Cubical Agda

Samuel Mimram

2025

École polytechnique

In order to define higher inductive types in practice, we need to explain the theory behind cubical Agda

Canonicity

Homotopy type theory is obtained by adding a new axiom, **univalence**, stating that the map

$$(A = B) \to (A \simeq B)$$

is an equivalence.

This can be performed axiomatically, by adding a new axiom

```
postulate
  isEquivPathToEquiv :
    {A B : Type ℓ} → isEquiv (pathToEquiv {A = A} {B = B})
```

Canonicity

Homotopy type theory is obtained by adding a new axiom, **univalence**, stating that the map

$$(A = B) \to (A \simeq B)$$

is an equivalence.

This can be performed axiomatically, by adding a new axiom

```
postulate
  isEquivPathToEquiv :
    {A B : Type ℓ} → isEquiv (pathToEquiv {A = A} {B = B})
```

but this has a defect: it breaks **canonicity**.

Canonicity

Definition
A type theory has the **canonicity** property when every term $t : \mathbb{N}$ is convertible to a natural number.

Canonicity

Definition
A type theory has the **canonicity** property when every term $t : \mathbb{N}$ is convertible to a natural number.

Note: we could have taken other types instead of $\mathbb{N}$ (e.g. the booleans).

Canonicity

Definition
A type theory has the **canonicity** property when every term $t : \mathbb{N}$ is convertible to a natural number.

Note: we could have taken other types instead of $\mathbb{N}$ (e.g. the booleans).

Theorem
Dependent type theory (without univalence) has the canonicity property.

Proof.
We can orient definitional equality into terminating rewriting system, e.g.

$$3 + \mathsf{id}\, 2 \to 3 + 2 \to 5$$

The only terms in normal form of type $\mathbb{N}$ are natural numbers. □

Canonicity

Adding axioms (such as univalence) breaks canonicity:

- consider the function $\sigma : \mathsf{List}\,1 \to \mathsf{List}\,1$ defined by

$$\sigma(l) \quad \hat{=} \quad \star :: l$$

Canonicity

Adding axioms (such as univalence) breaks canonicity:

- consider the function $\sigma : \mathsf{List}\,1 \to \mathsf{List}\,1$ defined by

$$\sigma(l) \quad \hat{=} \quad \star :: l$$

- we have

$$e : \mathsf{List}\ 1 \simeq \mathbb{N}$$

Canonicity

Adding axioms (such as univalence) breaks canonicity:

- consider the function $\sigma : \mathsf{List}\,1 \to \mathsf{List}\,1$ defined by

$$\sigma(l) \quad \hat{=} \quad \star :: l$$

- we have

$$e : \mathsf{List}\ 1 \simeq \mathbb{N}$$

- by univalence we deduce

$$\mathsf{ua}\, e : \mathsf{List}\ 1 = \mathbb{N}$$

Canonicity

Adding axioms (such as univalence) breaks canonicity:

- consider the function $\sigma : \mathsf{List}\,1 \to \mathsf{List}\,1$ defined by
$$\sigma(l) \quad \hat{=} \quad \star :: l$$
- we have
$$e : \mathsf{List}\ 1 \simeq \mathbb{N}$$
- by univalence we deduce
$$\mathsf{ua}\ e : \mathsf{List}\ 1 = \mathbb{N}$$
- by transport we have a function $s : \mathbb{N} \to \mathbb{N}$ defined by
$$s \quad \hat{=} \quad \mathsf{transport}^{X \mapsto X \to X}(\mathsf{ua}\ e)\,\sigma$$

Canonicity

Adding axioms (such as univalence) breaks canonicity:

- consider the function $\sigma : \mathsf{List}\,1 \to \mathsf{List}\,1$ defined by
$$\sigma(l) \quad \hat{=} \quad \star :: l$$
- we have
$$e : \mathsf{List}\ 1 \simeq \mathbb{N}$$
- by univalence we deduce
$$\mathsf{ua}\ e : \mathsf{List}\ 1 = \mathbb{N}$$
- by transport we have a function $s : \mathbb{N} \to \mathbb{N}$ defined by
$$s \quad \hat{=} \quad \mathsf{transport}^{X \mapsto X \to X}(\mathsf{ua}\ e)\,\sigma$$

Brunerie's number

A famous example comes from Brunerie's PhD thesis [Bru16] who shows in HoTT

$$\Sigma(n : \mathbb{N}).(\pi_4(\mathsf{S}^3) \simeq \mathbb{Z}/n\mathbb{Z})$$

Corollary 3.4.5. *We have*

$$\pi_4(\mathbb{S}^3) \simeq \mathbb{Z}/n\mathbb{Z},$$

where n is the absolute value of the image of $[i_2, i_2]$ by the equivalence $\pi_3(\mathbb{S}^2) \xrightarrow{\sim} \mathbb{Z}$.

This result is quite remarkable in that even though it is a constructive proof, it is not at all obvious how to actually compute this n. At the time of writing, we still haven't managed to extract its value from its definition. A complete and concise definition of this number n is presented in appendix B, for the benefit of someone wanting to implement it in a prospective proof assistant. In the rest of this thesis, we give a mathematical proof in homotopy type theory that $n = 2$.

