡ y
```

```
Cover (inr x) (inr y) = x ≡ y
```

```
Cover _ _ = ⊥
```

```
reflCode : {A B : Type} (c : A + B) → Cover c c
```

```
reflCode (inl x) = refl
```

```
reflCode (inr x) = refl
```

```
encode : {A B : Type} {c c' : A + B} → c ≡ c' → Cover c c'
```

```
encode {c = c} p = transport (λ i → Cover c (p i)) (reflCode c)
```

```
decode : {A B : Type} {c c' : A + B} → Cover c c' → c ≡ c'
```

```
decode {c = inl x} {c' = inl y} = cong inl
```

```
decode {c = inr x} {c' = inr y} = cong inr
```

```
decodeEncode : {A B : Type} {c c' : A + B} (p : c ≡ c') → decode (encode p) ≡ p
```

```
decodeEncode {c = inl x} = J (λ c' p → decode (encode p) ≡ p) (cong (cong inl) (transportRef1 refl))
```

```
decodeEncode {c = inr x} = J (λ c' p → decode (encode p) ≡ p) (cong (cong inr) (transportRef1 refl))
```

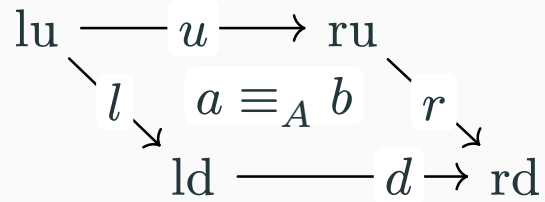
SqFill for Coproducts and Paths

- Theorem (SqFill-Coproduct): If $A B : \text{Type}$ have the SqFill property, then so does $A \uplus B$. Classic **encode-decode** proof [MGM04, Uni13].