Brunerie's number

A famous example comes from Brunerie's PhD thesis [Bru16] who shows in HoTT

$$\Sigma(n : \mathbb{N}).(\pi_4(\mathsf{S}^3) \simeq \mathbb{Z}/n\mathbb{Z})$$

Actually computing the natural number n was achieved by Ljungström and Mörtberg [LM23] who show that $n = \ldots$

Brunerie's number

A famous example comes from Brunerie's PhD thesis [Bru16] who shows in HoTT

$$\Sigma(n : \mathbb{N}).(\pi_4(S^3) \simeq \mathbb{Z}/n\mathbb{Z})$$

Actually computing the natural number n was achieved by Ljungström and Mörtberg [LM23] who show that $n = -2$.

Cubical type theory

In 2015, Cohen, Coquand, Huber and Mörtberg [CCHM18] presented a variant of dependent type theory in which

- one can manipulate n-cubes
- one can prove the univalence axiom
- supports higher inductive types [CHM18]
- has the canonicity property [Hub19]

Cubical type theory

The current implementation of cubical Agda is based on this
it can be activated with `--cubical`.

It is described in [VMA21] as well as in Agda's reference manual.

It comes with an alternative **standard library**:
`https://github.com/agda/cubical`

Cubical type theory

The cubical type theory can be considered as an "assembly language" for HoTT:

- most of the time we don't need to understand those details in order to make proofs (we only use the cubical Agda library)
- implementation might actually change in the future while keeping most proofs valid

But it is sometimes useful to know how it works.

The interval type

We begin by adding a new type I for the **interval** with two constructors $\mathsf{i}_0 : \mathsf{I}$ and $\mathsf{i}_1 : \mathsf{I}$

The idea is that a path in A corresponds to a function $\mathsf{I} \to A$.

The interval type

We begin by adding a new type I for the **interval** with two constructors $\mathsf{i}_0 : \mathsf{I}$ and $\mathsf{i}_1 : \mathsf{I}$

The idea is that a path in A corresponds to a function $\mathsf{I} \to A$.

More generally,

- a term $I \to I \to A$ corresponds to a square in A,
- a term $I \to I \to I \to A$ to a cube in A,
- etc.

This is why we are **cubical**.

The interval type

We begin by adding a new type I for the **interval** with two constructors $\mathsf{i}_0 : \mathsf{I}$ and $\mathsf{i}_1 : \mathsf{I}$

The idea is that a path in A corresponds to a function $\mathsf{I} \to A$.

More generally,

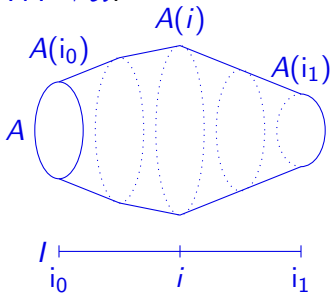
- a term $I \to I \to A$ corresponds to a square in A,
- a term $I \to I \to I \to A$ to a cube in A,
- etc.

This is why we are **cubical**.

We also want to consider paths $(i : \mathsf{I}) \to A(i)$ whose type is varying.

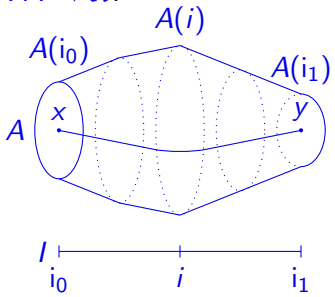
Paths

A *varying type* is a function $A : \mathsf{I} \to \mathcal{U}$:



Paths

A *varying type* is a function $A : \mathsf{I} \to \mathcal{U}$:



The type of *heterogeneous paths* is

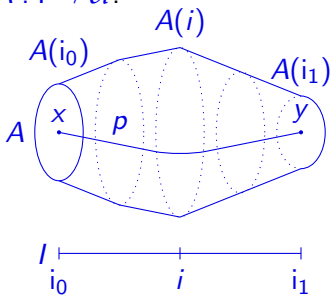
$$\mathsf{PathP} : (A : \mathsf{I} \to \mathcal{U}) \to A\ \mathsf{i}_0 \to A\, i1 \to \mathcal{U}$$

where $\mathsf{PathP}\ A\, x\, y$ can be thought of as the type of functions $p : (i : \mathsf{I}) \to A\, i$ such that

$$p\ \mathsf{i}_0 \mathrel{\hat{=}} x \qquad\qquad p\ \mathsf{i}_1 \mathrel{\hat{=}} y$$

Paths

A *varying type* is a function $A : \mathsf{I} \to \mathcal{U}$:



The type of *heterogeneous paths* is

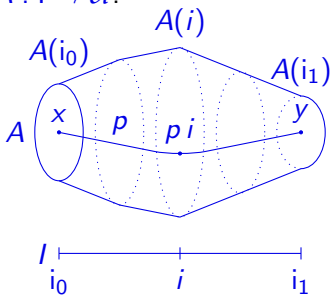
$$\mathsf{PathP} : (A : \mathsf{I} \to \mathcal{U}) \to A\ \mathsf{i}_0 \to A\, i1 \to \mathcal{U}$$

Given $p : \mathsf{PathP}\ A\ x\ y$ and $i : \mathsf{I}$, we have

$$p\, i : A\, i \qquad p\ \mathsf{i}_0 \doteq x \qquad p\ \mathsf{i}_1 \doteq y$$

Paths

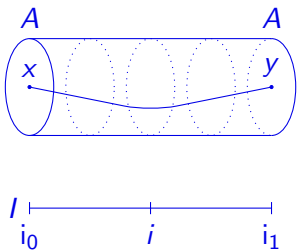
A *varying type* is a function $A : \mathsf{I} \to \mathcal{U}$:



The type of *heterogeneous paths* is

$$\mathsf{PathP} : (A : \mathsf{I} \to \mathcal{U}) \to A\ \mathsf{i}_0 \to A\, i1 \to \mathcal{U}$$

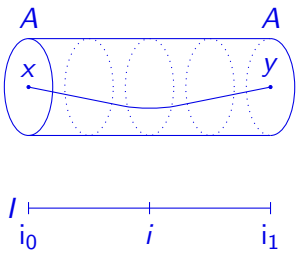
Identity types



Given $A : \mathcal{U}$, the type of **paths** in A is

$$x = y \quad \hat{=} \quad \mathsf{PathP}(\_ \mapsto A)\, x\, y$$

Identity types



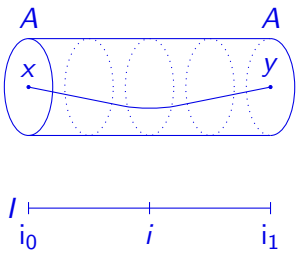
Given $A : \mathcal{U}$, the type of **paths** in A is

$$x = y \quad \hat{=} \quad \mathsf{PathP}(\_ \mapsto A)\, x\, y$$

We can define **reflexivity paths** by

$$\mathsf{refl} : (A : \mathit{Type})\, (x : A) \to x = x$$

Identity types



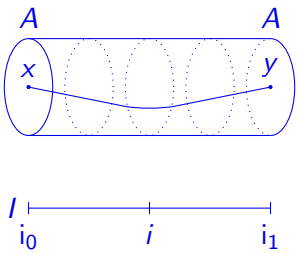
Given $A : \mathcal{U}$, the type of **paths** in A is

$$x = y \quad \hat{=} \quad \mathsf{PathP}(\_ \mapsto A)\, x\, y$$

We can define **reflexivity paths** by

$$\mathsf{refl} : (A : \mathit{Type})\, (x : A) \to x = x$$
$$A\, x \mapsto \lambda i.x$$

Identity types



Given $A : \mathcal{U}$, the type of **paths** in A is

$$x = y \quad \hat{=} \quad \mathsf{PathP}(\_ \mapsto A)\, x\, y$$

We can define **reflexivity paths** by

$$\mathsf{refl} : (A : \mathit{Type})\, (x : A) \to x = x$$
$$A\, x\, i \mapsto x$$

Functoriality

We can define ap (aka cong) by (with $x\,y : A$)

$$\mathsf{ap} : (f : A \to B)\,(p : x = y) \to f\,x = f\,y$$

Functoriality

We can define ap (aka cong) by (with $x\,y : A$)

$$\begin{aligned} \mathsf{ap} : (f : A \to B)\,(p : x = y) &\to f\,x = f\,y \\ f\,p\,i &\mapsto f\,(p\,i) \end{aligned}$$

Functoriality

We can define ap (aka cong) by (with $x\,y : A$)

$$\begin{aligned} \mathsf{ap} : (f : A \to B)\,(p : x = y) &\to f\,x = f\,y \\ f\,p\,i &\mapsto f\,(p\,i) \end{aligned}$$

and the dependent variant by

Functoriality

We can define ap (aka cong) by (with $x\,y : A$)

$$\begin{aligned}\mathsf{ap} : (f : A \to B)\,(p : x = y) &\to f\,x = f\,y \\ f\,p\,i &\mapsto f\,(p\,i)\end{aligned}$$

and the dependent variant by

$$\begin{aligned}\mathsf{apd} : (f : (x : A) \to B\,x)\,(p : x = y) &\to \mathsf{PathP}\ B\,(f\,x)\,(f\,y) \\ f\,p\,i &\mapsto f\,(p\,i)\end{aligned}$$

Functoriality

We can define ap (aka cong) by (with $x\,y : A$)

$$\begin{aligned}\mathsf{ap} : (f : A \to B)\,(p : x = y) &\to f\,x = f\,y \\ f\,p\,i &\mapsto f\,(p\,i)\end{aligned}$$

Things which used to require J can now be proved right away. For instance,

$$(f : A \to B)\,(g : B \to C)\,(p : x = y) \to \mathsf{ap}\,(g \circ f)\,p = \mathsf{ap}\,g\,(\mathsf{ap}\,f\,p)$$

Functoriality

We can define ap (aka cong) by (with $x\,y : A$)

$$\begin{aligned} \mathsf{ap} : (f : A \to B)\,(p : x = y) &\to f\,x = f\,y \\ f\,p\,i &\mapsto f\,(p\,i) \end{aligned}$$

Things which used to require J can now be proved right away. For instance,

$$\begin{aligned} (f : A \to B)\,(g : B \to C)\,(p : x = y) &\to \mathsf{ap}\,(g \circ f)\,p = \mathsf{ap}\,g\,(\mathsf{ap}\,f\,p) \\ f\,g\,p &\mapsto \mathsf{refl} \end{aligned}$$