```
SqFillCoproduct : SqFill (A + A')
SqFillCoproduct {inl lu} {inl ld} l {inl ru} {inl rd} r u d i j =
  (hcomp (λ where
    k (i = i0) → decodeEncode l k j
    k (i = i1) → decodeEncode r k j
    k (j = i0) → decodeEncode u k i
    k (j = i1) → decodeEncode d k i
    (inl {A} {A'} (SqFillA (encode l) (encode r) (encode u) (encode d) i j))))
SqFillCoproduct {inr lu} {inr ld} l {inr ru} {inr rd} r u d i j =
  (hcomp (λ where
    k (i = i0) → decodeEncode l k j
    k (i = i1) → decodeEncode r k j
    k (j = i0) → decodeEncode u k i
    k (j = i1) → decodeEncode d k i
    (inr {A} {A'} (SqFillA' (encode l) (encode r) (encode u) (encode d) i j))))
  SqFillCoproduct {inl x} {inr y} l _ _ _ = 1-elim (inl#inr x y l)
SqFillCoproduct {inr x} {inl y} l _ _ _ = 1-elim (inl#inr y x (sym l))
SqFillCoproduct {inl x} {inl y} _ {inr y} _ u _ = 1-elim (inl#inr x y u)
SqFillCoproduct {inr x} {inl y} _ {inl y} _ u _ = 1-elim (inl#inr y x (sym u))
SqFillCoproduct {inl x} _ {inl y} _ {inr y} _ _ d = 1-elim (inl#inr x y d)
SqFillCoproduct {inl x} _ {inl y} _ _ d = 1-elim (inl#inr y x (sym d))
```

SqFill for Coproducts and Paths

- Theorem (**SqFill-Coproduct**): If $A B : \text{Type}$ have the **SqFill** property, then so does $A \uplus B$. Classic **encode-decode** proof [MGM04, Uni13].
- Theorem (**SqFill-Path**): If $A : \text{Type}$ has the **SqFill** property, then for any $a b : A$, the path type $a \equiv_A b$ also has the **SqFill** property.



SqFill for Coproducts and Paths

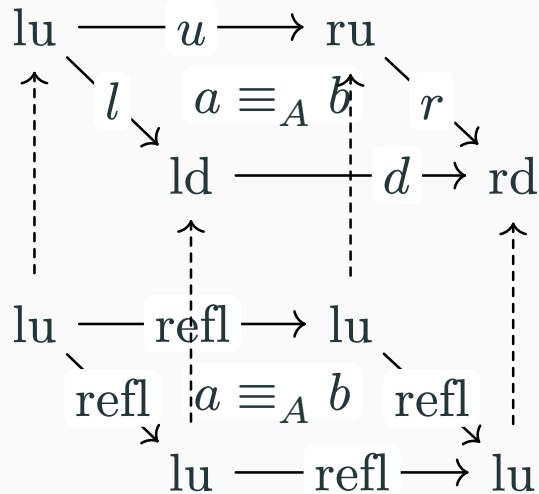
- Theorem (**SqFill**-Coproduct): If $A B : \text{Type}$ have the **SqFill** property, then so does $A \uplus B$. Classic **encode-decode** proof [MGM04, Uni13].
- Theorem (**SqFill**-Path): If $A : \text{Type}$ has the **SqFill** property, then for any $a b : A$, the path type $a \equiv_A b$ also has the **SqFill** property.