Function extensionality

We can even prove **function extensionality** by

$$\mathsf{funext} : (f\ g : A \to B)\,(p : (x : A) \to f\ x = g\ x) \to f = g$$

Function extensionality

We can even prove **function extensionality** by

$$\mathsf{funext} : (f\ g : A \to B)\,(p : (x : A) \to f\ x = g\ x) \to f = g$$
$$f\ g\ p\ i\ x \mapsto p\ x\ i$$

Transport

Another primitive operation is **transport** which is

$$\text{transport} : (A = B) \to A \to B$$

Transport

Another primitive operation is **transport** which is

$$\mathsf{transport} : (A = B) \to A \to B$$

from which we can derive the more traditional transport function as

$$\mathsf{subst} : (B : A \to \mathcal{U}) \to (x = y) \to B\,x \to B\,y$$

Transport

Another primitive operation is **transport** which is

$$\text{transport} : (A = B) \to A \to B$$

from which we can derive the more traditional transport function as

$$\text{subst} : (B : A \to \mathcal{U}) \to (x = y) \to B\,x \to B\,y$$
$$B\,p\,x' \mapsto \text{transport}\,(\lambda i.B(p\,i))\,x'$$

Boolean operations

It will be useful to have the following operations on the elements of I:

- supremum: $\vee$
- infimum: $\wedge$
- complement: $\sim$

Boolean operations

It will be useful to have the following operations on the elements of $\mathbb{I}$:

- supremum: $\vee$
- infimum: $\wedge$
- complement: $\sim$

Those satisfy definitionally all the laws of De Morgan algebras, e.g.

$$(i \vee j) \vee k \mathrel{\hat{=}} i \vee (j \vee k) \qquad (i \vee \mathsf{i}_1) \mathrel{\hat{=}} \mathsf{i}_1 \qquad \sim \mathsf{i}_0 \mathrel{\hat{=}} \mathsf{i}_1 \qquad \dots$$

$$i \vee (j \wedge k) \mathrel{\hat{=}} (i \vee j) \wedge (i \vee k) \qquad (i \vee \mathsf{i}_0) \mathrel{\hat{=}} i \qquad \sim\!\sim i \mathrel{\hat{=}} i \qquad \dots$$

Boolean operations

It will be useful to have the following operations on the elements of $\mathbb{I}$:

- supremum: $\vee$
- infimum: $\wedge$
- complement: $\sim$

Those satisfy definitionally all the laws of De Morgan algebras, e.g.

$$(i \vee j) \vee k \mathrel{\hat{=}} i \vee (j \vee k) \qquad (i \vee \mathrm{i}_1) \mathrel{\hat{=}} \mathrm{i}_1 \qquad \sim \mathrm{i}_0 \mathrel{\hat{=}} \mathrm{i}_1 \qquad \dots$$
$$i \vee (j \wedge k) \mathrel{\hat{=}} (i \vee j) \wedge (i \vee k) \qquad (i \vee \mathrm{i}_0) \mathrel{\hat{=}} i \qquad \sim\sim i \mathrel{\hat{=}} i \qquad \dots$$

i.e. all the laws of boolean algebras excepting

$$i \vee \sim i \mathrel{\hat{=}} \mathrm{i}_1 \qquad\qquad i \wedge \sim i \mathrel{\hat{=}} \mathrm{i}_0$$

Path reversal

We can define the path reversal operation by

$$\mathsf{sym} : (x = y) \to (y = x)$$

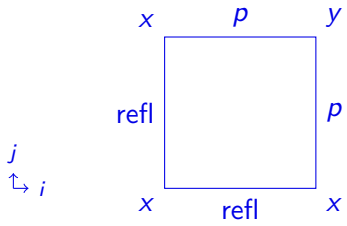
Path reversal

We can define the path reversal operation by

$$\begin{aligned} \mathsf{sym} : (x = y) &\to (y = x) \\ p\, i &\mapsto p\,(\sim i) \end{aligned}$$

A square

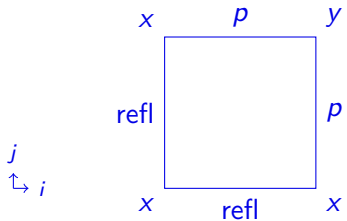
Given a path $p : x = y$, we can define a square



by

A square

Given a path $p : x = y$, we can define a square



by

$$\lambda i.\lambda j.p\,(i \wedge j)$$

J

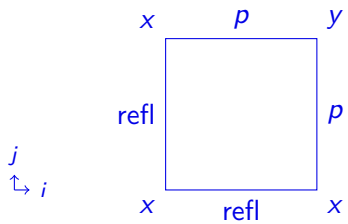
The J rule can similarly be shown for $x : A$ and $P : (y : A) \to x = y \to \mathit{Type}$ by

$$J : (r : P\ x\ \mathsf{refl})\ \{y : A\}\ (p : x = y) \to P\ y\ p$$
$$r\ p \mapsto$$

The J rule can similarly be shown for $x : A$ and $P : (y : A) \to x = y \to \mathit{Type}$ by

$$J : (r : P\, x\ \mathrm{refl})\ \{y : A\}\ (p : x = y) \to P\, y\, p$$
$$r\, p \mapsto$$

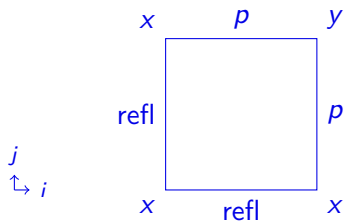
Recall that we have a square $\lambda i.\lambda j.p\,(i \wedge j)$



The J rule can similarly be shown for $x : A$ and $P : (y : A) \to x = y \to \mathit{Type}$ by

$$J : (r : P\, x\ \mathrm{refl})\ \{y : A\}\ (p : x = y) \to P\, y\, p$$
$$r\, p \mapsto \mathrm{transport}\,(\lambda j.P\,(p\, j)\,(\lambda i.p\,(i \wedge j)))\, r$$

Recall that we have a square $\lambda i.\lambda j.p\,(i \wedge j)$



J

J does not evaluate definitionally on refl

TODO: we however have transportRefl which allows showing this propositionally

TODO: definitional transportRefl is incompatible with cubical Agda [Swa18]

Defining composition

We could define composition by J as before

...

However this does not compute nicely because transport does not on path types...

In practice

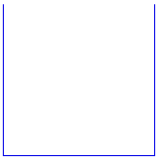
explain indices for wanted boundaries

Part I

Filling boxes

Defining composition

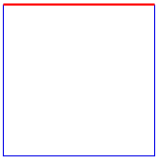
We deduce composition from an operation which states that for every “open cube”



we can define

Defining composition

We deduce composition from an operation which states that for every "open cube"

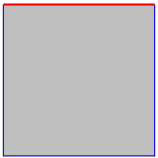


we can define

- the missing face: hcomp

Defining composition

We deduce composition from an operation which states that for every "open cube"



we can define

- the missing face: hcomp
- the interior cube: hfill

Partial types

We write

$$\mathsf{Partial}\ \varphi\ A$$

for the type of **partial types**: an element is a term of type A which is only defined when r is i_1, which can be thought of as a cube with missing faces.

Partial types

We write

$$\mathsf{Partial}\ \varphi\ A$$

for the type of **partial types**: an element is a term of type A which is only defined when r is i_1, which can be thought of as a cube with missing faces.

The only functions

$$\mathsf{I} \to \mathsf{Bool}$$

are the constant functions

$$\lambda i.\,\mathsf{true} \qquad\qquad \lambda i.\,\mathsf{false}$$

Partial types

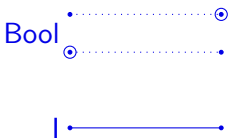
We write

$$\mathsf{Partial}\ \varphi\ A$$

for the type of **partial types**: an element is a term of type A which is only defined when r is i_1, which can be thought of as a cube with missing faces.

We can define a function

$$\begin{aligned} \mathsf{I} &\to \mathsf{Partial}\,(\sim i \vee i)\ \mathsf{Bool} \\ \mathsf{i}_0 &\mapsto \mathsf{false} \\ \mathsf{i}_1 &\mapsto \mathsf{true} \end{aligned}$$



Partial types

We write

$$\mathsf{Partial}\ \varphi\ A$$

for the type of **partial types**: an element is a term of type A which is only defined when r is i_1, which can be thought of as a cube with missing faces.

Agda checks that definitions match on their intersection so that we cannot define

$$\begin{aligned} \mathsf{I} &\to \mathsf{Bool} \\ \mathsf{i}_0 &\mapsto \mathsf{false} \\ \mathsf{i}_1 &\mapsto \mathsf{true} \\ i &\mapsto \mathsf{true} \end{aligned}$$

Partial types

We write

$$\mathsf{Partial}\ \varphi\ A$$

for the type of **partial types**: an element is a term of type A which is only defined when r is i_1, which can be thought of as a cube with missing faces.

Agda checks that all cases are covered so that we cannot define

$$\mathsf{I} \to \mathsf{Bool}$$
$$\mathsf{i}_0 \mapsto \mathsf{false}$$
$$\mathsf{i}_1 \mapsto \mathsf{true}$$

Partial types

We write

$$\mathsf{Partial}\ \varphi\ A$$

for the type of **partial types**: an element is a term of type A which is only defined when r is i_1, which can be thought of as a cube with missing faces.

The concrete syntax for pattern matching on interval arguments is

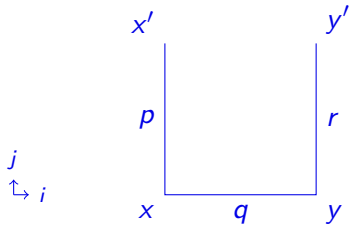
```
partialBool : (i : I) → Partial (~ i ∨ i) Bool
partialBool i (i = i0) = false
partialBool i (i = i1) = true
```

or

```
partialBool : (i : I) → Partial (~ i ∨ i) Bool
partialBool i = λ { (i = i0) → false ; (i = i1) → true }
```

Partial types

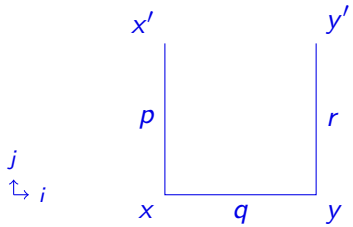
For instance, given $p : x = x'$, $q : x = y$ and $r : y = y'$, we can define



as

Partial types

For instance, given $p : x = x'$, $q : x = y$ and $r : y = y'$, we can define

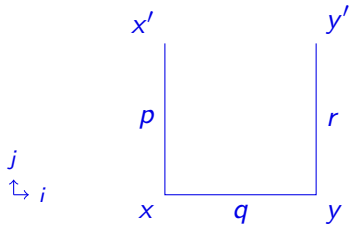


as

$$u : (i : \mathsf{I})\ (j : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee \sim j \vee i)\,A$$