$$\begin{array}{ccccc} \text{lu} & \xrightarrow{u} & \text{ru} & & \\ & \searrow l & & \swarrow r & \\ & & a \equiv_A b & & \\ & & \downarrow & & \\ & & \text{ld} & \xrightarrow{d} & \text{rd} \end{array}$$

$$\begin{array}{ccccc} \text{lu} & \xrightarrow{\text{refl}} & \text{lu} & & \\ & \searrow \text{refl} & & \swarrow \text{refl} & \\ & & a \equiv_A b & & \\ & & \downarrow & & \\ & & \text{lu} & \xrightarrow{\text{refl}} & \text{lu} \end{array}$$

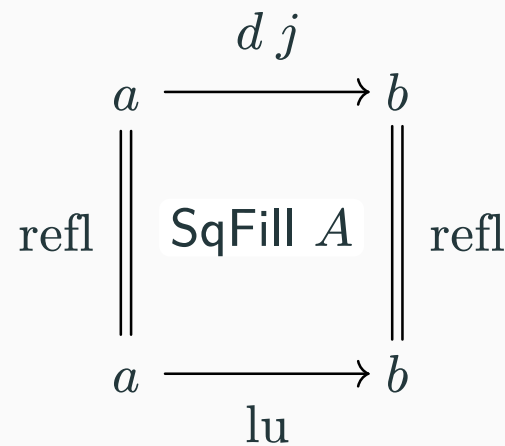
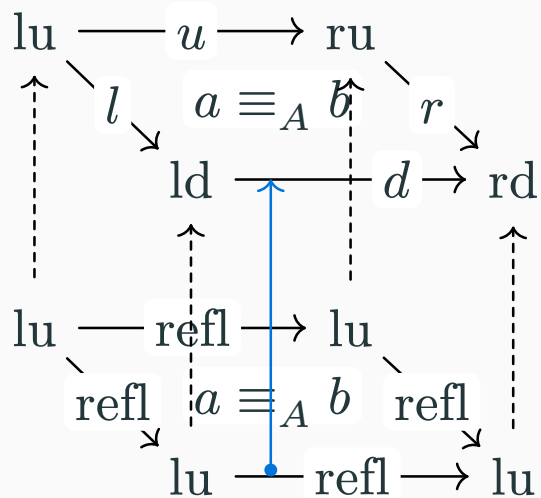
SqFill for Coproducts and Paths

- Theorem (**SqFill**-Coproduct): If $A B : \text{Type}$ have the **SqFill** property, then so does $A \uplus B$. Classic **encode-decode** proof [MGM04, Uni13].
- Theorem (**SqFill**-Path): If $A : \text{Type}$ has the **SqFill** property, then for any $a b : A$, the path type $a \equiv_A b$ also has the **SqFill** property.



SqFill for Coproducts and Paths

- Theorem (**SqFill-Coproduct**): If $A B : \text{Type}$ have the **SqFill** property, then so does $A \uplus B$. Classic **encode-decode** proof [MGM04, Uni13].
- Theorem (**SqFill-Path**): If $A : \text{Type}$ has the **SqFill** property, then for any $a b : A$, the path type $a \equiv_A b$ also has the **SqFill** property.



isProp: Every line on the sides is a square in A , which follows from the **SqFill** A assumption.

SqFill for Coproducts and Paths

- Theorem (**SqFill-Coproduct**): If $A B : \text{Type}$ have the **SqFill** property, then so does $A \uplus B$. Classic **encode-decode** proof [MGM04, Uni13].
- Theorem (**SqFill-Path**): If $A : \text{Type}$ has the **SqFill** property, then for any $a b : A$, the path type $a \equiv_A b$ also has the **SqFill** property. **Simple**.