Partial types

For instance, given $p : x = x'$, $q : x = y$ and $r : y = y'$, we can define



as

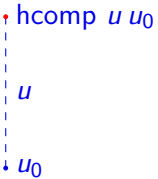
$$
\begin{array}{lll}
u : (i : \mathsf{I})\ (j : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee \sim j \vee i)\,A & & \\
\quad \mathsf{i}_0 & j & \mapsto p\,j \\
\quad i & \mathsf{i}_0 & \mapsto q\,i \\
\quad \mathsf{i}_1 & j & \mapsto r\,j
\end{array}
$$

Homogeneous composition

The **homogeneous composition** operation is

$$\mathsf{hcomp} : \{\varphi : I\}\ (u : I \to \mathsf{Partial}\ \varphi\ A)\ (u_0 : A) \to A$$

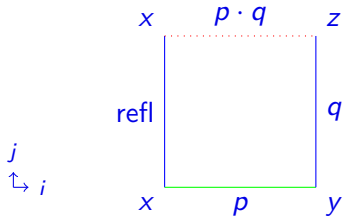
which can be pictured as



where Agda checks that u_0 agrees with u (for $i \mathrel{\hat{=}} \mathsf{i}_0$) when both are defined.

Defining composition

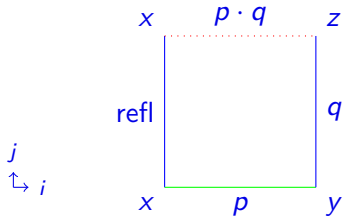
For instance, given $p : x = y$ and $q : y = z$, we can define their composite as



This means that we define

Defining composition

For instance, given $p : x = y$ and $q : y = z$, we can define their composite as



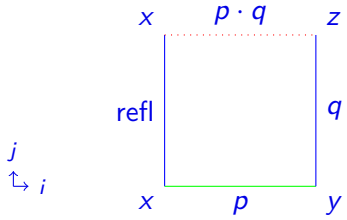
This means that we define

$$
\begin{array}{lll}
u : (i : \mathsf{I})\ (j : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee i)\,A & & \\
\quad \mathsf{i}_0 & j & \mapsto x \\
\quad \mathsf{i}_1 & j & \mapsto q\,j
\end{array}
$$

and

Defining composition

For instance, given $p : x = y$ and $q : y = z$, we can define their composite as



This means that we define

$$u : (i : \mathsf{I})\ (j : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee i)\,A$$
$$\mathsf{i}_0 \quad j \quad \mapsto x$$
$$\mathsf{i}_1 \quad j \quad \mapsto q\,j$$

and

$$p \cdot q \quad \hat{=} \quad \lambda i.\,\mathsf{hcomp}\,(u\,i)\,(p\,i)$$

Subtypes

We write

$$A[\varphi \mapsto u]$$

for the **subtype** of A whose elements are definitionally equal to u when φ is i_1.

Subtypes

We write

$$A[\varphi \mapsto u]$$

for the **subtype** of A whose elements are definitionally equal to u when φ is i_1.

It comes equipped with two operations

$$\begin{aligned} \mathsf{inS} &: \quad (u : A) \to A[\varphi \mapsto \lambda i.u] \\ \mathsf{outS} &: A[\varphi \mapsto u] \to A \end{aligned}$$

Homogeneous filler

The **homogeneous filler** is

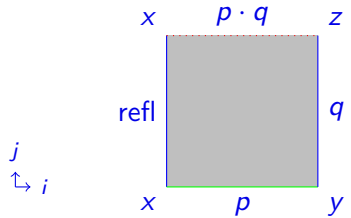
$$\mathsf{hfill} : \{\varphi : \mathsf{I}\}\ (u : (i : \mathsf{I}) \to \mathsf{Partial}\,\varphi A)\,(u_0 : A[\varphi \mapsto u\,\mathsf{i}_0])\,(i : I) \to A$$

which is

- u_0 when i is i_0
- $\mathsf{hcomp}\ u\ u_0$ when is is i_1

Homogeneous filler

For instance, if we apply hfill to the previous composition situation

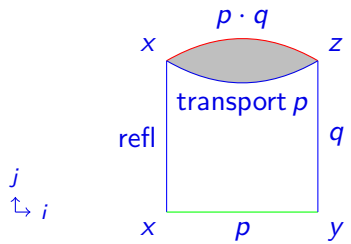


we obtain a path

$$f : \mathsf{PathP}\,(\lambda j.x = q\,j)\,p\,(p \cdot q)$$
$$j\,i \mapsto \mathsf{hfill}\,(u\,i)\,(\mathsf{inS}\,(p\,i))\,j$$

Homogeneous filler

For instance, if we apply hfill to the previous composition situation

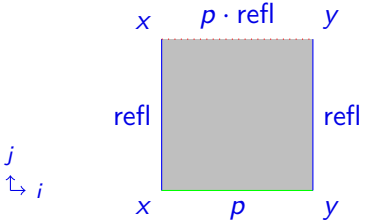


we obtain a path

$$f : \mathsf{PathP}\,(\lambda j.x = q\,j)\,p\,(p \cdot q)$$
$$j\,i \mapsto \mathsf{hfill}\,(u\,i)\,(\mathsf{inS}\,(p\,i))\,j$$

Composition is unital on the right

If we specialize to $q \hat{=} \mathsf{refl}$, we show composition is unital on the right

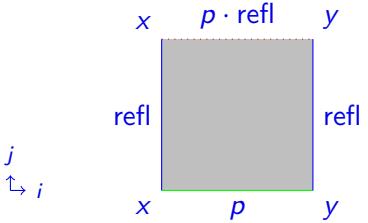


we obtain a path

$$f : \mathsf{PathP}\,(\lambda j.x = y)\,p\,(p \cdot \mathsf{refl})$$
$$j\,i \mapsto \mathsf{hfill}\,(u\,i)\,(\mathsf{inS}\,(p\,i))\,j$$

Composition is unital on the right

If we specialize to $q \hat{=} \mathsf{refl}$, we show composition is unital on the right



we obtain a path

$$f : p = p \cdot \mathsf{refl}$$
$$j\ i \mapsto \mathsf{hfill}\ (u\ i)\ (\mathsf{inS}\ (p\ i))\ j$$

Defining hfill

Note that hfill can be defined from hcomp:

$$\begin{array}{lll} \mathsf{hfill} : (u : (i : \mathsf{I}) \to \mathsf{Partial}\ \varphi)\ (u_0 : A[\varphi \mapsto u\ \mathsf{i}_0])\ (i : \mathsf{I}) \to A \\ \quad u \qquad\qquad\qquad\qquad\quad u_0 \qquad\qquad\qquad\quad i \qquad \mapsto \mathsf{hcomp}\ U\ (\mathsf{outS}\ u_0) \end{array}$$

where U is the function

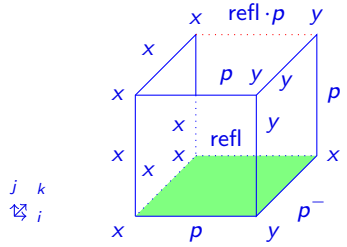
$$\begin{array}{l} j\,(\phi = \mathsf{i}_1) \mapsto u\,(i \wedge j) \\ j\,(i = \mathsf{i}_0) \mapsto \mathsf{outS}\ u_0 \end{array}$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

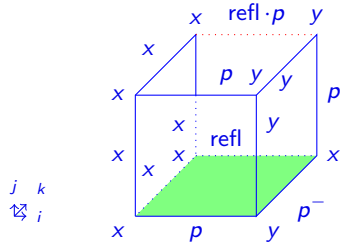


The "cube" can be defined as

$$u : (i : \mathsf{I})\ (j : \mathsf{I})\ (k : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee i \vee \sim k)\,A$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

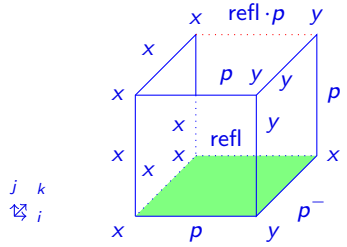


The "cube" can be defined as

$$u : (i : \mathbb{I})\ (j : \mathbb{I})\ (k : \mathbb{I}) \to \mathsf{Partial}\,(\sim i \vee i \vee \sim k)\ A$$
$$\mathsf{i}_0 \qquad j \qquad k \qquad \mapsto$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

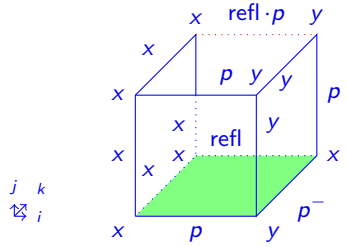


The "cube" can be defined as

$$u : (i : \mathsf{I})\ (j : \mathsf{I})\ (k : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee i \vee \sim k)\ A$$
$$\mathsf{i}_0 \qquad j \qquad k \qquad \mapsto x$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

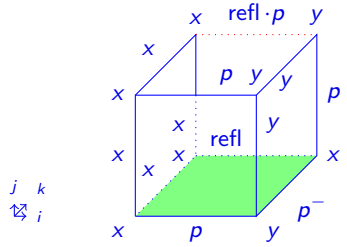


The "cube" can be defined as

$$
\begin{array}{llll}
u : (i : \mathsf{I})\ (j : \mathsf{I})\ (k : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee i \vee \sim k)\,A \\
\quad \mathsf{i}_0 \quad j \quad k \quad \mapsto x \\
\quad \mathsf{i}_1 \quad j \quad k \quad \mapsto
\end{array}
$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

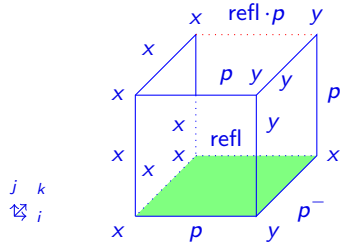


The "cube" can be defined as

$$
\begin{array}{llll}
u : (i : \mathsf{I})\ (j : \mathsf{I})\ (k : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee i \vee \sim k)\,A \\
\quad \mathsf{i}_0 \quad j \quad k \quad \mapsto x \\
\quad \mathsf{i}_1 \quad j \quad k \quad \mapsto p\,(j \vee \sim k)
\end{array}
$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