```
SqFillPath : {a b : A} → SqFill (a ≡ b)
```

```
SqFillPath {_} {_} {lu} l r u d i j =  
  hcomp (λ k → λ {(i = i0) → isPropa≡b lu (l j) k  
    ; (i = i1) → isPropa≡b lu (r j) k  
    ; (j = i0) → isPropa≡b lu (u i) k  
    ; (j = i1) → isPropa≡b lu (d i) k}) lu
```

where

```
isPropa≡b : {a b : A} (p q : a ≡ b) → p ≡ q  
isPropa≡b p q = SqFillA p q refl refl
```

Summary and Observations

	SqFill (<i>homogeneous</i> Square-Filling)	SqPFill (<i>heterogeneous</i> Square-Filling)
Pi	Trivial (no Kan operations)	Complicated : transport-fill-align ^{*†}
Sigma	Complicated : transport-fill-align [*]	Trivial (no Kan operations)
Coproducts	Standard encode-decode proof (J , irregularity [‡] , hcomp)	Standard encode-decode proof (J , irregularity [‡] , comp)
Path Types	Simple (a single hcomp)	Complicated : transport-fill-align ^{*^}

The proofs can be simplified if..

^{*} : the equality function was definable in a de Morgan algebra (specifically I) (?)

[†] : the $\text{coe}_k(i_0, i_1)$ function needs to have eta: $\text{coe}_k(i, i) = i$ at all $k : I$.

[‡] : irregularity [Swa18] in CubTT (very slightly) complicates the encode-decode proof.

[^] : trivial if using a stronger (course-of-values style) induction hypothesis.

Conclusion and Future Work

Conclusion

Contributions

- Implementation of a `--cubical=no-glue` variant in Agda.
- Computational UIP by “pushing through” type formers.
- **SqFill** and **SqPFill** as “generalisations” of UIP.

Preservation proofs for Π , Σ , Coproduct, and Path types in `--cubical=no-glue`.

Future Work

- Implement `--cubical=uij` (WIP)
- Preservation by inductive types (W -Types) and heterogeneous path types **PathP**.
 - **PathP** just slightly more heterogeneous than path types.
- Show nice properties of the resulting theory: Canonicity, Normalisation...

Open Questions

- Can we allow pattern matching with K on (Inductive) Identity types in `--cubical={no-glue/uip}`?
- What constitutes a **good** computational rule? How to measure which term is better?
- Which **SqFill** property to implement? (or, are Sigmas or Pis more common?)
- How to treat the Universe case such that e.g. `HCompU` behaves meaningfully for both Cubical-UIP and Cubical-Univalence?

Thank you!

Implementation in progress, feel free to join!

XTT [SAG22]

- Cubical Type Theory (without Glue Types) with definitional UIP: two paths are *definitionally* equal if they have the same endpoints
- Requires a non-standard universe where type constructors are injective up to *paths*
- also possible to show the consistency of our theory by a translation into XTT.

Setoid Type Theory [Alt99, Alt+19, Hof95]

- add functional extensionality, propositional extensionality, and quotient types to intensional type theory
- “More observational” than CubTT + UIP: equality between pairs is *definitionally equal* to the pointwise equalities of the first and second components, but only an isomorphism in Cubical Type Theories.

References

- [Agd25] The Agda Community. 2025. Cubical Agda Library. Retrieved from <https://github.com/agda/cubical>
- [Alt+19] Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. 2019. Setoid Type Theory - A Syntactic Translation. In *Mathematics of Program Construction - 13th International Conference, MPC 2019, Porto, Portugal, October 7-9, 2019, Proceedings (Lecture Notes in Computer Science)*, 2019. Springer, 155–196. https://doi.org/10.1007/978-3-030-33636-3_7
- [Alt99] Thorsten Altenkirch. 1999. Extensional Equality in Intensional Type Theory. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, 1999. IEEE Computer Society, 412–420. <https://doi.org/10.1109/LICS.1999.782636>
- [Coc19] Jesper Cockx. 2019. Comment: A variant of Cubical Agda that is consistent with UIP. Retrieved from <https://github.com/agda/agda/issues/3750#issuecomment-490070548>
- [Coh+17] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. 2017. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. *FLAP* 4, 10 (2017), 3127–3170. Retrieved from <http://collegepublications.co.uk/ifcolog/?00019>
- [HS94] Martin Hofmann and Thomas Streicher. 1994. The Groupoid Model Refutes Uniqueness of Identity Proofs. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94)*,

References

Paris, France, July 4-7, 1994, 1994. IEEE Computer Society, 208–212. <https://doi.org/10.1109/LICS.1994.316071>

[Hof95] Martin Hofmann. 1995. A Simple Model for Quotient Types. In *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA '95, Edinburgh, UK, April 10-12, 1995, Proceedings (Lecture Notes in Computer Science)*, 1995. Springer, 216–234. <https://doi.org/10.1007/BFB0014055>

[Hof97] Martin Hofmann. 1997. Syntax and Semantics of Dependent Types. In *Semantics and Logics of Computation*, Andrew M. Pitts and P.Editors Dybjer (eds.). Cambridge University Press, 79–130.

[MGM04] Conor McBride, Healdene Goguen, and James McKinna. 2004. A Few Constructions on Constructors. In *Types for Proofs and Programs, International Workshop, TYPES 2004, Jouy-en-Josas, France, December 15-18, 2004, Revised Selected Papers (Lecture Notes in Computer Science)*, 2004. Springer, 186–200. https://doi.org/10.1007/11617990_12

[Pit20] Andrew Pitts. 2020. Comment: A variant of Cubical Agda that is consistent with UIP. Retrieved from <https://github.com/agda/agda/issues/3750#issuecomment-575735235>

[Shu17] Michael Shulman. 2017. cubical type theory with UIP. Retrieved from <https://groups.google.com/g/homotopytypetheory/c/UegPjMrnx6I/m/UNBLHfZNBAAJ>

References

- [SAG22] Jonathan Sterling, Carlo Angiuli, and Daniel Gratzer. 2022. A Cubical Language for Bishop Sets. *Log. Methods Comput. Sci.* 18, 1 (2022). [https://doi.org/10.46298/LMCS-18\(1:43\)2022](https://doi.org/10.46298/LMCS-18(1:43)2022)
- [Swa18] Andrew Swan. 2018. Separating Path and Identity Types in Presheaf Models of Univalent Type Theory. Retrieved from <https://arxiv.org/abs/1808.00920>
- [Uni13] The Univalent Foundations Program. 2013. *Homotopy Type Theory: Univalent Foundations of Mathematics*. <https://homotopytypetheory.org/book>, Institute for Advanced Study. Retrieved from <https://homotopytypetheory.org/book>
- [VMA21] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. 2021. Cubical Agda: A dependently typed programming language with univalence and higher inductive types. *J. Funct. Program.* 31, (2021), e8. <https://doi.org/10.1017/S0956796821000034>