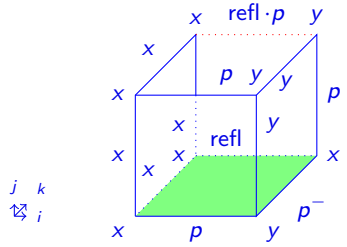


The "cube" can be defined as

$$
\begin{array}{llll}
u : (i : \mathsf{I})\ (j : \mathsf{I})\ (k : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee i \vee \sim k)\,A \\
\quad \mathsf{i}_0 & j & k & \mapsto x \\
\quad \mathsf{i}_1 & j & k & \mapsto p\,(j \vee \sim k) \\
\quad i & j & \mathsf{i}_0 & \mapsto
\end{array}
$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

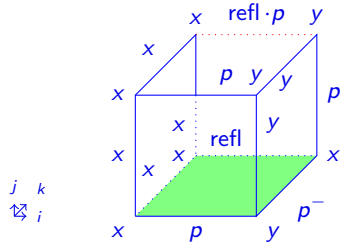


The "cube" can be defined as

$$
\begin{array}{llll}
u : (i : \mathsf{I})\ (j : \mathsf{I})\ (k : \mathsf{I}) \to \mathsf{Partial}\,(\sim i \vee i \vee \sim k)\,A \\
\quad \mathsf{i}_0 & j & k & \mapsto x \\
\quad \mathsf{i}_1 & j & k & \mapsto p\,(j \vee \sim k) \\
\quad i & j & \mathsf{i}_0 & \mapsto p\,i
\end{array}
$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

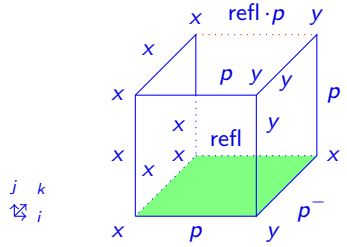


The bottom can be defined as

$$u_0 : (i : \mathbb{I})\ (k : \mathbb{I}) \to A$$
$$i \qquad k \qquad \mapsto$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.

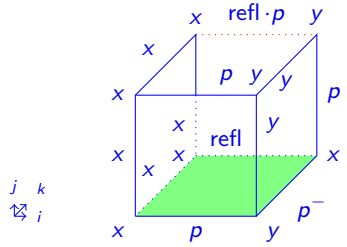


The bottom can be defined as

$$\begin{array}{lll} u_0 : (i : \mathsf{I})\ (k : \mathsf{I}) \to A \\ \quad i \qquad k \qquad\quad \mapsto p\,(i \wedge \sim k) \end{array}$$

Composition is unital on the left

The definition of composition is biased: showing $\mathsf{refl} \cdot p = p$ is more difficult.



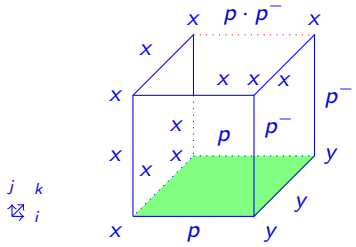
The filler is

$$f : (i : \mathbb{I})\ (j : \mathbb{I})\ (k : \mathbb{I}) \to A$$
$$i \quad j \quad k \quad \mapsto \mathsf{hfill}\,(\lambda j.u\ i\ j\ k)\,(\mathsf{inS}\,(u_0\ i\ k))$$

and we obtain the result as the top face ($j = \mathsf{i}_1$).

Composition is cancellative on the right

To show that composition is cancellative on the right, we can similarly use the "cube":



Glue types

Bibliography i

[Bru16] Guillaume Brunerie.
On the homotopy groups of spheres in homotopy type theory.
PhD thesis, Université de Nice Sophia Antipolis, 2016.
arXiv:1606.05916.

[CCHM18] Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg.
Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom.
In *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *LIPIcs*, pages 5:1–5:34. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
arXiv:1611.02108, doi:10.4230/LIPIcs.TYPES.2015.5.

[CHM18] Thierry Coquand, Simon Huber, and Anders Mörtberg.
On higher inductive types in cubical type theory.
In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 255–264, 2018.
arXiv:1802.01170, doi:10.1145/3209108.3209197.

[Hub19] Simon Huber.
Canonicity for cubical type theory.
Journal of Automated Reasoning, 63(2):173–210, 2019.
arXiv:1607.04156, doi:10.1007/s10817-018-9469-1.

[LM23] Axel Ljungström and Anders Mörtberg.
Formalizing $\pi_4(\mathbb{S}^3) \cong \mathbb{Z}/2\mathbb{Z}$ and computing a Brunerie number in cubical agda.
In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2023.
arXiv:2302.00151.

[Swa18] Andrew Swan.
Separating path and identity types in presheaf models of univalent type theory.
Preprint, 2018.
arXiv:1808.00920.

[VMA21] Andrea Vezzosi, Anders Mörtberg, and Andreas Abel.
Cubical agda: A dependently typed programming language with univalence and higher inductive types.
Journal of Functional Programming, 31, 2021.
doi:10.1017/S0956796821000034.